Bootstrapping In Angular: How It Works Internally

26 Comments / March 11, 2023 / 8 minutes of reading

⟵ Create new Project Angular Tutorial Angular Components ⟶

In this article on Bootstrapping in Angular, let us find out how Angular works internally and bootstraps our app. We used `ng new` to create a new Angular project. It generates lots of boilerplate codes. It also configures the Typescript, Webpack & Karma. The app, when run, displays a simple HTML page with several useful links to Angular. Now let us break up this app and look at what happens when the app starts until it displays the HTML page

Table of Contents

Create a new Angular Application
Bootstrapping in Angular

What is a Bootstrapping
Index.html loads first
Building Application
Loads Angular & Third-party libraries & App
Application Entry point
angular.json
Application entry point (main.ts)
What is platformBrowserDynamic
Root Module
AppComponent
Template
Summary

Create a new Angular Application

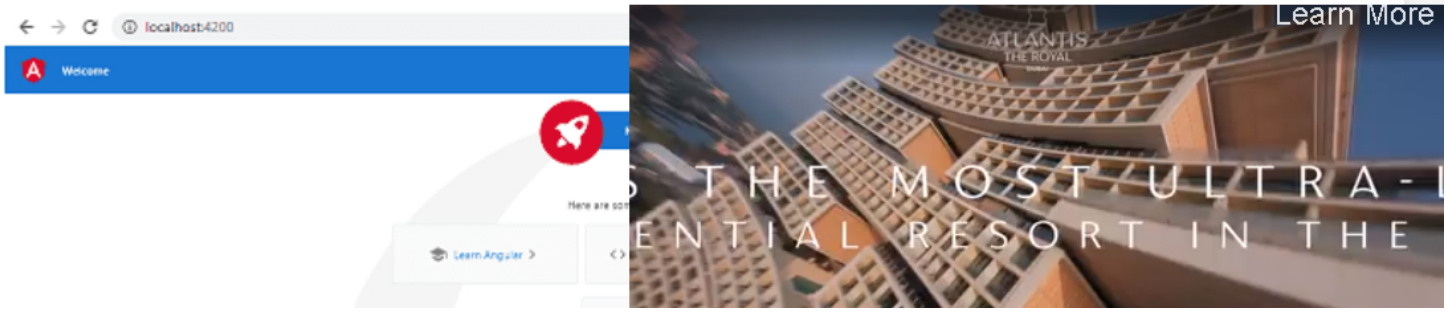
We'll start with a new application to learn how Angular works internally.

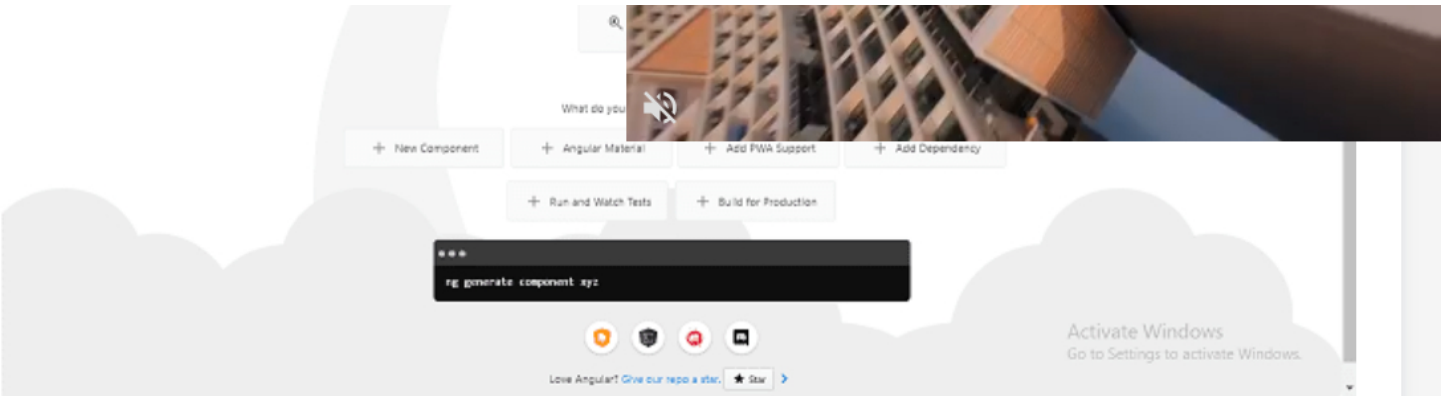
Install Angular and run the following command to create a new Angular application.

```
ng new GettingStarted
```

You can run the app using the following command, which will compile and run the app.

```
ng serve --open
```





Bootstrapping in Angular

What is a Bootstrapping

Bootstrapping is the process of initializing or loading our Angular application.

