CSS styles in angular.io

This document gives an overview of how angular.io CSS styles are implemented and organized in files.

General

Styles are implemented using Sass and stored in `.scss` files in src/styles/.

> **NOTE:** We do not use inline styles for components. Styles for components are defined in `.scss` files in src/styles/ and are not referenced from the component files.

File organization

The `.scss` files are organized in the following sub-directories: - 0-base/: General styles affecting the whole application. - 1-layouts/: Styles for areas/components related to the layout of the application, such as `top-menu`, `footer`, marketing pages, etc. - 2-modules/: Styles for specialized components (such as buttons, code, labels, etc.) and specific pages (such as API list page, "Features" page, etc.). - custom-themes/: Entry-points for the different themes available in the application (currently `light` and `dark`).

There are also some top-level files in `[src/styles/](.): - [_app-theme.scss](./_app-theme.scss): Defines a`theme()‘ Sass mixin for creating an application theme. - _constants.scss: Defines several constants to be used throughout the styles. - _mixins.scss: Defines Sass mixins to be used throughout the styles. - _print.scss: Contains styles to be applied when printing. - main.scss: Styles entry-point.

Styles for a specific area/component

For each area/component, there is a sub-directory in either `1-layouts/` or `2-modules/`.

Each such sub-directory contains a `<name>.scss` file with styles for the corresponding area/component and may also contain a `<name>-theme.scss` file with styles related to theming. See the next section for more details.

When appropriate, the styles in these files should be scoped to the targeted component (for example, by using the component's selector).

Theming

Angular.io supports choosing between themes. Currently, a `light` and a `dark` theme are supported. See also #41129 for more details/discussions around the theming implementation.

Styles for theming

Styles for each area/component are split between two files: `<name>.scss` and `<name>-theme.scss`.

The general styles go into `<name>.scss`. If an area/component has styles that could change based on the active theme (such as color or background), these go into the `*-theme.scss` file. Color-related styles in particular should always go into that file, even if the styles do not currently change between themes. This will make it easier to adjust/add more themes in the future.

Within each `*-theme.scss` file is a Sass mixin that takes in a Material theme configuration and generates the appropriate styling for that theme and component.

Advantages of the chosen approach: - Theming is contained to one file per component. Developers need only be aware of the existence of a single file when making theming related changes to a component - Themes can be lazy-loaded at runtime, preventing growth of the default styles chunk every time a new theme is implemented.

Disadvantages of the chosen approach: - Splitting styles into two files means that some selectors will be duplicated, resulting in an increase of the total styles chunk size.

Applying a theme at runtime

When building the app the following styles bundles are generated: - One based on `main.scss`, which is always included in `index.html` and contains the general (non-theme-specific) styles. - One bundle per theme, which is loaded "on demand" and contains theme-specific styles.

A theme bundle is loaded at runtime using the CSS `@import` rule. The appropriate theme is chosen based on the user's preference (either through an operating system setting or a user agent setting) using the `prefers-color-scheme` media query.

Once the application has bootstrapped, the theme may be updated based on a previously stored application-specific preference. Whenever the user explicitly changes the theme using the theme-toggle component, the new preference is stored for use in future visits.

> **NOTE:** The theming infrastructure is based on the material.angular.io implementation.