**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)

To build a complete web application with React from scratch, there are many important details you need to consider:

* Code has to be bundled using a bundler like webpack and transformed using a compiler like Babel.
* You need to do production optimizations such as code splitting.
* You might want to statically pre-render some pages for performance and SEO. You might also want to use server-side rendering or client-side rendering.
* You might have to write some server-side code to connect your React app to your data store.

A *framework* can solve these problems. But such a framework must have the right level of abstraction — otherwise it won’t be very useful. It also needs to have great "Developer Experience", ensuring you and your team have an amazing experience while writing code.

**Next.js: The React Framework**

Enter **Next.js**, the React Framework. Next.js provides a solution to all of the above problems. But more importantly, it puts you and your team in the [pit of success](https://blog.codinghorror.com/falling-into-the-pit-of-success/) when building React applications.

Next.js aims to have best-in-class developer experience and many built-in features, such as:

* An intuitive [page-based](https://nextjs.org/docs/basic-features/pages) routing system (with support for [dynamic routes](https://nextjs.org/docs/routing/dynamic-routes))
* [Pre-rendering](https://nextjs.org/docs/basic-features/pages#pre-rendering), both [static generation](https://nextjs.org/docs/basic-features/pages#static-generation-recommended) (SSG) and [server-side rendering](https://nextjs.org/docs/basic-features/pages#server-side-rendering) (SSR) are supported on a per-page basis
* Automatic code splitting for faster page loads
* [Client-side routing](https://nextjs.org/docs/routing/introduction#linking-between-pages) with optimized prefetching
* [Built-in CSS](https://nextjs.org/docs/basic-features/built-in-css-support) and [Sass support](https://nextjs.org/docs/basic-features/built-in-css-support#sass-support), and support for any [CSS-in-JS](https://nextjs.org/docs/basic-features/built-in-css-support#css-in-js) library
* Development environment with [Fast Refresh](https://nextjs.org/docs/basic-features/fast-refresh) support
* [API routes](https://nextjs.org/docs/api-routes/introduction) to build API endpoints with Serverless Functions
* Fully extendable

Next.js is used in tens of thousands of production-facing websites and web applications, including many of the world's largest brands.

**About This Tutorial**

This free interactive course will guide you through how to get started with Next.js.

In this tutorial, you’ll learn Next.js basics by creating a very simple **blog app**. Here’s an example of the final result:

[**https://next-learn-starter.vercel.app**](https://next-learn-starter.vercel.app/) ([source](https://github.com/vercel/next-learn/tree/master/basics/demo))

This tutorial assumes basic knowledge of JavaScript and React. If you’ve never written React code, you should go through [the official React tutorial](https://reactjs.org/tutorial/tutorial.html) first.

If you’re looking for documentation instead, [visit the Next.js documentation](https://nextjs.org/docs/getting-started).

**Join the Conversation**

If you have questions about anything related to Next.js or this course, you're welcome to ask our community on [Discord](https://discord.gg/Q3AsD4efFC).

Let’s get started!

**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/). 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/) 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@latest 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), which bootstraps a Next.js app for you. It uses [this template](https://github.com/vercel/next-learn/tree/master/basics/learn-starter) 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).

