ANGULAR 7

What is Angular 7?

Angular 7 is a JavaScript **(actually a TypeScript based open-source full-stack web application)**framework which makes you able to create reactive **Single Page Applications** (SPAs). Angular 7 is completely based on components. It consists of several components which forms a tree structure with parent and child components. Angular's versions beyond 2+ are generally known as **Angular** only. The very first version Angular 1.0 is known as **AngularJS.**

"Angular is a complete rewrite of AngularJS by the same team that built AngularJS."

What is Single Page Application (SPA)?

A single page application is a web application or a website which provides users a very fluid, reactive and fast experience similar to a desktop application. It contains menu, buttons and blocks on a single page and when a user clicks on any of them; it dynamically rewrites the current page rather than loading entire new pages from a server. That's the reason behind its reactive fast speed.

Difference between AngularJS and Angular

AngularJS	Angular
AngularJS is common and popular name of the first version of Angular1.0.	Angular is common and popular name of the Angular's version beyond 2+

AngularJS is a JavaScript-based open-source front-end web framework.	Angular is a TypeScript-based open-source full-stack web application framework.
AngularJS uses the concept of scope or controller.	Instead of scope and controller, Angular uses hierarchy of components as its primary architectural characteristic.
AngularJS has a simple syntax and used on HTML pages along with the source location.	Angular uses the different expression syntax. It uses "[ ]" for property binding, and "( )" for event binding.
AngularJS is a simple JavaScript file which is used with HTML pages and doesn't support the features of a server-side	Angular uses of Microsoft's TypeScript language, which provides Class-based Object Oriented Programming, Static Typing, Generics etc.

programming language.	which are the features of a server-side programming language.
AngularJS doesn't support dynamic loading of the page.	Angular supports dynamic loading of the page.

Angular Features

A list of most important features and benefits of Angular:

Angular supports multiple platforms

Angular is a cross platform language. It supports multiple platforms. You can build different types of apps by using Angular.

1. **Desktop applications:** Angular facilitates you to create desktop installed apps on different types of operating systems i.e. Windows, Mac or Linux by using the same Angular methods which we use for creating web and native apps.
2. **Native applications:** You can built native apps by using Angular with strategies from Cordova, Ionic, or NativeScript.
3. **Progressive web applications:**Progressive web applications are the most common apps which are built with Angular. Angular provides modern web platform capabilities to deliver high performance, offline, and zero-step installation apps.

High Speed, Ultimate Performance

Angular is amazingly fast and provides a great performance due to the following reasons:

1. **Universal support:** Angular can be used as a front-end web development tool for the programming languages like Node.js, .Net, PHP, Java Struts and Spring and other servers for near-instant rendering in just HTML and CSS. It also optimizes the website for better SEO.
2. **Code splitting:** Angular apps are fast and loads quickly with the new Component Router, which delivers automatic code-splitting so users only load code required to render the view they request.
3. **Code generation:** Angular makes your templates in highly optimized code for today?s JavaScript virtual machines which gives the benefits of hand-written code.

Productivity

Angular provides a better productivity due to its simple and powerful template syntax, command line tools and popular editors and IDEs.

1. **Powerful templates:** Angular provides simple and powerful template syntax to create UI view quickly.
2. **IDEs:** Angular provides intelligent code completion, instant errors, and other feedback in popular editors and IDEs.
3. **Angular CLI:** Angular CLI provides command line tools start building fast, add components and tests, and then instantly deploy.

Full Stack Development

Angular is a complete framework of JavaScript. It provides Testing, animation and Accessibility. It provides full stack development along with Node.js, Express.js, and MongoDB.

1. **Testing:** Angular provides Karma and Jasmine for unit testing. B y using it, you can check your broken things every time you save. Karma is a JavaScript test runner tool created by Angular team. Jasmine is the testing framework form unit testing in Angular apps, and Karma provides helpful tools that make it easier to us to call our Jasmine tests whilst we are writing code.
2. **Animation Support:** Angular facilitates you to create high-performance, complex choreographies and animation timelines with very little code through Angular's intuitive API.
3. **Accessibility:** In Angular, you can create accessible applications with ARIA-enabled components, developer guides, and built-in a11y test infrastructure.

How to install Angular 7?

Angular 7 Environment Setup

In this page, you will see how you can install the prerequisites needed to run your first Angular 7 app.

Install Visual Studio Code IDE or JetBrains WebStorm

You must have an IDE like Visual Studio Code IDE or JetBrains WebStorm to run your Angular 7 app.

VS Code is light and easy to setup, it has a great range of built-in code editing, formatting, and refactoring features. It is free to use. It also provides a huge number of extensions that will significantly increase your productivity.

You can download VS Code from here: **https://code.visualstudio.com**

JetBrains WebStorm is also a great IDE to develop Angular 7 apps. It is fast, attractive, and very easy to use software but, it is not free to use. You have to purchase it later, it only provides a trial period of 30 days for free.

You can download VS Code from
here: **https://www.jetbrains.com/webstorm/download/#section=windows**

Install Node.js

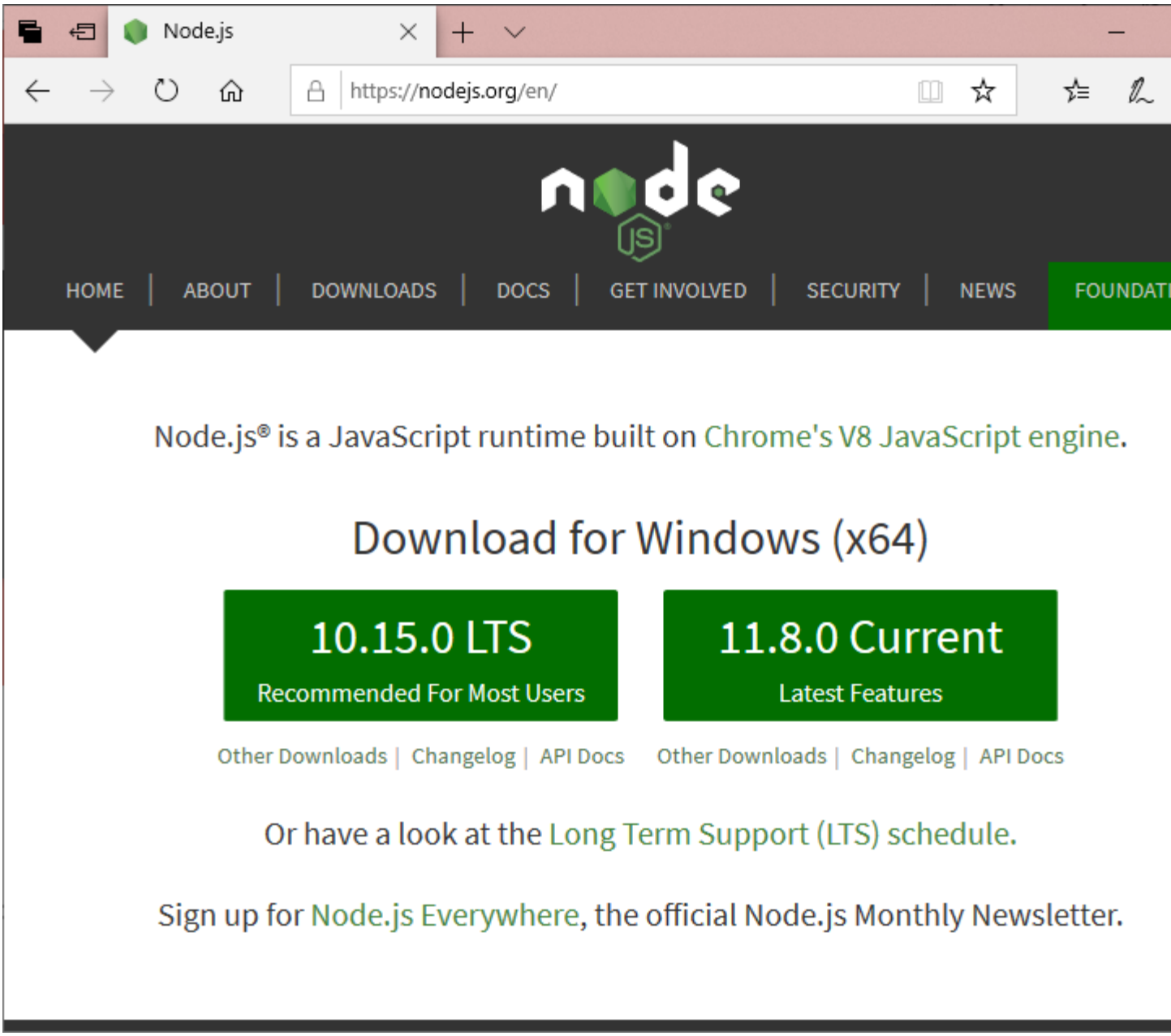
You should install node.js to run your Angular 7 app. It manages npm dependencies support some browsers when loading particular pages. It provides required libraries to run Angular project. Node.js serves your run-time environment as your localhost.

See how to install node.js: **install-nodejs**

Or

Just go to node.js official website **https://nodejs.org/en/**

Download and install latest version of node.js. In my case, it is 11.8.0



After the successful installation, you will see command prompt

You will see the whole cli command to create an Angular app. You need to run the first command to install Angular CLI. These steps are same for Windows and Mac.

1. npm install -g @angular/cli
2. ng new my-dream-app
3. cd my-dream-app
4. ng serve

Your Angular 7 Environment setup is complete now.

