



Docusaurus

GENERATED: 2026-09-01



Contents

1	Docs	6
1.1	Introduction	6
1.1.1	Fast Track 🕒	6
1.1.2	Docusaurus: Documentation Made Easy	7
1.1.3	Migrating from v1	7
1.1.4	Features	8
1.1.5	Design principles	8
1.1.6	Comparison with other tools	9
1.1.7	Staying informed	11
1.1.8	Something missing?	11
1.2	Getting Started	11
1.2.1	Installation	11
1.2.2	Configuration	16
1.2.3	Playground	24
1.2.4	TypeScript Support	25
1.3	Guides	30
1.3.1	Creating Pages	30
1.3.2	Docs Introduction	33
1.3.3	Docs	36
1.3.4	Create a doc	36
1.3.5	Sidebar	41
1.3.6	Sidebar items	51
1.3.7	Autogenerated	65
1.3.8	Using multiple sidebars	76
1.3.9	Versioning	80
1.3.10	Docs Multi-instance	88
1.3.11	Blog	93
1.3.12	Markdown Features	109
1.3.13	MDX and React	115
1.3.14	Tabs	126
1.3.15	Code blocks	133
1.3.16	Admonitions	152
1.3.17	Headings and Table of contents	162
1.3.18	Assets	168
1.3.19	Markdown links	173
1.3.20	MDX Plugins	174
1.3.21	Math Equations	180
1.3.22	Diagrams	185



1.3.23	Styling and Layout	188
1.3.24	Swizzling	199
1.3.25	Static Assets	207
1.3.26	Search	210
1.3.27	Browser support	222
1.3.28	Search engine optimization (SEO)	224
1.3.29	Using Plugins	230
1.3.30	Deployment	238
1.3.31	Deploying to GitHub Pages	243
1.3.32	Deploying to Netlify	249
1.3.33	Deploying to Vercel	250
1.3.34	i18n - Introduction	251
1.3.35	Internationalization	254
1.3.36	i18n - Tutorial	254
1.3.37	i18n - Using git	266
1.3.38	i18n - Using Crowdin	270
1.3.39	What's next?	288
1.3.40	Head metadata	288
1.4	Advanced Tutorials	290
1.5	Advanced Guides	291
1.5.1	Architecture	291
1.5.2	Plugins	293
1.5.3	Routing	296
1.5.4	Static site generation (SSG)	304
1.5.5	Client architecture	309
1.6	Upgrading Docusaurus	314
1.6.1	Troubleshooting upgrades	314
1.7	Upgrading	315
1.7.1	Upgrading to Docusaurus v3	315
1.7.2	To Docusaurus v2	341
1.7.3	Overview	341
1.7.4	Automated migration	343
1.7.5	Manual migration	345
1.7.6	Versioned sites	362
1.7.7	Translated sites	366
2	API	371
2.1	CLI	371
2.1.1	Docusaurus CLI commands	371
2.2	Docusaurus Client API	376
2.2.1	Components	376
2.2.2	Hooks	383



2.2.3	Functions	389
2.2.4	Modules	391
2.3	docusaurus.config.js	392
2.3.1	Overview	392
2.3.2	Required fields	393
2.3.3	Optional fields	395
2.4	Plugin Method References	418
2.4.1	Plugin module	418
2.4.2	Plugin constructor	418
2.4.3	Example	418
2.4.4	Lifecycle APIs	420
2.4.5	Extending infrastructure	432
2.4.6	I18n lifecycles	435
2.4.7	Static methods	438
2.5	Docusaurus plugins	440
2.5.1	Content plugins	440
2.5.2	Behavior plugins	440
2.6	Plugins	440
2.6.1	 plugin-content-docs	440
2.6.2	 plugin-content-blog	453
2.6.3	 plugin-content-pages	467
2.6.4	 plugin-client-redirects	471
2.6.5	 plugin-debug	474
2.6.6	 plugin-google-gtag	476
2.6.7	 plugin-rsdoctor	478
2.6.8	 plugin-svgr	479
2.6.9	 plugin-google-tag-manager	480
2.6.10	 plugin-ideal-image	481
2.6.11	 plugin-css-cascade-layers	484
2.6.12	 plugin-pwa	486
2.6.13	 plugin-sitemap	496
2.6.14	 plugin-vercel-analytics	499
2.7	Docusaurus themes	500
2.7.1	Main themes	500
2.7.2	Enhancement themes	501
2.8	Themes	501
2.8.1	Theme configuration	501
2.8.2	 theme-classic	527
2.8.3	 theme-live-codeblock	528
2.8.4	 theme-search-algolia	529
2.8.5	 theme-mermaid	529



2.9	Miscellaneous	530
2.9.1	 create-docusaurus	530
2.9.2	 eslint-plugin	531
2.9.3	no-html-links	534
2.9.4	no-untranslated-text	534
2.9.5	prefer-docusaurus-heading	535
2.9.6	string-literal-i18n-messages	536
2.9.7	 logger	537



1 Docs

1.1 Introduction

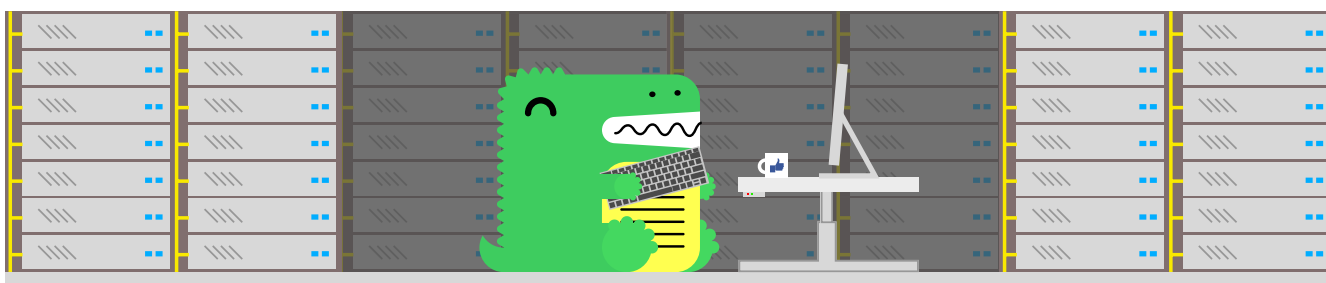
⚡ Docusaurus will help you ship a **beautiful documentation site in no time**.

📦 Building a custom tech stack is expensive. Instead, **focus on your content** and just write Markdown files.

💥 Ready for more? Use **advanced features** like versioning, i18n, search and theme customizations.

🔗 Check the **best Docusaurus sites** for inspiration.

🤖 Docusaurus is a **static-site generator**. It builds a **single-page application** with fast client-side navigation, leveraging the full power of **React** to make your site interactive. It provides out-of-the-box **documentation features** but can be used to create **any kind of site** (personal website, product, blog, marketing landing pages, etc).



1.1.1 Fast Track

Understand Docusaurus in **5 minutes** by playing!

Create a new Docusaurus site and follow the **very short** embedded tutorial.

Install [Node.js](#) and create a new Docusaurus site:

npm **Yarn** **pnpm** **Bun**

```
npx create-docusaurus@latest my-website classic
```

Start the site:

```
cd my-website
npx docusaurus start
```



Open `http://localhost:3000` and follow the tutorial.

**TIP**

Use docusaurus.new to test Docusaurus immediately in your browser!

Or read the [5-minute tutorial](#) online.

1.1.2 Docusaurus: Documentation Made Easy

In this presentation at [Algolia Community Event](#), [Meta Open Source team](#) shared a brief walk-through of Docusaurus. They covered how to get started with the project, enable plugins, and set up functionalities like documentation and blogging.



1.1.3 Migrating from v1

Docusaurus v2+ has been a total rewrite from Docusaurus v1, taking advantage of a completely modernized toolchain. After [v2's official release](#), we highly encourage you to **use Docusaurus v2+ over Docusaurus v1**, as Docusaurus v1 has been deprecated.

A [lot of users](#) are already using Docusaurus v2+ ([trends](#)).

Use Docusaurus v2+ if:

- ☒ You want a modern static documentation site
- ☒ You want a single-page application (SPA) with client-side routing
- ☒ You want the full power of React and MDX
- ☒ You do not need support for IE11

Use [Docusaurus v1](#) if:



-  You don't want a single-page application (SPA)
-  You need support for IE11 (...do you? IE [has already reached end-of-life](#) and is no longer officially supported)

For existing v1 users that are seeking to upgrade to v2+, you can follow our [migration guides](#).

1.1.4 Features

Docusaurus is built with high attention to the developer and contributor experience.

-  **Built with  and React:**
 - Extend and customize with React
 - Gain full control of your site's browsing experience by providing your own React components
-  **Pluggable:**
 - Bootstrap your site with a basic template, then use advanced features and plugins
 - Open source your plugins to share with the community
-  **Developer experience:**
 - Start writing your docs right now
 - Universal configuration entry point to make it more maintainable by contributors
 - Hot reloading with lightning-fast incremental build on changes
 - Route-based code and data splitting
 - Publish to GitHub Pages, Netlify, Vercel, and other deployment services with ease

Our shared goal—to help your users quickly find what they need and understand your products better. We share our best practices to help you build your docs site right and well.

-  **SEO friendly:**
 - HTML files are statically generated for every possible path.
 - Page-specific SEO to help your users land on your official docs directly relating their problems at hand.
-  **Powered by MDX:**
 - Write interactive components via JSX and React embedded in Markdown.
 - Share your code in live editors to get your users to love your products on the spot.
-  **Search:** Your full site is searchable.
-  **Document Versioning:** Helps you keep documentation in sync with project releases.
-  **Internationalization (i18n):** Translate your site in multiple locales.

Docusaurus v2+ is born to be compassionately accessible to all your users, and lightning-fast.

-  **Lightning-fast.** Docusaurus v2+ follows the [PRPL Pattern](#) that makes sure your content loads blazing fast.
-  **Accessible.** Attention to accessibility, making your site equally accessible to all users.

1.1.5 Design principles



- **Little to learn.** Docusaurus should be easy to learn and use as the API is quite small. Most things will still be achievable by users, even if it takes them more code and more time to write. Not having abstractions is better than having the wrong abstractions, and we don't want users to have to hack around the wrong abstractions. Mandatory talk—[Minimal API Surface Area](#).
- **Intuitive.** Users will not feel overwhelmed when looking at the project directory of a Docusaurus project or adding new features. It should look intuitive and easy to build on top of, using approaches they are familiar with.
- **Layered architecture.** The separations of concerns between each layer of our stack (content/theming/styling) should be clear—well-abstracted and modular.
- **Sensible defaults.** Common and popular performance optimizations and configurations will be done for users but they are given the option to override them.
- **No vendor lock-in.** Users are not required to use the default plugins or CSS, although they are highly encouraged to. Certain core infrastructures like React Loadable and React Router cannot be swapped because we do default performance optimization on them, but not higher-level ones. Choice of Markdown engines, CSS frameworks, CSS methodology, and other architectures will be entirely up to users.

We believe that, as developers, knowing how a library works helps us become better at using it. Hence we're dedicating effort to explaining the architecture and various components of Docusaurus with the hope that users reading it will gain a deeper understanding of the tool and be even more proficient in using it.

1.1.6 Comparison with other tools

Across all static site generators, Docusaurus has a unique focus on documentation sites and has many out-of-the-box features.

We've also studied other main static site generators and would like to share our insights on the comparison, hopefully helping you navigate through the prismatic choices out there.

Gatsby

[Gatsby](#) is packed with a lot of features, has a rich ecosystem of plugins, and is capable of doing everything that Docusaurus does. Naturally, that comes at a cost of a higher learning curve. Gatsby does many things well and is suitable for building many types of websites. On the other hand, Docusaurus tries to do one thing super well - be the best tool for writing and publishing content.

GraphQL is also pretty core to Gatsby, although you don't necessarily need GraphQL to build a Gatsby site. In most cases when building static websites, you won't need the flexibility that GraphQL provides.

Many aspects of Docusaurus v2+ were inspired by the best things about Gatsby and it's a great alternative.

[Docz](#) is a Gatsby theme to build documentation websites. It is currently less featured than Docusaurus.

Next.js

[Next.js](#) is another very popular hybrid React framework. It can help you build a good documentation website, but it is not opinionated toward the documentation use-case, and it will require a lot more work to implement what Docusaurus provides out-of-the-box.



[Nextra](#) is an opinionated static site generator built on top of Next.js. It is currently less featured than Docusaurus.

VitePress

[VitePress](#) has many similarities with Docusaurus - both focus heavily on content-centric websites and provides tailored documentation features out of the box. However, VitePress is powered by Vue, while Docusaurus is powered by React. If you want a Vue-based solution, VitePress would be a decent choice.

MkDocs

[MkDocs](#) is a popular Python static site generator with value propositions similar to Docusaurus.

It is a good option if you don't need a single-page application and don't plan to leverage React.

[Material for MkDocs](#) is a beautiful theme.

Docsify

[Docsify](#) makes it easy to create a documentation website, but is not a static-site generator and is not SEO friendly.

GitBook

[GitBook](#) has a very clean design and has been used by many open source projects. With its focus shifting towards a commercial product rather than an open-source tool, many of its requirements no longer fit the needs of open source projects' documentation sites. As a result, many have turned to other products. You may read about Redux's switch to Docusaurus [here](#).

Currently, GitBook is only free for open-source and non-profit teams. Docusaurus is free for everyone.

Jekyll

[Jekyll](#) is one of the most mature static site generators around and has been a great tool to use — in fact, before Docusaurus, most of Facebook's Open Source websites are/were built on Jekyll! It is extremely simple to get started. We want to bring a similar developer experience as building a static site with Jekyll.

In comparison with statically generated HTML and interactivity added using `<script />` tags, Docusaurus sites are React apps. Using modern JavaScript ecosystem tooling, we hope to set new standards on doc sites' performance, asset building pipeline and optimizations, and ease to set up.

Rspress

[Rspress](#) is a fast static site generator based on Rspack, a Rust-based bundler. It supports content writing with MDX (Markdown with React components), integrated text search, multilingual support (i18n), and extensibility through plugins. Designed for creating elegant documentation and static websites, Rspress produces static HTML files that are easy to deploy.

Rspress and Docusaurus are quite similar. They both have their pros and cons. Rspress was created more recently and benefits from a modern infrastructure that enables faster site builds. Docusaurus stands out for its maturity, comprehensive feature set, flexibility, and strong community. It is also [modernizing its infrastructure](#) regularly to remain competitive in terms of performance.



1.1.7 Staying informed

- [GitHub](#)
- [X](#)
- [Blog](#)
- [Discord](#)

1.1.8 Something missing?

If you find issues with the documentation or have suggestions on how to improve the documentation or the project in general, please [file an issue](#) for us, or send a tweet mentioning the [@docusaurus](#) X account.

For new feature requests, you can create a post on our [feature requests board \(Canny\)](#), which is a handy tool for road-mapping and allows for sorting by upvotes, which gives the core team a better indicator of what features are in high demand, as compared to GitHub issues which are harder to triage. Refrain from making a Pull Request for new features (especially large ones) as someone might already be working on it or will be part of our roadmap. Talk to us first!

1.2 Getting Started

1.2.1 Installation

Docusaurus consists of a set of npm [packages](#).

**TIP**

Use the [Fast Track](#) to understand Docusaurus in **5 minutes** 🕒!

Use [docusaurus.new](#) to test Docusaurus immediately in your browser!

Requirements

- [Node.js](#) version 24.14 or above (which can be checked by running `node -v`). You can use [nvm](#) to manage multiple Node.js versions on a single machine.
 - When installing Node.js, it is recommended to check all checkboxes related to dependencies.

Scaffold project website

The easiest way to install Docusaurus is to use the `create-docusaurus` command line tool that helps you scaffold a skeleton Docusaurus website. You can run this command anywhere in a new empty repository or within an existing repository, it will create a new directory containing the scaffolded files.

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)



```
npx create-docusaurus@latest my-website classic
```

We recommend the `classic` template so that you can get started quickly, and it contains features found in Docusaurus 1. The `classic` template contains `@docusaurus/preset-classic` which includes standard documentation, a blog, custom pages, and a CSS framework (with dark mode support). You can get up and running extremely quickly with the classic template and customize things later on when you have gained more familiarity with Docusaurus.

You can also use the template's TypeScript variant by passing the `--typescript` flag. See [TypeScript support](#) for more information.

npm Yarn pnpm Bun

```
npx create-docusaurus@latest my-website classic --typescript
```

⚠ META-ONLY

If you are setting up a new Docusaurus website for a Meta open source project, run this command inside an internal repository, which comes with some useful Meta-specific defaults:

```
scarf static-docs-bootstrap
```

Alternative installation commands

You can also initialize a new project using your preferred project manager:

npm Yarn pnpm Bun

```
npm init docusaurus
```




Run `npx create-docusaurus@latest --help`, or check out its [API docs](#) for more information about all available flags.

Project structure

Assuming you chose the classic template and named your site `my-website`, you will see the following files generated under a new directory `my-website/`:

```
my-website
├── blog
│   ├── 2019-05-28-hola.mdx
│   ├── 2019-05-29-hello-world.mdx
│   └── 2020-05-30-welcome.mdx
├── docs
│   ├── doc1.mdx
│   ├── doc2.mdx
│   ├── doc3.mdx
│   └── mdx.mdx
├── src
│   ├── css
│   │   └── custom.css
│   └── pages
│       ├── styles.module.css
│       └── index.js
├── static
│   └── img
├── docusaurus.config.js
├── package.json
├── README.md
├── sidebars.js
└── yarn.lock
```

Project structure rundown

- `/blog/` - Contains the blog Markdown files. You can delete the directory if you've disabled the blog plugin, or you can change its name after setting the `path` option. More details can be found in the [blog guide](#)
- `/docs/` - Contains the Markdown files for the docs. Customize the order of the docs sidebar in `sidebars.js`. You can delete the directory if you've disabled the docs plugin, or you can change its name after setting the `path` option. More details can be found in the [docs guide](#)
- `/src/` - Non-documentation files like pages or custom React components. You don't have to strictly put your non-documentation files here, but putting them under a centralized directory makes it easier to specify in case you need to do some sort of linting/processing
 - `/src/pages` - Any JSX/TSX/MDX file within this directory will be converted into a website page. More details can be found in the [pages guide](#)
- `/static/` - Static directory. Any contents inside here will be copied into the root of the final `build` directory



- `/docusaurus.config.js` - A config file containing the site configuration. This is the equivalent of `siteConfig.js` in Docusaurus v1
- `/package.json` - A Docusaurus website is a React app. You can install and use any npm packages you like in it
- `/sidebars.js` - Used by the documentation to specify the order of documents in the sidebar

Monorepos

If you are using Docusaurus for documentation of an existing project, a monorepo may be the solution for you. Monorepos allow you to share dependencies between similar projects. For example, your website may use your local packages to showcase latest features instead of depending on a released version. Then, your contributors can update the docs as they implement features. An example monorepo folder structure is below:

```
my-monorepo
├── package-a # Another package, your actual project
│   ├── src
│   └── package.json # Package A's dependencies
├── website # Docusaurus root
│   ├── docs
│   ├── src
│   └── package.json # Docusaurus' dependencies
└── package.json # Monorepo's shared dependencies
```

In this case, you should run `npx create-docusaurus` within the `./my-monorepo` folder.

If you're using a hosting provider such as Netlify or Vercel, you will need to change the `Base directory` of the site to where your Docusaurus root is. In this case, that would be `./website`. Read more about configuring ignore commands in the [deployment docs](#).

Read more about monorepos in the [Yarn documentation](#) (Yarn is not the only way to set up a monorepo, but it's a common solution), or check out [Docusaurus](#) and [Jest](#) for some real-world examples.

Running the development server

To preview your changes as you edit the files, you can run a local development server that will serve your website and reflect the latest changes.

npm **Yarn** **pnpm** **Bun**

```
cd my-website
npm run start
```



By default, a browser window will open at `http://localhost:3000`.

Congratulations! You have just created your first Docusaurus site! Browse around the site to see what's available.

Build

Docusaurus is a modern static website generator, so we need to build the website into a directory of static contents and put it on a web server so that it can be viewed. To build the website:

npm **Yarn** **pnpm** **Bun**

```
npm run build
```

and contents will be generated within the `/build` directory, which can be copied to any static file hosting service like [GitHub pages](#), [Vercel](#) or [Netlify](#). Check out the docs on [deployment](#) for more details.

Updating your Docusaurus version

There are many ways to update your Docusaurus version. One guaranteed way is to manually change the version number in `package.json` to the desired version. Note that all `@docusaurus/`-namespaced packages should be using the same version.

! INFO

You are browsing the documentation of an unreleased version. If you want to use any unreleased feature, you can use the [@canary](#) release.

package.json

```
{
  "dependencies": {
    "@docusaurus/core": "3.10.2",
    "@docusaurus/preset-classic": "3.10.2",
    // ...
  }
}
```

Then, in the directory containing `package.json`, run your package manager's install command:

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)

```
npm install
```

**TIP**

`npm install` may report several vulnerabilities and recommend running `npm audit` to address them. Typically, these reported vulnerabilities, such as RegExp DOS vulnerabilities, are harmless and can be safely ignored. Also read this article, which reflects our thinking: [npm audit: Broken by Design](#).

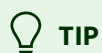
To check that the update occurred successfully, run:

```
npx docusaurus --version
```

You should see the correct version as output.

Alternatively, if you are using Yarn, you can do:

```
yarn add @docusaurus/core @docusaurus/preset-classic
```

**TIP**

Use new unreleased features of Docusaurus with the `@canary` npm dist tag

Problems?

Ask for help on [Stack Overflow](#), on our [GitHub repository](#), our [Discord server](#), or [X](#).

1.2.2 Configuration

**INFO**

Check the [docusaurus.config.js API reference](#) for an exhaustive list of options.

Docusaurus has a unique take on configurations. We encourage you to congregate information about your site into one place. We guard the fields of this file and facilitate making this data object accessible across your site.



Keeping a well-maintained `docusaurus.config.js` helps you, your collaborators, and your open source contributors to be able to focus on documentation while still being able to customize the site.

Syntax to declare `docusaurus.config.js`

The `docusaurus.config.js` file is run in Node.js and should export either:

- a **config object**
- a **function** that creates the config object

! INFO

The `docusaurus.config.js` file supports:

- **ES Modules**
- **CommonJS**
- **TypeScript**

Constraints:

- **Required:** use `export default /* your config*/` (or `module.exports`) to export your Docusaurus config
- **Optional:** use `import Lib from 'lib'` (or `require('lib')`) to import Node.js packages

Docusaurus gives us the ability to declare its configuration in various **equivalent ways**, and all the following config examples lead to the exact same result:

`docusaurus.config.js`

```
export default {  
  title: 'Docusaurus',  
  url: 'https://docusaurus.io',  
  // your site config ...  
};
```

`docusaurus.config.js`

```
module.exports = {  
  title: 'Docusaurus',  
  url: 'https://docusaurus.io',  
  // your site config ...  
};
```



docusaurus.config.ts

```
import type {Config} from '@docusaurus/types';

export default {
  title: 'Docusaurus',
  url: 'https://docusaurus.io',
  // your site config ...
} satisfies Config;
```

docusaurus.config.js

```
const config = {
  title: 'Docusaurus',
  url: 'https://docusaurus.io',
  // your site config ...
};

export default config;
```

docusaurus.config.js

```
export default function configCreator() {
  return {
    title: 'Docusaurus',
    url: 'https://docusaurus.io',
    // your site config ...
  };
}
```

docusaurus.config.js



```
export default async function createConfigAsync() {
  return {
    title: 'Docusaurus',
    url: 'https://docusaurus.io',
    // your site config ...
  };
}
```



USING ESM-ONLY PACKAGES

Using an async config creator can be useful to import ESM-only modules (notably most Remark plugins). It is possible to import such modules thanks to dynamic imports:

docusaurus.config.js

```
export default async function createConfigAsync() {
  // Use a dynamic import instead of require('esm-lib')
  const lib = await import('lib');

  return {
    title: 'Docusaurus',
    url: 'https://docusaurus.io',
    // rest of your site config...
  };
}
```

What goes into a `docusaurus.config.js`?

You should not have to write your `docusaurus.config.js` from scratch even if you are developing your site. All templates come with a `docusaurus.config.js` that includes defaults for the common options.

However, it can be helpful if you have a high-level understanding of how the configurations are designed and implemented.

The high-level overview of Docusaurus configuration can be categorized into:

Site metadata

Site metadata contains the essential global metadata such as `title`, `url`, `baseUrl`, and `favicon`.

They are used in several places such as your site's title and headings, browser tab icon, social sharing (Facebook, X) information or even to generate the correct path to serve your static files.

Deployment configurations



Deployment configurations such as `projectName`, `organizationName`, and optionally `deploymentBranch` are used when you deploy your site with the `deploy` command.

It is recommended to check the [deployment docs](#) for more information.

Theme, plugin, and preset configurations

List the [themes](#), [plugins](#), and [presets](#) for your site in the `themes`, `plugins`, and `presets` fields, respectively. These are typically npm packages:

docusaurus.config.js

```
export default {
  // ...
  plugins: [
    '@docusaurus/plugin-content-blog',
    '@docusaurus/plugin-content-pages',
  ],
  themes: ['@docusaurus/theme-classic'],
};
```

TIP

Docusaurus supports [module shorthands](#), allowing you to simplify the above configuration as:

docusaurus.config.js

```
export default {
  // ...
  plugins: ['content-blog', 'content-pages'],
  themes: ['classic'],
};
```

They can also be loaded from local directories:

docusaurus.config.js



```
import path from 'path';

export default {
  // ...
  themes: [path.resolve(__dirname, '/path/to/docusaurus-local-theme')],
};
```

To specify options for a plugin or theme, replace the name of the plugin or theme in the config file with an array containing the name and an options object:

docusaurus.config.js

```
export default {
  // ...
  plugins: [
    [
      'content-blog',
      {
        path: 'blog',
        routeBasePath: 'blog',
        include: ['*.md', '*.mdx'],
        // ...
      },
    ],
    'content-pages',
  ],
};
```

To specify options for a plugin or theme that is bundled in a preset, pass the options through the `presets` field. In this example, `docs` refers to `@docusaurus/plugin-content-docs` and `theme` refers to `@docusaurus/theme-classic`.

docusaurus.config.js



```
export default {
  // ...
  presets: [
    [
      '@docusaurus/preset-classic',
      {
        docs: {
          sidebarPath: './sidebars.js',
        },
        theme: {
          customCss: ['./src/css/custom.css'],
        },
      },
    ],
  ],
};
```

**TIP**

The `presets: [['classic', {...}]]` shorthand works as well.

For further help configuring themes, plugins, and presets, see [Using Plugins](#).

Custom configurations

Docusaurus guards `docusaurus.config.js` from unknown fields. To add custom fields, define them in `customFields`.

Example:

`docusaurus.config.js`

```
export default {
  // ...
  customFields: {
    image: '',
    keywords: [],
  },
  // ...
};
```

Accessing configuration from components



Your configuration object will be made available to all the components of your site. And you may access them via React context as `siteConfig`.

Basic example:

```
import React from 'react';
import useDocusaurusContext from '@docusaurus/useDocusaurusContext';

const Hello = () => {
  const {siteConfig} = useDocusaurusContext();
  const {title, tagline} = siteConfig;

  return <div>`${title} · ${tagline}`</div>;
};
```

**TIP**

If you just want to use those fields on the client side, you could create your own JS files and import them as ES6 modules, there is no need to put them in `docusaurus.config.js`.

Customizing Babel Configuration

Docusaurus transpiles your site's source code using Babel by default. If you want to customize the Babel configuration, you can do so by creating a `babel.config.js` file in your project root.

To use the built-in preset as a base configuration, install the following package and use it

npm **Yarn** **pnpm** **Bun**

```
npm install --save @docusaurus/babel
```

Then use the preset in your `babel.config.js` file:

babel.config.js

```
export default {
  presets: ['@docusaurus/babel/preset'],
};
```



Most of the time, the default preset configuration will work just fine. If you want to customize your Babel configuration (e.g. to add support for Flow), you can directly edit this file. For your changes to take effect, you need to restart the Docusaurus dev server.

1.2.3 Playground

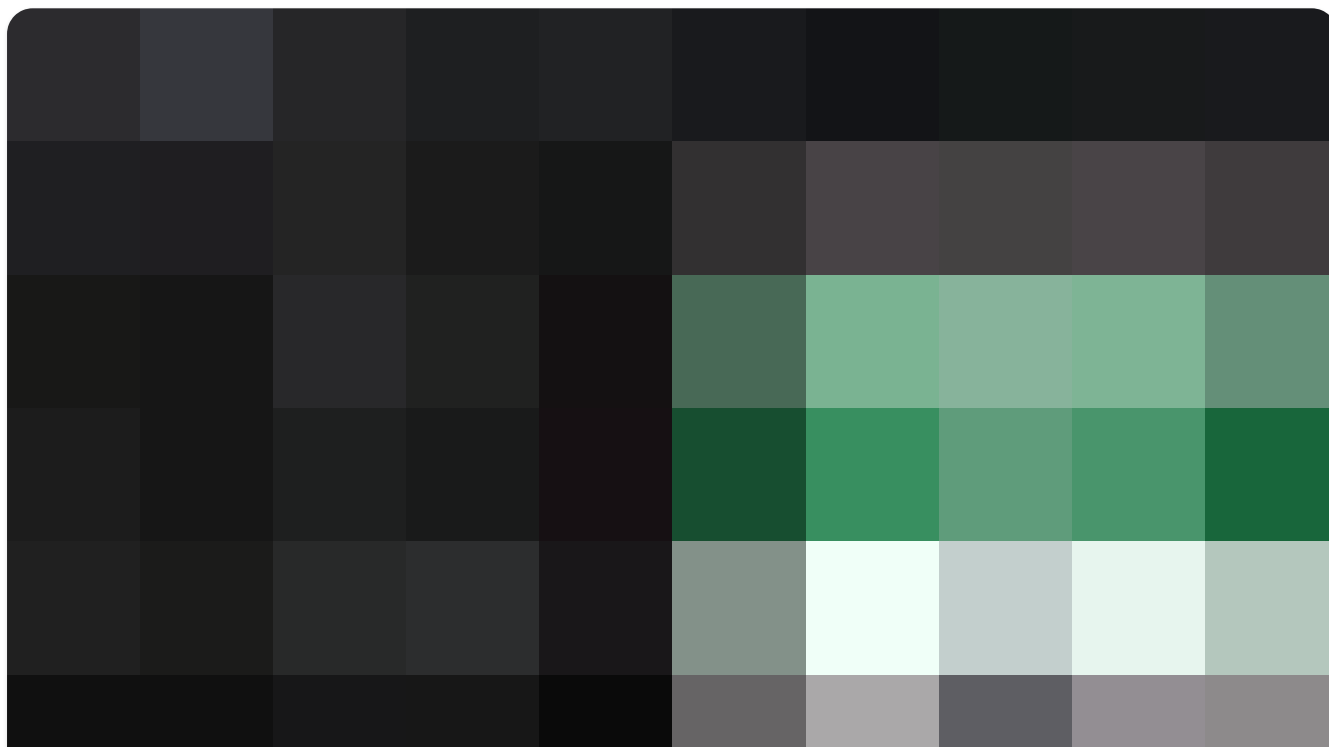
Playgrounds allow you to run Docusaurus **in your browser, without installing anything!**

They are mostly useful for:

- Testing Docusaurus
- Reporting bugs

Use docusaurus.new as an easy-to-remember shortcut.

Choose one of the available options below.



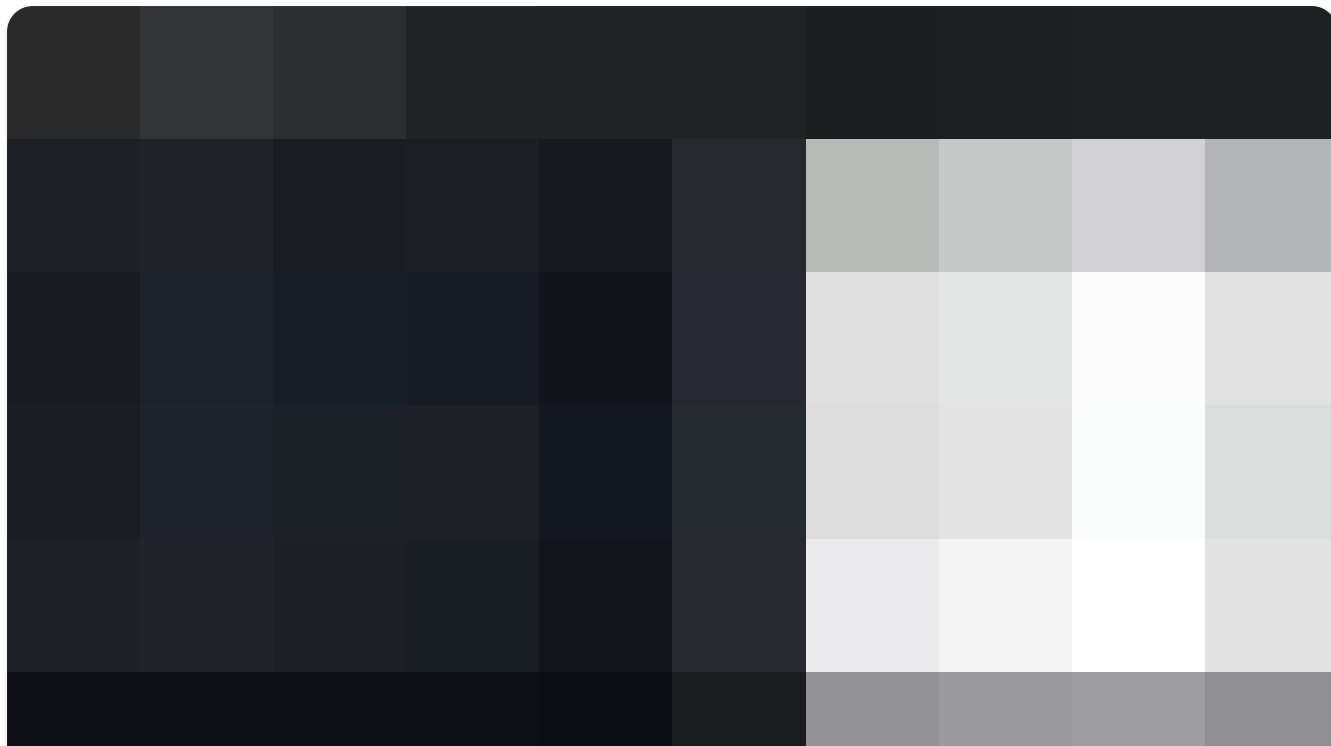
CodeSandbox

CodeSandbox is an online code editor and development environment that allows developers to create, share and collaborate on web development projects in a browser-based environment

Try it now!

[JavaScript](#)

[TypeScript](#)



StackBlitz uses a novel [WebContainers](#) technology to run Docusaurus directly in your browser.

Try it now!

JavaScript

TypeScript



TIP

For convenience, we'll remember your choice next time you visit [docusaurus.new](#).

1.2.4 TypeScript Support

Docusaurus is written in TypeScript and provides first-class TypeScript support.

The minimum required version is **TypeScript 6.0**.

Initialization

Docusaurus supports writing and using TypeScript theme components. If the init template provides a TypeScript variant, you can directly [initialize a site](#) with full TypeScript support by using the `--typescript` flag.

npm

Yarn

pnpm

Bun



```
npx create-docusaurus@latest my-website classic --typescript
```

Below are some guides on how to migrate an existing project to TypeScript.

Setup

Add the following packages to your project:

npm

Yarn

pnpm

Bun

```
npm install --save-dev typescript @docusaurus/module-type-aliases  
@docusaurus/tsconfig @docusaurus/types
```

Then add `tsconfig.json` to your project root with the following content:

tsconfig.json

```
{  
  "extends": "@docusaurus/tsconfig",  
  "compilerOptions": {}  
}
```

Docusaurus doesn't use this `tsconfig.json` to compile your project. It is added just for a nicer Editor experience, although you can choose to run `tsc` to type check your code for yourself or on CI.

Now you can start writing TypeScript theme components.

Ambient types for optional plugins

Some plugins and themes ship their own ambient `.d.ts` files that TypeScript does not load automatically.

For example, importing from `@theme/Playground` (provided by `@docusaurus/theme-live-codeblock`) fails with `TS2307: Cannot find module '@theme/Playground'` until you opt the plugin's types in.

Add a triple-slash reference in a project `.d.ts` file (recommended):

src/types.d.ts



```
/// <reference types="@docusaurus/theme-live-codeblock" />
```

Alternatively, add the package to the `types` compiler option of your `tsconfig.json`. This disables TypeScript's default loading of `@types/*` packages, so include any other type packages you rely on explicitly:

tsconfig.json

```
{
  "extends": "@docusaurus/tsconfig",
  "compilerOptions": {
    "types": ["@docusaurus/theme-live-codeblock"]
  }
}
```

The same applies to any other optional plugin or theme that ships ambient types.

Typing the config file

It is possible to use a TypeScript config file in Docusaurus.

docusaurus.config.ts



```
import type {Config} from '@docusaurus/types';
import type * as Preset from '@docusaurus/preset-classic';

const config: Config = {
  title: 'My Site',
  favicon: 'img/favicon.ico',

  /* Your site config here */

  presets: [
    [
      'classic',
      {
        /* Your preset config here */
      } satisfies Preset.Options,
    ],
  ],

  themeConfig: {
    /* Your theme config here */
  } satisfies Preset.ThemeConfig,
};

export default config;
```

It is also possible to use **JSDoc type annotations** within a `.js` file:

By default, the Docusaurus TypeScript config does not type-check JavaScript files.

The `// @ts-check` comment ensures the config file is properly type-checked when running `npm run tsc`.

```
docusaurus.config.js
```




```
// @ts-check

/** @type {import('@docusaurus/types').Config} */
const config = {
  tagline: 'Dinosaurs are cool',
  favicon: 'img/favicon.ico',

  /* Your site config here */

  presets: [
    [
      '@docusaurus/preset-classic',
      /** @type {import('@docusaurus/preset-classic').Options} */
      (
        {
          /* Your preset config here */
        }
      ),
    ],
  ],
  themeConfig:
    /** @type {import('@docusaurus/preset-classic').ThemeConfig} */
    (
      {
        /* Your theme config here */
      }
    ),
};

export default config;
```

TIP

Type annotations are very useful and help your IDE understand the type of config objects!

The best IDEs (VS Code, WebStorm, IntelliJ...) will provide a nice auto-completion experience.

Swizzling TypeScript theme components

For themes that support TypeScript theme components, you can add the `--typescript` flag to the end of the `swizzle` command to get TypeScript source code. For example, the following command will generate `index.tsx` and `styles.module.css` into `src/theme/Footer`.

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)

```
npm run swizzle @docusaurus/theme-classic Footer -- --typescript
```

All official Docusaurus themes support TypeScript theme components, including [theme-classic](#), [theme-live-codeblock](#), and [theme-search-algolia](#). If you are a Docusaurus theme package author who wants to add TypeScript support, see the [Lifecycle APIs docs](#).

1.3 Guides

1.3.1 Creating Pages

In this section, we will learn about creating pages in Docusaurus.

The [@docusaurus/plugin-content-pages](#) plugin empowers you to create **one-off standalone pages** like a showcase page, playground page, or support page. You can use React components, or Markdown.

NOTE

Pages do not have sidebars, only [docs](#) do.

INFO

Check the [Pages Plugin API Reference documentation](#) for an exhaustive list of options.

Add a React page

React is used as the UI library to create pages. Every page component should export a React component, and you can leverage the expressiveness of React to build rich and interactive content.

Create a file [/src/pages/helloReact.js](#):

```
/src/pages/helloReact.js
```



```
import React from 'react';
import Layout from '@theme/Layout';

export default function Hello() {
  return (
    <Layout title="Hello" description="Hello React Page">
      <div
        style={{
          display: 'flex',
          justifyContent: 'center',
          alignItems: 'center',
          height: '50vh',
          fontSize: '20px',
        }}>
        <p>
          Edit pages/helloReact.js and save to reload.
        </p>
      </div>
    </Layout>
  );
}
```

Once you save the file, the development server will automatically reload the changes. Now open <http://localhost:3000/helloReact> and you will see the new page you just created.

Each page doesn't come with any styling. You will need to import the `Layout` component from `@theme/Layout` and wrap your contents within that component if you want the navbar and/or footer to appear.

**TIP**

You can also create TypeScript pages with the `.tsx` extension (`helloReact.tsx`).

Add a Markdown page

Create a file `/src/pages/helloMarkdown.md`:

```
/src/pages/helloMarkdown.md
```



```
---
title: my hello page title
description: my hello page description
hide_table_of_contents: true
---

# Hello

How are you?
```

In the same way, a page will be created at `http://localhost:3000/helloMarkdown`.

Markdown pages are less flexible than React pages because it always uses the theme layout.

Here's an [example Markdown page](#).

**TIP**

You can use the full power of React in Markdown pages too, refer to the [MDX](#) documentation.

Routing

If you are familiar with other static site generators like Jekyll and Next, this routing approach will feel familiar to you. Any JavaScript file you create under `/src/pages/` directory will be automatically converted to a website page, following the `/src/pages/` directory hierarchy. For example:

- `/src/pages/index.js` → `[baseUrl]`
- `/src/pages/foo.js` → `[baseUrl]/foo`
- `/src/pages/foo/test.js` → `[baseUrl]/foo/test`
- `/src/pages/foo/index.js` → `[baseUrl]/foo/`

In this component-based development era, it is encouraged to co-locate your styling, markup, and behavior together into components. Each page is a component, and if you need to customize your page design with your own styles, we recommend co-locating your styles with the page component in its own directory. For example, to create a "Support" page, you could do one of the following:

- Add a `/src/pages/support.js` file
- Create a `/src/pages/support/` directory and a `/src/pages/support/index.js` file.

The latter is preferred as it has the benefits of letting you put files related to the page within that directory. For example, a CSS module file (`styles.module.css`) with styles meant to only be used on the "Support" page.

**NOTE**

This is merely a recommended directory structure, and you will still need to manually import the CSS module file within your component module (`support/index.js`).

By default, any Markdown or JavaScript file starting with `_` will be ignored and no routes will be created for that file (see the `exclude` option).

```
my-website
├── src
│   └── pages
│       ├── styles.module.css
│       ├── index.js
│       ├── _ignored.js
│       ├── _ignored-folder
│       │   ├── Component1.js
│       │   └── Component2.js
│       └── support
│           ├── index.js
│           └── styles.module.css
└── .
```

WARNING

All JavaScript/TypeScript files within the `src/pages/` directory will have corresponding website paths generated for them. If you want to create reusable components into that directory, use the `exclude` option (by default, files prefixed with `_`, test files (`.test.js`), and files in `__tests__` directory are not turned into pages).

Duplicate Routes

You may accidentally create multiple pages that are meant to be accessed on the same route. When this happens, Docusaurus will warn you about duplicate routes when you run `yarn start` or `yarn build` (behavior configurable through the `onDuplicateRoutes` config), but the site will still be built successfully. The page that was created last will be accessible, but it will override other conflicting pages. To resolve this issue, you should modify or remove any conflicting routes.

1.3.2 Docs Introduction

The docs feature provides users with a way to organize Markdown files in a hierarchical format.

INFO

Check the [Docs Plugin API Reference documentation](#) for an exhaustive list of options.

Your site's documentation is organized by four levels, from lowest to highest:



1. Individual pages.
2. Sidebars.
3. Versions.
4. Plugin instances.

The guide will introduce them in that order: starting from [how individual pages can be configured](#), to [how to create a sidebar or multiple ones](#), to [how to create and manage versions](#), to [how to use multiple docs plugin instances](#).

Docs-only mode

A freshly initialized Docusaurus site has the following structure:

<code>example.com/</code> <code>`src/pages/index.js`</code>	-> generated from
<code>example.com/docs/intro</code> <code>`docs/intro.md`</code>	-> generated from
<code>example.com/docs/tutorial-basics/...</code> <code>`docs/tutorial-basics/...`</code> ...	-> generated from
<code>example.com/blog/2021/08/26/welcome</code> <code>08-26-welcome/index.md`</code>	-> generated from <code>`blog/2021-</code>
<code>example.com/blog/2021/08/01/mdx-blog-post</code> <code>08-01-mdx-blog-post.mdx`</code> ...	-> generated from <code>`blog/2021-</code>

All docs will be served under the subroute `docs/`. But what if **your site only has docs**, or you want to prioritize your docs by putting them at the root?

Assume that you have the following in your configuration:

```
docusaurus.config.js
```



```
export default {
  // ...
  presets: [
    [
      '@docusaurus/preset-classic',
      {
        docs: {
          /* docs plugin options */
        },
        blog: {
          /* blog plugin options */
        },
        // ...
      },
    ],
  ],
};
```

To enter docs-only mode, change it to like this:

docusaurus.config.js

```
export default {
  // ...
  presets: [
    [
      '@docusaurus/preset-classic',
      {
        docs: {
          routeBasePath: '/', // Serve the docs at the site's root
          /* other docs plugin options */
        },
        blog: false, // Optional: disable the blog plugin
        // ...
      },
    ],
  ],
};
```

Note that you **don't necessarily have to give up on using the blog** or other plugins; all that `routeBasePath: '/'` does is that instead of serving the docs through `https://example.com/docs/some-doc`, they are now at the site root: `https://example.com/some-doc`. The blog, if enabled, can still be accessed through the `blog/` subroute.

Don't forget to put some page at the root (`https://example.com/`) through adding the front matter:



```
docs/intro.md
```

```
---
```

```
slug: /
```

```
---
```

This page will be the home page when users visit `https://example.com/`.

⚠ WARNING

If you added `slug: /` to a doc to make it the homepage, you should delete the existing homepage at `./src/pages/index.js`, or else there will be two files mapping to the same route!

Now, the site's structure will be like the following:

```
example.com/           -> generated from `docs/intro.md`
example.com/tutorial-basics/... -> generated from `docs/tutorial-basics/...`
...
```

💡 TIP

There's also a "blog-only mode" for those who only want to use the blog feature of Docusaurus. You can use the same method detailed above. Follow the setup instructions on [Blog-only mode](#).

1.3.3 Docs

1.3.4 Create a doc

Create a Markdown file, `greeting.md`, and place it under the `docs` directory.

```
website # root directory of your site
├── docs
│   └── greeting.md
├── src
│   └── pages
├── docusaurus.config.js
└── ...
```




```
---  
description: Create a doc page with rich content.  
---
```

Hello from Docusaurus

Are you ready to create the documentation site for your open source project?

Headers

will show up on the table of contents on the upper right

So that your users will know what this page is all about without scrolling down or even without reading too much.

Only h2 and h3 will be in the TOC by default.

You can configure the TOC heading levels either per-document or in the theme configuration.

The headers are well-spaced so that the hierarchy is clear.

- lists will help you
- present the key points
- that you want your users to remember
 - and you may nest them
 - multiple times

NOTE

All files prefixed with an underscore (`_`) under the `docs` directory are treated as "partial" pages and will be ignored by default.

Read more about [importing partial pages](#).

Doc front matter

The [front matter](#) is used to provide additional metadata for your doc page. Front matter is optional—Docusaurus will be able to infer all necessary metadata without the front matter. For example, the [doc tags](#) feature introduced below requires using front matter. For all possible fields, see [the API documentation](#).

Doc tags

Tags are declared in the front matter and introduce another dimension of categorization in addition to the [docs sidebar](#).



It is possible to define tags inline, or to reference predefined tags declared in a `tags file` (optional, usually `docs/tags.yml`).

In the following example:

- `docusaurus` references a predefined tag key declared in `docs/tags.yml`
- `Releases` is an inline tag, because it does not exist in `docs/tags.yml`

docs/my-doc.md

```
---
tags:
  - Releases
  - docusaurus
---

# Title

Content
```

docs/tags.yml

```
docusaurus:
  label: 'Docusaurus'
  permalink: '/docusaurus'
  description: 'Docs related to the Docusaurus framework'
```

**TIP**

Tags can also be declared with `tags: [Demo, Getting started]`.

Read more about all the possible [Yaml array syntaxes](#).

Organizing folder structure

How the Markdown files are arranged under the `docs` folder can have multiple impacts on Docusaurus content generation. However, most of them can be decoupled from the file structure.

Document ID

Every document has a unique `id`. By default, a document `id` is the name of the document (without the extension) relative to the root docs directory.



For example, the ID of `greeting.md` is `greeting`, and the ID of `guide/hello.md` is `guide/hello`.

```
website # Root directory of your site
├── docs
│   ├── greeting.md
│   └── guide
│       └── hello.md
```

However, the **last part** of the `id` can be defined by the user in the front matter. For example, if `guide/hello.md`'s content is defined as below, its final `id` is `guide/part1`.

```
---
id: part1
---

Lorem ipsum
```

The ID is used to refer to a document when hand-writing sidebars, or when using docs-related layout components or hooks.

Doc URLs

By default, the document's URL location is derived from the `document id`, which in turn is based on the document's file path.

If a file is named one of the following, the file name won't be included in the URL:

- Named as `index` (case-insensitive): `docs/Guides/index.md`
- Named as `README` (case-insensitive): `docs/Guides/README.mdx`
- Same name as parent folder: `docs/Guides/Guides.md`

In all cases, the default `slug` would only be `/Guides`, without the `/index`, `/README`, or duplicate `/Guides` segment.

NOTE

This convention is exactly the same as [the category index convention](#). However, the `isCategoryIndex` configuration does *not* affect the document URL.

Use the `slug` front matter to provide an explicit document URL and override the default one.

For example, suppose your site structure looks like this:



```
website # Root directory of your site
├── docs
│   └── guide
│       └── hello.md
```

By default, `hello.md` will be available at `/docs/guide/hello`. You can change its URL location to `/docs/bonjour`:

```
---
slug: /bonjour
---

Lorem ipsum
```

`slug` will be appended to the doc plugin's `routeBasePath`, which is `/docs` by default. See [Docs-only mode](#) for how to remove the `/docs` part from the URL.

NOTE

It is possible to use:

- absolute slugs: `slug: /mySlug`, `slug: /...`
- relative slugs: `slug: mySlug`, `slug: ../../mySlug...`

TIP

Changing a document's filename or `id`, will change its default URL. To prevent breaking permalinks when renaming files, we recommend setting an explicit `slug` to keep your URLs stable.

Making a document available at the root

If you want a document to be available at the root, and have a path like `https://docusaurus.io/docs/`, you can use the slug front matter:

```
---
id: my-home-doc
slug: /
---

Lorem ipsum
```



Sidebars

When using [autogenerated sidebars](#), the file structure will determine the sidebar structure.

Our recommendation for file system organization is: make your file system mirror the sidebar structure (so you don't need to handwrite your `sidebars.js` file), and use the `slug` front matter to customize URLs of each document.

1.3.5 Sidebar

Creating a sidebar is useful to:

- Group multiple **related documents** into an ordered tree
- **Display a common sidebar** on each of those documents
- Provide **paginated navigation**, with next/previous button

To use sidebars on your Docusaurus site:

1. Define a sidebars file that exports a dictionary of [sidebar objects](#).
2. Pass its path to the `@docusaurus/plugin-docs` plugin directly or via `@docusaurus/preset-classic`.

docusaurus.config.js

```
export default {
  presets: [
    [
      '@docusaurus/preset-classic',
      {
        docs: {
          sidebarPath: './sidebars.js',
        },
      ],
    ],
  ],
};
```

⚠️ NODE.JS RUNTIME

The sidebars file is run with Node.js. You can't use or import browsers APIs, React or JSX in it.

This section serves as an overview of miscellaneous features of the doc sidebar. In the following sections, we will more systematically introduce the following concepts:



Sidebar items

The sidebar supports various item types:



Autogenerated

Docusaurus can create a sidebar automatically from your filesystem structure: each folder creates a sidebar category, and each file creates a doc link.



Using multiple sidebars

You can create a sidebar for each set of Markdown files that you want to group together.

Default sidebar

If the `sidebarPath` is unspecified, Docusaurus [automatically generates a sidebar](#) for you, by using the filesystem structure of the `docs` folder:

sidebars.js

```
export default {
  mySidebar: [
    {
      type: 'autogenerated',
      dirName: '.', // generate sidebar from the docs folder (or
                    versioned_docs/<version>)
    },
  ],
};
```

You can also define your sidebars explicitly.



Sidebar object

A sidebar is a hierarchy of categories, doc links, and other hyperlinks.

```
type Sidebar =  
  // Normal syntax  
  | SidebarItem[]  
  // Shorthand syntax  
  | {[categoryLabel: string]: SidebarItem[]};
```

For example:

```
sidebars.js
```



```
export default {
  mySidebar: [
    {
      type: 'category',
      label: 'Getting Started',
      items: [
        {
          type: 'doc',
          id: 'doc1',
        },
      ],
    },
    {
      type: 'category',
      label: 'Docusaurus',
      items: [
        {
          type: 'doc',
          id: 'doc2',
        },
        {
          type: 'doc',
          id: 'doc3',
        },
      ],
    },
    {
      type: 'link',
      label: 'Learn more',
      href: 'https://example.com',
    },
  ],
};
```

This is a sidebars file that exports one sidebar, called `mySidebar`. It has three top-level items: two categories and one external link. Within each category, there are a few doc links.

A sidebars file can contain **multiple sidebar objects**, identified by their object keys.

```
type SidebarsFile = {
  [sidebarID: string]: Sidebar;
};
```

Theme configuration



Hideable sidebar

By enabling the `themeConfig.docs.sidebar.hideable` option, you can make the entire sidebar hideable, allowing users to better focus on the content. This is especially useful when content is consumed on medium-sized screens (e.g. tablets).

docusaurus.config.js

```
export default {
  themeConfig: {
    docs: {
      sidebar: {
        hideable: true,
      },
    },
  },
};
```

Auto-collapse sidebar categories

The `themeConfig.docs.sidebar.autoCollapseCategories` option would collapse all sibling categories when expanding one category. This saves the user from having too many categories open and helps them focus on the selected section.

docusaurus.config.js

```
export default {
  themeConfig: {
    docs: {
      sidebar: {
        autoCollapseCategories: true,
      },
    },
  },
};
```

Passing CSS classes

To pass CSS classes to a sidebar item, add the optional `className` attribute to any of the items. This is useful to apply visual customizations to specific sidebar items.



```
{
  type: 'doc',
  id: 'doc1',
  className: 'sidebar-item--highlighted',
};
```

Passing custom props

To pass in custom props to a sidebar item, add the optional `customProps` object to any of the items. This is useful to apply site customizations by swizzling React components rendering sidebar items.

```
{
  type: 'doc',
  id: 'doc1',
  customProps: {
    badges: ['new', 'green'],
    featured: true,
  },
};
```

Passing a unique key

Passing a unique `key` attribute can help uniquely identify a sidebar item. Sometimes other attributes (such as `label`) are not enough to distinguish two sidebar items from each other.

```
{
  type: 'category',
  label: 'API', // You may have multiple categories with this widespread label
  key: 'api-for-feature-1', // and now, they can be uniquely identified
};
```

❗ HOW IS THIS USEFUL?

Docusaurus only uses the `key` attribute to generate unique i18n translation keys. When a translation key conflict happens ([issue](#)), Docusaurus will tell you to apply a `key` to distinguish sidebar items.

Alternatively, you may have your own reasons for using the `key` attribute that will be passed to the respective sidebar item React components.



Sidebar Breadcrumbs

By default, breadcrumbs are rendered at the top, using the "sidebar path" of the current page.

This behavior can be disabled with plugin options:

docusaurus.config.js

```
export default {
  presets: [
    [
      '@docusaurus/preset-classic',
      {
        docs: {
          breadcrumbs: false,
        },
      ],
    ],
  ],
};
```

Complex sidebars example

A real-world example from the Docusaurus site:

sidebars.js



```
import type {SidebarsConfig} from '@docusaurus/plugin-content-docs';
const sidebars: SidebarsConfig = {
  docs: [
    'introduction',
    {
      type: 'category',
      label: 'Getting Started',
      link: {
        type: 'generated-index',
      },
      collapsed: false,
      items: [
        'installation',
        'configuration',
        'playground',
        'typescript-support',
      ],
    },
    {
      type: 'category',
      label: 'Guides',
      link: {
        type: 'generated-index',
        title: 'Docusaurus Guides',
        description:
          "Let's learn about the most important Docusaurus concepts!",
        keywords: ['guides'],
        image: '/img/docusaurus.png',
      },
      items: [
        'guides/creating-pages',
        {
          type: 'category',
          label: 'Docs',
          link: {
            type: 'doc',
            id: 'guides/docs/introduction',
          },
          items: [
            'guides/docs/create-doc',
            {
              type: 'category',
              label: 'Sidebar',
              link: {
                type: 'doc',
                id: 'guides/docs/sidebar/index',
              },
              items: [
                'guides/docs/sidebar/items',
              ],
            },
          ],
        },
      ],
    },
  ],
};
```



```
        'guides/docs/sidebar/autogenerated',
        'guides/docs/sidebar/multiple-sidebars',
      ],
    },
    'guides/docs/versioning',
    'guides/docs/multi-instance',
  ],
},
'blog',
{
  type: 'category',
  label: 'Markdown Features',
  link: {
    type: 'doc',
    id: 'guides/markdown-features/introduction',
  },
  items: [
    'guides/markdown-features/react',
    'guides/markdown-features/tabs',
    'guides/markdown-features/code-blocks',
    'guides/markdown-features/admonitions',
    'guides/markdown-features/toc',
    'guides/markdown-features/assets',
    'guides/markdown-features/links',
    'guides/markdown-features/plugins',
    'guides/markdown-features/math-equations',
    'guides/markdown-features/diagrams',
    'guides/markdown-features/head-metadata',
  ],
},
'styling-layout',
'swizzling',
'static-assets',
'search',
'browser-support',
'seo',
'using-plugins',
{
  type: 'category',
  label: 'Deployment',
  link: {type: 'doc', id: 'deployment/index'},
  items: [
    'deployment/github-pages',
    'deployment/netlify',
    'deployment/vercel',
  ],
},
{
  type: 'category',
```



```

    label: 'Internationalization',
    link: {type: 'doc', id: 'i18n/introduction'},
    items: [
      {
        type: 'doc',
        id: 'i18n/tutorial',
        label: 'Tutorial',
      },
      {
        type: 'doc',
        id: 'i18n/git',
        label: 'Using Git',
      },
      {
        type: 'doc',
        id: 'i18n/crowdin',
        label: 'Using Crowdin',
      },
    ],
  ],
  'guides/whats-next',
],
},
{
  type: 'category',
  label: 'Advanced Guides',
  link: {type: 'doc', id: 'advanced/index'},
  items: [
    'advanced/architecture',
    'advanced/plugins',
    'advanced/routing',
    'advanced/ssg',
    'advanced/client',
  ],
},
{
  type: 'category',
  label: 'Upgrading',
  link: {
    type: 'doc',
    id: 'migration/index',
  },
  items: [
    'migration/v3',
    {
      type: 'category',
      label: 'To Docusaurus v2',
      items: [
        'migration/v2/migration-overview',

```



```
        'migration/v2/migration-automated',
        'migration/v2/migration-manual',
        'migration/v2/migration-versioned-sites',
        'migration/v2/migration-translated-sites',
      ],
    },
  ],
},
],
api: [
  'cli',
  'docusaurus-core',
  {
    type: 'autogenerated',
    dirName: 'api',
  },
],
};
export default sidebars;
```

1.3.6 Sidebar items

The sidebar supports various item types:

- **Doc**: link to a doc page, associating it with the sidebar
- **Link**: link to any internal or external page
- **Category**: creates a dropdown of sidebar items
- **Autogenerated**: generate a sidebar slice automatically
- **HTML**: renders pure HTML in the item's position
- **Ref**: link to a doc page, without making the item take part in navigation generation

Doc: link to a doc

Use the `doc` type to link to a doc page and assign that doc to a sidebar:



```
type SidebarItemDoc =  
  // Normal syntax  
  | {  
    type: 'doc';  
    id: string;  
    label: string; // Sidebar label text  
    key?: string; // Sidebar key to uniquely identify the item  
    className?: string; // Class name for sidebar label  
    customProps?: Record<string, unknown>; // Custom props  
  }  
  
  // Shorthand syntax  
  | string; // docId shortcut
```

Example:

sidebars.js

```
export default {  
  mySidebar: [  
    // Normal syntax:  
    {  
      type: 'doc',  
      id: 'doc1', // document ID  
      label: 'Getting started', // sidebar label  
    },  
  
    // Shorthand syntax:  
    'doc2', // document ID  
  ],  
};
```

If you use the doc shorthand or [autogenerated](#) sidebar, you would lose the ability to customize the sidebar label through item definition. You can, however, use the `sidebar_label` Markdown front matter within that doc, which has higher precedence over the `label` key in the sidebar item. Similarly, you can use `sidebar_custom_props` to declare custom metadata for a doc page.

i NOTE

A `doc` item sets an [implicit sidebar association](#). Don't assign the same doc to multiple sidebars: change the type to `ref` instead.

 **TIP**

Sidebar custom props is a useful way to propagate arbitrary doc metadata to the client side, so you can get additional information when using any doc-related hook that fetches a doc object.

Link: link to any page

Use the `link` type to link to any page (internal or external) that is not a doc.

```
type SidebarItemLink = {
  type: 'link';
  label: string;
  href: string;
  description?: string;
  key?: string;
  className?: string;
  customProps?: Record<string, unknown>;
};
```

Example:

sidebars.js

```
export default {
  myLinksSidebar: [
    // External link
    {
      type: 'link',
      label: 'Facebook', // The link label
      href: 'https://facebook.com', // The external URL
    },

    // Internal link
    {
      type: 'link',
      label: 'Home', // The link label
      href: '/', // The internal path
    },
  ],
};
```

HTML: render custom markup



Use the `html` type to render custom HTML within the item's `<li>` tag.

This can be useful for inserting custom items such as dividers, section titles, ads, and images.

```
type SidebarItemHtml = {  
  type: 'html';  
  value: string;  
  defaultStyle?: boolean; // Use default menu item styles  
  key?: string;  
  className?: string;  
  customProps?: Record<string, unknown>;  
};
```

Example:

sidebars.js

```
export default {  
  myHtmlSidebar: [  
    {  
      type: 'html',  
      value: '', // The HTML to be  
        rendered  
      defaultStyle: true, // Use the default menu item styling  
    },  
  ],  
};
```

TIP

The menu item is already wrapped in an `<li>` tag, so if your custom item is simple, such as a title, just supply a string as the value and use the `className` property to style it:

sidebars.js



```
export default {  
  myHtmlSidebar: [  
    {  
      type: 'html',  
      value: 'Core concepts',  
      className: 'sidebar-title',  
    },  
  ],  
};
```

Category: create a hierarchy

Use the `category` type to create a hierarchy of sidebar items.

```
type SidebarItemCategory = {  
  type: 'category';  
  label: string; // Sidebar label text.  
  items: SidebarItem[]; // Array of sidebar items.  
  description?: string;  
  key?: string;  
  className?: string;  
  customProps?: Record<string, unknown>;  
  
  // Category options:  
  collapsible: boolean; // Set the category to be collapsible  
  collapsed: boolean; // Set the category to be initially collapsed or open  
  by default  
  link: SidebarItemCategoryLinkDoc | SidebarItemCategoryLinkGeneratedIndex;  
};
```

Example:

```
sidebars.js
```



```
export default {
  docs: [
    {
      type: 'category',
      label: 'Guides',
      collapsible: true,
      collapsed: false,
      items: [
        'creating-pages',
        {
          type: 'category',
          label: 'Docs',
          items: ['introduction', 'sidebar', 'markdown-features',
'versioning'],
        },
      ],
    },
  ],
},
];
```

TIP

Use the **shorthand syntax** when you don't need customizations:

sidebars.js

```
export default {
  docs: {
    Guides: [
      'creating-pages',
      {
        Docs: ['introduction', 'sidebar', 'markdown-features',
'versioning'],
      },
    ],
  },
};
```

Category links

With category links, clicking on a category can navigate you to another page.

 **TIP**

Use category links to introduce a category of documents.

Autogenerated categories can use the `_category_.yaml` file to declare the link.

Generated index page

You can auto-generate an index page that displays all the direct children of this category. The `slug` allows you to customize the generated page's route, which defaults to `/category/[categoryName]`.

sidebars.js

```
export default {
  docs: [
    {
      type: 'category',
      label: 'Guides',
      link: {
        type: 'generated-index',
        title: 'Docusaurus Guides',
        description: 'Learn about the most important Docusaurus concepts!',
        slug: '/category/docusaurus-guides',
        keywords: ['guides'],
        image: '/img/docusaurus.png',
      },
      items: ['pages', 'docs', 'blog', 'search'],
    },
  ],
};
```

See it in action on the [Docusaurus Guides page](#).

 TIP

Use `generated-index` links as a quick way to get an introductory document.

Doc link

A category can link to an existing document.

sidebars.js



```
export default {  
  docs: [  
    {  
      type: 'category',  
      label: 'Guides',  
      link: {type: 'doc', id: 'introduction'},  
      items: ['pages', 'docs', 'blog', 'search'],  
    },  
  ],  
};
```

See it in action on the [i18n introduction page](#).

Embedding generated index in doc page

You can embed the generated cards list in a normal doc page as well with the `DocCardList` component. It will display all the sidebar items of the parent category of the current document.

docs/sidebar/index.md

```
import DocCardList from '@theme/DocCardList';  
  
<DocCardList />
```



http://localhost:3000





Sidebar items

The sidebar supports various item types:



Autogenerated

Docusaurus can create a sidebar automatically from your filesystem structure: each folder creates a sidebar category, and each file creates a doc link.



Using multiple sidebars

You can create a sidebar for each set of Markdown files that you want to group together.

Collapsible categories

We support the option to expand/collapse categories. Categories are collapsible by default, but you can disable collapsing with `collapsible: false`.

```
sidebars.js
```



```
export default {
  docs: [
    {
      type: 'category',
      label: 'Guides',
      items: [
        'creating-pages',
        {
          type: 'category',
          collapsible: false,
          label: 'Docs',
          items: ['introduction', 'sidebar', 'markdown-features',
'versioning'],
        },
      ],
    },
  ],
};
```

To make all categories non-collapsible by default, set the `sidebarCollapsible` option in `plugin-content-docs` to `false`:

docusaurus.config.js

```
export default {
  presets: [
    [
      '@docusaurus/preset-classic',
      {
        docs: {
          sidebarCollapsible: false,
        },
      },
    ],
  ],
};
```

NOTE

The option in `sidebars.js` takes precedence over plugin configuration, so it is possible to make certain categories collapsible when `sidebarCollapsible` is set to `false` globally.

Expanded categories by default



Collapsible categories are collapsed by default. If you want them to be expanded on the first render, you can set `collapsed` to `false`:

sidebars.js

```
export default {
  docs: {
    Guides: [
      'creating-pages',
      {
        type: 'category',
        label: 'Docs',
        collapsed: false,
        items: ['markdown-features', 'sidebar', 'versioning'],
      },
    ],
  },
};
```

Similar to `collapsible`, you can also set the global configuration `options.sidebarCollapsed` to `false`. Individual `collapsed` options in `sidebars.js` will still take precedence over this configuration.

docusaurus.config.js

```
export default {
  presets: [
    [
      '@docusaurus/preset-classic',
      {
        docs: {
          sidebarCollapsed: false,
        },
      },
    ],
  ],
};
```

⚠ WARNING

When a category has `collapsed: true` but `collapsible: false` (either through `sidebars.js` or through plugin configuration), the latter takes precedence and the category is still rendered as expanded.



Using shorthands

You can express typical sidebar items without much customization more concisely with **shorthand syntaxes**. There are two parts to this: **doc shorthand** and **category shorthand**.

Doc shorthand

An item with type `doc` can be simply a string representing its ID:

Longhand

Shorthand

sidebars.js

```
export default {  
  sidebar: [  
    {  
      type: 'doc',  
      id: 'myDoc',  
    },  
  ],  
};
```

So it's possible to simplify the example above to:

sidebars.js



```
export default {
  mySidebar: [
    {
      type: 'category',
      label: 'Getting Started',
      items: [
        'doc1',
      ],
    },
    {
      type: 'category',
      label: 'Docusaurus',
      items: [
        'doc2',
        'doc3',
      ],
    },
    {
      type: 'link',
      label: 'Learn more',
      href: 'https://example.com',
    },
  ],
};
```

Category shorthand

A category item can be represented by an object whose key is its label, and the value is an array of subitems.

Longhand **Shorthand**

sidebars.js

```
export default {
  sidebar: [
    {
      type: 'category',
      label: 'Getting started',
      items: ['doc1', 'doc2'],
    },
  ],
};
```

This permits us to simplify that example to:



sidebars.js

```
export default {
  mySidebar: [
    {
      'Getting started': ['doc1'],
    },
    {
      Docusaurus: ['doc2', 'doc3'],
    },
    {
      type: 'link',
      label: 'Learn more',
      href: 'https://example.com',
    },
  ],
};
```

Each shorthand object after this transformation will contain exactly one entry. Now consider the further simplified example below:

sidebars.js

```
export default {
  mySidebar: [
    {
      'Getting started': ['doc1'],
      Docusaurus: ['doc2', 'doc3'],
    },
    {
      type: 'link',
      label: 'Learn more',
      href: 'https://example.com',
    },
  ],
};
```

Note how the two consecutive category shorthands are compressed into one object with two entries. This syntax generates a **sidebar slice**: you shouldn't see that object as one bulk item—this object is unwrapped, with each entry becoming a separate item, and they spliced together with the rest of the items (in this case, the "Learn more" link) to form the final sidebar level. Sidebar slices are also important when discussing [autogenerated sidebars](#).



Wherever you have an array of items that is reduced to one category shorthand, you can omit that enclosing array as well.

Before After

sidebars.js

```
export default {
  sidebar: [
    {
      'Getting started': ['doc1'],
      Docusaurus: [
        {
          'Basic guides': ['doc2', 'doc3'],
          'Advanced guides': ['doc4', 'doc5'],
        },
      ],
    },
  ],
};
```

1.3.7 Autogenerated

Docusaurus can **create a sidebar automatically** from your **filesystem structure**: each folder creates a sidebar category, and each file creates a doc link.

```
type SidebarItemAutogenerated = {
  type: 'autogenerated';
  dirName: string; // Source folder to generate the sidebar slice from
  (relative to docs)
};
```

Docusaurus can generate a full sidebar from your docs folder:

sidebars.js



```
export default {  
  myAutogeneratedSidebar: [  
    {  
      type: 'autogenerated',  
      dirName: '.', // '.' means the current docs folder  
    },  
  ],  
};
```

An `autogenerated` item is converted by Docusaurus to a **sidebar slice** (also discussed in [category shorthands](#)): a list of items of type `doc` or `category`, so you can splice **multiple autogenerated items** from multiple directories, interleaving them with regular sidebar items, in one sidebar level.

A real-world example

Consider this file structure:

```
docs  
├── api  
│   ├── product1-api  
│   │   └── api.md  
│   └── product2-api  
│       ├── basic-api.md  
│       └── pro-api.md  
├── intro.md  
└── tutorials  
    ├── advanced  
    │   ├── advanced1.md  
    │   ├── advanced2.md  
    │   └── read-more  
    │       ├── resource1.md  
    │       └── resource2.md  
    ├── easy  
    │   ├── easy1.md  
    │   └── easy2.md  
    ├── tutorial-end.md  
    ├── tutorial-intro.md  
    └── tutorial-medium.md
```

And assume every doc's ID is just its file name. If you define an autogenerated sidebar like this:



sidebars.js

```
export default {
  mySidebar: [
    'intro',
    {
      type: 'category',
      label: 'Tutorials',
      items: [
        'tutorial-intro',
        {
          type: 'autogenerated',
          dirName: 'tutorials/easy', // Generate sidebar slice from
docs/tutorials/easy
        },
        'tutorial-medium',
        {
          type: 'autogenerated',
          dirName: 'tutorials/advanced', // Generate sidebar slice from
docs/tutorials/advanced
        },
        'tutorial-end',
      ],
    },
    {
      type: 'autogenerated',
      dirName: 'api', // Generate sidebar slice from docs/api
    },
    {
      type: 'category',
      label: 'Community',
      items: ['team', 'chat'],
    },
  ],
};
```

It would be resolved as:

sidebars.js



```
export default {
  mySidebar: [
    'intro',
    {
      type: 'category',
      label: 'Tutorials',
      items: [
        'tutorial-intro',
        // Two files in docs/tutorials/easy
        'easy1',
        'easy2',
        'tutorial-medium',
        // Two files and a folder in docs/tutorials/advanced
        'advanced1',
        'advanced2',
        {
          type: 'category',
          label: 'read-more',
          items: ['resource1', 'resource2'],
        },
        'tutorial-end',
      ],
    },
    // Two folders in docs/api
    {
      type: 'category',
      label: 'product1-api',
      items: ['api'],
    },
    {
      type: 'category',
      label: 'product2-api',
      items: ['basic-api', 'pro-api'],
    },
    {
      type: 'category',
      label: 'Community',
      items: ['team', 'chat'],
    },
  ],
};
```

Note how the autogenerated source directories themselves don't become categories: only the items they contain do. This is what we mean by "sidebar slice".

Category index convention



Docusaurus can automatically link a category to its index document.

A category index document is a document following one of those filename conventions:

- Named as `index` (case-insensitive): `docs/Guides/index.md`
- Named as `README` (case-insensitive): `docs/Guides/README.mdx`
- Same name as parent folder: `docs/Guides/Guides.md`

This is equivalent to using a category with a [doc link](#):

sidebars.js

```
export default {
  docs: [
    {
      type: 'category',
      label: 'Guides',
      link: {type: 'doc', id: 'Guides/index'},
      items: [],
    },
  ],
};
```

TIP

Naming your introductory document `README.md` makes it show up when browsing the folder using the GitHub interface, while using `index.md` makes the behavior more in line with how HTML files are served.

TIP

If a folder only has one index page, it will be turned into a link instead of a category. This is useful for **asset collocation**:

```
some-doc
├── index.md
├── img1.png
└── img2.png
```

Customizing category index matching



It is possible to opt out any of the category index conventions, or define even more conventions. You can inject your own `isCategoryIndex` matcher through the `sidebarItemsGenerator` callback. For example, you can also pick `intro` as another file name eligible for automatically becoming the category index.

docusaurus.config.js

```
export default {
  plugins: [
    [
      '@docusaurus/plugin-content-docs',
      {
        async sidebarItemsGenerator({
          ...args,
          isCategoryIndex: defaultCategoryIndexMatcher, // The default
matcher implementation, given below
          defaultSidebarItemsGenerator,
        }) {
          return defaultSidebarItemsGenerator({
            ...args,
            isCategoryIndex(doc) {
              return (
                // Also pick intro.md in addition to the default ones
                doc.fileName.toLowerCase() === 'intro' ||
                defaultCategoryIndexMatcher(doc)
              );
            },
          });
        },
      ],
    ],
  ],
};
```

Or choose to not have any category index convention.

docusaurus.config.js



```
export default {
  plugins: [
    [
      '@docusaurus/plugin-content-docs',
      {
        async sidebarItemsGenerator({
          ...args,
          isCategoryIndex: defaultCategoryIndexMatcher, // The default
matcher implementation, given below
          defaultSidebarItemsGenerator,
        }) {
          return defaultSidebarItemsGenerator({
            ...args,
            isCategoryIndex() {
              // No doc will be automatically picked as category index
              return false;
            },
          });
        },
      ],
    ],
  ],
};
```

The `isCategoryIndex` matcher will be provided with three fields:

- `fileName`, the file's name without extension and with casing preserved
- `directories`, the list of directory names *from the lowest level to the highest level*, relative to the docs root directory
- `extension`, the file's extension, with a leading dot.

For example, for a doc file at `guides/sidebar/autogenerated.md`, the props the matcher receives are

```
const props = {
  fileName: 'autogenerated',
  directories: ['sidebar', 'guides'],
  extension: '.md',
};
```

The default implementation is:



```
function isCategoryIndex({fileName, directories}) {  
  const eligibleDocIndexNames = [  
    'index',  
    'readme',  
    directories[0].toLowerCase(),  
  ];  
  return eligibleDocIndexNames.includes(fileName.toLowerCase());  
}
```

Autogenerated sidebar metadata

For handwritten sidebar definitions, you would provide metadata to sidebar items through `sidebars.js`; for autogenerated, Docusaurus would read them from the item's respective file. In addition, you may want to adjust the relative position of each item because, by default, items within a sidebar slice will be generated in **alphabetical order** (using file and folder names).

Doc item metadata

The `label`, `className`, `key`, and `customProps` attributes are declared in front matter as `sidebar_label`, `sidebar_class_name`, `sidebar_key` and `sidebar_custom_props`, respectively. Position can be specified in the same way, via `sidebar_position` front matter.

docs/tutorials/tutorial-easy.md

```
---  
sidebar_position: 2  
sidebar_label: Easy  
sidebar_class_name: green  
sidebar_key: unique-sidebar-item-key  
---
```

Easy Tutorial

This is the easy tutorial!

Category item metadata

Add a `_category_.json` or `_category_.yaml` file in the respective folder. You can specify any category metadata and also the `position` metadata. `label`, `className`, `key`, `position`, and `customProps` will default to the respective values of the category's linked doc, if there is one.

JSON

YAML



docs/tutorials/_category_.json

```
{
  "position": 2.5,
  "label": "Tutorial",
  "key": "unique-sidebar-item-key",
  "collapsible": true,
  "collapsed": false,
  "className": "red",
  "link": {
    "type": "generated-index",
    "title": "Tutorial overview"
  },
  "customProps": {
    "description": "This description can be used in the swizzled DocCard"
  }
}
```

! INFO

If the `link` is explicitly specified, Docusaurus will not apply any [default conventions](#).

The doc links can be specified relatively, e.g. if the category is generated with the `guides` directory, `"link": {"type": "doc", "id": "intro"}` will be resolved to the ID `guides/intro`, only falling back to `intro` if a doc with the former ID doesn't exist.

You can also use `link: null` to opt out of default conventions and not generate any category index page.

! INFO

The position metadata is only used **within a sidebar slice**: Docusaurus does not re-order other items of your sidebar.

Using number prefixes

A simple way to order an autogenerated sidebar is to prefix docs and folders by number prefixes, which also makes them appear in the file system in the same order when sorted by file name:



```
docs
├── 01-Intro.md
├── 02-Tutorial Easy
│   ├── 01-First Part.md
│   ├── 02-Second Part.md
│   └── 03-End.md
├── 03-Tutorial Advanced
│   ├── 01-First Part.md
│   ├── 02-Second Part.md
│   ├── 03-Third Part.md
│   └── 04-End.md
└── 04-End.md
```

To make it **easier to adopt**, Docusaurus supports **multiple number prefix patterns**.

By default, Docusaurus will **remove the number prefix** from the doc id, title, label, and URL paths.

WARNING

Prefer using **additional metadata**.

Updating a number prefix can be annoying, as it can require **updating multiple existing Markdown links**:

```
docs/02-Tutorial Easy/01-First Part.md
```

- Check the [Tutorial End](../04-End.mdx);
- + Check the [Tutorial End](../05-End.mdx);

Customize the sidebar items generator

You can provide a custom `sidebarItemsGenerator` function in the docs plugin (or preset) config:

```
docusaurus.config.js
```



```
export default {
  plugins: [
    [
      '@docusaurus/plugin-content-docs',
      {
        async sidebarItemsGenerator({
          defaultSidebarItemsGenerator,
          numberPrefixParser,
          item,
          version,
          docs,
          categoriesMetadata,
          isCategoryIndex,
        }) {
          // Example: return an hardcoded list of static sidebar items
          return [
            {type: 'doc', id: 'doc1'},
            {type: 'doc', id: 'doc2'},
          ];
        },
      ],
    ],
  ],
};
```

TIP

Re-use and enhance the default generator instead of writing a generator from scratch: [the default generator we provide](#) is 250 lines long.

Add, update, filter, re-order the sidebar items according to your use case:

```
docusaurus.config.js
```

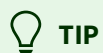


```
// Reverse the sidebar items ordering (including nested category items)
function reverseSidebarItems(items) {
  // Reverse items in categories
  const result = items.map((item) => {
    if (item.type === 'category') {
      return {...item, items: reverseSidebarItems(item.items)};
    }
    return item;
  });
  // Reverse items at current level
  result.reverse();
  return result;
}
```

```
export default {
  plugins: [
    [
      '@docusaurus/plugin-content-docs',
      {
        async sidebarItemsGenerator({defaultSidebarItemsGenerator,
...args}) {
          const sidebarItems = await
defaultSidebarItemsGenerator(args);
          return reverseSidebarItems(sidebarItems);
        },
      ],
    ],
  ],
};
```

1.3.8 Using multiple sidebars

You can create a sidebar for each **set of Markdown files** that you want to **group together**.



TIP

The Docusaurus site is a good example of using multiple sidebars:

- [Docs](#)
- [API](#)

Consider this example:

```
sidebars.js
```




```
export default {  
  tutorialSidebar: {  
    'Category A': ['doc1', 'doc2'],  
  },  
  apiSidebar: ['doc3', 'doc4'],  
};
```

When browsing `doc1` or `doc2`, the `tutorialSidebar` will be displayed; when browsing `doc3` or `doc4`, the `apiSidebar` will be displayed.

Understanding sidebar association

Following the example above, if a `commonDoc` is included in both sidebars:

sidebars.js

```
export default {  
  tutorialSidebar: {  
    'Category A': ['doc1', 'doc2', 'commonDoc'],  
  },  
  apiSidebar: ['doc3', 'doc4', 'commonDoc'],  
};
```

How does Docusaurus know which sidebar to display when browsing `commonDoc`? Answer: it doesn't, and we don't guarantee which sidebar it will pick.

When you add doc Y to sidebar X, it creates a two-way binding: sidebar X contains a link to doc Y, and when browsing doc Y, sidebar X will be displayed. But sometimes, we want to break either implicit binding:

1. *How do I generate a link to doc Y in sidebar X without making sidebar X displayed on Y?* For example, when I include doc Y in multiple sidebars as in the example above, and I want to explicitly tell Docusaurus to display one sidebar?
2. *How do I make sidebar X displayed when browsing doc Y, but sidebar X shouldn't contain the link to Y?* For example, when Y is a "doc home page" and the sidebar is purely used for navigation?

Front matter option `displayed_sidebar` will forcibly set the sidebar association. For the same example, you can still use doc shorthands without any special configuration:

sidebars.js



```
export default {  
  tutorialSidebar: {  
    'Category A': ['doc1', 'doc2'],  
  },  
  apiSidebar: ['doc3', 'doc4'],  
};
```

And then add a front matter:

commonDoc.md

```
---  
displayed_sidebar: apiSidebar  
---
```

Which explicitly tells Docusaurus to display `apiSidebar` when browsing `commonDoc`. Using the same method, you can make sidebar X which doesn't contain doc Y appear on doc Y:

home.md

```
---  
displayed_sidebar: tutorialSidebar  
---
```

Even when `tutorialSidebar` doesn't contain a link to `home`, it will still be displayed when viewing `home`.

If you set `displayed_sidebar: null`, no sidebar will be displayed whatsoever on this page, and subsequently, no pagination either.

Generating pagination

Docusaurus uses the sidebar to generate the "next" and "previous" pagination links at the bottom of each doc page. It strictly uses the sidebar that is displayed: if no sidebar is associated, it doesn't generate pagination either. However, the docs linked as "next" and "previous" are not guaranteed to display the same sidebar: they are included in this sidebar, but in their front matter, they may have a different `displayed_sidebar`.

If a sidebar is displayed by setting `displayed_sidebar` front matter, and this sidebar doesn't contain the doc itself, no pagination is displayed.



