Stella Technology

Exchange › Coordinate › Collaborate

Publisher's Guide

Sana Khan

Corporate Headquarters
6203 San Ignacio Ave, Suite 100
San Jose, CA 95119

Middle East
Kingdom of Saudi Arabia
P.O.Box 305488 / Riyadh-13317

Asia Office
Plot # 408, Main Double Road,
I-9/3 Industrial Area, Islamabad

This document contains publisher's guidelines for all projects.

Web: www.stellatechnology.com
Info: info@stellatechnology.com

Contents

1 Getting started 2
2 Components and the component decorators 2
3 HTML in angular components 2
4 Angular Material: 2
4.1 Step 1: Install Angular Material and Angular CDK 2
4.2 Step 2: Animations 2
4.3 Step 3: Import the component modules 3
4.4 Step 4: Include a theme 3
4.5 Step 5 (Optional): Add Material Icons 3
5 Adding ngx-bootstrap 4
6 Styling file structure 4
6.1 Location of files 4
6.2 File division 5
6.3 SCSS format 5
6.3.1 Nesting 5
6.3.2 Partials 5
6.3.3 Import 6
6.3.4 Mixins 6
6.3.5 Extend/Inheritance 6
6.3.6 Operators 6
6.4 Class and ID naming conventions 6
7 View encapsulation 6
8 References 7

Welcome! This is the publisher guide for angular 4 project.

1 Getting started

In this guide we are going to cover the basic UI/Publishing rules for any angular project.

To work on the publishing side, you should know about the angular 4 HTML structure and the way it's CSS is working.

2 Components and the component decorators

In Angular, your views are defined within HTML templates. Templates are defined within the **@Component** decorator. You're able to define inline HTML templates as well as external templates within HTML files. Every component has its own styling file, in which we write the direct styling code.

3 HTML in angular components

HTML is the language of the Angular template. Almost all HTML syntax is valid template syntax. The <script>element is a notable exception; it is forbidden, eliminating the risk of script injection attacks. In practice, <script>is ignored and a warning appears in the browser console.

Some legal HTML doesn't make much sense in a template.
The <html>, <body>, and <base> elements have no useful role. Pretty much everything else is fair game.

You can extend the HTML vocabulary of your templates with components and directives that appear as new elements and attributes.

4 Angular Material:

Angular material is basically a default material design for angular components.

Follow these steps to begin using Angular Material.

4.1 Step 1: Install Angular Material and Angular CDK

```
npm install --save @angular/material @angular/cdk
```

4.2 Step 2: Animations

Some Material components depend on the Angular animations module in order to be able to do more advanced transitions. If you want these

animations to work in your app, you have to install the @angular/animations module and include the BrowserAnimationsModule in your app.

```
npm install --save @angular/animations
```

```
import {BrowserAnimationsModule} from '@angular/platform-browser/animations';

@NgModule({
  ...
  imports: [BrowserAnimationsModule],
  ...
})
export class PizzaPartyAppModule { }
```

4.3 Step 3: Import the component modules

Import the NgModule for each component you want to use:

```
import {MdButtonModule, MdCheckboxModule} from '@angular/material';

@NgModule({
  ...
  imports: [MdButtonModule, MdCheckboxModule],
  ...
})
export class PizzaPartyAppModule { }
```

4.4 Step 4: Include a theme

Including a theme is required to apply all of the core and theme styles to your application.

To get started with a prebuilt theme, include one of Angular Material's prebuilt themes globally in your application. If you're using the Angular CLI, you can add this to your styles.css:

```
@import "~@angular/material/prebuilt-themes/indigo-pink.css";
```

4.5 Step 5 (Optional): Add Material Icons

If you want to use the md-icon component with the official Material Design Icons, load the icon font in your index.html.

```
<link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
```

Note that md-icon supports any font or svg icons; using Material Icons is one of many options.

5 Adding ngx-bootstrap

This is the best way to quickly integrate Bootstrap 3 or Bootstrap 4 Components with Angular. ngx-bootstrap contains all core (and not only) Bootstrap components powered by Angular. So you don't need to include original JS components, but we are using markup and responsive css provided by Bootstrap.

- Install ngx-bootstrap and bootstrap

```
npm install ngx-bootstrap bootstrap --save
```

- Add required module to **src/app/app.module.ts**

```
import { AlertModule } from 'ngx-bootstrap';
...

@NgModule({
   ...
   imports: [AlertModule.forRoot(), ... ],
    ...
})
```

- insert a new entry into the styles array to **.angular-cli.json**

```
"styles": [
   "../node_modules/bootstrap/dist/css/bootstrap.min.css",
  "styles.css",
],
```