**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.

**Create a Next.js App**

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

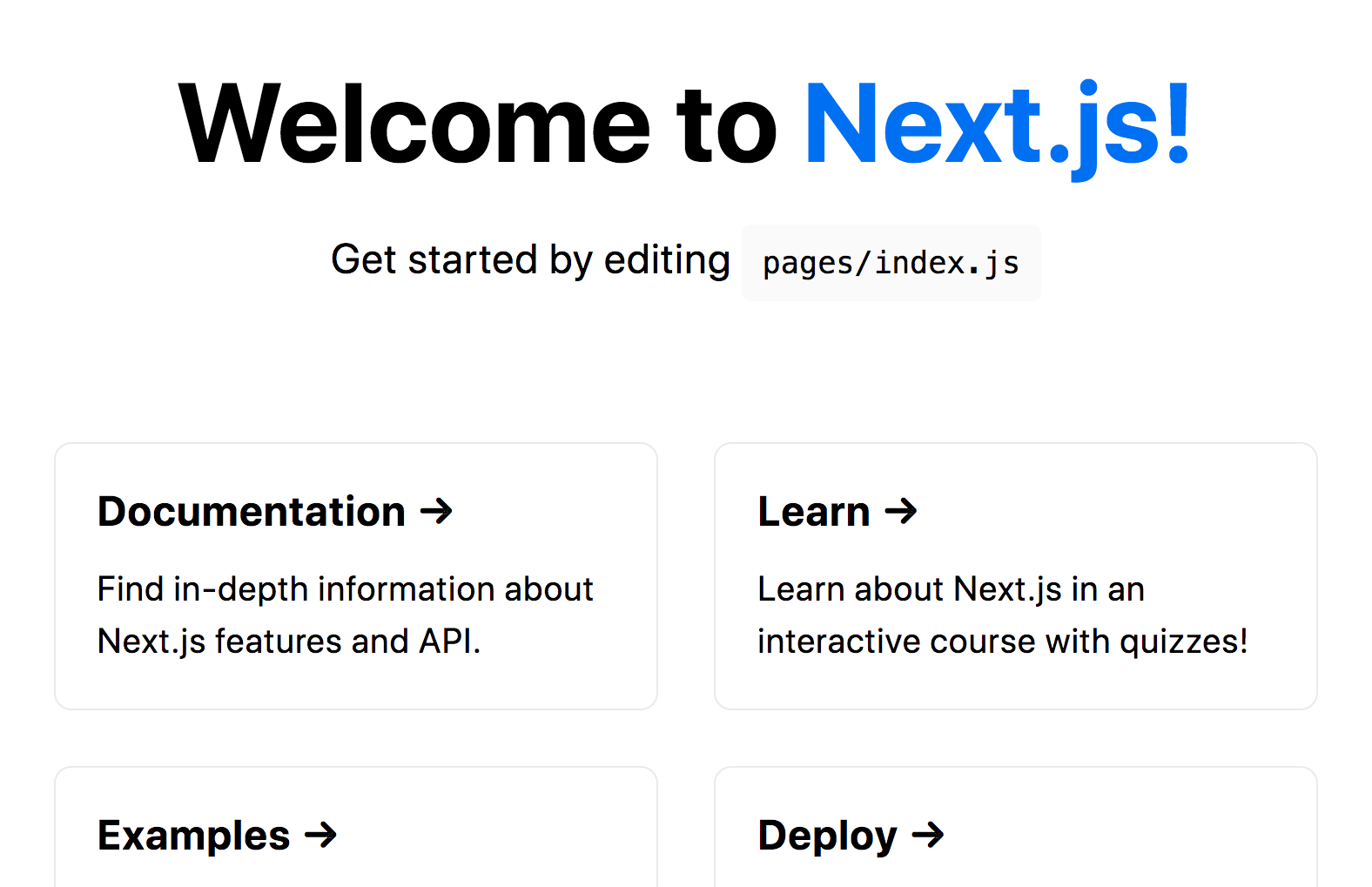
[2](https://nextjs.org/learn/basics/create-nextjs-app/setup)

[**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)

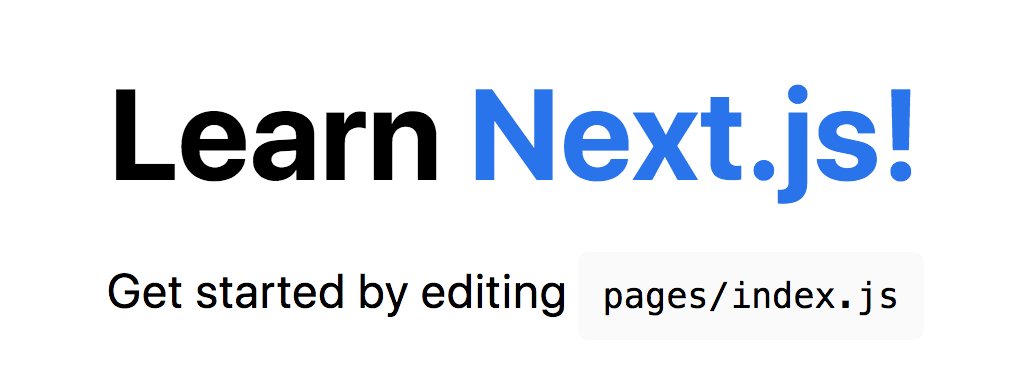
[**4**](https://nextjs.org/learn/basics/create-nextjs-app/editing-the-page)

**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.

**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).
* 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).

**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@latest 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).

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), 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.

