**BASICS**

* [**Create a Next.js App**](https://nextjs.org/learn/basics/create-nextjs-app)
  + [**Setup**](https://nextjs.org/learn/basics/create-nextjs-app/setup)
  + [Welcome to Next.js](https://nextjs.org/learn/basics/create-nextjs-app/welcome-to-nextjs)
  + [Editing the Page](https://nextjs.org/learn/basics/create-nextjs-app/editing-the-page)
* [Navigate Between Pages](https://nextjs.org/learn/basics/navigate-between-pages)
* [Assets, Metadata, and CSS](https://nextjs.org/learn/basics/assets-metadata-css)
* [Pre-rendering and Data Fetching](https://nextjs.org/learn/basics/data-fetching)
* [Dynamic Routes](https://nextjs.org/learn/basics/dynamic-routes)
* [API Routes](https://nextjs.org/learn/basics/api-routes)
* [Deploying Your Next.js App](https://nextjs.org/learn/basics/deploying-nextjs-app)

## Create a Next.js App

[1](https://nextjs.org/learn/basics/create-nextjs-app)

[**2**](https://nextjs.org/learn/basics/create-nextjs-app/setup)

[3](https://nextjs.org/learn/basics/create-nextjs-app/welcome-to-nextjs)

[4](https://nextjs.org/learn/basics/create-nextjs-app/editing-the-page)

## Setup

First, let’s make sure that your development environment is ready.

* If you don’t have **Node.js** installed, [install it from here](https://nodejs.org/en/" \t "_blank). You’ll need Node.js version **10.13** or later.
* You’ll be using your own text editor and terminal app for this tutorial.

If you are on Windows, we recommend [downloading Git for Windows](https://gitforwindows.org/" \t "_blank) and use Git Bash that comes with it, which supports the UNIX-specific commands in this tutorial. [Windows Subsystem for Linux (WSL)](https://docs.microsoft.com/en-us/windows/wsl/install-win10) is another option.

### Create a Next.js app

To create a Next.js app, open your terminal, cd into the directory you’d like to create the app in, and run the following command:

npx create-next-app nextjs-blog --use-npm --example "https://github.com/vercel/next-learn/tree/master/basics/learn-starter"

Under the hood, this uses the tool called [create-next-app](https://nextjs.org/docs/api-reference/create-next-app" \t "_blank), which bootstraps a Next.js app for you. It uses [this template](https://github.com/vercel/next-learn/tree/master/basics/learn-starter" \t "_blank) through the --example flag.

If it doesn’t work, please take a look at [this page](https://github.com/vercel/next-learn/blob/master/basics/errors/install.md" \t "_blank).

### Run the development server

You now have a new directory called nextjs-blog. Let’s cd into it:

cd nextjs-blog

Then, run the following command:

npm run dev

This starts your Next.js app’s "development server" (more on this later) on port **3000**.

Let’s check to see if it’s working. Open [http://localhost:3000](http://localhost:3000/) from your browser.

**Quick Review**: What text do you see on the page?

Welcome to Next.js!

Hello Next.js!

Submit

## Create a Next.js App

[1](https://nextjs.org/learn/basics/create-nextjs-app)

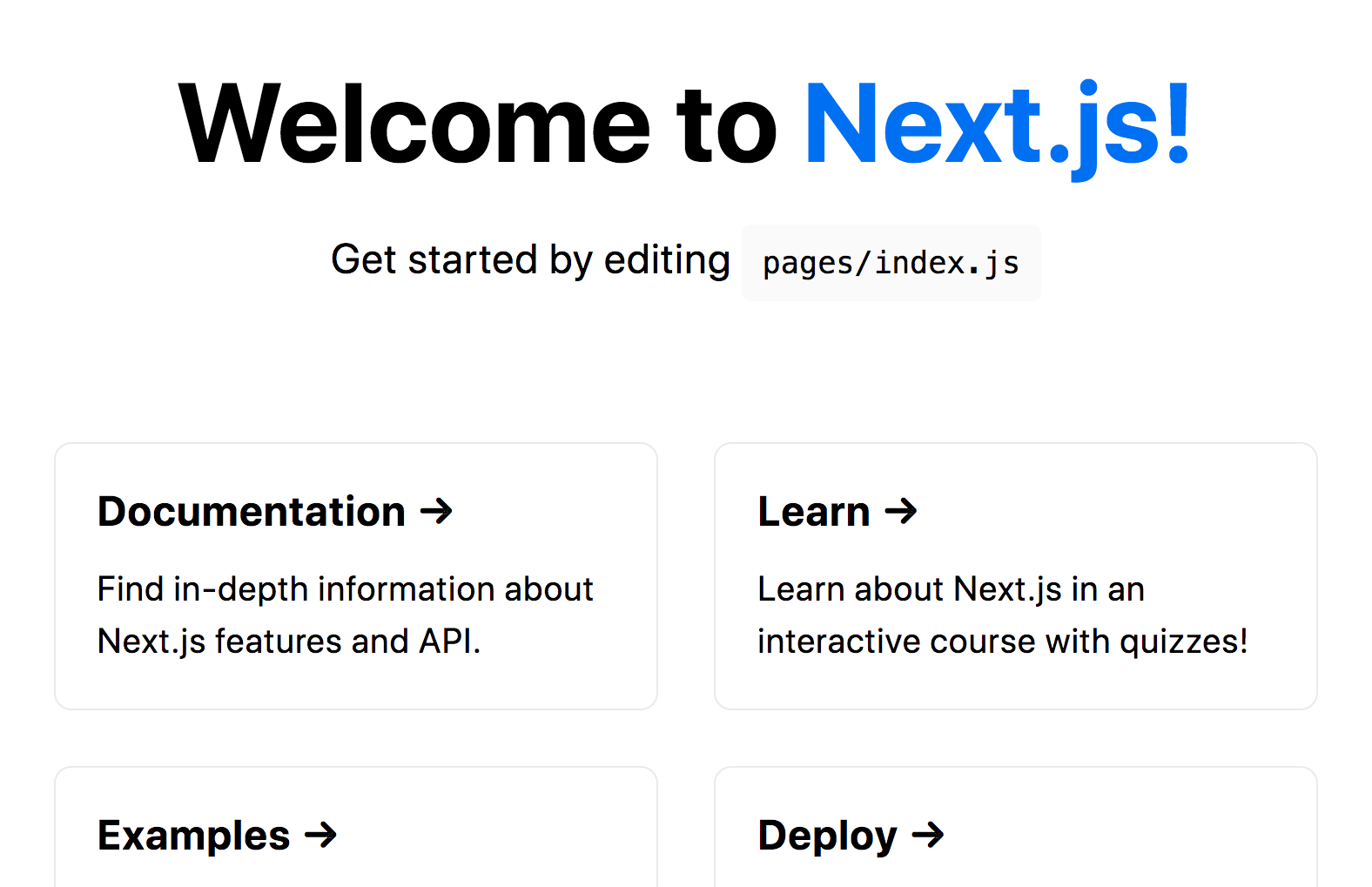
[2](https://nextjs.org/learn/basics/create-nextjs-app/setup)

[**3**](https://nextjs.org/learn/basics/create-nextjs-app/welcome-to-nextjs)

[4](https://nextjs.org/learn/basics/create-nextjs-app/editing-the-page)

## Welcome to Next.js

You should see a page like this when you access [http://localhost:3000](http://localhost:3000/). This is the starter template page which shows some helpful information about Next.js.



**Help is available:** If you get stuck, you can reach out to the community on [GitHub Discussions](https://github.com/vercel/next.js/discussions).

Let’s try to edit this page next!

## Create a Next.js App

[1](https://nextjs.org/learn/basics/create-nextjs-app)

[2](https://nextjs.org/learn/basics/create-nextjs-app/setup)

[3](https://nextjs.org/learn/basics/create-nextjs-app/welcome-to-nextjs)

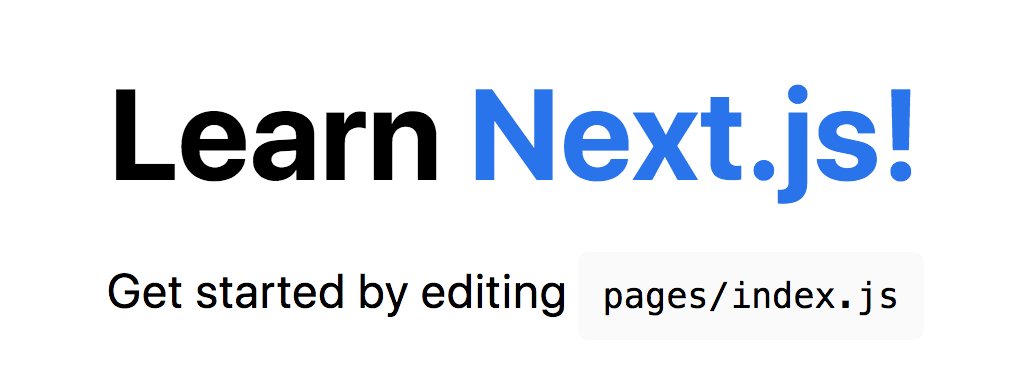
[**4**](https://nextjs.org/learn/basics/create-nextjs-app/editing-the-page)

## Editing the Page

Let’s try editing the starter page.

* Make sure the Next.js development server is still running.
* Open pages/index.js with your text editor.
* Find the text that says **“Welcome to”** under the <h1> tag and change it to **“Learn”**.
* Save the file.

As soon as you save the file, the browser automatically updates the page with the new text:



The Next.js development server has [Fast Refresh](https://nextjs.org/docs/basic-features/fast-refresh) enabled. When you make changes to files, Next.js automatically applies the changes in the browser almost instantly. No refresh needed! This will help you iterate on your app quickly.

### Next Up: Creating Pages

Great job! That’s it for the first lesson.

In the next lesson, we’ll talk about creating more pages and navigating between pages.

You should keep the development server running, but if you want to restart it, hit Ctrl + c to stop the server.

* [Create a Next.js App](https://nextjs.org/learn/basics/create-nextjs-app)
* [**Navigate Between Pages**](https://nextjs.org/learn/basics/navigate-between-pages)
  + [Setup](https://nextjs.org/learn/basics/navigate-between-pages/setup)
  + [Pages in Next.js](https://nextjs.org/learn/basics/navigate-between-pages/pages-in-nextjs)
  + [Link Component](https://nextjs.org/learn/basics/navigate-between-pages/link-component)
  + [Client-Side Navigation](https://nextjs.org/learn/basics/navigate-between-pages/client-side)
* [Assets, Metadata, and CSS](https://nextjs.org/learn/basics/assets-metadata-css)
* [Pre-rendering and Data Fetching](https://nextjs.org/learn/basics/data-fetching)
* [Dynamic Routes](https://nextjs.org/learn/basics/dynamic-routes)
* [API Routes](https://nextjs.org/learn/basics/api-routes)
* [Deploying Your Next.js App](https://nextjs.org/learn/basics/deploying-nextjs-app)

## Navigate Between Pages

[**1**](https://nextjs.org/learn/basics/navigate-between-pages)

[2](https://nextjs.org/learn/basics/navigate-between-pages/setup)

[3](https://nextjs.org/learn/basics/navigate-between-pages/pages-in-nextjs)

[4](https://nextjs.org/learn/basics/navigate-between-pages/link-component)

[5](https://nextjs.org/learn/basics/navigate-between-pages/client-side)

So far, the Next.js app we created only has one page. Websites and web applications generally have many different pages.