The latest version of Angular is Angular 7. It was released on October 18, 2018. It consists of many extensive features:

- Updates regarding Application Performance
- Angular Material & CDK
- Virtual Scrolling
- Improved Accessibility of Selects
- Supports Content Projection using web standard for custom elements
- Dependency updates regarding TypeScript 3.1, RxJS 6.3, Node 10

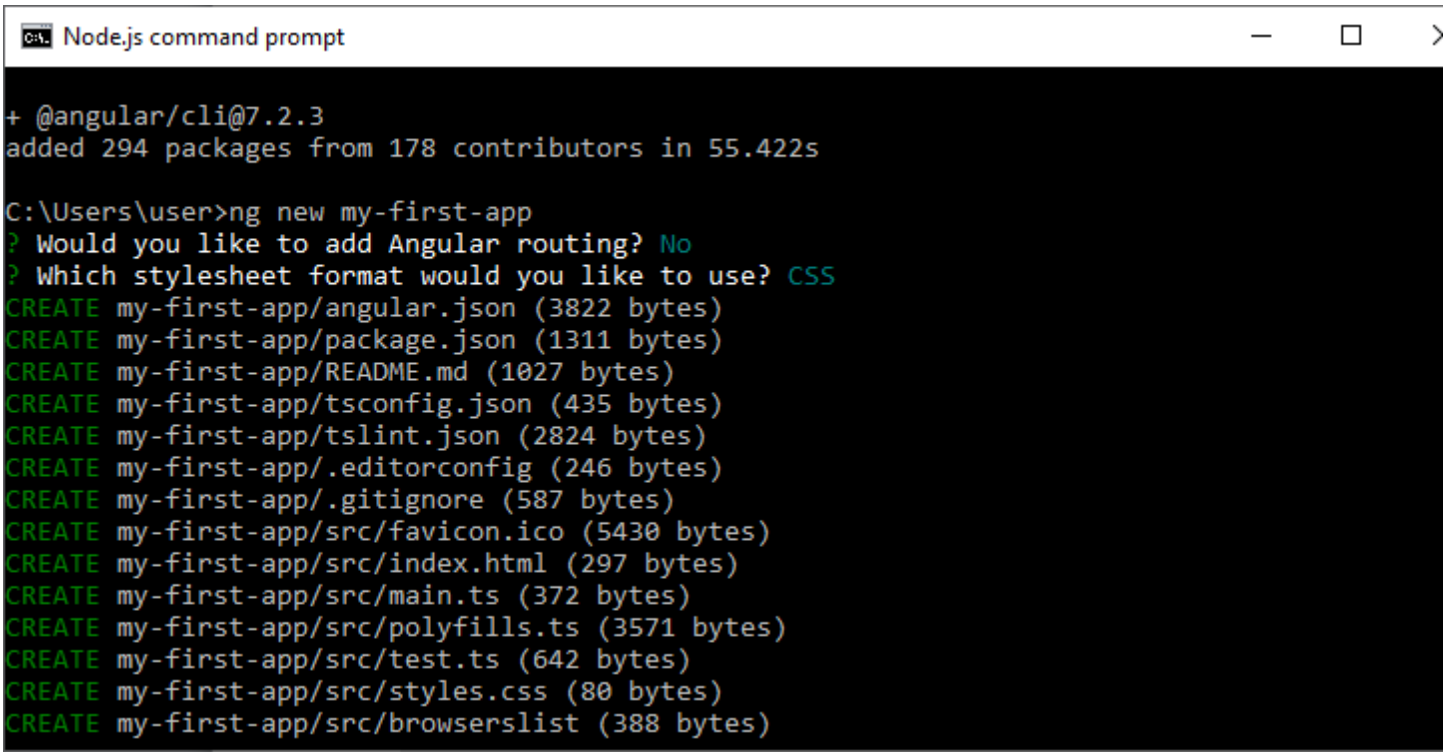
Angular 7 Project Setup (Create first app)

Following are the Angular CLI commands to create the first Angular app.

```
1. npm install -g @angular/cli
2. ng new my-dream-app
3. cd my-dream-app
4. ng serve
```

Run the following command to create your first Angular app.

```
1. ng new my-first-app
```

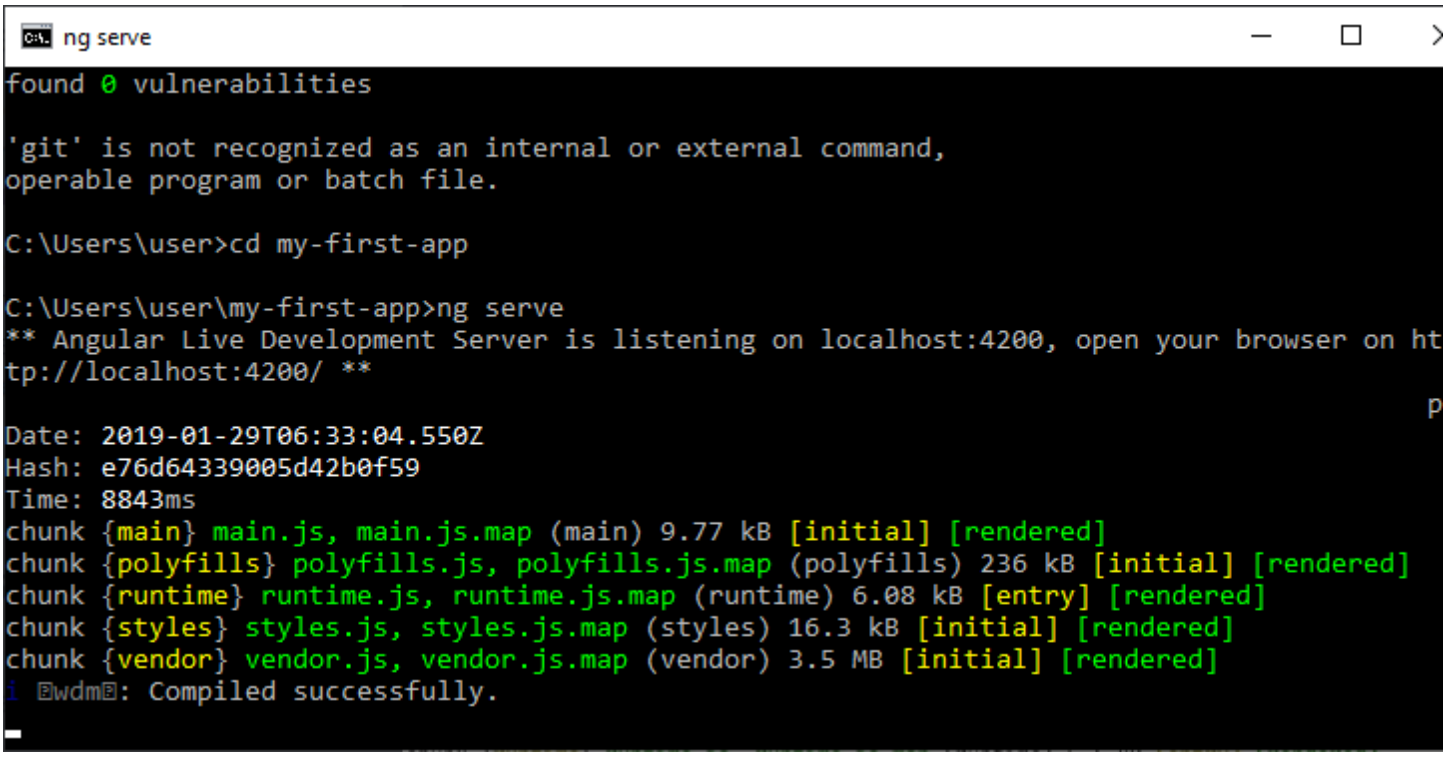


Navigate to your first app.

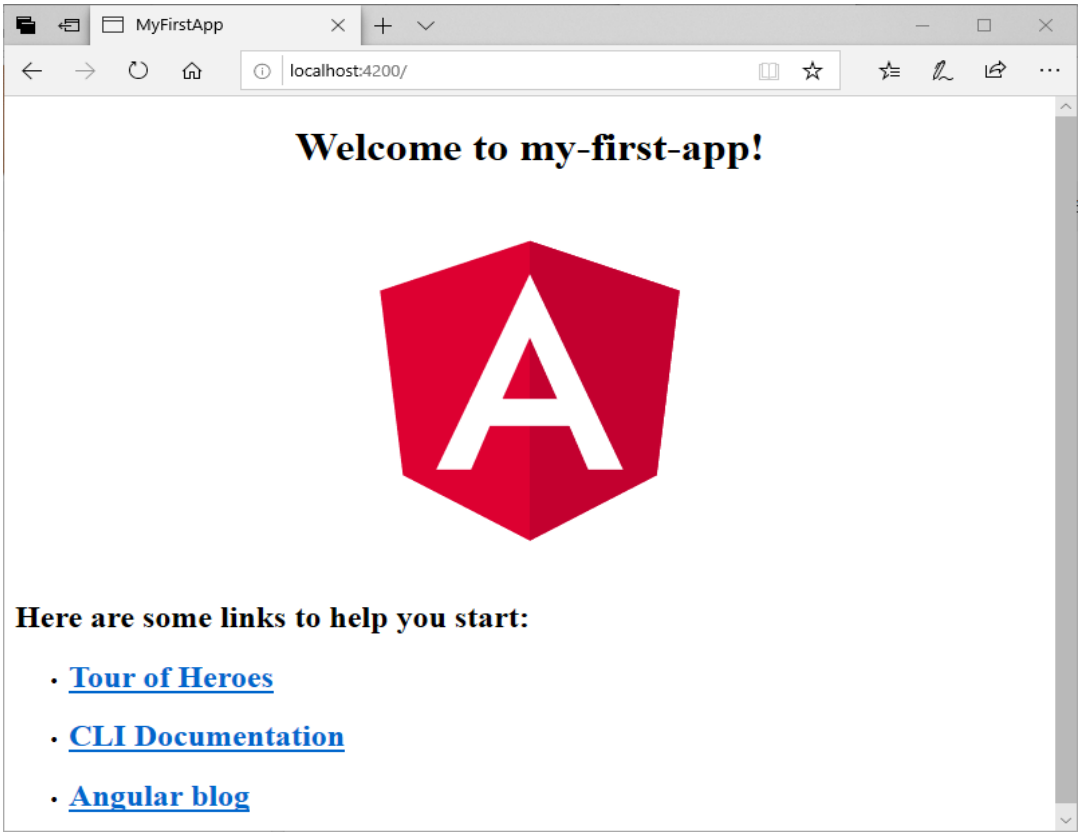
1. cd my-first-app

Start your server to run app.