**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 can use the Link Component [next/link](https://nextjs.org/docs/api-reference/next/link) to link between pages in your application. <Link> allows you to do client-side navigation and accepts [props](https://nextjs.org/docs/api-reference/next/link#:~:text=Link%20accepts%20the%20following%20props%3A) that give you better control over the navigation behavior.

**Using <Link>**

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

import Link from 'next/link';

Then find the h1 tag that looks like this:

<h1 className="title">

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

</h1>

And change it to:

<h1 className="title">

Read <Link href="/posts/first-post">this page!</Link>

</h1>

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="/">Back to home</Link>

</h2>

</>

);

}

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

**Note:** Before Next.js 12.2, it was required that the Link component wrapped an <a> tag, but [this is not required in versions 12.2 and above](https://nextjs.org/blog/next-12-2#:~:text=next/link%20no%20longer%20requires%20manually%20adding%20%3Ca%3E%20as%20a%20child.%20You%20can%20now%20opt%20into%20this%20behavior%20in%20a%20backward%2Dcompatible%20way.).

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



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

**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**

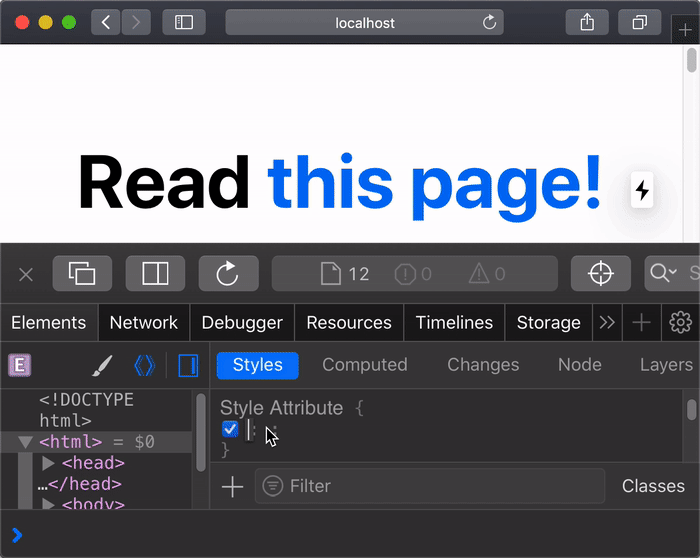
The [Link](https://nextjs.org/docs/api-reference/next/link) component enables **client-side navigation** between two pages in the same Next.js app.

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) 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).

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

**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#sass-support). 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) (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#adding-a-global-stylesheet) in [pages/\_app.js](https://nextjs.org/docs/advanced-features/custom-app).
* 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.

**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).

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

**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). Files inside public can be referenced from the root of the application similar to [pages](https://nextjs.org/docs/basic-features/pages).

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.

**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).
* 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).

**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#webp) 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/), 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).

Here's an example using [next/image](https://nextjs.org/docs/api-reference/next/image) 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).

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

**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)

**Third-Party JavaScript**

**Third-party JavaScript** refers to any scripts that are added from a third-party source. Usually, third-party scripts are included in order to introduce newer functionality into a site that does not need to be written from scratch, such as analytics, ads, and customer support widgets.

**Adding Third-Party JavaScript**

Let's dive into how we can add a third-party script to a Next.js page.

Open pages/posts/first-post.js in your editor and find the following lines:

<Head>

<title>First Post</title>

</Head>

In addition to metadata, scripts that need to load and execute as soon as possible are usually added within the <head> of a page. Using a regular HTML <script> element, an external script would be added as follows:

<Head>

<title>First Post</title>

<script src="https://connect.facebook.net/en\_US/sdk.js" />

</Head>

This script contains the [Facebook SDK](https://developers.facebook.com/docs/javascript/quickstart) which is commonly used to introduce Facebook social plugins and other functionality. Although this approach works, including scripts in this manner does not give a clear idea of when it would load with respect to the other JavaScript code fetched on the same page. If a particular script is render-blocking and can delay page content from loading, this can significantly impact performance.

