

Common Framework

—

Angular CLI

Table of Contents

What Is Angular CLI?	3
Prerequisites.....	3
Installation	4
Update Angular CLI.....	7
Adjusting Ports (Optional)	7
Application Structure	8
Adding Features	9
Create a Component.....	9
Create a Service.....	11
Adding External Styles.....	12
Adding External Scripts.....	13
Routing	13
Navigation.....	15
Environments.....	16
Tests	17
References / Bibliography	17

What Is Angular CLI?

Angular CLI, a complete development toolset, is a Command Line Interface (CLI) to automate your development workflow. It allows you to:

- create a new Angular application
- run a development server with Live Reload support to preview your application during development
- add features to your existing Angular application
- run your application's unit tests
- run your application's end-to-end (E2E) tests
- build your application for deployment to production

Prerequisites

Both the CLI and generated project have dependencies that require

- **Node 6.9.0** or higher. (Download Link: [Official Node.js Website](https://nodejs.org/en/))
- **NPM 3** or higher.

(npm is distributed with Node.js which means that when you download Node.js, you automatically get npm installed on your computer.)

Verify that Node.js and npm has been installed, you can verify their version by running:

```
$ node -v # => displays your Node.js version
```

```
$ npm -v # => displays your npm version
```



```
Command Prompt
C:\>node -v
v6.11.0

C:\>npm -v
3.10.10
```

Once you have Node.js installed, you can use the `npm` command to install [TypeScript](https://www.typescriptlang.org/).

```
$ npm install -g typescript
```

Although TypeScript is technically not an absolute requirement, it is highly recommended by the Angular team, so I recommend you install it to make working with Angular as comfortable as possible.

Now that you have **Node.js** and **TypeScript** installed, you can install **Angular CLI**.

Installation

BEFORE YOU INSTALL: Please read the [prerequisites](#).

- 1- Open Command Prompt and run the following command to install angular CLI globally on your system.

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

To verify whether your installation completed successfully, you can run:

```
$ ng version
```

which displays the version you have installed:



```
Command Prompt
C:\>ng version

Angular CLI
@angular/cli: 1.4.2
node: 6.11.0
os: win32 ia32
```

- 2- Navigate to the folder in command prompt where you want to create the new angular project.



```
Command Prompt
C:\>E:
E:\>
```

3- Create a new app using the `ng` command.

```
ng new PROJECT_NAME --style=scss
```

```
E:\>ng new commonFrameworkUI --style=scss
create commonFrameworkUI/e2e/app.e2e-spec.ts (301 bytes)
create commonFrameworkUI/e2e/app.po.ts (208 bytes)
create commonFrameworkUI/e2e/tsconfig.e2e.json (235 bytes)
create commonFrameworkUI/karma.conf.js (923 bytes)
create commonFrameworkUI/package.json (1331 bytes)
create commonFrameworkUI/protractor.conf.js (722 bytes)
create commonFrameworkUI/README.md (1115 bytes)
create commonFrameworkUI/tsconfig.json (363 bytes)
create commonFrameworkUI/tslint.json (3040 bytes)
create commonFrameworkUI/.angular-cli.json (1139 bytes)
create commonFrameworkUI/.editorconfig (245 bytes)
create commonFrameworkUI/.gitignore (516 bytes)
create commonFrameworkUI/src/assets/.gitkeep (0 bytes)
create commonFrameworkUI/src/environments/environment.prod.ts (51 bytes)
create commonFrameworkUI/src/environments/environment.ts (387 bytes)
create commonFrameworkUI/src/favicon.ico (5430 bytes)
create commonFrameworkUI/src/index.html (304 bytes)
create commonFrameworkUI/src/main.ts (370 bytes)
create commonFrameworkUI/src/polyfills.ts (2480 bytes)
create commonFrameworkUI/src/styles.scss (80 bytes)
create commonFrameworkUI/src/test.ts (1085 bytes)
create commonFrameworkUI/src/tsconfig.app.json (211 bytes)
create commonFrameworkUI/src/tsconfig.spec.json (304 bytes)
create commonFrameworkUI/src/typings.d.ts (104 bytes)
create commonFrameworkUI/src/app/app.module.ts (314 bytes)
create commonFrameworkUI/src/app/app.component.html (1075 bytes)
create commonFrameworkUI/src/app/app.component.spec.ts (986 bytes)
create commonFrameworkUI/src/app/app.component.ts (208 bytes)
create commonFrameworkUI/src/app/app.component.scss (0 bytes)
Installing packages for tooling via npm.
Installed packages for tooling via npm.
Successfully initialized git.
Project 'commonFrameworkUI' successfully created.
```

Note: You can always update the style Extension (e.g. from `css` to `scss` etc.) from the `"angular-cli.json"` file, which is found in the project directory.

