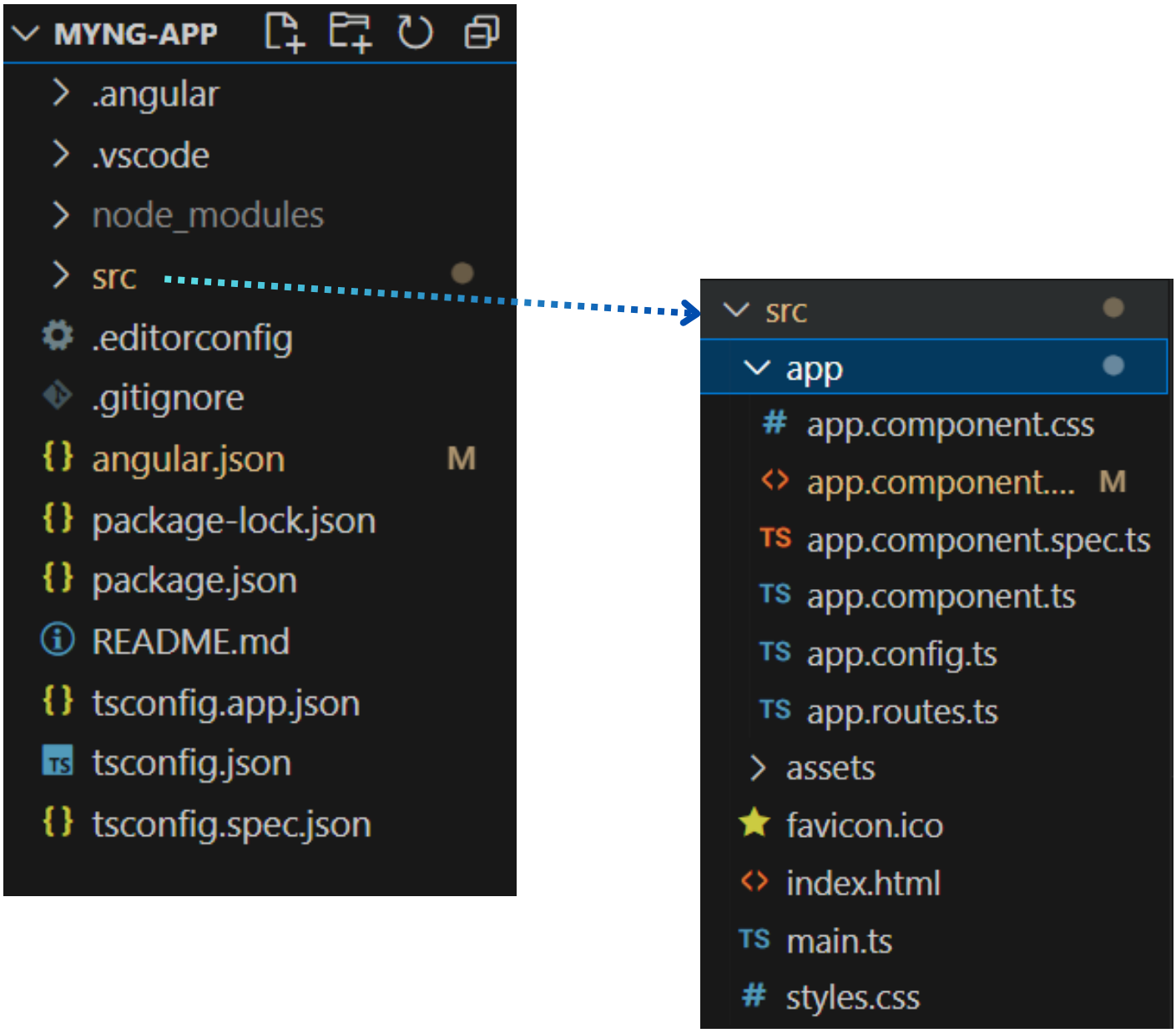
Exploring project structure and files

After creating a new project in the previous step and opening the project in VS Code, you will find the following structure:



Exploring project structure and files

The project structure in Angular follows a modular approach, logically organizing files and directories.Understanding the project structure is essential for managing and navigating an Angular application.

