# Style Library Interoperability

While you can use the emotion based styling solution provided by MUI to style your application, you can also use the one you already know and love (from plain CSS to styled-components).

This guide aims to document the most popular alternatives, but you should find that the principles applied here can be adapted to other libraries. There are examples for the following styling solutions:

- Plain CSS
- Global CSS
- Styled Components
- CSS Modules
- Emotion
- Tailwind CSS
- ~~JSS~~ TSS

## Plain CSS

Nothing fancy, just plain CSS.

{{"demo": "StyledComponents.js", "hideToolbar": true}}

Edit on **CodeSandbox**

**PlainCssSlider.css**

```css
.slider {
  color: #20b2aa;
}

.slider:hover {
  color: #2e8b57;
}
```

**PlainCssSlider.js**

```js
import * as React from 'react';
import Slider from '@mui/material/Slider';
import './PlainCssSlider.css';

export default function PlainCssSlider() {
  return (
    <div>
      <Slider defaultValue={30} />
      <Slider defaultValue={30} className="slider" />
    </div>
  );
}
```

### CSS injection order ⚠️

**Note:** Most CSS-in-JS solutions inject their styles at the bottom of the HTML `<head>`, which gives MUI precedence over your custom styles. To remove the need for **!important**, you need to change the CSS injection order. Here's a demo of how it can be done in MUI:

```
import * as React from 'react';
import { StyledEngineProvider } from '@mui/material/styles';

export default function GlobalCssPriority() {
  return (
    <StyledEngineProvider injectFirst>
      {/* Your component tree. Now you can override MUI's styles. */}
    </StyledEngineProvider>
  );
}
```

**Note:** If you are using emotion and have a custom cache in your app, that one will override the one coming from MUI. In order for the injection order to still be correct, you need to add the prepend option. Here is an example:

```
import * as React from 'react';
import { CacheProvider } from '@emotion/react';
import createCache from '@emotion/cache';

const cache = createCache({
  key: 'css',
  prepend: true,
});

export default function PlainCssPriority() {
  return (
    <CacheProvider value={cache}>
      {/* Your component tree. Now you can override MUI's styles. */}
    </CacheProvider>
  );
}
```

**Note:** If you are using styled-components and have `StyleSheetManager` with a custom `target`, make sure that the target is the first element in the HTML `<head>`. If you are curious to see how it can be done, you can take a look on the `StyledEngineProvider` implementation in the `@mui/styled-engine-sc` package.

### Deeper elements

If you attempt to style the Slider, you will likely need to affect some of the Slider's child elements, for example the thumb. In MUI, all child elements have an increased specificity of 2: `.parent .child {}`. When writing overrides, you need to do the same.

The following examples override the slider's `thumb` style in addition to the custom styles on the slider itself.

{{"demo": "StyledComponentsDeep.js", "hideToolbar": true}}

**PlainCssSliderDeep1.css**

```css
.slider {
  color: #20b2aa;
}

.slider:hover {
  color: #2e8b57;
}

.slider .MuiSlider-thumb {
  border-radius: 1px;
}
```

**PlainCssSliderDeep1.js**

```js
import * as React from 'react';
import Slider from '@mui/material/Slider';
import './PlainCssSliderDeep1.css';

export default function PlainCssSliderDeep1() {
  return (
    <div>
      <Slider defaultValue={30} />
      <Slider defaultValue={30} className="slider" />
    </div>
  );
}
```

The above demo relies on the default `className` values, but you can provide your own class name with the `componentsProps` API.

**PlainCssSliderDeep2.css**

```css
.slider {
  color: #20b2aa;
}

.slider:hover {
  color: #2e8b57;
}

.slider .thumb {
  border-radius: 1px;
}
```

**PlainCssSliderDeep2.js**

```js
import * as React from 'react';
import Slider from '@mui/material/Slider';
import './PlainCssSliderDeep2.css';

export default function PlainCssSliderDeep2() {
  return (
    <div>
      <Slider defaultValue={30} />
      <Slider
        defaultValue={30}
        className="slider"
        componentsProps={{ thumb: { className: 'thumb' } }}
      />
    </div>
  );
}
```

## Global CSS

Explicitly providing the class names to the component is too much effort? You can target the class names generated by MUI.

