Angular Folder Structure

Sep 12, 2021

Getting Started

1 Directory Tree Sections **3**

2 Directory Tree Structure **5**
- 2.1 Structure Overview . . . 5
- 2.2 Media Directory . . . 6
- 2.3 Core Module . . . 6
- 2.4 Data Module . . . 6
- 2.5 Layout Directory . . . 8
- 2.6 Module Directory . . . 9
- 2.7 Shared Module . . . 11
- 2.8 Styles Directory . . . 11
- 2.9 Default Directory Structure . . . 13
- 2.10 Neolithic Directory Structure . . . 15
- 2.11 Path Alias . . . 16
- 2.12 Furthur Reading . . . 17
- 2.13 Demonstration Application . . . 18
- 2.14 Install Locally . . . 18

3 Contributing **21**

Based on best practices from the community, other github Angular projects, developer experience from production Angular projects, and contributors, this project goal is to create a skeleton structure which is flexible for projects big or small.

This project defines directory structure in an Angular application and it is a working application. The code for this project can be found at https://github.com/mathisGarberg/angular-folder-structure

CHAPTER 1

Directory Tree Sections

These are the sections this repository proposes be added to a default Angular application structure. Each section is optional

- media
- core
- data
- layout
- modules
- shared
- styles

Please read through the Directory Structure Parts carefully to understand them all.

CHAPTER 2

Directory Tree Structure

This repository suggests two different tree structures. You are free to design your own tree structure if these are not suitable. See the Directory Structures section for details.

2.1 Structure Overview

The angular-folder-structure project goal is to create a skeleton structure which is flexible for projects big or small.

The Angular style guide has this to say on the subject of directory structure:

> Have a near-term view of implementation and a long-term vision. Start small but keep in mind where the app is heading down the road.
>
> All of the app's code goes in a folder named src. All feature areas are in their own folder, with their own NgModule.
>
> —Angular - Style Guide

While such instructions are nice to hear they don't give real-world skeleton exprience. But we've taken this advice to heart to build our skeleton and document all the parts in line with the official documentation.

2.1.1 Creating an Application

Start your project using the `ng` command:

```
ng new
```

You will be prompted for the project name. Create a name in lower case with dashes between the words like `album-collection-organizer`

Next you will be asked to add **Angular Routing**. We recommend you select **Yes** (which is not the default).

Next you will be asked to select the **Style Sheet Format**. We recommend you select **SCSS**

`ng` will now create a default skeleton applicaiton and install your vendors.

2.1.2 Adding Structure

The rest of this documentaion covers new structure which is built on top of the `ng` generated skeleton. Every part of `angular-folder-structure` is optional so we suggest you review each part in this documentation to see if it is appropriate for your project.

This is documentation for angular-folder-structure. If you find this useful please add your star to the project.

2.2 Media Directory

~/media

The `media` directory is used to store supporting files for the application. Things like requirement documentation, text outlines, etc. This is the junk drawer for the project.

2.2.1 Install

```
mkdir media
```

This is documentation for angular-folder-structure. If you find this useful please add your star to the project.

2.3 Core Module

~/src/app/core

This module is for classes used by app.module. Resources which are always loaded such as route guards, HTTP interceptors, and application level services, such as the ThemeService and logging belong in this directory.

Note: This module is recommended for a path alias to **@app**

2.3.1 Install

```
ng generate module Core
```

This is documentation for angular-folder-structure. If you find this useful please add your star to the project.

2.4 Data Module

~/src/app/data

The data module is a top level directory and holds the types (models/entities) and services (repositories) for data consumed by the application.

