API

GENERATED: 2026-09-01

Contents

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"
  }
}
```

1.1 Docusaurus CLI commands

Below is a list of Docusaurus CLI commands and their usages:

1.1.1 `docusaurus start [siteDir]`

Builds and serves a preview of your site locally with Webpack Dev Server.

Options

Name	Default	Description
`--port`	`3000`	Specifies the port of the dev server.
`--host`	`localhost`	Specify a host to use. For example, if you want your server to be accessible externally, you can use `--host 0.0.0.0`.
`--locale`		Specify site locale to be used.
`--hot-only`	`false`	Enables Hot Module Replacement without page refresh as a fallback in case of build failures. More information here.
`--no-open`	`false`	Do not open the page automatically in the browser.
`--config`	`undefined`	Path to Docusaurus config file, default to `[siteDir]/docusaurus.config.js`
`--poll [optionalIntervalMs]`	`false`	Use polling of files rather than watching for live reload as a fallback in environments where watching doesn't work. More information here.

Name	Default	Description
`--no-minify`	`false`	Build website without minimizing JS/CSS bundles.
`--https`	`false`	Serve the dev site over HTTPS using a self-signed certificate. Implied when both `--ssl-cert` and `--ssl-key` are provided.
`--ssl-cert`	`undefined`	Path to a TLS certificate file (implies HTTPS).
`--ssl-key`	`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
```

1.1.2 `docusaurus build [siteDir]`

Compiles your site for production.

Options

Name	Default	Description
`--dev`		Builds the website in dev mode, including full React error messages.
`--bundle-analyzer`	`false`	Analyze your bundle with the webpack bundle analyzer.
`--out-dir`	`build`	The full path for the new output directory, relative to the current workspace.
`--config`	`undefined`	Path to Docusaurus config file, default to `[siteDir]/docusaurus.config.js`
`--locale`		Build the site in the specified locale(s). If not specified, all known locales are built.
`--no-minify`	`false`	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`.

1.1.3 `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
`themeName`	The name of the theme to swizzle from.
`componentName`	The name of the theme component to swizzle.
`--list`	Display components available for swizzling
`--eject`	Eject the theme component
`--wrap`	Wrap the theme component
`--danger`	Allow immediate swizzling of unsafe components
`--typescript`	Swizzle the TypeScript variant component
`--javascript`	Swizzle the JavaScript variant component
`--config`	Path to docusaurus config file, default to `[siteDir]/docusaurus.config.js`

> **WARNING**
>
> Unsafe components have a higher risk of breaking changes due to internal refactorings.

1.1.4 `docusaurus deploy [siteDir]`

Deploys your site with GitHub Pages. Check out the docs on deployment for more details.

Options

Name	Default	Description
`--locale`		Deploy the site in the specified locale(s). If not specified, all known locales are deployed.
`--out-dir`	`build`	The full path for the new output directory, relative to the current workspace.
`--skip-build`	`false`	Deploy website without building it. This may be useful when using a custom deploy script.
`--target-dir`	`.`	Path to the target directory to deploy to.
`--config`	`undefined`	Path to Docusaurus config file, default to `[siteDir]/docusaurus.config.js`

1.1.5 `docusaurus serve [siteDir]`

Serve your built website locally.

Name	Default	Description
`--port`	`3000`	Use specified port
`--dir`	`build`	The full path for the output directory, relative to the current workspace
`--build`	`false`	Build website before serving
`--config`	`undefined`	Path to Docusaurus config file, default to `[siteDir]/docusaurus.config.js`
`--host`	`localhost`	Specify a host to use. For example, if you want your server to be accessible externally, you can use `--host 0.0.0.0`.
`--no-open`	`false` locally, `true` in CI	Do not open a browser window to the server location.

1.1.6 `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.

1.1.7 `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
`--locale`	`<defaultLocale>`	Define which locale folder you want to write translations the JSON files in
`--override`	`false`	Override existing translation messages
`--config`	`undefined`	Path to Docusaurus config file, default to `[siteDir]/docusaurus.config.js`
`--messagePrefix`	`''`	Allows adding a prefix to each translation message, to help you highlight untranslated strings

1.1.8 `docusaurus write-heading-ids [siteDir] [files]`

Add explicit heading IDs to the Markdown documents of your site.

Name	Default	Description
`files`	All MD files used by plugins	The files that you want heading IDs to be written to.
`--syntax`	`classic`	Heading ID syntax to use: `classic` (`{#id}`) or `mdx-comment` (`{/* #id */}`).
`--migrate`	`false`	Migrate existing heading IDs to the target `--syntax`, if they are using a different syntax.
`--overwrite`	`false`	Overwrite existing heading IDs, re-generate them from the heading text.
`--maintain-case`	`false`	Keep the headings' casing, otherwise make all lowercase.

2 Docusaurus Client API

Docusaurus provides some APIs on the clients that can be helpful to you when building your site.

2.1 Components

2.1.1 `<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.

2.1.2 `<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>
```

2.1.3 `<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>`.

2.1.4 `<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" />;
};
```

> **NOTE**
>
> `@docusaurus/router` implements React Router and supports its features.

2.1.5 `<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>
      {() => <span>page url = {window.location.href}</span>}
    </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>
  );
};
```

2.1.6 `<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>
  );
}
```

2.1.7 `<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 Hooks

2.2.1 `useDocusaurusContext`

React hook to access Docusaurus Context. The context contains the `siteConfig` object from docusaurus.config.js and some additional site metadata.

```ts
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:

```jsx
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.

2.2.2 `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	`useIsBrowser()`	`typeof window !== 'undefined'`
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>;
};
```

2.2.3 `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 <img src={imgSrc} />;
};
```

> **TIP**
>
> In most cases, you don't need `useBaseUrl`.

Prefer a `require()` call for assets:

```
<img src={require('@site/static/img/myImage.png').default} />
```

2.2.4 `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>;
};
```

2.2.5 `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`

2.2.6 `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>;
};
```

2.2.7 `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>;
};
```

2.2.8 `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} />;
}
```

2.3 Functions

2.3.1 `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'});
```

2.3.2 `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

```jsx
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>
  );
}
```

2.4 Modules

2.4.1 `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:

```jsx
import ExecutionEnvironment from '@docusaurus/ExecutionEnvironment';

if (ExecutionEnvironment.canUseDOM) {
  require('lib-that-only-works-client-side');
}
```

Field	Description
`ExecutionEnvironment.canUseDOM`	`true` if on client/browser, `false` on Node.js/prerendering.
`ExecutionEnvironment.canUseEventListeners`	`true` if on client and has `window.addEventListener`.
`ExecutionEnvironment.canUseIntersectionObserver`	`true` if on client and has `IntersectionObserver`.
`ExecutionEnvironment.canUseViewport`	`true` if on client and has `window.screen`.

2.4.2 `constants`

A module exposing useful constants to client-side theme code.