1. ng serve

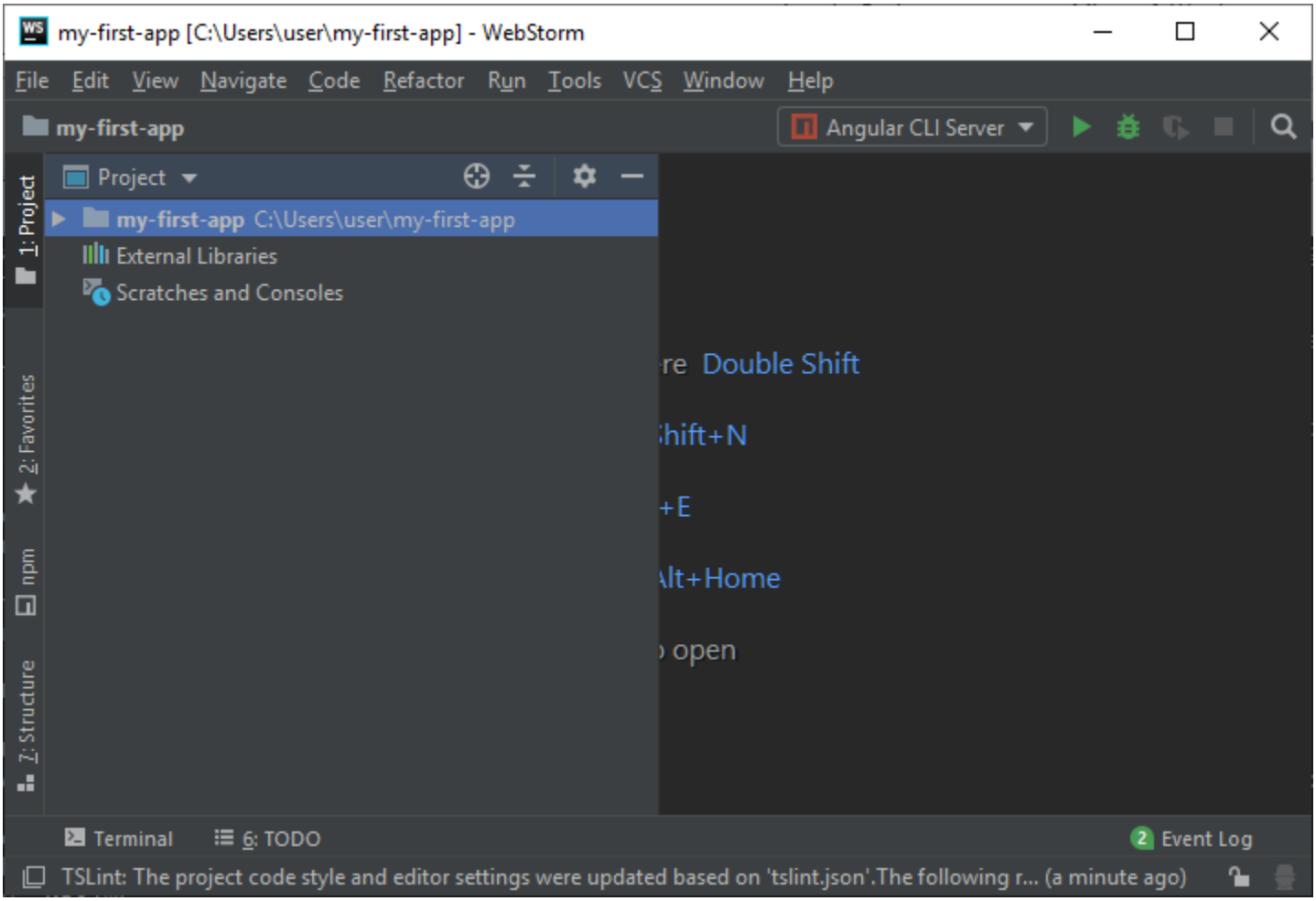


Your server is running on localhost:4200. Now, go to the browser and open it.

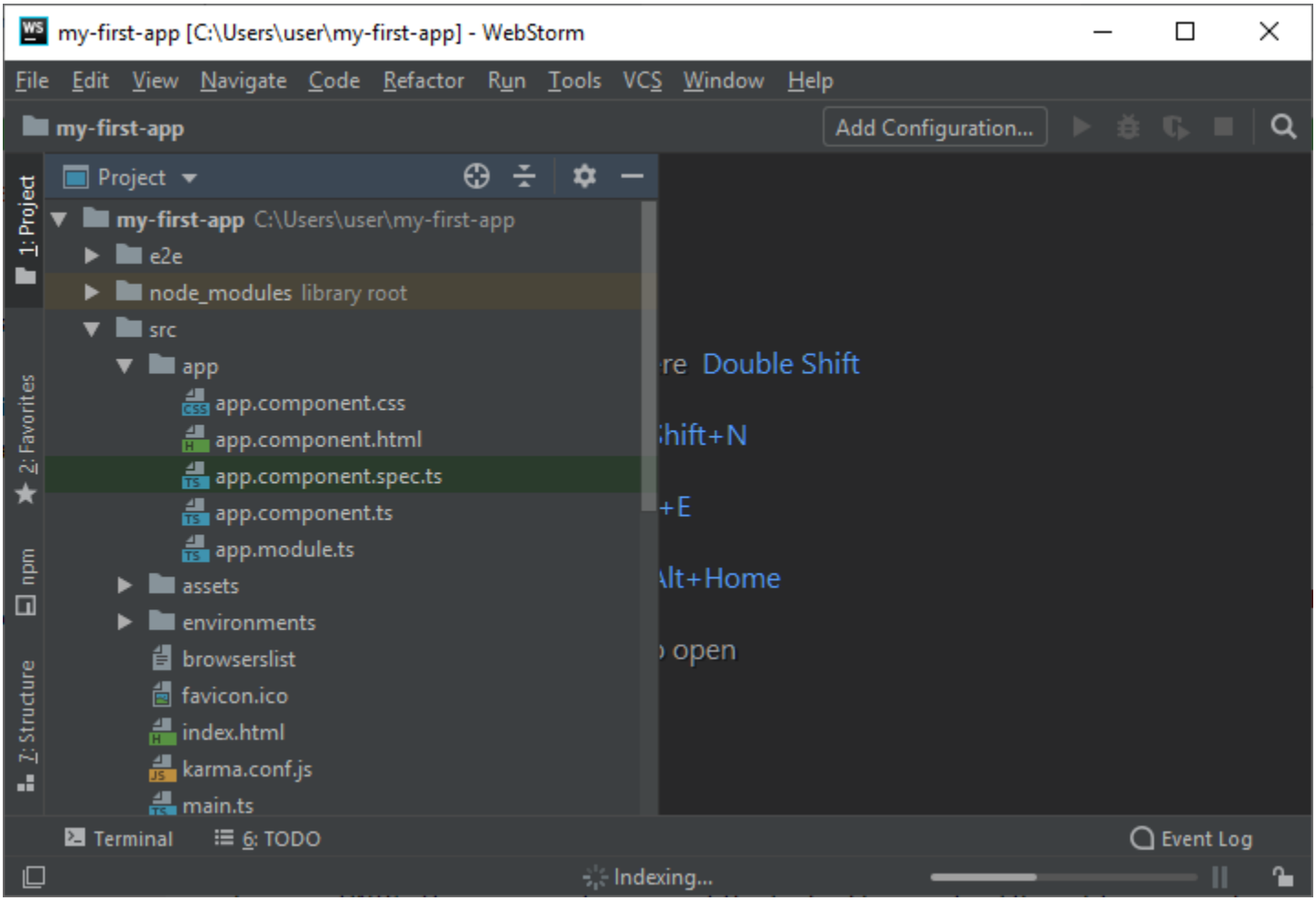


Now, you need an IDE to edit and run your app's code. Here, we are using WebStorm.

Open WebStorm and open your app "my-first-app" in the IDE. It will look like this:



Here, go to src folder, you will see app folder there. Expand the app folder.



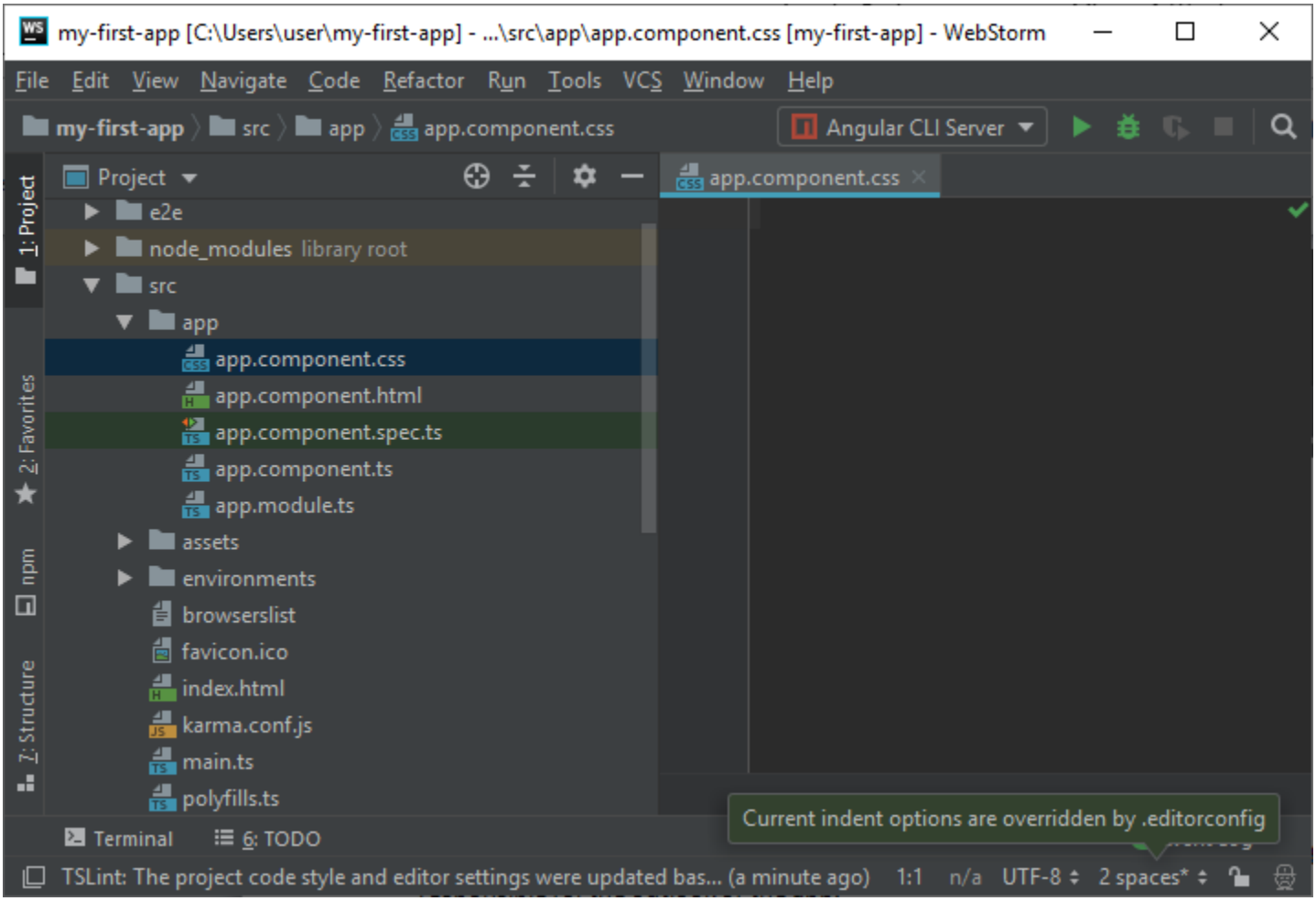
You will see 5 components there:

- app.component.css
- app.component.html
- app.component.spec.ts
- app.component.ts
- app.module.ts

You can see the code within the different components to understand what is going on and which part is responsible for the outlook of the app.