Let's explore how to add more pages to our application.

## What You’ll Learn in This Lesson

In this lesson, you will:

* Create a new page using the integrated [file system routing](https://nextjs.org/docs/routing/introduction" \t "_blank).
* Learn how to use the [Link](https://nextjs.org/docs/api-reference/next/link) component to enable client-side navigation between pages.
* Learn about built-in support for code splitting and prefetching.

If you’re looking for detailed documentation on Next.js routing, take a look at the [routing documentation](https://nextjs.org/docs/routing/introduction" \t "_blank).

[Start Now →](https://nextjs.org/learn/basics/navigate-between-pages/setup)

## Navigate Between Pages

[1](https://nextjs.org/learn/basics/navigate-between-pages)

[**2**](https://nextjs.org/learn/basics/navigate-between-pages/setup)

[3](https://nextjs.org/learn/basics/navigate-between-pages/pages-in-nextjs)

[4](https://nextjs.org/learn/basics/navigate-between-pages/link-component)

[5](https://nextjs.org/learn/basics/navigate-between-pages/client-side)

**If you’re continuing from the previous lesson,** you can skip this page. Click the button below to go to the next page.

[Next](https://nextjs.org/learn/basics/navigate-between-pages/pages-in-nextjs)

### Download Starter Code (Optional)

If you’re NOT continuing from the previous lesson, you can download, install, and run the starter code for this lesson below. This sets up a nextjs-blog directory such that it’s identical to the result of the previous lesson.

Again, this is NOT necessary if you’ve just finished the previous lesson.

npx create-next-app nextjs-blog --use-npm --example "https://github.com/vercel/next-learn/tree/master/basics/navigate-between-pages-starter"

Then follow the instructions from the command output (cd into the directory and start the development server).

## Navigate Between Pages

[1](https://nextjs.org/learn/basics/navigate-between-pages)

[2](https://nextjs.org/learn/basics/navigate-between-pages/setup)

[**3**](https://nextjs.org/learn/basics/navigate-between-pages/pages-in-nextjs)

[4](https://nextjs.org/learn/basics/navigate-between-pages/link-component)

[5](https://nextjs.org/learn/basics/navigate-between-pages/client-side)

## Pages in Next.js

In Next.js, a page is a React Component exported from a file in the [pages directory](https://nextjs.org/docs/basic-features/pages" \t "_blank).

Pages are associated with a route based on their **file name**. For example, in development:

* pages/index.js is associated with the / route.
* pages/posts/first-post.js is associated with the /posts/first-post route.

We already have the pages/index.js file, so let’s create pages/posts/first-post.js to see how it works.

### Create a New Page

Create the posts directory under pages.

Create a file called first-post.js inside the posts directory with the following content:

export default function FirstPost() {

return <h1>First Post</h1>

}

The component can have any name, but you must export it as a default export.

Now, make sure that the development server is running and visit <http://localhost:3000/posts/first-post>. You should see the page:



This is how you can create different pages in Next.js.

Simply create a JS file under the [pages directory](https://nextjs.org/docs/basic-features/pages" \t "_blank), and the path to the file becomes the URL path.

In a way, this is similar to building websites using HTML or PHP files. Instead of writing HTML you write JSX and use React Components.

Let's add a link to the newly added page so that we can navigate to it from the homepage.

**Quick Review**: If you wanted to add a new route /authors/me, what would the file name be (including the directory)?

authors/me.js

pages/authors/me.js

routes/authors/me.js

Submit

## Navigate Between Pages

[1](https://nextjs.org/learn/basics/navigate-between-pages)

[2](https://nextjs.org/learn/basics/navigate-between-pages/setup)

[3](https://nextjs.org/learn/basics/navigate-between-pages/pages-in-nextjs)

[**4**](https://nextjs.org/learn/basics/navigate-between-pages/link-component)

[5](https://nextjs.org/learn/basics/navigate-between-pages/client-side)

## Link Component

When linking between pages on websites, you use the <a> HTML tag.

In Next.js, you use the Link Component from [next/link](https://nextjs.org/docs/api-reference/next/link" \t "_blank) to wrap the <a> tag. <Link> allows you to do client-side navigation to a different page in the application.

### Using <Link>

First, open pages/index.js, and import the Link component from [next/link](https://nextjs.org/docs/api-reference/next/link" \t "_blank) by adding this line at the top:

import Link from 'next/link'

Then find the h1 tag that looks like this:

<h1 className="title">

Learn <a href="https://nextjs.org">Next.js!</a>

</h1>

And change it to:

<h1 className="title">

Read{' '}

<Link href="/posts/first-post">

<a>this page!</a>

</Link>

</h1>

{' '} adds an empty space, which is used to divide text over multiple lines.

Next, open pages/posts/first-post.js and replace its content with the following:

import Link from 'next/link'

export default function FirstPost() {

return (

<>

<h1>First Post</h1>

<h2>

<Link href="/">

<a>Back to home</a>

</Link>

</h2>

</>

)

}

As you can see, the Link component is similar to using <a> tags, but instead of <a href="…">, you use <Link href="…"> and put an <a> tag inside.

Let’s check to see if it works. You should now have a link on each page, allowing you to go back and forth:



## Navigate Between Pages

[1](https://nextjs.org/learn/basics/navigate-between-pages)

[2](https://nextjs.org/learn/basics/navigate-between-pages/setup)

[3](https://nextjs.org/learn/basics/navigate-between-pages/pages-in-nextjs)

[4](https://nextjs.org/learn/basics/navigate-between-pages/link-component)

[**5**](https://nextjs.org/learn/basics/navigate-between-pages/client-side)

## Client-Side Navigation

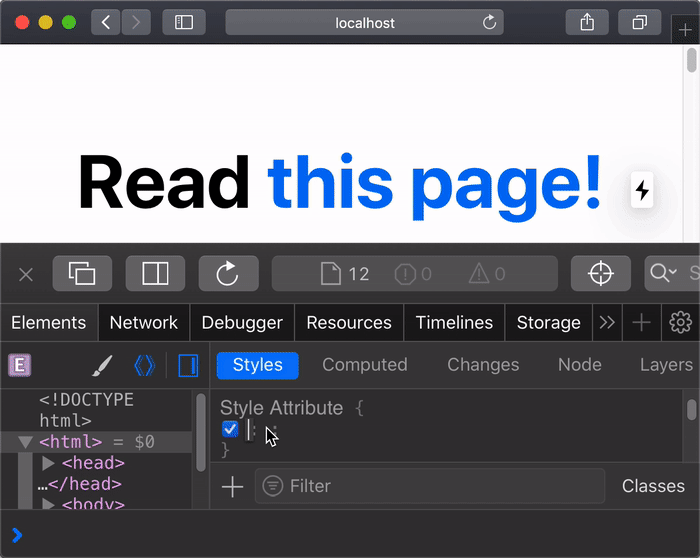
The [Link](https://nextjs.org/docs/api-reference/next/link) component enables **client-side navigation** between two pages in the same Next.js app.

Client-side navigation means that the page transition happens using JavaScript, which is faster than the default navigation done by the browser.

Here’s a simple way you can verify it:

* Use the browser’s developer tools to change the background CSS property of <html> to yellow.
* Click on the links to go back and forth between the two pages.
* You’ll see that the yellow background persists between page transitions.

This shows that the browser does not load the full page and client-side navigation is working.



If you’ve used <a href="…"> instead of <Link href="…"> and did this, the background color will be cleared on link clicks because the browser does a full refresh.

### Code splitting and prefetching

Next.js does code splitting automatically, so each page only loads what’s necessary for that page. That means when the homepage is rendered, the code for other pages is not served initially.

This ensures that the homepage loads quickly even if you have hundreds of pages.

Only loading the code for the page you request also means that pages become isolated. If a certain page throws an error, the rest of the application would still work.

Furthermore, in a production build of Next.js, whenever [Link](https://nextjs.org/docs/api-reference/next/link) components appear in the browser’s viewport, Next.js automatically **prefetches** the code for the linked page in the background. By the time you click the link, the code for the destination page will already be loaded in the background, and the page transition will be near-instant!

### Summary

Next.js automatically optimizes your application for the best performance by code splitting, client-side navigation, and prefetching (in production).

You create routes as files under [pages](https://nextjs.org/docs/basic-features/pages" \t "_blank) and use the built-in [Link](https://nextjs.org/docs/api-reference/next/link) component. No routing libraries are required.

You can learn more about the Link component [in the API reference for next/link](https://nextjs.org/docs/api-reference/next/link) and routing in general [in the routing documentation](https://nextjs.org/docs/routing/introduction).

<https://nextjs.org/docs/api-reference/next/link>

# next/link

**Examples**

* [Hello World](https://github.com/vercel/next.js/tree/canary/examples/hello-world)
* [Active className on Link](https://github.com/vercel/next.js/tree/canary/examples/active-class-name)

Before moving forward, we recommend you to read [Routing Introduction](https://nextjs.org/docs/routing/introduction) first.

Client-side transitions between routes can be enabled via the Link component exported by next/link.

For an example, consider a pages directory with the following files:

* pages/index.js
* pages/about.js
* pages/blog/[slug].js

We can have a link to each of these pages like so:

import Link from 'next/link'

function Home() {

return (

<ul>

<li>

<Link href="/">

<a>Home</a>

</Link>

</li>

<li>

<Link href="/about">

<a>About Us</a>

</Link>

</li>

<li>

<Link href="/blog/hello-world">

<a>Blog Post</a>

</Link>

</li>

</ul>

)

}

export default Home

Link accepts the following props:

* href - The path or URL to navigate to. This is the only required prop
* as - Optional decorator for the path that will be shown in the browser URL bar. Before Next.js 9.5.3 this was used for dynamic routes, check our [previous docs](https://nextjs.org/docs/tag/v9.5.2/api-reference/next/link" \l "dynamic-routes) to see how it worked. Note: when this path differs from the one provided in href the previous href/as behavior is used as shown in the [previous docs](https://nextjs.org/docs/tag/v9.5.2/api-reference/next/link" \l "dynamic-routes).
* [passHref](https://nextjs.org/docs/api-reference/next/link#if-the-child-is-a-custom-component-that-wraps-an-a-tag) - Forces Link to send the href property to its child. Defaults to false
* prefetch - Prefetch the page in the background. Defaults to true. Any <Link /> that is in the viewport (initially or through scroll) will be preloaded. Prefetch can be disabled by passing prefetch={false}. When prefetch is set to false, prefetching will still occur on hover. Pages using [Static Generation](https://nextjs.org/docs/basic-features/data-fetching#getstaticprops-static-generation) will preload JSON files with the data for faster page transitions. Prefetching is only enabled in production.
* [replace](https://nextjs.org/docs/api-reference/next/link#replace-the-url-instead-of-push) - Replace the current history state instead of adding a new url into the stack. Defaults to false
* [scroll](https://nextjs.org/docs/api-reference/next/link#disable-scrolling-to-the-top-of-the-page) - Scroll to the top of the page after a navigation. Defaults to true
* [shallow](https://nextjs.org/docs/routing/shallow-routing) - Update the path of the current page without rerunning [getStaticProps](https://nextjs.org/docs/basic-features/data-fetching" \l "getstaticprops-static-generation), [getServerSideProps](https://nextjs.org/docs/basic-features/data-fetching" \l "getserversideprops-server-side-rendering) or [getInitialProps](https://nextjs.org/docs/api-reference/data-fetching/getInitialProps). Defaults to false
* locale - The active locale is automatically prepended. locale allows for providing a different locale. When false href has to include the locale as the default behavior is disabled.