By default there are two subdirectories:

```
~/src/app/data
  /types
  /service
```

The types directory holds the class definition files for data structures. An example data structure:

```
export class Project {
  link: string;
  title: string;
  thumbnail: string;
}
```

The service directory holds the services for fetching data. The service files are not necessarily a 1:1 match with types files. An example service file:

```
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';

import { Project } from '../types/project';
import { ApiService } from './api.service';

const routes = {
    projects: '/projects',
    project: (id: number) =>  `/projects/${id}`
};

@Injectable({
  providedIn: 'root'
})
export class ProjectService {
    constructor(
      private apiService: ApiService) {}

    getAll(): Observable<Array<Project>> {
        return this.apiService.get(routes.projects);
    }

    getSingle(id: number): Observable<Project> {
        return this.apiService.get(routes.project(id));
    }
}
```

2.4.1 Multiple Data Sources

If your application consumes data from more than one source then the data directory should be restructured to contain subdirectories for each data source. Do not create multiple modules for each data source:

```
~/src/app/data
  /data-source-one
    /types
    /service
  /data-source-two
    /types
    /service
  /data.module.ts
```

2.4.2 Schema Naming Standard

A type file is very much like an entity file in an Object Relational Mapper. This type file is central to your application's consumption of data and therefore does not need cursory decorators such as calling it *ProjectSchema* or *ProjectModel*. Schemas are special because they are the only plain-named class in the application.

2.4.3 Install

```
ng generate module Data
```

This is documentation for angular-folder-structure. If you find this useful please add your star to the project.

2.5 Layout Directory

~/src/app/layout

and/or

`~/modules/custom/layout`

The layout directory contains one or more components which act as a layout or are parts of a layout such as a Header, Nav, Footer, etc. and have a:

```
<router-outlet></router-outlet>
```

in the html for other components to embed within. By convention the `app.component.html` in the `app` module acts as the top level layout for the entire application. From this top level you may embed other layouts which in turn embed other components.

There are two schools of thought on layouts. The first is to put all your layouts for all your modules into the `app` module's layout directory. This consolidates all layouts in a single location.

The second school of thought is to put a layout directory into each custom module which has its own layout[s]. This approach groups the layout for components of the module into the module in which the components reside. This may apply to the `app` module if you wish to create a layout[s] which most appropriatly fits there.

Components like Nav and Footer are handled the Angular way by importing them into a component template:

```
<app-nav></app-nav>
```

2.5.1 Routing

Each module can have its own routing and is often defined in a file separate from the `custom.module.ts` file. See the `app` module's `app-routing.module.ts` as an example.

Within routing layouts are handled in a rather clever way. By using child routes a top level route can define a layout to be used for all child routes. Defined in the `app` module's `app-routing.module.ts` file child routes are grouped in a single Route (as defined in `@angular/router`).

It is important to note that when layouts are imported from other modules (for this example the `CustomModule`) that does not cause the top-level `app` module to load the `CustomModule` from the server into the user's browser at the time only `app` level routes are displayed. This is important for performance! The `CustomModule` compiled `.js` file will only load when a route internal to the module is requested. That is, from the below configuration, only when a route beginning with `/custom` is requested.

```
use { CustomLayoutComponent } from './modules/custom/layout/custom-layout.component';

{
  path: 'custom',
  component: CustomLayoutComponent,
  loadChildren: () =>
    import('./modules/custom/custom.module').then(m => m.CustomModule)
}
```

When a route is called at `/custom` the `CustomLayoutComponent` is used as a layout and handling of the routing is handed off to the CustomModule. The `CustomLayoutComponent` has a `<router-outlet></router-outlet>`:

```
<div [class]="theme">
  <div class="mat-app-background">
    <app-nav></app-nav>

    <div class="container">
      <router-outlet></router-outlet>
    </div>

    <app-footer (changeTheme)="onThemeChange($event)"></app-footer>
  </div>
</div>
```

This `<router-outlet></router-outlet>` is used to display a route and component defined in the routing of the `CustomRoutingModule`:

So the routes are `/custom` and `/custom/projects/:id` and they use the CustomLayoutComponent for their layout.

2.5.2 Install

```
mkdir src/app/layout
ng generate component layout/AppCustomLayout
ng generate component layout/Header
ng generate component layout/Footer
```

This is documentation for angular-folder-structure. If you find this useful please add your star to the project.

2.6 Module Directory

~/src/app/modules

The modules directory contains a collection of modules which are each independent of each other. This allows Angular to load only the module it requires to display the request thereby saving bandwidth and speeding the entire application.