You can customize pagination with front matter `pagination_next` and `pagination_prev`. Consider this sidebar:

sidebars.js

```
export default {
  tutorial: [
    'introduction',
    {
      installation: ['windows', 'linux', 'macos'],
    },
    'getting-started',
  ],
};
```

The pagination next link on "windows" points to "linux", but that doesn't make sense: you would want readers to proceed to "getting started" after installation. In this case, you can set the pagination manually:

windows.md

```
---
pagination_next: getting-started
---

# Installation on Windows
```

You can also disable displaying a pagination link with `pagination_next: null` or `pagination_prev: null`.

The pagination label by default is the sidebar label. You can use the front matter `pagination_label` to customize how this doc appears in the pagination.

The `ref` item

The `ref` type is identical to the `doc` type in every way, except that it doesn't participate in generating navigation metadata. It only registers itself as a link. When [generating pagination](#) and [displaying sidebar](#), `ref` items are completely ignored.

It is particularly useful where you wish to link to the same document from multiple sidebars. The document only belongs to one sidebar (the one where it's registered as `type: 'doc'` or from an autogenerated directory), but its link will appear in all sidebars that it's registered in.

Consider this example:



sidebars.js

```
export default {
  tutorialSidebar: {
    'Category A': [
      'doc1',
      'doc2',
      {type: 'ref', id: 'commonDoc'},
      'doc5',
    ],
  },
  apiSidebar: ['doc3', 'doc4', 'commonDoc'],
};
```

You can think of the `ref` type as the equivalent to doing the following:

- Setting `displayed_sidebar: tutorialSidebar` for `commonDoc` (`ref` is ignored in sidebar association)
- Setting `pagination_next: doc5` for `doc2` and setting `pagination_prev: doc2` for `doc5` (`ref` is ignored in pagination generation)

1.3.9 Versioning

You can use the versioning CLI to create a new documentation version based on the latest content in the `docs` directory. That specific set of documentation will then be preserved and accessible even as the documentation in the `docs` directory continues to evolve.

⚠ WARNING

Think about it before starting to version your documentation - it can become difficult for contributors to help improve it!

Most of the time, you don't need versioning as it will just increase your build time, and introduce complexity to your codebase. Versioning is **best suited for websites with high-traffic and rapid changes to documentation between versions**. If your documentation rarely changes, don't add versioning to your documentation.

To better understand how versioning works and see if it suits your needs, you can read on below.

Overview

A typical versioned doc site looks like below:



```
website
├── sidebars.json      # sidebar for the current docs version
├── docs               # docs directory for the current docs version
│   ├── foo
│   │   └── bar.md     # https://mysite.com/docs/next/foo/bar
│   └── hello.md      # https://mysite.com/docs/next/hello
├── versions.json      # file to indicate what versions are available
├── versioned_docs
│   ├── version-1.1.0
│   │   ├── foo
│   │   │   └── bar.md # https://mysite.com/docs/foo/bar
│   │   └── hello.md
│   └── version-1.0.0
│       ├── foo
│       │   └── bar.md # https://mysite.com/docs/1.0.0/foo/bar
│       └── hello.md
├── versioned_sidebars
│   ├── version-1.1.0-sidebars.json
│   └── version-1.0.0-sidebars.json
├── docusaurus.config.js
└── package.json
```

The `versions.json` file is a list of version names, ordered from newest to oldest.

The table below explains how a versioned file maps to its version and the generated URL.

Path	Version	URL
<code>versioned_docs/version-1.0.0/hello.md</code>	1.0.0	<code>/docs/1.0.0/hello</code>
<code>versioned_docs/version-1.1.0/hello.md</code>	1.1.0 (latest)	<code>/docs/hello</code>
<code>docs/hello.md</code>	current	<code>/docs/next/hello</code>

TIP

The files in the `docs` directory belong to the `current` docs version.

By default, the `current` docs version is labeled as `Next` and hosted under `/docs/next/*`, but it is entirely configurable to fit your project's release lifecycle.

Terminology

Note the terminology we use here.

Current version

The version placed in the `./docs` folder.

Latest version / last version

The version served by default for docs navbar items. Usually has path `/docs`.



Current version is defined by the **file system location**, while latest version is defined by the **the navigation behavior**. They may or may not be the same version! (And the default configuration, as shown in the table above, would treat them as different: current version at `/docs/next` and latest at `/docs/`.)

Tutorials

Tagging a new version

1. First, make sure the current docs version (the `./docs` directory) is ready to be frozen.
2. Enter a new version number.

npm **Yarn** **pnpm** **Bun**

```
npm run docusaurus docs:version 1.1.0
```

When tagging a new version, the document versioning mechanism will:

- Copy the full `docs/` folder contents into a new `versioned_docs/version-[versionName]/` folder.
- Create a versioned sidebars file based from your current `sidebar` configuration (if it exists) - saved as `versioned_sidebars/version-[versionName]-sidebars.json`.
- Append the new version number to `versions.json`.

Creating new docs

1. Place the new file into the corresponding version folder.
2. Include the reference to the new file in the corresponding sidebar file according to the version number.

Current version structure **Older version structure**

```
# The new file.
docs/new.md

# Edit the corresponding sidebar file.
sidebars.js
```



TIP

Versioned sidebar files are, like standard sidebar files, relative to the content root for the given version — so for the example above, your versioned sidebar file may look like:



```
{
  "sidebar": [
    {
      "type": "autogenerated",
      "dirName": "."
    }
  ]
}
```

or for a manual sidebar:

```
{
  "sidebar": [
    {
      "type": "doc",
      "id": "new",
      "label": "New"
    }
  ]
}
```

Updating an existing version

You can update multiple docs versions at the same time because each directory in `versioned_docs/` represents specific routes when published.

1. Edit any file.
2. Commit and push changes.
3. It will be published to the version.

Example: When you change any file in `versioned_docs/version-2.6/`, it will only affect the docs for version `2.6`.

Deleting an existing version

You can delete/remove versions as well.

1. Remove the version from `versions.json`.

Example:



```
[  
  "2.0.0",  
  "1.9.0",  
  - "1.8.0"  
]
```

2. Delete the versioned docs directory. Example: `versioned_docs/version-1.8.0`.
3. Delete the versioned sidebars file. Example: `versioned_sidebars/version-1.8.0-sidebars.json`.

Configuring versioning behavior

The "current" version is the version name for the `./docs` folder. There are different ways to manage versioning, but two very common patterns are:

- You release v1, and start immediately working on v2 (including its docs). In this case, the **current version** is v2, which is in the `./docs` source folder, and can be browsed at `example.com/docs/next`. The **latest version** is v1, which is in the `./versioned_docs/version-1` source folder, and is browsed by most of your users at `example.com/docs`.
- You release v1, and will maintain it for some time before thinking about v2. In this case, the **current version** and **latest version** will both be point to v1, since the v2 docs doesn't even exist yet!

Docusaurus defaults work great for the first use case. We will label the current version as "next" and you can even choose not to publish it.

For the 2nd use case: if you release v1 and don't plan to work on v2 anytime soon, instead of versioning v1 and having to maintain the docs in 2 folders (`./docs` + `./versioned_docs/version-1.0.0`), you may consider "pretending" that the current version is a cut version by giving it a path and a label:

```
docusaurus.config.js
```




```
export default {
  presets: [
    '@docusaurus/preset-classic',
    docs: {
      lastVersion: 'current',
      versions: {
        current: {
          label: '1.0.0',
          path: '1.0.0',
        },
      },
    },
  ],
};
```

The docs in `./docs` will be served at `/docs/1.0.0` instead of `/docs/next`, and `1.0.0` will become the default version we link to in the navbar dropdown, and you will only need to maintain a single `./docs` folder.

We offer these plugin options to customize versioning behavior:

- `disableVersioning`: Explicitly disable versioning even with versions. This will make the site only include the current version.
- `includeCurrentVersion`: Include the current version (the `./docs` folder) of your docs.
 - **Tip**: turn it off if the current version is a work-in-progress, not ready to be published.
- `lastVersion`: Sets which version "latest version" (the `/docs` route) refers to.
 - **Tip**: `lastVersion: 'current'` makes sense if your current version refers to a major version that's constantly patched and released. The actual route base path and label of the latest version are configurable.
- `onlyIncludeVersions`: Defines a subset of versions from `versions.json` to be deployed.
 - **Tip**: limit to 2 or 3 versions in dev and deploy previews to improve startup and build time.
- `versions`: A dictionary of version metadata. For each version, you can customize the following:
 - `label`: the label displayed in the versions dropdown and banner.
 - `path`: the route base path of this version. By default, latest version has `/` and current version has `/next`.
 - `banner`: one of `'none'`, `'unreleased'`, and `'unmaintained'`. Determines what's displayed at the top of every doc page. Any version above the latest version would be "unreleased", and any version below would be "unmaintained".
 - `badge`: show a badge with the version name at the top of a doc of that version.
 - `className`: add a custom `className` to the `<html>` element of doc pages of that version.

See [docs plugin configuration](#) for more details.

Navbar items



We offer several docs navbar items to help you quickly set up navigation without worrying about versioned routes.

- `doc`: a link to a doc.
- `docSidebar`: a link to the first item in a sidebar.
- `docsVersion`: a link to the main doc of the currently viewed version.
- `docsVersionDropdown`: a dropdown containing all the versions available.

These links would all look for an appropriate version to link to, in the following order:

1. **Active version**: the version that the user is currently browsing, if she is on a page provided by this doc plugin. If she's not on a doc page, fall back to...
2. **Preferred version**: the version that the user last viewed. If there's no history, fall back to...
3. **Latest version**: the default version that we navigate to, configured by the `lastVersion` option.

docsVersionDropdown

By default, the `docsVersionDropdown` displays a dropdown with all the available docs versions.

The `versions` attribute allows you to display a subset of the available docs versions in a given order:

docusaurus.config.js

```
export default {
  themeConfig: {
    navbar: {
      items: [
        {
          type: 'docsVersionDropdown',
          versions: ['current', '3.0', '2.0'],
        },
      ],
    },
  },
};
```

Passing a `versions` object, lets you override the display label of each version:

docusaurus.config.js



```
export default {
  themeConfig: {
    navbar: {
      items: [
        {
          type: 'docsVersionDropdown',
          versions: {
            current: {label: 'Version 4.0'},
            '3.0': {label: 'Version 3.0'},
            '2.0': {label: 'Version 2.0'},
          },
        },
      ],
    },
  },
};
```

Recommended practices

Version your documentation only when needed

For example, you are building documentation for your npm package `foo` and you are currently in version 1.0.0. You then release a patch version for a minor bug fix and it's now 1.0.1.

Should you cut a new documentation version 1.0.1? **You probably shouldn't.** 1.0.1 and 1.0.0 docs shouldn't differ according to semver because there are no new features!. Cutting a new version for it will only just create unnecessary duplicated files.

Keep the number of versions small

As a good rule of thumb, try to keep the number of your versions below 10. You will **very likely** have a lot of obsolete versioned documentation that nobody even reads anymore. For example, [Jest](#) is currently in version `30.x`, however many older versions are archived but remain available, linking to immutable standalone deployment.



ARCHIVE OLDER VERSIONS

If you [deploy your site](#) on a static hosting provider, you can configure it to create a separate deployment at any point in time, and keep that deployment alive indefinitely as an immutable snapshot under a permanent URL. This means you can archive versioned docs as standalone immutable websites, and then link to these standalone websites. The [Jest](#), [React Native](#) and [Docusaurus](#) websites all use such pattern to keep the number of actively built versions low.

Use absolute import within the docs

Don't use relative paths import within the docs. Because when we cut a version the paths no longer work (the nesting level is different, among other reasons). You can utilize the `@site` alias provided by Docusaurus that points to the `website` directory. Example:



```
- import Foo from '../src/components/Foo';  
+ import Foo from '@site/src/components/Foo';
```

Link docs by file paths

Refer to other docs by relative file paths with the `.md` extension, so that Docusaurus can rewrite them to actual URL paths during building. Files will be linked to the correct corresponding version.

The `[@hello](hello.mdx#paginate)` document is great!

See the `[Tutorial](../getting-started/tutorial.mdx)` for more info.

Global or versioned collocated assets

You should decide if assets like images and files are per-version or shared between versions.

If your assets should be versioned, put them in the docs version, and use relative paths:

```
![img alt](./myImage.png)  
  
[download this file](./file.pdf)
```

If your assets are global, put them in `/static` and use absolute paths:

```
![img alt](/myImage.png)  
  
[download this file](/file.pdf)
```

1.3.10 Docs Multi-instance

The `@docusaurus/plugin-content-docs` plugin can support [multi-instance](#).

NOTE

This feature is only useful for [versioned documentation](#). It is recommended to be familiar with docs versioning before reading this page. If you just want [multiple sidebars](#), you can do so within one plugin.

Use-cases



Sometimes you want a Docusaurus site to host 2 distinct sets of documentation (or more).

These documentations may even have different versioning/release lifecycles.

Mobile SDKs documentation

If you build a cross-platform mobile SDK, you may have 2 documentations:

- Android SDK documentation (v1.0, v1.1)
- iOS SDK documentation (v1.0, v2.0)

In this case, you can use a distinct docs plugin instance per mobile SDK documentation.

WARNING

If each documentation instance is very large, you should rather create 2 distinct Docusaurus sites.

If someone edits the iOS documentation, is it really useful to rebuild everything, including the whole Android documentation that did not change?

Versioned and unversioned doc

Sometimes, you want some documents to be versioned, while other documents are more "global", and it feels useless to version them.

We use this pattern on the Docusaurus website itself:

- The `/docs/` section is versioned
- The `/community/` section is unversioned

Setup

Suppose you have 2 documentations:

- Product: some versioned doc about your product
- Community: some unversioned doc about the community around your product

In this case, you should use the same plugin twice in your site configuration.

WARNING

`@docusaurus/preset-classic` already includes a docs plugin instance for you!

When using the preset:

```
docusaurus.config.js
```



```
export default {
  presets: [
    [
      '@docusaurus/preset-classic',
      {
        docs: {
          // id: 'product', // omitted => default instance
          path: 'product',
          routeBasePath: 'product',
          sidebarPath: './sidebarsProduct.js',
          // ... other options
        },
      ],
    ],
  ],
  plugins: [
    [
      '@docusaurus/plugin-content-docs',
      {
        id: 'community',
        path: 'community',
        routeBasePath: 'community',
        sidebarPath: './sidebarsCommunity.js',
        // ... other options
      },
    ],
  ],
};
```

When not using the preset:

```
docusaurus.config.js
```



```
export default {
  plugins: [
    [
      '@docusaurus/plugin-content-docs',
      {
        // id: 'product', // omitted => default instance
        path: 'product',
        routeBasePath: 'product',
        sidebarPath: './sidebarsProduct.js',
        // ... other options
      },
    ],
    [
      '@docusaurus/plugin-content-docs',
      {
        id: 'community',
        path: 'community',
        routeBasePath: 'community',
        sidebarPath: './sidebarsCommunity.js',
        // ... other options
      },
    ],
  ],
};
```

Don't forget to assign a unique `id` attribute to plugin instances.

i NOTE

We consider that the `product` instance is the most important one, and make it the "default" instance by not assigning any ID.

Versioned paths

Each plugin instance will store versioned docs in a distinct folder.

The default plugin instance will use these paths:

- `website/versions.json`
- `website/versioned_docs`
- `website/versioned_sidebars`

The other plugin instances (with an `id` attribute) will use these paths:

- `website/[pluginId]_versions.json`
- `website/[pluginId]_versioned_docs`



- `website/[pluginId]_versioned_sidebars`

**TIP**

You can omit the `id` attribute (defaults to `default`) for one of the docs plugin instances.

The instance paths will be simpler, and retro-compatible with a single-instance setup.

Tagging new versions

Each plugin instance will have its own CLI command to tag a new version. They will be displayed if you run:

npm Yarn pnpm Bun

```
npm run docusaurus -- --help
```

To version the product/default docs plugin instance:

npm Yarn pnpm Bun

```
npm run docusaurus docs:version 1.0.0
```

To version the non-default/community docs plugin instance:

npm Yarn pnpm Bun

```
npm run docusaurus docs:version:community 1.0.0
```

Docs navbar items

Each docs-related [theme navbar items](#) take an optional `docsPluginId` attribute.

For example, if you want to have one version dropdown for each mobile SDK (iOS and Android), you could do:



docusaurus.config.js

```
export default {
  themeConfig: {
    navbar: {
      items: [
        {
          type: 'docsVersionDropdown',
          docsPluginId: 'ios',
        },
        {
          type: 'docsVersionDropdown',
          docsPluginId: 'android',
        },
      ],
    },
  },
};
```

1.3.11 Blog

The blog feature enables you to deploy a full-featured blog in no time.

! INFO

Check the [Blog Plugin API Reference documentation](#) for an exhaustive list of options.

Initial setup

To set up your site's blog, start by creating a `blog` directory.

Then, add an item link to your blog within `docusaurus.config.js`:

docusaurus.config.js



```
export default {
  themeConfig: {
    // ...
    navbar: {
      items: [
        // ...
        {to: 'blog', label: 'Blog', position: 'left'}, // or position:
        'right'
      ],
    },
  },
};
```

Adding posts

To publish in the blog, create a Markdown file within the blog directory.

For example, create a file at `website/blog/2019-09-05-hello-docusaurus.md`:

```
website/blog/2019-09-05-hello-docusaurus.md
```



```
---
title: Welcome Docusaurus
description: This is my first post on Docusaurus.
slug: welcome-docusaurus-v2
authors:
  - name: Joel Marcey
    title: Co-creator of Docusaurus 1
    url: https://github.com/JoelMarcey
    image_url: https://github.com/JoelMarcey.png
    socials:
      x: joelmarcey
      github: JoelMarcey
  - name: Sébastien Lorber
    title: Docusaurus maintainer
    url: https://sebastienlorber.com
    image_url: https://github.com/slorber.png
    socials:
      x: sebastienlorber
      github: slorber
      email: seb@example.com
tags: [hello, docusaurus-v2]
image: https://i.imgur.com/mErPwqL.png
hide_table_of_contents: false
---
```

Welcome to this blog. This blog is created with `[**Docusaurus**]`
(<https://docusaurus.io/>).

`<!-- truncate -->`

This is my first post on Docusaurus.

A whole bunch of exploration to follow.

The [front matter](#) is useful to add more metadata to your blog post, for example, author information, but Docusaurus will be able to infer all necessary metadata without the front matter. For all possible fields, see [the API documentation](#).

Blog list

The blog's index page (by default, it is at `/blog`) is the *blog list page*, where all blog posts are collectively displayed.

Use the `<!--truncate-->` marker in your blog post to represent what will be shown as the summary when viewing all published blog posts. Anything above `<!--truncate-->` will be part of the summary. Note that the portion above the truncate marker must be standalone renderable Markdown. For example:



website/blog/my-post.md

```
---
title: Markdown blog truncation example
---

All these will be part of the blog post summary.

<!-- truncate -->

But anything from here on down will not be.
```

For files using the `.mdx` extension, use a [MDX](#) comment `{/* truncate */}` instead:

website/blog/my-post.mdx

```
---
title: MDX blog truncation Example
---

All these will be part of the blog post summary.

{/* truncate */}

But anything from here on down will not be.
```

By default, 10 posts are shown on each blog list page, but you can control pagination with the `postsPerPage` option in the plugin configuration. If you set `postsPerPage: 'ALL'`, pagination will be disabled and all posts will be displayed on the first page. You can also add a meta description to the blog list page for better SEO:

docusaurus.config.js



```
export default {
  // ...
  presets: [
    [
      '@docusaurus/preset-classic',
      {
        blog: {
          blogTitle: 'Docusaurus blog!',
          blogDescription: 'A Docusaurus powered blog!',
          postsPerPage: 'ALL',
        },
      },
    ],
  ],
};
```

Blog sidebar

The blog sidebar displays recent blog posts. The default number of items shown is 5, but you can customize with the `blogSidebarCount` option in the plugin configuration. By setting `blogSidebarCount: 0`, the sidebar will be completely disabled, with the container removed as well. This will increase the width of the main container. Specially, if you have set `blogSidebarCount: 'ALL'`, *all* posts will be displayed.

You can also alter the sidebar heading text with the `blogSidebarTitle` option. For example, if you have set `blogSidebarCount: 'ALL'`, instead of the default "Recent posts", you may rather make it say "All posts":

docusaurus.config.js

```
export default {
  presets: [
    [
      '@docusaurus/preset-classic',
      {
        blog: {
          blogSidebarTitle: 'All posts',
          blogSidebarCount: 'ALL',
        },
      },
    ],
  ],
};
```

Blog post date



Docusaurus will extract a `YYYY-MM-DD` date from many patterns such as `YYYY-MM-DD-my-blog-post-title.md` or `YYYY/MM/DD/my-blog-post-title.md`. This enables you to easily group blog posts by year, by month, or to use a flat structure.

Supported date extraction patterns

Pattern	Example
Single file	<code>2021-05-28-my-blog-post-title.md</code>
MDX file	<code>2021-05-28-my-blog-post-title.mdx</code>
Single folder + <code>index.md</code>	<code>2021-05-28-my-blog-post-title/index.md</code>
Folder named by date	<code>2021-05-28/my-blog-post-title.md</code>
Nested folders by date	<code>2021/05/28/my-blog-post-title.md</code>
Partially nested folders by date	<code>2021/05-28-my-blog-post-title.md</code>
Nested folders + <code>index.md</code>	<code>2021/05/28/my-blog-post-title/index.md</code>
Date in the middle of path	<code>category/2021/05-28-my-blog-post-title.md</code>

Docusaurus can extract the date from the posts using any of the naming patterns above. It is advisable to choose one pattern and apply it to all posts to avoid confusion.

TIP

Using a folder can be convenient to co-locate blog post images alongside the Markdown file.

This naming convention is optional, and you can also provide the date as front matter. Since the front matter follows YAML syntax where the datetime notation is supported, you can use front matter if you need more fine-grained publish dates. For example, if you have multiple posts published on the same day, you can order them according to the time of the day:

earlier-post.md

```
---
date: 2021-09-13T10:00
---
```

later-post.md

```
---
date: 2021-09-13T18:00
---
```



Blog post authors

Use the `authors` front matter field to declare blog post authors. An author should have at least a `name` or an `image_url`. Docusaurus uses information like `url`, `email`, and `title`, but any other information is allowed.

Inline authors

Blog post authors can be declared directly inside the front matter:

Single author

Multiple authors

my-blog-post.md

```
---
authors:
  name: Joel Marcey
  title: Co-creator of Docusaurus 1
  url: https://github.com/JoelMarcey
  image_url: https://github.com/JoelMarcey.png
  email: jimarcey@gmail.com
  socials:
    x: joelmarcey
    github: JoelMarcey
---
```

TIP

This option works best to get started, or for casual, irregular authors.

INFO

Prefer using the `authors` front matter, but the legacy `author_*` front matter remains supported:

my-blog-post.md



```
---
author: Joel Marcey
author_title: Co-creator of Docusaurus 1
author_url: https://github.com/JoelMarcey
author_image_url: https://github.com/JoelMarcey.png
---
```

Global authors

For regular blog post authors, it can be tedious to maintain authors' information inlined in each blog post.

It is possible to declare those authors globally in a configuration file:

website/blog/authors.yml

```
jmarcey:
  name: Joel Marcey
  title: Co-creator of Docusaurus 1
  url: https://github.com/JoelMarcey
  image_url: https://github.com/JoelMarcey.png
  email: jimarcey@gmail.com
  socials:
    x: joelmarcey
    github: JoelMarcey

slorber:
  name: Sébastien Lorber
  title: Docusaurus maintainer
  url: https://sebastienlorber.com
  image_url: https://github.com/slorber.png
  socials:
    x: sebastienlorber
    github: slorber
    email: seb@example.com
```



TIP

Use the `authorsMapPath` plugin option to configure the path. JSON is also supported.

In blog posts front matter, you can reference the authors declared in the global configuration file:

Single author

Multiple authors



my-blog-post.md

```
---
authors: jmarcey
---
```

! INFO

The `authors` system is very flexible and can suit more advanced use-case:

Mix inline authors and global authors

You can use global authors most of the time, and still use inline authors:

my-blog-post.md

```
---
authors:
  - jmarcey
  - slorber
  - name: Inline Author name
    title: Inline Author Title
    url: https://github.com/inlineAuthor
    image_url: https://github.com/inlineAuthor
---
```

Local override of global authors

You can customize the global author's data on per-blog-post basis:

my-blog-post.md



```
---
authors:
  - key: jmarcey
    title: Joel Marcey's new title
  - key: slorber
    name: Sébastien Lorber's new name
---
```

Localize the author's configuration file

The configuration file can be localized, just create a localized copy of it at:

```
website/i18n/[locale]/docusaurus-plugin-content-blog/authors.yml
```

An author, either declared through front matter or through the authors map, needs to have a name or an avatar, or both. If all authors of a post don't have names, Docusaurus will display their avatars compactly. See [this test post](#) for the effect.

FEED GENERATION

[RSS feeds](#) require the author's email to be set for the author to appear in the feed.

Authors pages

The authors pages feature is optional, and mainly useful for multi-author blogs.

You can activate it independently for each author by adding a `page: true` attribute to the [global author configuration](#):

```
website/blog/authors.yml
```



```
slorber:
  name: Sébastien Lorber
  page: true # Turns the feature on - route will be /authors/slorber

jmarcey:
  name: Joel Marcey
  page:
    # Turns the feature on - route will be /authors/custom-author-url
  permalink: '/custom-author-url'
```

The blog plugin will now generate:

- a dedicated author page for each author ([example](#)) listing all the blog posts they contributed to
- an authors index page ([example](#)) listing all these authors, in the order they appear in `authors.yml`

ABOUT INLINE AUTHORS

Only [global authors](#) can activate this feature. [Inline authors](#) are not supported.

Blog post tags

Tags are declared in the front matter and introduce another dimension of categorization.

It is possible to define tags inline, or to reference predefined tags declared in a [tags file](#) (optional, usually `blog/tags.yml`).

In the following example:

- `docusaurus` references a predefined tag key declared in `blog/tags.yml`
- `Releases` is an inline tag, because it does not exist in `blog/tags.yml`

blog/my-post.md

```
---
title: 'My blog post'
tags:
  - Releases
  - docusaurus
---
```

Content



blog/tags.yml

```
docusaurus:
  label: 'Docusaurus'
  permalink: '/docusaurus'
  description: 'Blog posts related to the Docusaurus framework'
```

Reading time

Docusaurus generates a reading time estimation for each blog post based on word count. We provide an option to customize this.

docusaurus.config.js

```
export default {
  presets: [
    [
      '@docusaurus/preset-classic',
      {
        blog: {
          showReadingTime: true, // When set to false, the "x min read"
                                // won't be shown
          readingTime: ({content, locale, frontMatter, defaultReadingTime})
            => defaultReadingTime({
              content,
              locale,
              options: {wordsPerMinute: 300},
            }),
        },
      ],
    ],
  ],
};
```

The `readingTime` callback receives the following parameters:

- `content`: the blog content text as a string
- `frontMatter`: the front matter as a record of string keys and their values
- `locale`: the locale of the current Docusaurus site
- `defaultReadingTime`: the default built-in reading time function. It returns a number (reading time in minutes) or `undefined` (disable reading time for this page).

The default reading time is able to accept additional options:



- `wordsPerMinute` as a number (default: 300)

**TIP**

Use the callback for all your customization needs:

[Per-post disabling](#)[Passing options](#)[Using custom algorithms](#)**Disable reading time on one page:**`docusaurus.config.js`

```
export default {
  presets: [
    [
      '@docusaurus/preset-classic',
      {
        blog: {
          showReadingTime: true,
          readingTime: ({content, locale, frontMatter,
            defaultReadingTime}) =>
            frontMatter.hide_reading_time
              ? undefined
              : defaultReadingTime({content, locale}),
        },
      ],
    ],
  ],
};
```

Usage:

```
---
hide_reading_time: true
---
```

This page will no longer display the reading time stats!

Feed

You can generate RSS / Atom / JSON feed by passing `feedOptions`. By default, RSS and Atom feeds are generated. To disable feed generation, set `feedOptions.type` to `null`.



```
type FeedType = 'rss' | 'atom' | 'json';

type BlogOptions = {
  feedOptions?: {
    type?: FeedType | 'all' | FeedType[] | null;
    title?: string;
    description?: string;
    copyright: string;

    language?: string; // possible values: http://www.w3.org/TR/REC-  
html40/struct/dirlang.html#langcodes
    limit?: number | false | null; // defaults to 20
    // XSLT permits browsers to style and render nicely the feed XML files
    xslt?:
      | boolean
      | {
          //
          rss?: string | boolean;
          atom?: string | boolean;
        };
    // Allow control over the construction of BlogFeedItems
    createFeedItems?: (params: {
      blogPosts: BlogPost[];
      siteConfig: DocusaurusConfig;
      outDir: string;
      defaultCreateFeedItems: (params: {
        blogPosts: BlogPost[];
        siteConfig: DocusaurusConfig;
        outDir: string;
      }) => Promise<BlogFeedItem[]>;
    }) => Promise<BlogFeedItem[]>;
  };
};
```

Example usage:

```
docusaurus.config.js
```



```
export default {
  // ...
  presets: [
    [
      '@docusaurus/preset-classic',
      {
        blog: {
          feedOptions: {
            type: 'all',
            copyright: `Copyright © ${new Date().getFullYear()} Facebook,
Inc.`,
            createFeedItems: async (params) => {
              const {blogPosts, defaultCreateFeedItems, ...rest} = params;
              return defaultCreateFeedItems({
                // keep only the 10 most recent blog posts in the feed
                blogPosts: blogPosts.filter((item, index) => index < 10),
                ...rest,
              });
            },
          },
        },
      ],
    ],
  ],
};
```

The feeds can be found at:

RSS **Atom** **JSON**

<https://example.com/blog/rss.xml>

Advanced topics

Blog-only mode

You can run your Docusaurus site without a dedicated landing page and instead have your blog's post list page as the index page. Set the `routeBasePath` to be `'/'` to serve the blog through the root route `example.com/` instead of the subroute `example.com/blog/`.

`docusaurus.config.js`



```
export default {  
  // ...  
  presets: [  
    [  
      '@docusaurus/preset-classic',  
      {  
        docs: false, // Optional: disable the docs plugin  
        blog: {  
          routeBasePath: '/', // Serve the blog at the site's root  
          /* other blog options */  
        },  
      },  
    ],  
  ],  
};
```

⚠ WARNING

Don't forget to delete the existing homepage at `./src/pages/index.js` or else there will be two files mapping to the same route!

⚠ WARNING

If you disable the docs plugin, don't forget to delete references to the docs plugin elsewhere in your configuration file. Notably, make sure to remove the docs-related navbar items.

💡 TIP

There's also a "Docs-only mode" for those who only want to use the docs. Read [Docs-only mode](#) for detailed instructions or a more elaborate explanation of `routeBasePath`.

Multiple blogs

By default, the classic theme assumes only one blog per website and hence includes only one instance of the blog plugin. If you would like to have multiple blogs on a single website, it's possible too! You can add another blog by specifying another blog plugin in the `plugins` option for `docusaurus.config.js`.

Set the `routeBasePath` to the URL route that you want your second blog to be accessed on. Note that the `routeBasePath` here has to be different from the first blog or else there could be a collision of paths! Also, set `path` to the path to the directory containing your second blog's entries.

As documented for [multi-instance plugins](#), you need to assign a unique ID to the plugins.

```
docusaurus.config.js
```




```
export default {
  // ...
  plugins: [
    [
      '@docusaurus/plugin-content-blog',
      {
        /**
         * Required for any multi-instance plugin
         */
        id: 'second-blog',
        /**
         * URL route for the blog section of your site.
         * *DO NOT* include a trailing slash.
         */
        routeBasePath: 'my-second-blog',
        /**
         * Path to data on filesystem relative to site dir.
         */
        path: './my-second-blog',
      },
    ],
  ],
};
```

As an example, we host a second blog [here](#).

1.3.12 Markdown Features

Docusaurus uses **Markdown** as its main content authoring format.



LEARN MARKDOWN

You can [learn Markdown in 10 minutes](#).

Docusaurus uses modern tooling to help you create **interactive documentation**.

The **MDX** compiler transforms **Markdown files to React components**, and allows you to use JSX in your Markdown content. This enables you to easily interleave React components within your content, and create delightful learning experiences.



USE THE MDX PLAYGROUND

The **MDX playground** is your new best friend!

It is a very helpful debugging tool that shows how the MDX compiler transforms Markdown to React.



Options: select the right format (MDX or CommonMark) and the following plugins Docusaurus uses: `remark-gfm`, `remark-directive`, `rehype-raw`.

MDX vs. CommonMark

Docusaurus compiles both `.md` and `.mdx` files to React components using the MDX compiler, but **the syntax can be interpreted differently** depending on your settings.

The MDX compiler supports [2 formats](#):

- The [MDX format](#): a powerful parser allowing the usage of JSX
- The [CommonMark format](#): a standard-compliant Markdown parser that does not allow the usage of JSX

By default, **Docusaurus v3 uses the MDX format for all files** (including `.md` files) for historical reasons.

It is possible to **opt-in for CommonMark** using the `siteConfig.markdown.format` setting or the `mdx.format: md` front matter.



HOW TO USE COMMONMARK

If you plan to use CommonMark, we recommend the `siteConfig.markdown.format: 'detect'` setting. The appropriate format will be selected automatically, based on file extensions:

- `.md` files will use the CommonMark format
- `.mdx` files will use the MDX format



EXPERIMENTAL COMMONMARK SUPPORT

The CommonMark support is **experimental** and currently has a few [limitations](#).

Standard features

Markdown is a syntax that enables you to write formatted content in a readable syntax.

```
### My Doc Section
```

```
Hello world message with some bold text, some italic text, and a  
[link]()
```

```
![img alt]()img/docusaurus.png
```



http://localhost:3000





My Doc Section

Hello world message with some **bold** text, some *italic* text and a [link](#)



Markdown is declarative

Some may assume a 1-1 correlation between Markdown and HTML, e.g., `! [Preview] (/img/docusaurus.png)` will always become ``, as-is. However, *that is not the case*.

The Markdown syntax `! [message] (url)` only declaratively tells Docusaurus that an image needs to be inserted here, but we may do other things like transforming a [file path to URL path](#), so the generated markup may differ from the output of other Markdown renderers, or a naïve hand-transcription to the equivalent JSX/HTML code.

In general, you should only assume the *semantics* of the markup (``` fences become [code blocks](#); `>` becomes [quotes](#), etc.), but not the actual compiled output.

Front matter

Front matter is used to add metadata to your Markdown file. All content plugins have their own front matter schema, and use the front matter to enrich the default metadata inferred from the content or other configuration.

Front matter is provided at the very top of the file, enclosed by three dashes `---`. The content is parsed as [YAML](#).



```
---
title: My Doc Title
more_data:
  - Can be provided
  - as: objects
  or: arrays
---
```

! INFO

The API documentation of each official plugin lists the supported attributes:

- [Docs front matter](#)
- [Blog front matter](#)
- [Pages front matter](#)



ENHANCE YOUR FRONT MATTER

Use the [Markdown config](#) `parseFrontMatter` function to provide your own front matter parser, or to enhance the default parser.

It is possible to reuse the default parser to wrap it with your own custom proprietary logic. This makes it possible to implement convenient front matter transformations, shortcuts, or to integrate with external systems using front matter that Docusaurus plugins do not support.

```
docusaurus.config.js
```



```
export default {
  markdown: {
    parseFrontMatter: async (params) => {
      // Reuse the default parser
      const result = await params.defaultParseFrontMatter(params);

      // Process front matter description placeholders
      result.frontMatter.description =
        result.frontMatter.description?.replaceAll('{{MY_VAR}}',
'MY_VALUE');

      // Create your own front matter shortcut
      if (result.frontMatter.i_do_not_want_docs_pagination) {
        result.frontMatter.pagination_prev = null;
        result.frontMatter.pagination_next = null;
      }

      // Rename an unsupported front matter coming from another system
      if (result.frontMatter.cms_seo_summary) {
        result.frontMatter.description =
result.frontMatter.cms_seo_summary;
        delete result.frontMatter.cms_seo_summary;
      }

      return result;
    },
  },
};
```

Quotes

Markdown quotes are beautifully styled:

```
> Easy to maintain open source documentation websites.
>
> – Docusaurus
```



http://localhost:3000



Easy to maintain open source documentation websites.



Details

Markdown can embed HTML elements, and `details` HTML elements are beautifully styled:

Details element example

```
<details>
  <summary>Toggle me!</summary>

  This is the detailed content

  ```js
 console.log("Markdown features including the code block are available");
  ```

  You can use Markdown here including bold and italic text, and
  [inline link](https://docusaurus.io)
  <details>
    <summary>Nested toggle! Some surprise inside...</summary>

    🤔🤔🤔🤔🤔
  </details>
</details>
```

Details element example

Toggle me!

This is the detailed content

```
console.log("Markdown features including the code block are
available");
```

You can use Markdown here including **bold** and *italic* text, and inline link



Nested toggle! Some surprise inside...



! INFO

You may want to keep your `<summary>` on a single line. Keep in mind that MDX creates extra HTML `<p>` paragraphs for line breaks.. When in doubt, use the [MDX playground](#) to troubleshoot `<details>` rendering problems.

Tables

Markdown tables are supported via [GitHub Flavored Markdown \(GFM\)](#). They are an excellent way to display structured data in your documentation.

Feature	Support	Description
GFM Tables	✓ Yes	Standard Markdown table syntax
Alignment	✓ Yes	Left, center, and right alignment



http://localhost:3000



Feature	Support	Description
GFM Tables	✓ Yes	Standard Markdown table syntax
Alignment	✓ Yes	Left, center, and right alignment

1.3.13 MDX and React

Docusaurus has built-in support for [MDX](#), which allows you to write JSX within your Markdown files and render them as React components.

Check out the [MDX docs](#) to see what fancy stuff you can do with MDX.



DEBUGGING MDX

The MDX format is quite strict, and you may get compilation errors.



Use the [MDX playground](#) to debug them and make sure your syntax is valid.

! INFO

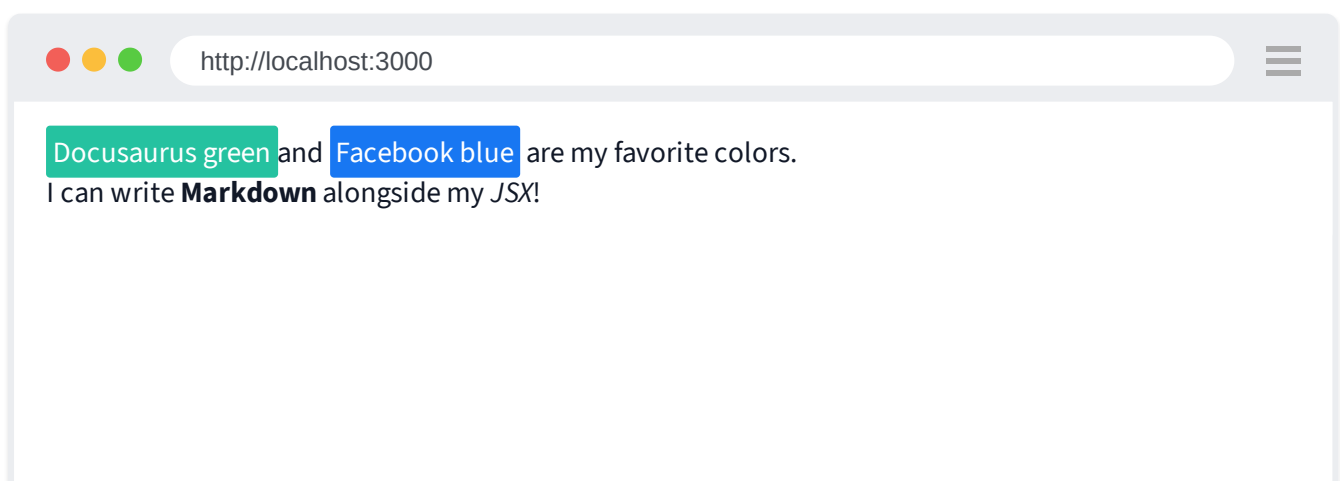
Prettier, the most popular formatter, [supports only the legacy MDX v1](#). If you get an unintentional formatting result, you may want to add `{/* prettier-ignore */}` before the problematic area, or add `*.mdx` to your `.prettiignore`, until Prettier has proper support for MDX v3. [One of the main authors of MDX recommends remark-cli with remark-mdx](#).

Exporting components

To define any custom component within an MDX file, you have to export it: only paragraphs that start with `export` will be parsed as components instead of prose.

```
export const Highlight = ({children, color}) => (  
  <span  
    style={{  
      backgroundColor: color,  
      borderRadius: '2px',  
      color: '#fff',  
      padding: '0.2rem',  
    }}>  
    {children}  
  </span>  
)  
;  
  
<Highlight color="#25c2a0">Docusaurus green</Highlight> and <Highlight  
color="#1877F2">Facebook blue</Highlight> are my favorite colors.  
  
I can write Markdown alongside my _JSX_!
```

Notice how it renders both the markup from your React component and the Markdown syntax:





! MDX IS JSX

Since all doc files are parsed using MDX, anything that looks like HTML is actually JSX. Therefore, if you need to inline-style a component, follow JSX flavor and provide style objects.

```
/* Instead of this: */
<span style="background-color: red">Foo</span>
/* Use this: */
<span style={{backgroundColor: 'red'}}>Foo</span>
```

Importing components

You can also import your own components defined in other files or third-party components installed via npm.

```
<!-- Docusaurus theme component -->
import TOCInline from '@theme/TOCInline';
<!-- External component -->
import Button from '@mui/material/Button';
<!-- Custom component -->
import BrowserWindow from '@site/src/components/BrowserWindow';
```

💡 TIP

The `@site` alias points to your website's directory, usually where the `docusaurus.config.js` file is. Using an alias instead of relative paths (`'../src/components/BrowserWindow'`) saves you from updating import paths when moving files around, or when [versioning docs](#) and [translating](#).

While declaring components within Markdown is very convenient for simple cases, it becomes hard to maintain because of limited editor support, risks of parsing errors, and low reusability. Use a separate `.js` file when your component involves complex JS logic:

```
src/components/Highlight.js
```



```
import React from 'react';

export default function Highlight({children, color}) {
  return (
    <span
      style={{
        backgroundColor: color,
        borderRadius: '2px',
        color: '#fff',
        padding: '0.2rem',
      }}>
      {children}
    </span>
  );
}
```

markdown-file.mdx

```
import Highlight from '@site/src/components/Highlight';

<Highlight color="#25c2a0">Docusaurus green</Highlight>
```

TIP

If you use the same component across a lot of files, you don't need to import it everywhere—consider adding it to the global scope. [See below](#)

MDX component scope

Apart from [importing a component](#) and [exporting a component](#), a third way to use a component in MDX is to **register it to the global scope**, which will make it automatically available in every MDX file, without any import statements.

For example, given this MDX file:

- a
- list!

And some `<Highlight>`custom markup`</Highlight>`...



It will be compiled to a React component containing `ul`, `li`, `p`, and `Highlight` elements. `Highlight` is not a native html element: you need to provide your own React component implementation for it.

In Docusaurus, the MDX component scope is provided by the `@theme/MDXComponents` file. It's not a React component, *per se*, unlike most other exports under the `@theme/` alias: it is a record from tag names like `Highlight` to their React component implementations.

If you [swizzle](#) this component, you will find all tags that have been implemented, and you can further customize our implementation by swizzling the respective sub-component, like `@theme/MDXComponents/Code` (which is used to render [Markdown code blocks](#)).

If you want to register extra tag names (like the `<Highlight>` tag above), you should consider [wrapping](#) `@theme/MDXComponents`, so you don't have to maintain all the existing mappings. Since the swizzle CLI doesn't allow wrapping non-component files yet, you should manually create the wrapper:

src/theme/MDXComponents.js

```
import React from 'react';
// Import the original mapper
import MDXComponents from '@theme-original/MDXComponents';
import Highlight from '@site/src/components/Highlight';

export default {
  // Re-use the default mapping
  ...MDXComponents,
  // Map the "<Highlight>" tag to our Highlight component
  // `Highlight` will receive all props that were passed to `<Highlight>`
  in MDX
  Highlight,
};
```

And now, you can freely use `<Highlight>` in every page, without writing the import statement:

I can conveniently use `<Highlight color="#25c2a0">Docusaurus green</Highlight>` everywhere!



http://localhost:3000



I can conveniently use `Docusaurus green` everywhere!



⚠ WARNING

We use **upper-case** tag names like `Highlight` on purpose.

From MDX v3+ onward (Docusaurus v3+), lower-case tag names are always rendered as native html elements, and will not use any component mapping you provide.

⚠ WARNING

This feature is powered by an `MDXProvider`. If you are importing Markdown in a React page, you have to supply this provider yourself through the `MDXContent` theme component.

src/pages/index.js

```
import React from 'react';
import FeatureDisplay from './_featureDisplay.mdx';
import MDXContent from '@theme/MDXContent';

export default function LandingPage() {
  return (
    <div>
      <MDXContent>
        <FeatureDisplay />
      </MDXContent>
    </div>
  );
}
```

If you don't wrap your imported MDX with `MDXContent`, the global scope will not be available.

Markdown and JSX interoperability

Docusaurus v3 is using [MDX v3](#).

The [MDX syntax](#) is mostly compatible with [CommonMark](#), but is much stricter because your `.mdx` files can use JSX and are compiled into real React components (check the [playground](#)).

Some valid CommonMark features won't work with MDX ([more info](#)), notably:

- Indented code blocks: use triple backticks instead
- Autolinks (`<http://localhost:3000>`): use regular link syntax instead (`[http://localhost:3000](http://localhost:3000)`)
- HTML syntax (`<p style="color: red;">`): use JSX instead (`<p style={{color: 'red'}}>`)
- Unescaped `{` and `<`: escape them with `\` instead (`\{` and `\<`)



EXPERIMENTAL COMMONMARK SUPPORT

Docusaurus v3 makes it possible to opt-in for a less strict, standard [CommonMark](#) support with the following options:

- The `mdx.format: md` front matter
- The `.md` file extension combined with the `siteConfig.markdown.format: "detect"` configuration

This feature is **experimental** and currently has a few [limitations](#).

Importing code snippets

You can not only import a file containing a component definition, but also import any code file as raw text, and then insert it in a code block, thanks to [Webpack raw-loader](#). In order to use `raw-loader`, you first need to install it in your project:

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)

```
npm install --save raw-loader
```

Now you can import code snippets from another file as it is:

myMarkdownFile.mdx

```
import CodeBlock from '@theme/CodeBlock';
import MyComponentSource from '!!raw-loader!./myComponent';

<CodeBlock language="jsx">{MyComponentSource}</CodeBlock>
```



http://localhost:3000





```
/**
 * Copyright (c) Facebook, Inc. and its affiliates.
 *
 * This source code is licensed under the MIT license found in the
 * LICENSE file in the root directory of this source tree.
 */

import React, {useState} from 'react';

export default function MyComponent() {
  const [bool, setBool] = useState(false);
  return (
    <div>
      <p>MyComponent rendered !</p>
      <p>bool={bool ? 'true' : 'false'}</p>
      <p>
        <button onClick={() => setBool((b) => !b)}>toggle
bool</button>
      </p>
    </div>
  );
}
```

See [using code blocks in JSX](#) for more details of the `<CodeBlock>` component.

NOTE

You have to use `<CodeBlock>` rather than the Markdown triple-backtick `````, because the latter will ship out any of its content as-is, but you want to interpolate the imported text here.

WARNING

This feature is experimental and might be subject to breaking API changes in the future.

Importing Markdown

You can use Markdown files as components and import them elsewhere, either in Markdown files or in React pages. Each MDX file default-exports its page content as a React component. In the `import` statement, you can default-import this component with any name, but it must be capitalized following React's naming rules.

By convention, using the `_` **filename prefix** will not create any doc page and means the Markdown file is a **"partial"**, to be imported by other files.



_markdown-partial-example.mdx

```
<span>Hello {props.name}</span>
```

This is text some content from `\_markdown-partial-example.mdx`.

someOtherDoc.mdx

```
import PartialExample from './_markdown-partial-example.mdx';
```

```
<PartialExample name="Sebastien" />
```



http://localhost:3000



Hello Sebastien

This is text some content from `_markdown-partial-example.md`.

This way, you can reuse content among multiple pages and avoid duplicating materials.

Available exports

Within the MDX page, the following variables are available as globals:

- `frontMatter`: the front matter as a record of string keys and values;
- `toc`: the table of contents, as a tree of headings. See also [Inline TOC](#) for a more concrete use-case.
- `contentTitle`: the Markdown title, which is the first `h1` heading in the Markdown text. It's `undefined` if there isn't one (e.g. title specified in the front matter).



```
import TOCInline from '@theme/TOCInline';  
import CodeBlock from '@theme/CodeBlock';
```

The table of contents for this page, serialized:

```
<CodeBlock className="language-json">{JSON.stringify(toc, null, 2)}  
</CodeBlock>
```

The front matter of this page:

```
<ul>  
  {Object.entries(frontMatter).map(([key, value]) => <li key={key}><b>{key}</b>: {value}</li>)}  
</ul>
```

```
<p>The title of this page is: <b>{contentTitle}</b></p>
```



http://localhost:3000



The table of contents for this page, serialized:



```
[
  {
    "value": "Exporting components",
    "id": "exporting-components",
    "level": 3
  },
  {
    "value": "Importing components",
    "id": "importing-components",
    "level": 3
  },
  {
    "value": "MDX component scope",
    "id": "mdx-component-scope",
    "level": 3
  },
  {
    "value": "Markdown and JSX interoperability",
    "id": "markdown-and-jsx-interoperability",
    "level": 3
  },
  {
    "value": "Importing code snippets",
    "id": "importing-code-snippets",
    "level": 2
  },
  {
    "value": "Importing Markdown",
    "id": "importing-markdown",
    "level": 2
  },
  {
    "value": "Available exports",
    "id": "available-exports",
    "level": 2
  }
]
```

The front matter of this page:

- **id:** react
- **description:** Using the power of React in Docusaurus Markdown documents, thanks to MDX
- **slug:** /markdown-features/react

The title of this page is: **MDX and React**

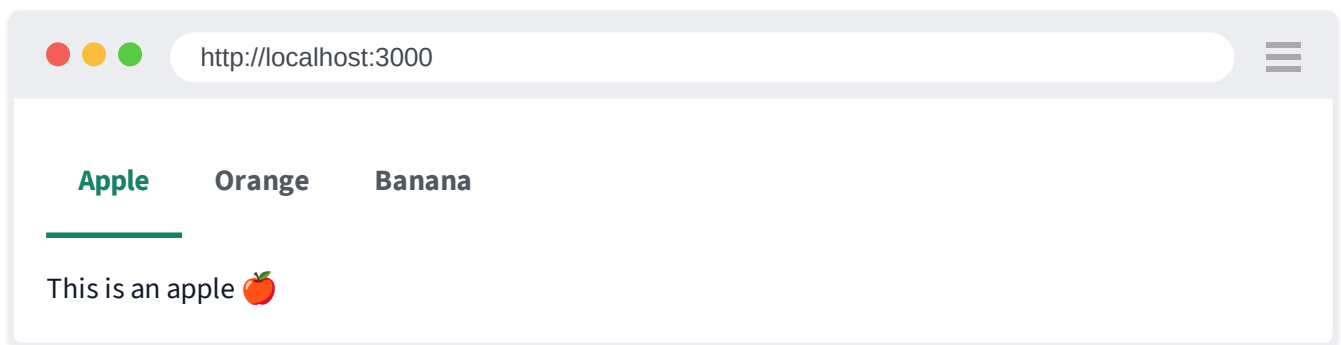


1.3.14 Tabs

Docusaurus provides the `<Tabs>` component that you can use in Markdown thanks to [MDX](#):

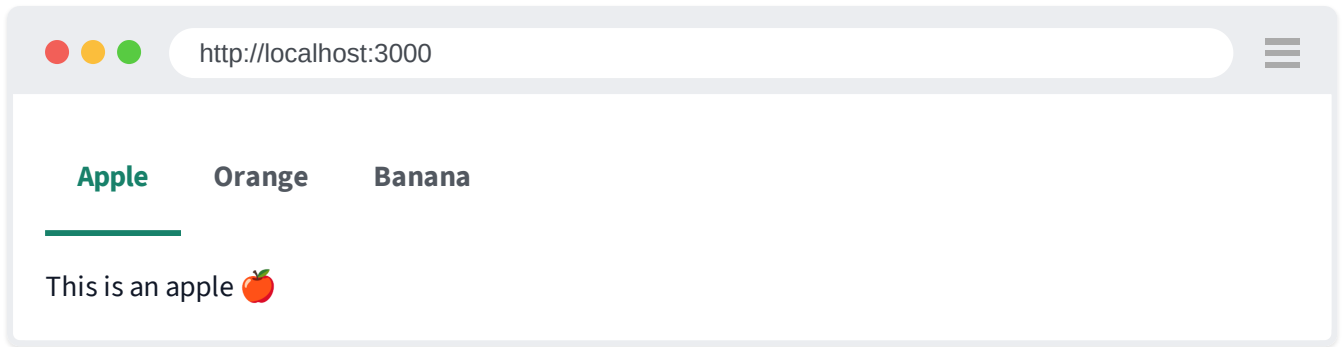
```
import Tabs from '@theme/Tabs';
import TabItem from '@theme/TabItem';

<Tabs>
  <TabItem value="apple" label="Apple" default>
    This is an apple 🍏
  </TabItem>
  <TabItem value="orange" label="Orange">
    This is an orange 🍊
  </TabItem>
  <TabItem value="banana" label="Banana">
    This is a banana 🍌
  </TabItem>
</Tabs>
```



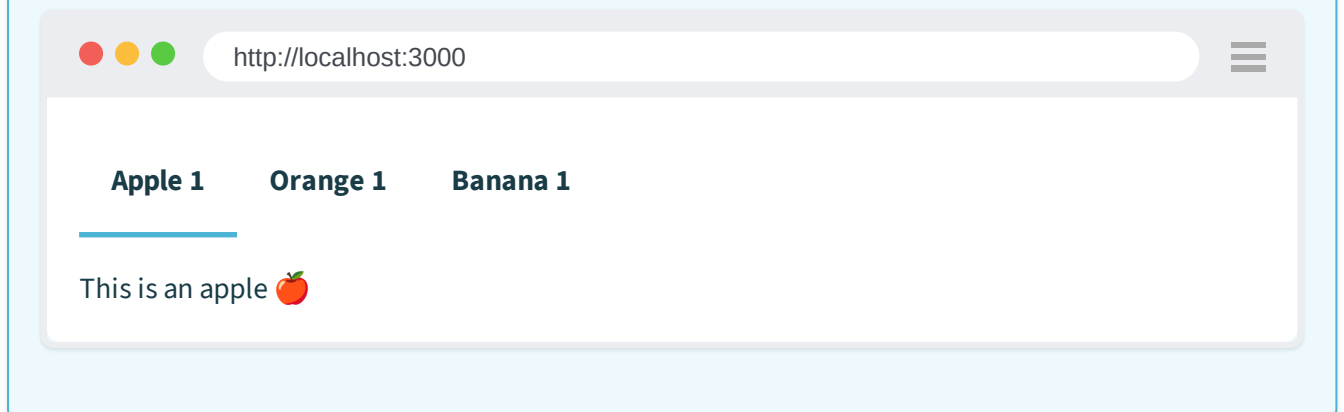
It is also possible to provide `values` and `defaultValue` props to `Tabs`:

```
<Tabs
  defaultValue="apple"
  values={[
    {label: 'Apple', value: 'apple'},
    {label: 'Orange', value: 'orange'},
    {label: 'Banana', value: 'banana'},
  ]}>
  <TabItem value="apple">This is an apple 🍏</TabItem>
  <TabItem value="orange">This is an orange 🍊</TabItem>
  <TabItem value="banana">This is a banana 🍌</TabItem>
</Tabs>
```



Tabs props take precedence over the **TabItem** props:

```
<Tabs
  defaultValue="apple"
  values={[
    {label: 'Apple 1', value: 'apple'},
    {label: 'Orange 1', value: 'orange'},
    {label: 'Banana 1', value: 'banana'},
  ]}>
  <TabItem value="apple" label="Apple 2">
    This is an apple 🍏
  </TabItem>
  <TabItem value="orange" label="Orange 2">
    This is an orange 🍊
  </TabItem>
  <TabItem value="banana" label="Banana 2" default>
    This is a banana 🍌
  </TabItem>
</Tabs>
```



TIP

By default, all tabs are rendered eagerly during the build process, and search engines can index hidden tabs.



It is possible to only render the default tab with `<Tabs lazy />`.

Displaying a default tab

The first tab is displayed by default, and to override this behavior, you can specify a default tab by adding `default` to one of the tab items. You can also set the `defaultValue` prop of the `Tabs` component to the label value of your choice. For example, in the example above, either setting `default` for the `value="apple"` tab or setting `defaultValue="apple"` for the tabs forces the "Apple" tab to be open by default.

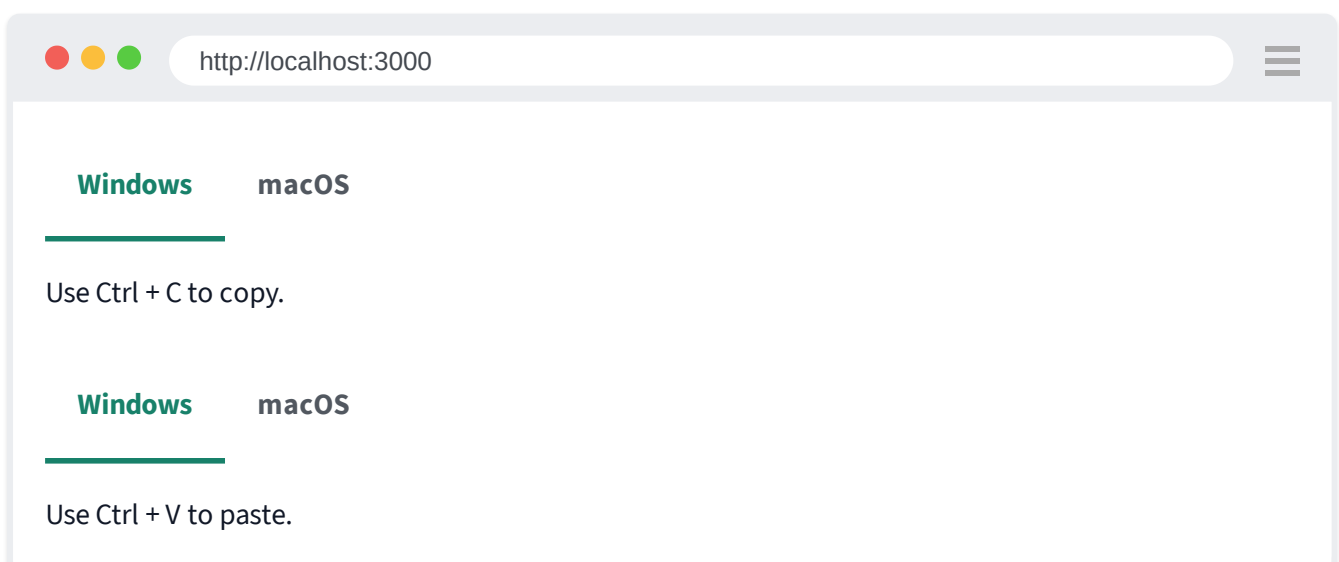
Docusaurus will throw an error if a `defaultValue` is provided for the `Tabs` but it refers to a non-existing value. If you want none of the tabs to be shown by default, use `defaultValue={null}`.

Syncing tab choices

You may want choices of the same kind of tabs to sync with each other. For example, you might want to provide different instructions for users on Windows vs users on macOS, and you want to change all OS-specific instructions tabs in one click. To achieve that, you can give all related tabs the same `groupId` prop. Note that doing this will persist the choice in `localStorage` and all `<Tab>` instances with the same `groupId` will update automatically when the value of one of them is changed. Note that group IDs are globally namespaced.

```
<Tabs groupId="operating-systems">
  <TabItem value="win" label="Windows">Use Ctrl + C to copy.</TabItem>
  <TabItem value="mac" label="macOS">Use Command + C to copy.</TabItem>
</Tabs>

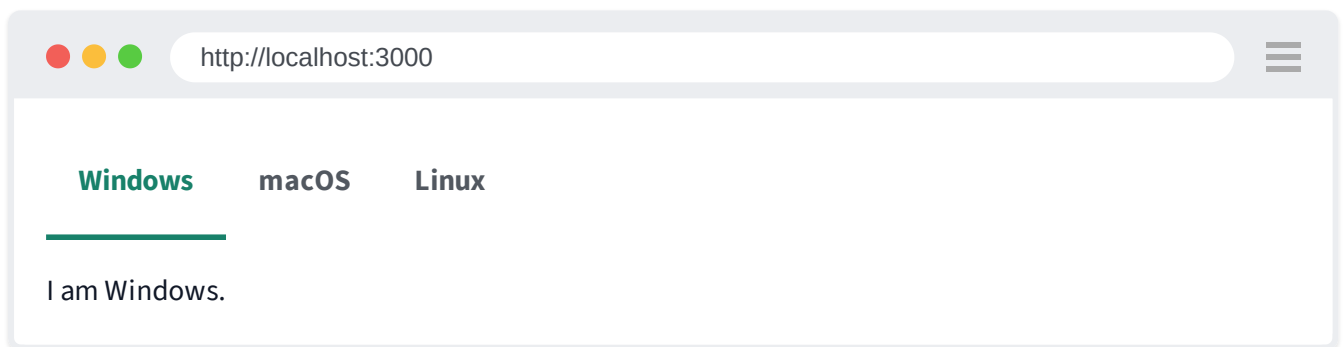
<Tabs groupId="operating-systems">
  <TabItem value="win" label="Windows">Use Ctrl + V to paste.</TabItem>
  <TabItem value="mac" label="macOS">Use Command + V to paste.</TabItem>
</Tabs>
```





For all tab groups that have the same `groupId`, the possible values do not need to be the same. If one tab group is chosen a value that does not exist in another tab group with the same `groupId`, the tab group with the missing value won't change its tab. You can see that from the following example. Try to select Linux, and the above tab groups don't change.

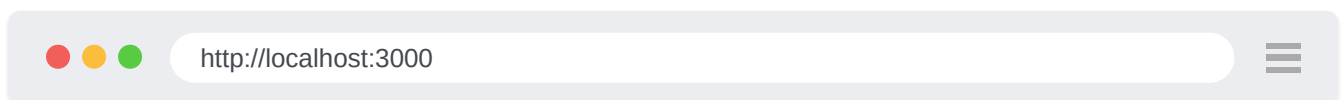
```
<Tabs groupId="operating-systems">
  <TabItem value="win" label="Windows">
    I am Windows.
  </TabItem>
  <TabItem value="mac" label="macOS">
    I am macOS.
  </TabItem>
  <TabItem value="linux" label="Linux">
    I am Linux.
  </TabItem>
</Tabs>
```



Tab choices with different group IDs will not interfere with each other:

```
<Tabs groupId="operating-systems">
  <TabItem value="win" label="Windows">Windows in windows.</TabItem>
  <TabItem value="mac" label="macOS">macOS is macOS.</TabItem>
</Tabs>

<Tabs groupId="non-mac-operating-systems">
  <TabItem value="win" label="Windows">Windows is windows.</TabItem>
  <TabItem value="unix" label="Unix">Unix is unix.</TabItem>
</Tabs>
```



**Windows**

macOS

Windows in windows.

Windows

Unix

Windows is windows.

Customizing tabs

You might want to customize the appearance of a certain set of tabs. You can pass the string in `className` prop, and the specified CSS class will be added to the `Tabs` component:

```
<Tabs className="unique-tabs">
  <TabItem value="Apple">This is an apple 🍏</TabItem>
  <TabItem value="Orange">This is an orange 🍊</TabItem>
  <TabItem value="Banana">This is a banana 🍌</TabItem>
</Tabs>
```



http://localhost:3000

**Apple****Orange****Banana**

This is an apple 🍏

Customizing tab headings

You can also customize each tab heading independently by using the `attributes` field. The extra props can be passed to the headings either through the `values` prop in `Tabs`, or props of each `TabItem`—in the same way as you declare `label`.

some-doc.mdx



```
import styles from './styles.module.css';

<Tabs>
  <TabItem value="apple" label="Apple" attributes={{className:
styles.red}}>
    This is an apple 🍏
  </TabItem>
  <TabItem value="orange" label="Orange" attributes={{className:
styles.orange}}>
    This is an orange 🍊
  </TabItem>
  <TabItem value="banana" label="Banana" attributes={{className:
styles.yellow}}>
    This is a banana 🍌
  </TabItem>
</Tabs>
```

styles.module.css

```
.red {
  color: red;
}
.red[aria-selected='true'] {
  border-bottom-color: red;
}

.orange {
  color: orange;
}
.orange[aria-selected='true'] {
  border-bottom-color: orange;
}

.yellow {
  color: yellow;
}
.yellow[aria-selected='true'] {
  border-bottom-color: yellow;
}
```



http://localhost:3000



**Apple**

Orange

Banana

This is an apple 🍏

**TIP**

`className` would be merged with other default class names. You may also use a custom `data-value` field (`{ 'data-value': 'apple' }`) paired with CSS attribute selectors:

styles.module.css

```
li[role='tab'][data-value='apple'] {  
  color: red;  
}
```

Query string

It is possible to persist the selected tab into the url search parameters. This enables you to share a link to a page which pre-selects the tab - linking from your Android app to documentation with the Android tabs pre-selected. This feature does not provide an anchor link - the browser will not scroll to the tab.

Use the `queryString` prop to enable this feature and define the search param name to use.

```
<Tabs queryString="current-os">  
  <TabItem value="android" label="Android">  
    Android  
  </TabItem>  
  <TabItem value="ios" label="iOS">  
    iOS  
  </TabItem>  
</Tabs>
```



http://localhost:3000

**Android**

iOS



Android

As soon as a tab is clicked, a search parameter is added at the end of the url: `?current-os=android` or `?current-os=ios`.



TIP

`queryString` can be used together with `groupId`.

For convenience, when the `queryString` prop is `true`, the `groupId` value will be used as a fallback.

```
<Tabs groupId="current-os" queryString>
  <TabItem value="android" label="Android">
    Android
  </TabItem>
  <TabItem value="ios" label="iOS">
    iOS
  </TabItem>
</Tabs>
```



http://localhost:3000



Android

iOS

Android

When the page loads, the tab query string choice will be restored in priority over the `groupId` choice (using `localStorage`).

1.3.15 Code blocks

Code blocks within documentation are super-powered 💪.

Code title

You can add a title to the code block by adding a `title` key after the language (leave a space between them).



```
```jsx title="/src/components/HelloCodeTitle.js"
function HelloCodeTitle(props) {
 return <h1>Hello, {props.name}</h1>;
}
```
```

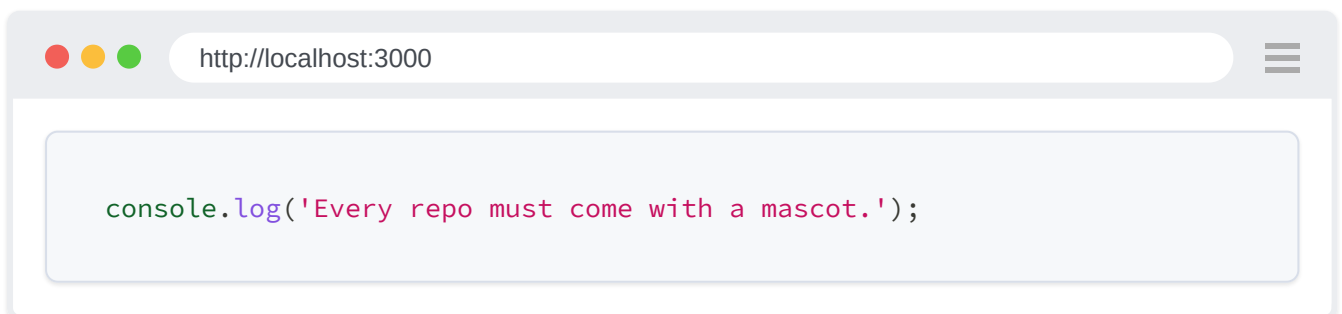


Syntax highlighting

Code blocks are text blocks wrapped around by strings of 3 backticks. You may check out [this reference](#) for the specifications of MDX.

```
```js
console.log('Every repo must come with a mascot.');
```

Use the matching language meta string for your code block, and Docusaurus will pick up syntax highlighting automatically, powered by [Prism React Renderer](#).



## Theming

By default, the Prism [syntax highlighting theme](#) we use is [Palenight](#). You can change this to another theme by passing `theme` field in `prism` as `themeConfig` in your `docusaurus.config.js`.



For example, if you prefer to use the `dracula` highlighting theme:

docusaurus.config.js

```
import {themes as prismThemes} from 'prism-react-renderer';

export default {
 themeConfig: {
 prism: {
 theme: prismThemes.dracula,
 },
 },
};
```

Because a Prism theme is just a JS object, you can also write your own theme if you are not satisfied with the default. Docusaurus enhances the `github` and `vsDark` themes to provide richer highlight, and you can check our implementations for the `light` and `dark` code block themes.

## Supported Languages

By default, Docusaurus comes with a subset of [commonly used languages](#).

### WARNING

Some popular languages like Java, C#, or PHP are not enabled by default.

To add syntax highlighting for any of the other [Prism-supported languages](#), define it in an array of additional languages.

### NOTE

Each additional language has to be a valid Prism component name. For example, Prism would map the language `cs` to `csharp`, but only `prism-csharp.js` exists as a *component*, so you need to use `additionalLanguages: ['csharp']`. You can look into `node_modules/prismjs/components` to find all components (languages) available.

For example, if you want to add highlighting for the PowerShell language:

docusaurus.config.js



```
export default {
 // ...
 themeConfig: {
 prism: {
 additionalLanguages: ['powershell'],
 },
 // ...
 },
};
```

After adding `additionalLanguages`, restart Docusaurus.

If you want to add highlighting for languages not yet supported by Prism, you can swizzle `prism-include-languages`:

**npm**   **Yarn**   **pnpm**   **Bun**

```
npm run swizzle @docusaurus/theme-classic prism-include-languages
```

It will produce `prism-include-languages.js` in your `src/theme` folder. You can add highlighting support for custom languages by editing `prism-include-languages.js`:

`src/theme/prism-include-languages.js`

```
const prismIncludeLanguages = (Prism) => {
 // ...

 additionalLanguages.forEach((lang) => {
 require(`prismjs/components/prism-${lang}`);
 });

 require('/path/to/your/prism-language-definition');

 // ...
};
```

You can refer to [Prism's official language definitions](#) when you are writing your own language definitions.

When adding a custom language definition, you do not need to add the language to the `additionalLanguages` config array, since Docusaurus only looks up the `additionalLanguages` strings



in languages that Prism provides. Adding the language import in `prism-include-languages.js` is sufficient.

## Line highlighting

### Highlighting with comments

You can use comments with `highlight-next-line`, `highlight-start`, and `highlight-end` to select which lines are highlighted.

```
```js
function HighlightSomeText(highlight) {
  if (highlight) {
    // highlight-next-line
    return 'This text is highlighted!';
  }

  return 'Nothing highlighted';
}

function HighlightMoreText(highlight) {
  // highlight-start
  if (highlight) {
    return 'This range is highlighted!';
  }
  // highlight-end

  return 'Nothing highlighted';
}
```
```



http://localhost:3000





```
function HighlightSomeText(highlight) {
 if (highlight) {
 return 'This text is highlighted!';
 }

 return 'Nothing highlighted';
}

function HighlightMoreText(highlight) {
 if (highlight) {
 return 'This range is highlighted!';
 }

 return 'Nothing highlighted';
}
```

Supported commenting syntax:

| Style      | Syntax                                         |
|------------|------------------------------------------------|
| C-style    | <code>/* ... */</code> and <code>// ...</code> |
| JSX-style  | <code>{/* ... */}</code>                       |
| Bash-style | <code># ...</code>                             |
| HTML-style | <code>&lt;!-- ... --&gt;</code>                |

We will do our best to infer which set of comment styles to use based on the language, and default to allowing *all* comment styles. If there's a comment style that is not currently supported, we are open to adding them! Pull requests welcome. Note that different comment styles have no semantic difference, only their content does.

You can set your own background color for highlighted code line in your `src/css/custom.css` which will better fit to your selected syntax highlighting theme. The color given below works for the default highlighting theme (Palenight), so if you are using another theme, you will have to tweak the color accordingly.

```
/src/css/custom.css
```



```
:root {
 --docusaurus-highlighted-code-line-bg: rgb(72, 77, 91);
}

/* If you have a different syntax highlighting theme for dark mode. */
[data-theme='dark'] {
 /* Color which works with dark mode syntax highlighting theme */
 --docusaurus-highlighted-code-line-bg: rgb(100, 100, 100);
}
```

If you also need to style the highlighted code line in some other way, you can target on `theme-code-block-highlighted-line` CSS class.

## Highlighting with metadata string

You can also specify highlighted line ranges within the language meta string (leave a space after the language). To highlight multiple lines, separate the line numbers by commas or use the range syntax to select a chunk of lines. This feature uses the `parse-number-range` library and you can find [more syntax](#) on their project details.

```
```jsx {1,4-6,11}  
import React from 'react';  
  
function MyComponent(props) {  
  if (props.isBar) {  
    return <div>Bar</div>;  
  }  
  
  return <div>Foo</div>;  
}  
  
export default MyComponent;  
```
```



http://localhost:3000





```
import React from 'react';

function MyComponent(props) {
 if (props.isBar) {
 return <div>Bar</div>;
 }

 return <div>Foo</div>;
}

export default MyComponent;
```



### PREFER COMMENTS

Prefer highlighting with comments where you can. By inlining highlight in the code, you don't have to manually count the lines if your code block becomes long. If you add/remove lines, you also don't have to offset your line ranges.

```
- ``jsx {3}
+ ``jsx {4}
function HighlightSomeText(highlight) {
 if (highlight) {
+ console.log('Highlighted text found');
 return 'This text is highlighted!';
 }

 return 'Nothing highlighted';
}
...`
```

Below, we will introduce how the magic comment system can be extended to define custom directives and their functionalities. The magic comments would only be parsed if a highlight metastring is not present.

## Custom magic comments

`// highlight-next-line` and `// highlight-start` etc. are called "magic comments", because they will be parsed and removed, and their purposes are to add metadata to the next line, or the section that the pair of start- and end-comments enclose.

You can declare custom magic comments through theme config. For example, you can register another magic comment that adds a `code-block-error-line` class name:





```
export default {
 themeConfig: {
 prism: {
 magicComments: [
 // Remember to extend the default highlight class name as well!
 {
 className: 'theme-code-block-highlighted-line',
 line: 'highlight-next-line',
 block: {start: 'highlight-start', end: 'highlight-end'},
 },
 {
 className: 'code-block-error-line',
 line: 'This will error',
 },
],
 },
 },
};
```

http://localhost:3000

In JavaScript, trying to access properties on `null` will error.

```
const name = null;
console.log(name.toUpperCase());
// Uncaught TypeError: Cannot read properties of null (reading 'toUpperCase')
```

If you use number ranges in metastring (the `{1,3-4}` syntax), Docusaurus will apply the **first** `magicComments` entry's class name. This, by default, is `theme-code-block-highlighted-line`, but if you change the `magicComments` config and use a different entry as the first one, the meaning of the metastring range will change as well.

You can disable the default line highlighting comments with `magicComments: []`. If there's no magic comment config, but Docusaurus encounters a code block containing a metastring range, it will error because there will be no class name to apply—the highlighting class name, after all, is just a magic comment entry.



Every magic comment entry will contain three keys: `className` (required), `line`, which applies to the directly next line, or `block` (containing `start` and `end`), which applies to the entire block enclosed by the two comments.

Using CSS to target the class can already do a lot, but you can unlock the full potential of this feature through [swizzling](#).

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)

```
npm run swizzle @docusaurus/theme-classic CodeBlock/Line
```

The `Line` component will receive the list of class names, based on which you can conditionally render different markup.

## Line numbering

You can enable line numbering for your code block by using `showLineNumbers` key within the language meta string (don't forget to add space directly before the key).

```
```jsx showLineNumbers
import React from 'react';

export default function MyComponent(props) {
  return <div>Foo</div>;
}
```
```



http://localhost:3000



```
1 import React from 'react';
2
3 export default function MyComponent(props) {
4 return <div>Foo</div>;
5 }
```

By default, the counter starts at line number 1. It's possible to pass a custom counter start value to split large code blocks for readability:



```
```jsx showLineNumbers=3
export default function MyComponent(props) {
  return <div>Foo</div>;
}
```
```



## Interactive code editor

(Powered by [React Live](#))

You can create an interactive coding editor with the `@docusaurus/theme-live-codeblock` plugin. First, add the plugin to your package.

**npm**   Yarn   pnpm   Bun

```
npm install --save @docusaurus/theme-live-codeblock
```

You will also need to add the plugin to your `docusaurus.config.js`.

```
export default {
 // ...
 themes: ['@docusaurus/theme-live-codeblock'],
 // ...
};
```

To use the plugin, create a code block with `live` attached to the language meta string.



```
```jsx live
function Clock(props) {
  const [date, setDate] = useState(new Date());

  useEffect(() => {
    const id = setInterval(() => {
      setDate(new Date());
    }, 1000);
    return () => clearInterval(id);
  }, []);

  return <h2>It is {date.toLocaleTimeString()}.</h2>;
}
```
```

The code block will be rendered as an interactive editor. Changes to the code will reflect on the result panel live.

http://localhost:3000

☰

**LIVE EDITOR**

↻ Reset

```
function Clock(props) {
 const [date, setDate] = useState(new Date());

 useEffect(() => {
 const id = setInterval(() => {
 setDate(new Date());
 }, 1000);
 return () => clearInterval(id);
 }, []);

 return <h2>It is {date.toLocaleTimeString()}.</h2>;
}
```

**RESULT**

Loading...

## Imports



### ⚠️ REACT-LIVE AND IMPORTS

It is not possible to import components directly from the react-live code editor, you have to define available imports upfront.

By default, all React imports are available. If you need more imports available, swizzle the react-live scope:

**npm**   **Yarn**   **pnpm**   **Bun**

```
npm run swizzle @docusaurus/theme-live-codeblock ReactLiveScope -- --eject
```

src/theme/ReactLiveScope/index.js

```
import React from 'react';

const ButtonExample = (props) => (
 <button
 {...props}
 style={{
 backgroundColor: 'white',
 color: 'black',
 border: 'solid red',
 borderRadius: 20,
 padding: 10,
 cursor: 'pointer',
 ...props.style,
 }}
 />
);

// Add react-live imports you need here
const ReactLiveScope = {
 React,
 ...React,
 ButtonExample,
};

export default ReactLiveScope;
```

The `ButtonExample` component is now available to use:



http://localhost:3000

LIVE EDITOR

Reset

```
function MyPlayground(props) {
 return (
 <div>
 <ButtonExample onClick={() => alert('hey!')}>Click me</ButtonExample>
 </div>
);
}
```

RESULT

Loading...

## Imperative Rendering (noInline)

The `noInline` option should be used to avoid errors when your code spans multiple components or variables.

```
```jsx live noInline  
const project = 'Docusaurus';  
  
const Greeting = () => <p>Hello {project}!</p>;  
  
render(<Greeting />);  
```
```

Unlike an ordinary interactive code block, when using `noInline` React Live won't wrap your code in an inline function to render it.

You will need to explicitly call `render()` at the end of your code to display the output.

http://localhost:3000



## LIVE EDITOR

Reset

```
const project = "Docusaurus";

const Greeting = () => (
 <p>Hello {project}!</p>
);

render(
 <Greeting />
);
```

## RESULT

Loading...

## Using JSX markup in code blocks

Code block in Markdown always preserves its content as plain text, meaning you can't do something like:

```
type EditUrlFunction = (params: {
 // This doesn't turn into a link (for good reason!)
 version: Version;
 versionDocsDirPath: string;
 docPath: string;
 permalink: string;
 locale: string;
}) => string | undefined;
```

If you want to embed HTML markup such as anchor links or bold type, you can use the `<pre>` tag, `<code>` tag, or `<CodeBlock>` component.

```
<pre>
 Input: 1 2 3 4{'\n'}
 Output: "366300745"{'\n'}
</pre>
```



http://localhost:3000





```
Input: 1 2 3 4
Output: "366300745"
```

### ⚠ MDX IS WHITESPACE INSENSITIVE

MDX is in line with JSX behavior: line break characters, even when inside `<pre>`, are turned into spaces. You have to explicitly write the new line character for it to be printed out.

### ⚠ WARNING

Syntax highlighting only works on plain strings. Docusaurus will not attempt to parse code block content containing JSX children.

## Multi-language support code blocks

With MDX, you can easily create interactive components within your documentation, for example, to display code in multiple programming languages and switch between them using a tabs component.

Instead of implementing a dedicated component for multi-language support code blocks, we've implemented a general-purpose `<Tabs>` component in the classic theme so that you can use it for other non-code scenarios as well.

The following example is how you can have multi-language code tabs in your docs. Note that the empty lines above and below each language block are **intentional**. This is a [current limitation of MDX](#): you have to leave empty lines around Markdown syntax for the MDX parser to know that it's Markdown syntax and not JSX.





```
import Tabs from '@theme/Tabs';
import TabItem from '@theme/TabItem';

<Tabs>
<TabItem value="js" label="JavaScript">

 ``js
function helloWorld() {
 console.log('Hello, world!');
}
 ``

</TabItem>
<TabItem value="py" label="Python">

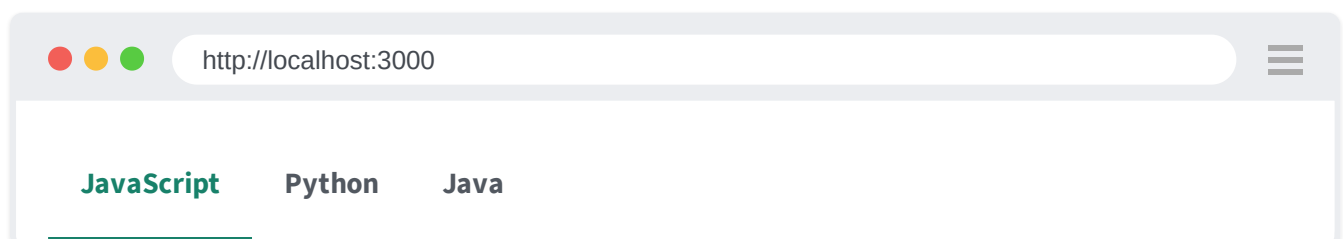
 ``py
def hello_world():
 print("Hello, world!")
 ``

</TabItem>
<TabItem value="java" label="Java">

 ``java
class HelloWorld {
 public static void main(String args[]) {
 System.out.println("Hello, World");
 }
}
 ``

</TabItem>
</Tabs>
```

And you will get the following:





```
function helloWorld() {
 console.log('Hello, world!');
}
```

If you have multiple of these multi-language code tabs, and you want to sync the selection across the tab instances, refer to the [Syncing tab choices section](#).

## Docusaurus npm2yarn remark plugin

Displaying CLI commands in both npm and Yarn is a very common need, for example:

**npm**   Yarn   pnpm   Bun

```
npm install @docusaurus/remark-plugin-npm2yarn
```

Docusaurus provides such a utility out of the box, freeing you from using the `Tabs` component every time. To enable this feature, first install the `@docusaurus/remark-plugin-npm2yarn` package as above, and then in `docusaurus.config.js`, for the plugins where you need this feature (doc, blog, pages, etc.), register it in the `remarkPlugins` option. (See [Docs configuration](#) for more details on configuration format)

```
docusaurus.config.js
```



```
export default {
 // ...
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 docs: {
 remarkPlugins: [
 [require('@docusaurus/remark-plugin-npm2yarn'), {sync: true}],
],
 },
 pages: {
 remarkPlugins: [require('@docusaurus/remark-plugin-npm2yarn')],
 },
 blog: {
 remarkPlugins: [
 [
 require('@docusaurus/remark-plugin-npm2yarn'),
 {converters: ['pnpm']},
],
],
 },
 // ...
 },
],
],
};
```

And then use it by adding the `npm2yarn` key to the code block:

```
```bash npm2yarn
npm install @docusaurus/remark-plugin-npm2yarn
```
```

## Configuration

| Option                  | Type    | Default        | Description                                                                                                                         |
|-------------------------|---------|----------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <code>sync</code>       | boolean | false          | Whether to sync the selected converter across all code blocks.                                                                      |
| <code>converters</code> | array   | 'yarn', 'pnpm' | The list of converters to use. The order of the converters is important, as the first converter will be used as the default choice. |

## Usage in JSX



Outside of Markdown, you can use the `@theme/CodeBlock` component to get the same output.