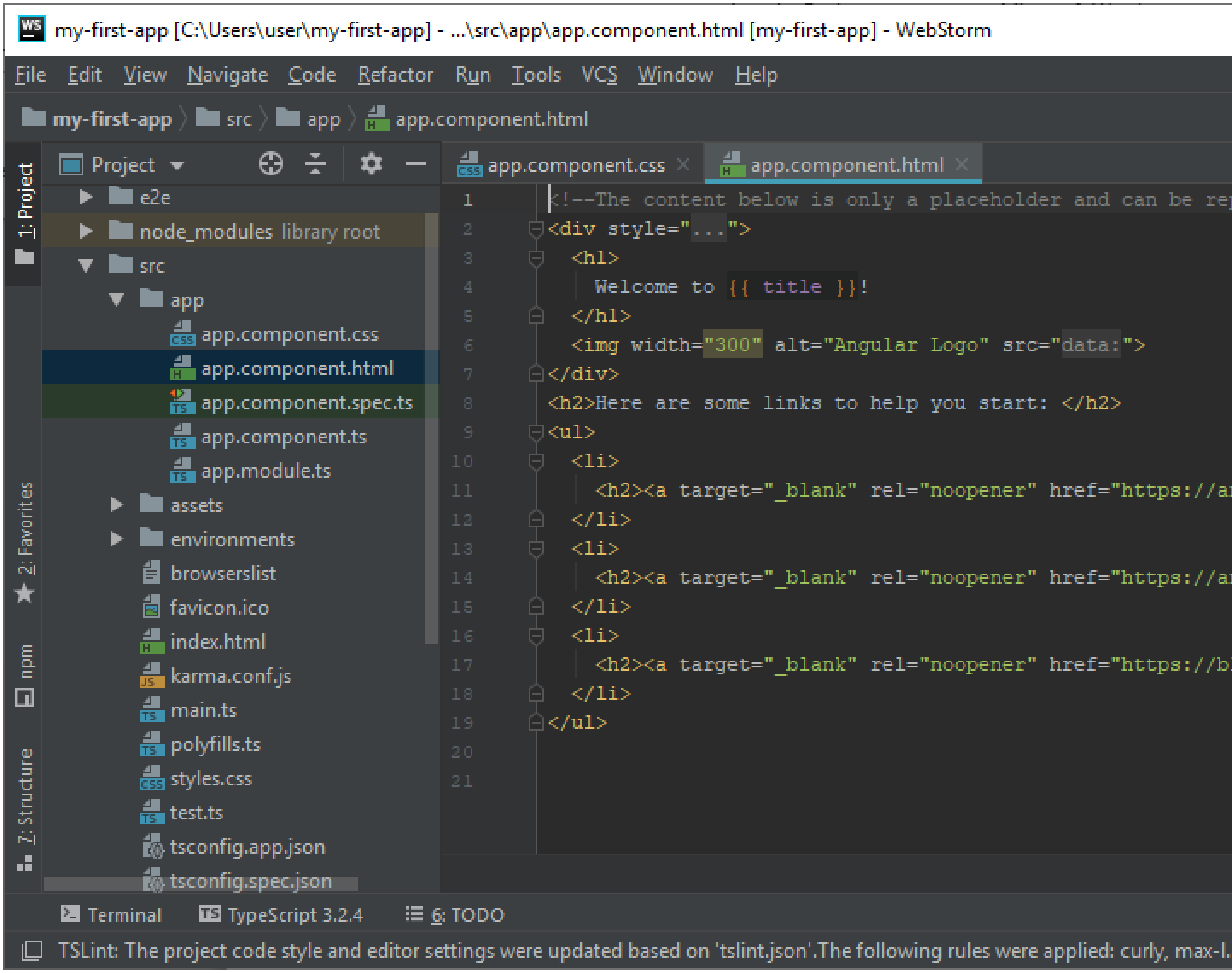
app.component.css

This part is empty because we don't specify any CSS here.



app.component.html

This is the most important component, the front page of your app. Here, you can change the salutation used before your app's name. You can also change the content on the front page and their respective links.



Code:

```
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
  <img width="300" alt="Angular Logo" src="
Wxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHZpZXdCb3g9IjAgMCAyNTAgMjU
wIj4KICAgIDxwYXRoIGZpbGw9IiNERDAwMzEiIGQ9Ik0xMjUgMzBMMzEuOSA2My4ybDE0
LjIgMTIzLjFMMTI1IDIzMGw3OC45LTQzLjcgMTQuMi0xMjMuMXoiIC8+CiAgICA8cGF0aCB
```

```
maWxsPSIjQzMwMDJGIiBkPSJNMTI1IDMwdjIyLjItLjFWMjMwbDc4LjktNDMuNyAxNC4yLTEyMy4xTDEyNSAzMHoiIC8+CiAgICA8cGF0aCAgZmlsbD0iI0ZGRkZGRiIgZD0iTTEyNSA1Mi4xTDY2LjggMTgyLjZoMjEuN2wxMS43LTI5LjJoNDkuNGwxMS43IDI5LjJIMTgzTDEyNSA1Mi4xem0xNyA4My4zaC0zNGwxNy00MC45IDE3IDQwLjl6IiAvPgogIDwvc3ZnPg==">
</div>
<h2>Here are some links to help you start: </h2>
<ul>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://angular.io/tutorial">Tour of Heroes</a></h2>
  </li>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://angular.io/cli">CLI Documentation</a></h2>
  </li>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://blog.angular.io/">Angular blog</a></h2>
  </li>
</ul>
```

app.component.spec.ts:

This file is used for testing purpose only.

```
import { TestBed, async } from '@angular/core/testing';
import { AppComponent } from './app.component';

describe('AppComponent', () => {
  beforeEach(async(() => {
    TestBed.configureTestingModule({
```

```
7.      declarations: [
8.        AppComponent
9.      ],
10    }).compileComponents();
11  }));
12
13  it('should create the app', () => {
14    const fixture = TestBed.createComponent(AppComponent);
15    const app = fixture.debugElement.componentInstance;
16    expect(app).toBeTruthy();
17  });
18
19  it(`should have as title 'my-first-app'`, () => {
20    const fixture = TestBed.createComponent(AppComponent);
21    const app = fixture.debugElement.componentInstance;
22    expect(app.title).toEqual('my-first-app');
23  });
24
25  it('should render title in a h1 tag', () => {
26    const fixture = TestBed.createComponent(AppComponent);
27    fixture.detectChanges();
28    const compiled = fixture.debugElement.nativeElement;
29    expect(compiled.querySelector('h1').textContent).toContain('Welcome to my-first-
app!');
30  });
31 });
```

app.component.ts

You can change the name of your app here. You just have to change the title.

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

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

app.module.ts

```
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 { }
```

All Angular CLI commands

Angular CLI is a command line interface tool which is used to initialize, develop, scaffold, and maintain Angular applications. You can use these command directly on command prompt or indirectly through an interactive UI i.e. Angular Console.

Command	Alias	Description
add		It is used to add support for an external library to your project.
build	b	It compiles an Angular app into an output directory named dist/ at the given output path. Must be executed from within a workspace directory.
config		It retrieves or sets Angular configuration values in the angular.json file for the workspace.
doc	d	It opens the official Angular documentation (angular.io) in a browser,

		and searches for a given keyword.
e2e	e	It builds and serves an Angular app, then runs end-to-end tests using Protractor.
generate	g	It generates and/or modifies files based on a schematic.
help		It provides a list of available commands and their short descriptions.
lint	l	It is used to run linting tools on Angular app code in a given project folder.
new	n	It creates a new workspace and an initial Angular app.
run		It runs an Architect target with an optional custom builder configuration defined in your project.

serve	s	It builds and serves your app, rebuilding on file changes.
test	t	It runs unit tests in a project.
update		It updates your application and its dependencies. See https://update.angular.io/
version	v	It utputs Angular CLI version.
xi18n		It extracts i18n messages from source code.

Angular 7 Architecture

Angular 7 is a platform and framework which is used to create client applications in HTML and TypeScript. Angular 7 is written in TypeScript.

Angular 7 implements core and optional functionality as a set of TypeScript libraries which you import in your app.

NgModules are the basic building blocks of an Angular 7 application. They provide a compilation context for components. An Angular 7 app is defined by a set of NgModules and NgModules collect related code into functional sets.

An Angular 7 app always has at least a root module which enables bootstrapping, and typically has many other feature modules.

- Components define views, which are the sets of screen elements that are chosen and modified according to your program logic and data by Angular 7.
- Components use services, which provide specific functionality not directly related to views. Service providers can be injected into components as dependencies, making your code modular, reusable, and efficient.

