

Enzigma Software Pvt. Ltd.

Introduction

Angular commonly referred to as "Angular 2+" or "Angular v2 and above" is a TypeScript-based open-source front-end web application framework led by the Angular Team at Google and by a community of individuals and corporations. Angular is a complete rewrite from the same team that built AngularJS.

Angular 6 was released on May 4, 2018. This is a major release focused less on the underlying framework, and more on the toolchain and on making it easier to move quickly with Angular in the future, like: ng update, ng add, Angular Elements, Angular Material + CDK Components, Angular Material Starter Components, CLI Workspaces, Library Support, Tree Shakable Providers, Animations Performance Improvements, and RxJS v6.

Angular 7 was released on October 18, 2018. Updates regarding Application Performance, Angular Material & CDK, Virtual Scrolling, Improved Accessibility of Selects, now supports Content Projection using web standard for custom elements, and dependency updates regarding Typescript 3.1, RxJS 6.3, Node 10 (still supporting Node 8).

Index

1. Guidelines: 101 Overview	6
What can you expect out of this training module?	6
Which Naming Convention is to be followed and why ?	6
What are the best Practice and Everything needed?	6
Why Choose Version Control?	7
What about after completing this module?	7
2. Estimated Time of Training	8
3. Module 1 - Introduction to Angular	9
What is Angular?	9
Brief History of Angular	9
Prerequisite for Angular	9
Basic Understanding of CSS	9
Basic Understanding on Javascript	9
Introduction : Typescript	10
Building Blocks of Angular	11
What is Linting? Why does it matter to us?	12
More on TSLint	14
Exploring TSLint	15
Playing With TSLint	16
Footnotes	18
Assignments	18
4. Module 2 - Modules	19
Modules: What makes it so cool?	19
Modules: What does a Module look like?	21
Modules: What does import do?	22
Modules: Why Declarations & imports?	23
Modules: Why Providers & Bootstrap?	24
Modules: What is Root Module?	24
Creating new module in Angular Application...	25
Creating new Component in new module...	25
Summary	26
Footnotes	26

Assignment	26
5. Module 3 - Directives	28
Directives: What makes them so awesome?	28
Directives: What really is a Directive?	28
Directives: Types of Directives	28
Directives: Flavours of Directives	32
Directives: How to create Directives	33
Directives: How does Directive look like?	33
Summary	34
Footnotes	34
Assignment	35
6. Module 4 - Components	36
What makes the Component so cool?	36
Component Lifecycle	36
Components: Components in Components	36
Components: Component Interaction	36
Components: More on Components	37
Summary	38
Footnotes	38
Assignment	39
Testing Components	41
7. Module 5 - Pipes	42
Pipes: What makes them so awesome?	42
Pipes: What really is a Pipes?	42
Pipes: Journey from filters to Pipes	42
Syntax of Pipes and the Connection with I	42
Pipes: Types of Pipes	43
Diving into Built-in Pipes	45
Pipes: How to create Pipes	49
Pipes: How does pipes look like?	50
Using Multiple Pipes...	52
Summary	52
Footnotes	52
Assignment	53

8. Module 6 - Services	54
Services: What makes them so awesome?	54
Services: What really is a Services?	54
Services: Types of Services	54
Services: How to create Services	55
Services: How does Services look like?	56
Summary	57
Footnotes	57
Assignment	57
9. Module 7 - Routing	58
Routing: What does it take to do Routing ?	58
Routing: What really is a Routing?	58
Routing: Who does the real Routing?	58
Angular Router: What is it?	58
Routing: How to Create Routing	59
Routing : How does Router look like	60
Summary	60
Footnotes	61
Assignment	61
10. Module 8 - Testing In Angular	67
What is Test Driven Development(TDD) And Behavioral Driven Development(BDD)?	67
Testing Frameworks in JS	67
Why to test angular code?	67
Types of Testing	68
How to run test?	69
Testing: Jasmine	69
Testing: Fundamental Concepts	70
Dependency Injection (DI)	70
Testing: Karma	70
Testing: Test Entry File	71
Component Testing	71
Service Testing	72
Directive Testing	72
Pipe Testing	72

Test debugging	72
Footnotes	74
11. Introduction to forms in Angular	74
12. Mini Project	75
Mini Project: User Dashboard	75
Using Bootstrap 4 in Angular Project	76
Mini Project: Reference Guide	76
Mini Project: API Reference	77

1. Guidelines: 101 Overview

What can you expect out of this training

This guide helps you getting started with the fundamentals & basics of Angular, and steadily increases the pace by introducing the How, What and Why's, as to give In-Depth understanding of each topic. As it isn't possible to cover all the topics In-Depth suggest you all to do your fair bit of research to get even more in-depth understanding of each.

Which Naming Convention is to be followed and why ?

Naming conventions are fundamental as it helps to encourages consistency overall, and makes it more easily understandable & readable. The Naming Convention that we would be as follows...

- *camelCase* is used for variable & function names (*var lastName*)
 - *camelCase* is used for filename (modules,
- Variable names Always start with *letter* (*var firstName*)
- *Spaces* around operator (*var x = y + z*)
- Encourage Code Readability & usage of Indentation
- Global variables & Constants should be written in UPPERCASE
- Add *//comments* where need.
- Choose Meaningful names while naming file names & folders.
- Use Conventional (Angular) suffix for files-
 - *.component.ts* , *.directive.ts* , *.module.ts*, *.pipe.ts* or *.service.ts*
- For more Information refer to official [@Style Guide](#)

What are the best Practice and Everything needed?

Best Practises ensure along with timely delivery ensure quality end-product making the code more readable, easily understand & ensure better maintenance.

- Declarations on the top
 - Single point to look on-to.
 - Reduced possibility of unwanted re-declarations.
- Follow Naming Convention mentioned above
 - Helps to have consistency overall.
 - Improves Readability.

-
- Encourage the use of Version Control.
 - Helps in reverting back to changes in-case of accidental changes.
 - Implement the Folder Structure systematically.
 - Follow Modular approach.
 - Avoid having one single file having more than 600 lines of code.
 - Created Separate Folders/Files.
 - Encourage Loose Coupled Approach.
 - Follow throughout the whole application a Consistent Pattern.
 - File/Folders Naming.
 - Components/Modules/Directives/Services Naming.

Why Choose Version Control?

In this whole course module we would be heavily using the version control in order to store,manage our code, the platform which we would be using for this would be [@Gitlab](#). Gitlab is no more different than github but instead offers us more features.

Prerequisites:

Need to ensure that you have gitlab account created, after doing so.

Create a gitlab repo which has access to required members.

While creating a gitlab repo, ensure Naming Convention

- “ng-training-”(followed by assignment name)

Example: ng-training-assignment -1

Make sure all the changes have being committed to the repo.

What about after completing this module?

Your journey of learning at Enigma never stops, so do your bit of research try making some tiny app out of nothing, Dive Deeper into how stuff works in the background, help your colleagues solving problems & you become **ng-ninja**

Guide to do so...

- Refer to [@medium](#) to get crisp and clear understanding of topics
- Get the latest trend and In-Depth understanding [@Go In-Depth](#).
- Stuck somewhere? [@Stack-overflow](#) for Rescue!

2. Estimated Time of Training

Module	Module Timeframe	Assignment Timeframe
Module 1: Introduction to Angular	½ day	½ day
Module 2 : Modules	½ day	½ day
Module 3 : Directives	1 day	-
Module 4 : Components	1 day	1 day
Module 5 : Pipes	½ day	½ day
Module 6 : Services	1 day	
Module 7 : Routing	½ day	½ day
Module 8 : Testing in Angular	2 days	
Introduction to forms in Angular	1 day	
Mini Project + Test Methods	--	3 Days

Grand Total = 14 days

3. Module 1 - Introduction to Angular

What is Angular?

AngularJS is a JavaScript-based open-source front-end web application framework mainly maintained by Google and by a community of individuals and corporations to address many of the challenges encountered in developing single-page applications.

Why Choose Angular?

[@Reasons to choose Angular](#)

Brief History of Angular

The History of AngularJS. AngularJS was created, as a side project, in 2009 by two developers, Misko Hevery and Adam Abrons. The two originally envisioned their project, GetAngular, to be an end-to-end tool that allowed web designers to interact with both the frontend and the backend, and the rest is a history...

Prerequisite for Angular

Basic Understanding of HTML.

Resource to Learn HTML

- [W3Schools](#)
- [Code Academy HTML](#)
- [Team Treehouse HTML](#)

Basic Understanding of CSS

Resources to Learn CSS

- [W3Schools CSS](#)
- [Codecademy CSS](#)
- [Hackr.io CSS](#)

Basic Understanding on Javascript

Resource to Learn JavaScript

- [W3Schools Javascript](#)
- [Channel9 MSDN - JavaScript](#)
- [Javascript MDN](#)
- [Hackr.io JavaScript](#)
- <https://www.freecodecamp.org/learn/>

Introduction : Typescript

- **What: TypeScript**

“TypeScript is an open-source programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript, and adds optional static typing to the language. TypeScript is designed for development of large applications and trans compiles to JavaScript”

- **Why: TypeScript.**

- Relation to other JavaScript targeting languages. ...
- Optionally static typing and type inference. ...
- Enhanced IDE support. ...
- Strict null checks. ...
- Compilation. ...
- JavaScript interoperability. ...
- Converting from JavaScript to TypeScript

- **When to choose:: TypeScript**

- When the Codebase is Large
- When the Dev-Team is already Pretty good with static typed languages
- When Typescript can turn as replacement to Babel
- When you feel the need for speed
-

- For Resource/Assignment refer to the Document.

[Enzigma TypeScript Training](#)

<https://www.typescriptlang.org/docs/>

Building Blocks of Angular

NodeJs

- What: NodeJS
 - Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser.
- History: NodeJS
 - Node.js was originally written by Ryan Dahl in 2009, about thirteen years after the introduction of the first server-side JavaScript environment, Netscape's LiveWire Pro Web.
- Why: NodeJS
 - [Reason to choose NodeJs](#)

NPM

- What is NPM?
 - npm is a package manager for the JavaScript programming language. It is the default package manager for the JavaScript runtime environment Node.js.
- Learn: NPM
 - [Official Documentation: NPM](#)

NG-CLI

- What is NG-CLI?
 - Angular Command Line Utility which is used for multiple purpose for acts like Swiss Army Knife in terms of Angular for variety.
- Installation: NG-CLI
 - [Official Link: Angular CLI](#)

Installation/Configuration : Angular

- Installation Setup and Configuration of Angular
 - Downloading/Installing/Configuration: NodeJS
 - Downloading Node JS
 - [Official Download NodeJS](#)

-
- Steps to Install Node JS (Windows)
 - [Walk through to install NodeJS](#)
 - What is Meant by Single Page Application?
 - [Exploring Single Page Application: Angular](#)
 - What are the different Version of Angular? What makes each of them different from the rest?
 - [Journey from Angular to Angular 2 , 3, 4, 5, 6](#)
 - Folder Structure of Angular Project.
 - [Angular Step-by-Step](#)
 - Version in Angular
 - [Changes/Updates in Angular](#)

What is Linting? Why does it matter to us?

- **What is Linting?**

“Linting can be called as process of checking the source code for Programmatic/Stylistic errors before actually running the code, its way to verify the code quality”

Linting available for lots of programming language. (JavaScript, CSS, HTML..etc)

Linter are the programs which enabling linting and which go ahead and check the code to find, formatting discrepancy, non-adherence to coding standards & conventions

- **Why choose Linting?**

“Linting can be cumbersome task but the end product that we get after which we enable linting is great.

- Ensure Coding Standards & Conventions are followed.
- Improves Readability of the Program.
- Find (syntax) errors before execution.
- Helps in code review (pre-code review).
- Avoid errors ,bugs
- Enforces a Uniform Coding Style.

- **Linting, in Angular(Typescript)**

“Linting in angular is possible using ts-lint. TSLint is static analysis tool that checks your code for readability, maintainability and functionality errors.

TSLint all about some set of rules and generic best practises, TSLint in fact comes with its own set of rules, we custom them as well according to our project needs as well.

