CREATE A NEW ANGULAR APPLICATION

AngularJS is based on the model view controller, whereas Angular 2/4/5 is based on the components structure. Angular 4 works on the same structure as Angular2 but is faster when compared to Angular2.

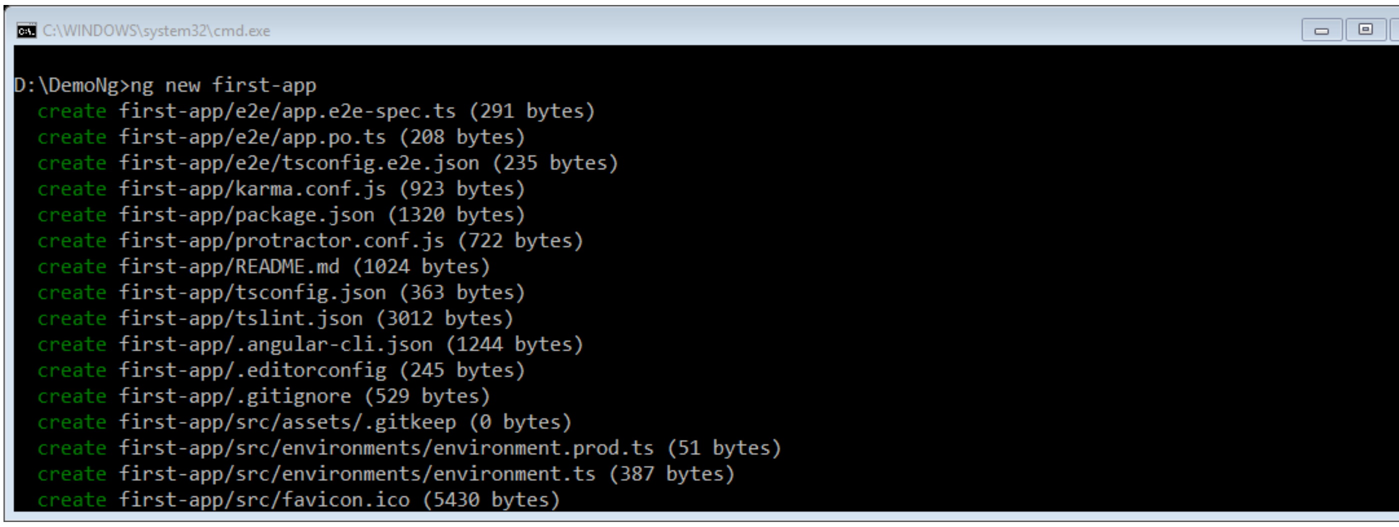
Angular4 uses TypeScript 2.1 version whereas Angular 2 uses TypeScript version 1.8. This brings a lot of difference in the performance.

To install Angular 4, the Angular team came up with Angular CLI which eases the installation. You need to run through a few commands to install Angular 4.

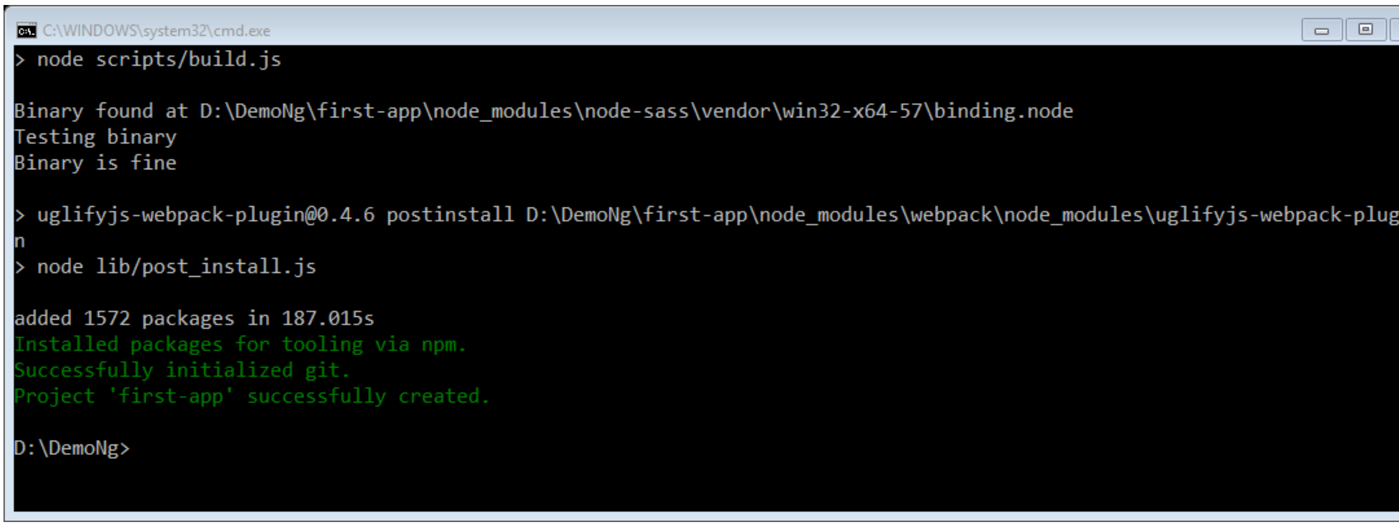
Go to this site https://cli.angular.io to install Angular CLI.

