R.M.K GROUP OF ENGINEERING INSTITUTIONS

R.M.K
GROUP OF
INSTITUTIONS

R.M.K GROUP OF INSTITUTIONS

R.M.K GROUP OF INSTITUTIONS

22CS402

Web Development Frameworks

Department : Computer Science & Engineering

Batch / Year : 2022 - 2026 / II

Created by : All Subject Staff

Date 25.02.2024

1. Contents

Content – Unit I

S.No.	Contents
1	Introduction- Angular First App,
2	Angular UI with Bootstrap CSS
3	Authentication, Authentication Service
4	- Authentication Service-Unsubscribe
5	Logout and Route Guard Cleanup
6	Customer Service
7	Http Service-Token Interceptor
8	Multi Provider, Compile-time Configuration
9	Runtime Configuration, Error Handling.

R.M.K GROUP OF INSTITUTIONS

2. Course Objectives

The Course will enable learners to:

- ❖ Simplify website development using Spring Boot as server-side technologies.
- ❖ Build single page applications using REACT as a reusable UI component technology as client-side technology.
- ❖ Assemble REACT as a front end technology and Node js as a server side technology to develop enterprise applications
- ❖ Develop a scalable and responsive web application
- ❖ Develop an industry ready application web enterprise feature

3. Prerequisites

22CS402
Web Development Frameworks

↑

22CS301
Advanced Java Programming

↑

22CS202
Java Programming

↑

22CS101 Problem Solving Using C++
22CS102 Software Development Practices

4. Syllabus

22CS301	Web Development Frameworks	L	T	P	C
		3	0	2	4

OBJECTIVES:

The Course will enable learners to:

- ❖ Simplify website development using Spring boot as server-side technologies.
- ❖ Build single page applications using REACT as a reusable UI component technology as client-side technology.
- ❖ Assemble REACT as a front end technology and Node js as a server side technology to develop enterprise applications
- ❖ Develop a scalable and responsive web application
- ❖ Develop an industry ready application web enterprise feature

UNIT I	SPRING BOOT AND STRUTS	9 +6

Spring Boot: Introducing Spring Boot, getting started with spring Boot, Common Spring Boot task-Managing configuration, creating custom properties, executing code on Spring Boot application startup, Database access with Spring data, Securing Spring Boot application.

List of Exercise/Experiments:

1. Use Spring Boot to build a Web Application
2. Create REST Service for an Education Site

UNIT II	JAVA REACT	9 +6

React: Introduction to React, Pure React- The Virtual DOM, React Elements, React with JSX, Props, State, and the Component Tree, Enhancing Components- Flux.

List of Exercise/Experiments:

1. Build Search filter in React
2. Display a list in React
3. Create Simple Login form in React

UNIT III	Node JS	9 +6

Node JS: Introduction to Node JS, Setting up Node.js, Node.js Modules- Finding and loading CommonJS and JSON modules using require, Hybrid CommonJS/Node.js/ES6 module scenarios, npm - the Node.js package management system.

List of Exercise/Experiments:

1. Write a node.js program for making external http calls
2. Write a program in node.js to parse the given url.

4. Syllabus Contd…

UNIT IV	WEB FRAMEWORK (ANGULAR) – I	9+6
Introduction- Angular First App, Angular UI with Bootstrap CSS Authentication, Authentication Service, Unsubscribe, Logout and Route Guard Cleanup, Customer Service ,Http Service, Token Interceptor, Multi Provider, Compile-time Configuration, Runtime Configuration, Error Handling. **List of Exercise/Experiments:** 1. Create a Dropdown using Angular UI bootstrap 2. Modify existing components and generating new components using Angular		

UNIT V	WEB FRAMEWORK (ANGULAR) – II	9+6
Dependancy injection in Angular,Reactive programming in Angular, Laying out pages with Flex Layout, Implementing component communications, Change detection and component lifecycle. **List of Exercise/Experiments:** 1. Launching your app with Angular root module		

4. Syllabus Contd...

OUTCOMES:

Upon completion of the course, the students will be able to:

CO1: Write Web API/RESTful API application programming interface to communicate with Spring boot as a serverside technology.

CO2: Build single page applications using REACT as a reusable UI component technology as client side technology

CO3: Build applications using Node Js as server side technologies

CO4: Able to develop a web application using latest Angular Framework

CO5: Apply various Angular features including directives, components, and services.

TEXT BOOK:

1. Somnath Musib, Spring Boot in Practice, Manning publication, June 2022 (https://www.manning.com/books/spring-boot-in-practice)
2. Alex Banks, Eve Porcello , "Learning React", May 2017, O'Reilly Media, Inc. ISBN: 9781491954621. (https://www.oreilly.com/library/view/learning- react/9781491954614/)
3. David Herron ,"Node.js Web Development - Fourth Edition",2018, Packt Publishing,ISBN: 9781788626859
4. Sukesh Marla, "A Journey to Angular Development Paperback ", BPB Publications.(https://in.bpbonline.com/products/a-journey-to-angular-development?_pos=1&_sid=0a0a0e9fb&_ss=r)
5. Yakov Fain Anton Moiseev, "Angular Development with TypeScript", 2nd Edition.(https://www.manning.com/books/angular-development-with-typescript-Second Edition.

REFERENCES:

1. Sue Spielman, The Struts Framework 1: A Practical guide for Java Programmers||, 1st Edition. Elsevier 2002

LIST OF EQUIPMENTS:

VSCode, Angular JS, React JS, Node JS, Ruby, Django

5. Course Outcomes

Upon completion of the course, the students will be able to:

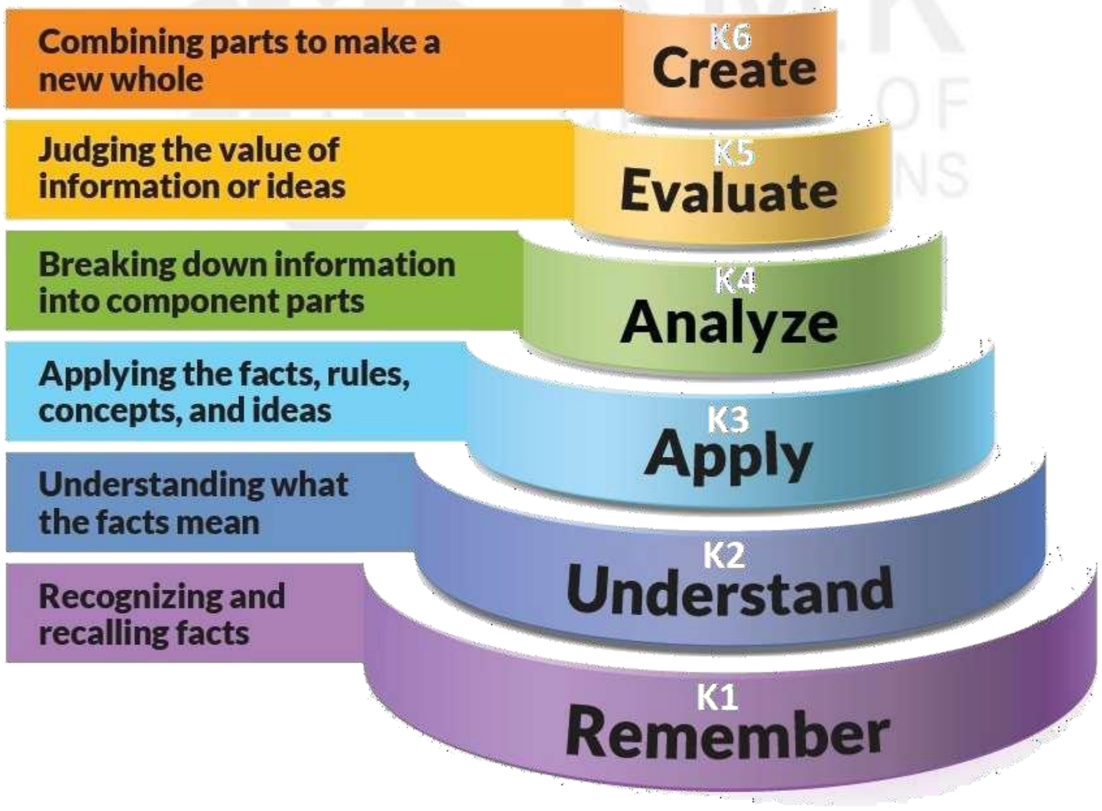
CO1: Write Web API/RESTful API application programming interface to communicate with Spring boot as a serverside technology.

CO2: Build single page applications using REACT as a reusable UI component technology as client side technology

CO3: Build applications using Node Js as server side technologies

CO4: Able to develop a web application using latest Angular Framework

CO5: Apply various Angular features including directives, components, and services.


Combining parts to make a new whole
K6
Create
Judging the value of information or ideas
K5
Evaluate
Breaking down information into component parts
K4
Analyze
Applying the facts, rules, concepts, and ideas
K3
Apply
Understanding what the facts mean
K2
Understand
Recognizing and recalling facts
K1
Remember

6. CO - PO Mapping

	POs and PSOs														
COs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	2	2	2	2	-	-	-	-	-	-	-	2	2	-
CO2	2	3	3	2	2	-	-	-	-	-	-	-	2	2	-
CO3	2	2	2	2	2	-	-	-	-	-	-	-	2	2	-
CO4	1	3	2	3	3	-	-	-	-	-	-	-	2	2	-
CO5	1	2	2	1	1	-	-	-	-	-	-	-	2	2	-

7. Lecture Plan - Unit IV

S. No.	Topic	No. of Periods	Proposed Date	Actual Lecture Date	Pertaining CO	Taxonomy Level	Mode of Delivery
1	Introduction- Angular First App,	1	12-03-2024		CO4	K2	Chalk & Talk
2	Angular UI with Bootstrap CSS	1	13-03-2024		CO4	K3	Chalk & Talk
3	Authentication, Authentication Service	1	16-03-2024		CO4	K2	Chalk & Talk
4	- Authentication Service-Unsubscribe	1	19-03-2024		CO4	K2	Chalk & Talk
5	Logout and Route Guard Cleanup	1	20-03-2024		CO4	K2	Chalk & Talk
6	Customer Service	1	23-03-2024		CO4	K2	Chalk & Talk
7	Http Service-Token Interceptor	1	26-03-2024		CO4	K3	Chalk & Talk
8	Multi Provider, Compile-time Configuration	1	30-03-2024		CO4	K2	Chalk & Talk
9	Runtime Configuration, Error Handling.	1	03-01-2024		CO4	K2	Chalk & Talk

R.M.K GROUP OF INSTITUTIONS

8. Activity Based Learning

Learning Method	Activity
Learn by Solving Problems	Tutorial Sessions available in iamneo Portal
Learn by Questioning	Quiz / MCQ Using RMK Nextgen App and iamneo Portal
Learn by doing Hands-on	Practice available in iamneo Portal

9. Lecture Notes

Introduction- Angular First App, Angular UI with Bootstrap CSS Authentication, Authentication Service, Unsubscribe, Logout and Route Guard Cleanup, Customer Service ,Http Service, Token Interceptor, Multi Provider, Compile-time Configuration, Runtime Configuration, Error Handling.

4.1 Angular Features

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.

Install Angular 7

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

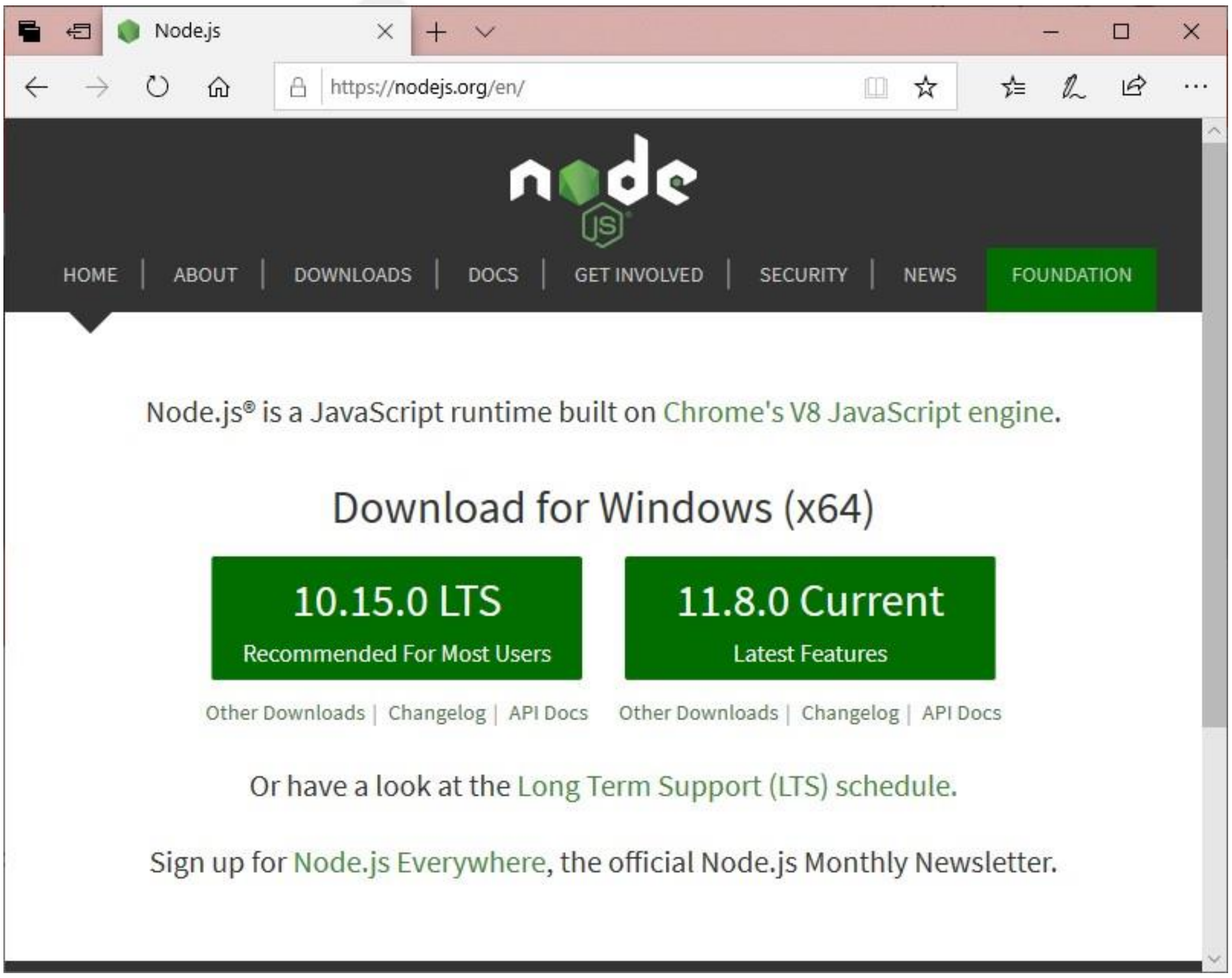
You should install node.js to run your Angular 7 app. It manages npm dependencies support some browsers when loading particular pages. It provides required libraries to run Angular project. Node.js serves your run-time environment as your localhost.

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


Node.js
https://nodejs.org/en/
node JS
HOME | ABOUT | DOWNLOADS | DOCS | GET INVOLVED | SECURITY | NEWS | FOUNDATION

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

Download for Windows (x64)

10.15.0 LTS
Recommended For Most Users

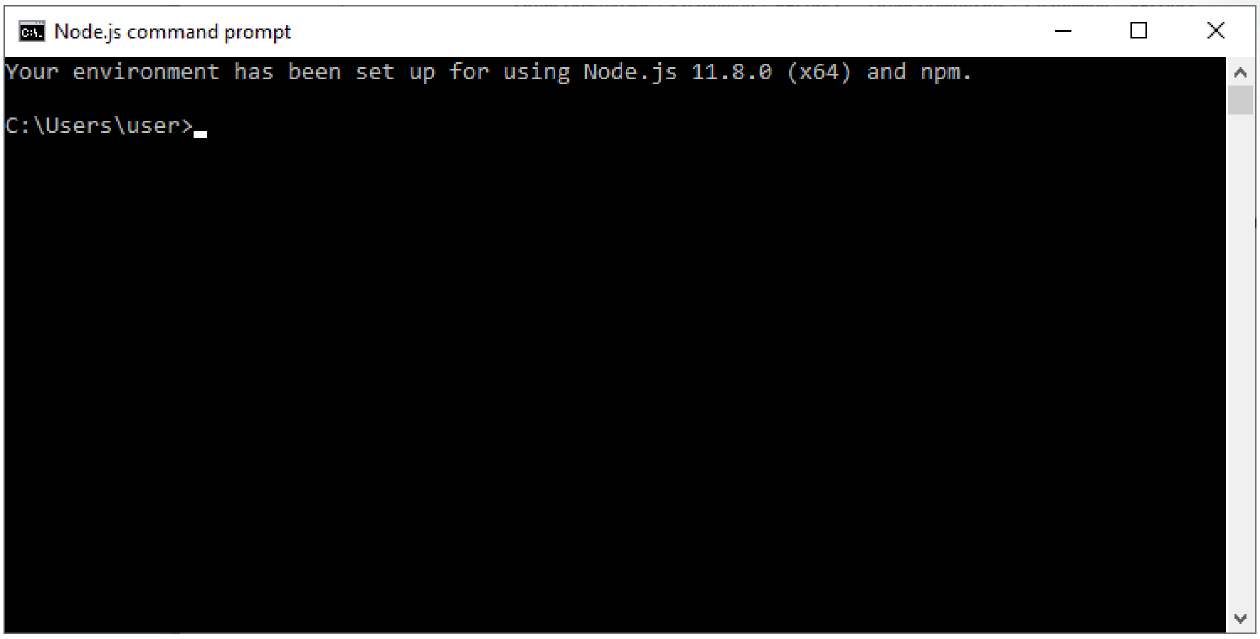
11.8.0 Current
Latest Features

Other Downloads | Changelog | API Docs
Other Downloads | Changelog | API Docs

Or have a look at the Long Term Support (LTS) schedule.

Sign up for Node.js Everywhere, the official Node.js Monthly Newsletter.

After the successful installation, you will see command prompt like this:

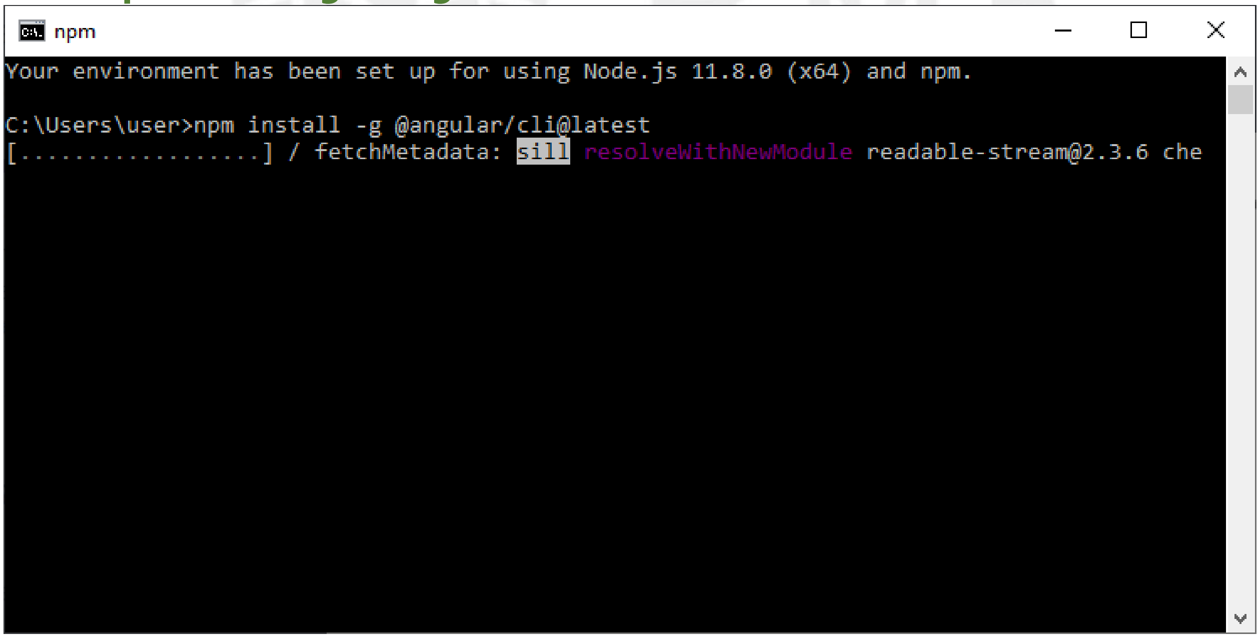


```
Your environment has been set up for using Node.js 11.8.0 (x64) and npm.

C:\Users\user>_
```

Use npm to install Angular CLI

Run the Angular CLI command to install Angular CLI

1. **npm install -g @angular/cli**

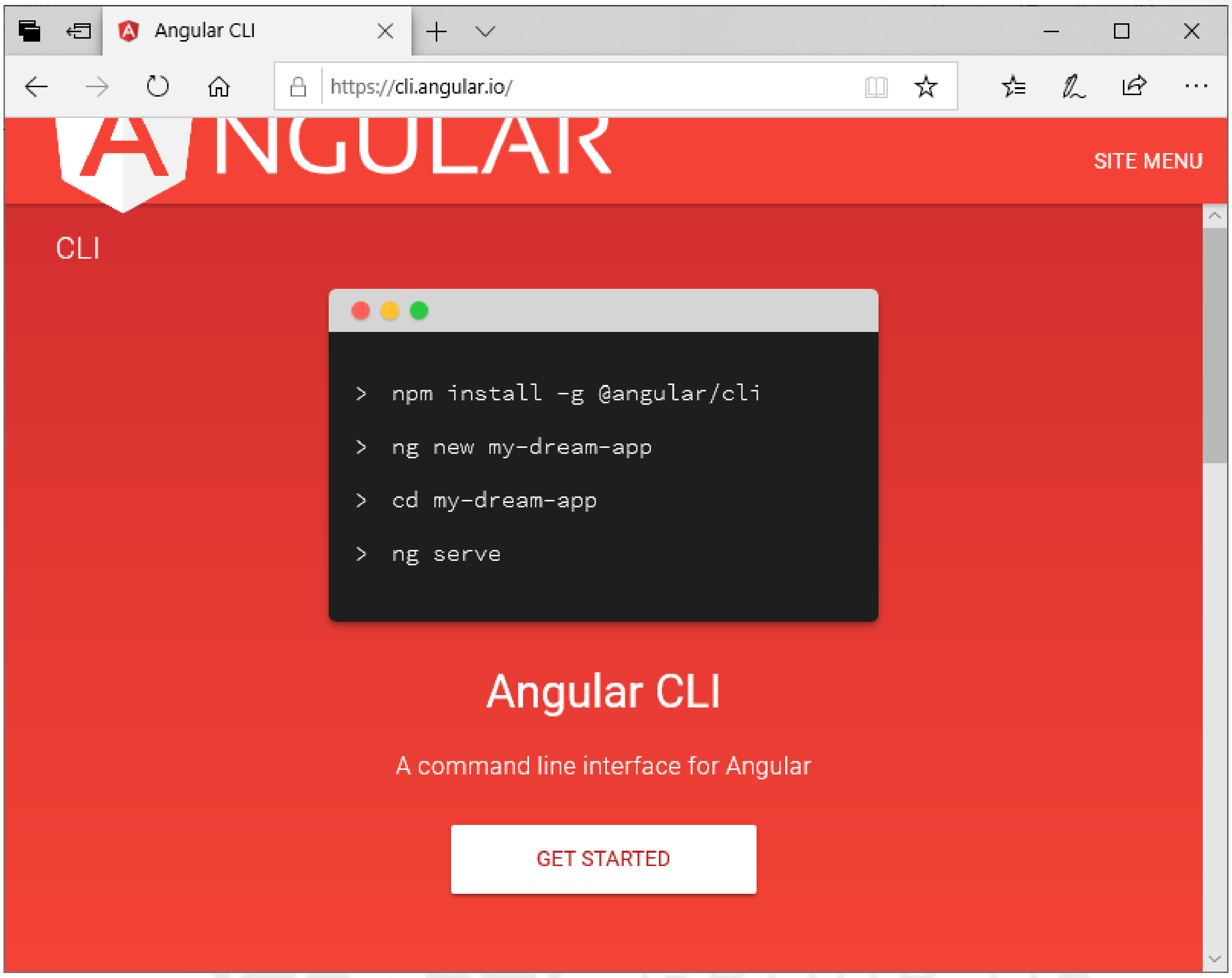


```
Your environment has been set up for using Node.js 11.8.0 (x64) and npm.

C:\Users\user>npm install -g @angular/cli@latest
[.................] / fetchMetadata: sill resolveWithNewModule readable-stream@2.3.6 che
```

or

Just go to Angular CLI official website **https://cli.angular.io/**

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.



```
> npm install -g @angular/cli
> ng new my-dream-app
> cd my-dream-app
> ng serve
```

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.

