Angular Js

Things to follow

V.1.0

By, Ruchit Suthar

Contents

Angular App Folder Structure

Below is the sample folder structure should be followed.

```
|-- app
        |-- assets
                |-- css
                |-- js
                |-- images
        |-- core
                |-- services
                        |-- http-client.service.ts | spec
        |-- layout
                |-- dashboard
                |-- header
                |-- sidebar
                |-- footer
        |-- pages
                |-- home
                        |-- home.component.ts | html | scss | spec
                        |-- home.module.ts
                        |-- home.routing.ts
                        |-- home.service.ts
                |-- contact
                        |-- contact-list
                                |-- contact-list.component.ts | html | scss | spec
                        |-- contact-add
                                |-- contact-add.component.ts | html | scss | spec
                        |-- contact-detail
                                |-- contact-detail.component.ts | html | scss | spec
                        |-- contact.module.ts
                        |-- contact.service.ts
        |-- shared
                |-- components
                        |-- multi-select.component.ts | html | scss | spec
                |-- directives
                        |-- drag-and-drop.directive.ts | html | scss | spec
                |-- pipes
                        |-- date.pipe.ts | html | scss
                |-- shared.module.ts
```

File name should be in lower case. Use - (dash) between words.

If you are creating component then add .component suffix in file name. Same like this add .service, .module, .directive, .pipe suffix for their respective file name.

app

This will be the root folder of the application. All the components, directives, pipes, services, module, assets should be residing in this folder.

assets

This folder contains sub folder for css, js and images. All the css, js, images should be maintained in their respective folder.

core

This folder contains core services or module. For an example, authentication, http interceptor, http client, or other common services will be maintained in this folder.

layout

This folder contains layout related components. Like if application has dashboard kind of layout or CRM, it commonly has dashboard, header, sidebar. So, all the separate component for each section should be created and maintained in this folder.

pages

This folder contains all the component of each page. Let say home page. There should be a folder named as ‘home’ and it contains home component, module, service, routing, html, css files.

If page itself is a big module then, divide module in sub pages like, let say there is a contact module which contains add, view, edit, list functionality then,

Create separate folder for each item like list component, add component, detail component as illustrated in image.

shared

Generic components, directive or pipe which are used in multiple page component then add those in this module.

Declare all the shared item in shared.module.ts file. Import this SharedModule in each module file of the page.

Angular App Naming Conventions

Class

Class name should be in **Pascal** case (the first letter is capitalized). i.e. Contact, ContactDetail

All class properties should be in pascal case.

```
export class SystemThemeItem {
    ThemeItemId: number | 0;
    SystemThemeId: number | 0;
    Name: string | undefined;
    Code: string | undefined;
    DataType: string | undefined;
    Value: string | undefined;
    CommunityId: number | 0;
    CommunityTypeId: number | 0;
}
```

Function

Function name should be in **camel** case (the first letter is lower). i.e. getContact(), deleteContactById()

```
getUserMember() {
    this.isLoading = true;
    this.memberService.getUserMember(this.memberFilter).subscribe(
      data => {
       // Some logic here
      },
      error => {
      // Error handling logic would be here
      },
      () => {
        this.isLoading = false;
      }
    );
  }
```

Item	Case	Example
Class	Pascal	Contact, ContactDetail
Class Properties	Pascal	FullName, Email, Phone
Function	Camel	getContact()
Global variable	Camel	isLoading
Private variable	Camel	isValid
Local variable	Camel	name
Service object	Camel	contactService
Constant	_ + Pascal	_ContactCode

Angular CLI Useful Commands

Use of Angular CLI

Install node.js. download latest stable version of node from its official website.

Install Angular CLI globally. For that open command prompt and run below command.

```
npm install -g @angular/cli
```

Create new project

Run below command in the folder or directory where you want to create a project.

```
ng new my-app
```

Run app

Run below command to run the app.

```
ng serve

OR

ng s --port 4500 (if want to run on specific port 4500)
```

by default, the app will be run on http://localhost:4200.

Build app

Run below command to build the app.

```
ng build
```

Run below command for production build.

```
ng build --prod --build-optimizer --aot
```

Run below command to build with specific environment configuration.

```
ng build --configuration=beta --build-optimizer --aot
```

Run below command to create new component.

```
ng g c component-name
```

Run below command to create new service.

```
ng g s service-name
```

Run below command to create new directive.

```
ng g d directive-name
```

Run below command to create new pipe.

```
ng g p pipe-name
```

URL to follow https://angular.io/cli

Angular App Best Practices

Single Responsibility Principle

Do not declare more than one component, service, directive, pipe etc. inside a single file. Each item should have its separate file.

Variable and function names

Name the variable and functions so the name itself describe the functionality and other developer can understand it by the name itself.