Edit on CodeSandbox

**GlobalCssSlider.css**

```css
.MuiSlider-root {
  color: #20b2aa;
}

.MuiSlider-root:hover {
  color: #2e8b57;
}
```

**GlobalCssSlider.js**

```js
import * as React from 'react';
import Slider from '@mui/material/Slider';
import './GlobalCssSlider.css';

export default function GlobalCssSlider() {
  return <Slider defaultValue={30} />;
}
```

### CSS injection order ⚠️

**Note:** Most CSS-in-JS solutions inject their styles at the bottom of the HTML `<head>`, which gives MUI precedence over your custom styles. To remove the need for **!important**, you need to change the CSS injection order. Here's a demo of how it can be done in MUI:

```
import * as React from 'react';
import { StyledEngineProvider } from '@mui/material/styles';

export default function GlobalCssPriority() {
  return (
    <StyledEngineProvider injectFirst>
      {/* Your component tree. Now you can override MUI's styles. */}
    </StyledEngineProvider>
  );
}
```

**Note:** If you are using emotion and have a custom cache in your app, that one will override the one coming from MUI. In order for the injection order to still be correct, you need to add the prepend option. Here is an example:

```
import * as React from 'react';
import { CacheProvider } from '@emotion/react';
import createCache from '@emotion/cache';

const cache = createCache({
  key: 'css',
  prepend: true,
});

export default function GlobalCssPriority() {
  return (
    <CacheProvider value={cache}>
      {/* Your component tree. Now you can override MUI's styles. */}
    </CacheProvider>
  );
}
```

**Note:** If you are using styled-components and have `StyleSheetManager` with a custom `target`, make sure that the target is the first element in the HTML `<head>`. If you are curious to see how it can be done, you can take a look on the `StyledEngineProvider` implementation in the `@mui/styled-engine-sc` package.

## Deeper elements

If you attempt to style the Slider, you will likely need to affect some of the Slider's child elements, for example the thumb. In MUI, all child elements have an increased specificity of 2: `.parent .child {}`. When writing overrides, you need to do the same.

The following example overrides the slider's `thumb` style in addition to the custom styles on the slider itself.

{{"demo": "StyledComponentsDeep.js", "hideToolbar": true}}

**GlobalCssSliderDeep.css**

```
.MuiSlider-root {
  color: #20b2aa;
}
```

```css
.MuiSlider-root:hover {
  color: #2e8b57;
}

.MuiSlider-root .MuiSlider-thumb {
  border-radius: 1px;
}
```

**GlobalCssSliderDeep.js**

```jsx
import * as React from 'react';
import Slider from '@mui/material/Slider';
import './GlobalCssSliderDeep.css';

export default function GlobalCssSliderDeep() {
  return <Slider defaultValue={30} />;
}
```

## Styled Components

Star 37k downloads 19M/month

### Change the default styled engine

By default, MUI components come with emotion as their style engine. If, however, you would like to use `styled-components`, you can configure your app by following the styled engine guide or starting with one of the example projects:

- Create React App with styled-components
- Create React App with styled-components and typescript

Following this approach reduces the bundle size, and removes the need to configure the CSS injection order.

After the style engine is configured properly, you can use the `styled()` utility from `@mui/material/styles` and have direct access to the theme.

{{"demo": "StyledComponents.js", "hideToolbar": true}}

Edit on CodeSandbox

```jsx
import * as React from 'react';
import Slider from '@mui/material/Slider';
import { styled } from '@mui/material/styles';

const CustomizedSlider = styled(Slider)`
  color: #20b2aa;

  :hover {
    color: #2e8b57;
  }
`;

export default function StyledComponents() {
  return <CustomizedSlider defaultValue={30} />;
}
```

### Deeper elements

If you attempt to style the Slider, you will likely need to affect some of the Slider's child elements, for example the thumb. In MUI, all child elements have an increased specificity of 2: `.parent .child {}`. When writing overrides, you need to do the same.

The following examples override the slider's `thumb` style in addition to the custom styles on the slider itself.

{{"demo": "StyledComponentsDeep.js", "defaultCodeOpen": true}}

The above demo relies on the [default `className` values](), but you can provide your own class name with the `componentsProps` API.

```
import * as React from 'react';
import { styled } from '@mui/material/styles';
import Slider from '@mui/material/Slider';

const CustomizedSlider = styled((props) => (
  <Slider componentsProps={{ thumb: { className: 'thumb' } }} {...props} />
))`
  color: #20b2aa;

  :hover {
    color: #2e8b57;
  }

  & .thumb {
    border-radius: 1px;
  }
`;

export default function StyledComponentsDeep2() {
  return (
    <div>
      <Slider defaultValue={30} />
      <CustomizedSlider defaultValue={30} />
    </div>
  );
}
```

### Theme

By using the MUI theme provider, the theme will be available in the theme context of the styled engine too (emotion or styled-components, depending on your configuration).

