Pages

> This document is for Next.js versions 9.3 and up. If you're using older versions of Next.js, refer to our previous documentation.

In Next.js, a **page** is a React Component exported from a `.js`, `.jsx`, `.ts`, or `.tsx` file in the `pages` directory. Each page is associated with a route based on its file name.

Example: If you create `pages/about.js` that exports a React component like below, it will be accessible at `/about`.

```
function About() {
  return <div>About</div>
}

export default About
```

Pages with Dynamic Routes

Next.js supports pages with dynamic routes. For example, if you create a file called `pages/posts/[id].js`, then it will be accessible at `posts/1`, `posts/2`, etc.

> To learn more about dynamic routing, check the Dynamic Routing documentation.

Pre-rendering

By default, Next.js **pre-renders** every page. This means that Next.js generates HTML for each page in advance, instead of having it all done by client-side JavaScript. Pre-rendering can result in better performance and SEO.

Each generated HTML is associated with minimal JavaScript code necessary for that page. When a page is loaded by the browser, its JavaScript code runs and makes the page fully interactive. (This process is called *hydration*.)

Two forms of Pre-rendering

Next.js has two forms of pre-rendering: **Static Generation** and **Server-side Rendering**. The difference is in **when** it generates the HTML for a page.

- **Static Generation (Recommended)**: The HTML is generated at **build time** and will be reused on each request.
- **Server-side Rendering**: The HTML is generated on **each request**.

Importantly, Next.js lets you **choose** which pre-rendering form you'd like to use for each page. You can create a "hybrid" Next.js app by using Static Generation for most pages and using Server-side Rendering for others.

We **recommend** using **Static Generation** over Server-side Rendering for performance reasons. Statically generated pages can be cached by CDN with no extra configuration to boost performance. However, in some cases, Server-side Rendering might be the only option.

You can also use **Client-side Rendering** along with Static Generation or Server-side Rendering. That means some parts of a page can be rendered entirely by client side JavaScript. To learn more, take a look at the Data Fetching documentation.

Static Generation (Recommended)

Examples

WordPress Example (Demo)

Blog Starter using markdown files (Demo)

DatoCMS Example (Demo)

TakeShape Example (Demo)

Sanity Example (Demo)

Prismic Example (Demo)

Contentful Example (Demo)

Strapi Example (Demo)

Prepr Example (Demo)

Agility CMS Example (Demo)

Cosmic Example (Demo)

ButterCMS Example (Demo)

Storyblok Example (Demo)

GraphCMS Example (Demo)

Kontent Example (Demo)

Builder.io Example (Demo)

Static Tweet (Demo)

If a page uses **Static Generation**, the page HTML is generated at **build time**. That means in production, the page HTML is generated when you run `next build` . This HTML will then be reused on each request. It can be cached by a CDN.

In Next.js, you can statically generate pages **with or without data**. Let's take a look at each case.

Static Generation without data

By default, Next.js pre-renders pages using Static Generation without fetching data. Here's an example:

```
function About() {
  return <div>About</div>
}

export default About
```

Note that this page does not need to fetch any external data to be pre-rendered. In cases like this, Next.js generates a single HTML file per page during build time.

Static Generation with data

Some pages require fetching external data for pre-rendering. There are two scenarios, and one or both might apply. In each case, you can use these functions that Next.js provides:

1. Your page **content** depends on external data: Use `getStaticProps`.
2. Your page **paths** depend on external data: Use `getStaticPaths` (usually in addition to `getStaticProps`).

Scenario 1: Your page content depends on external data **Example**: Your blog page might need to fetch the list of blog posts from a CMS (content management system).

```
// TODO: Need to fetch `posts` (by calling some API endpoint)
//       before this page can be pre-rendered.
function Blog({ posts }) {
  return (
    <ul>
      {posts.map((post) => (
        <li>{post.title}</li>
      ))}
    </ul>
  )
}

export default Blog
```

To fetch this data on pre-render, Next.js allows you to `export` an `async` function called `getStaticProps` from the same file. This function gets called at build time and lets you pass fetched data to the page's `props` on pre-render.

```
function Blog({ posts }) {
  // Render posts...
```

```
}

// This function gets called at build time
export async function getStaticProps() {
  // Call an external API endpoint to get posts
  const res = await fetch('https://.../posts')
  const posts = await res.json()

  // By returning { props: { posts } }, the Blog component
  // will receive `posts` as a prop at build time
  return {
    props: {
      posts,
    },
  }
}

export default Blog
```

To learn more about how `getStaticProps` works, check out the Data Fetching documentation.