Installing TSLint in Angular

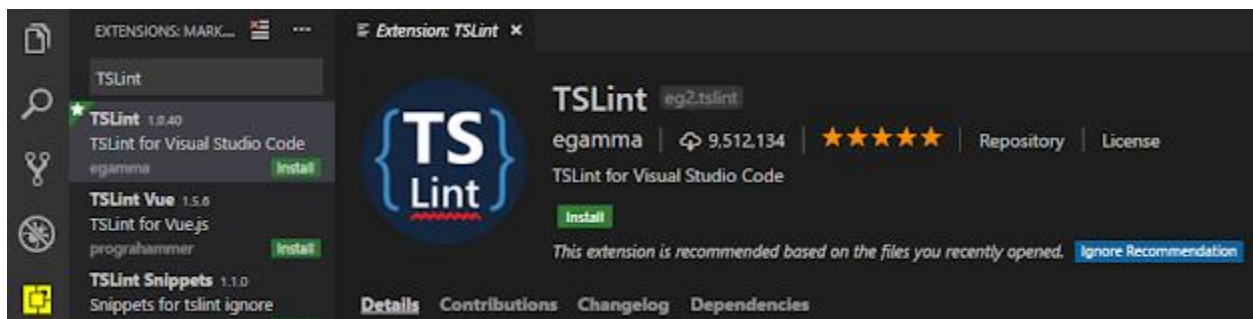
We can Install TSLint in Angular using the CLI or we can also setup and enable it using the VSCode Extension which is present.

Installing TSLint Extension (VSCode)

There are Couple of ways to install/setup TSLint on our machine, but on the purpose of this course we would be going ahead and installing TSLint Extension in VSCode.

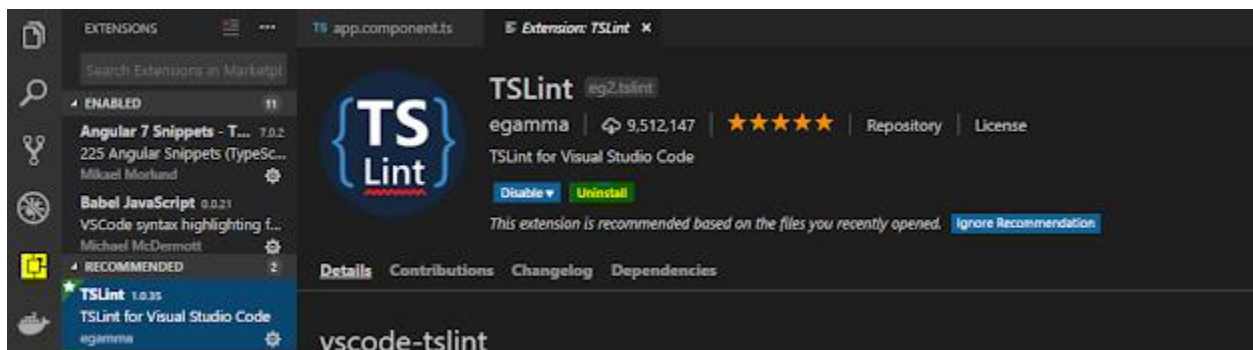
Steps to setup TSLint on VSCode.

1. Search TSLint under the extension section for VSCode & Click Install.



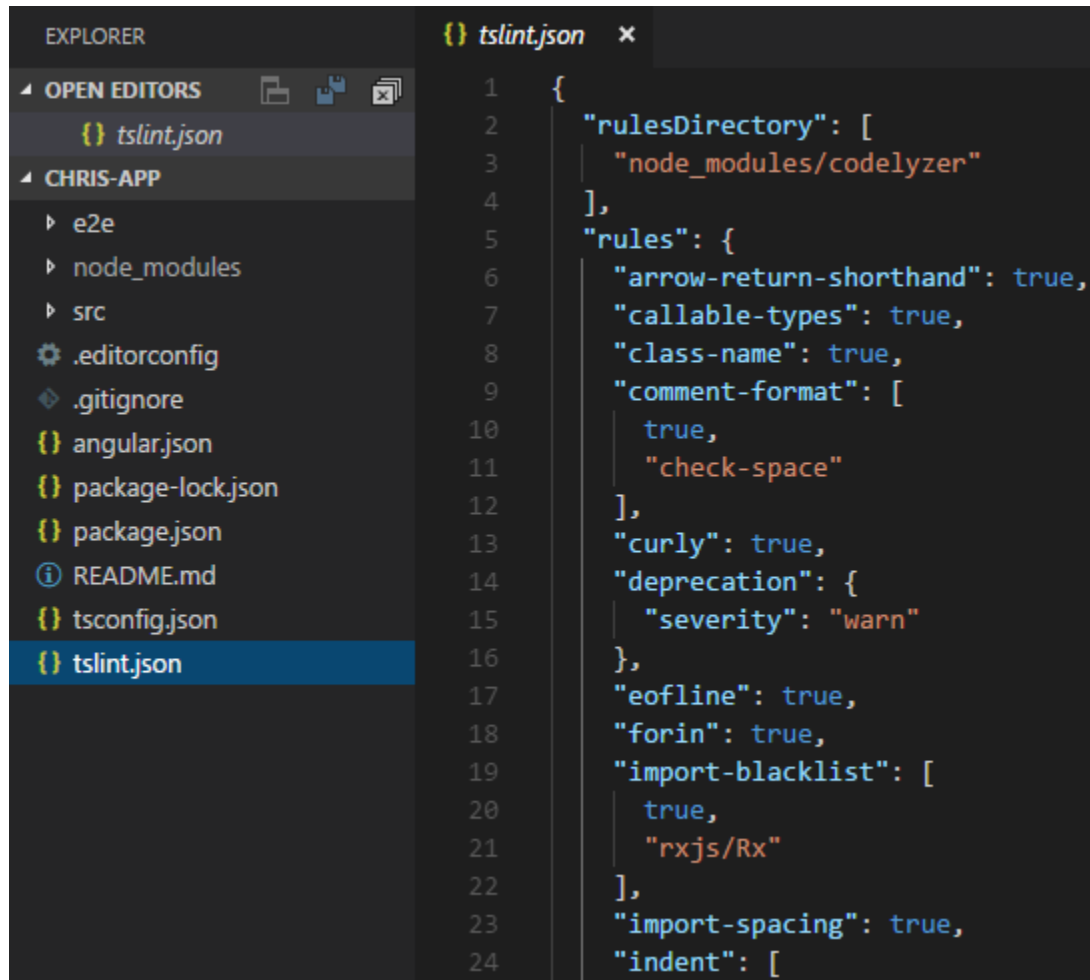
(After Installing the TSLint, go ahead and click on Reload/Restart VSCode)

2. To Verify if the TSLint is installed we can under the extension section and verify that.



More on TSLint

TSLint already comes with some set of rules, with its own set of best practises, In the case of Angular the default app that we have is (hero-app) comes preloaded with file called the *tslint.json*. We have tslint.json which is present at the root of the Angular project as well as the src folder of the angular project as well.



```
1 {
2   "rulesDirectory": [
3     "node_modules/codelyzer"
4   ],
5   "rules": {
6     "arrow-return-shorthand": true,
7     "callable-types": true,
8     "class-name": true,
9     "comment-format": [
10      true,
11      "check-space"
12    ],
13     "curly": true,
14     "deprecation": {
15       "severity": "warn"
16     },
17     "eofline": true,
18     "forin": true,
19     "import-blacklist": [
20       true,
21       "rxjs/Rx"
22     ],
23     "import-spacing": true,
24     "indent": [
```

(The Screenshot, is of the file tslint.json on the root of the Angular Project)

TSLint as well already can see is .json file which contains the standard best practises which are present which we can make changes accordingly to our project Specific requirements.

Exploring TSLint

TSLint can be used to touchbase on various aspects of coding in Angular, Majorly it focuses on Functionality, Maintainability, Styling (Formatting), Indentation. As TSLint is far more huge than the scope of this topics, i would just discuss a few TSLint Rules for more information, feel free to visit the [@OfficialDocumentation](#).

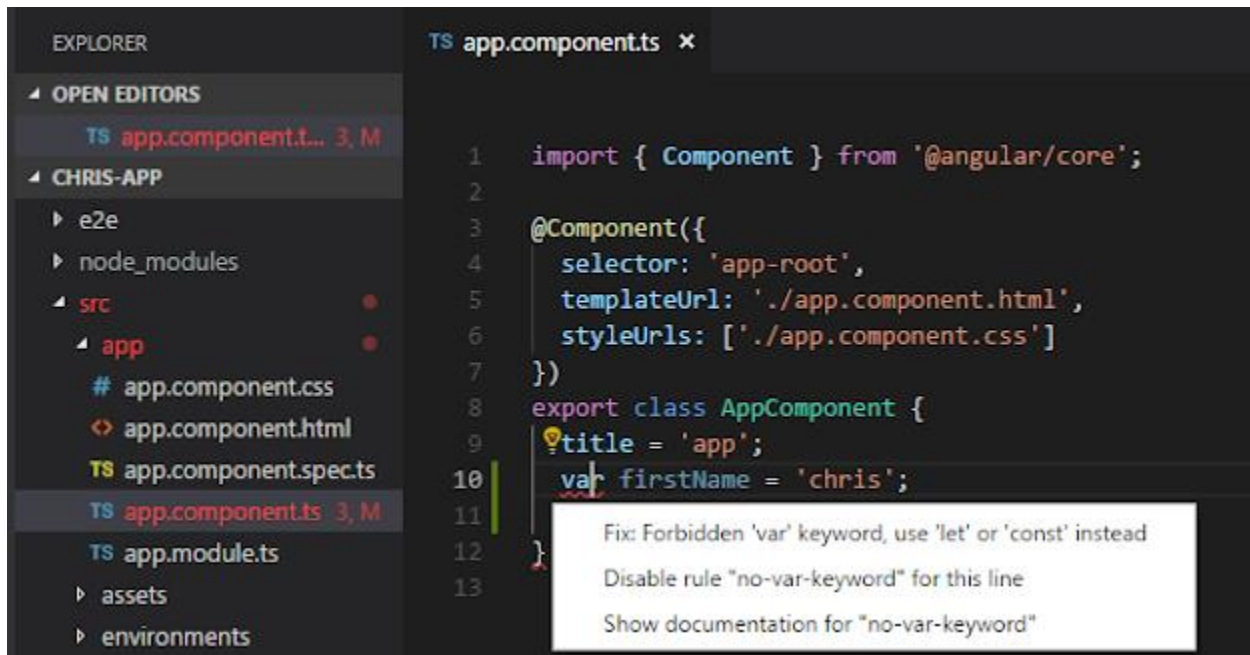
Table contains some rules that can be used, feel free to explore...

whitespaces	(true) Helps to make the code more readable & ensures a proper styling. Helps to white spaces around modules, operator, separator, typecast etc.
variable-name	Checks the variable names for various errors and implement camel-case or ban-keywords.
semicolon	Enforces the usage of semicolon at the end of every statement.
one-variable-per-declaration	Ensures that the variable is declared only once.
comment-format	Enforces formatting rules for single-line comments.
indent	Enforces indentation with tabs & spaces.
switch-default	Requires a default to be specified in every switch statements
no-unused-variable	Disallows unused imports, variables, functions...
no-null-keyword	Disallows use of null keyword literal
no-empty	Disallows use of empty blocks, blocks with nothing or just comments.
ban	With the help of ban, we can specify function which can be banned.

no-any	Disables the use of any data type.
--------	------------------------------------

Playing With TSLint

We can play around with TSLint by going ahead and mangling the code to check that TSLint is working in background, my not following the rules that are by default activated in TSLint.



In this case, in the root component i try to go ahead and declare a variable using variable, as per the TSLint it doesn't follow the best practise and hence it throws an error.

What makes TSLint more beautiful is the fact that TSLint, if we hover over the yellow bulb which appears right next to the error it gives us suggestions.

Fix: Forbidden 'var' keyword, use 'let' or 'const' instead
Disable rule "no-var-keyword" for this line
Show documentation for "no-var-keyword"

We can go ahead and fix the error automatically or suppress the error or refer to the Official document for more details.

Fuzzing with TSLint...

Adding the rule of never use Semicolon and whole application blows-up!!..

```
"semicolon": [  
  true,  
  "never"  
],
```

Adding this to TSLint Application and End-result it blows up the whole application having every statement ending with semicolon.

```
import { BrowserModule } from '@angular/platform-browser';  
import { NgModule } from '@angular/core';  
  
import { AppComponent } from './app.component';  
  
@NgModule({  
  declarations: [  

```

Specify Comment true with check space and lower case.

```
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
})  
export class AppComponent {  
  title: string;  
  //Comment  
}
```

[tslint] comment must start with a space (comment-format)
[tslint] comment must start with lowercase letter (comment-format)

This throws an error as we have added.

```
"comment-format": [  
  true,  
  "check-space",  
  "check-lowercase"  
],
```

Footnotes

- What do we mean by [Single Page Application](#)?
- How to use [ng-cli](#)?
- What is [package.json](#)?
- What is [package-lock.json](#)?
- What are some important [folders](#) while considering Angular?
- What is so special about the [node modules](#) folder?
- Understanding the working/use-case of the folders present
 - node_modules
 - src
 - dist
- Which port does Angular App run on?