<pre>"defaults": { "styleExt": "css", "component": {} }</pre>	to	<pre>"defaults": { "styleExt": "scss", "component": {} }</pre>
---	----	--

"commonFrameworkUI" has been created in your directory with all the specified files.

4- Navigate to your project using the `cd` command.

```
cd PROJECT_NAME
```

```
E:\>cd commonFrameworkUI  
E:\commonFrameworkUI>
```

5- Run `ng` command to run the application.

```
ng serve
```

```
E:\commonFrameworkUI>ng serve  
** NG live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **  
Date: 2017-09-18T15:53:31.821Z  
Hash: dc2b9764fe3a6da77fd5  
Time: 7071ms  
chunk {inline} inline.bundle.js, inline.bundle.js.map (inline) 5.83 kB [entry] [rendered]  
chunk {main} main.bundle.js, main.bundle.js.map (main) 8.63 kB {vendor} [initial] [rendered]  
chunk {polyfills} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 209 kB {inline} [initial] [rendered]  
chunk {styles} styles.bundle.js, styles.bundle.js.map (styles) 11.6 kB {inline} [initial] [rendered]  
chunk {vendor} vendor.bundle.js, vendor.bundle.js.map (vendor) 2.29 MB [initial] [rendered]  
webpack: Compiled successfully.
```

We now have a working application that can be accessed at <http://localhost:4200/>.



Welcome to app!



Here are some links to help you start:

- [Tour of Heroes](#)
- [CLI Documentation](#)
- [Angular blog](#)

Update Angular CLI

To update Angular CLI to a new version, you must update both the global package and your project's local package.

Global package:

```
npm uninstall -g @angular/cli
npm cache clean
npm install -g @angular/cli@latest
```

Local project package:

```
rm -rf node_modules dist
# use rmdir /S/Q node_modules dist in Windows Command Prompt;
# use rm -r -fo node_modules,dist in Windows PowerShell
npm install --save-dev @angular/cli@latest
npm install
```

Adjusting Ports (Optional)

To change the ports, run command

To adjust the **HTTP port** and **host address** use the command line options below, which would change your URL to <http://0.0.0.0:8080/>.

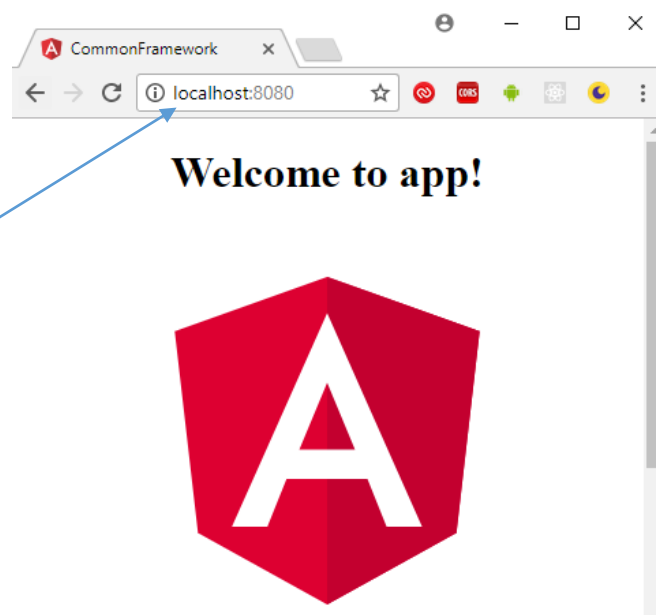
```
ng serve --port 8080 --host 0.0.0.0
```

To run a project on the different port, one way is to specify the port while you run ng serve command.

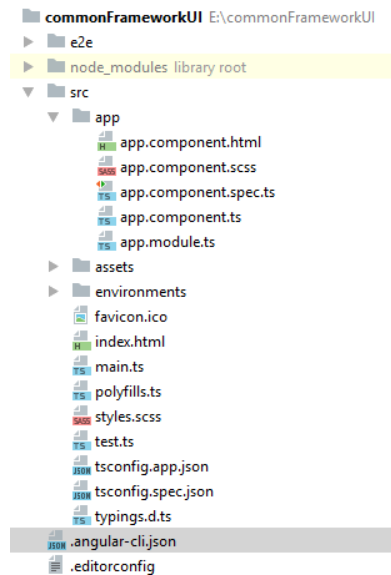
```
ng serve --port PORT_NUMBER
```

or add the port in the "angular-cli.json" file.

```
"defaults": {
  "styleExt": "scss",
  "component": {},
  "serve": {
    "port": 8080
  }
}
```



Application Structure



Here is the basic application structure you should have by now.

- 1- Create the following folders in the *"commonFrameworkUI/src/app"* directory. They will allow us to maintain and manage our application.

