Built-In CSS Support

Examples

Basic CSS Example

With Tailwind CSS

Next.js allows you to import CSS files from a JavaScript file. This is possible because Next.js extends the concept of `import` beyond JavaScript.

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`:

```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 if not already present. Then, `import` the `styles.css` file.

```js
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`**.

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`

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

Next.js supports 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:

```css
/*
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:

```jsx
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

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`:

```bash
npm install --save-dev sass
```

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

> **Note**: Sass supports two different syntaxes, each with their own extension. The `.scss` extension requires you use the SCSS syntax, while the `.sass` extension requires you use the Indented Syntax ("Sass").
>
> 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

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

For example to add `includePaths`:

```js
const path = require('path')

module.exports = {
  sassOptions: {
    includePaths: [path.join(__dirname, 'styles')],
  },
}
```

Sass Variables

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

For example, using the exported `primaryColor` Sass variable:

```scss
/* variables.module.scss */
$primary-color: #64FF00

:export {
  primaryColor: $primary-color
}
```

```jsx
// 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

Examples

Styled JSX

Styled Components

Emotion

Linaria

Tailwind CSS + Emotion

Styletron

Cxs

Aphrodite

Fela

Stitches

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 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.

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 for more examples.

FAQ

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.

Related

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

Customizing PostCSS Config: Extend the PostCSS config and plugins added by Next.js with your own.