Tomas Trajan Follow
I ❤ building useful things! It involves code & design 🛠 Passionate about Frontend, Angular, Material …
Jun 1 · 6 min read

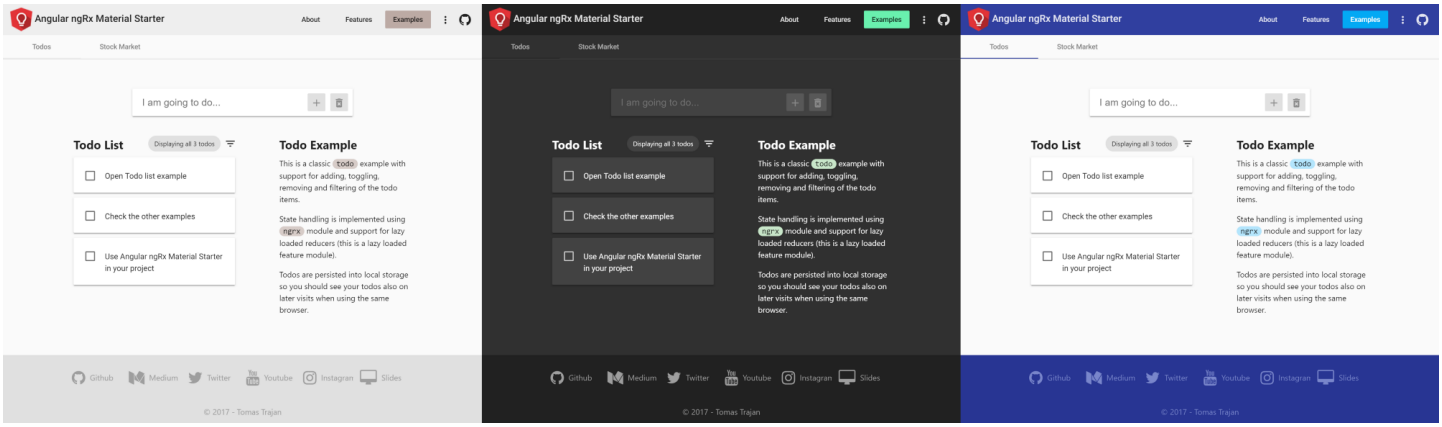
The complete guide to Angular Material Themes

Theme your Angular Material apps like a **PRO**, examples included ☺

For what is the web if not to express our ideas?

Theming support in an application can be pretty useful. From nice to have feature like letting user to chose from available color schemes to get personalized experience to more mission critical branding capabilities like building multi-tenant SaaS product where every client wants to use their own domain, logo and colors to offer branded services to their end customers.

Lately I have been working on Angular ngRx Material Starter project and surprisingly, one of the **best features of new Angular Material** component library proved to be its **theming** capabilities. It enables us to implement beautiful themes with only a little effort on our side!



Angular Material theming capabilities in practice (light, dark and default themes of our example project)

> *Unfortunately, as of June 2017, the official theming documentation seems to be a bit lackluster and it takes a while to figure out how the whole thing actually works and how to get access to the needed colors defined as a part of the theme.*

Anatomy of a theme

Defining a theme in Angular Material is extremely simple, all we need to do is to create a theme file, select three color palettes for the main theme colors—**primary**, **accent** and **warn**—and we are all set!

Feel free to explore all the palettes that are available out of the box.

