**Table of Contents**

* [Bootstrapping in Angular](https://www.tektutorialshub.com/angular/angular-bootstrapping-application/#bootstrapping-in-angular)
  + [What is a Bootstrapping](https://www.tektutorialshub.com/angular/angular-bootstrapping-application/#what-is-a-bootstrapping)
* [Index.html Loads First](https://www.tektutorialshub.com/angular/angular-bootstrapping-application/#indexhtml-loads-first)
  + [Building Application](https://www.tektutorialshub.com/angular/angular-bootstrapping-application/#building-application)
    - [What is Webpack?](https://www.tektutorialshub.com/angular/angular-bootstrapping-application/#what-is-webpack)
* [Application Loads](https://www.tektutorialshub.com/angular/angular-bootstrapping-application/#application-loads)
  + [Application Entry point](https://www.tektutorialshub.com/angular/angular-bootstrapping-application/#application-entry-point)
  + [angular.json](https://www.tektutorialshub.com/angular/angular-bootstrapping-application/#angularjson)
* [main.ts Application entry point](https://www.tektutorialshub.com/angular/angular-bootstrapping-application/#maints-application-entry-point)
  + [What is platformBrowserDynamic](https://www.tektutorialshub.com/angular/angular-bootstrapping-application/#what-is-platformbrowserdynamic)
* [Root Module](https://www.tektutorialshub.com/angular/angular-bootstrapping-application/#rootnbspmodule)
* [Component](https://www.tektutorialshub.com/angular/angular-bootstrapping-application/#component)
* [Template](https://www.tektutorialshub.com/angular/angular-bootstrapping-application/#template)
* [Source Code](https://www.tektutorialshub.com/angular/angular-bootstrapping-application/#source-code)
* [Summary](https://www.tektutorialshub.com/angular/angular-bootstrapping-application/#summary)

**Bootstrapping in Angular**

**What is a Bootstrapping**

Bootstrapping is a technique of initializing or loading our Angular application.

let’s walk through our code created in [Create your First new Angular project](https://www.tektutorialshub.com/angular/angular-create-first-application/) and see what happens at each stage and how our AppComponent gets loaded and displays “app works!”. The Angular takes the following steps to load our first view.

1. Index.html loads
2. Angular, Third-party libraries & Application loads
3. Main.ts the application entry point
4. Root Module
5. Root Component
6. Template

**Index.html Loads First**

Web apps need a starting point. Index.html is usually the first page to load. Let us open the file and find out what it contains. You will find it under the src folder.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <!doctype html>  <html lang="en">  <head>    <meta charset="utf-8">    <title>GettingStarted</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> |

There are no javascript files in the index.html. Neither you can see a stylesheet file. The body of the files has the following HTML tag.

|  |  |
| --- | --- |
| 1  2  3 | <app-root></app-root> |

How do Angular loads ?.  To Find out, let us build our application

[**BEST ANGULAR BOOKS**](https://www.tektutorialshub.com/angular/angular-best-books/)  
**The Top 8**[**Best Angular Books**](https://www.tektutorialshub.com/angular/angular-best-books/)**, which helps you to get started with Angular**

**Building Application**

To run our application, we use the [Angular CLI](https://www.tektutorialshub.com/angular/angular-cli-tutorial/) command ng serve or NPM command npm start (npm start command actually translates into ng serve.)

ng serve does build our application but does not save the compiled application to the disk. It saves it in memory and starts the development server.

We use ng build to build our app. Open the command prompt and run the command. This will build and copy the output files to the dist folder

|  |  |
| --- | --- |
| 1  2  3 | ng build |

Use ng build --prod to build and distribute the app for production. For testing/debugging use ng build. The production build optimizes, minimize and uglify the code.

