Table of Content

- Fundamental of Angular
    - Building Blocks of Angular
    - Angular Architecture
    - Steps to Prepare First Angular Application.
        - Creating Application Folder
        - Creating @angular/cli package
        - Craeting new angular application
        - Open the angular application in vs code
        - Modify app.component.html
        - Run the application
    - Folder Structure of Angular Application
        - package.json
        - packages of angular
        - tsconfig.json
        - protractor.config.js
        - karma.confg.js
        - angular-cli.json
        - src/style.css
        - src/index.html
        - src/main.ts
        - src/app/app.module.ts
        - src/app/app.component.ts
        - src/app/app.component.html
        - src/app/app.component.css
        - src/app/app.component.spec.ts
        - Etc..
    - Components
    - Modules
    - Data Bindings
    - Build in directives
        - ngIf
        - ngIf and else
        - ngSwitch
        - style
        - ngClass
    - ngFor
    - Working with Multiple components
    - Children of Components
    - Life Cycle Hooks
    - Services

- Pipes
- Forms And Validations
    - Template Driven Form
    - Reactive Form
- Routing
- Guards
- RxJS
- Angular Material
- Unit Testing

Course : Angular
No of Days : 20 days
Daily Session : 1 hour

Pre-Requisites
HTML,CSS,JS,TypeScript

DAY-1

Agenda:

1. What is Angular?
   - SPA
   - Goals of Angular
   - Versions
   - AngularJS vs Angular
2. Angular Set Up.
   - Installation of Node.js
   - Installation of VS Code
   - Insallation of Angular

Introduction to Angular:

- Angular is a client side framework, which is used to create web applications.
- The framework provides skeleton of the project and specifies clear guidelines, where to write which type of code.
- Angular can be used in combination with any server side platform such as Java, NodeJS, Asp.Net, PHP, Python etc.
- Angular is developed using "TypeScript" language, which is a superset of JavaScript
  language.
- Angular is the most-used client side framework.
- Angular was developed by Google.
- Angular is free-to-use (commercially too).
- Angular is open-source. That means the source code of angular is available online for free of cost.
- Angular is cross-platform. That means it works in all the operating systems.
- Angular is cross-browser compatible. That means it works in all the browsers, except less than IE 9 (which is completely out-dated).
- Angular is mainly used to create "data bindings". That means, we establish relation between a variable and html element; When the value of the variable

is changed, the same will be automatically effected in the corresponding html element; and vice versa.

- ❖ So that the developer need not write any code for DOM manipulations (updating values of html tags, based on user requirements. for example, updating the list of categories when a new category added by the user). Thus the developer can fully concentrate on the application logic, instead of writing huge code for DOM manipulations. So we can achieve clean separation between "application logic" and "DOM manipulations".
- ❖ Angular mainly works based on "Components". The component is a class, which reprensets a specific section (part) of the web page.

Goals of Angular:

- ❖ Make a Single Page Application:
  A Single Page Application (SPA) is a web application that loads and renders all necessary content on a single web page. Instead of navigating to different pages and reloading the entire page, SPAs dynamically update the content on the page as users interact with it, providing a more seamless and responsive user experience.
  - ➢ It make Development easier.
  - ➢ SPA provides client-side navigation system; but can communicate with server only through AJAX; the web page never gets refreshed fully.
  - ➢ **Ex:** Gmail, PayPal, Pinterest, Gmail, Facebook
- ❖ Seperation HTML logic from Application Logic.
- ❖ Performing Unit Testing

Versions:

Angular Version	Release Date	Features
AngularJS 1.x	Oct 2010	Initial release of AngularJS, a JavaScript-based framework for building dynamic web applications.

Angular 2	Sep 2016	Complete rewrite of AngularJS. Introduced a component-based architecture, improved performance, and better tooling support.
Angular 4	Mar 2017	Improved performance, smaller bundle sizes, and added new features such as the introduction of the 'else' clause in Angular templates.
Angular 5	Nov 2017	Improved build optimizer, support for progressive web apps (PWA), and introduction of Angular Universal Transfer API.
Angular 6	May 2018	Introduced Angular Elements for building reusable components, Angular Material starter components, and improved Angular CLI commands.
Angular 7	Oct 2018	Improved performance and introduced features like Angular CLI prompts, drag-and-drop capabilities, and Virtual Scroll.
Angular 8	May 2019	Introduced Ivy Renderer as an opt-in preview, differential loading for smaller bundles, and improved Angular CLI commands.
Angular 9	Feb 2020	Introduced the Ivy

		Renderer as the default rendering engine, improved performance, and updated dependencies to their latest versions.
Angular 10	June 2020	Improved performance and introduced stricter type checking with the 'strict' mode enabled by default.
Angular 11	Nov 2020	Improved performance, support for webpack 5, and updated dependencies.
Angular 12	May 2021	Introduced stricter type checking for templates, improved build and testing processes, and updated dependencies.
Angular 13	Nov 2021	Improved performance, enhanced developer experience, and updated dependencies.

Difference between AngularJS and Angular

AngularJS	**Angular**
AngularJS is a **JavaScript-based** framework.	Angular is a complete rewrite of AngularJS and is built with **TypeScript**.
It follows the MVC archteicture	It follows the components and directives.