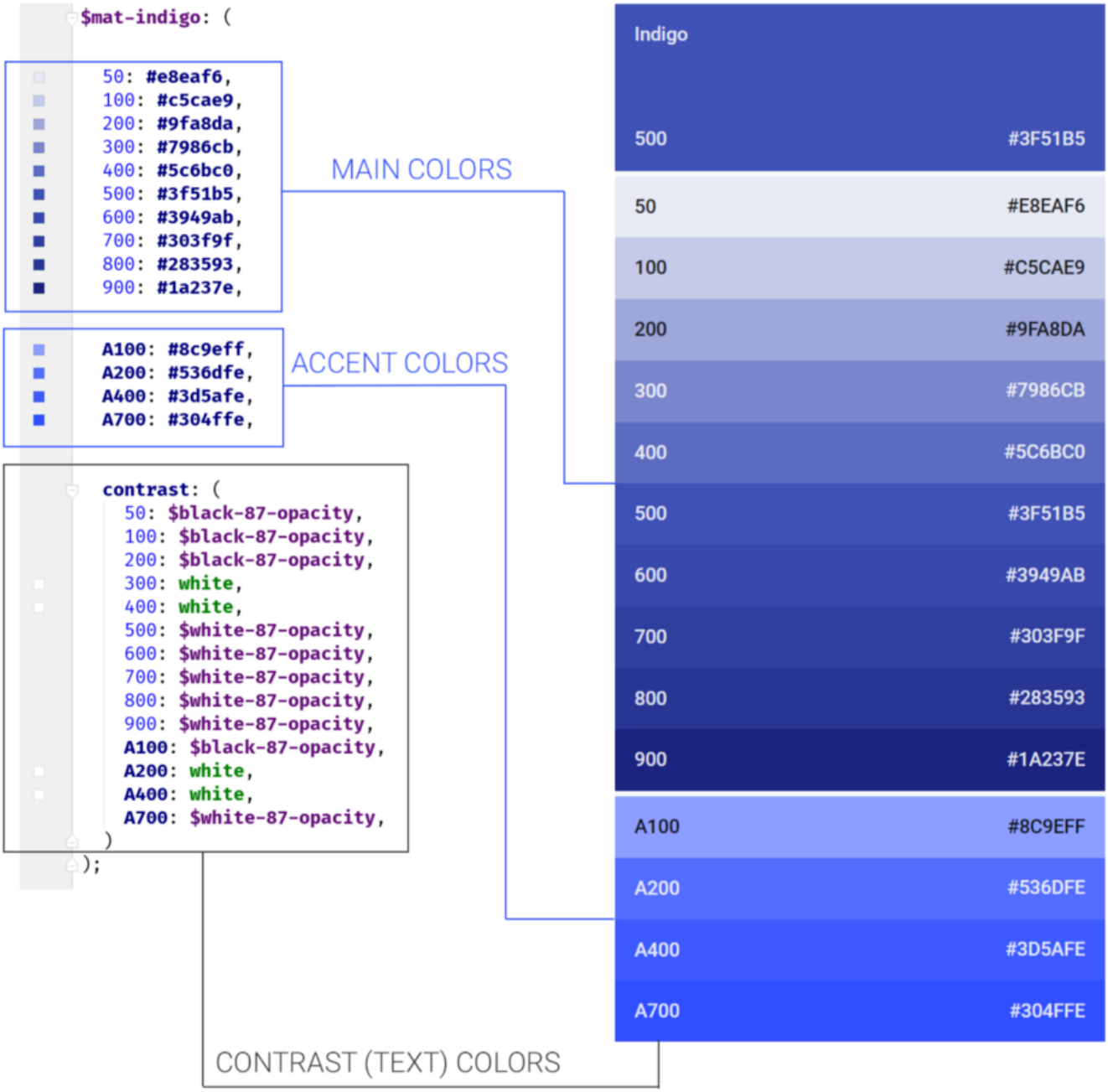
```
1  // define 3 theme color
2  // mat-palette accepts $palette-name, main, lighter an
3  $my-theme-primary: mat-palette($mat-indigo, 700, 300,
4  $my-theme-accent: mat-palette($mat-light-blue);
5  $my-theme-warn: mat-palette($mat-deep-orange, A200);
6
7  // create theme (use mat-dark-theme for themes with da
8  $my-theme: mat-light-theme(
9      $my-theme-primary
```

Angular Material theme definition file

The numbers behind the palette variable name select particular shades of chosen color for **default**, **lighter** and **darker** variants. If no numbers are provided Angular Material selects sensible defaults `$default: 500, $lighter: 100, $darker: 700`.