Components

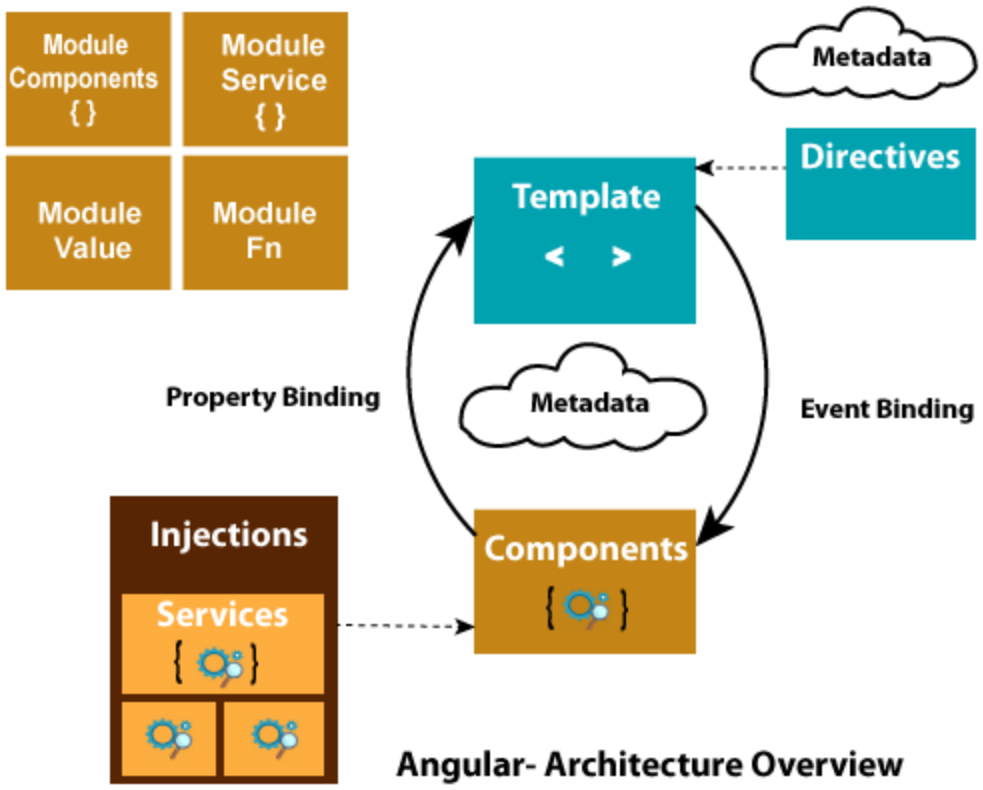
Components and services both are simply classes with decorators that mark their types and provide metadata which guide Angular to do things.

Every Angular application always has at least one component known as root component that connects a page hierarchy with page DOM. Each component defines a class that contains application data and logic, and is associated with an HTML template that defines a view to be displayed in a target environment.

Metadata of Component class:

- The metadata for a component class associates it with a template that defines a view. A template combines ordinary HTML with Angular directives and binding markup that allow Angular to modify the HTML before rendering it for display.
- The metadata for a service class provides the information Angular needs to make it available to components through dependency injection (DI).

Read more about Angular 7 component



Modules

Angular 7 NgModules are different from other JavaScript modules. Every Angular 7 app has a root module known as AppModule. It provides the bootstrap mechanism that launches the application.

Generally, every Angular 7 app contains many functional modules.

Some important features of Anngular 7 Modules:

- Angular 7 NgModules import the functionalities form other NgModules just like other JavaScript modules.
- NgModules allow their own functionality to be exported and used by other NgModules. For example, if you want to use the router service in your app, you can import the Router NgModule.

Template, Directives and Data Binding

In Angular 7, a template is used to combine HTML with Angular Markup and modify HTML elements before displaying them. Template directives provide

program logic, and binding markup connects your application data and the DOM.

There are two types of data binding:

- **Event Binding:** Event binding is used to bind events to your app and respond to user input in the target environment by updating your application data. Read more about event binding
- **Property Binding:** Property binding is used to pass data from component class and facilitates you to interpolate values that are computed from your application data into the HTML. Read more about property binding

Services and Dependency Injection

In Angular 7, developers create a service class for data or logic that isn't associated with a specific view, and they want to share across components.

Dependency Injection (DI) is used to make your component classes lean and efficient. DI doesn't fetch data from the server, validate user input, or log directly to the console; it simply renders such tasks to services.

Routing

In Angular 7, **Router** is an NgModule which provides a service that facilitates developers to define a navigation path among the different application states and view hierarchies in their app.

It works in the same way as a browser's navigation works. i.e.:

- Enter a URL in the address bar and the browser will navigate to that corresponding page.
- Click the link on a page and the browser will navigate to a new page.
- Click the browser's back or forward buttons and the browser will navigate backward or forward according to your seen history pages.

How does Router work?

The router maps URL-like paths to views instead of pages. Whenever a user performs an action, such as clicking a link that would load a new page in the browser, the router intercepts the browser's behavior, and shows or hides view hierarchies.

If the router finds that the current application state requires particular functionality, and the module that defines it hasn't been loaded, the router can lazy-load the module on demand.

The router interprets a link URL according to your app's view navigation rules and data state. You can navigate to new views when the user clicks a button or selects from a drop box, or in response to some other stimulus from any source. The router logs activity in the browser's history, so the back and forward buttons work as well.

To define navigation rules, you associate navigation paths with your components. A path uses a URL-like syntax that integrates your program data, in much the same way that template syntax integrates your views with your program data. You can then apply program logic to choose which views to show or to hide, in response to user input and your own access rules.

Angular 7 Components

Components are the key features of Angular. The whole application is built by using different components.

The core idea behind Angular is to build components. They make your complex application into reusable parts which you can reuse very easily.

How to create a new component?

Open WebStorm>> Go to your project source folder>> Expand the app directory and create a new directory named "server".

Now, create the component within server directory. Right click on the server directory and create a new file named as "server.component.ts". It is the newly created component.

Components are used to build webpages in Angular but they require modules to bundle them together. Now, you have to register our new components in module.

Creating component with CLI

Syntax

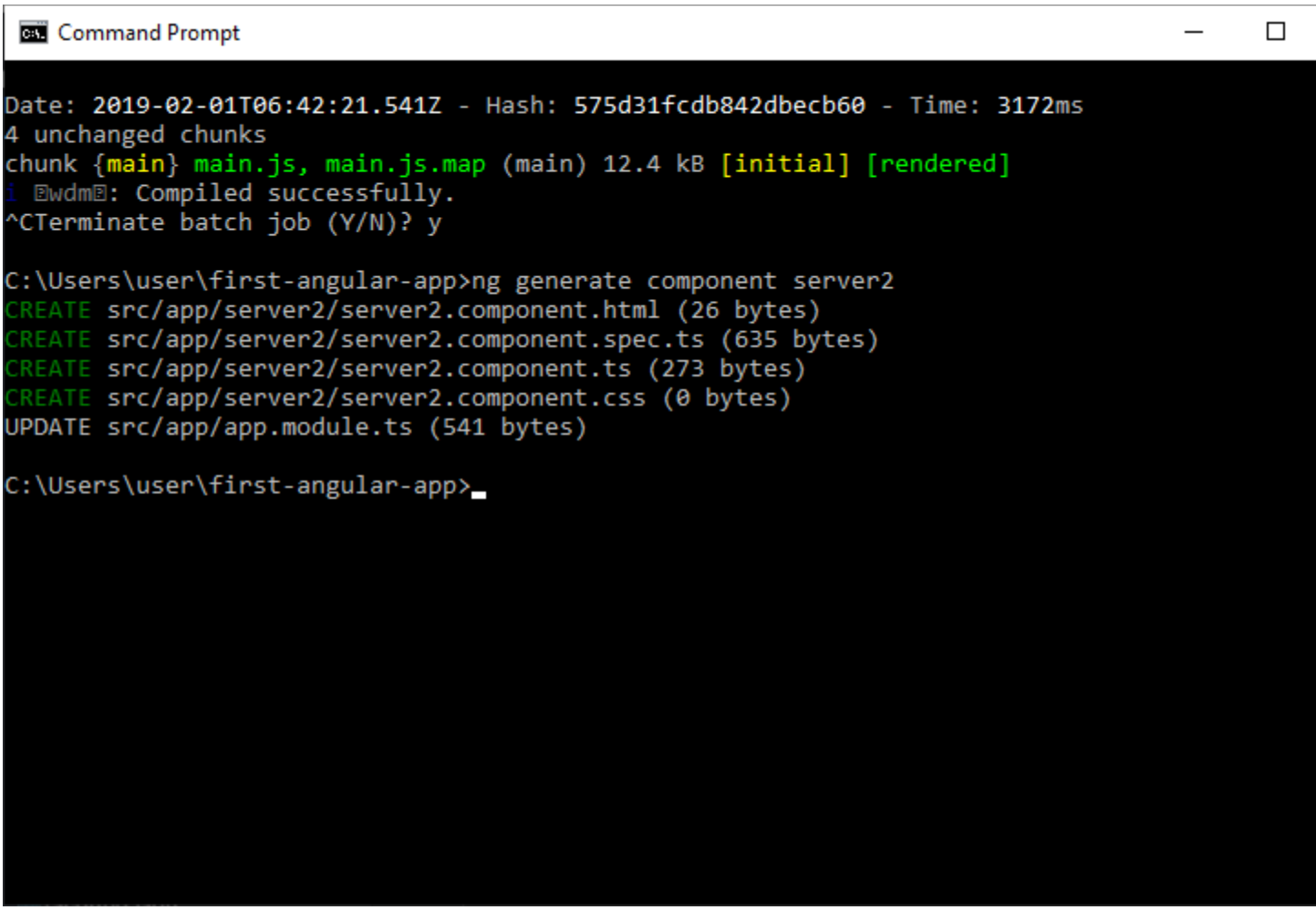
```
1. ng generate component component_name
2. Or
3. ng g c component_name
```