> ⚠️ *If you are **already** using a custom theme with styled-components or emotion, it might not be compatible with MUI's theme specification. If it's not compatible, you need to render MUI's ThemeProvider **first**. This will ensure the theme structures are isolated. This is ideal for the progressive adoption of MUI's components in the codebase.*

You are encouraged to share the same theme object between MUI and the rest of your project.

```
const CustomizedSlider = styled(Slider)(
  ({ theme }) => `
  color: ${theme.palette.primary.main};

  :hover {
    color: ${darken(theme.palette.primary.main, 0.2)};
  }
`,
);
```

{{"demo": "StyledComponentsTheme.js"}}

### Portals

The Portal provides a first-class way to render children into a DOM node that exists outside the DOM hierarchy of the parent component. Because of the way styled-components scopes its CSS, you may run into issues where styling is not applied.

For example, if you attempt to style the `tooltip` generated by the Tooltip component, you will need to pass along the `className` property to the element being rendered outside of it's DOM hierarchy. The following example shows a workaround:

```
import * as React from 'react';
import { styled } from '@mui/material/styles';
import Button from '@mui/material/Button';
import Tooltip from '@mui/material/Tooltip';

const StyledTooltip = styled(({ className, ...props }) => (
  <Tooltip {...props} classes={{ popper: className }} />
))`
  & .MuiTooltip-tooltip {
    background: navy;
  }
`;
```

{{"demo": "StyledComponentsPortal.js"}}

## CSS Modules

Star 16k

It's hard to know the market share of this styling solution as it's dependent on the bundling solution people are using.

{{"demo": "StyledComponents.js", "hideToolbar": true}}

Edit on CodeSandbox

**CssModulesSlider.module.css**

```css
.slider {
  color: #20b2aa;
}

.slider:hover {
  color: #2e8b57;
}
```

**CssModulesSlider.js**

```jsx
import * as React from 'react';
import Slider from '@mui/material/Slider';
// webpack, parcel or else will inject the CSS into the page
import styles from './CssModulesSlider.module.css';

export default function CssModulesSlider() {
  return (
    <div>
      <Slider defaultValue={30} />
      <Slider defaultValue={30} className={styles.slider} />
    </div>
  );
}
```

### CSS injection order ⚠️

**Note:** Most CSS-in-JS solutions inject their styles at the bottom of the HTML `<head>`, which gives MUI precedence over your custom styles. To remove the need for **!important**, you need to change the CSS injection order. Here's a demo of how it can be done in MUI:

```jsx
import * as React from 'react';
import { StyledEngineProvider } from '@mui/material/styles';

export default function GlobalCssPriority() {
  return (
    <StyledEngineProvider injectFirst>
      {/* Your component tree. Now you can override MUI's styles. */}
    </StyledEngineProvider>
  );
}
```

**Note:** If you are using emotion and have a custom cache in your app, that one will override the one coming from MUI. In order for the injection order to still be correct, you need to add the prepend option. Here is an example:

```
import * as React from 'react';
import { CacheProvider } from '@emotion/react';
import createCache from '@emotion/cache';

const cache = createCache({
  key: 'css',
  prepend: true,
});

export default function CssModulesPriority() {
  return (
    <CacheProvider value={cache}>
      {/* Your component tree. Now you can override MUI's styles. */}
    </CacheProvider>
  );
}
```

**Note:** If you are using styled-components and have `StyleSheetManager` with a custom `target`, make sure that the target is the first element in the HTML `<head>`. If you are curious to see how it can be done, you can take a look on the `StyledEngineProvider` implementation in the `@mui/styled-engine-sc` package.

## Deeper elements

If you attempt to style the Slider, you will likely need to affect some of the Slider's child elements, for example the thumb. In MUI, all child elements have an increased specificity of 2: `.parent .child {}`. When writing overrides, you need to do the same. It's important to keep in mind that CSS Modules adds an unique id to each class, and that id won't be present on the MUI provided children class. To escape from that, CSS Modules provides a functionality, the `:global` selector.

The following examples override the slider's `thumb` style in addition to the custom styles on the slider itself.

