React Typography component

Typography

Use typography to present your design and content as clearly and efficiently as possible.

Too many type sizes and styles at once can spoil any layout. A typographic scale has a limited set of type sizes that work well together along with the layout grid.

{{“component”: “modules/components/ComponentLinkHeader.js”}}

General

The *Roboto* font will **not** be automatically loaded by MUI. You are responsible for loading any fonts used in your application. Roboto Font has a few easy ways to get started. For more advanced configuration, check out the theme customization section.

Roboto Font CDN

Shown below is a sample link markup used to load the Roboto font from a CDN:

```
<link
  rel="stylesheet"
  href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&display=swap"
/>
```

Install with npm

You can install it by typing the below command in your terminal:

```
npm install @fontsource/roboto
```

Then, you can import it in your entry-point.

```
import '@fontsource/roboto/300.css';
import '@fontsource/roboto/400.css';
import '@fontsource/roboto/500.css';
import '@fontsource/roboto/700.css';
```

For more info check out Fontsource.

Fontsource can be configured to load specific subsets, weights and styles. MUI default typography configuration only relies on 300, 400, 500, and 700 font weights.

Component

The Typography component makes it easy to apply a default set of font weights and sizes in your application.

{{“demo”: “Types.js”}}

Theme

In some situations you might not be able to use the `Typography` component. Hopefully, you might be able to take advantage of the `typography` keys of the theme.

{{“demo”: “TypographyTheme.js”}}

Changing the semantic element

The Typography component uses the `variantMapping` prop to associate a UI variant with a semantic element. It’s important to realize that the style of a typography component is independent from the semantic underlying element.

- You can change the underlying element for a one-off situation with the `component` prop:

```
{
  /* There is already an h1 in the page, let's not duplicate it. */
}
<Typography variant="h1" component="h2">
  h1. Heading
</Typography>;
```

- You can change the mapping globally using the theme:

```
const theme = createTheme({
  components: {
    MuiTypography: {
      defaultProps: {
        variantMapping: {
          h1: 'h2',
          h2: 'h2',
          h3: 'h2',
          h4: 'h2',
          h5: 'h2',
```

```
          h6: 'h2',
          subtitle1: 'h2',
          subtitle2: 'h2',
          body1: 'span',
          body2: 'span',
        },
      },
    },
  },
});
```

Adding & disabling variants

In addition to using the default typography variants, you can add custom ones, or disable any you don't need. See the Adding & disabling variants example for more info.

System props

As a CSS utility component, the `Typography` supports all `system` properties. You can use them as prop directly on the component. For instance, a margin-top:

```
<Typography mt={2}>
```

Accessibility

A few key factors to follow for an accessible typography:

- **Color**. Provide enough contrast between text and its background, check out the minimum recommended WCAG 2.0 color contrast ratio (4.5:1).
- **Font size**. Use relative units (rem) to accommodate the user's settings.
- **Heading hierarchy**. Don't skip heading levels. In order to solve this problem, you need to separate the semantics from the style.