## [If the route has dynamic segments](https://nextjs.org/docs/api-reference/next/link#if-the-route-has-dynamic-segments)

There is nothing to do when linking to a [dynamic route](https://nextjs.org/docs/routing/dynamic-routes), including [catch all routes](https://nextjs.org/docs/routing/dynamic-routes#catch-all-routes), since Next.js 9.5.3 (for older versions check our [previous docs](https://nextjs.org/docs/tag/v9.5.2/api-reference/next/link" \l "dynamic-routes)). However, it can become quite common and handy to use [interpolation](https://nextjs.org/docs/routing/introduction" \l "linking-to-dynamic-paths) or an [URL Object](https://nextjs.org/docs/api-reference/next/link#with-url-object) to generate the link.

For example, the dynamic route pages/blog/[slug].js will match the following link:

import Link from 'next/link'

function Posts({ posts }) {

return (

<ul>

{posts.map((post) => (

<li key={post.id}>

<Link href={`/blog/${encodeURIComponent(post.slug)}`}>

<a>{post.title}</a>

</Link>

</li>

))}

</ul>

)

}

export default Posts

## [If the child is a custom component that wraps an <a> tag](https://nextjs.org/docs/api-reference/next/link#if-the-child-is-a-custom-component-that-wraps-an-a-tag)

If the child of Link is a custom component that wraps an <a> tag, you must add passHref to Link. This is necessary if you’re using libraries like [styled-components](https://styled-components.com/" \t "_blank). Without this, the <a> tag will not have the href attribute, which hurts your site's accessibility and might affect SEO. If you're using [ESLint](https://nextjs.org/docs/basic-features/eslint" \l "eslint-plugin), there is a built-in rule next/link-passhref to ensure correct usage of passHref.

import Link from 'next/link'

import styled from 'styled-components'

// This creates a custom component that wraps an <a> tag

const RedLink = styled.a`

color: red;

`

function NavLink({ href, name }) {

// Must add passHref to Link

return (

<Link href={href} passHref>

<RedLink>{name}</RedLink>

</Link>

)

}

export default NavLink

* If you’re using [emotion](https://emotion.sh/" \t "_blank)’s JSX pragma feature (@jsx jsx), you must use passHref even if you use an <a> tag directly.
* The component should support onClick property to trigger navigation correctly

## [If the child is a functional component](https://nextjs.org/docs/api-reference/next/link#if-the-child-is-a-functional-component)

If the child of Link is a functional component, in addition to using passHref, you must wrap the component in [React.forwardRef](https://reactjs.org/docs/react-api.html" \l "reactforwardref" \t "_blank):

import Link from 'next/link'

// `onClick`, `href`, and `ref` need to be passed to the DOM element

// for proper handling

const MyButton = React.forwardRef(({ onClick, href }, ref) => {

return (

<a href={href} onClick={onClick} ref={ref}>

Click Me

</a>

)

})

function Home() {

return (

<Link href="/about" passHref>

<MyButton />

</Link>

)

}

export default Home

## [With URL Object](https://nextjs.org/docs/api-reference/next/link#with-url-object)

Link can also receive a URL object and it will automatically format it to create the URL string. Here's how to do it:

import Link from 'next/link'

function Home() {

return (

<ul>

<li>

<Link

href={{

pathname: '/about',

query: { name: 'test' },

}}

>

<a>About us</a>

</Link>

</li>

<li>

<Link

href={{

pathname: '/blog/[slug]',

query: { slug: 'my-post' },

}}

>

<a>Blog Post</a>

</Link>

</li>

</ul>

)

}

export default Home

The above example has a link to:

* A predefined route: /about?name=test
* A [dynamic route](https://nextjs.org/docs/routing/dynamic-routes): /blog/my-post

You can use every property as defined in the [Node.js URL module documentation](https://nodejs.org/api/url.html#url_url_strings_and_url_objects).

## [Replace the URL instead of push](https://nextjs.org/docs/api-reference/next/link#replace-the-url-instead-of-push)

The default behavior of the Link component is to push a new URL into the history stack. You can use the replace prop to prevent adding a new entry, as in the following example:

<Link href="/about" replace>

<a>About us</a>

</Link>

## [Disable scrolling to the top of the page](https://nextjs.org/docs/api-reference/next/link#disable-scrolling-to-the-top-of-the-page)

The default behavior of Link is to scroll to the top of the page. When there is a hash defined it will scroll to the specific id, like a normal <a> tag. To prevent scrolling to the top / hash scroll={false} can be added to Link:

<Link href="/?counter=10" scroll={false}>

<a>Disables scrolling to the top</a>

</Link>

[next/router](https://nextjs.org/docs/api-reference/next/router)[next/image](https://nextjs.org/docs/api-reference/next/image)

<https://nextjs.org/docs/routing/introduction>

# Routing

Next.js has a file-system based router built on the [concept of pages](https://nextjs.org/docs/basic-features/pages).

When a file is added to the pages directory, it's automatically available as a route.

The files inside the pages directory can be used to define most common patterns.

#### [Index routes](https://nextjs.org/docs/routing/introduction#index-routes)

The router will automatically route files named index to the root of the directory.

* pages/index.js → /
* pages/blog/index.js → /blog

#### [Nested routes](https://nextjs.org/docs/routing/introduction#nested-routes)

The router supports nested files. If you create a nested folder structure, files will automatically be routed in the same way still.

* pages/blog/first-post.js → /blog/first-post
* pages/dashboard/settings/username.js → /dashboard/settings/username

#### [Dynamic route segments](https://nextjs.org/docs/routing/introduction#dynamic-route-segments)

To match a dynamic segment, you can use the bracket syntax. This allows you to match named parameters.

* pages/blog/[slug].js → /blog/:slug (/blog/hello-world)
* pages/[username]/settings.js → /:username/settings (/foo/settings)
* pages/post/[...all].js → /post/\* (/post/2020/id/title)

Check out the [Dynamic Routes documentation](https://nextjs.org/docs/routing/dynamic-routes) to learn more about how they work.

## [Linking between pages](https://nextjs.org/docs/routing/introduction#linking-between-pages)

The Next.js router allows you to do client-side route transitions between pages, similar to a single-page application.

A React component called Link is provided to do this client-side route transition.

import Link from 'next/link'

function Home() {

return (

<ul>

<li>

<Link href="/">

<a>Home</a>

</Link>

</li>

<li>

<Link href="/about">

<a>About Us</a>

</Link>

</li>

<li>

<Link href="/blog/hello-world">

<a>Blog Post</a>

</Link>

</li>

</ul>

)

}

export default Home

The example above uses multiple links. Each one maps a path (href) to a known page:

* / → pages/index.js
* /about → pages/about.js
* /blog/hello-world → pages/blog/[slug].js

Any <Link /> in the viewport (initially or through scroll) will be prefetched by default (including the corresponding data) for pages using [Static Generation](https://nextjs.org/docs/basic-features/data-fetching#getstaticprops-static-generation). The corresponding data for [server-rendered](https://nextjs.org/docs/basic-features/data-fetching#getserversideprops-server-side-rendering) routes is not prefetched.

### [Linking to dynamic paths](https://nextjs.org/docs/routing/introduction#linking-to-dynamic-paths)

You can also use interpolation to create the path, which comes in handy for [dynamic route segments](https://nextjs.org/docs/routing/introduction" \l "dynamic-route-segments). For example, to show a list of posts which have been passed to the component as a prop:

import Link from 'next/link'

function Posts({ posts }) {

return (

<ul>

{posts.map((post) => (

<li key={post.id}>

<Link href={`/blog/${encodeURIComponent(post.slug)}`}>

<a>{post.title}</a>

</Link>

</li>

))}

</ul>

)

}

export default Posts

[encodeURIComponent](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/encodeURIComponent) is used in the example to keep the path utf-8 compatible.

Alternatively, using a URL Object:

import Link from 'next/link'

function Posts({ posts }) {

return (

<ul>

{posts.map((post) => (

<li key={post.id}>

<Link

href={{

pathname: '/blog/[slug]',

query: { slug: post.slug },

}}

>

<a>{post.title}</a>

</Link>

</li>

))}

</ul>

)

}

export default Posts

Now, instead of using interpolation to create the path, we use a URL object in href where:

* pathname is the name of the page in the pages directory. /blog/[slug] in this case.
* query is an object with the dynamic segment. slug in this case.

## [Injecting the router](https://nextjs.org/docs/routing/introduction#injecting-the-router)

**Examples**

To access the [router object](https://nextjs.org/docs/api-reference/next/router" \l "router-object) in a React component you can use [useRouter](https://nextjs.org/docs/api-reference/next/router" \l "userouter) or [withRouter](https://nextjs.org/docs/api-reference/next/router" \l "withrouter).

In general we recommend using [useRouter](https://nextjs.org/docs/api-reference/next/router" \l "userouter).

## [Learn more](https://nextjs.org/docs/routing/introduction#learn-more)

The router is divided in multiple parts:

#### [next/link](https://nextjs.org/docs/api-reference/next/link)

[Handle client-side navigations.](https://nextjs.org/docs/api-reference/next/link)

#### [next/router](https://nextjs.org/docs/api-reference/next/router)

[Leverage the router API in your pages.](https://nextjs.org/docs/api-reference/next/router)

**Note:** If you need to link to an external page outside the Next.js app, just use an <a> tag without Link.

If you need to add attributes like, for example, className, add it to the a tag, not to the Link tag. [Here’s an example](https://github.com/vercel/next-learn/blob/master/basics/snippets/link-classname-example.js" \t "_blank).

|  |
| --- |
| // Example: Adding className with <Link> |
|  | import Link from 'next/link' |
|  |  |
|  | export default function LinkClassnameExample() { |
|  | // To add attributes like className, target, rel, etc. |
|  | // add them to the <a> tag, not to the <Link> tag. |
|  | return ( |
|  | <Link href="/"> |
|  | <a className="foo" target="\_blank" rel="noopener noreferrer"> |
|  | Hello World |
|  | </a> |
|  | </Link> |
|  | ) |
|  | } |
|  |  |
|  | // Take a look at https://nextjs.org/docs/api-reference/next/link |
|  | // to learn more! |

**Quick Review**: A user opens their browser and navigates to your-blog.com/posts/first-post. What JavaScript is initially loaded for this page?