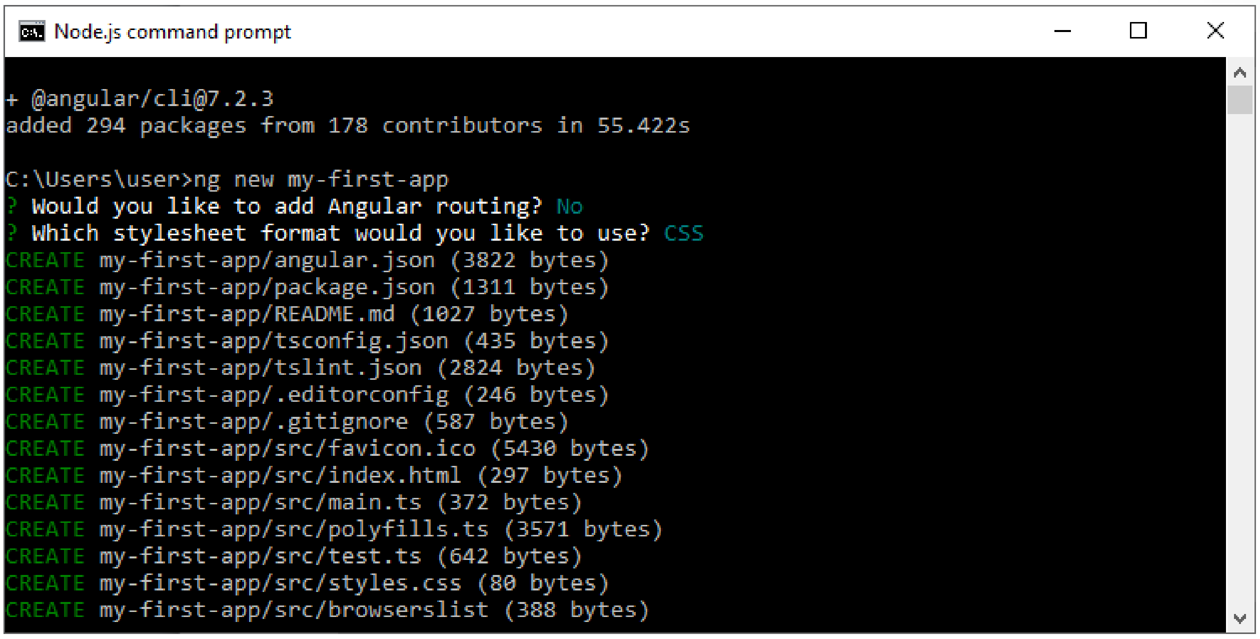
Angular 7 Project Setup (Create first app)

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



Node.js command prompt

```
+ @angular/cli@7.2.3
added 294 packages from 178 contributors in 55.422s

C:\Users\user>ng new my-first-app
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? CSS
CREATE my-first-app/angular.json (3822 bytes)
CREATE my-first-app/package.json (1311 bytes)
CREATE my-first-app/README.md (1027 bytes)
CREATE my-first-app/tsconfig.json (435 bytes)
CREATE my-first-app/tslint.json (2824 bytes)
CREATE my-first-app/.editorconfig (246 bytes)
CREATE my-first-app/.gitignore (587 bytes)
CREATE my-first-app/src/favicon.ico (5430 bytes)
CREATE my-first-app/src/index.html (297 bytes)
CREATE my-first-app/src/main.ts (372 bytes)
CREATE my-first-app/src/polyfills.ts (3571 bytes)
CREATE my-first-app/src/test.ts (642 bytes)
CREATE my-first-app/src/styles.css (80 bytes)
CREATE my-first-app/src/browserslist (388 bytes)
```

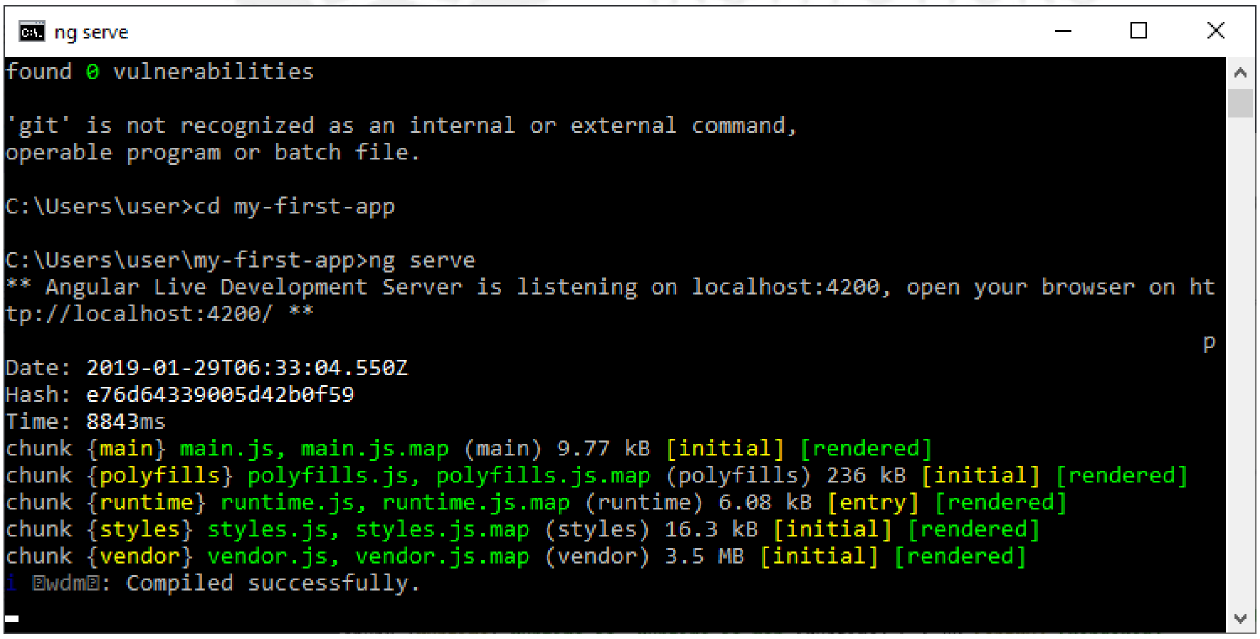
Navigate to your first app.

1. cd my-first-app

Start your server to run app.

Backward Skip 10sPlay VideoForward Skip 10s

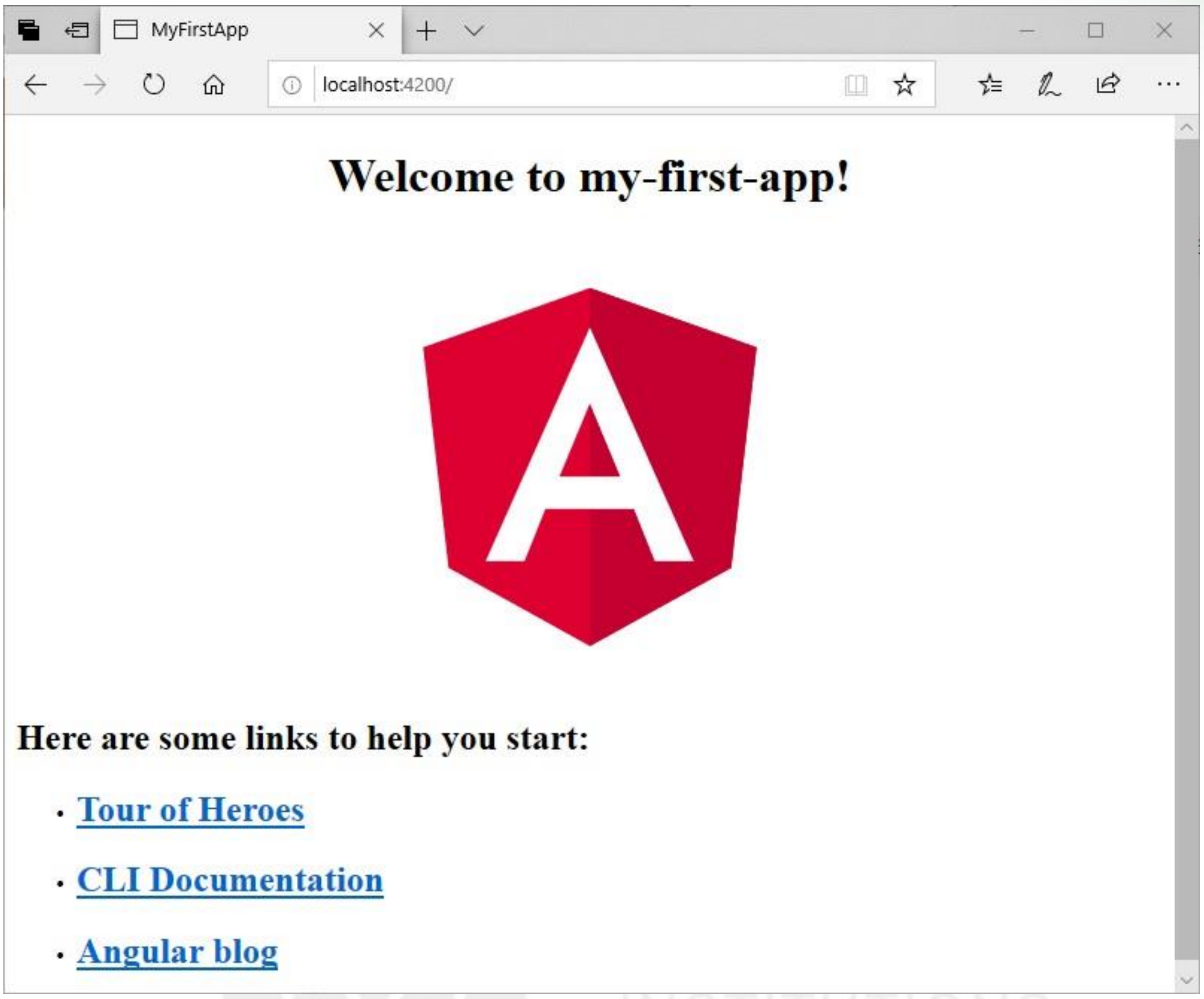
1. ng serve



ng serve

```
found 0 vulnerabilities

'git' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\user>cd my-first-app

C:\Users\user\my-first-app>ng serve
** Angular Live Development Server is listening on localhost:4200, open your browser on ht
tp://localhost:4200/ **
                                                                                         p
Date: 2019-01-29T06:33:04.550Z
Hash: e76d64339005d42b0f59
Time: 8843ms
chunk {main} main.js, main.js.map (main) 9.77 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 236 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.08 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 16.3 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3.5 MB [initial] [rendered]
i ｢wdm｣: Compiled successfully.
```

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



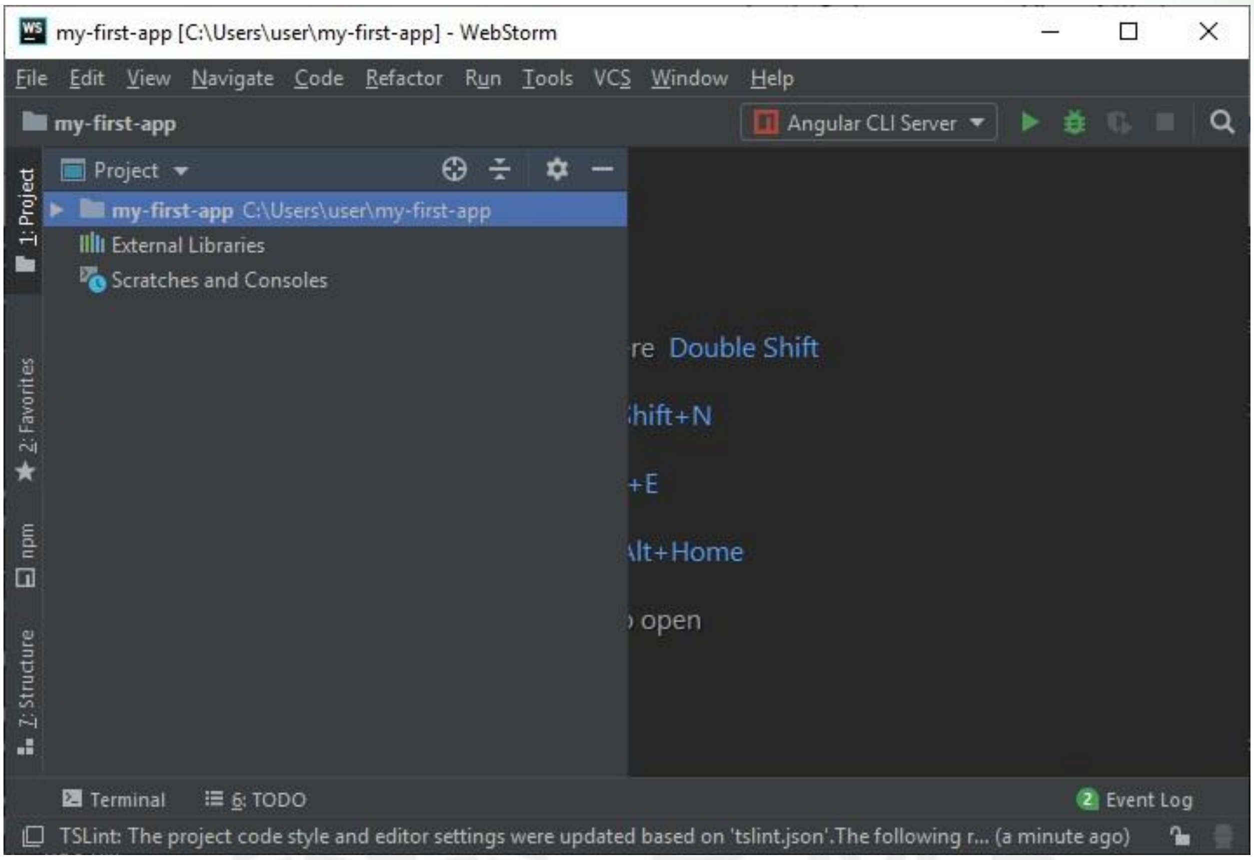
Welcome to my-first-app!

Here are some links to help you start:

- Tour of Heroes
- CLI Documentation
- Angular blog

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

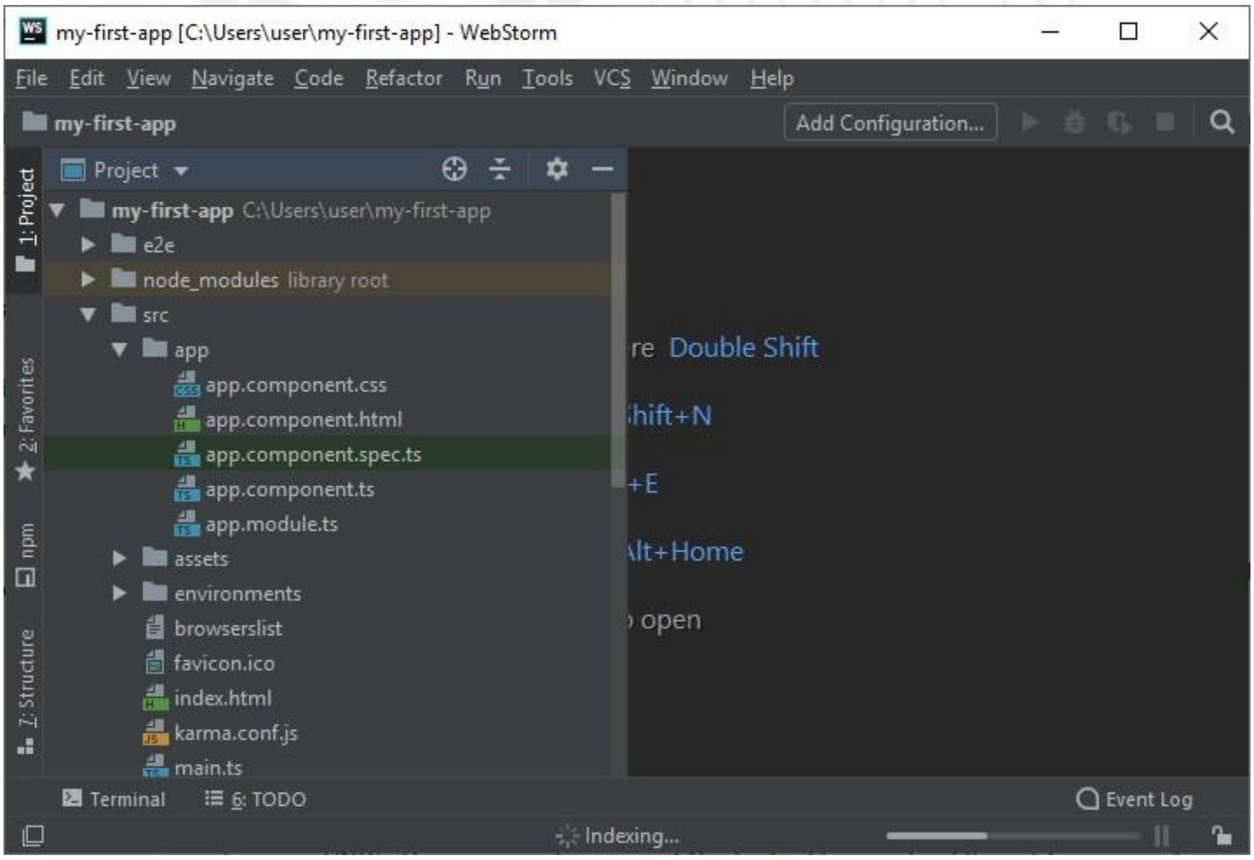
Open WebStorm and open your app "my-first-app" in the IDE. It will look like this:



my-first-app [C:\Users\user\my-first-app] - WebStorm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
my-first-app
Angular CLI Server
Project
my-first-app C:\Users\user\my-first-app
External Libraries
Scratches and Consoles
re Double Shift
hift+N
+E
lt+Home
open
Terminal
6: TODO
Event Log

TSLint: The project code style and editor settings were updated based on 'tslint.json'.The following r... (a minute ago)