In order to accomplish this each module must have its own routing which is a `loadChildren` route resource defined in the `AppRoutingModule`. This is also covered in the layout documentation

A route can have children and each child can have a loadChildren property. From `app-routing.module.ts`:

```
{
  path: '',
  component: ContentLayoutComponent,
  canActivate: [NoAuthGuard], // Should be replaced with actual auth guard
  children: [
  {
    path: 'dashboard',
    loadChildren: () =>
      import('./modules/home/home.module').then(m => m.HomeModule)
  },
  {
    path: 'about',
    loadChildren: () =>
      import('./modules/about/about.module').then(m => m.AboutModule)
  },
  {
    path: 'contact',
    loadChildren: () =>
      import('./modules/contact/contact.module').then(m => m.ContactModule)
  }
  ]
},
```

Each child must have its own base path from which it can load children from a module in the `modules` directory. Here is the routing for the About page:

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { AboutComponent } from './pages/about/about.component';

const routes: Routes = [
  {
    path: '',
    component: AboutComponent
  }
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class AboutRoutingModule { }
```

It is necessary to add the child routes to the RouterModule through `forChild`.

Besides routing any module inside the `modules` directory can be as simple or complicated as you wish.

2.6.1 Install

```
mkdir src/app/modules
```

For each new module run `ng generate module modules/NewModule`

This is documentation for angular-folder-structure. If you find this useful please add your star to the project.

2.7 Shared Module

~/src/app/shared

The shared module contains classes and resources which are used in more than one dynamically loaded module. By always loading with the application the shared components are ready whenever a module requests them.

The shared module is a good place to import and export the `FormsModule` and the `ReactiveFormsModule`. It is also good for the `FontAwesomeModule` and any other resource used by some modules some of the time but not all modules all of the time.

Note: This module is recommended for a path alias to **@shared**

2.7.1 Install

```
ng generate module Shared
```

This is documentation for angular-folder-structure. If you find this useful please add your star to the project.

2.8 Styles Directory

~/src/styles

The `~/src/styles` directory is used to store scss style sheets for the application. It can contain themes, Bootstrap, Angular Material, and any other styles.

~/src/styles.scss is installed in the default Angular skeleton. It should contain **@import** statements for all your global application scss files. For instance it can import theme files stored in the `~/src/styles` directory.

2.8.1 Themes

The `~/src/styles/themes` directory should contain the application-wide themes. This application includes two theme-files, *black-theme.scss* and *light-theme.scss*.

A theme file generates the color-palette that composes the final theme and is constructed of three main palettes: primary, accent and warn. These palettes are defined using the *mat-palette* mixin, which accepts a mat-color and a hue-number that represents different shades of the chosen color. In terms of code, this is what we have:

```
$my-black-primary: mat-palette($mat-grey, 700, 300, 900);
$my-black-accent: mat-palette($mat-blue-grey, 400);
$my-black-warn: mat-palette($mat-red, 500);

$my-black-theme: mat-dark-theme(
  $my-black-primary,
  $my-black-accent,
  $my-black-warn
);
```

The themes are included in the *styles.scss* file along with the *mat-core* mixin, which adds the base styles to material components.

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

@import './styles/themes/black-theme.scss';
@import './styles/themes/light-theme.scss';

@include mat-core();

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

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

The downside here is that the approach above only will style material components and not custom ones.

To achieve this, we've added a file called `project-container.component.scss-theme.scss`. This file imports the material theme and defines a mixin that styles the content with the appropriate color values - pulling color-palettes from the theme.

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

@mixin my-project-container-component-theme($theme) {
  $accent: map-get($theme, accent);

  .active {
    color: mat-color($accent, default-contrast);
    background-color: mat-color($accent);

    &:hover {
      color: mat-color($accent, default-contrast);
      background-color: mat-color($accent);
    }
  }
}
```