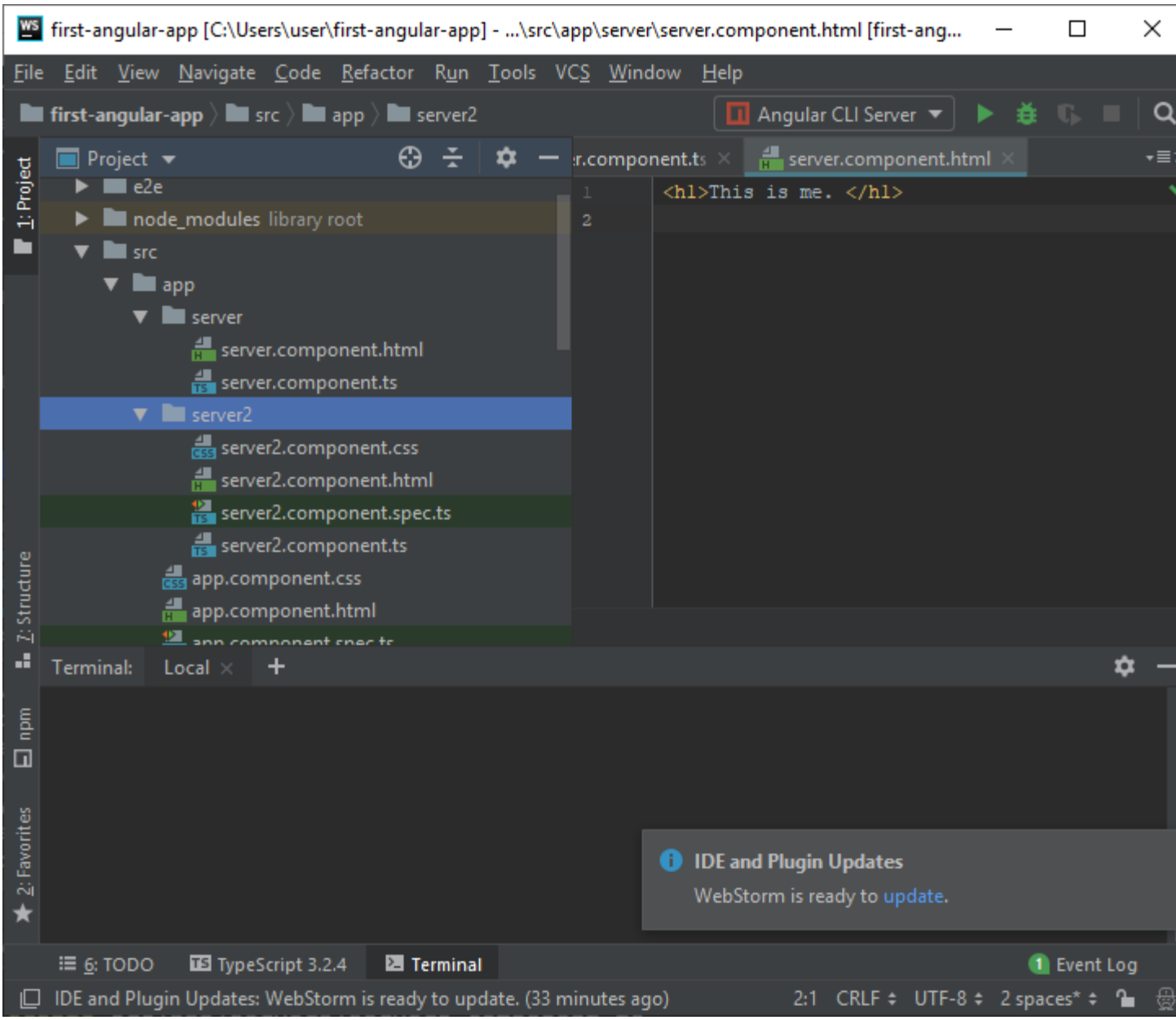
Let's see how to create a new component by using command line.

Open Command prompt and stop **ng serve** command if it is running on the browser.

Type **ng generate component server2** to create a new component named server2.

You can also use a shortcut **ng g c server2** to do the same task.


Command Prompt
Date: 2019-02-01T06:42:21.541Z - Hash: 575d31fcdb842dbecb60 - Time: 3172ms
4 unchanged chunks
chunk {main} main.js, main.js.map (main) 12.4 kB [initial] [rendered]
i ｢wdm｣: Compiled successfully.
^CTerminate batch job (Y/N)? y
C:\Users\user\first-angular-app>ng generate component server2
CREATE src/app/server2/server2.component.html (26 bytes)
CREATE src/app/server2/server2.component.spec.ts (635 bytes)
CREATE src/app/server2/server2.component.ts (273 bytes)
CREATE src/app/server2/server2.component.css (0 bytes)
UPDATE src/app/app.module.ts (541 bytes)
C:\Users\user\first-angular-app>



In the above image, you can see that a new component named "server2" is created. It contains the same other components which you have seen when we create a first app..

Angular 7 Directives

Directives are instructions in the DOM. They specify how to place your components and business logic in the Angular.

Directives are js class and declared as @directive. There are 3 directives in Angular.

- Component Directives
- Structural Directives
- Attribute Directives

Component Directives: Component directives are used in main class. They contain the detail of how the component should be processed, instantiated and used at runtime.

Structural Directives: Structural directives start with a * sign. These directives are used to manipulate and change the structure of the DOM elements. For example, *ngIf and *ngFor.

Attribute Directives: Attribute directives are used to change the look and behavior of the DOM elements. For example: ngClass, ngStyle etc.

Difference between Attribute Directive and Structural Directive

Attribute Directives	Structural Directives
Attribute directives look like a normal HTML Attribute and mainly used in databinding and event binding.	Structural Directives start with a * symbol and look different.
Attribute Directives affect only the element they are added to.	Structural Directives affect the whole area in the DOM.

Angular Databinding

Databinding is a powerful feature of Angular. Angular Databinding is used for communication. It is used to communicate between your TypeScript code (your business logic) and the other component which is shown to the users i.e. HTML layout.

Databinding is necessary because when we write the code in TypeScript, it is compiled to JavaScript and the result is shown on HTML layout. Thus, to show the correct and spontaneous result to the users, a proper communication is necessary. That's why databinding is used in Angular.

There is two type of databinding:

One-way databinding

One way databinding is a simple one way communication where HTML template is changed when we make changes in TypeScript code.

Or

In one-way databinding, the value of the Model is used in the View (HTML page) but you can't update Model from the View. Angular Interpolation / String Interpolation, Property Binding, and Event Binding are the example of one-way databinding.

Two-way databinding

In two-way databinding, automatic synchronization of data happens between the Model and the View. Here, change is reflected in both components. Whenever you make changes in the Model, it will be reflected in the View and when you make changes in View, it will be reflected in Model.

This happens immediately and automatically, ensures that the HTML template and the TypeScript code are updated at all times.

Angular 7 String Interpolation

In Angular, String interpolation is used to display dynamic data on HTML template (at user end). It facilitates you to make changes on component.ts file and fetch data from there to HTML template (component.html file).

For example:

component.ts file:

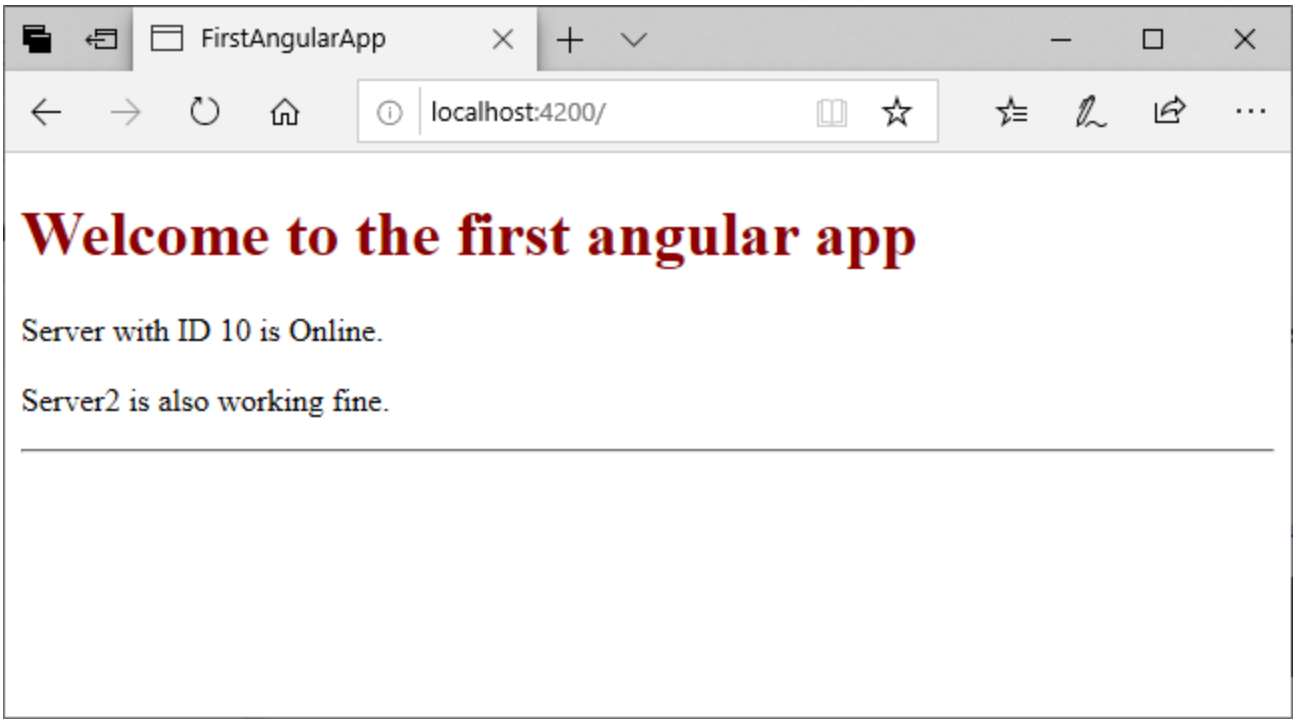
```
import {Component} from '@angular/core';
@Component(
  {selector: 'app-server',
 templateUrl: 'server.component.html'})
export class ServerComponent {
  serverID: number = 10;
    serverStatus: string = 'Online';
}
```

Here, we have specified serverID and serverStatus with some values. Let's use this in "component.html" file.

component.html file:

```
<p>Server with ID {{serverID}} is {{serverStatus}}. </p>
```

Output:



Property Binding

```
1. import { Component } from '@angular/core';
2. @Component({
3.     selector: 'my-app',
4.     template: `
5.                 <h1 [innerHtml]='fullName'></h1>
6.               `
7. })
8. export class AppComponent {
9.     fullName: string = 'Robert Junior';
10 }
```

In Property binding, see how Angular pulls the value from fullName property from the component and inserts it using the html property innerHtml of <h1> element.