**Using the Script Component**

[next/script](https://nextjs.org/docs/api-reference/next/script) is an extension of the HTML <script> element and optimizes when additional scripts are fetched and executed.

In the same file, add an import for Script from next/script at the beginning of the file:

import Script from 'next/script';

Now, update the FirstPost component to include the Script component:

export default function FirstPost() {

return (

<>

<Head>

<title>First Post</title>

</Head>

<Script

src="https://connect.facebook.net/en\_US/sdk.js"

strategy="lazyOnload"

onLoad={() =>

console.log(`script loaded correctly, window.FB has been populated`)

}

/>

<h1>First Post</h1>

<h2>

<Link href="/">← Back to home</Link>

</h2>

</>

);

}

Notice that a few additional properties have been defined in the Script component:

* strategy controls when the third-party script should load. A value of lazyOnload tells Next.js to load this particular script lazily during browser idle time
* onLoad is used to run any JavaScript code immediately after the script has finished loading. In this example, we log a message to the console that mentions that the script has loaded correctly

Try accessing <http://localhost:3000/posts/first-post>. By using your browser’s developer tools, you should see the message above logged in the Console panel. In addition, you can run window.FB to see that the script has populated this global variable.

**Note:** The Facebook SDK was only used as a contrived example to show how to add third-party scripts to your application in a performant way. Now that you understand the basics of including third-party functionality in Next.js, you can remove the Script component from FirstPost before proceeding.

To learn more about the Script component, check out the [documentation](https://nextjs.org/docs/basic-features/script).

**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)

**CSS Styling**

Let’s now talk about **CSS styling**.

As you can see, our index page ([http://localhost:3000](http://localhost:3000/)) already has some styles. If you look at your file structure, you'll see a folder called styles with two CSS files: a global stylesheet (globals.css), and a CSS module (Home.module.css).

If your project doesn't have those files, you can download the starter code here:

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

**CSS Modules**

[CSS modules](https://nextjs.org/docs/basic-features/built-in-css-support) allow you to locally scope CSS at the component-level by automatically creating unique class names. This allows you to use the same CSS class name in different files without worrying about class name collisions.

In addition to CSS modules, you can style your Next.js application in a variety of ways, including:

* Sass which allows you to import .css and .scss files.
* PostCSS libraries like [Tailwind CSS](https://github.com/vercel/next.js/tree/canary/examples/with-tailwindcss).
* CSS-in-JS libraries such as [styled-jsx](https://github.com/vercel/styled-jsx), [styled-components](https://github.com/vercel/next.js/tree/canary/examples/with-styled-components), and [emotion](https://github.com/vercel/next.js/tree/canary/examples/with-emotion)

In this lesson, we’ll talk about how to use [CSS Modules](https://nextjs.org/docs/basic-features/built-in-css-support#adding-component-level-css) and [Sass](https://nextjs.org/docs/basic-features/built-in-css-support#sass-support) in Next.js. Let’s dive in!

**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)

**Layout Component**

First, let’s create a **Layout** component which will be shared across all pages.

* Create a top-level directory called components.
* Inside components, create a file called layout.js with the following content:

export default function Layout({ children }) {

return <div>{children}</div>;

}

Then, open pages/posts/first-post.js, add an import for the Layout component, and make it the outermost component:

import Head from 'next/head';

import Link from 'next/link';

import Layout from '../../components/layout';

export default function FirstPost() {

return (

<Layout>

<Head>

<title>First Post</title>

</Head>

<h1>First Post</h1>

<h2>

<Link href="/">← Back to home</Link>

</h2>

</Layout>

);

}

**Adding CSS**

Now, let’s add some styles to the Layout component. To do so, we’ll use [CSS Modules](https://nextjs.org/docs/basic-features/built-in-css-support#adding-component-level-css), which lets you import CSS files in a React component.

Create a file called components/layout.module.css with the following content:

.container {

max-width: 36rem;

padding: 0 1rem;

margin: 3rem auto 6rem;

}

**Important:** To use [CSS Modules](https://nextjs.org/docs/basic-features/built-in-css-support#adding-component-level-css), the CSS file name must end with .module.css.