Create a new project named `first-app` with this CLI command.

```
> ng new first-app
```



```
D:\DemoNg>ng new first-app
  create first-app/e2e/app.e2e-spec.ts (291 bytes)
  create first-app/e2e/app.po.ts (208 bytes)
  create first-app/e2e/tsconfig.e2e.json (235 bytes)
  create first-app/karma.conf.js (923 bytes)
  create first-app/package.json (1320 bytes)
  create first-app/protractor.conf.js (722 bytes)
  create first-app/README.md (1024 bytes)
  create first-app/tsconfig.json (363 bytes)
  create first-app/tslint.json (3012 bytes)
  create first-app/.angular-cli.json (1244 bytes)
  create first-app/.editorconfig (245 bytes)
  create first-app/.gitignore (529 bytes)
  create first-app/src/assets/.gitkeep (0 bytes)
  create first-app/src/environments/environment.prod.ts (51 bytes)
  create first-app/src/environments/environment.ts (387 bytes)
  create first-app/src/favicon.ico (5430 bytes)
```



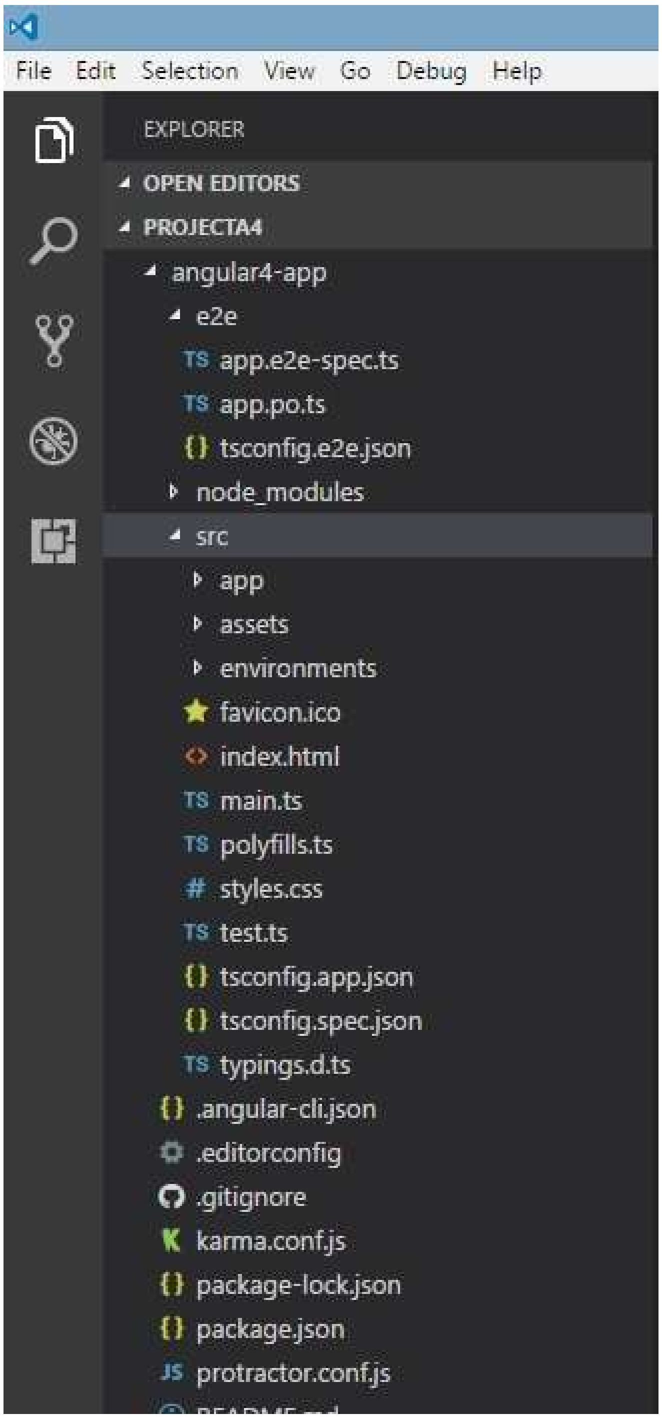
```
> node scripts/build.js

Binary found at D:\DemoNg\first-app\node_modules\node-sass\vendor\win32-x64-57\binding.node
Testing binary
Binary is fine

> uglifyjs-webpack-plugin@0.4.6 postinstall D:\DemoNg\first-app\node_modules\webpack\node_modules\uglifyjs-webpack-plug
n
> node lib/post_install.js

added 1572 packages in 187.015s
Installed packages for tooling via npm.
Successfully initialized git.
Project 'first-app' successfully created.

D:\DemoNg>
```

The project **first-app** is created successfully. It installs all the required packages necessary for our project to run in Angular 4. Let us now switch to the project created, which is in the directory **first-app**. Change the directory in the command line

```
> cd first-app
```

We will use Visual Studio Code IDE for working with Angular 4; you can use any IDE, i.e., Atom, WebStorm, etc.