```
import CodeBlock from '@theme/CodeBlock';

export default function MyReactPage() {
 return (
 <div>
 <CodeBlock
 language="jsx"
 title="/src/components/HelloCodeTitle.js"
 showLineNumbers>
 {`function HelloCodeTitle(props) {
 return <h1>Hello, {props.name}</h1>;
}`}
 </CodeBlock>
 </div>
);
}
```



The props accepted are `language`, `title` and `showLineNumbers`, in the same way as you write Markdown code blocks.

Although discouraged, you can also pass in a `metastring` prop like `metastring='{1-2} title="/src/components/HelloCodeTitle.js" showLineNumbers'`, which is how Markdown code blocks are handled under the hood. However, we recommend you [use comments for highlighting lines](#).

As [previously stated](#), syntax highlighting is only applied when the children is a simple string.

## 1.3.16 Admonitions

In addition to the basic Markdown syntax, we have a special admonitions syntax by wrapping text with a set of 3 colons, followed by a label denoting its type.

Example:



:::note

Some **content** with *Markdown* ``syntax``. Check `[this `api`](#)`.

:::

:::tip

Some **content** with *Markdown* ``syntax``. Check `[this `api`](#)`.

:::

:::info

Some **content** with *Markdown* ``syntax``. Check `[this `api`](#)`.

:::

:::warning

Some **content** with *Markdown* ``syntax``. Check `[this `api`](#)`.

:::

:::danger

Some **content** with *Markdown* ``syntax``. Check `[this `api`](#)`.

:::



http://localhost:3000



 **NOTE**

Some **content** with *Markdown* `syntax`. Check `this api`.

 **TIP**

Some **content** with *Markdown* `syntax`. Check `this api`.

 **INFO**



Some **content** with *Markdown* `syntax`. Check [this](#) `api`.

#### WARNING

Some **content** with *Markdown* `syntax`. Check [this](#) `api`.

#### DANGER

Some **content** with *Markdown* `syntax`. Check [this](#) `api`.

## Usage with Prettier

If you use [Prettier](#) to format your Markdown files, Prettier might auto-format your code to invalid admonition syntax. To avoid this problem, add empty lines around the starting and ending directives. This is also why the examples we show here all have empty lines around the content.

```
<!-- Prettier doesn't change this -->
```

```
:::note
```

```
Hello world
```

```
:::
```

```
<!-- Prettier changes this -->
```

```
:::note
```

```
Hello world
```

```
:::
```

```
<!-- to this -->
```

```
::: note Hello world:::
```

## Specifying title

You may also specify an optional title.

```
:::note[Your Title with some Markdown 'syntax'!]
```

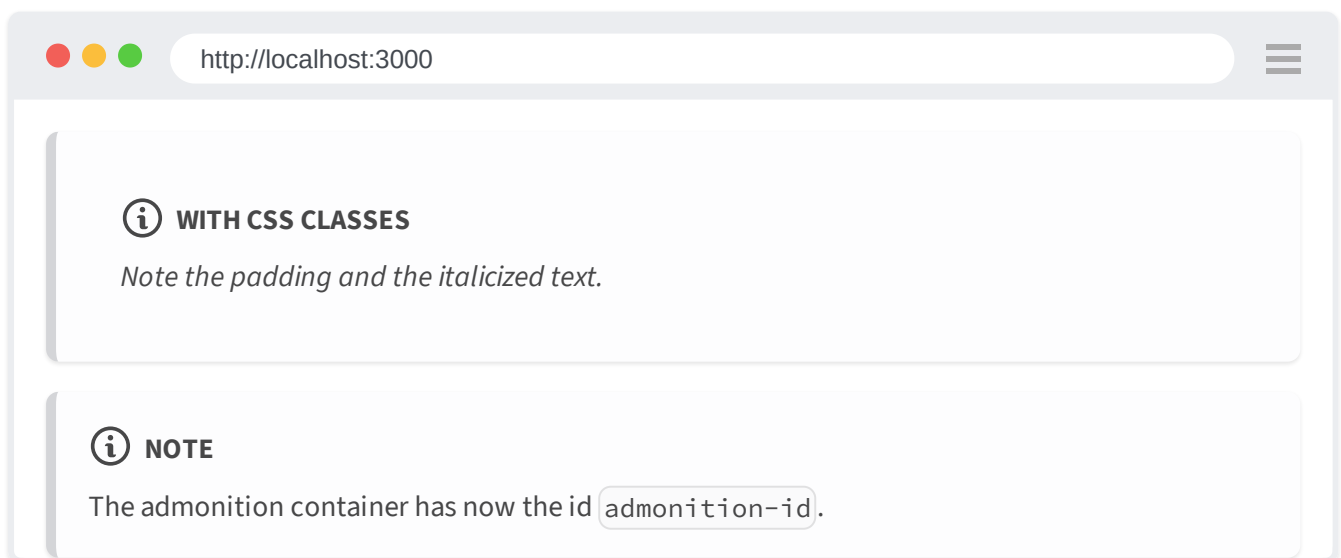
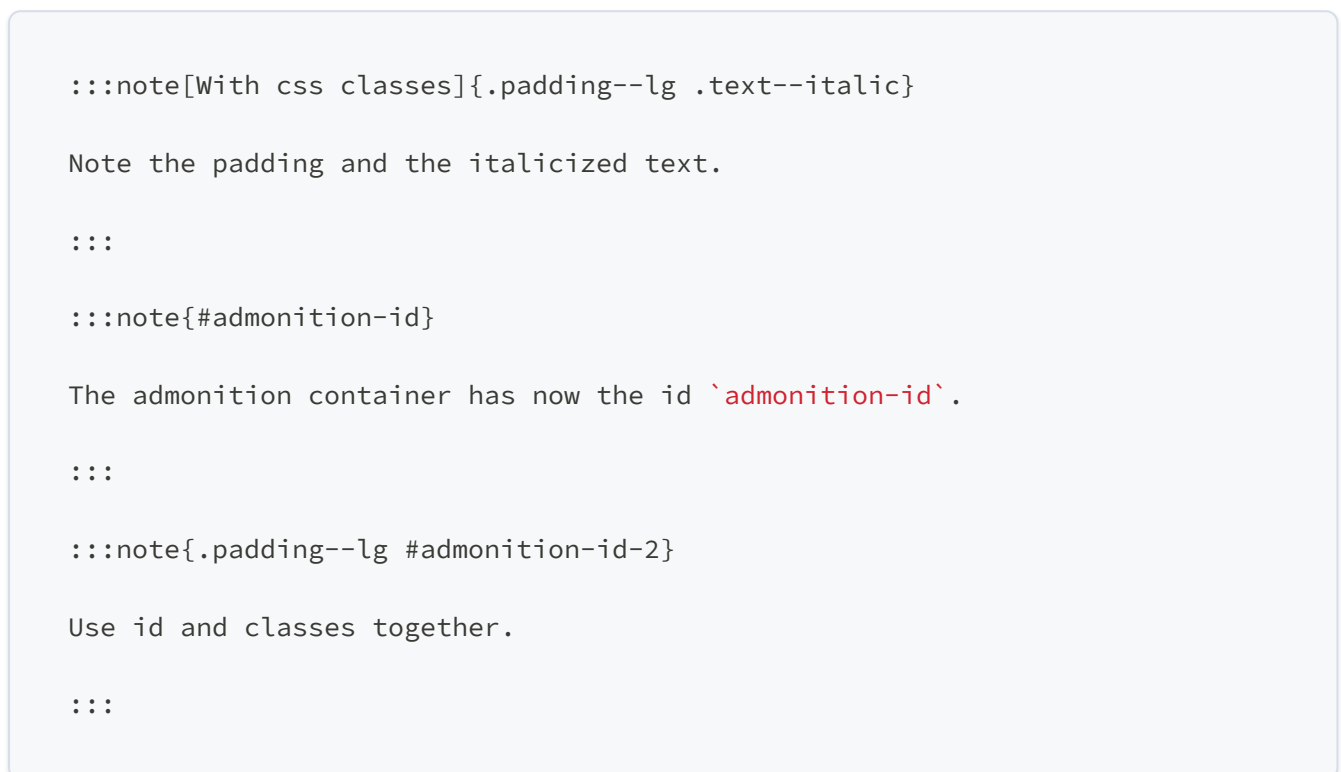
```
Some content with some Markdown 'syntax'.
```

```
:::
```



## Specifying attributes

You may also provide classes or IDs to admonitions.



**i NOTE**

Use id and classes together.

## Nested admonitions

Admonitions can be nested. Use more colons `:` for each parent admonition level.

```
:::::info[Parent]
```

Parent content

```
::::danger[Child]
```

Child content

```
:::tip[Deep Child]
```

Deep child content

```
:::
```

```
::::
```

```
:::::
```



http://localhost:3000

**i PARENT**

Parent content

**🔥 CHILD**

Child content

**💡 DEEP CHILD**





Deep child content

## Admonitions with MDX

You can use MDX inside admonitions too!

```
import Tabs from '@theme/Tabs';

import TabItem from '@theme/TabItem';

:::tip[Use tabs in admonitions]

<Tabs>
 <TabItem value="apple" label="Apple">This is an apple 🍏</TabItem>
 <TabItem value="orange" label="Orange">This is an orange 🍊</TabItem>
 <TabItem value="banana" label="Banana">This is a banana 🍌</TabItem>
</Tabs>

:::
```



http://localhost:3000



### USE TABS IN ADMONITIONS

Apple

Orange

Banana

This is an apple 🍏

## Usage in JSX

Outside of Markdown, you can use the `@theme/Admonition` component to get the same output.

MyReactPage.jsx



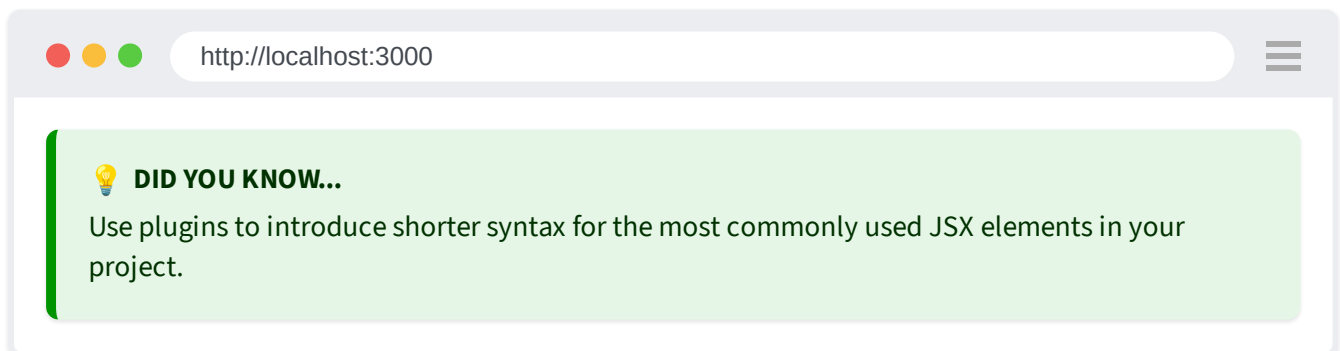
```
import Admonition from '@theme/Admonition';

export default function MyReactPage() {
 return (
 <div>
 <Admonition type="info">
 <p>Some information</p>
 </Admonition>
 </div>
);
}
```

The types that are accepted are the same as above: `note`, `tip`, `danger`, `info`, `warning`. Optionally, you can specify an icon by passing a JSX element or a string, or a title:

MyReactPage.jsx

```
<Admonition type="tip" icon="💡" title="Did you know...">
 Use plugins to introduce shorter syntax for the most commonly used JSX
 elements in your project.
</Admonition>
```



## Customizing admonitions

There are two kinds of customizations possible with admonitions: **parsing** and **rendering**.

### Customizing rendering behavior

You can customize how each individual admonition type is rendered through [swizzling](#). You can often achieve your goal through a simple wrapper. For example, in the follow example, we swap out the icon for `info` admonitions only.

src/theme/Admonition.js



```
import React from 'react';
import Admonition from '@theme-original/Admonition';
import MyCustomNoteIcon from '@site/static/img/info.svg';

export default function AdmonitionWrapper(props) {
 if (props.type !== 'info') {
 return <Admonition title="My Custom Admonition Title" {...props} />;
 }
 return <Admonition icon={<MyCustomNoteIcon />} {...props} />;
}
```

## Customizing parsing behavior

Admonitions are implemented with a [Remark plugin](#). The plugin is designed to be configurable. To customize the Remark plugin for a specific content plugin (docs, blog, pages), pass the options through the `admonitions` key.

docusaurus.config.js

```
export default {
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 docs: {
 admonitions: {
 keywords: ['note', 'tip', 'info', 'warning', 'danger'],
 extendDefaults: true,
 },
 },
],
],
],
};
```

The plugin accepts the following options:

- `keywords`: An array of keywords that can be used as the type for the admonition.
- `extendDefaults`: Should the provided options (such as `keywords`) be merged into the existing defaults. Defaults to `true`.

The `keyword` will be passed as the `type` prop of the `Admonition` component.

## Custom admonition type components



By default, the theme doesn't know what to do with custom admonition keywords such as `:::my-custom-admonition`. It is your responsibility to map each admonition keyword to a React component so that the theme knows how to render them.

If you registered a new admonition type `my-custom-admonition` via the following config:

docusaurus.config.js

```
export default {
 // ...
 presets: [
 [
 'classic',
 {
 // ...
 docs: {
 admonitions: {
 keywords: ['my-custom-admonition'],
 extendDefaults: true,
 },
 },
],
],
],
};
```

You can provide the corresponding React component for `:::my-custom-admonition` by creating the following file (unfortunately, since it's not a React component file, it's not swizzlable):

src/theme/Admonition/Types.js



```
import React from 'react';
import DefaultAdmonitionTypes from '@theme-original/Admonition/Types';

function MyCustomAdmonition(props) {
 return (
 <div style={{border: 'solid red', padding: 10}}>
 <h5 style={{color: 'blue', fontSize: 30}}>{props.title}</h5>
 <div>{props.children}</div>
 </div>
);
}

const AdmonitionTypes = {
 ...DefaultAdmonitionTypes,

 // Add all your custom admonition types here...
 // You can also override the default ones if you want
 'my-custom-admonition': MyCustomAdmonition,
};

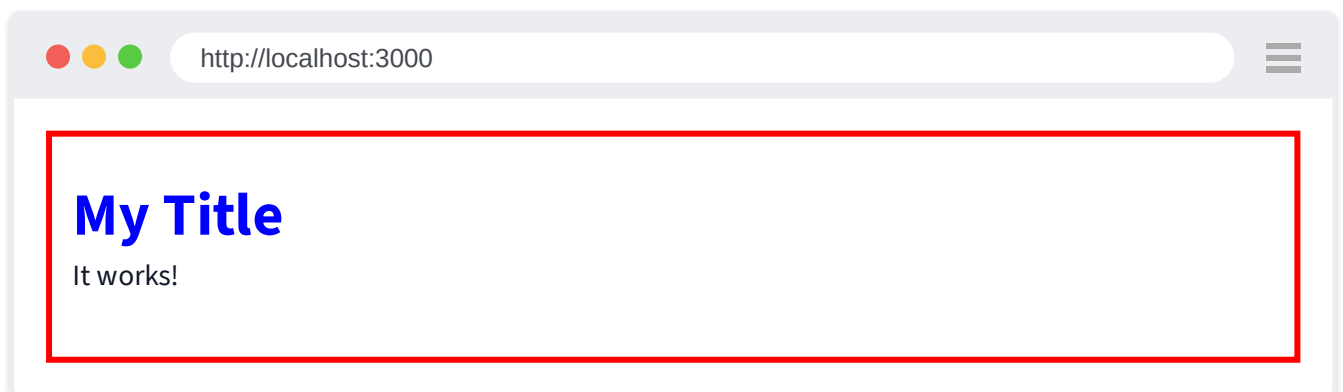
export default AdmonitionTypes;
```

Now you can use your new admonition keyword in a Markdown file, and it will be parsed and rendered with your custom logic:

```
:::my-custom-admonition[My Title]

It works!

:::
```





## 1.3.17 Headings and Table of contents

### Markdown headings

You can use regular Markdown headings.

```
Level 2 title

Level 3 title

Level 4 title
```

Each Markdown heading will appear as a table of contents entry.

#### Heading IDs

Each heading has an ID that can be automatically generated or explicitly specified. Heading IDs allow you to link to a specific document heading in Markdown or JSX:

```
[link](#heading-id)
```

```
<Link to="#heading-id">link</Link>
```

By default, Docusaurus will generate heading IDs for you, based on the heading text. For example, `### Hello World` will have ID `hello-world`.

Generated IDs have **some limitations**:

- The ID might not look good
- You might want to **change or translate** the text without updating the existing ID to avoid breaking links

A special syntax lets you set an **explicit heading id**.

**MDX**

**CommonMark**



```
Hello World {/* #my-explicit-id */}

Hello World {#my-explicit-id}
```

The heading id comment must start with `#`, be placed at the **end** of the heading and will be stripped from the rendered output.

### TIP

Use the `write-heading-ids` CLI command to add explicit IDs to all your Markdown documents.

The `--syntax` option lets you choose which syntax you prefer:

- The `classic` syntax for `{#headingId}`
- The `mdx-comment` syntax for `{/* #headingId */}`

### AVOID THE CLASSIC `{#id}` SYNTAX FOR MDX FILES

For MDX files, the `{#id}` syntax should be avoided. Since Docusaurus v3 and MDX v2, it is **not valid MDX syntax anymore**. It can break external tools that support MDX (IDEs and linters). It is only supported in Docusaurus for backward compatibility, thanks to the `markdown.mdx1Compat.headingIds` config option. The comment-based syntax should be preferred for MDX documents.

### AVOID COLLIDING IDS

Generated heading IDs will be guaranteed to be unique on each page, but if you use custom IDs, make sure each one appears exactly once on each page, or there will be two DOM elements with the same ID, which is invalid HTML semantics, and will lead to one heading being unclickable.

## Table of contents heading level

Each Markdown document displays a table of contents on the top-right corner. By default, this table only shows h2 and h3 headings, which should be sufficient for an overview of the page structure. In case you need to change the range of headings displayed, you can customize the minimum and maximum heading level — either per page or globally.

To set the heading level for a particular page, use the `toc_min_heading_level` and `toc_max_heading_level` front matter.

```
myDoc.md
```



```

Display h2 to h5 headings
toc_min_heading_level: 2
toc_max_heading_level: 5

```

To set the heading level for *all* pages, use the `themeConfig.tableOfContents` option.

docusaurus.config.js

```
export default {
 themeConfig: {
 tableOfContents: {
 minHeadingLevel: 2,
 maxHeadingLevel: 5,
 },
 },
};
```

If you've set the options globally, you can still override them locally via front matter.

### **i** NOTE

The `themeConfig` option would apply to all TOC on the site, including [inline TOC](#), but front matter options only affect the top-right TOC. You need to use the `minHeadingLevel` and `maxHeadingLevel` props to customize each `<TOCInline />` component.

## Inline table of contents

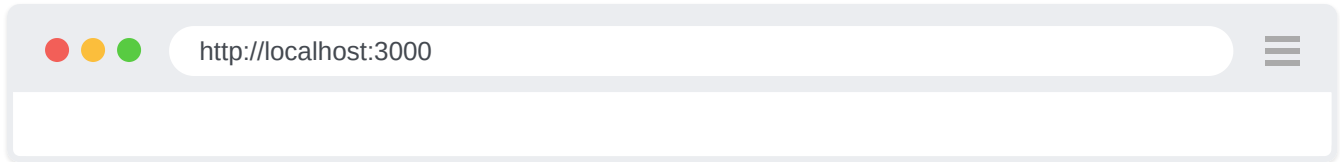
It is also possible to display an inline table of contents directly inside a Markdown document, thanks to MDX.

The `toc` variable is available in any MDX document and contains all the headings of an MDX document. By default, only `h2` and `h3` headings are displayed in the TOC. You can change which heading levels are visible by setting `minHeadingLevel` or `maxHeadingLevel` for individual `TOCInline` components.

```
import TOCInline from '@theme/TOCInline';

<TOCInline toc={toc} />
```





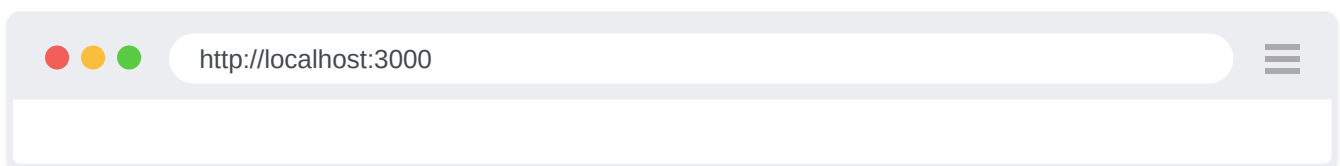
The `toc` global is just a list of heading items:

```
declare const toc: {
 value: string;
 id: string;
 level: number;
}[];
```

Note that the `toc` global is a flat array, so you can easily cut out unwanted nodes or insert extra nodes, and create a new TOC tree.

```
import TOCInline from '@theme/TOCInline';

<TOCInline
 // Only show h2 and h4 headings
 toc={toc.filter((node) => node.level === 2 || node.level === 4)}
 minHeadingLevel={2}
 // Show h4 headings in addition to the default h2 and h3 headings
 maxHeadingLevel={4}
>
```



## Customizing table of contents generation

The table-of-contents is generated by parsing the Markdown source with a [Remark plugin](#). There are known edge-cases where it generates false-positives and false-negatives.

Markdown headings within hideable areas will still show up in the TOC. For example, headings within `Tabs` and `details` will not be excluded.

Non-Markdown headings will not show up in the TOC. This can be used to your advantage to tackle the aforementioned issue.



```
<details>
<summary>Some details containing headings</summary>
<h2 id="#heading-id">I'm a heading that will not show up in the TOC</h2>

Some content...

</details>
```

The ability to ergonomically insert extra headings or ignore certain headings is a work-in-progress. If this feature is important to you, please report your use-case in [this issue](#).

### WARNING

Below is just some dummy content to have more table of contents items available on the current page.

## Example Section 1

Lorem ipsum

### Example Subsection 1 a

Lorem ipsum

#### Example subsubsection 1 a I

#### Example subsubsection 1 a II

#### Example subsubsection 1 a III

### Example Subsection 1 b

Lorem ipsum

#### Example subsubsection 1 b I

#### Example subsubsection 1 b II

#### Example subsubsection 1 b III

### Example Subsection 1 c

Lorem ipsum

#### Example subsubsection 1 c I

#### Example subsubsection 1 c II



### **Example subsubsection 1 c III**

## **Example Section 2**

Lorem ipsum

### **Example Subsection 2 a**

Lorem ipsum

#### **Example subsubsection 2 a I**

#### **Example subsubsection 2 a II**

#### **Example subsubsection 2 a III**

### **Example Subsection 2 b**

Lorem ipsum

#### **Example subsubsection 2 b I**

#### **Example subsubsection 2 b II**

#### **Example subsubsection 2 b III**

### **Example Subsection 2 c**

Lorem ipsum

#### **Example subsubsection 2 c I**

#### **Example subsubsection 2 c II**

#### **Example subsubsection 2 c III**

## **Example Section 3**

Lorem ipsum

### **Example Subsection 3 a**

Lorem ipsum

#### **Example subsubsection 3 a I**

#### **Example subsubsection 3 a II**

#### **Example subsubsection 3 a III**

### **Example Subsection 3 b**

Lorem ipsum



### Example subsubsection 3 b I

### Example subsubsection 3 b II

### Example subsubsection 3 b III

### Example Subsection 3 c

Lorem ipsum

### Example subsubsection 3 c I

### Example subsubsection 3 c II

### Example subsubsection 3 c III

## 1.3.18 Assets

Sometimes you want to link to assets (e.g. docx files, images...) directly from Markdown files, and it is convenient to co-locate the asset next to the Markdown file using it.

Let's imagine the following file structure:

```
Your doc
/website/docs/myFeature.mdx

Some assets you want to use
/website/docs/assets/docusaurus-asset-example-banner.png
/website/docs/assets/docusaurus-asset-example.docx
```

## Images

You can display images in three different ways: Markdown syntax, CJS require, or ES imports syntax.

### Markdown syntax

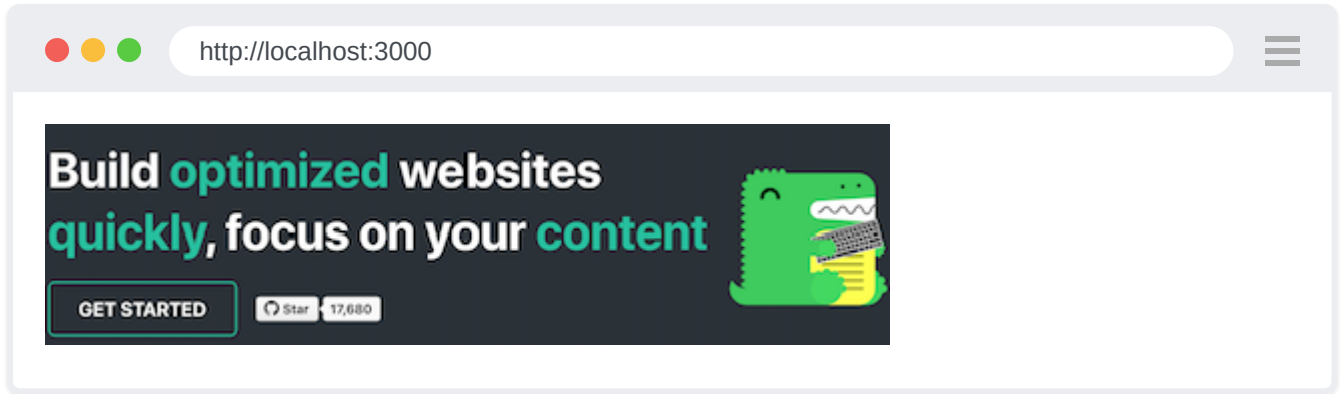
### CommonJS require

### Import statement

Display images using simple Markdown syntax:

```
![Example banner](./assets/docusaurus-asset-example-banner.png)
```

All of the above result in displaying the image:



### **i** NOTE

If you are using `@docusaurus/plugin-ideal-image`, you need to use the dedicated image component, as documented.

## Files

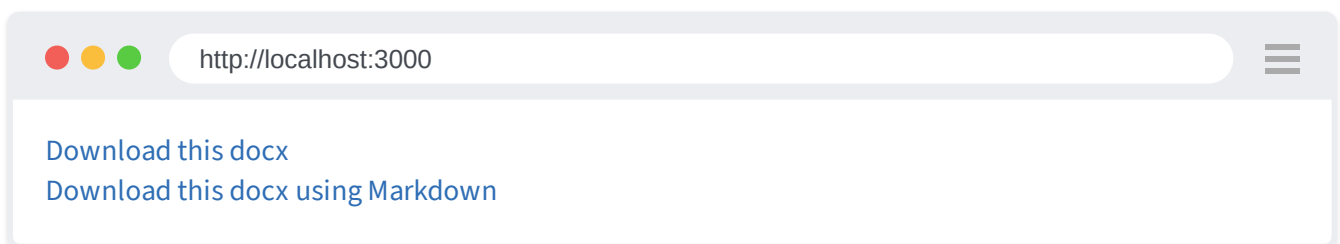
In the same way, you can link to existing assets by `require`'ing them and using the returned URL in `video`s, `a` anchor links, etc.

### # My Markdown page

```
 Download this docx
```

or

```
[Download this docx using Markdown](./assets/docusaurus-asset-example.docx)
```



### **!** MARKDOWN LINKS ARE ALWAYS FILE PATHS

If you use the Markdown image or link syntax, all asset paths will be resolved as file paths by Docusaurus and automatically converted to `require()` calls. You don't need to use `require()` in Markdown unless you use the JSX syntax, which you do have to handle yourself.

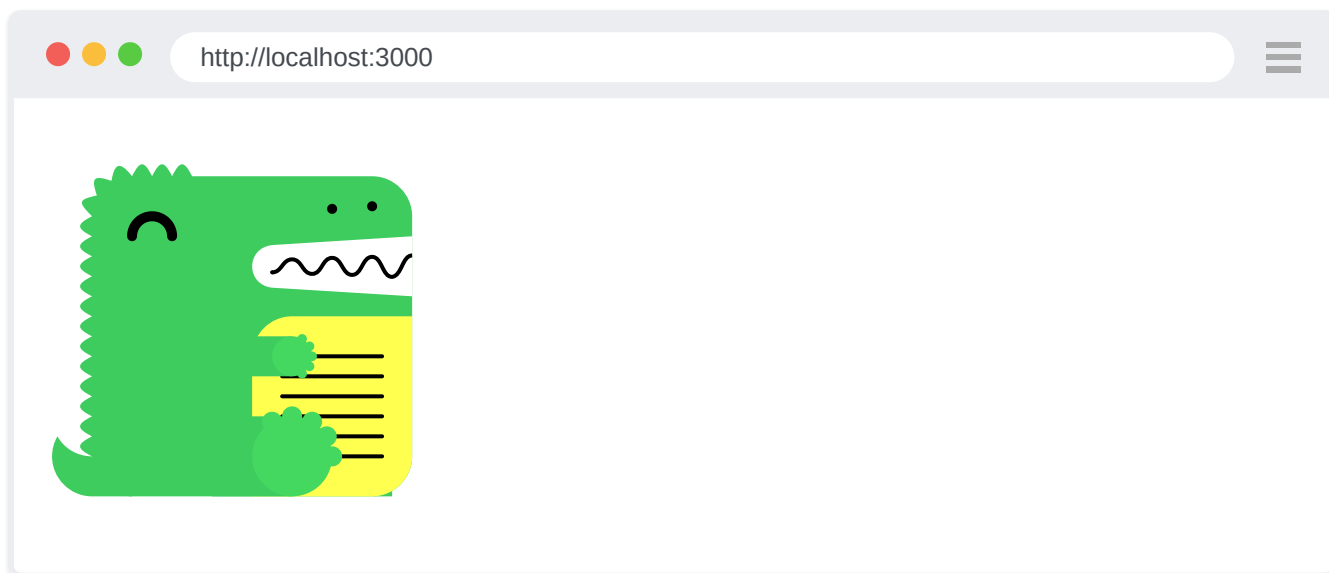
## Inline SVGs



Docusaurus supports inlining SVGs out of the box.

```
import DocusaurusSvg from './docusaurus.svg';

<DocusaurusSvg />;
```



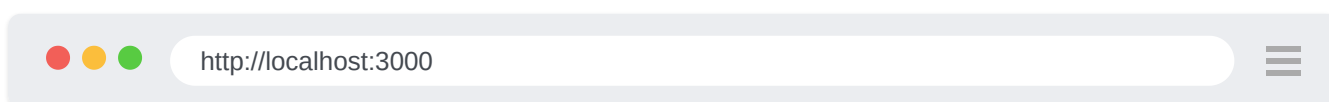
This can be useful if you want to alter the part of the SVG image via CSS. For example, you can change one of the SVG colors based on the current theme.

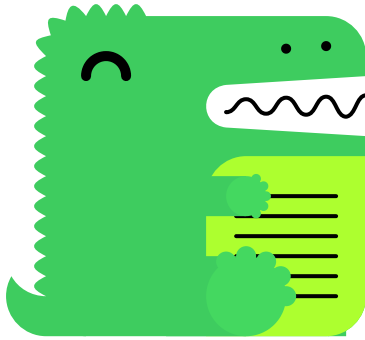
```
import DocusaurusSvg from './docusaurus.svg';

<DocusaurusSvg className="themedDocusaurus" />;
```

```
[data-theme='light'] .themedDocusaurus [fill='#FFFF50'] {
 fill: greenyellow;
}

[data-theme='dark'] .themedDocusaurus [fill='#FFFF50'] {
 fill: seagreen;
}
```



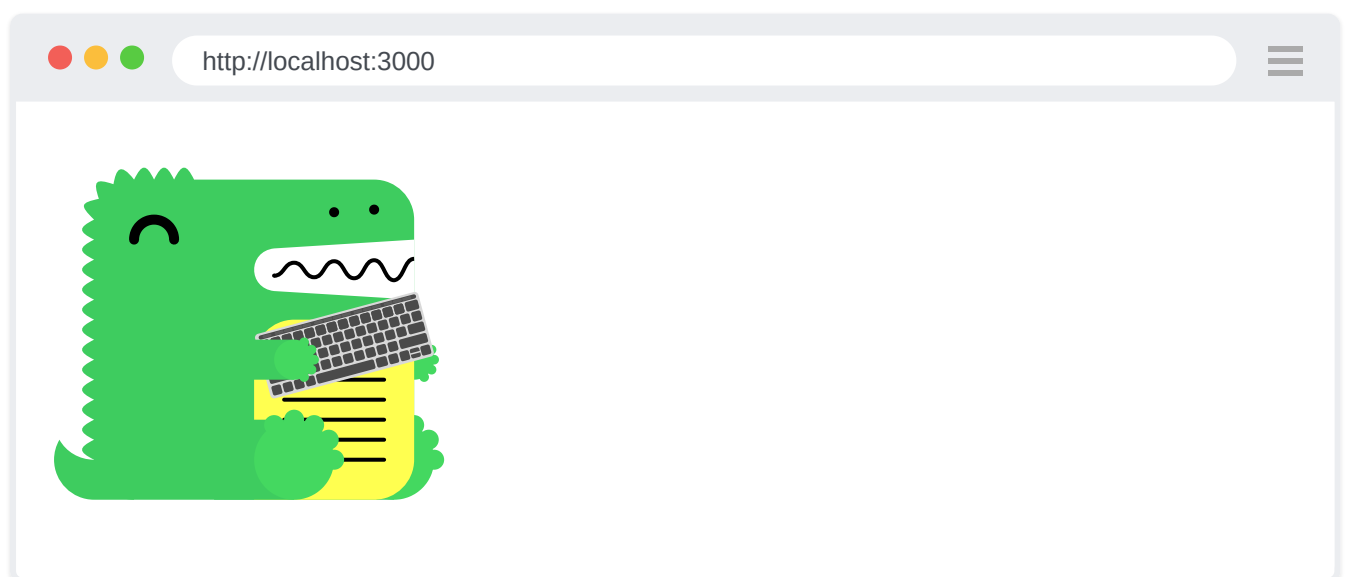


## Themed Images

Docusaurus supports themed images: the `ThemedImage` component (included in the themes) allows you to switch the image source based on the current theme.

```
import useBaseUrl from '@docusaurus/useBaseUrl';
import ThemedImage from '@theme/ThemedImage';

<ThemedImage
 alt="Docusaurus themed image"
 sources={{
 light: useBaseUrl('/img/docusaurus_light.svg'),
 dark: useBaseUrl('/img/docusaurus_dark.svg'),
 }}
/>;
```



### GitHub-style themed images

GitHub uses its own [image theming approach](#) with path fragments, which you can easily implement yourself.

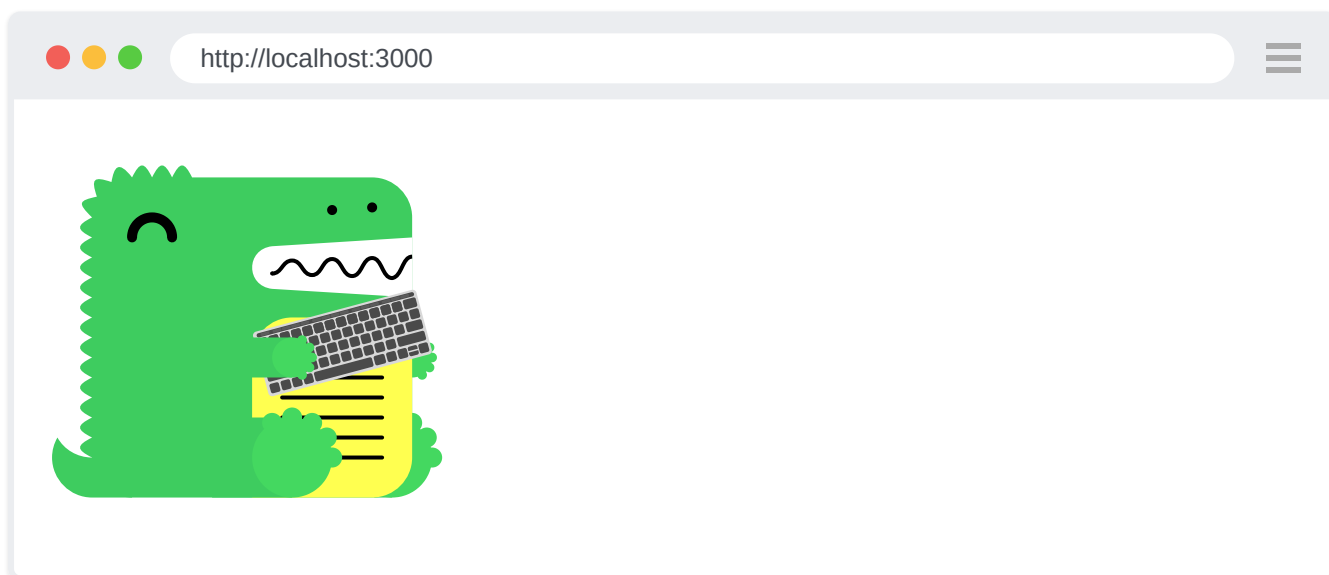


To toggle the visibility of an image using the path fragment (for GitHub, it's `#gh-dark-mode-only` and `#gh-light-mode-only`), add the following to your custom CSS (you can also use your own suffix if you don't want to be coupled to GitHub):

src/css/custom.css

```
[data-theme='light'] img[src$='#gh-dark-mode-only'],
[data-theme='dark'] img[src$='#gh-light-mode-only'] {
 display: none;
}
```

```
![Docusaurus themed image](/img/docusaurus_keytar.svg#gh-light-mode-only)!
[Docusaurus themed image](/img/docusaurus_speed.svg#gh-dark-mode-only)
```



## Static assets

If a Markdown link or image has an absolute path, the path will be seen as a file path and will be resolved from the static directories. For example, if you have configured `static directories` to be `['public', 'static']`, then for the following image:

my-doc.md

```
![An image from the static](/img/docusaurus.png)
```

Docusaurus will try to look for it in both `static/img/docusaurus.png` and `public/img/docusaurus.png`. The link will then be converted to a `require()` call instead of staying as





a URL. This is desirable in two regards:

1. You don't have to worry about the base URL, which Docusaurus will take care of when serving the asset;
2. The image enters Webpack's build pipeline and its name will be appended by a hash, which enables browsers to aggressively cache the image and improves your site's performance.

If you intend to write URLs, you can use the `pathname://` protocol to disable automatic asset linking.

```
! [banner] (pathname:///img/docusaurus-asset-example-banner.png)
```

This link will be generated as ``, without any processing or file existence checking.

## 1.3.19 Markdown links

There are two ways of adding a link to another page: through a **URL path** and a **file path**.

- `[URL path to another document](./installation)`
- `[file path to another document](./installation.mdx)`

URL paths are unprocessed by Docusaurus, and you can see them as directly rendering to `<a href="./installation">`, i.e. it will be resolved according to the page's URL location, rather than its file-system location.

If you want to reference another Markdown file **included by the same plugin**, you could use the relative path of the document you want to link to. Docusaurus' Markdown loader will convert the file path to the target file's URL path (and hence remove the `.md` extension).

For example, if you are in `docs/folder/doc1.md` and you want to reference `docs/folder/doc2.md`, `docs/folder/subfolder/doc3.md`, and `docs/otherFolder/doc4.md`:

`docs/folder/doc1.md`

I am referencing a `[document](doc2.mdx)`.

Reference to another `[document in a subfolder](subfolder/doc3.mdx)`.

`[Relative document](../otherFolder/doc4.mdx)` referencing works as well.

Relative file paths are resolved against the current file's directory. Absolute file paths, on the other hand, are resolved relative to the **content root**, usually `docs/`, `blog/`, or **localized ones** like `i18n/zh-Hans/plugin-content-docs/current`.



Here are some examples of file path links and how they get resolved, assuming the current file is `website/docs/category/source.mdx`:

- `[link](./target.mdx)` is resolved from the current file's directory `website/docs/category`.
- `[link](../target.mdx)` is resolved from the parent file's directory `website/docs`.
- `[link](/target.mdx)` is resolved from the docs content root `website/docs`, using in priority the localized docs.
- `[link](target.mdx)` is resolved from the current directory `website/docs/category`, then from the docs content roots, then from the site root.

Absolute file paths can also be relative to the site directory. However, beware that links that begin with `/docs/`, `/blog/` or `@site/` are **not portable** as you would need to manually update them if you create new doc versions or localize them:

- `[link](/docs/target.mdx)` is resolved from the site root `website` (⚠ less portable).
- `[link](@site/docs/target.mdx)` is relative to the site root `website` (⚠ less portable).

Using relative *file* paths (with `.md` extensions) instead of relative *URL* links provides the following benefits:

- Links will keep working on the GitHub interface and many Markdown editors
- You can customize the files' slugs without having to update all the links
- Moving files around the folders can be tracked by your editor, and some editors may automatically update file links
- A [versioned doc](#) will link to another doc of the exact same version
- Relative URL links are very likely to break if you update the `trailingSlash` config

### ⚠ WARNING

Markdown file references only work when the source and target files are processed by the same plugin instance. This is a technical limitation of our Markdown processing architecture and will be fixed in the future. If you are linking files between plugins (e.g. linking to a doc page from a blog post), you have to use URL links.

## 1.3.20 MDX Plugins

Sometimes, you may want to extend or tweak your Markdown syntax. For example:

- How do I embed youtube videos using the image syntax (``)?
- How do I style links that are on their own lines differently, e.g., as a social card?
- How do I make every page start with a copyright notice?

And the answer is: create an MDX plugin! MDX has a built-in [plugin system](#) that can be used to customize how the Markdown files will be parsed and transformed to JSX. There are three typical use-cases of MDX plugins:

- Using existing [remark plugins](#) or [rehype plugins](#);
- Creating remark/rehype plugins to transform the elements generated by existing MDX syntax;
- Creating remark/rehype plugins to introduce new syntaxes to MDX.



If you play with the [MDX playground](#), you would notice that the MDX transpilation has two intermediate steps: Markdown AST (MDAST), and Hypertext AST (HAST), before arriving at the final JSX output. MDX plugins also come in two forms:

- **Remark**: processes the Markdown AST.
- **Rehype**: processes the Hypertext AST.

**TIP**

Use plugins to introduce shorter syntax for the most commonly used JSX elements in your project. The [admonition](#) syntax that we offer is also generated by a Remark plugin, and you could do the same for your own use case.

## Default plugins

Docusaurus injects [some default Remark plugins](#) during Markdown processing. These plugins would:

- Generate the table of contents;
- Add anchor links to each heading;
- Transform images and links to `require()` calls.
- ...

These are all typical use-cases of Remark plugins, which can also be a source of inspiration if you want to implement your own plugin.

## Installing plugins

An MDX plugin is usually an npm package, so you install them like other npm packages using npm. Take the [math plugins](#) as an example.

**npm**

Yarn

pnpm

Bun

```
npm install --save remark-math@5 rehype-katex@6
```

### How are `remark-math` and `rehype-katex` different?

In case you are wondering how Remark and Rehype are different, here is a good example. `remark-math` operates on the Markdown AST, where it sees text like `$...$`, and all it does is transform that to the JSX `<span class="math math-inline">...</span>` without doing too much with the content. This decouples the extraction of math formulae from their rendering, which means you can



swap KaTeX<sup>KATEX</sup> out with other math renderers, like MathJax (with `rehype-mathjax`), just by replacing the Rehype plugin.

Next, the `rehype-katex` operates on the Hypertext AST where everything has been converted to HTML-like tags already. It traverses all the elements with `math` class and uses KaTeX<sup>KATEX</sup> to parse and render the content to actual HTML.

### ⚠ WARNING

Many official Remark/Rehype plugins are **ES Modules only**, a JavaScript module system, which Docusaurus supports. We recommend using an **ES Modules** config file to make it easier to import such packages.

Next, import your plugins and add them to the plugin options through plugin or preset config in `docusaurus.config.js`:

`docusaurus.config.js`

```
import remarkMath from 'remark-math';
import rehypeKatex from 'rehype-katex';

export default {
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 docs: {
 path: 'docs',
 remarkPlugins: [remarkMath],
 rehypePlugins: [rehypeKatex],
 },
],
],
],
};
```

### Using a CommonJS config file?

If you decide to use a CommonJS config file, it is possible to load those ES module plugins thanks to dynamic imports and an async config creator function:



docusaurus.config.js

```
module.exports = async function createConfigAsync() {
 return {
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 docs: {
 path: 'docs',
 remarkPlugins: [(await import('remark-math')).default],
 rehypePlugins: [(await import('rehype-katex')).default],
 },
 },
],
],
 };
};
```

## Configuring plugins

Some plugins can be configured and accept their own options. In that case, use the `[plugin, pluginOptions]` syntax, like this:

docusaurus.config.js

```
import rehypeKatex from 'rehype-katex';

export default {
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 docs: {
 rehypePlugins: [
 [rehypeKatex, {strict: false}],
],
 },
],
],
],
};
```



You should check your plugin's documentation for the options it supports.

## Creating new rehype/remark plugins

If there isn't an existing package that satisfies your customization need, you can create your own MDX plugin.

### NOTE

The writeup below is **not** meant to be a comprehensive guide to creating a plugin, but just an illustration of how to make it work with Docusaurus. Visit the [Remark](#) or [Rehype](#) documentation for a more in-depth explanation of how they work.

For example, let's make a plugin that visits every `h2` heading and adds a `Section X.` prefix. First, create your plugin source file anywhere—you can even publish it as a separate npm package and install it like explained above. We would put ours at `src/remark/section-prefix.js`. A remark/rehype plugin is just a function that receives the `options` and returns a `transformer` that operates on the AST.

```
import {visit} from 'unist-util-visit';

const plugin = (options) => {
 const transformer = async (ast) => {
 let number = 1;
 visit(ast, 'heading', (node) => {
 if (node.depth === 2 && node.children.length > 0) {
 node.children.unshift({
 type: 'text',
 value: `Section ${number}. `,
 });
 number++;
 }
 });
 };
 return transformer;
};

export default plugin;
```

You can now import your plugin in `docusaurus.config.js` and use it just like an installed plugin!

```
docusaurus.config.js
```



```
import sectionPrefix from './src/remark/section-prefix';

export default {
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 docs: {
 remarkPlugins: [sectionPrefix],
 },
 },
],
],
};
```

### TIP

The `transformer` has a second parameter `vfile` which is useful if you need to access the current Markdown file's path.

```
const plugin = (options) => {
 const transformer = async (ast, vfile) => {
 ast.children.unshift({
 type: 'text',
 value: `The current file path is ${vfile.path}`,
 });
 };
 return transformer;
};
```

Our `transformImage` plugin uses this parameter, for example, to transform relative image references to `require()` calls.

### NOTE

The default plugins of Docusaurus would operate before the custom remark plugins, and that means the images or links have been converted to JSX with `require()` calls already. For example, in the example above, the table of contents generated is still the same even when all `h2` headings are now prefixed by `Section X.`, because the TOC-generating plugin is called before our custom plugin. If you need to process the MDAST before the default plugins do, use the `beforeDefaultRemarkPlugins` and `beforeDefaultRehypePlugins`.



docusaurus.config.js

```
export default {
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 docs: {
 beforeDefaultRemarkPlugins: [sectionPrefix],
 },
 },
],
],
};
```

This would make the table of contents generated contain the `Section X.` prefix as well.

## 1.3.21 Math Equations

Mathematical equations can be rendered using [KaTeX](#).

See [below](#) how to activate them.

## Usage

Please read the [KaTeX](#) documentation for more details.

### Inline

Write inline math equations by wrapping LaTeX equations between `$`:

Let  $f\colon[a,b]\rightarrow\mathbb{R}$  be Riemann integrable. Let  $F\colon[a,b]\rightarrow\mathbb{R}$  be  $F(x)=\int_a^x f(t)\,dt$ . Then  $F$  is continuous, and at all  $x$  such that  $f$  is continuous at  $x$ ,  $F$  is differentiable at  $x$  with  $F'(x)=f(x)$ .



http://localhost:3000



Let  $f : [a, b] \rightarrow \mathbb{R}$  be Riemann integrable. Let  $F : [a, b] \rightarrow \mathbb{R}$  be  $F(x) = \int_a^x f(t) dt$ . Then  $F$  is continuous, and at all  $x$  such that  $f$  is continuous at  $x$ ,  $F$  is differentiable at  $x$  with  $F'(x) = f(x)$ .





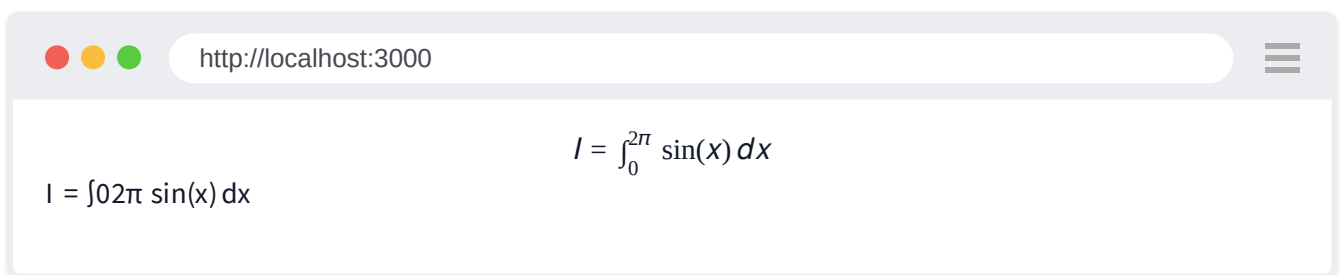
## Blocks

For equation block or display mode, use ````math` fenced code blocks.

```
```math
I = \int_0^{2\pi} \sin(x) \, dx
```
```

You can also use line breaks and `$$`, although this syntax relies on a [Markdown syntax extension](#) and is less portable:

```
$$
I = \int_0^{2\pi} \sin(x) \, dx
$$
```



## Enabling math equations

Enable KaTeX:

1. Install the `remark-math` and `rehype-katex` plugins:

**npm**

Yarn

pnpm

Bun

```
npm install --save remark-math@6 rehype-katex@7
```



### WARNING

Make sure to use `remark-math 6` and `rehype-katex 7` for Docusaurus v3 (using MDX v3). We can't guarantee other versions will work.

2. These 2 plugins are **only available as ES Modules**. We recommended to use an **ES Modules** config file:



ES module docusaurus.config.js

```
import remarkMath from 'remark-math';
import rehypeKatex from 'rehype-katex';

export default {
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 docs: {
 path: 'docs',
 remarkPlugins: [remarkMath],
 rehypePlugins: [rehypeKatex],
 },
],
],
],
};
```

### Using a CommonJS config file?

If you decide to use a CommonJS config file, it is possible to load those ES module plugins thanks to dynamic imports and an async config creator function:

CommonJS module docusaurus.config.js



```
module.exports = async function createConfigAsync() {
 return {
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 docs: {
 path: 'docs',
 remarkPlugins: [(await import('remark-math')).default],
 rehypePlugins: [(await import('rehype-
katex')).default],
 },
],
],
],
 };
};
```

3. Include the KaTeX CSS in your config under `stylesheets`:

```
export default {
 //...
 stylesheets: [
 {
 href:
'https://cdn.jsdelivr.net/npm/katex@0.13.24/dist/katex.min.css',
 type: 'text/css',
 integrity:
 'sha384-
odtC+0UGzzFL/6PNoE8rX/SPcQDXBJ+uRepguP4QkPCm2LBxH3FA3y+fKSiJ+AmM',
 crossorigin: 'anonymous',
 },
],
};
```

### See a config file example

docusaurus.config.js



```
import remarkMath from 'remark-math';
import rehypeKatex from 'rehype-katex';

export default {
 title: 'Docusaurus',
 tagline: 'Build optimized websites quickly, focus on your content',
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 docs: {
 path: 'docs',
 remarkPlugins: [remarkMath],
 rehypePlugins: [rehypeKatex],
 },
],
],
],
 stylesheets: [
 {
 href:
 'https://cdn.jsdelivr.net/npm/katex@0.13.24/dist/katex.min.css',
 type: 'text/css',
 integrity:
 'sha384-odtC+0UGzzFL/6PNoE8rX/SPcQDXBJ+uRepguP4QkPCm2LBxH3FA3y+fKSiJ+AmM',
 crossorigin: 'anonymous',
 },
],
};
```

## Self-hosting KaTeX assets

Loading stylesheets, fonts, and JavaScript libraries from CDN sources is a good practice for popular libraries and assets, since it reduces the amount of assets you have to host. In case you prefer to self-host the `katex.min.css` (along with required KaTeX fonts), you can download the latest version from [KaTeX GitHub releases](#), extract and copy `katex.min.css` and `fonts` directory (only `.woff2` font types should be enough) to your site's `static` directory, and in `docusaurus.config.js`, replace the stylesheet's `href` from the CDN URL to your local path (say, `/katex/katex.min.css`).

`docusaurus.config.js`



```
export default {
 stylesheets: [
 {
 href: '/katex/katex.min.css',
 type: 'text/css',
 },
],
};
```

## 1.3.22 Diagrams

Diagrams can be rendered using [Mermaid](#) in a code block.

## Installation

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)

```
npm install --save @docusaurus/theme-mermaid
```

Enable Mermaid functionality by adding plugin `@docusaurus/theme-mermaid` and setting `markdown.mermaid` to `true` in your `docusaurus.config.js`.

docusaurus.config.js

```
export default {
 markdown: {
 mermaid: true,
 },
 themes: ['@docusaurus/theme-mermaid'],
};
```

## Usage

Add a code block with language `mermaid`:

Example Mermaid diagram



```
```mermaid
graph TD;
  A-->B;
  A-->C;
  B-->D;
  C-->D;
```
```

See the [Mermaid syntax documentation](#) for more information on the Mermaid syntax.

## Theming

The diagram dark and light themes can be changed by setting `mermaid.theme` values in the `themeConfig` in your `docusaurus.config.js`. You can set themes for both light and dark mode.

docusaurus.config.js

```
export default {
 themeConfig: {
 mermaid: {
 theme: {light: 'neutral', dark: 'forest'},
 },
 },
};
```

See the [Mermaid theme documentation](#) for more information on theming Mermaid diagrams.

## Mermaid Config

Options in `mermaid.options` will be passed directly to `mermaid.initialize`:

docusaurus.config.js



```
export default {
 themeConfig: {
 mermaid: {
 options: {
 maxTextSize: 50,
 },
 },
 },
};
```

See the [Mermaid config documentation](#) and the [Mermaid config types](#) for the available config options.

## Dynamic Mermaid Component

To generate dynamic diagrams, you can use the `Mermaid` component:

Example of dynamic Mermaid component

```
import Mermaid from '@theme/Mermaid';

<Mermaid
 value={`graph TD;
 A-->B;
 A-->C;
 B-->D;
 C-->D;`} />
```

## Layouts

Mermaid supports different [layout engines](#):

- The `dagre` layout engine is supported by default in Docusaurus.
- The `elk` layout engine is heavier and can be enabled by installing the optional `@mermaid-js/layout-elk` dependency.



```
```mermaid
---
config:
  layout: elk
---
graph TD;
  A-->B;
  A-->C;
  B-->D;
  C-->D;
```
```

## 1.3.23 Styling and Layout

**TIP**

This section is focused on styling through stylesheets. For more advanced customizations (DOM structure, React code...), refer to the [swizzling guide](#).

A Docusaurus site is a single-page React application. You can style it the way you style React apps.

There are a few approaches/frameworks which will work, depending on your preferences and the type of website you are trying to build. Websites that are highly interactive and behave more like web apps will benefit from more modern styling approaches that co-locate styles with the components. Component styling can also be particularly useful when you wish to customize or swizzle a component.

## Global styles

This is the most traditional way of styling that most developers (including non-front-end developers) would be familiar with. It works fine for small websites that do not have much customization.

If you're using `@docusaurus/preset-classic`, you can create your own CSS files (e.g. `/src/css/custom.css`) and import them globally by passing them as an option of the classic theme.

Your project structure should look like this:

```
my-website/
├── src/
│ └── css/
│ └── custom.css ← add your custom styles here
├── docusaurus.config.js
└── package.json
```





docusaurus.config.js

```
export default {
 // ...
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 theme: {
 customCss: ['./src/css/custom.css'],
 },
],
],
],
};
```

Any CSS you write within that file will be available globally and can be referenced directly using string literals.

/src/css/custom.css

```
.purple-text {
 color: rebeccapurple;
}
```

```
function MyComponent() {
 return (
 <main>
 <h1 className="purple-text">Purple Heading!</h1>
 </main>
);
}
```

If you want to add CSS to any element, you can open the DevTools in your browser to inspect its class names. Class names come in several kinds:

- **Theme class names.** These class names are listed exhaustively in [the next subsection](#). They don't have any default properties. You should always prioritize targeting those stable class names in your custom CSS.
- **Infima class names.** These class names are found in the classic theme and usually follow the [BEM convention](#) of `block__element--modifier`. They are usually stable but are still considered



implementation details, so you should generally avoid targeting them. However, you can [modify Infima CSS variables](#).

- **CSS module class names.** These class names end with a hash which may change over time (`codeBlockContainer_RIuc`). They are considered implementation details and you should almost always avoid targeting them in your custom CSS. If you must, you can use an [attribute selector](#) (`[class*='codeBlockContainer']`) that ignores the hash.

## Theme Class Names

We provide some stable CSS class names for robust and maintainable global layout styling. These names are theme-agnostic and meant to be targeted by custom CSS.



### TIP

If you can't find a way to create a robust CSS selector, please [report your customization use-case](#) and we will consider adding new class names.

### Exhaustive list of stable class names



```
export const ThemeClassNames = {
 page: {
 blogListPage: 'blog-list-page',
 blogPostPage: 'blog-post-page',
 blogTagsListPage: 'blog-tags-list-page',
 blogTagPostListPage: 'blog-tags-post-list-page',
 blogAuthorsListPage: 'blog-authors-list-page',
 blogAuthorsPostsPage: 'blog-authors-posts-page',
 docsDocPage: 'docs-doc-page',
 docsTagsListPage: 'docs-tags-list-page',
 docsTagDocListPage: 'docs-tags-doc-list-page',
 mdxPage: 'mdx-page',
 },
 wrapper: {
 main: 'main-wrapper',
 blogPages: 'blog-wrapper',
 docsPages: 'docs-wrapper',
 mdxPages: 'mdx-wrapper',
 },
 common: {
 editThisPage: 'theme-edit-this-page',
 lastUpdated: 'theme-last-updated',
 backToTopButton: 'theme-back-to-top-button',
 codeBlock: 'theme-code-block',
 admonition: 'theme-admonition',
 unlistedBanner: 'theme-unlisted-banner',
 draftBanner: 'theme-draft-banner',
 admonitionType: (type: string) => `theme-admonition-${type}`,
 },
 announcementBar: {
 container: 'theme-announcement-bar',
 },
 tabs: {
 container: 'theme-tabs-container',
 },
 layout: {
 navbar: {
 container: 'theme-layout-navbar',
 containerLeft: 'theme-layout-navbar-left',
 containerRight: 'theme-layout-navbar-right',
 mobileSidebar: {
 container: 'theme-layout-navbar-sidebar',
 panel: 'theme-layout-navbar-sidebar-panel',
 },
 },
 main: {
 container: 'theme-layout-main',
 },
 },
}
```



```
 footer: {
 container: 'theme-layout-footer',
 column: 'theme-layout-footer-column',
 },
 },
 docs: {
 docVersionBanner: 'theme-doc-version-banner',
 docVersionBadge: 'theme-doc-version-badge',
 docBreadcrumbs: 'theme-doc-breadcrumbs',
 docMarkdown: 'theme-doc-markdown',
 docTocMobile: 'theme-doc-toc-mobile',
 docTocDesktop: 'theme-doc-toc-desktop',
 docFooter: 'theme-doc-footer',
 docFooterTagsRow: 'theme-doc-footer-tags-row',
 docFooterEditMetaRow: 'theme-doc-footer-edit-meta-row',
 docSidebarContainer: 'theme-doc-sidebar-container',
 docSidebarMenu: 'theme-doc-sidebar-menu',
 docSidebarItemCategory: 'theme-doc-sidebar-item-category',
 docSidebarItemLink: 'theme-doc-sidebar-item-link',
 docSidebarItemCategoryLevel: (level: number) =>
 `theme-doc-sidebar-item-category-level-${level}` as const,
 docSidebarItemLinkLevel: (level: number) =>
 `theme-doc-sidebar-item-link-level-${level}` as const,
 docCard: {
 container: 'theme-doc-card-container',
 heading: 'theme-doc-card-heading',
 icon: 'theme-doc-card-icon',
 title: 'theme-doc-card-title',
 description: 'theme-doc-card-description',
 },
 },
},
blog: {
 blogFooterTagsRow: 'theme-blog-footer-tags-row',
 blogFooterEditMetaRow: 'theme-blog-footer-edit-meta-row',
},
pages: {
 pageFooterEditMetaRow: 'theme-pages-footer-edit-meta-row',
},
} as const;
```

## Styling your site with Infima

`@docusaurus/preset-classic` uses [Infima](#) as the underlying styling framework. Infima provides a flexible layout and common UI components styling suitable for content-centric websites (blogs, documentation, landing pages). For more details, check out the [Infima website](#).

When you scaffold your Docusaurus project with `create-docusaurus`, the website will be generated with basic Infima stylesheets and default styling. You can override Infima CSS variables globally.



```
/src/css/custom.css
```

```
:root {
 --ifm-color-primary: #25c2a0;
 --ifm-code-font-size: 95%;
}
```

Infima uses 7 shades of each color. We recommend using [ColorBox](#) to find the different shades of colors for your chosen primary color.

Alternatively, use the following tool to generate the different shades for your website and copy the variables into `/src/css/custom.css`.

**TIP**

Aim for at least [WCAG-AA contrast ratio](#) for the primary color to ensure readability. Use the Docusaurus website itself to preview how your color palette would look like. You can use alternative palettes in dark mode because one color doesn't usually work in both light and dark mode.

Primary Color:

[Edit dark mode](#)[Reset](#)

Background:



| CSS Variable Name                         | Hex                                                                                         | Adjustment | Contrast Rating                                                                            |
|-------------------------------------------|---------------------------------------------------------------------------------------------|------------|--------------------------------------------------------------------------------------------|
| <code>--ifm-color-primary-lightest</code> |  #3cad6e | -30        | Fail  |
| <code>--ifm-color-primary-lighter</code>  |  #359962 | -15        | Fail  |
| <code>--ifm-color-primary-light</code>    |  #33925d | -10        | Fail  |
| <code>--ifm-color-primary</code>          |  #2e8555 | 0          | AA    |
| <code>--ifm-color-primary-dark</code>     |  #29784c | 10         | AA    |
| <code>--ifm-color-primary-darker</code>   |  #277148 | 15         | AA    |
| <code>--ifm-color-primary-darkest</code>  |  #205d3b | 30         | AAA   |

Replace the variables in `src/css/custom.css` with these new variables.

```
/src/css/custom.css
```



```
:root {
 --ifm-color-primary: #2e8555;
 --ifm-color-primary-dark: #29784c;
 --ifm-color-primary-darker: #277148;
 --ifm-color-primary-darkest: #205d3b;
 --ifm-color-primary-light: #33925d;
 --ifm-color-primary-lighter: #359962;
 --ifm-color-primary-lightest: #3cad6e;
}
```

## Dark Mode

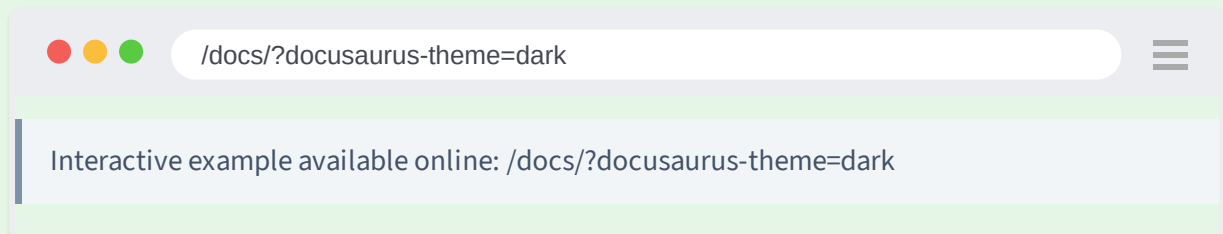
In light mode, the `<html>` element has a `data-theme="light"` attribute; in dark mode, it's `data-theme="dark"`. Therefore, you can scope your CSS to dark-mode-only by targeting `html` with a specific attribute.

```
/* Overriding root Infima variables */
[data-theme='dark'] {
 --ifm-color-primary: #4e89e8;
}
/* Styling one class specially in dark mode */
[data-theme='dark'] .purple-text {
 color: plum;
}
```

### TIP

It is possible to initialize the Docusaurus theme directly from a `docusaurus-theme` query string parameter.

Examples:





/docs/configuration?docusaurus-theme=light

Interactive example available online: /docs/configuration?docusaurus-theme=light

## Data Attributes

It is possible to inject `<html>` data attributes with query string parameters following the `docusaurus-data-<key>` pattern. This gives you the flexibility to style a page differently based on the query string.

For example, let's render one of our pages with a red border and no navbar:

/src/css/custom.css

```
html[data-navbar='false'] .navbar {
 display: none;
}

html[data-red-border] div#__docusaurus {
 border: red solid thick;
}
```

/docs/?docusaurus-data-navbar=false&docusaurus-data-red-border

Interactive example available online: /docs/?docusaurus-data-navbar=false&docusaurus-data-red-border

### IFRAME MODE

If you plan to embed some Docusaurus pages on another site through an iframe, it can be useful to create an alternative display mode and use iframe urls such as `https://mysite.com/docs/myDoc?docusaurus-data-mode=iframe`. It is your responsibility to provide the additional styles and decide which UI elements you want to keep or hide.

## Mobile View

Docusaurus uses `996px` as the cutoff between mobile screen width and desktop. If you want your layout to be different in the mobile view, you can use media queries.



```
.banner {
 padding: 4rem;
}
/** In mobile view, reduce the padding */
@media screen and (max-width: 996px) {
 .heroBanner {
 padding: 2rem;
 }
}
```



### CUSTOMIZING THE BREAKPOINT

Some React components, such as the header and the sidebar, implement different JavaScript logic when in mobile view. If you change the breakpoint value in your custom CSS, you probably also want to update the invocations of the `useWindowSize` hook by [swizzling](#) the components it's used in and passing an explicit option argument.

## CSS modules

To style your components using [CSS Modules](#), name your stylesheet files with the `.module.css` suffix (e.g. `welcome.module.css`). Webpack will load such CSS files as CSS modules and you have to reference the class names as properties of the imported CSS module (as opposed to using plain strings). This is similar to the convention used in [Create React App](#).

styles.module.css

```
.main {
 padding: 12px;
}

.heading {
 font-weight: bold;
}

.contents {
 color: #ccc;
}
```





```
import styles from './styles.module.css';

function MyComponent() {
 return (
 <main className={styles.main}>
 <h1 className={styles.heading}>Hello!</h1>
 <article className={styles.contents}>Lorem Ipsum</article>
 </main>
);
}
```

The class names will be processed by webpack into a globally unique class name during build.

## CSS-in-JS

### ⚠ WARNING

CSS-in-JS support is a work in progress, so libs like MUI may have display quirks. [Welcoming PRs](#).

## Sass/SCSS

To use Sass/SCSS as your CSS preprocessor, install the unofficial Docusaurus plugin `docusaurus-plugin-sass`. This plugin works for both global styles and the CSS modules approach:

1. Install `docusaurus-plugin-sass`:

**npm**    **Yarn**    **pnpm**    **Bun**

```
npm install --save docusaurus-plugin-sass sass
```

2. Include the plugin in your `docusaurus.config.js` file:

docusaurus.config.js

```
export default {
 // ...
 plugins: ['docusaurus-plugin-sass'],
 // ...
};
```



3. Write and import your stylesheets in Sass/SCSS as normal.

## Global styles using Sass/SCSS

You can now set the `customCss` property of `@docusaurus/preset-classic` to point to your Sass/SCSS file:

docusaurus.config.js

```
export default {
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 // ...
 theme: {
 customCss: ['./src/css/custom.scss'],
 },
 // ...
 },
],
],
};
```

## Modules using Sass/SCSS

Name your stylesheet files with the `.module.scss` suffix (e.g. `welcome.module.scss`) instead of `.css`. Webpack will use `sass-loader` to preprocess your stylesheets and load them as CSS modules.

styles.module.scss

```
.main {
 padding: 12px;

 article {
 color: #ccc;
 }
}
```



```
import styles from './styles.module.scss';

function MyComponent() {
 return (
 <main className={styles.main}>
 <article>Lorem Ipsum</article>
 </main>
);
}
```

## TypeScript support

To enable TypeScript support for Sass/SCSS modules, the TypeScript configuration should be updated to add the `docusaurus-plugin-sass` type definitions. This can be done in the `tsconfig.json` file:

```
{
 "extends": "@docusaurus/tsconfig",
 "compilerOptions": {
 ...
+ "types": ["docusaurus-plugin-sass"]
 }
}
```

## 1.3.24 Swizzling

In this section, we will introduce how customization of layout is done in Docusaurus.

Déjà vu...?

This section is similar to [Styling and Layout](#), but this time, we will customize React components themselves, rather than what they look like. We will talk about a central concept in Docusaurus: **swizzling**, which allows **deeper site customizations**.

In practice, swizzling permits to **swap a theme component with your own implementation**, and it comes in 2 patterns:

- **Ejecting**: creates a **copy** of the original theme component, which you can fully **customize**
- **Wrapping**: creates a **wrapper** around the original theme component, which you can **enhance**

### Why is it called swizzling?

The name comes from Objective-C and Swift-UI: [method swizzling](#) is the process of changing the implementation of an existing selector (method).



**For Docusaurus, component swizzling means providing an alternative component that takes precedence over the component provided by the theme.**

You can think of it as [Monkey Patching](#) for React components, enabling you to override the default implementation. Gatsby has a similar concept called [theme shadowing](#).

To gain a deeper understanding of this, you have to understand [how theme components are resolved](#).

## Swizzling Process

### Overview

Docusaurus provides a convenient **interactive CLI** to swizzle components. You generally only need to remember the following command:

**npm**    **Yarn**    **pnpm**    **Bun**

```
npm run swizzle
```

It will generate a new component in your `src/theme` directory, which should look like this example:

**Ejecting**    **Wrapping**

src/theme/SomeComponent.js

```
import React from 'react';

export default function SomeComponent(props) {
 // You can fully customize this implementation
 // including changing the JSX, CSS and React hooks
 return (
 <div className="some-class">
 <h1>Some Component</h1>
 <p>Some component implementation details</p>
 </div>
);
}
```

To get an overview of all the themes and components available to swizzle, run:

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)

```
npm run swizzle -- --list
```

Use `--help` to see all available CLI options, or refer to the reference [swizzle CLI documentation](#).



### REMOVING UNNEEDED SWIZZLED COMPONENTS

If you decide that a previously swizzled component is no longer necessary, you can simply remove its file(s) from the `src/theme` directory. After removing the component, make sure to restart your development server to ensure the changes are properly reflected.



### NOTE

After swizzling a component, **restart your dev server** in order for Docusaurus to know about the new component.



### PREFER STAYING ON THE SAFE SIDE

Be sure to understand [which components are safe to swizzle](#). Some components are **internal implementation details** of a theme.



### INFO

`docusaurus swizzle` is only an automated way to help you swizzle the component. You can also create the `src/theme/SomeComponent.js` file manually, and Docusaurus will [resolve it](#). There's no internal magic behind this command!

## Ejecting

Ejecting a theme component is the process of **creating a copy** of the original theme component, which you can **fully customize and override**.

To eject a theme component, use the swizzle CLI interactively, or with the `--eject` option:

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)

```
npm run swizzle [theme name] [component name] -- --eject
```



An example:

**npm**

**Yarn**

**pnpm**

**Bun**

```
npm run swizzle @docusaurus/theme-classic Footer -- --eject
```

This will copy the current `<Footer />` component's implementation to your site's `src/theme` directory. Docusaurus will now use this `<Footer>` component copy instead of the original one. You are now free to completely re-implement the `<Footer>` component.

src/theme/Footer/index.js

```
import React from 'react';

export default function Footer(props) {
 return (
 <footer>
 <h1>This is my custom site footer</h1>
 <p>And it is very different from the original</p>
 </footer>
);
}
```

### ⚠ WARNING

Ejecting an **unsafe** component can sometimes lead to copying a large amount of internal code, which you now have to maintain yourself. It can make Docusaurus upgrades more difficult, as you will need to migrate your customizations if the props received or internal theme APIs used have changed.

**Prefer wrapping whenever possible:** the amount of code to maintain is smaller.

### 💡 RE-SWIZZLING

To keep ejected components up-to-date after a Docusaurus upgrade, re-run the eject command and compare the changes with `git diff`. You are also recommended to write a brief comment at the top of the file explaining what changes you have made, so that you could more easily re-apply your changes after re-ejection.

## Wrapping



Wrapping a theme component is the process of **creating a wrapper** around the original theme component, which you can **enhance**.

To wrap a theme component, use the swizzle CLI interactively, or with the `--wrap` option:

**npm**   **Yarn**   **pnpm**   **Bun**

---

```
npm run swizzle [theme name] [component name] -- --wrap
```

An example:

**npm**   **Yarn**   **pnpm**   **Bun**

---

```
npm run swizzle @docusaurus/theme-classic Footer -- --wrap
```

This will create a wrapper in your site's `src/theme` directory. Docusaurus will now use the `<FooterWrapper>` component instead of the original one. You can now add customizations around the original component.

src/theme/Footer/index.js

```
import React from 'react';
import Footer from '@theme-original/Footer';

export default function FooterWrapper(props) {
 return (
 <>
 <section>
 <h2>Extra section</h2>
 <p>This is an extra section that appears above the original
footer</p>
 </section>
 <Footer {...props} />
 </>
);
}
```



### What is this `@theme-original` thing?

Docusaurus uses [theme aliases](#) to resolve the theme components to use. The newly created wrapper takes the `@theme/SomeComponent` alias. `@theme-original/SomeComponent` permits to import original component that the wrapper shadows without creating an infinite import loop where the wrapper imports itself.



#### TIP

Wrapping a theme is a great way to **add extra components around existing one** without [ejecting](#) it. For example, you can easily add a custom comment system under each blog post:

src/theme/BlogPostItem.js

```
import React from 'react';
import BlogPostItem from '@theme-original/BlogPostItem';
import MyCustomCommentSystem from '@site/src/MyCustomCommentSystem';

export default function BlogPostItemWrapper(props) {
 return (
 <>
 <BlogPostItem {...props} />
 <MyCustomCommentSystem />
 </>
);
}
```

## What is safe to swizzle?

With great power comes great responsibility

Some theme components are **internal implementation details** of a theme. Docusaurus allows you to swizzle them, but it **might be risky**.

### Why is it risky?

Theme authors (including us) might have to update their theme over time: changing the component props, name, file system location, types... For example, consider a component that receives two props `name` and `age`, but after a refactor, it now receives a `person` prop with the above two properties. Your component, which still expects these two props, will render `undefined` instead.





Moreover, internal components may simply disappear. If a component is called `Sidebar` and it's later renamed to `DocSidebar`, your swizzled component will be completely ignored.

**Theme components marked as unsafe may change in a backward-incompatible way between theme minor versions.** When upgrading a theme (or Docusaurus), your customizations might **behave unexpectedly**, and can even **break your site**.

For each theme component, the swizzle CLI will indicate **3 different levels of safety** declared by theme authors:

- **Safe:** this component is safe to be swizzled, its public API is considered stable, and no breaking changes should happen within a theme **major version**
- **Unsafe:** this component is a theme implementation detail, not safe to be swizzled, and breaking changes might happen within a theme **minor version**
- **Forbidden:** the swizzle CLI will prevent you from swizzling this component, because it is not designed to be swizzled at all

#### NOTE

Some components might be safe to wrap, but not safe to eject.

#### INFO

Don't be too **afraid to swizzle unsafe components**: just keep in mind that **breaking changes** might happen, and you might need to upgrade your customizations manually on minor version upgrades.

#### REPORT YOUR USE-CASE

If you have a **strong use-case for swizzling an unsafe component**, please [report it here](#) and we will work together to find a solution to make it safe.

## Which component should I swizzle?

It is not always clear which component you should swizzle exactly to achieve the desired result.

`@docusaurus/theme-classic`, which provides most of the theme components, has about [100 components!](#)

#### TIP

To print an overview of all the `@docusaurus/theme-classic` components:

**npm**

**Yarn**

**pnpm**

**Bun**



```
npm run swizzle @docusaurus/theme-classic -- --list
```

You can follow these steps to locate the appropriate component to swizzle:

1. **Component description.** Some components provide a short description, which is a good way to find the right one.
2. **Component name.** Official theme components are semantically named, so you should be able to infer its function from the name. The swizzle CLI allows you to enter part of a component name to narrow down the available choices. For example, if you run `yarn swizzle @docusaurus/theme-classic`, and enter `Doc`, only the docs-related components will be listed.
3. **Start with a higher-level component.** Components form a tree with some components importing others. Every route will be associated with one top-level component that the route will render (most of them listed in [Routing in content plugins](#)). For example, all blog post pages have `@theme/BlogPostPage` as the topmost component. You can start with swizzling this component, and then go down the component tree to locate the component that renders just what you are targeting. Don't forget to unswizzle the rest by deleting the files after you've found the correct one, so you don't maintain too many components.
4. **Read the [theme source code](#)** and use search wisely.

### JUST ASK!

If you still have no idea which component to swizzle to achieve the desired effect, you can reach out for help in one of our [support channels](#).

We also want to understand better your fanciest customization use-cases, so please [report them](#).

## Do I need to swizzle?

Swizzling ultimately means you have to maintain some additional React code that interact with Docusaurus internal APIs. If you can, think about the following alternatives when customizing your site:

1. **Use CSS.** CSS rules and selectors can often help you achieve a decent degree of customization. Refer to [styling and layout](#) for more details.
2. **Use translations.** It may sound surprising, but translations are ultimately just a way to customize the text labels. For example, if your site's default language is `en`, you can still run `yarn write-translations -l en` and edit the `code.json` emitted. Refer to the [i18n tutorial](#) for more details.

### TIP

**The smaller, the better.** If swizzling is inevitable, prefer to swizzle only the relevant part and maintain as little code on your own as possible. Swizzling a small component often means less risk of **breaking changes** during upgrade.



Wrapping is also a far safer alternative to ejecting.

## Wrapping your site with `<Root>`

The `<Root>` component is rendered at the **very top** of the React tree, above the theme `<Layout>`, and **never unmounts**. It is the perfect place to add stateful logic that should not be re-initialized across navigations (user authentication status, shopping cart state...).

Swizzle it **manually** by creating a file at `src/theme/Root.js`:

`src/theme/Root.js`

```
import React from 'react';

// Default implementation, that you can customize
export default function Root({children}) {
 return <>{children}</>;
}
```



**TIP**

Use this component to render React Context providers.

### 1.3.25 Static Assets

Static assets are the non-code files that are directly copied to the build output. They include images, stylesheets, favicons, fonts, etc.

By default, you are suggested to put these assets in the `static` folder. Every file you put into **that directory will be copied** into the root of the generated `build` folder with the directory hierarchy preserved. E.g. if you add a file named `sun.jpg` to the static folder, it will be copied to `build/sun.jpg`.

This means that:

- for site `baseUrl: '/'`, the image `/static/img/docusaurus.png` will be served at `/img/docusaurus.png`.
- for site `baseUrl: '/subpath/'`, the image `/static/img/docusaurus.png` will be served at `/subpath/img/docusaurus.png`.

You can customize the static directory sources in `docusaurus.config.js`. For example, we can add `public` as another possible path:

`docusaurus.config.js`



```
export default {
 title: 'My site',
 staticDirectories: ['public', 'static'],
 // ...
};
```

Now, all files in `public` as well as `static` will be copied to the build output.

## Referencing your static asset

### In JSX

In JSX, you can reference assets from the `static` folder in your code using absolute URLs, but this is not ideal because changing the site `baseUrl` will **break those links**. For the image `` served at `https://example.com/test`, the browser will try to resolve it from the URL root, i.e. as `https://example.com/img/docusaurus.png`, which will fail because it's actually served at `https://example.com/test/img/docusaurus.png`.

You can `import()` or `require()` the static asset (recommended), or use the `useBaseUrl` utility function: both prepend the `baseUrl` to paths for you.

Examples:

MyComponent.js

```
import DocusaurusImageUrl from '@site/static/img/docusaurus.png';

;
```

MyComponent.js

```

```

MyComponent.js

```
import useBaseUrl from '@docusaurus/useBaseUrl';

;
```



You can also import SVG files: they will be transformed into React components.

MyComponent.js

```
import DocusaurusLogoWithKeytar from
 '@site/static/img/docusaurus_keytar.svg';

<DocusaurusLogoWithKeytar title="Docusaurus Logo" className="logo" />;
```

## In Markdown

In Markdown, you can stick to using absolute paths when writing links or images **in Markdown syntax** because Docusaurus handles them as `require` calls instead of URLs when parsing the Markdown. See [Markdown static assets](#).

You write a link like this: `[Download this document](/files/note.docx)`

Docusaurus changes that to: `<a href={require('static/files/note.docx')}>Download this document</a>`

### ⚠️ USE MARKDOWN SYNTAX

Docusaurus will only parse links that are in Markdown syntax. If your asset references are using the JSX tag `<a>` / `<img>`, nothing will be done.

## In CSS

In CSS, the `url()` function is commonly used to reference assets like fonts and images. To reference a static asset, use absolute paths:

```
@font-face {
 font-family: 'Caroline';
 src: url('/font/Caroline.otf');
}
```

The `static/font/Caroline.otf` asset will be loaded by the bundler.

### ⚠️ IMPORTANT TAKEAWAY

One important takeaway: **never hardcode your base URL!** The base URL is considered an implementation detail and should be easily changeable. All paths, even when they look like URL slugs,



are actually file paths.

If you find the URL slug mental model more understandable, here's a rule of thumb:

- Pretend you have a base URL like `/test/` when writing JSX so you don't use an absolute URL path like `src="/img/thumbnail.png"` but instead `require` the asset.
- Pretend it's `/` when writing Markdown or CSS so you always use absolute paths without the base URL.

## Caveats

Keep in mind that:

- By default, none of the files in the `static` folder will be post-processed, hashed, or minified.
  - However, as we've demonstrated above, we are usually able to convert them to `require` calls for you so they do get processed. This is good for aggressive caching and better user experience.
- Missing files referenced via hard-coded absolute paths will not be detected at compilation time and will result in a 404 error.
- By default, GitHub Pages runs published files through [Jekyll](#). Since Jekyll will discard any files that begin with `_`, it is recommended that you disable Jekyll by adding an empty file named `.nojekyll` file to your `static` directory if you are using GitHub pages for hosting.

### 1.3.26 Search

There are a few options you can use to add search to your website:

- [Algolia DocSearch](#) (**official**)
- [Typesense DocSearch](#)
- [Local Search](#)
- Your own `SearchBar` component

#### INFO

Docusaurus provides **first-class support** for [Algolia DocSearch](#).

Other options are **maintained by the community**: please report bugs to their respective repositories.



## Using Algolia DocSearch

Docusaurus has **official support** for [Algolia DocSearch](#).

The service is **free** for any developer documentation or technical blog: just make sure to read the [checklist](#) and [apply to the DocSearch program](#).

DocSearch crawls your website once a week (the schedule is configurable from the web interface) and aggregates all the content in an Algolia index. This content is then queried directly from your front-end using



the Algolia API.

If your website is [not eligible](#) for the free, hosted version of DocSearch, or if your website sits behind a firewall and is not public, then you can [run your own](#) DocSearch crawler.

#### NOTE

By default, the Docusaurus preset generates a [sitemap.xml](#) that the Algolia crawler can use.

#### FROM THE OLD DOCSEARCH?

You can read more about migration from the legacy DocSearch infra in [our blog post](#) or [the DocSearch migration docs](#).

## Index Configuration

After your application has been approved and deployed, you will receive an email with all the details for you to add DocSearch to your project. Editing and managing your crawls can be done via [the web interface](#). Indices are readily available after deployment, so manual configuration usually isn't necessary.

#### USE THE RECOMMENDED CRAWLER CONFIG

It is highly recommended to use our official [Docusaurus v3 crawler configuration](#). We cannot support you if you choose a different crawler configuration.

#### WHEN UPDATING YOUR CRAWLER CONFIG

The crawler configuration contains a `initialIndexSettings`, which will only be used to initialize your Algolia index if it does not exist yet.

If you update your `initialIndexSettings` crawler setting, it is possible to update the index manually through the interface, but [the Algolia team recommends to delete your index and then restart a crawl](#) to fully reinitialize it with the new settings.

## Connecting Algolia

Docusaurus' own `@docusaurus/preset-classic` supports Algolia DocSearch integration. If you use the classic preset, no additional installation is needed.

### Installation steps when not using `@docusaurus/preset-classic`

1. Install the package:



npm

Yarn

pnpm

Bun

```
npm install --save @docusaurus/theme-search-algolia
```

2. Register the theme in `docusaurus.config.js`:

`docusaurus.config.js`

```
export default {
 title: 'My site',
 // ...
 themes: ['@docusaurus/theme-search-algolia'],
 themeConfig: {
 // ...
 },
};
```

Then, add an `algolia` field in your `themeConfig`. [Apply for DocSearch](#) to get your Algolia index and API key.

`docusaurus.config.js`





```
export default {
 // ...
 themeConfig: {
 // ...
 algolia: {
 // The application ID provided by Algolia
 appId: 'YOUR_APP_ID',

 // Public API key: it is safe to commit it
 apiKey: 'YOUR_SEARCH_API_KEY',

 indexName: 'YOUR_INDEX_NAME',

 // Optional: see doc section below
 contextualSearch: true,

 // Optional: Specify domains where the navigation should occur
 // through window.location instead on history.push. Useful when our Algolia
 // config crawls multiple documentation sites and we want to navigate with
 // window.location.href to them.
 externalUrlRegex: 'external\\.com|domain\\.com',

 // Optional: Replace parts of the item URLs from Algolia. Useful when
 // using the same search index for multiple deployments using a different
 // baseUrl. You can use regexp or string in the `from` param. For example:
 // localhost:3000 vs myCompany.com/docs
 replaceSearchResultPathname: {
 from: '/docs/', // or as RegExp: /\sdocs\s/
 to: '/',
 },

 // Optional: Algolia search parameters
 searchParameters: {},

 // Optional: path for search page that enabled by default (`false` to
 // disable it)
 searchPagePath: 'search',

 // Optional: whether the insights feature is enabled or not on
 // Docsearch (`false` by default)
 insights: false,

 // Optional: whether you want to use the new Ask AI feature
 // (undefined by default)
 askAi: 'YOUR_ALGOLIA_ASK_AI_ASSISTANT_ID',

 //... other Algolia params
 },
 },
},
```



```
};
```

### ! INFO

The `searchParameters` option used to be named `algoliaOptions` in Docusaurus v1.

Refer to its [official DocSearch documentation](#) for possible values.

### ! WARNING

The search feature will not work reliably until Algolia crawls your site.

If search doesn't work after any significant change, please use the Algolia dashboard to **trigger a new crawl**.

## Contextual search

Contextual search is **enabled by default**.

It ensures that search results are **relevant to the current language and version**.

docusaurus.config.js

```
export default {
 // ...
 themeConfig: {
 // ...
 algolia: {
 contextualSearch: true,
 },
 },
};
```

Let's consider you have 2 docs versions (**v1** and **v2**) and 2 languages (**en** and **fr**).

When browsing v2 docs, it would be odd to return search results for the v1 documentation. Sometimes v1 and v2 docs are quite similar, and you would end up with duplicate search results for the same query (one result per version).

Similarly, when browsing the French site, it would be odd to return search results for the English docs.

To solve this problem, the contextual search feature understands that you are browsing a specific docs version and language, and will create the search query filters dynamically.



- on `/en/docs/v1/myDoc`, search results will only include **English** results for the **v1** docs (+ other unversioned pages)
- on `/fr/docs/v2/myDoc`, search results will only include **French** results for the **v2** docs (+ other unversioned pages)

### ! INFO

When using `contextualSearch: true` (default), the contextual facet filters will be merged with the ones provided with `algolia.searchParameters.facetFilters`.

For specific needs, you can disable `contextualSearch` and define your own `facetFilters`:

docusaurus.config.js

```
export default {
 // ...
 themeConfig: {
 // ...
 algolia: {
 contextualSearch: false,
 searchParameters: {
 facetFilters: ['language:en', ['filter1', 'filter2'],
 'filter3'],
 },
 },
 },
};
```

Refer to the relevant [Algolia faceting documentation](#).

### ! CONTEXTUAL SEARCH DOESN'T WORK?

If you only get search results when Contextual Search is disabled, this is very likely because of an [index configuration issue](#).

## Ask AI

Ask AI is a new feature that allows you to ask questions about your documentation.

Ask AI brings the power of AI to your documentation:

- **Conversational search:** Users can ask questions in natural language and get context-aware answers.
- **Completely free:** Ask AI is available at no additional cost. You'll pay only for the LLM usage to your provider.



- **BYOLLM (Bring Your Own LLM):** You can use your own language model provider, giving you full control over the AI experience.
- **Direct Algolia index integration:** The provider/models retrieve relevant context directly from your Algolia index, ensuring accurate and up-to-date answers.
- **Recently asked & conversation history:** Easily revisit your recent questions and jump back into previous Ask AI conversations.
- **Seamless integration:** Ask AI is available directly from the search modal.

To enable it, add an `askAi` field in your `algolia` config:

docusaurus.config.js

```
export default {
 // ...
 themeConfig: {
 // ...
 algolia: {
 askAi: 'YOUR_ALGOLIA_ASK_AI_ASSISTANT_ID',

 // OR with custom parameters
 askAi: {
 assistantId: 'YOUR_ALGOLIA_ASK_AI_ASSISTANT_ID',
 indexName: 'YOUR_ALGOLIA_INDEX_NAME',
 apiKey: 'YOUR_ALGOLIA_API_KEY',
 appId: 'YOUR_ALGOLIA_APP_ID',
 suggestedQuestions: true, // Optional: enable suggested questions
 (default: false)
 },

 //... other Algolia params
 },
 },
};
```

### ! INFO

To use Ask AI, you need to have an Algolia index with the Ask AI assistant enabled. Learn more about [how to setup Ask AI](#).

## Styling your Algolia search

By default, DocSearch comes with a fine-tuned theme that was designed for accessibility, making sure that colors and contrasts respect standards.

Still, you can reuse the [Infima CSS variables](#) from Docusaurus to style DocSearch by editing the `/src/css/custom.css` file.