Assignments

Assignment 1:

- Create a Default Angular application named as “Module-1”
- Make the changes accordingly in have the below output

Introduction to Angular

Assignment 2:

- In the same application go ahead and install the npm package (Locally) | Package Name: hello-world-npm
- In the same application go ahead and install the npm package (Globally) | Package Name: typescript

4. Module 2 - Modules

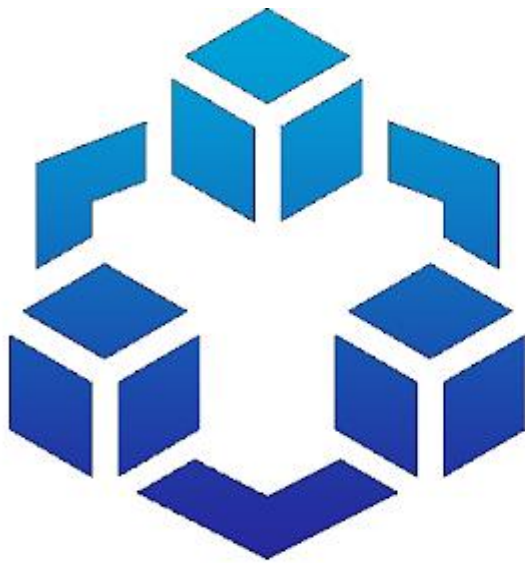
Modules: What makes it so cool?

Most of today's, Modern day application is being made of smaller manageable units which combine together to form an Application. This type of an approach is termed as Modular Approach. Modular Approach is great as it helps to make the code more maintainable & have a strong degree of isolation between individual units.

Let's first understand in general what is a *Modular Design* approach, *Modular design*, is a process in which the whole application is divided into manageable units. Modular design helps to separate the Application into smaller units & helps to reduce redundant code overall. As we achieved a degree of isolation, it helps multiple teams to work on a given application without disrupting the whole application which helps them to get the work done faster and more efficiently. It goes ahead and encourages the code reusability as well. Concept of Module isn't a new concept invented by Angular, Javascript also has the feature of [Modules](#) which serves the similar purpose.

Angular implements Modular Approach using *Modules*. An Angular Application at a higher level is made up of Multiple Modules all combined together. Each of Modules is used to serve a specific purpose and contains a group of Components, Services, Directives & Pipes, which combined together often share a common purpose at higher level. Modules encourage code-reusability with the help of exporting and importing parts of the modules into other modules.

A Modern-Day Fully Feature-loaded application that is present contains multiple features integrated together, in order to segregate and make them work perfectly, we encourage the use of creating multiple modules, each focusing on a given feature. An Angular application contains by default one root module called as AppModule (app.module.ts), after which we can have Feature Modules as well, which focus on the different features inside of the application, We can also have other modules which focus on the Frequently used-code/Reusable Module.



"A whole application is made of smaller manageable & isolated units called Modules. Modules Intergrate Together to form a complete Angular Application."

SideNote:

Modules is a logical container used to package together (Components, Directives, Services, Pipes) inside of it, which can be consumed & reused by the other Modules with the help of importing, Similarly how we have packages in Java or Namespace in the case of C#...

Modules: What does a Module look like?

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { WarningAlertComponent } from './warning-alert/warning-alert.component';
import { SuccessAlertComponent } from './success-alert/success-alert.component';
import { ServerComponent } from './server/server.component';
import { EnigmaComponent } from './enigma/enigma.component';

@NgModule({
  declarations: [
    AppComponent,
    WarningAlertComponent,
    SuccessAlertComponent,
    ServerComponent,
    EnigmaComponent
  ],
  imports: [
    BrowserModule, FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

The Screenshot above shows AppModule (app.module.ts) which is the *root module* in the angular application. Starting with *import* which gives the relative path to the Modules, Components, Services.. etc along with it *declarations* which specify the components, pipes to be used followed by *imports* which strictly define module which are already imported. *providers* contains Service which is used. *bootstrap* contains the name of the component present in index.html & @NgModule() with all these metadata/properties makes the class mentioned below as Module Class

Modules: What does import do?

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { WarningAlertComponent } from './warning-alert/warning-alert.component';
import { SuccessAlertComponent } from './success-alert/success-alert.component';
import { ServerComponent } from './server/server.component';
import { EnigmaComponent } from './enigma/enigma.component';
```

import: "import component by specifying the relative path to resource"

Import as the name suggest is use to go ahead and import the Components, Modules, Pipes which are present by specifying the relative path of the components,Modules,Pipes...

```
@NgModule({
  declarations: [
    AppComponent,
    WarningAlertComponent,
    SuccessAlertComponent,
    ServerComponent,
    EnigmaComponent
  ],
  imports: [
    BrowserModule,FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
```

@NgModule() this basically a Decorator which has list of metadata/properties which contains information of the modules as well @NgModule helps to define that the class defined below the NgModule Decorator is Module class.

Modules: Why Declarations & imports?

What do we declare in declarations ?

```
declarations: [  
  AppComponent,  
  WarningAlertComponent,  
  SuccessAlertComponent,  
  ServerComponent,  
  EnigmaComponent  
],
```

Declaration: “declares the list of components,directives, pipes”

Keeping it fair and simple, Declarations as the name suggest is used to go ahead and perform declarations by specifying the name of the components that we have already imported in import section at the start. Declaration is an array which contains the names of the Components which are present.

What do we really import in imports?

```
imports: [  
  BrowserModule,FormsModule  
],
```

imports: “specify list of components that already imported in import section”

In the import section of the Module we define the path to the module along with it we specify the name of the Module, Imports in this case is useful to import the names of the module that would be imported in the current module.

Modules: Why Providers & Bootstrap?

What does **Bootstrap** do?

```
bootstrap: [AppComponent]
```

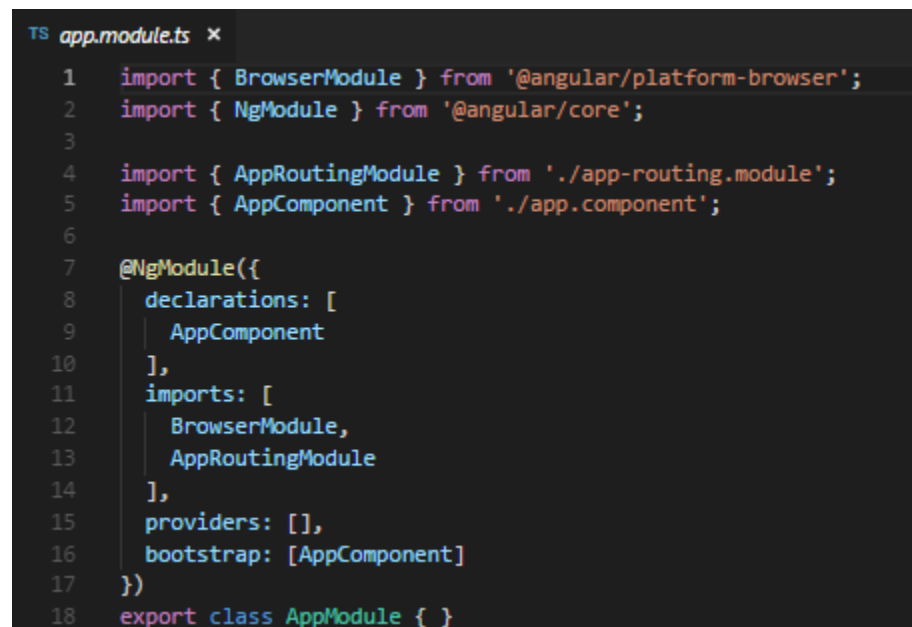
In this case, bootstrap is an array which can contain the names of the component that we want to render, and which needs to be inserted inside of the index.html file.

What does **Providers** provider?

```
providers: [],
```

In this case, Providers are used to inject the services required by components, directives, pipes in our module. It's again an array which contains the Service names. By Default when we create a Service using the NG-CLI it doesn't automatically enter the name of the Service in the providers array. Service are more inclined to be Application wide, but still we can limit the usage of service to particular component or module.

Modules: What is Root Module?



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

Above is a screenshot of AppModule (app.module.ts) which is root Module for the angular Application, alongside of the root module we can also have multiple features modules which

focus on a particular feature. As discussed previously, it imports some well know Modules BrowserModule which is responsible if we want to render application on the browser, which contains loads browser specific functions which involves DOM Manipulation feature as well.

Creating new module in Angular Application...

We can have a Single Module do all the task and heavy-lifting, but it's better to segregate the application into manageable units by creating modules or features modules, as we already know feature modules focus on a primary goal or an area of interest.

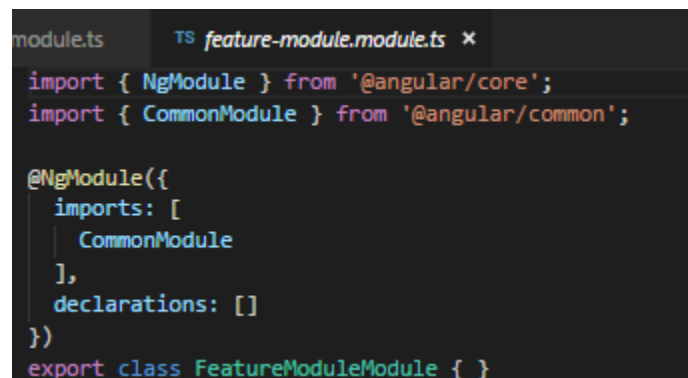
We can create a module in an Angular Application by navigating to the root of the project by executing the command using the ng-cli tool.

Syntax: `ng generate module module_name`

```
PS E:\Angular\Modules> ng generate module feature-module
CREATE src/app/feature-module/feature-module.module.ts (197 bytes)
```

This command goes ahead and creates a folder named after the module specified in the above command.

Inside of the folder it creates a module.



```
module.ts TS feature-module.module.ts x
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

@NgModule({
  imports: [
    CommonModule
  ],
  declarations: []
})
export class FeatureModuleModule { }
```

By default the Module that is created contains no components or any other file besides the Module.ts file.

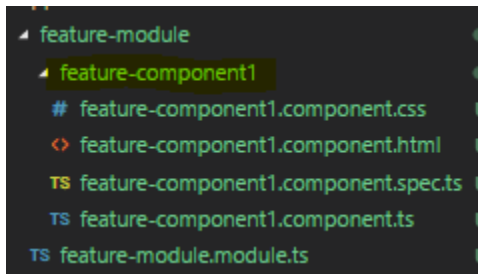
Creating a new Component in a new module...

As we already know that a Module will just contain a Module.ts file we want to create a Component inside of the module we need to make minor changes in the command while creating the component inside of the module.

Syntax: ng generate component modulename/componentname

```
PS E:\Angular\Modules> ng generate component feature-module/feature-component1
CREATE src/app/feature-module/feature-component1/feature-component1.component.html (37 bytes)
CREATE src/app/feature-module/feature-component1/feature-component1.component.spec.ts (706 bytes)
CREATE src/app/feature-module/feature-component1/feature-component1.component.ts (316 bytes)
CREATE src/app/feature-module/feature-component1/feature-component1.component.css (0 bytes)
UPDATE src/app/feature-module/feature-module.module.ts (319 bytes)
```

In this case, we want to create component in the module we need to use the same syntax, followed by the modulename forward-slash (/) component name.