To download Visual Studio Code, go to https://code.visualstudio.com/ and click **Download for Windows**.

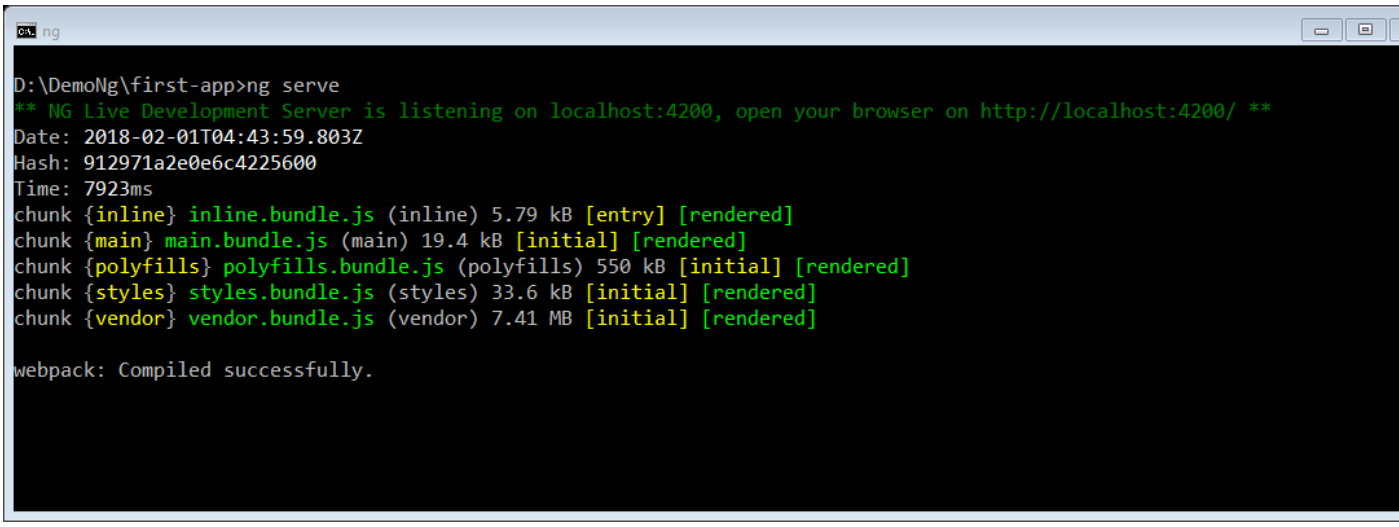


```
angular4-app
  e2e
    app.e2e-spec.ts
    app.po.ts
    tsconfig.e2e.json
  node_modules
  src
    app
    assets
    environments
    favicon.ico
    index.html
    main.ts
    polyfills.ts
    styles.css
    test.ts
    tsconfig.app.json
    tsconfig.spec.json
    typings.d.ts
  .angular-cli.json
  .editorconfig
  .gitignore
  karma.conf.js
  package-lock.json
  package.json
  protractor.conf.js
```

Now that we have the file structure for our project, let us compile our project with the following command –

```
> ng serve
```