```
/src/css/custom.css
```

```
[data-theme='light'] .DocSearch {
 /* --docsearch-primary-color: var(--ifm-color-primary); */
 /* --docsearch-text-color: var(--ifm-font-color-base); */
 --docsearch-muted-color: var(--ifm-color-secondary-darkest);
 --docsearch-container-background: rgba(94, 100, 112, 0.7);
 /* Modal */
 --docsearch-modal-background: var(--ifm-color-secondary-lighter);
 /* Search box */
 --docsearch-searchbox-background: var(--ifm-color-secondary);
 --docsearch-searchbox-focus-background: var(--ifm-color-white);
 /* Hit */
 --docsearch-hit-color: var(--ifm-font-color-base);
 --docsearch-hit-active-color: var(--ifm-color-white);
 --docsearch-hit-background: var(--ifm-color-white);
 /* Footer */
 --docsearch-footer-background: var(--ifm-color-white);
}

[data-theme='dark'] .DocSearch {
 --docsearch-text-color: var(--ifm-font-color-base);
 --docsearch-muted-color: var(--ifm-color-secondary-darkest);
 --docsearch-container-background: rgba(47, 55, 69, 0.7);
 /* Modal */
 --docsearch-modal-background: var(--ifm-background-color);
 /* Search box */
 --docsearch-searchbox-background: var(--ifm-background-color);
 --docsearch-searchbox-focus-background: var(--ifm-color-black);
 /* Hit */
 --docsearch-hit-color: var(--ifm-font-color-base);
 --docsearch-hit-active-color: var(--ifm-color-white);
 --docsearch-hit-background: var(--ifm-color-emphasis-100);
 /* Footer */
 --docsearch-footer-background: var(--ifm-background-surface-color);
 --docsearch-key-gradient: linear-gradient(
 -26.5deg,
 var(--ifm-color-emphasis-200) 0%,
 var(--ifm-color-emphasis-100) 100%
);
}
```

## Customizing the Algolia search behavior

Algolia DocSearch supports a [list of options](#) that you can pass to the `algolia` field in the `docusaurus.config.js` file.



docusaurus.config.js

```
export default {
 themeConfig: {
 // ...
 algolia: {
 apiKey: 'YOUR_API_KEY',
 indexName: 'YOUR_INDEX_NAME',
 // Options...
 },
 },
};
```

## Editing the Algolia search component

If you prefer to edit the Algolia search React component, [swizzle](#) the `SearchBar` component in `@docusaurus/theme-search-algolia`:

[npm](#)   [Yarn](#)   [pnpm](#)   [Bun](#)

```
npm run swizzle @docusaurus/theme-search-algolia SearchBar
```

## Troubleshooting

Here are the most common issues Docusaurus users face when using Algolia DocSearch.

### No Search Results

Seeing no search results is usually related to an **index configuration problem**.

#### How to check if I have a config problem?

Docusaurus uses [Algolia faceting](#) for its [Contextual Search](#) feature, to create dynamic queries such as:



```
[
 "language:en",
 [
 "docusaurus_tag:default",
 "docusaurus_tag:docs-default-3.2.1",
 "docusaurus_tag:docs-community-current",
 "docusaurus_tag:docs-docs-tests-current"
]
]
```

On the Algolia UI, your index should allow to create facet queries on fields `docusaurus_tag`, `language`, `lang`, `version`, `type`, as shown in the screenshot below:

The screenshot shows the Elasticsearch Index Management interface for an index named 'docusaurus-2'. The left sidebar contains navigation menus for 'CONFIGURE', 'OBSERVE', and 'ENHANCE'. The main panel displays the 'Index' details, including the number of records (67,763), events (0), and average record size (974.07B). Below this, there are tabs for 'Browse', 'Configuration', 'Replicas', 'Search API Logs', 'Stats', and 'UI Demos'. The 'Browse' tab is active, showing a search bar with '30K hits matched in 1 ms'. On the left side of the main panel, there is a 'Facets' section with a dropdown menu. Five red arrows point to the following facet categories: 'docusaurus\_tag', 'lang', 'language', 'type', and 'version'. Each category lists several options with their respective counts. On the right side of the main panel, there are two result snippets. The first snippet shows fields like 'objectID', 'content', 'weight', 'docusaurus\_tag', 'lang', 'language', 'type', 'version', and 'hierarchy'. The second snippet shows similar fields but with different values. At the bottom of the page, there is a footer indicating 'Page 220 of 539'.

Application: docusaurus-2 Index: docusaurus-2

### Index

# records 67,763 # events 0 avg. record size 974.07B

Enhance search relevance! Send event data to unlock more value from your application.

New... Manage index

Browse Configuration Replicas Search API Logs Stats UI Demos

Search Search 30K hits matched in 1 ms

+ Add Query Parameter Display Preferences Query alternatives

#### Facets

##### docusaurus\_tag

- ☐ docs-default-current (12,143)
- ☐ docs-default-3.2.0 (12,109)
- ☐ docs-default-3.1.1 (11,936)
- ☐ docs-default-3.0.1 (11,868)
- ☐ docs-default-2.x (10,896)

Show more values (5)

##### lang

- ☐ fr (13,649)
- ☐ pt-BR (13,598)
- ☐ zh-CN (13,555)
- ☐ ko (13,489)
- ☐ en (13,472)

##### language

- ☐ fr (13,649)
- ☐ pt-BR (13,598)
- ☐ zh-CN (13,555)
- ☐ ko (13,489)
- ☐ en (13,472)

##### type

- ☐ content (33,324)
- ☐ lvl2 (11,882)
- ☐ lvl3 (8,636)
- ☐ lvl5 (8,379)
- ☐ lvl1 (2,884)

Show more values (1)

##### version

- ☐ current (12,818)
- ☐ 3.2.0 (12,109)
- ☐ 3.1.1 (11,936)
- ☐ 3.0.1 (11,868)
- ☐ 2.x (10,896)

#### 1

objectID "0-https://docusaurus.CN/examples/markdown"

content null

weight { pageRank: 0, level: 90,

docusaurus\_tag "default"

lang "zh-CN"

language "zh-CN"

type "lvl1"

version ""

hierarchy { lvl0: "Documentation",

Show more attributes (8)

#### 2

objectID "0-https://docusaurus.CN/docs/next/category"

content null

weight { pageRank: 0, level: 90,

docusaurus\_tag "docs-default-current"

lang "zh-CN"

language "zh-CN"

type "lvl1"

version [ "current" ] (1 item)

hierarchy { lvl0: "指南", lvl1: "Docu

This resu  
It is  
rank  
ed.b  
ased  
on o  
hies

Show more attributes (8)





Alternatively, if you disable [Contextual Search](#) with `{contextualSearch: false}` (which we don't particularly recommend), Docusaurus will not use facet queries, and you should start seeing results.



### USE THE RECOMMENDED CONFIGURATION

We [recommend a specific crawler configuration](#) for a good reason. We cannot support you if you choose to use a different configuration.

You can fix index configuration problems by following those steps:

1. Use the [recommend crawler configuration](#)
2. Delete your index through the UI
3. Trigger a new crawl through the UI
4. Check your index is recreated with the appropriate faceting fields: `docusaurus_tag`, `language`, `lang`, `version`, `type`
5. See that you now get search results, even with [Contextual Search](#) enabled

## Support

The Algolia DocSearch team can help you figure out search problems on your site.

You can reach out to Algolia via [their support page](#) or on [Discord](#).

Docusaurus also has an `#algolia` channel on [Discord](#).



## Using Typesense DocSearch

[Typesense](#) DocSearch works similar to Algolia DocSearch, except that your website is indexed into a Typesense search cluster.

Typesense is an [open source](#) instant-search engine that you can either:

- [Self-Host](#) on your own servers or
- Use the Managed [Typesense Cloud](#) service.

Similar to Algolia DocSearch, there are two components:

- [typesense-docsearch-scraper](#) - which scrapes your website and indexes the data in your Typesense cluster.
- [docusaurus-theme-search-typesense](#) - a search bar UI component to add to your website.

Read a step-by-step walk-through of how to [run typesense-docsearch-scraper here](#) and how to [install the Search Bar in your Docusaurus Site here](#).



## Using Local Search

You can use a local search plugin for websites where the search index is small and can be downloaded to your users' browsers when they visit your website.



You'll find a list of community-supported [local search plugins listed here](#).

## Using your own search

To use your own search, swizzle the `SearchBar` component in `@docusaurus/theme-classic`

**npm**

Yarn

pnpm

Bun

```
npm run swizzle @docusaurus/theme-classic SearchBar
```

This will create an `src/theme/SearchBar` file in your project folder. Restart your dev server and edit the component, you will see that Docusaurus uses your own `SearchBar` component now.

**Notes:** You can alternatively [swizzle from Algolia SearchBar](#) and create your own search component from there.

### 1.3.27 Browser support

Docusaurus allows sites to define the list of supported browsers through a [browserslist configuration](#).

## Purpose

Websites need to balance between backward compatibility and bundle size. As old browsers do not support modern APIs or syntax, more code is needed to implement the same functionality.

For example, you may use the [optional chaining syntax](#):

```
const value = obj?.prop?.val;
```

...which unfortunately is only recognized by browser versions released after 2020. To be compatible with earlier browser versions, when building your site for production, our JS loader will transpile your code to a more verbose syntax:



```
var _obj, _obj$prop;

const value =
 (_obj = obj) === null || _obj === void 0
 ? void 0
 : (_obj$prop = _obj.prop) === null || _obj$prop === void 0
 ? void 0
 : _obj$prop.val;
```

However, this penalizes all other users with increased site load time because the 29-character line now becomes 168 characters—a 6-fold increase! (In practice, it will be better because the names used will be shorter.) As a tradeoff, the JS loader only transpiles the syntax to the degree that's supported by all browser versions defined in the browser list.

The browser list by default is provided through the `package.json` file as a root `browserslist` field.

### WARNING

On old browsers, the compiled output will use unsupported (too recent) JS syntax, causing React to fail to initialize and end up with a static website with only HTML/CSS and no JS.

## Default values

Websites initialized with the default classic template has the following in `package.json`:

package.json

```
{
 "name": "docusaurus",
 // ...
 "browserslist": {
 "production": [">0.5%", "not dead", "not op_mini all"],
 "development": [
 "last 1 chrome version",
 "last 1 firefox version",
 "last 1 safari version"
]
 },
 // ...
}
```

Explained in natural language, the browsers supported in production are those:



- With more than 0.5% of market share; *and*
- Has official support or updates in the past 24 months (the opposite of "dead"); *and*
- Is not Opera Mini.

And browsers used in development are:

- The latest version of Chrome *or* Firefox *or* Safari.

You can "evaluate" any config with the `browserslist` CLI to obtain the actual list:

```
npx browserslist --env="production"
```

The output is all browsers supported in production. Below is the output in January 2022:

```
and_chr 96
and_uc 12.12
chrome 96
chrome 95
chrome 94
edge 96
firefox 95
firefox 94
ie 11
ios_saf 15.2
ios_saf 15.0-15.1
ios_saf 14.5-14.8
ios_saf 14.0-14.4
ios_saf 12.2-12.5
opera 82
opera 81
safari 15.1
safari 14.1
safari 13.1
```

## Read more

You may wish to visit the [browserslist documentation](#) for more specifications, especially the accepted [query values](#) and [best practices](#).

## 1.3.28 Search engine optimization (SEO)

Docusaurus supports search engine optimization in a variety of ways.

## Global metadata



Provide global meta attributes for the entire site through the [site configuration](#). The metadata will all be rendered in the HTML `<head>` using the key-value pairs as the prop name and value. The `metadata` attribute is a convenient shortcut to declare `<meta>` tags, but it is also possible to inject arbitrary tags in `<head>` with the `headTags` attribute.

docusaurus.config.js

```
export default {
 themeConfig: {
 // Declare some <meta> tags
 metadata: [
 {name: 'keywords', content: 'cooking, blog'},
 {name: 'twitter:card', content: 'summary_large_image'},
],
 },
 headTags: [
 // Declare a <link> preconnect tag
 {
 tagName: 'link',
 attributes: {
 rel: 'preconnect',
 href: 'https://example.com',
 },
 },
 // Declare some json-ld structured data
 {
 tagName: 'script',
 attributes: {
 type: 'application/ld+json',
 },
 innerHTML: JSON.stringify({
 '@context': 'https://schema.org/',
 '@type': 'Organization',
 name: 'Meta Open Source',
 url: 'https://opensource.fb.com/',
 logo: 'https://opensource.fb.com/img/logos/Meta-Open-Source.svg',
 }),
 },
],
};
```

Docusaurus adds some metadata out-of-the-box. For example, if you have configured `i18n`, you will get a `hreflang` alternate link.

To read more about types of meta tags, visit [the MDN docs](#).

## Single page metadata



Similar to [global metadata](#), Docusaurus also allows for the addition of meta-information to individual pages. Follow [this guide](#) for configuring the `<head>` tag. In short:

my-markdown-page.mdx

## # A cooking guide

```
<head>
 <meta name="keywords" content="cooking, blog" />
 <meta name="twitter:card" content="summary_large_image" />
 <link rel="preconnect" href="https://example.com" />
 <script type="application/ld+json">
 {JSON.stringify({
 '@context': 'https://schema.org/',
 '@type': 'Organization',
 name: 'Meta Open Source',
 url: 'https://opensource.fb.com/',
 logo: 'https://opensource.fb.com/img/logos/Meta-Open-Source.svg',
 })}
 </script>
</head>

Some content...
```

Docusaurus automatically adds `description`, `title`, canonical URL links, and other useful metadata to each Markdown page. They are configurable through [front matter](#):

```

title: Title for search engines; can be different from the actual heading
description: A short description of this page
image: a thumbnail image to be shown in social media cards
keywords: [keywords, describing, the main topics]

```

When creating your React page, adding these fields in `Layout` would also improve SEO.

### TIP

Prefer to use [front matter](#) for fields like `description` and `keywords`: Docusaurus will automatically apply this to both `description` and `og:description`, while you would have to manually declare two metadata tags when using the `<head>` tag.



### ! INFO

The official plugins all support the following **front matter**: `title`, `description`, `keywords` and `image`. Refer to their respective API documentation for additional **front matter** support:

- [Docs front matter](#)
- [Blog front matter](#)
- [Pages front matter](#)

For JSX pages, you can use the Docusaurus `<Head>` component.

my-react-page.jsx

```
import React from 'react';
import Layout from '@theme/Layout';
import Head from '@docusaurus/Head';

export default function page() {
 return (
 <Layout title="Page" description="A React page demo">
 <Head>
 <meta property="og:image" content="image.png" />
 <meta name="twitter:card" content="summary_large_image" />
 <link rel="preconnect" href="https://example.com" />
 <script type="application/ld+json">
 {JSON.stringify({
 '@context': 'https://schema.org/',
 '@type': 'Organization',
 name: 'Meta Open Source',
 url: 'https://opensource.fb.com/',
 logo: 'https://opensource.fb.com/img/logos/Meta-Open-Source.svg',
 })}
 </script>
 </Head>
 {/* ... */}
 </Layout>
);
}
```

### 💡 TIP

For convenience, the default theme `<Layout>` component accept `title` and `description` as props.



## Static HTML generation

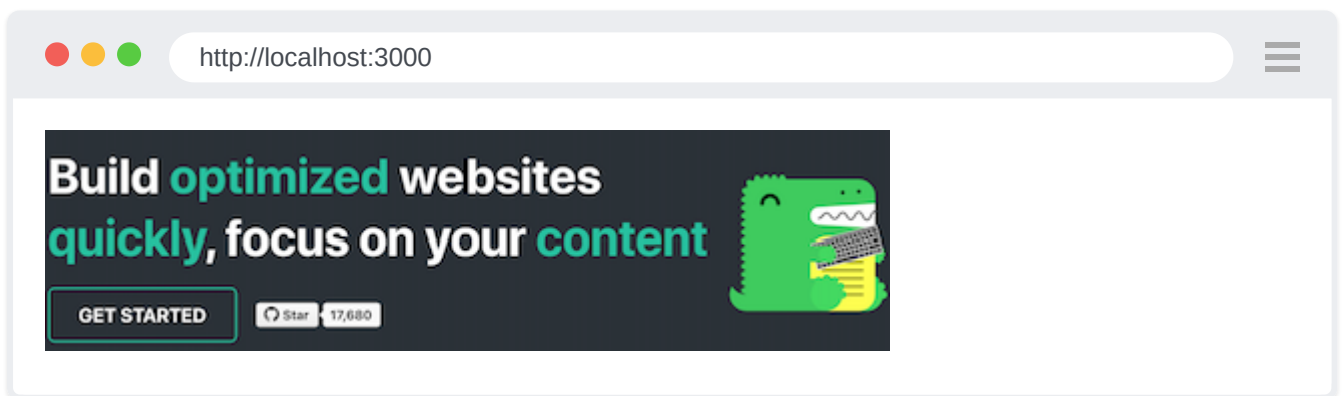
Docusaurus is a static site generator—HTML files are statically generated for every URL route, which helps search engines discover your content more easily.

## Image meta description

The alt tag for an image tells the search engine what the image is about, and is used when the image can't be visually seen, e.g. when using a screen reader, or when the image is broken. Alt tags are commonly supported in Markdown.

You may also add a title for your image—this doesn't impact SEO much but is displayed as a tooltip when hovering above the image, usually used to provide hints.

```
![Docusaurus banner](./assets/docusaurus-asset-example-banner.png 'Image title')
```



## Rich search information

Docusaurus blogs support [rich search results](#) out-of-the-box to get maximum search engine experience. The information is created depending on your meta information in blog/global configuration. In order to get the benefits of the rich search information, fill in the information about the post's publish date, authors, and image, etc. Read more about the meta-information [here](#).

## Robots file

A `robots.txt` file regulates search engines' behavior about which should be displayed and which shouldn't. You can provide it as [static asset](#). The following would allow access to all sub-pages from all requests:

```
static/robots.txt
```





```
User-agent: *
Disallow:
```

Read more about the robots file in [the Google documentation](#).

### ⚠ WARNING

**Important:** the `robots.txt` file does **not** prevent HTML pages from being indexed.

To prevent your whole Docusaurus site from being indexed, use the `noIndex` site config. Some [hosting providers](#) may also let you configure a `X-Robots-Tag: noindex` HTTP header (GitHub Pages does not support this).

To prevent a single page from being indexed, use `<meta name="robots" content="noindex">` as [page metadata](#). Read more about the [robots meta tag](#).

## Sitemap file

Docusaurus provides the `@docusaurus/plugin-sitemap` plugin, which is shipped with `preset-classic` by default. It autogenerates a `sitemap.xml` file which will be available at `https://example.com/[baseUrl]/sitemap.xml` after the production build. This sitemap metadata helps search engine crawlers crawl your site more accurately.

### 💡 TIP

The sitemap plugin automatically filters pages containing a `noindex` [robots meta directive](#).

For example, `/examples/noIndex` is not included in the `Docusaurus sitemap.xml` file because it contains the following [page metadata](#):

```
<head>
 <meta name="robots" content="noindex, nofollow" />
</head>
```

## Human readable links

Docusaurus uses your file names as links, but you can always change that using slugs, see this [tutorial](#) for more details.

## Structured content



Search engines rely on the HTML markup such as `<h2>`, `<table>`, etc., to understand the structure of your webpage. When Docusaurus renders your pages, semantic markup, e.g. `<aside>`, `<nav>`, `<main>`, are used to divide the different sections of the page, helping the search engine to locate parts like sidebar, navbar, and the main page content.

Most [CommonMark](#) syntaxes have their corresponding HTML tags. By using Markdown consistently in your project, you will make it easier for search engines to understand your page content.

## 1.3.29 Using Plugins

**The Docusaurus core doesn't provide any feature of its own.** All features are delegated to individual plugins: the [docs](#) feature provided by the [docs plugin](#); the [blog](#) feature provided by the [blog plugin](#); or individual [pages](#) provided by the [pages plugin](#). If there are no plugins installed, the site won't contain any routes.

You may not need to install common plugins one-by-one though: they can be distributed as a bundle in a [preset](#). For most users, plugins are configured through the preset configuration.

We maintain a [list of official plugins](#), but the community has also created some [unofficial plugins](#). If you want to add any feature: autogenerating doc pages, executing custom scripts, integrating other services... be sure to check out the list: someone may have implemented it for you!

If you are feeling energetic, you can also read [the plugin guide](#) or [plugin method references](#) for how to make a plugin yourself.

## Installing a plugin

A plugin is usually an npm package, so you install them like other npm packages using npm.

[npm](#)   [Yarn](#)   [pnpm](#)   [Bun](#)

```
npm install --save docusaurus-plugin-name
```

Then you add it in your site's `docusaurus.config.js`'s `plugins` option:

`docusaurus.config.js`

```
export default {
 // ...
 plugins: ['@docusaurus/plugin-content-pages'],
};
```

Docusaurus can also load plugins from your local directory, with something like the following:



docusaurus.config.js

```
export default {
 // ...
 plugins: ['./src/plugins/docusaurus-local-plugin'],
};
```

Paths should be absolute or relative to the config file.

## Configuring plugins

For the most basic usage of plugins, you can provide just the plugin name or the path to the plugin.

However, plugins can have options specified by wrapping the name and an options object in a two-member tuple inside your config. This style is usually called "Babel Style".

docusaurus.config.js

```
export default {
 // ...
 plugins: [
 [
 '@docusaurus/plugin-xxx',
 {
 /* options */
 },
],
],
};
```

Example:

docusaurus.config.js



```
export default {
 plugins: [
 // Basic usage.
 '@docusaurus/plugin-debug',

 // With options object (babel style)
 [
 '@docusaurus/plugin-sitemap',
 {
 changefreq: 'weekly',
 },
],
],
};
```

## Multi-instance plugins and plugin IDs

All Docusaurus content plugins can support multiple plugin instances. For example, it may be useful to have [multiple docs plugin instances](#) or [multiple blogs](#). It is required to assign a unique ID to each plugin instance, and by default, the plugin ID is `default`.

docusaurus.config.js

```
export default {
 plugins: [
 [
 '@docusaurus/plugin-content-docs',
 {
 id: 'docs-1',
 // other options
 },
],
 [
 '@docusaurus/plugin-content-docs',
 {
 id: 'docs-2',
 // other options
 },
],
],
};
```



NOTE



At most one plugin instance can be the "default plugin instance", by omitting the `id` attribute, or using `id: 'default'`.

## Using themes

Themes are loaded in the exact same way as plugins. From the consumer perspective, the `themes` and `plugins` entries are interchangeable when installing and configuring a plugin. The only nuance is that themes are loaded after plugins, and it's possible for [a theme to override a plugin's default theme components](#).



### TIP

The `themes` and `plugins` options lead to different [shorthand resolutions](#), so if you want to take advantage of shorthands, be sure to use the right entry!

docusaurus.config.js

```
export default {
 // ...
 themes: ['@docusaurus/theme-classic', '@docusaurus/theme-live-
codeblock'],
};
```

## Using presets

Presets are bundles of plugins and themes. For example, instead of letting you register and configure `@docusaurus/plugin-content-docs`, `@docusaurus/plugin-content-blog`, etc. one after the other in the config file, we have `@docusaurus/preset-classic` preset allows you to configure them in one centralized place.

### `@docusaurus/preset-classic`

The classic preset is shipped by default to new Docusaurus websites created with `create-docusaurus`. It contains the following themes and plugins:

- `@docusaurus/theme-classic`
- `@docusaurus/theme-search-algolia`
- `@docusaurus/plugin-content-docs`
- `@docusaurus/plugin-content-blog`
- `@docusaurus/plugin-content-pages`
- `@docusaurus/plugin-debug`
- `@docusaurus/plugin-google-gtag`
- `@docusaurus/plugin-google-tag-manager`
- `@docusaurus/plugin-sitemap`



- `@docusaurus/plugin-svgr`

The classic preset will relay each option entry to the respective plugin/theme.

docusaurus.config.js

```
export default {
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 // Debug defaults to true in dev, false in prod
 debug: undefined,
 // Will be passed to @docusaurus/theme-classic.
 theme: {
 customCss: ['./src/css/custom.css'],
 },
 // Will be passed to @docusaurus/plugin-content-docs (false to
 // disable)
 docs: {},
 // Will be passed to @docusaurus/plugin-content-blog (false to
 // disable)
 blog: {},
 // Will be passed to @docusaurus/plugin-content-pages (false to
 // disable)
 pages: {},
 // Will be passed to @docusaurus/plugin-sitemap (false to disable)
 sitemap: {},
 // Will be passed to @docusaurus/plugin-svgr (false to disable)
 svgr: {},
 // Will be passed to @docusaurus/plugin-google-gtag (only enabled
 // when explicitly specified)
 gtag: {},
 // Will be passed to @docusaurus/plugin-google-tag-manager (only
 // enabled when explicitly specified)
 googleTagManager: {},
 },
],
],
};
```

## Installing presets

A preset is usually an npm package, so you install them like other npm packages using npm.

npm

Yarn

pnpm

Bun



```
npm install --save @docusaurus/preset-classic
```

Then, add it in your site's `docusaurus.config.js`'s `presets` option:

docusaurus.config.js

```
export default {
 // ...
 presets: ['@docusaurus/preset-classic'],
};
```

Preset paths can be relative to the config file:

docusaurus.config.js

```
export default {
 // ...
 presets: ['./src/presets/docusaurus-local-preset'],
};
```

## Creating presets

A preset is a function with the same shape as the [plugin constructor](#). It should return an object of `{ plugins: PluginConfig[], themes: PluginConfig[] }`, in the same as how they are accepted in the site config. For example, you can specify a preset that includes the following themes and plugins:

src/presets/docusaurus-preset-multi-docs.js



```
export default function preset(context, opts = {}) {
 return {
 themes: [['docusaurus-theme-awesome', opts.theme]],
 plugins: [
 // Using three docs plugins at the same time!
 // Assigning a unique ID for each without asking the user to do it
 ['@docusaurus/plugin-content-docs', {...opts.docs1, id: 'docs1'}],
 ['@docusaurus/plugin-content-docs', {...opts.docs2, id: 'docs2'}],
 ['@docusaurus/plugin-content-docs', {...opts.docs3, id: 'docs3'}],
],
 };
}
```

Then in your Docusaurus config, you may configure the preset:

docusaurus.config.js

```
export default {
 presets: [
 [
 './src/presets/docusaurus-preset-multi-docs.js',
 {
 theme: {hello: 'world'},
 docs1: {path: '/docs'},
 docs2: {path: '/community'},
 docs3: {path: '/api'},
 },
],
],
};
```

This is equivalent to doing:

docusaurus.config.js





```
export default {
 themes: [['docusaurus-theme-awesome', {hello: 'world'}]],
 plugins: [
 ['@docusaurus/plugin-content-docs', {id: 'docs1', path: '/docs'}],
 ['@docusaurus/plugin-content-docs', {id: 'docs2', path: '/community'}],
 ['@docusaurus/plugin-content-docs', {id: 'docs3', path: '/api'}],
],
};
```

This is especially useful when some plugins and themes are intended to be used together. You can even link their options together, e.g. pass one option to multiple plugins.

## Module shorthands

Docusaurus supports shorthands for plugins, themes, and presets. When it sees a plugin/theme/preset name, it tries to load one of the following, in that order:

- `[name]` (like `content-docs`)
- `@docusaurus/[moduleType]-[name]` (like `@docusaurus/plugin-content-docs`)
- `docusaurus-[moduleType]-[name]` (like `docusaurus-plugin-content-docs`)

where `moduleType` is one of `'preset'`, `'theme'`, `'plugin'`, depending on which field the module name is declared in. The first module name that's successfully found is loaded.

If the name is scoped (beginning with `@`), the name is first split into scope and package name by the first slash:

```
@scope
^-----^
 scope (no name!)

@scope/awesome
^-----^ ^-----^
 scope name

@scope/awesome/main
^-----^ ^-----^
 scope name
```

If there is no name (like `@jquery`), `[scope]/docusaurus-[moduleType]` (i.e. `@jquery/docusaurus-plugin`) is loaded. Otherwise, the following are attempted:

- `[scope]/[name]` (like `@jquery/content-docs`)



- `[scope]/docusaurus-[moduleType]-[name]` (like `@jquery/docusaurus-plugin-content-docs`)

Below are some examples, for a plugin registered in the `plugins` field. Note that unlike [ESLint](#) or [Babel](#) where a consistent naming convention for plugins is mandated, Docusaurus permits greater naming freedom, so the resolutions are not certain, but follows the priority defined above.

Declaration	May be resolved as
<code>awesome</code>	<code>docusaurus-plugin-awesome</code>
<code>sitemap</code>	<code>@docusaurus/plugin-sitemap</code>
<code>@my-company</code>	<code>@my-company/docusaurus-plugin</code> (the only possible resolution!)
<code>@my-company/awesome</code>	<code>@my-company/docusaurus-plugin-awesome</code>
<code>@my-company/awesome/web</code>	<code>@my-company/docusaurus-plugin-awesome/web</code>

## 1.3.30 Deployment

To build the static files of your website for production, run:

`npm`   `Yarn`   `pnpm`   `Bun`

```
npm run build
```

Once it finishes, the static files will be generated within the `build` directory.

### NOTE

The only responsibility of Docusaurus is to build your site and emit static files in `build`.

It is now up to you to choose how to host those static files.

You can deploy your site to static site hosting services such as [Vercel](#), [GitHub Pages](#), [Netlify](#), [Render](#), and [Surge](#).

A Docusaurus site is statically rendered, and it can generally work without JavaScript!

## Configuration

The following parameters are required in `docusaurus.config.js` to optimize routing and serve files from the correct location:



Name	Description
<code>url</code>	URL for your site. For a site deployed at <code>https://my-org.com/my-project/</code> , <code>url</code> is <code>https://my-org.com/</code> .
<code>baseUrl</code>	Base URL for your project, with a trailing slash. For a site deployed at <code>https://my-org.com/my-project/</code> , <code>baseUrl</code> is <code>/my-project/</code> .

## Testing your Build Locally

It is important to test your build locally before deploying it for production. Docusaurus provides a `docusaurus serve` command for that:

`npm`

`Yarn`

`pnpm`

`Bun`

```
npm run serve
```

By default, this will load your site at `http://localhost:3000/`.

## Trailing slash configuration

Docusaurus has a `trailingSlash` config to allow customizing URLs/links and emitted filename patterns.

The default value generally works fine. Unfortunately, each static hosting provider has a **different behavior**, and deploying the exact same site to various hosts can lead to distinct results. Depending on your host, it can be useful to change this config.



**TIP**

Use [slorber/trailing-slash-guide](#) to understand better the behavior of your host and configure `trailingSlash` appropriately.

## Using environment variables

Putting potentially sensitive information in the environment is common practice. However, in a typical Docusaurus website, the `docusaurus.config.js` file is the only interface to the Node.js environment (see [our architecture overview](#)), while everything else (MDX pages, React components, etc.) are client side and do not have direct access to the `process` global variable. In this case, you can consider using `customFields` to pass environment variables to the client side.

```
docusaurus.config.js
```



```
// If you are using dotenv (https://www.npmjs.com/package/dotenv)
import 'dotenv/config';

export default {
 title: '...',
 url: process.env.URL, // You can use environment variables to control
 // site specifics as well
 customFields: {
 // Put your custom environment here
 teamEmail: process.env.EMAIL,
 },
};
```

home.jsx

```
import useDocusaurusContext from '@docusaurus/useDocusaurusContext';

export default function Home() {
 const {
 siteConfig: {customFields},
 } = useDocusaurusContext();
 return <div>Contact us through {customFields.teamEmail}</div>;
}
```

## Choosing a hosting provider

There are a few common hosting options:

- [Self-hosting](#) with an HTTP server like Apache2 or Nginx.
- Static hosting providers (e.g. [Netlify](#) and [Vercel](#)). We will use them as references, but the same reasoning can apply to other providers.
- [GitHub Pages](#) (by definition, it is also static hosting provider, but we compare it separately).

If you are unsure of which one to choose, ask the following questions:

### How many resources (money, person-hours, etc.) am I willing to invest in this?

-  Self-hosting requires experience in networking as well as Linux and web server administration. It's the most difficult option, and would require the most time to manage successfully. Expense-wise, cloud services are almost never free, and purchasing/deploying an onsite server can be even more costly.



-  Static hosting providers can help you set up a working website in almost no time and offer features like server-side redirects that are easily configurable. Many providers offer generous build-time quotas even for free plans that you would almost never exceed. However, free plans have limits, and you would need to pay once you hit those limits. Check the pricing page of your provider for details.
-  The GitHub Pages deployment workflow can be tedious to set up. (Evidence: see the length of [Deploying to GitHub Pages!](#)) However, this service (including build and deployment) is always free for public repositories, and we have detailed instructions to help you make it work.

### How much server-side customization do I need?

-  With self-hosting, you have access to the entire server's configuration. You can configure the virtual host to serve different content based on the request URL, you can do complicated server-side redirects, you can implement authentication, and so on. If you need a lot of server-side features, self-host your website.
-  Static hosting providers usually offers some server-side configuration (e.g. URL formatting (trailing slashes), server-side redirects, etc.).
-  GitHub Pages doesn't expose server-side configuration besides enforcing HTTPS and setting CNAME records.

### Do I need collaboration-friendly deployment workflows?

-  Self-hosted services can leverage continuous deployment functionality like Netlify, but more heavy-lifting is involved. Usually, you would designate a specific person to manage the deployment, and the workflow wouldn't be very git-based as opposed to the other two options.
-  Netlify and Vercel have deploy previews for every pull request, which is useful for a team to review work before merging to production. You can also manage a team with different member access to the deployment.
-  GitHub Pages cannot do deploy previews in a non-convoluted way. One repo can only be associated with one site deployment. On the other hand, you can control who has write access to the site's deployment.

There isn't a silver bullet. You need to weigh your needs and resources before making a choice.

## Self-Hosting

Docusaurus can be self-hosted using `docusaurus serve`. Change port using `--port` and `--host` to change host.

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)

```
npm run serve -- --build --port 80 --host 0.0.0.0
```

### WARNING

It is not the best option, compared to a static hosting provider / CDN.

## Hosting provider guides

Here's a list of popular hosting providers and how they should be configured to deploy Docusaurus sites most efficiently. Docusaurus is not affiliated with any of these services, and this information is provided for convenience only.



### GitHub Pages

Deploy your Docusaurus app for production on GitHub Pages.



### Netlify

Deploy your Docusaurus app for production on Netlify.



### Vercel

Deploy your Docusaurus app for production on Vercel.



### OTHER HOSTING SOLUTIONS



We can only officially document a small subset of hosting providers.

However, a Docusaurus site is just static files, and almost any hosting provider able to serve static websites can deploy it.

Check our [Deployment Platforms](#) GitHub Discussions category: the community shares its experience with other platforms there, and hosting providers are welcome to explain how to deploy Docusaurus on their service.

## 1.3.31 Deploying to GitHub Pages

Docusaurus provides an easy way to publish to [GitHub Pages](#), which comes free with every GitHub repository.

### Overview

Usually, there are two repositories (at least two branches) involved in a publishing process: the branch containing the source files, and the branch containing the build output to be served with GitHub Pages. In the following tutorial, they will be referred to as **"source"** and **"deployment"**, respectively.

Each GitHub repository is associated with a GitHub Pages service. If the deployment repository is called `my-org/my-project` (where `my-org` is the organization name or username), the deployed site will appear at `https://my-org.github.io/my-project/`. If the deployment repository is called `my-org/my-org.github.io` (the *organization GitHub Pages repo*), the site will appear at `https://my-org.github.io/`.

#### ! INFO

In case you want to use your custom domain for GitHub Pages, create a `CNAME` file in the `static` directory. Anything within the `static` directory will be copied to the root of the `build` directory for deployment. When using a custom domain, you should be able to move back from `baseUrl: '/projectName/'` to `baseUrl: '/'`, and also set your `url` to your custom domain.

You may refer to GitHub Pages' documentation [User, Organization, and Project Pages](#) for more details.

GitHub Pages picks up deploy-ready files (the output from `docusaurus build`) from the default branch (`master` / `main`, usually) or the `gh-pages` branch, and either from the root or the `/docs` folder. You can configure that through `Settings > Pages` in your repository. This branch will be called the "deployment branch".

We provide a `docusaurus deploy` command that helps you deploy your site from the source branch to the deployment branch in one command: clone, build, and commit.

## `docusaurus.config.js` settings

First, modify your `docusaurus.config.js` and add the following params:



Name	Description
<code>organizationName</code>	The GitHub user or organization that owns the deployment repository.
<code>projectName</code>	The name of the deployment repository.
<code>deploymentBranch</code>	The name of the deployment branch. It defaults to <code>'gh-pages'</code> for non-organization GitHub Pages repos ( <code>projectName</code> not ending in <code>.github.io</code> ). Otherwise, it needs to be explicit as a config field or environment variable.

These fields also have their environment variable counterparts which have a higher priority:

`ORGANIZATION_NAME`, `PROJECT_NAME`, and `DEPLOYMENT_BRANCH`.

### WARNING

GitHub Pages adds a trailing slash to Docusaurus URLs by default. It is recommended to set a `trailingSlash` config (`true` or `false`, not `undefined`).

Example:

docusaurus.config.js

```
export default {
 // ...
 url: 'https://endiliey.github.io', // Your website URL
 baseUrl: '/',
 projectName: 'endiliey.github.io',
 organizationName: 'endiliey',
 trailingSlash: false,
 // ...
};
```

### WARNING

By default, GitHub Pages runs published files through [Jekyll](#). Since Jekyll will discard any files that begin with `_`, it is recommended that you disable Jekyll by adding an empty file named `.nojekyll` file to your `static` directory.

## Environment settings





Name	Description
USE_SSH	Set to <code>true</code> to use SSH instead of the default HTTPS for the connection to the GitHub repo. If the source repo URL is an SSH URL (e.g. <code>git@github.com:facebook/docusaurus.git</code> ), <code>USE_SSH</code> is inferred to be <code>true</code> .
GIT_USER	The username for a GitHub account that <b>has push access to the deployment repo</b> . For your own repositories, this will usually be your GitHub username. Required if not using SSH, and ignored otherwise.
GIT_PASS	Personal access token of the git user (specified by <code>GIT_USER</code> ), to facilitate non-interactive deployment (e.g. continuous deployment)
CURRENT_BRANCH	The source branch. Usually, the branch will be <code>main</code> or <code>master</code> , but it could be any branch except for <code>gh-pages</code> . If nothing is set for this variable, then the current branch from which <code>docusaurus deploy</code> is invoked will be used.
GIT_USER_NAME	The <code>git config user.name</code> value to use when pushing to the deployment repo
GIT_USER_EMAIL	The <code>git config user.email</code> value to use when pushing to the deployment repo

GitHub enterprise installations should work in the same manner as `github.com`; you only need to set the organization's GitHub Enterprise host as an environment variable:

Name	Description
GITHUB_HOST	The domain name of your GitHub enterprise site.
GITHUB_PORT	The port of your GitHub enterprise site.

## Deploy

Finally, to deploy your site to GitHub Pages, run:

**Bash**   **Windows**   **PowerShell**

```
GIT_USER=<GITHUB_USERNAME> yarn deploy
```

### WARNING

Beginning in August 2021, GitHub requires every command-line sign-in to use the **personal access token** instead of the password. When GitHub prompts for your password, enter the PAT instead. See the [GitHub documentation](#) for more information.

Alternatively, you can use SSH (`USE_SSH=true`) to log in.

## Triggering deployment with GitHub Actions



[GitHub Actions](#) allow you to automate, customize, and execute your software development workflows right in your repository.

The workflow examples below assume your website source resides in the `main` branch of your repository (the *source branch* is `main`), and your [publishing source](#) is configured for [publishing with a custom GitHub Actions Workflow](#).

Our goal is that:

1. When a new pull request is made to `main`, there's an action that ensures the site builds successfully, without actually deploying. This job will be called `test-deploy`.
2. When a pull request is merged to the `main` branch or someone pushes to the `main` branch directly, it will be built and deployed to GitHub Pages. This job will be called `deploy`.

Here are two approaches to deploying your docs with GitHub Actions. Based on the location of your deployment repository, choose the relevant tab below:

- Source repo and deployment repo are the **same** repository.
- The deployment repo is a **remote** repository, different from the source. Instructions for this scenario assume [publishing source](#) is the `gh-pages` branch.

## Same Remote

While you can have both jobs defined in the same workflow file, the original `deploy` workflow will always be listed as skipped in the PR check suite status, which is not indicative of the actual status and provides no value to the review process. We therefore propose to manage them as separate workflows instead.

### GitHub action files

Add these two workflow files:

#### ⚠️ TWEAK THE PARAMETERS FOR YOUR SETUP

If your Docusaurus project is not at the root of your repo, you may need to configure a [default working directory](#), and adjust the paths accordingly.

## npm Yarn

```
.github/workflows/deploy.yml
```



```
name: Deploy to GitHub Pages

on:
 push:
 branches:
 - main
 # Review gh actions docs if you want to further define triggers,
 # paths, etc
 # https://docs.github.com/en/actions/using-workflows/workflow-
 # syntax-for-github-actions#on

jobs:
 build:
 name: Build Docusaurus
 runs-on: ubuntu-latest
 steps:
 - uses: actions/checkout@v4
 with:
 fetch-depth: 0
 - uses: actions/setup-node@v4
 with:
 node-version: 24
 cache: npm

 - name: Install dependencies
 run: npm ci
 - name: Build website
 run: npm run build

 - name: Upload Build Artifact
 uses: actions/upload-pages-artifact@v3
 with:
 path: build

 deploy:
 name: Deploy to GitHub Pages
 needs: build

 # Grant GITHUB_TOKEN the permissions required to make a Pages
 # deployment
 permissions:
 pages: write # to deploy to Pages
 id-token: write # to verify the deployment originates from an
 # appropriate source

 # Deploy to the github-pages environment
 environment:
 name: github-pages
```



```
url: ${ steps.deployment.outputs.page_url }}

runs-on: ubuntu-latest
steps:
 - name: Deploy to GitHub Pages
 id: deployment
 uses: actions/deploy-pages@v4
```

.github/workflows/test-deploy.yml

```
name: Test deployment

on:
 pull_request:
 branches:
 - main
 # Review gh actions docs if you want to further define triggers,
 paths, etc
 # https://docs.github.com/en/actions/using-workflows/workflow-
 syntax-for-github-actions#on

jobs:
 test-deploy:
 name: Test deployment
 runs-on: ubuntu-latest
 steps:
 - uses: actions/checkout@v4
 with:
 fetch-depth: 0
 - uses: actions/setup-node@v4
 with:
 node-version: 24
 cache: npm

 - name: Install dependencies
 run: npm ci
 - name: Test build website
 run: npm run build
```

**Site not deployed properly?**



After pushing to main, if you don't see your site published at the desired location (for example, it says "There isn't a GitHub Pages site here", or it's showing your repo's README.md file), try the following:

- Wait about three minutes and refresh. It may take a few minutes for GitHub pages to pick up the new files.
- Check your repo's landing page for a little green tick next to the last commit's title, indicating the CI has passed. If you see a cross, it means the build or deployment failed, and you should check the log for more debugging information.
- Click on the tick and make sure you see a "Deploy to GitHub Pages" workflow. Names like "pages build and deployment / deploy" are GitHub's default workflows, indicating your custom deployment workflow failed to be triggered at all. Make sure the YAML files are placed under the `.github/workflows` folder, and that the trigger condition is set correctly (e.g., if your default branch is "master" instead of "main", you need to change the `on.push` property).
- Under your repo's Settings > Pages, make sure the "Source" (which is the source for the *deployment* files, not "source" as in our terminology) is set to "gh-pages" + "/" (root), since we are using `gh-pages` as the deployment branch.

If you are using a custom domain:

- Verify that you have the correct DNS records set up if you're using a custom domain. See [GitHub pages documentation on configuring custom domains](#). Also, please be aware that it may take up to 24 hours for DNS changes to propagate through the internet.

## 1.3.32 Deploying to Netlify

To deploy your Docusaurus sites to [Netlify](#), first make sure the following options are properly configured:

docusaurus.config.js

```
export default {
 url: 'https://docusaurus-2.netlify.app', // Url to your site with no
 trailing slash
 baseUrl: '/', // Base directory of your site relative to your repo
 // ...
};
```

Then, [create your site with Netlify](#).

While you set up the site, specify the build commands and directories as follows:

- build command: `npm run build`
- publish directory: `build`



If you did not configure these build options, you may still go to "Site settings" -> "Build & deploy" after your site is created.

Once properly configured with the above options, your site should deploy and automatically redeploy upon merging to your deploy branch, which defaults to `main`.

### ⚠ WARNING

Some Docusaurus sites put the `docs` folder outside of `website` (most likely former Docusaurus v1 sites):

```
repo # git root
├── docs # MD files
└── website # Docusaurus root
```

If you decide to use the `website` folder as Netlify's base directory, Netlify will not trigger builds when you update the `docs` folder, and you need to configure a [custom ignore command](#):

website/netlify.toml

```
[build]
 ignore = "git diff --quiet $CACHED_COMMIT_REF $COMMIT_REF . ../docs/"
```

### ⚠ WARNING

By default, Netlify adds trailing slashes to Docusaurus URLs.

It is recommended to disable the Netlify setting `Post Processing > Asset Optimization > Pretty Urls` to prevent lowercase URLs, unnecessary redirects, and 404 errors.

**Be very careful:** the `Disable asset optimization` global checkbox is broken and does not really disable the `Pretty URLs` setting in practice. Please make sure to **uncheck it independently**.

If you want to keep the `Pretty Urls` Netlify setting on, adjust the `trailingSlash` Docusaurus config appropriately.

Refer to [slorber/trailing-slash-guide](#) for more information.

## 1.3.33 Deploying to Vercel

Deploying your Docusaurus project to [Vercel](#) will provide you with [various benefits](#) in the areas of performance and ease of use.



To deploy your Docusaurus project with a [Vercel for Git Integration](#), make sure it has been pushed to a Git repository.

Import the project into Vercel using the [Import Flow](#). During the import, you will find all relevant options preconfigured for you; however, you can choose to change any of these [options](#).

After your project has been imported, all subsequent pushes to branches will generate [Preview Deployments](#), and all changes made to the [Production Branch](#) (usually "main" or "master") will result in a [Production Deployment](#).

## 1.3.34 i18n - Introduction

It is **easy to translate a Docusaurus website** with its internationalization ([i18n](#)) support.

### Goals

It is important to understand the **design decisions** behind the Docusaurus i18n support.

For more context, you can read the initial [RFC](#) and [PR](#).

#### i18n goals

The goals of the Docusaurus i18n system are:

- **Simple:** just put the translated files in the correct filesystem location
- **Flexible translation workflows:** use Git (monorepo, forks, or submodules), SaaS software, FTP
- **Flexible deployment options:** single, multiple domains, or hybrid
- **Modular:** allow plugin authors to provide i18n support
- **Low-overhead runtime:** documentation is mostly static and does not require heavy JS libraries or polyfills
- **Scalable build-times:** allow building and deploying localized sites independently
- **Localize assets:** an image of your site might contain text that should be translated
- **No coupling:** not forced to use any SaaS, yet integrations are possible
- **Easy to use with Crowdin:** a lot of Docusaurus v1 sites use Crowdin and should be able to migrate to v2
- **Good SEO defaults:** we set useful SEO headers like `hreflang` for you
- **RTL support:** locales reading right-to-left (Arabic, Hebrew, etc.) are supported and easy to implement
- **Default translations:** classic theme labels are translated for you in [many languages](#)

#### i18n non-goals

We don't provide support for:

- **Automatic locale detection:** opinionated, and best done on the [server \(your hosting provider\)](#)
- **Translation SaaS software:** you are responsible to understand the external tools of your choice
- **Translation of slugs:** technically complicated, little SEO value

## Translation workflow



## Overview

Overview of the workflow to create a translated Docusaurus website:

1. **Configure:** declare the default locale and alternative locales in `docusaurus.config.js`
2. **Translate:** put the translation files at the correct filesystem location
3. **Deploy:** build and deploy your site using a single or multi-domain strategy

## Translation files

You will work with three kinds of translation files.

## Markdown files

This is the main content of your Docusaurus website.

Markdown and MDX documents are translated as a whole, to fully preserve the translation context, instead of splitting each sentence as a separate string.

## JSON files

JSON is used to translate:

- Your React code: standalone React pages in `src/pages`, or other components
- Layout labels provided through `themeConfig`: navbar, footer
- Layout labels provided through plugin options: docs sidebar category labels, blog sidebar title...

The JSON format used is called **Chrome i18n**:

```
{
 "myTranslationKey1": {
 "message": "Translated message 1",
 "description": "myTranslationKey1 is used on the homepage"
 },
 "myTranslationKey2": {
 "message": "Translated message 2",
 "description": "myTranslationKey2 is used on the FAQ page"
 }
}
```

The choice was made for 2 reasons:

- **Description attribute:** to help translators with additional context
- **Widely supported:** [Chrome extensions](#), [Crowdin](#), [Transifex](#), [Phrase](#), [Applanga](#), etc.

## Data files

Some plugins may read from external data files that are localized as a whole. For example, the blog plugin uses an `authors.yml` file that can be translated by creating a copy under `i18n/[locale]/docusaurus-`





```
plugin-content-blog/authors.yml.
```

## Translation files location

The translation files should be created at the correct filesystem location.

Each locale and plugin has its own `i18n` subfolder:

```
website/i18n/[locale]/[pluginName]/...
```

### NOTE

For multi-instance plugins, the path is `website/i18n/[locale]/[pluginName]-[pluginId]/...`.

Translating a very simple Docusaurus site in French would lead to the following tree:

```
website/i18n
├── fr
│ ├── code.json # Any text label present in the React code
│ │ # Includes text labels from the themes' code
│ ├── docusaurus-plugin-content-blog # translation data the blog plugin
│ │ └── needs
│ │ └── 2020-01-01-hello.md
│ ├── docusaurus-plugin-content-docs # translation data the docs plugin
│ │ └── needs
│ │ ├── current
│ │ │ ├── doc1.md
│ │ │ └── doc2.mdx
│ │ └── current.json
│ └── docusaurus-theme-classic # translation data the classic theme needs
│ ├── footer.json # Text labels in your footer theme config
│ └── navbar.json # Text labels in your navbar theme config
```

The JSON files are initialized with the `docusaurus write-translations` CLI command. Each plugin sources its own translated content under the corresponding folder, while the `code.json` file defines all text labels used in the React code.

Each content plugin or theme is different, and **defines its own translation files location**:

- [Docs i18n](#)
- [Blog i18n](#)



- [Pages i18n](#)
- [Themes i18n](#)

## 1.3.35 Internationalization

### 1.3.36 i18n - Tutorial

This tutorial will walk you through the basics of the **Docusaurus i18n system**.

We will add **French** translations to a **newly initialized English Docusaurus website**.

Initialize a new site (like [this one](#)):

**npm**    Yarn    pnpm    Bun

```
npx create-docusaurus@latest website classic
```

## Configure your site

Modify `docusaurus.config.js` to add the i18n support for the French language.

### Site configuration

Use the [site i18n configuration](#) to declare the i18n locales:

docusaurus.config.js

```
export default {
 i18n: {
 defaultLocale: 'en',
 locales: ['en', 'fr', 'fa'],
 localeConfigs: {
 en: {
 htmlLang: 'en-GB',
 },
 // You can omit a locale (e.g. fr) if you don't need to override the
 defaults
 fa: {
 direction: 'rtl',
 },
 },
 },
};
```



The locale names are used for the translation files' locations, as well as your translated locales' base URL. When building all locales, only the default locale will have its name omitted in the base URL.

Docusaurus uses the locale names to provide **sensible defaults**: the `<html lang="...">` attribute, locale label, calendar format, etc. You can customize these defaults with the `localeConfigs`.

## Theme configuration

Add a **navbar item** of type `localeDropdown` so that users can select the locale they want:

docusaurus.config.js

```
export default {
 themeConfig: {
 navbar: {
 items: [
 {
 type: 'localeDropdown',
 position: 'left',
 },
],
 },
 },
};
```

### TIP

You can pass a query parameter that will be appended to the URL when a user changes the locale using the dropdown (e.g. `queryString: '?persistLocale=true'`).

This is useful for implementing an automatic locale detection on your server. For example, you can use this parameter to store the user's preferred locale in a cookie.

## Start your site

Start your localized site in dev mode, using the locale of your choice:

**npm**   **Yarn**   **pnpm**   **Bun**

```
npm run start -- --locale fr
```

Your site is accessible at <http://localhost:3000/fr/>.



We haven't provided any translation yet, so the site is mostly untranslated.

### TIP

Docusaurus provides **default translations** for generic theme labels, such as "Next" and "Previous" for the pagination.

Please help us complete those [default translations](#).

### WARNING

Each locale is a **distinct standalone single-page application**: it is not possible to start the Docusaurus sites in all locales at the same time.

## Translate your site

All translation data for the French locale is stored in `website/i18n/fr`. Each plugin sources its own translated content under the corresponding folder, while the `code.json` file defines all text labels used in the React code.

### NOTE

After copying files around, restart your site with `npm run start -- --locale fr`. Hot-reload will work better when editing existing files.

## Translate your React code

For any React code you've written yourself: React pages, React components, etc., you will use the [translation APIs](#).

Locate all text labels in your React code that will be visible to your users, and mark them with the translation APIs. There are two kinds of APIs:

- The `<Translate>` component wraps a string as a JSX element;
- The `translate()` callback takes a message and returns a string.

Use the one that better fits the context semantically. For example, the `<Translate>` can be used as React children, while for props that expect a string, the callback can be used.

### WARNING

A JSX element is an *object*, not a string. Using it in contexts expecting strings (such as the children of `<option>`) would [coerce it to a string](#), which returns `"[object Object]"`. While we encourage you to use `<Translate>` as JSX children, only use the element form when it actually works.



## Before After

src/pages/index.js

```
import React from 'react';
import Layout from '@theme/Layout';
import Link from '@docusaurus/Link';

export default function Home() {
 return (
 <Layout>
 <h1>Welcome to my website</h1>
 <main>
 You can also visit my
 <Link to="https://docusaurus.io/blog">blog</Link>

 </main>
 </Layout>
);
}
```

### ! INFO

Docusaurus provides a **very small and lightweight translation runtime** on purpose, and only supports basic [placeholders interpolation](#), using a subset of the [ICU Message Format](#).

Most documentation websites are generally **static** and don't need advanced i18n features (**plurals**, **genders**, etc.). Use a library like [react-intl](#) for more advanced use-cases.

The `docusaurus write-translations` command will statically analyze all React code files used in your site, extract calls to these APIs, and aggregate them in the `code.json` file. The translation files will be stored as maps from IDs to translation message objects (including the translated label and the description of the label). In your calls to the translation APIs (`<Translate>` or `translate()`), you need to specify either the default untranslated message or the ID, in order for Docusaurus to correctly correlate each translation entry to the API call.



### TEXT LABELS MUST BE STATIC



The `docusaurus write-translations` command only does **static analysis** of your code. It doesn't actually run your site. Therefore, dynamic messages can't be extracted, as the message is an *expression*, not a *string*:

```
const items = [
 {id: 1, title: 'Hello'},
 {id: 2, title: 'World'},
];

function ItemsList() {
 return (

 {/* DON'T DO THIS: doesn't work with the write-translations
command */}
 {items.map((item) => (
 <li key={item.id}>
 <Translate>{item.title}</Translate>

))}

);
}
```

This still behaves correctly at runtime. However, in the future, we may provide a "no-runtime" mechanism, allowing the translations to be directly inlined in the React code through Babel transformations, instead of calling the APIs at runtime. Therefore, to be future-proof, you should always prefer statically analyzable messages. For example, we can refactor the code above to:

```
const items = [
 {id: 1, title: <Translate>Hello</Translate>},
 {id: 2, title: <Translate>World</Translate>},
];

function ItemsList() {
 return (

 {/* The titles are now already translated when rendering! */}
 {items.map((item) => (
 <li key={item.id}>{item.title}
))}

);
}
```



You can see the calls to the translation APIs as purely *markers* that tell Docusaurus that "here's a text label to be replaced with a translated message".

## Pluralization

When you run `write-translations`, you will notice that some labels are pluralized:

i18n/en/code.json

```
{
 // ...
 "theme.blog.post.plurals": "One post|{count} posts"
 // ...
}
```

Every language will have a list of [possible plural categories](#). Docusaurus will arrange them in the order of `["zero", "one", "two", "few", "many", "other"]`. For example, because English (`en`) has two plural forms ("one" and "other"), the translation message has two labels separated by a pipe (`|`). For Polish (`pl`) which has three plural forms ("one", "few", and "many"), you would provide three labels in that order, joined by pipes.

You can pluralize your own code's messages as well:

```
import {translate} from '@docusaurus/Translate';
import {usePluralForm} from '@docusaurus/theme-common';

function ItemsList({items}) {
 // `usePluralForm` will provide the plural selector for the current
 locale
 const {selectMessage} = usePluralForm();
 // Select the appropriate pluralized label based on `items.length`
 const message = selectMessage(
 items.length,
 translate(
 {message: 'One item|{count} items'},
 {count: items.length},
),
);
 return (
 <>
 <h2>{message}</h2>
 {items.map((item) => <li key={item.id}>{item.title})}
 </>
);
}
```

**i NOTE**

Docusaurus uses `Intl.PluralRules` to resolve and select plural forms. It is important to provide the right number of plural forms in the right order for `selectMessage` to work.

## Translate plugin data

JSON translation files are used for everything that is interspersed in your code:

- React code, including the translated labels you have marked above
- Navbar and footer labels in theme config
- Docs sidebar category labels in `sidebars.js`
- Blog sidebar title in plugin options
- ...

Run the `write-translations` command:

**npm**

Yarn

pnpm

Bun

```
npm run write-translations -- --locale fr
```

It will extract and initialize the JSON translation files that you need to translate. The `code.json` file at the root includes all translation API calls extracted from the source code, which could either be written by you or provided by the themes, some of which may already be translated by default.

```
i18n/fr/code.json
```





```
{
 // No ID for the <Translate> component: the default message is used as ID
 "Welcome to my website": {
 "message": "Welcome to my website"
 },
 "home.visitMyBlog": {
 "message": "You can also visit my {blog}",
 "description": "The homepage message to ask the user to visit my blog"
 },
 "homepage.visitMyBlog.linkLabel": {
 "message": "Blog",
 "description": "The label for the link to my blog"
 },
 "Home icon": {
 "message": "Home icon",
 "description": "The homepage icon alt message"
 }
}
```

Plugins and themes will also write their own JSON translation files, such as:

i18n/fr/docusaurus-theme-classic/navbar.json

```
{
 "title": {
 "message": "My Site",
 "description": "The title in the navbar"
 },
 "item.label.Docs": {
 "message": "Docs",
 "description": "Navbar item with label Docs"
 },
 "item.label.Blog": {
 "message": "Blog",
 "description": "Navbar item with label Blog"
 },
 "item.label.GitHub": {
 "message": "GitHub",
 "description": "Navbar item with label GitHub"
 }
}
```

Translate the `message` attribute in the JSON files of `i18n/fr`, and your site layout and homepage should now be translated.



## Translate Markdown files

Official Docusaurus content plugins extensively use Markdown/MDX files and allow you to translate them.

### Translate the docs

Copy your docs Markdown files from `docs/` to `i18n/fr/docusaurus-plugin-content-docs/current`, and translate them:

```
mkdir -p i18n/fr/docusaurus-plugin-content-docs/current
cp -r docs/. i18n/fr/docusaurus-plugin-content-docs/current
```

#### ! INFO

Notice that the `docusaurus-plugin-content-docs` plugin always divides its content by versions. The data in `./docs` folder will be translated in the `current` subfolder and `current.json` file. See [the doc versioning guide](#) for more information about what "current" means.

### Translate the blog

Copy your blog Markdown files to `i18n/fr/docusaurus-plugin-content-blog`, and translate them:

```
mkdir -p i18n/fr/docusaurus-plugin-content-blog
cp -r blog/. i18n/fr/docusaurus-plugin-content-blog
```

### Translate the pages

Copy your pages Markdown files to `i18n/fr/docusaurus-plugin-content-pages`, and translate them:

```
mkdir -p i18n/fr/docusaurus-plugin-content-pages
cp -r src/pages/. i18n/fr/docusaurus-plugin-content-pages
```

#### ! WARNING

We only copy `.md` and `.mdx` files, as React pages are translated through JSON translation files already.



#### USE EXPLICIT HEADING IDS

By default, a Markdown heading `### Hello World` will have a generated ID `hello-world`. Other documents can link it with `[link] (#hello-world)`. However, after translation, the heading



becomes `### Bonjour le Monde`, with ID `bonjour-le-monde`.

Generated IDs are not always a good fit for localized sites, as it requires you to localize all the anchor links:

```
- [link](#hello-world).
+ [link](#bonjour-le-monde)
```

For localized sites, it is recommended to use **explicit heading IDs**.

## Deploy your site

You can choose to deploy your site under a **single domain** or use **multiple (sub)domains**.

### ABOUT LOCALIZED BASEURLS

Docusaurus will automatically add a `/<locale>/` path segment to your site for locales except the default one. This heuristic works well for most sites but can be configured on a per-locale basis depending on your deployment requirements.

Read more on the `siteConfig.baseUrl` docs.

## Single-domain deployment

Run the following command:

**npm**    **Yarn**    **pnpm**    **Bun**

```
npm run build
```

Docusaurus will build **one single-page application per locale**:

- `website/build`: for the default, English language
- `website/build/fr`: for the French language

You can now **deploy** the `build` folder to the static hosting solution of your choice.

### NOTE

The Docusaurus website uses this strategy:



- `https://docusaurus.io`
- `https://docusaurus.io/fr`

### TIP

Static hosting providers generally redirect `/unknown/url` to `/404.html` by convention, always showing an **English 404 page**.

**Localize your 404 pages** by configuring your host to redirect `/fr/*` to `/fr/404.html`.

This is not always possible, and depends on your host: GitHub Pages can't do this, [Netlify](#) can.

## Multi-domain deployment

You can also build your site for a single locale:

**npm**    **Yarn**    **pnpm**    **Bun**

```
npm run build -- --locale fr
```

When building a single locale at a time, Docusaurus will not add the `/fr/` URL prefix automatically, assuming you want to deploy each locale to a distinct domain.

On your [static hosting provider](#):

- create one deployment per locale
- configure the appropriate build command, using the `--locale` option
- configure the (sub)domain of your choice for each deployment

### CONFIGURING URLS FOR EACH LOCALE

Use the `siteConfig.i18n.localeConfigs[<locale>].url` attribute to configure a distinct site URL for each locale:



```
const config = {
 i18n: {
 localeConfigs: {
 en: {
 url: 'https://en.docusaurus.io',
 baseUrl: '/',
 },
 fr: {
 url: 'https://fr.docusaurus.io',
 baseUrl: '/',
 },
 },
 },
};
```

This helps [search engines like Google](#) know about localized versions of your page thanks to `<Link rel="alternate" hreflang="<locale>" />` meta tags.

This also permits Docusaurus themes to redirect users to the appropriate URL when they switch locale, usually through the [Navbar locale dropdown](#).

Read more on the [siteConfig.url](#) and [siteConfig.baseUrl](#) docs.

### WARNING

This strategy is **not possible** with GitHub Pages, as it is only possible to **have a single deployment**.

## Hybrid

It is possible to have some locales using sub-paths, and others using subdomains.

It is also possible to deploy each locale as a separate subdomain, assemble the subdomains in a single unified domain at the CDN level:

- Deploy your site as `fr.docusaurus.io`
- Configure a CDN to serve it from `docusaurus.io/fr`

## Managing translations

Docusaurus doesn't care about how you manage your translations: all it needs is that all translation files (JSON, Markdown, or other data files) are available in the file system during building. However, as site creators, you would need to consider how translations are managed so your translation contributors could collaborate well.

We will share two common translation collaboration strategies: [using git](#) and [using Crowdin](#).



## 1.3.37 i18n - Using git

A **possible translation strategy** is to **version control the translation files** with Git (or any other [VCS](#)).

### Tradeoffs

This strategy has advantages:

- **Easy to get started:** just commit the `i18n` folder to Git
- **Easy for developers:** Git, GitHub and pull requests are mainstream developer tools
- **Free** (or without any additional cost, assuming you already use Git)
- **Low friction:** does not require signing up to an external tool
- **Rewarding:** contributors are happy to have a nice contribution history

Using Git also present some shortcomings:

- **Hard for non-developers:** they do not master Git and pull-requests
- **Hard for professional translators:** they are used to SaaS translation software and advanced features
- **Hard to maintain:** you have to keep the translated files **in sync** with the untranslated files

#### NOTE

Some **large-scale technical projects** (React, Vue.js, MDN, TypeScript, Nuxt.js, etc.) use Git for translations.

Refer to the [Docusaurus i18n RFC](#) for our notes and links studying these systems.

## Initialization

This is a walk-through of using Git to translate a newly initialized English Docusaurus website into French, and assume you already followed the [i18n tutorial](#).

### Prepare the Docusaurus site

Initialize a new Docusaurus site:

**npm**   Yarn   pnpm   Bun

```
npx create-docusaurus@latest website classic
```

Add the site configuration for the French language:



docusaurus.config.js

```
export default {
 i18n: {
 defaultLocale: 'en',
 locales: ['en', 'fr'],
 },
 themeConfig: {
 navbar: {
 items: [
 // ...
 {
 type: 'localeDropdown',
 position: 'left',
 },
 // ...
],
 },
 },
 // ...
};
```

Translate the homepage:

src/pages/index.js

```
import React from 'react';
import Translate from '@docusaurus/Translate';
import Layout from '@theme/Layout';

export default function Home() {
 return (
 <Layout>
 <h1 style={{margin: 20}}>
 <Translate description="The homepage main heading">
 Welcome to my Docusaurus translated site!
 </Translate>
 </h1>
 </Layout>
);
}
```

## Initialize the `i18n` folder

Use the [write-translations](#) CLI command to initialize the JSON translation files for the French locale:

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)

```
npm run write-translations -- --locale fr
```

```
1 translations written at i18n/fr/code.json
11 translations written at i18n/fr/docusaurus-theme-classic/footer.json
4 translations written at i18n/fr/docusaurus-theme-classic/navbar.json
3 translations written at i18n/fr/docusaurus-plugin-content-
docs/current.json
```

**TIP**

Use the `--messagePrefix '(fr) '` option to make the untranslated strings stand out.

`Hello` will appear as `(fr) Hello` and makes it clear a translation is missing.

Copy your untranslated Markdown files to the French folder:

```
mkdir -p i18n/fr/docusaurus-plugin-content-docs/current
cp -r docs/. i18n/fr/docusaurus-plugin-content-docs/current

mkdir -p i18n/fr/docusaurus-plugin-content-blog
cp -r blog/. i18n/fr/docusaurus-plugin-content-blog

mkdir -p i18n/fr/docusaurus-plugin-content-pages
cp -r src/pages/. i18n/fr/docusaurus-plugin-content-pages
```

Add all these files to Git.

## Translate the files

Translate the Markdown and JSON files in `i18n/fr` and commit the translation.

You should now be able to start your site in French and see the translations:

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)

```
npm run start -- --locale fr
```





You can also build the site locally or on your CI:

**npm**   **Yarn**   **pnpm**   **Bun**

```
npm run build
or
npm run build -- --locale fr
```

## Repeat

Follow the same process for each locale you need to support.

# Maintenance

Keeping translated files **consistent** with the originals **can be challenging**, in particular for Markdown documents.

## Markdown translations

When an untranslated Markdown document is edited, it is **your responsibility to maintain the respective translated files**, and we unfortunately don't have a good way to help you do so.

To keep your translated sites consistent, when the `website/docs/doc1.md` doc is edited, you need **backport these edits** to `i18n/fr/docusaurus-plugin-content-docs/current/doc1.md`.

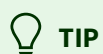
## JSON translations

To help you maintain the JSON translation files, it is possible to run again the [write-translations](#) CLI command:

**npm**   **Yarn**   **pnpm**   **Bun**

```
npm run write-translations -- --locale fr
```

New translations will be appended, and existing ones will not be overridden.



**TIP**

Reset your translations with the `--override` option.

## Localize edit URLs



When the user is browsing a page at `/fr/doc1`, the edit button will link by default to the unlocalized doc at `website/docs/doc1.md`.

Your translations are on Git, and you can use the `editLocalizedFiles: true` option of the docs and blog plugins.

The edit button will link to the localized doc at `i18n/fr/docusaurus-plugin-content-docs/current/doc1.md`.

### 1.3.38 i18n - Using Crowdin

The i18n system of Docusaurus is **decoupled from any translation software**.

You can integrate Docusaurus with the **tools and SaaS of your choice**, as long as you put the **translation files at the correct location**.

We document the usage of [Crowdin](#), as **one possible integration example**.

#### WARNING

This is **not an endorsement of Crowdin** as the unique choice to translate a Docusaurus site, but it is successfully used by Facebook to translate documentation projects such as [Jest](#), [Docusaurus](#), and [ReasonML](#).

Refer to the [Crowdin documentation](#) and [Crowdin support](#) for help.

#### TIP

Use this [community-driven GitHub discussion](#) to discuss anything related to Docusaurus + Crowdin.

## Crowdin overview

Crowdin is a translation SaaS, offering a [free plan for open-source projects](#).

We recommend the following translation workflow:

- **Upload sources** to Crowdin (untranslated files)
- Use Crowdin to **translate the content**
- **Download translations** from Crowdin (localized translation files)

Crowdin provides a [CLI](#) to **upload sources** and **download translations**, allowing you to automate the translation process.

The `crowdin.yml` [configuration file](#) is convenient for Docusaurus, and permits to **download the localized translation files at the expected location** (in `i18n/[locale]/..`).

Read the [official documentation](#) to know more about advanced features and different translation workflows.



# Crowdin tutorial

This is a walk-through of using Crowdin to translate a newly initialized English Docusaurus website into French, and assume you already followed the [i18n tutorial](#).

The end result can be seen at [docusaurus-crowdin-example.netlify.app](https://docusaurus-crowdin-example.netlify.app) ([repository](#)).

## Prepare the Docusaurus site

Initialize a new Docusaurus site:

**npm**   Yarn   pnpm   Bun

```
npx create-docusaurus@latest website classic
```

Add the site configuration for the French language:

docusaurus.config.js

```
export default {
 i18n: {
 defaultLocale: 'en',
 locales: ['en', 'fr'],
 },
 themeConfig: {
 navbar: {
 items: [
 // ...
 {
 type: 'localeDropdown',
 position: 'left',
 },
 // ...
],
 },
 },
 // ...
};
```

Translate the homepage:



src/pages/index.js

```
import React from 'react';
import Translate from '@docusaurus/Translate';
import Layout from '@theme/Layout';

export default function Home() {
 return (
 <Layout>
 <h1 style={{margin: 20}}>
 <Translate description="The homepage main heading">
 Welcome to my Docusaurus translated site!
 </Translate>
 </h1>
 </Layout>
);
}
```

## Create a Crowdin project

Sign up on [Crowdin](#), and create a project.

Use English as the source language, and French as the target language.



# Create Crowdin Project

Project name

docusaurus-crowdin-example

Project address

https://crowdin.com/project/docusaurus-crowdin-example

☒ Public project

Visible to anyone. You can restrict access to specific languages after the project is created.

☐ Private project

Visible only to the invited project members.

Source language

English

Target languages

fre



☒ French

☐ French, Belgium

☐ French, Canada

☐ French, Luxembourg

☐ French, Quebec

☐ French, Switzerland

1 language selected



French

Your project is created, but it is empty for now. We will upload the files to translate in the next steps.

## Create the Crowdin configuration

This configuration ([doc](#)) provides a mapping for the Crowdin CLI to understand:

- Where to find the source files to upload (JSON and Markdown)
- Where to download the files after translation (in `i18n/[locale]`)

Create `crowdin.yml` in `website`:

```
crowdin.yml
```



```
project_id: '123456'
api_token_env: CROWDIN_PERSONAL_TOKEN
preserve_hierarchy: true
files:
 # JSON translation files
 - source: /i18n/en/**/*
 translation: /i18n/%two_letters_code%/**/%original_file_name%
 # Docs Markdown files
 - source: /docs/**/*
 translation: /i18n/%two_letters_code%/docusaurus-plugin-content-docs/current/**/%original_file_name%
 # Blog Markdown files
 - source: /blog/**/*
 translation: /i18n/%two_letters_code%/docusaurus-plugin-content-blog/**/%original_file_name%
```

Crowdin has its own syntax for declaring source/translation paths:

- `**/*`: everything in a subfolder
- `%two_letters_code%`: the 2-letters variant of Crowdin target languages (`fr` in our case)
- `**/%original_file_name%`: the translations will preserve the original folder/file hierarchy

### ! INFO

The Crowdin CLI warnings are not always easy to understand.

We advise to:

- change one thing at a time
- re-upload sources after any configuration change
- use paths starting with `/` (`./` does not work)
- avoid fancy globbing patterns like `/docs/**/*.(md|mdx)` (does not work)

## Access token

The `api_token_env` attribute defines the **env variable name** read by the Crowdin CLI.

You can obtain a `Personal Access Token` on [your personal profile page](#).

### 💡 TIP

You can keep the default value `CROWDIN_PERSONAL_TOKEN`, and set this environment variable and on your computer and on the CI server to the generated access token.



### ⚠ WARNING

A Personal Access Tokens grant **read-write access to all your Crowdin projects**.

You should **not commit** it, and it may be a good idea to create a dedicated **Crowdin profile for your company** instead of using a personal account.

## Other configuration fields

- `project_id`: can be hardcoded, and is found on `https://crowdin.com/project/<MY_PROJECT_NAME>/settings#api`
- `preserve_hierarchy`: preserve the folder's hierarchy of your docs on Crowdin UI instead of flattening everything

## Install the Crowdin CLI

This tutorial uses the CLI version `3.5.2`, but we expect `3.x` releases to keep working.

Install the Crowdin CLI as an npm package to your Docusaurus site:

**npm**   Yarn   pnpm   Bun

```
npm install @crowdin/cli@3
```

Add a `crowdin` script:

package.json

```
{
 "scripts": {
 // ...
 "write-translations": "docusaurus write-translations",
 "crowdin": "crowdin"
 }
}
```

Test that you can run the Crowdin CLI:

**npm**   Yarn   pnpm   Bun



```
npm run crowdin -- --version
```

Set the `CROWDIN_PERSONAL_TOKEN` env variable on your computer, to allow the CLI to authenticate with the Crowdin API.

**TIP**

Temporarily, you can hardcode your personal token in `crowdin.yml` with `api_token: 'MY-TOKEN'`.

## Upload the sources

Generate the JSON translation files for the default language in `website/i18n/en`:

**npm**   Yarn   pnpm   Bun

---

```
npm run write-translations
```

Upload all the JSON and Markdown translation files:

**npm**   Yarn   pnpm   Bun

---

