Angular File Structure and Best Practices (that help to scale)

Shijin Nath · Follow
6 min read · Apr 25, 2020

Introduction

When working in a large team with many developers that are responsible for the same codebase, having a common understanding of how the application should be structured is vital. The time spent arguing over architectural decisions takes up a

large portion of our day to day operations, and having a common understanding comes a long way in solving them.

The goal of this article is to help those developers by proposing a scalable and maintainable structure for medium to large applications. We'll explore the structure in the context of the Angular documentation by using their statement to further highlight the advantages here.

Note! There's almost impossible to find a structure that suits every single use-case. The structure of an application will change a lot depending on the project, and there's no blueprint here. This article aims to improve the development process by ***proposing*** *a structure that should work well for medium to large applications.*

The Angular Structure

```
∇ app
    ∇ core
        ∇ guards
            auth.guard.ts
            module-import.guard.ts
            no-auth.guard.ts
        ∇ interceptor
            token.interceptor.ts
            error.interceptor.ts
        ∇ services
            service-a.service.ts
            service-b.service.ts
        ∇ components
            ∇ navbar
                navbar.component.html|scss|ts
            ∇ page-not-found
                page-not-found.component.html|scss|ts
        ∇ constants
            constant-a.ts
            constant-b.ts
        ∇ enums
            enum-a.ts
            enum-b.ts
        ∇ models
            model-a.ts
            model-b.ts
        ∇ utils
            common-functions.ts
    ∇ features
        ∇ feature-a
            ∇ components
                ∇ scoped-shared-component-a
                        scoped-shared-component-a.component.html|scss|ts
                ∇ scope-shared-component-b
                        scoped-shared-component-b.component.html|scss|ts
            ∇ pages
                ∇ page-a
                    page-a.component.html|scss|ts
                ∇ page-b
                    page-b.component.html|scss|ts
            ∇ models
                scoped-model-a.model.ts
                scoped-model-b.model.ts
            ∇ services
                scoped-service-a.service.ts
                scoped-service-b.service.ts
            feature-a-routing.module.ts
            feature-a.module.ts
```

```
                feature-a.component.html|scss|ts
    ∇ shared
        ∇ components
            ∇ shared-button
                shared-button.component.html|scss|ts
        ∇ directives
            shared-directive.ts
        ∇ pipes
            shared-pipe.ts
        shared.module.ts
    styles.scss
    ▽ styles
        app-loading.scss
        company-colors.scss
        spinners.scss
        variables.scss
    ▽ assets
        ▽ i18n
            lang-a.json
            lang-b.json
        ▽ images
            image-a.svg
            image-b.svg
        ▽ static
            structure-a.json
            structure-b.json
        ▽ icons
            custom-icon-a.svg
            custom-icon-b.svg
```

> 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`.
>
> Do structure the app such that you can Locate code quickly, Identify the code at a glance, keep the Flattest structure you can, and Try to be DRY, but not on the expense of readability.
>
> Do name the file such that you instantly know what it contains and represents.
>
> Do keep a flat folder structure as long as possible.

The `ng new` command from the Angular CLI provides us with an initial skeleton structure at the root level of the application and is easy to run and build on top of.

This default behavior is suitable for new applications, and is the perfect entry point for the structure. You should use the Angular schematics to generate the files in the appropriate areas.

The App Module

> Do create an NgModule in the app's root folder, for example, in `/src/app`.

In Angular, everything is organized in modules, and every application have at least one of them, the app root module. The app module is the entry point of the application, and is the module that Angular uses to bootstrap the application. The setup instructions when creating a new application produces a minimal `AppModule` with a single component. You'll evolve this module as the application grows.

The Core Module

```
∇ core
        ∇ guards
            auth.guard.ts
            module-import.guard.ts
            no-auth.guard.ts
        ∇ interceptor
            token.interceptor.ts
            error.interceptor.ts
        ∇ services
            service-a.service.ts
            service-b.service.ts
        ∇ components
            ∇ navbar
                navbar.component.html|scss|ts
            ∇ page-not-found
                page-not-found.component.html|scss|ts
        ∇ constants
            constant-a.ts
            constant-b.ts
        ∇ enums
            enum-a.ts
            enum-b.ts
        ∇ models
            model-a.ts
            model-b.ts
        ∇ utils
            common-functions.ts
```

angular-core-module hosted with ❤ by GitHub view raw

The `CoreModule` takes on the role of the app root module, but is not the module that gets bootstrapped by Angular at run-time. The common denominator between the files present here is that we only need to load them once, and that is at run-time, which makes them singleton. The module contains root-scoped services, static components like the navbar and footer, interceptors, guard, constants, enums, utils, and universal models. To prevent re-importing the module elsewhere, we should add a module-import-guard in it's constructor method.