Workspace configuration files

WORKSPACE CONFIGURATION FILES	PURPOSE
.editorconfig	This is the configuration file for the Visual Studio code editor. You can visit http://editorconfig.org for more information.
.gitignore	Git configuration to ensure autogenerated files are not committed to source control.The files which you want to ignore in the git repository, you need to put within this gitignore file.
README.md	README is the file that contains the description or description of the project which we would like to give to the end-users so that they can start using our application in a great manner. This file contains the basic documentation for your project, also contains some pre-filled CLI command information.
angular.json	This is one of the most important files and it contains all the configuration of your Angular Project. It contains the configurations such as what is the project name, what is the root directory as source folder (src) name which contains all the components, services, directives, pipes, what is the starting point of your angular application (index.html file), What is the starting point of typescript file (main.ts), etc.
package.json	Configures npm package dependencies that are available to all projects in the workspace. See npm documentation for the specific format and contents of this file.

Exploring project structure and files

Workspace configuration files

WORKSPACE CONFIGURATION FILES	PURPOSE
package-lock.json	package-lock.json provides version information for all packages installed into node_modules by the npm client.
src/	Source files for the root-level application project.
node_modules/	Provides npm packages to the entire workspace. Workspace-wide node_modules dependencies are visible to all projects.
tsconfig.json	The tsconfig.json is the Typescript compiler configuration file. This file specifies the compiler options required for the Typescript to compile (transpile) the project
tsconfig.app.json	The tsconfig.app.json is used for compiling the code,
tsconfig.spec.json	tsconfig.spec.json for compiling the tests

Exploring project structure and files

Application source files

Files at the top level of src/ support testing and running your application. Subfolders contain the application source and application-specific configuration.

APPLICATION SUPPORT FILES	PURPOSE
app/	Contains the component files in which your application logic and data are defined.
assets/	Contains image and other asset files to be copied as-is when you build your application.
favicon.ico	An icon to use for this application in the bookmark bar.
index.html	The main HTML page that is served when someone visits your site. The CLI automatically adds all JavaScript and CSS files when building your app, so you typically don't need to add any <script> or <link> tags here manually.
main.ts	The main entry point for your application. Compiles the application with the JIT compiler and bootstraps the application's root module (AppModule) to run in the browser. You can also use the AOT compiler without changing any code by appending the --aot flag to the CLI build and serve commands.
styles.css	Lists CSS files that supply styles for a project. The extension reflects the style preprocessor you have configured for the project.

Exploring project structure and files

App Folder

Inside the src folder, the app folder contains your project's logic and data. Angular components, templates, and styles go here.

SRC/APP/ FILES	PURPOSE
app/app.config.ts	Defines the application config logic that tells Angular how to assemble the application. As you add more providers to the app, they must be declared here.
app/app.component.ts	Defines the logic for the application's root component, named AppComponent.
app/app.component.html	Defines the HTML template associated with the root AppComponent.
app/app.component.css	Defines the base CSS stylesheet for the root AppComponent.
app/app.component.spec.ts	Defines a unit test for the root AppComponent.
app/app.module.ts	Defines the root module, named AppModule, that tells Angular how to assemble the application. Initially declares only the AppComponent. As you add more components to the app, they must be declared here.

Exploring project structure and files

App Folder

The Angular CLI creates a simple application that works out of the box. It creates the root component, a root module, and a unit test class to test the component. Now let us see each component of the application one at a time

The Component

The app.component.ts is the component that is added to the project by Angular CLI. You will find it under the folder app/src

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

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

The component class is the most essential part of our application. It represents the view of the application. A view is a region on the screen. It consists of three main parts, i.e. a class, a class decorator, and an import statement

Exploring project structure and files

App Folder

Import statement

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

The import statement imports the libraries used in our component class. Our Component is decorated with the @Component decorator, part of the @angular/core module. So, we need to refer to it in our class. This is done using the import method, as shown above.

Component class

```
export class AppComponent {
  title = 'myng-app';
}
```

The component is a simple class. We define it using the export keyword. The other parts of the app can import it use it. The above component class has one property title. This title is displayed, when the application is run.

The component class can have many methods and properties. The main purpose of the component is to supply logic to our view.