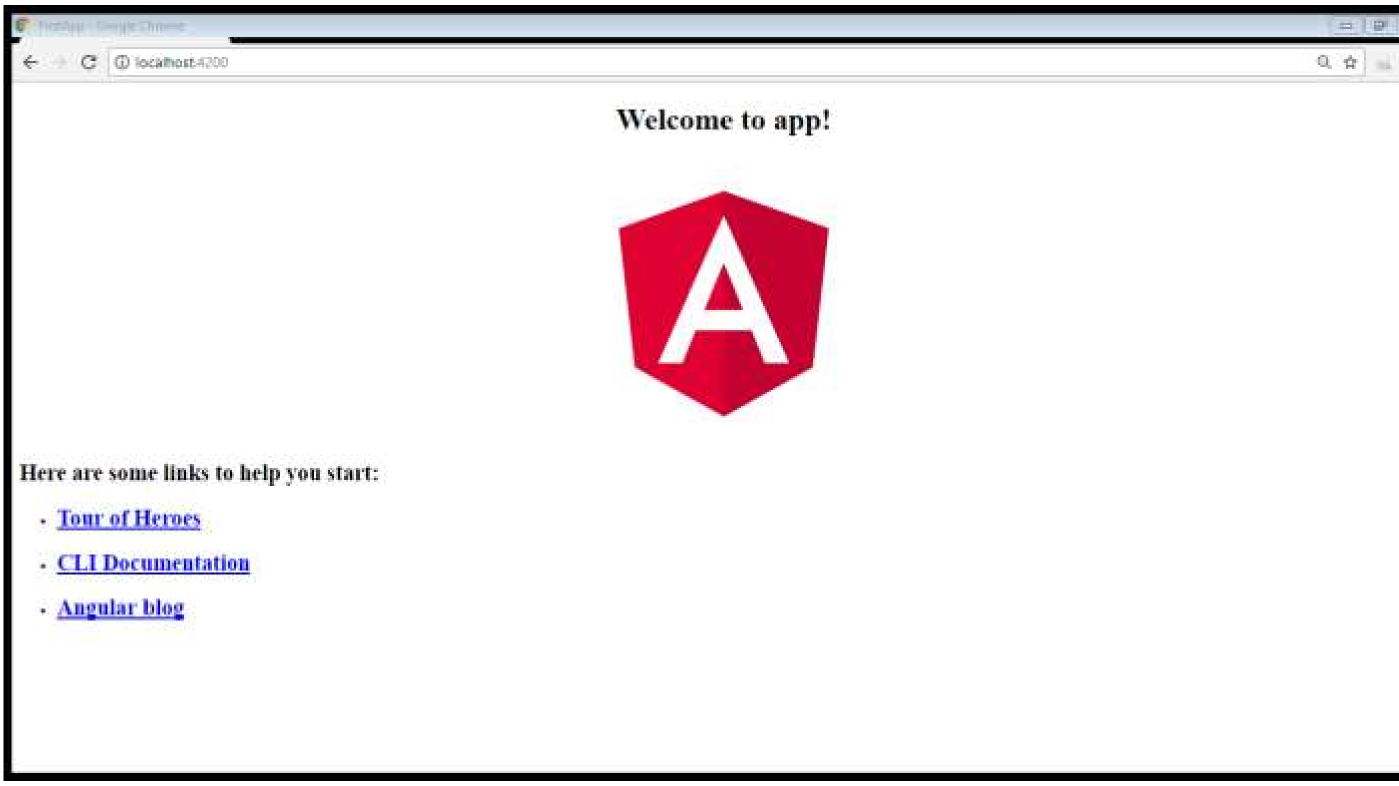
The `ng serve` command builds the application and starts the web server.



```
D:\DemoNg\first-app>ng serve
** NG Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
Date: 2018-02-01T04:43:59.803Z
Hash: 912971a2e0e6c4225600
Time: 7923ms
chunk {inline} inline.bundle.js (inline) 5.79 kB [entry] [rendered]
chunk {main} main.bundle.js (main) 19.4 kB [initial] [rendered]
chunk {polyfills} polyfills.bundle.js (polyfills) 550 kB [initial] [rendered]
chunk {styles} styles.bundle.js (styles) 33.6 kB [initial] [rendered]
chunk {vendor} vendor.bundle.js (vendor) 7.41 MB [initial] [rendered]

webpack: Compiled successfully.
```

The web server starts on port 4200. Type the url **http://localhost:4200/** in the browser and see the output. Once the project is compiled, you will receive the following output –

Once you run **http://localhost:4200/** in the browser, you will be directed to the following screen –



Welcome to app!

Here are some links to help you start:

- Tour of Heroes
- CLI Documentation
- Angular blog

PROJECT FOLDER STRUCTURE

The Angular 4 app folder has the following **folder structure** –

- **e2e** – end to end test folder. Mainly e2e is used for integration testing and helps ensure the application works fine.
- **node_modules** – The npm package installed is node_modules. You can open the folder and see the packages available.
- **src** – This folder is where we will work on the project using Angular 4.
    - **app** - The Angular 4 **app** folder has the following **file structure** –
    - **.angular-cli.json** – It basically holds the project name, version of cli, etc.
    - **.editorconfig** – This is the config file for the editor.
    - **.gitignore** – A .gitignore file should be committed into the repository, in order to share the ignore rules with any other users that clone the repository.
    - **karma.conf.js** – This is used for unit testing via the protractor. All the information required for the project is provided in karma.conf.js file.
    - **package.json** – The package.json file tells which libraries will be installed into node_modules when you run npm install.
- **protractor.conf.js** – This is the testing configuration required for the application.
- **tsconfig.json** – This basically contains the compiler options required during compilation.
- **tslint.json** – This is the config file with rules to be considered while compiling.

The **src** folder is the main folder, which internally has a different file structure.

APP

It contains the files described below. These files are installed by angular-cli by default.

1. **app.module.ts** – If you open the file, you will see that the code has reference to different libraries, which are imported. Angular-cli has used these default libraries for the import – angular/core, platform-browser. The names itself explain the usage of the libraries.