To use this container class inside components/layout.js, you need to:

* Import the CSS file and assign a name to it, like styles
* Use styles.container as the className

Open components/layout.js and replace its content with the following:

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

export default function Layout({ children }) {

return <div className={styles.container}>{children}</div>;

}

If you go to <http://localhost:3000/posts/first-post> now, you should see that the text is now inside a centered container:



**Automatically Generates Unique Class Names**

Now, if you take a look at the HTML in your browser’s devtools, you’ll notice that the div rendered by the Layout component has a class name that looks like layout\_container\_\_...:



This is what [CSS Modules](https://nextjs.org/docs/basic-features/built-in-css-support#adding-component-level-css) does: *It automatically generates unique class names*. As long as you use CSS Modules, you don’t have to worry about class name collisions.

Furthermore, Next.js’s code splitting feature works on [CSS Modules](https://nextjs.org/docs/basic-features/built-in-css-support#adding-component-level-css) as well. It ensures the minimal amount of CSS is loaded for each page. This results in smaller bundle sizes.

[CSS Modules](https://nextjs.org/docs/basic-features/built-in-css-support#adding-component-level-css) are extracted from the JavaScript bundles at build time and generate .css files that are loaded automatically by Next.js.

**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)

**Global Styles**

[CSS Modules](https://nextjs.org/docs/basic-features/built-in-css-support#adding-component-level-css) are useful for component-level styles. But if you want some CSS to be loaded by **every page**, Next.js has support for that as well.

To load [global CSS](https://nextjs.org/docs/basic-features/built-in-css-support#adding-a-global-stylesheet) to your application, create a file called pages/\_app.js with the following content:

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

return <Component {...pageProps} />;

}

The default export of \_app.js is a top-level React component that wraps all the pages in your application. You can use this component to keep state when navigating between pages, or to add global styles as we're doing here. [Learn more about \_app.js file](https://nextjs.org/docs/advanced-features/custom-app).

**Restart the Development Server**

**Important:** You need to restart the development server when you add [pages/\_app.js](https://nextjs.org/docs/advanced-features/custom-app). Press Ctrl + c to stop the server and run:

npm run dev

**Adding Global CSS**

In Next.js, you can add [global CSS](https://nextjs.org/docs/basic-features/built-in-css-support#adding-a-global-stylesheet) files by importing them from [pages/\_app.js](https://nextjs.org/docs/advanced-features/custom-app). You **cannot** import global CSS anywhere else.

The reason that [global CSS](https://nextjs.org/docs/basic-features/built-in-css-support#adding-a-global-stylesheet) can't be imported outside of pages/\_app.js is that global CSS affects all elements on the page.

If you were to navigate from the homepage to the /posts/first-post page, global styles from the homepage would affect /posts/first-post unintentionally.

You can place the global CSS file anywhere and use any name. So let’s do the following:

* Create a top-level styles directory and a global.css file.
* Add the following CSS inside styles/global.css. This code resets some styles and changes the color of the a tag:

html,

body {

padding: 0;

margin: 0;

font-family: -apple-system, BlinkMacSystemFont, Segoe UI, Roboto, Oxygen, Ubuntu,

Cantarell, Fira Sans, Droid Sans, Helvetica Neue, sans-serif;

line-height: 1.6;

font-size: 18px;

}

\* {

box-sizing: border-box;

}

a {

color: #0070f3;

text-decoration: none;

}

a:hover {

text-decoration: underline;

}

img {

max-width: 100%;

display: block;

}

Finally, import the CSS file inside the pages/\_app.js file you've created earlier on:

// `pages/\_app.js`

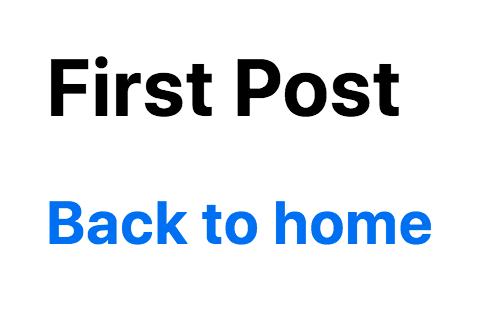
import '../styles/global.css';

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

return <Component {...pageProps} />;

}

Now, if you access <http://localhost:3000/posts/first-post>, you’ll see that the styles are applied. Any styles imported in \_app.js will be applied globally, to all pages of the application.