```
import {DEFAULT_PLUGIN_ID} from '@docusaurus/constants';
```

Named export	Value
`DEFAULT_PLUGIN_ID`	`default`

3 docusaurus.config.js

> **INFO**
>
> Refer to the Getting Started **Configuration** for examples.

3.1 Overview

`docusaurus.config.js` contains configurations for your site and is placed in the root directory of your site.

> **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.

3.2 Required fields

3.2.1 `title`

- Type: `string`

Title for your website. Will be used in metadata and as browser tab title.

docusaurus.config.js

```
export default {
  title: 'Docusaurus',
};
```

3.2.2 `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.

3.2.3 `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.

3.3 Optional fields

3.3.1 `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',
};
```

3.3.2 `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.

3.3.3 `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

3.3.4 `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`.

3.3.5 `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';
```

3.3.6 `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`
};
```

3.3.7 `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`).

3.3.8 `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.

3.3.9 `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.

3.3.10 `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`.

3.3.11 `tagline`

- Type: `string`

The tagline for your website.

docusaurus.config.js

```
export default {
  tagline:
    'Docusaurus makes it easy to maintain Open Source documentation
websites.',
};
```

3.3.12 `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',
};
```

3.3.13 `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',
};
```

3.3.14 `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',
};
```

3.3.15 `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',
};
```

3.3.16 `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',
};
```

3.3.17 `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
    },
  },
};
```

3.3.18 `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');
      },
    }),
  ],
};
```

3.3.19 `themes`

- Type: `PluginConfig[]`

docusaurus.config.js

```
export default {
  themes: ['@docusaurus/theme-classic'],
};
```

3.3.20 `presets`

- Type: `PresetConfig[]`

```
type PresetConfig = string | [string, any];
```

docusaurus.config.js

```
export default {
  presets: [],
};
```

3.3.21 `markdown`

The global Docusaurus Markdown config.

- Type: `MarkdownConfig`

```ts
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: `.md` vs `.mdx`.
`mermaid`	`boolean`	`false`	When `true`, allows Docusaurus to render Markdown code blocks with `mermaid` language as Mermaid diagrams.
`emoji`	`boolean`	`true`	When `true`, allows Docusaurus to render emoji shortcodes (e.g., `:+1:`) as Unicode emoji (👍). When `false`, emoji shortcodes are left as-is.
`preprocessor`	`MarkdownPreprocessor`	`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
`parseFrontMatter`	`ParseFrontMatter`	`undefined`	Gives you the ability to provide your own front matter parser, or to enhance the default parser. Read our front matter guide for details.
`mdx1Compat`	`MDX1CompatOptions`	`{comments: true, admonitions: true, headingIds: true}`	Compatibility options to make it easier to upgrade to Docusaurus v3+. Defaults to all `false` when `future.v4.mdx1CompatDisabledByDefault` is enabled.
`anchors`	`MarkdownAnchorsConfig`	`{maintainCase: false}`	Options to control the behavior of anchors generated from Markdown headings
`remarkRehypeOptions`	`object`	`undefined`	Makes it possible to pass custom `remark-rehype` options.
`hooks`	`MarkdownHooks`	`object`	Make it possible to customize the MDX loader behavior with callbacks or built-in options.
`hooks.onBrokenMarkdownLinks`	`ReportingSeverity | OnBrokenMarkdownLinksFunction`	`'warn'`	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 mdast node.
`hooks.onBrokenMarkdownImages`	`ReportingSeverity | OnBrokenMarkdownImagesFunction`	`'throw'`	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 mdast node.
`hooks.onUnusedMarkdownDirectives`	`ReportingSeverity | OnUnusedMarkdownDirectivesFunction`	`'warn'`	Hook to customize the behavior when encountering an unused Markdown directive. Also accepts a callback function, giving you access to the raw `Directive` AST nodes.

3.3.22 `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
```

3.3.23 `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'],
};
```

3.3.24 `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.

3.3.25 `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,
    },
  ],
};
```

3.3.26 `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.

3.3.27 `clientModules`

An array of client modules to load globally on your site.

Example:

docusaurus.config.js

```
export default {
  clientModules: ['./mySiteGlobalJs.js', './mySiteGlobalCss.css'],
};
```

3.3.28 `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>`,
};
```

3.3.29 `titleDelimiter`

- Type: `string`

Will be used as title delimiter in the generated `<title>` tag.

Example:

docusaurus.config.js

```
export default {
  titleDelimiter: '🦖', // Defaults to `|`
};
```

