CSS variables

Learn about the experimental API for using CSS variables with Material UI components.

CSS variables provide significant improvements in developer experience related to theming and customization. With these variables, you can inject a theme into your app's stylesheet *at build time* to apply the user's selected settings before the whole app is rendered.

This solves the problem of dark-mode SSR flickering; lets you provide your users with multiple themes beyond light and dark; and offers a better debugging experience overall, among other benefits.

Previously, these CSS variables were only available as an experimental API in the MUI System package. Now they are ready for experimental use with Material UI components.

> *If you want to see wider support for this API across Material UI's component library, please feel free to contribute to the ongoing development. Make sure to check the GitHub issue that keeps track of our progress, to see if anyone else is currently working on a component you're interested in.*
>
> *We'd appreciate any feedback about this API, as it is still in development.*

Introduction

The CSS variables API relies on a new experimental provider for the theme called `Experimental_CssVarsProvider` to inject styles into Material UI components. In addition to providing the theme in the inner React context, this new provider also generates CSS variables out of all tokens in the theme that are not functions, and makes them available in the context as well.

All of these variables are accessible in an object in the theme called `vars`. The structure of this object is nearly identical to the theme structure, the only difference is that the values represent CSS variables.

Usage

`Experimental_CssVarsProvider` is a new experimental provider that attaches all generated CSS variables to the theme and puts them in React's context. Children elements under this provider will also be able to read the CSS variables from the theme.

```
import { Experimental_CssVarsProvider as CssVarsProvider } from
'@mui/material/styles';

function App() {
  return <CssVarsProvider>...</CssVarsProvider>;
}
```

Customizing components

Because the CSS variables API is an experimental feature, it is currently only supported by the `Button` component. To customize it using CSS variables, you'll need to wrap your application with `Experimental_CssVarsProvider`.

Play around with the demo below!

{{"demo": "CssVariablesCustomization.js", "iframe": true }}

If you are using TypeScript you should use module augmentation to update the `Theme` structure:

```
import { Theme as MuiTheme } from '@mui/material/styles';

declare module '@mui/material/styles' {
  interface Theme {
    vars: Omit<
      MuiTheme,
      'typography' | 'mixins' | 'breakpoints' | 'direction' | 'transitions'
    >;
  }
}
```

Customizing the theme

If you want, for example, to override Material UI's default color schemes, you can use the `experimental_extendTheme` utility.

```
const theme = experimental_extendTheme({
  colorSchemes: {
    light: {
      palette: {
        primary: teal,
        secondary: deepOrange,
      },
    },
    dark: {
      palette: {
        primary: cyan,
        secondary: orange,
      },
    },
  },
});
```

{{"demo": "CssVarsCustomTheme.js", "iframe": true }}

If you are using `ThemeProvider`, you can replace it with the new experimental provider.

```
- import { ThemeProvider, createTheme } from '@mui/material/styles';
+ import { Experimental_CssVarsProvider as CssVarsProvider } from
'@mui/material/styles';

function App() {
  return (
-     <ThemeProvider theme={createTheme()}>
-       ...
-     </ThemeProvider>
+     <CssVarsProvider>
+       ...
+     </CssVarsProvider>
```

```
  )
}
```

Toggle between light and dark mode

`Experimental_CssVarsProvider` provides light and dark mode by default. It stores the user's selected mode and syncs it with the browser's local storage internally. You can read and update the mode via the `useColorScheme` API.

```
import {
  Experimental_CssVarsProvider as CssVarsProvider,
  useColorScheme,
} from '@mui/material/styles';

const ModeSwitcher = () => {
  const { mode, setMode } = useColorScheme();
  const [mounted, setMounted] = React.useState(false);

  React.useEffect(() => {
    setMounted(true);
  }, []);

  if (!mounted) {
    // for server-side rendering
    // Read more on https://github.com/pacocoursey/next-themes#avoid-hydration-
mismatch
    return null;
  }

  return (
    <Button
      variant="outlined"
      onClick={() => {
        if (mode === 'light') {
          setMode('dark');
        } else {
          setMode('light');
        }
      }}
    >
      {mode === 'light' ? 'Dark' : 'Light'}
    </Button>
  );
};

function App() {
  return (
    <CssVarsProvider>
      <ModeSwitcher />
    </CssVarsProvider>
  );
}
```

Server-side rendering

To prevent the dark-mode SSR flickering during the hydration phase, place `getInitColorSchemeScript()` before the `<Main />` tag.

Next.js

To use the API with a Next.js project, add the following code to the custom [`pages/_document.js`]() file:

```
import Document, { Html, Head, Main, NextScript } from 'next/document';
import { getInitColorSchemeScript } from '@mui/material/styles';

export default class MyDocument extends Document {
  render() {
    return (
      <Html>
        <Head>...</Head>
        <body>
          {getInitColorSchemeScript()}
          <Main />
          <NextScript />
        </body>
      </Html>
    );
  }
}
```

Gatsby

To use the API with a Gatsby project, add the following code to the custom [`gatsby-ssr.js](https://www.gatsbyjs.com/docs/reference/config-files/gatsby-ssr/) file`:

```
import React from 'react';
import { getInitColorSchemeScript } from '@mui/material/styles';

export function onRenderBody({ setPreBodyComponents }) {
  setPreBodyComponents([getInitColorSchemeScript()]);
}
```

API

`<CssVarsProvider>` props

- `defaultMode?: 'light' | 'dark' | 'system'` - Application's default mode (`light` by default)
- `disableTransitionOnChange : boolean` - Disable CSS transitions when switching between modes
- `enableColorScheme: boolean` - Indicate to the browser which color scheme is used (light or dark) for rendering built-in UI

- `prefix: string` - CSS variable prefix
- `theme: ThemeInput` - the theme provided to React's context
- `modeStorageKey?: string` - localStorage key used to store application `mode`
- `attribute?: string` - DOM attribute for applying color scheme

`useColorScheme: () => ColorSchemeContextValue`

- `mode: string` - The user's selected mode
- `setMode: mode => {…}` - Function for setting the `mode`. The `mode` is saved to internal state and local storage; if `mode` is null, it will be reset to the default mode

`getInitColorSchemeScript: (options) => React.ReactElement`

options

- `enableSystem?: boolean`: - If `true` and the selected mode is not `light` or `dark`, the system mode is used
- `modeStorageKey?: string`: - localStorage key used to store application `mode`
- `attribute?: string` - DOM attribute for applying color scheme