- Models
- Components
- Services
- Directives
- Pipes
- Scss

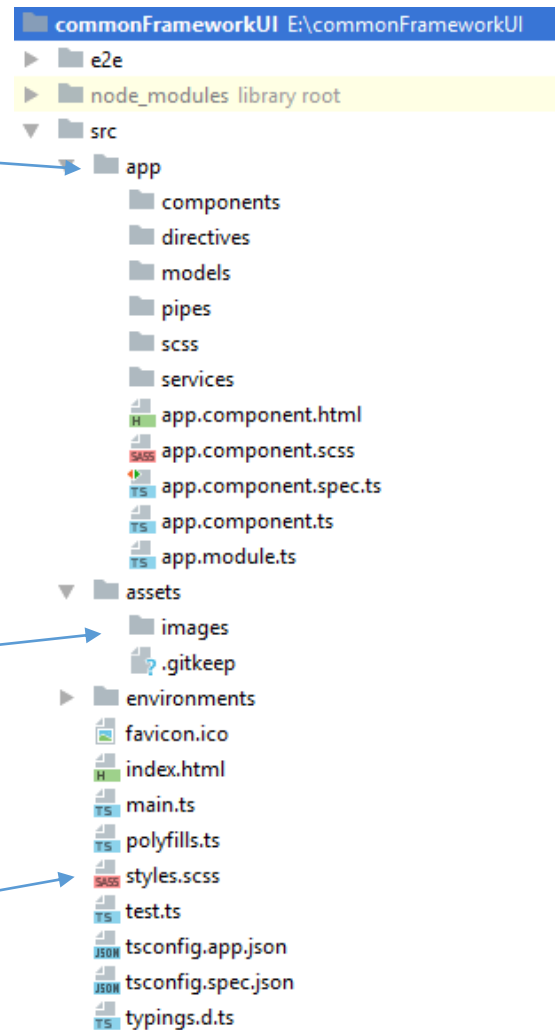
The app folder contains all the source code for the applications.

- 2- Create the following folder in the *"commonFrameworkUI/src/assets"* directory.

- images

All images used in the application are placed in this folder

- 3- Notice the *"styles.scss"* file. It contains the global styles. Component related styles can be defined in the scss file in specific component folder.



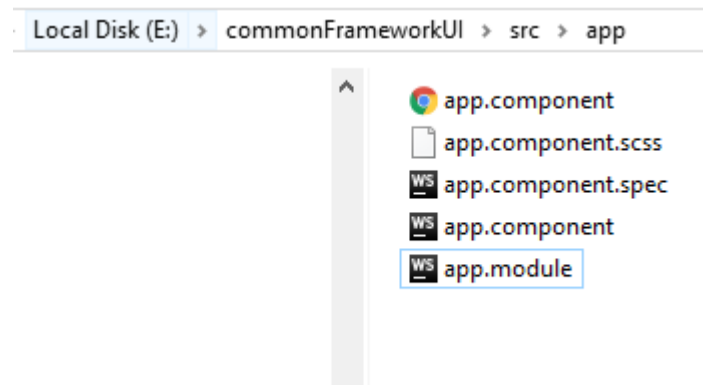
Adding Features

You can use the `ng generate` command to add features to your existing application:

- `ng generate class my-new-class`: add a class to your application
- `ng generate component my-new-component`: add a component to your application
- `ng generate directive my-new-directive`: add a directive to your application
- `ng generate enum my-new-enum`: add an enum to your application
- `ng generate module my-new-module`: add a module to your application
- `ng generate pipe my-new-pipe`: add a pipe to your application
- `ng generate service my-new-service`: add a service to your application

Create a Component

This basic app comes with a single component (AppComponent) that can be found at `/src/app/app.component.ts`.



- 1- Navigate to the "components directory" in "*ProjectName/src/app*" through command line.

```
E:\>cd commonFrameworkUI/src/app/components  
E:\commonFrameworkUI\src\app\components>
```

- 2- Add another component using the `ng generate` command.

```
ng generate component COMPONENT_NAME
```