Angular takes the following steps to load our first view.

1. Loads Index.html
2. Loads Angular & Third-party libraries & Application
3. Executes application entry point (main.ts)
4. Load & execute Root Module (app.module.ts)
5. Executes the Root Component (app.component.ts)
6. Displayes the template (app.component.html)

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.

```

<!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">
```

```
  <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 there is a stylesheet file. The body of the files has the following HTML tag.

```

<app-root></app-root>

```

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

Building Application

To run our application, we use the Angular CLI command `ng serve`.

`ng serve` builds our application but does not save the compiled application to the disk. It saves it in memory and starts the development server. `ng serve` also watches over the project. If we make any changes to the application, it will re-compile and update the file.

To view the compiled application, we need to build the project. We can do that using the `ng build`. Open the command prompt and run the `ng build` command. This will build and copy the output files to the `dist\getting-started` folder (Older versions copied the files to the `dist` folder).

```

ng build

```

Now open the `dist\getting-started` and open the `index.html`.

```html

<!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" />
    <link rel="stylesheet" href="styles.56e3e15f5d5be678.css" />
  </head>
  <body>
    <app-root></app-root>
    <script src="runtime.87dd532f2ee308a5.js" type="module"></script>
    <script src="polyfills.96a2a660c10c5395.js" type="module"></script>
    <script src="main.6ea3cc439cb6e266.js" type="module"></script>
  </body>
</html>

```

You can see that the compiler included four script files. They are `runtime`, `polyfills`, `styles`, & `main`.

`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 a 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 Webpack module loader adds these files.

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 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.

Angular 13 Update:
Angular discontinues the support for IE 11. Hence removes the differential loading from version 13.

Angular 12 Update:
ng build does a production build. In the prior versions of Angular, we needed to specify the flag `ng build --prod` for a production build. The production build optimizes, minimizes, and uglifies the code.

You can use the `ng build --optimization=false` to stop the compiler from optimizing, minimizing, and uglifing the code.

Angular 7 Update:
Since Angular 7, we have had a new feature called conditional polyfill loading (differential loading). Now Angular builds two versions of script files, one for es2015 (runtime-es2015.js, polyfills-es2015.js, vendor-es2015.js, main-es2015.js) & another for es5 (runtime-es5.js, polyfills-es5.js, vendor-es5.js, main-es5.js). The es2015 (es6) is for modern browsers, and es5 is for older browsers, which do not support the new features of es2015.

Angular used the nomodule attribute, which tells the modern browser to ignore and

not load the script. Hence `es5` scripts are not loaded in modern browsers.

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

Loads Angular & Third-party libraries & Application

So when `index.html` is loaded, the Angular core libraries and 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 from `angular.json` is shown below

```

{
  "$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",
```

```
16        "options": {
17          "outputPath": "dist/getting-started",
18          "index": "src/index.html",
19          "main": "src/main.ts",            ===>
20          "polyfills": [
21            "zone.js"
22          ],
23
```

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

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

Application entry point (main.ts)

The main.ts file is shown below.

```
1
2 import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
3
4 import { AppModule } from './app/app.module';
5
6 platformBrowserDynamic().bootstrapModule(AppModule)
7   .catch(err => console.error(err));
8
```

Let us look at the relevant code in detail.

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

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

`@angular/platform-browser-dynamic`.

What is platformBrowserDynamic

`platformBrowserDynamic` is the module responsible for loading the Angular application in the desktop browser?

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` it from nativescript-angular/platform library and will be calling platformNativeScriptDynamic().bootstrapModule(AppModule). Read more about the Angular Nativescript bootstrap process here.

```

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 loaded first when the application is loaded is called a root module.

```

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

```

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

Angular 15: update

Angular 13: update

The prior versions of main.ts also conta

longer needed, hence removed from Ar

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.

```

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

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

```

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

```

We use `@NgModule` class decorator to define an Angular Module and provide metadata about the Modules.

```

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

```

The @NgModule has several metadata properties.

imports

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

Declarations

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

Providers

The Providers array is where we register the services we create. The Angular dependency injection framework injects these services in components and 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`

AppComponent

Finally, we arrived at `AppComponent`, which is the root component of the `AppModule`. The code of our `AppComponent` is shown below

```

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 Angular. It has 3 properties in the above code. `Selector`, `templateURL` & `styleUrls`

templateURL

This property contains an HTML template, which will 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.html** and the CSS Selector is **app-root**

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

```

<body>
  <app-root></app-root>
</body>

```

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

Summary

We use ng build to build the application. The Angular compiler uses the angular.json to find the locations of application entry points (main.ts) & index.html. The compiler copies the code to the dist folder.

index.html loads the angular and third-party libraries.