The Shared Module

```
∇ shared
        ∇ components
            ∇ shared-button
                shared-button.component.html|scss|ts
        ∇ directives
            shared-directive.ts
        ∇ pipes
            shared-pipe.ts
        shared.module.ts
```

angular-shared-module hosted with ❤ by GitHub view raw

> Do create a feature module named `SharedModule` in a `shared` folder; for example, `app/shared/shared.module.ts` defines `SharedModule`.
>
> Do declare components, directives, and pipes in a shared module when those items will be re-used and referenced by the components declared in other feature

> Do export all symbols from the `SharedModule` that other feature modules need to use.

When working on large applications, the Angular team suggest us to consider lazy loading of our modules. This decreases the bundle size of our application and therefore the initial build-time, and this is where the SharedModule truly shines.

The SharedModule allows us to organize and streamline our code. The shared module shouldn't have any dependency to the rest of the application, and should therefore not rely on any other module. It should contain all the reusable modules,

lazy loaded feature modules required to operate. You should add commonly used directives, pipes and components here. Many third-party libraries are available as NgModules such as Material Design, and exposing them through the SharedModule might be a good idea.

We can easily publish and share these components between applications.

The Feature Modules

```
∇ features
          ∇ feature-a
               ∇ components
                     ∇ scoped-component-a
                              scoped-component-a.component.html|scss|ts
                     ∇ scope-component-b
                              scoped-component-b.component.html|scss|ts
               ∇ pages
                    ∇ page-a
                         page-a.component.html|scss|ts
                    ∇ page-b
                         page-b.component.html|scss|ts
               ∇ models
                     scoped-model-a.model.ts
                     scoped-model-b.model.ts
               ∇ services
                     scoped-service-a.service.ts
                     scoped-service-b.service.ts
               feature-a-routing.module.ts
               feature-a.module.ts
               feature-a.component.html|scss|ts
```

angular-feature-modules hosted with ❤ by GitHub view raw

> Do create an NgModule for all distinct features in an application; for example, a `Heroes` feature. All feature areas are in their own folder, with their own NgModule.
>
> Do place the feature module in the same named folder as the feature area; for example, in `app/heroes`.
>
> Do name the feature module file reflecting the name of the feature area and folder; for example, `app/heroes/heroes.module.ts`.
>
> Do name the feature module symbol reflecting the name of the feature area, folder, and file; for example, `app/heroes/heroes.module.ts` defines `HeroesModule`.

The initial Angular application does only have one single module, which works great for small applications. But as the application grows, you'll need to consider subdividing it into multiple feature modules, some which can be lazy loaded. These modules should only depend on the SharedModule, and their functionality should be scoped to the module.

Feature modules deliver user experience dedicated to a particular application feature like the user- or the administration-part of the app. We're grouping the components, services, models and other functionality that belongs together. They typically have a top component that acts as the feature root and private, supporting sub-components descend from it. They might be imported by the root `AppModule` of a small application that lacks routing or need to show some initial content, but can also be lazy loaded with references in the app routing file.

Domain feature modules rarely have providers, but when they do, the lifetime of the provided services should be the same as the lifetime of the module. Beginning with Angular 6.0, the preferred way pf creating a singleton is to set `providedIn` to `root` on the service' `@Injectable` decorator. This tells Angular to provide the service in the application root. But we can also use this

to create a singleton service is to set `providedIn` to `root` on the service's `@Injectable()` decorator. This tells Angular to provide the service in the application root. We can use this in the context of a feature by using the `providedIn` property on the module instead, resulting in an error when using it elsewhere.

When you need the service in other modules as well, it probably belongs in the CoreModule' service's declaration instead.

Styles

In a similar way to how we want to avoid bloating up the `AppModule` as the application grows, it's also true for the `styles.scss` file.You should instead create a styles folder, which contains mixins or css-functions, responsible for their own areas. These files are then imported in the appropriate order inside the styles.scss file, providing their global styles to the rest of the app. Create mixins for reusable css-snippets, and scope the associated logic together.

Assets

The assets folder is generated for us by the Angular CLI with the `ng new` command, and is the perfect place for storing all our media files. Using it in combination with a PathLocationStrategy gives easily referable files across the app. This folder persists at build time.

Going Beyond!