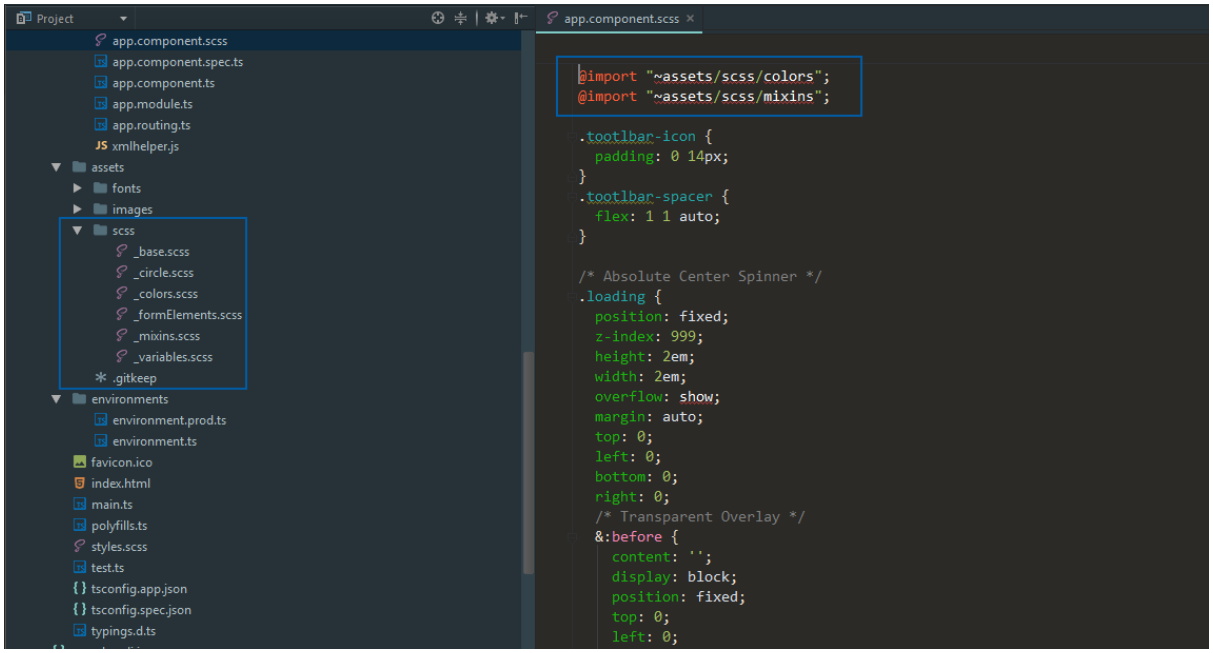
6 Styling file structure

6.1 Location of files

Style file of every component is located inside that component. You can create partial Sass files that contain little snippets of CSS that you can include in other Sass files and reuse that snippet in other components. This is a great way to modularize your CSS and help keep things easier to maintain. A partial is simply a Sass file named with a leading underscore. You might name it something like `_partial.scss`. The underscore lets Sass

know that the file is only a partial file and that it should not be generated into a CSS file. Sass partials are used with the `@import` directive. These partial Sass files should locate inside the assets folder.

The location of common partial files along with fonts and images should be inside the “assets” folder. In this case we can access the path directly.



6.2 File division

Style file division would be completely according to the project but there are few divisions which should be done for each project.

Color codes should be in a separate Sass partial file and there would be a unique variable name for each color.

Every form element (e.g input fields, buttons, select, checkboxes) should be in a separate Sass partial file.

Mixins and other variables should be in separate files, in order to reuse them again for other components.

6.3 SCSS format

We are going to use SCSS syntax.

6.3.1 Nesting

Sass will let you nest your CSS selectors in a way that follows the same visual hierarchy of your HTML.

6.3.2 Partials

You can create partial Sass files that contain little snippets of CSS that you can include in other Sass files. This is a great way to modularize your CSS and help keep things easier to maintain. A partial is simply a Sass file named with a leading underscore. You might name it something like `_partial.scss`.

6.3.3 Import

CSS has an import option that lets you split your CSS into smaller, more maintainable portions. The only drawback is that each time you use `@import` in CSS it creates another HTTP request. Sass builds on top of the current CSS `@import` but instead of requiring an HTTP request, Sass will take the file that you want to import and combine it with the file you're importing into so you can serve a single CSS file to the web browser.

6.3.4 Mixins

Some things in CSS are a bit tedious to write, especially with CSS3 and the many vendor prefixes that exist. A mixin lets you make groups of CSS declarations that you want to reuse throughout your site. You can even pass in values to make your mixin more flexible.

6.3.5 Extend/Inheritance

This is one of the most useful features of Sass. Using `@extend` lets you share a set of CSS properties from one selector to another. It helps keep your Sass very DRY.

6.3.6 Operators

Doing math in your CSS is very helpful. Sass has a handful of standard math operators like `+`, `-`, `*`, `/`, and `%`.

6.4 Class and ID naming conventions

The class and ID naming conventions we are going to follow is lower case having hyphen between two separate words.

7 View encapsulation

By default, Angular component styles are scoped to affect the component's view. This means that the styles you write will affect all the elements in your component template. They will not affect elements that are children of other components within your template. In this case Style rules don't leak out and page styles don't bleed in.

You can disable this encapsulation adding encapsulation: ViewEncapsulation.None to the @Component() decorator. After disabling the encapsulation the surrogate ID from each element will be removed and you will able to style accordingly.

Three types of values are available for encapsulation:

- encapsulation: ViewEncapsulation.None
- encapsulation: ViewEncapsulation.Emulated
- encapsulation: ViewEncapsulation.Native

These are some general publishing rules, which are going to implement on every angular 4 HTML.

8 References

- https://material.angular.io/guides
- http://sass-lang.com/guide