Only the JavaScript for /posts/first-post is loaded.

JavaScript for the entire application is loaded.

Submit

* [**Assets, Metadata, and CSS**](https://nextjs.org/learn/basics/assets-metadata-css)
  + [Setup](https://nextjs.org/learn/basics/assets-metadata-css/setup)
  + [Assets](https://nextjs.org/learn/basics/assets-metadata-css/assets)
  + [Metadata](https://nextjs.org/learn/basics/assets-metadata-css/metadata)
  + [Third-Party JavaScript](https://nextjs.org/learn/basics/assets-metadata-css/third-party-javascript)
  + [CSS Styling](https://nextjs.org/learn/basics/assets-metadata-css/css-styling)
  + [Layout Component](https://nextjs.org/learn/basics/assets-metadata-css/layout-component)
  + [Global Styles](https://nextjs.org/learn/basics/assets-metadata-css/global-styles)
  + [Polishing Layout](https://nextjs.org/learn/basics/assets-metadata-css/polishing-layout)
  + [Styling Tips](https://nextjs.org/learn/basics/assets-metadata-css/styling-tips)
* [Pre-rendering and Data Fetching](https://nextjs.org/learn/basics/data-fetching)
* [Dynamic Routes](https://nextjs.org/learn/basics/dynamic-routes)
* [API Routes](https://nextjs.org/learn/basics/api-routes)
* [Deploying Your Next.js App](https://nextjs.org/learn/basics/deploying-nextjs-app)

## Assets, Metadata, and CSS

[**1**](https://nextjs.org/learn/basics/assets-metadata-css)

[2](https://nextjs.org/learn/basics/assets-metadata-css/setup)

[3](https://nextjs.org/learn/basics/assets-metadata-css/assets)

[4](https://nextjs.org/learn/basics/assets-metadata-css/metadata)

[5](https://nextjs.org/learn/basics/assets-metadata-css/third-party-javascript)

[6](https://nextjs.org/learn/basics/assets-metadata-css/css-styling)

[7](https://nextjs.org/learn/basics/assets-metadata-css/layout-component)

[8](https://nextjs.org/learn/basics/assets-metadata-css/global-styles)

[9](https://nextjs.org/learn/basics/assets-metadata-css/polishing-layout)

[10](https://nextjs.org/learn/basics/assets-metadata-css/styling-tips)

The second page we added currently does not have styling. Let's add some CSS to style the page.

Next.js has built-in support for [CSS](https://nextjs.org/docs/basic-features/built-in-css-support) and [Sass](https://nextjs.org/docs/basic-features/built-in-css-support" \l "sass-support" \t "_blank). For this course, we will use CSS.

This lesson will also talk about how Next.js handles static assets like images and page metadata like the <title> tag.

### What You’ll Learn in This Lesson

In this lesson, you’ll learn:

* How to add [static files](https://nextjs.org/docs/basic-features/static-file-serving" \t "_blank) (images, etc) to Next.js.
* How to customize what goes inside the <head> for each page.
* How to create a reusable React component which is styled using [CSS Modules](https://nextjs.org/docs/basic-features/built-in-css-support#adding-component-level-css).
* How to [add global CSS](https://nextjs.org/docs/basic-features/built-in-css-support" \l "adding-a-global-stylesheet" \t "_blank) in [pages/\_app.js](https://nextjs.org/docs/advanced-features/custom-app" \t "_blank).
* Some useful tips for styling in Next.js.

### Prerequisites

* Basic CSS knowledge. This course will go over how to add CSS in a Next.js app, but it won't cover CSS fundamentals.

If you’re looking for detailed documentation on Next.js styling, take a look [at the CSS documentation](https://nextjs.org/docs/basic-features/built-in-css-support).

<https://nextjs.org/docs/basic-features/built-in-css-support>

# Built-In CSS Support

**Examples**

* [Basic CSS Example](https://github.com/vercel/next.js/tree/canary/examples/basic-css)
* [With Tailwind CSS](https://github.com/vercel/next.js/tree/canary/examples/with-tailwindcss)

Next.js allows you to import CSS files from a JavaScript file. This is possible because Next.js extends the concept of [import](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import" \t "_blank) beyond JavaScript.

## [Adding a Global Stylesheet](https://nextjs.org/docs/basic-features/built-in-css-support#adding-a-global-stylesheet)

To add a stylesheet to your application, import the CSS file within pages/\_app.js.

For example, consider the following stylesheet named styles.css:

body {

font-family: 'SF Pro Text', 'SF Pro Icons', 'Helvetica Neue', 'Helvetica',

'Arial', sans-serif;

padding: 20px 20px 60px;

max-width: 680px;

margin: 0 auto;

}

Create a [pages/\_app.js file](https://nextjs.org/docs/advanced-features/custom-app) if not already present. Then, [import](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import" \t "_blank) the styles.css file.

import '../styles.css'

// This default export is required in a new `pages/\_app.js` file.

export default function MyApp({ Component, pageProps }) {

return <Component {...pageProps} />

}

These styles (styles.css) will apply to all pages and components in your application. Due to the global nature of stylesheets, and to avoid conflicts, you may **only import them inside [pages/\_app.js](https://nextjs.org/docs/advanced-features/custom-app)**.

In development, expressing stylesheets this way allows your styles to be hot reloaded as you edit them—meaning you can keep application state.

In production, all CSS files will be automatically concatenated into a single minified .css file.

### [Import styles from node\_modules](https://nextjs.org/docs/basic-features/built-in-css-support#import-styles-from-node_modules)

Since Next.js **9.5.4**, importing a CSS file from node\_modules is permitted anywhere in your application.

For global stylesheets, like bootstrap or nprogress, you should import the file inside pages/\_app.js. For example:

// pages/\_app.js

import 'bootstrap/dist/css/bootstrap.css'

export default function MyApp({ Component, pageProps }) {

return <Component {...pageProps} />

}

For importing CSS required by a third party component, you can do so in your component. For example:

// components/ExampleDialog.js

import { useState } from 'react'

import { Dialog } from '@reach/dialog'

import VisuallyHidden from '@reach/visually-hidden'

import '@reach/dialog/styles.css'

function ExampleDialog(props) {

const [showDialog, setShowDialog] = useState(false)

const open = () => setShowDialog(true)

const close = () => setShowDialog(false)

return (

<div>

<button onClick={open}>Open Dialog</button>

<Dialog isOpen={showDialog} onDismiss={close}>

<button className="close-button" onClick={close}>

<VisuallyHidden>Close</VisuallyHidden>

<span aria-hidden>×</span>

</button>

<p>Hello there. I am a dialog</p>

</Dialog>

</div>

)

}

## [Adding Component-Level CSS](https://nextjs.org/docs/basic-features/built-in-css-support#adding-component-level-css)

Next.js supports [CSS Modules](https://github.com/css-modules/css-modules) using the [name].module.css file naming convention.

CSS Modules locally scope CSS by automatically creating a unique class name. This allows you to use the same CSS class name in different files without worrying about collisions.

This behavior makes CSS Modules the ideal way to include component-level CSS. CSS Module files **can be imported anywhere in your application**.

For example, consider a reusable Button component in the components/ folder:

First, create components/Button.module.css with the following content:

/\*

You do not need to worry about .error {} colliding with any other `.css` or

`.module.css` files!

\*/

.error {

color: white;

background-color: red;

}

Then, create components/Button.js, importing and using the above CSS file:

import styles from './Button.module.css'

export function Button() {

return (

<button

type="button"

// Note how the "error" class is accessed as a property on the imported

// `styles` object.

className={styles.error}

>

Destroy

</button>

)

}

CSS Modules are an optional feature and are **only enabled for files with the .module.css extension**. Regular <link> stylesheets and global CSS files are still supported.

In production, all CSS Module files will be automatically concatenated into **many minified and code-split** .css files. These .css files represent hot execution paths in your application, ensuring the minimal amount of CSS is loaded for your application to paint.

## [Sass Support](https://nextjs.org/docs/basic-features/built-in-css-support#sass-support)

Next.js allows you to import Sass using both the .scss and .sass extensions. You can use component-level Sass via CSS Modules and the .module.scss or .module.sass extension.

Before you can use Next.js' built-in Sass support, be sure to install [sass](https://github.com/sass/sass" \t "_blank):

npm install sass

Sass support has the same benefits and restrictions as the built-in CSS support detailed above.

**Note**: Sass supports [two different syntaxes](https://sass-lang.com/documentation/syntax" \t "_blank), each with their own extension. The .scss extension requires you use the [SCSS syntax](https://sass-lang.com/documentation/syntax#scss), while the .sass extension requires you use the [Indented Syntax ("Sass")](https://sass-lang.com/documentation/syntax" \l "the-indented-syntax" \t "_blank).

If you're not sure which to choose, start with the .scss extension which is a superset of CSS, and doesn't require you learn the Indented Syntax ("Sass").

### [Customizing Sass Options](https://nextjs.org/docs/basic-features/built-in-css-support#customizing-sass-options)

If you want to configure the Sass compiler you can do so by using sassOptions in next.config.js.

For example to add includePaths:

const path = require('path')

module.exports = {

sassOptions: {

includePaths: [path.join(\_\_dirname, 'styles')],

},

}

### [Sass Variables](https://nextjs.org/docs/basic-features/built-in-css-support#sass-variables)

Next.js supports Sass variables exported from CSS Module files.

For example, using the exported primaryColor Sass variable:

/\* variables.module.scss \*/

$primary-color: #64FF00

:export {

primaryColor: $primary-color

}

// pages/\_app.js

import variables from '../styles/variables.module.scss'

export default function MyApp({ Component, pageProps }) {

return (

<Layout color={variables.primaryColor}>

<Component {...pageProps} />

</Layout>

)

}

## [CSS-in-JS](https://nextjs.org/docs/basic-features/built-in-css-support#css-in-js)

**Examples**

It's possible to use any existing CSS-in-JS solution. The simplest one is inline styles:

function HiThere() {

return <p style={{ color: 'red' }}>hi there</p>

}

export default HiThere

We bundle [styled-jsx](https://github.com/vercel/styled-jsx" \t "_blank) to provide support for isolated scoped CSS. The aim is to support "shadow CSS" similar to Web Components, which unfortunately [do not support server-rendering and are JS-only](https://github.com/w3c/webcomponents/issues/71).

See the above examples for other popular CSS-in-JS solutions (like Styled Components).

A component using styled-jsx looks like this:

function HelloWorld() {

return (

<div>

Hello world

<p>scoped!</p>

<style jsx>{`

p {

color: blue;

}

div {

background: red;

}

@media (max-width: 600px) {

div {

background: blue;

}

}

`}</style>

<style global jsx>{`

body {

background: black;

}

`}</style>

</div>

)

}

export default HelloWorld

Please see the [styled-jsx documentation](https://github.com/vercel/styled-jsx" \t "_blank) for more examples.