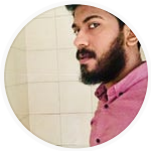
Even though the structure described above solves a lot of problems, we've still left out one big area, and that is state management in Angular. The reason for this is that there's so many different ways to solve this exact problem, and one solution might not apply to another. The context of Angular Material and other component libraries are another area that will define large areas of the application structure. The context

of The emergence of Nrwl with micro-frontends have also gained hype. Would you like to see this kind of article of some point, please let me know.

Summary

Choosing an appropriate folder structure is not an easy task. You need to agree with the team on the structure that suits the application, and what might suit one need, might not suit another. Should hopefully be valid for the next years to come. What would you do differently to ensure a good, maintainable structure?

Don’t forget to clap , share and follow !!



Follow

Written by Shijin Nath

75 Followers

I’m full-stack developer interested in web trends, science and technology. Cloud-native and open source enthusiast. http://shijinnath-portfolio.herokuapp.com/

More from Shijin Nath

Shijin Nath

Typescript Rest API with Express.js, MySQL and TypeORM

Today, we are going to use TypeScript Express.js and TypeORM to create an enterprise level Rest API. The objective is to create a...

11 min read · Apr 26, 2020

217

Shijin Nath

Difference between Concurrency and Parallelism

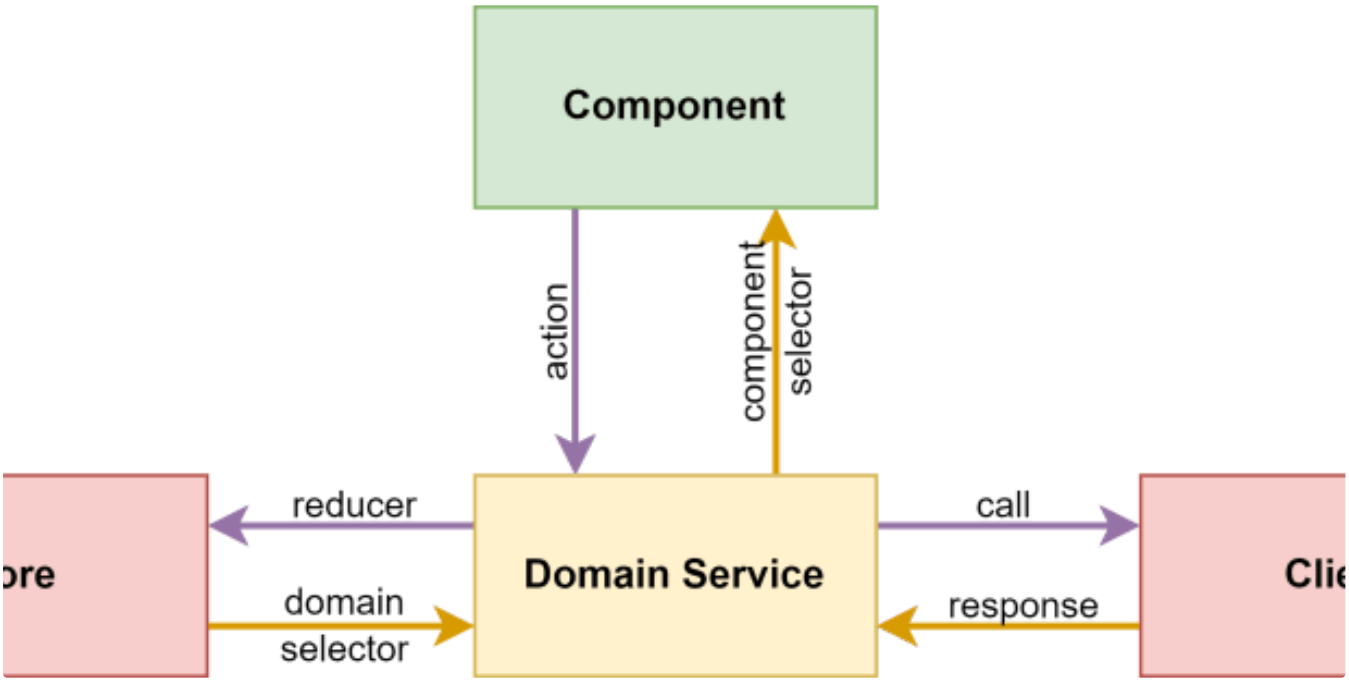
Concurrency: Concurrency relates to an application that is making progress more than one task at the same time. Concurrency is a approach...

3 min read · Apr 26, 2020

63

See all from Shijin Nath

Recommended from Medium



Jérôme Navez

A simple and clean architecture for your Angular projects

Conceptually, this architecture could be applied to any frontend framework. Since I have more knowledge of Angular than the other frontend...