Scenario 2: Your page paths depend on external data Next.js allows you to create pages with **dynamic routes**. For example, you can create a file called `pages/posts/[id].js` to show a single blog post based on `id`. This will allow you to show a blog post with `id: 1` when you access `posts/1`.

> To learn more about dynamic routing, check the Dynamic Routing documentation.

However, which `id` you want to pre-render at build time might depend on external data.

Example: suppose that you've only added one blog post (with `id: 1`) to the database. In this case, you'd only want to pre-render `posts/1` at build time.

Later, you might add the second post with `id: 2`. Then you'd want to pre-render `posts/2` as well.

So your page **paths** that are pre-rendered depend on external data**.** To handle this, Next.js lets you `export` an `async` function called `getStaticPaths` from a dynamic page (`pages/posts/[id].js` in this case). This function gets called at build time and lets you specify which paths you want to pre-render.

```
// This function gets called at build time
export async function getStaticPaths() {
  // Call an external API endpoint to get posts
  const res = await fetch('https://.../posts')
  const posts = await res.json()
```

```
  // Get the paths we want to pre-render based on posts
  const paths = posts.map((post) => ({
    params: { id: post.id },
  }))

  // We'll pre-render only these paths at build time.
  // { fallback: false } means other routes should 404.
  return { paths, fallback: false }
}
```

Also in `pages/posts/[id].js`, you need to export `getStaticProps` so that you can fetch the data about the post with this `id` and use it to pre-render the page:

```
function Post({ post }) {
  // Render post...
}

export async function getStaticPaths() {
  // ...
}

// This also gets called at build time
export async function getStaticProps({ params }) {
  // params contains the post `id`.
  // If the route is like /posts/1, then params.id is 1
  const res = await fetch(`https://.../posts/${params.id}`)
  const post = await res.json()

  // Pass post data to the page via props
  return { props: { post } }
}

export default Post
```

To learn more about how `getStaticPaths` works, check out the Data Fetching documentation.

When should I use Static Generation?

We recommend using **Static Generation** (with and without data) whenever possible because your page can be built once and served by CDN, which makes it much faster than having a server render the page on every request.

You can use Static Generation for many types of pages, including:

- Marketing pages
- Blog posts and portfolios

- E-commerce product listings
- Help and documentation

You should ask yourself: "Can I pre-render this page **ahead** of a user's request?" If the answer is yes, then you should choose Static Generation.

On the other hand, Static Generation is **not** a good idea if you cannot pre-render a page ahead of a user's request. Maybe your page shows frequently updated data, and the page content changes on every request.

In cases like this, you can do one of the following:

- Use Static Generation with **Client-side Rendering:** You can skip pre-rendering some parts of a page and then use client-side JavaScript to populate them. To learn more about this approach, check out the Data Fetching documentation.
- Use **Server-Side Rendering:** Next.js pre-renders a page on each request. It will be slower because the page cannot be cached by a CDN, but the pre-rendered page will always be up-to-date. We'll talk about this approach below.

Server-side Rendering

> Also referred to as "SSR" or "Dynamic Rendering".

If a page uses **Server-side Rendering**, the page HTML is generated on **each request**.

To use Server-side Rendering for a page, you need to `export` an `async` function called `getServerSideProps`. This function will be called by the server on every request.

For example, suppose that your page needs to pre-render frequently updated data (fetched from an external API). You can write `getServerSideProps` which fetches this data and passes it to `Page` like below:

```
function Page({ data }) {
  // Render data...
}

// This gets called on every request
export async function getServerSideProps() {
  // Fetch data from external API
  const res = await fetch(`https://.../data`)
  const data = await res.json()

  // Pass data to the page via props
  return { props: { data } }
}
```

```
export default Page
```

As you can see, `getServerSideProps` is similar to `getStaticProps`, but the difference is that `getServerSideProps` is run on every request instead of on build time.

To learn more about how `getServerSideProps` works, check out our Data Fetching documentation

Summary

We've discussed two forms of pre-rendering for Next.js.

- **Static Generation (Recommended):** The HTML is generated at **build time** and will be reused on each request. To make a page use Static Generation, either export the page component, or export `getStaticProps` (and `getStaticPaths` if necessary). It's great for pages that can be pre-rendered ahead of a user's request. You can also use it with Client-side Rendering to bring in additional data.
- **Server-side Rendering:** The HTML is generated on **each request**. To make a page use Server-side Rendering, export `getServerSideProps`. Because Server-side Rendering results in slower performance than Static Generation, use this only if absolutely necessary.

Learn more

We recommend you to read the following sections next:

Data Fetching: Learn more about data fetching in Next.js.

Preview Mode: Learn more about the preview mode in Next.js.

Routing: Learn more about routing in Next.js.

TypeScript: Add TypeScript to your pages.