## [FAQ](https://nextjs.org/docs/basic-features/built-in-css-support#faq)

### [Does it work with JavaScript disabled?](https://nextjs.org/docs/basic-features/built-in-css-support#does-it-work-with-javascript-disabled)

Yes, if you disable JavaScript the CSS will still be loaded in the production build (next start). During development, we require JavaScript to be enabled to provide the best developer experience with [Fast Refresh](https://nextjs.org/blog/next-9-4#fast-refresh).

## [Related](https://nextjs.org/docs/basic-features/built-in-css-support#related)

For more information on what to do next, we recommend the following sections:

#### [Customizing PostCSS Config](https://nextjs.org/docs/advanced-features/customizing-postcss-config)

[Extend the PostCSS config and plugins added by Next.js with your own.](https://nextjs.org/docs/advanced-features/customizing-postcss-config)

[Start Now →](https://nextjs.org/learn/basics/assets-metadata-css/setup)

## Assets, Metadata, and CSS

[1](https://nextjs.org/learn/basics/assets-metadata-css)

[**2**](https://nextjs.org/learn/basics/assets-metadata-css/setup)

[3](https://nextjs.org/learn/basics/assets-metadata-css/assets)

[4](https://nextjs.org/learn/basics/assets-metadata-css/metadata)

[5](https://nextjs.org/learn/basics/assets-metadata-css/third-party-javascript)

[6](https://nextjs.org/learn/basics/assets-metadata-css/css-styling)

[7](https://nextjs.org/learn/basics/assets-metadata-css/layout-component)

[8](https://nextjs.org/learn/basics/assets-metadata-css/global-styles)

[9](https://nextjs.org/learn/basics/assets-metadata-css/polishing-layout)

[10](https://nextjs.org/learn/basics/assets-metadata-css/styling-tips)

**If you’re continuing from the previous lesson,** you can skip this page. Click the button below to go to the next page.

[Next](https://nextjs.org/learn/basics/assets-metadata-css/assets)

### Download Starter Code (Optional)

If you’re NOT continuing from the previous lesson, you can download, install, and run the starter code for this lesson below. This sets up a nextjs-blog directory such that it’s identical to the result of the previous lesson.

Again, this is NOT necessary if you’ve just finished the previous lesson.

npx create-next-app nextjs-blog --use-npm --example "https://github.com/vercel/next-learn/tree/master/basics/assets-metadata-css-starter"

Then follow the instructions from the command output (cd into the directory and start the development server).

## Assets, Metadata, and CSS

[1](https://nextjs.org/learn/basics/assets-metadata-css)

[2](https://nextjs.org/learn/basics/assets-metadata-css/setup)

[**3**](https://nextjs.org/learn/basics/assets-metadata-css/assets)

[4](https://nextjs.org/learn/basics/assets-metadata-css/metadata)

[5](https://nextjs.org/learn/basics/assets-metadata-css/third-party-javascript)

[6](https://nextjs.org/learn/basics/assets-metadata-css/css-styling)

[7](https://nextjs.org/learn/basics/assets-metadata-css/layout-component)

[8](https://nextjs.org/learn/basics/assets-metadata-css/global-styles)

[9](https://nextjs.org/learn/basics/assets-metadata-css/polishing-layout)

[10](https://nextjs.org/learn/basics/assets-metadata-css/styling-tips)

## Assets

Next.js can serve **static assets**, like images, under **the top-level [public directory](https://nextjs.org/docs/basic-features/static-file-serving" \t "_blank)**. Files inside public can be referenced from the root of the application similar to [pages](https://nextjs.org/docs/basic-features/pages" \t "_blank).

The public directory is also useful for robots.txt, Google Site Verification, and any other static assets. Check out the documentation for [Static File Serving](https://nextjs.org/docs/basic-features/static-file-serving) to learn more.

<https://nextjs.org/docs/basic-features/static-file-serving>

# Static File Serving

Next.js can serve static files, like images, under a folder called public in the root directory. Files inside public can then be referenced by your code starting from the base URL (/).

For example, if you add an image to public/me.png, the following code will access the image:

import Image from 'next/image'

function Avatar() {

return <Image src="/me.png" alt="me" width="64" height="64" />

}

export default Avatar

Note: next/image requires Next.js 10 or later.

This folder is also useful for robots.txt, favicon.ico, Google Site Verification, and any other static files (including .html)!

**Note**: Don't name the public directory anything else. The name cannot be changed and is the only directory used to serve static assets.

**Note**: Be sure to not have a static file with the same name as a file in the pages/ directory, as this will result in an error.

Read more: <https://nextjs.org/docs/messages/conflicting-public-file-page>

**Note**: Only assets that are in the public directory at [build time](https://nextjs.org/docs/api-reference/cli" \l "build) will be served by Next.js. Files added at runtime won't be available. We recommend using a third party service like [AWS S3](https://aws.amazon.com/s3/) for persistent file storage.

### Download Your Profile Picture

First, let's retrieve your profile picture.

* **Download** your profile picture in .jpg format (or [use this file](https://github.com/vercel/next-learn/blob/master/basics/basics-final/public/images/profile.jpg)).



* Create an images directory inside of the [public directory](https://nextjs.org/docs/basic-features/static-file-serving" \t "_blank).
* Save the picture as profile.jpg in the public/images directory.
* The image size can be around 400px by 400px.
* You may remove the unused SVG logo file directly under the [public directory](https://nextjs.org/docs/basic-features/static-file-serving" \t "_blank).

### Unoptimized Image

With regular HTML, you would add your profile picture as follows:

<img src="/images/profile.jpg" alt="Your Name" />

However, this means you have to manually handle:

* Ensuring your image is responsive on different screen sizes
* Optimizing your images with a third-party tool or library
* Only loading images when they enter the viewport

And more. Instead, Next.js provides an Image component out of the box to handle this for you.

### Image Component and Image Optimization

[next/image](https://nextjs.org/docs/api-reference/next/image) is an extension of the HTML <img> element, evolved for the modern web.

Next.js also has support for Image Optimization by default. This allows for resizing, optimizing, and serving images in modern formats like [WebP](https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Image_types" \l "webp" \t "_blank) when the browser supports it. This avoids shipping large images to devices with a smaller viewport. It also allows Next.js to automatically adopt future image formats and serve them to browsers that support those formats.

Automatic Image Optimization works with any image source. Even if the image is hosted by an external data source, like a CMS, it can still be optimized.

### Using the Image Component

Instead of optimizing images at build time, Next.js optimizes images on-demand, as users request them. Unlike static site generators and static-only solutions, your build times aren't increased, whether shipping 10 images or 10 million images.

Images are lazy loaded by default. That means your page speed isn't penalized for images outside the viewport. Images load as they are scrolled into viewport.

Images are always rendered in such a way as to avoid [Cumulative Layout Shift](https://web.dev/cls/" \t "_blank), a [Core Web Vital](https://web.dev/vitals/#core-web-vitals) that Google is going to [use in search ranking](https://webmasters.googleblog.com/2020/05/evaluating-page-experience.html" \t "_blank).

Here's an example using [next/image](https://nextjs.org/docs/api-reference/next/image.md" \t "_blank) to display our profile picture. The height and width props should be the desired rendering size, with an aspect ratio identical to the source image.

**Note:** We'll use this component later in "Polishing Layout", no need to copy it yet.

import Image from 'next/image'

const YourComponent = () => (

<Image

src="/images/profile.jpg" // Route of the image file

height={144} // Desired size with correct aspect ratio

width={144} // Desired size with correct aspect ratio

alt="Your Name"

/>

)

To learn more about Automatic Image Optimization, check out the [documentation](https://nextjs.org/docs/basic-features/image-optimization).

<https://nextjs.org/docs/basic-features/image-optimization>

# Image Component and Image Optimization

**Examples**

* [Image Component](https://github.com/vercel/next.js/tree/canary/examples/image-component)

The Next.js Image component, [next/image](https://nextjs.org/docs/api-reference/next/image), is an extension of the HTML <img> element, evolved for the modern web. It includes a variety of built-in performance optimizations to help you achieve good [Core Web Vitals](https://nextjs.org/learn/seo/web-performance). These scores are an important measurement of user experience on your website, and are [factored into Google's search rankings](https://nextjs.org/learn/seo/web-performance/seo-impact).

Some of the optimizations built into the Image component include:

* **Improved Performance:** Always serve correctly sized image for each device, using modern image formats.
* **Visual Stability:** Prevent [Cumulative Layout Shift](https://nextjs.org/learn/seo/web-performance/cls) automatically.
* **Faster Page Loads:** Images are only loaded when they enter the viewport, with optional blur-up placeholders
* **Asset Flexibility:** On-demand image resizing, even for images stored on remote servers

## [Using the Image Component](https://nextjs.org/docs/basic-features/image-optimization#using-the-image-component)

To add an image to your application, import the [next/image](https://nextjs.org/docs/api-reference/next/image) component:

import Image from 'next/image'

Now, you can define the src for your image (either local or remote).

### [Local Images](https://nextjs.org/docs/basic-features/image-optimization#local-images)

To use a local image, import your .jpg, .png, or .webp files:

import profilePic from '../public/me.png'

Dynamic await import() or require() are not supported. The import must be static so it can be analyzed at build time.

Next.js will automatically determine the width and height of your image based on the imported file. These values are used to prevent [Cumulative Layout Shift](https://nextjs.org/learn/seo/web-performance/cls) while your image is loading.

import Image from 'next/image'

import profilePic from '../public/me.png'

function Home() {

return (

<>

<h1>My Homepage</h1>

<Image

src={profilePic}

alt="Picture of the author"

// width={500} automatically provided

// height={500} automatically provided

// blurDataURL="data:..." automatically provided

// placeholder="blur" // Optional blur-up while loading

/>

<p>Welcome to my homepage!</p>

</>

)

}

### [Remote Images](https://nextjs.org/docs/basic-features/image-optimization#remote-images)

To use a remote image, the src property should be a URL string, which can be [relative](https://nextjs.org/docs/basic-features/image-optimization#loaders) or [absolute](https://nextjs.org/docs/basic-features/image-optimization#domains). Because Next.js does not have access to remote files during the build process, you'll need to provide the [width](https://nextjs.org/docs/api-reference/next/image" \l "width), [height](https://nextjs.org/docs/api-reference/next/image" \l "height) and optional [blurDataURL](https://nextjs.org/docs/api-reference/next/image" \l "blurdataurl) props manually:

import Image from 'next/image'

export default function Home() {

return (

<>

<h1>My Homepage</h1>

<Image

src="/me.png"

alt="Picture of the author"

width={500}

height={500}

/>

<p>Welcome to my homepage!</p>

</>

)

}

Learn more about the [sizing requirements](https://nextjs.org/docs/basic-features/image-optimization" \l "image-sizing) in next/image.

### [Domains](https://nextjs.org/docs/basic-features/image-optimization#domains)

Sometimes you may want to access a remote image, but still use the built-in Next.js Image Optimization API. To do this, leave the loader at its default setting and enter an absolute URL for the Image src.