It uses ng-bind in order to bind data from view to model and vice versa.	It uses () and [] to bind the data between view and model.
It uses $routeprovider.when() for routing configuration	It uses @Route Config{(..)} for routing configuration.
It does'nt use the CLI tool	It uses the CLI tool

Browser Compatability of Angular

S.NO	Browser	Support Version
1.	Google Chrome	Any version
2.	Mozilla Firfox	Any version
3.	MS Internet Explorer	9+

DAY-2

2. Angular Set Up.

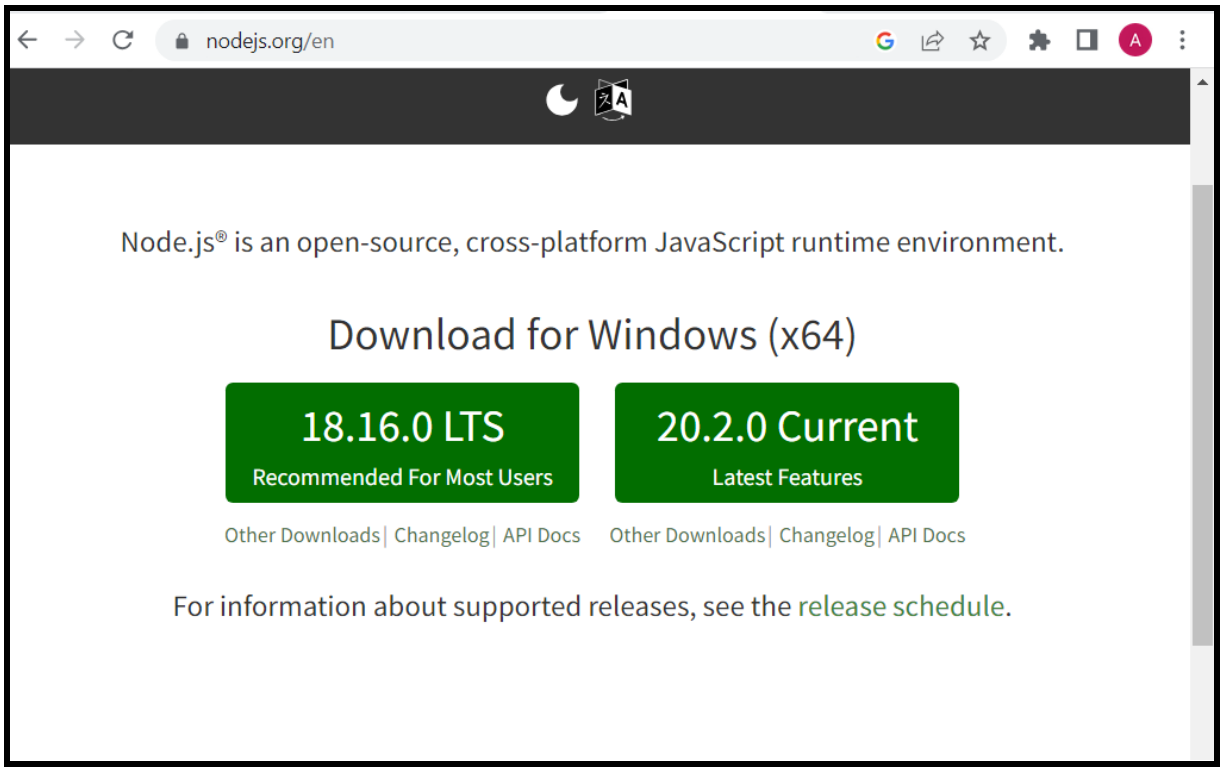
Angular

==========

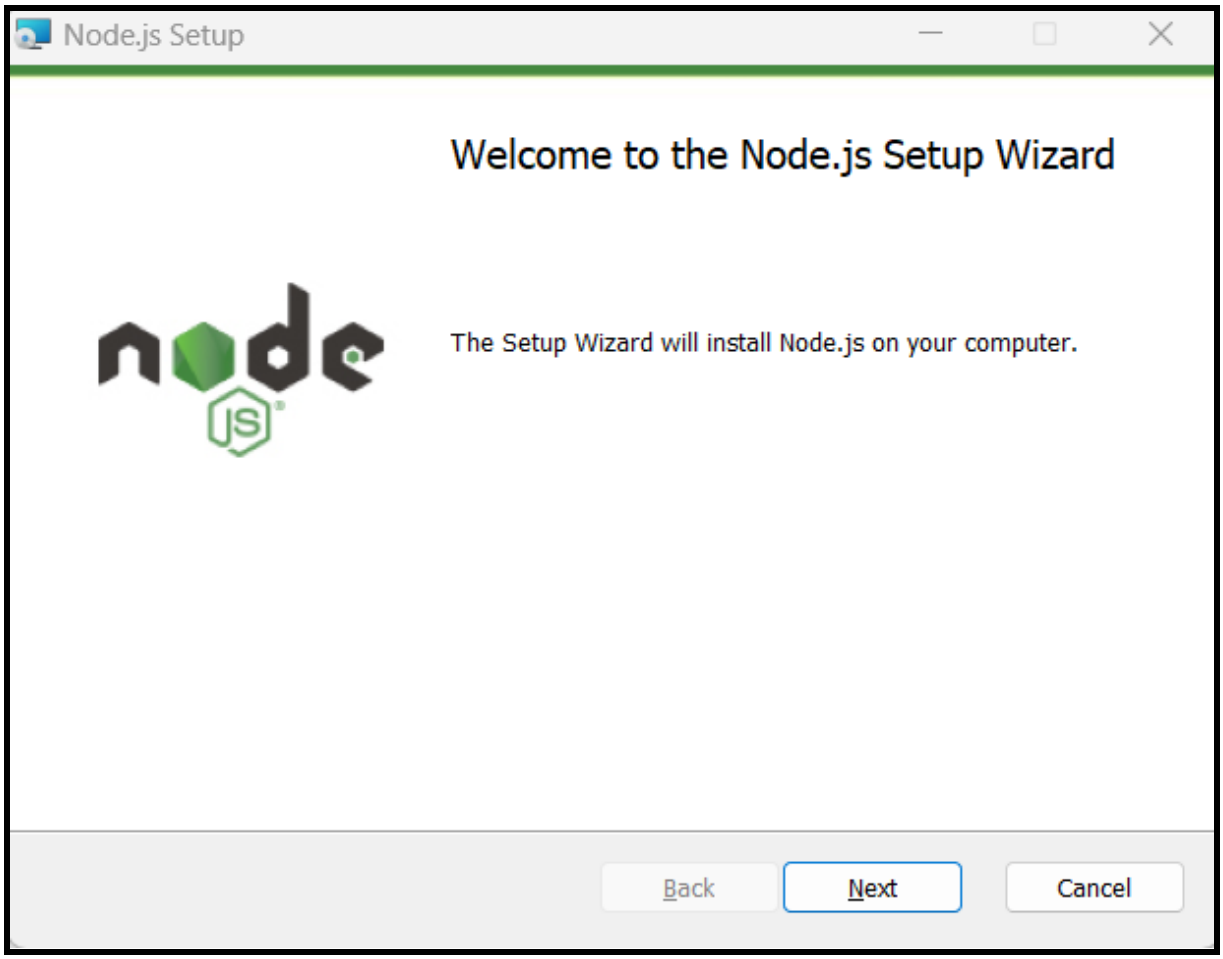
The following are the steps involved to install Angular in the window operating system.

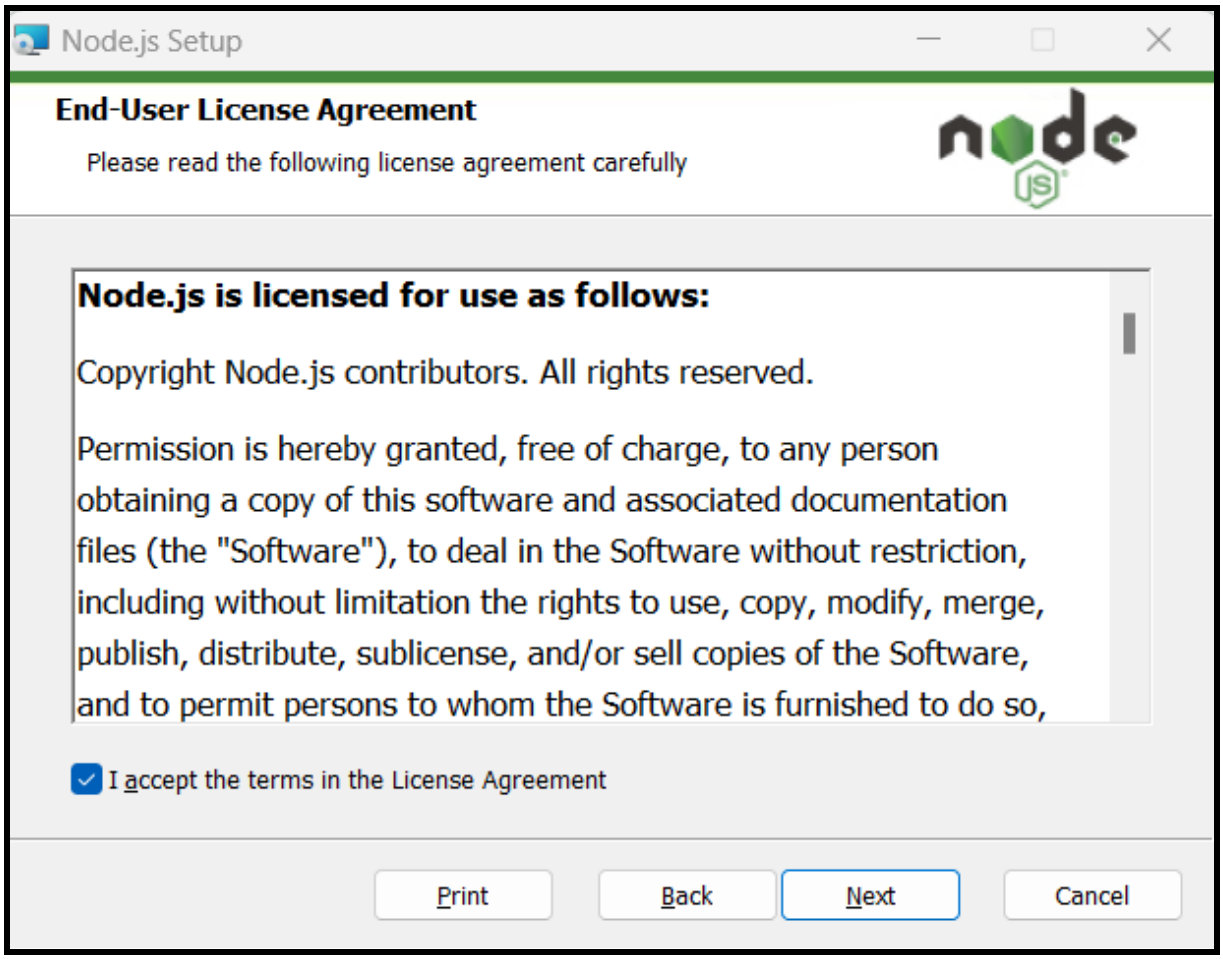
Step1: Install NodeJS in the windows. In order to download we use the following url:

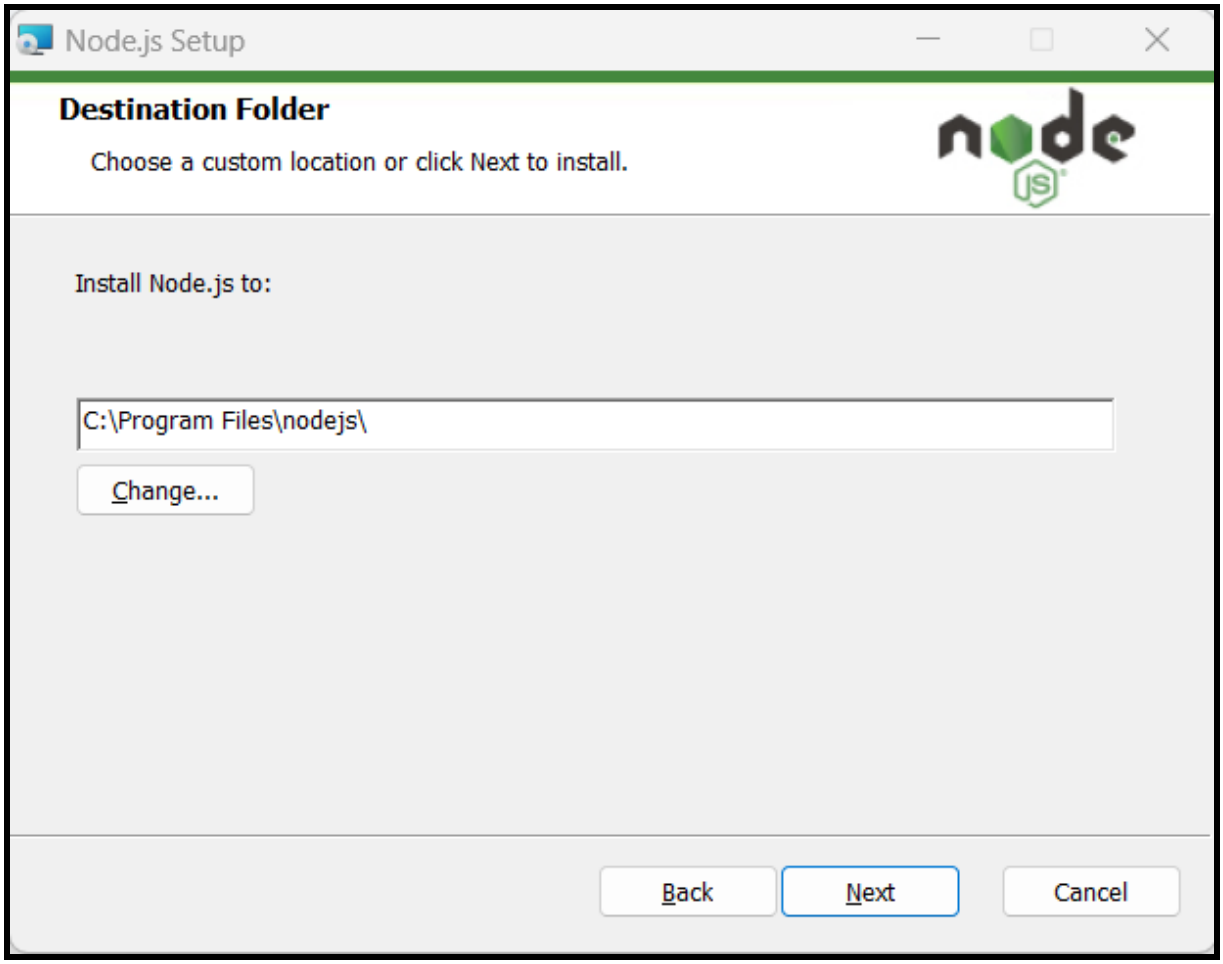
https://nodejs.org/en

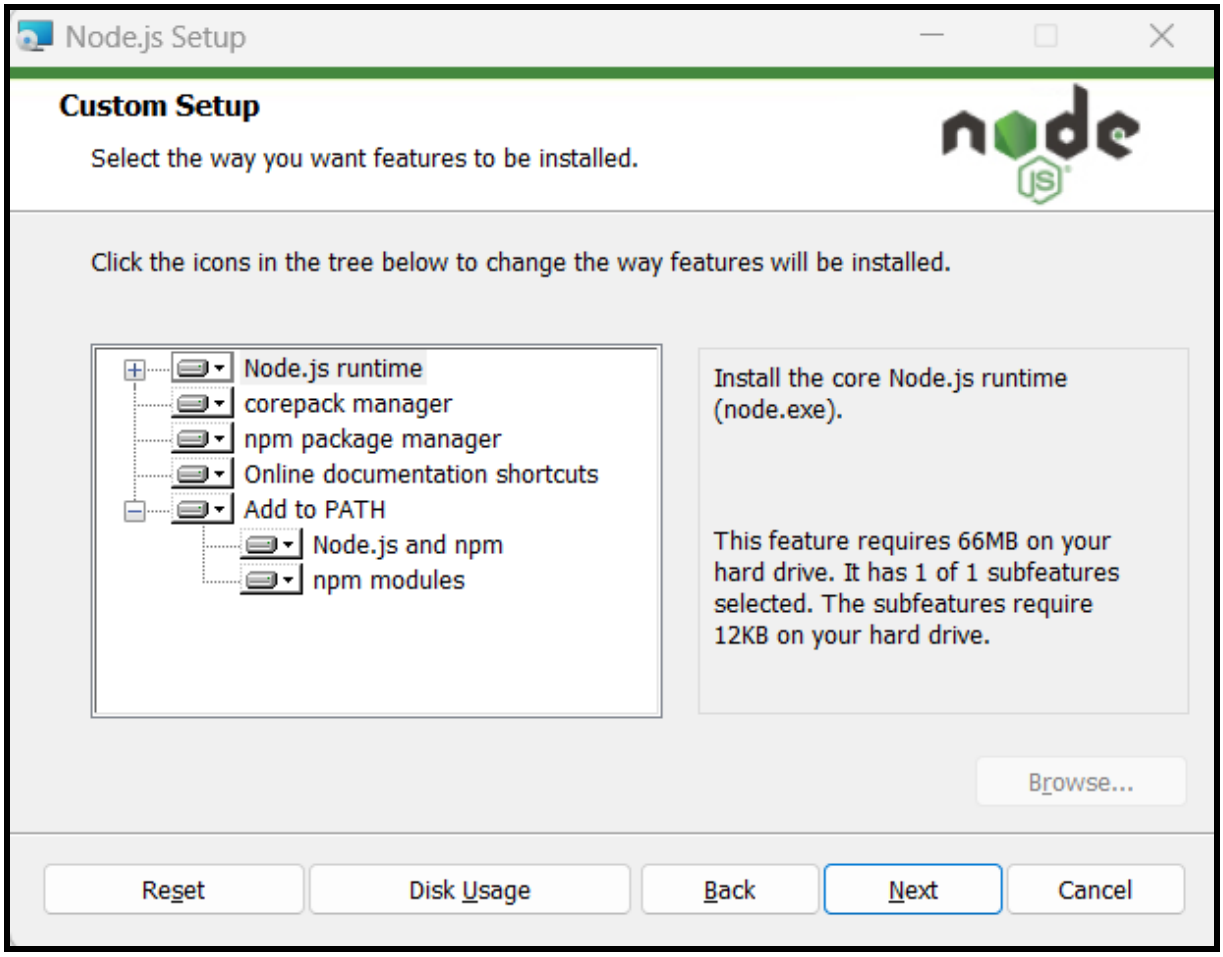


After download the nodejs the following is the way of installation of nodejs:

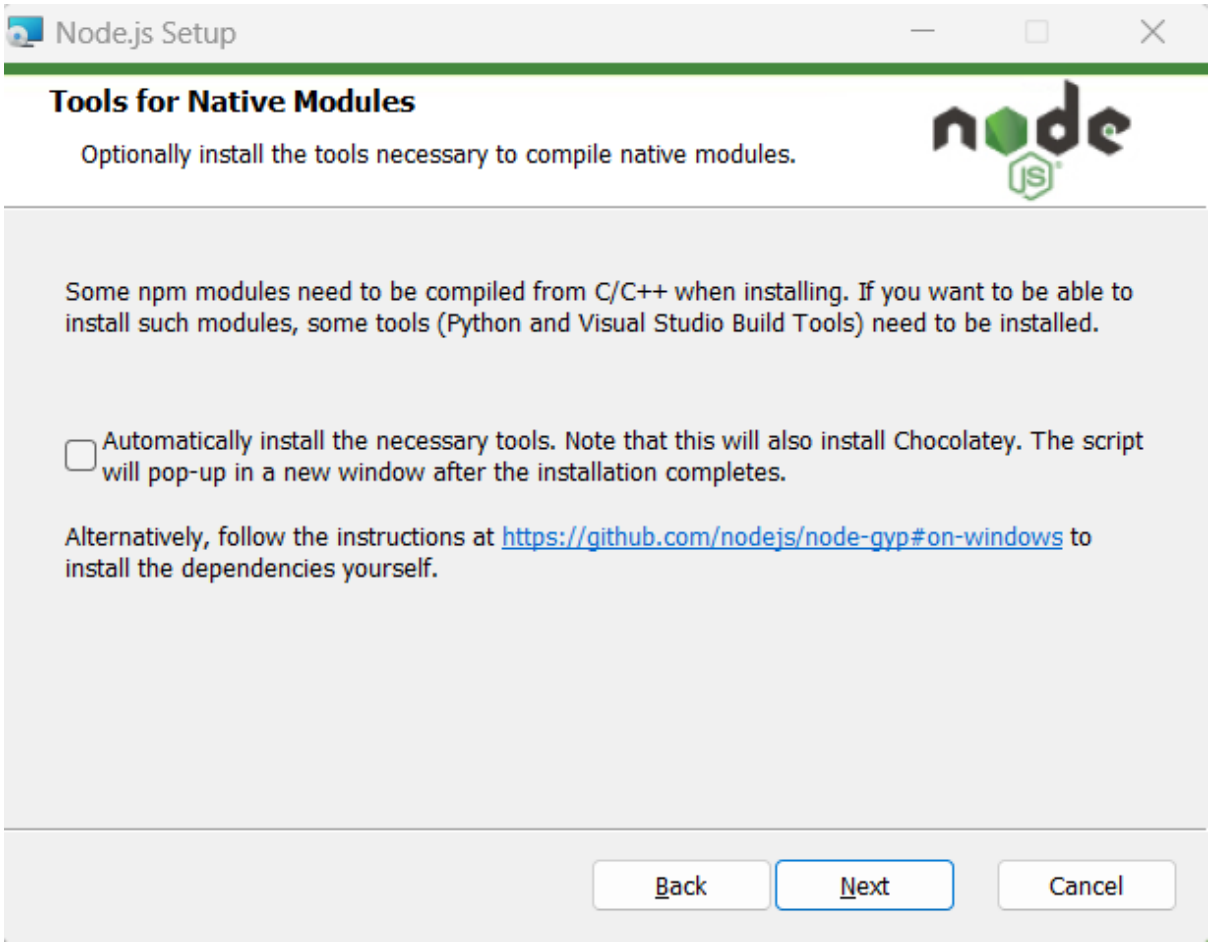

Node.js Setup
Welcome to the Node.js Setup Wizard
The Setup Wizard will install Node.js on your computer.
Back
Next
Cancel


Node.js Setup
End-User License Agreement
Please read the following license agreement carefully
Node.js is licensed for use as follows:
Copyright Node.js contributors. All rights reserved.
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so,
I accept the terms in the License Agreement
Print
Back
Next
Cancel