Then those files are referred to in the *styles.scss* files:

```
@import 'app/modules/home/page/project-item/project-container.component.scss-theme.
↪scss';

@mixin custom-components-theme($theme) {
  // ...
  @include my-project-container-component-theme($theme);
}

.my-light-theme {
  // ...
  @include custom-components-theme($my-light-theme);
}

.my-dark-theme {
  // ...
  @include custom-components-theme($my-black-theme);
}
```

The application content needs to be placed inside either a `mat-sidenav-container` element or have the

`mat-app-background` class applied to work. This application follows the last approach by appending this class to the `div` that wraps the app-content in the `src/app/layout/content-layout/content-layout.component.html` file:

```
<div class="my-dark-theme">
  <div class="mat-app-background">
    <app-nav></app-nav>

      <div class="container">
        <router-outlet></router-outlet>
      </div>

    <app-footer></app-footer>
  </div>
</div>
```

The height of the viewport the theme should affects is also defined:

```
.mat-app-background {
  height: 100%;
}
```

2.8.2 Bootstrap

The `~/src/styles` directory can be used for compiling bootstrap and storing other scss resources. To install a custom bootstrap download the source files, extract bootstrap into `~/src/styles/bootstrap`, then modify the `bootstrap/scss/_variables.scss`. Include boostrap in the `styles.scss`:

```
@import './styles/bootstrap/scss/bootstrap.scss';
```

2.8.3 Install

```
mkdir src/styles
```

This is documentation for angular-folder-structure. If you find this useful please add your star to the project.

2.9 Default Directory Structure

This is the directory structure this repository recommends. It is listed here in one place as a reference.

2.9.1 About

Inspired by the original blog post, this structure uses all the directory structure parts and is a strait-forward installation.

2.9.2 Tree Structure

```
.
├── e2e
│   └── src
├── media
└── src
    ├── app
    │   ├── core (@app)
    │   ├── data
    │   ├── layout
    │   ├── modules
    │   └── shared (@shared)
    ├── assets
    ├── environments (@env) [@env links to environment file]
    └── styles
        └── themes
```

2.9.3 Install

These instructions are to install this directory structure to a brand new `ng` **version 9 or below** created application:

```
mkdir media
ng generate module Core
ng generate module Shared
ng generate module Data
mkdir src/app/layout
mkdir src/app/modules
mkdir src/styles && mkdir src/styles/themes
json --version || sudo npm install -g json
json -f tsconfig.json -I -c "this.baseUrl = './'"
json -f tsconfig.json -I -c "this.compilerOptions.paths = {}"
json -f tsconfig.json -I \
  -e "this.compilerOptions.paths['@app/*'] = ['src/app/core/*']" \
  -e "this.compilerOptions.paths['@shared/*'] = ['src/app/shared/*']" \
  -e "this.compilerOptions.paths['@env'] = ['src/environments/environment']"
```

These instructions are to install this directory structure to a brand new `ng` **version 10 or above** created application. Before you can execute these instructions you must remove the comments from the `tsconfig.base.json` file because comments in a json file are not valid json:

```
mkdir media
ng generate module Core
ng generate module Shared
ng generate module Data
mkdir src/app/layout
mkdir src/app/modules
mkdir src/styles && mkdir src/styles/themes
json --version || sudo npm install -g json
json -f tsconfig.base.json -I -c "this.baseUrl = './'"
json -f tsconfig.base.json -I -c "this.compilerOptions.paths = {}"
json -f tsconfig.base.json -I \
  -e "this.compilerOptions.paths['@app/*'] = ['src/app/core/*']" \
  -e "this.compilerOptions.paths['@shared/*'] = ['src/app/shared/*']" \
  -e "this.compilerOptions.paths['@env'] = ['src/environments/environment']"
```

This is documentation for angular-folder-structure. If you find this useful please add your star to the project.

2.10 Neolithic Directory Structure

– Tom H Anderson <tom.h.anderson@gmail.com>

This is an alternative to the primary directory structure this repository promotes.

2.10.1 About

This directory structure corrects `ng` creation of all files inside the `app` directory. The default `ng` structure forces all code to exist as a subdirectory to `app` without giving a clear space for files which should exist beneath the `app` dir.