```
feature-module
├── feature-component1
│   ├── feature-component1.component.css
│   ├── feature-component1.component.html
│   ├── feature-component1.component.spec.ts
│   └── feature-component1.component.ts
└── feature-module.module.ts
```

Now if we observe that it goes ahead and creates a folder named after the module created (feature-module) inside of which we have the (feature-module.module.ts) file. Followed by which we have sub-folder under the module folder which will contain the Component specific files.

Summary

“An Angular Application is made up of Modules, Modules helps to package (Components, Directives , Pipes) together have specific purpose that helps them bind together, Modules help achieve code reusability by exporting the resources and later on importing it. An Application will have one root Module (AppModule) and other Feature Modules. @NgModule() decorator is used to defined a Module Class, we can create Module using ng-cli using the command:

ng generate module modulename.”

Footnotes

- [@More Information of Modules](#)
- [Multiple Modules Structure](#)
- [Understanding Ng-Modules & Scopes](#)

Assignment

Assignment 1:

- Create and Angular Application Named Modules
- Create a Module named login
- Import the login Module in the AppModule.

Assignment 2:

- Create two more modules under the same application.
 - featuremodule1
 - featuremodule2
- Examine the Folder Structure & Import both the modules in the main AppModule

5. Module 3 - Directives

Directives: What makes them so awesome?

Boring to play along with the same old HTML elements `<p>`, `<div>` ``, `<blah>` `<blah>` `<blah>`... But wait does `<blah>` `<blah>` make sense ,if i put it in the html file! (Noo)

Directives to the rescue, Directives help to replace the boring standard html attributes/elements and add your own custom magic to it. Besides all the goodness directives come pre-loaded. What makes Angular even more cool, is the fact that Angular also provides its own standard directives , but it doesn't just stop there we can have our own directives if needed to make something special out of nowhere.Let's unfold the mystery for Directives.

Directives: What really is a Directive?

Directives, are just a DOM element besides the standard HTML elements,which do get replaced with a custom behaviour which either has standard described by Angular (i.e. *Standard Directives*) or we can define our own custom behaviour (i.e. *Custom Directives*) according to which it gets rendered.Directives are multi-purpose classes that are meant to interact with the visual part of your application, namely what the user sees in front of him.

Directives: Types of Directives

Directives: What are the types of Directives?

Directives, are great what makes it even more fruitful is that Directives comes in two types...

- Built-In Directives
- Custom Directives

Built-in Directives

Built-in Directives, there are a tons of inbuilt Directives provided by Angular that save your efforts, we would just discuss a few.Inbuilt Directives, as the name suggest are the directives which are provided by Angular as a standard which we can consume, without to worry about the reinventing the wheel.

Angular has provided tons of Directives, some of which are listed below.

-
- *ngif
 - *ngfor
 - ngSwitch
 - ngClass
 - ngStyle

Creating a Custom Directive

Magic!, time to add our own custom logic now, with the help of Custom Directives we can go ahead and create our own Directives in angular, which means <blah> <blah> which was mentioned before will make sense now.

Really? How can we make <blah> as element display or do something, with the help of Custom Directives we can go ahead and create custom directive and add it in the html so accordingly it will render it.

We need to make sure while creating the Directive the name that we assign to the directive should not proceed with ng, as its already applicable for Built-in Directive and there shouldn't be any conflicts.

Purpose of the training we would go ahead and create a directive which would have highlight the text around it (set the background to yellow)

Simple and straightforward, Let's get started we would name it as "my-highlighter"

We would go and run "ng generate directive my-highlighter"

(This is Hassle free of creating Directive in Angular, so that we don't miss on the syntax)

```
PS E:\ng-assignments\Module2\Directives> ng generate directive my-highlighter
CREATE src/app/my-highlighter.directive.spec.ts (253 bytes)
CREATE src/app/my-highlighter.directive.ts (155 bytes)
UPDATE src/app/app.module.ts (490 bytes)
```

By running the command, we would create the basic Directive and also adds the details in the Module as well.

More on Custom Directive

Now, since we have the Directive, we need to go ahead and add some magic to it. Now in order to make changes, we will be needing access to the Element, in order to do so we have ElementRef which we need to import first.

```
import { Directive, ElementRef } from '@angular/core';

@Directive({
  selector: '[appMyHighlighter]'
})
export class MyHighlighterDirective {

  constructor(el: ElementRef) {
    el.nativeElement.style.background='yellow';
  }
}
```

In this case, after Importing Element Ref, we create reference of it and then using the style.background property we would now be able to set the value to yellow. As we have created the directive automatically the details of the Directive are added automatically in the AppModule, when we create it manually in that case we need to go ahead and specify it manually in the AppModule so that it can be usable in the components present inside of the Modules. We cannot simply add a directive as a tag, how we can for a HTML tag or a component, If we try to do so, it will give an error, because the whole purpose of a directive is that it goes ahead and is responsible to either enhance or improve the behaviour of the existing element.

```
<appMyHighlighter></appMyHighlighter>
```

More on Custom Directive

Now, let's observe the entries that are created in AppModule in this case, it goes ahead and creates the entry in the respective places import first and later on adding the details in the declarations array.

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { MyHighlighterDirective } from './my-highlighter.directive';

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

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

@Component({
  selector: 'app-root',
  template: `
    <h1 appMyHighlighter>Testing My-Highligther</h1>
  `,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Directives';
}
```

Now if we observe we can see that directive is used as attribute inside of h1 tag.

Output:

Testing My-Highligther

Here's how the final magic looks like... Great! Vibrant!

Directives: Flavours of Directives

Directives: What are the flavours of Directives?

Directives, as we already knew are of two types, the own which comes pre-package with Angular and the Custom Directives as well. Directives come in three flavour (Attribute Directives / Structural Directives / Components)

Attribute Directives

Attribute Directives, change the behaviour of an element, component or another directive thus, cannot add/remove elements in DOM.

There are couple of Attribute Directives

- `ngClass` | `ngStyle` | `ng-init` | `ng-app`

*Structural Directives

Structural Directives, are responsible to change the DOM layout, which involves adding/removing & manipulating the elements. They start with a *

- `*ngIf` | `*ngFor` | `*ngSwitch`

Components

They can be called as directives with a template. Directives will be more focussed in terms of behavioural aspect, Components rather more focused on the view & a template align to it as well as User Interface

Directives: How to create Directives

Directives: Ways to Create Directives ?

Directives in Angular can be created using the two ways...

- Automatic
- Manually

Automatic Approach

Automatic approach is simple and straightforward in which need to make use of Angular Command-Line tool (ng-cli). By using Automatic Approach it automatically creates the Directive Files and make sure footprints are added.

```
PS C:\Users\chris.rego\Desktop\Angular\testapp> ng generate directive testdirective
CREATE src/app/testdirective.directive.spec.ts (252 bytes)
CREATE src/app/testdirective.directive.ts (155 bytes)
UPDATE src/app/app.module.ts (505 bytes)
```

(In this case, it goes ahead and creates directive file inside of the AppModule)

Manual Approach

Manual approach is when we manually had to go ahead and create the required files along with the Manually going ahead and entering the details where needed.

Directives: How does Directive look like?

Deeper Look: Directive

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

@Directive({
  selector: '[appTestdirective]'
})
export class TestdirectiveDirective {

  constructor() { }

}
```

(Screenshot: This image is Directive file create above using ng-cli tool)

1. In this case, we go ahead and import the Directive Decorator from the @angular/core library

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

2. @Directive (Decorator) is used to define the typescript class is a Directive.

```
@Directive({  
  selector: '[appTestdirective]'  
})
```

3. At the end of the day, it's a class hence we define a class to define our Custom logic.

```
export class TestdirectiveDirective {  
  
  constructor() { }  
  
}
```

Summary

“Directives are used to extended HTML elements, They can be Custom or Built-in which either change the behaviour element (Attribute) or add/remove/modify DOM (*Structural), as well Components. Built-in Directives Usually preceding with ng,”

Footnotes

- [@Official Guide to Angular Directives](#)
- [Custom Directives](#)
- [List of Custom Directives](#)
- [Angular Directive Guide](#)

Assignment

Assignment 1:

- Create application name Directive
- Create a Directive Automatic approach using ng-cli
 - Name: Red-Marker
- Create a Directive manually without the use of ng-cli
 - Name: Green-Marker

Output:

Red Marker

Green Marker

Assignment 2:

- In the same application above go ahead and create Directive.
 - Name: Hover Focus

Hover Focus Directive will be used to in the case, when the mouse hovers over element the element background-color should change to blue, and once it hovers out of the specified element for the Directive, background-color should change to grey.

Hint: Use @HostBinding & @HostListener

Enter the text

This is random value

(Mouse out - at the left) - Background Color: White, Color: black

(Mouse over - at the right) - Background Color: Blue, Color: white

In this case, we need to have input text which will be using this Directive.

Write Unit-Test for the Above Implemented Directive.

6. Module 4 - Components

What makes the Component so cool?

What is [Component](#)?

“A component controls a patch of screen called a view. For example, individual components define and control each of the following views from the Tutorial”

Components can also be said to be basic building blocks in terms of Angular Application which is present.

More on Component

“Let me ask you, can i put a custom tag in html, and would that work?”

Let's try to put enigma in our default home page (index.html)

Do we see any changes in it. Nope!

Here's why?

“Browser is a great piece of software which reads through the whole document and renders the output based up tags used,Default tags which are present have a semantic meaning to it and hence the browser understanding these HTML tags and accordingly the browser renders the DOM and page gets displayed”

Lets Specify the a custom tag named as <enigma> save it into our html file and after doing so we would be.. Argh!! Arghh! disaster

Now how do we make the browser understand our custom tags and make the browser render something that we wish based upon using the custom tag.Components we can create custom tag which can use in an html page.

Component Lifecycle (Lifecycle Hooks):

Refer => <https://angular.io/guide/lifecycle-hooks#lifecycle-hooks>

Components: Components in Components

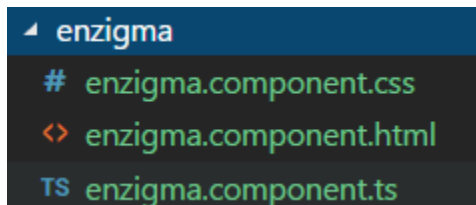
- Understanding the components that make a Components

“Components are termed as the basic building blocks in terms of angular application”

Component in Angular boils down to a simple folder which contains certain set of files, which together go ahead form a single component. But are Component just folders which contains files? Component can also be a file which contains all the entities that make the view viewable as inline rather than having separate files (.html , .css) . Let's solve the mystery behind it, One Application can have multiple components which are bind together form a Module, and then as we already know that modules combined together form an Application. Components encourage code reusability along with it also makes us go ahead creates certain degree of isolation. Components are also of types Directives.

Components contains the below components/files when created using ng-cli

- Html file (View)
- Css file (Stylesheet)
- Typescript (Class - @Component)
- Typescript/Jasmine test file



Components: Component Interaction

Reference:

<https://angular.io/guide/component-interaction>

Sharing data between child and parent directives and components:

<https://angular.io/guide/component-interaction>

Components: More on Components

“Understanding the components that make a Components”

One Component can also be used to go ahead and reuse the feature of another component. Component can intercommunicate with other components which are present. Related service binded together along with services, etc form a Module

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

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

  constructor() { }

  ngOnInit() {
  }

}
```

@Component Decorator specifies that this class is Component.

Summary

“Components are types of Directives, which are more inclined towards a view & UI which needs to be rendered, Component is composed of a view(HTML Template with CSS Styling) & Typescript class containing the data & logic to render the details accordingly. Identified with the help Decorator @Component()”

Footnotes

- How to create a component?
- What is meant by nesting a component?
- What does .ts file mean? What does it contain
- What is use of .html file and .css file
- Can we override the default component?
- Which is the Root Component in Angular 6?

-
- Understanding the working/use-case of the files present in a component
 - How to pass data from child component to parent component and vice versa?
 - .html
 - .css
 - .ts
 - .spec.ts

Assignment

Assignment 1:

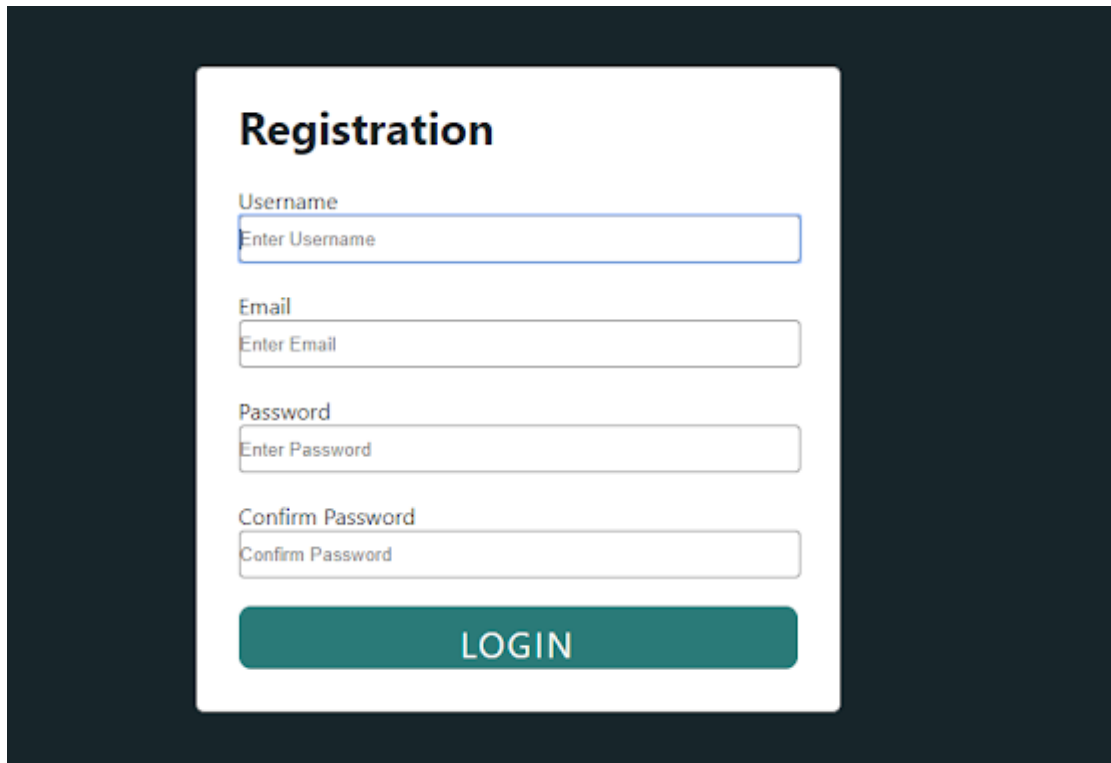
- Create a new Angular Application using ng-cli.
 - Name: Components
 -
- Create three components in Application
 - Name: FirstComponent
 - Name: SecondComponent

Go ahead and use all the components (AppComponent,FirstComponent,SecondComponent) on one screen.