{{"demo": "StyledComponentsDeep.js", "hideToolbar": true}}

**CssModulesSliderDeep1.module.css**

```
.slider {
  color: #20b2aa;
}

.slider:hover {
  color: #2e8b57;
}

.slider :global .MuiSlider-thumb {
  border-radius: 1px;
}
```

**CssModulesSliderDeep1.js**

```
import * as React from 'react';
// webpack, parcel or else will inject the CSS into the page
import styles from './CssModulesSliderDeep1.module.css';
import Slider from '@mui/material/Slider';

export default function CssModulesSliderDeep1() {
  return (
    <div>
      <Slider defaultValue={30} />
      <Slider defaultValue={30} className={styles.slider} />
    </div>
  );
}
```

The above demo relies on the default `className` values, but you can provide your own class name with the `componentsProps` API.

**CssModulesSliderDeep2.module.css**

```
.slider {
  color: #20b2aa;
}

.slider:hover {
  color: #2e8b57;
}

.slider .thumb {
  border-radius: 1px;
}
```

**CssModulesSliderDeep2.js**

```
import * as React from 'react';
// webpack, parcel or else will inject the CSS into the page
import styles from './CssModulesSliderDeep2.module.css';
import Slider from '@mui/material/Slider';

export default function CssModulesSliderDeep2() {
  return (
    <div>
      <Slider defaultValue={30} />
      <Slider
        defaultValue={30}
        className={styles.slider}
        componentsProps={{ thumb: { className: styles.thumb } }}
      />
    </div>
  );
}
```

## Emotion

Star 15k downloads 18M/month

### The `css` prop

Emotion's **css()** method works seamlessly with MUI.

{{"demo": "EmotionCSS.js", "defaultCodeOpen": true}}

### Theme

It works exactly like styled components. You can use the same guide.

### The `styled()` API

It works exactly like styled components. You can use the same guide.

## Tailwind CSS

Star 60k downloads 16M/month

### Setup

If you are used to Tailwind CSS and want to use it together with the MUI components, you can start by cloning the Tailwind CSS example project. If you use a different framework, or already have set up your project, follow these steps:

1. Add Tailwind CSS to your project, following the instructions in https://tailwindcss.com/docs/installation.
2. Remove Tailwind's `base` directive in favor of the `CssBaseline` component provided by `@mui/material`, as it plays nicer with the MUI components.

**index.css**

```diff
-@tailwind base;
 @tailwind components;
 @tailwind utilities;
```

3. Add the `important` option, using the id of your app wrapper. For example, `#__next` for Next.js and `"#root"` for CRA:

**tailwind.config.js**

```diff
 module.exports = {
   content: [
     "./src/**/*.{js,jsx,ts,tsx}",
   ],
+  important: '#root',
   theme: {
     extend: {},
   },
   plugins: [],
 }
```

Most of the CSS used by Material UI has as specificity of 1, hence this `important` property is unnecessary. However, in a few edge cases, MUI uses nested CSS selectors that win over Tailwind CSS. Use this step to help ensure that the deeper elements can always be customized using Tailwind's utility classes. More details on this option can be found here https://tailwindcss.com/docs/configuration#selector-strategy

4. Fix the CSS injection order. Most CSS-in-JS solutions inject their styles at the bottom of the HTML `<head>`, which gives MUI precedence over Tailwind CSS. To reduce the need for the `important` property, you need to change the CSS injection order. Here's a demo of how it can be done in MUI:

```
import * as React from 'react';
import { StyledEngineProvider } from '@mui/material/styles';

export default function GlobalCssPriority() {
  return (
    <StyledEngineProvider injectFirst>
      {/* Your component tree. Now you can override MUI's styles. */}
    </StyledEngineProvider>
  );
}
```

**Note:** If you are using emotion and have a custom cache in your app, it will override the one coming from MUI. In order for the injection order to still be correct, you need to add the prepend option. Here is an example:

```
import * as React from 'react';
import { CacheProvider } from '@emotion/react';
import createCache from '@emotion/cache';

const cache = createCache({
  key: 'css',
  prepend: true,
});

export default function PlainCssPriority() {
  return (
    <CacheProvider value={cache}>
      {/* Your component tree. Now you can override MUI's styles. */}
    </CacheProvider>
  );
}
```

**Note:** If you are using styled-components and have `StyleSheetManager` with a custom `target`, make sure that the target is the first element in the HTML `<head>`. If you are curious to see how it can be done, you can take a look at the `StyledEngineProvider` implementation in the `@mui/styled-engine-sc` package.