```
npm run crowdin upload
```





```
✓ Fetching project info
✓ Directory 'i18n'
✓ Directory 'i18n/en'
✓ Directory 'i18n/en/docusaurus-theme-classic'
✓ Directory 'i18n/en/docusaurus-plugin-content-docs'
✓ File 'i18n/en/docusaurus-theme-classic/footer.json'
✓ File 'i18n/en/docusaurus-theme-classic/navbar.json'
✓ File 'i18n/en/docusaurus-plugin-content-docs/current.json'
✓ Directory 'docs'
✓ File 'docs/doc3.md'
✓ File 'docs/mdx.md'
✓ File 'docs/doc2.md'
✓ File 'docs/doc1.md'
✓ Directory 'blog'
✓ File 'blog/2019-05-30-welcome.md'
✓ File 'blog/2019-05-29-hello-world.md'
✓ File 'blog/2019-05-28-hola.md'
```

Your source files are now visible on the Crowdin interface:

[https://crowdin.com/project/<MY\\_PROJECT\\_NAME>/settings#files](https://crowdin.com/project/<MY_PROJECT_NAME>/settings#files)

S docusaurus-crowdin-example



Project Home

## docusaurus-crowdin-example settings

General	Translations	Files	Members	Vendors	Reports	Screenshots	TM	Glossary	Integrations	API & Webhooks	Strings	More ▾
📁 Add File ▾		New Folder	New Version Branch	🗑️							Search files	
Name						Strings		Revision				
▶	📁	blog				38				↑	...	
▶	📁	docs				166				↑	...	
▶	📁	i18n				18				↑	...	

## Translate the sources

On [https://crowdin.com/project/<MY\\_PROJECT\\_NAME>](https://crowdin.com/project/<MY_PROJECT_NAME>), click on the French target language.



S / docusaurus-crowdin-example

# French translation

Filter Files		Translate All			
▼	blog				...
	2019-05-28-hola.md		0% • 0%		...
	2019-05-29-hello-world.md		0% • 0%		...
	2019-05-30-welcome.md		0% • 0%		...
▼	docs				...
	doc1.md		0% • 0%		...
	doc2.md		0% • 0%		...
	doc3.md		0% • 0%		...
	mdx.md		0% • 0%		...
▼	i18n				...
▼	en				...
▼	docusaurus-plugin-content-docs				...
	current.json		0% • 0%		...
▼	docusaurus-theme-classic				...
	footer.json		0% • 0%		...
	navbar.json		0% • 0%		...

Translate some Markdown files.

FRENCH > DOC1.MD

Search in file

id: doc1  
title: Style Guide  
sidebar\_label: Style Guide  
slug: /

Tu peux écrire du contenu en utilisant la syntaxe Markdown de GitHub.

**Syntaxe Markdown**

Sert de page d'exemple pour styliser un site Docusaurus basé sur Markdown.

**Headers**

**H1 - Create the best documentation**

**H2 - Create the best documentation**

**H3 - Create the best documentation**

**H4 - Create the best documentation**

**H5 - Create the best documentation**

**H6 - Create the best documentation**

**Emphasis**

Emphasis, aka Italics, with *asterisks* or underscores.

Strong emphasis, aka bold, with **asterisks** or underscores.

SOURCE STRING

To serve as an example page when styling markdown based Docusaurus sites.

CONTEXT EDIT

Paragraph text  
XPath: /p[2]

Sert de page d'exemple pour styliser un site Docusaurus basé sur Markdown.

73 / 74

SAVE

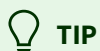
FRENCH TRANSLATIONS

No translations suggested yet

TM AND MT SUGGESTIONS

No suggestions

OTHER LANGUAGES

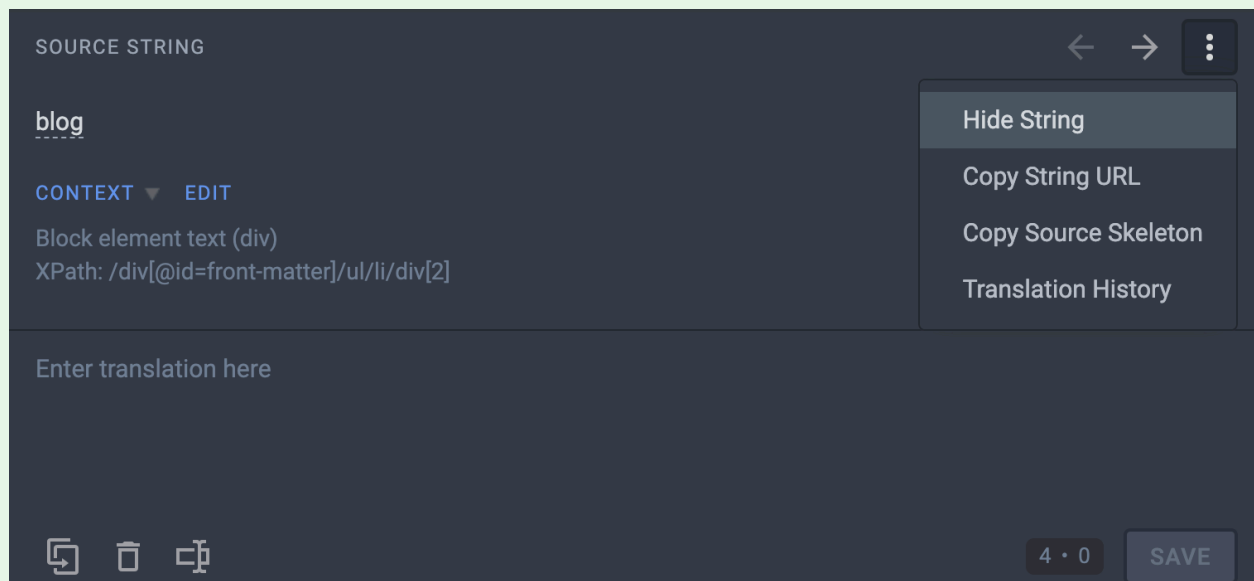


TIP

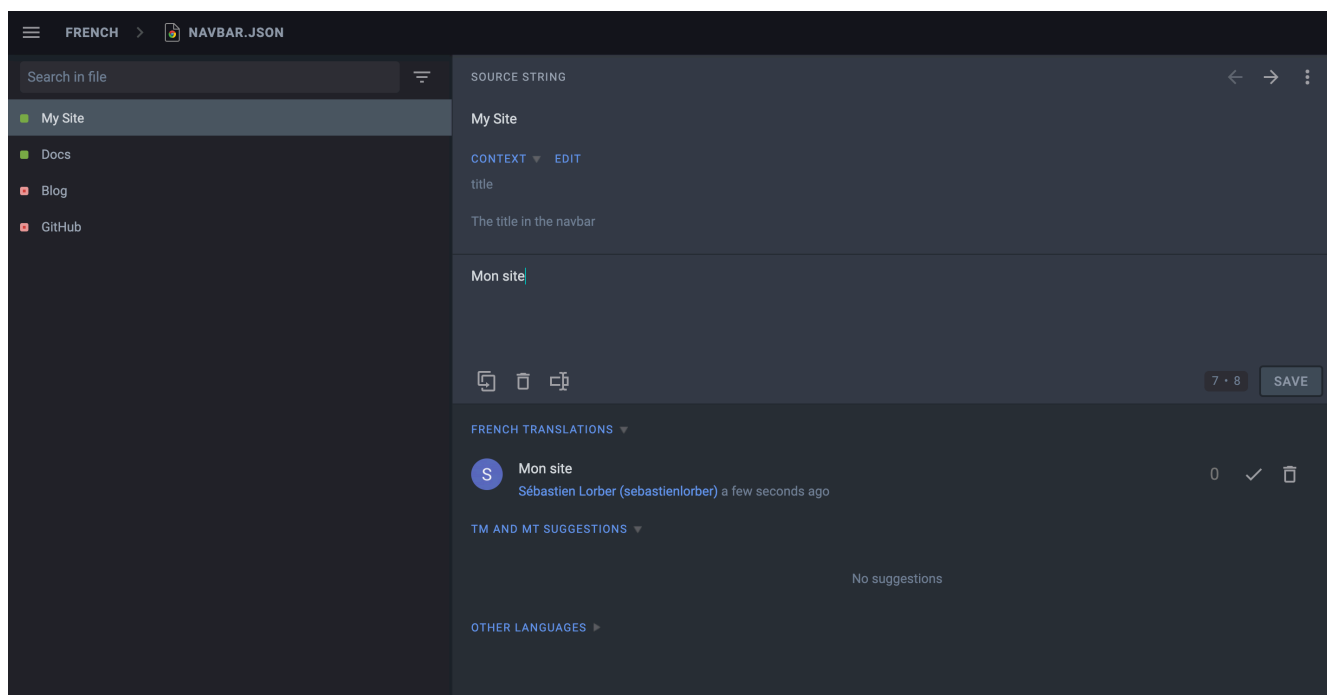


Use `Hide String` to make sure translators **don't translate things that should not be**:

- Front matter: `id`, `slug`, `tags` ...
- Admonitions: `:::`, `:::note`, `:::tip` ...



Translate some JSON files.



### ! INFO

The `description` attribute of JSON translation files is visible on Crowdin to help translate the strings.

**TIP**

**Pre-translate** your site, and **fix pre-translation mistakes manually** (enable the Global Translation Memory in settings first).

Use the `Hide String` feature first, as Crowdin is pre-translating things too optimistically.

## Download the translations

Use the Crowdin CLI to download the translated JSON and Markdown files.

**npm**

Yarn

pnpm

Bun

```
npm run crowdin download
```

The translated content should be downloaded in `i18n/fr`.

Start your site on the French locale:

**npm**

Yarn

pnpm

Bun

```
npm run start -- --locale fr
```

Make sure that your website is now translated in French at `http://localhost:3000/fr/`.

## Automate with CI

We will configure the CI to **download the Crowdin translations at build time** and keep them outside of Git.

Add `website/i18n` to `.gitignore`.

Set the `CROWDIN_PERSONAL_TOKEN` env variable on your CI.

Create an npm script to `sync` Crowdin (extract sources, upload sources, download translations):

```
package.json
```



```
{
 "scripts": {
 "crowdin:sync": "docusaurus write-translations && crowdin upload && crowdin download"
 }
}
```

Call the `npm run crowdin:sync` script in your CI, just before building the Docusaurus site.

**TIP**

Keep your deploy-previews fast: don't download translations, and use `npm run build -- --locale en` for feature branches.

**WARNING**

Crowdin does not support well multiple concurrent uploads/downloads: it is preferable to only include translations to your production deployment, and keep deploy previews untranslated.

## Advanced Crowdin topics

### MDX

**WARNING**

Pay special attention to the JSX fragments in MDX documents!

Crowdin **does not support officially MDX**, but they added **support for the `.mdx` extension**, and interpret such files as Markdown (instead of plain text).

### MDX problems

Crowdin thinks that the JSX syntax is embedded HTML and can mess up with the JSX markup when you download the translations, leading to a site that fails to build due to invalid JSX.

Simple JSX fragments using simple string props like `<Username name="Sebastien"/>` will work fine; more complex JSX fragments using object/array props like `<User person={{name: "Sebastien"}}/>` are more likely to fail due to a syntax that does not look like HTML.

### MDX solutions

We recommend extracting the complex embedded JSX code as separate standalone components. We also added an `mdx-code-block` escape hatch syntax:



# How to deploy Docusaurus

To deploy Docusaurus, run the following command:

```
````mdx-code-block

import Tabs from '@theme/Tabs';

import TabItem from '@theme/TabItem';

<Tabs>
  <TabItem value="bash" label="Bash">

    ```bash
 GIT_USER=<GITHUB_USERNAME> yarn deploy
    ```

  </TabItem>
  <TabItem value="windows" label="Windows">

    ```batch
 cmd /C "set "GIT_USER=<GITHUB_USERNAME>" && yarn deploy"
    ```

  </TabItem>
</Tabs>
````
```

This will:

- be interpreted by Crowdin as code blocks (and not mess-up with the markup on download)
- be interpreted by Docusaurus as regular JSX (as if it was not wrapped by any code block)
- unfortunately opt-out of MDX tooling (IDE syntax highlighting, Prettier...)

## Docs versioning

Configure translation files for the `website/versioned_docs` folder.

When creating a new version, the source strings will generally be quite similar to the current version (`website/docs`), and you don't want to translate the new version docs again and again.

Crowdin provides a `Duplicate Strings` setting.



## Translations

### Duplicate Strings

Hide – all duplicates will share the same translation



You can save time by translating all duplicates with the same translation and hiding these instances from translators. This may, however, affect accuracy.

We recommend using `Hide`, but the ideal setting depends on how much your versions are different.

### ! WARNING

Not using `Hide` leads to a much larger amount of `source strings` in quotas, and will affect the pricing.

## Multi-instance plugins

You need to configure translation files for each plugin instance.

If you have a docs plugin instance with `id=ios`, you will need to configure those source files as well

- `website/ios`
- `website/ios_versioned_docs` (if versioned)

## Maintaining your site

Sometimes, you will **remove or rename a source file** on Git, and Crowdin will display CLI warnings:

```
2:53:51 PM: ✓ Building ZIP archive with the latest translations
2:53:51 PM: [●●●●●] Fetching project info
2:53:51 PM: [●●●●●] Fetching project info
2:53:51 PM: ✓ Building translation (100%)
2:53:51 PM: [●●●●●] Downloading translation
2:53:51 PM: ✓ Downloading translation
2:53:52 PM: ✓ Extracted: 'i18n/fr/code.json'
2:53:52 PM: ✓ Extracted: 'i18n/fr/docusaurus-plugin-content-blog/2019-05-28-hola.md'
2:53:52 PM: ✓ Extracted: 'i18n/fr/docusaurus-plugin-content-blog/2019-05-29-hello-world.md'
2:53:52 PM: ✓ Extracted: 'i18n/fr/docusaurus-plugin-content-blog/2019-05-30-welcome.md'
2:53:52 PM: ✓ Extracted: 'i18n/fr/docusaurus-plugin-content-docs/current.json'
2:53:52 PM: ✓ Extracted: 'i18n/fr/docusaurus-plugin-content-docs/current/doc1.md'
2:53:52 PM: ✓ Extracted: 'i18n/fr/docusaurus-plugin-content-docs/current/doc2.md'
2:53:52 PM: ✓ Extracted: 'i18n/fr/docusaurus-plugin-content-docs/current/doc3.md'
2:53:52 PM: ✓ Extracted: 'i18n/fr/docusaurus-plugin-content-docs/current/interactiveDoc.mdx'
2:53:52 PM: ✓ Extracted: 'i18n/fr/docusaurus-theme-classic/footer.json'
2:53:52 PM: ✓ Extracted: 'i18n/fr/docusaurus-theme-classic/navbar.json'
2:53:52 PM: ⚠ Downloaded translations don't match the current project configuration. The translations for the following sources
will be omitted (use --verbose to get the list of the omitted translations):
- docs/doc4.md (1)
2:53:52 PM: Done in 20.84s.
```

When your sources are refactored, you should use the Crowdin UI to **update your Crowdin files manually**:



## docusaurus-crowdin-example settings

| General Translations Files Members Vendors Reports Screenshots TM Glossary Integrations API & Webhooks Strings More ▾ |         |   |          |   |       |  |  |  |  |
|-----------------------------------------------------------------------------------------------------------------------|---------|---|----------|---|-------|--|--|--|--|
| Add File ▾ New Folder New Version Branch                                                                              |         |   |          |   |       |  |  |  |  |
| Search files                                                                                                          |         |   |          |   |       |  |  |  |  |
| Name                                                                                                                  | Strings |   | Revision |   |       |  |  |  |  |
| ► blog                                                                                                                | 18      |   |          |   | ↑ ... |  |  |  |  |
| ▼ docs                                                                                                                |         |   |          |   | ↑ ... |  |  |  |  |
| doc1.md                                                                                                               | 100     | 1 | Update   | ↑ | ...   |  |  |  |  |
| doc2.md                                                                                                               | 2       | 1 | Update   | ↑ | ...   |  |  |  |  |
| doc3.md                                                                                                               | 42      | 1 | Update   | ↑ | ...   |  |  |  |  |
| interactiveDoc.mdx                                                                                                    | 4       | 2 | Update   | ↑ | ...   |  |  |  |  |
| ▼ i18n                                                                                                                |         |   |          |   | ↑ ... |  |  |  |  |
| ▼ en                                                                                                                  |         |   |          |   | ↑ ... |  |  |  |  |
| ► docusaurus-plug                                                                                                     | 2       |   |          | ↑ | ...   |  |  |  |  |
| ► docusaurus-ther                                                                                                     | 12      |   |          | ↑ | ...   |  |  |  |  |
| code.json                                                                                                             | 1       | 1 | Update   | ↑ | ...   |  |  |  |  |

## VCS (Git) integrations

Crowdin has multiple VCS integrations for [GitHub](#), [GitLab](#), [Bitbucket](#).



### TL;DR

We recommend avoiding them.

It could have been helpful to be able to edit the translations in both Git and Crowdin, and have a **bi-directional sync** between the 2 systems.

In practice, **it didn't work very reliably** for a few reasons:

- The Crowdin -> Git sync works fine (with a pull request)
- The Git -> Crowdin sync is manual (you have to press a button)
- The heuristics used by Crowdin to match existing Markdown translations to existing Markdown sources are not 100% reliable, and you have to verify the result on Crowdin UI after any sync from Git
- 2 users concurrently editing on Git and Crowdin can lead to a translation loss
- It requires the `crowdin.yml` file to be at the root of the repository

## In-Context localization

Crowdin has an [In-Context localization](#) feature.



### WARNING

Unfortunately, it does not work yet for technical reasons, but we have good hope it can be solved.

Crowdin replaces Markdown strings with technical IDs such as `crowdin:id12345`, but it does so too aggressively, including hidden strings, and messes up with front matter, admonitions, JSX...





## Localize edit URLs

When the user is browsing a page at `/fr/doc1`, the edit button will link by default to the unlocalized doc at `website/docs/doc1.md`.

You may prefer the edit button to link to the Crowdin interface instead by using the `editUrl` function to customize the edit URLs on a per-locale basis.

docusaurus.config.js

```
const DefaultLocale = 'en';

export default {
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 docs: {
 editUrl: ({locale, versionDocsDirPath, docPath}) => {
 // Link to Crowdin for French docs
 if (locale !== DefaultLocale) {
 return `https://crowdin.com/project/docusaurus-v2/${locale}`;
 }
 // Link to GitHub for English docs
 return
 `https://github.com/facebook/docusaurus/edit/main/website/${versionDocsDirPath}/${docPath}`;
 },
 blog: {
 editUrl: ({locale, blogDirPath, blogPath}) => {
 if (locale !== DefaultLocale) {
 return `https://crowdin.com/project/docusaurus-v2/${locale}`;
 }
 return
 `https://github.com/facebook/docusaurus/edit/main/website/${blogDirPath}/${blogPath}`;
 },
 },
],
],
],
],
};
```



NOTE



It is currently **not possible to link to a specific file** in Crowdin.

## Example configuration

The **Docusaurus configuration file** is a good example of using versioning and multi-instance:

```
crowdin.yml
```



```
project_id: '428890'
api_token_env: CROWDIN_PERSONAL_TOKEN
preserve_hierarchy: true
languages_mapping: &languages_mapping
 two_letters_code:
 pt-BR: pt-BR
mdx_file_type: &mdx_file_type mdx_v2_4
files:
 - source: /website/i18n/en/**/*
 translation: /website/i18n/%two_letters_code%/**/%original_file_name%
 languages_mapping: *languages_mapping
 - source: /website/docs/**/*.*mdx
 translation: /website/i18n/%two_letters_code%/docusaurus-plugin-
content-docs/current/**/%original_file_name%
 languages_mapping: *languages_mapping
 type: *mdx_file_type
 - source: /website/docs/**/*
 translation: /website/i18n/%two_letters_code%/docusaurus-plugin-
content-docs/current/**/%original_file_name%
 languages_mapping: *languages_mapping
 ignore: [/**/*.*mdx]
 - source: /website/versioned_docs/**/*.*mdx
 translation: /website/i18n/%two_letters_code%/docusaurus-plugin-
content-docs/**/%original_file_name%
 languages_mapping: *languages_mapping
 type: *mdx_file_type
 - source: /website/versioned_docs/**/*
 translation: /website/i18n/%two_letters_code%/docusaurus-plugin-
content-docs/**/%original_file_name%
 languages_mapping: *languages_mapping
 ignore: [/**/*.*mdx]
 - source: /website/community/**/*
 translation: /website/i18n/%two_letters_code%/docusaurus-plugin-
content-docs-community/current/**/%original_file_name%
 languages_mapping: *languages_mapping
 - source: /website/blog/**/*
 translation: /website/i18n/%two_letters_code%/docusaurus-plugin-
content-blog/**/%original_file_name%
 languages_mapping: *languages_mapping
 - source: /website/src/pages/**/*
 translation: /website/i18n/%two_letters_code%/docusaurus-plugin-
content-pages/**/%original_file_name%
 ignore: [/**/*.js, /**/*.jsx, /**/*.ts, /**/*.tsx, /**/*.css]
 languages_mapping: *languages_mapping
```

## Machine Translation (MT) issue: links/image handling

Crowdin recently rolled out some major changes to the markdown file format and now the links are treated differently than they were before. Before they were considered as tags, but now they appear as plain text.



Because of these changes the plain text links are passed to the MT engine which attempts to translate the target, thus breaking the translation (for instance: this string `Allez voir [ma merveilleuse page] (/ma-merveilleuse-page)` is translated `Check out [my wonderful page] (/my-wonderful-page)`, and this breaks docusaurus i18n workflow as the page name should not be translated).

As of 2023 Dec.7, they are not planning to change the logic of how links are treated, so you should have this in mind if you plan to use Crowdin with MT.

## 1.3.39 What's next?

Congratulations! You have understood most core features of Docusaurus now. You have:

- [Used the pages plugin](#) to create a standalone React / Markdown page
- [Used the docs plugin](#) to create documentation pages. This includes [configuring the sidebar](#), and even [versioning](#)
- [Used the blog plugin](#) to create a fully featured blog
- Tried your hands on [a range of Markdown features](#), which are useful for all content plugins
- [Used stylesheets](#) or [swizzling](#) to customize your site's appearance
- [Included images and other assets](#) in your pages
- [Added search](#) to your site
- Understood how [browser support](#) and [SEO](#) are done through standard Docusaurus APIs
- Learned about how [individual plugins](#) are installed and configured
- [Deployed](#) your site to a content host
- [Internationalized](#) your site to include multiple languages

At this point, you probably have a big `docusaurus.config.js` already 😊 However, you haven't written much code yet! Most of the features are implemented through calling encapsulated Docusaurus APIs. As you continue your journey, you can take three paths:

- Learn more advanced Docusaurus concepts. This will help you gain a deeper understand of what the APIs do.
- Read about [all existing APIs](#). Many of them have not been covered in the Guides!
- Learn to [develop a plugin](#) to extend the functionality of your site.

## 1.3.40 Head metadata

### Customizing head metadata

Docusaurus automatically sets useful page metadata in `<html>`, `<head>` and `<body>` for you. It is possible to add extra metadata (or override existing ones) with the `<head>` tag in Markdown files:

```
markdown-features-head-metadata.mdx
```



```

id: head-metadata
title: Head Metadata

```

```
<head>
 <html className="some-extra-html-class" />
 <body className="other-extra-body-class" />
 <title>Head Metadata customized title!</title>
 <meta charset="utf-8" />
 <meta name="twitter:card" content="summary" />
 <link rel="canonical" href="https://docusaurus.io/docs/markdown-
features/head-metadata" />
</head>
```

## # Head Metadata

My text

This `<head>` declaration has been added to the current Markdown doc as a demo. Open your browser DevTools and check how this page's metadata has been affected.

### NOTE

This feature is built on top of the Docusaurus `<Head>` component. Refer to [react-helmet](#) for exhaustive documentation.

### YOU DON'T NEED THIS FOR REGULAR SEO

Content plugins (e.g. docs and blog) provide front matter options like `description`, `keywords`, and `image`, which will be automatically applied to both `description` and `og:description`, while you would have to manually declare two metadata tags when using the `<head>` tag.

## Markdown page description

The Markdown pages' description metadata may be used in more places than the head metadata. For example, the docs plugin's [generated category index](#) uses the description metadata for the doc cards.

By default, the description is the first content-ful line, with some efforts to convert it to plain text. For example, the following file...



## # Title

Main content... May contain some `[links](./file.mdx)` or **emphasis**.

...will have the default description "Main content... May contain some links or emphasis". However, **it's not designed to be fully functional**. Where it fails to produce reasonable descriptions, you can explicitly provide one through front matter:

```

description: This description will override the default.

```

## # Title

Main content... May contain some `[links](./file.mdx)` or **emphasis**.

# 1.4 Advanced Tutorials

This section is not going to be very structured, but we will cover the following topics:



## Architecture

How Docusaurus works to build your app



## Plugins

Plugins are the building blocks of features in a Docusaurus site. Each plugin handles its own individual feature. Plugins may work and be distributed as part of a bundle via presets.



## Routing

Docusaurus' routing system follows single-page application conventions: one route, one component.



## Static site generation

Docusaurus statically renders your React code into HTML, allowing faster load speed and better SEO.



## Client architecture

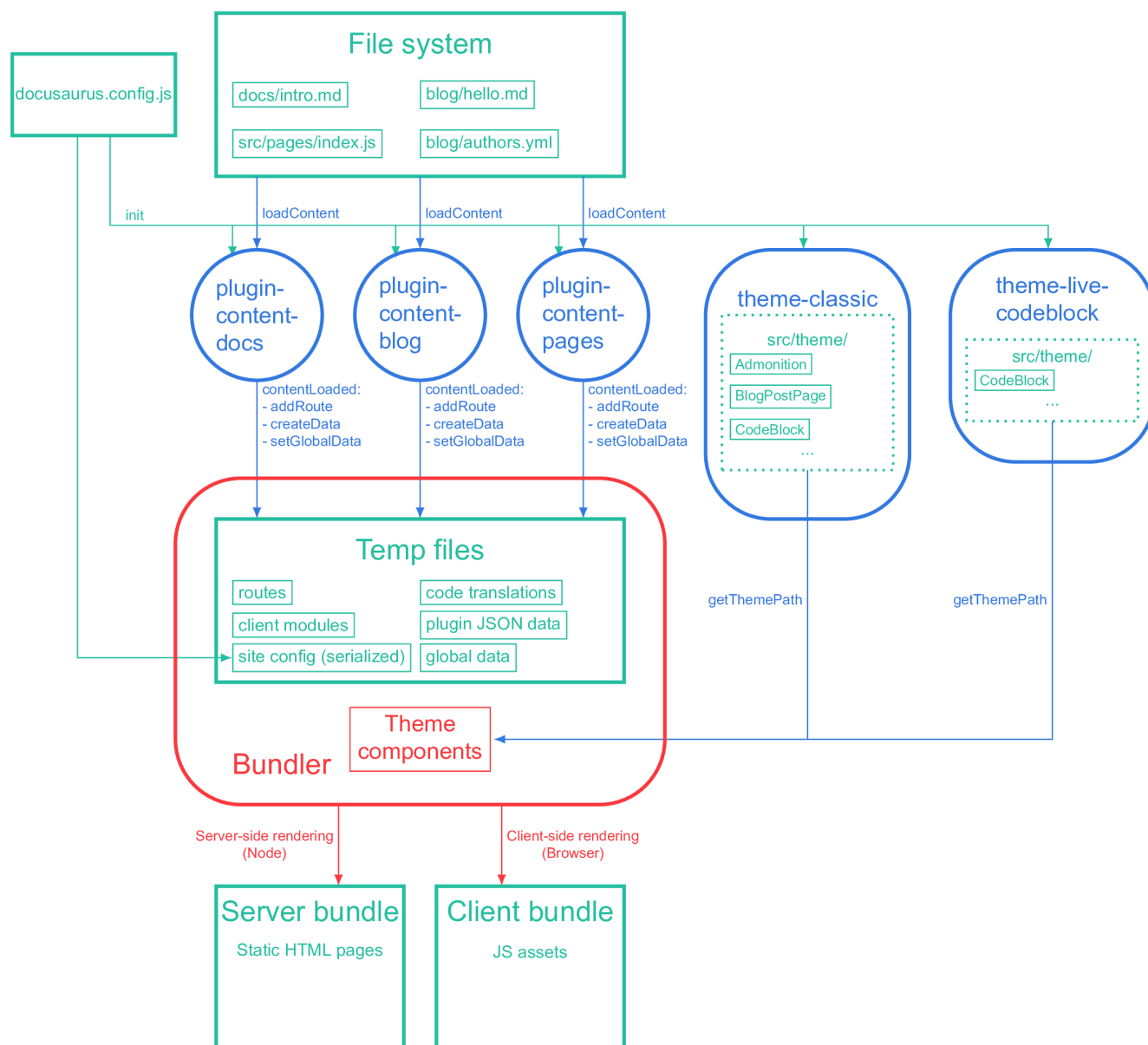
How the Docusaurus client is structured

We will assume that you have finished the guides, and know the basics like how to configure plugins, how to write React components, etc. These sections will have plugin authors and code contributors in mind, so we may occasionally refer to [plugin APIs](#) or other architecture details. Don't panic if you don't understand everything 😊

## 1.5 Advanced Guides



## 1.5.1 Architecture



This diagram shows how Docusaurus works to build your app. Plugins each collect their content and emit JSON data; themes provide layout components which receive the JSON data as route modules. The bundler bundles all the components and emits a server bundle and a client bundle.

Although you (either plugin authors or site creators) are writing JavaScript all the time, bear in mind that the JS is actually run in different environments:

- All plugin lifecycle methods are run in Node. Therefore, until we support ES Modules in our codebase, plugin source code must be provided as ES modules that can be imported, or CommonJS that can be `require`'d.
- The theme code is built with Webpack. They can be provided as ESM—following React conventions.

Plugin code and theme code never directly import each other: they only communicate through protocols (in our case, through JSON temp files and calls to `addRoute`). A useful mental model is to imagine that the plugins are not written in JavaScript, but in another language like Rust. The only way to interact with plugins





for the user is through `docusaurus.config.js`, which itself is run in Node (hence you can use `require` and pass callbacks as plugin options).

During bundling, the config file itself is serialized and bundled, allowing the theme to access config options like `themeConfig` or `baseUrl` through `useDocusaurusContext()`. However, the `siteConfig` object only contains **serializable values** (values that are preserved after `JSON.stringify()`). Functions, regexes, etc. would be lost on the client side. The `themeConfig` is designed to be entirely serializable.

## 1.5.2 Plugins

Plugins are the building blocks of features in a Docusaurus site. Each plugin handles its own individual feature. Plugins may work and be distributed as part of a bundle via presets.

## Creating plugins

A plugin is a function that takes two parameters: `context` and `options`. It returns a plugin instance object (or a promise). You can create plugins as functions or modules. For more information, refer to the [plugin method references section](#).

### Function definition

You can use a plugin as a function directly included in the Docusaurus config file:

`docusaurus.config.js`

```
export default {
 // ...
 plugins: [
 async function myPlugin(context, options) {
 // ...
 return {
 name: 'my-plugin',
 async loadContent() {
 // ...
 },
 async contentLoaded({content, actions}) {
 // ...
 },
 /* other lifecycle API */
 };
 },
],
};
```

### Module definition

You can use a plugin as a module path referencing a separate file or npm package:



docusaurus.config.js

```
export default {
 // ...
 plugins: [
 // without options:
 './my-plugin',
 // or with options:
 ['./my-plugin', options],
],
};
```

Then in the folder `my-plugin`, you can create an `index.js` such as this:

my-plugin/index.js

```
export default async function myPlugin(context, options) {
 // ...
 return {
 name: 'my-plugin',
 async loadContent() {
 /* ... */
 },
 async contentLoaded({content, actions}) {
 /* ... */
 },
 /* other lifecycle API */
 };
}
```

You can view all plugins installed in your site using the [debug plugin's metadata panel](#).

Plugins come as several types:

- `package`: an external package you installed
- `project`: a plugin you created in your project, given to Docusaurus as a local file path
- `local`: a plugin created using the function definition
- `synthetic`: a "fake plugin" Docusaurus created internally, so we take advantage of our modular architecture and don't let the core do much special work. You won't see this in the metadata because it's an implementation detail.



You can access them on the client side with

```
useDocusaurusContext().siteMetadata.pluginVersions.
```

## Plugin design

Docusaurus' implementation of the plugins system provides us with a convenient way to hook into the website's lifecycle to modify what goes on during development/build, which involves (but is not limited to) extending the webpack config, modifying the data loaded, and creating new components to be used in a page.

### Theme design

When plugins have loaded their content, the data is made available to the client side through actions like `createData` + `addRoute` or `setGlobalData`. This data has to be *serialized* to plain strings, because *plugins and themes run in different environments*. Once the data arrives on the client side, the rest becomes familiar to React developers: data is passed along components, components are bundled with Webpack, and rendered to the window through `ReactDOM.render`...

**Themes provide the set of UI components to render the content.** Most content plugins need to be paired with a theme in order to be actually useful. The UI is a separate layer from the data schema, which makes swapping designs easy.

For example, a Docusaurus blog may consist of a blog plugin and a blog theme.

#### NOTE

This is a contrived example: in practice, `@docusaurus/theme-classic` provides the theme for docs, blog, and layouts.

docusaurus.config.js

```
export default {
 themes: ['theme-blog'],
 plugins: ['plugin-content-blog'],
};
```

And if you want to use Bootstrap styling, you can swap out the theme with `theme-blog-bootstrap` (another fictitious non-existing theme):

docusaurus.config.js



```
export default {
 themes: ['theme-blog-bootstrap'],
 plugins: ['plugin-content-blog'],
};
```

Now, although the theme receives the same data from the plugin, how the theme chooses to *render* the data as UI can be drastically different.

While themes share the exact same lifecycle methods with plugins, themes' implementations can look very different from those of plugins based on themes' designed objectives.

Themes are designed to complete the build of your Docusaurus site and supply the components used by your site, plugins, and the themes themselves. A theme still acts like a plugin and exposes some lifecycle methods, but most likely they would not use `loadContent`, since they only receive data from plugins, but don't generate data themselves; themes are typically also accompanied by an `src/theme` directory full of components, which are made known to the core through the `getThemePath` lifecycle.

To summarize:

- Themes share the same lifecycle methods with Plugins
- Themes are run after all existing Plugins
- Themes add component aliases by providing `getThemePath`.

## 1.5.3 Routing

Docusaurus' routing system follows single-page application conventions: one route, one component. In this section, we will begin by talking about routing within the three content plugins (docs, blog, and pages), and then go beyond to talk about the underlying routing system.

### Routing in content plugins

Every content plugin provides a `routeBasePath` option. It defines where the plugins append their routes to. By default, the docs plugin puts its routes under `/docs`; the blog plugin, `/blog`; and the pages plugin, `/`. You can think about the route structure like this:

Any route will be matched against this nested route config until a good match is found. For example, when given a route `/docs/configuration`, Docusaurus first enters the `/docs` branch, and then searches among the subroutes created by the docs plugin.

Changing `routeBasePath` can effectively alter your site's route structure. For example, in **Docs-only mode**, we mentioned that configuring `routeBasePath: '/'` for docs means that all routes that the docs plugin create would not have the `/docs` prefix, yet it doesn't prevent you from having more subroutes like `/blog` created by other plugins.

Next, let's look at how the three plugins structure their own "boxes of subroutes".

### Pages routing



Pages routing are straightforward: the file paths directly map to URLs, without any other way to customize. See the [pages docs](#) for more information.

The component used for Markdown pages is `@theme/MDXPage`. React pages are directly used as the route's component.

## Blog routing

The blog creates the following routes:

- **Posts list pages:** `/`, `/page/2`, `/page/3`...
  - The route is customizable through the `pageBasePath` option.
  - The component is `@theme/BlogListPage`.
- **Post pages:** `/2021/11/21/algolia-docsearch-migration`, `/2021/05/12/announcing-docusaurus-two-beta`...
  - Generated from each Markdown post.
  - The routes are fully customizable through the `slug` front matter.
  - The component is `@theme/BlogPostPage`.
- **Tags list page:** `/tags`
  - The route is customizable through the `tagsBasePath` option.
  - The component is `@theme/BlogTagsListPage`.
- **Tag pages:** `/tags/adoption`, `/tags/beta`...
  - Generated through the tags defined in each post's front matter.
  - The routes always have base defined in `tagsBasePath`, but the subroutes are customizable through the tag's `permalink` field.
  - The component is `@theme/BlogTagsPostsPage`.
- **Archive page:** `/archive`
  - The route is customizable through the `archiveBasePath` option.
  - The component is `@theme/BlogArchivePage`.

## Docs routing

The docs is the only plugin that creates **nested routes**. At the top, it registers **version paths**: `/`, `/next`, `/2.0.0-beta.13`... which provide the version context, including the layout and sidebar. This ensures that when switching between individual docs, the sidebar's state is preserved, and that you can switch between versions through the navbar dropdown while staying on the same doc. The component used is `@theme/DocPage`.

The individual docs are rendered in the remaining space after the navbar, footer, sidebar, etc. have all been provided by the `DocPage` component. For example, this page, `/docs/next/advanced/routing`, is generated from the file at `./docs/advanced/routing.md`. The component used is `@theme/DocItem`.

The doc's `slug` front matter customizes the last part of the route, but the base route is always defined by the plugin's `routeBasePath` and the version's `path`.

## File paths and URL paths

Throughout the documentation, we always try to be unambiguous about whether we are talking about file paths or URL paths. Content plugins usually map file paths directly to URL paths, for example,



`./docs/advanced/routing.md` will become `/docs/advanced/routing`. However, with `slug`, you can make URLs totally decoupled from the file structure.

When writing links in Markdown, you could either mean a *file path*, or a *URL path*, which Docusaurus would use several heuristics to determine.

- If the path has a `@site` prefix, it is *always* an asset file path.
- If the path has an `http(s)://` prefix, it is *always* a URL path.
- If the path doesn't have an extension, it is a URL path. For example, a link `[page](../plugins)` on a page with URL `/docs/advanced/routing` will link to `/docs/plugins`. Docusaurus will only detect broken links when building your site (when it knows the full route structure), but will make no assumptions about the existence of a file. It is exactly equivalent to writing `<a href="../plugins">page</a>` in a JSX file.
- If the path has an `.md(x)` extension, Docusaurus would try to resolve that Markdown file to a URL, and replace the file path with a URL path.
- If the path has any other extension, Docusaurus would treat it as [an asset](#) and bundle it.

The following directory structure may help you visualize this file → URL mapping. Assume that there's no slug customization in any page.

#### A sample site structure



```
.
├── blog # blog plugin has routeBasePath:
│ ├── 2019-05-28-first-blog-post.md # ->
│ │ └── /blog/2019/05/28/first-blog-post
│ ├── 2019-05-29-long-blog-post.md # -> /blog/2019/05/29/long-
│ │ └── blog-post
│ ├── 2021-08-01-mdx-blog-post.mdx # -> /blog/2021/08/01/mdx-
│ │ └── blog-post
│ └── 2021-08-26-welcome
│ ├── docusaurus-plushie-banner.jpeg
│ └── index.md # ->
│ └── /blog/2021/08/26/welcome
├── docs # docs plugin has routeBasePath:
│ ├── intro.md # -> /docs/intro
│ ├── tutorial-basics
│ │ ├── _category_.json
│ │ └── congratulations.md # -> /docs/tutorial-
│ │ └── basics/congratulations
│ │ └── markdown-features.mdx # -> /docs/tutorial-
│ │ └── basics/markdown-features
│ └── tutorial-extras
│ ├── _category_.json
│ └── manage-docs-versions.md # -> /docs/tutorial-
│ └── extras/manage-docs-versions
│ └── translate-your-site.md # -> /docs/tutorial-
│ └── extras/translate-your-site
├── src
│ └── pages # pages plugin has routeBasePath:
│ ├── index.module.css
│ ├── index.tsx # -> /
│ └── markdown-page.md # -> /markdown-page
└── versioned_docs
 ├── version-1.0.0 # version has base path '/1.0.0'
 │ ├── intro.md # -> /docs/1.0.0/intro
 │ ├── tutorial-basics
 │ │ ├── _category_.json
 │ │ └── congratulations.md # -> /docs/1.0.0/tutorial-
 │ │ └── basics/congratulations
 │ │ └── markdown-features.mdx # -> /docs/1.0.0/tutorial-
 │ │ └── basics/markdown-features
 │ └── tutorial-extras
 │ ├── _category_.json
 │ └── manage-docs-versions.md # -> /docs/1.0.0/tutorial-
 │ └── extras/manage-docs-versions
```



```
└─ translate-your-site.md # -> /docs/1.0.0/tutorial-
extras/translate-your-site
```

So much about content plugins. Let's take one step back and talk about how routing works in a Docusaurus app in general.

## Routes become HTML files

Because Docusaurus is a server-side rendering framework, all routes generated will be server-side rendered into static HTML files. If you are familiar with the behavior of HTTP servers like [Apache2](#), you will understand how this is done: when the browser sends a request to the route `/docs/advanced/routing`, the server interprets that as request for the HTML file `/docs/advanced/routing/index.html`, and returns that.

The `/docs/advanced/routing` route can correspond to either `/docs/advanced/routing/index.html` or `/docs/advanced/routing.html`. Some hosting providers differentiate between them using the presence of a trailing slash, and may or may not tolerate the other. Read more in the [trailing slash guide](#).

For example, the build output of the directory above is (ignoring other assets and JS bundle):

**Output of the above workspace**





```
build
├── 404.html # /404/
├── blog
│ ├── archive
│ │ └── index.html # /blog/archive/
│ ├── first-blog-post
│ │ └── index.html # /blog/first-blog-post/
│ ├── index.html # /blog/
│ ├── long-blog-post
│ │ └── index.html # /blog/long-blog-post/
│ ├── mdx-blog-post
│ │ └── index.html # /blog/mdx-blog-post/
│ ├── tags
│ │ ├── docusaurus
│ │ │ └── index.html # /blog/tags/docusaurus/
│ │ ├── hola
│ │ │ └── index.html # /blog/tags/hola/
│ │ └── index.html # /blog/tags/
│ └── welcome
│ └── index.html # /blog/welcome/
├── docs
│ ├── 1.0.0
│ │ ├── intro
│ │ │ └── index.html # /docs/1.0.0/intro/
│ │ ├── tutorial-basics
│ │ │ ├── congratulations
│ │ │ │ └── index.html # /docs/1.0.0/tutorial-
│ │ │ │ basics/congratulations/
│ │ │ └── markdown-features
│ │ │ └── index.html # /docs/1.0.0/tutorial-
│ │ │ basics/markdown-features/
│ │ └── tutorial-extras
│ │ ├── manage-docs-versions
│ │ │ └── index.html # /docs/1.0.0/tutorial-extras/manage-
│ │ │ docs-versions/
│ │ └── translate-your-site
│ │ └── index.html # /docs/1.0.0/tutorial-
│ │ extras/translate-your-site/
│ ├── intro
│ │ └── index.html # /docs/1.0.0/intro/
│ ├── tutorial-basics
│ │ ├── congratulations
│ │ │ └── index.html # /docs/tutorial-
│ │ │ basics/congratulations/
│ │ └── markdown-features
│ │ └── index.html # /docs/tutorial-basics/markdown-
│ │ features/
│ └── tutorial-extras
```



```
├── manage-docs-versions
│ └── index.html # /docs/tutorial-extras/manage-docs-
versions/
│ └── translate-your-site
│ └── index.html # /docs/tutorial-extras/translate-
your-site/
├── index.html # /
└── markdown-page
 └── index.html # /markdown-page/
```

If `trailingSlash` is set to `false`, the build would emit `intro.html` instead of `intro/index.html`.

All HTML files will reference its JS assets using absolute URLs, so in order for the correct assets to be located, you have to configure the `baseUrl` field. Note that `baseUrl` doesn't affect the emitted bundle's file structure: the base URL is one level above the Docusaurus routing system. You can see the aggregate of `url` and `baseUrl` as the actual location of your Docusaurus site.

For example, the emitted HTML would contain links like `<link rel="preload" href="/assets/js/runtime~main.7ed5108a.js" as="script">`. Because absolute URLs are resolved from the host, if the bundle placed under the path `https://example.com/base/`, the link will point to `https://example.com/assets/js/runtime~main.7ed5108a.js`, which is, well, non-existent. By specifying `/base/` as base URL, the link will correctly point to `/base/assets/js/runtime~main.7ed5108a.js`.

Localized sites have the locale as part of the base URL as well. For example, `https://docusaurus.io/zh-CN/docs/advanced/routing/` has base URL `/zh-CN/`.

## Generating and accessing routes

The `addRoute` lifecycle action is used to generate routes. It registers a piece of route config to the route tree, giving a route, a component, and props that the component needs. The props and the component are both provided as paths for the bundler to `require`, because as explained in the [architecture overview](#), server and client only communicate through temp files.

All routes are aggregated in `.docusaurus/routes.js`, which you can view with the debug plugin's [routes panel](#).

On the client side, we offer `@docusaurus/router` to access the page's route. `@docusaurus/router` is a re-export of the `react-router-dom` package. For example, you can use `useLocation` to get the current page's [location](#), and `useHistory` to access the [history object](#). (They are not the same as the browser API, although similar in functionality. Refer to the React Router documentation for specific APIs.)

This API is **SSR safe**, as opposed to the browser-only `window.location`.

```
myComponent.js
```

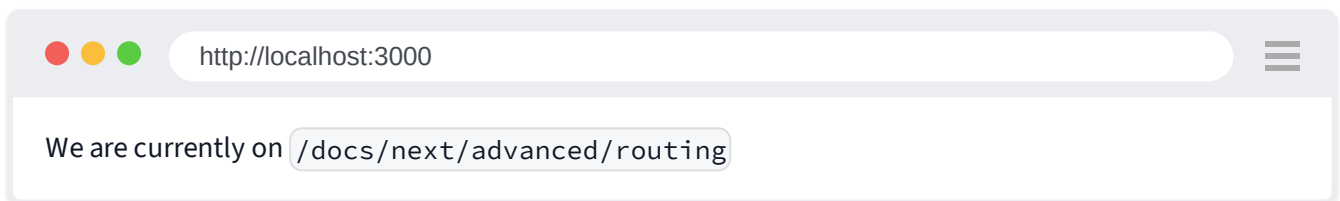


```
import React from 'react';
import {useLocation} from '@docusaurus/router';

export function PageRoute() {
 // React router provides the current component's route, even in SSR
 const location = useLocation();
 return (

 We are currently on <code>{location.pathname}</code>

);
}
```

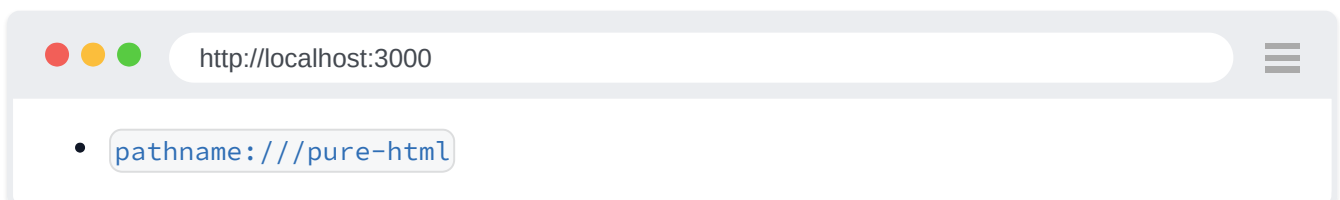


## Escaping from SPA redirects

Docusaurus builds a [single-page application](#), where route transitions are done through the `history.push()` method of React router. This operation is done on the client side. However, the prerequisite for a route transition to happen this way is that the target URL is known to our router. Otherwise, the router catches this path and displays a 404 page instead.

If you put some HTML pages under the `static` folder, they will be copied to the build output and therefore become accessible as part of your website, yet it's not part of the Docusaurus route system. We provide a `pathname://` protocol that allows you to redirect to another part of your domain in a non-SPA fashion, as if this route is an external link.

- `[pathname:///pure-html] (pathname:///pure-html)`



The `pathname://` protocol is useful for referencing any content in the static folder. For example, Docusaurus would convert [all Markdown static assets to require\(\) calls](#). You can use `pathname://` to keep it a regular link instead of being hashed by Webpack.



my-doc.md

```
![An image from the static](pathname:///img/docusaurus.png)

[An asset from the static](pathname:///files/asset.pdf)
```

Docusaurus will only strip the `pathname: //` prefix without processing the content.

## 1.5.4 Static site generation (SSG)

In [architecture](#), we mentioned that the theme is run in Webpack. But beware: that doesn't mean it always has access to browser globals! The theme is built twice:

- During **server-side rendering**, the theme is compiled in a sandbox called [React DOM Server](#). You can see this as a "headless browser", where there is no `window` or `document`, only React. SSR produces static HTML pages.
- During **client-side rendering**, the theme is compiled to JavaScript that gets eventually executed in the browser, so it has access to browser variables.

### ❗ SSR OR SSG?

*Server-side rendering* and *static site generation* can be different concepts, but we use them interchangeably.

Strictly speaking, Docusaurus is a static site generator, because there's no server-side runtime—we statically render to HTML files that are deployed on a CDN, instead of dynamically pre-rendering on each request. This differs from the working model of [Next.js](#).

Therefore, while you probably know not to access Node globals like `process` ([or can we?](#)) or the `'fs'` module, you can't freely access browser globals either.

```
import React from 'react';

export default function WhereAmI() {
 return {window.location.href};
}
```

This looks like idiomatic React, but if you run `docusaurus build`, you will get an error:

```
ReferenceError: window is not defined
```



This is because during server-side rendering, the Docusaurus app isn't actually run in browser, and it doesn't know what `window` is.

### What about `process.env.NODE_ENV`?

One exception to the "no Node globals" rule is `process.env.NODE_ENV`. In fact, you can use it in React, because Webpack injects this variable as a global:

```
import React from 'react';

export default function expensiveComp() {
 if (process.env.NODE_ENV === 'development') {
 return <>This component is not shown in development</>;
 }
 const res = someExpensiveOperationThatLastsALongTime();
 return <>{res}</>;
}
```

During Webpack build, the `process.env.NODE_ENV` will be replaced with the value, either `'development'` or `'production'`. You will then get different build results after dead code elimination:

#### Development      Production

```
import React from 'react';

export default function expensiveComp() {
 if ('development' === 'development') {
+ return <>This component is not shown in development</>;
 }
- const res = someExpensiveOperationThatLastsALongTime();
- return <>{res}</>;
}
```

## Understanding SSR

React is not just a dynamic UI runtime—it's also a templating engine. Because Docusaurus sites mostly contain static contents, it should be able to work without any JavaScript (which React runs in), but only plain HTML/CSS. And that's what server-side rendering offers: statically rendering your React code into



HTML, without any dynamic content. An HTML file has no concept of client state (it's purely markup), hence it shouldn't rely on browser APIs.

These HTML files are the first to arrive at the user's browser screen when a URL is visited (see [routing](#)). Afterwards, the browser fetches and runs other JS code to provide the "dynamic" parts of your site—anything implemented with JavaScript. However, before that, the main content of your page is already visible, allowing faster loading.

In CSR-only apps, all DOM elements are generated on client side with React, and the HTML file only ever contains one root element for React to mount DOM to; in SSR, React is already facing a fully built HTML page, and it only needs to correlate the DOM elements with the virtual DOM in its model. This step is called "hydration". After React has hydrated the static markup, the app starts to work as any normal React app.

Note that Docusaurus is ultimately a single-page application, so static site generation is only an optimization (*progressive enhancement*, as it's called), but our functionality does not fully depend on those HTML files. This is contrary to site generators like [Jekyll](#) and [Docusaurus v1](#), where all files are statically transformed to markup, and interactiveness is added through external JavaScript linked with `<script>` tags. If you inspect the build output, you will still see JS assets under `build/assets/js`, which are, really, the core of Docusaurus.

## Escape hatches

If you want to render any dynamic content on your screen that relies on the browser API to be functional at all, for example:

- Our [live codeblock](#), which runs in the browser's JS runtime
- Our [themed image](#) that detects the user's color scheme to display different images
- The JSON viewer of our debug panel which uses the `window` global for styling

You may need to escape from SSR since static HTML can't display anything useful without knowing the client state.

### WARNING

It is important for the first client-side render to produce the exact same DOM structure as server-side rendering, otherwise, React will correlate virtual DOM with the wrong DOM elements.

Therefore, the naïve attempt of 

```
if (typeof window !== 'undefined') { /* render something */ }
```

 won't work appropriately as a browser vs. server detection, because the first client render would instantly render different markup from the server-generated one.

You can read more about this pitfall in [The Perils of Rehydration](#).

We provide several more reliable ways to escape SSR.

### **<BrowserOnly>**

If you need to render some component in browser only (for example, because the component relies on browser specifics to be functional at all), one common approach is to wrap your component with



`<BrowserOnly>` to make sure it's invisible during SSR and only rendered in CSR.

```
import BrowserOnly from '@docusaurus/BrowserOnly';

function MyComponent(props) {
 return (
 <BrowserOnly fallback={<div>Loading...</div>}>
 {() => {
 const LibComponent =
 require('some-lib-that-accesses-window').LibComponent;
 return <LibComponent {...props} />;
 }}
 </BrowserOnly>
);
}
```

It's important to realize that the children of `<BrowserOnly>` is not a JSX element, but a function that *returns* an element. This is a design decision. Consider this code:

```
import BrowserOnly from '@docusaurus/BrowserOnly';

function MyComponent() {
 return (
 <BrowserOnly>
 {/* DON'T DO THIS - doesn't actually work */}
 page url = {window.location.href}
 </BrowserOnly>
);
}
```

While you may expect that `BrowserOnly` hides away the children during server-side rendering, it actually can't. When the React renderer tries to render this JSX tree, it does see the `{window.location.href}` variable as a node of this tree and tries to render it, although it's actually not used! Using a function ensures that we only let the renderer see the browser-only component when it's needed.

## useIsBrowser

You can also use the `useIsBrowser()` hook to test if the component is currently in a browser environment. It returns `false` in SSR and `true` in CSR, after first client render. Use this hook if you only need to perform certain conditional operations on client-side, but not render an entirely different UI.



```
import useIsBrowser from '@docusaurus/useIsBrowser';

function MyComponent() {
 const isBrowser = useIsBrowser();
 const location = isBrowser ? window.location.href : 'fetching
location...';
 return {location};
}
```

## useEffect

Lastly, you can put your logic in `useEffect()` to delay its execution until after first CSR. This is most appropriate if you are only performing side-effects but don't get data from the client state.

```
function MyComponent() {
 useEffect(() => {
 // Only logged in the browser console; nothing is logged during server-
 side rendering
 console.log("I'm now in the browser");
 }, []);
 return Some content...;
}
```

## ExecutionEnvironment

The `ExecutionEnvironment` namespace contains several values, and `canUseDOM` is an effective way to detect browser environment.

Beware that it essentially checked `typeof window !== 'undefined'` under the hood, so you should not use it for rendering-related logic, but only imperative code, like reacting to user input by sending web requests, or dynamically importing libraries, where DOM isn't updated at all.

a-client-module.js

```
import ExecutionEnvironment from '@docusaurus/ExecutionEnvironment';

if (ExecutionEnvironment.canUseDOM) {
 document.title = "I'm loaded!";
}
```





## 1.5.5 Client architecture

### Theme aliases

A theme works by exporting a set of components, e.g. `Navbar`, `Layout`, `Footer`, to render the data passed down from plugins. Docusaurus and users use these components by importing them using the `@theme` webpack alias:

```
import Navbar from '@theme/Navbar';
```

The alias `@theme` can refer to a few directories, in the following priority:

1. A user's `website/src/theme` directory, which is a special directory that has the higher precedence.
2. A Docusaurus theme package's `theme` directory.
3. Fallback components provided by Docusaurus core (usually not needed).

This is called a *layered architecture*: a higher-priority layer providing the component would shadow a lower-priority layer, making swizzling possible. Given the following structure:

```
website
├── node_modules
│ └── @docusaurus/theme-classic
│ └── theme
│ └── Navbar.js
└── src
 └── theme
 └── Navbar.js
```

`website/src/theme/Navbar.js` takes precedence whenever `@theme/Navbar` is imported. This behavior is called component swizzling. If you are familiar with Objective C where a function's implementation can be swapped during runtime, it's the exact same concept here with changing the target `@theme/Navbar` is pointing to!

We already talked about how the "userland theme" in `src/theme` can re-use a theme component through the `@theme-original` alias. One theme package can also wrap a component from another theme, by importing the component from the initial theme, using the `@theme-init` import.

Here's an example of using this feature to enhance the default theme `CodeBlock` component with a `react-live` playground feature.



```
import InitialCodeBlock from '@theme-init/CodeBlock';
import React from 'react';

export default function CodeBlock(props) {
 return props.live ? (
 <ReactLivePlayground {...props} />
) : (
 <InitialCodeBlock {...props} />
);
}
```

Check the code of `@docusaurus/theme-live-codeblock` for details.

### ⚠ WARNING

Unless you want to publish a re-usable "theme enhancer" (like `@docusaurus/theme-live-codeblock`), you likely don't need `@theme-init`.

It can be quite hard to wrap your mind around these aliases. Let's imagine the following case with a super convoluted setup with three themes/plugins and the site itself all trying to define the same component. Internally, Docusaurus loads these themes as a "stack".

```
+-----+
| `website/src/theme/CodeBlock.js` | <-- `@theme/CodeBlock`
| always points to the top |
+-----+
| `theme-live-codeblock/theme/CodeBlock/index.js` | <-- `@theme-
| original/CodeBlock` points to the topmost non-swizzled component
+-----+
| `plugin-awesome-codeblock/theme/CodeBlock.js` |
+-----+
| `theme-classic/theme/CodeBlock/index.js` | <-- `@theme-
| init/CodeBlock` always points to the bottom
+-----+
```

The components in this "stack" are pushed in the order of `preset plugins > preset themes > plugins > themes > site`, so the swizzled component in `website/src/theme` always comes out on top because it's loaded last.

`@theme/*` always points to the topmost component—when `CodeBlock` is swizzled, all other components requesting `@theme/CodeBlock` receive the swizzled version.

`@theme-original/*` always points to the topmost non-swizzled component. That's why you can import `@theme-original/CodeBlock` in the swizzled component—it points to the next one in the "component



stack", a theme-provided one. Plugin authors should not try to use this because your component could be the topmost component and cause a self-import.

`@theme-init/*` always points to the bottommost component—usually, this comes from the theme or plugin that first provides this component. Individual plugins / themes trying to enhance code block can safely use `@theme-init/CodeBlock` to get its basic version. Site creators should generally not use this because you likely want to enhance the *topmost* instead of the *bottommost* component. It's also possible that the `@theme-init/CodeBlock` alias does not exist at all—Docusaurus only creates it when it points to a different one from `@theme-original/CodeBlock`, i.e. when it's provided by more than one theme. We don't waste aliases!

## Client modules

Client modules are part of your site's bundle, just like theme components. However, they are usually side-effect-ful. Client modules are anything that can be `import`ed by Webpack—CSS, JS, etc. JS scripts usually work on the global context, like registering event listeners, creating global variables...

These modules are imported globally before React even renders the initial UI.

```
@docusaurus/core/App.tsx
```

```
// How it works under the hood
import '@generated/client-modules';
```

Plugins and sites can both declare client modules, through `getClientModules` and `siteConfig.clientModules`, respectively.

Client modules are called during server-side rendering as well, so remember to check the [execution environment](#) before accessing client-side globals.

```
mySiteGlobalJs.js
```

```
import ExecutionEnvironment from '@docusaurus/ExecutionEnvironment';

if (ExecutionEnvironment.canUseDOM) {
 // As soon as the site loads in the browser, register a global event
 listener
 window.addEventListener('keydown', (e) => {
 if (e.code === 'Period') {
 location.assign(location.href.replace('.com', '.dev'));
 }
 });
}
```



CSS stylesheets imported as client modules are [global](#).

```
mySiteGlobalCss.css
```

```
/* This stylesheet is global. */
.globalSelector {
 color: red;
}
```

## Client module lifecycles

Besides introducing side-effects, client modules can optionally export two lifecycle functions:

`onRouteUpdate` and `onRouteDidUpdate`.

Because Docusaurus builds a single-page application, `script` tags will only be executed the first time the page loads, but will not re-execute on page transitions. These lifecycles are useful if you have some imperative JS logic that should execute every time a new page has loaded, e.g., to manipulate DOM elements, to send analytics data, etc.

For every route transition, there will be several important timings:

1. The user clicks a link, which causes the router to change its current location.
2. Docusaurus preloads the next route's assets, while keeping displaying the current page's content.
3. The next route's assets have loaded.
4. The new location's route component gets rendered to DOM.

`onRouteUpdate` will be called at event (2), and `onRouteDidUpdate` will be called at (4). They both receive the current location and the previous location (which can be `null`, if this is the first screen).

`onRouteUpdate` can optionally return a "cleanup" callback, which will be called at (3). For example, if you want to display a progress bar, you can start a timeout in `onRouteUpdate`, and clear the timeout in the callback. (The classic theme already provides an `nprogress` integration this way.)

Note that the new page's DOM is only available during event (4). If you need to manipulate the new page's DOM, you'll likely want to use `onRouteDidUpdate`, which will be fired as soon as the DOM on the new page has mounted.

```
myClientModule.js
```



```
export function onRouteDidUpdate({location, previousLocation}) {
 // Don't execute if we are still on the same page; the lifecycle may be
 // fired
 // because the hash changes (e.g. when navigating between headings)
 if (location.pathname !== previousLocation?.pathname) {
 const title = document.getElementsByTagName('h1')[0];
 if (title) {
 title.innerText += '❤️';
 }
 }
}

export function onRouteUpdate({location, previousLocation}) {
 if (location.pathname !== previousLocation?.pathname) {
 const progressBarTimeout = window.setTimeout(() => {
 nprogress.start();
 }, delay);
 return () => window.clearTimeout(progressBarTimeout);
 }
 return undefined;
}
```

Or, if you are using TypeScript and you want to leverage contextual typing:

myClientModule.ts

```
import type {ClientModule} from '@docusaurus/types';

const module: ClientModule = {
 onRouteUpdate({location, previousLocation}) {
 // ...
 },
 onRouteDidUpdate({location, previousLocation}) {
 // ...
 },
};
export default module;
```

Both lifecycles will fire on first render, but they will not fire on server-side, so you can safely access browser globals in them.



**PREFER USING REACT**



Client module lifecycles are purely imperative, and you can't use React hooks or access React contexts within them. If your operations are state-driven or involve complicated DOM manipulations, you should consider [swizzling components](#) instead.

## 1.6 Upgrading Docusaurus

Docusaurus versioning is based on the `major.minor.patch` scheme and respects [Semantic Versioning](#).

**Breaking changes** are only released on major version upgrades, and thoroughly documented in the following upgrade guides.



### To Docusaurus v3

This documentation will help you upgrade your site from Docusaurus v2 to Docusaurus v3.



### To Docusaurus v2

5 items

### 1.6.1 Troubleshooting upgrades

When upgrading Docusaurus you may experience issues caused by mismatching cached dependencies - there are a few troubleshooting steps you should perform to resolve these common issues before reporting a bug or seeking support.

#### Run the `clear` command

This CLI command is used to clear a Docusaurus site's generated assets, caches and build artifacts.

**npm**

Yarn

pnpm

Bun

```
npm run clear
```

#### Remove `node_modules` and your lock file(s)

Remove the `node_modules` folder and your package manager's lock file using the following:



**Bash**   **PowerShell**

```
rm -rf node_modules yarn.lock package-lock.json
```

Then reinstall packages and regenerate the `lock` file using:

**npm**   **Yarn**   **pnpm**   **Bun**

```
npm install
```

## 1.7 Upgrading

### 1.7.1 Upgrading to Docusaurus v3

This documentation will help you upgrade your site from Docusaurus v2 to Docusaurus v3.

Docusaurus v3 is a new **major version**, including **breaking changes** requiring you to adjust your site accordingly. We will guide to during this process, and also mention a few optional recommendations.

This is not a full rewrite, and the breaking changes are relatively easy to handle. The simplest sites will eventually upgrade by simply updating their npm dependencies.

The main breaking change is the upgrade from MDX v1 to MDX v3. Read the [MDX v2](#) and [MDX v3](#) release notes for details. MDX will now compile your Markdown content **more strictly** and with **subtle differences**.



#### BEFORE UPGRADING

Before upgrading, we recommend [preparing your site for Docusaurus v3](#). There are changes that you can already **handle incrementally, under Docusaurus v2**. Doing so will help reduce the work needed to finally upgrade to Docusaurus v3.

For complex sites, we also recommend to set up [visual regression tests](#), a good way to ensure your site stays visually identical. Docusaurus v3 mainly upgrades dependencies, and is not expected to produce any visual changes.



#### NOTE

Check the release notes for [Docusaurus v3.0.0](#), and browse the pull-requests for additional useful information and the motivation behind each change mentioned here.



# Upgrading Dependencies

Upgrading to Docusaurus v3 requires upgrading core Docusaurus dependencies (`@docusaurus/name`), but also other related packages.

Docusaurus v3 now uses the following dependencies:

- Node.js v18.0+
- React v18.0+
- MDX v3.0+
- TypeScript v5.1+
- prism-react-renderer v2.0+
- react-live v4.0+
- remark-emoji v4.0+
- mermaid v10.4+

## UPGRADING COMMUNITY PLUGINS

If your site uses third-party community plugins and themes, you might need to upgrade them.

Make sure those plugins are compatible with Docusaurus v3 before attempting an upgrade.

A typical `package.json` dependency upgrade example:

```
package.json
```





```
{
 "dependencies": {
 // upgrade to Docusaurus v3
 - "@docusaurus/core": "2.4.3",
 - "@docusaurus/preset-classic": "2.4.3",
 + "@docusaurus/core": "3.0.0",
 + "@docusaurus/preset-classic": "3.0.0",
 // upgrade to MDX v3
 - "@mdx-js/react": "^1.6.22",
 + "@mdx-js/react": "^3.0.0",
 // upgrade to prism-react-renderer v2.0+
 - "prism-react-renderer": "^1.3.5",
 + "prism-react-renderer": "^2.1.0",
 // upgrade to React v18.0+
 - "react": "^17.0.2",
 - "react-dom": "^17.0.2"
 + "react": "^18.2.0",
 + "react-dom": "^18.2.0"
 },
 "devDependencies": {
 // upgrade Docusaurus dev dependencies to v3
 - "@docusaurus/module-type-aliases": "2.4.3",
 - "@docusaurus/types": "2.4.3"
 + "@docusaurus/module-type-aliases": "3.0.0",
 + "@docusaurus/types": "3.0.0"
 }
 "engines": {
 // require Node.js 18.0+
 - "node": ">=16.14"
 + "node": ">=18.0"
 }
}
```

For TypeScript users:

```
package.json
```



```
{
 "devDependencies": {
 // swap the external TypeScript config package for the new official
 one
 - "@tsconfig/docusaurus": "^1.0.7",
 + "@docusaurus/tsconfig": "3.0.0",
 // upgrade React types to v18.0+
 - "@types/react": "^17.0.69",
 + "@types/react": "^18.2.29",
 // upgrade TypeScript to v5.1+
 - "typescript": "~4.7.4"
 + "typescript": "~5.2.2"
 }
}
```

## Upgrading MDX

MDX is a major dependency of Docusaurus responsible for compiling your `.md` and `.mdx` files to React components.

The transition from MDX v1 to MDX v3 is the **main challenge** to the adoption of Docusaurus v3. Most breaking changes come from MDX v2, and MDX v3 is a relatively small release.

Some documents that compiled successfully under Docusaurus v2 might now **fail to compile** under Docusaurus v3.



### FIND PROBLEMATIC CONTENT AHEAD OF TIME

Run `npx docusaurus-mdx-checker` on your site to get a list of files that will now fail to compile under Docusaurus v3.

This command is also a good way to estimate the amount of work to be done to make your content compatible. Remember most of this work can be executed ahead of the upgrade by [preparing your content for Docusaurus v3](#).

Other documents might also **render differently**.



### USE VISUAL REGRESSION TESTS

For large sites where a manual review of all pages is complicated, we recommend you to setup [visual regression tests](#).

Upgrading MDX comes with all the breaking changes documented on the [MDX v2](#) and [MDX v3](#) release blog posts. Most breaking changes come from MDX v2, and MDX v3 is a relatively small release. The [MDX v2](#)



[migration guide](#) has a section on how to [update MDX files](#) that will be particularly relevant to us. Also make sure to read the [Troubleshooting MDX](#) page that can help you interpret common MDX error messages.

Make sure to also read our updated [MDX and React](#) documentation page.

## Using the MDX playground

The MDX playground is your new best friend. It permits to understand how your content is **compiled to React components**, and troubleshoot compilation or rendering issues in isolation.

- [MDX playground - current version](#)
- [MDX playground - v1](#)

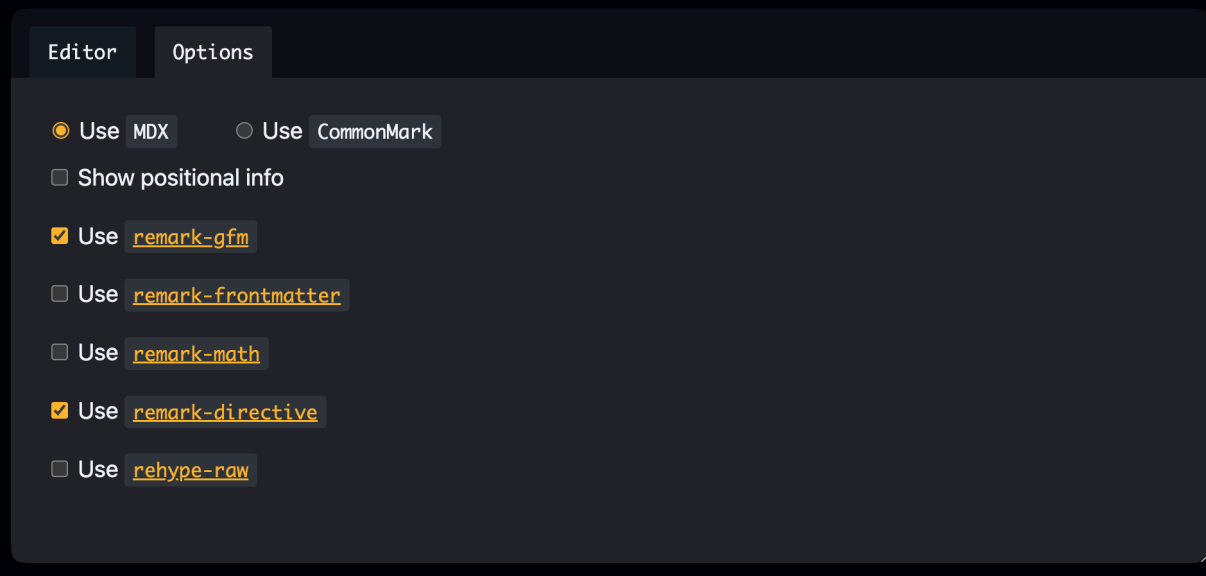
### Configuring the MDX playground options for Docusaurus

To obtain a compilation behavior similar to what Docusaurus v2 uses, please turn on these options on the [MDX playground](#):

- Use `MDX`
- Use `remark-gfm`
- Use `remark-directive`

## Playground

Here you can play with the MDX format. Write some MDX to find out what it turns into. You can see the rendered result, the generated code, and the intermediary ASTs. This can be helpful for debugging or exploring. To read about how the MDX format works, we recommend that you start with [§ What is MDX](#).



The screenshot shows the MDX Playground interface with the 'Options' tab selected. The 'Editor' tab is also visible. The options are as follows:

- ☒ Use `MDX`    ☐ Use `CommonMark`
- ☐ Show positional info
- ☒ Use `remark-gfm`
- ☐ Use `remark-frontmatter`
- ☐ Use `remark-math`
- ☒ Use `remark-directive`
- ☐ Use `rehype-raw`

Using the two MDX playgrounds side-by-side, you will soon notice that some content is compiled differently or fails to compile in v2.



## MAKING YOUR CONTENT FUTURE-PROOF

The goal will be to refactor your problematic content so that it **works fine with both versions of MDX**. This way, when you upgrade to Docusaurus v3, this content will already work out-of-the-box.

## Using the MDX checker CLI

We provide a [docusaurus-mdx-checker](#) CLI that permits to easily spot problematic content. Run this command on your site to obtain a list of files that will fail to compile under MDX v3.

```
npx docusaurus-mdx-checker
```

For each compilation issue, the CLI will log the file path and a line number to look at.

```
[+ docusaurus-v2 git:(main) npx docusaurus-mdx-checker
[ERROR] 7/248 MDX files couldn't compile!

Error while compiling file _dogfooding/_pages tests/markdownPageTests.md
Details: These MDX global variables do not seem to be available in scope: b

Error while compiling file docs/api/plugins/plugin-pwa.md (Line=123 Column=48)
Details: Unexpected character '=' (U+003D) before name, expected a character that can start a name, such as a letter, '$', or '_'

Error while compiling file docs/deployment.mdx (Line=443 Column=1)
Details: Unexpected closing tag '</details>', expected corresponding closing tag for '<TabItem>' (320:1-320:48)

Error while compiling file docs/introduction.md (Line=57 Column=6)
Details: Unexpected character '/' (U+002F) after self-closing slash, expected '>' to end the tag (note: JS comments in JSX tags are not supported in MDX)

Error while compiling file versioned_docs/version-2.0.0-rc.1/api/plugins/plugin-pwa.md (Line=123 Column=48)
Details: Unexpected character '=' (U+003D) before name, expected a character that can start a name, such as a letter, '$', or '_'

Error while compiling file versioned_docs/version-2.0.0-rc.1/deployment.mdx (Line=443 Column=1)
Details: Unexpected closing tag '</details>', expected corresponding closing tag for '<TabItem>' (320:1-320:48)

Error while compiling file versioned_docs/version-2.0.0-rc.1/introduction.md (Line=57 Column=6)
Details: Unexpected character '/' (U+002F) after self-closing slash, expected '>' to end the tag (note: JS comments in JSX tags are not supported in MDX)
```



## TIP

Use this CLI to estimate of how much work will be required to make your content compatible with MDX v3.



## WARNING

This CLI is a best effort, and will **only report compilation errors**.

It will not report subtle compilation changes that do not produce errors but can affect how your content is displayed. To catch these problems, we recommend using [visual regression tests](#).

## Common MDX problems

Docusaurus cannot document exhaustively all the changes coming with MDX. That's the responsibility of the [MDX v2](#) and [MDX v3](#) migration guides.



However, by upgrading a few Docusaurus sites, we noticed that most of the issues come down to only a few cases that we have documented for you.

## Bad usage of `{`

The `{` character is used for opening [JavaScript expressions](#). MDX will now fail if what you put inside `{expression}` is not a valid expression.

example.md

```
The object shape looks like {username: string, age: number}
```



### ERROR MESSAGE

Could not parse expression with acorn: Unexpected content after expression



### HOW TO UPGRADE

Available options to fix this error:

- Use inline code: `{username: string, age: number}`
- Use the HTML code: `&#123;`
- Escape it: `\{`

## Bad usage of `<`

The `<` character is used for opening [JSX tags](#). MDX will now fail if it thinks your JSX is invalid.

example.md

```
Use Android version <5
```

```
You can use a generic type like Array<T>
```

```
Follow the template "Road to <YOUR_MINOR_VERSION>"
```



### ERROR MESSAGES



Unexpected character `5` (U+0035) before name, expected a character that can start a name, such as a letter, `$`, or `_`

Expected a closing tag for `<T>` (1:6-1:9) before the end of `paragraph` end-tag-mismatch mdast-util-mdx-jsx

Expected a closing tag for `<YOUR_MINOR_VERSION>` (134:19-134:39) before the end of `paragraph`



### HOW TO UPGRADE

Available options to fix this error:

- Use inline code: `Array<T>`
- Use the HTML code: `&lt;` or `&#60;`
- Escape it: `\<`

## Bad usage of GFM Autolink

Docusaurus supports [GitHub Flavored Markdown \(GFM\)](#), but [autolink](#) using the `<link>` syntax is not supported anymore by MDX.

example.md

```
<sebastien@thisweekinreact.com>
```

```
<http://localhost:3000>
```



### ERROR MESSAGES

Unexpected character `@` (U+0040) in name, expected a name character such as letters, digits, `$`, or `_`; whitespace before attributes; or the end of the tag (note: to create a link in MDX, use `[text](url)`)

Unexpected character `/` (U+002F) before local name, expected a character that can start a name, such as a letter, `$`, or `_` (note: to create a link in MDX, use `[text](url)`)



### HOW TO UPGRADE

Use regular Markdown links, or remove the `<` and `>`. MDX and GFM are able to autolink literals already.



example.md

```
sebastien@thisweekinreact.com
[sebastien@thisweekinreact.com] (mailto:sebastien@thisweekinreact.com)

http://localhost:3000
[http://localhost:3000] (http://localhost:3000)
```

## Lower-case MDXComponent mapping

For users providing a **custom MDXComponent mapping**, components are now "sandboxed":

- a **MDXComponent** mapping for **h1** only gets used for **# hi** but not for **<h1>hi</h1>**
- a **lower-cased** custom element name will not be substituted by its respective **MDXComponent** component anymore



### VISUAL DIFFERENCE

Your **MDXComponent component mapping** might not be applied as before, and your custom components might no longer be used.



### HOW TO UPGRADE

For native Markdown elements, you can keep using **lower-case**: **p**, **h1**, **img**, **a**...

For any other element, **use upper-case names**.

src/theme/MDXComponents.js

```
import MDXComponents from '@theme-original/MDXComponents';

export default {
 ...MDXComponents,
 p: (props) => <p {...props} className="my-paragraph"/>
- myElement: (props) => <div {...props} className="my-class" />,
+ MyElement: (props) => <div {...props} className="my-class" />,
};
```

## Unintended extra paragraphs

In MDX v3, it is now possible to interleave JSX and Markdown more easily without requiring extra line breaks. Writing content on multiple lines can also produce new expected **<p>** tags.



## VISUAL DIFFERENCE

See how this content is rendered differently by MDX v1 and v3.

example.md

```
<div>Some **Markdown** content</div>
<div>
 Some **Markdown** content
</div>
```

MDX v1 output

```
<div>Some **Markdown** content</div>
<div>Some **Markdown** content</div>
```

MDX v3 output

```
<div>Some Markdown content</div>
<div><p>Some Markdown content</p></div>
```



## HOW TO UPGRADE

If you don't want an extra `<p>` tag, refactor content on a case by case basis to use a single-line JSX tag.

```
<figure>

- <figcaption>
- My image caption
- </figcaption>
+ <figcaption>My image caption</figcaption>
</figure>
```

You can also wrap such content with `{` and `}` to avoid extra `<p>` tags if you don't intend to use Markdown syntax there yet.





```
-<figure>
+{<figure>

 <figcaption>
 My image caption
 </figcaption>
-</figure>
+</figure>}
```

## Unintended usage of directives

Docusaurus v3 now uses [Markdown Directives](#) (implemented with [remark-directive](#)) as a generic way to provide support for admonitions, and other upcoming Docusaurus features.

example.md

```
This is a :textDirective

::leafDirective

:::containerDirective
Container directive content
:::
```

### VISUAL CHANGE

Directives are parsed with the purpose of being handled by other Remark plugins. Unhandled directives will be ignored, and won't be rendered back in their original form.

example.md

```
The AWS re:Invent conf is great
```

Due to `:Invent` being parsed as a text directive, this will now be rendered as:



The AWS re  
conf is great



#### HOW TO UPGRADE

- Use the HTML code: `&#58;`
- Add a space after `:` (if it makes sense): `: text`
- Escape it: `\:`

## Unsupported indented code blocks

MDX does not transform indented text as code blocks anymore.

example.md

```
console.log("hello");
```



#### VISUAL CHANGE

The upgrade does not generally produce new MDX compilation errors, but can lead to content being rendered in an unexpected way because there isn't a code block anymore.



#### HOW TO UPGRADE

Use the regular code block syntax instead of indentation:

example.md

```
```js
console.log('hello');
```
```

## Other Markdown incompatibilities

### Emphasis starting or ending with a space or a punctuation

New MDX parser now strictly complies with the CommonMark spec. CommonMark spec has introduced rules for emphasis around spaces and punctuation, which are incompatible especially with languages that do not



use a space to split words, since v0.14.

Japanese and Chinese are most affected by this, but there are some other languages that can be affected (e.g. Thai and Khmer), for example when you try to emphasize an inline code or a link. Languages that use a space to split words are much less affected.

`**` (other than ``**``) in the following example were parsed as intended in Docusaurus 2, but are not now in Docusaurus 3.

example.md

```
**Do not end a range of emphasis with a space. **Or `**` will not work as intended.
```

```
<!-- Japanese -->
```

```
「。」の後に文を続けると`**`が意図した動作をしません。 **また、[リンク]
(https://docusaurus.io/)**や**`コード`**のすぐ外側に`**`、そのさらに外側に句読点以外
がある場合も同様です。
```

### See the detailed conditions and how to upgrade

If `*` or `**` matches either of the following conditions, it will not work as the beginning of an emphasis mark anymore:

- The next character is a space (e.g. `word* word`)
- The previous character is a punctuation character and the next character is a letter (not a space or punctuation character) (e.g. `文** (文)`)

On the contrary, if `*` or `**` matches either of the following conditions, it will not work as the end of an emphasis mark anymore:

- The previous character is a space (e.g. `word *word`)
- The next character is a punctuation character and the previous character is a letter (e.g. `文。 **文`)

“A punctuation character” includes non-ASCII ones, brackets, quotation marks and some symbols including `%` and `@`. More strictly speaking, a character whose 2-letters Unicode category starts with `P` is treated as a punctuation character here.



#### HOW TO UPGRADE

If the offending emphasis mark is next to a space, move the space out of the range of emphasis:



english.md

```
Do not end a range of emphasis with a space. Or `**` will not work.
```

If the offending emphasis mark is surrounded by both a punctuation character and a letter, you can fix it without modifying the content by:

1. Convert the document to MDX if it is a vanilla Markdown.
2. replace the offending emphasis mark with a raw HTML tag (`<em>` or `<strong>`) instead:

japanese.mdx

```
「。」の後に文を続けると`**`が意図した動作をしません。また、
[リンク](https://docusaurus.io/)や`コード`
のすぐ外側に`**`、そのさらに外側に句読点以外がある場合も同様です。
```

While not an ideal solution, you can also use either of the following without converting the document to MDX:

- Move the most outside punctuation character out of the emphasis mark.

japanese.md

```
「。」の後に文を続けると`**`が意図した動作をしません。また、[**リンク**]
**](https://docusaurus.io/)や・・・
```

- Put a space just outside of the offending `*` or `**`. This solution does not force you to convert the document to MDX.

japanese.md



\*\*「。」」の後に文を続けると`\*\*`が意図した動作をしません。\*\* また、\*\*[リンク](https://docusaurus.io/)\*\* や \*\*`コード`\*\* のすぐ外側に`\*\*`、そのさらに外側に句読点以外がある場合も同様です。

An unofficial remark plugin [remark-cjk-friendly](#) can fix this issue in most cases without modifying the Markdown source written in Chinese, Japanese, and Korean like the above.

## MDX plugins

All the official packages (Unified, Remark, Rehype...) in the MDX ecosystem are now **ES Modules only** and do not support **CommonJS** anymore.

In practice this means that you can't do `require("remark-plugin")` anymore.



### HOW TO UPGRADE

Docusaurus v3 now supports **ES Modules** configuration files. We recommend that you migrate your config file to ES module, that enables you to import the Remark plugins easily:

docusaurus.config.js

```
import remarkPlugin from 'remark-plugin';

export default {
 title: 'Docusaurus',
 /* site config using remark plugins here */
};
```

If you want to keep using CommonJS modules, you can use dynamic imports as a workaround that enables you to import ES modules inside a CommonJS module. Fortunately, the [Docusaurus config](#) supports the usage of an [async function](#) to let you do so.

docusaurus.config.js



```
module.exports = async function () {
 const myPlugin = (await import('remark-plugin')).default;
 return {
 // site config...
 };
};
```

### ! FOR PLUGIN AUTHORS

If you created custom Remark or Rehype plugins, you may need to refactor those, or eventually rewrite them completely, due to how the new AST is structured. We have created a [dedicated support discussion](#) to help plugin authors upgrade their code.

## Formatters

Prettier, the most common formatter, supports only the legacy MDX v1, not v3 yet as of Docusaurus v3.0.0. You can add `{/* prettier-ignore */}` before the incompatible parts of your code to make it work with Prettier.