Now open the dist and open the index.html.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | <!doctype html>  <html lang="en">  <head>    <meta charset="utf-8">    <title>GettingStarted</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>      <script src="runtime-es2015.js" type="module"></script>    <script src="runtime-es5.js" nomodule defer></script>    <script src="polyfills-es5.js" nomodule defer></script>    <script src="polyfills-es2015.js" type="module"></script>    <script src="styles-es2015.js" type="module"></script>    <script src="styles-es5.js" nomodule defer></script>    <script src="vendor-es2015.js" type="module"></script>    <script src="vendor-es5.js" nomodule defer></script>    <script src="main-es2015.js" type="module"></script>    <script src="main-es5.js" nomodule defer></script></body>  </html> |

You can see that the compiler included five script files. They are runtime, polyfills, styles, vendor, & main. All these files have two versions one is es5 & the other one es2015

Since the Angular 7**,**we have new feature called **conditional polyfill loading**. Now Angular builds two script files, one for es2015 & another for es5. The es2015 (es6) is for modern browser and es5 is older browsers, which do not support the new features of es2015.  
  
Note the nomodule attribute, which tells the modern browser to ignore the script and do not load it. Hence es5 scripts are not loaded in the modern browsers

runtime.js: Webpack runtime file  
polyfills.js – Polyfill scripts for supporting the variety of the latest modern browsers  
styles.js – This file contains the global style rules bundled as javascript file.  
vendor.js – contains the scripts from the Angular core library and any other 3rd party library.  
main.js – code of the application.

The Angular Version 2 generated only three script files ( inline.js, styles.bundle.js & main.bundle.js).

These files are added by the Webpack module loader.

**What is Webpack?**

Webpack is a bundler. it scans our application looking for javascript files and merges them into one ( or more) big file. Webpack has the ability to bundle any kind of file like JavaScript, CSS, SASS, LESS, images, HTML, & fonts, etc.

The [Angular CLI](https://www.tektutorialshub.com/angular/angular-cli-tutorial/) uses Webpack as a module bundler. Webpack needs a lot of configuration options to work correctly. The Angular CLI sets up all these configuration options behind the scene.

The Webpack traverses through our application looking for javascript and other files and merges all of them into one or more bundles. In our example application, it has created five files.

Ad by Valueimpression

**Application Loads**

So when index.html is loaded, the Angular core libraries, third-party libraries are loaded. Now the angular needs to locate the entry point.

**Application Entry point**

The entry point of our application is main.ts. You will find it under the src folder.

**angular.json**

The Angular finds out the entry point from the configuration file angular.json. This file is located in the root folder of the project. The relevant part of the angular.json is shown below

The angular-cli.json was the configuration file in Angular 5 and before. It is now angular.json since the version Angular 6.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | {    "$schema": "./node\_modules/@angular/cli/lib/config/schema.json",    "version": 1,    "newProjectRoot": "projects",    "projects": {      "GettingStarted": {        "projectType": "application",        "schematics": {},        "root": "",        "sourceRoot": "src",        "prefix": "app",        "architect": {          "build": {            "builder": "@angular-devkit/build-angular:browser",            "options": {              "outputPath": "dist/GettingStarted",              "index": "src/index.html",              "main": "src/main.ts",                        <====              "polyfills": "src/polyfills.ts",              "tsConfig": "tsconfig.app.json",              "aot": false,              "assets": [                "src/favicon.ico",                "src/assets"              ],              "styles": [                "src/styles.css"              ],              "scripts": []            }, |

The main entry under the node *projects -> GettingStarted -> architect -> build -> options*points towards the src/main.ts. This file is the entry point of our application.

**main.ts Application entry point**

The main.ts file is as shown below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | 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)    .catch(err => console.error(err)); |

Let us look at the relevant code in detail.

|  |  |
| --- | --- |
| 1  2  3 | import { platformBrowserDynamic } from '@angular/platform-browser-dynamic'; |

This line imports the module platformBrowserDynamic from the library@angular/platform-browser-dynamic.

**What is platformBrowserDynamic**

platformBrowserDynamic is the module, which is responsible for loading the Angular application in the desktop browser.

The Angular Applications can be bootstrapped in many ways and in many platforms. For example, we can load our application in a Desktop Browser or in a mobile device with Ionic or NativeScript.