Give the proper name to the function. Try to name it like the name itself describe the functionality written in that function.

Avoid below kind of function and variable naming.

```
function div(x, y)) {
  const val = x / y;
  return val;
}
```

Hopefully more like this:

```
function divide(divident, divisor) {
  const quotient = divident / divisor;
  return quotient;
}
```

Check of blank, null or undefine value

Most of the developers currently performing this check in below way.

```
let firstName: stirng;
if (firstName !== '' || firstName !== undefined || firstName !== null) {
      // firstName has some value which is not blank, null or undefined.
}
```

Below if the optimized way.

```
let firstName: stirng;
if (firstName) {
      // firstName has some value which is not blank, null or undefined.
}
```

Write small and pure functions

When you write functions to execute some business logic, you should keep them smell and clean. It is easier to test and maintain. When you start noticing you function is getting long, it's good idea to abstract some of the logic in separate new function.

Avoid functions like this:

```
addOrUpdateData(data: Data, status: boolean) {
  if (status) {
    return this.http.post<Data>(url, data)
      .pipe(this.catchHttpErrors());
  }
  return this.http.put<Data>(`${url}/${data.id}`, data)
    .pipe(this.catchHttpErrors());
  }
}
```

Hopefully more like this:

```
addData(data: Data) {
  return this.http.post<Data>(url, data)
    .pipe(this.catchHttpErrors());
}
updateData(data: Data) {
  return this.http.put<Data>(`${url}/${data.id}`, data)
    .pipe(this.catchHttpErrors());
}
```

Variable declaration

Avoid taking too much global variable declaration. Rather create a class that contains meaningful properties and user the class object.

Use interface to create a Type of the object. Mostly used when you need to map the data return from API to the local object.

Boolean variable name should be always declared with "is" prefix. i.e. isLoading, isSuccess.

Event functions name should be always declared with "on" prefix. i.e. onClick(), onChange().

Use of Interfaces

Using interfaces is perfect way of describing our object literals. If our object is of any interface type, it is necessary to implement all the interfaces properties.

Interface name must be start with “I” prefix. i.e. IUser, IContact

This is more important when you are deserializing data coming from services to the local object.

Optional properties of an interface can be declared with “?”.

```
export interface IUser{
  firstName: string;
  lastName: string;
  phone?: string;
}
```

User of Constructor

We should use constructor method to create object of dependencies like service, pipe etc. Do not write any business logic in constructor.

```
constructor(private commonService: CommonService,
    private router: Router,
    private memberService: MembersService,
    private notificationService: NotificationService,
    private dragulaService: DragulaService,
    private errorMessage: ErrorMessage,
    private resolver: ComponentFactoryResolver,
    private contactService: ContactService,
    private propertyService: PropertyService,
    private sharedService: SharedService,
    private shareModalService: ShareModalService,
    private manageMyAccountService: ManageMyAccountService,
    private membersService: MembersService,
    private staticContentService: StaticContentService,
    private fb: FormBuilder,
    private cdr: ChangeDetectorRef,
  ) { }
```

Use of ngOnInit

Use this method for initialization of class, properties or methods to be called on load of the component.

```
ngOnInit() {
    DocumentReady.TabInit();
    DocumentReady.RightSidebarInit();
    this.getThemeItemByCommunityTypeId();
    this.initializeObject();
    this.getStaticContent();
    this.getContactSelectOptions();
    this.setUserSubscriptionFeature();
    this.onScrollEnd();
    this.onScrollEndPendingRequest();
    this.subscribeDrag();
    this.showHelpModal();
    this.subscribeNotification();
  }
```

Safe Navigation Operator (?) in HTML Template

To be on the safe side we should always use the safe navigation operator while accessing a property from an object in a component's template.

If the object is null and we try to access a property, we are going to get an exception. But if we use the save navigation (?) operator, the template will ignore the null value and will access the property once the object is not the null anymore.

```
<div class="col-md-3">
  {{user?.name}}
</div>
```

Module Organization and Lazy Loading

Even though an Angular application is going to work just fine if we create just one module, the recommendation is to split our application into multi-modules.

There are a lot of advantages to this approach. The project structure is better organized, it is more maintainable, readable and reusable and we are able to use the lazy-loading feature.

The best practice is to use separate routing module.

```
const appRoutes: Route[] = [
  { path: 'home', component: HomeComponent },
  { path: '404', component: NotFoundComponent }
]

@NgModule({
  imports: [
    CommonModule,
    RouterModule.forRoot(appRoutes)
  ],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

then register new routing module in app module.

Lazy Loading Module

Implementation of lazy loaded module is recommended. The advantage of lazy loading module that we can load module on demand not on when app load for the first time.

Lazy loaded module will be loaded when its URL requested.