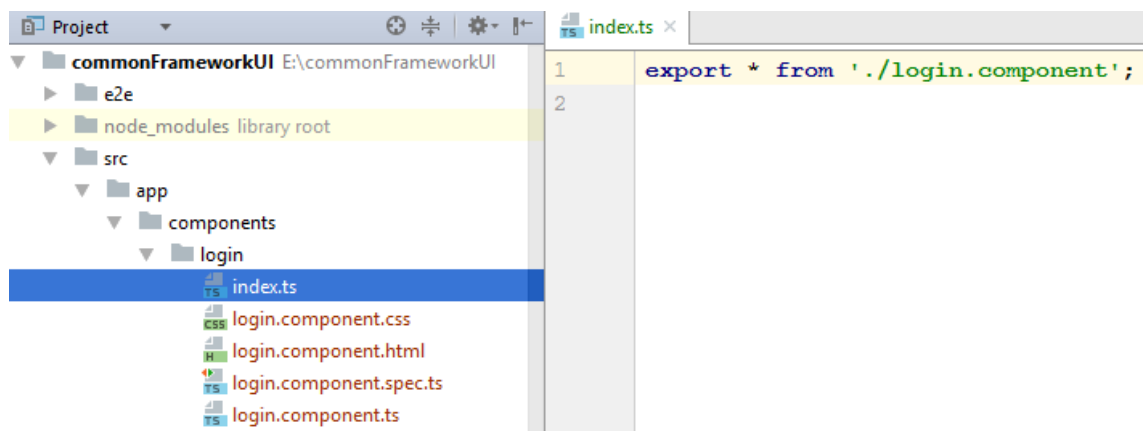
or shortcut notation

```
ng g c COMPONENT_NAME
```

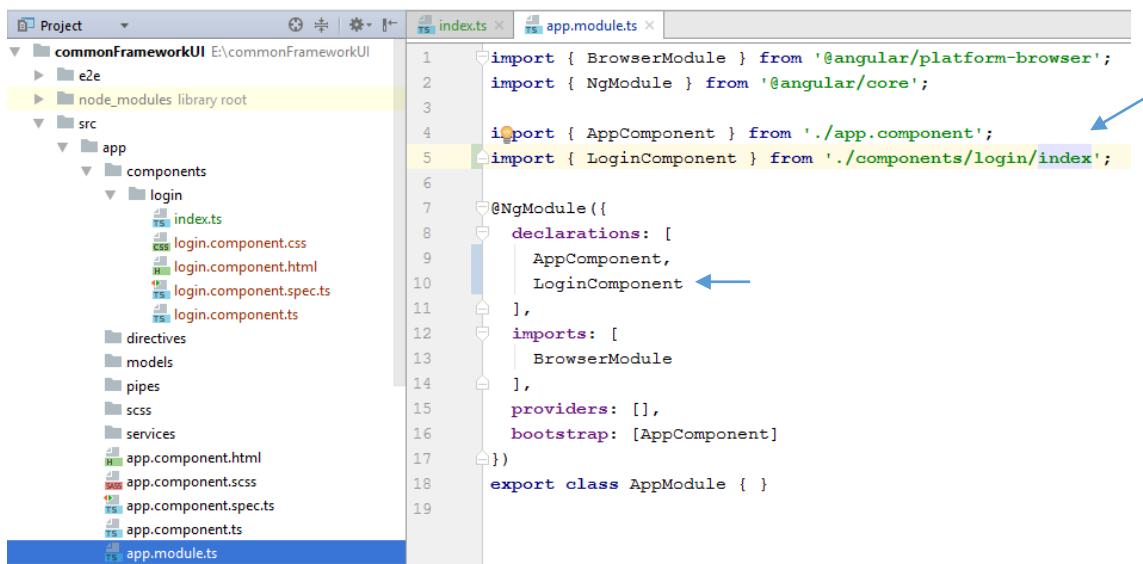
Anytime we create a new component using the Angular CLI we end up with four (4) new files. We now have the below files under the `/src/app/new-test/` directory.

```
E:\commonFrameworkUI\src\app\components>ng generate component login
create src/app/components/login/login.component.html (24 bytes)
create src/app/components/login/login.component.spec.ts (621 bytes)
create src/app/components/login/login.component.ts (265 bytes)
create src/app/components/login/login.component.css (0 bytes)
update src/app/app.module.ts (403 bytes)
```

- 3- Create a new TypeScript file called "index.ts" inside the created component directory and write command to export all the elements from that specific component directory.



- 4- Notice that the `ng generate` command has already declared and imported the component in our "app module" and "`@NgModule`". Update the import statement from "`component/component-name-directory/ component-name`" to "`component /component-name-directory/index`".



Note: Directives are created the same way as components

Create a Service

- 1- Navigate to the "service directory" in "ProjectName/src/app" through command line.

Command Prompt

```
E:\commonFrameworkUI\src\app>cd services  
E:\commonFrameworkUI\src\app\services>
```

- 2- Add another component using the `ng generate` command.

```
ng generate service SERVICE_NAME
```

or shortcut notation

```
ng g s SERVICE_NAME
```

Anytime we create a new service using the Angular CLI we end up with four (2) new files. We now have the below files under the `/src/app/services/` directory.

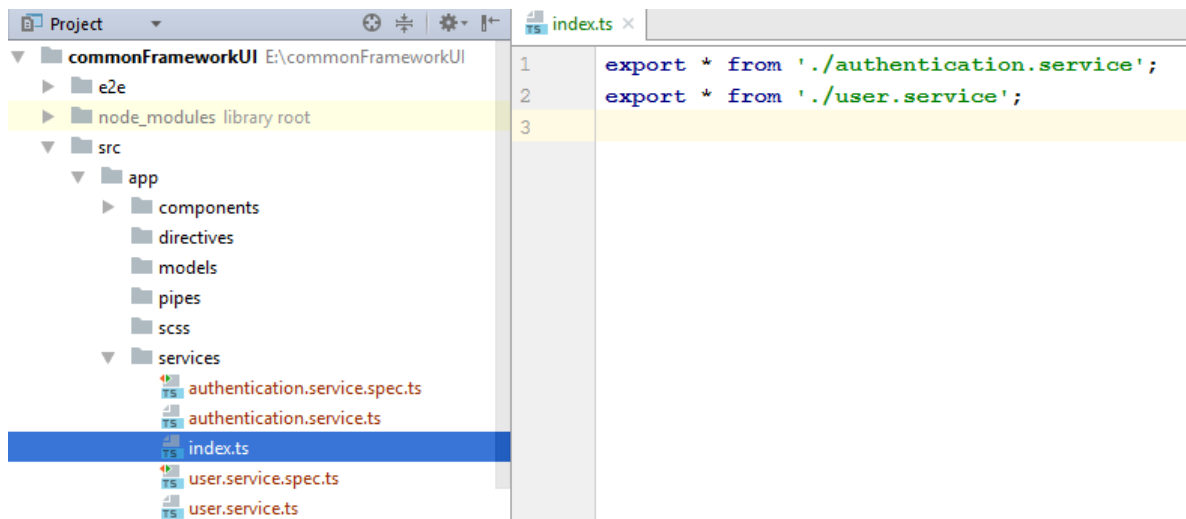
Command Prompt

```
E:\commonFrameworkUI\src\app\services>ng generate service authentication  
create src/app/services/authentication.service.spec.ts (422 bytes)  
create src/app/services/authentication.service.ts (120 bytes)
```

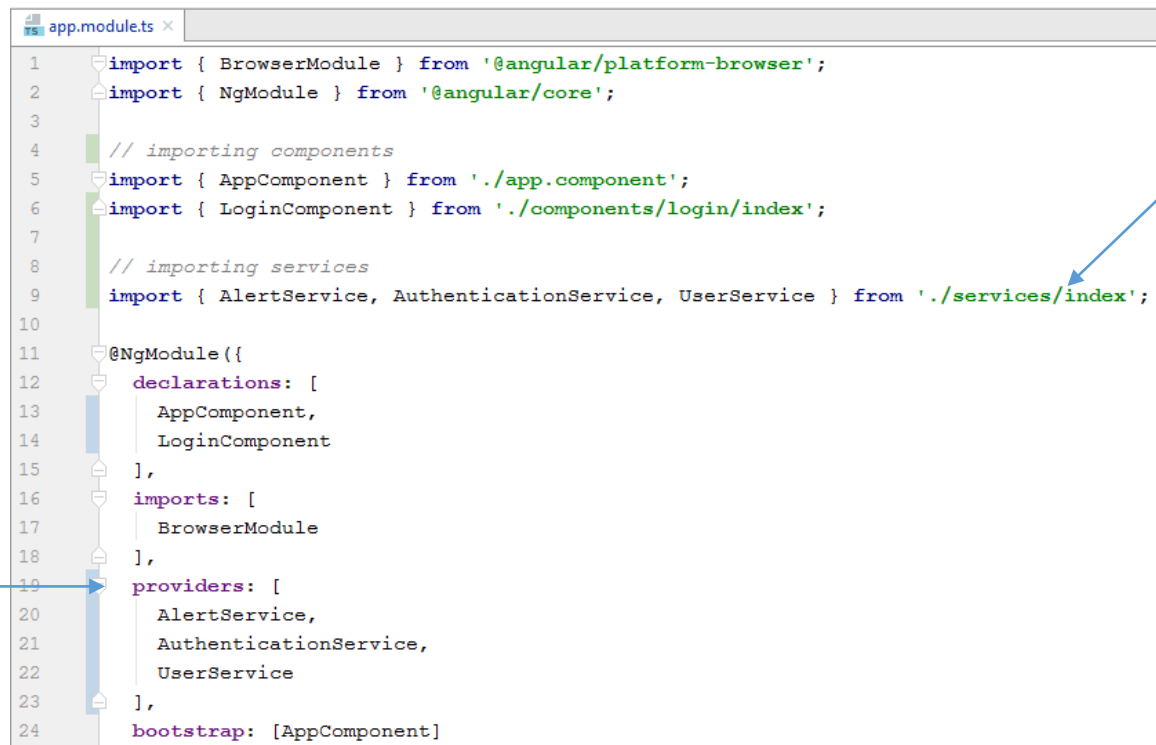
Note: Inorder to create a file without the `spec.ts` file, use command:

```
ng generate service SERVICE_NAME --spec= false
```

- 3- Create an index file in the "services" folder and export all the services in the file. I have created two services just to demonstrate the use of "index.ts" file.



- 4- Import the service in "app module" and inform the application that a service has been included in the application. Add the service name in "providers" of the "@ngModule".

A screenshot of a code editor showing the file 'app.module.ts'. The code is as follows:

```
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4 // importing components
5 import { AppComponent } from './app.component';
6 import { LoginComponent } from './components/login/index';
7
8 // importing services
9 import { AlertService, AuthenticationService, UserService } from './services/index';
10
11 @NgModule({
12   declarations: [
13     AppComponent,
14     LoginComponent
15   ],
16   imports: [
17     BrowserModule
18   ],
19   providers: [
20     AlertService,
21     AuthenticationService,
22     UserService
23   ],
24   bootstrap: [AppComponent]
```

Two blue arrows are present: one points from the right margin to the import statement on line 9, and another points from the left margin to the 'providers' array on line 19.

(Note 1: Angular CLI will not add the services to the "app.module.ts". You have to do it manually.)

(Note 2: Pipes are created the same way as services)

Adding External Styles

In order to use third party styles to your application. Install it through node and provide its path in the "styles" tag in the .angular-cli.json file located in the Project directory commonFrameworkUI/angular-cli.json.

```
"apps": [
  {
    "root": "src",
    "outDir": "dist",
    "assets": [
      "assets",
      "favicon.ico"
    ],
    "index": "index.html",
    "main": "main.ts",
    "polyfills": "polyfills.ts",
    "test": "test.ts",
    "tsconfig": "tsconfig.app.json",
    "testTsconfig": "tsconfig.spec.json",
    "prefix": "app",
    "styles": [
      "../node_modules/font-awesome/css/font-awesome.css",
      "../node_modules/datatables.net-dt/css/jquery.dataTables.css",
      "../node_modules/bootstrap/dist/css/bootstrap.min.css",
      "styles.scss"
    ]
  },
]
```

Adding External Scripts

Add external scripts to be available throughout the application include it in the “script” tag of the `.angular-cli.json` file.

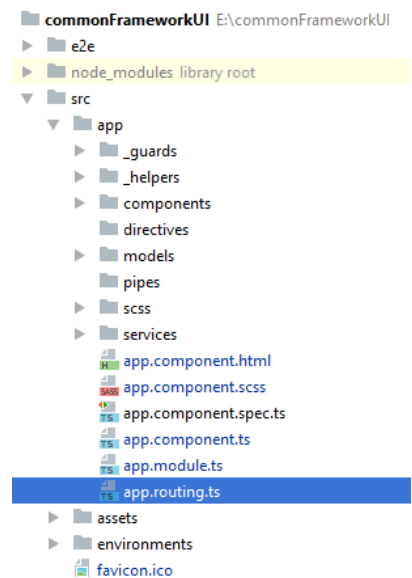
```
"apps": [  
  {  
    "root": "src",  
    "outDir": "dist",  
    "assets": [  
      "assets",  
      "favicon.ico"  
    ],  
    "index": "index.html",  
    "main": "main.ts",  
    "polyfills": "polyfills.ts",  
    "test": "test.ts",  
    "tsconfig": "tsconfig.app.json",  
    "testTsconfig": "tsconfig.spec.json",  
    "prefix": "app",  
    "styles": [  
      "../node_modules/font-awesome/css/font-awesome.css",  
      "../node_modules/datatables.net-dt/css/jquery.dataTables.css",  
      "../node_modules/bootstrap/dist/css/bootstrap.min.css",  
      "styles.scss"  
    ],  
    "scripts": [  
      "../node_modules/jquery/dist/jquery.js",  
      "../node_modules/datatables.net/js/jquery.dataTables.js"  
    ],  
  },  
],
```

Routing

- 1- Create an `app.routing.ts` file in the “app” directory (refer to figure on right side).
- 2- Import “RouterModules” and “Routes” from the angular library package.

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

```
app.routing.ts  
1 import { Routes, RouterModule } from '@angular/router';  
2
```



3- Import components in the "app.routing.ts" file.

```
app.routing.ts
1 import { Routes, RouterModule } from '@angular/router';
2
3 import { HomeComponent } from './components/home/index';
4 import { LoginComponent } from './components/login/index';
5 import { RegisterComponent } from './components/register/index';
6 import { AuthGuard } from './_guards/index';
7
```

4- Create an array of type Routes and define the corresponding path and component for the navigation. "canActivate" (optional) authenticates user before allowing entry to in the application.

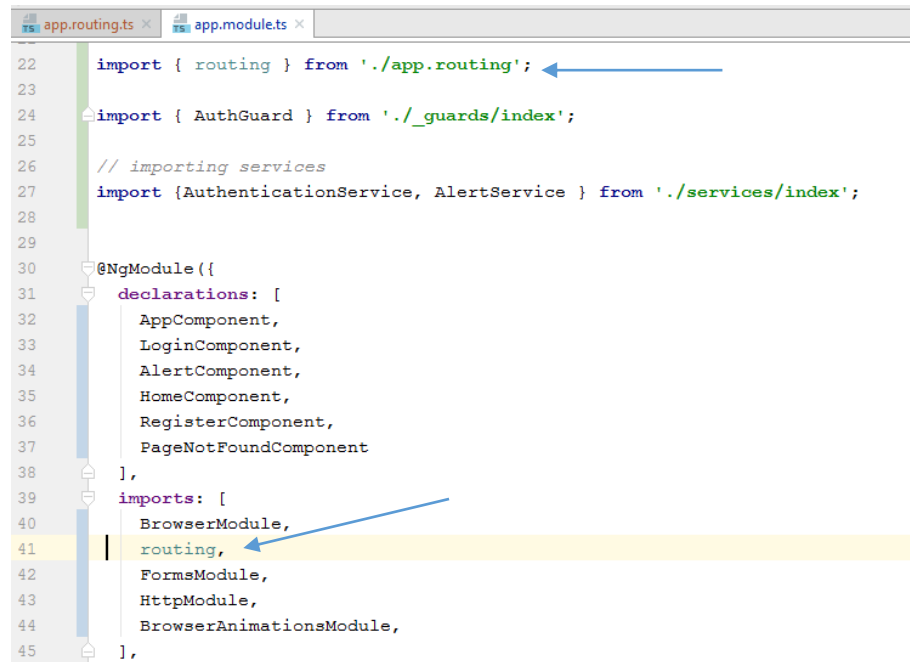
```
7 import { AuthGuard } from './_guards/index';
8
9 const appRoutes: Routes = [
10   { path: '', redirectTo: '/login', pathMatch: 'full' },
11   { path: 'home', component: HomeComponent, canActivate: [AuthGuard] },
12   { path: 'register', component: RegisterComponent },
13   { path: 'login', component: LoginComponent },
14   // otherwise redirect to Page Not Found Page
15   { path: '**', component: PageNotFoundComponent }
16 ];
```

A routed Angular application has one singleton instance of the [Router](#) service. When the browser's URL changes, that router looks for a corresponding [Route](#) from which it can determine the component to display.

5- Configure the router through the "RouterModule.forRoot" method and export it.

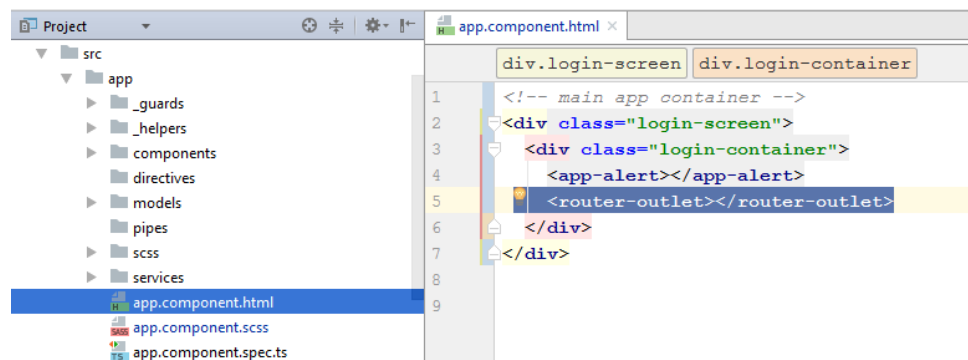
```
app.routing.ts  app.modules.ts
1 import { Routes, RouterModule } from '@angular/router';
2
3 import { HomeComponent } from './components/home/index';
4 import { LoginComponent } from './components/login/index';
5 import { RegisterComponent } from './components/register/index';
6 import { PageNotFoundComponent } from './components/page-not-found/index';
7 import { AuthGuard } from './_guards/index';
8
9 const appRoutes: Routes = [
10   { path: '', redirectTo: '/login', pathMatch: 'full' },
11   { path: 'home', component: HomeComponent, canActivate: [AuthGuard] },
12   { path: 'register', component: RegisterComponent },
13   { path: 'login', component: LoginComponent },
14   // otherwise redirect to Page Not Found Page
15   { path: '**', component: PageNotFoundComponent }
16 ];
17
18 export const routing = RouterModule.forRoot(appRoutes);
```

- 6- Import the exported routing information from the app.routing.ts to the app.module.ts file.



```
22 import { routing } from './app.routing';  
23  
24 import { AuthGuard } from './_guards/index';  
25  
26 // importing services  
27 import { AuthenticationService, AlertService } from './services/index';  
28  
29  
30 @NgModule({  
31   declarations: [  
32     AppComponent,  
33     LoginComponent,  
34     AlertComponent,  
35     HomeComponent,  
36     RegisterComponent,  
37     PageNotFoundComponent  
38   ],  
39   imports: [  
40     BrowserModule,  
41     routing,  
42     FormsModule,  
43     HttpClientModule,  
44     BrowserAnimationsModule,  
45   ],
```

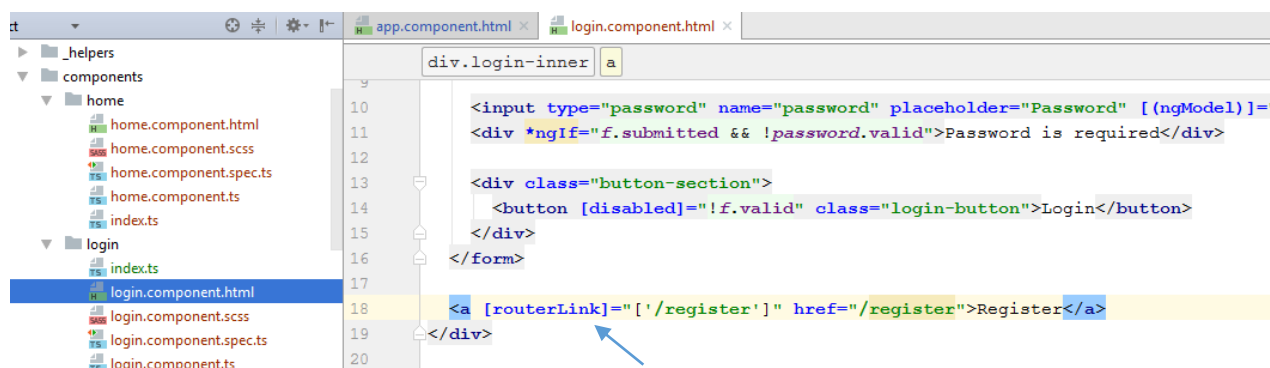
- 7- Router outlet is used to load the URLs from the routing modules.



```
1 <!-- main app container -->  
2 <div class="login-screen">  
3   <div class="login-container">  
4     <app-alert></app-alert>  
5     <router-outlet></router-outlet>  
6   </div>  
7 </div>  
8  
9
```

Navigation

Navigate to a route within the template using "routerLink" directive.



```
10 <input type="password" name="password" placeholder="Password" [(ngModel)]=  
11 <div *ngIf="f.submitted && !password.valid">Password is required</div>  
12  
13 <div class="button-section">  
14   <button [disabled]="!f.valid" class="login-button">Login</button>  
15 </div>  
16 </form>  
17  
18 <a [routerLink]="['/register']" href="/register">Register</a>  
19 </div>  
20
```

Environments

Environments let you specify settings to customize your application behavior.

You can define your own environments in the `.angular-cli.json` file. The default ones are:

- `source`: use settings defined in `environments/environment.ts`
- `dev`: use settings defined in `environments/environment.ts`
- `prod`: use settings defined in `environments/environment.prod.ts`

where `environments/environment.ts` equals:

```
export const environment = {  
  production: false };
```

and `environments/environment.prod.ts` equals:

```
export const environment = {  
  production: true };
```

The build process will use the `dev` environment by default.

If you specify a different environment, the build process will use the corresponding environment:

```
# Uses environments/environment.ts  
$ ng build  
  
# Also uses environments/environment.ts  
$ ng build --environment=dev  
  
# Uses environments/environment.prod.ts  
$ ng build --environment=prod
```

As you can see in `src/main.ts`, you can access the environment settings from your code by importing `environments/environment`:

```
import { enableProdMode } from '@angular/core';  
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';  
  
import { AppModule } from './app/app.module';  
import { environment } from './environments/environment';
```



```
if (environment.production) {  
  enableProdMode();  
}  
  
platformBrowserDynamic().bootstrapModule(AppModule);
```

The build process will make sure the right environment is provided when you import it.

Tests

The spec file listed above is where we can add tests, it comes with a simple test that makes sure that the component can be created. Let's go ahead and run our tests using the below command.

```
ng test
```

References / Bibliography

- <https://www.sitepoint.com/ultimate-angular-cli-reference/>
- <http://onehungrymind.com/named-router-outlets-in-angular-2/>