3.3.30 `baseUrlIssueBanner`

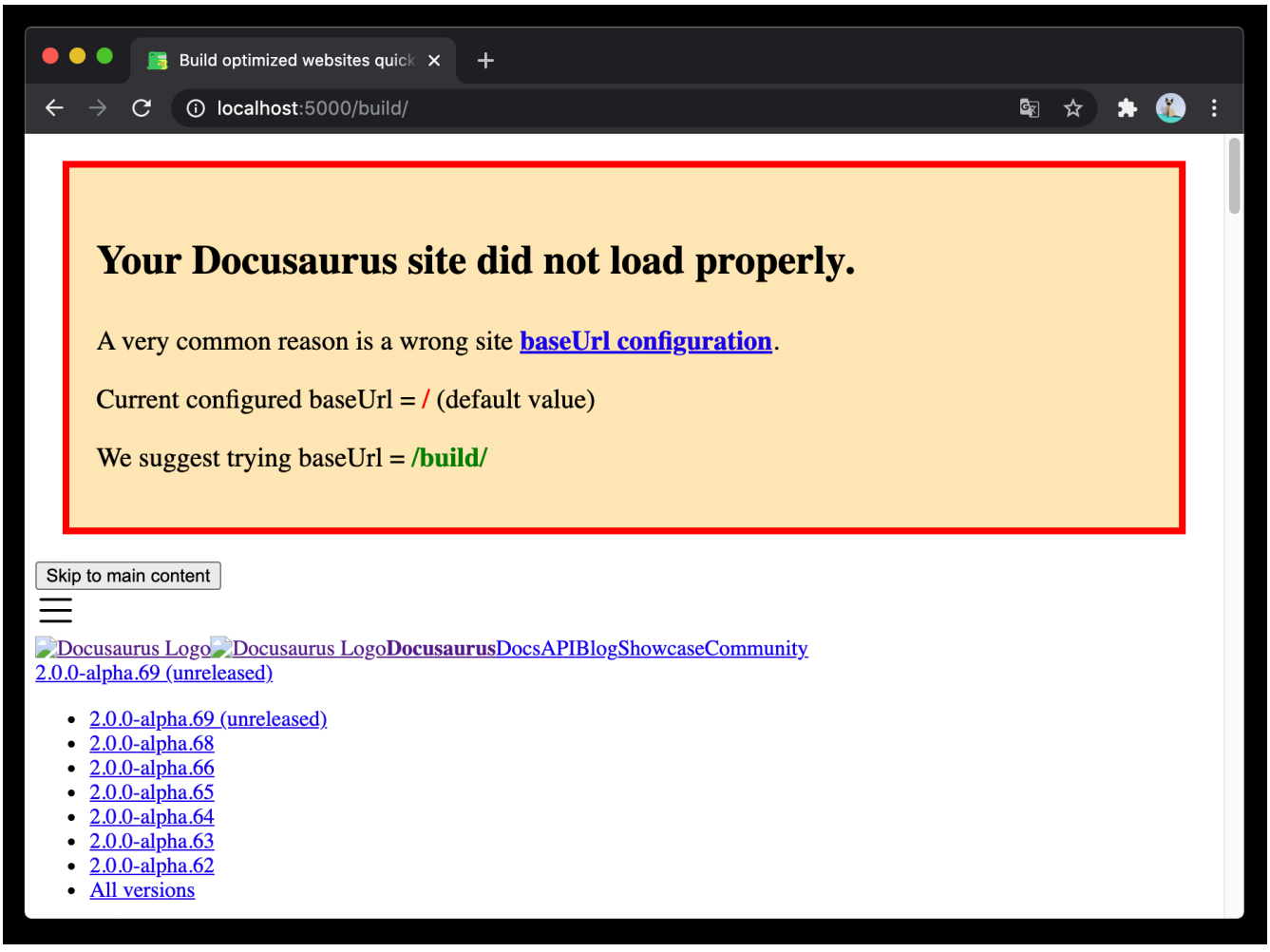
- Type: `boolean`

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.

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.

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.

4.2 Plugin constructor

The plugin module's default export is a constructor function with the signature `(context: LoadContext, options: PluginOptions) => Plugin | Promise<Plugin>`.

4.2.1 `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;
};
```

4.2.2 `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.

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;
  }
```

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.

4.4.1 `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);
    },
  };
}
```

4.4.2 `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.

```ts
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]: pathToPropModule` (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,
      });
    },
  };
}
```

4.4.3 `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/
        },
      };
    },
  };
}
```

4.4.4 `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;
    },
  };
}
```

4.4.5 `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);
      });
    },
  };
}
```

4.4.6 `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.

```ts
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.

4.4.7 `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];
    },
  };
}
```

4.5 Extending infrastructure

Docusaurus has some infrastructure like hot reloading, CLI, and swizzling, that can be extended by external plugins.

4.5.1 `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}`];
    },
  };
}
```

4.5.2 `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));
        });
    },
  };
}
```

4.5.3 `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';
    },
  };
}
```

4.5.4 `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';
    },
  };
}
```

4.5.5 `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 swizzlable 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',
  ];
}
```

4.6 I18n lifecycles

Plugins use these lifecycles to load i18n-related data.

4.6.1 `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,
          },
        },
      ];
    },
  };
}
```

4.6.2 `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,
      };
    },
  };
}
```

4.6.3 `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,
      };
    },
  };
}
```

4.6.4 `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`);
    },
  };
}
```

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.

4.7.1 `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;
}
```

4.7.2 `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;
}
```

5 Docusaurus plugins

We provide official Docusaurus plugins.

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

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

6 Plugins

6.1 📦 plugin-content-docs

Provides the Docs functionality and is the default docs plugin for Docusaurus.

6.1.1 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.

6.1.2 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 | EditUrlFunction`	`undefined`	Base URL to edit your site. The final URL is computed by `editUrl + relativeDocPath`. 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 `editUrl` is a function.
`editCurrentVersion`	`boolean`	`false`	The edit URL will always target the current version doc instead of older versions. Ignored when `editUrl` is a function.
`routeBasePath`	`string`	`'docs'`	URL route for the docs section of your site. **DO NOT** include a trailing slash. Use `/` for shipping docs without base path.
`tagsBasePath`	`string`	`'tags'`	URL route for the tags list page of your site. It is prepended to the `routeBasePath`.

Name	Type	Default	Description
`include`	`string[]`	`['**/*.{md,mdx}']`	Array of glob patterns matching Markdown files to be built, relative to the content path.
`exclude`	`string[]`	*See example configuration*	Array of glob patterns matching Markdown files to be excluded. Serves as refinement based on the `include` option.
`sidebarPath`	`false | string`	`undefined`	Path to a sidebars configuration file, loaded in a Node.js context. Use `false` to disable sidebars, or `undefined` to create a fully autogenerated sidebar.
`sidebarCollapsible`	`boolean`	`true`	Whether sidebar categories are collapsible by default. See also Collapsible categories
`sidebarCollapsed`	`boolean`	`true`	Whether sidebar categories are collapsed by default. See also Expanded categories by default
`sidebarItemsGenerator`	`SidebarGenerator`	*Omitted*	Function used to replace the sidebar items of type `'autogenerated'` with real sidebar items (docs, categories, links...). See also Customize the sidebar items generator
`numberPrefixParser`	`boolean | PrefixParser`	*Omitted*	Custom parsing logic to extract number prefixes from file names. Use `false` to disable this behavior and leave the docs untouched, and `true` to use the default parser. See also Using number prefixes
`docsRootComponent`	`string`	`'@theme/DocsRoot'`	Parent component of all the docs plugin pages (including all versions). Stays mounted when navigation between docs pages and versions.
`docVersionRootComponent`	`string`	`'@theme/DocVersionLayout'`	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.
`docRootComponent`	`string`	`'@theme/DocRoot'`	Parent component of all doc pages with sidebars (regular docs pages, category generated index pages). Stays mounted when navigation between such pages.
`docItemComponent`	`string`	`'@theme/DocItem'`	Main doc container, with TOC, pagination, etc.
`docTagsListComponent`	`string`	`'@theme/DocTagsListPage'`	Root component of the tags list page
`docTagDocListComponent`	`string`	`'@theme/DocTagDocListPage'`	Root component of the "docs containing tag X" page.
`docCategoryGeneratedIndexComponent`	`string`	`'@theme/DocCategoryGeneratedIndexPage'`	Root component of the generated category index page.
`remarkPlugins`	`any[]`	`[]`	Remark plugins passed to MDX.
`rehypePlugins`	`any[]`	`[]`	Rehype plugins passed to MDX.
`recmaPlugins`	`any[]`	`[]`	Recma plugins passed to MDX.

Name	Type	Default	Description
`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`	Whether to display the author who last updated the doc.
`showLastUpdateTime`	`boolean`	`false`	**Only for Markdown pages**. 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