> *Yeah cool, variable names and numbers, but what does it mean in practice?!*

No worries I got your back ☺



Anatomy of a Angular Material Palette

Even though every palette contains **accent** shades (eg: A100), we don't necessarily need to use them as the accent color of our theme, we may simply choose different palette instead.

> *In example above, theme variable* `$my-theme-accent` *is set to* `$mat-light-blue` *palette instead of accent shades of* `$mat-indigo` *palette which is used for* `$my-theme-primary` *variable…*

In the code example above we created our theme using `mat-light-theme` function. This means that all the other "neutral" colors of our theme will be various shades of black and white with mostly white (or very light grey) backgrounds and mostly dark elements like text, dividers, shadows …

Angular Material provides also `mat-dark-theme` function which does exactly the opposite—dark backgrounds and light elements. We will get into more details of these two later…

Using our custom themes

Great, we defined our custom theme but that doesn't do anything by itself. Next step is to include our custom theme in main style file of our application (usually `styles.scss`).

```
@import '~@angular/material/theming';

// always include only once per project
@include mat-core();

// import our custom theme
@import 'my-theme.scss';

// specify theme class eg: <body class="my-theme"> ...
.my-theme {
```

Using custom theme in our application

> *Using specific css class for single theme is not necessary per se but it will help us when adding additional themes*

Adding multiple custom themes

To use multiple themes we simply need to import additional themes and create respective css classes for each theme.

```scss
@import '~@angular/material/theming';

// always include only once per project
@include mat-core();

// import our custom themes
@import 'my-theme.scss';
@import 'my-light-theme.scss';
@import 'my-dark-theme.scss';

.my-theme {
  @include angular-material-theme($my-theme);
}

// additional css classes
// you can hardcode one of them on the <body> tag
```

Use multiple custom themes

Theme class and overlay handling

Depending on our particular use case we might need to implement some dynamic css class switching (with `[class]`) to enable user to switch themes using application preferences during runtime or use parametrized build (eg: define variable in webpack) to build our application using desired theme by adding correct css class to the `<body>` tag during build.

Angular Material contains components like dropdown or dialog which create overlay over the application's default layout, to theme these elements we have to set theme class also on the `overlayContainer`.

```
import { OverlayContainer } from '@angular/material';

export class AppComponent implements OnInit {

  // use this to set correct theme class on app holder
  // eg: <div [class]="themeClass">...</div>
  themeClass: string;

  overlayContainer;

  constructor(
    private overlayContainer: OverlayContainer
  ) {}

```

Setting theme on the overlayContainer of Angular Material during runtime

Cool our application now supports use of potentially unlimited number of different themes🎨🎶. This is useful in itself but these themes will only style components provided by the Angular Material library itself. While there are quite some components available (and even more are waiting in the pipeline for the future releases) almost no application can do without implementing some custom components.

Theming our custom components

Let's say we are going to implement our own custom "big input" component in shared module. Using `@angular/cli` we will execute command which will look something like `ng generate component shared/big-input --module shared.module`. This will generate component, template and style files. We will put our general layout and styling to the `big-input.component.scss` but we will also create new file `big-input.component.scss-theme.scss` where we will use style rules which have anything to do with color.

> *While* `<component-name>.component.scss-theme.scss` *is not official naming convention, I found it very helpful because IDE will put original styles file and theme file next to each other based on default alphabetical ordering.*

The custom component theme file then will look something like this…

```scss
@import '~@angular/material/theming';

// mixin name will be used in main style.scss
@mixin big-input-component-theme($theme) {

  // retrieve variables from theme
  // (all possible variables, use only what you really
  $primary: map-get($theme, primary);
  $accent: map-get($theme, accent);
  $warn: map-get($theme, accent);
  $foreground: map-get($theme, foreground);
  $background: map-get($theme, background);

  // all of these variables contain many additional va

  big-input {

    input {
```

Styling custom component using theme variables

> *In our mixin, we retrieved all the available theme variables to demonstrate what is possible but in real project we should only retrieve what is necessary...*

These variables then contain many sub-values based on the variable type. `$primary`, `$accent` and `$warn` variables contain reference to the whole palette as we defined in our custom theme definition in the beginning of article.