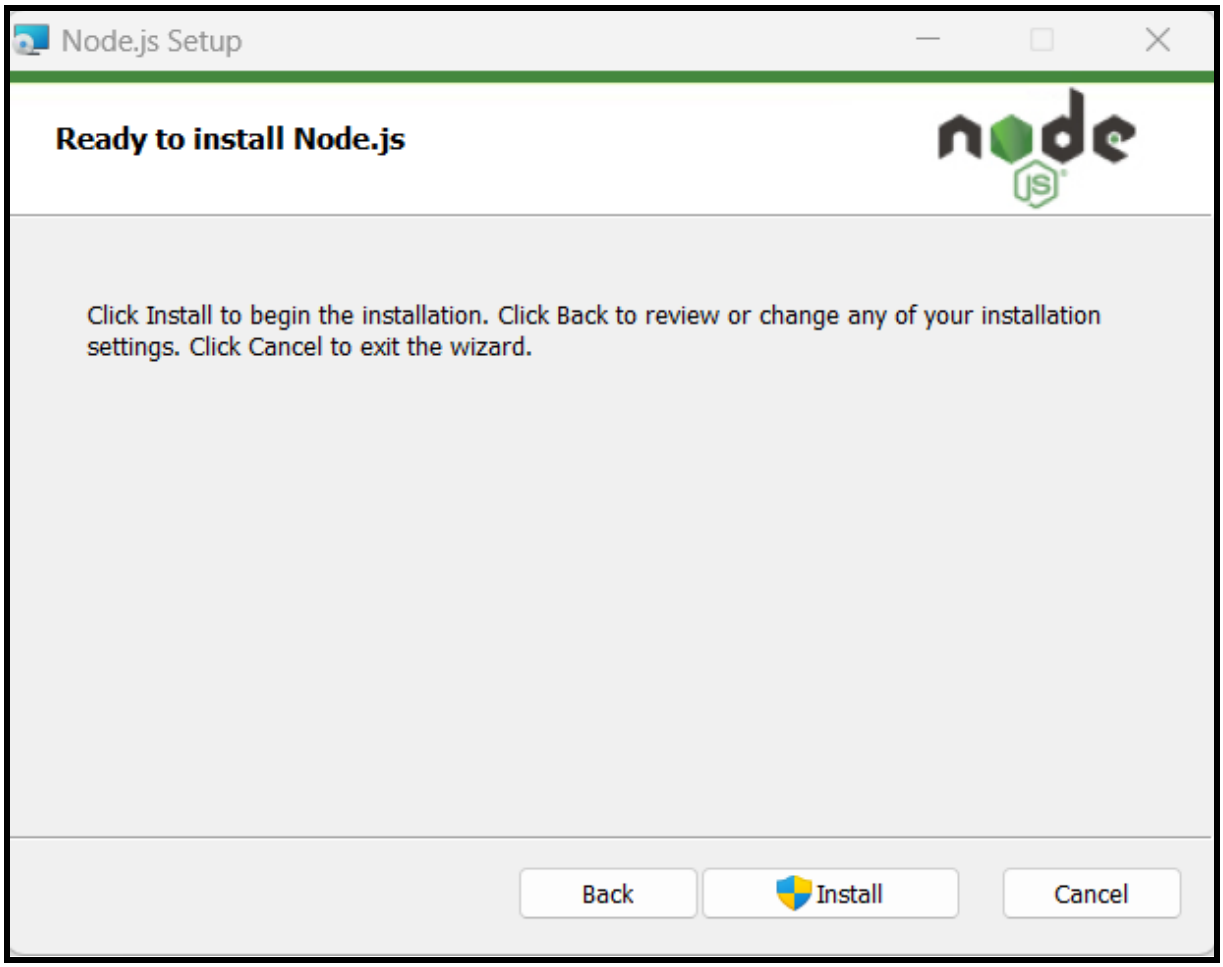

Node.js Setup
Destination Folder
Choose a custom location or click Next to install.
Install Node.js to:
C:\Program Files\nodejs\
Change...
Back
Next
Cancel