```ts
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/${versionDocsDirP
ath}/${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'],
        },
      },
    ],
  ],
};
```

6.1.3 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 Autogenerated sidebar metadata.
`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 custom props 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 multiple sidebars guide 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 `null` to disable showing "Next" for this page.
`pagination_prev`	`string | null`	Previous doc in the sidebar	The ID of the documentation you want the "Previous" pagination to link to. Use `null` to disable showing "Previous" for this page.
`parse_number_prefixes`	`boolean`	`numberPrefixParser` plugin option	Whether number prefix parsing is disabled on this doc. See also Using number prefixes.

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
```

6.1.4 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:

```ts
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

```yaml
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

```mdx
---
tags: [releases, announcements, emptyTag]
---

# Title

Content
```

6.1.5 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
```

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**.

6.2.1 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.

6.2.2 Configuration

Accepted fields:

Name	Type	Default	Description
`path`	`string`	`'blog'`	Path to the blog content directory on the file system, relative to site dir.
`editUrl`	`string | EditUrlFn`	`undefined`	Base URL to edit your site. The final URL is computed by `editUrl + relativePostPath`. 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 `editUrl` is a function.
`blogTitle`	`string`	`'Blog'`	Blog page title for better SEO.
`blogDescription`	`string`	`'Blog'`	Blog page meta description for better SEO.
`blogSidebarCount`	`number | 'ALL'`	`5`	Number of blog post elements to show in the blog sidebar. `'ALL'` to show all blog posts; `0` to disable.
`blogSidebarTitle`	`string`	`'Recent posts'`	Title of the blog sidebar.
`routeBasePath`	`string`	`'blog'`	URL route for the blog section of your site. **DO NOT** include a trailing slash. Use `/` to put the blog at root path.
`tagsBasePath`	`string`	`'tags'`	URL route for the tags section of your blog. Will be appended to `routeBasePath`.
`pageBasePath`	`string`	`'page'`	URL route for the pages section of your blog. Will be appended to `routeBasePath`.
`archiveBasePath`	`string | null`	`'archive'`	URL route for the archive section of your blog. Will be appended to `routeBasePath`. **DO NOT** include a trailing slash. Use `null` to disable generation of archive.
`authorsBasePath`	`string`	`'authors'`	URL route for the authors pages of your blog. Will be appended to `path`.
`include`	`string[]`	`['**/*.{md,mdx}']`	Array of glob patterns matching Markdown files to be built, relative to the content path.
`exclude`	`string[]`	*See example configuration*	Array of glob patterns matching Markdown files to be excluded.

Name	Type	Default	Description
			Serves as refinement based on the `include` option.
`postsPerPage`	`number | 'ALL'`	`10`	Number of posts to show per page in the listing page. Use `'ALL'` to display all posts on one listing page.
`blogListComponent`	`string`	`'@theme/BlogListPage'`	Root component of the blog listing page.
`blogPostComponent`	`string`	`'@theme/BlogPostPage'`	Root component of each blog post page.
`blogTagsListComponent`	`string`	`'@theme/BlogTagsListPage'`	Root component of the tags list page.
`blogTagsPostsComponent`	`string`	`'@theme/BlogTagsPostsPage'`	Root component of the "posts containing tag" page.
`blogArchiveComponent`	`string`	`'@theme/BlogArchivePage'`	Root component of the blog archive page.
`blogAuthorsPostsComponent`	`string`	`'@theme/Blog/Pages/BlogAuthorsPostsPage'`	Root component of the blog author page.
`blogAuthorsListComponent`	`string`	`'@theme/Blog/Pages/BlogAuthorsListPage'`	Root component of the blog authors page index.
`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.
`truncateMarker`	`RegExp`	`/<!--\s*truncate\s*-->/ | \{\/\*\s*truncate\s*\*\/\}/`	Truncate marker marking where the summary ends.
`showReadingTime`	`boolean`	`true`	Show estimated reading time for the blog post.
`readingTime`	`ReadingTimeFn`	The default reading time	A callback to customize the reading time number displayed.
`authorsMapPath`	`string`	`'authors.yml'`	Path to the authors map file, relative to the blog content directory.
`feedOptions`	*See below*	`{type: ['rss', 'atom']}`	Blog feed.
`feedOptions.type`	`FeedType | FeedType[] | 'all' | null`	**Required**	Type of feed to be generated. Use `null` to disable generation.
`feedOptions.createFeedItems`	`CreateFeedItemsFn | undefined`	`undefined`	An optional function which can be used to transform and / or filter the items in the feed.

Name	Type	Default	Description
`feedOptions.limit`	`number | null | false`	`20`	Limits the feed to the specified number of posts, `false` or `null` for all entries. Defaults to `20`.
`feedOptions.title`	`string`	`siteConfig.title`	Title of the feed.
`feedOptions.description`	`string`	`` `${siteConfig.title} Blog` ``	Description of the feed.
`feedOptions.copyright`	`string`	`undefined`	Copyright message.
`feedOptions.xslt`	`boolean | FeedXSLTOptions`	`undefined`	Permits to style the blog XML feeds with XSLT so that browsers render them nicely.
`feedOptions.language`	`string` (See documentation for possible values)	`undefined`	Language metadata of the feed.
`sortPosts`	`'descending' | 'ascending'`	`'descending'`	Governs the direction of blog post sorting.
`processBlogPosts`	`ProcessBlogPostsFn`	`undefined`	An optional function which can be used to transform blog posts (filter, modify, delete, etc...).
`showLastUpdateAuthor`	`boolean`	`false`	Whether to display the author who last updated the blog post.
`showLastUpdateTime`	`boolean`	`false`	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 `actions/checkout`, use `fetch-depth: 0`. When deploying to Vercel, set the environment variable `VERCEL_DEEP_CLONE=true`.
`tags`	`string | false | null | undefined`	`tags.yml`	Path to the YAML tags file listing pre-defined tags. Relative to the blog content directory.
`onInlineTags`	`'ignore' | 'log' | 'warn' | 'throw'`	`warn`	The plugin behavior when blog posts contain inline tags (not appearing in the list of pre-defined tags, usually `tags.yml`).
`onUntruncatedBlogPosts`	`'ignore' | 'log' | 'warn' | 'throw'`	`warn`	The plugin behavior when blog posts do not contain a truncate marker.

Types

`EditUrlFn`

```ts
type EditUrlFunction = (params: {
  blogDirPath: string;
  blogPath: string;
  permalink: string;
  locale: string;
}) => string | undefined;
```

`ReadingTimeFn`

```ts
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`

```ts
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.

