CSS Modules

Examples

- Basic CSS Example ↗

Next.js has built-in support for CSS Modules using the `.module.css` extension.

CSS Modules locally scope CSS by automatically creating a unique class name. This allows you to use the same class name in different files without worrying about collisions. This behavior makes CSS Modules the ideal way to include component-level CSS.

Example

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

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

Button.module.css

```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:

components/Button.js

```
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.

Global Styles

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

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

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

pages/_app.js

```
import '../styles.css';
```

```
2
3  // This default export is required in a new `pages/_app.js` file.
4  export default function MyApp({ Component, pageProps }) {
5    return <Component {...pageProps} />;
6  }
```

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.

External Stylesheets

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.

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

```
1  import 'bootstrap/dist/css/bootstrap.css';
2
3  export default function MyApp({ Component, pageProps }) {
4    return <Component {...pageProps} />;
5  }
```

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

components/example-dialog.js

```
1  import { useState } from 'react';
2  import { Dialog } from '@reach/dialog';
3  import VisuallyHidden from '@reach/visually-hidden';
4  import '@reach/dialog/styles.css';
```

```
5
6  function ExampleDialog(props) {
7    const [showDialog, setShowDialog] = useState(false);
8    const open = () => setShowDialog(true);
9    const close = () => setShowDialog(false);
10
11   return (
12     <div>
13       <button onClick={open}>Open Dialog</button>
14       <Dialog isOpen={showDialog} onDismiss={close}>
15         <button className="close-button" onClick={close}>
16           <VisuallyHidden>Close</VisuallyHidden>
17           <span aria-hidden>×</span>
18         </button>
19         <p>Hello there. I am a dialog</p>
20       </Dialog>
21     </div>
22   );
23 }
```

Additional Features

Next.js includes additional features to improve the authoring experience of adding styles:

- When running locally with `next dev`, local stylesheets (either global or CSS modules) will take advantage of Fast Refresh to instantly reflect changes as edits are saved.
- When building for production with `next build`, CSS files will be bundled into fewer minified `.css` files to reduce the number of network requests needed to retrieve styles.
- If you disable JavaScript, styles will still be loaded in the production build (`next start`). However, JavaScript is still required for `next dev` to enable Fast Refresh.