```
{/* prettier-ignore */
<SomeComponent>Some long text in the component</SomeComponent>
```

If you get tired of too many `{/* prettier-ignore */}` insertions, you can consider disabling MDX formatting by Prettier by adding the following to your `.prettiignore` file, until it starts supporting MDX v3:

```
.prettiignore
```

```
*.mdx
```

## Other Breaking Changes

Apart the MDX v3 upgrade, here is an exhaustive list of breaking changes coming with Docusaurus v3.

### Node.js v18.0

Node.js 16 [reached End-of-Life](#), and Docusaurus v3 now requires **Node.js >= 18.0**.



### HOW TO UPGRADE



Install Node.js 18.0+ on your computer.

Eventually, configure your continuous integration, CDN or host to use this new Node.js version.

You can also update your site `package.json` to prevent usage of an older unsupported version:

package.json

```
{
 "engines": {
 - "node": ">=16.14"
 + "node": ">=18.0"
 }
}
```

Upgrade your Docusaurus v2 site to Node.js 18 before upgrading to Docusaurus v3.

## React v18.0+

Docusaurus v3 now requires **React >= 18.0**.

React 18 comes with its own breaking changes that should be relatively easy to handle, depending on the amount of custom React code you created for your site. The official themes and plugins are compatible with React 18.

### ❗ HOW TO UPGRADE

Read the official [React v18.0](#) and [How to Upgrade to React 18](#), and look at your own React code to figure out which components might be affected by this upgrade.

We recommend to particularly look for:

- Automatic batching for stateful components
- New React hydration errors reported to the console



### EXPERIMENTAL SUPPORT FOR REACT 18 FEATURES

React 18 comes with new features:

- `<Suspense>`
- `React.lazy()`
- `startTransition`



Their Docusaurus support is considered as experimental. We might have to adjust the integration in the future, leading to a different runtime behavior.

## Prism-React-Renderer v2.0+

Docusaurus v3 upgrades `prism-react-renderer` to v2.0+. This library is used for code block syntax highlighting.

### ! HOW TO UPGRADE

This is a new major library version containing breaking changes, and we can't guarantee a strict retro-compatibility. The `prism-react-renderer` v2 release notes are not super exhaustive, but there are 3 major changes to be aware of for Docusaurus users.

The dependency should be upgraded:

package.json

```
{
 "dependencies": {
 - "prism-react-renderer": "^1.3.5",
 + "prism-react-renderer": "^2.1.0",
 }
}
```

The API to import themes in your Docusaurus config file has been updated:

docusaurus.config.js

```
- const lightTheme = require('prism-react-renderer/themes/github');
- const darkTheme = require('prism-react-renderer/themes/dracula');
+ const {themes} = require('prism-react-renderer');
+ const lightTheme = themes.github;
+ const darkTheme = themes.dracula;
```

Previously, `react-prism-renderer` v1 included more languages by default. From v2.0+, fewer languages are included by default. You may need to add extra languages to your Docusaurus config:

docusaurus.config.js





```
const siteConfig = {
 themeConfig: {
 prism: {
 additionalLanguages: ['bash', 'diff', 'json'],
 },
 },
};
```

## React-Live v4.0+

For users of the `@docusaurus/theme-live-codeblock` package, Docusaurus v3 upgrades `react-live` to v4.0+.

### ! HOW TO UPGRADE

In theory, you have nothing to do, and your existing interactive code blocks should keep working as before.

However, this is a new major library version containing breaking changes, and we can't guarantee a strict retro-compatibility. Read the [v3](#) and [v4](#) changelogs in case of problems.

## remark-emoji v4.0+

Docusaurus v3 upgrades `remark-emoji` to v4.0+. This library is to support `:emoji:` shortcuts in Markdown.

### ! HOW TO UPGRADE

Most Docusaurus users have nothing to do. Users of emoji shortcodes should read the [changelog](#) and double-check their emojis keep rendering as expected.

**Breaking Change** Update `node-emoji` from v1 to v2. This change introduces support for many new emojis and removes old emoji short codes which are no longer valid on GitHub.

## Mermaid v10.4+

For users of the `@docusaurus/theme-mermaid` package, Docusaurus v3 upgrades `mermaid` to v10.4+.

### ! HOW TO UPGRADE

In theory, you have nothing to do, and your existing diagrams should keep working as before.

However, this is a new major library version containing breaking changes, and we can't guarantee a strict retro-compatibility. Read the [v10](#) changelog in case of problem.



## TypeScript v5.1+

Docusaurus v3 now requires **TypeScript >= 5.1**.

### ! HOW TO UPGRADE

Upgrade your dependencies to use TypeScript 5+

package.json

```
{
 "devDependencies": {
 - "typescript": "~4.7.4"
 + "typescript": "~5.2.2"
 }
}
```

## TypeScript base config

The official Docusaurus TypeScript config has been re-internalized from the external package `@tsconfig/docusaurus` to our new monorepo package `@docusaurus/tsconfig`.

This new package is versioned alongside all the other Docusaurus core packages, and will be used to ensure TypeScript retro-compatibility and breaking changes on major version upgrades.

### ! HOW TO UPGRADE

Swap the external TypeScript config package for the new official one

package.json

```
{
 "devDependencies": {
 - "@tsconfig/docusaurus": "^1.0.7",
 + "@docusaurus/tsconfig": "3.0.0",
 }
}
```

Use it in your `tsconfig.json` file:

tsconfig.json



```
{
 - "extends": "@tsconfig/docusaurus/tsconfig.json",
 + "extends": "@docusaurus/tsconfig",
 "compilerOptions": {
 "baseUrl": "."
 }
}
```

## New Config Loader

Docusaurus v3 changes its internal config loading library from `import-fresh` to `jiti`. It is responsible for loading files such as `docusaurus.config.js` or `sidebars.js`, and Docusaurus plugins.

### ! HOW TO UPGRADE

In theory, you have nothing to do, and your existing config files should keep working as before.

However, this is a major dependency swap and subtle behavior changes could occur.

## Admonition Warning

For historical reasons, we support an undocumented admonition `:::warning` that renders with a red color.

### WARNING

This is a Docusaurus v2 `:::warning` admonition.

However, the color and icon have always been wrong. Docusaurus v3 re-introduces `:::warning` admonition officially, documents it, and fixes the color and icon.

### WARNING

This is a Docusaurus v3 `:::warning` admonition.

### ! HOW TO UPGRADE

If you previously used the undocumented `:::warning` admonition, make sure to verify for each usage if yellow is now an appropriate color. If you want to keep the red color, use `:::danger` instead.

Docusaurus v3 also deprecated the `:::caution` admonition. Please refactor `:::caution` (yellow) to either `:::warning` (yellow) or `:::danger` (red).

If you want to keep the title “caution”, you might want to refactor it to `:::warning[caution]` (yellow).



## Versioned Sidebars

This breaking change will only affect **Docusaurus v2 early adopters** who versioned their docs before `v2.0.0-beta.10` (December 2021).

When creating version `v1.0.0`, the sidebar file contained a prefix `version-v1.0.0/` that **Docusaurus v3** does not support anymore.

```
versioned_sidebars/version-v1.0.0-sidebars.json
```

```
{
 "version-v1.0.0/docs": [
 "version-v1.0.0/introduction",
 "version-v1.0.0/prerequisites"
]
}
```

### ! HOW TO UPGRADE

Remove the useless versioned prefix from your versioned sidebars.

```
versioned_sidebars/version-v1.0.0-sidebars.json
```

```
{
 "docs": ["introduction", "prerequisites"]
}
```

## Blog Feed Limit

The `@docusaurus/plugin-content-blog` now limits the RSS feed to the last 20 entries by default. For large Docusaurus blogs, this is a more sensible default value to avoid an increasingly large RSS file.

### ! HOW TO UPGRADE

In case you don't like this new default behavior, you can revert to the former "unlimited feed" behavior with the new `limit: false` feed option:

```
docusaurus.config.js
```



```
const blogOptions = {
 feedOptions: {
 limit: false,
 },
};
```

## Docs Theme Refactoring

For users that swizzled docs-related theme components (like `@theme/DocPage`), these components have been significantly refactored to make it easier to customize.

Technically, **this is not a breaking change** because these components are **flagged as unsafe to swizzle**, however many Docusaurus sites ejected docs-related components, and will be interested to know their customizations might break Docusaurus.

### ! HOW TO UPGRADE

Delete all your swizzled components, re-swizzle them, and re-apply your customizations on top of the newly updated components.

Alternatively, you can look at the [pull-request notes](#) to understand the new theme component tree structure, and eventually try to patch your swizzled components manually.

## Optional Changes

Some changes are not mandatory, but remain useful to be aware of to plainly leverage Docusaurus v3.

### Automatic JSX runtime

Docusaurus v3 now uses the React 18 "automatic" JSX runtime.

It is not needed anymore to import React in JSX files that do not use any React API.

src/components/MyComponent.js

```
- import React from 'react';

export default function MyComponent() {
 return <div>Hello</div>;
}
```

## ESM and TypeScript Configs



Docusaurus v3 supports ESM and TypeScript config files, and it might be a good idea to adopt those new options.

docusaurus.config.js

```
export default {
 title: 'Docusaurus',
 url: 'https://docusaurus.io',
 // your site config ...
};
```

docusaurus.config.ts

```
import type {Config} from '@docusaurus/types';
import type * as Preset from '@docusaurus/preset-classic';

const config: Config = {
 title: 'My Site',
 favicon: 'img/favicon.ico',
 presets: [
 [
 'classic',
 {
 /* Your preset config here */
 } satisfies Preset.Options,
],
],
 themeConfig: {
 /* Your theme config here */
 } satisfies Preset.ThemeConfig,
};

export default config;
```

## Using the `.mdx` extension

We recommend using the `.mdx` extension whenever you use JSX, `import`, or `export` (i.e. MDX features) inside a Markdown file. It is semantically more correct and improves compatibility with external tools (IDEs, formatters, linters, etc.).

In future versions of Docusaurus, `.md` files will be parsed as standard [CommonMark](#), which does not support these features. In Docusaurus v3, `.md` files keep being compiled as MDX files, but it will be possible to [opt-in for CommonMark](#).



## Upgrading math packages

If you use Docusaurus to render [Math Equations](#), you should upgrade the MDX plugins.

Make sure to use `remark-math 6` and `rehype-katex 7` for Docusaurus v3 (using MDX v3). We can't guarantee other versions will work.

```
{
- "remark-math": "^3.0.0",
+ "remark-math": "^6.0.0",
- "rehype-katex": "^5.0.0"
+ "rehype-katex": "^7.0.0"
}
```

`hast-util-is-element` is now unnecessary in Docusaurus v3. If you have installed it and don't use it somewhere else, you can just remove it by running `npm uninstall hast-util-is-element`.

## Turn off MDX v1 compat

Docusaurus v3 comes with [MDX v1 compatibility options](#), that are turned on by default.

docusaurus.config.js

```
export default {
 markdown: {
 mdx1Compat: {
 comments: true,
 admonitions: true,
 headingIds: true,
 },
 },
};
```

### comments option

This option allows the usage of HTML comments inside MDX, while HTML comments are officially not supported anymore.

For MDX files, we recommend to progressively use MDX `{/* comments */}` instead of HTML `<!-- comments -->`, and then turn this compatibility option off.



**BLOG TRUNCATE MARKER**



The default blog truncate marker now supports both `<!-- truncate -->` and `{/* truncate */}`.

### admonitions option

This option allows the usage of the Docusaurus v2 [admonition title](#) syntax:

```
:::note Your Title

content

:::
```

Docusaurus now implements admonitions with [Markdown Directives](#) (implemented with [remark-directive](#)), and the syntax to provide a directive label requires square brackets:

```
:::note[Your Title]

content

:::
```

We recommend to progressively use the new Markdown directive label syntax, and then turn this compatibility option off.

### headingIds option

This option allows the usage of the Docusaurus v2 [explicit heading id](#) syntax:

```
Hello World {#my-explicit-id}
```

This syntax is now invalid MDX, and would require to escape the `{` character: `\{#my-explicit-id}`.

We recommend to keep this compatibility option on for now, until we provide a new syntax compatible with newer versions of MDX.

## Troubleshooting

In case of any upgrade problem, the first things to try are:

- make sure all your docs compile in the [MDX playground](#), or using `npx docusaurus-mdx-checker`
- delete `node_modules` and `package-lock.json`, and then run `npm install` again





- run `docusaurus clear` to clear the caches
- remove third-party plugins that might not support Docusaurus v3
- delete all your swizzled components

Once you have tried that, you can ask for support through the following support channels:

- [Docusaurus v3 - Upgrade Support](#)
- [Docusaurus v3 - Discord channel #migration-v2-to-v3](#)
- [MDX v3 - Upgrade Support](#)
- [MDX v3 - Remark/Rehype Plugins Support](#)
- [MDX v3 - Discord channel #migration-mdx-v3](#)

Please consider **our time is precious**. To ensure that your support request is not ignored, we kindly ask you to:

- provide a **minimal** reproduction that we can easily run, ideally created with [docusaurus.new](#)
- provide a live deployment url showing the problem in action (if your site can build)
- explain clearly the problem, much more than an ambiguous "it doesn't work"
- include as much relevant material as possible: code snippets, repo url, git branch urls, full stack traces, screenshots and videos
- present your request clearly, concisely, showing us that you have made an effort to help us help you

Alternatively, you can look for a paid [Docusaurus Service Provider](#) to execute this upgrade for you. If your site is open source, you can also ask our community for [free, benevolent help](#).

## 1.7.2 To Docusaurus v2

### 1.7.3 Overview

This doc guides you through migrating an existing Docusaurus 1 site to Docusaurus 2.

We try to make this as easy as possible, and provide a migration CLI.

## Main differences

Docusaurus 1 is a pure documentation site generator, using React as a server-side template engine, but not loading React on the browser.

Docusaurus 2, rebuilt from the ground up, generates a single-page-application, using the full power of React in the browser. It allows for more customizability but preserved the best parts of Docusaurus 1 - easy to get started, versioned docs, and i18n.

Beyond that, Docusaurus 2 is a **performant static site generator** and can be used to create common content-driven websites (e.g. Documentation, Blogs, Product Landing and Marketing Pages, etc) extremely quickly.

While our main focus will still be helping you get your documentations right and well, it is possible to build any kind of website using Docusaurus 2 as it is just a React application. **Docusaurus can now be used to**



**build any website, not just documentation websites.**

## Docusaurus 1 structure

Your Docusaurus 1 site should have the following structure:

```
├── docs
└── website
 ├── blog
 ├── core
 │ └── Footer.js
 ├── package.json
 ├── pages
 ├── sidebars.json
 ├── siteConfig.js
 └── static
```

## Docusaurus 2 structure

After the migration, your Docusaurus 2 site could look like:

```
├── docs
└── website
 ├── blog
 ├── src
 │ ├── components
 │ ├── css
 │ └── pages
 ├── static
 ├── package.json
 ├── sidebars.json
 └── docusaurus.config.js
```

### ! INFO

This migration does not change the `/docs` folder location, but Docusaurus v2 sites generally have the `/docs` folder inside `/website`

You are free to put the `/docs` folder anywhere you want after having migrated to v2.

## Migration process

There are multiple things to migrate to obtain a fully functional Docusaurus 2 website:



- packages
- CLI commands
- site configuration
- Markdown files
- sidebars file
- pages, components and CSS
- versioned docs
- i18n support 🚧

## Automated migration process

The [migration CLI](#) will handle many things of the migration for you.

However, some parts can't easily be automated, and you will have to fallback to the manual process.

### NOTE

We recommend running the migration CLI, and complete the missing parts thanks to the manual migration process.

## Manual migration process

Some parts of the migration can't be automated (particularly the pages), and you will have to migrate them manually.

The [manual migration guide](#) will give you all the manual steps.

## Support

For any questions, you can ask in the [#migration-v1-to-v2](#) [Discord channel](#).

Feel free to tag [@slorber](#) in any migration PRs if you would like us to have a look.

We also have volunteers willing to [help you migrate your v1 site](#).

## Example migration PRs

You might want to refer to our migration PRs for [Create React App](#) and [Flux](#) as examples of how a migration for a basic Docusaurus v1 site can be done.

### 1.7.4 Automated migration

The migration CLI automatically migrates your v1 website to a v2 website.

### INFO



Manual work is still required after using the migration CLI, as we can't automate a full migration

The migration CLI migrates:

- Site configurations (from `siteConfig.js` to `docusaurus.config.js`)
- `package.json`
- `sidebars.json`
- `/docs`
- `/blog`
- `/static`
- `versioned_sidebar.json` and `/versioned_docs` if your site uses versioning

To use the migration CLI, follow these steps:

1. Before using the migration CLI, ensure that `/docs`, `/blog`, `/static`, `sidebars.json`, `siteConfig.js`, `package.json` follow the expected structure.
2. To migrate your v1 website, run the migration CLI with the appropriate filesystem paths:

```
migration command format
npx @docusaurus/migrate migrate <v1 website directory> <desired v2 website directory>

example
npx @docusaurus/migrate migrate ./v1-website ./v2-website
```

3. To view your new website locally, go into your v2 website's directory and start your development server.

**npm**   **Yarn**   **pnpm**   **Bun**

```
cd ./v2-website
npm install
npm start
```

### DANGER

The migration CLI updates existing files. Be sure to have committed them first!

## Options



You can add option flags to the migration CLI to automatically migrate Markdown content and pages to v2. It is likely that you will still need to make some manual changes to achieve your desired result.

| Name                | Description                                            |
|---------------------|--------------------------------------------------------|
| <code>--mdx</code>  | Add this flag to convert Markdown to MDX automatically |
| <code>--page</code> | Add this flag to migrate pages automatically           |

```
example using options
npx @docusaurus/migrate migrate --mdx --page ./v1-website ./v2-website
```

### DANGER

The migration of pages and MDX is still a work in progress.

We recommend you to try to run the pages without these options, commit, and then try to run the migration again with the `--page` and `--mdx` options.

This way, you'd be able to easily inspect and fix the diff.

## 1.7.5 Manual migration

This manual migration process should be run after the [automated migration process](#), to complete the missing parts, or debug issues in the migration CLI output.

## Project setup

### `package.json`

### Scoped package names

In Docusaurus 2, we use scoped package names:

- `docusaurus` → `@docusaurus/core`

This provides a clear distinction between Docusaurus' official packages and community maintained packages. In another words, all Docusaurus' official packages are namespaced under `@docusaurus/`.

Meanwhile, the default doc site functionalities provided by Docusaurus 1 are now provided by `@docusaurus/preset-classic`. Therefore, we need to add this dependency as well:

```
package.json
```



```
{
 dependencies: {
 - "docusaurus": "^1.x.x",
 + "@docusaurus/core": "^2.0.0-beta.0",
 + "@docusaurus/preset-classic": "^2.0.0-beta.0",
 }
}
```

**TIP**

Please use the most recent Docusaurus 2 version, which you can check out [here](#) (using the `latest` tag).

## CLI commands

Meanwhile, CLI commands are renamed to `docusaurus <command>` (instead of `docusaurus-command`).

The `"scripts"` section of your `package.json` should be updated as follows:

package.json

```
{
 "scripts": {
 "start": "docusaurus start",
 "build": "docusaurus build",
 "swizzle": "docusaurus swizzle",
 "deploy": "docusaurus deploy"
 // ...
 }
}
```

A typical Docusaurus 2 `package.json` may look like this:

package.json



```
{
 "scripts": {
 "docusaurus": "docusaurus",
 "start": "docusaurus start",
 "build": "docusaurus build",
 "swizzle": "docusaurus swizzle",
 "deploy": "docusaurus deploy",
 "serve": "docusaurus serve",
 "clear": "docusaurus clear"
 },
 "dependencies": {
 "@docusaurus/core": "^2.0.0-beta.0",
 "@docusaurus/preset-classic": "^2.0.0-beta.0",
 "clsx": "^1.1.1",
 "react": "^17.0.2",
 "react-dom": "^17.0.2"
 },
 "browserslist": {
 "production": [">0.5%", "not dead", "not op_mini all"],
 "development": [
 "last 1 chrome version",
 "last 1 firefox version",
 "last 1 safari version"
]
 }
}
```

## Update references to the `build` directory

In Docusaurus 1, all the build artifacts are located within `website/build/<PROJECT_NAME>`.

In Docusaurus 2, it is now moved to just `website/build`. Make sure that you update your deployment configuration to read the generated files from the correct `build` directory.

If you are deploying to GitHub pages, make sure to run `yarn deploy` instead of `yarn publish-gh-pages` script.

### `.gitignore`

The `.gitignore` in your `website` should contain:

```
.gitignore
```



```
dependencies
/node_modules

production
/build

generated files
.docusaurus
.cache-loader

misc
.DS_Store
.env.local
.env.development.local
.env.test.local
.env.production.local

npm-debug.log*
yarn-debug.log*
yarn-error.log*
```

### README

The D1 website may have an existing README file. You can modify it to reflect the D2 changes, or copy the default [Docusaurus v2 README](#).

## Site configurations

### docusaurus.config.js

Rename `siteConfig.js` to `docusaurus.config.js`.

In Docusaurus 2, we split each functionality (blog, docs, pages) into plugins for modularity. Presets are bundles of plugins and for backward compatibility we built a `@docusaurus/preset-classic` preset which bundles most of the essential plugins present in Docusaurus 1.

Add the following preset configuration to your `docusaurus.config.js`.

```
docusaurus.config.js
```





```
module.exports = {
 // ...
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 docs: {
 // Docs folder path relative to website dir.
 path: '../docs',
 // Sidebars file relative to website dir.
 sidebarPath: require.resolve('./sidebars.json'),
 },
 // ...
 },
],
],
};
```

We recommend moving the `docs` folder into the `website` folder and that is also the default directory structure in v2. [Vercel](#) supports [Docusaurus project deployments out-of-the-box](#) if the `docs` directory is within the `website`. It is also generally better for the docs to be within the website so that the docs and the rest of the website code are co-located within one `website` directory.

If you are migrating your Docusaurus v1 website, and there are pending documentation pull requests, you can temporarily keep the `/docs` folder to its original place, to avoid producing conflicts.

Refer to migration guide below for each field in `siteConfig.js`.

## Updated fields

**`baseUrl`, `tagline`, `title`, `url`, `favicon`, `organizationName`, `projectName`, `githubHost`, `scripts`, `stylesheets`**

No actions needed, these configuration fields were not modified.

### **`colors`**

Deprecated. We wrote a custom CSS framework for Docusaurus 2 called [Infima](#) which uses CSS variables for theming. The docs are not quite ready yet and we will update here when it is. To overwrite Infima's CSS variables, create your own CSS file (e.g. `./src/css/custom.css`) and import it globally by passing it as an option to `@docusaurus/preset-classic`:

```
docusaurus.config.js
```



```
module.exports = {
 // ...
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 theme: {
 customCss: [require.resolve('./src/css/custom.css')],
 },
],
],
],
};
```

Infima uses 7 shades of each color.

/src/css/custom.css

```
/**
 * You can override the default Infima variables here.
 * Note: this is not a complete list of --ifm- variables.
 */
:root {
 --ifm-color-primary: #25c2a0;
 --ifm-color-primary-dark: rgb(33, 175, 144);
 --ifm-color-primary-darker: rgb(31, 165, 136);
 --ifm-color-primary-darkest: rgb(26, 136, 112);
 --ifm-color-primary-light: rgb(70, 203, 174);
 --ifm-color-primary-lighter: rgb(102, 212, 189);
 --ifm-color-primary-lightest: rgb(146, 224, 208);
}
```

We recommend using [ColorBox](#) to find the different shades of colors for your chosen primary color.

Alternatively, use the following tool to generate the different shades for your website and copy the variables into `src/css/custom.css`.

### TIP

Aim for at least [WCAG-AA contrast ratio](#) for the primary color to ensure readability. Use the Docusaurus website itself to preview how your color palette would look like. You can use alternative palettes in dark mode because one color doesn't usually work in both light and dark mode.



Primary Color:

#2e8555



Edit dark mode

Reset

Background:



| CSS Variable Name            | Hex                                                                                       | Adjustment | Contrast Rating                                                                          |
|------------------------------|-------------------------------------------------------------------------------------------|------------|------------------------------------------------------------------------------------------|
| --ifm-color-primary-lightest |  #3cad6e | -30        | Fail  |
| --ifm-color-primary-lighter  |  #359962 | -15        | Fail  |
| --ifm-color-primary-light    |  #33925d | -10        | Fail  |
| --ifm-color-primary          |  #2e8555 | 0          | AA    |
| --ifm-color-primary-dark     |  #29784c | 10         | AA    |
| --ifm-color-primary-darker   |  #277148 | 15         | AA    |
| --ifm-color-primary-darkest  |  #205d3b | 30         | AAA   |

Replace the variables in `src/css/custom.css` with these new variables.

`/src/css/custom.css`

```
:root {
 --ifm-color-primary: #2e8555;
 --ifm-color-primary-dark: #29784c;
 --ifm-color-primary-darker: #277148;
 --ifm-color-primary-darkest: #205d3b;
 --ifm-color-primary-light: #33925d;
 --ifm-color-primary-lighter: #359962;
 --ifm-color-primary-lightest: #3cad6e;
}
```

**footerIcon**, **copyright**, **ogImage**, **twitterImage**,  
**docsSideNavCollapsible**

Site meta info such as assets, SEO, copyright info are now handled by themes. To customize them, use the `themeConfig` field in your `docusaurus.config.js`:

`docusaurus.config.js`



```
module.exports = {
 // ...
 themeConfig: {
 footer: {
 logo: {
 alt: 'Meta Open Source Logo',
 src: '/img/meta_oss_logo.png',
 href: 'https://opensource.facebook.com/',
 },
 copyright: `Copyright © ${new Date().getFullYear()} Facebook, Inc.`,
 // You can also put own HTML here.
 },
 image: 'img/docusaurus.png',
 // ...
 },
};
```

## headerIcon, headerLinks

In Docusaurus 1, header icon and header links were root fields in `siteConfig`:

siteConfig.js

```
headerIcon: 'img/docusaurus.svg',
headerLinks: [
 { doc: "doc1", label: "Getting Started" },
 { page: "help", label: "Help" },
 { href: "https://github.com/", label: "GitHub" },
 { blog: true, label: "Blog" },
],
```

Now, these two fields are both handled by the theme:

docusaurus.config.js



```
module.exports = {
 // ...
 themeConfig: {
 navbar: {
 title: 'Docusaurus',
 logo: {
 alt: 'Docusaurus Logo',
 src: 'img/docusaurus.svg',
 },
 items: [
 {to: 'docs/doc1', label: 'Getting Started', position: 'left'},
 {to: 'help', label: 'Help', position: 'left'},
 {
 href: 'https://github.com/',
 label: 'GitHub',
 position: 'right',
 },
 {to: 'blog', label: 'Blog', position: 'left'},
],
 },
 // ...
 },
};
```

## algolia

docusaurus.config.js

```
module.exports = {
 // ...
 themeConfig: {
 algolia: {
 apiKey: '47ecd3b21be71c5822571b9f59e52544',
 indexName: 'docusaurus-2',
 algoliaOptions: { //... },
 },
 // ...
 },
};
```

### WARNING

Your Algolia DocSearch v1 config (found [here](#)) should be updated for Docusaurus v2 ([example](#)).



You can contact the DocSearch team (@shortcuts, @s-pace) for support. They can update it for you and trigger a recrawl of your site to restore the search (otherwise you will have to wait up to 24h for the next scheduled crawl)

## blogSidebarCount

Deprecated. Pass it as a blog option to `@docusaurus/preset-classic` instead:

docusaurus.config.js

```
module.exports = {
 // ...
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 blog: {
 postsPerPage: 10,
 },
 // ...
 },
],
],
};
```

## cname

Deprecated. Create a `CNAME` file in your `static` folder instead with your custom domain. Files in the `static` folder will be copied into the root of the `build` folder during execution of the build command.

## customDocsPath, docsUrl, editUrl, enableUpdateBy, enableUpdateTime

**BREAKING:** `editUrl` should point to (website) Docusaurus project instead of `docs` directory.

Deprecated. Pass it as an option to `@docusaurus/preset-classic` docs instead:

docusaurus.config.js



```
module.exports = {
 // ...
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 docs: {
 // Equivalent to `customDocsPath`.
 path: 'docs',
 // Equivalent to `editUrl` but should point to `website` dir
 // instead of `website/docs`.
 editUrl:
 'https://github.com/facebook/docusaurus/edit/main/website',
 // Equivalent to `docsUrl`.
 routeBasePath: 'docs',
 // Remark and Rehype plugins passed to MDX. Replaces
 // `markdownOptions` and `markdownPlugins`.
 remarkPlugins: [],
 rehypePlugins: [],
 // Equivalent to `enableUpdateBy`.
 showLastUpdateAuthor: true,
 // Equivalent to `enableUpdateTime`.
 showLastUpdateTime: true,
 },
 // ...
 },
],
],
};
```

## gaTrackingId

docusaurus.config.js



```
module.exports = {
 // ...
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 // ...
 googleAnalytics: {
 trackingID: 'UA-141789564-1',
 },
 },
],
],
};
```

## gaGtag

docusaurus.config.js

```
module.exports = {
 // ...
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 // ...
 gtag: {
 trackingID: 'UA-141789564-1',
 },
 },
],
],
};
```

## Removed fields

The following fields are all deprecated, you may remove from your configuration file.

- `blogSidebarTitle`
- `cleanUrl` - Clean URL is used by default now.
- `defaultVersionShown` - Versioning is not ported yet. You'd be unable to migration to Docusaurus 2 if you are using versioning. Stay tuned.
- `disableHeaderTitle`
- `disableTitleTagline`





- `docsSideNavCollapsible` is available at `docsPluginOptions.sidebarCollapsible`, and this is turned on by default now.
- `facebookAppId`
- `facebookComments`
- `facebookPixelId`
- `fonts`
- `highlight` - We now use [Prism](#) instead of [highlight.js](#).
- `markdownOptions` - We use MDX in v2 instead of Remarkable. Your Markdown options have to be converted to Remark/Rehype plugins.
- `markdownPlugins` - We use MDX in v2 instead of Remarkable. Your Markdown plugins have to be converted to Remark/Rehype plugins.
- `manifest`
- `onPageNav` - This is turned on by default now.
- `separateCss` - It can imported in the same manner as `custom.css` mentioned above.
- `scrollToTop`
- `scrollToTopOptions`
- `translationRecruitingLink`
- `twitter`
- `twitterUsername`
- `useEnglishUrl`
- `users`
- `usePrism` - We now use [Prism](#) instead of [highlight.js](#)
- `wrapPagesHTML`

We intend to implement many of the deprecated config fields as plugins in future. Help will be appreciated!

## Urls

In v1, all pages were available with or without the `.html` extension.

For example, these 2 pages exist:

- <https://v1.docusaurus.io/docs/en/installation>
- <https://v1.docusaurus.io/docs/en/installation.html>

If `cleanUrl` was:

- `true`: links would target `/installation`
- `false`: links would target `/installation.html`

In v2, by default, the canonical page is `/installation`, and not `/installation.html`.

If you had `cleanUrl: false` in v1, it's possible that people published links to `/installation.html`.



For SEO reasons, and avoiding breaking links, you should configure server-side redirect rules on your hosting provider.

As an escape hatch, you could use [@docusaurus/plugin-client-redirects](#) to create client-side redirects from `/installation.html` to `/installation`.

```
module.exports = {
 plugins: [
 [
 '@docusaurus/plugin-client-redirects',
 {
 fromExtensions: ['html'],
 },
],
],
};
```

If you want to keep the `.html` extension as the canonical URL of a page, docs can declare a `slug: installation.html` front matter.

## Components

### Sidebar

In previous version, nested sidebar category is not allowed and sidebar category can only contain doc ID. However, v2 allows infinite nested sidebar and we have many types of [Sidebar Item](#) other than document.

You'll have to migrate your sidebar if it contains category type. Rename `subcategory` to `category` and `ids` to `items`.

sidebars.json

```
{
 - type: 'subcategory',
 + type: 'category',
 label: 'My Example Subcategory',
 + items: ['doc1'],
 - ids: ['doc1']
},
```

### Footer

`website/core/Footer.js` is no longer needed. If you want to modify the default footer provided by Docusaurus, [swizzle](#) it:

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)

```
npm run swizzle @docusaurus/theme-classic Footer
```

This will copy the current `<Footer />` component used by the theme to a `src/theme/Footer` directory under the root of your site, you may then edit this component for customization.

Do not swizzle the Footer just to add the logo on the left. The logo is intentionally removed in v2 and moved to the bottom. Just configure the footer in `docusaurus.config.js` with `themeConfig.footer`:

```
module.exports = {
 themeConfig: {
 footer: {
 logo: {
 alt: 'Meta Open Source Logo',
 src: '/img/meta_oss_logo.png',
 href: 'https://opensource.facebook.com',
 },
 },
 },
};
```

## Pages

Please refer to [creating pages](#) to learn how Docusaurus 2 pages work. After reading that, notice that you have to move `pages/en` files in v1 to `src/pages` instead.

In Docusaurus v1, pages received the `siteConfig` object as props.

In Docusaurus v2, get the `siteConfig` object from `useDocusaurusContext` instead.

In v2, you have to apply the theme layout around each page. The Layout component takes metadata props.

`CompLibrary` is deprecated in v2, so you have to write your own React component or use Infima styles (Docs will be available soon, sorry about that! In the meanwhile, inspect the V2 website or view <https://infima.dev/> to see what styles are available).

You can migrate CommonJS to ES6 imports/exports.

Here's a typical Docusaurus v2 page:



```
import React from 'react';
import Link from '@docusaurus/Link';
import useDocusaurusContext from '@docusaurus/useDocusaurusContext';
import Layout from '@theme/Layout';

const MyPage = () => {
 const {siteConfig} = useDocusaurusContext();
 return (
 <Layout title={siteConfig.title} description={siteConfig.tagline}>
 <div className="hero text--center">
 <div className="container">
 <div className="padding-vert--md">
 <h1 className="hero__title">{siteConfig.title}</h1>
 <p className="hero__subtitle">{siteConfig.tagline}</p>
 </div>
 <div>
 <Link
 to="/docs/get-started"
 className="button button--lg button--outline button--
primary">
 Get started
 </Link>
 </div>
 </div>
 </div>
 </Layout>
);
};

export default MyPage;
```

The following code could be helpful for migration of various pages:

- Index page - [Flux](#) (recommended), [Docusaurus 2](#), [Hermes](#)
- Help/Support page - [Docusaurus 2](#), [Flux](#)

## Content

### Replace AUTOGENERATED\_TABLE\_OF\_CONTENTS

This feature is replaced by [inline table of content](#)

### Update Markdown syntax to be MDX-compatible

In Docusaurus 2, the Markdown syntax has been changed to [MDX](#). Hence there might be some broken syntax in the existing docs which you would have to update. A common example is self-closing tags like `<img>` and `<br>` which are valid in HTML would have to be explicitly closed now ( `<img/>` and `<br/>` ). All tags in MDX documents have to be valid JSX.



Front matter is parsed by [gray-matter](#). If your front matter use special characters like `:`, you now need to quote it: `title: Part 1: my part1 title` → `title: "Part 1: my part1 title"`.

**Tips:** You might want to use some online tools like [HTML to JSX](#) to make the migration easier.

## Language-specific code tabs

Refer to the [multi-language support code blocks](#) section.

## Front matter

The Docusaurus front matter fields for the blog have been changed from camelCase to snake\_case to be consistent with the docs.

The fields `authorFBID` and `authorTwitter` have been deprecated. They are only used for generating the profile image of the author which can be done via the `authors` field.

## Deployment

The `CNAME` file used by GitHub Pages is not generated anymore, so be sure you have created it in `/static/CNAME` if you use a custom domain.

The blog RSS feed is now hosted at `/blog/rss.xml` instead of `/blog/feed.xml`. You may want to configure server-side redirects so that users' subscriptions keep working.

## Test your site

After migration, your folder structure should look like this:

```
my-project
├── docs
├── website
│ ├── blog
│ ├── src
│ │ ├── css
│ │ │ └── custom.css
│ │ └── pages
│ │ └── index.js
│ ├── package.json
│ ├── sidebars.json
│ ├── .gitignore
│ ├── docusaurus.config.js
│ └── static
```

Start the development server and fix any errors:

**npm**

**Yarn**

**pnpm**

**Bun**



```
cd website
npm start
```

You can also try to build the site for production:

**npm**   Yarn   pnpm   Bun

```
npm run build
```

## 1.7.6 Versioned sites

Read up <https://docusaurus.io/blog/2018/09/11/Towards-Docusaurus-2#versioning> first for problems in v1's approach.

### NOTE

The versioned docs should normally be migrated correctly by the [migration CLI](#)

## Migrate your `versioned_docs` front matter

Unlike v1, The Markdown header for each versioned doc is no longer altered by using `version-${version}-${original_id}` as the value for the actual ID field. See scenario below for better explanation.

For example, if you have a `docs/hello.md`.

```

id: hello
title: Hello, World !

Hi, Endilie here :)
```

When you cut a new version 1.0.0, in Docusaurus v1, `website/versioned_docs/version-1.0.0/hello.md` looks like this:



```

id: version-1.0.0-hello
title: Hello, World !
original_id: hello

```

Hi, Endilie here :)

In comparison, Docusaurus 2 `website/versioned_docs/version-1.0.0/hello.md` looks like this (exactly same as original)

```

id: hello
title: Hello, World !

```

Hi, Endilie here :)

Since we're going for snapshot and allow people to move (and edit) docs easily inside version. The `id` front matter is no longer altered and will remain the same. Internally, it is set as `version-${version}/${id}`.

Essentially, here are the necessary changes in each `versioned_docs` file:

```

- id: version-1.0.0-hello
+ id: hello
title: Hello, World !
- original_id: hello

```

Hi, Endilie here :)

## Migrate your `versioned_sidebars`

- Refer to `versioned_docs` ID as `version-${version}/${id}` (v2) instead of `version-${version}-${original_id}` (v1).

Because in v1 there is a good chance someone created a new file with front matter ID `"version-${version}-${id}"` that can conflict with `versioned_docs` ID.

For example, Docusaurus 1 can't differentiate `docs/xxx.md`



```

id: version-1.0.0-hello

Another content
```

vs `website/versioned_docs/version-1.0.0/hello.md`

```

id: version-1.0.0-hello
title: Hello, World !
original_id: hello

Hi, Endilie here :)
```

Since we don't allow `/` in v1 & v2 for front matter, conflicts are less likely to occur.

So v1 users need to migrate their `versioned_sidebars` file

Example `versioned_sidebars/version-1.0.0-sidebars.json`:

`versioned_sidebars/version-1.0.0-sidebars.json`

```
{
+ "version-1.0.0/docs": {
- "version-1.0.0-docs": {
 "Test": [
+ "version-1.0.0/foo/bar",
- "version-1.0.0-foo/bar",
],
 "Guides": [
+ "version-1.0.0/hello",
- "version-1.0.0-hello"
]
}
}
```

## Populate your `versioned_sidebars` and `versioned_docs`





In v2, we use snapshot approach for documentation versioning. **Every versioned docs does not depends on other version.** It is possible to have `foo.md` in `version-1.0.0` but it doesn't exist in `version-1.2.0`. This is not possible in previous version due to Docusaurus v1 fallback functionality (<https://v1.docusaurus.io/docs/en/versioning#fallback-functionality>).

For example, if your `versions.json` looks like this in v1

`versions.json`

```
["1.1.0", "1.0.0"]
```

Docusaurus v1 creates versioned docs **if and only if the doc content is different**. Your docs structure might look like this if the only doc changed from v1.0.0 to v1.1.0 is `hello.md`.

```
website
├── versioned_docs
│ ├── version-1.1.0
│ │ └── hello.md
│ └── version-1.0.0
│ ├── foo
│ │ └── bar.md
│ └── hello.md
├── versioned_sidebars
└── version-1.0.0-sidebars.json
```

In v2, you have to populate the missing `versioned_docs` and `versioned_sidebars` (with the right front matter and ID reference too).

```
website
├── versioned_docs
│ ├── version-1.1.0
│ │ ├── foo
│ │ │ └── bar.md
│ │ └── hello.md
│ └── version-1.0.0
│ ├── foo
│ │ └── bar.md
│ └── hello.md
├── versioned_sidebars
│ ├── version-1.1.0-sidebars.json
│ └── version-1.0.0-sidebars.json
```



# Convert style attributes to style objects in MDX

Docusaurus 2 uses JSX for doc files. If you have any style attributes in your Docusaurus 1 docs, convert them to style objects, like this:

```

id: demo
title: Demo

Section

hello world

- <pre style="background: black">zzz</pre>
+ <pre style={{background: 'black'}}>zzz</pre>
```

## 1.7.7 Translated sites

This page explains how migrate a translated Docusaurus v1 site to Docusaurus v2.

## i18n differences

Docusaurus v2 i18n is conceptually quite similar to Docusaurus v1 i18n with a few differences.

It is not tightly coupled to Crowdin, and you can use Git or another SaaS instead.

### Different filesystem paths

On Docusaurus v2, localized content is generally found at `website/i18n/[locale]`.

Docusaurus v2 is modular based on a plugin system, and each plugin is responsible to manage its own translations.

Each plugin has its own i18n subfolder, like: `website/i18n/fr/docusaurus-plugin-content-blog`

### Updated translation APIs

With Docusaurus v1, you translate your pages with `<translate>`:



```
const translate = require('../server/translate.js').translate;

<h2>
 <translate desc="the header description">
 This header will be translated
 </translate>
</h2>;
```

On Docusaurus v2, you translate your pages with `<Translate>`

```
import Translate from '@docusaurus/Translate';

<h2>
 <Translate id="header.translation.id" description="the header
description">
 This header will be translated
 </Translate>
</h2>;
```

### NOTE

The `write-translations` CLI still works to extract translations from your code.

The code translations are now added to `i18n/[locale]/code.json` using Chrome i18n JSON format.

## Stricter Markdown parser

Docusaurus v2 is using [MDX](#) to parse Markdown files.

MDX compiles Markdown files to React components, is stricter than the Docusaurus v1 parser, and will make your build fail on error instead of rendering some bad content.

Also, the HTML elements must be replaced by JSX elements.

This is particularly important for i18n because if your translations are not good on Crowdin and use invalid Markup, your v2 translated site might fail to build: you may need to do some translation cleanup to fix the errors.

## Migration strategies

This section will help you figure out how to **keep your existing v1 translations after you migrate to v2**.



There are **multiple possible strategies** to migrate a Docusaurus v1 site using Crowdin, with different tradeoffs.

### WARNING

This documentation is a best-effort to help you migrate, please help us improve it if you find a better way!

Before all, we recommend to:

- Migrate your v1 Docusaurus site to v2 without the translations
- Get familiar with the [new i18n system of Docusaurus v2](#) and
- Make Crowdin work for your v2 site, using a new and untranslated Crowdin project and the [Crowdin tutorial](#)

### DANGER

Don't try to migrate without understanding both Crowdin and Docusaurus v2 i18n.

## Create a new Crowdin project

To avoid any **risk of breaking your v1 site in production**, one possible strategy is to duplicate the original v1 Crowdin project.

### INFO

This strategy was used to [upgrade the Jest website](#).

Unfortunately, Crowdin does not have any "Duplicate/clone Project" feature, which makes things complicated.

- Download the translation memory of your original project in `.tmx` format ([https://crowdin.com/project/<ORIGINAL\\_PROJECT>/settings#tm](https://crowdin.com/project/<ORIGINAL_PROJECT>/settings#tm) > View Records)
- Upload the translation memory to your new project ([https://crowdin.com/project/<NEW\\_PROJECT>/settings#tm](https://crowdin.com/project/<NEW_PROJECT>/settings#tm) > View Records)
- Reconfigure `crowdin.yml` for Docusaurus v2 according to the i18n docs
- Upload the Docusaurus v2 source files with the Crowdin CLI to the new project
- Mark sensitive strings like `id` or `slug` as "hidden string" on Crowdin
- On the "Translations" tab, click on "Pre-Translation > via TM" ([https://crowdin.com/project/<NEW\\_PROJECT>/settings#translations](https://crowdin.com/project/<NEW_PROJECT>/settings#translations))
- Try first with "100% match" (more content will be translated than "Perfect"), and pre-translate your sources
- Download the Crowdin translations locally
- Try to run/build your site and see if there are any errors



You will likely have errors on your first-try: the pre-translation might try to translate things that it should not be translated (front matter, admonition, code blocks...), and the translated MD files might be invalid for the MDX parser.

You will have to fix all the errors until your site builds. You can do that by modifying the translated MD files locally, and fix your site for one locale at a time using `docusaurus build --locale fr`.

There is no ultimate guide we could write to fix these errors, but common errors are due to:

- Not marking enough strings as "hidden strings" in Crowdin, leading to pre-translation trying to translate these strings.
- Having bad v1 translations, leading to invalid markup in v2: bad HTML elements inside translations and unclosed tags
- Anything rejected by the MDX parser, like using HTML elements instead of JSX elements (use the [MDX playground](#) for debugging)

You might want to repeat this pre-translation process, eventually trying the "Perfect" option and limiting pre-translation only some languages/files.

**TIP**

Use `mdx-code-block` around problematic Markdown elements: Crowdin is less likely mess things up with code blocks.

**NOTE**

You will likely notice that some things were translated on your old project, but are now untranslated in your new project.

The Crowdin Markdown parser is evolving other time and each Crowdin project has a different parser version, which can lead to pre-translation not being able to pre-translate all the strings.

This parser version is undocumented, and you will have to ask the Crowdin support to know your project's parser version and fix one specific version.

Using the same CLI version and parser version across the 2 Crowdin projects might give better results.

**DANGER**

Crowdin has an "upload translations" feature, but in our experience it does not give very good results for Markdown

## Use the existing Crowdin project

If you don't mind modifying your existing Crowdin project and risking to mess things up, it may be possible to use the Crowdin branch system.

 **WARNING**

This workflow has not been tested in practice, please report us how good it is.

This way, you wouldn't need to create a new Crowdin project, transfer the translation memory, apply pre-translations, and try to fix the pre-translations errors.

You could create a Crowdin branch for Docusaurus v2, where you upload the v2 sources, and merge the Crowdin branch to main once ready.

## Use Git instead of Crowdin

It is possible to migrate away of Crowdin, and add the translation files to Git instead.

Use the Crowdin CLI to download the v1 translated files, and put these translated files at the correct Docusaurus v2 filesystem location.



## 2 API

### 2.1 CLI

Docusaurus provides a set of scripts to help you generate, serve, and deploy your website.

Once your website is bootstrapped, the website source will contain the Docusaurus scripts that you can invoke with your package manager:

package.json

```
{
 // ...
 "scripts": {
 "docusaurus": "docusaurus",
 "start": "docusaurus start",
 "build": "docusaurus build",
 "swizzle": "docusaurus swizzle",
 "deploy": "docusaurus deploy",
 "clear": "docusaurus clear",
 "serve": "docusaurus serve",
 "write-translations": "docusaurus write-translations",
 "write-heading-ids": "docusaurus write-heading-ids"
 }
}
```

#### 2.1.1 Docusaurus CLI commands

Below is a list of Docusaurus CLI commands and their usages:

**docusaurus start [siteDir]**

Builds and serves a preview of your site locally with [Webpack Dev Server](#).

#### Options

Name	Default	Description
<code>--port</code>	<code>3000</code>	Specifies the port of the dev server.
<code>--host</code>	<code>localhost</code>	Specify a host to use. For example, if you want your server to be accessible externally, you can use <code>--host 0.0.0.0</code> .
<code>--locale</code>		Specify site locale to be used.
<code>--hot-only</code>	<code>false</code>	Enables Hot Module Replacement without page refresh as a fallback in case of build failures. More information <a href="#">here</a> .
<code>--no-open</code>	<code>false</code>	Do not open the page automatically in the browser.
<code>--config</code>	<code>undefined</code>	Path to Docusaurus config file, default to <code>[siteDir]/docusaurus.config.js</code>



Name	Default	Description
<code>--poll</code> <code>[optionalIntervalMs]</code>	false	Use polling of files rather than watching for live reload as a fallback in environments where watching doesn't work. More information <a href="#">here</a> .
<code>--no-minify</code>	false	Build website without minimizing JS/CSS bundles.
<code>--https</code>	false	Serve the dev site over HTTPS using a self-signed certificate. Implied when both <code>--ssl-cert</code> and <code>--ssl-key</code> are provided.
<code>--ssl-cert</code>	undefined	Path to a TLS certificate file (implies HTTPS).
<code>--ssl-key</code>	undefined	Path to a TLS private key file (implies HTTPS).

### ! INFO

Please note that some functionality (for example, anchor links) will not work in development. The functionality will work as expected in production.

### ! DEVELOPMENT OVER NETWORK

When forwarding port 3000 from a remote server or VM (e.g. GitHub Codespaces), you can run the dev server on `0.0.0.0` to make it listen on the local IP.

**npm**   **Yarn**   **pnpm**   **Bun**

```
npm run start -- --host 0.0.0.0
```

## Enabling HTTPS

There are multiple ways to obtain a certificate. We will use [mkcert](#) as an example.

1. Run `mkcert localhost` to generate `localhost.pem` + `localhost-key.pem`
2. Run `mkcert -install` to install the cert in your trust store, and restart your browser
3. Start the app with the Docusaurus HTTPS CLI options:

**npm**   **Yarn**   **pnpm**   **Bun**

```
npm run start -- --ssl-cert localhost.pem --ssl-key localhost-key.pem
```





4. Open `https://localhost:3000/`



### SELF-SIGNED CERTIFICATE

Use the `--https` option to serve the dev site over HTTPS with an automatically generated self-signed certificate:

**npm**

**Yarn**

**pnpm**

**Bun**

```
npm run start -- --https
```

## **docusaurus build [siteDir]**

Compiles your site for production.

### Options

Name	Default	Description
<code>--dev</code>		Builds the website in dev mode, including full React error messages.
<code>--bundle-analyzer</code>	<code>false</code>	Analyze your bundle with the <a href="#">webpack bundle analyzer</a> .
<code>--out-dir</code>	<code>build</code>	The full path for the new output directory, relative to the current workspace.
<code>--config</code>	<code>undefined</code>	Path to Docusaurus config file, default to <code>[siteDir]/docusaurus.config.js</code>
<code>--locale</code>		Build the site in the specified locale(s). If not specified, all known locales are built.
<code>--no-minify</code>	<code>false</code>	Build website without minimizing JS/CSS bundles.



### INFO

For advanced minification of CSS bundle, we use the [advanced cssnano preset](#) (along with additional several PostCSS plugins) and [level 2 optimization of clean-css](#). If as a result of this advanced CSS minification you find broken CSS, build your website with the environment variable `USE_SIMPLE_CSS_MINIFIER=true` to minify CSS with the [default cssnano preset](#). **Please fill out an issue if you experience CSS minification bugs.**

You can skip the HTML minification with the environment variable

```
SKIP_HTML_MINIFICATION=true.
```

## **docusaurus swizzle [themeName] [componentName] [siteDir]**

[Swizzle](#) a theme component to customize it.

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)

```
npm run swizzle [themeName] [componentName] [siteDir]

Example (leaving out the siteDir to indicate this directory)
npm run swizzle @docusaurus/theme-classic Footer -- --eject
```

The swizzle CLI is interactive and will guide you through the whole [swizzle process](#).

## Options

Name	Description
<code>themeName</code>	The name of the theme to swizzle from.
<code>componentName</code>	The name of the theme component to swizzle.
<code>--list</code>	Display components available for swizzling
<code>--eject</code>	<a href="#">Eject</a> the theme component
<code>--wrap</code>	<a href="#">Wrap</a> the theme component
<code>--danger</code>	Allow immediate swizzling of unsafe components
<code>--typescript</code>	Swizzle the TypeScript variant component
<code>--javascript</code>	Swizzle the JavaScript variant component
<code>--config</code>	Path to docusaurus config file, default to <code>[siteDir]/docusaurus.config.js</code>

### WARNING

Unsafe components have a higher risk of breaking changes due to internal refactorings.

## **docusaurus deploy [siteDir]**

Deploys your site with [GitHub Pages](#). Check out the docs on [deployment](#) for more details.

## Options

Name	Default	Description
<code>--locale</code>		Deploy the site in the specified locale(s). If not specified, all known locales are deployed.
<code>--out-dir</code>	<code>build</code>	The full path for the new output directory, relative to the current workspace.
<code>--skip-build</code>	<code>false</code>	Deploy website without building it. This may be useful when using a custom deploy script.
<code>--target-dir</code>	<code>.</code>	Path to the target directory to deploy to.
<code>--config</code>	<code>undefined</code>	Path to Docusaurus config file, default to <code>[siteDir]/docusaurus.config.js</code>



## docusaurus serve [siteDir]

Serve your built website locally.

Name	Default	Description
<code>--port</code>	<code>3000</code>	Use specified port
<code>--dir</code>	<code>build</code>	The full path for the output directory, relative to the current workspace
<code>--build</code>	<code>false</code>	Build website before serving
<code>--config</code>	<code>undefined</code>	Path to Docusaurus config file. default to <code>[siteDir]/docusaurus.config.js</code>
<code>--host</code>	<code>localhost</code>	Specify a host to use. For example, if you want your server to be accessible externally, you can use <code>--host 0.0.0.0</code> .
<code>--no-open</code>	<code>false</code> locally, <code>true</code> in CI	Do not open a browser window to the server location.

## docusaurus clear [siteDir]

Clear a Docusaurus site's generated assets, caches, build artifacts.

We recommend running this command before reporting bugs, after upgrading versions, or anytime you have issues with your Docusaurus site.

## docusaurus write-translations [siteDir]

Write the JSON translation files that you will have to translate.

By default, the files are written in `website/i18n/<defaultLocale>/...`.

Name	Default	Description
<code>--locale</code>	<code>&lt;defaultLocale&gt;</code>	Define which locale folder you want to write translations the JSON files in
<code>--override</code>	<code>false</code>	Override existing translation messages
<code>--config</code>	<code>undefined</code>	Path to Docusaurus config file. default to <code>[siteDir]/docusaurus.config.js</code>
<code>--messagePrefix</code>	<code>''</code>	Allows adding a prefix to each translation message, to help you highlight untranslated strings

## docusaurus write-heading-ids [siteDir] [files]

Add [explicit heading IDs](#) to the Markdown documents of your site.

Name	Default	Description
<code>files</code>	All MD files used by plugins	The files that you want heading IDs to be written to.
<code>--syntax</code>	<code>classic</code>	Heading ID syntax to use: <code>classic</code> ( <code>{#id}</code> ) or <code>mdx-comment</code> ( <code>{/* #id */}</code> ).
<code>--migrate</code>	<code>false</code>	Migrate existing heading IDs to the target <code>--syntax</code> , if they are using a different syntax.
<code>--overwrite</code>	<code>false</code>	Overwrite existing heading IDs, re-generate them from the heading text.



Name	Default	Description
<code>--maintain-case</code>	<code>false</code>	Keep the headings' casing, otherwise make all lowercase.

## 2.2 Docusaurus Client API

Docusaurus provides some APIs on the clients that can be helpful to you when building your site.

### 2.2.1 Components

#### `<ErrorBoundary />`

This component creates a [React error boundary](#).

Use it to wrap components that might throw, and display a fallback when that happens instead of crashing the whole app.

```
import React from 'react';
import ErrorBoundary from '@docusaurus/ErrorBoundary';

const SafeComponent = () => (
 <ErrorBoundary
 fallback={({error, tryAgain}) => (
 <div>
 <p>This component crashed because of error: {error.message}</p>
 <button onClick={tryAgain}>Try Again!</button>
 </div>
)}>
 <SomeDangerousComponentThatMayThrow />
 </ErrorBoundary>
);
```

#### TIP

To see it in action, click here:

**Boom!**

#### INFO

Docusaurus uses this component to catch errors within the theme's layout, and also within the entire app.

#### NOTE



This component doesn't catch build-time errors and only protects against client-side render errors that can happen when using stateful React components.

## Props

- `fallback`: an optional render callback returning a JSX element. It will receive an object with 2 attributes: `error`, the error that was caught, and `tryAgain`, a function `() => void` callback to reset the error in the component and try rendering it again. If not present, `@theme/Error` will be rendered instead. `@theme/Error` is used for the error boundaries wrapping the site, above the layout.

### ⚠ WARNING

The `fallback` prop is a callback, and **not a React functional component**. You can't use React hooks inside this callback.

## <Head/>

This reusable React component will manage all of your changes to the document head. It takes plain HTML tags and outputs plain HTML tags and is beginner-friendly. It is a wrapper around [React Helmet](#).

Usage Example:

```
import React from 'react';
import Head from '@docusaurus/Head';

const MySEO = () => (
 <Head>
 <meta property="og:description" content="My custom description" />
 <meta charset="utf-8" />
 <title>My Title</title>
 <link rel="canonical" href="http://mysite.com/example" />
 </Head>
);
```

Nested or latter components will override duplicate usages:



```
<Parent>
 <Head>
 <title>My Title</title>
 <meta name="description" content="Helmet application" />
 </Head>
 <Child>
 <Head>
 <title>Nested Title</title>
 <meta name="description" content="Nested component" />
 </Head>
 </Child>
</Parent>
```

Outputs:

```
<head>
 <title>Nested Title</title>
 <meta name="description" content="Nested component" />
</head>
```

### <Link/>

This component enables linking to internal pages as well as a powerful performance feature called preloading. Preloading is used to prefetch resources so that the resources are fetched by the time the user navigates with this component. We use an `IntersectionObserver` to fetch a low-priority request when the `<Link>` is in the viewport and then use an `onMouseOver` event to trigger a high-priority request when it is likely that a user will navigate to the requested resource.

The component is a wrapper around react-router's `<Link>` component that adds useful enhancements specific to Docusaurus. All props are passed through to react-router's `<Link>` component.

External links also work, and automatically have these props: `target="_blank" rel="noopener noreferrer"`.



```
import React from 'react';
import Link from '@docusaurus/Link';

const Page = () => (
 <div>
 <p>
 Check out my <Link to="/blog">blog</Link>!
 </p>
 <p>
 Follow me on <Link to="https://x.com/docusaurus">X</Link>!
 </p>
 </div>
);
```

### to: string

The target location to navigate to. Example: `/docs/introduction`.

```
<Link to="/courses" />
```



#### TIP

Prefer this component to vanilla `<a>` tags because Docusaurus does a lot of optimizations (e.g. broken path detection, prefetching, applying base URL...) if you use `<Link>`.

### <Redirect/>

Rendering a `<Redirect>` will navigate to a new location. The new location will override the current location in the history stack like server-side redirects (HTTP 3xx) do. You can refer to [React Router's Redirect documentation](#) for more info on available props.

Example usage:

```
import React from 'react';
import {Redirect} from '@docusaurus/router';

const Home = () => {
 return <Redirect to="/docs/test" />;
};
```

**i NOTE**

`@docusaurus/router` implements [React Router](#) and supports its features.

**<BrowserOnly/>**

The `<BrowserOnly>` component permits to render React components only in the browser after the React app has hydrated.

**💡 TIP**

Use it for integrating with code that can't run in Node.js, because the `window` or `document` objects are being accessed.

**Props**

- `children`: render function prop returning browser-only JSX. Will not be executed in Node.js
- `fallback` (optional): JSX to render on the server (Node.js) and until React hydration completes.

**Example with code**

```
import BrowserOnly from '@docusaurus/BrowserOnly';

const MyComponent = () => {
 return (
 <BrowserOnly>
 {() => page url = {window.location.href}}
 </BrowserOnly>
);
};
```

**Example with a library**

```
import BrowserOnly from '@docusaurus/BrowserOnly';

const MyComponent = (props) => {
 return (
 <BrowserOnly fallback={<div>Loading...</div>}>
 {() => {
 const LibComponent = require('some-lib').LibComponent;
 return <LibComponent {...props} />;
 }}
 </BrowserOnly>
);
};
```





## <Interpolate/>

A simple interpolation component for text containing dynamic placeholders.

The placeholders will be replaced with the provided dynamic values and JSX elements of your choice (strings, links, styled elements...).

### Props

- `children`: text containing interpolation placeholders like `{placeholderName}`
- `values`: object containing interpolation placeholder values

```
import React from 'react';
import Link from '@docusaurus/Link';
import Interpolate from '@docusaurus/Interpolate';

export default function VisitMyWebsiteMessage() {
 return (
 <Interpolate
 values={{
 firstName: 'Sébastien',
 website: (
 <Link to="https://docusaurus.io" className="my-website-class">
 website
 </Link>
),
 }}>
 {'Hello, {firstName}! How are you? Take a look at my {website}'}
 </Interpolate>
);
}
```

## <Translate/>

When [localizing your site](#), the `<Translate/>` component will allow providing **translation support to React components**, such as your homepage. The `<Translate>` component supports [interpolation](#).

The translation strings will statically extracted from your code with the [docusaurus write-translations](#) CLI and a `code.json` translation file will be created in `website/i18n/[locale]`.

### NOTE

The `<Translate/>` props **must be hardcoded strings**.

Apart from the `values` prop used for interpolation, it is **not possible to use variables**, or the static extraction wouldn't work.



## Props

- `children`: untranslated string in the default site locale (can contain [interpolation placeholders](#))
- `id`: optional value to be used as the key in JSON translation files
- `description`: optional text to help the translator
- `values`: optional object containing interpolation placeholder values

## Example

src/pages/index.js

```
import React from 'react';
import Layout from '@theme/Layout';

import Translate from '@docusaurus/Translate';

export default function Home() {
 return (
 <Layout>
 <h1>
 <Translate
 id="homepage.title"
 description="The homepage welcome message">
 Welcome to my website
 </Translate>
 </h1>
 <main>
 <Translate values={{firstName: 'Sébastien'}}>
 {'Welcome, {firstName}! How are you?'}
 </Translate>
 </main>
 </Layout>
);
}
```

### NOTE

You can even omit the `children` prop and specify a translation string in your `code.json` file manually after running the `docusaurus write-translations` CLI command.

```
<Translate id="homepage.title" />
```

**! INFO**

The `<Translate>` component supports interpolation. You can also implement [string pluralization](#) through some custom code and the `translate` imperative API.

## 2.2.2 Hooks

### **useDocusaurusContext**

React hook to access Docusaurus Context. The context contains the `siteConfig` object from `docusaurus.config.js` and some additional site metadata.

```
type PluginVersionInformation =
 | {readonly type: 'package'; readonly version?: string}
 | {readonly type: 'project'}
 | {readonly type: 'local'}
 | {readonly type: 'synthetic'};

type SiteMetadata = {
 readonly docusaurusVersion: string;
 readonly siteVersion?: string;
 readonly pluginVersions: Record<string, PluginVersionInformation>;
};

type I18nLocaleConfig = {
 label: string;
 direction: string;
};

type I18n = {
 defaultLocale: string;
 locales: [string, ...string[]];
 currentLocale: string;
 localeConfigs: Record<string, I18nLocaleConfig>;
};

type DocusaurusContext = {
 siteConfig: DocusaurusConfig;
 siteMetadata: SiteMetadata;
 globalData: Record<string, unknown>;
 i18n: I18n;
 codeTranslations: Record<string, string>;
};
```

Usage example:



```
import React from 'react';
import useDocusaurusContext from '@docusaurus/useDocusaurusContext';

const MyComponent = () => {
 const {siteConfig, siteMetadata} = useDocusaurusContext();
 return (
 <div>
 <h1>{siteConfig.title}</h1>
 <div>{siteMetadata.siteVersion}</div>
 <div>{siteMetadata.docusaurusVersion}</div>
 </div>
);
};
```

### NOTE

The `siteConfig` object only contains **serializable values** (values that are preserved after `JSON.stringify()`). Functions, regexes, etc. would be lost on the client side.

## useIsBrowser

Returns `true` after initial hydration completes in the browser.

### WARNING

Use this hook instead of `typeof window !== 'undefined'` in React rendering logic.

The initial render output must be consistent between server and client — components should not depend on browser-only state during the initial render. See [The Perils of Rehydration](#).

`useIsBrowser()` and `typeof window !== 'undefined'` serve different purposes:

- **`useIsBrowser()`** — tracks hydration state. Use it to gate UI behavior that should only activate after initial hydration.
- **`typeof window !== 'undefined'`** — detects the runtime environment. Use it when you need to check DOM availability outside of React rendering logic.

Context	<code>useIsBrowser()</code>	<code>typeof window !== 'undefined'</code>
Node.js / SSR	false	false
During initial client render	false	true (DOM is available)
After initial hydration	true	true

Usage example:



```
import React from 'react';
import useIsBrowser from '@docusaurus/useIsBrowser';

const MyComponent = () => {
 const isBrowser = useIsBrowser();
 return <div>{isBrowser ? 'Browser' : 'Server'}</div>;
};
```

## useBaseUrl

React hook to prepend your site `baseUrl` to a string.

### ⚠ WARNING

#### Don't use it for regular links!

The `/baseUrl/` prefix is automatically added to all **absolute paths** by default:

- Markdown: `[link] (/my/path)` will link to `/baseUrl/my/path`
- React: `<Link to="/my/path/">link</Link>` will link to `/baseUrl/my/path`

## Options

```
type BaseUrlOptions = {
 forcePrependBaseUrl: boolean;
 absolute: boolean;
};
```

## Example usage:

```
import React from 'react';
import useBaseUrl from '@docusaurus/useBaseUrl';

const SomeImage = () => {
 const imgSrc = useBaseUrl('/img/myImage.png');
 return ;
};
```

### 💡 TIP

In most cases, you don't need `useBaseUrl`.



Prefer a `require()` call for `assets`:

```

```

## useBaseUrlUtils

Sometimes `useBaseUrl` is not good enough. This hook return additional utils related to your site's base URL.

- `withBaseUrl`: useful if you need to add base URLs to multiple URLs at once.

```
import React from 'react';
import {useBaseUrlUtils} from '@docusaurus/useBaseUrl';

const Component = () => {
 const urls = ['/a', '/b'];
 const {withBaseUrl} = useBaseUrlUtils();
 const urlsWithBaseUrl = urls.map(withBaseUrl);
 return <div>{/* ... */</div>;
};
```

## useGlobalData

React hook to access Docusaurus global data created by all the plugins.

Global data is namespaced by plugin name then by plugin ID.

### ! INFO

Plugin ID is only useful when a plugin is used multiple times on the same site. Each plugin instance is able to create its own global data.

```
type GlobalData = Record<
 PluginName,
 Record<
 PluginId, // "default" by default
 any // plugin-specific data
 >
>;
```

Usage example:



```
import React from 'react';
import useGlobalData from '@docusaurus/useGlobalData';

const MyComponent = () => {
 const globalData = useGlobalData();
 const myPluginData = globalData['my-plugin']['default'];
 return <div>{myPluginData.someAttribute}</div>;
};
```

**TIP**

Inspect your site's global data at `.docusaurus/globalData.json`

### usePluginData

Access global data created by a specific plugin instance.

This is the most convenient hook to access plugin global data and should be used most of the time.

`pluginId` is optional if you don't use multi-instance plugins.

```
function usePluginData(
 pluginName: string,
 pluginId?: string,
 options?: {failfast?: boolean},
);
```

Usage example:

```
import React from 'react';
import {usePluginData} from '@docusaurus/useGlobalData';

const MyComponent = () => {
 const myPluginData = usePluginData('my-plugin');
 return <div>{myPluginData.someAttribute}</div>;
};
```

### useAllPluginInstancesData

Access global data created by a specific plugin. Given a plugin name, it returns the data of all the plugins instances of that name, by plugin id.



```
function useAllPluginInstancesData(
 pluginName: string,
 options?: {failfast?: boolean},
);
```

Usage example:

```
import React from 'react';
import {useAllPluginInstancesData} from '@docusaurus/useGlobalData';

const MyComponent = () => {
 const allPluginInstancesData = useAllPluginInstancesData('my-plugin');
 const myPluginData = allPluginInstancesData['default'];
 return <div>{myPluginData.someAttribute}</div>;
};
```

## useBrokenLinks

React hook to access the Docusaurus broken link checker APIs, exposing a way for a Docusaurus pages to report and collect their links and anchors.

### WARNING

This is an **advanced** API that **most Docusaurus users don't need to use directly**.

It is already **built-in** in existing high-level components:

- the `<Link>` component will collect links for you
- the `@theme/Heading` (used for Markdown headings) will collect anchors

Use `useBrokenLinks()` if you implement your own `<Heading>` or `<Link>` component.

Usage example:

MyHeading.js





```
import useBrokenLinks from '@docusaurus/useBrokenLinks';

export default function MyHeading(props) {
 useBrokenLinks().collectAnchor(props.id);
 return <h2 {...props} />;
}
```

MyLink.js

```
import useBrokenLinks from '@docusaurus/useBrokenLinks';

export default function MyLink(props) {
 useBrokenLinks().collectLink(props.href);
 return <a {...props} />;
}
```

## 2.2.3 Functions

### interpolate

The imperative counterpart of the `<Interpolate>` component.

### Signature

```
// Simple string interpolation
function interpolate(text: string, values: Record<string, string>): string;

// JSX interpolation
function interpolate(
 text: string,
 values: Record<string, ReactNode>,
): ReactNode;
```

### Example

```
import {interpolate} from '@docusaurus/Interpolate';

const message = interpolate('Welcome {firstName}', {firstName:
'Sébastien'});
```



## translate

The imperative counterpart of the `<Translate>` component. Also supporting [placeholders interpolation](#).

### TIP

Use the imperative API for the **rare cases** where a **component cannot be used**, such as:

- the page `title` metadata
- the `placeholder` props of form inputs
- the `aria-label` props for accessibility

## Signature

```
function translate(
 translation: {message: string; id?: string; description?: string},
 values: Record<string, string>,
): string;
```

## Example

```
src/pages/index.js
```



```
import React from 'react';
import Layout from '@theme/Layout';

import {translate} from '@docusaurus/Translate';

export default function Home() {
 return (
 <Layout
 title={translate({message: 'My page meta title'})}>
 <img
 src={'https://docusaurus.io/logo.png'}
 aria-label={
 translate(
 {
 message: 'The logo of site {siteName}',
 // Optional
 id: 'homepage.logo.ariaLabel',
 description: 'The home page logo aria label',
 },
 {siteName: 'Docusaurus'}
)
 }
 />
 </Layout>
);
}
```

## 2.2.4 Modules

### ExecutionEnvironment

A module that exposes a few boolean variables to check the current rendering environment.

#### WARNING

For React rendering logic, use `useIsBrowser()` or `<BrowserOnly>` instead.

Example:

```
import ExecutionEnvironment from '@docusaurus/ExecutionEnvironment';

if (ExecutionEnvironment.canUseDOM) {
 require('lib-that-only-works-client-side');
}
```



Field	Description
<code>ExecutionEnvironment.canUseDOM</code>	<code>true</code> if on client/browser, <code>false</code> on Node.js/prerendering.
<code>ExecutionEnvironment.canUseEventListeners</code>	<code>true</code> if on client and has <code>window.addEventListener</code> .
<code>ExecutionEnvironment.canUseIntersectionObserver</code>	<code>true</code> if on client and has <code>IntersectionObserver</code> .
<code>ExecutionEnvironment.canUseViewport</code>	<code>true</code> if on client and has <code>window.screen</code> .

### constants

A module exposing useful constants to client-side theme code.

```
import {DEFAULT_PLUGIN_ID} from '@docusaurus/constants';
```

Named export	Value
<code>DEFAULT_PLUGIN_ID</code>	<code>default</code>

## 2.3 docusaurus.config.js

### ! INFO

Refer to the Getting Started [Configuration](#) for examples.

### 2.3.1 Overview

`docusaurus.config.js` contains configurations for your site and is placed in the root directory of your site.

### i NOTE

With a [TypeScript](#) Docusaurus codebase your config file may be called `docusaurus.config.ts`. The syntax is broadly identical to the `js` config file with the addition of types. You can see an example on the [Docusaurus Website](#) itself.

This file is **run in Node.js** and should export a site configuration object, or a function that creates it.

The `docusaurus.config.js` file supports:

- [ES Modules](#)
- [CommonJS](#)
- [TypeScript](#)

Examples:



docusaurus.config.js

```
export default {
 title: 'Docusaurus',
 url: 'https://docusaurus.io',
 // your site config ...
};
```

docusaurus.config.js

```
export default async function createConfigAsync() {
 return {
 title: 'Docusaurus',
 url: 'https://docusaurus.io',
 // your site config ...
 };
}
```

**TIP**

Refer to [Syntax to declare docusaurus.config.js](#) for a more exhaustive list of examples and explanations.

## 2.3.2 Required fields

**title**

- Type: `string`

Title for your website. Will be used in metadata and as browser tab title.

docusaurus.config.js

```
export default {
 title: 'Docusaurus',
};
```

**url**

- Type: `string`



URL for your website. This can also be considered the top-level hostname. For example, `https://facebook.github.io` is the URL of `https://facebook.github.io/metro/`, and `https://docusaurus.io` is the URL for `https://docusaurus.io`. This field is related to the `baseUrl` field.

docusaurus.config.js

```
export default {
 url: 'https://docusaurus.io',
};
```

### ! SPECIAL CASE FOR I18N SITES

If your site uses multiple locales, it is possible to provide a distinct `url` for each locale thanks to the `siteConfig.i18n.localeConfigs[<locale>].url` attribute. This makes it possible to deploy a localized Docusaurus site over multiple domains.

## baseUrl

- Type: `string`

The base URL of your site is the path segment appearing just after the `url`, letting you eventually host your site under a subpath instead of at the root of the domain.

For example, let's consider you want to host a site at `https://facebook.github.io/metro/`, then you must configure it accordingly:

- `url` should be `'https://facebook.github.io'`
- `baseUrl` should be  `'/metro/'`

By default, a Docusaurus site is hosted at the root of the domain:

docusaurus.config.js

```
export default {
 baseUrl: '/',
};
```

### ! SPECIAL CASE FOR I18N SITES

If your site uses multiple locales, then Docusaurus will automatically localize the `baseUrl` of your site based on smart heuristics:



- For the default locale, `baseUrl` will be `/<siteBaseUrl>/`
- For other locales, `baseUrl` will be `/<siteBaseUrl>/<locale>/`
- When building a single locale at a time (with `docusaurus build --locale <locale>`), `baseUrl` will be `/<siteBaseUrl>/`, assuming the intent is to [deploy each locale on distinct domains](#).

When the localized `baseUrl` Docusaurus computes doesn't satisfy you, it's always possible to override it by providing an explicit localized `baseUrl` thanks to the `siteConfig.i18n.localeConfigs[<locale>].baseUrl` attribute.

## 2.3.3 Optional fields

### **favicon**

- Type: `string | undefined`

Path to your site favicon; must be a URL that can be used in link's href. For example, if your favicon is in `static/img/favicon.ico`:

docusaurus.config.js

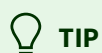
```
export default {
 favicon: '/img/favicon.ico',
};
```

### **trailingSlash**

- Type: `boolean | undefined`

Allow to customize the presence/absence of a trailing slash at the end of URLs/links, and how static HTML files are generated:

- `undefined` (default): keeps URLs untouched, and emit `/docs/myDoc/index.html` for `/docs/myDoc.md`
- `true`: add trailing slashes to URLs/links, and emit `/docs/myDoc/index.html` for `/docs/myDoc.md`
- `false`: remove trailing slashes from URLs/links, and emit `/docs/myDoc.html` for `/docs/myDoc.md`



#### **TIP**

Each static hosting provider serves static files differently (this behavior may even change over time).

Refer to the [deployment guide](#) and [slorber/trailing-slash-guide](#) to choose the appropriate setting.

### **i18n**

- Type: `Object`



The `i18n` configuration object to [localize your site](#).

Example:

docusaurus.config.js

```
export default {
 i18n: {
 defaultLocale: 'en',
 locales: ['en', 'fa'],
 path: 'i18n',
 localeConfigs: {
 en: {
 label: 'English',
 direction: 'ltr',
 htmlLang: 'en-US',
 calendar: 'gregory',
 path: 'en',
 translate: false,
 url: 'https://en.example.com',
 baseUrl: '/',
 },
 fa: {
 label: 'فارسی',
 direction: 'rtl',
 htmlLang: 'fa-IR',
 calendar: 'persian',
 path: 'fa',
 translate: true,
 url: 'https://fa.example.com',
 baseUrl: '/',
 },
 },
 },
};
```

- `defaultLocale`: The locale that (1) does not have its name in the base URL (2) gets started with `docusaurus start` without `--locale` option (3) will be used for the `<link hrefLang="x-default">` tag
- `locales`: List of locales deployed on your site. Must contain `defaultLocale`.
- `path`: Root folder which all locale folders are relative to. Can be absolute or relative to the config file. Defaults to `i18n`.
- `localeConfigs`: Individual options for each locale.
  - `label`: The label displayed for this locale in the locales dropdown.
  - `direction`: `ltr` (default) or `rtl` (for [right-to-left languages](#) like Farsi, Arabic, Hebrew, etc.). Used to select the locale's CSS and HTML meta attribute.





- `htmlLang`: BCP 47 language tag to use in `<html lang="...">` (or any other DOM tag name) and in `<link ... hreflang="...">`
- `calendar`: the `calendar` used to calculate the date era. Note that it doesn't control the actual string displayed: `MM/DD/YYYY` and `DD/MM/YYYY` are both `gregory`. To choose the format (`DD/MM/YYYY` or `MM/DD/YYYY`), set your locale name to `en-GB` or `en-US` (`en` means `en-US`).
- `path`: Root folder that all plugin localization folders of this locale are relative to. Will be resolved against `i18n.path`. Defaults to the locale's name (`i18n/<locale>`). Note: this has no effect on the locale's `baseUrl`—customization of base URL is a work-in-progress.
- `translate`: Should we run the translation process for this locale? By default, it is enabled if the `i18n/<locale>` folder exists
- `url`: This lets you override the `siteConfig.url`, particularly useful if your site is [deployed over multiple domains](#).
- `baseUrl`: This lets you override the default localized `baseUrl` Docusaurus infers from your `siteConfig.baseUrl`, giving you more control to host your localized site in less common ways, in particularly [deployments over multi-domains](#)

## storage

- Type: `Object`

Site-wide browser storage options that theme authors should strive to respect.

docusaurus.config.js

```
export default {
 storage: {
 type: 'localStorage',
 namespace: true,
 },
};
```

- `type`: The browser storage theme authors should use. Possible values are `localStorage` and `sessionStorage`. Defaults to `localStorage`.
- `namespace`: Whether to namespace the browser storage keys to avoid storage key conflicts when Docusaurus sites are hosted under the same domain, or on localhost. Possible values are `string` | `boolean`. The namespace is appended at the end of the storage keys `key-namespace`. Use `true` to automatically generate a random namespace from your site `url + baseUrl`. Defaults to `false` (no namespace, historical behavior). Use the `future.v4.siteStorageNamespacing` flag to default this to `true`.

## future

- Type: `Object`

The `future` configuration object permits to opt-in for upcoming/unstable/experimental Docusaurus features that are not ready for prime time.



It is also a way to opt-in for upcoming breaking changes coming in the next major versions, enabling you to prepare your site for the next version while staying on the previous one. The [Remix Future Flags blog post](#) greatly explains this idea.

### BREAKING CHANGES IN MINOR VERSIONS

Features prefixed by `experimental_` or `unstable_` are subject to changes in **minor versions**, and not considered as [Semantic Versioning breaking changes](#).

Features namespaced by `v<MajorVersion>` (`v6`, `v7`, etc.) are future flags that are expected to be turned on by default in the next major versions. These are less likely to change, but we keep the possibility to do so.

`future` API breaking changes should be easy to handle, and will be documented in minor/major version blog posts.

Example:

docusaurus.config.js

```
export default {
 future: {
 v4: {
 useCssCascadeLayers: true,
 siteStorageNamespacing: true,
 fasterByDefault: true,
 mdx1CompatDisabledByDefault: true,
 },
 faster: {
 swcJsLoader: true,
 swcJsMinimizer: true,
 swcHtmlMinimizer: true,
 lightningCssMinimizer: true,
 rspackBundler: true,
 rspackPersistentCache: true,
 ssgWorkerThreads: true,
 mdxCrossCompilerCache: true,
 },
 experimental_router: 'hash',
 },
};
```

- `v4`: Permits to opt-in for upcoming Docusaurus v4 breaking changes and features, to prepare your site in advance for this new version. Use `true` as a shorthand to enable all the flags.
  - `useCssCascadeLayers`: This enables the [Docusaurus CSS Cascade Layers plugin](#) with pre-configured layers that we plan to apply by default for Docusaurus v4.



- `siteStorageNamespacing`: Defaults the `storage.namespace` config to `true` instead of `false`. This enables automatic browser storage key namespacing, which avoids storage key conflicts when multiple Docusaurus sites are hosted under the same domain, or on localhost.
- `fasterByDefault`: Defaults all `future.faster` flags to `true` instead of `false`. This enables [Docusaurus Faster](#) features by default. Requires having `@docusaurus/faster` in your dependencies. If you explicitly set individual `faster` flags, those explicit values take precedence.
- `mdx1CompatDisabledByDefault`: Defaults all `markdown.mdx1Compat` flags to `false` instead of `true`. This prepares your site for Docusaurus v4, which will not enable MDX v1 compatibility by default. If you explicitly set individual `mdx1Compat` flags, those explicit values take precedence.
- `faster`: An object containing feature flags to make the Docusaurus build faster. This requires adding the `@docusaurus/faster` package to your site's dependencies. Use `true` as a shorthand to enable all flags. Read more on the [Docusaurus Faster](#) issue. Available feature flags:
  - `swcJsLoader`: Use [SWC](#) to transpile JS (instead of [Babel](#)).
  - `swcJsMinimizer`: Use [SWC](#) to minify JS (instead of [Terser](#)).
  - `swcHtmlMinimizer`: Use [SWC](#) to minify HTML and inlined JS/CSS (instead of [html-minifier-terser](#)).
  - `lightningCssMinimizer`: Use [Lightning CSS](#) to minify CSS (instead of [cssnano](#) and [clean-css](#)).
  - `rspackBundler`: Use [Rspack](#) to bundle your app (instead of [webpack](#)).
  - `rspackPersistentCache`: Use [Rspack Persistent Cache](#) to re-build your app faster on subsequent builds. Requires `rspackBundler: true`. Requires persisting `./node_modules/.cache` across rebuilds.
  - `mdxCrossCompilerCache`: Compile MDX files only once for both browser/Node.js environments instead of twice.
  - `ssgWorkerThreads`: Using a Node.js worker thread pool to execute the static site generation phase faster.
  - `gitEagerVcs`: Upgrades the default [VCS strategy](#) to `default-v2`, that reads your whole Git repository at once instead of per-file, making Git operations faster on large repositories.
- `experimental_router`: The router type to use. Possible values are `browser` and `hash`. Defaults to `browser`. The `hash` router is only useful for rare cases where you want to opt-out of static site generation, have a fully client-side app with a single `index.html` entrypoint file. This can be useful to distribute a Docusaurus site as a `.zip` archive that you can [browse locally without running a web server](#).
- `experimental_vcs`: The Version Control System (VCS) implementation to use to read file info (creation/last update date/author). Read the [dedicated section](#) below for details.

## experimental\_vcs

This exposes an API that lets you provide your own Version Control System (VCS) implementation to read file info (creation/last update date/author).