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

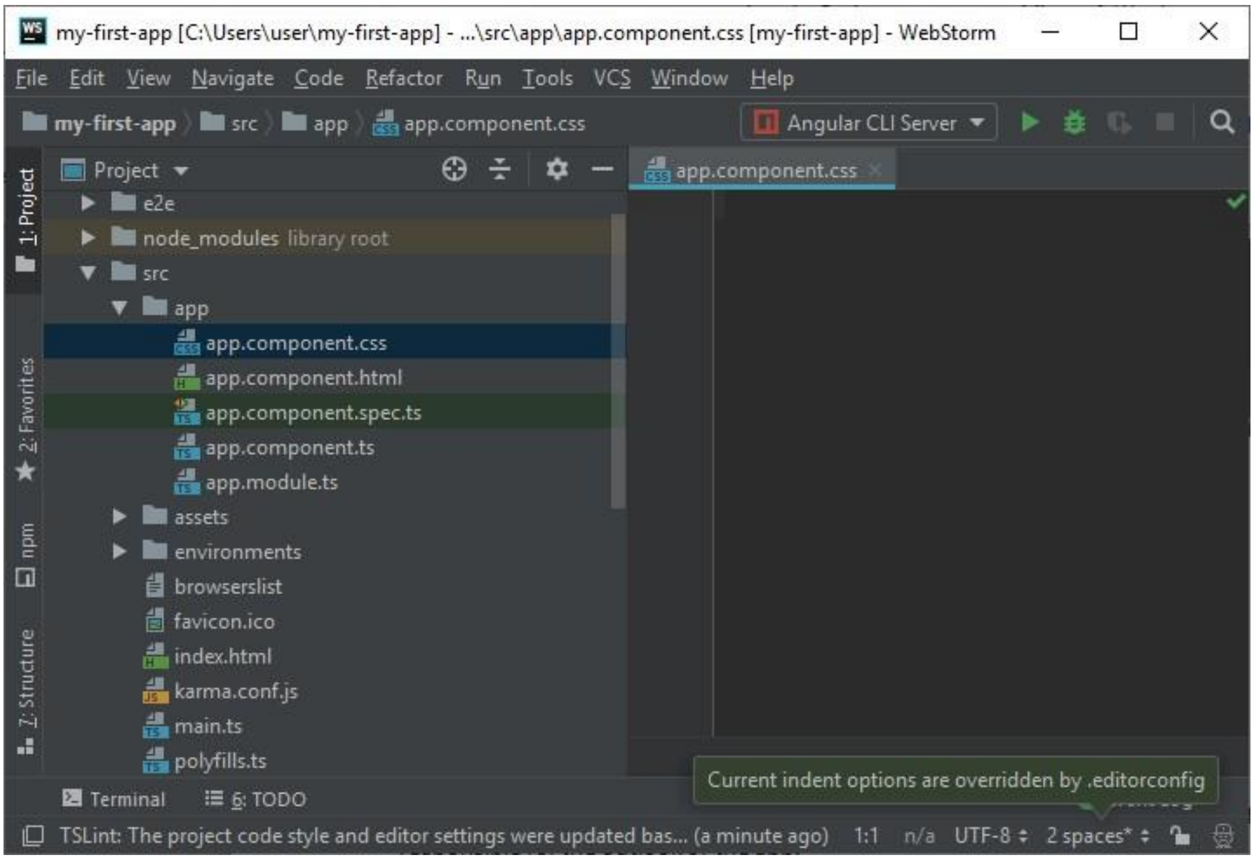


You will see 5 components there:

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

app.component.css

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



my-first-app [C:\Users\user\my-first-app] - ...\src\app\app.component.css [my-first-app] - WebStorm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
my-first-app src app app.component.css
Angular CLI Server
Project
app.component.css
1: Project
2: Favorites
npm
7: Structure
Terminal
6: TODO

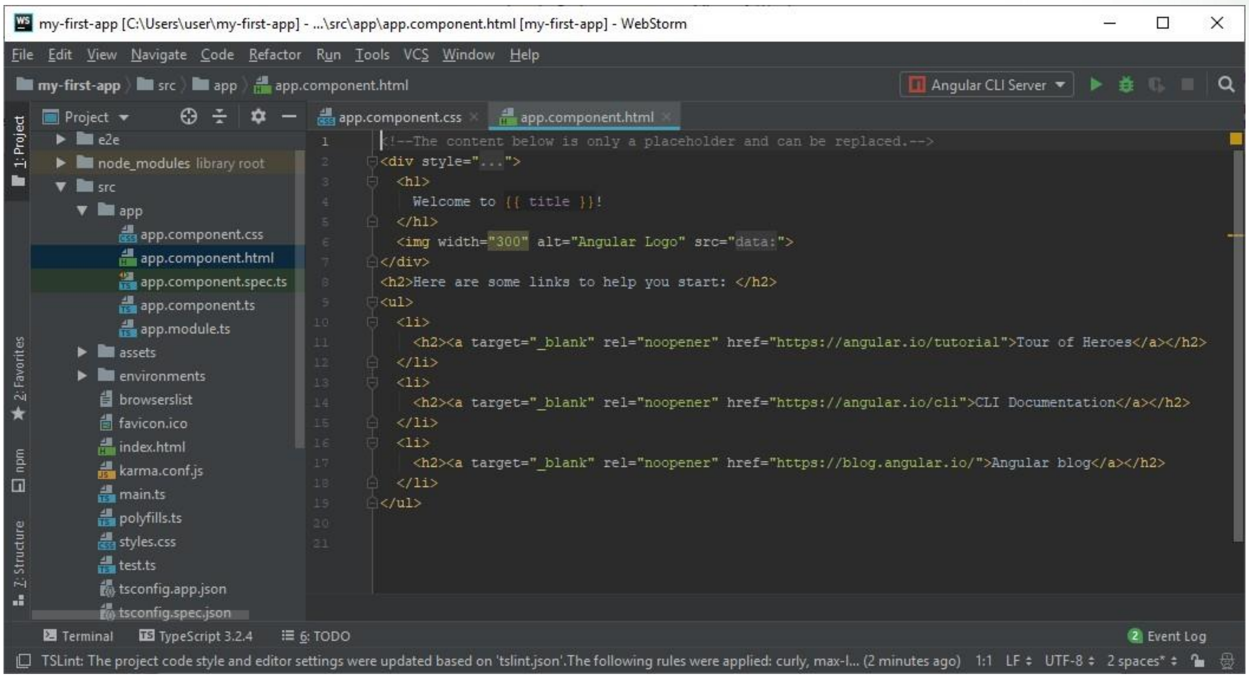
```
e2e
node_modules library root
src
  app
    app.component.css
    app.component.html
    app.component.spec.ts
    app.component.ts
    app.module.ts
  assets
  environments
  browserslist
  favicon.ico
  index.html
  karma.conf.js
  main.ts
  polyfills.ts
```

Current indent options are overridden by .editorconfig

TSLint: The project code style and editor settings were updated bas... (a minute ago) 1:1 n/a UTF-8 2 spaces*

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.



```
<!--The content below is only a placeholder and can be replaced.-->
<div style="...">
  <h1>
    Welcome to {{ title }}!
  </h1>
  <img width="300" alt="Angular Logo" src="data:">
</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>
```

TSLint: The project code style and editor settings were updated based on 'tslint.json'.The following rules were applied: curly, max-l... (2 minutes ago)

Code:

```
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
  <img width="300" alt="Angular Logo" src="data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHZpZXdCb3g9IjAgMCAyNTAgMjUwIj4KICAgIDxwYXRoIGZpbGw9IiNERDAwMzEiIGQ9Ik0xMjUgMzBMMzEuOSA2My4ybDE0LjIgMTIzLjFMMTI1IDIzMGw3OC45LTQzLjcgMTQuMi0xMjMuMXoiIC8+CiAgICA8cGF0aCBmaWxsPSIjQzMwMDJGIiBkPSJNMTI1IDMwdjIyLjItLjFWMjMwbDc4LjktNDMuNyAxNC4yLTEyMy4xTDEyNSAzMHoiIC8+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({
      declarations: [
        AppComponent
      ],
    }).compileComponents();
  }));

  it('should create the app', () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app).toBeTruthy();
  });

  it(`should have as title 'my-first-app'`, () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app.title).toEqual('my-first-app');
  });

  it('should render title in a h1 tag', () => {
    const fixture = TestBed.createComponent(AppComponent);
    fixture.detectChanges();
    const compiled = fixture.debugElement.nativeElement;
    expect(compiled.querySelector('h1').textContent).toContain('Welcome to my-first-app!');
  });
});
```

app.component.ts

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

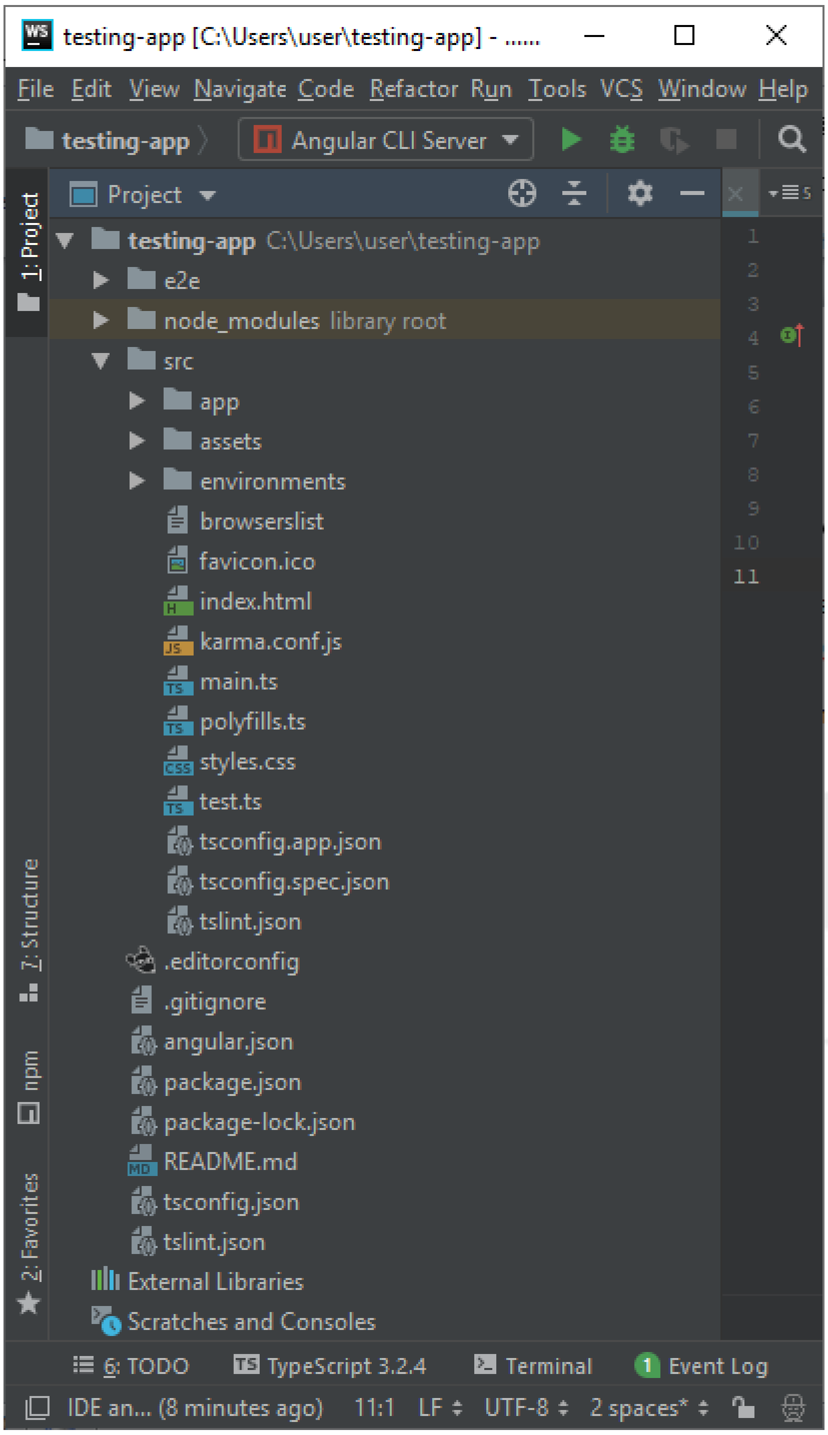
```
import { Component } from '@angular/core';

@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: [],
14.   bootstrap: [AppComponent]
15. })
16. export class AppModule { }
```

Angular 7 App files explanation

See the structure of the Angular 7 app on WebStorm IDE (how it looks on IDE). For Angular 7 development, you can use either Visual Studio Code IDE or WebStorm IDE. Both are good. Here, we are using JetBrains WebStorm IDE.


testing-app [C:\Users\user\testing-app] - ......
File Edit View Navigate Code Refactor Run Tools VCS Window Help
testing-app
Angular CLI Server
Project
1: Project
7: Structure
npm
2: Favorites
6: TODO
TypeScript 3.2.4
Terminal
Event Log
IDE an... (8 minutes ago) 11:1 LF UTF-8 2 spaces*

```
testing-app C:\Users\user\testing-app
  e2e
  node_modules library root
  src
    app
    assets
    environments
    browserslist
    favicon.ico
    index.html
    karma.conf.js
    main.ts
    polyfills.ts
    styles.css
    test.ts
    tsconfig.app.json
    tsconfig.spec.json
    tslint.json
  .editorconfig
  .gitignore
  angular.json
  package.json
  package-lock.json
  README.md
  tsconfig.json
  tslint.json
External Libraries
Scratches and Consoles
```

Files used in Angular 7 App folder

Angular 7 App files which are mainly used in your project are given below:

- **src folder:** This is the folder which contains the main code files related to your angular application.
- **app folder:** The app folder contains the files, you have created for app components.

- **app.component.css:** This file contains the cascading style sheets code for your app component.
- **app.component.html:** This file contains the html file related to app component. This is the template file which is used by angular to do the data binding.
- **app.component.spec.ts:** This file is a unit testing file related to app component. This file is used along with other unit tests. It is run from Angular CLI by the command ng test.
- **app.component.ts:** This is the most important typescript file which includes the view logic behind the component.
- **app.module.ts:** This is also a typescript file which includes all the dependencies for the website. This file is used to define the needed modules to be imported, the components to be declared and the main component to be bootstrapped.

Other Important files

- **package.json:** This is npm configuration file. It includes details about your website's package dependencies along with details about your own website being a package itself.
- **package-lock.json :** This is an auto-generated and modified file that gets updated whenever npm does an operation related to node_modules or package.json
- **angular.json:** It is very important configuration file related to your angular application. **It defines the structure of your app and includes any settings associated with your application.** Here, you can specify environments on this file (development, production). This is the file where we add Bootstrap file to work with Angular 7.
- **.gitignore:** This file is related to the source control git.
- **.editorconfig:** This is a simple file which is used to maintain consistency in code editors to organize some basics such as indentation and whitespaces.
- **assets folder:** This folder is a placeholder for resource files which are used in the application such as images, locales, translations etc.
- **environments folder:** The environments folder is used to hold the environment configuration constants that help when building the angular application. The constants are defined in 2 separate .ts files (environment.ts and environment.prod.ts), where these constants are used within the angular.json file by the Angular CLI. For example, if you run the ng build command, it will build the

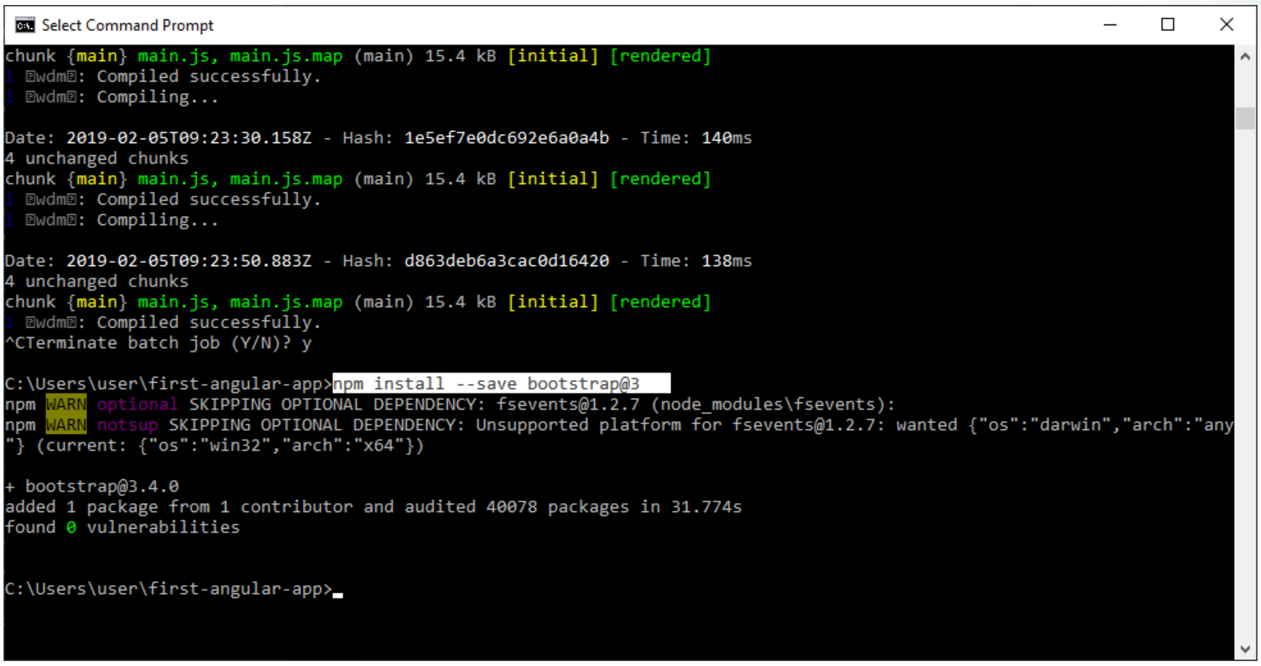
application using the development environment settings, whereas the command ng build ?prod will build the project using the production environment settings.

- **browserlist:** This file is used by autoprefixer that adjusts the CSS to support a list of defined browsers.
- **favicon.ico:** This file specifies a small icon that appears next to the browser tab of a website.
- **index.html:** This is the entry file which holds the high level container for the angular application.
- **karma.config.js:** This file specifies the config file for the Karma Test Runner, Karma has been developed by the AngularJS team which can run tests for both AngularJS and Angular 2+
- **main.ts:** As defined in angular.json file, this is the main ts file that will first run. This file bootstraps (starts) the AppModule from app.module.ts , and it can be used to define global configurations.
- **polyfills.ts:** This file is a set of code that can be used to provide compatibility support for older browsers. Angular 7 code is written mainly in ES6+ language specifications which is getting more adopted in front-end development, so since not all browsers support the full ES6+ specifications, pollyfills can be used to cover whatever feature missing from a given browser.
- **styles.css:/** This is a global css file which is used by the angular application.
- **tests.ts:** This is the main test file that the Angular CLI command ng test will use to traverse all the unit tests within the application and run them.
- **tsconfig.json:** This is a typescript compiler configuration file.
- **tsconfig.app.json:** This is used to override the tsconfig.json file with app specific configurations.
- **tsconfig.spec.json:** This overrides the tsconfig.json file with app specific unit test configurations.

install Bootstrap for Angular project?

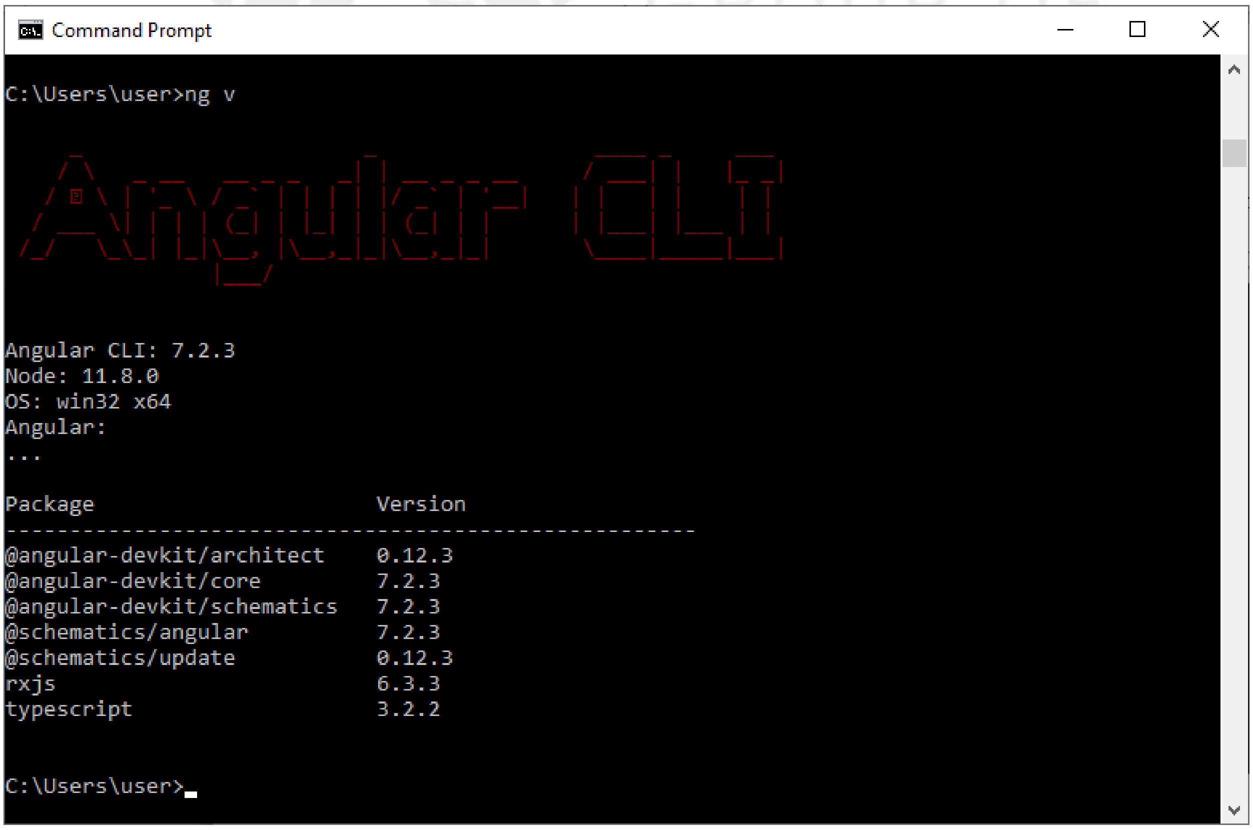
Run the following command on command prompt:

1. npm install --save bootstrap@3 => The @3 is important!



```
chunk {main} main.js, main.js.map (main) 15.4 kB [initial] [rendered]
ℹ ｢wdm｣: Compiled successfully.
ℹ ｢wdm｣: Compiling...

Date: 2019-02-05T09:23:30.158Z - Hash: 1e5ef7e0dc692e6a0a4b - Time: 140ms
4 unchanged chunks
chunk {main} main.js, main.js.map (main) 15.4 kB [initial] [rendered]
ℹ ｢wdm｣: Compiled successfully.
ℹ ｢wdm｣: Compiling...

Date: 2019-02-05T09:23:50.883Z - Hash: d863deb6a3cac0d16420 - Time: 138ms
4 unchanged chunks
chunk {main} main.js, main.js.map (main) 15.4 kB [initial] [rendered]
ℹ ｢wdm｣: Compiled successfully.
^CTerminate batch job (Y/N)? y

C:\Users\user\first-angular-app>npm install --save bootstrap@3
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.7 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.7: wanted {"os":"darwin","arch":"any
"} (current: {"os":"win32","arch":"x64"})

+ bootstrap@3.4.0
added 1 package from 1 contributor and audited 40078 packages in 31.774s
found 0 vulnerabilities


C:\Users\user\first-angular-app>_
```

Further, when you use a project created with Angular CLI 6+ (check via **ng v**), you'll have an angular.json file instead of an **.angular-cli.json** file. In that file, you still need to add Bootstrap to the **styles[]** array, but the path should be **node_modules/bootstrap/dist/css/bootstrap.min.css** **, NOT ../node_modules/bootstrap/dist/css/bootstrap.min.css** . The leading **../** must not be included.