Assignment 2:

- Create a new Angular Application using ng-cli
 - Name: Components
- Create a new module
 - Name: Feature Module
- Inside of the Feature Module, create a two new Component
 - Name(Component): login
 - Name(Component): register
- Now go ahead and load the RegisterComponent in the AppModule

Output: (Registration Page)

A registration form titled "Registration" is displayed on a dark blue background. The form is a white card with rounded corners. It contains four input fields: "Username" with placeholder text "Enter Username", "Email" with placeholder text "Enter Email", "Password" with placeholder text "Enter Password", and "Confirm Password" with placeholder text "Confirm Password". Below the input fields is a teal button with the text "LOGIN" in white capital letters.

Registration

Username
Enter Username

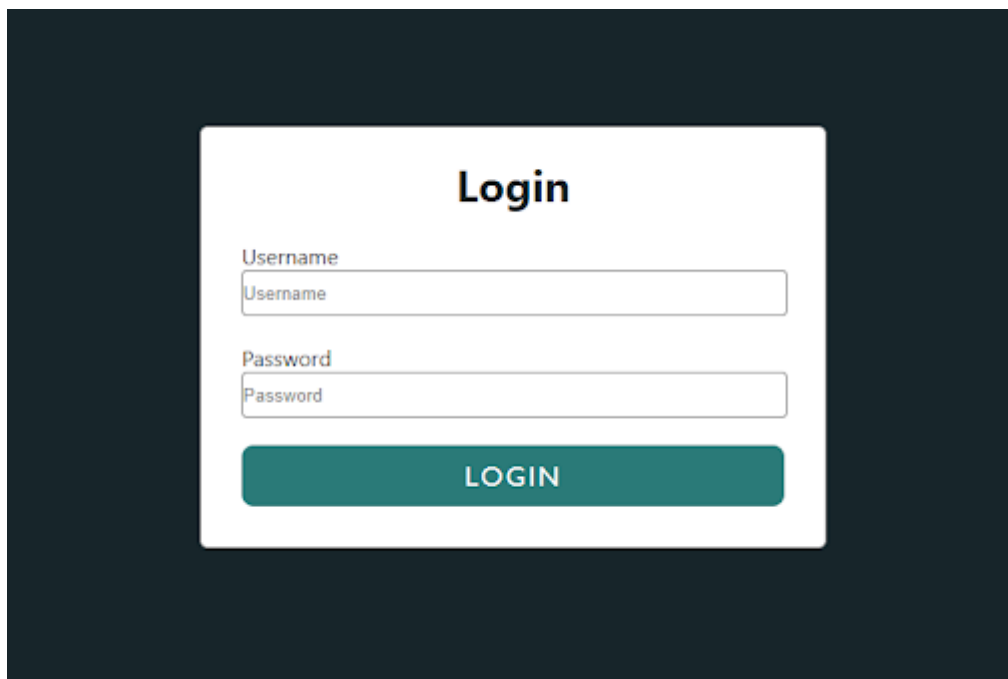
Email
Enter Email

Password
Enter Password

Confirm Password
Confirm Password

LOGIN

Output: (Login Page)

A login form titled "Login" is displayed on a dark blue background. The form is a white card with rounded corners. It contains two input fields: "Username" with placeholder text "Username" and "Password" with placeholder text "Password". Below the input fields is a teal button with the text "LOGIN" in white capital letters.

Login

Username
Username

Password
Password

LOGIN

Testing Components

Ever wondered, why running the command `ng generate component demo component` goes ahead and generates a component in Angular. If you watch closely it goes ahead and creates 4 files (Typescript Component Class, HTML, CSS and `.spec.ts` file)....

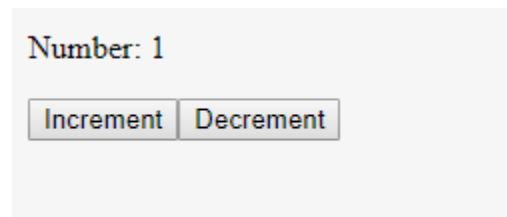
We have discussed already on what is `.ts` class file and the HTML and CSS files, But the `.spec.ts` is special one out of all...

This is a proof of how angular goes ahead and enforces Testability right from the ground level, right from the very initial stage of creating a single component, it makes sure that the component is unit tested.

Angular follows the Test Drive approach, we have a separate topic below which is dedicated to the overall details of the Testing in Angular.

Let's go ahead and test a Basic Component to understand the beauty of Angular Testing... Let's take the initial steps by creating a component named as counter.

Our final output would look like this.



In this case, we have to have a simple Application which would increment & decrement the counter after clicking the Increment & Decrement button which respectively calls the `increment()` & `decrement()` function mentioned.

Make sure you write unit test for the application.

7. Module 5 - Pipes

Pipes: What makes them so awesome?

There are times when we enter a given data and the output that we need to get out of that input that we have provided, needs to be fancy, stylish or may be something out of this world, Basically we want to format a value or variable in different format. Let say have my account balance to be prefixed with a Dollar symbol, how easy it is in Angular?. I just have to convert my account balance into a string and then append a '\$' dollar sign to it and then display the string. What if i told you all of this can be done with just a single line in Angular? But how can i do that ?

Pipes is the answer, pipes will help us go ahead and format the data that we have while it gets rendered on the view without writing any new things in your model. Pipes besides all the goodness also comes with loads of other features which make it a pretty good deal.

Pipes: What really is a Pipes?

Pipes, take a value or values and then returns a value based upon some operation which is performed. Pipes are ways to transform and format the data. Pipes is a way to display-value transformation right into your HTML. Pipes are operators that are used to format data in Angular.

Pipes: Journey from filters to Pipes

Pipes, where previously called filters in AngularJS 1.x Era.

But with the new era of Angular(ditching the JS) from Angular 2 onwards, filters with more advancements are now called as Pipes. When you use a pipe, you don't have to change your data model nor make a copy of it to make it "fit" what it should look like on the screen.

Syntax of Pipes and the Connection with |

Pipes are created with the help of OR symbol (|) surround by the data to be transformed and the name of pipe. The basic syntax of pipe looks like this

{{ price | currency : 'USD' }} . The | Symbol which is present helps us to go ahead and format the data at the left according to the pipe which is specified at the right and then go ahead and render the output accordingly.

Pipes: Types of Pipes

Pipes: What are the types of Directives?

Pipes, provide loads of benefits and they come in two types.

- **Built-In Pipes**
- **Custom Pipes**

Built-in Pipes

Built-in Pipes, there are couple of built-in pipes provided by Angular that save your efforts, we would just discuss only a few. Built-in pipes, as the name suggest are stock pipes provided by Angular and provide a unique way of transforming the input data. They can be used for basics transformation of data which involves making the string as uppercase, lowercase or formatting the date as well as displays metrics or percentile values, along with all of that it also contains the ways to display currency of different countries.

Angular has provided couple of built-in pipes , some of which are listed below. More Information of the Built-in Pipes. Refer to the [@Official link](#)

- uppercase
- lowercase
- titlecase
- date
- currency
- percent
- slice
- decimal (number)
- Jsonpipe
- async

uppercase

It goes ahead and transform the data and makes it uppercase.

Example: {{ 'chris' | uppercase }}

Output: CHRIS

lowercase

It goes ahead and transform the data and makes it lowercase.

Example: {{ 'REGO' | lowercase }}

Output: rego

titlecase

It will transform the data, so as the first letter of each word would become uppercase.

Example: {{ 'enzigma software pvt ltd' | titlecase }}

Output: Enzigma Software Pvt. Ltd.

currency

It goes ahead and transform the data & to format it into given currency value.

Example: {{ 65000 | currency: 'INR' }}

Output: ₹65,000.00

percent

It goes ahead and transform the data & evaluates into percent format.

Example: {{ 0.4 | percent }}

Output: 40%

Diving into Built-in Pipes

Lets better understand Built-in Pipes by implementing it with the help of an example...

```
<h2> {{ "customer details" | titlecase }}</h2>
<table border="1px">
  <tr>
    <td>Customer Name:</td>
    <td>{{ customerName | uppercase }}</td>
  </tr>
  <tr>
    <td>Customer ID:</td>
    <td>{{ customerId | lowercase }}</td>
  </tr>
  <tr>
    <td>Account Balance:</td>
    <td>{{ accountBalance | currency: "USD" }}</td>
  </tr>
  <tr>
    <td>Account Date:</td>
    <td>{{ accountDate | date }}</td>
  </tr>
  <tr>
    <td>Sybil Score:</td>
    <td>{{ sybilScore | percent }}</td>
  </tr>
</table>
```

In this case , we have declared the view with the built-in pipes ranging from uppercase, lowercase, currency, date and with it also the sybil score of the Customer. Now the data which is present in the html is plain unformatted without any changes, but along with | operator and the respective pipes which are specified it goes ahead and transforms the data into different type itself.

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

@Component({
  selector: 'balance-component',
  templateUrl: './balance.component.html',
  styleUrls: ['./balance.component.css']
})
export class BalanceComponent {
  customerName = 'chris rego';
  customerID = 'AED-123-EG2';
  accountBalance = 123.456;
  accountDate = new Date(2012,0,1);
  sybilScore = 0.4;
}
```

Model that we have along contains data which is in raw/unformatted format, the final output that we receive is render based upon the transformation which is done with the help of pipes which we have used for each data.

Customer Details

Customer Name:	CHRIS REGO
Customer ID:	aed-123-eg2
Account Balance:	\$123.46
Account Date:	Jan 1, 2012
Sybil Score:	40%

Final output which is present is render based upon the pipes that we have used,

Custom Pipes

Built-In Pipes already provide us most of the basic transformation which we need, but along with it if we have the power to write our own pipes. Custom pipes helps us do that, in this case we can define custom pipes with the help of custom logic and transform the data.

There are steps in-order to define, register & use a custom pipe in angular.

Define/Creation Phase.

Primarily, we need first create pipe to do that, we need to create a file and name it as 'nameofpipe'. pipe.ts or simply run ng cli command (ng generate pipe balance)

Example: `ts balance.pipe.ts`

This is how custom Pipe looks like

```

import { Pipe,PipeTransform } from "@angular/core";
import { BalanceComponent } from './balance.component';

@Pipe({
  name: 'professionPrefix'
})

export class BalanceTitlePipe implements PipeTransform{

  balancetitle = new BalanceComponent();

  transform(value: string):string {

    if(this.balancetitle.profession=== 'doctor') {
      return "Dr. ";
    }
    else if(value.toLowerCase()=='engineer'){
      return "Er. ";
    }
    throw new Error("Method not implemented.");
  }

}

```

Register Phase:

We need to make sure that the details of the pipe need to do added in the module as well. If in case if we forget to add the details in the module it will go ahead and throw an error stating that its not able to find the Pipe.

```

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

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

import { BalanceTitlePipe } from './balance/balance.pipe';
import { CustomerdetailPipe } from './balance/customerdetail.pipe';

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

```

Usage Phase:

In this case, we are using the custom pipe in the html file.