```ts
type FeedXSLTOptions =
  | boolean
  | undefined
  | null
  | {
      rss?: string | boolean | null | undefined;
      atom?: string | boolean | null | undefined;
    };
```

`CreateFeedItemsFn`

```ts
type CreateFeedItemsFn = (params: {
  blogPosts: BlogPost[];
  siteConfig: DocusaurusConfig;
  outDir: string;
  defaultCreateFeedItemsFn: CreateFeedItemsFn;
}) => Promise<BlogFeedItem[]>;
```

`ProcessBlogPostsFn`

```ts
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,
                });
              },
            },
          },
        },
      ],
    ],
  };
```

6.2.3 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
`authors`	`Authors`	`undefined`	List of blog post authors (or unique author). Read the `authors` guide for more explanations. Prefer `authors` over the `author_*` front matter fields, even for single author blog posts.
`author`	`string`	`undefined`	⚠️ Prefer using `authors`. The blog post author's name.
`author_url`	`string`	`undefined`	⚠️ Prefer using `authors`. The URL that the author's name will be linked to. This could be a GitHub, X, Facebook profile URL, etc.
`author_image_url`	`string`	`undefined`	⚠️ Prefer using `authors`. The URL to the author's thumbnail image.
`author_title`	`string`	`undefined`	⚠️ Prefer using `authors`. A description of the author.
`title`	`string`	Markdown title	The blog post title.
`title_meta`	`string`	`frontMatter.title`	The blog post SEO metadata title, used in `<head>` for `<title>` and `og:title`. Permits to override `title` when the displayed title and SEO title should be different.
`sidebar_label`	`string`	`title`	A custom label for the blog sidebar, replacing the default one (`title`).
`date`	`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, `2021-04-15-blog-post.mdx`, `2021-04-15-blog-post/index.mdx`, `2021/04/15/blog-post.mdx`. Otherwise, it is the Markdown file creation time.
`tags`	`Tag[]`	`undefined`	A list of strings or objects of two string fields `label` and `permalink` to tag to your post. Strings can be a reference to keys of a tags file (usually `tags.yml`)

Name	Type	Default	Description
`draft`	`boolean`	`false`	Draft blog posts will only be available during development.
`unlisted`	`boolean`	`false`	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.
`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.
`keywords`	`string[]`	`undefined`	Keywords meta tag, which will become the `<meta name="keywords" content="keyword1,keyword2,..."/>` in `<head>`, used by 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 blog post URL (`/<routeBasePath>/<slug>`). Supports multiple patterns: `slug: my-blog-post`, `slug: /my/path/to/blog/post`, slug: `/`.
`last_update`	`FrontMatterLastUpdate`	`undefined`	Allows overriding the last update author/date. Date can be any parsable date string.

```ts
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
```

6.2.4 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:

```ts
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

```yaml
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

```md
---
tags: [releases, announcements, emptyTag]
---

# Title

Content
```

6.2.5 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:

```ts
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

```yaml
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: Joel 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

```yaml
---
authors: [slorber, jmarcey]
---

# My Blog Post

Content
```

6.2.6 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
```

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.

6.3.1 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.

6.3.2 Configuration

Accepted fields:

Name	Type	Default	Description
`path`	`string`	`'src/pages'`	Path to data on filesystem relative to site dir. Components in this directory will be automatically converted to pages.
`editUrl`	`string | EditUrlFn`	`undefined`	**Only for Markdown pages**. Base URL to edit your site. The final URL is computed by `editUrl + relativePostPath`. 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`	**Only for Markdown pages**. The edit URL will target the localized file, instead of the original unlocalized file. Ignored when `editUrl` is a function.
`routeBasePath`	`string`	`'/'`	URL route for the pages section of your site. **DO NOT** include a trailing slash.
`include`	`string[]`	`['**/*.{js,jsx,ts,tsx,md,mdx}']`	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`	**Only for Markdown pages**. Whether to display the author who last updated the page.
`showLastUpdateTime`	`boolean`	`false`	**Only for Markdown pages**. 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 `fetch-depth: 0`. When deploying to Vercel, set the environment variable `VERCEL_DEEP_CLONE=true`.

Types

`EditUrlFn`

```ts
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: [],
        },
      },
    ],
  ],
};
```

6.3.3 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 `<meta name="description" content="..."/>` and `<meta property="og:description" content="..."/>` in `<head>`, used by search engines.
`keywords`	`string[]`	`undefined`	Keywords meta tag, which will become the `<meta name="keywords" content="keyword1,keyword2,..."/>` 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 page URL (`/<routeBasePath>/<slug>`). Supports multiple patterns: `slug: my-page`, `slug: /my/page`, slug: `/`.
`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
```

6.3.4 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
```

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.

6.4.1 Installation

npm | Yarn | pnpm | Bun

```
npm install --save @docusaurus/plugin-client-redirects
```

6.4.2 Configuration

Accepted fields:

Option	Type	Default	Description
`fromExtensions`	`string[]`	`[]`	The extensions to be removed from the route after redirecting.
`toExtensions`	`string[]`	`[]`	The extensions to be appended to the route after redirecting.
`redirects`	`RedirectRule[]`	`[]`	The list of redirect rules.
`createRedirects`	`CreateRedirectsFn`	`undefined`	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.

> **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`

```ts
type RedirectRule = {
  to: string;
  from: string | string[];
};
```

> **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
        },
      },
    ],
  ],
};
```

6.5 📦 plugin-debug

The debug plugin will display useful debug information at `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.

6.5.1 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.

6.5.2 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
      },
    ],
  ],
};
```

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.

6.6.1 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.

6.6.2 Configuration

Accepted fields:

Name	Type	Default	Description
`trackingID`	`string | string[]`	**Required**	The tracking ID of your gtag service. It is possible to provide multiple ids.
`anonymizeIP`	`boolean`	`false`	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,
        },
      },
    ],
  ],
};
```

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.

6.7.1 Installation

npm | Yarn | pnpm | Bun

```
npm install --save @docusaurus/plugin-rsdoctor
```

6.7.2 Configuration

Accepted fields:

Name	Type	Default	Description
`rsdoctorOptions`	`object`	`{}`	The Rsdoctor 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',
        },
      },
    ],
  ],
};
```

6.8 📦 plugin-svgr

An SVGR plugin to transform SVG files into React components automatically at build time.