They are imported and saved into variables such as **declarations, imports, providers**, and **bootstrap**.

```
import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';


@NgModule({

   declarations: [

      AppComponent

   ],

   imports: [

      BrowserModule
   ],
   providers: [],
   bootstrap: [AppComponent]
})

export class AppModule { }
```

- **declarations** – In declarations, the reference to the components is stored. The Appcomponent is the default component that is created whenever a new project is initiated. We will learn about creating new components in a different section.
- **imports** – This will have the modules imported as shown above. At present, BrowserModule is part of the imports which is imported from @angular/platform-browser.
- **providers** – This will have reference to the services created. The service will be discussed in a subsequent chapter.
- **bootstrap** – This has reference to the default component created, i.e., AppComponent.

2. **app.component.css** – You can write your css structure over here. Right now, we have added the background color to the div as shown below.
3. **app.component.html** – The html code will be available in this file.
4. **app.component.spec.ts** – These are automatically generated files which contain unit tests for source component.
5. **app.component.ts** – The class for the component is defined over here. You can do the processing of the html structure in the .ts file. The processing will include activities such as connecting to the database, interacting with other components, routing, services, etc.

```
import { Component } from '@angular/core';


@Component({
   selector: 'app-root',
   templateUrl: './app.component.html',
   styleUrls: ['./app.component.css']
})
export class AppComponent {
   title = 'app';
}
```

ASSETS

You can save your images, js files in this folder.

ENVIRONMENT

This folder has the details for the production or the dev environment. The folder contains two files.

- environment.prod.ts
- environment.ts

Both the files have details of whether the final file should be compiled in the production environment or the dev environment.

The additional file structure of Angular 4 app folder includes the following –

FAVICON.ICO

This is a file that is usually found in the root directory of a website.

INDEX.HTML

```
     <!doctype html>

     <html lang = "en">

        <head>

           <meta charset = "utf-8">

           <title>HTTP Search Param</title>

           <base href = "/">

           <link href = "https://fonts.googleapis.com/icon?family=Material+Icons"
rel="stylesheet">

           <link href = "https://fonts.googleapis.com/css?family=Roboto|Roboto+Mono"
rel="stylesheet">

           <link href = "styles.c7c7b8bf22964ff954d3.bundle.css" rel="stylesheet">

           <meta name = "viewport" content="width=device-width, initial-scale=1">

           <link rel = "icon" type="image/x-icon" href="favicon.ico">

        </head>


        <body>

           <app-root></app-root>

        </body>

     </html>
```

The body has **<app-root></app-root>**. This is the selector which is used in **app.component.ts** file and will display the details from app.component.html file.

MAIN.TS

main.ts is the file from where we start our project development. It starts with importing the basic module which we need. Right now if you see angular/core, angular/platform-browser-dynamic, app.module and environment is imported by default during angular-cli installation and project setup.

```
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 **platformBrowserDynamic().bootstrapModule(AppModule)** has the parent module reference **AppModule**. Hence, when it executes in the browser, the file that is called is index.html. Index.html internally refers to main.ts which calls the parent module, i.e., AppModule when the following code executes –

```
platformBrowserDynamic().bootstrapModule(AppModule);
```

When AppModule is called, it calls app.module.ts which further calls the AppComponent based on the boostrap as follows –