To protect your application from malicious users, you must define a list of remote domains that you intend to access this way. This is configured in your next.config.js file, as shown below:

module.exports = {

images: {

domains: ['example.com', 'example2.com'],

},

}

### [Loaders](https://nextjs.org/docs/basic-features/image-optimization#loaders)

Note that in the [example earlier](https://nextjs.org/docs/basic-features/image-optimization" \l "remote-images), a partial URL ("/me.png") is provided for a remote image. This is possible because of the next/image [loader](https://nextjs.org/docs/api-reference/next/image" \l "loader) architecture.

A loader is a function that generates the URLs for your image. It appends a root domain to your provided src, and generates multiple URLs to request the image at different sizes. These multiple URLs are used in the automatic [srcset](https://developer.mozilla.org/en-US/docs/Web/API/HTMLImageElement/srcset" \t "_blank) generation, so that visitors to your site will be served an image that is the right size for their viewport.

The default loader for Next.js applications uses the built-in Image Optimization API, which optimizes images from anywhere on the web, and then serves them directly from the Next.js web server. If you would like to serve your images directly from a CDN or image server, you can use one of the [built-in loaders](https://nextjs.org/docs/api-reference/next/image" \l "built-in-loaders) or write your own with a few lines of JavaScript.

Loaders can be defined per-image, or at the application level.

### [Priority](https://nextjs.org/docs/basic-features/image-optimization#priority)

You should add the priority property to the image that will be the [Largest Contentful Paint (LCP) element](https://web.dev/lcp/" \l "what-elements-are-considered" \t "_blank) for each page. Doing so allows Next.js to specially prioritize the image for loading (e.g. through preload tags or priority hints), leading to a meaningful boost in LCP.

The LCP element is typically the largest image or text block visible within the viewport of the page. When you run next dev, you'll see a console warning if the LCP element is an <Image> without the priority property.

Once you've identified the LCP image, you can add the property like this:

import Image from 'next/image'

export default function Home() {

return (

<>

<h1>My Homepage</h1>

<Image

src="/me.png"

alt="Picture of the author"

width={500}

height={500}

priority

/>

<p>Welcome to my homepage!</p>

</>

)

}

See more about priority in the [next/image component documentation](https://nextjs.org/docs/api-reference/next/image" \l "priority).

## [Image Sizing](https://nextjs.org/docs/basic-features/image-optimization#image-sizing)

One of the ways that images most commonly hurt performance is through layout shift, where the image pushes other elements around on the page as it loads in. This performance problem is so annoying to users that it has its own Core Web Vital, called [Cumulative Layout Shift](https://web.dev/cls/" \t "_blank). The way to avoid image-based layout shifts is to [always size your images](https://web.dev/optimize-cls/" \l "images-without-dimensions" \t "_blank). This allows the browser to reserve precisely enough space for the image before it loads.

Because next/image is designed to guarantee good performance results, it cannot be used in a way that will contribute to layout shift, and **must** be sized in one of three ways:

1. Automatically, using a [static import](https://nextjs.org/docs/basic-features/image-optimization" \l "local-images)
2. Explicitly, by including a height **and** width property
3. Implicitly, by using layout="fill" which causes the image to expand to fill its parent element.

### [What if I don't know the size of my images?](https://nextjs.org/docs/basic-features/image-optimization#what-if-i-dont-know-the-size-of-my-images)

If you are accessing images from a source without knowledge of the images' sizes, there are several things you can do:

**Use layout='fill'**

The fill layout mode allows your image to be sized by its parent element. Consider using CSS to give the image's parent element space on the page, then using the [objectFit property](https://nextjs.org/docs/api-reference/next/image" \l "objectfit) with fill, contain, or cover, along with the [objectPosition property](https://nextjs.org/docs/api-reference/next/image" \l "objectposition) to define how the image should occupy that space.

**Normalize your images**

If you're serving images from a source that you control, consider modifying your image pipeline to normalize the images to a specific size.

**Modify your API calls**

If your application is retrieving image URLs using an API call (such as to a CMS), you may be able to modify the API call to return the image dimensions along with the URL.

If none of the suggested methods works for sizing your images, the next/image component is designed to work well on a page alongside standard <img> elements.

## [Styling](https://nextjs.org/docs/basic-features/image-optimization#styling)

Styling the Image component is not that different from styling a normal <img> element, but there are a few guidelines to keep in mind:

**Pick the correct layout mode**

The image component has several different [layout modes](https://nextjs.org/docs/api-reference/next/image" \l "layout) that define how it is sized on the page. If the styling of your image isn't turning out the way you want, consider experimenting with other layout modes.

**Target the image with className, not based on DOM structure**

Regardless of the layout mode used, the Image component will have a consistent DOM structure of one <img> tag wrapped by exactly one <span>. For some modes, it may also have a sibling <span> for spacing. These additional <span> elements are critical to allow the component to prevent layout shifts.

The recommended way to style the inner <img> is to set the className prop on the Image component to the value of an imported [CSS Module](https://nextjs.org/docs/basic-features/built-in-css-support#adding-component-level-css). The value of className will be automatically applied to the underlying <img> element.

Alternatively, you can import a [global stylesheet](https://nextjs.org/docs/basic-features/built-in-css-support#adding-a-global-stylesheet) and manually set the className prop to the same name used in the global stylesheet.

You cannot use [styled-jsx](https://nextjs.org/docs/basic-features/built-in-css-support" \l "css-in-js) because it's scoped to the current component.

You cannot use the style prop because the <Image> component does not pass it through to the underlying <img>.

**When using layout='fill', the parent element must have position: relative**

This is necessary for the proper rendering of the image element in that layout mode.

**When using layout='responsive', the parent element must have display: block**

This is the default for <div> elements but should be specified otherwise.

## [Properties](https://nextjs.org/docs/basic-features/image-optimization#properties)

[**View all properties available to the next/image component.**](https://nextjs.org/docs/api-reference/next/image)

### [Styling Examples](https://nextjs.org/docs/basic-features/image-optimization#styling-examples)

For examples of the Image component used with the various fill modes, see the [Image component example app](https://image-component.nextjs.gallery/).

## [Configuration](https://nextjs.org/docs/basic-features/image-optimization#configuration)

The next/image component and Next.js Image Optimization API can be configured in the [next.config.js file](https://nextjs.org/docs/api-reference/next.config.js/introduction). These configurations allow you to [enable remote domains](https://nextjs.org/docs/api-reference/next/image" \l "domains), [define custom image breakpoints](https://nextjs.org/docs/api-reference/next/image" \l "device-sizes), [change caching behavior](https://nextjs.org/docs/api-reference/next/image" \l "caching-behavior) and more.

[**Read the full image configuration documentation for more information.**](https://nextjs.org/docs/api-reference/next/image#configuration-options)

## [Related](https://nextjs.org/docs/basic-features/image-optimization#related)

For more information on what to do next, we recommend the following sections:

#### [next/image](https://nextjs.org/docs/api-reference/next/image)

[See all available properties for the Image component](https://nextjs.org/docs/api-reference/next/image)

To learn more about the Image component, check out the [API reference for next/image](https://nextjs.org/docs/api-reference/next/image).

# next/image

**ExamplesVersion History**

**Note: This is API documentation for the Image Component and Image Optimization. For a feature overview and usage information for images in Next.js, please see**[**Images**](https://nextjs.org/docs/basic-features/image-optimization)**.**

## [Required Props](https://nextjs.org/docs/api-reference/next/image#required-props)

The <Image /> component requires the following properties.

### [src](https://nextjs.org/docs/api-reference/next/image#src)

Must be one of the following:

1. A [statically imported](https://nextjs.org/docs/basic-features/image-optimization" \l "local-images) image file, or
2. A path string. This can be either an absolute external URL, or an internal path depending on the [loader](https://nextjs.org/docs/api-reference/next/image" \l "loader) prop or [loader configuration](https://nextjs.org/docs/api-reference/next/image" \l "loader-configuration).

When using an external URL, you must add it to [domains](https://nextjs.org/docs/api-reference/next/image" \l "domains) in next.config.js.

### [width](https://nextjs.org/docs/api-reference/next/image#width)

The width of the image, in pixels. Must be an integer without a unit.

Required, except for statically imported images, or those with [layout="fill"](https://nextjs.org/docs/api-reference/next/image" \l "layout).

### [height](https://nextjs.org/docs/api-reference/next/image#height)

The height of the image, in pixels. Must be an integer without a unit.

Required, except for statically imported images, or those with [layout="fill"](https://nextjs.org/docs/api-reference/next/image" \l "layout).

## [Optional Props](https://nextjs.org/docs/api-reference/next/image#optional-props)

The <Image /> component accepts a number of additional properties beyond those which are required. This section describes the most commonly-used properties of the Image component. Find details about more rarely-used properties in the [Advanced Props](https://nextjs.org/docs/api-reference/next/image" \l "advanced-props) section.

### [layout](https://nextjs.org/docs/api-reference/next/image#layout)

The layout behavior of the image as the viewport changes size.

| **layout** | **Behavior** | **srcSet** | **sizes** |
| --- | --- | --- | --- |
| intrinsic (default) | Scale down to fit width of container, up to image size | 1x, 2x (based on [imageSizes](https://nextjs.org/docs/api-reference/next/image" \l "image-sizes)) | N/A |
| fixed | Sized to width and height exactly | 1x, 2x (based on [imageSizes](https://nextjs.org/docs/api-reference/next/image" \l "image-sizes)) | N/A |
| responsive | Scale to fit width of container | 640w, 750w, ... 2048w, 3840w (based on [imageSizes](https://nextjs.org/docs/api-reference/next/image" \l "image-sizes) and [deviceSizes](https://nextjs.org/docs/api-reference/next/image" \l "device-sizes)) | 100vw |
| fill | Grow in both X and Y axes to fill container | 640w, 750w, ... 2048w, 3840w (based on [imageSizes](https://nextjs.org/docs/api-reference/next/image" \l "image-sizes) and [deviceSizes](https://nextjs.org/docs/api-reference/next/image" \l "device-sizes)) | 100vw |

* [Demo the intrinsic layout (default)](https://image-component.nextjs.gallery/layout-intrinsic)
  + When intrinsic, the image will scale the dimensions down for smaller viewports, but maintain the original dimensions for larger viewports.
* [Demo the fixed layout](https://image-component.nextjs.gallery/layout-fixed)
  + When fixed, the image dimensions will not change as the viewport changes (no responsiveness) similar to the native img element.
* [Demo the responsive layout](https://image-component.nextjs.gallery/layout-responsive)
  + When responsive, the image will scale the dimensions down for smaller viewports and scale up for larger viewports.
  + Ensure the parent element uses display: block in their stylesheet.
* [Demo the fill layout](https://image-component.nextjs.gallery/layout-fill)
  + When fill, the image will stretch both width and height to the dimensions of the parent element, provided the parent element is relative.
  + This is usually paired with the [objectFit](https://nextjs.org/docs/api-reference/next/image" \l "objectfit) property.
  + Ensure the parent element has position: relative in their stylesheet.
* [Demo background image](https://image-component.nextjs.gallery/background)

### [loader](https://nextjs.org/docs/api-reference/next/image#loader)

A custom function used to resolve URLs. Setting the loader as a prop on the Image component overrides the default loader defined in the [images section of next.config.js](https://nextjs.org/docs/api-reference/next/image" \l "loader-configuration).