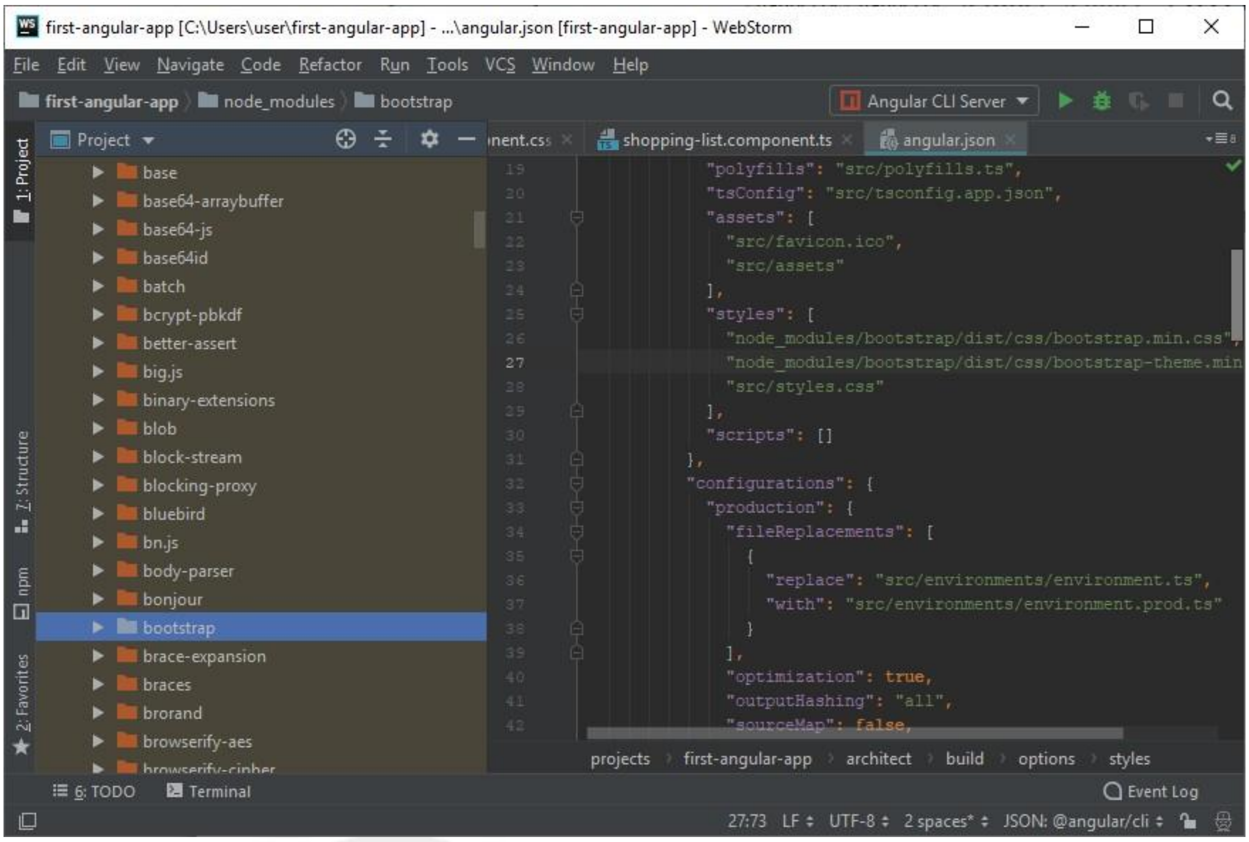


```
C:\Users\user>ng v

Angular CLI: 7.2.3
Node: 11.8.0
OS: win32 x64
Angular:
...

Package                          Version
------------------------------------------------------
@angular-devkit/architect        0.12.3
@angular-devkit/core             7.2.3
@angular-devkit/schematics       7.2.3
@schematics/angular              7.2.3
@schematics/update               0.12.3
rxjs                             6.3.3
typescript                       3.2.2


C:\Users\user>_
```

Here, we are using the Angular 7.2.3 version.

How to add bootstrap.css file in the project?

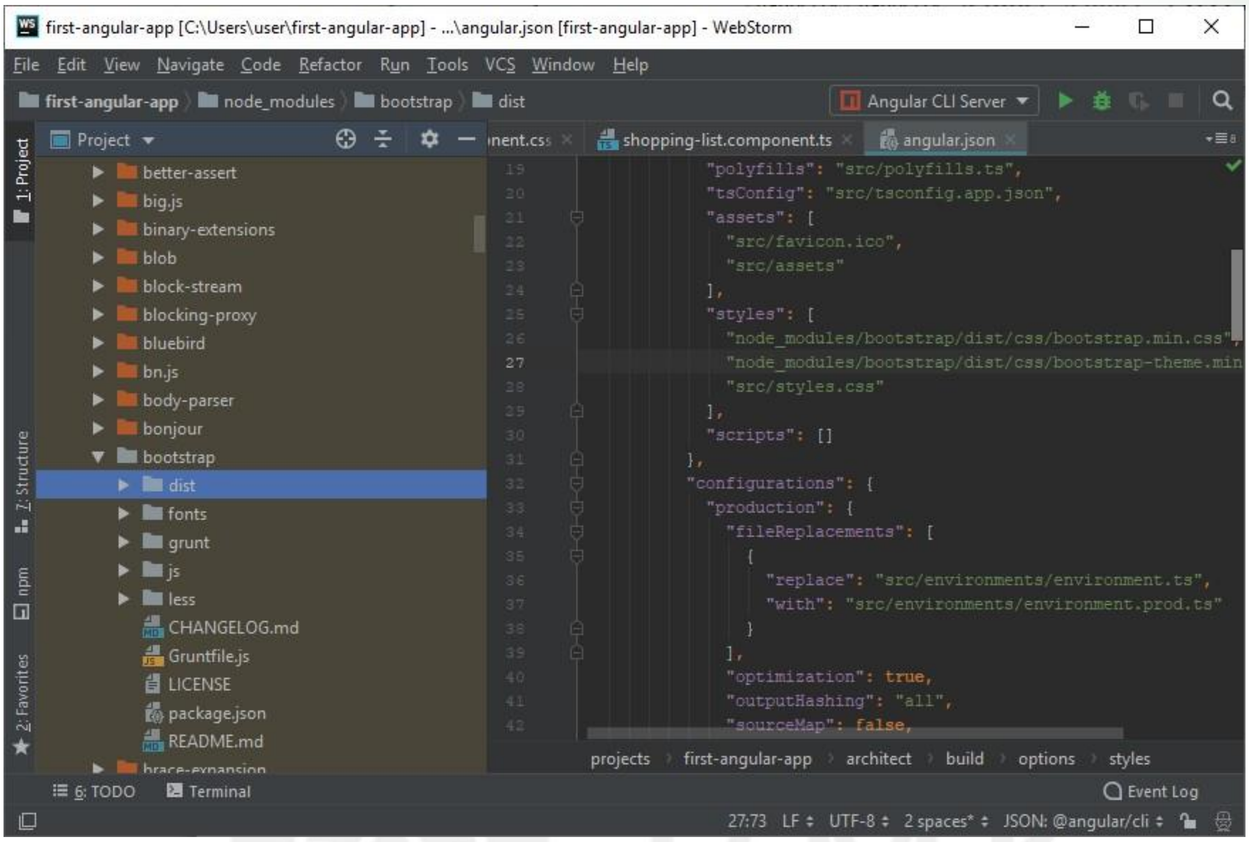
Expand Node Module (library root folder)



```
            "polyfills": "src/polyfills.ts",
            "tsConfig": "src/tsconfig.app.json",
            "assets": [
              "src/favicon.ico",
              "src/assets"
            ],
            "styles": [
              "node_modules/bootstrap/dist/css/bootstrap.min.css",
              "node_modules/bootstrap/dist/css/bootstrap-theme.min
              "src/styles.css"
            ],
            "scripts": []
          },
          "configurations": {
            "production": {
              "fileReplacements": [
                {
                  "replace": "src/environments/environment.ts",
                  "with": "src/environments/environment.prod.ts"
                }
              ],
              "optimization": true,
              "outputHashing": "all",
              "sourceMap": false,
```

first-angular-app [C:\Users\user\first-angular-app] - ...\angular.json [first-angular-app] - WebStorm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
first-angular-app 〉 node_modules 〉 bootstrap
Angular CLI Server
Project
nent.css
shopping-list.component.ts
angular.json
1: Project
7: Structure
npm
2: Favorites
base
base64-arraybuffer
base64-js
base64id
batch
bcrypt-pbkdf
better-assert
big.js
binary-extensions
blob
block-stream
blocking-proxy
bluebird
bn.js
body-parser
bonjour
bootstrap
brace-expansion
braces
brorand
browserify-aes
browserify-cipher
projects 〉 first-angular-app 〉 architect 〉 build 〉 options 〉 styles
6: TODO
Terminal
Event Log
27:73 LF ÷ UTF-8 ÷ 2 spaces* ÷ JSON: @angular/cli ÷

Go to bootstrap folder and expand it.

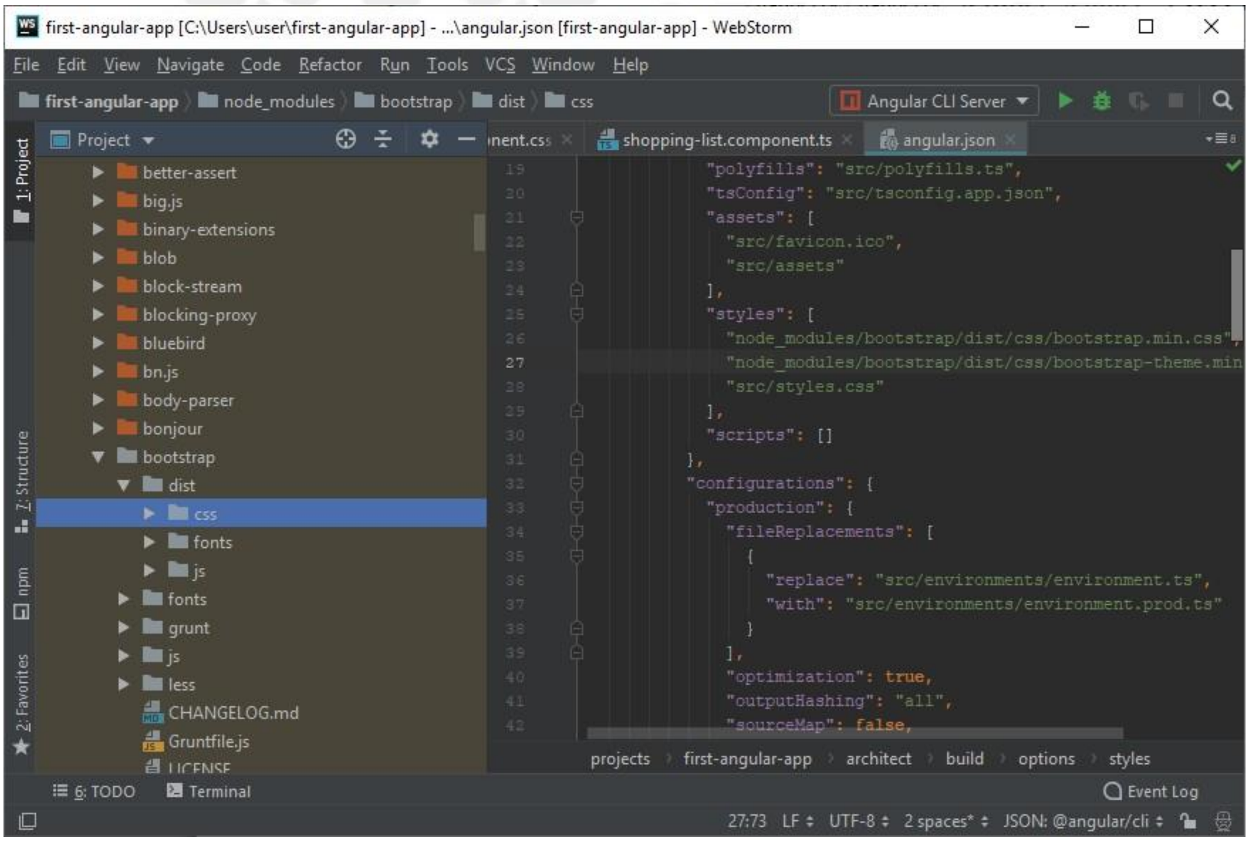

first-angular-app [C:\Users\user\first-angular-app] - ...\angular.json [first-angular-app] - WebStorm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
first-angular-app 〉 node_modules 〉 bootstrap 〉 dist
Angular CLI Server
Project
nent.css
shopping-list.component.ts
angular.json
1: Project
7: Structure
npm
2: Favorites
6: TODO
Terminal
Event Log
27:73 LF UTF-8 2 spaces* JSON: @angular/cli

```
better-assert
big.js
binary-extensions
blob
block-stream
blocking-proxy
bluebird
bn.js
body-parser
bonjour
bootstrap
  dist
  fonts
  grunt
  js
  less
  CHANGELOG.md
  Gruntfile.js
  LICENSE
  package.json
  README.md
brace-expansion
```

```
        "polyfills": "src/polyfills.ts",
        "tsConfig": "src/tsconfig.app.json",
        "assets": [
          "src/favicon.ico",
          "src/assets"
        ],
        "styles": [
          "node_modules/bootstrap/dist/css/bootstrap.min.css",
          "node_modules/bootstrap/dist/css/bootstrap-theme.min
          "src/styles.css"
        ],
        "scripts": []
      },
      "configurations": {
        "production": {
          "fileReplacements": [
            {
              "replace": "src/environments/environment.ts",
              "with": "src/environments/environment.prod.ts"
            }
          ],
          "optimization": true,
          "outputHashing": "all",
          "sourceMap": false,
```

projects 〉 first-angular-app 〉 architect 〉 build 〉 options 〉 styles

Go to dist folder and expand dist.

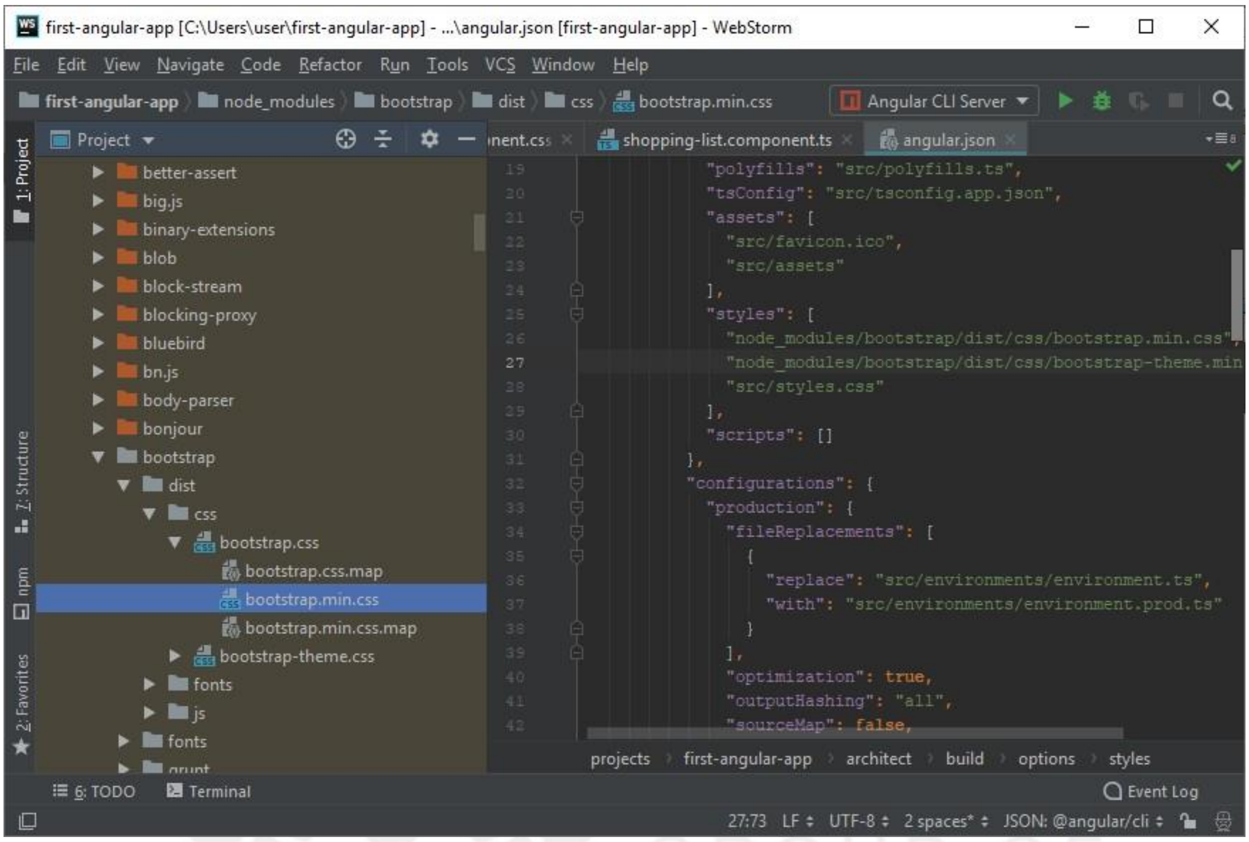

first-angular-app [C:\Users\user\first-angular-app] - ...\angular.json [first-angular-app] - WebStorm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
first-angular-app 〉 node_modules 〉 bootstrap 〉 dist 〉 css
Angular CLI Server
Project
nent.css
shopping-list.component.ts
angular.json
1: Project
7: Structure
npm
2: Favorites
6: TODO
Terminal
Event Log
27:73 LF UTF-8 2 spaces* JSON: @angular/cli

```
better-assert
big.js
binary-extensions
blob
block-stream
blocking-proxy
bluebird
bn.js
body-parser
bonjour
bootstrap
  dist
    css
    fonts
    js
  fonts
  grunt
  js
  less
  CHANGELOG.md
  Gruntfile.js
  LICENSE
```

```
        "polyfills": "src/polyfills.ts",
        "tsConfig": "src/tsconfig.app.json",
        "assets": [
          "src/favicon.ico",
          "src/assets"
        ],
        "styles": [
          "node_modules/bootstrap/dist/css/bootstrap.min.css",
          "node_modules/bootstrap/dist/css/bootstrap-theme.min
          "src/styles.css"
        ],
        "scripts": []
      },
      "configurations": {
        "production": {
          "fileReplacements": [
            {
              "replace": "src/environments/environment.ts",
              "with": "src/environments/environment.prod.ts"
            }
          ],
          "optimization": true,
          "outputHashing": "all",
          "sourceMap": false,
```

projects 〉 first-angular-app 〉 architect 〉 build 〉 options 〉 styles

Expand css and you will find "bootstrap.css". Expand bootstrap.css and you will see bootstrap.min.css


first-angular-app [C:\Users\user\first-angular-app] - ...\angular.json [first-angular-app] - WebStorm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
first-angular-app › node_modules › bootstrap › dist › css › bootstrap.min.css
Angular CLI Server
Project
better-assert
big.js
binary-extensions
blob
block-stream
blocking-proxy
bluebird
bn.js
body-parser
bonjour
bootstrap
dist
css
bootstrap.css
bootstrap.css.map
bootstrap.min.css
bootstrap.min.css.map
bootstrap-theme.css
fonts
js
fonts
grunt
1: Project
7: Structure
npm
2: Favorites
nent.css
shopping-list.component.ts
angular.json
projects › first-angular-app › architect › build › options › styles
6: TODO
Terminal
Event Log
27:73 LF UTF-8 2 spaces* JSON: @angular/cli

```
          "polyfills": "src/polyfills.ts",
          "tsConfig": "src/tsconfig.app.json",
          "assets": [
            "src/favicon.ico",
            "src/assets"
          ],
          "styles": [
            "node_modules/bootstrap/dist/css/bootstrap.min.css",
            "node_modules/bootstrap/dist/css/bootstrap-theme.min
            "src/styles.css"
          ],
          "scripts": []
        },
        "configurations": {
          "production": {
            "fileReplacements": [
              {
                "replace": "src/environments/environment.ts",
                "with": "src/environments/environment.prod.ts"
              }
            ],
            "optimization": true,
            "outputHashing": "all",
            "sourceMap": false,
```

Open angular.json file and add the bootstrap.min.css in style section.

```
"styles": [
  "node_modules/bootstrap/dist/css/bootstrap.min.css",
  "src/styles.css"
],
```

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.