7 min read · Jun 29

133 6

Mirza Leka

10 Angular Dos & Don’ts

Improve the development, modularity, and performance of your Angular apps.

10 min read · Oct 23

325 3

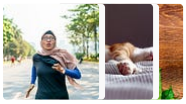
Lists

Staff Picks

516 stories · 471 saves

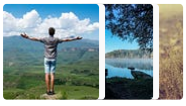
Stories to Help You Level-Up at Work

19 stories · 327 saves



Self-Improvement 101

20 stories · 955 saves



Productivity 101

20 stories · 864 saves

Igor Martins

State management with NgRx in Angular

Modern front-end applications commonly use component concepts, which represent independent, reusable, and autonomous parts of a user...

9 min read · Aug 2

18

Jaydeep Patil

Observable and Subjects in Angular

In this article, we are going to discuss the basics of observable and subject. Also, the difference between them is explained with the help...

5 min read · Aug 4

147 2

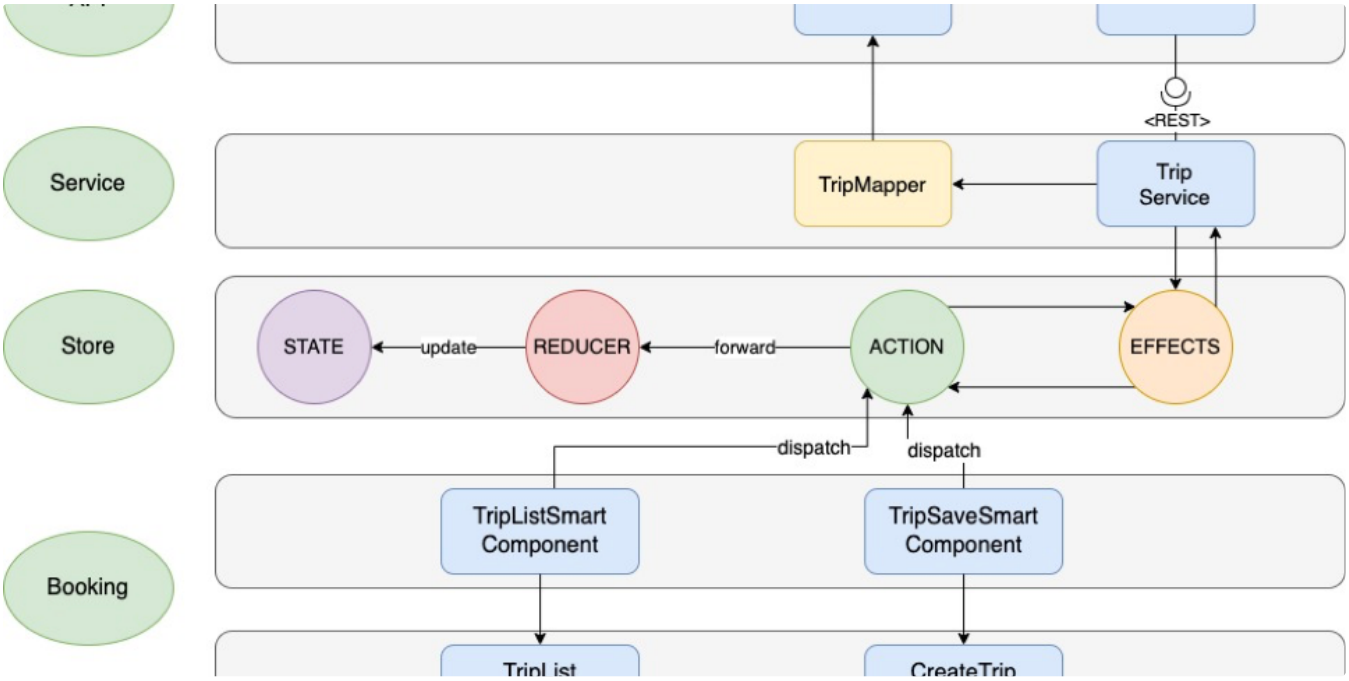
Preston Lamb in ngconf

Functional CanDeactivate Guards in Angular

CanDeactivate guards in Angular can be very helpful in your app. Preventing users from leaving a route can keep the user from losing their p

4 min read · Aug 10

76



Robert Maier-Silldorff in Bits and Pieces

Clean Frontend Architecture

An overview of some of the principles associated with a clean frontend architecture (SOLID, KISS, DRY, and more).

6 min read · Jul 14

1.9K 17

See more recommendations