Theming

Customize Material UI with your theme. You can change the colors, the typography and much more.

The theme specifies the color of the components, darkness of the surfaces, level of shadow, appropriate opacity of ink elements, etc.

Themes let you apply a consistent tone to your app. It allows you to **customize all design aspects** of your project in order to meet the specific needs of your business or brand.

To promote greater consistency between apps, light and dark theme types are available to choose from. By default, components use the light theme type.

Theme provider

If you wish to customize the theme, you need to use the `ThemeProvider` component in order to inject a theme into your application. However, this is optional; Material UI components come with a default theme.

`ThemeProvider` relies on the context feature of React to pass the theme down to the components, so you need to make sure that `ThemeProvider` is a parent of the components you are trying to customize. You can learn more about this in the API section.

Theme configuration variables

Changing the theme configuration variables is the most effective way to match Material UI to your needs. The following sections cover the most important theme variables:

- `.palette`
- `.typography`
- `.spacing`

- `.breakpoints`
- `.zIndex`
- `.transitions`
- `.components`

You can check out the default theme section to view the default theme in full.

Custom variables

When using Material UI's theme with MUI System or any other styling solution, it can be convenient to add additional variables to the theme so you can use them everywhere. For instance:

```
const theme = createTheme({
  status: {
    danger: orange[500],
  },
});
```

⚠ `vars` is a private field for CSS theme variables. It will throw an error if you try to pass a value to it:

```
createTheme({
  vars: { ... }, // ❌ error
})
```

TypeScript

You have to use module augmentation to add new variables to the `Theme` and `ThemeOptions`.

```
declare module '@mui/material/styles' {
  interface Theme {
    status: {
      danger: string;
    };
  }
  // allow configuration using `createTheme`
  interface ThemeOptions {
    status?: {
      danger?: string;
    };
  }
}
```

```
<ThemeProvider theme={theme}>
  <CustomCheckbox defaultChecked />
</ThemeProvider>
```

To add extra variables to the `theme.palette`, see palette customization.

Theme builder

The community has built great tools to build a theme:

- mui-theme-creator: A tool to help design and customize themes for the Material UI component library. Includes basic site templates to show various components and how they are affected by the theme
- Material palette generator : The Material palette generator can be used to generate a palette for any color you input.

Accessing the theme in a component

You can access the theme variables inside your functional React components using the `useTheme` hook:

```
import { useTheme } from '@mui/material/styles';

function DeepChild() {
  const theme = useTheme();
  return <span>{`spacing ${theme.spacing}`}</span>;
}
```

Nesting the theme

You can nest multiple theme providers.

```
<ThemeProvider theme={outerTheme}>
  <Checkbox defaultChecked />
  <ThemeProvider theme={innerTheme}>
    <Checkbox defaultChecked />
  </ThemeProvider>
</ThemeProvider>
```

The inner theme will **override** the outer theme. You can extend the outer theme by providing a function:

API

`createTheme(options, ...args) => theme`

Generate a theme base on the options received. Then, pass it as a prop to `ThemeProvider`.

Arguments

1. `options` (*object*): Takes an incomplete theme object and adds the missing parts.
2. `...args` (*object[]*): Deep merge the arguments with the about to be returned theme.

⚠ Only the first argument (`options`) is processed by the `createTheme` function. If you want to actually merge two themes' options and create a new one based on them, you may want to deep merge the two options and provide them as a first argument to the `createTheme` function.

```
import { deepmerge } from '@mui/utils';
import { createTheme } from '@mui/material/styles';

const theme = createTheme(deepmerge(options1, options2));
```

Returns

`theme` (*object*): A complete, ready-to-use theme object.

Examples

```
import { createTheme } from '@mui/material/styles';
import { green, purple } from '@mui/material/colors';

const theme = createTheme({
  palette: {
    primary: {
      main: purple[500],
    },
    secondary: {
      main: green[500],
    },
  },
});
```

Theme composition: using theme options to define other options

When the value for a theme option is dependent on another theme option, you should compose the theme in steps.

```
import { createTheme } from '@mui/material/styles';

let theme = createTheme({
  palette: {
    primary: {
      main: '#0052cc',
    },
    secondary: {
      main: '#edf2ff',
    },
  },
});

theme = createTheme(theme, {
  palette: {
    info: {
      main: theme.palette.secondary.main,
    },
  },
```

Think of creating a theme as a two-step composition process: first, you define the basic design options; then, you'll use these design options to compose other options.

WARNING: `theme.vars` is a private field used for CSS variables support. Please use another name for a custom object.

`responsiveFontSizes(theme, options) => theme`

Generate responsive typography settings based on the options received.

Arguments

1. `theme` (*object*): The theme object to enhance.
2. `options` (*object* [optional]):

- `breakpoints` (*array<string>* [optional]): Default to `['sm', 'md', 'lg']`. Array of breakpoints (identifiers).
- `disableAlign` (*bool* [optional]): Default to `false`. Whether font sizes change slightly so line heights are preserved and align to Material Design's 4px line height grid. This requires a unitless line height in the theme's styles.
- `factor` (*number* [optional]): Default to `2`. This value determines the strength of font size resizing. The higher the value, the less difference there is between font sizes on small screens. The lower the value, the bigger font sizes for small screens. The value must be greater than 1.
- `variants` (*array<string>* [optional]): Default to all. The typography variants to handle.

Returns

`theme` (*object*): The new theme with a responsive typography.

Examples

```
import { createTheme, responsiveFontSizes } from '@mui/material/styles';

let theme = createTheme();
theme = responsiveFontSizes(theme);
```

`unstable_createMuiStrictModeTheme(options, ...args) => theme`

WARNING: Do not use this method in production.

Generates a theme that reduces the amount of warnings inside `React.StrictMode` like `Warning: findDOMNode is deprecated in StrictMode`.

Requirements

Currently `unstable_createMuiStrictModeTheme` adds no additional requirements.

Arguments

1. `options` (*object*): Takes an incomplete theme object and adds the missing parts.
2. `...args` (*object[]*): Deep merge the arguments with the about to be returned theme.

Returns

`theme` (*object*): A complete, ready-to-use theme object.

Examples

```
import { unstable_createMuiStrictModeTheme } from '@mui/material/styles';

const theme = unstable_createMuiStrictModeTheme();

function App() {
  return (
    <React.StrictMode>
      <ThemeProvider theme={theme}>
        <LandingPage />
      </ThemeProvider>
    </React.StrictMode>
  );
}
```

`ThemeProvider`

This component takes a `theme` prop and applies it to the entire React tree that it is wrapping around. It should preferably be used at **the root of your component tree**.

Props

Name	Type	Description
children *	node	Your component tree.
theme *	union: object | func	A theme object, usually the result of `createTheme()`. The provided theme will be merged with the default theme. You can provide a function to extend the outer theme.

Examples

```
import * as React from 'react';
import { red } from '@mui/material/colors';
import { ThemeProvider, createTheme } from '@mui/material/styles';

const theme = createTheme({
  palette: {
    primary: {
      main: red[500],
    },
  },
});
```

```
function App() {
  return <ThemeProvider theme={theme}>...</ThemeProvider>;
}
```