```
export default {
 future: {
 experimental_vcs: {
 initialize: ({siteDir}) => {
 // Initialize your VCS client here.
 // If you want to read your VCS eagerly/incrementally on startup,
 // this is the place to do it.
 // This function is synchronous on purpose and not awaited
 // It should not delay Docusaurus startup, but be run in parallel.
 },
 getFileCreationInfo: async (filePath: string) => {
 // Provide your own implementation to read file creation info.
 return getFileCreationInfo(filePath);
 },
 getFileLastUpdateInfo: async (filePath: string) => {
 // Provide your own implementation to read file creation info.
 return getFileLastUpdateInfo(filePath);
 },
 },
 },
};
```

## VCS Presets

It is possible to pass a boolean VCS value:

- `true`: enables the default VCS preset (`default-v1` or `default-v2`, depending on the Docusaurus Faster `gitEagerVcs` flag value)
- `false`: disables the VCS, always returns `null` for all files

```
export default {
 future: {
 experimental_vcs: true, // Enables the default VCS preset
 },
};
```

It is also possible to choose VCS preset we provide out of the box by its name.

```
export default {
 future: {
 experimental_vcs: 'presetName',
 },
};
```



The available preset names are:

- `git-ad-hoc`: the historical `git log <filename>` based strategy.
- `git-eager`: the new Git strategy that reads your whole repository upfront.
- `hardcoded`: returns hardcoded value, useful in dev/tests to speed up developer experience.
- `disabled`: returns `null` for all files, considering them untracked.
- `default-v1`: the historical default (`git-ad-hoc` in prod, `hardcoded` in dev)
- `default-v2`: the upcoming default (`git-eager` in prod, `hardcoded` in dev)

Unless you have specific needs, we recommend using the default presets (`default-v1` or `default-v2`), that skip reading file info in development mode for better performance.

## VCS Types

```
type VcsChangeInfo = {timestamp: number; author: string};

type VscInitializeParams = {
 siteDir: string;
};

type VcsConfig = {
 initialize: (params: VscInitializeParams) => void;
 getFileCreationInfo: (filePath: string) => Promise<VcsChangeInfo | null>;
 getFileLastUpdateInfo: (filePath: string) => Promise<VcsChangeInfo |
null>;
};

type VcsPreset =
 | 'git-ad-hoc'
 | 'git-eager'
 | 'hardcoded'
 | 'disabled'
 | 'default-v1'
 | 'default-v2';
```

## noIndex

- Type: `boolean`

This option adds `<meta name="robots" content="noindex, nofollow">` to every page to tell search engines to avoid indexing your site (more information [here](#)).

Example:

```
docusaurus.config.js
```



```
export default {
 noIndex: true, // Defaults to `false`
};
```

### onBrokenLinks

- Type: `'ignore' | 'log' | 'warn' | 'throw'`

The behavior of Docusaurus when it detects any broken link.

By default, it throws an error, to ensure you never ship any broken link.

#### NOTE

The broken links detection is only available for a production build (`docusaurus build`).

### onBrokenAnchors

- Type: `'ignore' | 'log' | 'warn' | 'throw'`

The behavior of Docusaurus when it detects any broken anchor declared with the `Heading` component of Docusaurus.

By default, it prints a warning, to let you know about your broken anchors.

### onBrokenMarkdownLinks

#### DEPRECATED

Deprecated in Docusaurus v3.9, and will be removed in Docusaurus v4.

Replaced by `siteConfig.markdown.hooks.onBrokenMarkdownLinks`

- Type: `'ignore' | 'log' | 'warn' | 'throw'`

The behavior of Docusaurus when it detects any broken Markdown link.

By default, it prints a warning, to let you know about your broken Markdown link.

### onDuplicateRoutes

- Type: `'ignore' | 'log' | 'warn' | 'throw'`

The behavior of Docusaurus when it detects any `duplicate routes`.

By default, it displays a warning after you run `yarn start` or `yarn build`.



## tagline

- Type: `string`

The tagline for your website.

docusaurus.config.js

```
export default {
 tagline:
 'Docusaurus makes it easy to maintain Open Source documentation
websites.',
};
```

## organizationName

- Type: `string`

The GitHub user or organization that owns the repository. You don't need this if you are not using the `docusaurus deploy` command.

docusaurus.config.js

```
export default {
 // Docusaurus' organization is facebook
 organizationName: 'facebook',
};
```

## projectName

- Type: `string`

The name of the GitHub repository. You don't need this if you are not using the `docusaurus deploy` command.

docusaurus.config.js

```
export default {
 projectName: 'docusaurus',
};
```

## deploymentBranch



- Type: `string`

The name of the branch to deploy the static files to. You don't need this if you are not using the `docusaurus deploy` command.

docusaurus.config.js

```
export default {
 deploymentBranch: 'gh-pages',
};
```

## githubHost

- Type: `string`

The hostname of your server. Useful if you are using GitHub Enterprise. You don't need this if you are not using the `docusaurus deploy` command.

docusaurus.config.js

```
export default {
 githubHost: 'github.com',
};
```

## githubPort

- Type: `string`

The port of your server. Useful if you are using GitHub Enterprise. You don't need this if you are not using the `docusaurus deploy` command.

docusaurus.config.js

```
export default {
 githubPort: '22',
};
```

## themeConfig

- Type: `Object`

The [theme configuration](#) object to customize your site UI like navbar and footer.





Example:

```
docusaurus.config.js
```



```
export default {
 themeConfig: {
 docs: {
 sidebar: {
 hideable: false,
 autoCollapseCategories: false,
 },
 },
 colorMode: {
 defaultMode: 'light',
 disableSwitch: false,
 respectPrefersColorScheme: true,
 },
 navbar: {
 title: 'Site Title',
 logo: {
 alt: 'Site Logo',
 src: 'img/logo.svg',
 width: 32,
 height: 32,
 },
 items: [
 {
 to: 'docs/docusaurus.config.js',
 activeBasePath: 'docs',
 label: 'docusaurus.config.js',
 position: 'left',
 },
 // ... other links
],
 },
 footer: {
 style: 'dark',
 links: [
 {
 title: 'Docs',
 items: [
 {
 label: 'Docs',
 to: 'docs/doc1',
 },
],
 },
 // ... other links
],
 logo: {
 alt: 'Meta Open Source Logo',
 src: 'img/meta_oss_logo.png',
 href: 'https://opensource.fb.com',
 },
 },
 },
}
```



```
 width: 160,
 height: 51,
 },
 copyright: `Copyright © ${new Date().getFullYear()} Facebook, Inc.`,
 // You can also put own HTML here
},
},
};
```

## plugins

- Type: `PluginConfig[]`

```
type PluginConfig = string | [string, any] | PluginModule | [PluginModule, any];
```

See [plugin method references](#) for the shape of a `PluginModule`.

docusaurus.config.js

```
export default {
 plugins: [
 'docusaurus-plugin-awesome',
 ['docusuarus-plugin-confetti', {fancy: false}],
 () => ({
 postBuild() {
 console.log('Build finished');
 },
 }),
],
};
```

## themes

- Type: `PluginConfig[]`

docusaurus.config.js

```
export default {
 themes: ['@docusaurus/theme-classic'],
};
```



## presets

- Type: `PresetConfig[]`

```
type PresetConfig = string | [string, any];
```

docusaurus.config.js

```
export default {
 presets: [],
};
```

## markdown

The global Docusaurus Markdown config.

- Type: `MarkdownConfig`



```
type MarkdownPreprocessor = (args: {
 filePath: string;
 fileContent: string;
}) => string;

type MDX1CompatOptions =
 | boolean
 | {
 comments: boolean;
 admonitions: boolean;
 headingIds: boolean;
 };

type ParseFrontMatter = (params: {
 filePath: string;
 fileContent: string;
 defaultParseFrontMatter: ParseFrontMatter;
}) => Promise<{
 frontMatter: {[key: string]: unknown};
 content: string;
}>;

type MarkdownAnchorsConfig = {
 maintainCase: boolean;
};

type OnBrokenMarkdownLinksFunction = (params: {
 sourceFilePath: string; // MD/MDX source file relative to cwd
 url: string; // Link url
 node: Link | Definition; // mdast node
}) => void | string;

type OnBrokenMarkdownImagesFunction = (params: {
 sourceFilePath: string; // MD/MDX source file relative to cwd
 url: string; // Image url
 node: Image; // mdast node
}) => void | string;

type OnUnusedMarkdownDirectivesFunction = (params: {
 sourceFilePath: string; // MD/MDX source file relative to cwd
 directives: Directives[]; // mdast nodes
}) => void | string;

type ReportingSeverity = 'ignore' | 'log' | 'warn' | 'throw';

type MarkdownHooks = {
 onBrokenMarkdownLinks: ReportingSeverity | OnBrokenMarkdownLinksFunction;
 onBrokenMarkdownImages: ReportingSeverity |
 OnBrokenMarkdownImagesFunction;
```



```
onUnusedMarkdownDirectives:
 | ReportingSeverity
 | OnUnusedMarkdownDirectivesFunction;
};

type MarkdownConfig = {
 format: 'mdx' | 'md' | 'detect';
 mermaid: boolean;
 emoji: boolean;
 preprocessor?: MarkdownPreprocessor;
 parseFrontMatter?: ParseFrontMatter;
 mdx1Compat: MDX1CompatOptions;
 remarkRehypeOptions: object; // see https://github.com/remarkjs/remark-rehype#options
 anchors: MarkdownAnchorsConfig;
 hooks: MarkdownHooks;
};
```

Example:

```
docusaurus.config.js
```



```
export default {
 markdown: {
 format: 'mdx',
 mermaid: true,
 emoji: true,
 preprocessor: ({filePath, fileContent}) => {
 return fileContent.replaceAll('{MY_VAR}', 'MY_VALUE');
 },
 parseFrontMatter: async (params) => {
 const result = await params.defaultParseFrontMatter(params);
 result.frontMatter.description =
 result.frontMatter.description?.replaceAll('{MY_VAR}',
'MY_VALUE');
 return result;
 },
 mdx1Compat: {
 comments: true,
 admonitions: true,
 headingIds: true,
 },
 anchors: {
 maintainCase: true,
 },
 hooks: {
 onBrokenMarkdownLinks: 'warn',
 onBrokenMarkdownImages: 'throw',
 onUnusedMarkdownDirectives: 'warn',
 },
 },
};
```

Name	Type	Default	Description
format	'mdx'   'md'   'detect'	'mdx'	The default parser format to use for Markdown content. Using 'detect' will select the appropriate format automatically based on file extensions: <code>.md</code> vs <code>.mdx</code> .
mermaid	boolean	false	When <code>true</code> , allows Docusaurus to render Markdown code blocks with <code>mermaid</code> language as Mermaid diagrams.
emoji	boolean	true	When <code>true</code> , allows Docusaurus to render emoji shortcodes (e.g., <code>:+1:</code> ) as Unicode emoji (👍). When <code>false</code> , emoji shortcodes are left as-is.
preprocessor	MarkdownPreprocessor or	undefined	Gives you the ability to alter the Markdown content string before parsing. Use it as a last-resort escape hatch or workaround: it is almost always better to implement a Remark/Rehype plugin.



Name	Type	Default	Description
<code>parseFrontMatter</code>	<code>ParseFrontMatter</code>	<code>undefined</code>	Gives you the ability to provide your own front matter parser, or to enhance the default parser. Read our <a href="#">front matter guide</a> for details.
<code>mdx1Compat</code>	<code>MDX1CompatOptions</code>	<code>{comments: true, admonitions: true, headingIds: true}</code>	Compatibility options to make it easier to upgrade to Docusaurus v3+. Defaults to all <code>false</code> when <code>future.v4.mdx1CompatDisabledByDefault</code> is enabled.
<code>anchors</code>	<code>MarkdownAnchorsConfig</code>	<code>{maintainCase: false}</code>	Options to control the behavior of anchors generated from Markdown headings
<code>remarkRehypeOptions</code>	<code>object</code>	<code>undefined</code>	Makes it possible to pass custom <a href="#">remark-rehype options</a> .
<code>hooks</code>	<code>MarkdownHooks</code>	<code>object</code>	Make it possible to customize the MDX loader behavior with callbacks or built-in options.
<code>hooks.onBrokenMarkdownLinks</code>	<code>ReportingSeverity   OnBrokenMarkdownLinksFunction</code>	<code>'warn'</code>	Hook to customize the behavior when encountering a broken Markdown link URL. With the callback function, you can return a new link URL, or alter the link <a href="#">mdast node</a> .
<code>hooks.onBrokenMarkdownImages</code>	<code>ReportingSeverity   OnBrokenMarkdownImagesFunction</code>	<code>'throw'</code>	Hook to customize the behavior when encountering a broken Markdown image URL. With the callback function, you can return a new image URL, or alter the image <a href="#">mdast node</a> .
<code>hooks.onUnusedMarkdownDirectives</code>	<code>ReportingSeverity   OnUnusedMarkdownDirectivesFunction</code>	<code>'warn'</code>	Hook to customize the behavior when encountering an unused <a href="#">Markdown directive</a> . Also accepts a callback function, giving you access to the raw <code>Directive</code> AST nodes.

## customFields

Docusaurus guards `docusaurus.config.js` from unknown fields. To add a custom field, define it on `customFields`.

- Type: `Object`

`docusaurus.config.js`

```
export default {
 customFields: {
 admin: 'endi',
 superman: 'lol',
 },
};
```

Attempting to add unknown fields in the config will lead to errors during build time:





Error: The field(s) 'foo', 'bar' are not recognized in docusaurus.config.js

## staticDirectories

An array of paths, relative to the site's directory or absolute. Files under these paths will be copied to the build output as-is.

- Type: `string[]`

Example:

docusaurus.config.js

```
export default {
 staticDirectories: ['static'],
};
```

## headTags

An array of tags that will be inserted in the HTML `<head>`. The values must be objects that contain two properties: `tagName` and `attributes`. `tagName` must be a string that determines the tag being created; eg "link". `attributes` must be an attribute-value map. When custom html elements are needed, set `customElement: true`.

- Type: `{ tagName: string; attributes: Object; customElement?: boolean; }[]`

Example:

docusaurus.config.js

```
export default {
 headTags: [
 {
 tagName: 'link',
 attributes: {
 rel: 'icon',
 href: '/img/docusaurus.png',
 },
 },
],
};
```



This would become `<link rel="icon" href="img/docusaurus.png" />` in the generated HTML.

## scripts

An array of scripts to load. The values can be either strings or plain objects of attribute-value maps. The `<script>` tags will be inserted in the HTML `<head>`. If you use a plain object, the only required attribute is `src`, and any other attributes are permitted (each one should have boolean/string values).

Note that `<script>` added here are render-blocking, so you might want to add `async: true`/`defer: true` to the objects.

- Type: `(string | Object)[]`

Example:

docusaurus.config.js

```
export default {
 scripts: [
 // String format.
 'https://docusaurus.io/script.js',
 // Object format.
 {
 src:
 'https://cdnjs.cloudflare.com/ajax/libs/clipboard.js/2.0.0/clipboard.min.js',
 async: true,
 },
],
};
```

## stylesheets

An array of CSS sources to load. The values can be either strings or plain objects of attribute-value maps. The `<link>` tags will be inserted in the HTML `<head>`. If you use an object, the only required attribute is `href`, and any other attributes are permitted (each one should have boolean/string values).

- Type: `(string | Object)[]`

Example:

docusaurus.config.js



```
export default {
 stylesheets: [
 // String format.
 'https://docusaurus.io/style.css',
 // Object format.
 {
 href: 'http://mydomain.com/style.css',
 },
],
};
```

### ! INFO

By default, the `<link>` tags will have `rel="stylesheet"`, but you can explicitly add a custom `rel` value to inject any kind of `<link>` tag, not necessarily stylesheets.

## clientModules

An array of [client modules](#) to load globally on your site.

Example:

docusaurus.config.js

```
export default {
 clientModules: ['./mySiteGlobalJs.js', './mySiteGlobalCss.css'],
};
```

## ssrTemplate

An HTML template written in [Eta's syntax](#) that will be used to render your application. This can be used to set custom attributes on the `body` tags, additional `meta` tags, customize the `viewport`, etc. Please note that Docusaurus will rely on the template to be correctly structured in order to function properly, once you do customize it, you will have to make sure that your template is compliant with the requirements from upstream.

- Type: `string`

Example:

docusaurus.config.js



```
export default {
 ssrTemplate: `<!DOCTYPE html>
<html <%~ it.htmlAttributes %>>
 <head>
 <meta charset="UTF-8">
 <meta name="generator" content="Docusaurus v<%= it.version %>">
 <% it.metaAttributes.forEach((metaAttribute) => { %>
 <%~ metaAttribute %>
 <% }); %>
 <%~ it.headTags %>
 <% it.stylesheets.forEach((stylesheet) => { %>
 <link rel="stylesheet" href="<%= it.baseUrl %><%= stylesheet %>" />
 <% }); %>
 <% it.scripts.forEach((script) => { %>
 <link rel="preload" href="<%= it.baseUrl %><%= script %>"
as="script">
 <% }); %>
 </head>
 <body <%~ it.bodyAttributes %>>
 <%~ it.preBodyTags %>
 <div id="__docusaurus">
 <%~ it.appHtml %>
 </div>
 <% it.scripts.forEach((script) => { %>
 <script src="<%= it.baseUrl %><%= script %>"></script>
 <% }); %>
 <%~ it.postBodyTags %>
 </body>
</html>`,
};
```

### titleDelimiter

- Type: `string`

Will be used as title delimiter in the generated `<title>` tag.

Example:

docusaurus.config.js

```
export default {
 titleDelimiter: '🦖', // Defaults to `|`
};
```



## baseUrlIssueBanner

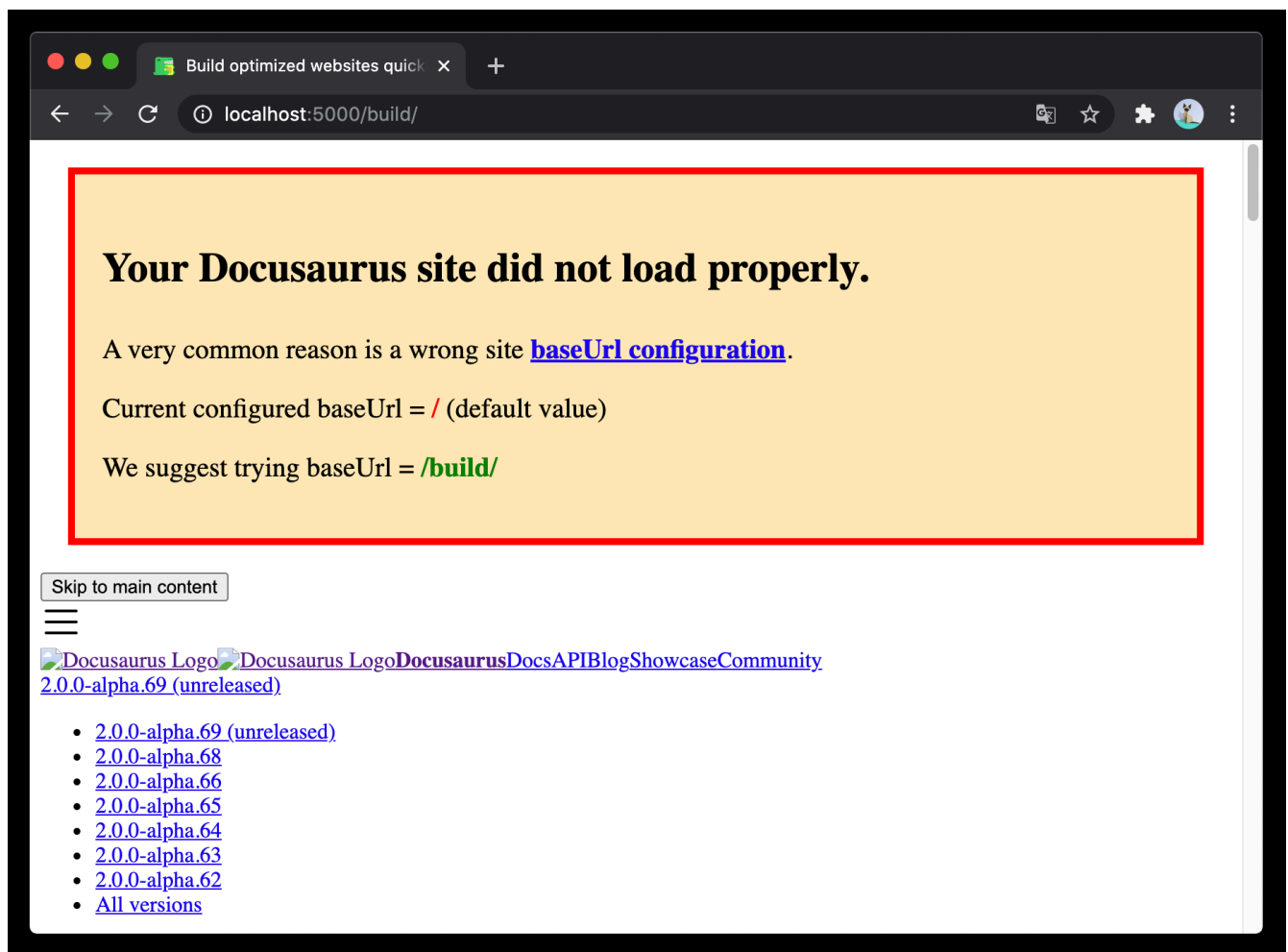
- Type: `boolean`

When enabled, will show a banner in case your site can't load its CSS or JavaScript files, which is a very common issue, often related to a wrong `baseUrl` in site config.

Example:

docusaurus.config.js

```
export default {
 baseUrlIssueBanner: true, // Defaults to `true`
};
```



### WARNING

This banner needs to inline CSS / JS in case all asset loading fails due to wrong base URL.



If you have a strict [Content Security Policy](#), you should rather disable it.

## 2.4 Plugin Method References

### ⚠ WARNING

This section is a work in progress. Anchor links or even URLs are not guaranteed to be stable.

Plugin APIs are shared by themes and plugins—themes are loaded just like plugins.

### 2.4.1 Plugin module

Every plugin is imported as a module. The module is expected to have the following members:

- A **default export**: the constructor function for the plugin.
- **Named exports**: the [static methods](#) called before plugins are initialized.

### 2.4.2 Plugin constructor

The plugin module's default export is a constructor function with the signature `(context: LoadContext, options: PluginOptions) => Plugin | Promise<Plugin>`.

#### context

`context` is plugin-agnostic, and the same object will be passed into all plugins used for a Docusaurus website. The `context` object contains the following fields:

```
type LoadContext = {
 siteDir: string;
 generatedFilesDir: string;
 siteConfig: DocusaurusConfig;
 outDir: string;
 baseUrl: string;
};
```

#### options

`options` are the [second optional parameter when the plugins are used](#). `options` are plugin-specific and are specified by users when they use them in `docusaurus.config.js`. If there's a `validateOptions` function exported, the `options` will be validated and normalized beforehand.

Alternatively, if a preset contains the plugin, the preset will then be in charge of passing the correct options into the plugin. It is up to the individual plugin to define what options it takes.

### 2.4.3 Example

Here's a mental model for a presumptuous plugin implementation.



```
// A JavaScript function that returns an object.
// `context` is provided by Docusaurus. Example: siteConfig can be accessed
from context.
// `opts` is the user-defined options.
export default async function myPlugin(context, opts) {
 return {
 // A compulsory field used as the namespace for directories to cache
 // the intermediate data for each plugin.
 // If you're writing your own local plugin, you will want it to
 // be unique in order not to potentially conflict with imported
 plugins.
 // A good way will be to add your own project name within.
 name: 'docusaurus-my-project-cool-plugin',

 async loadContent() {
 // The loadContent hook is executed after siteConfig and env has been
 loaded.
 // You can return a JavaScript object that will be passed to
 contentLoaded hook.
 },

 async contentLoaded({content, actions}) {
 // The contentLoaded hook is done after loadContent hook is done.
 // `actions` are set of functional API provided by Docusaurus (e.g.
 addRoute)
 },

 async postBuild(props) {
 // After docusaurus <build> finish.
 },

 // TODO
 async postStart(props) {
 // docusaurus <start> finish
 },

 configureWebpack(config, isServer, utils, content) {
 // Modify internal webpack config. If returned value is an Object, it
 // will be merged into the final config using webpack-merge;
 // If the returned value is a function, it will receive the config as
 the 1st argument and an isServer flag as the 2nd argument.
 },

 getPathsToWatch() {
 // Paths to watch.
 },

 getThemePath() {
 // Returns the path to the directory where the theme components can
```



```
// be found.
},

getClientModules() {
 // Return an array of paths to the modules that are to be imported
 // in the client bundle. These modules are imported globally before
 // React even renders the initial UI.
},

extendCli(cli) {
 // Register an extra command to enhance the CLI of Docusaurus
},

injectHtmlTags({content}) {
 // Inject head and/or body HTML tags.
},

async getTranslationFiles({content}) {
 // Return translation files
},

translateContent({content, translationFiles}) {
 // translate the plugin content here
},

translateThemeConfig({themeConfig, translationFiles}) {
 // translate the site themeConfig here
},

async getDefaultCodeTranslationMessages() {
 // return default theme translations here
},
};
}

export function validateOptions({options, validate}) {
 const validatedOptions = validate(myValidationSchema, options);
 return validatedOptions;
}

export function validateThemeConfig({themeConfig, validate}) {
 const validatedThemeConfig = validate(myValidationSchema, options);
 return validatedThemeConfig;
}
```

## 2.4.4 Lifecycle APIs

During the build, plugins are loaded in parallel to fetch their own contents and render them to routes. Plugins may also configure webpack or post-process the generated files.





## async loadContent()

Plugins should use this lifecycle to fetch from data sources (filesystem, remote API, headless CMS, etc.) or do some server processing. The return value is the content it needs.

For example, this plugin below returns a random integer between 1 and 10 as content.

docusaurus-plugin/src/index.js

```
export default function (context, options) {
 return {
 name: 'docusaurus-plugin',
 async loadContent() {
 return 1 + Math.floor(Math.random() * 10);
 },
 };
}
```

## async contentLoaded({content, actions})

The data that was loaded in `loadContent` will be consumed in `contentLoaded`. It can be rendered to routes, registered as global data, etc.

### content

`contentLoaded` will be called *after* `loadContent` is done. The return value of `loadContent()` will be passed to `contentLoaded` as `content`.

### actions

`actions` contain three functions:

### addRoute(config: RouteConfig): void

Create a route to add to the website.



```
export type RouteConfig = {
 /**
 * With leading slash. Trailing slash will be normalized by config.
 */
 path: string;
 /**
 * Component used to render this route, a path that the bundler can
 `require`.
 */
 component: string;
 /**
 * Props. Each entry should be `[propName]: pathToPropsModule` (created
 with
 * `createData`)
 */
 modules?: RouteModules;
 /**
 * The route context will wrap the `component`. Use `useRouteContext` to
 * retrieve what's declared here. Note that all custom route context
 declared
 * here will be namespaced under {@link RouteContext.data}.
 */
 context?: RouteModules;
 /**
 * Nested routes config, useful for "layout routes" having subroutes.
 */
 routes?: RouteConfig[];
 /**
 * React router config option: `exact` routes would not match subroutes.
 */
 exact?: boolean;
 /**
 * React router config option: `strict` routes are sensitive to the
 presence
 * of a trailing slash.
 */
 strict?: boolean;
 /**
 * Used to sort routes.
 * Higher-priority routes will be matched first.
 */
 priority?: number;
 /**
 * Optional route metadata
 */
 metadata?: RouteMetadata;
 /**
 * Extra props; will be available on the client side.
 */
}
```



```
[propName: string]: unknown;
};

/**
 * Plugin authors can assign extra metadata to the created routes
 * It is only available on the Node.js side, and not sent to the browser
 * Optional: plugin authors are encouraged but not required to provide it
 *
 * Some plugins might use this data to provide additional features.
 * This is the case of the sitemap plugin to provide support for "lastmod".
 * See also: https://github.com/facebook/docusaurus/pull/9954
 */
export type RouteMetadata = {
 /**
 * The source code file path that led to the creation of the current
 route
 * In official content plugins, this is usually a Markdown or React file
 * This path is expected to be relative to the site directory
 */
 sourceFilePath?: string;
 /**
 * The last updated date of this route
 * This is generally read from the Git history of the sourceFilePath
 * but can also be provided through other means (usually front matter)
 *
 * This has notably been introduced for adding "lastmod" support to the
 * sitemap plugin, see https://github.com/facebook/docusaurus/pull/9954
 */
 lastUpdatedAt?: number;
};

type RouteModules = {
 [module: string]: Module | RouteModules | RouteModules[];
};

type Module =
 | {
 path: string;
 __import?: boolean;
 query?: ParsedUrlQueryInput;
 }
 | string;
```

### **createData(name: string, data: any): Promise<string>**

A declarative callback to create static data (generally JSON or string) which can later be provided to your routes as props. Takes the file name and data to be stored, and returns the actual data file's path.

For example, this plugin below creates a `/friends` page which displays `Your friends are: Yangshun, Sebastien`:



website/src/components/Friends.js

```
import React from 'react';

export default function FriendsComponent({friends}) {
 return <div>Your friends are {friends.join(',')}</div>;
}
```

docusaurus-friends-plugin/src/index.js

```
export default function friendsPlugin(context, options) {
 return {
 name: 'docusaurus-friends-plugin',
 async contentLoaded({content, actions}) {
 const {createData, addRoute} = actions;
 // Create friends.json
 const friends = ['Yangshun', 'Sebastien'];
 const friendsJsonPath = await createData(
 'friends.json',
 JSON.stringify(friends),
);

 // Add the '/friends' routes, and ensure it receives the friends
 props
 addRoute({
 path: '/friends',
 component: '@site/src/components/Friends.js',
 modules: {
 // propName -> JSON file path
 friends: friendsJsonPath,
 },
 exact: true,
 });
 },
 };
}
```

### **setGlobalData(data: any): void**

This function permits one to create some global plugin data that can be read from any page, including the pages created by other plugins, and your theme layout.

This data becomes accessible to your client-side/theme code through the `useGlobalData` and `usePluginData` hooks.

**! WARNING**

Global data is... global: its size affects the loading time of all pages of your site, so try to keep it small. Prefer `createData` and page-specific data whenever possible.

For example, this plugin below creates a `/friends` page which displays `Your friends are: Yangshun, Sebastien`:

website/src/components/Friends.js

```
import React from 'react';
import {usePluginData} from '@docusaurus/useGlobalData';

export default function FriendsComponent() {
 const {friends} = usePluginData('docusaurus-friends-plugin');
 return <div>Your friends are {friends.join(',')}</div>;
}
```

docusaurus-friends-plugin/src/index.js

```
export default function friendsPlugin(context, options) {
 return {
 name: 'docusaurus-friends-plugin',
 async contentLoaded({content, actions}) {
 const {setGlobalData, addRoute} = actions;
 // Create friends global data
 setGlobalData({friends: ['Yangshun', 'Sebastien']});

 // Add the '/friends' routes
 addRoute({
 path: '/friends',
 component: '@site/src/components/Friends.js',
 exact: true,
 });
 },
 };
}
```

## **configureWebpack(config, isServer, utils, content)**

Modifies the internal webpack config. If the return value is a JavaScript object, it will be merged into the final config using `webpack-merge`. If it is a function, it will be called and receive `config` as the first argument



and an `isServer` flag as the second argument.

### ⚠ WARNING

The API of `configureWebpack` will be modified in the future to accept an object (`configureWebpack({config, isServer, utils, content})`)

### config

`configureWebpack` is called with `config` generated according to client/server build. You may treat this as the base config to be merged with.

### isServer

`configureWebpack` will be called both in server build and in client build. The server build receives `true` and the client build receives `false` as `isServer`.

### utils

`configureWebpack` also receives an util object:

- `getStyleLoaders(isServer: boolean, cssOptions: {[key: string]: any}): Loader[]`
- `getJSLoader(isServer: boolean, cacheOptions?: {}): Loader | null`

You may use them to return your webpack configuration conditionally.

For example, this plugin below modify the webpack config to transpile `.foo` files.

docusaurus-plugin/src/index.js

```
export default function (context, options) {
 return {
 name: 'custom-docusaurus-plugin',
 configureWebpack(config, isServer, utils) {
 const {getJSLoader} = utils;
 return {
 module: {
 rules: [
 {
 test: /\.foo$/,
 use: [getJSLoader(isServer), 'my-custom-webpack-loader'],
 },
],
 },
 };
 },
 };
}
```



## content

`configureWebpack` will be called both with the content loaded by the plugin.

## Merge strategy

We merge the Webpack configuration parts of plugins into the global Webpack config using [webpack-merge](#).

It is possible to specify the merge strategy. For example, if you want a webpack rule to be prepended instead of appended:

docusaurus-plugin/src/index.js

```
export default function (context, options) {
 return {
 name: 'custom-docusaurus-plugin',
 configureWebpack(config, isServer, utils) {
 return {
 mergeStrategy: {'module.rules': 'prepend'},
 module: {rules: [myRuleToPrepend]},
 };
 },
 };
};
```

Read the [webpack-merge strategy doc](#) for more details.

## Configuring dev server

The dev server can be configured through returning a `devServer` field.

docusaurus-plugin/src/index.js



```
export default function (context, options) {
 return {
 name: 'custom-docusaurus-plugin',
 configureWebpack(config, isServer, utils) {
 return {
 devServer: {
 open: '/docs', // Opens localhost:3000/docs instead of
 localhost:3000/
 },
 };
 },
 };
}
```

## configurePostCss(options)

Modifies `postcssOptions` of `postcss-loader` during the generation of the client bundle.

Should return the mutated `postcssOptions`.

By default, `postcssOptions` looks like this:

```
const postcssOptions = {
 ident: 'postcss',
 plugins: [require('autoprefixer')],
};
```

Example:

docusaurus-plugin/src/index.js

```
export default function (context, options) {
 return {
 name: 'docusaurus-plugin',
 configurePostCss(postcssOptions) {
 // Appends new PostCSS plugin.
 postcssOptions.plugins.push(require('postcss-import'));
 return postcssOptions;
 },
 };
}
```





## postBuild(props)

Called when a (production) build finishes.

```
interface Props {
 siteDir: string;
 generatedFilesDir: string;
 siteConfig: DocusaurusConfig;
 outDir: string;
 baseUrl: string;
 headTags: string;
 preBodyTags: string;
 postBodyTags: string;
 routesPaths: string[];
 routesBuildMetadata: {[location: string]: {noIndex: boolean}};
 plugins: Plugin<any>[];
 content: Content;
}
```

Example:

docusaurus-plugin/src/index.js

```
export default function (context, options) {
 return {
 name: 'docusaurus-plugin',
 async postBuild({siteConfig = {}, routesPaths = [], outDir}) {
 // Print out to console all the rendered routes.
 routesPaths.map((route) => {
 console.log(route);
 });
 },
 };
}
```

## injectHtmlTags({content})

Inject head and/or body HTML tags to Docusaurus generated HTML.

`injectHtmlTags` will be called both with the content loaded by the plugin.



```
function injectHtmlTags(): {
 headTags?: HtmlTags;
 preBodyTags?: HtmlTags;
 postBodyTags?: HtmlTags;
};

type HtmlTags = string | HtmlTagObject | (string | HtmlTagObject)[];

type HtmlTagObject = {
 /**
 * Attributes of the HTML tag
 * E.g. `{ 'disabled': true, 'value': 'demo', 'rel': 'preconnect' }`
 */
 attributes?: {
 [attributeName: string]: string | boolean;
 };
 /**
 * The tag name e.g. `div`, `script`, `link`, `meta`
 */
 tagName: string;
 /**
 * The inner HTML
 */
 innerHTML?: string;
};
```

Example:

```
docusaurus-plugin/src/index.js
```



```
export default function (context, options) {
 return {
 name: 'docusaurus-plugin',
 loadContent: async () => {
 return {remoteHeadTags: await fetchHeadTagsFromAPI()};
 },
 injectHtmlTags({content}) {
 return {
 headTags: [
 {
 tagName: 'link',
 attributes: {
 rel: 'preconnect',
 href: 'https://www.github.com',
 },
 },
 ...content.remoteHeadTags,
],
 preBodyTags: [
 {
 tagName: 'script',
 attributes: {
 charset: 'utf-8',
 src: '/noflash.js',
 },
 },
],
 postBodyTags: [`<div> This is post body </div>`],
 };
 },
 };
}
```

Tags will be added as follows:

- `headTags` will be inserted before the closing `</head>` tag after scripts added by config.
- `preBodyTags` will be inserted after the opening `<body>` tag before any child elements.
- `postBodyTags` will be inserted before the closing `</body>` tag after all child elements.

## getClientModules()

Returns an array of paths to the `client modules` that are to be imported into the client bundle.

As an example, to make your theme load a `customCss` or `customJs` file path from `options` passed in by the user:



my-theme/src/index.js

```
export default function (context, options) {
 const {customCss, customJs} = options || {};
 return {
 name: 'name-of-my-theme',
 getClientModules() {
 return [customCss, customJs];
 },
 };
}
```

## 2.4.5 Extending infrastructure

Docusaurus has some infrastructure like hot reloading, CLI, and swizzling, that can be extended by external plugins.

### getPathsToWatch()

Specifies the paths to watch for plugins and themes. The paths are watched by the dev server so that the plugin lifecycles are reloaded when contents in the watched paths change. Note that the plugins and themes modules are initially called with `context` and `options` from Node, which you may use to find the necessary directory information about the site.

Use this for files that are consumed server-side, because theme files are automatically watched by Webpack dev server.

Example:

docusaurus-plugin/src/index.js

```
import path from 'path';

export default function (context, options) {
 return {
 name: 'docusaurus-plugin',
 getPathsToWatch() {
 const contentPath = path.resolve(context.siteDir, options.path);
 return [`${contentPath}/**/*.${ts,tsx}`];
 },
 };
}
```

### extendCli(cli)



Register an extra command to enhance the CLI of Docusaurus. `cli` is a [commander](#) object.

### ⚠ WARNING

The commander version matters! We use commander v5, and make sure you are referring to the right version documentation for available APIs.

Example:

docusaurus-plugin/src/index.js

```
export default function (context, options) {
 return {
 name: 'docusaurus-plugin',
 extendCli(cli) {
 cli
 .command('roll')
 .description('Roll a random number between 1 and 1000')
 .action(() => {
 console.log(Math.floor(Math.random() * 1000 + 1));
 });
 },
 };
}
```

## getThemePath()

Returns the path to the directory where the theme components can be found. When your users call `swizzle`, `getThemePath` is called and its returned path is used to find your theme components. Relative paths are resolved against the folder containing the entry point.

For example, your `getThemePath` can be:

my-theme/src/index.js

```
export default function (context, options) {
 return {
 name: 'my-theme',
 getThemePath() {
 return './theme';
 },
 };
}
```



## getTypeScriptThemePath()

Similar to `getThemePath()`, it should return the path to the directory where the source code of TypeScript theme components can be found. This path is purely for swizzling TypeScript theme components, and theme components under this path will **not** be resolved by Webpack. Therefore, it is not a replacement for `getThemePath()`. Typically, you can make the path returned by `getTypeScriptThemePath()` be your source directory, and make the path returned by `getThemePath()` be the compiled JavaScript output.

### TIP

For TypeScript theme authors: you are strongly advised to make your compiled output as human-readable as possible. Only strip type annotations and don't transpile any syntaxes, because they will be handled by Webpack's Babel loader based on the targeted browser versions.

You should also format these files with Prettier. Remember—JS files can and will be directly consumed by your users.

Example:

my-theme/src/index.js

```
export default function (context, options) {
 return {
 name: 'my-theme',
 getThemePath() {
 // Where compiled JavaScript output lives
 return '../lib/theme';
 },
 getTypeScriptThemePath() {
 // Where TypeScript source code lives
 return '../src/theme';
 },
 };
}
```

## getSwizzleComponentList()

**This is a static method, not attached to any plugin instance.**

Returns a list of stable components that are considered safe for swizzling. These components will be swizzable without `--danger`. All components are considered unstable by default. If an empty array is returned, all components are considered unstable. If `undefined` is returned, all components are considered stable.



my-theme/src/index.js

```
export function getSwizzleComponentList() {
 return [
 'CodeBlock',
 'DocSidebar',
 'Footer',
 'NotFound',
 'SearchBar',
 'hooks/useTheme',
 'prism-include-languages',
];
}
```

## 2.4.6 I18n lifecycles

Plugins use these lifecycles to load i18n-related data.

### **getTranslationFiles({content})**

Plugins declare the JSON translation files they want to use.

Returns translation files `{path: string, content: ChromeI18nJSON}`:

- `path`: relative to the plugin localized folder `i18n/[locale]/[pluginName]`. Extension `.json` should be omitted to remain generic.
- `content`: using the Chrome i18n JSON format.

These files will be written by the `write-translations` CLI to the plugin i18n subfolder, and will be read in the appropriate locale before calling `translateContent()` and `translateThemeConfig()`

Example:

my-plugin.js



```
export default function (context, options) {
 return {
 name: 'my-plugin',
 async getTranslationFiles({content}) {
 return [
 {
 path: 'sidebar-labels',
 content: {
 someSidebarLabel: {
 message: 'Some Sidebar Label',
 description: 'A label used in my plugin in the sidebar',
 },
 someLabelFromContent: content.myLabel,
 },
 },
];
 },
 };
}
```

## **translateContent({content, translationFiles})**

Translate the plugin content, using the localized translation files.

Returns the localized plugin content.

The `contentLoaded()` lifecycle will be called with the localized plugin content returned by `translateContent()`.

Example:

```
my-plugin.js
```





```
export default function (context, options) {
 return {
 name: 'my-plugin',
 translateContent({content, translationFiles}) {
 const myTranslationFile = translationFiles.find(
 (f) => f.path === 'myTranslationFile',
);
 return {
 ...content,
 someContentLabel: myTranslationFile.someContentLabel.message,
 };
 },
 };
}
```

## translateThemeConfig({themeConfig, translationFiles})

Translate the site `themeConfig` labels, using the localized translation files.

Returns the localized `themeConfig`.

Example:

my-plugin.js

```
export default function (context, options) {
 return {
 name: 'my-theme',
 translateThemeConfig({themeConfig, translationFiles}) {
 const myTranslationFile = translationFiles.find(
 (f) => f.path === 'myTranslationFile',
);
 return {
 ...themeConfig,
 someThemeConfigLabel:
 myTranslationFile.someThemeConfigLabel.message,
 };
 },
 };
}
```

## async getDefaultCodeTranslationMessages()



Themes using the `<Translate>` API can provide default code translation messages.

It should return messages in `Record<string, string>`, where keys are translation IDs and values are messages (without the description) localized using the site's current locale.

Example:

my-plugin.js

```
export default function (context, options) {
 return {
 name: 'my-theme',
 async getDefaultCodeTranslationMessages() {
 return readJsonFile(`${context.i18n.currentLocale}.json`);
 },
 };
}
```

## 2.4.7 Static methods

Static methods are not part of the plugin instance—they are attached to the constructor function. These methods are used to validate and normalize the plugin options and theme config, which are then used as constructor parameters to initialize the plugin instance.

### **validateOptions({options, validate})**

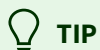
Returns validated and normalized options for the plugin. This method is called before the plugin is initialized. You must return the options since they will be passed to the plugin during initialization.

#### **options**

`validateOptions` is called with `options` passed to plugin for validation and normalization.

#### **validate**

`validateOptions` is called with `validate` function which takes a **Joi** schema and options as the arguments, returns validated and normalized options. `validate` will automatically handle error and validation config.



**TIP**

**Joi** is recommended for validation and normalization of options.

To avoid mixing Joi versions, use `import {Joi} from '@docusaurus/utils-validation'`

If you don't use **Joi** for validation you can throw an Error in case of invalid options and return options in case of success.



my-plugin/src/index.js

```
export default function myPlugin(context, options) {
 return {
 name: 'docusaurus-plugin',
 // rest of methods
 };
}

export function validateOptions({options, validate}) {
 const validatedOptions = validate(myValidationSchema, options);
 return validatedOptions;
}
```

## validateThemeConfig({themeConfig, validate})

Return validated and normalized configuration for the theme.

### themeConfig

`validateThemeConfig` is called with `themeConfig` provided in `docusaurus.config.js` for validation and normalization.

### validate

`validateThemeConfig` is called with `validate` function which takes a **Joi** schema and `themeConfig` as the arguments, returns validated and normalized options. `validate` will automatically handle error and validation config.



**TIP**

**Joi** is recommended for validation and normalization of theme config.

To avoid mixing Joi versions, use `import {Joi} from '@docusaurus/utils-validation'`

If you don't use **Joi** for validation you can throw an Error in case of invalid options.

my-theme/src/index.js



```
export default function myPlugin(context, options) {
 return {
 name: 'docusaurus-plugin',
 // rest of methods
 };
}

export function validateThemeConfig({themeConfig, validate}) {
 const validatedThemeConfig = validate(myValidationSchema, options);
 return validatedThemeConfig;
}
```

## 2.5 Docusaurus plugins

We provide official Docusaurus plugins.

### 2.5.1 Content plugins

These plugins are responsible for loading your site's content, and creating pages for your theme to render.

- [@docusaurus/plugin-content-docs](#)
- [@docusaurus/plugin-content-blog](#)
- [@docusaurus/plugin-content-pages](#)

### 2.5.2 Behavior plugins

These plugins will add a useful behavior to your Docusaurus site.

- [@docusaurus/plugin-debug](#)
- [@docusaurus/plugin-sitemap](#)
- [@docusaurus/plugin-svgr](#)
- [@docusaurus/plugin-rsdoctor](#)
- [@docusaurus/plugin-pwa](#)
- [@docusaurus/plugin-client-redirects](#)
- [@docusaurus/plugin-ideal-image](#)
- [@docusaurus/plugin-google-gtag](#)
- [@docusaurus/plugin-google-tag-manager](#)
- [@docusaurus/plugin-css-cascade-layers](#)

## 2.6 Plugins

### 2.6.1 plugin-content-docs

Provides the [Docs](#) functionality and is the default docs plugin for Docusaurus.

## Installation



npm

Yarn

pnpm

Bun

```
npm install --save @docusaurus/plugin-content-docs
```

**TIP**

If you use the preset `@docusaurus/preset-classic`, you don't need to install this plugin as a dependency.

You can configure this plugin through the [preset options](#).

## Configuration

Accepted fields:

Name	Type	Default	Description
path	string	'docs'	Path to the docs content directory on the file system, relative to site directory.
editUrl	string   <a href="#">EditUrlFunction</a>	undefined	Base URL to edit your site. The final URL is computed by <code>editUrl + relativeDocPath</code> . Using a function allows more nuanced control for each file. Omitting this variable entirely will disable edit links.
editLocalizedFiles	boolean	false	The edit URL will target the localized file, instead of the original unlocalized file. Ignored when <code>editUrl</code> is a function.
editCurrentVersion	boolean	false	The edit URL will always target the current version doc instead of older versions. Ignored when <code>editUrl</code> is a function.
routeBasePath	string	'docs'	URL route for the docs section of your site. <b>DO NOT</b> include a trailing slash. Use <code>/</code> for shipping docs without base path.
tagsBasePath	string	'tags'	URL route for the tags list page of your site. It is prepended to the <code>routeBasePath</code> .
include	string[]	['**/*.md', '**/*.mdx']	Array of glob patterns matching Markdown files to be built, relative to the content path.
exclude	string[]	See <i>example configuration</i>	Array of glob patterns matching Markdown files to be excluded. Serves as refinement based on the <code>include</code> option.
sidebarPath	false   string	undefined	Path to a sidebars configuration file, loaded in a Node.js context. Use <code>false</code>



Name	Type	Default	Description
			to disable sidebars, or <code>undefined</code> to create a fully autogenerated sidebar.
<code>sidebarCollapsible</code>	<code>boolean</code>	<code>true</code>	Whether sidebar categories are collapsible by default. See also <a href="#">Collapsible categories</a>
<code>sidebarCollapsed</code>	<code>boolean</code>	<code>true</code>	Whether sidebar categories are collapsed by default. See also <a href="#">Expanded categories by default</a>
<code>sidebarItemsGenerator</code>	<a href="#">SidebarGenerator</a>	<i>Omitted</i>	Function used to replace the sidebar items of type <code>'autogenerated'</code> with real sidebar items (docs, categories, links...). See also <a href="#">Customize the sidebar items generator</a>
<code>numberPrefixParser</code>	<code>boolean</code>   <a href="#">PrefixParser</a>	<i>Omitted</i>	Custom parsing logic to extract number prefixes from file names. Use <code>false</code> to disable this behavior and leave the docs untouched, and <code>true</code> to use the default parser. See also <a href="#">Using number prefixes</a>
<code>docsRootComponent</code>	<code>string</code>	<code>'@theme/DocsRoot'</code>	Parent component of all the docs plugin pages (including all versions). Stays mounted when navigation between docs pages and versions.
<code>docVersionRootComponent</code>	<code>string</code>	<code>'@theme/DocVersionLayout'</code>	Parent component of all docs pages of an individual version (doc pages with sidebars, tags pages). Stays mounted when navigation between pages of that specific version.
<code>docRootComponent</code>	<code>string</code>	<code>'@theme/DocRoot'</code>	Parent component of all doc pages with sidebars (regular docs pages, category generated index pages). Stays mounted when navigation between such pages.
<code>docItemComponent</code>	<code>string</code>	<code>'@theme/DocItem'</code>	Main doc container, with TOC, pagination, etc.
<code>docTagsListComponent</code>	<code>string</code>	<code>'@theme/DocTagsListPage'</code>	Root component of the tags list page
<code>docTagDocListComponent</code>	<code>string</code>	<code>'@theme/DocTagDocListPage'</code>	Root component of the "docs containing tag X" page.
<code>docCategoryGeneratedIndexComponent</code>	<code>string</code>	<code>'@theme/DocCategoryGeneratedIndexPage'</code>	Root component of the generated category index page.
<code>remarkPlugins</code>	<code>any[]</code>	<code>[]</code>	Remark plugins passed to MDX.
<code>rehypePlugins</code>	<code>any[]</code>	<code>[]</code>	Rehype plugins passed to MDX.
<code>recmaPlugins</code>	<code>any[]</code>	<code>[]</code>	Recma plugins passed to MDX.
<code>beforeDefaultRemarkPlugins</code>	<code>any[]</code>	<code>[]</code>	Custom Remark plugins passed to MDX before the default Docusaurus Remark plugins.
<code>beforeDefaultRehypePlugins</code>	<code>any[]</code>	<code>[]</code>	Custom Rehype plugins passed to MDX before the default Docusaurus Rehype plugins.
<code>showLastUpdateAuthor</code>	<code>boolean</code>	<code>false</code>	Whether to display the author who last updated the doc.



Name	Type	Default	Description
showLastUpdateTime	boolean	false	<b>Only for Markdown pages.</b> Whether to display the last date the doc was updated. This requires access to git history during the build, so will not work correctly with shallow clones (a common default for CI systems). With GitHub actions/checkout, use fetch-depth: 0. When deploying to Vercel, set the environment variable VERCEL_DEEP_CLONE=true.
breadcrumbs	boolean	true	Enable or disable the breadcrumbs on doc pages.
disableVersioning	boolean	false	Explicitly disable versioning even when multiple versions exist. This will make the site only include the current version. Will error if includeCurrentVersion: false and disableVersioning: true.
includeCurrentVersion	boolean	true	Include the current version of your docs.
lastVersion	string	First version in versions.json	The version navigated to in priority and displayed by default for docs navbar items.
onlyIncludeVersions	string[]	All versions available	Only include a subset of all available versions.
versions	VersionsConfig	{}	Independent customization of each version's properties.
tags	string   false   null   undefined	tags.yml	Path to a YAML file listing pre-defined tags. Relative to the docs version content directories.
onInlineTags	'ignore'   'log'   'warn'   'throw'	warn	The plugin behavior when docs contain inline tags (not appearing in the list of pre-defined tags, usually docs/tags.yml).

## Types

### EditUrlFunction

```
type EditUrlFunction = (params: {
 version: string;
 versionDocsDirPath: string;
 docPath: string;
 permalink: string;
 locale: string;
}) => string | undefined;
```

### PrefixParser



```
type PrefixParser = (filename: string) => {
 filename: string;
 numberPrefix?: number;
};
```

## SidebarGenerator





```
type SidebarGenerator = (generatorArgs: {
 /** The sidebar item with type "autogenerated" to be transformed. */
 item: {type: 'autogenerated'; dirName: string};
 /** Useful metadata for the version this sidebar belongs to. */
 version: {contentPath: string; versionName: string};
 /** All the docs of that version (unfiltered). */
 docs: {
 id: string;
 title: string;
 frontMatter: DocFrontMatter & Record<string, unknown>;
 source: string;
 sourceDirName: string;
 sidebarPosition?: number | undefined;
 }[];
 /** Number prefix parser configured for this plugin. */
 numberPrefixParser: PrefixParser;
 /** The default category index matcher which you can override. */
 isCategoryIndex: CategoryIndexMatcher;
 /**
 * key is the path relative to the doc content directory, value is the
 * category metadata file's content.
 */
 categoriesMetadata: {[filePath: string]: CategoryMetadata};
 /**
 * Useful to re-use/enhance the default sidebar generation logic from
 * Docusaurus.
 */
 defaultSidebarItemsGenerator: SidebarGenerator;
 // Returns an array of sidebar items – same as what you can declare in
 // sidebars.js, except for shorthands. See
 // https://docusaurus.io/docs/sidebar/items
}) => Promise<SidebarItem[]>;

type CategoryIndexMatcher = (param: {
 /** The file name, without extension */
 fileName: string;
 /**
 * The list of directories, from lowest level to highest.
 * If there's no dir name, directories is ['.']
 */
 directories: string[];
 /** The extension, with a leading dot */
 extension: string;
}) => boolean;
```

## VersionsConfig



```
type VersionConfig = {
 /**
 * The base path of the version, will be appended to `baseUrl` +
 * `routeBasePath`.
 */
 path?: string;
 /** The label of the version to be used in badges, dropdowns, etc. */
 label?: string;
 /** The banner to show at the top of a doc of that version. */
 banner?: 'none' | 'unreleased' | 'unmaintained';
 /** Show a badge with the version label at the top of each doc. */
 badge?: boolean;
 /** Prevents search engines from indexing this version */
 noIndex?: boolean;
 /** Add a custom class name to the <html> element of each doc */
 className?: string;
};

type VersionsConfig = {[versionName: string]: VersionConfig};
```

## Example configuration

You can configure this plugin through preset options or plugin options.



### TIP

Most Docusaurus users configure this plugin through the preset options.

### Preset options

### Plugin options

If you use a preset, configure this plugin through the [preset options](#):

```
docusaurus.config.js
```



```
module.exports = {
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 docs: {
 path: 'docs',
 breadcrumbs: true,
 // Simple use-case: string editUrl
 // editUrl:
 'https://github.com/facebook/docusaurus/edit/main/website/',
 // Advanced use-case: functional editUrl
 editUrl: ({versionDocsDirPath, docPath}) =>
 `https://github.com/facebook/docusaurus/edit/main/website/${versionDocsDirPath}/${docPath}`,
 editLocalizedFiles: false,
 editCurrentVersion: false,
 routeBasePath: 'docs',
 include: ['**/*.md', '**/*.mdx'],
 exclude: [
 '**/_*. {js,jsx,ts,tsx,md,mdx}',
 '**/_*/**',
 '**/*.test.{js,jsx,ts,tsx}',
 '**/__tests__/**',
],
 sidebarPath: 'sidebars.js',
 async sidebarItemsGenerator({
 defaultSidebarItemsGenerator,
 numberPrefixParser,
 item,
 version,
 docs,
 isCategoryIndex,
 }) {
 // Use the provided data to generate a custom sidebar slice
 return [
 {type: 'doc', id: 'intro'},
 {
 type: 'category',
 label: 'Tutorials',
 items: [
 {type: 'doc', id: 'tutorial1'},
 {type: 'doc', id: 'tutorial2'},
],
 },
];
 },
 numberPrefixParser(filename) {
```



```
// Implement your own logic to extract a potential number
prefix

const numberPrefix = findNumberPrefix(filename);
// Prefix found: return it with the cleaned filename
if (numberPrefix) {
 return {
 numberPrefix,
 filename: filename.replace(prefix, ''),
 };
}

// No number prefix found
return {numberPrefix: undefined, filename};
},
docsRootComponent: '@theme/DocsRoot',
docVersionRootComponent: '@theme/DocVersionRoot',
docRootComponent: '@theme/DocRoot',
docItemComponent: '@theme/DocItem',
remarkPlugins: [require('./my-remark-plugin')],
rehypePlugins: [],
beforeDefaultRemarkPlugins: [],
beforeDefaultRehypePlugins: [],
showLastUpdateAuthor: false,
showLastUpdateTime: false,
disableVersioning: false,
includeCurrentVersion: true,
lastVersion: undefined,
versions: {
 current: {
 label: 'Android SDK v2.0.0 (WIP)',
 path: 'android-2.0.0',
 banner: 'none',
 },
 '1.0.0': {
 label: 'Android SDK v1.0.0',
 path: 'android-1.0.0',
 banner: 'unmaintained',
 },
},
onlyIncludeVersions: ['current', '1.0.0', '2.0.0'],
},
],
],
};
```

## Markdown front matter

Markdown documents can use the following Markdown [front matter](#) metadata fields, enclosed by a line `---` on either side.



Accepted fields:

Name	Type	Default	Description
id	string	file path (including folders, without the extension)	A unique document ID.
title	string	Markdown title or id	The text title of your document. Used for the page metadata and as a fallback value in multiple places (sidebar, next/previous buttons...). Automatically added at the top of your doc if it does not contain any Markdown title.
pagination_label	string	sidebar_label or title	The text used in the document next/previous buttons for this document.
sidebar_label	string	title	The text shown in the document sidebar for this document.
sidebar_position	number	Default ordering	Controls the position of a doc inside the generated sidebar slice when using autogenerated sidebar items. See also <a href="#">Autogenerated sidebar metadata</a> .
sidebar_class_name	string	undefined	Gives the corresponding sidebar label a special class name when using autogenerated sidebars.
sidebar_key	string	undefined	Gives the corresponding sidebar item a key to uniquely identify the sidebar item. This is mostly useful for i18n sites to generate unique translation keys for categories sharing the same labels. An error will tell you when this extra metadata is needed.
sidebar_custom_props	object	undefined	Assign <a href="#">custom props</a> to the sidebar item referencing this doc
displayed_sidebar	string	undefined	Force the display of a given sidebar when browsing the current document. Read the <a href="#">multiple sidebars guide</a> for details.
hide_title	boolean	false	Whether to hide the title at the top of the doc. It only hides a title declared through the front matter, and have no effect on a Markdown title at the top of your document.
hide_table_of_contents	boolean	false	Whether to hide the table of contents to the right.
toc_min_heading_level	number	2	The minimum heading level shown in the table of contents. Must be between 2 and 6 and lower or equal to the max value.
toc_max_heading_level	number	3	The max heading level shown in the table of contents. Must be between 2 and 6.
pagination_next	string   null	Next doc in the sidebar	The ID of the documentation you want the "Next" pagination to link to. Use <code>null</code> to disable showing "Next" for this page.
pagination_previous	string   null	Previous doc in the sidebar	The ID of the documentation you want the "Previous" pagination to link to. Use <code>null</code> to disable showing "Previous" for this page.
parse_number_prefixes	boolean	numberPrefix-Parser plugin option	Whether number prefix parsing is disabled on this doc. See also <a href="#">Using number prefixes</a> .



Name	Type	Default	Description
custom_edit_url	string   null	Computed using the editUrl plugin option	The URL for editing this document. Use null to disable showing "Edit this page" for this page.
keywords	string[]	undefined	Keywords meta tag for the document page, for search engines.
description	string	The first line of Markdown content	The description of your document, which will become the <meta name="description" content="..." /> and <meta property="og:description" content="..." /> in <head>, used by search engines.
image	string	undefined	Cover or thumbnail image that will be used as the <meta property="og:image" content="..." /> in the <head>, enhancing link previews on social media and messaging platforms.
slug	string	File path	Allows to customize the document URL (/<routeBasePath>/<slug>). Supports multiple patterns: slug: my-doc, slug: /my/path/myDoc, slug: /.
tags	Tag[]	undefined	A list of strings or objects of two string fields label and permalink to tag to your docs. Strings can be a reference to keys of a tags file (usually tags.yml)
draft	boolean	false	Draft documents will only be available during development.
unlisted	boolean	false	Unlisted documents will be available in both development and production. They will be "hidden" in production, not indexed, excluded from sitemaps, and can only be accessed by users having a direct link.
last_update	FrontMatterLastUpdate	undefined	Allows overriding the last update author/date. Date can be any parsable date string.

```
type FrontMatterLastUpdate = {date?: string; author?: string};
```

```
type Tag = string | {label: string; permalink: string};
```

Example:



```

id: doc-markdown
title: Docs Markdown Features
hide_title: false
hide_table_of_contents: false
sidebar_label: Markdown
sidebar_position: 3
pagination_label: Markdown features
custom_edit_url: https://github.com/facebook/docusaurus/edit/main/docs/api-
doc-markdown.md
description: How do I find you when I cannot solve this problem
keywords:
 - docs
 - docusaurus
tags: [docusaurus]
image: https://i.imgur.com/mErPwqL.png
slug: /myDoc
last_update:
 date: 1/1/2000
 author: custom author name

Markdown Features

My Document Markdown content
```

## Tags File

Use the `tags` plugin option to configure the path of a YAML tags file.

By convention, the plugin will look for a `tags.yml` file at the root of your content folder(s).

This file can contain a list of predefined tags. You can reference these tags by their keys in Markdown files thanks to the `tags` front matter.



### KEEPING TAGS CONSISTENT

Using a tags file, you can ensure that your tags usage is consistent across your plugin content set. Use the `onInlineTags: 'throw'` plugin option to enforce this consistency and prevent usage of inline tags declared on the fly.

## Types

The YAML content of the provided tags file should respect the following shape:



```
type Tag = {
 label?: string; // Tag display label
 permalink?: string; // Tag URL pathname segment
 description?: string; // Tag description displayed in the tag page
};

type TagsFileInput = Record<string, Partial<Tag> | null>;
```

## Example

tags.yml

```
releases:
 label: 'Product releases'
 permalink: '/product-releases'
 description: 'Content related to product releases.'

A partial tag definition is also valid
announcements:
 label: 'Announcements'

An empty tag definition is also valid
Other attributes will be inferred from the key
emptyTag:
```

content.md

```

tags: [releases, announcements, emptyTag]

Title

Content
```

## i18n

Read the [i18n introduction](#) first.

### Translation files location

- **Base path:** `website/i18n/[locale]/docusaurus-plugin-content-docs`





- **Multi-instance path:** `website/i18n/[locale]/docusaurus-plugin-content-docs-[pluginId]`
- **JSON files:** extracted with `docusaurus write-translations`
- **Markdown files:** `website/i18n/[locale]/docusaurus-plugin-content-docs/[versionName]`

## Example file-system structure

```
website/i18n/[locale]/docusaurus-plugin-content-docs
├── # translations for website/docs
│ ├── current
│ │ ├── api
│ │ │ └── config.md
│ │ └── getting-started.md
│ └── current.json
├── # translations for website/versioned_docs/version-1.0.0
│ ├── version-1.0.0
│ │ ├── api
│ │ │ └── config.md
│ │ └── getting-started.md
│ └── version-1.0.0.json
```

## 2.6.2 plugin-content-blog

Provides the [Blog](#) feature and is the default blog plugin for Docusaurus.

### SOME FEATURES PRODUCTION ONLY

The [feed feature](#) works by extracting the build output, and is **only active in production**.

## Installation

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)

```
npm install --save @docusaurus/plugin-content-blog
```

**TIP**



If you use the preset `@docusaurus/preset-classic`, you don't need to install this plugin as a dependency.

You can configure this plugin through the [preset options](#).

## Configuration

Accepted fields:

Name	Type	Default	Description
<code>path</code>	<code>string</code>	<code>'blog'</code>	Path to the blog content directory on the file system, relative to site dir.
<code>editUrl</code>	<code>string</code>   <code>EditUrlFn</code>	<code>undefined</code>	Base URL to edit your site. The final URL is computed by <code>editUrl</code> + <code>relativePostPath</code> . Using a function allows more nuanced control for each file. Omitting this variable entirely will disable edit links.
<code>editLocalizedFiles</code>	<code>boolean</code>	<code>false</code>	The edit URL will target the localized file, instead of the original unlocalized file. Ignored when <code>editUrl</code> is a function.
<code>blogTitle</code>	<code>string</code>	<code>'Blog'</code>	Blog page title for better SEO.
<code>blogDescription</code>	<code>string</code>	<code>'Blog'</code>	Blog page meta description for better SEO.
<code>blogSidebarCount</code>	<code>number</code>   <code>'ALL'</code>	<code>5</code>	Number of blog post elements to show in the blog sidebar. <code>'ALL'</code> to show all blog posts; <code>0</code> to disable.
<code>blogSidebarTitle</code>	<code>string</code>	<code>'Recent posts'</code>	Title of the blog sidebar.
<code>routeBasePath</code>	<code>string</code>	<code>'blog'</code>	URL route for the blog section of your site. <b>DO NOT</b> include a trailing slash. Use <code>/</code> to put the blog at root path.
<code>tagsBasePath</code>	<code>string</code>	<code>'tags'</code>	URL route for the tags section of your blog. Will be appended to <code>routeBasePath</code> .
<code>pageBasePath</code>	<code>string</code>	<code>'page'</code>	URL route for the pages section of your blog. Will be appended to <code>routeBasePath</code> .
<code>archiveBasePath</code>	<code>string</code>   <code>null</code>	<code>'archive'</code>	URL route for the archive section of your blog. Will be appended to <code>routeBasePath</code> . <b>DO NOT</b> include a trailing slash. Use <code>null</code> to disable generation of archive.
<code>authorsBasePath</code>	<code>string</code>	<code>'authors'</code>	URL route for the authors pages of your blog. Will be appended to <code>path</code> .
<code>include</code>	<code>string[]</code>	<code>['**/*.md', '**/*.mdx']</code>	Array of glob patterns matching Markdown files to be built, relative to the content path.
<code>exclude</code>	<code>string[]</code>	<i>See example configuration</i>	Array of glob patterns matching Markdown files to be excluded.



Name	Type	Default	Description
			Serves as refinement based on the <code>include</code> option.
<code>postsPerPage</code>	<code>number</code>   <code>'ALL'</code>	<code>10</code>	Number of posts to show per page in the listing page. Use <code>'ALL'</code> to display all posts on one listing page.
<code>blogListComponent</code>	<code>string</code>	<code>'@theme/BlogListPage'</code>	Root component of the blog listing page.
<code>blogPostComponent</code>	<code>string</code>	<code>'@theme/BlogPostPage'</code>	Root component of each blog post page.
<code>blogTagsListComponent</code>	<code>string</code>	<code>'@theme/BlogTagsListPage'</code>	Root component of the tags list page.
<code>blogTagsPostsComponent</code>	<code>string</code>	<code>'@theme/BlogTagsPostsPage'</code>	Root component of the "posts containing tag" page.
<code>blogArchiveComponent</code>	<code>string</code>	<code>'@theme/BlogArchivePage'</code>	Root component of the blog archive page.
<code>blogAuthorsPostsComponent</code>	<code>string</code>	<code>'@theme/Blog/Pages/BlogAuthorsPostsPage'</code>	Root component of the blog author page.
<code>blogAuthorsListComponent</code>	<code>string</code>	<code>'@theme/Blog/Pages/BlogAuthorsListPage'</code>	Root component of the blog authors page index.
<code>remarkPlugins</code>	<code>any[]</code>	<code>[]</code>	Remark plugins passed to MDX.
<code>rehypePlugins</code>	<code>any[]</code>	<code>[]</code>	Rehype plugins passed to MDX.
<code>recmaPlugins</code>	<code>any[]</code>	<code>[]</code>	Recma plugins passed to MDX.
<code>beforeDefaultRemarkPlugins</code>	<code>any[]</code>	<code>[]</code>	Custom Remark plugins passed to MDX before the default Docusaurus Remark plugins.
<code>beforeDefaultRehypePlugins</code>	<code>any[]</code>	<code>[]</code>	Custom Rehype plugins passed to MDX before the default Docusaurus Rehype plugins.
<code>truncateMarker</code>	<code>RegExp</code>	<code>/&lt;!--\s*truncate\s*--&gt;/   {\s*\s*\s*\s*\s*truncate\s*\s*\s*\s*}/</code>	Truncate marker marking where the summary ends.
<code>showReadingTime</code>	<code>boolean</code>	<code>true</code>	Show estimated reading time for the blog post.
<code>readingTime</code>	<code>ReadingTimeFn</code>	The default reading time	A callback to customize the reading time number displayed.
<code>authorsMapPath</code>	<code>string</code>	<code>'authors.yml'</code>	Path to the authors map file, relative to the blog content directory.
<code>feedOptions</code>	See below	<code>{type: ['rss', 'atom']}</code>	Blog feed.
<code>feedOptions.type</code>	<code>FeedType</code>   <code>FeedType[]</code>   <code>'all'</code>   <code>null</code>	<b>Required</b>	Type of feed to be generated. Use <code>null</code> to disable generation.
<code>feedOptions.createFeedItems</code>	<code>CreateFeedItemsFn</code>   <code>undefined</code>	<code>undefined</code>	An optional function which can be used to transform and / or filter the items in the feed.



Name	Type	Default	Description
<code>feedOptions.limit</code>	<code>number   null   false</code>	<code>20</code>	Limits the feed to the specified number of posts, <code>false</code> or <code>null</code> for all entries. Defaults to <code>20</code> .
<code>feedOptions.title</code>	<code>string</code>	<code>siteConfig.title</code>	Title of the feed.
<code>feedOptions.description</code>	<code>string</code>	<code>`\${siteConfig.title} Blog`</code>	Description of the feed.
<code>feedOptions.copyright</code>	<code>string</code>	<code>undefined</code>	Copyright message.
<code>feedOptions.xslt</code>	<code>boolean   FeedXSLTOptions</code>	<code>undefined</code>	Permits to style the blog XML feeds with XSLT so that browsers render them nicely.
<code>feedOptions.language</code>	<code>string</code> (See <a href="#">documentation</a> for possible values)	<code>undefined</code>	Language metadata of the feed.
<code>sortPosts</code>	<code>'descending'   'ascending'</code>	<code>'descending'</code>	Governs the direction of blog post sorting.
<code>processBlogPosts</code>	<code>ProcessBlogPostsFn</code>	<code>undefined</code>	An optional function which can be used to transform blog posts (filter, modify, delete, etc...).
<code>showLastUpdateAuthor</code>	<code>boolean</code>	<code>false</code>	Whether to display the author who last updated the blog post.
<code>showLastUpdateTime</code>	<code>boolean</code>	<code>false</code>	Whether to display the last date the blog post was updated. This requires access to git history during the build, so will not work correctly with shallow clones (a common default for CI systems). With GitHub <code>actions/checkout</code> , use <code>fetch-depth: 0</code> . When deploying to Vercel, set the environment variable <code>VERCEL_DEEP_CLONE=true</code> .
<code>tags</code>	<code>string   false   null   undefined</code>	<code>tags.yml</code>	Path to the YAML tags file listing pre-defined tags. Relative to the blog content directory.
<code>onInlineTags</code>	<code>'ignore'   'log'   'warn'   'throw'</code>	<code>warn</code>	The plugin behavior when blog posts contain inline tags (not appearing in the list of pre-defined tags, usually <code>tags.yml</code> ).
<code>onUntruncatedBlogPosts</code>	<code>'ignore'   'log'   'warn'   'throw'</code>	<code>warn</code>	The plugin behavior when blog posts do not contain a truncate marker.

## Types

`EditUrlFn`



```
type EditUrlFunction = (params: {
 blogDirPath: string;
 blogPath: string;
 permalink: string;
 locale: string;
}) => string | undefined;
```

## ReadingTimeFn

```
type ReadingTimeOptions = {
 wordsPerMinute: number;
};

type ReadingTimeCalculator = (params: {
 content: string;
 locale: string;
 frontMatter?: BlogPostFrontMatter & Record<string, unknown>;
 options?: ReadingTimeOptions;
}) => number;

type ReadingTimeFn = (params: {
 content: string;
 locale: string;
 frontMatter: BlogPostFrontMatter & Record<string, unknown>;
 defaultReadingTime: ReadingTimeCalculator;
}) => number | undefined;
```

## FeedType

```
type FeedType = 'rss' | 'atom' | 'json';
```

## FeedXSLTOptions

Permits to style the blog XML feeds so that browsers render them nicely with [XSLT](#).

- Use `true` to let the blog use its built-in `.xsl` and `.css` files to style the blog feed
- Use a falsy value (`undefined` | `null` | `false`) to disable the feature
- Use a `string` to provide a file path to a custom `.xsl` file relative to the blog content folder. By convention, you must provide a `.css` file with the exact same name.



```
type FeedXSLTOptions =
 | boolean
 | undefined
 | null
 | {
 rss?: string | boolean | null | undefined;
 atom?: string | boolean | null | undefined;
 };
```

### CreateFeedItemsFn

```
type CreateFeedItemsFn = (params: {
 blogPosts: BlogPost[];
 siteConfig: DocusaurusConfig;
 outDir: string;
 defaultCreateFeedItemsFn: CreateFeedItemsFn;
}) => Promise<BlogFeedItem[]>;
```

### ProcessBlogPostsFn

```
type ProcessBlogPostsFn = (params: {
 blogPosts: BlogPost[];
}) => Promise<void | BlogPost[]>;
```

## Example configuration

You can configure this plugin through preset options or plugin options.

**TIP**

Most Docusaurus users configure this plugin through the preset options.

#### Preset options

#### Plugin options

If you use a preset, configure this plugin through the [preset options](#):

```
docusaurus.config.js
```



```
module.exports = {
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 blog: {
 path: 'blog',
 // Simple use-case: string editUrl
 // editUrl:
 'https://github.com/facebook/docusaurus/edit/main/website/',
 // Advanced use-case: functional editUrl
 editUrl: ({locale, blogDirPath, blogPath, permalink}) =>
 `https://github.com/facebook/docusaurus/edit/main/website/${blogDirPath}/${blogPath}`,
 editLocalizedFiles: false,
 blogTitle: 'Blog title',
 blogDescription: 'Blog',
 blogSidebarCount: 5,
 blogSidebarTitle: 'All our posts',
 routeBasePath: 'blog',
 include: ['**/*.md', '**/*.mdx'],
 exclude: [
 '**/*.{js,jsx,ts,tsx,md,mdx}',
 '**/*/*/*',
 '**/*test.{js,jsx,ts,tsx}',
 '**/__tests__/*',
],
 postsPerPage: 10,
 blogListComponent: '@theme/BlogListPage',
 blogPostComponent: '@theme/BlogPostPage',
 blogTagsListComponent: '@theme/BlogTagsListPage',
 blogTagsPostsComponent: '@theme/BlogTagsPostsPage',
 remarkPlugins: [require('./my-remark-plugin')],
 rehypePlugins: [],
 beforeDefaultRemarkPlugins: [],
 beforeDefaultRehypePlugins: [],
 truncateMarker: /<!--\s*(truncate)\s*-->/,
 showReadingTime: true,
 feedOptions: {
 type: '',
 title: '',
 description: '',
 copyright: '',
 language: undefined,
 createFeedItems: async (params) => {
 const {blogPosts, defaultCreateFeedItems, ...rest} = params;
 return defaultCreateFeedItems({
 // keep only the 10 most recent blog posts in the feed

```



```
blogPosts: blogPosts.filter((item, index) => index < 10),
...rest,
});
},
},
},
},
],
],
};
```

## Markdown front matter

Markdown documents can use the following Markdown [front matter](#) metadata fields, enclosed by a line `---` on either side.

Accepted fields:

Name	Type	Default	Description
<code>authors</code>	Authors	undefined	List of blog post authors (or unique author). Read the <a href="#">authors guide</a> for more explanations. Prefer <code>authors</code> over the <code>author_*</code> front matter fields, even for single author blog posts.
<code>author</code>	string	undefined	⚠ Prefer using <code>authors</code> . The blog post author's name.
<code>author_url</code>	string	undefined	⚠ Prefer using <code>authors</code> . The URL that the author's name will be linked to. This could be a GitHub, X, Facebook profile URL, etc.
<code>author_image_url</code>	string	undefined	⚠ Prefer using <code>authors</code> . The URL to the author's thumbnail image.
<code>author_title</code>	string	undefined	⚠ Prefer using <code>authors</code> . A description of the author.
<code>title</code>	string	Markdown title	The blog post title.
<code>title_meta</code>	string	<code>frontMatter.title</code>	The blog post SEO metadata title, used in <code>&lt;head&gt;</code> for <code>&lt;title&gt;</code> and <code>og:title</code> . Permits to override <code>title</code> when the displayed title and SEO title should be different.
<code>sidebar_label</code>	string	<code>title</code>	A custom label for the blog sidebar, replacing the default one ( <code>title</code> ).
<code>date</code>	string	File name or file creation time	The blog post creation date. If not specified, this can be extracted from the file or folder name, e.g, <code>2021-04-15-blog-post.mdx</code> , <code>2021-04-15-blog-post/index.mdx</code> , <code>2021/04/15/blog-post.mdx</code> . Otherwise, it is the Markdown file creation time.
<code>tags</code>	Tag[]	undefined	A list of strings or objects of two string fields <code>label</code> and <code>permalink</code> to tag to your post. Strings can be a reference to keys of a <a href="#">tags file</a> (usually <code>tags.yml</code> )





Name	Type	Default	Description
<code>draft</code>	<code>boolean</code>	<code>false</code>	Draft blog posts will only be available during development.
<code>unlisted</code>	<code>boolean</code>	<code>false</code>	Unlisted blog posts will be available in both development and production. They will be "hidden" in production, not indexed, excluded from sitemaps, and can only be accessed by users having a direct link.
<code>hide_table_of_contents</code>	<code>boolean</code>	<code>false</code>	Whether to hide the table of contents to the right.
<code>toc_min_heading_level</code>	<code>number</code>	<code>2</code>	The minimum heading level shown in the table of contents. Must be between 2 and 6 and lower or equal to the max value.
<code>toc_max_heading_level</code>	<code>number</code>	<code>3</code>	The max heading level shown in the table of contents. Must be between 2 and 6.
<code>keywords</code>	<code>string[]</code>	<code>undefined</code>	Keywords meta tag, which will become the <code>&lt;meta name="keywords" content="keyword1, keyword2, ..." /&gt;</code> in <code>&lt;head&gt;</code> , used by search engines.
<code>description</code>	<code>string</code>	The first line of Markdown content	The description of your document, which will become the <code>&lt;meta name="description" content="..." /&gt;</code> and <code>&lt;meta property="og:description" content="..." /&gt;</code> in <code>&lt;head&gt;</code> , used by search engines.
<code>image</code>	<code>string</code>	<code>undefined</code>	Cover or thumbnail image that will be used as the <code>&lt;meta property="og:image" content="..." /&gt;</code> in the <code>&lt;head&gt;</code> , enhancing link previews on social media and messaging platforms.
<code>slug</code>	<code>string</code>	File path	Allows to customize the blog post URL ( <code>/&lt;routeBasePath&gt;/&lt;slug&gt;</code> ). Supports multiple patterns: <code>slug: my-blog-post</code> , <code>slug: /my/path/to/blog/post</code> , <code>slug: /</code> .
<code>last_update</code>	<code>FrontMatterLastUpdate</code>	<code>undefined</code>	Allows overriding the last update author/date. Date can be any <a href="#">parsable date string</a> .



```
type FrontMatterLastUpdate = {date?: string; author?: string};

type Tag = string | {label: string; permalink: string};

// An author key references an author from the global plugin authors.yml
// file
type AuthorKey = string;

// Social platform name -> Social platform link
// Example: {MyPlatform: 'https://myplatform.com/myusername'}
// Pre-defined platforms
// ("x", "github", "twitter", "linkedin", "stackoverflow", "instagram",
// "bluesky", "mastodon", "threads", "twitch", "youtube", "email") accept
// handles:
// Example: {github: 'slorber'}
type AuthorSocials = Record<string, string>;

type Author = {
 key?: AuthorKey;
 name: string;
 title?: string;
 url?: string;
 image_url?: string;
 socials?: AuthorSocials;
};

// The front matter authors field allows various possible shapes
type Authors = AuthorKey | Author | (AuthorKey | Author)[];
```

Example:



```

title: Welcome Docusaurus
authors:
 - slorber
 - yangshun
 - name: Joel Marcey
 title: Co-creator of Docusaurus 1
 url: https://github.com/JoelMarcey
 image_url: https://github.com/JoelMarcey.png
 socials:
 x: joelmarcey
 github: JoelMarcey
tags: [docusaurus]
description: This is my first post on Docusaurus.
image: https://i.imgur.com/mErPwqL.png
hide_table_of_contents: false

```

A Markdown blog post

## Tags File

Use the `tags` plugin option to configure the path of a YAML tags file.

By convention, the plugin will look for a `tags.yml` file at the root of your content folder(s).

This file can contain a list of predefined tags. You can reference these tags by their keys in Markdown files thanks to the `tags` front matter.



### KEEPING TAGS CONSISTENT

Using a tags file, you can ensure that your tags usage is consistent across your plugin content set. Use the `onInlineTags: 'throw'` plugin option to enforce this consistency and prevent usage of inline tags declared on the fly.

## Types

The YAML content of the provided tags file should respect the following shape:



```
type Tag = {
 label?: string; // Tag display label
 permalink?: string; // Tag URL pathname segment
 description?: string; // Tag description displayed in the tag page
};

type TagsFileInput = Record<string, Partial<Tag> | null>;
```

## Example

tags.yml

```
releases:
 label: 'Product releases'
 permalink: '/product-releases'
 description: 'Content related to product releases.'