```
bootstrap: [AppComponent]
```

In app.component.ts, there is a **selector: app-root** which is used in the index.html file. This will display the contents present in app.component.html.

POLYFILL.TS

This is mainly used for backward compatibility.

STYLES.CSS

This is the style file required for the project.

TEST.TS

Here, the unit test cases for testing the project will be handled.

TSCONFIG.APP.JSON

This is used during compilation, it has the config details that need to be used to run the application.

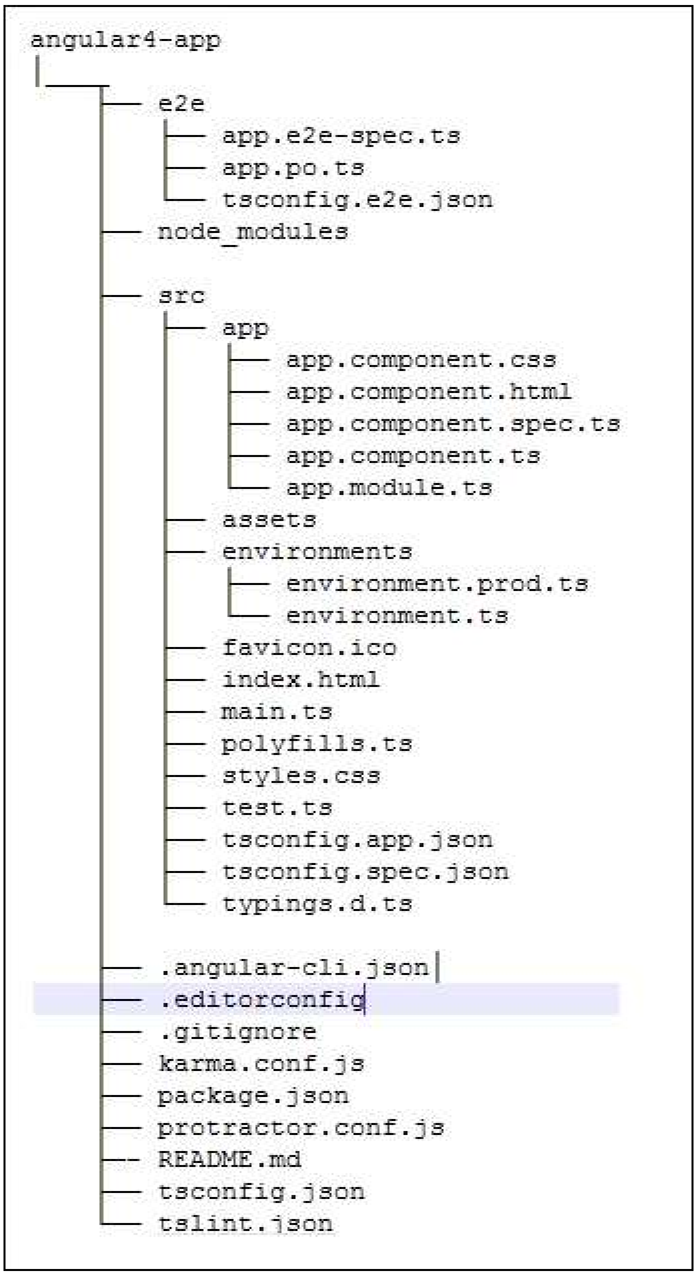
TSCONFIG.SPEC.JSON

This helps maintain the details for testing.

TYPINGS.D.TS

It is used to manage the TypeScript definition.

The final file structure looks as follows –



```
angular4-app
|____
    ├── e2e
    |     ├── app.e2e-spec.ts
    |     ├── app.po.ts
    |     └── tsconfig.e2e.json
    ├── node_modules
    |
    ├── src
    |     ├── app
    |     |     ├── app.component.css
    |     |     ├── app.component.html
    |     |     ├── app.component.spec.ts
    |     |     ├── app.component.ts