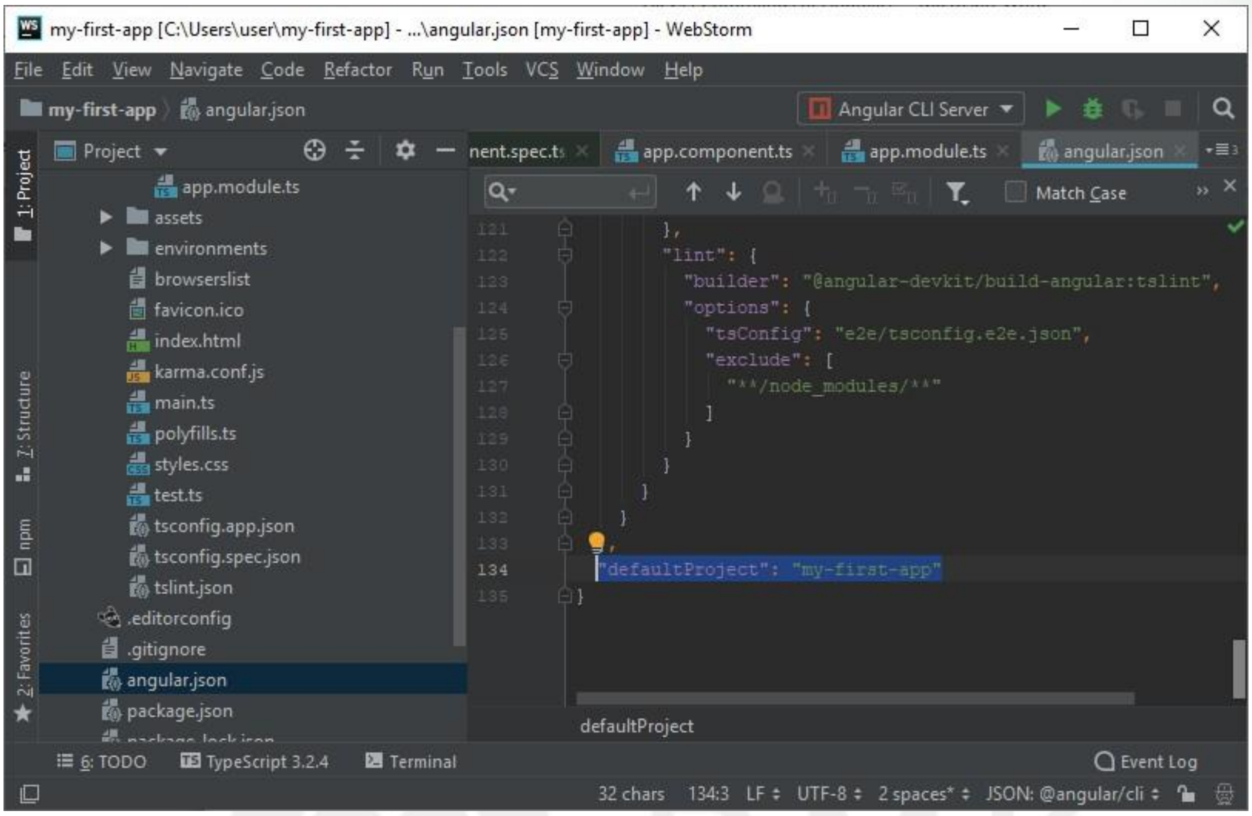
Command	Alias	Description
add		It is used to add support for an external library to your project.

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 outputs Angular CLI version.
xi18n		It extracts i18n messages from source code.

ng add Command

The ng add command is used to add support for an external library to your project. It adds the npm package for a published library to your workspace, and makes your default app project to use that library, in whatever way is specified by the library's schematic. For example, if you add @angular/pwa, then it will configure your project for PWA support.

The default app project is the value of defaultProject in angular.json.



```
        },
        "lint": {
          "builder": "@angular-devkit/build-angular:tslint",
          "options": {
            "tsConfig": "e2e/tsconfig.e2e.json",
            "exclude": [
              "**/node_modules/**"
            ]
          }
        }
      }
    }
  },
  "defaultProject": "my-first-app"
}
```

Syntax:

1. ng add **<collection>** [options]

Parameter Explanation:

<collection>: It specifies the package which you want to add.

Options

--defaults=true|false: When true, it disables interactive input prompts for options with a default.

--help=true|false|json|JSON: It is used to show a help message in the console. Default: false

--interactive=true|false: When false, it disables interactive input prompts.

ng build Command

The ng build command is used to compile an Angular app into an output directory named dist/ at the given output path. It must be executed from within a workspace directory.

Syntax:

1. ng build **<project>** [options]

2. ng b **<project>** [options]

Parameter Explanation:

<project>: It specifies the name of the project to build. It can be an app or a library.

Options

--aot=true|false: It builds using Ahead of Time compilation. Default: false

--baseHref=baseHref: It specifies the base url for the application being built.

--buildEventLog=buildEventLog: (experimental) Output file path for Build Event Protocol events.

--buildOptimizer=true|false: It enables '@angular-devkit/build-optimizer' optimizations when using the 'aot' option. Default: false

--commonChunk=true|false: It uses a separate bundle containing code used across multiple bundles. Default: true

--configuration=configuration: A named build target, as specified in the "configurations" section of angular.json. Each named target is accompanied by a configuration of option defaults for that target.

Aliases: -c

--deleteOutputPath=true|false: It is used to delete the output path before building. Default: true

--deployUrl=deployUrl: URL where files will be deployed.

--es5BrowserSupport=true|false: Enables conditionally loaded ES2015 polyfills. Default: false

--extractCss=true|false: It is used to extract css from global styles into css files instead of js ones. Default: false

--extractLicenses=true|false: It is used to extract all licenses in a separate file. Default: false

--forkTypeChecker=true|false: It is used to run the TypeScript type checker in a forked process. Default: true

--help=true|false|json|JSON: It is used to show a help message for this command in the console. Default: false

--i18nFile=i18nFile: Localization file to use for i18n.

--i18nFormat=i18nFormat: Format of the localization file specified with --i18n-file.

--i18nLocale=i18nLocale: Locale to use for i18n.

--i18nMissingTranslation=i18nMissingTranslation: How to handle missing translations for i18n.

--index=index: The name of the index HTML file.

--lazyModules: List of additional NgModule files that will be lazy loaded. Lazy router modules will be discovered automatically.

--main=main: The full path for the main entry point to the app, relative to the current workspace.

--namedChunks=true|false: Use file name for lazy loaded chunk Default: true

--ngswConfigPath=ngswConfigPath: Path to ngsw-config.json.

--optimization=true|false: Enables optimization of the build output.

--outputHashing= none|all|media|bundles: Define the output filename cache-busting hashing mode.

Default: none

--outputPath=outputPath: The full path for the new output directory, relative to the current workspace.

By default, writes output to a folder named dist/ in the current project.

--poll: Enable and define the file watching poll time period in milliseconds.

--polyfills=polyfills: The full path for the polyfills file, relative to the current workspace.

--preserveSymlinks=true|false: Do not use the real path when resolving modules.

Default: false

ADVERTISEMENT

--prod=true|false: When true, sets the build configuration to the production target. All builds make use of bundling and limited tree-shaking. A production build also runs limited dead code elimination.

--profile=true|false: Output profile events for Chrome profiler.

Default: false

--progress=true|false: Log progress to the console while building.

--resourcesOutputPath= resourcesOutputPath: The path where style resources will be placed, relative to outputPath.

--serviceWorker=true|false: Generates a service worker config for production builds. Default: false

--showCircularDependencies=true|false: Show circular dependency warnings on builds. Default: true

--sourceMap=true|false: It is used to show Output sourcemaps.

Default: true

--statsJson=true|false: It generates a 'stats.json' file which can be analyzed using tools such as: 'webpack-bundle-analyzer' or https://webpack.github.io/analyse.

Default: false

--subresourceIntegrity=true|false: It enables the use of subresource integrity validation.

Default: false

--tsConfig=tsConfig: The full path for the TypeScript configuration file, relative to the current workspace.

--vendorChunk=true|false: It uses a separate bundle containing only vendor libraries.

Default: true

--verbose=true|false: It adds more details to output logging.

Default: false

--watch=true|false: It runs build when files change.

Default: false

ng config Command

The ng config command is used to retrieve or set Angular configuration values in the angular.json file for the workspace.

Syntax:

1. ng config **<jsonPath>** **<value>** [options]

Parameter Explanation:

<jsonPath>: The configuration key to set or query, in JSON path format. For example: "a[3].foo.bar[2]". If no new value is provided, returns the current value of this key.

<value>: If provided, a new value for the given configuration key.

Options

--global=true|false: When true, it accesses the global configuration in the caller's home directory.

Default: false

Aliases: -g

--help= true|false|json|JSON: It is used to show a help message for this command in the console.

Default: false

ng doc Command

The ng doc command is used to open the official Angular documentation (angular.io) in a browser, and searches for a given keyword.

Syntax:

1. ng doc **<keyword>** [options]
2. ng d **<keyword>** [options]

Parameter Explanation:

<keyword>: It is used to specify the keyword to search for, as provided in the search bar in angular.io.

Options

--help=true|false|json|JSON: It is used to show a help message for this command in the console.

Default: false

--search=true|false: When true, it searches all of angular.io. Otherwise, searches only API reference documentation.

Default: false

Aliases: -s

ng e2e Command

It is used to build and serve an Angular app, then runs end-to-end tests using Protractor.

Syntax:

1. ng e2e **<project>** [options]
2. ng e **<project>** [options]

It must be executed from within a workspace directory. When you don't specify the project name, it executes for all projects.

Parameter Explanation:

<project>: It specifies the name of the project you want to build. It can be an app or a library.

Options

--baseUrl=baseUrl: It specifies the base URL for protractor to connect to.

--configuration=configuration: It is used to specify named build target, as specified in the "configurations" section of angular.json. Each named target is accompanied by a configuration of option defaults for that target.

Aliases: -c

--devServerTarget=devServerTarget: It specifies dev server target to run tests against.

--elementExplorer=true|false: It starts Protractor's Element Explorer for debugging.

Default: false

--help= true|false|json|JSON: It shows a help message for this command in the console.

Default: false

--host=host: Host to listen on.

Default: localhost

--port: It specifies the port to serve the application.

--prod=true|false: When true, it sets the build configuration to the production target. All builds make use of bundling and limited tree-shaking. A production build also runs limited dead code elimination.

--protractorConfig= protractorConfig: It specifies the name of the Protractor configuration file.

--specs: It overrides specs in the protractor config.

--suite=suite: It overrides suite in the protractor config.

--webdriverUpdate=true|false: It is used to update webdriver.

Default: true

ng generate Command

The ng generate command is used to generate and/or modify file based on schematic.

1. ng generate **<schematic>** [options]
2. ng g **<schematic>** [options]

Parameter Explanation:

<schematic >: It specifies the schematic or collection:schematic which you want to generate. It can take one of the following sub-commands.

- appShell
- application
- class
- component

- directive
- enum
- guard
- interface
- library
- module
- pipe
- service
- serviceWorker
- universal

Schematic Command Explanation

appShell:

It is used to generate an app shell for running a server-side version of an app.

Syntax:

1. ng generate appShell [options]
2. ng g appShell [options]

application

It is used to create a new basic app definition in the "projects" subfolder of the workspace.

Syntax:

1. ng generate application **<name>** [options]
2. ng g application **<name>** [options]

class

It is used to create a new generic class definition in the given or default project.

Syntax:

1. ng generate class **<name>** [options]
2. ng g class **<name>** [options]

component

It is used to create a new generic component definition in the given or default project.

Syntax:

1. ng generate component **<name>** [options]
2. ng g component **<name>** [options]

directive

It is used to create a new generic directive definition in the given or default project.

Syntax:

1. ng generate directive **<name>** [options]
2. ng g directive **<name>** [options]

enum

It is used to create a new, generic enum definition for the given or default project.

Syntax:

1. ng generate enum **<name>** [options]
2. ng g enum **<name>** [options]

enum

It is used to create a new, generic enum definition for the given or default project.

Syntax:

1. ng generate enum **<name>** [options]
2. ng g enum **<name>** [options]

guard

It is used to generate a new, generic route guard definition in the given or default project.

Syntax:

1. ng generate enum **<name>** [options]
2. ng g enum **<name>** [options]

interface

It is used to create a new generic interface definition in the given or default project.

Syntax:

1. ng generate interface **<name>** **<type>** [options]
2. ng g interface **<name>** **<type>** [options]

library

It is used to create a new generic library project in the current workspace.

Syntax:

1. ng generate library **<name>** [options]
2. ng g library **<name>** [options]

module

It is used to create a new generic NgModule definition in the given or default project.

Syntax:

1. ng generate module **<name>** [options]
2. ng g module **<name>** [options]

pipe

It is used to create a new generic pipe definition in the given or default project.

Syntax:

1. ng generate pipe **<name>** [options]
2. ng g pipe **<name>** [options]

service

It is used to create a new, generic service definition in the given or default project.

Syntax:

1. ng generate service **<name>** [options]
2. ng g service **<name>** [options]

serviceWorker

This is used to pass this schematic to the "run" command to create a service worker.

Syntax:

1. ng generate serviceWorker [options]
2. ng g serviceWorker [options]

universal

This command is used to pass this schematic to the "run" command to set up server-side rendering for an app.

Syntax:

1. ng generate universal [options]
2. ng g universal [options]

Options

--defaults=true|false: When true, it disables interactive input prompts for options with a default.

--dryRun=true|false: When true, it runs through and reports activity without writing out results.

Default: false

Aliases: -d

--force=true|false: When true, it forces overwriting of existing files.

Default: false

Aliases: -f

--help=true|false|json|JSON: It is used to show a help message in the console.

Default: false

--interactive=true|false: When false, it disables interactive input prompts.

4.2 ANGULAR UI WITH BOOTSTRAP

Approach:

- First, add Angular UI bootstrap scripts needed for your project.

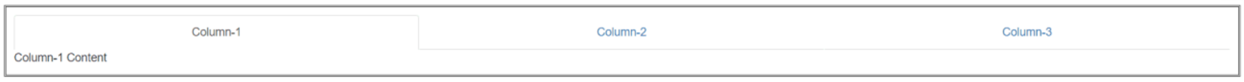
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular.js"></script> <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular-animate.js"></script> <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular-sanitize.js"></script> <script src="https://angular-ui.github.io/bootstrap/ui-bootstrap-tpls-2.5.0.js"></script>

- Make tab with its UIBootStrap classes which will set the UI look for the tab.
- Now make different types of tab using different classes and run the code.

Example 1: Making tabs with justified content.

```
<!DOCTYPE html>
<html ng-app="gfg">
<head>

        <!-- Adding CDN scripts required for our page -->
        <script src=
"https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular.js">
        </script>
        <script src=
"https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular-
animate.js">
        </script>
        <script src=
"https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular-
sanitize.js">
        </script>
        <script src=
"https://angular-ui.github.io/bootstrap/ui-bootstrap-tpls-2.5.0.js">
        </script>
        <link href=
"https://netdna.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
                rel="stylesheet">
        <script>
        // Adding Modules
        angular.module('gfg', ['ngAnimate', 'ngSanitize', 'ui.bootstrap']);
        angular.module('gfg').controller('tab', function ($scope) {
        });
        </script>
</head>

<body>
        <div ng-controller="tab"
                style="padding: 10px; margin: 10px; border-style: double;">
        <!-- making a tab -->
        <uib-tabset active="activeJustified" justified="true">
                <uib-tab index="0" heading="Column-1">
                Column-1 Content
                </uib-tab>
                <uib-tab index="1" heading="Column-2">
                Column-2 Content
                </uib-tab>
                <uib-tab index="2" heading="Column-3">
                Column-3 Content
                </uib-tab>
        </uib-tabset>
        </div>
</body>
</html>
```

Output:



Column-1 Column-2 Column-3

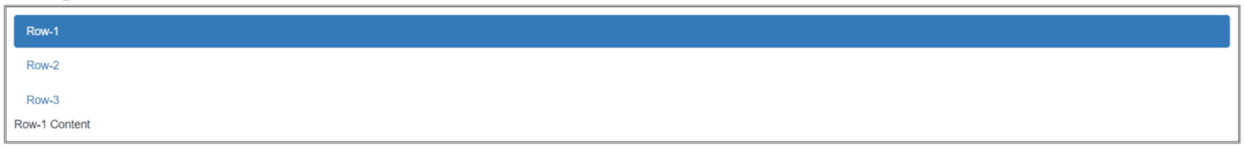
Column-1 Content

Example 2:

```
<!DOCTYPE html>
<html ng-app="gfg">
<head>

        <!-- Adding CDN scripts required for our page -->
        <script src=
"https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular.js">
        </script>
        <script src=
"https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular-
animate.js">
        </script>
        <script src=
"https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular-
sanitize.js">
        </script>
        <script src=
"https://angular-ui.github.io/bootstrap/ui-bootstrap-tpls-2.5.0.js">
        </script>
        <link href=
"https://netdna.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
                rel="stylesheet">

        <script>
        // Adding Modules
        angular.module('gfg', ['ngAnimate', 'ngSanitize', 'ui.bootstrap']);
        angular.module('gfg').controller('tab', function ($scope) {
        });
        </script>
</head>

<body>
        <div ng-controller="tab"
                style="padding: 10px; margin: 10px; border-style: double;">
                <!-- making a tab -->
                <uib-tabset active="activePill" vertical="true" type="pills">
                        <uib-tab index="0" heading="Row-1">
                        Row-1 Content
                </uib-tab>
                        <uib-tab index="1" heading="Row-2">
                        Row-2 Content
                </uib-tab>
                        <uib-tab index="2" heading="Row-3">
                        Row-3 Content
                </uib-tab>
                </uib-tabset>
        </div>
```

```
</body>
</html>
```

Output:


```
Row-1
Row-2
Row-3
Row-1 Content
```

Using Icons From Angular UI Bootstrap

Step 1) Get The Angular UI Bootstrap Library Files

via npm:

```
npm install angular-ui-bootstrap
```

via bower:

```
bower install angular-bootstrap
```

Step 2) Import The Angular UI Bootstrap Module

```
angular.module('myModule', ['ui.bootstrap']);
```

Step 3) Use Some Icons!

You can then add an icon anywhere by creating an tag (yes, the "i" is short for icon) and giving it two classes: one that is always just "glyphicon" and one for the specific glyphicon you want to use. Here is an example:

```
<i class="glyphicon glyphicon-fire"></i>
```

4.2.1 ANGULAR CSS AUTHENTICATION

login form using angular and bootstrap

Login.html

```
<div ng-app="myApp" ng-controller="myController" class="container">
    <form name="myForm">
      <div class="login-container">
        <div class="row">
          <div class="col-md-12">
            <div class="form-group">
              <label>Email</label>
        <input type="email" name="email" ng-model="user.username"
required class="form-control">
        <div ng-show="myForm.email.$error.email" class="error">Invalid
email!</div>
        <div ng-show="myForm.email.$error.required &amp;&amp;
myForm.email.$touched" class="error">Required!</div>
       </div>
      </div>
      <div class="col-md-12">
       <div class="form-group">
        <label>Password</label>
        <input type="password" name="pass" ng-model="user.password"
required class="form-control">
        <div ng-show="myForm.pass.$error.required &amp;&amp;
myForm.pass.$touched" class="error">Required!</div>
       </div>
      </div>
      <div class="col-md-12">
       <div class="form-group">
        <button ng-click="authenticate()" ng-disabled="myForm.$invalid"
class="btn btn-success pull-right">Login</button>
        <div class="clearfix"></div>
       </div>
      </div>
     </div>
     <div class="row">
      <div class="col-md-12">
       <div ng-show="showError" class="alert alert-danger">Wrong
credentials!</div>
       <div ng-show="showSuccess" class="alert alert-success">Login
Successful!</div>
      </div>
     </div>
    </div>
   </form>
  </div>
```

Login.css

```
.login-container{
 max-width: 400px;
 margin: 60px auto 0;
}

.error{
 color: tomato;
 font-size: 12px;
 padding-top: 4px
}
```

Login.js

```
var app = angular.module('myApp',[]);

app.controller('myController',function($scope){

        $scope.user={'username':'','password':''};

        // ---- Users json
        var validUsers= [
                {'username':'chandler@friends.com', 'password':'1234'},
                {'username':'ross@friends.com', 'password':'1234'},
                {'username':'joey@friends.com', 'password':'1234'},
                {'username':'rechal@friends.com', 'password':'1234'}
        ];

        $scope.showError = false; // set Error flag
        $scope.showSuccess = false; // set Success Flag

        // ------ Authenticate function
        $scope.authenticate = function (){
                var flag= false;

                for(var i in validUsers){ // loop on users array
                        if($scope.user.username == validUsers[i].username &&
$scope.user.password == validUsers[i].password){
                                flag = true;
                                break;
                        }
                        else{
                                flag = false;
                        }
                }

                // ------- set error or success flags
                if(flag){
                        $scope.showError = false;
                        $scope.showSuccess = true;
                }
                else{
                        $scope.showError = true;
                        $scope.showSuccess = false;
                }


        }

});
```

Output:
Email

Password

Login

4.3 Authentication and Authorization

Authentication is the process matching the visitor of a web application with the pre-defined set of user identity in the system. In other word, it is the process of recognizing the user's identity. Authentication is very important process in the system with respect to security.

Authorization is the process of giving permission to the user to access certain resource in the system. Only the authenticated user can be authorised to access a resource.

Working example

Let us try to add login functionality to our application and secure it using CanActivate guard.

Open command prompt and go to project root folder.

```
cd /go/to/expense-manager
```

Start the application.

```
ng serve
```

Create a new service, AuthService to authenticate the user.

```
ng generate service auth
CREATE src/app/auth.service.spec.ts (323 bytes)
CREATE src/app/auth.service.ts (133 bytes)
```

Open **AuthService** and include below code.