```
<tr>
  <td>Full Name:</td>
  <td>{{ profession | professionPrefix }} {{ customerName | titlecase }}</td>
</tr>
```

In this case it goes ahead and checks the profession if its doctor it gets prefixed as “Dr” or “Er.” if the profession is an engineer. Accordingly it displays the output.

Customer Details

Customer Name:	CHRIS REGO
Profession:	Engineer
Customer ID:	aed-123-eg2
Account Balance:	\$123.46
Account Date:	Jan 1, 2012
Sybil Score:	40%
Full Name:	Er. Chris Rego

```
TS balance.component.ts x
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'balance-component',
5    templateUrl: './balance.component.html',
6    styleUrls: ['./balance.component.css']
7  })
8  export class BalanceComponent {
9    customerName = 'chris rego';
10   customerID = 'AED-123-EG2';
11   accountBalance = 123.456;
12   accountDate = new Date(2012,0,1);
13   sybilScore = 0.4;
14   profession = 'engineer';
```

(When Engineer is the profession, in response to it pipe transform the data)

Customer Details

Customer Name:	CHRIS REGO
Profession:	Doctor
Customer ID:	aed-123-eg2
Account Balance:	\$123.46
Account Date:	Jan 1, 2012
Sybil Score:	40%
Full Name:	Dr. Chris Rego

```
TS balance.component.ts x
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'balance-component',
5    templateUrl: './balance.component.html',
6    styleUrls: ['./balance.component.css']
7  })
8  export class BalanceComponent {
9    customerName = 'chris rego';
10   customerID = 'AED-123-EG2';
11   accountBalance = 123.456;
12   accountDate = new Date(2012,0,1);
13   sybilScore = 0.4;
14   profession = 'doctor';
```

(When Doctor is the profession, in response to it pipe transform the data)

Pipes: How to create Pipes

Pipes: Ways to create pipes?

Pipes in Angular can be created using the two ways...

- Automatic
- Manually

Automatic Approach

Automatic approach is simple and straightforward in which need to make use of Angular Command-Line tool (ng-cli). By using Automatic Approach it automatically creates the Directive Files and make sure footprints are added.

Example: ng generate pipe pipename

```
PS C:\chris-app> ng generate pipe balance/customerdetail
CREATE src/app/balance/customerdetail.pipe.spec.ts (219 bytes)
CREATE src/app/balance/customerdetail.pipe.ts (217 bytes)
UPDATE src/app/app.module.ts (564 bytes)
```

(In this case, it goes ahead and creates pipe file which includes .spec.ts as well as .ts)

Manual Approach

Manual approach is when we manually had to go ahead and create the required file along with the Manually going ahead and entering the details where needed.(module)

Pipes: How does pipes look like?

```
import { Pipe,PipeTransform } from "@angular/core";
import { BalanceComponent } from './balance.component';

@Pipe({
  name:'professionPrefix'
})

export class BalanceTitlePipe implements PipeTransform{

  balancetitle = new BalanceComponent();

  transform(value: string):string {

    if(this.balancetitle.profession==='doctor') {
      return "Dr. ";
    }
    else if(value.toLowerCase()=='engineer'){
      return "Er. ";
    }
    throw new Error("Method not implemented.");
  }
}
```

(Screenshot: This image is a Pipe file created manually)

Import

Import the Pipe Decorator from the @angular/core library

Import the PipeTransform interface from @angular/core library

```
import { Pipe,PipeTransform } from "@angular/core";
```

Decorator

@Pipe (Pipe) is used to define the typescript file is Pipe. so it can be used in the html

```
@Pipe({
  name:'professionPrefix'
})
```

Logic/Class:

```
export class BalanceTitlePipe implements PipeTransform {

    balancetitle = new BalanceComponent();

    transform(value: string):string {

        if(this.balancetitle.profession==='doctor') {
            return "Dr. ";
        }
        else if(value.toLowerCase()=='engineer'){
            return "Er. ";
        }
        throw new Error("Method not implemented.");
    }
}
```

In this case, as it implement the interface PipeTransform it has a method that needs to be overridden named as transform which contains by default value which is the data that which is present at the left side of pipe expression along with it we can also pass multiple parameters to it as well.

Parameterized Pipes

Parameterized Pipes in this in order to fine tune the pipe we can pass 'n' number of arguments to pipe expression, parameterized which are passed to the pipes can be separated using (:) colon.

```
import { Component } from '@angular/core'
@Component({
    selector: 'birthday-component',
    templateUrl: 'birthday.component.html'
})

export class BirthdayComponent {
    birthday = new Date(2012,1,1);
}
```

```
birthday.component.html ✕
1 <p>Chris is born on {{ birthday | date:"MM/dd/yy"}}</p>
```

Final Output:

Chris is born on 02/01/12

In this case, we have used parameter which is passed to date pipe by using (:) colon followed by the format in which the date needs to be displayed, in which we have fine tuned the output data which we receive.

Using Multiple Pipes...

There are often times when we want to transform the data & then additional transform the data into much different format. This can be done by using multiple pipes in single statement each Separated by the subsequent pipe using | (OR Symbol).

Examples:

```
<p>Chris is born on {{ birthday | date:"MM/dd/yy"}}</p>  
<p>First three Character of person {{ name | slice:0:3 | uppercase }} </p>
```

In this case ,the name contains 'chris' which is in lowercase, after which perform the operation using slice:0:3 and after which we have applied another pipe to to convert 'chr' to convert it to the UPPERCASE which is 'CHR'

Output:

First three Character of person CHR

Summary

“Whether you are displaying dates, modifying strings or formatting currencies, Pipes allows you to do all of that with ease. Pipes come built-in or custom ”

Footnotes

- [@Official Guide to Angular Pipes](#)

Assignment

Assignment 1: (Performing Unit testing for below Assignment).

- Create a Pipe a Manual (without using ng-cli)
 - Name: SortInfo

This Pipe will ensure that the data which is entered gets sorted in ascending order

(Hint: use sort() function)

Make sure all the details

Assignment 2: (Performing Unit testing for below Assignment).

- Create a Pipe manually (using ng-cli)
 - Name: Gender

This Pipe will have parameter which is passed which defines the gender(male or female) and depending upon it value should precede with Mr. OR Mrs. followed by the (value/name)

Mr. Chris

8. Module 6 - Services

Services: What makes them so awesome?

Let me ask you something? What is purpose of the function? Yes, some might say let's you modularize the code make you look cool and blah blah... Makes you look like a coding Ninja... But out of all the hooks and crooks functions in Programming aspect helps us to avoid the redundant code to rewritten and rather simple reuse the function where needed (Call the function) , In the very same way does Angular by change provide us a way to go ahead and avoid rewriting same code again and again which is common throughout an application?

Services to the rescue, Services help to replace the redundant lines of extra code by having a single code which will be reused in the Application by just consuming the service, which far more efficient than having repetitive code throughout the whole application.

Services: What really is a Services?

services, are just normal typescript class? Like are they just normal Class files! (No...)Service helps us to create reusable component which are global and that can be reused whenever needed.

Services: Types of Services

Services: What are the types of Services?

Services, are great feature in Angular which come in two types...

- **Built-In Services**
- **Custom Services**

Built-in Services

Built-in Services, there are a 30 of built-in Services provided by Angular that save your efforts, we would just discuss a few.Built-in Services provide their own set of standard functionalities. Built-in Services precedes with \$ symbol.

Angular has provided tons of Services, some of which are listed below.

- \$http
- \$resource
- \$location
- \$timeout
- \$log

Custom Services

Service!,can be written in our way using custom Services.Service as well already know contain.

```
angular.  
module('myServiceModule', []).  
  controller('MyController', ['$scope', 'notify', function($scope, notify) {  
    $scope.callNotify = function(msg) {  
      notify(msg);  
    };  
  }]).  
  factory('notify', ['$window', function(win) {  
    var msgs = [];  
    return function(msg) {  
      msgs.push(msg);  
      if (msgs.length === 3) {  
        win.alert(msgs.join('\n'));  
        msgs = [];  
      }  
    };  
  }]);
```

Above is an Example of Custom Service which is written in Angular

Services: How to create Services

Services: Ways to Create Services?

Directives in Angular can be created using the two ways...

- Automatic (using ng-cli)
- Manually

Automatic Approach

Automatic approach is simple and straightforward in which need to make use of Angular Command-Line tool (ng-cli). By using Automatic Approach it automatically creates the Service Files.

```
PS C:\Users\chris.rego\Desktop\Angular\new-app> ng generate service testservice
CREATE src/app/testservice.service.spec.ts (404 bytes)
CREATE src/app/testservice.service.ts (140 bytes)
```

(In this case, it goes ahead and creates directive file inside of the AppModule)

Manual Approach

Manual approach is when we manually had to go ahead and create the required files along with the Manually going ahead and entering the details where needed.

Services: How does Services look like?

Deeper Look: Services

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

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

  constructor() { }

}
```

(Screenshot: This image is Service file create above using ng-cli tool)

1. In this case, we go ahead and import the Service Decorator from the @angular/core library

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

2. @Injectable (Decorator) is used to define the typescript file is Service.

```
@Injectable({
  providedIn: 'root'
})
```

3. At the end of the day, it's a class hence we define a class to define our Custom logic.

```
export class TestserviceService {  
  
  constructor() { }  
}
```

Summary

“Services are reusable instances of code that we can go ahead and access wherever needed rather than having redundant code all over the application, Services is more driven towards the approach to write the logic which can be shared and used by multiple components as to encourage code reusability,”

Footnotes

- [Understand Why & Hows of Service @Official Tour](#)
- [@Understanding providers & scopes](#)

-

9. Module 7 - Routing

Routing: What does it take to do Routing ?

Before we even dive into Angular specific routing as a concept, let's just understand exactly what does the word *routing* mean.

The Dictionary definition of Routing :

"the use of a particular path or direction for something to travel or be accessed"

In the same fashion, Angular Routing helps us to access Components based on the URL routes. As Angular helps us to develop Single page application (SPA) which are web application that fits on a single page. Navigation between pages performed without refreshing the whole page which indeed increases the performance and makes it more optimized, fast response & No page reloads. Single Page Application all the interaction to the backend happens behind the scenes. Changes in the view can be done by clicks or navigating using the path entered in the address bar for URL.

Routing: What really is a Routing?

Routing are key essence which help us to create Single Page Application

With the help of routing if the user navigates from one page to another, the page is updated dynamically without page reload, even if the URL changes. It also updates the browsing history so that user can navigate back and forth between pages. Routing helps to route to specific Components, but it's at the Module level.

Routing: Who does the real Routing?

Routing, in the case of Angular is performed using *Angular Router*.

It's actually Javascript implementation of Javascript Routing. Angular Router is packaged and is present in `@angular/router`

Angular Router: What is it?

Angular Router, is an official Angular routing library, written and maintained by the Angular Core Team. Angular Router(@angular/router) takes care of certain things it activates all required Angular components to compose a page when a user navigates to a certain URL.

When user clicks/navigate to a page, Angular Router performs the following steps:

- It checks the Browser URL (user wants to navigate)
- It applies URL redirect
- Depending on URL, check the router state matching the URL
- Depending on the Router State - resolves the Data
- Loads the Angular Components required to Display the page.

We can have one or many Angular Router Service Instances, its recommended to have one generic router service which takes care of the routing needs.

Routing: How to Create Routing

Router can be created manually as well as Automatically, Both the ways are completely fine, but the Automatic approach to it isn't as straightforward cannot be created after creation of the module.

Routing: Manual Approach

Manual approach involves going ahead and creating the routing file manually inside of the particular Module or the root Module which is present.

Routing: Automatic Approach

Automatic approach isn't as straightforward where we something “*ng generate route aladdin*” Boom! That's not the case.

We can create a Route file at the time of creation of Angular Project or we can also specify the same while creating a module in the.

```
PS C:\Users\chris.rego\Desktop\Angular\rouTex> ng generate module testmodule --routing
CREATE src/app/testmodule/testmodule-routing.module.ts (253 bytes)
CREATE src/app/testmodule/testmodule.module.spec.ts (307 bytes)
CREATE src/app/testmodule/testmodule.module.ts (295 bytes)
```

- Ng generate module test_module --routing

Routing : How does Router look like

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

import { ModuleWithProviders } from "@angular/core";
import { ProductListComponent } from '../app/product-list/product-list.component';
import { ProductDetailComponent } from '../app/product-detail/product-detail.component';

const routes: Routes = [
  { path: '', redirectTo: '/products', pathMatch: 'full' },|
  { path: 'products', component: ProductListComponent },
  { path: "product/:id", component: ProductDetailComponent }
];

export const routingModule: ModuleWithProviders = RouterModule.forRoot(routes);
```

(Screenshot: This image is Routing file (app.routing.ts))

1. Importing the Required artifacts (Importing the Components as well)

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

import { ModuleWithProviders } from "@angular/core";
import { ProductListComponent } from '../app/product-list/product-list.component';
import { ProductDetailComponent } from '../app/product-detail/product-detail.component';
```

2. Declaring the Array of the path and the Component to be redirected to.

```
const routes: Routes = [
  { path: '', redirectTo: '/products', pathMatch: 'full' },|
  { path: 'products', component: ProductListComponent },
  { path: "product/:id", component: ProductDetailComponent }
];
```

3. Importing the routes and reusing the it.

```
export const routingModule: ModuleWithProviders = RouterModule.forRoot(routes);
```

Summary

“Routing is a mechanism in Angular which helps to establish purely Single Page Application Principle, by able to loading components on the fly with the help of paths which are bonded to it, Routers at its core is binding on of path to the its relative components, with some more advance features...”