If you are using the nativescript, then you will be using platformNativeScriptDynamic from nativescript-angular/platform library and will be calling platformNativeScriptDynamic().bootstrapModule(AppModule). Read more about [Angular Nativescript bootstrap process from here](https://docs.nativescript.org/404.html?path=core-concepts/angular-bootstrap)

|  |  |
| --- | --- |
| 1  2  3 | import { AppModule } from './app/app.module'; |

The above line imports AppModule. The AppModule is the Root Module of the app. The Angular applications are organized as modules. Every application built in Angular must have at least one module. The module, which is loaded first when the application is loaded is called a root module.

|  |  |
| --- | --- |
| 1  2  3  4 | platformBrowserDynamic().bootstrapModule(AppModule)    .catch(err => console.error(err)); |

TheplatformBrowserDynamic loads the root module by invoking the bootstrapModule and giving it the reference to our Root module i.e AppModule

**Root Module**

The angular bootstrapper loads our root module AppModule. The AppModule is located under the folder src/app. The code of our Root module is shown below

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | import { BrowserModule } from '@angular/platform-browser';  import { NgModule } from '@angular/core';    import { AppRoutingModule } from './app-routing.module';  import { AppComponent } from './app.component';    @NgModule({    declarations: [      AppComponent    ],    imports: [      BrowserModule,      AppRoutingModule    ],    providers: [],    bootstrap: [AppComponent]  })  export class AppModule { } |

The root module must have at least one root component. The root component is loaded, when the module is loaded by the Angular.

In our example, AppComponent is our root component. Hence we import it.

|  |  |
| --- | --- |
| 1  2  3 | import { AppComponent } from './app.component'; |

We use @NgModule class decorator to define a [Angular Module](https://www.tektutorialshub.com/angular/angular-modules/) and provide metadata about the Modules.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | @NgModule({    declarations: [      AppComponent    ],    imports: [      BrowserModule,      AppRoutingModule    ],    providers: [],    bootstrap: [AppComponent]  })  export class AppModule { } |

The [@NgModule](https://www.tektutorialshub.com/angular/angular-modules/#ngmodule) has several metadata properties.

**imports**

We need to list all the external modules required including other Angular modules, that is used by this Angular Module

**Declarations**

The Declarations array contains the list of components, directives, & pipes that belong to this Angular Module. We have only one component in our application AppComponent.

**Providers**

The [Providers](https://www.tektutorialshub.com/angular/angular-providers/) array, is where we register the services we create. The [Angular Dependency injection](https://www.tektutorialshub.com/angular/angular-dependency-injection/) framework injects these [services](https://www.tektutorialshub.com/angular/angular-services/) in components, directives. pipes and other services.

**Bootstrap**

The component that angular should load, when this Angular Module loads. The component must be part of this module. We want AppComponent load when AppModule loads, hence we list it here.

The Angular reads the bootstrap metadata and loads the AppComponent

**Component**

Finally, we arrive at AppComponent, which is the root [component](https://www.tektutorialshub.com/angular/angular-component/) of the AppModule. The code of our AppComponent is shown below

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | import { Component } from '@angular/core';    @Component({    selector: 'app-root',    templateUrl: './app.component.html',    styleUrls: ['./app.component.css']  })  export class AppComponent {    title = 'GettingStarted';  } |

The Class AppComponent is decorated with @Component Class Decorator.

The @Component class decorator provides the metadata about the class to the Angular. It has 3 properties in the above code. Selector, templateURL & styleUrls

**templateURL**

This property contains an HTML template, which is going to be displayed in the browser. The template file is app.component.html

**Selector**

This property specifies the CSS Selector, where our template will be inserted into the HTML. The CSS Selector in our code is app-root

**Template**

The AppComponent defines the template as app.component.htmland the CSS Selector is app-root

Our index.html already have the app-root CSS selector defined

|  |  |
| --- | --- |
| 1  2  3  4  5 | <body>    <app-root></app-root>  </body> |

The Angular locates app-root in our index.html and renders our template between those tags.

**Source Code**