    |     |     └── app.module.ts
    |     ├── assets
    |     ├── environments
    |     |     ├── environment.prod.ts
    |     |     └── environment.ts
    |     ├── favicon.ico
    |     ├── index.html
    |     ├── main.ts
    |     ├── polyfills.ts
    |     ├── styles.css
    |     ├── test.ts
    |     ├── tsconfig.app.json
    |     ├── tsconfig.spec.json
    |     └── typings.d.ts
    |
    ├── .angular-cli.json
    ├── .editorconfig
    ├── .gitignore
    ├── karma.conf.js
    ├── package.json
    ├── protractor.conf.js
    ├── README.md
    ├── tsconfig.json
    └── tslint.json
```

HOW ANGULAR WORKS

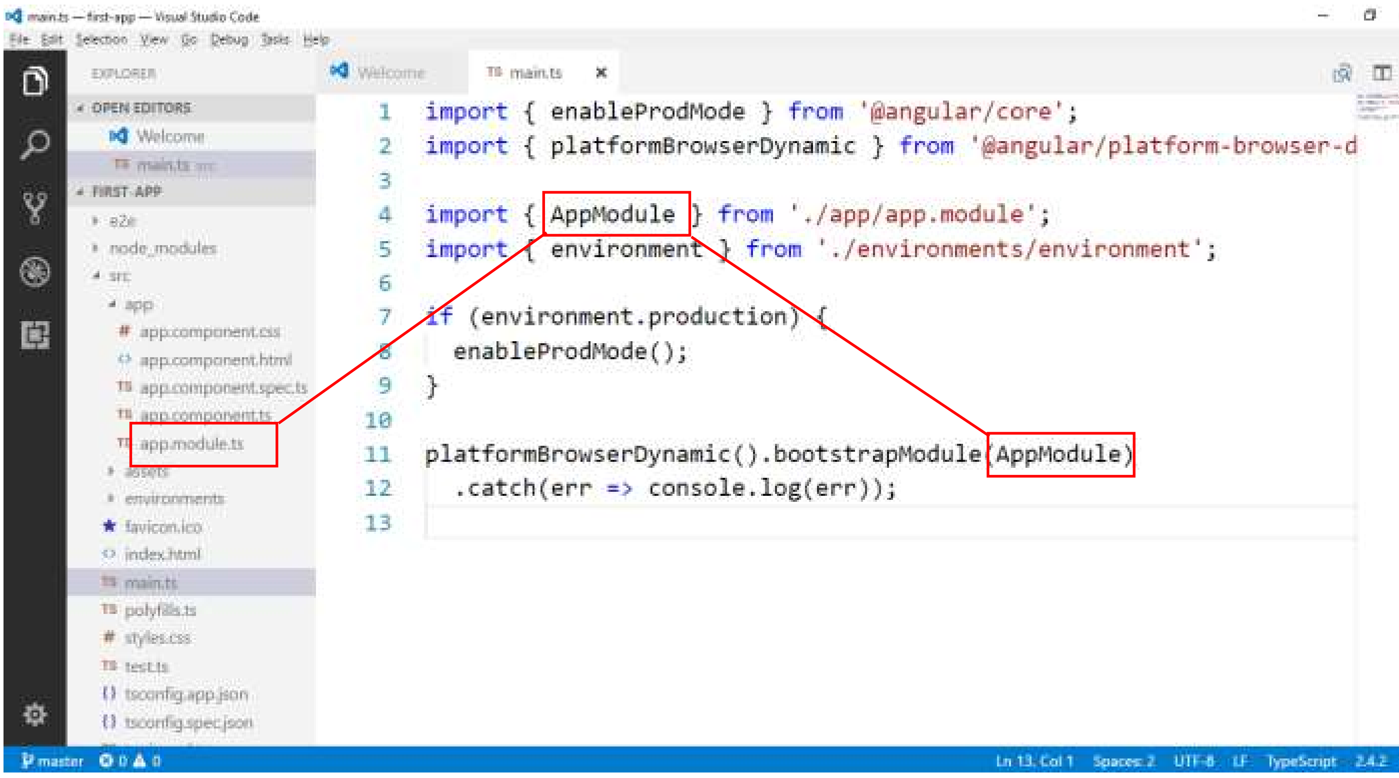
The first big idea is that an Angular application is made up of Components. One way to think of Components is a way to teach the browser new tags. If you have an Angular 1 background, Components are analogous to directives in AngularJS 1.x

BOOTING THE APP

Every app has a main entry point. This application was built using Angular CLI (which is built on a tool called Webpack). We run this app by calling the command:

```
> ng serve
```

- ng will look at the file `.angular-cli.json` to find the entry point to our app
- `.angular-cli.json` specifies a "main" file, which in this case is `main.ts`
    - `main.ts` is the entry-point for our app and it bootstraps our application
    - The bootstrap process boots an Angular module ("AppModule")
- We use the AppModule to bootstrap the app. AppModule is specified in src/app/app.module.ts
- AppModule specifies which component to use as the top-level component. In this case it is AppComponent
- AppComponent has `<app-root>` tags in the template and this renders output



```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-d

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.log(err));
```

Angular has a powerful concept of modules. When you boot an Angular app, you're not booting a component directly, but instead you create an NgModule which points to the component you want to load.

FIRST-APP/SRC/APP/APP.MODULE.TS

```
@NgModule({
declarations: [
    AppComponent
    ],
imports: [
    BrowserModule,
    ],
providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }
```

The first thing we see is an @NgModule decorator. Like all decorators, this @NgModule(...) code adds metadata to the class immediately following (in this case, AppModule).

Our @NgModule decorator has four keys: declarations, imports, providers, and bootstrap.

DECLARATIONS

declarations specifies the components that are defined in this module. This is an important idea in Angular: You have to declare components in a NgModule before you can use them in your templates. You can think of an NgModule a bit like a "package" and declarations states what components are "owned by" this module.

You may have noticed that when we used ng generate, the tool automatically added our components to this declarations list! The idea is that when we generated a new component, the ng tool assumed we wanted it to belong to the current NgModule.

IMPORTS

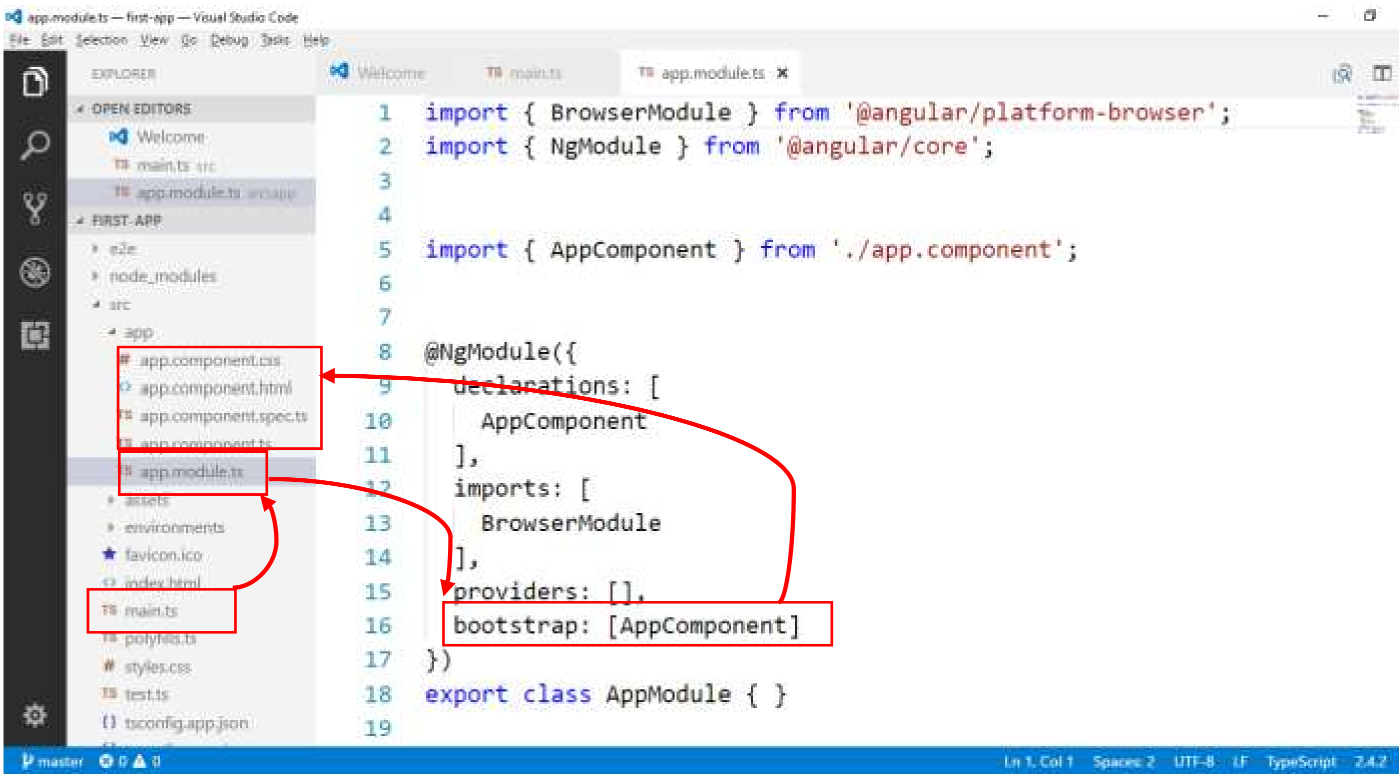
imports describes which dependencies this module has. We're creating a browser app, so we want to import the BrowserModule. If your module depends on other modules, you list them here.

PROVIDERS

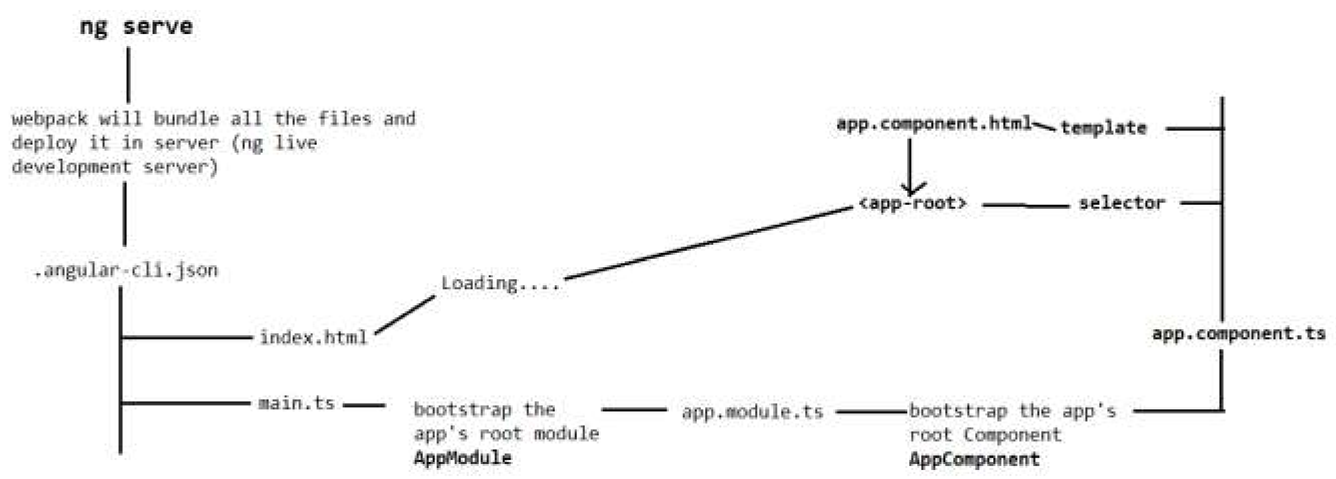
providers are used for dependency injection. So, to make a service available to be injected throughout our application, we will add it here.

BOOTSTRAP

bootstrap tells Angular that when this module is used to bootstrap an app, we need to load the AppComponent component as the top-level component.



```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';


import { AppComponent } from './app.component';


@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```


ng serve
webpack will bundle all the files and deploy it in server (ng live development server)
.angular-cli.json
index.html
Loading....
main.ts
bootstrap the app's root module AppModule
app.module.ts
bootstrap the app's root Component AppComponent
app.component.html
template
<app-root>
selector
app.component.ts