6.8.1 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.

6.8.2 Configuration

Accepted fields:

Name	Type	Default	Description
`svgrConfig`	`object`	`{}`	The SVGR config options, 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 */
          },
        },
      },
    ],
  ],
};
```

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 gtag 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.

6.9.1 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.

6.9.2 Configuration

Accepted fields:

Name	Type	Default	Description
`containerId`	`string`	**Required**	Your Tag Manager container Id (usually starts with `GTM-`).

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',
        },
      },
    ],
  ],
};
```

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`.

6.10.1 Installation

npm | Yarn | pnpm | Bun

```
npm install --save @docusaurus/plugin-ideal-image
```

6.10.2 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')} />
```

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"]
  }
}
```

6.10.3 Configuration

Accepted fields:

Option	Type	Default	Description
`name`	`string`	`ideal-img/[name].[hash:hex:7].[width].[ext]`	Filename template for output files.
`sizes`	`number[]`	*original size*	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
`disableIn-Dev`	`boolean`	`true`	You can test ideal image behavior in dev mode by setting this to `false`. **Tip**: use network throttling 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,
      },
    ],
  ],
};
```

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.

6.11.1 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.

6.11.2 Configuration

Accepted fields:

Name	Type	Default	Description
`layers`	`Layers`	**Built-in layers**	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

:::warning 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'),
  },
};
```

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.

6.12.1 Installation

npm | Yarn | pnpm | Bun

```
npm install --save @docusaurus/plugin-pwa
```

6.12.2 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)',
          },
        ],
      },
    ],
  ],
};
```

6.12.3 Progressive Web App

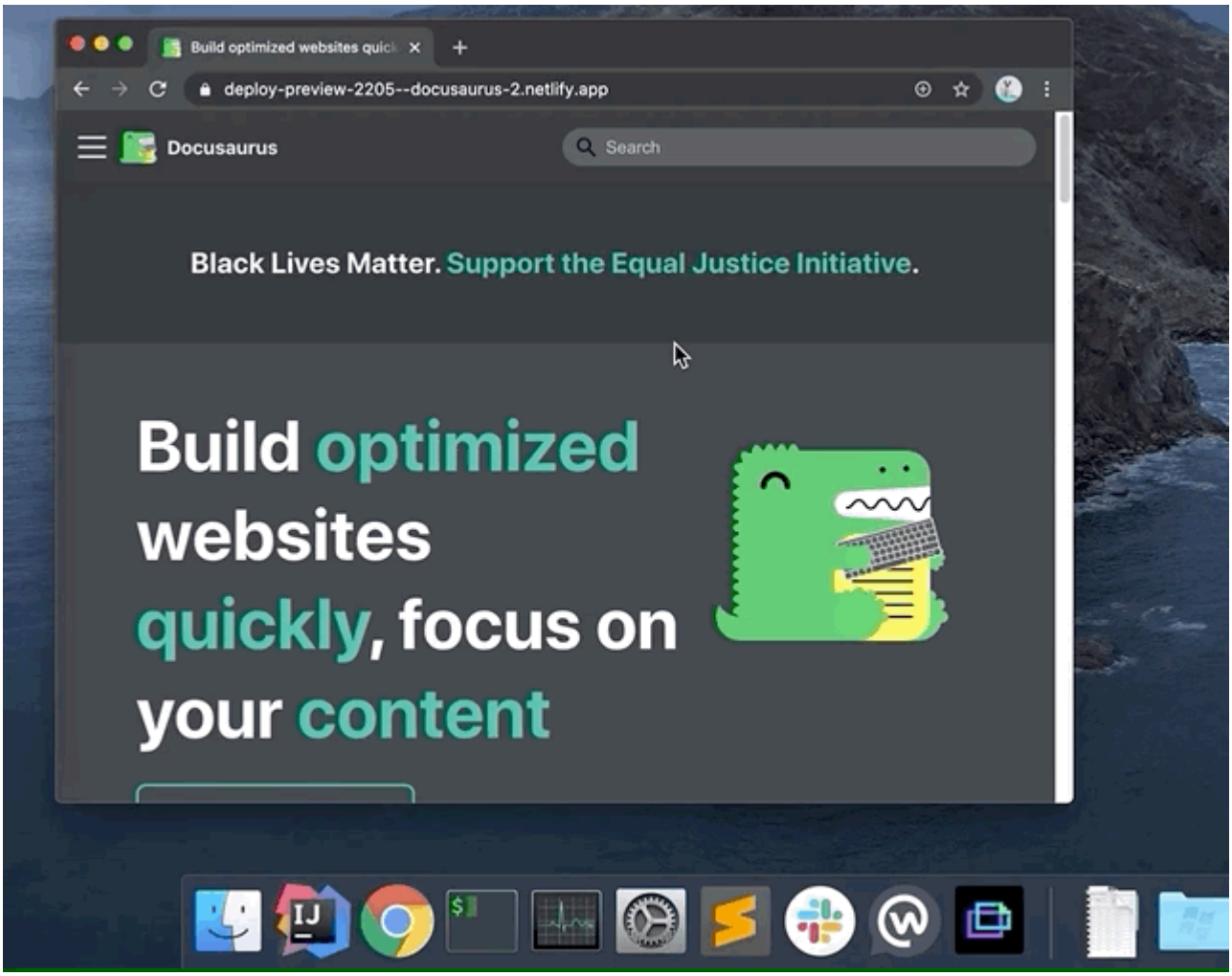
Having a service worker installed is not enough to make your application a PWA. You'll need to at least include a Web App Manifest and have the correct tags in `<head>` (Options > pwaHead).

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

6.12.4 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.

6.12.5 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.

6.12.6 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 WARNING

Use this carefully: some users may not like to be forced to use the offline mode.

:::

:::danger 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.

6.12.7 Manifest example

The Docusaurus site manifest can serve as an inspiration:

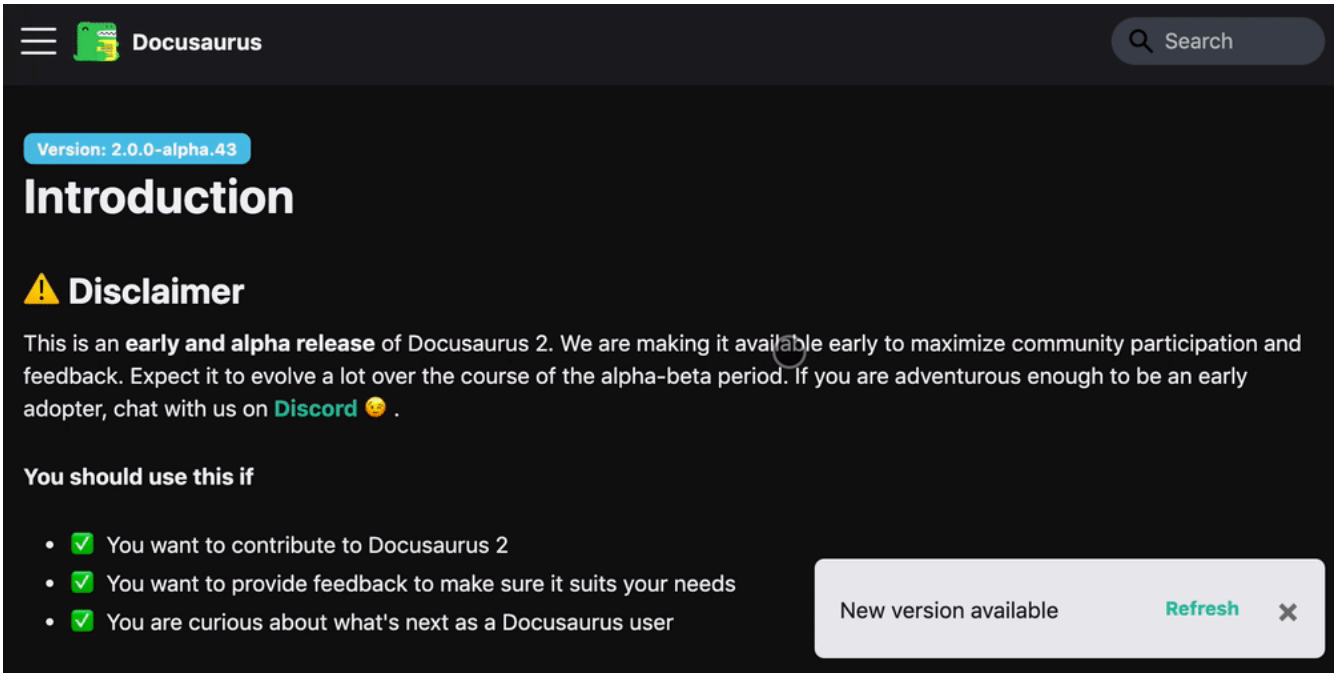
```
{
  "name": "Docusaurus",
  "short_name": "Docusaurus",
  "theme_color": "#2196f3",
  "background_color": "#424242",
  "display": "standalone",
  "scope": "./",
  "start_url": "./index.html",
  "related_applications": [
    {
      "platform": "webapp",
      "url": "https://docusaurus.io/manifest.json"
    }
  ],
  "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"
      }
    ]
  }
```

6.12.8 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.

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.

6.13.1 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.

6.13.2 Configuration

Accepted fields:

Name	Type	Default	Description
`lastmod`	`'date' | 'datetime' | null`	`null`	`date` is YYYY-MM-DD. `datetime` is a ISO 8601 datetime. `null` is disabled. See sitemap docs.
`changefreq`	`string | null`	`'weekly'`	See sitemap docs
`priority`	`number | null`	`0.5`	See sitemap docs
`ignorePatterns`	`string[]`	`[]`	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.
`filename`	`string`	`sitemap.xml`	The path to the created sitemap file, relative to the output directory. Useful if you have two plugin instances outputting two files.
`createSitemapItems`	`CreateSitemapItemsFn | undefined`	`undefined`	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

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`.

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.

6.14.1 Installation

npm | Yarn | pnpm | Bun