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

import { Observable, of } from 'rxjs';
import { tap, delay } from 'rxjs/operators';

@Injectable({
   providedIn: 'root'
})
export class AuthService {

   isUserLoggedIn: boolean = false;

   login(userName: string, password: string): Observable {
      console.log(userName);
      console.log(password);
      this.isUserLoggedIn = userName == 'admin' && password == 'admin';
      localStorage.setItem('isUserLoggedIn', this.isUserLoggedIn ? "true" :
"false");

   return of(this.isUserLoggedIn).pipe(
      delay(1000),
      tap(val => {
         console.log("Is User Authentication is successful: " + val);
      })
   );
   }

   logout(): void {
   this.isUserLoggedIn = false;
      localStorage.removeItem('isUserLoggedIn');
   }

   constructor() { }
}
```

Here,

- We have written two methods, **login** and **logout**.
- The purpose of the **login** method is to validate the user and if the user successfully validated, it stores the information in **localStorage** and then returns true.
- Authentication validation is that the user name and password should be **admin.**
- We have not used any backend. Instead, we have simulated a delay of 1s using Observables.
- The purpose of the **logout** method is to invalidate the user and removes the information stored in **localStorage.**

Create a **login** component using below command –

```
ng generate component login
CREATE src/app/login/login.component.html (20 bytes)
CREATE src/app/login/login.component.spec.ts (621 bytes)
CREATE src/app/login/login.component.ts (265 bytes)
CREATE src/app/login/login.component.css (0 bytes)
UPDATE src/app/app.module.ts (1207 bytes)
```

Open **LoginComponent** and include below code –

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

import { FormGroup, FormControl } from '@angular/forms';
import { AuthService } from '../auth.service';
import { Router } from '@angular/router';

@Component({
   selector: 'app-login',
   templateUrl: './login.component.html',
   styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {

   userName: string;
   password: string;
   formData: FormGroup;

   constructor(private authService : AuthService, private router : Router) { }

   ngOnInit() {
      this.formData = new FormGroup({
         userName: new FormControl("admin"),
         password: new FormControl("admin"),
      });
   }

   onClickSubmit(data: any) {
      this.userName = data.userName;
      this.password = data.password;

      console.log("Login page: " + this.userName);
      console.log("Login page: " + this.password);

      this.authService.login(this.userName, this.password)
         .subscribe( data => {
            console.log("Is Login Success: " + data);

           if(data) this.router.navigate(['/expenses']);
      });
   }
}
```

Here,

- Used reactive forms.
- Imported AuthService and Router and configured it in constructor.
- Created an instance of FormGroup and included two instance of FormControl, one for user name and another for password.
- Created a onClickSubmit to validate the user using authService and if successful, navigate to expense list.

Open **LoginComponent** template and include below template code.

```
<!-- Page Content -->
<div class="container">
   <div class="row">
      <div class="col-lg-12 text-center" style="padding-top: 20px;">
         <div class="container box" style="margin-top: 10px; padding-left: 0px;
padding-right: 0px;">
            <div class="row">
               <div class="col-12" style="text-align: center;">
```

```
                        <form [formGroup]="formData"
(ngSubmit)="onClickSubmit(formData.value)"
                            class="form-signin">
                        <h2 class="form-signin-heading">Please sign
in</h2>
                        <label for="inputEmail" class="sr-only">Email
address</label>
                        <input type="text" id="username" class="form-
control"
                            formControlName="userName"
placeholder="Username" required autofocus>
                        <label for="inputPassword" class="sr-
only">Password</label>
                        <input type="password" id="inputPassword"
class="form-control"
                            formControlName="password"
placeholder="Password" required>
                        <button class="btn btn-lg btn-primary btn-block"
type="submit">Sign in</button>
                        </form>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

Here,

Created a reactive form and designed a login form.

Attached the **onClickSubmit** method to the form submit action.

Open **LoginComponent** style and include below CSS Code.

```
.form-signin {
  max-width: 330px;

  padding: 15px;
  margin: 0 auto;
}

input {
   margin-bottom: 20px;
}
```

Here, some styles are added to design the login form.

Create a logout component using below command –

```
ng generate component logout
CREATE src/app/logout/logout.component.html (21 bytes)
CREATE src/app/logout/logout.component.spec.ts (628 bytes)
CREATE src/app/logout/logout.component.ts (269 bytes)
CREATE src/app/logout/logout.component.css (0 bytes)
UPDATE src/app/app.module.ts (1368 bytes)
```

Open **LogoutComponent** and include below code.

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

import { AuthService } from '../auth.service';
import { Router } from '@angular/router';

@Component({
   selector: 'app-logout',
   templateUrl: './logout.component.html',
   styleUrls: ['./logout.component.css']
})
export class LogoutComponent implements OnInit {

   constructor(private authService : AuthService, private router: Router) { }

   ngOnInit() {
      this.authService.logout();
      this.router.navigate(['/']);
   }

}
```

Here,

- Used logout method of AuthService.
- Once the user is logged out, the page will redirect to home page (/).

Create a guard using below command –

```
ng generate guard expense
CREATE src/app/expense.guard.spec.ts (364 bytes)
CREATE src/app/expense.guard.ts (459 bytes)
```

Open ExpenseGuard and include below code –

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, Router,
UrlTree } from '@angular/router';
import { Observable } from 'rxjs';

import { AuthService } from './auth.service';

@Injectable({
  providedIn: 'root'
})
export class ExpenseGuard implements CanActivate {

  constructor(private authService: AuthService, private router: Router) {}

  canActivate(
  next: ActivatedRouteSnapshot,
  state: RouterStateSnapshot): boolean | UrlTree {
    let url: string = state.url;

        return this.checkLogin(url);
    }

    checkLogin(url: string): true | UrlTree {
      console.log("Url: " + url)
      let val: string = localStorage.getItem('isUserLoggedIn');

      if(val != null && val == "true"){
        if(url == "/login")
          this.router.parseUrl('/expenses');
        else
          return true;
      } else {
        return this.router.parseUrl('/login');
      }
    }
}
```

Here,

- checkLogin will check whether the localStorage has the user information and if it is available, then it returns true.
- If the user is logged in and goes to login page, it will redirect the user to expenses page
- If the user is not logged in, then the user will be redirected to login page.

Open **AppRoutingModule (src/app/app-routing.module.ts)** and update below code –

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { ExpenseEntryComponent } from './expense-entry/expense-
entry.component';
import { ExpenseEntryListComponent } from './expense-entry-list/expense-
entry-list.component';
import { LoginComponent } from './login/login.component';
import { LogoutComponent } from './logout/logout.component';

import { ExpenseGuard } from './expense.guard';

const routes: Routes = [
   { path: 'login', component: LoginComponent },
   { path: 'logout', component: LogoutComponent },
   { path: 'expenses', component: ExpenseEntryListComponent, canActivate:
[ExpenseGuard]},
   { path: 'expenses/detail/:id', component: ExpenseEntryComponent,
canActivate: [ExpenseGuard]},
   { path: '', redirectTo: 'expenses', pathMatch: 'full' }
];

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

Here,

- Imported LoginComponent and LogoutComponent.
- Imported ExpenseGuard.
- Created two new routes, login and logout to access LoginComponent and LogoutComponent respectively.
- Add new option canActivate for ExpenseEntryComponent and ExpenseEntryListComponent.

Open **AppComponent** template and add two login and logout link.

```
<div class="collapse navbar-collapse" id="navbarResponsive">
   <ul class="navbar-nav ml-auto">
      <li class="nav-item active">
         <a class="nav-link" href="#">Home
            <span class="sr-only" routerLink="/">(current)</span>

         </a>
      </li>
      <li class="nav-item">
         <a class="nav-link" routerLink="/expenses">Report</a>
      </li>
      <li class="nav-item">
         <a class="nav-link" href="#">Add Expense</a>
      </li>
      <li class="nav-item">

         <a class="nav-link" href="#">About</a>
      </li>
      <li class="nav-item">
                  <div *ngIf="isUserLoggedIn; else isLogOut">
                        <a class="nav-link" routerLink="/logout">Logout</a>
                  </div>

                  <ng-template #isLogOut>
                              <a class="nav-link" routerLink="/login">Login</a>
                  </ng-template>
      </li>
   </ul>
</div>
```

Open **AppComponent** and update below code –

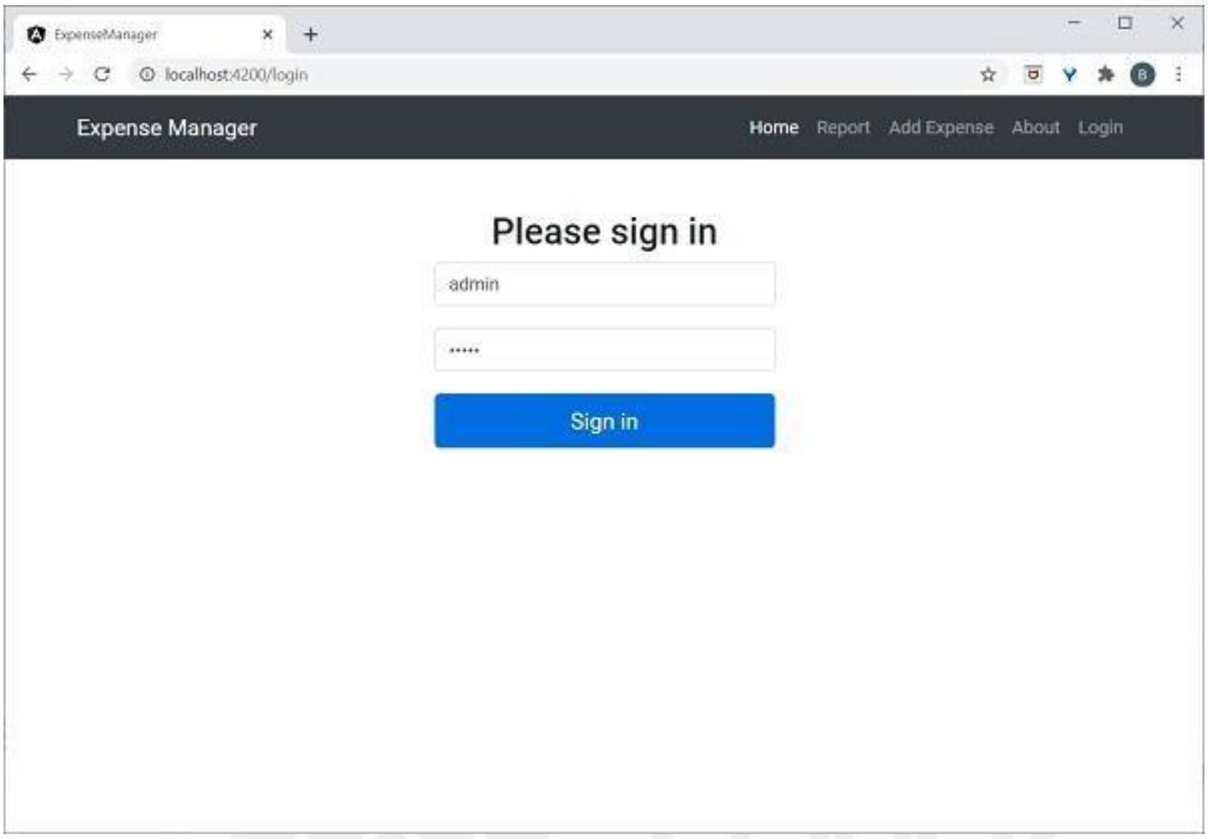
```
import { Component } from '@angular/core';

import { AuthService } from './auth.service';

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

   title = 'Expense Manager';
   isUserLoggedIn = false;

   constructor(private authService: AuthService) {}

   ngOnInit() {
      let storeData = localStorage.getItem("isUserLoggedIn");
      console.log("StoreData: " + storeData);

      if( storeData != null && storeData == "true")
         this.isUserLoggedIn = true;
      else


         this.isUserLoggedIn = false;
   }
}
```

Here, we have added the logic to identify the user status so that we can show login / logout functionality.

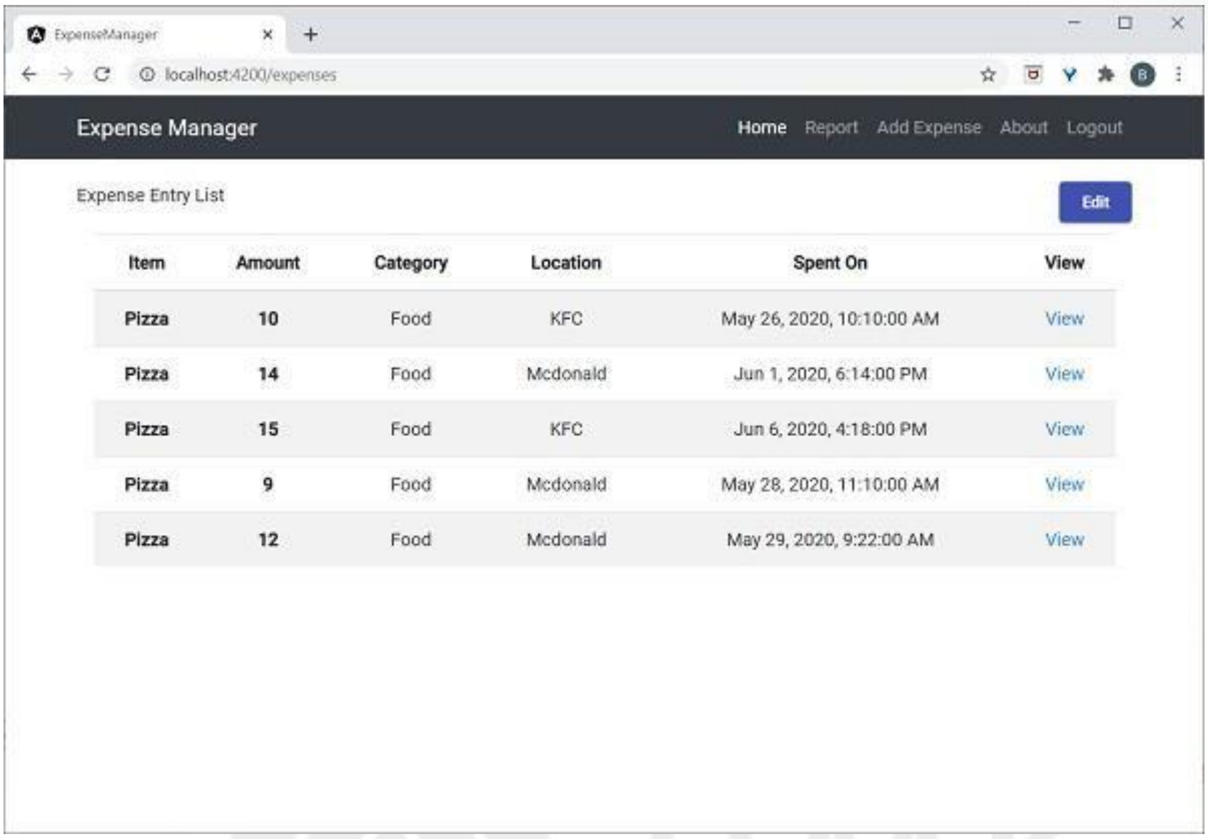
Open **AppModule (src/app/app.module.ts)** and configure **ReactiveFormsModule**

```
import { ReactiveFormsModule } from '@angular/forms';
imports: [
   ReactiveFormsModule
]
```

Now, run the application and the application opens the login page.



Enter admin and admin as username and password and then, click submit. The application process the login and redirects the user to expense list page as shown below –



Expense Entry List

Item	Amount	Category	Location	Spent On	View
Pizza	10	Food	KFC	May 26, 2020, 10:10:00 AM	View
Pizza	14	Food	Mcdonald	Jun 1, 2020, 6:14:00 PM	View
Pizza	15	Food	KFC	Jun 6, 2020, 4:18:00 PM	View
Pizza	9	Food	Mcdonald	May 28, 2020, 11:10:00 AM	View
Pizza	12	Food	Mcdonald	May 29, 2020, 9:22:00 AM	View

Finally, you can click logout and exit the application.

4.4 Customer Service, Http Service in Angular

Using Angular HttpClient

The HttpClient is a separate model in Angular and is available under the @angular/common/http package. The following steps show you how to use the HttpClient in an Angular app.

Import HttpClient Module in Root Module

We need to import it into our root module app.module. Also, we need to add it to the imports metadata array.

```

import { NgModule } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';

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

```

Import Required Module in Component/Service

Then you should import HttpClient the @angular/common/http in the component or service.

```

import { HttpClient } from '@angular/common/http';

```

Inject HttpClient service

Inject the HttpClient service in the constructor.

```

constructor(public http: HttpClient) {
}

```

Call the HttpClient.Get method

Use HttpClient.Get method to send an HTTP Request. The request is sent when we Subscribe to the get() method. When the response arrives map it the desired object and display the result.

```

public getData() {
  this.HttpClient.get<any[]>(this.baseUrl+'users/'+this.userName+'/repos')
           .subscribe(data => {
               this.repos= data;
           },
           error => {
           }
  );
}

```

What is Observable?

The Angular HTTPClient makes use of observable. Hence it is important to understand the basics of it

Observable help us to manage async data. You can think of Observables as an array of items, which arrive asynchronously over time.

The observables implement the observer design pattern, where observables maintain a list of dependents. We call these dependents as observers. The observable notifies them automatically of any state changes, usually by calling one of their methods.

Observer subscribes to an Observable. The observer reacts when the value of the Observable changes. An Observable can have multiple subscribers and all the subscribers are notified when the state of the Observable changes.
When an Observer subscribes to an observable, it needs to pass (optional) the three callbacks. next(), error() & complete(). The observable invokes the next() callback, when it receives an value. When the observable completes it invokes the complete() callback. And when the error occurs it invokes the error() callback with details of error and subscriber finishes.
The Observables are used extensively in Angular. The new HTTPClient Module and Event system are all Observable based.

Observables Operators

Operators are methods that operate on an Observable and return a new observable. Each Operator modifies the value it receives. These operators are applied one after the other in a chain.

The RxJs provides several Operators, which allows you to filter, select, transform, combine and compose Observables. Examples of Operators are map, filter, take, merge, etc

How to use RxJs

The RxJs is a very large library. Hence Angular exposes a stripped-down version of Observables. You can import it using the following import statement

```

import { Observable } from 'rxjs';

```

The above import imports only the necessary features. It does not include any of the Operators.

To use observables operators, you need to import them. The following code imports the map & catchError operators.

```

import { map, catchError } from 'rxjs/operators';

```

HTTP GET

The HttpClient.get sends the HTTP Get Request to the API endpoint and parses the returned result to the desired type. By default, the body of the response is parsed as JSON. If you want any other type, then you need to specify explicitly using the observe & responseType options.

Syntax

```

get(url: string,
    options: {
        headers?: HttpHeaders | { [header: string]: string | string[]; };
        params?: HttpParams | { [param: string]: string | string[]; };
        observe?: "body|events|response|";
        responseType: "arraybuffer|json|blob|text";
        reportProgress?: boolean;
        withCredentials?: boolean;}
   ): Observable<>

```

Options

under the options, we have several configuration options, which we can use to configure the request.

headers It allows you to add HTTP headers to the outgoing requests.
observe The HttpClient.get method returns the body of the response parsed as JSON (or type specified by the responseType). Sometimes you may need to read the entire response along with the headers and status codes. To do this you can set the observe property to the **response**.
The allowed options are

- a response which returns the entire response
- body which returns only the body
- events which return the response with events.

params Allows us to Add the URL parameters or Get Parameters to the Get Request
reportProgress This is a boolean property. Set this to true, if you want to get notified of the progress of the Get Request. This is a pretty useful feature when you have a large amount of data to download (or upload) and you want the user to notify of the progress.
responseType Json is the default response type. In case you want a different type of response, then you need to use this parameter. The Allowed Options are arraybuffer, blob, JSON, and text.
withCredentials It is of boolean type. If the value is true then HttpClient.get will request data with credentials (cookies)

HTTP Post

The HttpClient.post() sends the HTTP POST request to the endpoint. Similar to the get(), we need to subscribe to the post() method to send the request. The post method parsed the body of the response as JSON and returns it. This is the default behavior. If you want any other type, then you need to specify explicitly using the observe & responseType options.

The syntax of the HTTP Post is similar to the HTTP Get.

```

post(url: string,
     body: any,
     options: {
        headers?: HttpHeaders | { [header: string]: string | string[]; };
        observe?: "body|events|response|";
        params?: HttpParams | { [param: string]: string | string[]; };
        reportProgress?: boolean;
        responseType: "arraybuffer|json|blob|text";
        withCredentials?: boolean;
     }
): Observable

```

The following is an example of HTTP Post

```

addPerson(person:Person): Observable<any> {
    const headers = { 'content-type': 'application/json'}
    const body=JSON.stringify(person);
    this.http.post(this.baseURL + 'people', body,{'headers':headers , observe: 'response'})
      .subscribe(
       response=> {
            console.log("POST completed sucessfully. The response received "+response);
        },
        error => {
            console.log("Post failed with the errors");
        },
        () => {
            console.log("Post Completed");
        }
}


```

HTTP PUT

The HttpClient.put() sends the HTTP PUT request to the endpoint. The syntax and usage are very similar to the HTTP POST method.

```
put(url: string,
    body: any,
    options: {
      headers?: HttpHeaders | { [header: string]: string | string[]; };
      observe?: "body|events|response|";
      params?: HttpParams | { [param: string]: string | string[]; };
      reportProgress?: boolean;
      responseType: "arraybuffer|json|blob|text";
      withCredentials?: boolean;
    }
): Observable
```

HTTP PATCH

The HttpClient.patch() sends the HTTP PATCH request to the endpoint. The syntax and usage are very similar to the HTTP POST method.

```
patch(url: string,
    body: any,
    options: {
      headers?: HttpHeaders | { [header: string]: string | string[]; };
      observe?: "body|events|response|";
      params?: HttpParams | { [param: string]: string | string[]; };
      reportProgress?: boolean;
      responseType: "arraybuffer|json|blob|text";
      withCredentials?: boolean;
    }
): Observable
```

HTTP DELETE

The HttpClient.delete() sends the HTTP DELETE request to the endpoint. The syntax and usage are very similar to the HTTP GET method.

```
delete(url: string,
     options: {
        headers?: HttpHeaders | { [header: string]: string | string[]; };
        params?: HttpParams | { [param: string]: string | string[]; };
        observe?: "body|events|response|";
        responseType: "arraybuffer|json|blob|text";
        reportProgress?: boolean;
        withCredentials?: boolean;}
    ): Observable<>
```

HttpClient Example

Now, We have a basic understanding of HttpClient model & observables, let us build an HttpClient example app.

Create a new Angular app

```
ng new httpClient
```

import HttpClientModule

In the app,module.ts import the HttpClientModule module as shown below. We also add it to the imports array

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

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


```

Component

Now, open the app.component.ts and copy the following code.