Footnotes

- [@About Angular Router](#)
- [@Official Guide Angular](#)
- [@More on Angular Routing](#)

Assignment

Assignment 1: Now since we have understood the basic building blocks of angular we would try to create a really simple application, which we would implement the concepts of modules, component, services & routing. For the purpose of this assignment we would be using Bootstrap 4 as UI framework, we need to make sure that we don't provide the links to bootstrap4 in the cdn but rather install as a package in our project using npm. We have complete page dedicated to the setup of bootstrap for the application

Refer to the link and read the description about the assignment.

Link: [Bootstrap Setup](#)

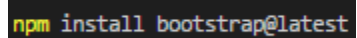
The Application that we would develop will display the list of the favourite bikes and cars. Let's start by creating a new Angular application using ng-cli

■ Name (Application): RoutingBasic

After creating the application make sure to the install bootstrap package along with its dependencies in the project folder itself (locally)

- Run the command: (Installing the latest Bootstrap package)

npm install bootstrap@latest



(This goes ahead and install bootstrap locally and creates the entry in the package.json and adds the bootstrap files in the specific bootstrap directly present in node_modules)

- Run the command: (Installing the dependencies)

npm install jquery popper.js --save

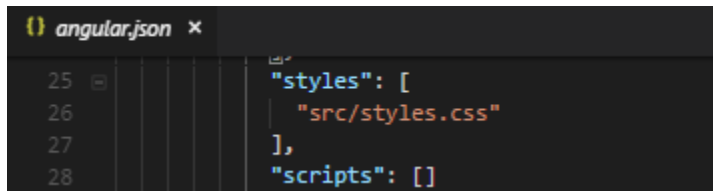
```
npm install jquery popper.js --save
```

(This goes ahead and install the dependencies needed for bootstrap 4 which includes the jquery & popper, which further the details are added in the Package.json & files are added in the node_modules)

Now since we have installed the dependencies we need to go ahead and need to add the reference to bootstrap and its dependencies at the required places in the project, to ensure that we can use Bootstrap 4 in our project.

Navigate to the file (angular.json) present in the root folder of the Application

We need to make the entries in the angular.json under styles & scripts array to add relative path to bootstrap file and jquery and popper.js



```
{
  "styles": [
    "src/styles.css"
  ],
  "scripts": [

```

We will now be adding the relative paths in the styles & scripts arrays we also need to make sure that while declaring the paths the order is important and we shouldn't change them.

```
"styles": [
  "src/styles.css",
  "./node_modules/bootstrap/dist/css/bootstrap.min.css"
],
"scripts": [
  "./node_modules/jquery/dist/jquery.slim.min.js",
  "./node_modules/popper.js/dist/popper.min.js",
  "./node_modules/bootstrap/dist/js/bootstrap.min.js"
]
```

```
angular.json x
24     ],
25     "styles": [
26       "src/styles.css",
27       "./node_modules/bootstrap/dist/css/bootstrap.min.css"
28     ],
29     "scripts": [
30       "./node_modules/jquery/dist/jquery.slim.min.js",
31       "./node_modules/popper.js/dist/popper.min.js",
32       "./node_modules/bootstrap/dist/js/bootstrap.min.js"
33     ]
  }
```

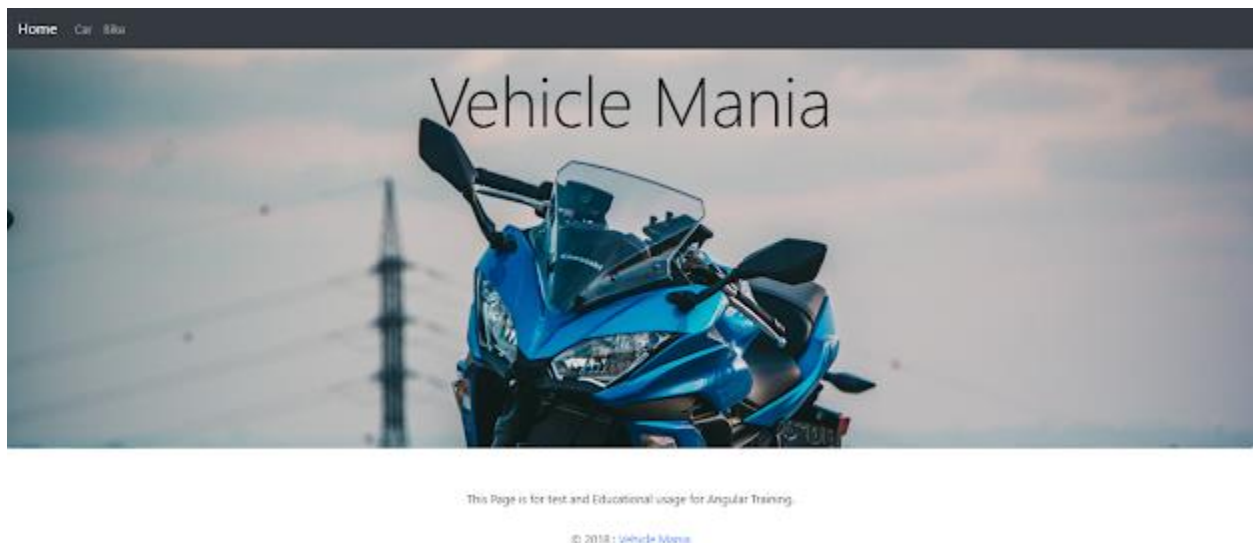
After ensuring that we have specified the path now we can go ahead and start working on the rest of the Application.

In this case, we would be performing routing between components present in the same AppModule

- Go ahead and create a two new Components without the use of ng-cli and add the entries in the AppModule.
 - Name: Bike
 - Name: Car
 - Name: Home
 - Name: PageNotFound

Routing will help us to route between different components service on the other hand will contain functions which will be used to display the data.

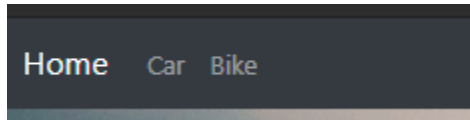
Home Component - As the name suggest is used to display the default home page.



Here's a jist of how the default homepage should look like.

Which contains the navigation bar with links to redirect to different section of the page.
Along with it, it also contains the footer which contains basic information of what the project is.

Nav Bar - We need to make sure no matter what component we redirect to, but the navbar is always present at the top.



Footer:

This contains the basic information of the Web Application.

This Page is for test and Educational usage for Angular Trainin

© 2018 : [Vehicle Mania](#)

Bike Component - Will be used to display information of the bike.

Home Car Bike		
Bike Mania		
BikeName	Country	Price
KTM	India	1000
DUCATI	Italia	2000
MV AUGUSTA	Italia	4000
BMW SR 10000	Germania	11000
This Page is for test and Educational usage for Angular Training.		
© 2018 : Vehicle Mania		

Car Component - Will be used to display information of the car .

- CarName | Country | Price

Car Mania

CarName	Country	Price
HONDA CITY	Japan	1000
HYUNDAI ELITE I20	Japan	2000
MARUTI SWIFT XRR	India	4000
BMW Z 10000	Germania	11000

This Page is for test and Educational usage for Angular Training.

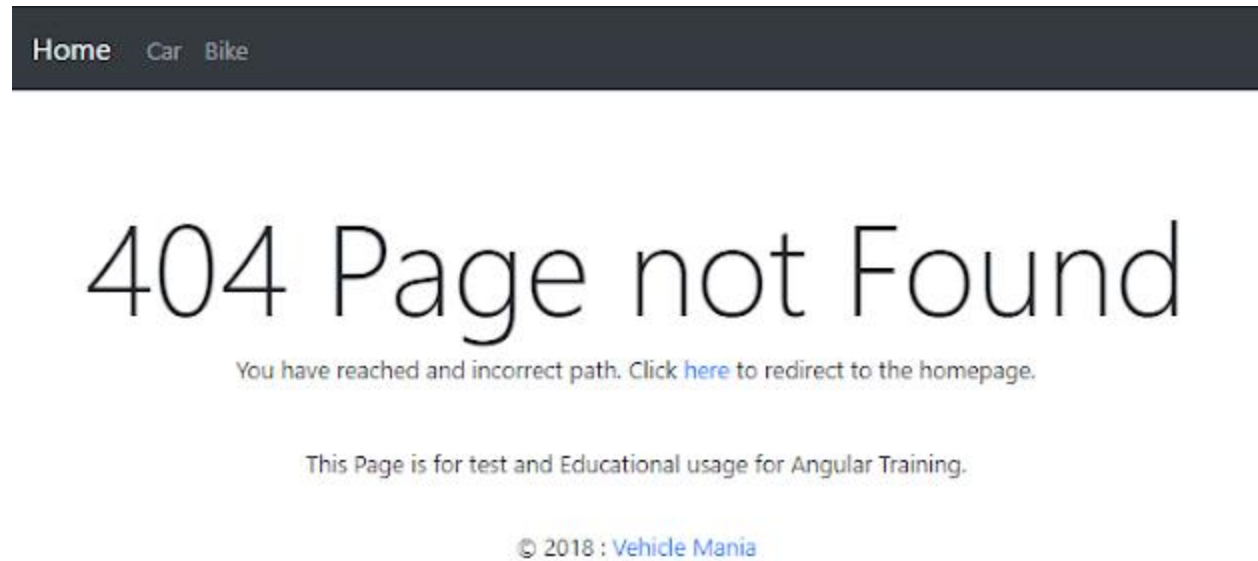
© 2018 : [Vehicle Mania](#)

Service:

In this case, the Data that is getting displayed we can write the logic of it using the service instead of writing it in the component.

We need to create a service named as Vehicle and then write the two function will return the list of details of both the Bike & Car.

In case, if the user enters and incorrect URL we should also have a page dedicated for displaying Page Not Found error. (404) in order to serve the purpose we have a component called as Page Not Found Component.



As a whole the application will have 4 components, which will display the information of the HomePage, Bikes, Cars & Page Not Found. the information can be written in the service as function which can then be consumed by the components and then

10. Module 8 - Testing In Angular

What is Test Driven Development(TDD) And Behavioral Driven Development(BDD)?

Test driven development is a methodology for writing the tests first for a given module and for the actual implementation afterward. If you write your tests before your application code, that saves you from the cognitive load of keeping all the implementation details in mind, for the time you have to write your tests.

BDD allows you to write software requirements as specifications in a human-readable format, and use those specifications to run tests that make sure that the software does what is expected.

Testing Frameworks in JS

- Mocha : Mocha is a Javascript test framework running on Node Js. It tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases.
-
- Jasmine : Jasmine is a behavior-driven development framework for testing JavaScript code. It does not depend on any other JavaScript frameworks.
-
- Tape: Tape is a tool for tap-producing tests on Meteor for node and browser that requires a few amount of APIs. It can be used for testing also unit and integration.

Why to test angular code?

- **Improve the design of implementations**

Start coding a feature without giving it a lot of thought to the design is a very common mistake among developers. Using unit testing is going to enforce to think and rethink the design, and if you are using TDD the impact is even bigger.

- **Allows refactoring**

Since you already have tests that ensure you that everything is working as expected, you can easily add changes to that code with the certainty that you are not adding any bugs.