We declare the root module in main.ts. Hence it will load next.

The root module declares its app.component.ts as its root component. Hence app.component loads next.

app.component.ts declares app.component.html as its template and `app-root` as its selector. Hence the template is loaded and displayed to the user.

Read More

1. Angular Tutorial
2. Introduction to Angular
3. Angular Architecture Overview & Concepts
4. Install Angular
5. How to Create a new project in Angular
6. Bootstrapping in Angular

⟵ Create new Project | Angular Tutorial | Angular Components ⟶

26 thoughts on "Bootstrapping in Angular: How It Works Internally"

ANONYMOUS
MARCH 21, 2023 AT 1:11 PM

The best explanation!

Reply

KAKA
NOVEMBER 15, 2022 AT 4:01 AM

APPRECIATED!

Reply

SCOTTY
DECEMBER 1, 2021 AT 1:52 PM

very nice, except there are too many ads in-between the paragraphs…
Anyway, really appreciate your explanation, quite helpful.

Reply

ANONYMOUS
SEPTEMBER 2, 2021 AT 10:48 PM

Excellent, Superb !!! A MUST read for beginners.

Reply

ANONYMOUS
SEPTEMBER 2, 2021 AT 12:10 AM

best explaination

Reply

DEBASMIT

JULY 24, 2021 AT 6:02 PM

Best of the best. What a classic way of explanation on Angular basics. Well done.

Reply

MURALI KRISHNA

JULY 11, 2021 AT 11:06 PM

Best of the best

Reply

KIRAN

JULY 10, 2021 AT 8:56 PM

Too good sir. Thank you for your efforts. very usefull.

Reply

ANONYMOUS

JULY 6, 2021 AT 9:42 AM

Good Breakdown and explaination

Reply

NAVJYOT

JUNE 28, 2021 AT 1:36 PM

Awesome explanation

Reply

CHINMAYA

JUNE 20, 2021 AT 10:33 AM

Really good articles !

Reply

RAMANANDA

MAY 24, 2021 AT 11:55 AM

Best Tutorial so far!

Reply

ANONYMOUS

MAY 20, 2021 AT 7:24 AM

Nice article. You make it simple and understandable.

Reply

DESH

FEBRUARY 16, 2021 AT 11:37 AM

What is dist folder can you explain please?

Reply

IVAN

MARCH 21, 2021 AT 5:54 PM

Hi, “dist” is a folder which is created once you have run the “ng build” command in order to build you Angular application for the production use. If the command finishes sucessfully, that folder will contain the build version of your Angular application which is then ready to deploy to the server.

Reply

ANONYMOUS

APRIL 29, 2021 AT 4:51 PM

Once you run the command “ng build”, after successful build one dist folder will create in your project. Angular will do minification for your scripts and place files inside the folder. Minification means removing whitespaces and save file size.

Reply

SATHISH

JANUARY 28, 2021 AT 5:02 AM

Nice Explanation on bootstrap…

Reply

MARIAM

JANUARY 2, 2021 AT 3:25 AM

It’s more helpful and simple than the Angular’s Documentation.

Reply

DEEPAK

JANUARY 1, 2021 AT 4:55 PM

Very good article. Simple and clear explanations. Kudos!!

Reply

HIMANSHU VERMA

MAY 30, 2021 AT 6:23 PM

Wow wow. Just wow. Very very nicely written.
Please keep posting. I am bookmarking your Site.

Reply

VIKAS UNIYAL

OCTOBER 8, 2020 AT 4:39 PM

its good for deep learning with good practices ….keep it up bro and it helps me a lot ..thank you soo much …please add some picture output with each code … 🙂

Reply

ANONYMOUS

SEPTEMBER 30, 2020 AT 6:28 PM

Excellent blog/doc. could not resist to share this..

Reply

ANURAG

DECEMBER 26, 2020 AT 8:38 PM

Really nice explanation.

Reply

SATHISHKUMAR

SEPTEMBER 29, 2020 AT 11:09 AM

Excellent article. Explanation about entryComponents in the Module section is missing. You can add that too.

Reply

VENKAT

JULY 13, 2020 AT 10:57 PM

Very well structured and best angular tutorial. I am desperately looking for this kind of tutorial. Really Kudos to your work. Thank you.

Reply

Reply

ANONYMOUS

APRIL 4, 2020 AT 1:37 PM

Exactly what I was looking for!

Reply

Leave a Comment

Your email address will not be published. Required fields are marked *

Type here..

Name*

Email*

Website

Post Comment »

This site uses Akismet to reduce spam. Learn how your comment data is processed.

Copyright © 2(