Exploring project structure and files

App Folder

@Component decorator

The AppComponent class is then, decorated with @Component decorator. The @Component (called class decorator) provides Metadata about our component. The Angular uses this Metadata to create the view

```
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
```

The @component Metadata above has three fields. The selector, templateURL & styleUrls

templateUrl

The templateUrl contains the path to the HTML template file. The Angular uses this HTML file to render the view. In the above example, it points to the app.component.html file.

styleUrls

The styleUrls is an array of Style Sheets that Angular uses to style our HTML file. In the above example, it points towards to app.component.css style sheet. The app.component.css file is in the same folder as the AppComponent. The file is empty. You can create styles for the component and put it here

Exploring project structure and files

App Folder

selector

The selector tells angular where to display the template. The example above selector is app-root. The Angular, whenever it encounters the above tag in the HTML file, it replaces it with the template (app.component.html).

The app-root selector is used in index.html

Template

The app.component.html is our template. The templateUrl in component class above points to this class.
The app.component.html file is in the same folder as the AppComponent. The code is almost 500 lines of code, and almost all of it is standard HTML & CSS
Note that {{title}} is placed inside the h1 tags in line no 344. The double curly braces are the angular way of telling our app to read the title property from the component (AppComponent). We call this data binding (interpolation).

```
<span>{{ title }} app is running!</span>
```

Root Module

Angular organizes the application code as Angular modules. The Modules are closely related blocks of code in functionality. Every application must have at least one module.
The Module which loads first is the root Module. This Module is our root module.
The root module is called app.module.ts. (under src/app folder). It contains the following code.

Exploring project structure and files

App Folder

Root Module

Angular organizes the application code as Angular modules. The Modules are closely related blocks of code in functionality. Every application must have at least one module. The Module which loads first is the root Module. This Module is our root module. The root module is called app.module.ts. (under src/app folder). It contains the following code.

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

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

The structure of the Angular module is similar to the component class. Like a Component, it consists of three parts. A class, class decorator and import statement

Exploring project structure and files

App Folder

Module class

```
export class AppModule { }
```

Similar to the component, the Module class is defined with the export keyword. Exporting the class ensures that you can use this module in other modules.

@NgModule class decorator

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

We use the @component decorator to define our component. The Angular Modules require a **@ngModule** decorator. @ngModue decorator passes the metadata about the module.

The @ngModule Metadata above has four fields. The declarations, imports, providers, & bootstrap

Exploring project structure and files

App Folder

Imports Metadata tells the angular list of other modules used by this module. We are importing **BrowserModule** and **AppRoutingModule**. The BrowserModule is the core angular module, which contains critical services, directives, and pipes, etc. The AppRoutingModule is defines the application Routes and is mentioned in later section.

Declaration Metadata lists the components, directives & pipes that belong to this module.

Providers are services that belong to this module, which we can use in other modules.

Bootstrap Metadata identifies the root component of the module. When Angular loads, the appModule it looks for bootstrap Metadata and loads all the components listed here. We want our module to load AppComponent , hence we have listed it here.

Import statement

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
```

The import statement is used to import the Angular libraries required by AppModule. like NgModule & BrowserModule. We also need to import AppComponent, as we want to load AppComponent, when we load the AppModule

Exploring project structure and files

App Folder

App Routing Module

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

const routes: Routes = [ ];

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

The AppRoutingModule in the file app-routing.module.ts defines the Routes of the application. These Routes tells Angular how to move from one part of the application to another part or one View to another View.

The Routes defined in the constant const routes: Routes = [ ];, which is empty

This Module is defined as a separate Module and is imported into AppModule.

Exploring project structure and files

App Folder

Bootstrapping our root module

The app.component.html is the file we need to show the user. It is bound to the AppComponent component. We indicated that the AppComponent is to be bootstrapped when AppModule is loaded.

We must ask Angular to load AppModule when the application is loaded. This is done in the main.ts file

The main.ts file is found under the src folder. The code is as follows.

```
import { bootstrapApplication } from '@angular/platform-browser';
import { appConfig } from './app/app.config';
import { AppComponent } from './app/app.component';

bootstrapApplication(AppComponent, appConfig)
 .catch((err) => console.error(err));
```

The first line of the import is **bootstrapApplication a platform-browser** library. This library contains all the functions required to bootstrap the angular application.

Exploring project structure and files

App Folder

index.html

Index.html is the entry point of our application.

```
<!doctype html>
<html lang="en">
<head>
 <meta charset="utf-8">
 <title>MyngApp</title>
 <base href="/">
 <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 selector app-root, which we defined in our component metadata, is used as an HTML tag. The Angular scans the HTML page, and when it finds the tag <app-root><app-root> replaces the entire content with the content of app.component.html