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

export class Repos {
  id: string;
  name: string;
  html_url: string;
  description: string;
}

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent implements OnInit {

  userName: string = "tektutorialshub"
  baseURL: string = "https://api.github.com/";
  repos: Repos[];


  constructor(private http: HttpClient) {
  }

  ngOnInit() {
    this.getRepos()
  }


  public getRepos() {

    return this.http.get<Repos[]>(this.baseURL + 'users/' + this.userName + '/repos')
      .subscribe(
        (response) => {                           //Next callback
          console.log('response received')
          console.log(response);
          this.repos = response;
        },
        (error) => {                              //Error callback
          console.error('Request failed with error')
          alert(error);
        },
        () => {                                   //Complete callback
          console.log('Request completed')
        })
  }
}

```

Import HTTPClient

HTTPClient is a service, which is a major component of the HTTP Module. It contains methods like `GET`, `POST`, `PUT` etc. We need to import it.

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

Repository Model

The model to handle our data.

```
export class Repos {
  id: string;
  name: string;
  html_url: string;
  description: string;
}
```

Inject HttpClient

Inject the HttpClient service into the component.

```
constructor(private http: HttpClient) {
}
```

Subscribe to HTTP Get

The GetRepos method, we invoke the get() method of the HttpClient Service. The HttpClient.get method allows us to cast the returned response object to a type we require. We make use of that feature and supply the type for the returned value http.get<repos[]>
The get() method returns an observable. Hence we subscribe to it.

```
public getRepos() {
  return this.http.get<Repos[]>(this.baseURL + 'users/' + this.userName + '/repos')
     .subscribe(
```

When we subscribe to any observable, we optionally pass the three callbacks. next(), error() & complete(). In this example we pass only two callbacks next() & error().

Receive the Response

We receive the data in the next() callback. By default, the Angular reads the body of the response as JSON and casts it to an object and returns it back. Hence we can use directly in our app.

```
      (response) => {                        //Next callback
        console.log('response received')
        console.log(response);
        this.repos = response;
      },

```

Handle the errors

We handle the errors in `error` callback.

```

     (error) => {                              //Error callback
        console.error('Request failed with error')
        alert(error);
      },

```

Template

```

<h1 class="heading"><strong>HTTPClient </strong> Example</h1>


<table class='table'>
  <thead>
    <tr>
      <th>ID</th>
      <th>Name</th>
      <th>HTML Url</th>
      <th>description</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let repo of repos;">
      <td>{{repo.id}}</td>
      <td>{{repo.name}}</td>
      <td>{{repo.html_url}}</td>
      <td>{{repo.description}}</td>
    </tr>
  </tbody>
</table>

<pre>{{repos | json}}</pre>

```

4.5 HTTP Token Interceptors in Angular

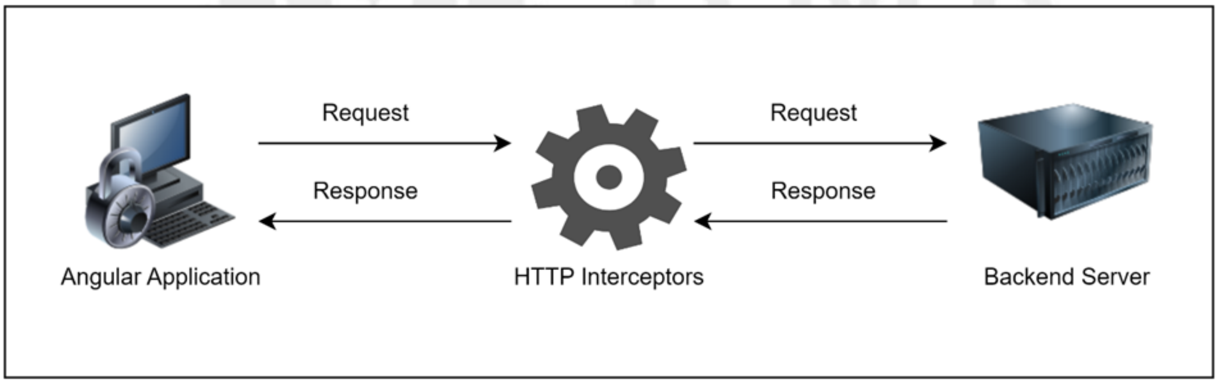
What is an HTTP Interceptor?

· HTTP Interceptors are a concept in web development and server-side programming, typically associated with web frameworks and libraries.

· These interceptors allow developers to intercept and handle HTTP requests and responses globally within an application.

HTTP Interceptors in Angular

· HTTP Interceptors in Angular are classes that implement the **HttpInterceptor** interface.



HTTP Interceptor

· They can be used to perform various tasks related to HTTP requests and responses, such as adding headers, handling errors, modifying the request or response data, logging, authentication, etc.

· **HttpInterceptor** defines a single method called **intercept**, which takes two parameters: the **HttpRequest** and the **HttpHandler**.

Benefits of HTTP Interceptors

Following are some of the key benefits of using HTTP Interceptors in Angular:

Testability and reusability: Interceptors are easy to test in isolation, allowing you to ensure that each interceptor behaves correctly

Centralized code for cross-cutting concerns: HTTP Interceptors allow you to define logic for common tasks, such as authentication, logging, error handling, or adding headers, in a centralized location.

Global application-level modifications: Interceptors operate globally, intercepting all HTTP requests and responses made by the Angular application. This means you can apply changes or perform actions consistently across multiple API calls without having to modify each individual request or response manually.

Error handling and logging: Interceptors can be utilized to handle errors globally, providing a consistent approach to error reporting and handling throughout the application.

Caching and request/response manipulation: HTTP Interceptors can be leveraged to implement caching mechanisms, reducing redundant requests and optimizing the application's performance.

Separation of concerns: By using HTTP Interceptors, you can keep concerns related to data fetching and communication (HTTP) separate from the business logic of your components and services.

Security and authentication: Interceptors are commonly used for adding authorization headers or authentication tokens to outgoing requests. This ensures that the user's authentication status is automatically included in API calls without the need to explicitly set headers in every request.

Easy integration with third-party libraries: Interceptors can be used to integrate with third-party libraries or APIs seamlessly. For example, you can apply a specific format to API responses that are expected by a charting library or a data visualization tool.

Practical Implementation

Let's start with practical implementation; for that, we need to create a new Angular application using the following command.

```
ng new angular-http-interceptor-demo
```

Now, we are going to create different interceptors one-by-one with the help of angular.

1. Logging Interceptor

In Angular, logging interceptors can be used for audit log purposes. If we want to log different incoming and outgoing requests with request and response objects, we can do so with the help of a logging interceptor.

Step 1

Create a new logging interceptor with the help of the following command.

```
ng g interceptor logging
```

This command will create the logging interface with a default implementation. So, modify the same as I have shown below.

```
import { Injectable } from '@angular/core';
import {
  HttpEvent,
  HttpInterceptor,
  HttpHandler,
  HttpRequest,
  HttpResponse,
} from '@angular/common/http';
import { Observable, tap } from 'rxjs';

@Injectable()
export class LoggingInterceptor implements HttpInterceptor {
  constructor() {}

  intercept(
    request: HttpRequest<any>,
    next: HttpHandler
  ): Observable<HttpEvent<any>> {
    console.log('Outgoing HTTP request', request);
    return next.handle(request).pipe(
      tap((event: HttpEvent<any>) => {
        console.log('Incoming HTTP response', event);
      })
    );
  }
}
```

· Here, we import the necessary modules and classes from Angular’s HTTP package.

· The **HttpInterceptor** interface allows us to create our custom interceptor, and **HttpRequest**, **HttpHandler**, and **HttpEvent** are classes used for handling HTTP requests and responses.

· We also import Observable and Tap from the RxJS library, which is used for handling asynchronous operations.

· We call **next.handle(request)** to pass the request to the next interceptor in the chain, or the backend server.

· Then, we use the **pipe** method along with the **tap** operator to intercept the incoming response.

· The tap operator allows us to execute a side effect (in this case, log the response) without modifying the response itself.

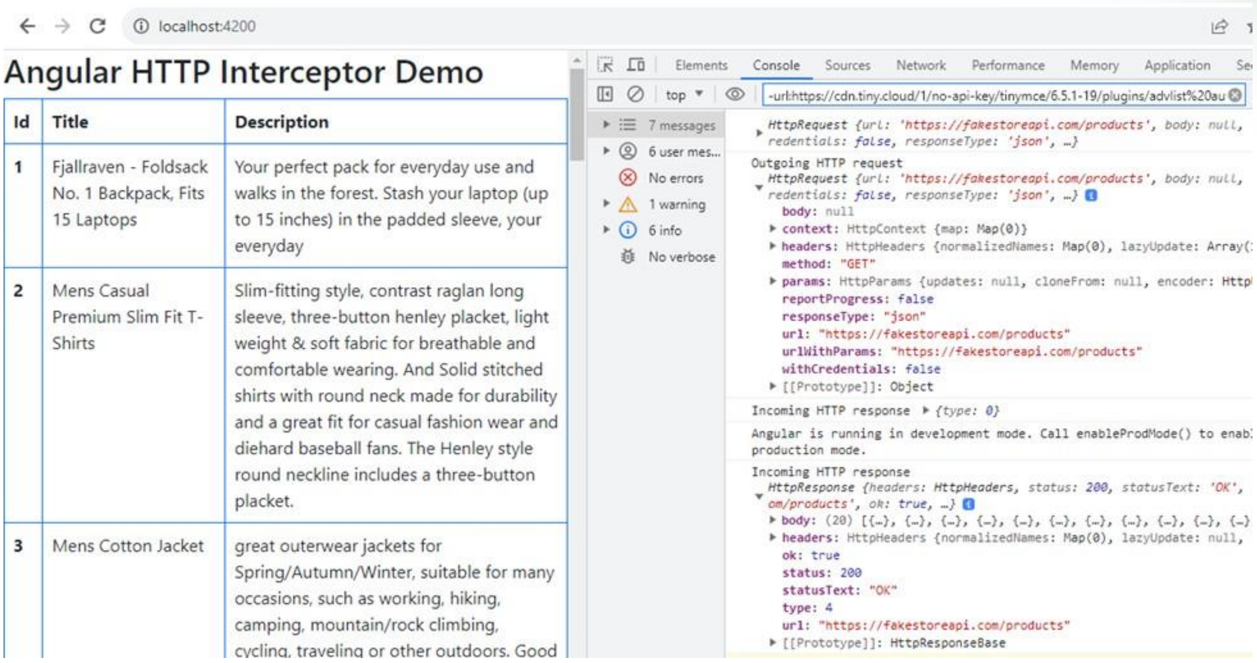
Step 2

Provide an interceptor in the app module:

```
import { LoggingInterceptor } from './interceptors/logging.interceptor'

 providers: [
   {
     provide: HTTP_INTERCEPTORS, useClass: LoggingInterceptor, multi: true
   }
  ]
```

In the AppModule, we provide the LoggingInterceptor class as an interceptor using the HTTP_INTERCEPTORS token. The multi: true option ensures that the interceptor is appended to the existing array of interceptors rather than replacing them.

When you make an HTTP request, it will get logged with the following request and response:



localhost:4200

Angular HTTP Interceptor Demo

Id	Title	Description
1	Fjallraven - Foldsack No. 1 Backpack, Fits 15 Laptops	Your perfect pack for everyday use and walks in the forest. Stash your laptop (up to 15 inches) in the padded sleeve, your everyday
2	Mens Casual Premium Slim Fit T-Shirts	Slim-fitting style, contrast raglan long sleeve, three-button henley placket, light weight & soft fabric for breathable and comfortable wearing. And Solid stitched shirts with round neck made for durability and a great fit for casual fashion wear and diehard baseball fans. The Henley style round neckline includes a three-button placket.
3	Mens Cotton Jacket	great outerwear jackets for Spring/Autumn/Winter, suitable for many occasions, such as working, hiking, camping, mountain/rock climbing, cycling, traveling or other outdoors. Good

```
-url:https://cdn.tiny.cloud/1/no-api-key/tinymce/6.5.1-19/plugins/advlist%20au

 HttpRequest {url: 'https://fakestoreapi.com/products', body: null,
 redentials: false, responseType: 'json', …}
Outgoing HTTP request
 HttpRequest {url: 'https://fakestoreapi.com/products', body: null,
 redentials: false, responseType: 'json', …}
   body: null
 ▸ context: HttpContext {map: Map(0)}
 ▸ headers: HttpHeaders {normalizedNames: Map(0), lazyUpdate: Array(
   method: "GET"
 ▸ params: HttpParams {updates: null, cloneFrom: null, encoder: Http
   reportProgress: false
   responseType: "json"
   url: "https://fakestoreapi.com/products"
   urlWithParams: "https://fakestoreapi.com/products"
   withCredentials: false
 ▸ [[Prototype]]: Object
Incoming HTTP response ▸ {type: 0}
Angular is running in development mode. Call enableProdMode() to enab
production mode.
Incoming HTTP response
 HttpResponse {headers: HttpHeaders, status: 200, statusText: 'OK',
 om/products', ok: true, …}
 ▸ body: (20) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}
 ▸ headers: HttpHeaders {normalizedNames: Map(0), lazyUpdate: null,
   ok: true
   status: 200
   statusText: "OK"
   type: 4
   url: "https://fakestoreapi.com/products"
 ▸ [[Prototype]]: HttpResponseBase
```

Logging Interceptor

In a real-time scenario, you can log this response in a third-party service as per need and requirement.

2. Adding Headers to Requests

In Angular, we can modify HTTP Requests and add some extra value to the request header with the help of an interceptor.

Step 1

Create a new header interceptor with the help of the following command:

ng g interceptor headers

```
import { Injectable } from '@angular/core';
import {
  HttpRequest,
  HttpHandler,
  HttpEvent,
  HttpInterceptor
} from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable()
export class HeadersInterceptor implements HttpInterceptor {

  constructor() {}

  intercept(request: HttpRequest<unknown>, next: HttpHandler):
Observable<HttpEvent<unknown>> {
    console.log(request)
    const GUID = 'f4179b26-21ac-432c-bcd8-cb4bc6e50981'
    const modifiedRequest = request.clone({
      setHeaders:{
        GUID
      }
    })
    return next.handle(modifiedRequest);
  }
}
```

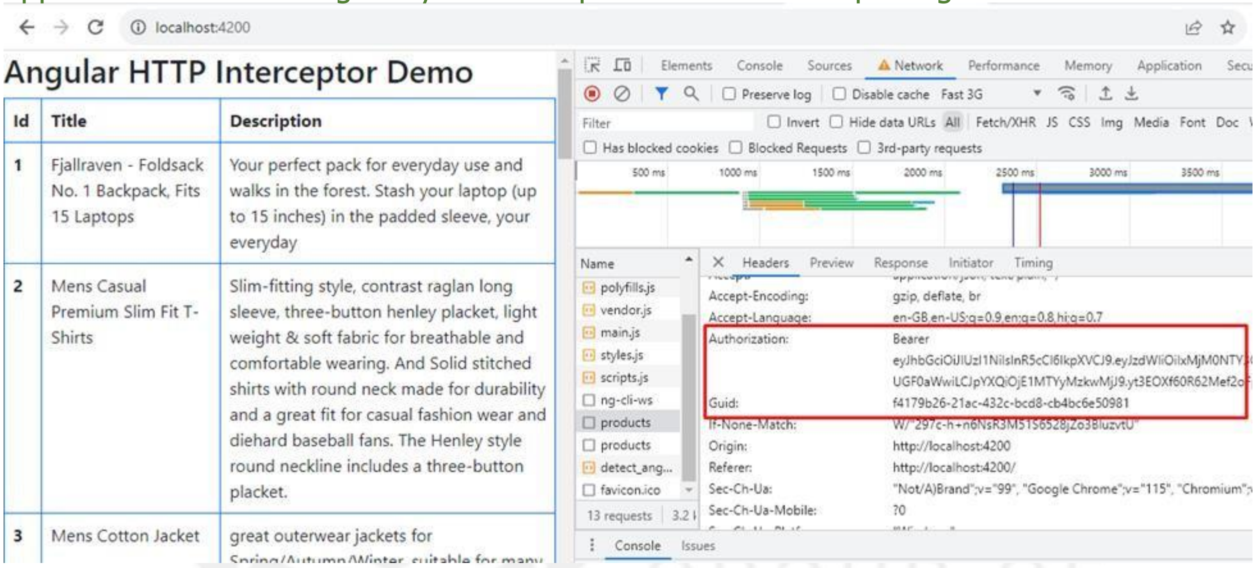
Here we first hardcode one GUID that we are going to set inside the header. So, first, we need to clone that HTTP request and use the set headers property to set the value in the request header.

Step 2

Provide an interceptor in the app module:

```
import { HeadersInterceptor } from './interceptors/headers.interceptor'

providers: [
    {
      provide: HTTP_INTERCEPTORS, useClass: HeadersInterceptor, multi: true
      }
    ]
```

In the AppModule, we provide the HeadersInterceptor class as an interceptor using the HTTP_INTERCEPTORS token. The multi: true option ensures that the interceptor is appended to the existing array of interceptors rather than replacing them.



Angular HTTP Interceptor Demo

Id	Title	Description
1	Fjallraven - Foldsack No. 1 Backpack, Fits 15 Laptops	Your perfect pack for everyday use and walks in the forest. Stash your laptop (up to 15 inches) in the padded sleeve, your everyday
2	Mens Casual Premium Slim Fit T-Shirts	Slim-fitting style, contrast raglan long sleeve, three-button henley placket, light weight & soft fabric for breathable and comfortable wearing. And Solid stitched shirts with round neck made for durability and a great fit for casual fashion wear and diehard baseball fans. The Henley style round neckline includes a three-button placket.
3	Mens Cotton Jacket	great outerwear jackets for Spring/Autumn/Winter, suitable for many

```
Name
polyfills.js
vendor.js
main.js
styles.js
scripts.js
ng-cli-ws
products
products
detect_ang...
favicon.ico
13 requests
```

```
Accept-Encoding:      gzip, deflate, br
Accept-Language:      en-GB,en-US;q=0.9,en;q=0.8,hi;q=0.7
Authorization:        Bearer
                      eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY
                      UGF0aWwiLCJpYXQiOjE1MTYyMzkwMjJ9.yt3EOXf60R62Mef2o
Guid:                 f4179b26-21ac-432c-bcd8-cb4bc6e50981
If-None-Match:        W/"297c-h+n6NsR3M51S6528jZo3BluzvtU"
Origin:               http://localhost:4200
Referer:              http://localhost:4200/
Sec-Ch-Ua:            "Not/A)Brand";v="99", "Google Chrome";v="115", "Chromium";
Sec-Ch-Ua-Mobile:     ?0
```

Set Headers Interceptor

In a real-time scenario, you can use these header values for further processing, like validating requests, and in many other cases.

3. Error Handling Interceptor

In Angular, The Error interceptor is an HTTP interceptor that allows you to handle HTTP errors globally within your application.

When you make HTTP requests to a server, there might be scenarios where the server responds with an error status code, such as 404 or 500.

Handling these errors in each individual HTTP request can be tedious and repetitive.

The Error Interceptor helps you centralize the error-handling logic and provides a consistent way to manage errors across your application.

Step 1

Create a new error interceptor with the help of the following command.

ng g interceptor error

```
import { Injectable } from '@angular/core';
import {
  HttpRequest,
  HttpHandler,
  HttpEvent,
  HttpInterceptor,
  HttpErrorResponse
} from '@angular/common/http';
import { Observable, catchError, throwError } from 'rxjs';

@Injectable()
export class ErrorInterceptor implements HttpInterceptor {

  constructor() {}

  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>>
{
    return next.handle(request).pipe(
      catchError((error: HttpErrorResponse) => {
        // Handle the error here
        console.error('error occurred:', error);
        //throw error as per requirement
        return throwError(error);
      })
    );
  }
}
```

· Inside the **intercept()** method, you can use the **catchError** operator from RxJS to catch any errors that occur during the HTTP request or response handling.

· This operator allows you to intercept the error, handle it as needed, and optionally re-throw the error to propagate it further up the observable chain.

Step 2

· Provide interceptor in the app module:

```
import { ErrorInterceptor } from './interceptors/error.interceptor';
  providers: [
    {
      provide: HTTP_INTERCEPTORS, useClass: ErrorInterceptor, multi: true
    }
  ]
```

· In the AppModule, we provide the HeadersInterceptor class as an interceptor using the HTTP_INTERCEPTORS token. The multi: true option ensures that the interceptor is appended to the existing array of interceptors rather than replacing them.

4. Authentication Interceptor

In Angular, an authentication interceptor can be used to add authentication tokens or headers to every outgoing HTTP request. This is helpful when you need to ensure that all API requests are authenticated.

Step 1

Create a new authentication interceptor with the help of the following command.