```
npm install --save @docusaurus/plugin-vercel-analytics
```

6.14.2 Configuration

Accepted fields:

Name	Type	Default	Description
`mode`	`string`	`'auto'`	Override the automatic environment detection. Read the official docs for details.
`debug`	`boolean`	`undefined`	Enable browser console logging of analytics events. Read the official docs for details.

Example configuration

You can configure this plugin through plugin options.

docusaurus.config.js

```
export default {
  plugins: [
    [
      'vercel-analytics',
      {
        debug: true,
        mode: 'auto',
      },
    ],
  ],
};
```

7 Docusaurus themes

We provide official Docusaurus themes.

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.

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

8 Themes

8.1 Theme configuration

This configuration applies to all main themes.

8.1.1 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
`defaultMode`	`'light' | 'dark'`	`'light'`	The color mode when user first visits the site.
`disableSwitch`	`boolean`	`false`	Hides the switch in the navbar. Useful if you want to support a single color mode.
`respectPrefersColorScheme`	`boolean`	`false`	Whether to use the `prefers-color-scheme` media-query, using user system preferences, instead of the hardcoded `defaultMode`.

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
`image`	`string`	`undefined`	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
`metadata`	`Metadata[]`	`[]`	Any field will be directly passed to the `<meta />` tag. Possible fields include `id`, `name`, `property`, `content`, `itemprop`, 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
`id`	`string`	`'announcement-bar'`	Any value that will identify this message.
`content`	`string`	`''`	The text content of the announcement. HTML will be interpolated.
`backgroundColor`	`string`	`'#fff'`	Background color of the entire bar.
`textColor`	`string`	`'#000'`	Announcement text color.
`isCloseable`	`boolean`	`true`	Whether this announcement can be dismissed with a '×' button.

Example configuration:

docusaurus.config.js

```
export default {
  themeConfig: {
    announcementBar: {
      id: 'support_us',
      content:
        'We are looking to revamp our docs, please fill <a target="_blank" rel="noopener noreferrer" href="#">this survey</a>',
      backgroundColor: '#fafbfc',
      textColor: '#091E42',
      isCloseable: false,
    },
  },
};
```

8.1.2 Plugins

Our main themes offer additional theme configuration options for Docusaurus core content plugins.

Docs

Name	Type	Default	Description
`versionPersistence`	`'localStorage' | 'none'`	`'localStorage'`	Defines the browser persistence of the preferred docs version.
`sidebar.hideable`	`boolean`	`false`	Show a hide button at the bottom of the sidebar.
`sidebar.autoCollapseCategories`	`boolean`	`false`	Automatically collapse all sibling categories of the one you navigate

Name	Type	Default	Description
			to.

Example configuration:

docusaurus.config.js