A partial tag definition is also valid
announcements:
 label: 'Announcements'

An empty tag definition is also valid
Other attributes will be inferred from the key
emptyTag:
```

content.md

```

tags: [releases, announcements, emptyTag]

Title

Content
```

## Authors File

Use the `authors` plugin option to configure the path of a YAML authors file.

By convention, the plugin will look for a `authors.yml` file at the root of your blog content folder(s).



This file can contain a list of predefined [global blog authors](#). You can reference these authors by their keys in Markdown files thanks to the `authors` [front matter](#).

## Types

The YAML content of the provided authors file should respect the following shape:

```
type AuthorsMapInput = {
 [authorKey: string]: AuthorInput;
};

type AuthorInput = {
 name?: string;
 title?: string;
 description?: string;
 imageURL?: string;
 url?: string;
 email?: string;
 page?: boolean | {permalink: string};
 socials?: Record<string, string>;
 [customAuthorAttribute: string]: unknown;
};
```

## Example

authors.yml



```
slorber:
 name: Sébastien Lorber
 title: Docusaurus maintainer
 url: https://sebastienlorber.com
 image_url: https://github.com/slorber.png
 page: true
 socials:
 x: sebastienlorber
 github: slorber
 email: seb@example.com

jmarcey:
 name: Joël Marcey
 title: Co-creator of Docusaurus 1
 url: https://github.com/JoelMarcey
 image_url: https://github.com/JoelMarcey.png
 email: jimarcey@gmail.com
 page:
 permalink: '/joel-marcey'
 socials:
 x: joelmarcey
 github: JoelMarcey
```

blog/my-blog-post.md

```

authors: [slorber, jmarcey]

My Blog Post

Content
```

## i18n

Read the [i18n introduction](#) first.

### Translation files location

- **Base path:** `website/i18n/[locale]/docusaurus-plugin-content-blog`
- **Multi-instance path:** `website/i18n/[locale]/docusaurus-plugin-content-blog-[pluginId]`
- **JSON files:** extracted with `docusaurus write-translations`
- **Markdown files:** `website/i18n/[locale]/docusaurus-plugin-content-blog`



## Example file-system structure

```
website/i18n/[locale]/docusaurus-plugin-content-blog
├── # translations for website/blog
│ ├── authors.yml
│ ├── first-blog-post.md
│ └── second-blog-post.md
└── # translations for the plugin options that will be rendered
 └── options.json
```

### 2.6.3 plugin-content-pages

The default pages plugin for Docusaurus. The classic template ships with this plugin with default configurations. This plugin provides [creating pages](#) functionality.

## Installation

[npm](#)   [Yarn](#)   [pnpm](#)   [Bun](#)

```
npm install --save @docusaurus/plugin-content-pages
```



#### TIP

If you use the preset `@docusaurus/preset-classic`, you don't need to install this plugin as a dependency.

You can configure this plugin through the [preset options](#).

## Configuration

Accepted fields:

Name	Type	Default	Description
<code>path</code>	<code>string</code>	<code>'src/pages'</code>	Path to data on filesystem relative to site dir. Components in this directory will be automatically converted to pages.
<code>editUrl</code>	<code>string</code>   <a href="#">EditUrlFn</a>	<code>undefined</code>	<b>Only for Markdown pages.</b> Base URL to edit your site. The final URL is computed by <code>editUrl + relativePostPath</code> . Using a function allows more nuanced control for each



Name	Type	Default	Description
			file. Omitting this variable entirely will disable edit links.
editLocalizedFiles	boolean	false	<b>Only for Markdown pages.</b> The edit URL will target the localized file, instead of the original unlocalized file. Ignored when <code>editUrl</code> is a function.
routeBasePath	string	'/'	URL route for the pages section of your site. <b>DO NOT</b> include a trailing slash.
include	string[]	<code>['**/*.js,*.jsx,*.ts,*.tsx,*.md,*.mdx']</code>	Matching files will be included and processed.
exclude	string[]	See example configuration	No route will be created for matching files.
mdxPageComponent	string	'@theme/MDXPage'	Component used by each MDX page.
remarkPlugins	any[]	[]	Remark plugins passed to MDX.
rehypePlugins	any[]	[]	Rehype plugins passed to MDX.
recmaPlugins	any[]	[]	Recma plugins passed to MDX.
beforeDefaultRemarkPlugins	any[]	[]	Custom Remark plugins passed to MDX before the default Docusaurus Remark plugins.
beforeDefaultRehypePlugins	any[]	[]	Custom Rehype plugins passed to MDX before the default Docusaurus Rehype plugins.
showLastUpdateAuthor	boolean	false	<b>Only for Markdown pages.</b> Whether to display the author who last updated the page.
showLastUpdateTime	boolean	false	<b>Only for Markdown pages.</b> Whether to display the last date the page post was updated. This requires access to git history during the build, so will not work correctly with shallow clones (a common default for CI systems). With GitHub actions/checkout, use <code>fetch-depth: 0</code> . When deploying to Vercel, set the environment variable <code>VERCEL_DEEP_CLONE=true</code> .

## Types

### EditUrlFn

```
type EditUrlFunction = (params: {
 blogDirPath: string;
 blogPath: string;
 permalink: string;
 locale: string;
}) => string | undefined;
```

## Example configuration

You can configure this plugin through preset options or plugin options.



 **TIP**

Most Docusaurus users configure this plugin through the preset options.

**Preset options****Plugin options**

If you use a preset, configure this plugin through the [preset options](#):

docusaurus.config.js

```
module.exports = {
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 pages: {
 path: 'src/pages',
 routeBasePath: '',
 include: ['**/*.{js,jsx,ts,tsx,md,mdx}'],
 exclude: [
 '**/_*.{js,jsx,ts,tsx,md,mdx}',
 '**/_*/**',
 '**/*.test.{js,jsx,ts,tsx}',
 '**/__tests__/**',
],
 mdxPageComponent: '@theme/MDXPage',
 remarkPlugins: [require('./my-remark-plugin')],
 rehypePlugins: [],
 beforeDefaultRemarkPlugins: [],
 beforeDefaultRehypePlugins: [],
 },
],
],
],
};
```

## Markdown front matter

Markdown pages can use the following Markdown [front matter](#) metadata fields, enclosed by a line `---` on either side.

Accepted fields:



Name	Type	Default	Description
title	string	Markdown title	The blog post title.
description	string	The first line of Markdown content	The description of your page, which will become the <code>&lt;meta name="description" content="..." /&gt;</code> and <code>&lt;meta property="og:description" content="..." /&gt;</code> in <code>&lt;head&gt;</code> , used by search engines.
keywords	string[]	undefined	Keywords meta tag, which will become the <code>&lt;meta name="keywords" content="keyword1,keyword2,..." /&gt;</code> in <code>&lt;head&gt;</code> , used by search engines.
image	string	undefined	Cover or thumbnail image that will be used as the <code>&lt;meta property="og:image" content="..." /&gt;</code> in the <code>&lt;head&gt;</code> , enhancing link previews on social media and messaging platforms.
slug	string	File path	Allows to customize the page URL ( <code>/&lt;routeBasePath&gt;/&lt;slug&gt;</code> ). Supports multiple patterns: <code>slug: my-page</code> , <code>slug: /my/page</code> , <code>slug: /</code> .
wrapperClassName	string		Class name to be added to the wrapper element to allow targeting specific page content.
hide_table_of_contents	boolean	false	Whether to hide the table of contents to the right.
draft	boolean	false	Draft pages will only be available during development.
unlisted	boolean	false	Unlisted pages will be available in both development and production. They will be "hidden" in production, not indexed, excluded from sitemaps, and can only be accessed by users having a direct link.

Example:

```

title: Markdown Page
description: Markdown page SEO description
wrapperClassName: markdown-page
hide_table_of_contents: false
draft: true
slug: /markdown-page

Markdown page content
```

## i18n

Read the [i18n introduction](#) first.

### Translation files location

- **Base path:** `website/i18n/[locale]/docusaurus-plugin-content-pages`



- **Multi-instance path:** `website/i18n/[locale]/docusaurus-plugin-content-pages-[pluginId]`
- **JSON files:** extracted with `docusaurus write-translations`
- **Markdown files:** `website/i18n/[locale]/docusaurus-plugin-content-pages`

## Example file-system structure

```
website/i18n/[locale]/docusaurus-plugin-content-pages
├── # translations for website/src/pages
├── first-markdown-page.md
└── second-markdown-page.md
```

## 2.6.4 plugin-client-redirects

Docusaurus Plugin to generate **client-side redirects**.

This plugin will write additional HTML pages to your static site that redirect the user to your existing Docusaurus pages with JavaScript.

### **PRODUCTION ONLY**

This plugin is always inactive in development and **only active in production** because it works on the build output.

### **WARNING**

It is better to use server-side redirects whenever possible.

Before using this plugin, you should look if your hosting provider doesn't offer this feature.

## Installation

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)

```
npm install --save @docusaurus/plugin-client-redirects
```

## Configuration

Accepted fields:



Option	Type	Default	Description
<code>fromExtensions</code>	<code>string[]</code>	<code>[]</code>	The extensions to be removed from the route after redirecting.
<code>toExtensions</code>	<code>string[]</code>	<code>[]</code>	The extensions to be appended to the route after redirecting.
<code>redirects</code>	<code>RedirectRule[]</code>	<code>[]</code>	The list of redirect rules.
<code>createRedirects</code>	<code>CreateRedirectsFn</code>	<code>undefined</code>	A callback to create a redirect rule. Docusaurus query this callback against every path it has created, and use its return value to output more paths.

### **i** NOTE

This plugin will also read the `siteConfig.onDuplicateRoutes` config to adjust its logging level when multiple files will be emitted to the same location.

## Types

### RedirectRule

```
type RedirectRule = {
 to: string;
 from: string | string[];
};
```

### **i** NOTE

The idea of "from" and "to" is central in this plugin. "From" means a path that you want to *create*, i.e. an extra HTML file that will be written; "to" means a path to want to redirect *to*, usually a route that Docusaurus already knows about.

This is why you can have multiple "from" for the same "to": we will create multiple HTML files that all redirect to the same destination. On the other hand, one "from" can never have more than one "to": the written HTML file needs to have a determinate destination.

### CreateRedirectsFn



```
// The parameter `path` is a route that Docusaurus has already created. It
can
// be seen as the "to", and your return value is the "from". Returning a
falsy
// value will not create any redirect pages for this particular path.
type CreateRedirectsFn = (path: string) => string[] | string | null |
undefined;
```

## Example configuration

Here's an example configuration:

```
docusaurus.config.js
```



```
export default {
 plugins: [
 [
 '@docusaurus/plugin-client-redirects',
 {
 fromExtensions: ['html', 'htm'], // /myPage.html -> /myPage
 toExtensions: ['exe', 'zip'], // /myAsset -> /myAsset.zip (if
 // latter exists)
 redirects: [
 // /docs/oldDoc -> /docs/newDoc
 {
 to: '/docs/newDoc',
 from: '/docs/oldDoc',
 },
 // Redirect from multiple old paths to the new path
 {
 to: '/docs/newDoc2',
 from: ['/docs/oldDocFrom2019', '/docs/legacyDocFrom2016'],
 },
],
 createRedirects(existingPath) {
 if (existingPath.includes('/community')) {
 // Redirect from /docs/team/X to /community/X and
 // /docs/support/X to /community/X
 return [
 existingPath.replace('/community', '/docs/team'),
 existingPath.replace('/community', '/docs/support'),
];
 }
 return undefined; // Return a falsy value: no redirect created
 },
],
],
],
};
```

## 2.6.5 plugin-debug

The debug plugin will display useful debug information at

[http://localhost:3000/\\_\\_docusaurus/debug](http://localhost:3000/__docusaurus/debug).

It is mostly useful for plugin authors, that will be able to inspect more easily the content of the `.docusaurus` folder (like the creates routes), but also be able to inspect data structures that are never written to disk, like the plugin data loaded through the `contentLoaded` lifecycle.



INFO



If you use the plugin via the classic preset, the preset will **enable the plugin in development and disable it in production** by default (`debug: undefined`) to avoid exposing potentially sensitive information. You can use `debug: true` to always enable it or `debug: false` to always disable it.

If you use a standalone plugin, you may need to achieve the same effect by checking the environment:

docusaurus.config.js

```
export default {
 plugins: [
 process.env.NODE_ENV !== 'production' && '@docusaurus/plugin-
 debug',
].filter(Boolean),
};
```

### NOTE

If you report a bug, we will probably ask you to have this plugin turned on in the production, so that we can inspect your deployment config more easily.

If you don't have any sensitive information, you can keep it on in production [like we do](#).

## Installation

**npm**   Yarn   pnpm   Bun

```
npm install --save @docusaurus/plugin-debug
```

### TIP

If you use the preset `@docusaurus/preset-classic`, you don't need to install this plugin as a dependency.

You can configure this plugin through the [preset options](#).

## Configuration

This plugin currently has no options.



## Example configuration

You can configure this plugin through preset options or plugin options.

### TIP

Most Docusaurus users configure this plugin through the preset options.

#### Preset options

#### Plugin Options

If you use a preset, configure this plugin through the [preset options](#):

docusaurus.config.js

```
export default {
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 debug: true, // This will enable the plugin in production
 },
],
],
};
```

## 2.6.6 plugin-google-gtag

The default [Global Site Tag \(gtag.js\)](#) plugin. It is a JavaScript tagging framework and API that allows you to send event data to Google Analytics, Google Ads, and Google Marketing Platform. This section describes how to configure a Docusaurus site to enable global site tag for Google Analytics.

### TIP

You can use [Google's Tag Assistant](#) tool to check if your gtag is set up correctly!



#### PRODUCTION ONLY

This plugin is always inactive in development and **only active in production** to avoid polluting the analytics statistics.

## Installation



[npm](#)[Yarn](#)[pnpm](#)[Bun](#)

```
npm install --save @docusaurus/plugin-google-gtag
```

**TIP**

If you use the preset `@docusaurus/preset-classic`, you don't need to install this plugin as a dependency.

You can configure this plugin through the [preset options](#).

## Configuration

Accepted fields:

Name	Type	Default	Description
<code>trackingID</code>	<code>string   string[]</code>	<b>Required</b>	The tracking ID of your gtag service. It is possible to provide multiple ids.
<code>anonymizeIP</code>	<code>boolean</code>	<code>false</code>	Whether the IP should be anonymized when sending requests.

### Example configuration

You can configure this plugin through preset options or plugin options.

**TIP**

Most Docusaurus users configure this plugin through the preset options.

[Preset options](#)[Plugin options](#)

If you use a preset, configure this plugin through the [preset options](#):

```
docusaurus.config.js
```



```
module.exports = {
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 gtag: {
 trackingID: 'G-999X9XX9XX',
 anonymizeIP: true,
 },
 },
],
],
};
```

## 2.6.7 plugin-rsdoctor

A [Rsdoctor](#) plugin can help you troubleshoot the bundling phase of your Docusaurus site, supporting both Webpack and Rspack.



### TIP

Use it to figure out which plugin or loader is slowing down the bundler, and focus your efforts on optimizing the bottleneck.

## Installation

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)

```
npm install --save @docusaurus/plugin-rsdoctor
```

## Configuration

Accepted fields:

Name	Type	Default	Description
<code>rsdoctorOptions</code>	<code>object</code>	<code>{}</code>	The <a href="#">Rsdoctor</a> bundler plugin options, forwarded as is

### Example configuration

You can configure this plugin through plugin options.



docusaurus.config.js

```
export default {
 plugins: [
 [
 'rsdoctor',
 {
 rsdoctorOptions: {
 mode: 'lite',
 },
 },
],
],
};
```

## 2.6.8 plugin-svgr

An [SVGR](#) plugin to transform SVG files into React components automatically at build time.

## Installation

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)

```
npm install --save @docusaurus/plugin-svgr
```



### TIP

If you use the preset `@docusaurus/preset-classic`, you don't need to install this plugin as a dependency.

You can configure this plugin through the [preset options](#).

## Configuration

Accepted fields:

Name	Type	Default	Description
svgrConfig	object	{}	The <a href="#">SVGR config options</a> , forwarded as is

### Example configuration

You can configure this plugin through plugin options.



## Preset options

## Plugin options

If you use a preset, configure this plugin through the [preset options](#):

docusaurus.config.js

```
module.exports = {
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 svgr: {
 svgrConfig: {
 /* SVGR config */
 },
 },
],
],
],
};
```

## 2.6.9 plugin-google-tag-manager

A plugin for adding [Google Tag Manager \(gtm.js\)](#) to a Docusaurus site. Use this plugin in conjunction with the standard [tag plugin](#) for in-depth analysis of how users are using your site.



### TIP

You can use [Google's Tag Assistant](#) tool to check if tag manager is set up correctly!



### PRODUCTION ONLY

This plugin is always inactive in development and **only active in production** to avoid polluting the analytics statistics.

## Installation

[npm](#)

[Yarn](#)

[pnpm](#)

[Bun](#)

```
npm install --save @docusaurus/plugin-google-tag-manager
```



### TIP

If you use the preset `@docusaurus/preset-classic`, you don't need to install this plugin as a dependency.

You can configure this plugin through the [preset options](#).

## Configuration

Accepted fields:

Name	Type	Default	Description
<code>containerId</code>	<code>string</code>	<b>Required</b>	Your Tag Manager container Id (usually starts with <code>GTM-</code> ).

### Example configuration

You can configure this plugin through preset options or plugin options.

### TIP

Most Docusaurus users configure this plugin through the preset options.

#### Preset options

#### Plugin options

If you use a preset, configure this plugin through the [preset options](#):

docusaurus.config.js

```
module.exports = {
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 googleTagManager: {
 containerId: 'GTM-12345',
 },
 },
],
],
};
```

## 2.6.10 plugin-ideal-image

Docusaurus Plugin to generate an almost ideal image (responsive, lazy-loading, and low quality placeholder).



### ! INFO

By default, the plugin is **inactive in development** so you could always view full-scale images. If you want to debug the ideal image behavior, you could set the `disableInDev` option to `false`.

## Installation

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)

```
npm install --save @docusaurus/plugin-ideal-image
```

## Usage

This plugin supports the PNG and JPG formats only.

```
import Image from '@theme/IdealImage';
import thumbnail from './path/to/img.png';

// your React code
<Image img={thumbnail} />

// or
<Image img={require('./path/to/img.png')} />
```

### i NOTE

When `plugin-ideal-image` is installed, importing a supported image file no longer returns a plain URL string. Instead, it returns an object with the shape: `{ preSrc: string, src: { src: string, width: number, height: number } }`.

If you need to use the image in a standard `<img>` tag, access it like this:



```
<img
 src={image.src.src}
 width={image.src.width}
 height={image.src.height}
 alt="description"
/>
```

To avoid dealing with this object shape directly, it is recommended to use the `<Image>` component provided by this plugin instead.

### ⚠ WARNING

This plugin registers a [Webpack loader](#) that changes the type of imported/require images:

- Before: `string`
- After: `{preSrc: string, src: import("@theme/IdealImage").SrcImage}`

### ⚠ FOR PNPM USERS

Starting with [pnpm 10](#), running `pnpm install` won't run dependency install scripts by default. You'll need additional pnpm configuration ([issue](#)) for our `sharp` image resizing dependency to install correctly, such as:

package.json

```
{
 "pnpm": {
 "onlyBuiltDependencies": ["fsevents"]
 }
}
```

## Configuration

Accepted fields:

Option	Type	Default	Description
<code>name</code>	<code>string</code>	<code>ideal-img/[name].[hash:hex:7].[width].[ext]</code>	Filename template for output files.
<code>sizes</code>	<code>number[]</code>	<i>original size</i>	Specify all widths you want to use. If a specified size exceeds the original image's



Option	Type	Default	Description
			width, the latter will be used (i.e. images won't be scaled up).
size	number	original size	Specify one width you want to use; if the specified size exceeds the original image's width, the latter will be used (i.e. images won't be scaled up)
min	number		As an alternative to manually specifying sizes, you can specify min, max and steps, and the sizes will be generated for you.
max	number		See min above
steps	number	4	Configure the number of images generated between min and max (inclusive)
quality	number	85	JPEG compression quality
disableInDev	boolean	true	You can test ideal image behavior in dev mode by setting this to false. <b>Tip:</b> use <a href="#">network throttling</a> in your browser to simulate slow networks.

## Example configuration

Here's an example configuration:

docusaurus.config.js

```
export default {
 plugins: [
 [
 '@docusaurus/plugin-ideal-image',
 {
 quality: 70,
 max: 1030, // max resized image's size.
 min: 640, // min resized image's size. if original is lower, use
 // that size.
 steps: 2, // the max number of images generated between min and max
 // (inclusive)
 disableInDev: false,
 },
],
],
};
```

## 2.6.11 plugin-css-cascade-layers

### EXPERIMENTAL

This plugin is mostly designed to be used internally by the classic preset through the [Docusaurus future.v4.useCssCascadeLayers](#) flag, although it can also be used as a standalone plugin. Please





let us know [here](#) if you have a use case for it and help us design an API that makes sense for the future of Docusaurus.

A plugin for wrapping CSS modules of your Docusaurus site in [CSS Cascade Layers](#). This modern CSS feature is widely supported by all browsers. It allows grouping CSS rules in layers of specificity and gives you more control over the CSS cascade.

Use this plugin to:

- apply a top-level `@layer myLayer { ... }` block rule around any CSS module, including un-layered third-party CSS.
- define an explicit layer ordering

### ⚠ CAUTION

To use this plugin properly, it's recommended to have a solid understanding of [CSS Cascade Layers](#), the [CSS Cascade](#) and [specificity](#).

## Installation

[npm](#)   [Yarn](#)   [pnpm](#)   [Bun](#)

```
npm install --save @docusaurus/plugin-css-cascade-layers
```

### 💡 TIP

If you use the preset `@docusaurus/preset-classic`, this plugin is automatically configured for you with the `siteConfig.future.v4.useCssCascadeLayers` flag.

## Configuration

Accepted fields:

Name	Type	Default	Description
<code>layers</code>	<code>Layers</code>	<b>Built-in layers</b>	An object representing all the CSS cascade layers you want to use, and whether the layer should be applied to a given file path. See examples and types below.

### Types

`Layers`



```
type Layers = Record<
 string, // layer name
 (filePath: string) => boolean // layer matcher
>;
```

The `layers` object is defined by:

- key: the name of a layer
- value: a function to define if a given CSS module file should be in that layer

### ORDER MATTERS

The object order matters:

- the keys order defines an explicit CSS layer order
- when multiple layers match a file path, only the first layer will apply

## Example configuration

You can configure this plugin through plugin options.

```
const options = {
 layers: {
 'docusaurus.infima': (filePath) =>
 filePath.includes('/node_modules/infima/dist'),
 'docusaurus.theme-classic': (filePath) =>
 filePath.includes('/node_modules/@docusaurus/theme-classic/lib'),
 },
};
```

## 2.6.12 plugin-pwa

Docusaurus Plugin to add PWA support using [Workbox](#). This plugin generates a [Service Worker](#) in production build only, and allows you to create fully PWA-compliant documentation site with offline and installation support.

## Installation

[npm](#)

[Yarn](#)

[pnpm](#)

[Bun](#)



```
npm install --save @docusaurus/plugin-pwa
```

## Configuration

Create a [PWA manifest](#) at `./static/manifest.json`.

Modify `docusaurus.config.js` with a minimal PWA config, like:

`docusaurus.config.js`

```
export default {
 plugins: [
 [
 '@docusaurus/plugin-pwa',
 {
 debug: true,
 offlineModeActivationStrategies: [
 'appInstalled',
 'standalone',
 'queryString',
],
 },
],
 pwaHead: [
 {
 tagName: 'link',
 rel: 'icon',
 href: '/img/docusaurus.png',
 },
 {
 tagName: 'link',
 rel: 'manifest',
 href: '/manifest.json', // your PWA manifest
 },
 {
 tagName: 'meta',
 name: 'theme-color',
 content: 'rgb(37, 194, 160)',
 },
],
],
};
```



## Progressive Web App

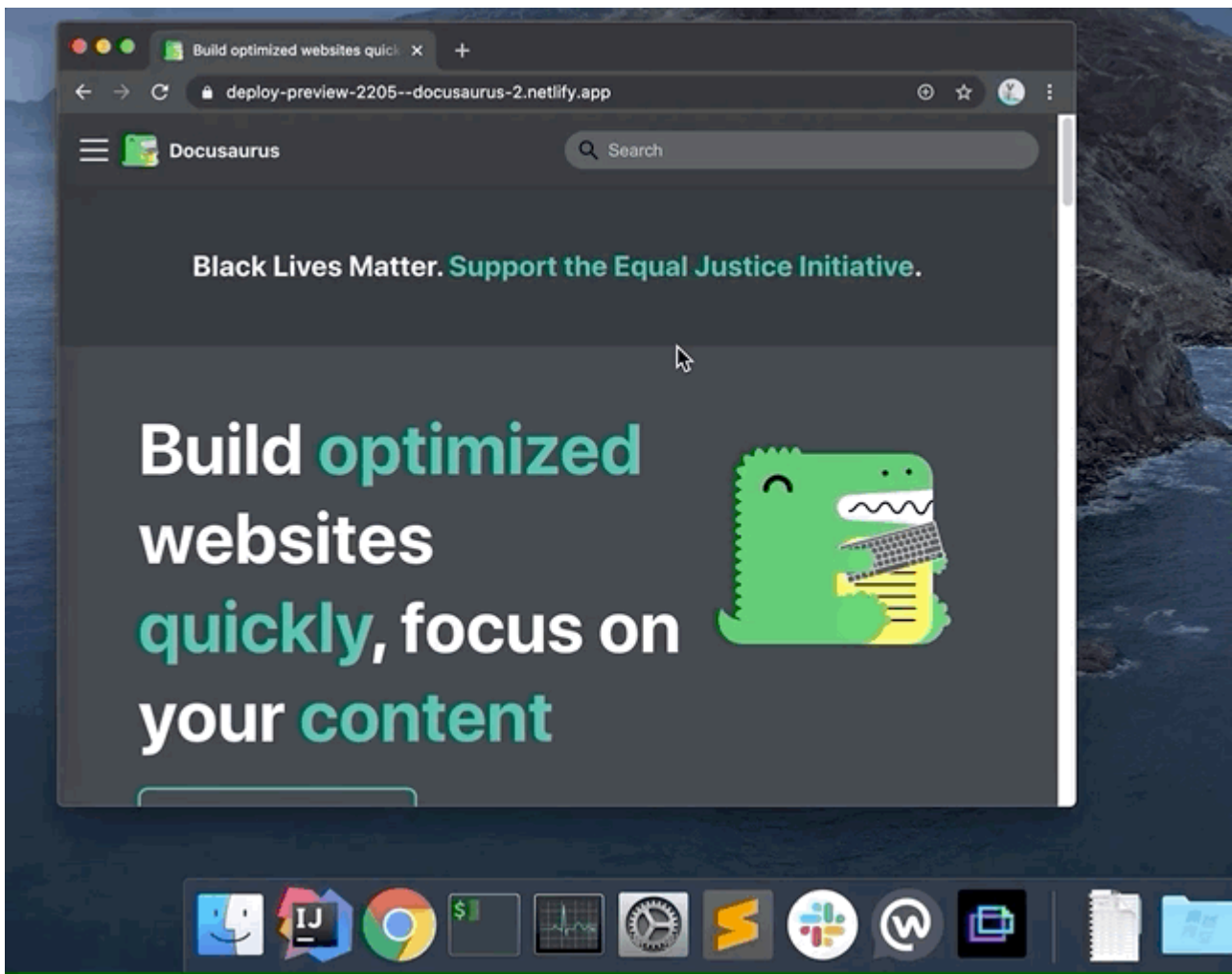
Having a service worker installed is not enough to make your application a PWA. You'll need to at least include a [Web App Manifest](#) and have the correct tags in `<head>` ([Options > pwaHead](#)).

After deployment, you can use [Lighthouse](#) to run an audit on your site.

For a more exhaustive list of what it takes for your site to be a PWA, refer to the [PWA Checklist](#)

## App installation support

If your browser supports it, you should be able to install a Docusaurus site as an app.



### NOTE

App installation requires the HTTPS protocol and a valid manifest.

## Offline mode (precaching)

We enable users to browse a Docusaurus site offline, by using service-worker precaching.



The [workbox-precaching](#) page explains the idea:

One feature of service workers is the ability to save a set of files to the cache when the service worker is installing. This is often referred to as "precaching", since you are caching content ahead of the service worker being used.

The main reason for doing this is that it gives developers control over the cache, meaning they can determine when and how long a file is cached as well as serve it to the browser without going to the network, meaning it can be used to create web apps that work offline.

Workbox takes a lot of the heavy lifting out of precaching by simplifying the API and ensuring assets are downloaded efficiently.

By default, offline mode is enabled when the site is installed as an app. See the `offlineModeActivationStrategies` option for details.

After the site has been precached, the service worker will serve cached responses for later visits. When a new build is deployed along with a new service worker, the new one will begin installing and eventually move to a waiting state. During this waiting state, a reload popup will show and ask the user to reload the page for new content. Until the user either clears the application cache or clicks the `reload` button on the popup, the service worker will continue serving the old content.

### WARNING

Offline mode / precaching requires downloading all the static assets of the site ahead of time, and can consume unnecessary bandwidth. It may not be a good idea to activate it for all kind of sites.

## Options

### `debug`

- Type: `boolean`
- Default: `false`

Turn debug mode on:

- Workbox logs
- Additional Docusaurus logs
- Unoptimized SW file output
- Source maps

### `offlineModeActivationStrategies`

- Type: `('appInstalled' | 'mobile' | 'saveData' | 'queryString' | 'always')[]`
- Default: `['appInstalled', 'queryString', 'standalone']`

Strategies used to turn the offline mode on:

- `appInstalled`: activates for users having installed the site as an app (not 100% reliable)



- `standalone`: activates for users running the app as standalone (often the case once a PWA is installed)
- `queryString`: activates if `queryString` contains `offlineMode=true` (convenient for PWA debugging)
- `mobile`: activates for mobile users (`width <= 996px`)
- `saveData`: activates for users with `navigator.connection.saveData === true`
- `always`: activates for all users

### WARNING

Use this carefully: some users may not like to be forced to use the offline mode.

### DANGER

It is not possible to detect if a page is rendered as a PWA in a reliable manner.

The `appinstalled` event has been [removed from the specification](#), and the `navigator.getInstalledRelatedApps()` API is only supported in recent Chrome versions and require `related_applications` declared in the manifest.

The `standalone` strategy is a nice fallback to activate the offline mode (at least when running the installed app).

## injectManifestConfig

[Workbox options](#) to pass to `workbox.injectManifest()`. This gives you control over which assets will be precached, and be available offline.

- Type: `InjectManifestOptions`
- Default: `{}`

```
docusaurus.config.js
```



```
export default {
 plugins: [
 [
 '@docusaurus/plugin-pwa',
 {
 injectManifestConfig: {
 manifestTransforms: [
 //...
],
 modifyURLPrefix: {
 //...
 },
 // We already add regular static assets (HTML, images...) to be
 // available offline
 // You can add more files according to your needs
 globPatterns: ['**/*.pdf,docx,xlsx'],
 // ...
 },
],
],
],
};
```

### pwaHead

- Type: `({ tagName: string; [attributeName: string]: string })[]`
- Default: `[]`

Array of objects containing `tagName` and key-value pairs for attributes to inject into the `<head>` tag. Technically you can inject any head tag through this, but it's ideally used for tags to make your site PWA compliant. Here's a list of tag to make your app fully compliant:



```
export default {
 plugins: [
 [
 '@docusaurus/plugin-pwa',
 {
 pwaHead: [
 {
 tagName: 'link',
 rel: 'icon',
 href: '/img/docusaurus.png',
 },
 {
 tagName: 'link',
 rel: 'manifest',
 href: '/manifest.json',
 },
 {
 tagName: 'meta',
 name: 'theme-color',
 content: 'rgb(37, 194, 160)',
 },
 {
 tagName: 'meta',
 name: 'apple-mobile-web-app-capable',
 content: 'yes',
 },
 {
 tagName: 'meta',
 name: 'apple-mobile-web-app-status-bar-style',
 content: '#000',
 },
 {
 tagName: 'link',
 rel: 'apple-touch-icon',
 href: '/img/docusaurus.png',
 },
 {
 tagName: 'link',
 rel: 'mask-icon',
 href: '/img/docusaurus.svg',
 color: 'rgb(37, 194, 160)',
 },
 {
 tagName: 'meta',
 name: 'msapplication-TileImage',
 content: '/img/docusaurus.png',
 },
 {
 tagName: 'meta',
```





```
 name: 'msapplication-TileColor',
 content: '#000',
 },
],
 },
],
],
};
```

### swCustom

- Type: `string | undefined`
- Default: `undefined`

Useful for additional Workbox rules. You can do whatever a service worker can do here, and use the full power of workbox libraries. The code is transpiled, so you can use modern ES6+ syntax here.

For example, to cache files from external routes:

```
import {registerRoute} from 'workbox-routing';
import {StaleWhileRevalidate} from 'workbox-strategies';

// default fn export receiving some useful params
export default function swCustom(params) {
 const {
 debug, // :boolean
 offlineMode, // :boolean
 } = params;

 // Cache responses from external resources
 registerRoute((context) => {
 return [
 /graph\.facebook\.com\/.*\/picture/,
 /netlify\.com\/img/,
 /avatars1\.githubusercontent/,
].some((regex) => context.url.href.match(regex));
 }, new StaleWhileRevalidate());
}
```

The module should have a `default` function export, and receives some params.

### swRegister

- Type: `string | false`
- Default: `'docusaurus-plugin-pwa/src/registerSW.js'`



Adds an entry before the Docusaurus app so that registration can happen before the app runs. The default `registerSW.js` file is enough for simple registration.

Passing `false` will disable registration entirely.

## Manifest example

The Docusaurus site manifest can serve as an inspiration:



```
{
 "name": "Docusaurus",
 "short_name": "Docusaurus",
 "theme_color": "#2196f3",
 "background_color": "#424242",
 "display": "standalone",
 "scope": "./",
 "start_url": "./index.html",
 "related_applications": [
 {
 "platform": "webapp",
 "url": "https://docusaurus.io/manifest.json"
 }
],
 "icons": [
 {
 "src": "img/icons/icon-72x72.png",
 "sizes": "72x72",
 "type": "image/png"
 },
 {
 "src": "img/icons/icon-96x96.png",
 "sizes": "96x96",
 "type": "image/png"
 },
 {
 "src": "img/icons/icon-128x128.png",
 "sizes": "128x128",
 "type": "image/png"
 },
 {
 "src": "img/icons/icon-144x144.png",
 "sizes": "144x144",
 "type": "image/png"
 },
 {
 "src": "img/icons/icon-152x152.png",
 "sizes": "152x152",
 "type": "image/png"
 },
 {
 "src": "img/icons/icon-192x192.png",
 "sizes": "192x192",
 "type": "image/png"
 },
 {
 "src": "img/icons/icon-384x384.png",
 "sizes": "384x384",
 "type": "image/png"
 }
]
}
```

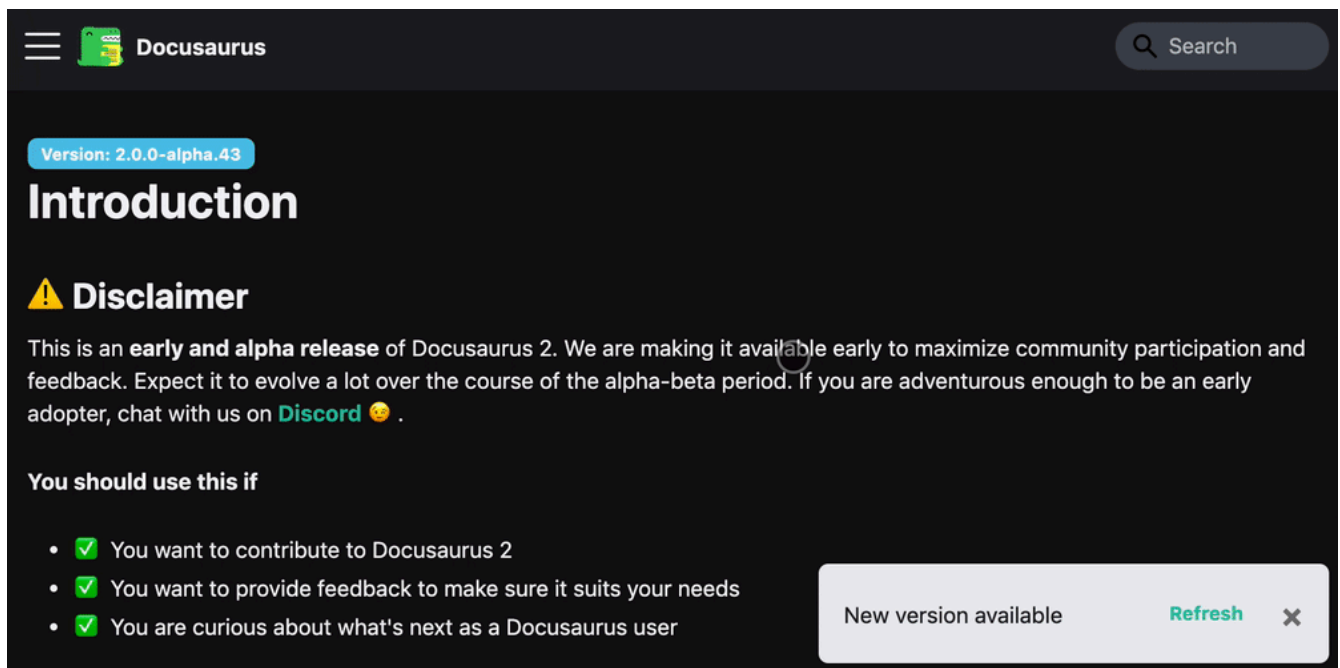


```
 },
 {
 "src": "img/icons/icon-512x512.png",
 "sizes": "512x512",
 "type": "image/png"
 }
]
}
```

## Customizing reload popup

The `@theme/PwaReloadPopup` component is rendered when a new service worker is waiting to be installed, and we suggest a reload to the user. You can [swizzle](#) this component and implement your own UI. It will receive an `onReload` callback as props, which should be called when the `reload` button is clicked. This will tell the service worker to install the waiting service worker and reload the page.

The default theme includes an implementation for the reload popup and uses [Infima Alerts](#).



Your component can render `null`, but this is not recommended: users won't have a way to get up-to-date content.

### 2.6.13 plugin-sitemap

This plugin creates sitemaps for your site so that search engine crawlers can crawl your site more accurately.

#### PRODUCTION ONLY

This plugin is always inactive in development and **only active in production** because it works on the build output.



# Installation

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)

```
npm install --save @docusaurus/plugin-sitemap
```

## TIP

If you use the preset `@docusaurus/preset-classic`, you don't need to install this plugin as a dependency.

You can configure this plugin through the [preset options](#).

# Configuration

Accepted fields:

Name	Type	Default	Description
<code>lastmod</code>	<code>'date'   'datetime'   null</code>	<code>null</code>	<code>date</code> is YYYY-MM-DD. <code>datetime</code> is a ISO 8601 datetime. <code>null</code> is disabled. See <a href="#">sitemap docs</a> .
<code>changefreq</code>	<code>string   null</code>	<code>'weekly'</code>	See <a href="#">sitemap docs</a>
<code>priority</code>	<code>number   null</code>	<code>0.5</code>	See <a href="#">sitemap docs</a>
<code>ignorePatterns</code>	<code>string[]</code>	<code>[]</code>	A list of glob patterns; matching route paths will be filtered from the sitemap. Note that you may need to include the base URL in here.
<code>filename</code>	<code>string</code>	<code>sitemap.xml</code>	The path to the created sitemap file, relative to the output directory. Useful if you have two plugin instances outputting two files.
<code>createSitemapItems</code>	<a href="#">CreateSitemapItemsFn</a>   <code>undefined</code>	<code>undefined</code>	An optional function which can be used to transform and / or filter the items in the sitemap.

## Types

`CreateSitemapItemsFn`



```
type CreateSitemapItemsFn = (params: {
 siteConfig: DocusaurusConfig;
 routes: RouteConfig[];
 defaultCreateSitemapItems: CreateSitemapItemsFn;
}) => Promise<SitemapItem[]>;
```

### ! INFO

This plugin also respects some site config:

- `noIndex`: results in no sitemap generated
- `trailingSlash`: determines if the URLs in the sitemap have trailing slashes

### i ABOUT `lastmod`

The `lastmod` option will only output a sitemap `<lastmod>` tag if plugins provide [route metadata](#) attributes `sourceFilePath` and/or `lastUpdatedAt`.

All the official content plugins provide the metadata for routes backed by a content file (Markdown, MDX or React page components), but it is possible third-party plugin authors do not provide this information, and the plugin will not be able to output a `<lastmod>` tag for their routes.

## Example configuration

You can configure this plugin through preset options or plugin options.

### 💡 TIP

Most Docusaurus users configure this plugin through the preset options.

#### Preset options

#### Plugin options

If you use a preset, configure this plugin through the [preset options](#):

```
docusaurus.config.js
```



```
module.exports = {
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 sitemap: {
 lastmod: 'date',
 changefreq: 'weekly',
 priority: 0.5,
 ignorePatterns: ['/tags/**'],
 filename: 'sitemap.xml',
 createSitemapItems: async (params) => {
 const {defaultCreateSitemapItems, ...rest} = params;
 const items = await defaultCreateSitemapItems(rest);
 return items.filter((item) => !item.url.includes('/page/'));
 },
 },
],
],
],
};
```

You can find your sitemap at `/sitemap.xml`.

## 2.6.14 plugin-vercel-analytics

**Vercel Analytics** provides comprehensive insights into your website's visitors, tracking top pages, referrers, and demographics like location, operating systems, and browser info.

### **PRODUCTION ONLY**

This plugin is always inactive in development and **only active in production** (`docusaurus build`) to avoid polluting the analytics statistics.

## Installation

**npm**   **Yarn**   **pnpm**   **Bun**

```
npm install --save @docusaurus/plugin-vercel-analytics
```

## Configuration



Accepted fields:

Name	Type	Default	Description
mode	string	'auto'	Override the automatic environment detection. Read the <a href="#">official docs</a> for details.
debug	boolean	undefined	Enable browser console logging of analytics events. Read the <a href="#">official docs</a> for details.

## Example configuration

You can configure this plugin through plugin options.

docusaurus.config.js

```
export default {
 plugins: [
 [
 'vercel-analytics',
 {
 debug: true,
 mode: 'auto',
 },
],
],
};
```

## 2.7 Docusaurus themes

We provide official Docusaurus themes.

### 2.7.1 Main themes

The main themes implement the user interface for the [docs](#), [blog](#) and [pages](#) plugins.

- [@docusaurus/theme-classic](#)
- 🚧 other themes are planned

#### ⚠️ WARNING

The goal is to have all themes share the exact same features, user-experience and configuration.

Only the UI design and underlying styling framework should change, and you should be able to change theme easily.

We are not there yet: only the classic theme is production ready.





## 2.7.2 Enhancement themes

These themes will enhance the existing main themes with additional user-interface related features.

- [@docusaurus/theme-live-codeblock](#)
- [@docusaurus/theme-search-algolia](#)

## 2.8 Themes

### 2.8.1 Theme configuration

This configuration applies to all [main themes](#).

## Common

### Color mode

The classic theme provides by default light and dark mode support, with a navbar switch for the user.

It is possible to customize the color mode support within the `colorMode` object.

Accepted fields:

Name	Type	Default	Description
<code>defaultMode</code>	<code>'light'   'dark'</code>	<code>'light'</code>	The color mode when user first visits the site.
<code>disableSwitch</code>	<code>boolean</code>	<code>false</code>	Hides the switch in the navbar. Useful if you want to support a single color mode.
<code>respectPrefersColorScheme</code>	<code>boolean</code>	<code>false</code>	Whether to use the <code>prefers-color-scheme</code> media-query, using user system preferences, instead of the hardcoded <code>defaultMode</code> .

Example configuration:

docusaurus.config.js

```
export default {
 themeConfig: {
 colorMode: {
 defaultMode: 'light',
 disableSwitch: false,
 respectPrefersColorScheme: false,
 },
 },
};
```

**! WARNING**

With `respectPrefersColorScheme: true`, the `defaultMode` is overridden by user system preferences.

If you only want to support one color mode, you likely want to ignore user system preferences.

## Meta image

You can configure a default image that will be used for your meta tag, in particular `og:image` and `twitter:image`.

Accepted fields:

Name	Type	Default	Description
<code>image</code>	<code>string</code>	<code>undefined</code>	The meta image URL for the site. Relative to your site's "static" directory. Cannot be SVGs. Can be external URLs too.

Example configuration:

docusaurus.config.js

```
export default {
 themeConfig: {
 image: 'img/docusaurus.png',
 },
};
```

## Metadata

You can configure additional HTML metadata (and override existing ones).

Accepted fields:

Name	Type	Default	Description
<code>metadata</code>	<code>Metadata[]</code>	<code>[]</code>	Any field will be directly passed to the <code>&lt;meta /&gt;</code> tag. Possible fields include <code>id</code> , <code>name</code> , <code>property</code> , <code>content</code> , <code>itemprop</code> , etc.

Example configuration:

docusaurus.config.js



```
export default {
 themeConfig: {
 metadata: [{name: 'twitter:card', content: 'summary'}],
 },
};
```

## Announcement bar

Sometimes you want to announce something in your website. Just for such a case, you can add an announcement bar. This is a non-fixed and optionally dismissible panel above the navbar. All configuration are in the `announcementBar` object.

Accepted fields:

Name	Type	Default	Description
<code>id</code>	string	<code>'announcement-bar'</code>	Any value that will identify this message.
<code>content</code>	string	<code>''</code>	The text content of the announcement. HTML will be interpolated.
<code>backgroundColor</code>	string	<code>'#fff'</code>	Background color of the entire bar.
<code>textColor</code>	string	<code>'#000'</code>	Announcement text color.
<code>isCloseable</code>	boolean	<code>true</code>	Whether this announcement can be dismissed with a 'x' button.

Example configuration:

docusaurus.config.js

```
export default {
 themeConfig: {
 announcementBar: {
 id: 'support_us',
 content:
 'We are looking to revamp our docs, please fill this survey',
 backgroundColor: '#fafbfc',
 textColor: '#091E42',
 isCloseable: false,
 },
 },
};
```

## Plugins

Our [main themes](#) offer additional theme configuration options for Docusaurus core content plugins.



## Docs

Name	Type	Default	Description
<code>versionPersistence</code>	<code>'localStorage'   'none'</code>	<code>'localStorage'</code>	Defines the browser persistence of the preferred docs version.
<code>sidebar.hideable</code>	<code>boolean</code>	<code>false</code>	Show a hide button at the bottom of the sidebar.
<code>sidebar.autoCollapseCategories</code>	<code>boolean</code>	<code>false</code>	Automatically collapse all sibling categories of the one you navigate to.

Example configuration:

docusaurus.config.js

```
export default {
 themeConfig: {
 docs: {
 versionPersistence: 'localStorage',
 sidebar: {
 hideable: false,
 autoCollapseCategories: false,
 },
 },
 },
};
```

## Blog

Name	Type	Default	Description
<code>sidebar.groupByYear</code>	<code>boolean</code>	<code>true</code>	Group sidebar blog posts by years.

Example configuration:

docusaurus.config.js



```
export default {
 themeConfig: {
 blog: {
 sidebar: {
 groupByYear: true,
 },
 },
 },
};
```

## Navbar

Accepted fields:

Name	Type	Default	Description
title	string	undefined	Title for the navbar.
logo	See below	undefined	Customization of the logo object.
items	NavbarItem[]	[]	A list of navbar items. See specification below.
hideOnScroll	boolean	false	Whether the navbar is hidden when the user scrolls down.
style	'primary'   'dark'	Same as theme	Sets the navbar style, ignoring the dark/light theme.

### Navbar logo

The logo can be placed in [static folder](#). Logo URL is set to base URL of your site by default. Although you can specify your own URL for the logo, if it is an external link, it will open in a new tab. In addition, you can override a value for the target attribute of logo link, it can come in handy if you are hosting docs website in a subdirectory of your main website, and in which case you probably do not need a link in the logo to the main website will open in a new tab.

To improve dark mode support, you can also set a different logo for this mode.

Accepted fields:

Name	Type	Default	Description
alt	string	undefined	Alt tag for the logo image.
src	string	<b>Required</b>	URL to the logo image. Base URL is appended by default.
srcDark	string	logo.src	An alternative image URL to use in dark mode.
href	string	siteConfig.baseUrl	Link to navigate to when the logo is clicked.
width	string   number	undefined	Specifies the <code>width</code> attribute.



Name	Type	Default	Description
height	string   number	undefined	Specifies the height attribute.
target	string	Calculated based on href (external links will open in a new tab, all others in the current one).	The target attribute of the link; controls whether the link is opened in a new tab, the current one, or otherwise.
className	string	undefined	CSS class applied to the image.
style	object	undefined	CSS inline style object. React/JSX flavor, using camelCase properties.

Example configuration:

docusaurus.config.js

```
export default {
 themeConfig: {
 navbar: {
 title: 'Site Title',
 logo: {
 alt: 'Site Logo',
 src: 'img/logo.svg',
 srcDark: 'img/logo_dark.svg',
 href: 'https://docusaurus.io/',
 target: '_self',
 width: 32,
 height: 32,
 className: 'custom-navbar-logo-class',
 style: {border: 'solid red'},
 },
 },
 },
};
```

## Navbar items

You can add items to the navbar via `themeConfig.navbar.items`.

docusaurus.config.js



```
export default {
 themeConfig: {
 navbar: {
 items: [
 {
 type: 'doc',
 position: 'left',
 docId: 'introduction',
 label: 'Docs',
 },
 {to: 'blog', label: 'Blog', position: 'left'},
 {
 type: 'docsVersionDropdown',
 position: 'right',
 },
 {
 type: 'localeDropdown',
 position: 'right',
 },
 {
 href: 'https://github.com/facebook/docusaurus',
 position: 'right',
 className: 'header-github-link',
 'aria-label': 'GitHub repository',
 },
],
 },
 },
};
```

The items can have different behaviors based on the `type` field. The sections below will introduce you to all the types of navbar items available.

## Navbar link

By default, Navbar items are regular links (internal or external).

React Router should automatically apply active link styling to links, but you can use `activeBasePath` in edge cases. For cases in which a link should be active on several different paths (such as when you have multiple doc folders under the same sidebar), you can use `activeBaseRegex`. `activeBaseRegex` is a more flexible alternative to `activeBasePath` and takes precedence over it -- Docusaurus parses it into a regular expression that is tested against the current URL.

Outbound (external) links automatically get `target="_blank" rel="noopener noreferrer"` attributes.

Accepted fields:



Name	Type	Default	Description
<code>type</code>	<code>'default'</code>	Optional	Sets the type of this item to a link.
<code>label</code>	<code>string</code>	<b>Required</b>	The name to be shown for this item.
<code>html</code>	<code>string</code>	Optional	Same as <code>label</code> , but renders pure HTML instead of text content.
<code>to</code>	<code>string</code>	<b>Required</b>	Client-side routing, used for navigating within the website. The <code>baseUrl</code> will be automatically prepended to this value.
<code>href</code>	<code>string</code>	<b>Required</b>	A full-page navigation, used for navigating outside of the website. <b>Only one of <code>to</code> or <code>href</code> should be used.</b>
<code>prependBaseUrl-ToHref</code>	<code>boolean</code>	<code>false</code>	Prepends the <code>baseUrl</code> to <code>href</code> values.
<code>position</code>	<code>'left'   'right'</code>	<code>'left'</code>	The side of the navbar this item should appear on.
<code>activeBasePath</code>	<code>string</code>	<code>to / href</code>	To apply the active class styling on all routes starting with this path. This usually isn't necessary.
<code>activeBaseRegex</code>	<code>string</code>	<code>undefined</code>	Alternative to <code>activeBasePath</code> if required.
<code>className</code>	<code>string</code>	<code>''</code>	Custom CSS class (for styling any item).

#### NOTE

In addition to the fields above, you can specify other arbitrary attributes that can be applied to a HTML link.

Example configuration:

```
docusaurus.config.js
```





```
export default {
 themeConfig: {
 navbar: {
 items: [
 {
 to: 'docs/introduction',
 // Only one of "to" or "href" should be used
 // href: 'https://www.facebook.com',
 label: 'Introduction',
 // Only one of "label" or "html" should be used
 // html: 'Introduction'
 position: 'left',
 activeBaseRegex: 'docs/(next|v8)',
 target: '_blank',
 },
],
 },
 },
};
```

## Navbar dropdown

Navbar items of the type `dropdown` has the additional `items` field, an inner array of navbar items.

Navbar dropdown items only accept the following **"link-like" item types**:

- [Navbar link](#)
- [Navbar doc link](#)
- [Navbar docs version](#)
- [Navbar doc sidebar](#)
- [Navbar with custom HTML](#)

Note that the dropdown base item is a clickable link as well, so this item can receive any of the props of a [plain navbar link](#).

Accepted fields:

Name	Type	Default	Description
<code>type</code>	<code>'dropdown'</code>	Optional	Sets the type of this item to a dropdown.
<code>label</code>	<code>string</code>	<b>Required</b>	The name to be shown for this item.
<code>items</code>	<code>LinkLikeItem[]</code>	<b>Required</b>	The items to be contained in the dropdown.
<code>position</code>	<code>'left'   'right'</code>	<code>'left'</code>	The side of the navbar this item should appear on.

Example configuration:



docusaurus.config.js

```
export default {
 themeConfig: {
 navbar: {
 items: [
 {
 type: 'dropdown',
 label: 'Community',
 position: 'left',
 items: [
 {
 label: 'Facebook',
 href: 'https://www.facebook.com',
 },
 {
 type: 'doc',
 label: 'Social',
 docId: 'social',
 },
 // ... more items
],
 },
],
 },
 },
};
```

## Navbar doc link

If you want to link to a specific doc, this special navbar item type will render the link to the doc of the provided `docId`. It will get the class `navbar__link--active` as long as you browse a doc of the same sidebar.

Accepted fields:

Name	Type	Default	Description
<code>type</code>	<code>'doc'</code>	<b>Required</b>	Sets the type of this item to a doc link.
<code>docId</code>	string	<b>Required</b>	The ID of the doc that this item links to.
<code>label</code>	string	<code>docId</code>	The name to be shown for this item.
<code>position</code>	<code>'left'</code>   <code>'right'</code>	<code>'left'</code>	The side of the navbar this item should appear on.
<code>docsPluginId</code>	string	<code>'default'</code>	The ID of the docs plugin that the doc belongs to.

Example configuration:



docusaurus.config.js

```
export default {
 themeConfig: {
 navbar: {
 items: [
 {
 type: 'doc',
 position: 'left',
 docId: 'introduction',
 label: 'Docs',
 },
],
 },
 },
};
```

## Navbar linked to a sidebar

You can link a navbar item to the first document link (which can be a doc link or a generated category index) of a given sidebar without having to hardcode a doc ID.

Accepted fields:

Name	Type	Default	Description
type	'docSidebar'	<b>Required</b>	Sets the type of this navbar item to a sidebar's first document.
sidebarId	string	<b>Required</b>	The ID of the sidebar that this item is linked to.
label	string	First document link's sidebar label	The name to be shown for this item.
position	'left'   'right'	'left'	The side of the navbar this item should appear on.
docsPluginId	string	'default'	The ID of the docs plugin that the sidebar belongs to.



### TIP

Use this navbar item type if your sidebar is updated often and the order is not stable.

Example configuration:

docusaurus.config.js



```
export default {
 themeConfig: {
 navbar: {
 items: [
 {
 type: 'docSidebar',
 position: 'left',
 sidebarId: 'api',
 label: 'API',
 },
],
 },
 },
};
```

sidebars.js

```
export default {
 tutorial: [
 {
 type: 'autogenerated',
 dirName: 'guides',
 },
],
 api: [
 'cli', // The navbar item will be linking to this doc
 'docusaurus-core',
 {
 type: 'autogenerated',
 dirName: 'api',
 },
],
};
```

## Navbar docs version dropdown

If you use docs with versioning, this special navbar item type that will render a dropdown with all your site's available versions.

The user will be able to switch from one version to another, while staying on the same doc (as long as the doc ID is constant across versions).

Accepted fields:



Name	Type	Default	Description
type	'docsVersionDropdown'	<b>Required</b>	Sets the type of this item to a docs version dropdown.
position	'left'   'right'	'left'	The side of the navbar this item should appear on.
dropdownItemsBefore	<a href="#">LinkLikeItem</a> []	[]	Add additional dropdown items at the beginning of the dropdown.
dropdownItemsAfter	<a href="#">LinkLikeItem</a> []	[]	Add additional dropdown items at the end of the dropdown.
docsPluginId	string	'default'	The ID of the docs plugin that the doc versioning belongs to.
dropdownActiveClassDisabled	boolean	false	Do not add the link active class when browsing docs.
versions	DropdownVersions	undefined	Specify a custom list of versions to include in the dropdown. See <a href="#">the versioning guide</a> for details.

Types:

```
type DropdownVersion = {
 /** Allows you to provide a custom display label for each version. */
 label?: string;
};

type DropdownVersions = string[] | {[versionName: string]:
 DropdownVersion};
```

Example configuration:

```
docusaurus.config.js
```



```
export default {
 themeConfig: {
 navbar: {
 items: [
 {
 type: 'docsVersionDropdown',
 position: 'left',
 dropdownItemsAfter: [{to: '/versions', label: 'All versions'}],
 dropdownActiveClassDisabled: true,
 },
],
 },
 },
};
```

## Navbar docs version

If you use docs with versioning, this special navbar item type will link to the active/browsed version of your doc (depends on the current URL), and fallback to the latest version.

Accepted fields:

Name	Type	Default	Description
type	'docsVersion'	<b>Required</b>	Sets the type of this item to a doc version link.
label	string	The active/latest version label.	The name to be shown for this item.
to	string	The active/latest version.	The internal link that this item points to.
position	'left'   'right'	'left'	The side of the navbar this item should appear on.
docsPluginId	string	'default'	The ID of the docs plugin that the doc versioning belongs to.

Example configuration:

```
docusaurus.config.js
```



```
export default {
 themeConfig: {
 navbar: {
 items: [
 {
 type: 'docsVersion',
 position: 'left',
 to: '/path',
 label: 'label',
 },
],
 },
 },
};
```

## Navbar locale dropdown

If you use the [i18n feature](#), this special navbar item type will render a dropdown with all your site's available locales.

The user will be able to switch from one locale to another, while staying on the same page.

Accepted fields:

Name	Type	Default	Description
type	'localeDropdown'	<b>Required</b>	Sets the type of this item to a locale dropdown.
position	'left'   'right'	'left'	The side of the navbar this item should appear on.
dropdownItemsBefore	<a href="#">LinkLikeItem</a> []	[]	Add additional dropdown items at the beginning of the dropdown.
dropdownItemsAfter	<a href="#">LinkLikeItem</a> []	[]	Add additional dropdown items at the end of the dropdown.
queryString	string	undefined	The query string to be appended to the URL.

Example configuration:

```
docusaurus.config.js
```



```
export default {
 themeConfig: {
 navbar: {
 items: [
 {
 type: 'localeDropdown',
 position: 'left',
 dropdownItemsAfter: [
 {
 to: 'https://my-site.com/help-us-translate',
 label: 'Help us translate',
 },
],
 },
],
 },
 },
};
```

## Navbar search

If you use the [search](#), the search bar will be the rightmost element in the navbar.

However, with this special navbar item type, you can change the default location.

Name	Type	Default	Description
type	'search'	<b>Required</b>	Sets the type of this item to a search bar.
position	'left'   'right'	'left'	The side of the navbar this item should appear on.
className	string	/	Custom CSS class for this navbar item.

docusaurus.config.js

```
export default {
 themeConfig: {
 navbar: {
 items: [
 {
 type: 'search',
 position: 'right',
 },
],
 },
 },
};
```





## Navbar with custom HTML

You can also render your own HTML markup inside a navbar item using this navbar item type.

Name	Type	Default	Description
type	'html'	<b>Required</b>	Sets the type of this item to a HTML element.
position	'left'   'right'	'left'	The side of the navbar this item should appear on.
className	string	''	Custom CSS class for this navbar item.
value	string	''	Custom HTML to be rendered inside this navbar item.

docusaurus.config.js

```
export default {
 themeConfig: {
 navbar: {
 items: [
 {
 type: 'html',
 position: 'right',
 value: '<button>Give feedback</button>',
 },
],
 },
 },
};
```

## Auto-hide sticky navbar

You can enable this cool UI feature that automatically hides the navbar when a user starts scrolling down the page, and show it again when the user scrolls up.

docusaurus.config.js

```
export default {
 themeConfig: {
 navbar: {
 hideOnScroll: true,
 },
 },
};
```

## Navbar style



You can set the static Navbar style without disabling the theme switching ability. The selected style will always apply no matter which theme user have selected.

Currently, there are two possible style options: `dark` and `primary` (based on the `--ifm-color-primary` color). You can see the styles preview in the [Infima documentation](#).

docusaurus.config.js

```
export default {
 themeConfig: {
 navbar: {
 style: 'primary',
 },
 },
};
```

## CodeBlock

Docusaurus uses [Prism React Renderer](#) to highlight code blocks. All configuration are in the `prism` object.

Accepted fields:

Name	Type	Default	Description
<code>theme</code>	<code>PrismTheme</code>	<code>palenight</code>	The Prism theme to use for light-theme code blocks.
<code>darkTheme</code>	<code>PrismTheme</code>	<code>palenight</code>	The Prism theme to use for dark-theme code blocks.
<code>defaultLanguage</code>	<code>string</code>	<code>undefined</code>	The default language to use for code blocks not declaring any explicit language.
<code>magicComments</code>	<code>MagicCommentConfig</code> <code>[]</code>	<i>see below</i>	The list of <a href="#">magic comments</a> .

```
type MagicCommentConfig = {
 className: string;
 line?: string;
 block?: {start: string; end: string};
};
```



```
const defaultMagicComments = [
 {
 className: 'theme-code-block-highlighted-line',
 line: 'highlight-next-line',
 block: {start: 'highlight-start', end: 'highlight-end'}},
],
];
```

## Theme

By default, we use [Palenight](#) as syntax highlighting theme. You can specify a custom theme from the [list of available themes](#). You may also use a different syntax highlighting theme when the site is in dark mode.

Example configuration:

docusaurus.config.js

```
import {themes as prismThemes} from 'prism-react-renderer';

export default {
 themeConfig: {
 prism: {
 theme: prismThemes.github,
 darkTheme: prismThemes.dracula,
 },
 },
};
```

### NOTE

If you use the line highlighting Markdown syntax, you might need to specify a different highlight background color for the dark mode syntax highlighting theme. Refer to the [docs for guidance](#).

## Default language

You can set a default language for code blocks if no language is added after the opening triple backticks (i.e. ````). Note that a valid [language name](#) must be passed.

Example configuration:

docusaurus.config.js



```
export default {
 themeConfig: {
 prism: {
 defaultLanguage: 'javascript',
 },
 },
};
```

## Footer

You can add logo and a copyright to the footer via `themeConfig.footer`. Logo can be placed in [static folder](#). Logo URL works in the same way of the navbar logo.

Accepted fields:

Name	Type	Default	Description
logo	Logo	undefined	Customization of the logo object. See <a href="#">Navbar logo</a> for details.
copyright	string	undefined	The copyright message to be displayed at the bottom, also supports custom HTML.
style	'dark'   'light'	'light'	The color theme of the footer component.
links	(Column   FooterLink) []	[]	The link groups to be present.

Example configuration:

docusaurus.config.js

```
export default {
 themeConfig: {
 footer: {
 logo: {
 alt: 'Meta Open Source Logo',
 src: 'img/meta_oss_logo.png',
 href: 'https://opensource.fb.com',
 width: 160,
 height: 51,
 },
 copyright: `Copyright © ${new Date().getFullYear()} My Project, Inc.
Built with Docusaurus.`,
 },
 },
};
```



## Footer Links

You can add links to the footer via `themeConfig.footer.links`. There are two types of footer configurations: **multi-column footers** and **simple footers**.

Multi-column footer links have a `title` and a list of `FooterItems` for each column.

Name	Type	Default	Description
<code>title</code>	<code>string</code>	<code>undefined</code>	Label of the section of these links.
<code>items</code>	<code>FooterItem[]</code>	<code>[]</code>	Links in this section.

Accepted fields of each `FooterItem`:

Name	Type	Default	Description
<code>label</code>	<code>string</code>	<b>Required</b>	Text to be displayed for this link.
<code>to</code>	<code>string</code>	<b>Required</b>	Client-side routing, used for navigating within the website. The <code>baseUrl</code> will be automatically prepended to this value.
<code>href</code>	<code>string</code>	<b>Required</b>	A full-page navigation, used for navigating outside of the website. <b>Only one of <code>to</code> or <code>href</code> should be used.</b>
<code>html</code>	<code>string</code>	<code>undefined</code>	Renders the HTML pass-through instead of a simple link. In case <code>html</code> is used, no other options should be provided.

Example multi-column configuration:

```
docusaurus.config.js
```



```
export default {
 footer: {
 links: [
 {
 title: 'Docs',
 items: [
 {
 label: 'Style Guide',
 to: 'docs/',
 },
 {
 label: 'Second Doc',
 to: 'docs/doc2/',
 },
],
 },
 {
 title: 'Community',
 items: [
 {
 label: 'Stack Overflow',
 href: 'https://stackoverflow.com/questions/tagged/docusaurus',
 },
 {
 label: 'Discord',
 href: 'https://discordapp.com/invite/docusaurus',
 },
 {
 label: 'X',
 href: 'https://x.com/docusaurus',
 },
 {
 html: `
 <a href="https://www.netlify.com" target="_blank"
rel="noreferrer noopener" aria-label="Deploys by Netlify">

 `,
 },
],
 },
],
 },
};
```

A simple footer just has a list of `FooterItem`s displayed in a row.



Example simple configuration:

docusaurus.config.js

```
export default {
 footer: {
 links: [
 {
 label: 'Stack Overflow',
 href: 'https://stackoverflow.com/questions/tagged/docusaurus',
 },
 {
 label: 'Discord',
 href: 'https://discordapp.com/invite/docusaurus',
 },
 {
 label: 'X',
 href: 'https://x.com/docusaurus',
 },
 {
 html: `
 <a href="https://www.netlify.com" target="_blank"
 rel="noreferrer noopener" aria-label="Deploys by Netlify">

 `,
 },
],
 },
};
```

## Table of Contents

You can adjust the default table of contents via `themeConfig.tableOfContents`.

Name	Type	Default	Description
<code>minHeadingLevel</code>	number	2	The minimum heading level shown in the table of contents. Must be between 2 and 6 and lower or equal to the max value.
<code>maxHeadingLevel</code>	number	3	Max heading level displayed in the TOC. Should be an integer between 2 and 6.

Example configuration:

docusaurus.config.js



```
export default {
 themeConfig: {
 tableOfContents: {
 minHeadingLevel: 2,
 maxHeadingLevel: 5,
 },
 },
};
```

## Hooks

### useColorMode

A React hook to access the color context. This context contains functions for selecting light/dark/system mode and exposes the current color mode and the choice from the user. The color mode values **should not be used for dynamic content rendering** (see below).

Usage example:

```
import {useColorMode} from '@docusaurus/theme-common';

const MyColorModeButton = () => {
 const {
 colorMode, // the "effective" color mode, never null
 colorModeChoice, // the color mode chosen by the user, can be null
 setColorMode, // set the color mode chosen by the user
 } = useColorMode();

 return (
 <button
 onClick={() => {
 const nextColorMode = colorModeChoice === 'dark' ? 'light' :
'dark';
 setColorMode(nextColorMode);
 }}>
 Toggle color mode
 </button>
);
};
```

Attributes:

- `colorMode: 'light' | 'dark'`: The effective color mode currently applied to the UI. It cannot be null.





- `colorModeChoice: 'light' | 'dark' | null`: The color mode explicitly chosen by the user. It can be `null` if user has not made any choice yet, or if they reset their choice to the system/default value.
- `setColorMode(colorModeChoice: 'light' | 'dark' | null, options: {persist: boolean}): void`: A function to call when the user explicitly chose a color mode. `null` permits to reset the choice to the system/default value. By default, the choice is persisted in `localStorage` and restored on page reload, but you can opt out with `{persist: false}`.

### ⚠ WARNING

Don't use `colorMode` and `colorModeChoice` while rendering React components. Doing so is likely to produce [FOUC](#), layout shifts and [React hydration](#) mismatches if you use them to render JSX content dynamically.

However, these values are safe to use **after React hydration**, in `useEffect` and event listeners, like in the `MyColorModeButton` example above.

If you need to render content dynamically depending on the current theme, the only way to avoid FOUC, layout shifts and hydration mismatch is to rely on CSS selectors to render content dynamically, based on the `html` data attributes that we set before the page displays anything:

```
<html data-theme="<light | dark>" data-theme-choice="<light | dark |
system>">
 <!-- content -->
</html>
```

```
[data-theme='light']
[data-theme='dark']

[data-theme-choice='light']
[data-theme-choice='dark']
[data-theme-choice='system']
```

### Why are `colorMode` and `colorModeChoice` unsafe when rendering?

To understand the problem, you need to understand how [React hydration](#) works.

During the static site generation phase, Docusaurus doesn't know what the user color mode choice is, and `useColorMode()` returns the following static values:



- `colorMode = themeConfig.colorMode.defaultMode`
- `colorModeChoice = null`

During the very first React client-side render (the hydration), React must produce the exact same HTML markup, and will also use these static values.

The correct `colorMode` and `colorModeChoice` values will only be provided in the second React render.

Typically, the following component will lead to **React hydration mismatches**. The label may switch from `light` to `dark` while React hydrates, leading to a confusing user experience.

```
import {useColorMode} from '@docusaurus/theme-common';

const DisplayCurrentColorMode = () => {
 const {colorMode} = useColorMode();
 return {colorMode};
};
```

### **i** NOTE

The component calling `useColorMode` must be a child of the `Layout` component.

```
function ExamplePage() {
 return (
 <Layout>
 <Example />
 </Layout>
);
}
```

## i18n

Read the [i18n introduction](#) first.

### Translation files location

- **Base path:** `website/i18n/[locale]/docusaurus-theme-[themeName]`
- **Multi-instance path:** N/A
- **JSON files:** extracted with `docusaurus write-translations`
- **Markdown files:** N/A



## Example file-system structure

```
website/i18n/[locale]/docusaurus-theme-classic
├── # translations for the theme
├── navbar.json
└── footer.json
```

## 2.8.2 theme-classic

The classic theme for Docusaurus.

You can refer to the [theme configuration page](#) for more details on the configuration.

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)

```
npm install --save @docusaurus/theme-classic
```

**TIP**

If you have installed `@docusaurus/preset-classic`, you don't need to install it as a dependency.

## Configuration

Accepted fields:

Option	Type	Default	Description
<code>customCss</code>	<code>string[]</code>   <code>string</code>	<code>[]</code>	Stylesheets to be imported globally as <a href="#">client modules</a> . Relative paths are resolved against the site directory.

**NOTE**

Most configuration for the theme is done in `themeConfig`, which can be found in [theme configuration](#).

## Example configuration

You can configure this theme through preset options or plugin options.

 **TIP**

Most Docusaurus users configure this plugin through the preset options.

**Preset options****Plugin options**

If you use a preset, configure this plugin through the [preset options](#):

docusaurus.config.js

```
module.exports = {
 presets: [
 [
 '@docusaurus/preset-classic',
 {
 theme: {
 customCss: './src/css/custom.css',
 },
],
],
],
};
```

## 2.8.3 theme-live-codeblock

This theme provides a `@theme/CodeBlock` component that is powered by [react-live](#). You can read more on [interactive code editor](#) documentation.

**npm**

Yarn

pnpm

Bun

```
npm install --save @docusaurus/theme-live-codeblock
```

## Configuration

docusaurus.config.js



```
export default {
 plugins: ['@docusaurus/theme-live-codeblock'],
 themeConfig: {
 liveCodeBlock: {
 /**
 * The position of the live playground, above or under the editor
 * Possible values: "top" | "bottom"
 */
 playgroundPosition: 'bottom',
 },
 },
};
```

## 2.8.4 theme-search-algolia

This theme provides a `@theme/SearchBar` component that integrates with Algolia DocSearch easily. Combined with `@docusaurus/theme-classic`, it provides a very easy search integration. You can read more on [search](#) documentation.

[npm](#)   [Yarn](#)   [pnpm](#)   [Bun](#)

```
npm install --save @docusaurus/theme-search-algolia
```

This theme also adds search page available at `/search` (as swizzlable `SearchPage` component) path with OpenSearch support. You can change this default path via `themeConfig.algolia.searchPagePath`. Use `false` to disable search page.



### TIP

If you have installed `@docusaurus/preset-classic`, you don't need to install it as a dependency.

## 2.8.5 theme-mermaid

This theme provides a `@theme/Mermaid` component that is powered by [mermaid](#). You can read more on [diagrams](#) documentation.

[npm](#)   [Yarn](#)   [pnpm](#)   [Bun](#)



```
npm install --save @docusaurus/theme-mermaid
```

## Configuration

docusaurus.config.js

```
export default {
 themes: ['@docusaurus/theme-mermaid'],
 // In order for Mermaid code blocks in Markdown to work,
 // you also need to enable the Remark plugin with this option
 markdown: {
 mermaid: true,
 },
};
```

## 2.9 Miscellaneous

### 2.9.1 create-docusaurus

A scaffolding utility to help you instantly set up a functional Docusaurus app.

## Usage

**npm**   Yarn   pnpm   Bun

```
npx create-docusaurus@latest [name] [template] [rootDir]
```

The `name` argument will be used as the site's path as well as the `name` field in the created app's `package.json`. It can be an absolute path, or a path relative to `rootDir`.

The `template` argument can be one of the following:

- `classic`: Uses the classic template (recommended)
- `facebook`: Uses the Facebook/Meta template, which contains some Meta-specific setup
- A git repo URL (beginning with `https://` or `git@`), which can be cloned to the destination
- A local file path relative to CWD, which contains the files to be copied to destination

The `rootDir` will be used to resolve the absolute path to the site directory. The default is CWD.



### ⚠ WARNING

This command should be preferably used in an interactive shell so all features are available.

## Options

### **-t, --typescript**

Used when the template argument is a recognized name. Currently, only `classic` provides a TypeScript variant.

### **-g, --git-strategy**

Used when the template argument is a git repo. It needs to be one of:

- `deep`: preserves full git history
- `shallow`: clones with `--depth=1`
- `copy`: does a shallow clone, but does not create a git repo
- `custom`: enter your custom git clone command. We will prompt you for it. You can write something like `git clone --depth 10`, and we will append the repository URL and destination directory.

### **-p, --package-manager**

Value should be one of `npm`, `yarn`, `pnpm`, or `bun`. If it's not explicitly provided, Docusaurus will infer one based on:

- The lockfile already present in the CWD (e.g. if you are setting up website in an existing project)
- The command used to invoke `create-docusaurus` (e.g. `npm init`, `npx`, `yarn create`, `bunx`, etc.)
- Interactive prompting, in case all heuristics are not present

### **-s, --skip-install**

If provided, Docusaurus will not automatically install dependencies after creating the app. The `--package-manager` option is only useful when you are actually installing dependencies.

## 2.9.2 eslint-plugin

[ESLint](#) is a tool that statically analyzes your code and reports problems or suggests best practices through editor hints and command line. Docusaurus provides an ESLint plugin to enforce best Docusaurus practices.

This ESLint plugin supports ESLint `>= 8.57.0`, flat configs and legacy configs.

## Installation

[npm](#)[Yarn](#)[pnpm](#)[Bun](#)



```
npm install --save-dev @docusaurus/eslint-plugin
```

## Supported configs

Both built-in configs exist in flat and legacy variants:

- Recommended: recommended rule set for most Docusaurus sites that should be extended from.
- All: **all** rules enabled. This will change between minor versions, so you should not use this if you want to avoid unexpected breaking changes.

## Supported rules

Name	Description	
<code>@docusaurus/no-untranslated-text</code>	Enforce text labels in JSX to be wrapped by <code>translate</code> calls	
<code>@docusaurus/string-literal-i18n-messages</code>	Enforce <code>translate</code> APIs to be called on plain text labels	✓
<code>@docusaurus/no-html-links</code>	Ensures <code>@docusaurus/Link</code> is used instead of <code>&lt;a&gt;</code> tags	✓
<code>@docusaurus/prefer-docusaurus-heading</code>	Ensures <code>@theme/Heading</code> is used instead of <code>&lt;h&gt;</code> tags for headings	✓

✓ = recommended

## Usage - Flat

### Recommended config

Import `@docusaurus/eslint-plugin` and add `docusaurus.configs.flat.recommended` to your flat config array:

eslint.config.js

```
import {defineConfig} from 'eslint/config';
import docusaurus from '@docusaurus/eslint-plugin';

export default defineConfig(
 docusaurus.configs.flat.recommended,
 {
 // Other config
 },
);
```





This will enable the `@docusaurus` eslint plugin and use the `recommended` config. See [Supported rules](#) above for a list of rules that this will enable.

## Manual config

For more fine-grained control, you can also enable the plugin manually and configure the rules you want to use directly:

eslint.config.js

```
import {defineConfig} from 'eslint/config';
import docusaurus from '@docusaurus/eslint-plugin';

export default defineConfig({
 plugins: {docusaurus},
 rules: {
 '@docusaurus/string-literal-i18n-messages': 'error',
 '@docusaurus/no-untranslated-text': 'warn',
 },
});
```

## Usage - Legacy

### Recommended config

Add `plugin:@docusaurus/recommended` to the `extends` section of your `.eslintrc` configuration file:

.eslintrc

```
{
 "extends": ["plugin:@docusaurus/recommended"]
}
```

This will enable the `@docusaurus` eslint plugin and use the `recommended` config. See [Supported rules](#) above for a list of rules that this will enable.

## Manual config

For more fine-grained control, you can also enable the plugin manually and configure the rules you want to use directly:

.eslintrc



```
{
 "plugins": ["@docusaurus"],
 "rules": {
 "@docusaurus/string-literal-i18n-messages": "error",
 "@docusaurus/no-untranslated-text": "warn"
 }
}
```

## 2.9.3 no-html-links

Ensure that the Docusaurus `<Link>` component is used instead of `<a>` tags.

The `<Link>` component has prefetching and preloading built-in. It also does build-time broken link detection, and helps Docusaurus understand your site's structure better.

## Rule Details

Examples of **incorrect** code for this rule:

```
go to page!

X
```

Examples of **correct** code for this rule:

```
import Link from '@docusaurus/Link'

<Link to="/page">go to page!</Link>

<Link to="https://x.com/docusaurus">X</Link>
```

## Rule Configuration

Accepted fields:

Option	Type	Default	Description
<code>ignoreFullyResolved</code>	<code>boolean</code>	<code>false</code>	Set to true will not report any <code>&lt;a&gt;</code> tags with absolute URLs including a protocol.

## 2.9.4 no-untranslated-text

Enforce text labels in JSX to be wrapped by translate calls.



When the [i18n feature](#) is used, this rule ensures that all labels appearing on the website are translatable, so no string accidentally slips through untranslated.

## Rule Details

Examples of **incorrect** code for this rule:

```
// Hello World is not translated
<Component>Hello World</Component>
```

Examples of **correct** code for this rule:

```
// Hello World is translated
<Component>
 <Translate>Hello World</Translate>
</Component>
```

## Rule Configuration

Accepted fields:

Option	Type	Default	Description
<code>ignoredStrings</code>	<code>string[]</code>	<code>[]</code>	Text labels that only contain strings in this list will not be reported.

## When Not To Use It

If you're not using the [i18n feature](#), you can disable this rule. You can also disable this rule where the text is not supposed to be translated.

## Further Reading

- <https://docusaurus.io/docs/docusaurus-core#translate>
- <https://docusaurus.io/docs/docusaurus-core#translate-imperative>

### 2.9.5 prefer-docusaurus-heading

Ensures that the `@theme/Heading` theme component provided by Docusaurus `theme-classic` is used instead of `<hn>` tags for headings.

## Rule Details

Examples of **incorrect** code for this rule:



```
<h1>This is heading 1</h1>

<h2>This is heading 2</h2>

<h3>This is heading 3</h3>
```

Examples of **correct** code for this rule:

```
import Heading from '@theme/Heading'

<Heading as='h1'>This is heading 1</Heading>

<Heading as='h2'>This is heading 2</Heading>

<Heading as='h3'>This is heading 3</Heading>
```

## 2.9.6 string-literal-i18n-messages

Enforce translate APIs to be called on plain text labels.

Docusaurus offers the [docusaurus write-translations](#) API, which statically extracts the text labels marked as translatable. Dynamic values used in `<Translate>` or `translate()` calls will fail to be extracted. This rule will ensure that all translate calls are statically extractable.

## Rule Details

Examples of **incorrect** code for this rule:

```
const text = 'Some text to be translated'

// Invalid <Translate> child
<Translate>{text}</Translate>

// Invalid message attribute
translate({message: text})
```

Examples of **correct** code for this rule:



```
// Valid <Translate> child
<Translate>Some text to be translated</Translate>

// Valid message attribute
translate({message: 'Some text to be translated'})

// Valid <Translate> child using values object as prop
<Translate values={{firstName: 'Sébastien'}}>
 {'Welcome, {firstName}! How are you?'}
</Translate>

// Valid message attribute using values object as second argument
translate({message: 'The logo of site {siteName}'}, {siteName:
 'Docusaurus'})
```

## When Not To Use It

If you're not using the [i18n feature](#), you can disable this rule.

## Further Reading

- <https://docusaurus.io/docs/docusaurus-core#translate>
- <https://docusaurus.io/docs/docusaurus-core#translate-imperative>

## 2.9.7 logger

An encapsulated logger for semantically formatting console messages.

Authors of packages in the Docusaurus ecosystem are encouraged to use this package to provide unified log formats.

## APIs

It exports a single object as default export: `logger`. `logger` has the following properties:

- Some useful colors.
  - `red`
  - `yellow`
  - `green`
  - `bold`
  - `dim`
- Formatters. These functions all have the signature `(msg: unknown) => string`. Note that their implementations are not guaranteed. You should only care about their semantics.
  - `path`: formats a file path.
  - `url`: formats a URL.
  - `name`: formats an identifier.



- `code`: formats a code snippet.
- `subdue`: subdues the text.
- `num`: formats a number.
- The `interpolate` function. It is a template literal tag. The syntax can be found below.
- Logging functions. All logging functions can both be used as normal functions (similar to the `console.log` family, but only accepts one parameter) or template literal tags.
  - `info`: prints information.
  - `warn`: prints a warning that should be paid attention to.
  - `error`: prints an error (not necessarily halting the program) that signals significant problems.
  - `success`: prints a success message.
- The `report` function. It takes a `ReportingSeverity` value (`ignore`, `log`, `warn`, `throw`) and reports a message according to the severity.

### ! A WORD ON THE `error` FORMATTER

Beware that an `error` message, even when it doesn't hang the program, is likely going to cause confusion. When users inspect logs and find an `[ERROR]`, even when the build succeeds, they will assume something is going wrong. Use it sparingly.

Docusaurus only uses `logger.error` when printing messages immediately before throwing an error, or when user has set the reporting severity of `onBrokenLink`, etc. to `"error"`.

In addition, `warn` and `error` will color the **entire** message for better attention. If you are printing large blocks of help text about an error, better use `logger.info`.

## Using the template literal tag

The template literal tag evaluates the template and expressions embedded. `interpolate` returns a new string, while other logging functions prints it. Below is a typical usage:

```
import logger from '@docusaurus/logger';

logger.info`Hello name=${name}! You have number=${money} dollars. Here are
the ${
 items.length > 1 ? 'items' : 'item'
} on the shelf: ${items}
To buy anything, enter code=${'buy x'} where code=${'x'} is the item's
name; to quit, press code=${'Ctrl + C'}.`;
```

An embedded expression is optionally preceded by a flag in the form `[a-z]+=` (a few lowercase letters, followed by an equals sign, directly preceding the embedded expression). If the expression is not preceded by any flag, it's printed out as-is. Otherwise, it's formatted with one of the formatters:

- `path=`: `path`
- `url=`: `url`



- name=: name
- code=: code
- subdue=: subdue
- number=: num

If the expression is an array, it's formatted by ``\n- ${array.join('\n- ')}\n`` (note it automatically gets a leading line end). Each member is formatted by itself and the bullet is not formatted. So you would see the above message printed as:

```
[INFO] Hello Josh! You have 100 dollars. Here are the items on the shelf:
- Hotdogs
- Macbooks
- Vinyl discs
To buy anything, enter `buy x` where `x` is the item's name; to quit, press `Ctrl + C`.
```