- Add new features without breaking anything

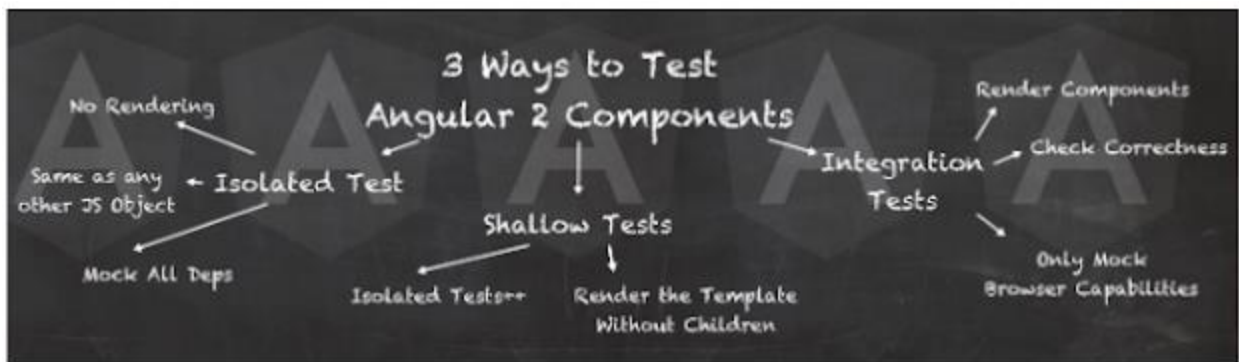
When you are adding a new feature you can run the tests to ensure that you ain't breaking any other part of the application.

- Tests are good documentation.
- Guard against breaking changes
- Tests make developers more confident about their work.
- Analyze code behaviour(expected and Unexpected)

"You can say that all their benefits come at a great cost: TIME, but this is completely false. All the time that using unit testing may cost you is going to be small compared to the time they are going to save you later when you are introducing new features or making any refactors. The time spent resolving bugs is going to be drastically smaller than if you are not using unit testing."

Types of Testing

- Unit test - Test individual comp independently
- Integration Test - Test cmp with its dependencies.
- End To End Test - Test the entire application as whole



How to run test?

This command is going to execute the tests, open the browser, show a console and a browser report and, not less important, leave the tests execution in watch mode. The test file extension must be .spec.ts so that tooling can identify it as a file with tests.

Run Command: **ng test**



```
describe('Component: User', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [UserComponent]
    });
  });

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

  it('should use the user name from the service', () => {
    let fixture = TestBed.createComponent(UserComponent);
    let app = fixture.debugElement.componentInstance;
    let userService = fixture.debugElement.injector.get(UserService);
    fixture.detectChanges();
    expect(userService.user.name).toEqual(app.user.name);
  })
});
```

Testing: Jasmine

Jasmine is a behavior-driven development framework for testing JavaScript code. It does not depend on any other JavaScript frameworks. It does not require a DOM. And it has a clean, obvious syntax so that you can easily write tests.

When you create the project all the dependencies get installed among them everything you are going to need to create the tests.

describe block - global Jasmine function that accepts two parameters. The first parameter is the title of the test suite, and the second one is its actual implementation. The specs are defined using an **it** function that takes two parameters, similar to that of the **describe** block.

expect function - used by Jasmine to determine whether a spec should pass or fail. The **expect** function takes a parameter which is known as the actual value. It is then chained with another

function that takes the expected value. These functions are called matcher functions, and we will be using the matcher functions like `toBeTruthy()`, `toBeDefined()`, `toBe()`, and `toContain()`

Refer this link [Jasmine](#)

Testing: Fundamental Concepts

- Suites — `describe(string, function)` functions, take a title and a function containing one or more specs.
- Specs — `it(string, function)` functions, take a title and a function containing one or more expectations.
- Expectations — are assertions that evaluate to true or false. Basic syntax reads `expect(actual).toBe(expected)`
- Matchers — are predefined helpers for common assertions. Eg: `toBe(expected)`, `toEqual(expected)`.

Find a complete list [here](#).

Jasmine offers four handlers to add our setup and teardown code: `beforeEach`, `afterEach` executed for each spec and `beforeAll`, `afterAll` executed once per suite. Use `beforeEach` and `afterEach` to do changes before and after each spec. A good practice to avoid code duplication on our specs is to refactor repetitive code into the setup.

Dependency Injection (DI)

`TestBed`, similarly to `@NgModule` helps us to set up the dependencies for our tests. We call `TestBed.configureTestingModule` passing our configuration. This information will then be used to resolve any dependencies.

Testing: Karma

`karma.conf.js` is the configuration file for the Karma test runner and the only configuration file that we will need for writing unit tests in Angular. By default, Chrome is the default browser-launcher used by Karma to capture tests. We will create a custom launcher for running the headless Chrome and add it to the `browsers` array.

All the test specs should be located inside the application's `src/app/` directory because that's where Karma looks for the test specs. If you create a new component or a service, it is important that you place your test specs inside the same directory that the code for the component or service resides in.

Let's take a look at the karma configuration file created by angular-cli.

- frameworks: this is where jasmine gets set as a testing framework. If you want to use another framework this is the place to do it.
- reporters: this is where you set the reporters. You can change them or add new ones.
- autoWatch: if this is set to true, the tests run in watch mode. If you change any test and save the file the tests are re-build and re-run.
- browsers: this is where you set the browser where the test should run. By default is google but you can install and use other browsers launchers.

Testing: Test Entry File

The angular-cli configuration of karma uses the file “test.ts” as the entry point of the tests for the application.

- An environment to run angular tests is being created using all the imports at the begging of the file.
- TestBed is a powerful unit testing tool provided by angular, and it is initialized in this file.
- Finally, karma loads all the tests files of the application matching their names against a regular expression. All files inside our app folder that has “spec.ts” on its name are considered a test.

For more details [test-method-with-example](#)

Component Testing

Component, unlike all other parts of an Angular application, combines an HTML template and a TypeScript class. The component truly is the template and the class working together. and to adequately test a component, you should test that they work together as intended.

Refer link : [basic-component-test](#)

The following link, comprising most of this guide, explore common component testing scenarios [Component Test Scenarios](#) :

Service Testing

Refer link for service testing [service-testing](#)

Directive Testing

Attribute directive modifies the behavior of an element, component or another directive. Its name reflects the way the directive is applied: as an attribute on a host element.

Testing a single use case is unlikely to explore the full range of a directive's capabilities. Finding and testing all components that use the directive is tedious, brittle, and almost as unlikely to afford full coverage. Class-only tests might be helpful, but attribute directives like this one tend to manipulate the DOM. Isolated unit tests don't touch the DOM and, therefore, do not inspire confidence in the directive's efficacy. A better solution is to create an artificial test component that demonstrates all ways to apply the directive.

Refer this link [directive-testing](#)

Pipe Testing

Pipes are easy to test without the Angular testing utilities.

A pipe class has one method, transform, that manipulates the input value into a transformed output value. The transform implementation rarely interacts with the DOM. Most pipes have no dependence on Angular other than the @Pipe metadata and an interface.

Refer this link for example - [pipe-testing](#)

Test debugging

Debug specs in the browser in the same way that you debug an application.

1. Reveal the karma browser window (hidden earlier).
2. Click the DEBUG button; it opens a new browser tab and re-runs the tests.
3. Open the browser's "Developer Tools" (Ctrl-Shift-I on windows; Command-Option-I in OSX).
4. Pick the "sources" section.
5. Open the 1st.spec.ts test file (Control/Command-P, then start typing the name of the file).
6. Set a breakpoint in the test.
7. Refresh the browser, and it stops at the breakpoint.

Footnotes

- [Detail-example-angular-testing](#)
- [Class-and-pipe-testing](#)
- [Official-angular-doc-testing](#)

11. Introduction to forms in Angular

- What is a Reactive Form ?
- How data-flow works in template-driven forms ?
- Form Validations

Resources to learn Angular Forms:

<https://angular.io/guide/forms-overview>

12. Mini Project

Mini Project: User Dashboard

1. Please refer the [requirement](#) for creating user dashboard also write test methods for all components and service parallelly .

(In this case, we need to use Bootstrap 4 as UI Framework for the Assignment

More instruction on how to include bootstrap 4 and its dependencies is mentioned [@linkbelow](#))

2. It needs to be strictly observed that the code is committed to git regularly.

3. Create component for each endpoint url [given](#) with required fields.

4. Create one service named “api.service.ts” containing all apis (GET, POST, PUT, DELETE) and session management. (4 hrs)

5. Create a Login component with following fields

- a. Username (Email)
- b. Password (password should be alphanumeric and minimum length 8 characters)
 - All fields are required.

On clicking login button, after successful login user should be redirected to user dashboard component. (2 hrs)

6. Maintain sessionId of user in “api.service.ts”. After login sessionId should be created. Store that Id in cookie or any local storage. After logout sessionId must get deleted from cookie or local storage. (2 hrs)

6.

Using Bootstrap 4 in Angular Project

We need to use Bootstrap 4 in our project, in order to use Bootstrap 4 in our project we need to make sure, the required dependencies for the Bootstrap are also installed along with it.

Below is the step which contains the step-by-step overview to setup Bootstrap 4 on our machine.

1. Run the following command: `npm install --save bootstrap`

```
PS C:\Users\chris.rego\Desktop\Angular\route> npm install --save bootstrap
npm WARN bootstrap@4.1.3 requires a peer of jquery@1.9.1 - 3 but none is installed. You must install peer dependencies yourself.
npm WARN bootstrap@4.1.3 requires a peer of popper.js@^1.14.3 but none is installed. You must install peer dependencies yourself.
```

(it goes ahead and install the latest version of bootstrap@4.1.3 and gives a warning that the dependencies are not installed and we need to install it)

2. Let's run the commands to install other dependencies

```
> npm install jquery --save
```

```
> npm install popper.js --save
```

* `npm install jquery --save`

* `npm install popper.js --save`

- Bootstrap is now included in the `node_modules` folder.
- Now we have all the dependencies (bootstrap, jquery, popper) installed.

3. Now enter the details in the `angular-cli.json` in your project

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

```
"scripts": [ "../node_modules/jquery/dist/jquery.min.js",
"../node_modules/popper.js/dist/umd/popper.min.js",
"../node_modules/bootstrap/dist/js/bootstrap.min.js" ],
```

Note: Don't change the sequence of the js file which are added.

Mini Project: Reference Guide

Step	Action	Outcome
Clone	1. Got to command prompt.	Your local clone

repository	2. Change the current working directory to the location where you want the cloned directory to be made.	will be created.
	3. Make sure you have enigma gitlab account and following repository shared to you.	
	4. Type git clone https://gitlab.com/enigma-training/ng-training.git	
	5. Press Enter.	
Start Server	1. Go to cloned repository folder.	Your server will start at port 5000.
	2. Hit command npm install.	
	3. Hit command node server/server.js	
	4. This will be start server on port 5000(to change default port got to server/config/config.json and change port number).	
Configure MongoDB	1. Install mongodb to run locally on localhost:27017.	Your database will be ready.
Check server	1. Open postman app (Google chrome app).	API is ready to use.
	2. Create new request.	
	3. Add http://127.0.0.1:5000/users/register in url.	
	4. Change method to POST.	
	5. Add {"email":"mini@project.com","password":"test12345","firstName":"monika","lastName":"pingale"} in body.	
	6. Press Send.	
	7. Check status code 200 and response contain user object.	

Mini Project: API Reference

API's				
Endpoint	Method	Request		Output
domain- http://localhost:PORT		Body	Header	
/register	Post	{ "email": "email@example.com", "firstName": "first_name", "lastName": "last_name", "userType": "Admin/trainee" }		{ "email": "email@example.com" }
/login	POST	{"email": "email_id@example.com", "password": password , "userType": "Admin/Trainee" }	empty	{ "email": "email@example.com" }
/logout	POST	empty	empty	{ "message": "user logout successfully" }
/forgetPassword/:username	get	empty		{ "message": "forgotten password send on mail Id" }
/updatePassword	put	{ "username": "username", }		{ "message": "pass"

		<pre> “oldpassword” : “oldpassword”, “newpassword” : “newpassword” } </pre>		<pre> word updated successfully } </pre>
/question	post	<pre> { “technology” : “java”, “username” : “email”, “question ” : “What is java ”, “options” : [“library”, “platform”], “Answer” : “platform”, “Mark” : 1 } </pre>		<pre> { “message” : “question created successfully” } </pre>
/questionSet	post	<pre> { “Technology” : “Java”, “noOfQuestions” : 2, “Username” : “Email” } </pre>		<pre> { “questionSetID”: “ObjectID” } </pre>
/inviteTrainee	post	<pre> { “traineeEmail” : “email”, “questionSetId” : “objectId” } </pre>		<pre> { “message”: “invitation send to trainee” } </pre>
/startTest/:questionSetId	get	<pre> { “email” : “Email” } </pre>		<pre> { “questionset”: [], } </pre>
/submitTest	post	<pre> { “email” : “email”, “questionSet” : [{ “question” : </pre>	questionSetID : “ObjectID”	<pre> { message : “”, “score” : “score” } </pre>

		<pre>"question", "answer" : "answer" }] }</pre>		}