Interoperabilidade da Biblioteca de Estilo

Enquanto você pode usar a solução de estilo baseada em emotion fornecida pelo Material-UI para estilizar sua aplicação, você também pode usar o que você já conhece e ama (desde CSS simples a styled-components).

Este guia tem como objetivo documentar as alternativas mais populares, mas você deve descobrir que os princípios aplicados aqui podem ser adaptados para outras bibliotecas. Existem exemplos para as seguintes soluções de estilo:

- CSS puro
- CSS global
- Styled Components
- Módulos CSS
- Emotion
- Tailwind CSS
- ~~JSS~~ TSS

CSS puro

Nada extravagante, apenas CSS.

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

Edit on CodeSandbox

PlainCssSlider.css

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

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

PlainCssSlider.js

```
import * as React from 'react';
import Slider from '@material-ui/core/Slider';
import './PlainCssSlider.css';

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

Ordem de injeção do CSS ⚠

Nota: A maioria das soluções CSS-in-JS injetam seus estilos na parte inferior do HTML `<head>`, que dá precedência ao Material-UI sobre seus estilos customizados. Para remover a necessidade de **!important**, você precisa alterar a ordem de injeção do CSS. Here's a demo of how it can be done in Material-UI:

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

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

Note: If you are using emotion and have a custom cache in your app, that one will override the one coming from Material-UI. In order for the injection order to still be correct, you need to add the prepend option. Aqui está um exemplo:

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

export default function GlobalCssPriority() {
  return (
    <StyledEngineProvider injectFirst>
      {/* 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 `StylesProvider` implementation in the `@material-ui/styled-engine-sc` package.

Elementos mais profundos

Se você tentar estilizar o Slider, você provavelmente gostaria de afetar alguns dos elementos filhos de Slider, por exemplo o thumb. No Material-UI, todos os elementos filhos têm uma especificidade aumentada de 2: `.parent .child {}`. Ao escrever uma sobrescrita, você precisa fazer o mesmo.

Os exemplos a seguir substituem o estilo de `thumb` do controle slider, além dos estilos customizados no slider em si.

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

PlainCssSliderDeep1.css

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

```
}

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

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

PlainCssSliderDeep1.js

```
import * as React from 'react';
import Slider from '@material-ui/core/Slider';
import './PlainCssSliderDeep1.css';

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

A demonstração acima depende dos [valores padrão de `className`], mas você pode fornecer seu próprio nome de classe com a API `componentsProps`.

PlainCssSliderDeep2.css

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

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

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

PlainCssSliderDeep2.js

```
import * as React from 'react';
import Slider from '@material-ui/core/Slider';
import './PlainCssSliderDeep2.css';

export default function PlainCssSliderDeep2() {
  return (
```

```
    <div>
      <Slider defaultValue={30} />
      <Slider
        defaultValue={30}
        className="slider"
        componentsProps={{ thumb: { className: 'thumb' } }}
      />
    </div>
  );
}
```

CSS global

Fornecer explicitamente os nomes das classes ao componente é um esforço excessivo? Você pode segmentar os nomes de classe gerados por Material-UI.

Edit on CodeSandbox

GlobalCssSlider.css

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

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

GlobalCssSlider.js

```
import * as React from 'react';
import Slider from '@material-ui/core/Slider';
import './GlobalCssSlider.css';

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

Ordem de injeção do CSS ⚠

Para remover a necessidade de **!important**, você precisa alterar a ordem de injeção do CSS. **Nota:** A maioria das soluções CSS-in-JS injetam seus estilos na parte inferior do HTML `<head>`, que dá precedência ao Material-UI sobre seus estilos customizados. Here's a demo of how it can be done in Material-UI:

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

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

Note: If you are using emotion and have a custom cache in your app, that one will override the one coming from Material-UI. In order for the injection order to still be correct, you need to add the prepend option. Aqui está um exemplo:

```
Agora você pode sobrescrever os estilos do Material-UI. import * as React from
'react';
import { StylesProvider } from '@material-ui/core';

export default function GlobalCssPriority() {
  return (
    <StylesProvider injectFirst>
      {/* Your component tree. 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}>
      {/* Sua árvore de componentes. */}
    </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 `StylesProvider` implementation in the `@material-ui/styled-engine-sc` package.

Elementos mais profundos

Se você tentar estilizar o Slider, você provavelmente gostaria de afetar alguns dos elementos filhos de Slider, por exemplo o thumb. No Material-UI, todos os elementos filhos têm uma especificidade aumentada de 2: `.parent .child {}`. Ao escrever uma sobrescrita, você precisa fazer o mesmo.

O exemplo a seguir substituem o estilo de `thumb` do controle slider, além dos estilos customizados no slider em si.

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

GlobalCssSliderDeep.css

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

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

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

GlobalCssSliderDeep.js

```
import * as React from 'react';
import Slider from '@material-ui/core/Slider';
import './GlobalCssSliderDeep.css';

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

Styled Components

Alterar o motor de estilo padrão

Por padrão, os componentes do Material-UI vêm com emotion como seu motor de estilo. Se, no entanto, você gostaria de usar `styled-components`, você pode configurar sua aplicação seguindo este projeto de exemplo.