Both examples for string interpolation and property binding will provide the same result.

Angular 7 Event Binding

Angular facilitates us to bind the events along with the methods. This process is known as event binding. Event binding is used with parenthesis ().

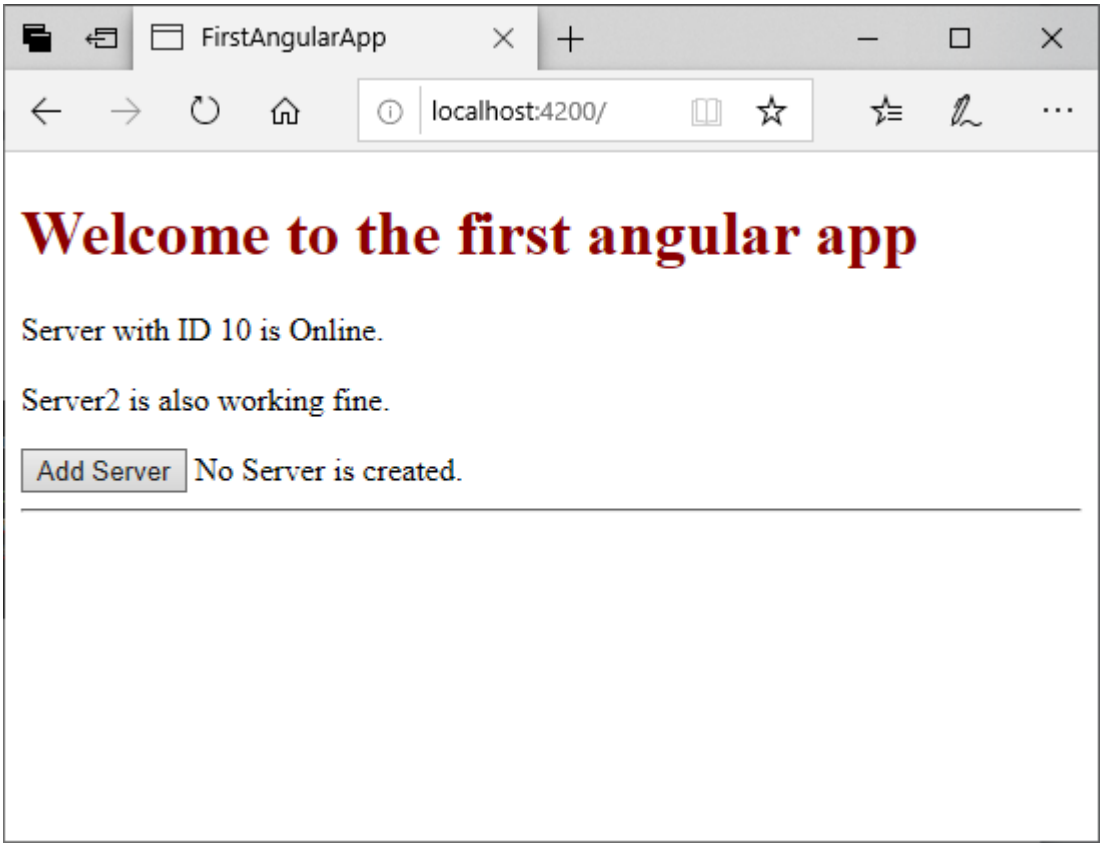
Let's see it by an example:

```
@Component({
  selector: 'app-server2',
  templateUrl: './server2.component.html',
  styleUrls: ['./server2.component.css']
})
export class Server2Component implements OnInit {
 allowNewServer = false;
 serverCreationStatus= 'No Server is created.';
  constructor() {
    setTimeout(() =>{
      this.allowNewServer = true;
    }, 5000);
  }

  ngOnInit() {
  }

}
```

component.html file:

```
<p>
  Server2 is also working fine.
```

```
</p>
<button class="btn btn-primary"
        [disabled]="!allowNewServer" >Add Server</button>
<!--<h3 [innerText]= "allowNewServer"></h3>-->

{{serverCreationStatus}}
```



It will give an output that "No Server is created".

Now, we are going to bind an event with button.

Add another method onCreateServer() in component.ts file which will call the event.

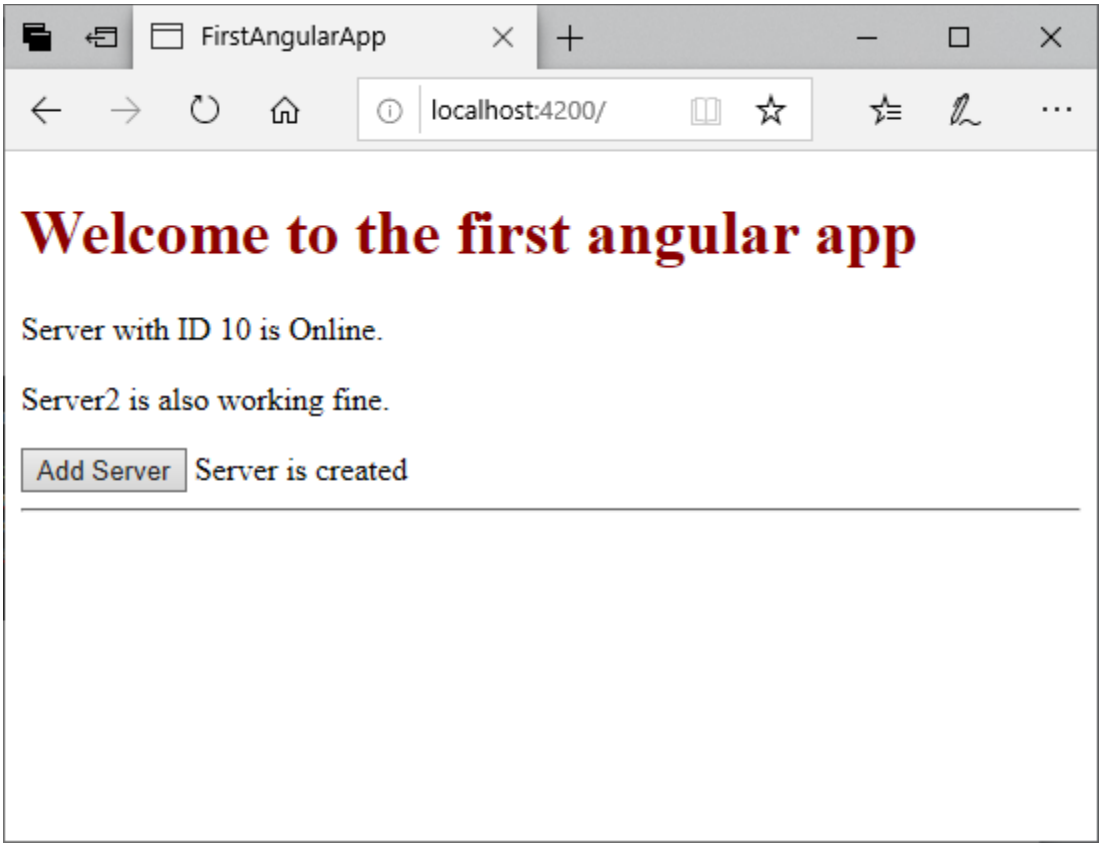
component.html file:

```
<p>
  Server2 is also working fine.
```

```
</p>
<button class="btn btn-primary"
        [disabled]="!allowNewServer"
(click)="onCreateServer()">Add Server</button>
<!--<h3 [innerText]= "allowNewServer"></h3>-->


{{serverCreationStatus}}
```

Output:

Now, after clicking on the button, you will see that it is showing server is created. This is an example of event binding.



Angular 7 Pipes

In Angular 1, filters are used which are later called Pipes onwards Angular2. In Angular 7, it is known as pipe and used to transform data. It is denoted by symbol |

Syntax:

```
{{title | uppercase}}
```

Pipe takes integers, strings, arrays, and date as input separated with |. It transforms the data in the format as required and displays the same in the browser.

Let's see an example using pipes. Here, we display the title text in upper and lower case by using pipes.

Example:

Define a variable named "title" in component.ts file.

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

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

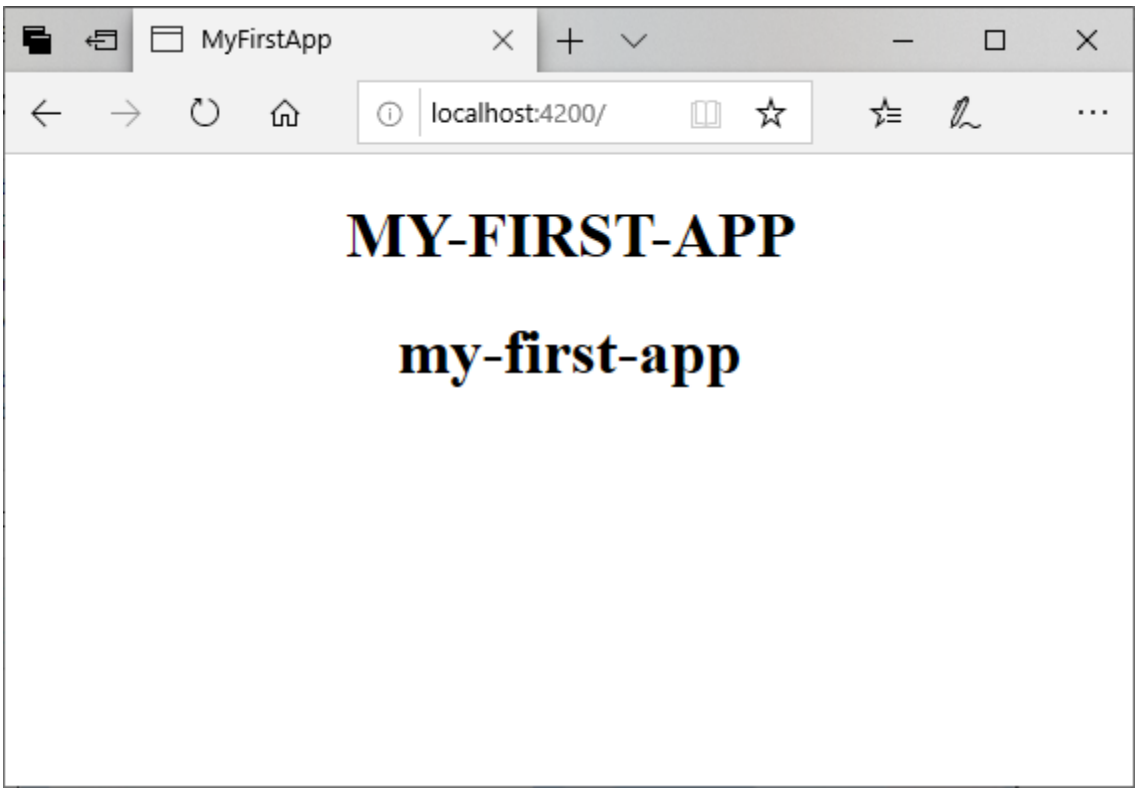
Use the pipe symbol in component.html file:

```
<h1>
   {{ title | uppercase }} <br/></h1>
<h1>
```

4. {{ title | lowercase }}
</h1>

Output:

Run ng serve and see the result. You will see the following result.