A loader is a function returning a URL string for the image, given the following parameters:

* [src](https://nextjs.org/docs/api-reference/next/image#src)
* [width](https://nextjs.org/docs/api-reference/next/image#width)
* [quality](https://nextjs.org/docs/api-reference/next/image#quality)

Here is an example of using a custom loader with next/image:

import Image from 'next/image'

const myLoader = ({ src, width, quality }) => {

return `https://example.com/${src}?w=${width}&q=${quality || 75}`

}

const MyImage = (props) => {

return (

<Image

loader={myLoader}

src="me.png"

alt="Picture of the author"

width={500}

height={500}

/>

)

}

### [sizes](https://nextjs.org/docs/api-reference/next/image#sizes)

A string that provides information about how wide the image will be at different breakpoints. Defaults to 100vw (the full width of the screen) when using layout="responsive" or layout="fill".

If you are using layout="fill" or layout="responsive", it's important to assign sizes for any image that takes up less than the full viewport width.

For example, when the parent element will constrain the image to always be less than half the viewport width, use sizes="50vw". Without sizes, the image will be sent at twice the necessary resolution, decreasing performance.

If you are using layout="intrinsic" or layout="fixed", then sizes is not needed because the upper bound width is constrained already.

[Learn more](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img#attr-sizes).

### [quality](https://nextjs.org/docs/api-reference/next/image#quality)

The quality of the optimized image, an integer between 1 and 100 where 100 is the best quality. Defaults to 75.

### [priority](https://nextjs.org/docs/api-reference/next/image#priority)

When true, the image will be considered high priority and [preload](https://web.dev/preload-responsive-images/" \t "_blank). Lazy loading is automatically disabled for images using priority.

You should use the priority property on any image detected as the [Largest Contentful Paint (LCP)](https://nextjs.org/learn/seo/web-performance/lcp) element. It may be appropriate to have multiple priority images, as different images may be the LCP element for different viewport sizes.

Should only be used when the image is visible above the fold. Defaults to false.

### [placeholder](https://nextjs.org/docs/api-reference/next/image#placeholder)

A placeholder to use while the image is loading. Possible values are blur or empty. Defaults to empty.

When blur, the [blurDataURL](https://nextjs.org/docs/api-reference/next/image" \l "blurdataurl) property will be used as the placeholder. If src is an object from a [static import](https://nextjs.org/docs/basic-features/image-optimization" \l "local-images) and the imported image is .jpg, .png, .webp, or .avif, then blurDataURL will be automatically populated.

For dynamic images, you must provide the [blurDataURL](https://nextjs.org/docs/api-reference/next/image" \l "blurdataurl) property. Solutions such as [Plaiceholder](https://github.com/joe-bell/plaiceholder" \t "_blank) can help with base64 generation.

When empty, there will be no placeholder while the image is loading, only empty space.

Try it out:

* [Demo the blur placeholder](https://image-component.nextjs.gallery/placeholder)
* [Demo the shimmer effect with blurDataURL prop](https://image-component.nextjs.gallery/shimmer)
* [Demo the color effect with blurDataURL prop](https://image-component.nextjs.gallery/color)

## [Advanced Props](https://nextjs.org/docs/api-reference/next/image#advanced-props)

In some cases, you may need more advanced usage. The <Image /> component optionally accepts the following advanced properties.

### [objectFit](https://nextjs.org/docs/api-reference/next/image#objectfit)

Defines how the image will fit into its parent container when using layout="fill".

This value is passed to the [object-fit CSS property](https://developer.mozilla.org/en-US/docs/Web/CSS/object-fit" \t "_blank) for the src image.

### [objectPosition](https://nextjs.org/docs/api-reference/next/image#objectposition)

Defines how the image is positioned within its parent element when using layout="fill".

This value is passed to the [object-position CSS property](https://developer.mozilla.org/en-US/docs/Web/CSS/object-position" \t "_blank) applied to the image.

### [onLoadingComplete](https://nextjs.org/docs/api-reference/next/image#onloadingcomplete)

A callback function that is invoked once the image is completely loaded and the [placeholder](https://nextjs.org/docs/api-reference/next/image" \l "placeholder) has been removed.

The onLoadingComplete function accepts one parameter, an object with the following properties:

* [naturalWidth](https://developer.mozilla.org/en-US/docs/Web/API/HTMLImageElement/naturalWidth)
* [naturalHeight](https://developer.mozilla.org/en-US/docs/Web/API/HTMLImageElement/naturalHeight)

### [loading](https://nextjs.org/docs/api-reference/next/image#loading)

**Attention**: This property is only meant for advanced usage. Switching an image to load with eager will normally **hurt performance**.

We recommend using the [priority](https://nextjs.org/docs/api-reference/next/image" \l "priority) property instead, which properly loads the image eagerly for nearly all use cases.

The loading behavior of the image. Defaults to lazy.

When lazy, defer loading the image until it reaches a calculated distance from the viewport.

When eager, load the image immediately.

[Learn more](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img#attr-loading)

### [blurDataURL](https://nextjs.org/docs/api-reference/next/image#blurdataurl)

A [Data URL](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Data_URIs) to be used as a placeholder image before the src image successfully loads. Only takes effect when combined with [placeholder="blur"](https://nextjs.org/docs/api-reference/next/image" \l "placeholder).

Must be a base64-encoded image. It will be enlarged and blurred, so a very small image (10px or less) is recommended. Including larger images as placeholders may harm your application performance.

Try it out:

* [Demo the default blurDataURL prop](https://image-component.nextjs.gallery/placeholder)
* [Demo the shimmer effect with blurDataURL prop](https://image-component.nextjs.gallery/shimmer)
* [Demo the color effect with blurDataURL prop](https://image-component.nextjs.gallery/color)

You can also [generate a solid color Data URL](https://png-pixel.com/" \t "_blank) to match the image.

### [lazyBoundary](https://nextjs.org/docs/api-reference/next/image#lazyboundary)

A string (with similar syntax to the margin property) that acts as the bounding box used to detect the intersection of the viewport with the image and trigger lazy [loading](https://nextjs.org/docs/api-reference/next/image" \l "loading). Defaults to "200px".

[Learn more](https://developer.mozilla.org/en-US/docs/Web/API/IntersectionObserver/rootMargin)

### [unoptimized](https://nextjs.org/docs/api-reference/next/image#unoptimized)

When true, the source image will be served as-is instead of changing quality, size, or format. Defaults to false.

## [Other Props](https://nextjs.org/docs/api-reference/next/image#other-props)

Other properties on the <Image /> component will be passed to the underlying img element with the exception of the following:

* style. Use className instead.
* srcSet. Use [Device Sizes](https://nextjs.org/docs/api-reference/next/image#device-sizes) instead.
* ref. Use [onLoadingComplete](https://nextjs.org/docs/api-reference/next/image" \l "onloadingcomplete) instead.
* decoding. It is always "async".

## [Configuration Options](https://nextjs.org/docs/api-reference/next/image#configuration-options)

### [Domains](https://nextjs.org/docs/api-reference/next/image#domains)

To protect your application from malicious users, you must define a list of image provider domains that you want to be served from the Next.js Image Optimization API. This is configured in with the domains property in your next.config.js file, as shown below:

module.exports = {

images: {

domains: ['assets.acme.com'],

},

}

### [Loader Configuration](https://nextjs.org/docs/api-reference/next/image#loader-configuration)

If you want to use a cloud provider to optimize images instead of using the Next.js built-in Image Optimization API, you can configure the loader and path prefix in your next.config.js file. This allows you to use relative URLs for the Image [src](https://nextjs.org/docs/api-reference/next/image" \l "src) and automatically generate the correct absolute URL for your provider.

module.exports = {

images: {

loader: 'imgix',

path: 'https://example.com/myaccount/',

},

}

### [Built-in Loaders](https://nextjs.org/docs/api-reference/next/image#built-in-loaders)

The following Image Optimization cloud providers are included:

* Default: Works automatically with next dev, next start, or a custom server
* [Vercel](https://vercel.com/): Works automatically when you deploy on Vercel, no configuration necessary. [Learn more](https://vercel.com/docs/next.js/image-optimization" \t "_blank)
* [Imgix](https://www.imgix.com/): loader: 'imgix'
* [Cloudinary](https://cloudinary.com/): loader: 'cloudinary'
* [Akamai](https://www.akamai.com/): loader: 'akamai'
* Custom: loader: 'custom' use a custom cloud provider by implementing the [loader](https://nextjs.org/docs/api-reference/next/image" \l "loader) prop on the next/image component

If you need a different provider, you can use the [loader](https://nextjs.org/docs/api-reference/next/image" \l "loader) prop with next/image.

The next/image component's default loader is not supported when using [next export](https://nextjs.org/docs/advanced-features/static-html-export). However, other loader options will work.

The next/image component's default loader uses [squoosh](https://www.npmjs.com/package/@squoosh/lib" \t "_blank) because it is quick to install and suitable for a development environment. When using next start in your production environment, it is strongly recommended that you install [sharp](https://www.npmjs.com/package/sharp) by running yarn add sharp in your project directory. This is not necessary for Vercel deployments, as sharp is installed automatically.

## [Advanced](https://nextjs.org/docs/api-reference/next/image#advanced)

The following configuration is for advanced use cases and is usually not necessary. If you choose to configure the properties below, you will override any changes to the Next.js defaults in future updates.

### [Device Sizes](https://nextjs.org/docs/api-reference/next/image#device-sizes)

If you know the expected device widths of your users, you can specify a list of device width breakpoints using the deviceSizes property in next.config.js. These widths are used when the [next/image](https://nextjs.org/docs/api-reference/next/image) component uses layout="responsive" or layout="fill" to ensure the correct image is served for user's device.

If no configuration is provided, the default below is used.

module.exports = {

images: {

deviceSizes: [640, 750, 828, 1080, 1200, 1920, 2048, 3840],

},

}

### [Image Sizes](https://nextjs.org/docs/api-reference/next/image#image-sizes)

You can specify a list of image widths using the images.imageSizes property in your next.config.js file. These widths are concatenated with the array of [device sizes](https://nextjs.org/docs/api-reference/next/image" \l "device-sizes) to form the full array of sizes used to generate image [srcset](https://developer.mozilla.org/en-US/docs/Web/API/HTMLImageElement/srcset" \t "_blank)s.

The reason there are two separate lists is that imageSizes is only used for images which provide a [sizes](https://nextjs.org/docs/api-reference/next/image" \l "sizes) prop, which indicates that the image is less than the full width of the screen. **Therefore, the sizes in imageSizes should all be smaller than the smallest size in deviceSizes.**

If no configuration is provided, the default below is used.

module.exports = {

images: {

imageSizes: [16, 32, 48, 64, 96, 128, 256, 384],

},

}

### [Acceptable Formats](https://nextjs.org/docs/api-reference/next/image#acceptable-formats)

The default [Image Optimization API](https://nextjs.org/docs/api-reference/next/image#loader-configuration) will automatically detect the browser's supported image formats via the request's Accept header.