```
ng g interceptor auth
```

```
import { Injectable } from '@angular/core';
import {
  HttpEvent,
  HttpInterceptor,
  HttpHandler,
  HttpRequest,
} from '@angular/common/http';
import { Observable } from 'rxjs';
//import { AuthService } from './auth.service';

@Injectable()
export class AuthInterceptor implements HttpInterceptor {
  constructor(/*private authService: AuthService*/) {}

  intercept(
    req: HttpRequest<any>,
    next: HttpHandler
  ): Observable<HttpEvent<any>> {
    const authToken =
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpheW
RlZXAgUGF0aWwiLCJpYXQiOjE1MTYyMzkwMjJ9.yt3EOXf60R62Mef2oFpbFh2ihkP5qZ4fM8bjV
nF8YhA";//his.authService.getToken();

    if (authToken) {
      // Clone the request and attach the token
      const authReq = req.clone({
      setHeaders: {
          Authorization: `Bearer ${authToken}`
        }
      });

      return next.handle(authReq);
    }

    // If there is no token, pass the original request
    return next.handle(req);
  }
}
```

Here we first hardcode one token that we are going to set inside the header. So, for that first, we need to clone that HTTP request and need to use the set headers property to set the value in the request header.

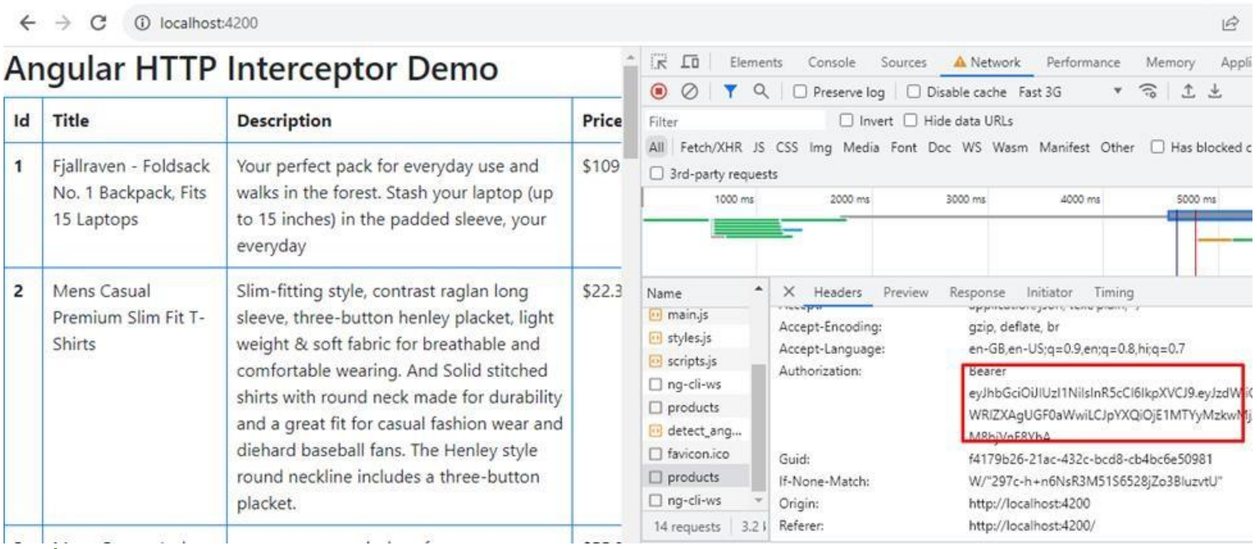
Step 2

Provide an interceptor in the app module:

```
import { AuthInterceptor } from './interceptors/auth.interceptor';

 providers: [
  {
   provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi: true
  }
```

In the AppModule, we provide the HeadersInterceptor class as an interceptor using the HTTP_INTERCEPTORS token. The multi: true option ensures that the interceptor is appended to the existing array of interceptors rather than replacing them.

When you make an HTTP request, it will set a token inside the header, as shown below.



Angular HTTP Interceptor Demo

Id	Title	Description	Price
1	Fjallraven - Foldsack No. 1 Backpack, Fits 15 Laptops	Your perfect pack for everyday use and walks in the forest. Stash your laptop (up to 15 inches) in the padded sleeve, your everyday	$109
2	Mens Casual Premium Slim Fit T-Shirts	Slim-fitting style, contrast raglan long sleeve, three-button henley placket, light weight & soft fabric for breathable and comfortable wearing. And Solid stitched shirts with round neck made for durability and a great fit for casual fashion wear and diehard baseball fans. The Henley style round neckline includes a three-button placket.	$22.3

```
Accept-Encoding:     gzip, deflate, br
Accept-Language:     en-GB,en-US;q=0.9,en;q=0.8,hi;q=0.7
Authorization:       Bearer
                     eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdW
                     WRlZXAgUGF0aWwiLCJpYXQiOjE1MTYyMzkwMj
                     M8hjVnF8YhA
Guid:                f4179b26-21ac-432c-bcd8-cb4bc6e50981
If-None-Match:       W/"297c-h+n6NsR3M51S6528jZo3BluzvtU"
Origin:              http://localhost:4200
Referer:             http://localhost:4200/
```

Authentication Interceptor

4.6 Providing Compile Time and Runtime Configuration in Angular

Compile time configuration: environment files

The main idea is to use the environment mechanism provided by Angular. It is possible to create multiple environment files:

```
//environment.dev.ts
                    export const environment = {
                      appInsightKey: 'A Key',
                    };


                    //environment.test.ts
                    export const environment = {
                      appInsightKey: 'B Key',
                    };


                    //environment.prod.ts
                    export const environment = {
                      appInsightKey: 'C Key',
                    };
```

Next, declare different configurations in angular.json to replace the default environment.ts with the correct one. Like this:

```
"configurations":
{
                  "test-env": {
                    "fileReplacements": [
                      {
                      "replace": "src/environments/environment.ts",
                      "with": "src/environments/environment.test.ts"
                      }
                    ],
                  },
                  ...
                }
```

And finally, build the application with `-- configuration` option specifying the target environment:

```
ng build --configuration test-env
```

The main drawback of this solution is that it does not work well with CI/CD pipelines. The environment files are consumed at build time by Angular, meaning that you will need to **build the application for each environment**. No build-once => deploy everywhere.

Furthermore, you will have to maintain multiple environment files, which might not be ideal.

Runtime configuration

A better solution is to set up runtime configuration in the Angular app so that the instrumentation key is set during application bootstrap.

This solution allows us to reuse the same build artifact for any target environment. Build once => deploy everywhere 😃.

First of all, you will need to create a config file in JSON format under `assets/`folder (we don't want this file to be bundled during the build process) called`app.config.json` :

```
{
        "appInsightsKey": ""
     }
```

Next, create a new Angular service, responsible for loading the configuration and providing it to any component/service requiring it:

```
//AppConfig.d.ts
                         export interface AppConfig {
                             appInsightsKey: string;
}


//AppConfigService.ts
@Injectable()
export class AppConfigService {
    private config: AppConfig;
    loaded = false;
    constructor(private http: HttpClient) {}
    loadConfig(): Promise<void> {
        return this.http
            .get<AppConfig>('/assets/app.config.json')
            .toPromise()
            .then(data => {
                this.config = data;
                this.loaded = true;
            });
    }

    getConfig(): AppConfig {
        return this.config;
    }
}
```

Problem: How can we ensure that this configuration is loaded **before** the application is up and running (i.e. avoid cases when the config is required but has not been fully loaded yet)?

The ***solution*** is to use Angular's APP_INITIALIZER token to hook in the application bootstrap step and load the config. To do this, edit the `app.module`and add the following:

```
import              {
NgModule,
APP_INITIALIZER
}             from
'@angular/core';


export function initConfig(appConfig: AppConfigService) {
    return () => appConfig.loadConfig();
}


@NgModule({
  providers: [
    {
      provide: APP_INITIALIZER,
      useFactory: initConfig,
      deps: [AppConfigService],
      multi: true,
    },
  ],
  ...
})
```

Using the AppConfigService

This service can be now used to provide the instrumentation key required by the Application Insight service.

```
export class
ApplicationInsightsService
{
  appInsights: ApplicationInsights;

  constructor(private appConfigService: AppConfigService) {
    const appInsightsKey =
this.appConfigService.getConfig().appInsightsKey;
    ...
  }
  ...
}
```

ERROR HANDLING
ErrorHandler
CLASSFINAL

Provides a hook for centralized exception handling.

See more...

```
class                    ErrorHandler                    {
handleError(error: any): void

}
```

Description

The default implementation of ErrorHandler prints error messages to the `console`. To intercept error handling, write a custom exception handler that replaces this default as appropriate for your app.

Further information is available in the Usage Notes...

Methods

handleError()
code

handleError(error: any): void
Parameters

error	any

Returns

void

Usage notes
Example

```
content_copyclass MyErrorHandler implements ErrorHandler {

  handleError(error) {

    // do something with the exception

  }

}

@NgModule({

  providers: [{provide: ErrorHandler, useClass: MyErrorHandler}]

})

class MyModule {}
```

9. Assignment Questions

Category - 1

S.No.	Write Programs for the following	K - Level	COs
1	Create a responsive user interface using Bootstrap's grid system within an Angular component. Explain how Bootstrap classes are utilized for layout design.	K2	CO4
2	Set up an Angular application for hosting the chatbot. Explain the basics of Angular CLI and project structure.	K3	CO4

Category - 2

S.No.	Write Programs for the following	K - Level	COs
1	Handle errors gracefully, such as invalid authentication tokens or failed API requests, by displaying appropriate error messages to users.	K5	CO4
2	Implement an authentication service to manage user registration and login. Discuss the importance of authentication in tracking user progress and personalization.	K3	CO4

Category - 3

S.No.	Write Programs for the following	K - Level	COs
1	Implement error handling mechanisms to handle validation errors, server-side errors, and synchronization errors when updating budget data.	K5	CO4
2	Implement error handling mechanisms to handle invalid input errors, calculation errors, and server-side errors when saving calculator history.	K5	CO4

R.M.K GROUP OF INSTITUTIONS

10. Assignment Questions

Category - 4

S.No.	Write Programs for the following	K - Level	COs
1	Implement error handling mechanisms to handle validation errors, server-side errors, and payment processing errors during the checkout process.	K5	CO4
2	Implement error handling mechanisms to handle network errors, server-side errors, and user input validation errors on mobile devices.	K5	CO4

Category - 5

S.No.	Write Programs for the following	K - Level	COs
1	Implement error handling mechanisms tohandle network errors, server-side errors, and user input validation errors.	K5	CO4
2	Use route guards to restrict access to the calculator history feature to authenticated users only. Discuss the importance of data privacy in personal finance Tools	K5	CO4

PART A

1. What is Angular?

Angular is a platform and framework for building single-page client applications using HTML and TypeScript. Angular is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that you import into your applications..

2. What is feature modules in Angular?

In Angular, a feature module is a way to organize and encapsulate a specific set of features and functionality within an application. · It contains a group of related components, directives, services, and a few other files. It helps us maintain and manage the application codebase.

3. What are directives? Name some of the most commonly used directives in AngularJS application.

A directive is something that introduces new syntax. They are like markers on the DOM element, which attaches a special behavior to it. In any AngularJS application, directives are the most important components.

Some of the commonly used directives are:

1) ng-model 2) ng-App 3) ng-bind 4) ng-repeat 5) ng-show

4. Explain the purpose of Angular's ngIf directive.

Angular's ngIf directive is used to conditionally render or remove an element from the DOM based on a boolean expression. It evaluates the expression assigned to it and renders the associated element if the expression evaluates to true, otherwise, it removes the element from the DOM. This is particularly useful for dynamically showing or hiding elements based on certain conditions, improving performance by not rendering unnecessary DOM elements

5. What are steps to create Angular First App:

Angular provides a powerful framework for building single-page web applications. To create your first Angular app, you can use the Angular CLI (Command Line Interface) to generate a new project scaffold. Here are the steps:

```
npm install -g @angular/cli      # Install Angular CLI globally
ng new my-first-app              # Create a new Angular project
cd my-first-app                  # Navigate into the project directory
ng serve --open                  # Launch the development server and open the app in the
```

PART A

6. Explain directives and their types

During compilation process, when specific HTML function is triggered, it is referred to as directive. It is executed when the compiler encounters it in the DOM.

Different types of directives are:

1) Element directives

2) Attribute directives

3) CSS class directives

4) Comment directives.

7. What is DI (Dependency Injection) and how an object or function can get a hold ofits dependencies?

DI or Dependency Injection is a software design pattern that deals with how code gets hold of its dependencies. In order to retrieve elements of the application which is required to be configured when the module gets loaded, the operation "config" uses dependency injection.

These are the ways that object uses to hold of its dependenciesTypically using the new operator, dependency can be created By referring to a global variable, dependency can be looked up Dependency can be passed into where it is required.

8. What is an HTTP Interceptor?

An HTTP Interceptor in Angular is a middleware component that intercepts outgoing HTTP requests or incoming HTTP responses, allowing you to perform various actions such as modifying headers, logging requests, handling errors, and more. Interceptors are a powerful tool for managing HTTP communication across an Angular application. These interceptors allow developers to intercept and handle HTTP requests and responses globally within an application.

HttpInterceptor defines a single method called intercept, which takes two parameters: the HttpRequest and the HttpHandler.

PART A

8. Explain ng-click directives in AngularJS with example

Ng-click directives can be used in a scenario when you have to click on the button or want to perform any operation.

Example:

<button ng-click="count = count ++">Click</button>

9. What is Authentication?

The authentication is a service that is used to login and logout of Angular application. The credentials of users pass to API on the server. Then post server-side validation these credentials, JSON Web Token is returned, which as detail about the current user.

10. What is Angular CLI?

Angular CLI is also called as the command line interface tool. It is used to build, initialize, and maintain Angular apps. CLI software can be used through very interactive UI like a command shell or Angular Console.

11. What are the directives and mention some of the important types of directives?

Ans: AngularJs extend the HTML with new attributes to create directives. Angularjs has a set of built-in directives that provide functionalities to your applications. It allows you to create directives to develop your applications. These are the distinctive attributes with ng-prefix, that notifies the AngularJs compiler to attach a specified behavior to that DOM element.

The following are the AngularJs built-in directives:

ngBind ngModel ngClass

ngBind: It is used to bind the variable values/models to angular js application and also controls the HTML tag attributes like and more, but it does not support two-way binding. Using ngBind, you can see the output of the model values.

ngModel: This directive is used to bind the model values in the angularJs application. It provides two-way data binding with the model value, and in some cases, it is used for data-binding.

ngClass: This directive allows you to set CSS classes on an HTML element by data binding an expression that demonstrates all classes to be added.

PART A

12. Write Angular script for displaying greeting hello world

```
// app.component.ts
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: '<h1>{{ greeting }}</h1>',
})
export class AppComponent {
  greeting = 'Hello, World!';
}
```

13. Write Angular script for binding to user input

```
// app.component.ts
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: '<input type="text" [(ngModel)]="name"> <p>Hello, {{ name }}</p>',})
export class AppComponent {
  name = '';
}
```

14. Write Angular script for ng-click event

```
// app.component.ts
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: '<button (click)="onClick()">Click me</button>',
})
export class AppComponent {
  onClick() {
    alert('Button clicked!');
  }
}
```

PART A

15. What are benefits of HTTP Interceptors

Following are some of the key benefits of using HTTP Interceptors in Angular:

Testability and reusability: Interceptors are easy to test in isolation, allowing you to ensure that each interceptor behaves correctly

Global application-level modifications: Interceptors operate globally, intercepting all HTTP requests and responses made by the Angular application. This means you can apply changes or perform actions consistently across multiple API calls without havingto modify each individual request or response manually.

Error handling and logging: Interceptors can be utilized to handle errors globally, providing a consistent approach to error reporting and handling throughout the application.

Caching and request/response manipulation: HTTP Interceptors can be leveraged to implement caching mechanisms, reducing redundant requests and optimizing the application's performance.

Security and authentication: Interceptors are commonly used for adding authorization headers or authentication tokens to outgoing requests. This ensures that the user's authentication status is automatically included in API calls without the need to explicitlyset headers in every request.

Easy integration with third-party libraries: Interceptors can be used to integrate with third-party libraries or APIs seamlessly. For example, you can apply a specific format toAPI responses that are expected by a charting library or a data visualization tool.

PART B

1. Explain the steps involved in creating your first Angular application using Angular CLI. CO4 K1
2. Discuss the purpose of the main files and directories generated by Angular CLI in a new project. CO4 K1
3. Describe the process of integrating Bootstrap CSS into an Angular application.CO4 K3
4. Explain how Angular directives and components can be styled using Bootstrap classes.CO4 K3
5. Discuss the role of an authentication service in an Angular application.CO4 K2
6. Explain the methods provided by an authentication service for user authentication and session management.CO4 K2
7. Describe the importance of unsubscribing from observables in Angular applications.CO4 K3
8. Discuss the steps involved in implementing logout functionality and route guard cleanup in Angular.CO4 K2
9. Discuss the role of a token interceptor in Angular's HTTP communication.CO4 K2
10. Explain how a token interceptor can be used to attach authentication tokens to outgoing HTTP requests.CO4 K3
11. Differentiate between compile-time configuration and runtime configuration in Angular applications.CO4 K4
12. Provide examples of compile-time and runtime configurations in Angular.CO4 K4
13. Discuss strategies for error handling in Angular applications.CO4 K2
14. Explain how to implement global error handling and display user-friendly error messages in Angular.CO4 K2

13. Supportive online courses

Online courses

1. https://www.udemy.com/topic/spring-boot/
2. https://www.guvi.in/mlp/join-full-stack-program
3. https://www.codingninjas.com/careercamp/professionals/
4. https://www.coursera.org/learn/spring-repositories
5. https://www.coursera.org/learn/google-cloud-java-spring

External Links for Additional Resources

1. https://spring.io/guides/gs/spring-boot/
2. https://www.baeldung.com/spring-boot-start
3. https://www.interviewbit.com/spring-boot-interview-questions/

14.Real Time Applications

E-commerce Platform:
Social Media Dashboard
Collaborative Document Editing

14. Content Beyond Syllabus

Performance monitoring and analytics in AngularJS

Performance monitoring and analytics in AngularJS involve tracking and optimizing the performance of AngularJS applications to ensure they meet desired speed and efficiency standards. Here's how you can approach this:

AngularJS Batarang: AngularJS Batarang is a Chrome browser extension that helps developers debug and profile AngularJS applications. It provides insights into the performance of AngularJS components, directives, controllers, and services. By using Batarang, you can identify performance bottlenecks and optimize your application accordingly.

$watch/$digest Cycle Optimization: One of the key performance areas in AngularJS is the $digest cycle. It's essential to minimize the number of watchers and reduce the complexity of digest cycles to improve the overall performance of your application. You can use tools like Batarang to analyze the digest cycle and identify areas for optimization.

Instrumentation and Logging: Instrument your AngularJS application with logging and monitoring tools to track key performance metrics such as page load times, API response times, and user interactions. Tools like Google Analytics, New Relic, or custom logging solutions can provide valuable insights into your application's performance.

Lazy Loading and Code Splitting: Implement lazy loading and code splitting techniques to load only the necessary code and resources when they are needed. This can improve initial page load times and reduce the overall size of your application, leading to better performance.

Performance Budgeting: Define performance budgets for your AngularJS application to set targets for key performance metrics such as page load time, time to interactive, and first contentful paint. Monitor these metrics regularly and take proactive measures to ensure your application stays within the defined performance budget.

Caching and Optimization: Implement caching strategies for static assets, API responses, and server-side rendered content to reduce network latency and improve overall performance. Use techniques like browser caching, CDN caching, and server-side caching to optimize content delivery.

Continuous Performance Monitoring: Set up continuous integration and deployment pipelines with performance testing and monitoring integrated into your development workflow. Use tools like Lighthouse, WebPageTest, or custom performance monitoring scripts to automate performance testing and identify regressions early in the development lifecycle.

15. Assessment Schedule

Tentative schedule for the Assessment During 2023-2024 Even Semester

S. No.	Name of the Assessment	Start Date	End Date	Portion
1	Unit Test 1			Unit 1
2	IAT 1	12.02.2024	17.02.2024	Unit 1 & 2
3	Unit Test 2			Unit 3
4	IAT 2	01.04.2024	06.04.2024	Unit 3 & 4
5	Revision 1			Unit 5, 1 & 2
6	Revision 2			Unit 3 & 4
7	Model	20.04.2024	30.04.2024	All 5 Units

16. Text Books & References

TEXT BOOKS:

1. Somnath Musib, Spring Boot in Practice, Manning publication, June 2022 (https://www.manning.com/books/spring-boot-in-practice)
2. Alex Banks, Eve Porcello , "Learning React", May 2017, O'Reilly Media, Inc. ISBN: 9781491954621. (https://www.oreilly.com/library/view/learning-react/9781491954614/)
3. David Herron ,"Node.js Web Development - Fourth Edition",2018, Packt Publishing,ISBN: 9781788626859
4. Sukesh Marla, "A Journey to Angular Development Paperback ", BPB Publications.(https://in.bpbonline.com/products/a-journey-to-angular-development?_pos=1&_sid=0a0a0e9fb&_ss=r)
5. Yakov Fain Anton Moiseev, "Angular Development with TypeScript", 2nd Edition.(https://www.manning.com/books/angular-development-with-typescript-Second Edition).

Reference Books:

REFERENCES:

1. Sue Spielman, The Struts Framework 1: A Practical guide for Java Programmers||, 1st Edition. Elsevier 2002

WEB REFERENCES:

1. https://www.manning.com/books/spring-boot-in-practice
2. https://www.oreilly.com/library/view/learning- react/9781491954614
3. https://in.bpbonline.com/products/a-journey-to-angular-development?_pos=1&_sid=0a0a0e9fb&_ss=r
4. https://in.bpbonline.com/products/a-journey-to-angular-development?_pos=1&_sid=0a0a0e9fb&_ss=r
5. https://www.manning.com/books/angular-development-with-typescript-~~Second Edition~~

17. Mini Project Suggestions

To Do App

A app that keeps track of your to-do actions, you can add any number of tasks to it, search those tasks, click on the checkbox to complete the task, and filter using buttons like active tasks, completed tasks and all the tasks. This is another great project for you to add to your portfolio as a beginner React developer.

E Commerce

An e-commerce website built using React and Typescript, where we can filter clothes products using their customer preferred sizes as M, L or XL etc. We have a button called "Add to cart" below each product shown on the web page, once user selects any product, it will go to cart. At the end it can be used to checkout. These terms must be familiar to everyone now-a-days since it tries to mock popular e- commerce websites like Amazon, Flipkart and Myntra etc.

Category -2

1. **To-Do List App**: Create a simple to-do list app with features to add, mark as done, and delete tasks.
2. **Counter App**: Build a counter application that increments or decrements a value when buttons are clicked.

Category -3

1. **Calculator**: Develop a basic calculator that performs arithmetic operations on user inputs.
2. **Random Quote Generator**: Create an app that displays random quotes fetched from an API.

18. Mini Project Suggestions

Category – 4

3. Weather App: Build an app that fetches weather data based on user input location.
4. Image Gallery: Design a gallery where users can view and search for images.

Category – 5

8. Timer/Stopwatch: Build a timer or stopwatch application with start, pause, and reset functionalities.
9. Simple Blog: Create a basic blog platform where users can read and write posts.

[

Thank you