This directory structure moves `~/src/app` to `~/src/module/app` then creates a symlink from `~/src` to `~/src/app` (not shown and hidden from vscode) so `ng` will still place new files where they belong. By moving `app` to a subdirectory of the `module` directory it clears space for files which truly belong under the `app` module. This removes the requirement of a `core` module.

All modules exist in the same directory. This removes the special handling for the `app` module and flattens the modules making them all equally important.

2.10.2 Tree Structure

```
.
├── e2e
│   └── src
├── media
└── src
    ├── assets
    ├── environments (@env) [@env links to environment file]
    ├── module (@module)
    │   ├── app (@app)
    │   ├── [custom module 1]
    │   │   └── layout
    │   ├── [custom module 2]
    │   │   └── layout
    │   ├── [more custom modules...]
    │   ├── data (@data)
    │   └── shared (@shared)
    └── styles
        └── themes
```

2.10.3 Install

These instructions are to install this directory structure to a brand new `ng` **version 9 or below** created application:

```
mkdir media
mkdir src/module
mkdir src/styles && mkdir src/styles/themes
mv src/app src/module
cd src && ln -s . app & cd .
sed -i .bak 's/.\/app/.\/module\/app/g' src/main.ts
ng generate module module/Shared
ng generate module module/Data
json --version || sudo npm install -g json
```

(continues on next page)

(continued from previous page)

```
json -f tsconfig.json -I -c "this.baseUrl = './'"
json -f tsconfig.json -I -c "this.compilerOptions.paths = {}"
json -f tsconfig.json -I \
  -e "this.compilerOptions.paths['@app/*'] = ['src/module/app/*']" \
  -e "this.compilerOptions.paths['@shared/*'] = ['src/module/shared/*']" \
  -e "this.compilerOptions.paths['@module/*'] = ['src/module/*']" \
  -e "this.compilerOptions.paths['@env'] = ['src/environments/environment']" \
  -e "this.compilerOptions.paths['@data/*'] = ['src/module/data/*']"
mkdir -p .vscode
test -f .vscode/settings.json || echo "{}" > .vscode/settings.json
json -f .vscode/settings.json -I -e "this['files.exclude'] = {'**src/app': true}"
```

These instructions are to install this directory structure to a brand new `ng` **version 10 or above** created application. Before you can execute these instructions you must remove the comments from the `tsconfig.base.json` file because comments in a json file are not valid json:

```
mkdir media
mkdir src/module
mkdir src/styles && mkdir src/styles/themes
mv src/app src/module
mkdir src/module/app/layout
cd src && ln -s . app & cd .
sed -i .bak 's/.\/app/.\/module\/app/g' src/main.ts
ng generate module module/Shared
ng generate module module/Data
json --version || sudo npm install -g json
json -f tsconfig.base.json -I -c "this.baseUrl = './'"
json -f tsconfig.base.json -I -c "this.compilerOptions.paths = {}"
json -f tsconfig.base.json -I \
  -e "this.compilerOptions.paths['@app/*'] = ['src/module/app/*']" \
  -e "this.compilerOptions.paths['@shared/*'] = ['src/module/shared/*']" \
  -e "this.compilerOptions.paths['@module/*'] = ['src/module/*']" \
  -e "this.compilerOptions.paths['@env'] = ['src/environments/environment']" \
  -e "this.compilerOptions.paths['@data/*'] = ['src/module/data/*']"
mkdir -p .vscode
test -f .vscode/settings.json || echo "{}" > .vscode/settings.json
json -f .vscode/settings.json -I -e "this['files.exclude'] = {'**src/app': true}"
```

This is documentation for angular-folder-structure. If you find this useful please add your star to the project.

2.11 Path Alias

Path aliases simplify paths by giving a link to the path rather than using the the fully qualified path name. Using them will make your code easier to read and maintain.

Path aliases are relative to `compilerOptions.baseUrl`. By default this is set to `"./"` so all path aliases we create must be fully qualified from the `baseUrl`.

2.11.1 Create Aliases

This documentation uses a json program to modify the `tsconfig.json` file to simplify editing for everyone.

