> *Note: this feature is available with `react-scripts@2.0.0` and higher.*

Generally, we recommend that you don't reuse the same CSS classes across different components. For example, instead of using a `.Button` CSS class in `<AcceptButton>` and `<RejectButton>` components, we recommend creating a `<Button>` component with its own `.Button` styles, that both `<AcceptButton>` and `<RejectButton>` can render (but not inherit).

Following this rule often makes CSS preprocessors less useful, as features like mixins and nesting are replaced by component composition. You can, however, integrate a CSS preprocessor if you find it valuable.

To use Sass, first install `sass`:

```
$ npm install sass
# or
$ yarn add sass
```

Now you can rename `src/App.css` to `src/App.scss` and update `src/App.js` to import `src/App.scss`. This file and any other file will be automatically compiled if imported with the extension `.scss` or `.sass`.

To share variables between Sass files, you can use Sass's `@use` rule. For example, `src/App.scss` and other component style files could include `@use "./shared.scss";` with variable definitions.

This will allow you to do imports like

```
@use 'styles/_colors.scss'; // assuming a styles directory under src/
@use '~nprogress/nprogress'; // loading a css file from the nprogress node module
```

> ***Note:** You can prefix paths with `~`, as displayed above, to resolve modules from `node_modules`.*

`sass` also supports the `SASS_PATH` variable.

To use imports relative to a path you specify, you can add a `.env` file at the project root with the path specified in the `SASS_PATH` environment variable. To specify more directories you can add them to `SASS_PATH` separated by a `:` like `path1:path2:path3`.

> ***Note:** For the Windows operating system, separate your paths by semicolons.*
>
> ```
> SASS_PATH=path1;path2;path3
> ```

> ***Tip:** You can opt into using this feature with CSS modules too!*

> ***Note:** If you're using Flow, override the module.file_ext setting in your `.flowconfig` so it'll recognize `.sass` or `.scss` files. You will also need to include the `module.file_ext` default settings for `.js`, `.jsx`, `.mjs` and `.json` files.*
>
> ```
> [options]
> module.file_ext=.js
> module.file_ext=.jsx
> module.file_ext=.mjs
> module.file_ext=.json
> module.file_ext=.sass
> module.file_ext=.scss
> ```

> ***Note:*** *LibSass and the packages built on top of it, including Node Sass, are deprecated. If you're a user of Node Sass, you can migrate to Dart Sass by replacing* `node-sass` *in your* `package.json` *file with* `sass` *or by running the following commands:*
>
> ```
> $ npm uninstall node-sass
> $ npm install sass
> # or
> $ yarn remove node-sass
> $ yarn add sass
> ```