Here, you can see that pipes have changed the tittle in upper and lowercase.

Angular 7 Built-in Pipes

Angular 7 provides some built-in pipes:

- Lowercasepipe
- Uppercasepipe
- Datepipe
- Currencypipe
- Jsonpipe

- Percentpipe
- Decimalpipe
- Slicepipe

How to create a custom pipe?

To create a custom pipe, create a new ts file and use the code according to the work you have to do. You have to import Pipe, PipeTransform from Angular/Core. Let's create a sqrt custom pipe.

component.ts file:

```
1. import {Pipe, PipeTransform} from '@angular/core';
2. @Pipe ({
3.   name : 'sqrt'
4. })
5. export class SqrtPipe implements PipeTransform {
6.   transform(val : number) : number {
7.     return Math.sqrt(val);
8.   }
9. }
```

Now, it's turn to make changes in the app.module.ts. Create a class named as SqrtPipe. This class will implement the PipeTransform. The transform method defined in the class will take argument as the number and will return the number after taking the square root.

As, we have created a new file so, we need to add the same in app.module.ts.

Module.ts file:

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { AppComponent } from './app.component';
4. import { NewCmpComponent } from './new-cmp/new-cmp.component';
```

```
import { ChangeTextDirective } from './change-text.directive';
import { SqrtPipe } from './app.sqrt';
@NgModule({
  declarations: [
    SqrtPipe,
    AppComponent,
    NewCmpComponent,
    ChangeTextDirective
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Now, use sqrt pipe in component.html file.

component.html file:

```
<h1>Example of Custom Pipe</h1>
<h2>Square root of 625 is: {{625 | sqrt}}</h2><br/>
<h2>Square root of 169 is: {{169 | sqrt}}</h2>
```

Output:

```
Example of Custom Pipe

Square root of 625 is: {{625 | sqrt}}

Square root of 169 is: {{169 | sqrt}}
```

ANGULAR 7 | ANGULAR 8 HTTP INTERACTIONS

Now that we have created the project, before making of HttpClient to send HTTP requests let's first create the basic buildings of our demo application which are simply an `HttpService` that interfaces with the REST server and a bunch of components for making a CRUD interface that calls the methods of `HttpService`.

Creating an Angular Service

Let's start with the `HttpService`. In your terminal, run:

```
$ cd frontend
$ ng g s http
```

The command will generate the `src/app/http.service.spec.ts` (for tests) and `src/app/http.service.ts` files.

> ***Note**: Make sure you have navigated inside the `frontend` folder before running Angular CLI commands.*

Creating Angular Components

Next, let's create four components for displaying (list and by id) and creating/updating the customers:

```
$ ng g c customer-list
$ ng g c customer-details
$ ng g c customer-create
$ ng g c customer-update
```

Adding Routing & Navigation

To be able to navigate between these components, we need to add them to the router configuration. Open the `src/app/app-routing.module.ts` file and update it accordingly:

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


import { CustomerListComponent } from './customer-list/customer-
list.component';
import { CustomerDetailsComponent } from './customer-details/customer-
details.component';
```

```
import { CustomerCreateComponent } from './customer-create/customer-
create.component';
import { CustomerUpdateComponent } from './customer-update/customer-
update.component';



const routes: Routes = [
  { path:  '', pathMatch:  'full', redirectTo:  'list'},
  { path: 'list', component: CustomerListComponent},
  { path: 'details/:id', component: CustomerDetailsComponent},
  { path: 'create', component: CustomerCreateComponent},
  { path: 'update', component: CustomerUpdateComponent},
];


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

Example of Making HTTP GET Requests using HttpClient in Angular 8|7

In this section we suppose that we have a component that displays a list of customers from a server.

First let's see the required steps:

- Import *HttpClient* from *@angular/common/http*
- Inject HttpClient via component constructor
- Make HTTP GET Requests using .get(endpoint) method
- Subscribe to the returned observable and show results

Here is the source code of our example:

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


import { Observable } from  "rxjs/Observable";
```

```
import { HttpClient } from  "@angular/common/http";



class  Customer {

id : number;

name: string;

email: string;

tel: string;

}



@Component({

selector:  'customers',

template:  `

<ul *ngIf="customersObservable | async as customers else empty">

<li *ngFor="let customer of customers">



</li>

</ul>
```

```
<ng-template #empty> No Customers Yet </ng-template>

`})

export  class  CustomerComponent  implements  OnInit {

customersObservable : Observable<Customer[]>;



constructor(private  httpClient:HttpClient) {}



ngOnInit() {

this.customersObservable = this.httpClient

.get<Customer[]>("127.0.0.1:3000/customers")

.do(console.log);

}

}
```

HTTP GET Request Parameters: HttpParams

In many situations, we need to feed some HTTP parameters to the API endpoint we are querying. In this section we'll see how to use the *HttpParams* class to use parameters in the *HttpClient* module.

For instance, let's suppose that we need to make a GET request to this http://127.0.0.1:3000/customers?_page=1&_limit=1 URL for getting the first two customers of the first page.

We start by importing the *HttpParams* class using:

```
import {HttpParams} from  "@angular/common/http";
```

Next, we create an instance of the *HttpParams*class:

```
const  params = new  HttpParams().set('_page', "1").set('_limit', "1");
```

Finally, we call *httpClient.get()* method with these parameters, then assign the returned Observable to the *customersObservable*variable:

```
this.customersObservable =
this.httpClient.get("http://127.0.0.1:3000/customers", {params});
```

Using fromString to easily create HttpParams

We can also build HTTP parameters directly from a query string, for example for our previous example
URL http://127.0.0.1:3000/customers?_page=1&_limit=1 we can create an instance of *HttpParams* class from the query string `_page=1&_limit=1` by simply using the *fromString* variable:

```
const  params = new  HttpParams({fromString:  '_page=1&_limit=1'});
```

Generic HttpClient `request()` method

We have previously seen how to use the `.get()` method to send HTTP GET requests. Now we'll see a generic method to

send GET and the other HTTP methods such as POST, PUT and Delete etc.

Using the `.request()` method of the *HttpClient* module we can re-write our previous example to the following code:

```
const  params = new  HttpParams({fromString:  '_page=1&_limit=1'});
```

```
this.customersObservable =
this.http.request("GET","http://127.0.0.1:3000/customers",{responseType:"json",params});
```

Adding custom HTTP Headers to requests

We can also add custom HTTP headers to our HTTP requests using the *HttpHeaders* class.

First create an instance of the *HttpHeaders*class and then set your custom HTTP header. For example:

```
const  headers = new  HttpHeaders().set("X-CustomHttpHeader",
"CUSTOM_VALUE");
```

Next, you can send the GET request using:

```
this.customersObservable =
this.httpClient.get("http://127.0.0.1:3000/customers", {headers});
```

Sending HTTP PUT Requests in Angular 8

The HTTP PUT method is used to completely replace a resource on the API server. We can use the *HttpClient* module to send a PUT request to an API server using the the `put()`method. For example:

```
this.httpClient.put("http://127.0.0.1:3000/customers/1",

{

"name":  "NewCustomer001",

"email":  "newcustomer001@email.com",
```

```
"tel":  "0000252525"

})

.subscribe(

data  => {

console.log("PUT Request is successful ", data);

},

error  => {

console.log("Rrror", error);

}

);
```

Sending HTTP PATCH Requests

The HTTP PATCH method is used to update a resource on the server. The *HttpClient* class provides the `patch()` method tha can be used to send UPDATE requests. For example:

```
this.httpClient.patch("http://127.0.0.1:3000/customers/1",

{

"email":  "newcustomer001@email.com"

}).subscribe(

data  => {
```

```
console.log("PUT Request is successful ", data);

},

error  => {

console.log("Error", error);

}

);
```

Sending HTTP DELETE Requests

Now let's see an example of how we can send an HTTP DELETE request to delete a resource from the API server using `delete()` method provided by the *HttpClient* class:

```
this.httpClient.patch("http://127.0.0.1:3000/customers/1")

.subscribe(

data  => {

console.log("PATCH Request is successful ", data);

},

error  => {

console.log("Error", error);

}

);
```

Sending HTTP POST Requests in Angular 8

The HTTP POST method has many uses but mostly used when we need to add new data on the server so let's take an example of adding a new customer to our REST API server database using the `post()` method of the *HttpClient* class:

```
this.httpClient.post("http://127.0.0.1:3000/customers",
{
"name":  "Customer004",
"email":  "customer004@email.com",
"tel":  "0000252525"
})
.subscribe(
data  => {
console.log("POST Request is successful ", data);
},
error  => {

console.log("Error", error);

}

);
```

We are calling the `post()` method from the injected instance of *HttpClient*. The first parameter is the API endpoint and the second parameter is the *customer* data object. We also subscribe to the observable returned by the `post()` method. If the operation is successful we display *POST Request is successful* and the data on the console. If there is an error we log the error on the console