**If it didn’t work**: Make sure you restart the development server when you update pages/\_app.js.

**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)

**Polishing Layout**

So far, we’ve only added minimal React and CSS code just to illustrate concepts such as [CSS Modules](https://nextjs.org/docs/basic-features/built-in-css-support#adding-component-level-css). Before we move on to our next lesson about [data fetching](https://nextjs.org/docs/basic-features/data-fetching), let’s polish our page styling and code.

**Update components/layout.module.css**

First, open components/layout.module.css and replace its content with the following more polished styles for the layout and profile picture:

.container {

max-width: 36rem;

padding: 0 1rem;

margin: 3rem auto 6rem;

}

.header {

display: flex;

flex-direction: column;

align-items: center;

}

.backToHome {

margin: 3rem 0 0;

}

**Create styles/utils.module.css**

Second, let’s create a set of CSS utility classes (for text styles) that can be re-used across multiple components.

Add a new CSS file called styles/utils.module.css with the following content:

.heading2Xl {

font-size: 2.5rem;

line-height: 1.2;

font-weight: 800;

letter-spacing: -0.05rem;

margin: 1rem 0;

}

.headingXl {

font-size: 2rem;

line-height: 1.3;

font-weight: 800;

letter-spacing: -0.05rem;

margin: 1rem 0;

}

.headingLg {

font-size: 1.5rem;

line-height: 1.4;

margin: 1rem 0;

}

.headingMd {

font-size: 1.2rem;

line-height: 1.5;

}

.borderCircle {

border-radius: 9999px;

}

.colorInherit {

color: inherit;

}

.padding1px {

padding-top: 1px;

}

.list {

list-style: none;

padding: 0;

margin: 0;

}

.listItem {

margin: 0 0 1.25rem;

}

.lightText {

color: #666;

}

You can reuse these utility classes throughout your application, and you may even use utility classes in your global.css file. Utility classes refer to an approach of writing CSS selectors rather than a method (e.g. global styles, CSS modules, Sass, etc). Learn more about [utility-first CSS](https://tailwindcss.com/docs/utility-first).

**Update components/layout.js**

Third, open components/layout.js and replace its content with the following code, **changing** Your Name to an actual name:

import Head from 'next/head';

import Image from 'next/image';

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

import utilStyles from '../styles/utils.module.css';

import Link from 'next/link';

const name = 'Your Name';

export const siteTitle = 'Next.js Sample Website';

export default function Layout({ children, home }) {

return (

<div className={styles.container}>

<Head>

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

<meta

name="description"

content="Learn how to build a personal website using Next.js"

/>

<meta

property="og:image"

content={`https://og-image.vercel.app/${encodeURI(

siteTitle,

)}.png?theme=light&md=0&fontSize=75px&images=https%3A%2F%2Fassets.vercel.com%2Fimage%2Fupload%2Ffront%2Fassets%2Fdesign%2Fnextjs-black-logo.svg`}

/>

<meta name="og:title" content={siteTitle} />

<meta name="twitter:card" content="summary\_large\_image" />

</Head>

<header className={styles.header}>

{home ? (

<>

<Image

priority

src="/images/profile.jpg"

className={utilStyles.borderCircle}

height={144}

width={144}

alt=""

/>

<h1 className={utilStyles.heading2Xl}>{name}</h1>

</>

) : (

<>

<Link href="/">

<Image

priority

src="/images/profile.jpg"

className={utilStyles.borderCircle}

height={108}

width={108}

alt=""

/>

</Link>

<h2 className={utilStyles.headingLg}>

<Link href="/" className={utilStyles.colorInherit}>

{name}

</Link>

</h2>

</>

)}

</header>

<main>{children}</main>

{!home && (

<div className={styles.backToHome}>

<Link href="/">← Back to home</Link>

</div>

)}

</div>

);

}

Here’s what’s new:

* [meta tags](https://en.wikipedia.org/wiki/Meta_element) (like og:image), which are used to describe a page's content
* Boolean home prop which will adjust the size of the title and the image
* “Back to home” link at the bottom if home is false
* Added images with next/image, which are preloaded with the [priority](https://nextjs.org/docs/api-reference/next/image#priority) attribute

**Update pages/index.js**

Finally, let's update the homepage.