If the Accept head matches more than one of the configured formats, the first match in the array is used. Therefore, the array order matters. If there is no match, the Image Optimization API will fallback to the original image's format.

If no configuration is provided, the default below is used.

module.exports = {

images: {

formats: ['image/webp'],

},

}

You can enable AVIF support with the following configuration.

module.exports = {

images: {

formats: ['image/avif', 'image/webp'],

},

}

Note: AVIF generally takes 20% longer to encode but it compresses 20% smaller compared to WebP. This means that the first time an image is requested, it will typically be slower and then subsequent requests that are cached will be faster.

## [Caching Behavior](https://nextjs.org/docs/api-reference/next/image#caching-behavior)

The following describes the caching algorithm for the default [loader](https://nextjs.org/docs/api-reference/next/image" \l "loader). For all other loaders, please refer to your cloud provider's documentation.

Images are optimized dynamically upon request and stored in the <distDir>/cache/images directory. The optimized image file will be served for subsequent requests until the expiration is reached. When a request is made that matches a cached but expired file, the cached file is deleted before generating a new optimized image and caching the new file.

The expiration (or rather Max Age) is defined by either the [minimumCacheTTL](https://nextjs.org/docs/api-reference/next/image" \l "minimum-cache-ttl) configuration or the upstream server's Cache-Control header, whichever is larger. Specifically, the max-age value of the Cache-Control header is used. If both s-maxage and max-age are found, then s-maxage is preferred.

* You can configure [minimumCacheTTL](https://nextjs.org/docs/api-reference/next/image" \l "minimum-cache-ttl) to increase the cache duration when the upstream image does not include Cache-Control header or the value is very low.
* You can configure [deviceSizes](https://nextjs.org/docs/api-reference/next/image" \l "device-sizes) and [imageSizes](https://nextjs.org/docs/api-reference/next/image" \l "device-sizes) to reduce the total number of possible generated images.
* You can configure [formats](https://nextjs.org/docs/basic-features/image-optimization" \l "acceptable-formats) to disable multiple formats in favor of a single image format.

### [Minimum Cache TTL](https://nextjs.org/docs/api-reference/next/image#minimum-cache-ttl)

You can configure the Time to Live (TTL) in seconds for cached optimized images. In many cases, it's better to use a [Static Image Import](https://nextjs.org/docs/basic-features/image-optimization#local-images) which will automatically hash the file contents and cache the image forever with a Cache-Control header of immutable.

module.exports = {

images: {

minimumCacheTTL: 60,

},

}

If you need to add a Cache-Control header for the browser (not recommended), you can configure [headers](https://nextjs.org/docs/api-reference/next.config.js/headers) on the upstream image e.g. /some-asset.jpg not /\_next/image itself.

### [Disable Static Imports](https://nextjs.org/docs/api-reference/next/image#disable-static-imports)

The default behavior allows you to import static files such as import icon from './icon.png and then pass that to the src property.

In some cases, you may wish to disable this feature if it conflicts with other plugins that expect the import to behave differently.

You can disable static image imports inside your next.config.js:

module.exports = {

images: {

disableStaticImages: true,

},

}

## [Related](https://nextjs.org/docs/api-reference/next/image#related)

For an overview of the Image component features and usage guidelines, see:

#### [Images](https://nextjs.org/docs/basic-features/image-optimization)

[Learn how to display and optimize images with the Image component.](https://nextjs.org/docs/basic-features/image-optimization)

**Quick Review**: What does next/image simplify for you?

Uploading & storing images

Resizing & optimizing images

Cropping & color correcting images

Submit

## Assets, Metadata, and CSS

[1](https://nextjs.org/learn/basics/assets-metadata-css)

[2](https://nextjs.org/learn/basics/assets-metadata-css/setup)

[3](https://nextjs.org/learn/basics/assets-metadata-css/assets)

[**4**](https://nextjs.org/learn/basics/assets-metadata-css/metadata)

[5](https://nextjs.org/learn/basics/assets-metadata-css/third-party-javascript)

[6](https://nextjs.org/learn/basics/assets-metadata-css/css-styling)

[7](https://nextjs.org/learn/basics/assets-metadata-css/layout-component)

[8](https://nextjs.org/learn/basics/assets-metadata-css/global-styles)

[9](https://nextjs.org/learn/basics/assets-metadata-css/polishing-layout)

[10](https://nextjs.org/learn/basics/assets-metadata-css/styling-tips)

## Metadata

What if we wanted to modify the metadata of the page, such as the <title> HTML tag?

<title> is part of the <head> HTML tag, so let's dive into how we can modify the <head> tag in a Next.js page.

Open pages/index.js in your editor and find the following lines:

<Head>

<title>Create Next App</title>

<link rel="icon" href="/favicon.ico" />

</Head>

Notice that <Head> is used instead of the lowercase <head>. <Head> is a React Component that is built into Next.js. It allows you to modify the <head> of a page.

You can import the Head component from the [next/head](https://nextjs.org/docs/api-reference/next/head" \t "_blank) module.

### Adding Head to first-post.js

We haven't added a <title> to our /posts/first-post route. Let's add one.

Open the pages/posts/first-post.js file and add an import for Head from [next/head](https://nextjs.org/docs/api-reference/next/head" \t "_blank) at the beginning of the file:

import Head from 'next/head'

Then, update the exported FirstPost component to include the Head component. For now, we‘ll add just the title tag:

export default function FirstPost() {

return (

<>

<Head>

<title>First Post</title>

</Head>

<h1>First Post</h1>

<h2>

<Link href="/">

<a>Back to home</a>

</Link>

</h2>

</>

)

}

Try accessing <http://localhost:3000/posts/first-post>. The browser tab should now say “First Post”. By using your browser’s developer tools, you should see that the title tag is added to <head>.

To learn more about the Head component, check out the [API reference for next/head](https://nextjs.org/docs/api-reference/next/head).

<https://nextjs.org/docs/api-reference/next/head>

# next/head

**Examples**

We expose a built-in component for appending elements to the head of the page:

import Head from 'next/head'

function IndexPage() {

return (

<div>

<Head>

<title>My page title</title>

<meta name="viewport" content="initial-scale=1.0, width=device-width" />

</Head>

<p>Hello world!</p>

</div>

)

}

export default IndexPage

To avoid duplicate tags in your head you can use the key property, which will make sure the tag is only rendered once, as in the following example:

import Head from 'next/head'

function IndexPage() {

return (

<div>

<Head>

<title>My page title</title>

<meta property="og:title" content="My page title" key="title" />

</Head>

<Head>

<meta property="og:title" content="My new title" key="title" />

</Head>

<p>Hello world!</p>

</div>

)

}

export default IndexPage

In this case only the second <meta property="og:title" /> is rendered. meta tags with duplicate key attributes are automatically handled.

The contents of head get cleared upon unmounting the component, so make sure each page completely defines what it needs in head, without making assumptions about what other pages added.

title, meta or any other elements (e.g. script) need to be contained as **direct** children of the Head element, or wrapped into maximum one level of <React.Fragment> or arrays—otherwise the tags won't be correctly picked up on client-side navigations.

We recommend using [next/script](https://nextjs.org/docs/basic-features/script) in your component instead of manually creating a <script> in next/head.

If you want to customize the <html> tag, for example to add the lang attribute, you can do so by creating a pages/\_document.js file. Learn more in the [custom Document documentation](https://nextjs.org/docs/advanced-features/custom-document" \t "_blank).

<https://nextjs.org/docs/advanced-features/custom-document>

# Custom Document

A custom Document is commonly used to augment your application's <html> and <body> tags. This is necessary because Next.js pages skip the definition of the surrounding document's markup.

To override the default Document, create the file ./pages/\_document.js and extend the Document class as shown below:

import Document, { Html, Head, Main, NextScript } from 'next/document'

class MyDocument extends Document {

static async getInitialProps(ctx) {

const initialProps = await Document.getInitialProps(ctx)

return { ...initialProps }

}

render() {

return (

<Html>

<Head />

<body>

<Main />

<NextScript />

</body>

</Html>

)

}

}

export default MyDocument

The code above is the default Document added by Next.js. Feel free to remove the getInitialProps or render function from MyDocument if you don't need to change them.

<Html>, <Head />, <Main /> and <NextScript /> are required for the page to be properly rendered.

Custom attributes are allowed as props, like lang:

<Html lang="en">

The <Head /> component used here is not the same one from [next/head](https://nextjs.org/docs/api-reference/next/head). The <Head /> component used here should only be used for any <head> code that is common for all pages. For all other cases, such as <title> tags, we recommend using [next/head](https://nextjs.org/docs/api-reference/next/head) in your pages or components.

The ctx object is equivalent to the one received in [getInitialProps](https://nextjs.org/docs/api-reference/data-fetching/getInitialProps" \l "context-object), with one addition:

* renderPage: Function - a callback that runs the actual React rendering logic (synchronously). It's useful to decorate this function in order to support server-rendering wrappers like Aphrodite's [renderStatic](https://github.com/Khan/aphrodite" \l "server-side-rendering" \t "_blank)

## [Caveats](https://nextjs.org/docs/advanced-features/custom-document#caveats)

* Document is only rendered in the server, event handlers like onClick won't work.
* React components outside of <Main /> will not be initialized by the browser. Do not add application logic here or custom CSS (like styled-jsx). If you need shared components in all your pages (like a menu or a toolbar), take a look at the [App](https://nextjs.org/docs/advanced-features/custom-app) component instead.
* Document's getInitialProps function is not called during client-side transitions, nor when a page is [statically optimized](https://nextjs.org/docs/advanced-features/automatic-static-optimization).
* Document currently does not support Next.js [Data Fetching methods](https://nextjs.org/docs/basic-features/data-fetching) like [getStaticProps](https://nextjs.org/docs/basic-features/data-fetching" \l "getstaticprops-static-generation) or [getServerSideProps](https://nextjs.org/docs/basic-features/data-fetching" \l "getserversideprops-server-side-rendering).

## [Customizing renderPage](https://nextjs.org/docs/advanced-features/custom-document#customizing-renderpage)

It should be noted that the only reason you should be customizing renderPage is for usage with **css-in-js** libraries that need to wrap the application to properly work with server-side rendering.

It takes as argument an options object for further customization:

import Document from 'next/document'

class MyDocument extends Document {

static async getInitialProps(ctx) {

const originalRenderPage = ctx.renderPage

ctx.renderPage = () =>

originalRenderPage({

// useful for wrapping the whole react tree

enhanceApp: (App) => App,

// useful for wrapping in a per-page basis

enhanceComponent: (Component) => Component,

})

// Run the parent `getInitialProps`, it now includes the custom `renderPage`

const initialProps = await Document.getInitialProps(ctx)

return initialProps

}

}

export default MyDocument

## [TypeScript](https://nextjs.org/docs/advanced-features/custom-document#typescript)

You can use the built-in DocumentContext type and change the file name to ./pages/\_document.tsx like so:

import Document, { DocumentContext } from 'next/document'

class MyDocument extends Document {

static async getInitialProps(ctx: DocumentContext) {

const initialProps = await Document.getInitialProps(ctx)

return initialProps

}

}

export default MyDocument