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

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

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

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

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

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

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

Elementos mais profundos

Se você tentar estilizar o Slider, você provavelmente gostaria de afetar alguns dos elementos filhos de Slider, por exemplo o thumb. No Material-UI, todos os elementos filhos têm uma especificidade aumentada de 2: `.parent .child {}`. Ao escrever uma sobrescrita, você precisa fazer o mesmo.

Os exemplos a seguir substituem o estilo de `thumb` do controle slider, além dos estilos customizados no slider em si.

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

Ao usar o provedor de tema do Material-UI, o tema estará disponível no contexto do tema do motor de estilo também (emotion ou styled-components, dependendo da sua configuração).

```
import * as React from 'react';
import { styled } from '@material-ui/core/styles';
import Slider from '@material-ui/core/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>
  );
}
```

Tema

Você é encorajado a compartilhar o mesmo objeto de tema entre Material-UI e o resto de seu projeto.

> ⚠️ *Se você já **estiver** usando um tema customizando com styled-components ou emotion, ele pode não ser compatível com a especificação do tema do Material-UI. Se ele não é compatível, você precisa renderizar o ThemeProvider do Material-UI **primeiro**. Isto irá garantir que as estruturas do tema estejam isoladas. Isso é ideal para a adoção progrressiva dos componentes da base de código do Material-UI.*

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"}}

Portais

O Portal fornece uma maneira de primeira classe para renderizar filhos em um nó DOM que existe fora da hierarquia DOM do componente pai. Devido a maneira como o escopo de CSS do styled-components funciona, você pode encontrar problemas nos quais o estilo não é aplicado.

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. O exemplo a seguir mostra uma solução alternativa:

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

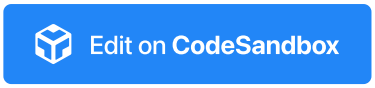
```
const StyledTooltip = styled(({ className, ...props }) => (
  <Tooltip {...props} classes={{ popper: className }} />
))`
  & . MuiTooltip-tooltip {
    background: navy;
  }
`;
```

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

Módulos CSS

É difícil saber a participação de mercado nesta solução de estilo, pois é dependente da solução de empacotamento que as pessoas estão usando.

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



CssModulesSlider.module.css

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

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

CssModulesSlider.js

```
import * as React from 'react';
import Slider from '@material-ui/core/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>
  );
}
```

Ordem de injeção do CSS ⚠️

Para remover a necessidade de **!important**, você precisa alterar a ordem de injeção do CSS. **Nota:** A maioria das soluções CSS-in-JS injetam seus estilos na parte inferior do HTML `<head>`, que dá precedência ao Material-UI sobre seus estilos customizados. Here's a demo of how it can be done in Material-UI:

```
import * as React from 'react';
import { StyledEngineProvider } from '@material-ui/core/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 Material-UI. In order for the injection order to still be correct, you need to add the prepend option. Aqui está um exemplo:

```
*/}
    </StylesProvider>
  );
} 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}>
      {/* Sua árvore de componentes. 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 `StylesProvider` implementation in the `@material-ui/styled-engine-sc` package.

Elementos mais profundos

Se você tentar estilizar o Slider, você provavelmente gostaria de afetar alguns dos elementos filhos de Slider, por exemplo o thumb. No Material-UI, todos os elementos filhos têm uma especificidade aumentada de 2: `.parent .child {}`. Ao escrever uma sobrescrita, você precisa fazer o mesmo.

Os exemplos a seguir substituem o estilo de `thumb` do controle slider, além dos estilos customizados no slider em si.

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

CssModulesSliderDeep1.module.css

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

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

.slider . 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 '@material-ui/core/Slider';

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

A demonstração acima depende dos valores padrão de `className`, mas você pode fornecer seu próprio nome de classe com a API `componentsProps`.

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 '@material-ui/core/Slider';

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

Emotion

A propriedade `css`

O método **css()** do Emotion funciona perfeitamente com Material-UI.

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

Tema

Funciona exatamente como styled components. Você pode usar o mesmo guia.

A API `styled()`

Funciona exatamente como styled components. Você pode usar o mesmo guia.

Tailwind CSS

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

```
-@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

```
 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:

```
*/}
    </StylesProvider>
```

```
  );
} 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}>
      {/* Sua árvore de componentes. 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.*