## Usage

Now it's all set up and you can start using Tailwind CSS on the MUI components!

{{"demo": "StyledComponents.js", "hideToolbar": true}}

Open in StackBlitz

**index.tsx**

```
import * as React from 'react';
import Slider from '@mui/material/Slider';

export default function App() {
  return (
    <div>
      <Slider defaultValue={30} />
      <Slider defaultValue={30} className="text-teal-600" />
    </div>
  );
}
```

## Deeper elements

If you attempt to style the Slider, for example, you'll likely want to customize its child elements.

This example showcases how to override the Slider's `thumb` style.

{{"demo": "StyledComponentsDeep.js", "hideToolbar": true}}

**SliderThumbOverrides.tsx**

```
import * as React from 'react';
import Slider from '@mui/material/Slider';

export default function SliderThumbOverrides() {
  return (
    <div>
      <Slider defaultValue={30} />
      <Slider
        defaultValue={30}
        className="text-teal-600"
        componentsProps={{ thumb: { className: 'rounded-sm' } }}
      />
    </div>
  );
}
```

## Styling pseudo states

If you want to style a component's pseudo-state, you can use the appropriate key in the `classes` prop. Here is an example of how you can style the Slider's active state:

**SliderPseudoStateOverrides.tsx**

```
import * as React from 'react';
import Slider from '@mui/material/Slider';

export default function SliderThumbOverrides() {
  return <Slider defaultValue={30} classes={{ active: 'shadow-none' }} />;
}
```

## ~~JSS~~ TSS

JSS itself is no longer supported in MUI however, if you like the hook-based API ( `makeStyles` → `useStyles` ) that `react-jss` was offering you can opt for `tss-react` .

TSS integrates well with MUI and provide a better TypeScript support than JSS.

> *If you are updating from `@material-ui/core` (v4) to `@mui/material` (v5) checkout migration guide.*

```
import { render } from 'react-dom';
import { CacheProvider } from '@emotion/react';
import createCache from '@emotion/cache';
import { ThemeProvider } from '@mui/material/styles';

export const muiCache = createCache({
  key: 'mui',
  prepend: true,
});

//NOTE: Don't use <StyledEngineProvider injectFirst/>
render(
  <CacheProvider value={muiCache}>
    <ThemeProvider theme={myTheme}>
      <Root />
    </ThemeProvider>
  </CacheProvider>,
  document.getElementById('root'),
);
```

Now you can simply `import { makeStyles, withStyles } from 'tss-react/mui'` .
The theme object that will be passed to your callbacks functions will be the one you get with `import { useTheme } from '@mui/material/styles'` .

If you want to take controls over what the `theme` object should be, you can re-export `makeStyles` and `withStyles` from a file called, for example, `makesStyles.ts` :

```
import { useTheme } from '@mui/material/styles';
//WARNING: tss-react require TypeScript v4.4 or newer. If you can't update use:
//import { createMakeAndWithStyles } from "tss-react/compat";
import { createMakeAndWithStyles } from 'tss-react';
```

```
export const { makeStyles, withStyles } = createMakeAndWithStyles({
  useTheme,
  /*
    OR, if you have extended the default mui theme adding your own custom
properties:
    Let's assume the myTheme object that you provide to the <ThemeProvider /> is of
    type MyTheme then you'll write:
    */
  //"useTheme": useTheme as (()=> MyTheme)
});
```

Then, the library is used like this:

```
import { makeStyles } from 'tss-react/mui';

export function MyComponent(props: Props) {
  const { className } = props;

  const [color, setColor] = useState<'red' | 'blue'>('red');

  const { classes, cx } = useStyles({ color });

  //Thanks to cx, className will take priority over classes.root
  return <span className={cx(classes.root, className)}>hello world</span>;
}

const useStyles = makeStyles<{ color: 'red' | 'blue' }>()((theme, { color }) => ({
  root: {
    color,
    '&:hover': {
      backgroundColor: theme.palette.primary.main,
    },
  },
}));
```

For info on how to setup SSR or anything else, please refer to the TSS documentation.

> ⚠ ***Keep*** **`@emotion/styled`** ***as a dependency of your project***. *Even if you never use it explicitly, it's a peer dependency of* `@mui/material` .

> ⚠ *For Storybook: As of writing this lines storybook still uses by default emotion 10.*
> *Material UI and TSS runs emotion 11 so there is some changes to be made to your* `.storybook/main.js` *to make it uses emotion 11.*