Open pages/index.js and replace its content with:

import Head from 'next/head';

import Layout, { siteTitle } from '../components/layout';

import utilStyles from '../styles/utils.module.css';

export default function Home() {

return (

<Layout home>

<Head>

<title>{siteTitle}</title>

</Head>

<section className={utilStyles.headingMd}>

<p>[Your Self Introduction]</p>

<p>

(This is a sample website - you’ll be building a site like this on{' '}

<a href="https://nextjs.org/learn">our Next.js tutorial</a>.)

</p>

</section>

</Layout>

);

}

Then replace [Your Self Introduction] with your self-introduction. Here’s an example with the author’s profile:



That’s it! We now have the polished layout code and we're ready to move on to our data fetching lessons.

Before we wrap up this lesson, let’s talk about some helpful techniques related to Next.js’s CSS support on the next page.

**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)

**Styling Tips**

Here are some styling tips that might be helpful.

You can just **read through** the following sections. No need to make changes to our app!

**Using clsx library to toggle classes**

[clsx](https://www.npmjs.com/package/clsx) is a simple library that lets you toggle class names easily. You can install it using npm install clsx or yarn add clsx.

Please take a look at its [documentation](https://github.com/lukeed/clsx) for more details, but here's the basic usage:

* Suppose that you want to create an Alert component which accepts type, which can be 'success' or 'error'.
* If it's 'success', you want the text color to be green. If it's 'error', you want the text color to be red.

You can first write a CSS module (e.g. alert.module.css) like this:

.success {

color: green;

}

.error {

color: red;

}

And use clsx like this:

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

import { clsx } from 'clsx';

export default function Alert({ children, type }) {

return (

<div

className={clsx({

[styles.success]: type === 'success',

[styles.error]: type === 'error',

})}

>

{children}

</div>

);

}

**Customizing PostCSS Config**

Out of the box, with no configuration, Next.js compiles CSS using [PostCSS](https://postcss.org/).

To customize PostCSS config, you can create a top-level file called [postcss.config.js](https://nextjs.org/docs/advanced-features/customizing-postcss-config#customizing-plugins). This is useful if you're using libraries like [Tailwind CSS](https://tailwindcss.com/).

Here are the steps to add [Tailwind CSS](https://tailwindcss.com/). First, install the packages:

npm install -D tailwindcss autoprefixer postcss

Then, create a [postcss.config.js](https://nextjs.org/docs/advanced-features/customizing-postcss-config#customizing-plugins):

// postcss.config.js

module.exports = {

plugins: {

tailwindcss: {},

autoprefixer: {},

},

};

We also recommend [configuring content sources](https://tailwindcss.com/docs/content-configuration) by specifying the content option on tailwind.config.js:

// tailwind.config.js

module.exports = {

content: [

'./pages/\*\*/\*.{js,ts,jsx,tsx}',

'./components/\*\*/\*.{js,ts,jsx,tsx}',

// For the best performance and to avoid false positives,

// be as specific as possible with your content configuration.

],

};

To learn more about custom PostCSS configuration, check out the [documentation for PostCSS](https://nextjs.org/docs/advanced-features/customizing-postcss-config).

To easily get started with Tailwind CSS, [check out our example](https://github.com/vercel/next.js/tree/canary/examples/with-tailwindcss).

**Using Sass**

Out of the box, Next.js allows you to import [Sass](https://nextjs.org/docs/basic-features/built-in-css-support#sass-support) using both the .scss and .sass extensions. You can use component-level Sass via [CSS Modules](https://nextjs.org/docs/basic-features/built-in-css-support#adding-component-level-css) 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):

npm install -D sass

**That's it for this lesson!**

To learn more about Next.js's built-in CSS Support and CSS Modules, check out the [CSS Documentation](https://nextjs.org/docs/basic-features/built-in-css-support).