```
export default {
  themeConfig: {
    docs: {
      versionPersistence: 'localStorage',
      sidebar: {
        hideable: false,
        autoCollapseCategories: false,
      },
    },
  },
};
```

Blog

Name	Type	Default	Description
`sidebar.groupByYear`	`boolean`	`true`	Group sidebar blog posts by years.

Example configuration:

docusaurus.config.js

```
export default {
  themeConfig: {
    blog: {
      sidebar: {
        groupByYear: true,
      },
    },
  },
};
```

8.1.3 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`	**Required**	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 `width` attribute.
`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
`type`	`'default'`	Optional	Sets the type of this item to a link.
`label`	`string`	**Required**	The name to be shown for this item.
`html`	`string`	Optional	Same as `label`, but renders pure HTML instead of text content.
`to`	`string`	**Required**	Client-side routing, used for navigating within the website. The baseUrl will be automatically prepended to this value.
`href`	`string`	**Required**	A full-page navigation, used for navigating outside of the website. **Only one of `to` or `href` should be used.**
`prependBaseUrlToHref`	`boolean`	`false`	Prepends the baseUrl to `href` values.
`position`	`'left' | 'right'`	`'left'`	The side of the navbar this item should appear on.
`activeBasePath`	`string`	`to` / `href`	To apply the active class styling on all routes starting with this path. This usually isn't necessary.
`activeBaseRegex`	`string`	`undefined`	Alternative to `activeBasePath` if required.
`className`	`string`	`''`	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: '<b>Introduction</b>'
          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
`type`	`'dropdown'`	Optional	Sets the type of this item to a dropdown.
`label`	`string`	**Required**	The name to be shown for this item.
`items`	`LinkLikeItem[]`	**Required**	The items to be contained in the dropdown.
`position`	`'left' | 'right'`	`'left'`	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
`type`	`'doc'`	**Required**	Sets the type of this item to a doc link.
`docId`	`string`	**Required**	The ID of the doc that this item links to.
`label`	`string`	`docId`	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 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'`	**Required**	Sets the type of this navbar item to a sidebar's first document.
`sidebarId`	`string`	**Required**	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'`	**Required**	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`	`LinkLikeItem[]`	`[]`	Add additional dropdown items at the beginning of the dropdown.
`dropdownItemsAfter`	`LinkLikeItem[]`	`[]`	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 the versioning guide for details.

Types:

```ts
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'`	**Required**	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'`	**Required**	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`	`LinkLikeItem[]`	`[]`	Add additional dropdown items at the beginning of the dropdown.
`dropdownItemsAfter`	`LinkLikeItem[]`	`[]`	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'`	**Required**	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'`	**Required**	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',
    },
  },
};
```

8.1.4 CodeBlock

Docusaurus uses Prism React Renderer to highlight code blocks. All configuration are in the `prism` object.

Accepted fields:

Name	Type	Default	Description
`theme`	`PrismTheme`	`palenight`	The Prism theme to use for light-theme code blocks.
`darkTheme`	`PrismTheme`	`palenight`	The Prism theme to use for dark-theme code blocks.
`defaultLanguage`	`string`	`undefined`	The default language to use for code blocks not declaring any explicit language.
`magicComments`	`MagicCommentConfig[]`	*see below*	The list of magic comments.

```
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',
      },
    },
  };
```

8.1.5 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 Navbar logo 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 `FooterItem`s for each column.

Name	Type	Default	Description
`title`	`string`	`undefined`	Label of the section of these links.
`items`	`FooterItem[]`	`[]`	Links in this section.

Accepted fields of each `FooterItem`:

Name	Type	Default	Description
`label`	`string`	**Required**	Text to be displayed for this link.
`to`	`string`	**Required**	Client-side routing, used for navigating within the website. The baseUrl will be automatically prepended to this value.
`href`	`string`	**Required**	A full-page navigation, used for navigating outside of the website. **Only one of `to` or `href` should be used.**
`html`	`string`	`undefined`	Renders the HTML pass-through instead of a simple link. In case `html` 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">
                  <img
src="https://www.netlify.com/img/global/badges/netlify-color-accent.svg"
alt="Deploys by Netlify" width="114" height="51" />
                </a>
              `,
          },
        ],
      },
    ],
  },
};
```

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">
              <img src="https://www.netlify.com/img/global/badges/netlify-
color-accent.svg" alt="Deploys by Netlify" width="114" height="51" />
            </a>
          `,
      },
    ],
  },
};
```

8.1.6 Table of Contents

You can adjust the default table of contents via `themeConfig.tableOfContents`.

Name	Type	Default	Description
`minHeadingLevel`	`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.
`maxHeadingLevel`	`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,
    },
  },
};
```

8.1.7 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 <span>{colorMode}</span>;
};
```

NOTE

The component calling `useColorMode` must be a child of the `Layout` component.

```
function ExamplePage() {
  return (
    <Layout>
      <Example />
    </Layout>
  );
}
```

8.1.8 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
```

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.

8.2.1 Configuration

Accepted fields:

Option	Type	Default	Description
`customCss`	`string[] | string`	`[]`	Stylesheets to be imported globally as client modules. 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',
        },
      },
    ],
  ],
};
```

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',
    },
  },
};
```

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.

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
```

8.5.1 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,
  },
};
```

9 Miscellaneous

9.1 📦 create-docusaurus

A scaffolding utility to help you instantly set up a functional Docusaurus app.

9.1.1 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.

9.1.2 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.

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.

9.2.1 Installation

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

```
npm install --save-dev @docusaurus/eslint-plugin
```

9.2.2 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.

9.2.3 Supported rules

Name	Description	
`@docusaurus/no-untranslated-text`	Enforce text labels in JSX to be wrapped by translate calls	
`@docusaurus/string-literal-i18n-messages`	Enforce translate APIs to be called on plain text labels	✅
`@docusaurus/no-html-links`	Ensures @docusaurus/Link is used instead of `<a>` tags	✅
`@docusaurus/prefer-docusaurus-heading`	Ensures @theme/Heading is used instead of `<hn>` tags for headings	✅

✅ = recommended

9.2.4 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',
  },
});
```

9.2.5 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"
  }
}
```

9.2.6 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:

```
<a href="/page">go to page!</a>

<a href="https://x.com/docusaurus" target="_blank">X</a>
```

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
`ignoreFullyResolved`	`boolean`	`false`	Set to true will not report any `<a>` tags with absolute URLs including a protocol.

9.2.7 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
`ignoredStrings`	`string[]`	`[]`	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

9.2.8 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>
```

9.2.9 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

9.3 📦 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.

9.3.1 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: