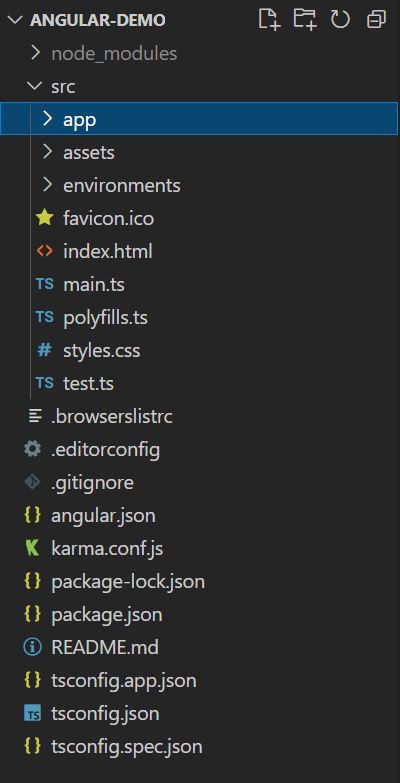
Project Folder Structure

Angular CLI create a new folder as per project name i.e. AngularDemo. So, now open that project in any code editor like Visual Studio Code or Microsoft Visual Studio. The project folder contains the below folder structure –



The created project contains the following folders –

1. e2e - This folder is for an end to end testing purposes. It contains the configuration files related to performing the unit test of the projects.
2. node\_modules - This folder contains the downloaded packages as per the configuration.
3. src - This folder contains the actual source code. It contains 3 subfolders as –
   * app - App folder contains the Angular project-related files like components, HTML files, etc.
   * assets - Assets folder contains any static files like images, stylesheets, custom javascript library files (if any required), etc.
   * environments - Environments folder contains the environment-related files which are required during deployment or build of the projects.

* index.html: Main html file for your application.

Config Files

When we create any Angular based project using Angular CLI, then every time it will create 3 different configuration files which help us to configure the projects along with its related dependencies. These configuration files are –

**tsconfig.json** – If tsconfig.json files exist within the project root folder, that means that the project is a basically TypeScript project. The tsconfig.json file specifies the root files and the compiler options required to compile the project. tsconfig.json as below –

1. {
2. "compileOnSave": **true**,
3. "compilerOptions": {
4. "baseUrl": "./",
5. "outDir": "./dist/out-tsc",
6. "sourceMap": **true**,
7. "declaration": **false**,
8. "module": "esnext",
9. "moduleResolution": "node",
10. "emitDecoratorMetadata": **true**,
11. "experimentalDecorators": **true**,
12. "importHelpers": **true**,
13. "target": "es2015",
14. "typeRoots": [
15. "node\_modules/@types"
16. ],
17. "lib": [
18. "es2018",
19. "dom"
20. ]
21. }
22. }

**package.json** – package.json is basically a JSON file that contains all information related to the required packages for the project. Also, with the help of this configuration files, we can maintain the Project Name and its related version by using the “name” and “version” property. Also, we can provide the build definition of the project using this file.

1. {
2. "name": "angular8-demo",
3. "version": "0.0.0",
4. "scripts": {
5. "ng": "ng",
6. "start": "ng serve",
7. "build": "ng build",
8. "test": "ng test",
9. "lint": "ng lint",
10. "e2e": "ng e2e"
11. },
12. "private": **true**,
13. "dependencies": {
14. "@angular/animations": "~8.0.0",
15. "@angular/common": "~8.0.0",
16. "@angular/compiler": "~8.0.0",
17. "@angular/core": "~8.0.0",
18. "@angular/forms": "~8.0.0",
19. "@angular/platform-browser": "~8.0.0",
20. "@angular/platform-browser-dynamic": "~8.0.0",
21. "@angular/router": "~8.0.0",
22. "rxjs": "~6.4.0",
23. "tslib": "^1.9.0",
24. "zone.js": "~0.9.1"
25. },
26. "devDependencies": {
27. "@angular-devkit/build-angular": "~0.800.0",
28. "@angular/cli": "~8.0.2",
29. "@angular/compiler-cli": "~8.0.0",
30. "@angular/language-service": "~8.0.0",
31. "@types/node": "~8.9.4",
32. "@types/jasmine": "~3.3.8",
33. "@types/jasminewd2": "~2.0.3",
34. "codelyzer": "^5.0.0",
35. "jasmine-core": "~3.4.0",
36. "jasmine-spec-reporter": "~4.2.1",
37. "karma": "~4.1.0",
38. "karma-chrome-launcher": "~2.2.0",
39. "karma-coverage-istanbul-reporter": "~2.0.1",
40. "karma-jasmine": "~2.0.1",
41. "karma-jasmine-html-reporter": "^1.4.0",
42. "protractor": "~5.4.0",
43. "ts-node": "~7.0.0",
44. "tslint": "~5.15.0",
45. "typescript": "~3.4.3"
46. }

**angular.json** – angular.json file is an Angular Application Environment based JSON file which contains all the information related to the project build and deployment. It tells the system which files need to change when we use ng build or ng serve command.

**main.ts** - The main.ts file acts as the main entry point of our Angular application. This file is responsible for the bootstrapper operation of our Angular modules. It contains some important statements related to the modules and some initial setup configurations like

* **enableProdMode**– This option is used to disable Angular’s development mode and enable Productions mode. Disabling Development mode turns off assertions and other model-related checks within the framework.
* **platformBrowserDynamic**– This option is required to bootstrap the Angular app n the browser.
* **AppModule**– This option indicates which module acts as a root module in the applications.
* **environment**– This option stores the values of the different environment constants.

1. **import** { enableProdMode } from '@angular/core';
2. **import** { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
4. **import** { AppModule } from './app/app.module';
5. **import** { environment } from './environments/environment';
7. **if** (environment.production) {
8. enableProdMode();
9. }
11. platformBrowserDynamic().bootstrapModule(AppModule)
12. .**catch**(err => console.error(err));

@ngModule Metadata

In every Angular application, at least one angular module file is required. An Angular application may contain more than one Angular module. Angular modules is a process or system to assemble multiple angular elements, like components, directives, pipes, service, etc. so that these Angular elements can be combined in such a way that all elements can be related with each other and ultimately create an application.

In Angular, @NgModule decorator is used to defining the Angular module class. Sometimes, this class is called a NgModule class. @NgModule always takes a metadata object, which tells Angular how to compile and launch the application in the browser. So, to define the Angular module, we need to define some steps as follows:

1. First, we need to import Angular BrowserModule into the Angular module file at the beginning. This BrowserModule class is responsible for running the application in the browser.
2. In the next step, we need to declare the Angular elements like component within the Angular module so that those components or elements can be associated with the Angular module.
3. In the last step, we need to mention one Angular component as a root component for the Angular module. This component is always known as a bootstrap component. So, one Angular module can contain hundreds of components. But out of those components, one component needs to be a root or bootstrap component that will be executed first when the Angular module will be bootstrapped in the browser.
4. **import** { BrowserModule } from '@angular/platform-browser';
5. **import** { NgModule } from '@angular/core';
7. **import** { AppComponent } from './app.component';
9. @NgModule({
10. declarations: [
11. AppComponent
12. ],
13. imports: [
14. BrowserModule
15. ],
16. providers: [],
17. bootstrap: [AppComponent]
18. })
19. **export** **class** AppModule { }

So, when we create an Angular Project using Angular CLI, it creates new projects along with a module and a default component file. These files normally exist within the app folder. So, first, simply run the Angular Project using ng serve command output will be visible in the browser.