Retrieving color

We can retrieve particular colors using `mat-color` function. We can select default, lighter and darker shades (eg: `mat-color($primary, lighter)`) as defined by our theme or even any available color of the palette when necessary (`mat-color($primary, A200)`).

Contrast colors

It can be useful to retrieve appropriate contrast color when using main color as a background for some component which also contains text. In that case we can retrieve corresponding contrast color, eg: `mat-color($primary, lighter-contrast)` for backgrounds which `mat-color($primary, lighter)`.

Foreground and background color sub-values

While `$primary`, `$accent` and `$warn` variables contain same sub-values because they are created in a same way using `mat-palette` function, `$foreground` and `$background` are a bit different. We don't define them by hand but Angular Material sets their colors based on the function we use to create our custom theme. There are two available functions— `mat-light-theme` & `mat-dark-theme` and available sub-values are…

```scss
// Background palette for light themes.
$mat-light-theme-background: (
  status-bar: map_get($mat-grey, 300),
  app-bar:    map_get($mat-grey, 100),
  background: map_get($mat-grey, 50),
  hover:      rgba(black, 0.04),
  card:       white,
  dialog:     white,
  disabled-button: $black-12-opacity,
  raised-button: white,
  focused-button: $black-6-opacity,
  selected-button: map_get($mat-grey, 300),
  selected-disabled-button: map_get($mat-grey, 400),
  disabled-button-toggle: map_get($mat-grey, 200),
);
```

```scss
// Foreground palette for light themes.
$mat-light-theme-foreground: (
  base:              black,
  divider:           $black-12-opacity,
  dividers:          $black-12-opacity,
  disabled:          rgba(black, 0.38),
  disabled-button:   rgba(black, 0.38),
  disabled-text:     rgba(black, 0.38),
  hint-text:         rgba(black, 0.38),
  secondary-text:    rgba(black, 0.54),
  icon:              rgba(black, 0.54),
  icons:             rgba(black, 0.54),
  text:              rgba(black, 0.87)
);
```

Background and foreground Angular Material variables

Using custom component themes

To use our custom component theme we have to include it in main styles.scss file.

> *It is a good practice to create separate* `custom-components-theme` *mixin where we can collect all of our custom components so that we don't have to repeat all of them for every custom theme class*

```scss
@import '~@angular/material/theming';

@include mat-core();

@import 'my-theme.scss';

// import custom componenet themes
// unofficial naming convention to support nice orderi
// to see theme under the original style file of the c
@import 'app/shared/big-input/big-input.component.scss
@import 'app/shared/flip-toggle-button/flip-toggle-but

// you only have to add additional componenets here (i
@mixin custom-components-theme($theme) {
  @include big-input-theme($theme);
  @include flip-toggle-button-theme($theme);
```

Including custom component themes in main style.scss file

Defining custom palette

Angular material provides quite some palettes out of the box and many times it will be enough but sometimes we will need to use colors defined in company's design manual to achieve consistent branding of the product.

In that case we can simply create `custom-palettes.scss` file and create our own palette variables (eg: `$my-company-blue`) with all the sub-values which are contained in standard Angular Material palettes like 100, 200, contrast, …

Then we simply import this file in the custom theme definition file so that we can use the provided values.

That's it!

We learned how to theme our Angular Material applications! Check out the example **website**, **github repo** and don't forget to **recommend** and **share** this article if you liked it!

Also, feel free to check out other interesting front-end related posts like…

Angular Model Pattern

How to simply handle state in your Angular applications in standardized way without using 3rd party dependencies

medium.com

How does Angular teach you to be a better a Software Engineer

Frameworks like Angular implement many of the industry's best practices to tackle challenges faced by the frontend devs…

blog.ngconsultant.io

Follow me on Twitter to get notified about the **newest blog posts** and **useful** front-end stuff.

> And never forget, future is bright

Obviously the bright future