Run this command to install the json program:

```
npm install -g json
```

To create an alias run this command from your root application directory:

```
json -f tsconfig.json -I -e "this.compilerOptions.paths['@app/*'] = ['src/app/core/*']
↪"
```

The `@app` is the alias. The path, `src/app/core/*` in this example, is the path from the root `compilerOptions.baseUrl` directory to the directory you would like to alias.

For json to be able to add paths to your tsconfig.json "paths" has to exist under compilerOptions. Else you get an error when running the commands to add specific paths. "paths" can be added like this:

> json -f tsconfig.json -I -e "this.compilerOptions['paths'] = {}"

2.11.2 Recommended Aliases

Recommended are aliases to core, shared, modules and `environment`. These aliases will simplify your development:

```
json -f tsconfig.json -I -e "this.compilerOptions.paths['@app/*'] = ['src/app/core/*']
↪"
json -f tsconfig.json -I -e "this.compilerOptions.paths['@shared/*'] = ['src/app/
↪shared/*']"
json -f tsconfig.json -I -e "this.compilerOptions.paths['@modules/*'] = ['src/app/
↪modules/*']"
json -f tsconfig.json -I -e "this.compilerOptions.paths['@env'] = ['src/environments/
↪environment']"
```

Note the alias for @env goes directly to the environment file.

2.11.3 Using Aliases

When you have aliases defined such as `@shared` you can shortcut your `use` statements by using the alias:

```
import { SharedModule } from '../../shared/shared.module';
```

becomes:

```
import { SharedModule } from '@shared/shared.module';
```

The environment `@env` alias to a file is used this way:

```
import { environment } from '@env';
```

This is documentation for angular-folder-structure. If you find this useful please add your star to the project.

2.12 Furthur Reading

- Read the original blog post which started this module: How to define a highly scalable folder structure for your Angular project

- Path Aliasing is a popular way to make your import statements more tidy and is used in this demonstration application.
- Some Best Practices which follow the advice here closely.

2.12.1 Alternative Directory Structure Projects

These projects have similar goals to this project and understanding them will give a wider view of directory structure: the problem and solutions.

- **angular-starter** This is a very popular starter kit with lots of tools built in. It is designed to be used when starting a new application. Common directory parts include **~/src/styles**.
- **$ngx ROCKET** Extensible Angular 8+ enterprise-grade project generator based on angular-cli with best practices from the community. Includes PWA, Cordova & Electron support, coding guides and more!
- **angular-ngrx-material-starter** Angular, NgRx, Angular CLI & Angular Material Starter Project
- **Angular-Full-Stack** Angular Full Stack project built using Angular 2+, Express, Mongoose and Node. Whole stack in TypeScript.

This is documentation for angular-folder-structure. If you find this useful please add your star to the project.

2.13 Demonstration Application

See this application running on our demonstation application

This is documentation for angular-folder-structure. If you find this useful please add your star to the project.

2.14 Install Locally

You can install this application locally and run it. Assuming you already have `typescript`, `npm`, and `ng` installed, clone this repository, cd to the directory and run `npm install`:

```
git clone https://github.com/mathisGarberg/angular-folder-structure.git
cd angular-folder-structure
npm install
```

Included with this package are some custom npm scripts. Here is a list of npm run commands and their descriptions:

```
npm start      -> Run dev. server on http://localhost:4200/
npm run build -> Lint code and build app for production in dist folder
npm run test   -> Run unit tests via Karma in watch mode
npm test:ci    -> Lint code and run unit tests once for continuous integration
npm run e2e    -> Run e2e tests using Protractor
npm run lint   -> Lint code
```

2.14.1 Run Locally

To run the application type `ng serve` then browse to http://localhost:4200/

This is documentation for angular-folder-structure. If you find this useful please add your star to the project.

CHAPTER 3

Contributing

We welcome contributions of all kinds to this repository. Submit a PR or create an issue for anything you can help us improve.

For documentation contributions we recommend you install restructuredText into your copy of vscode so you can lint and preview your work before it is submitted.

This is documentation for angular-folder-structure. If you find this useful please add your star to the project.