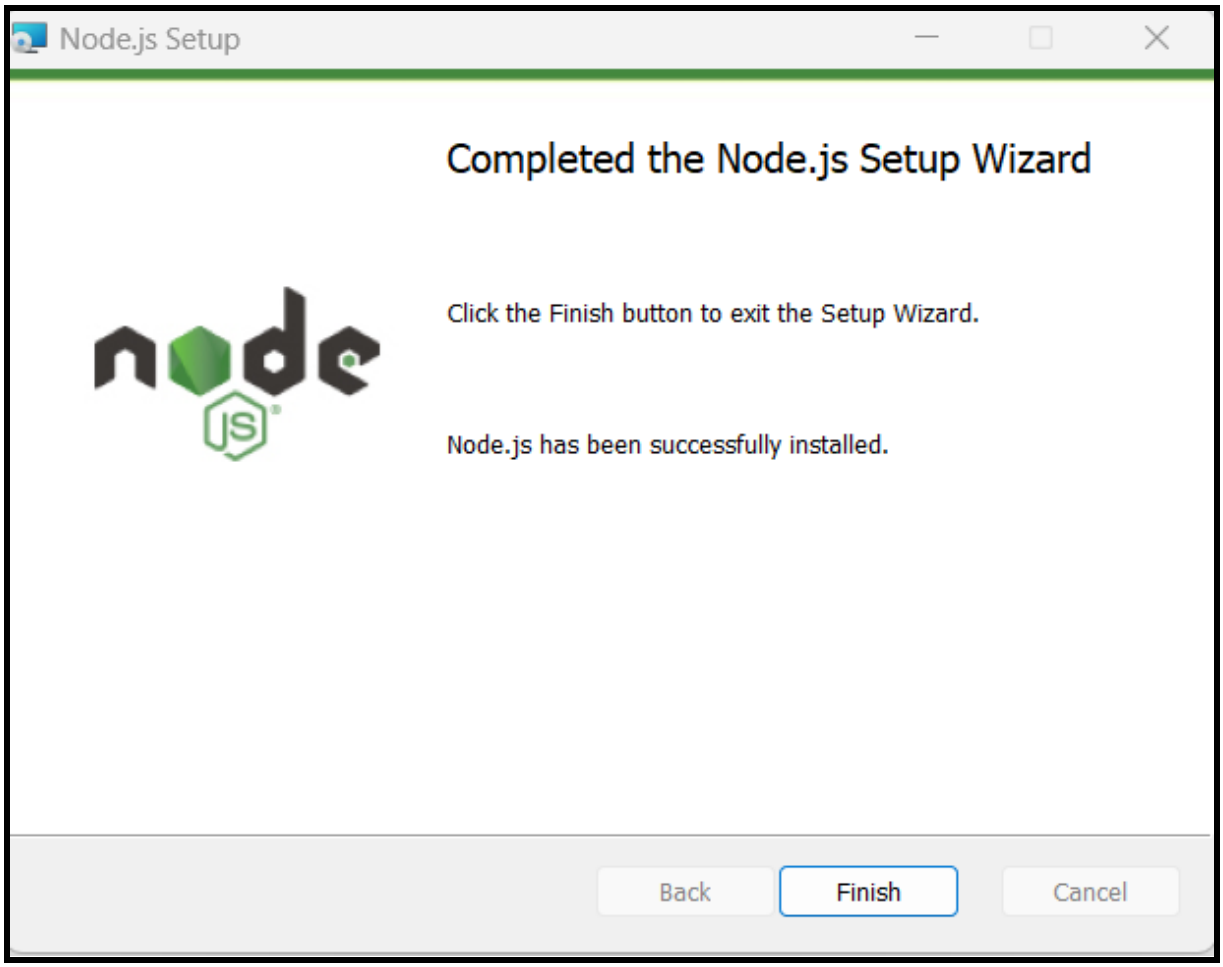
Click on Next button



Click On Next Button

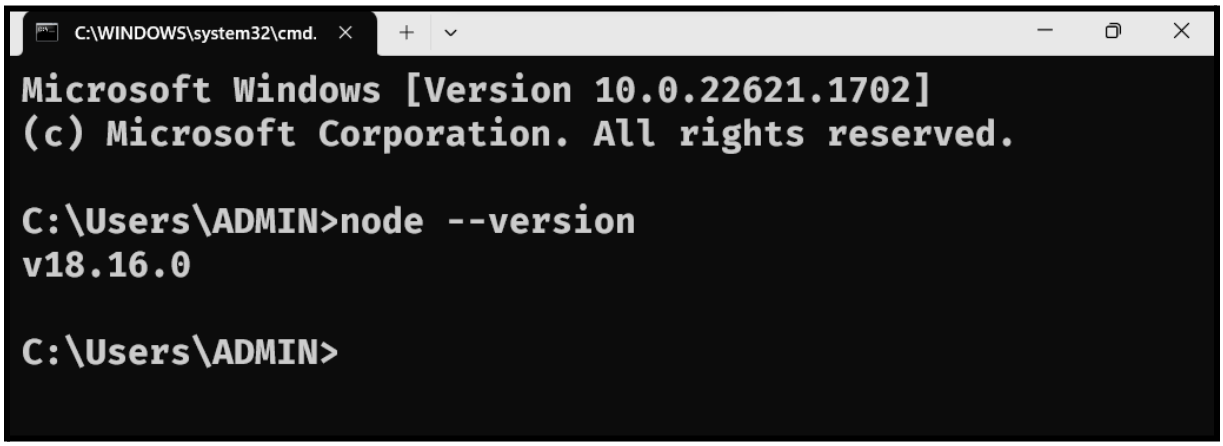


Now Click on Install button



Click on Finish Button

Once we install the nodejs we can cross check the node js by opening the command prompt.



Let’s learn about the npm which is by default installed while we install a node js.

What is NPM?

- NPM – or "Node Package Manager" – is the default package manager for JavaScript's runtime Node.js.

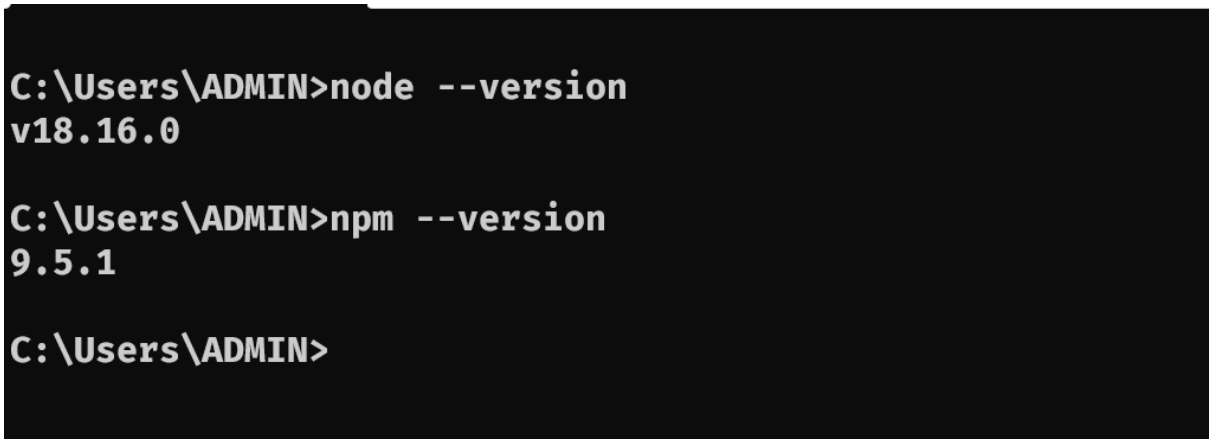
- It's also known as "Ninja Pumpkin Mutants", "Nonprofit Pizza Makers", and a host of other random names that you can explore and probably contribute to over at npm-expansions.

- **NPM consists of two main parts:**

    - a CLI (command-line interface) tool for publishing and downloading packages.
    - an online repository that hosts JavaScript packages.
      https://www.npmjs.com/

Let’s learn how to install angular in the window operating system.
The following are the below steps to install the angular:

Step1 : Check the version of nodejs, npm as below command:



Step2: Now install the angular by visiting the official website of angular
https://angular.io/docs
Step3: Open the command prompt and install the angular 10

Note : If angular is already there in your system you can uninstall by using following command:

```
C:\Users\ADMIN>npm uninstall -g angular-cli

up to date in 82ms

C:\Users\ADMIN>
```

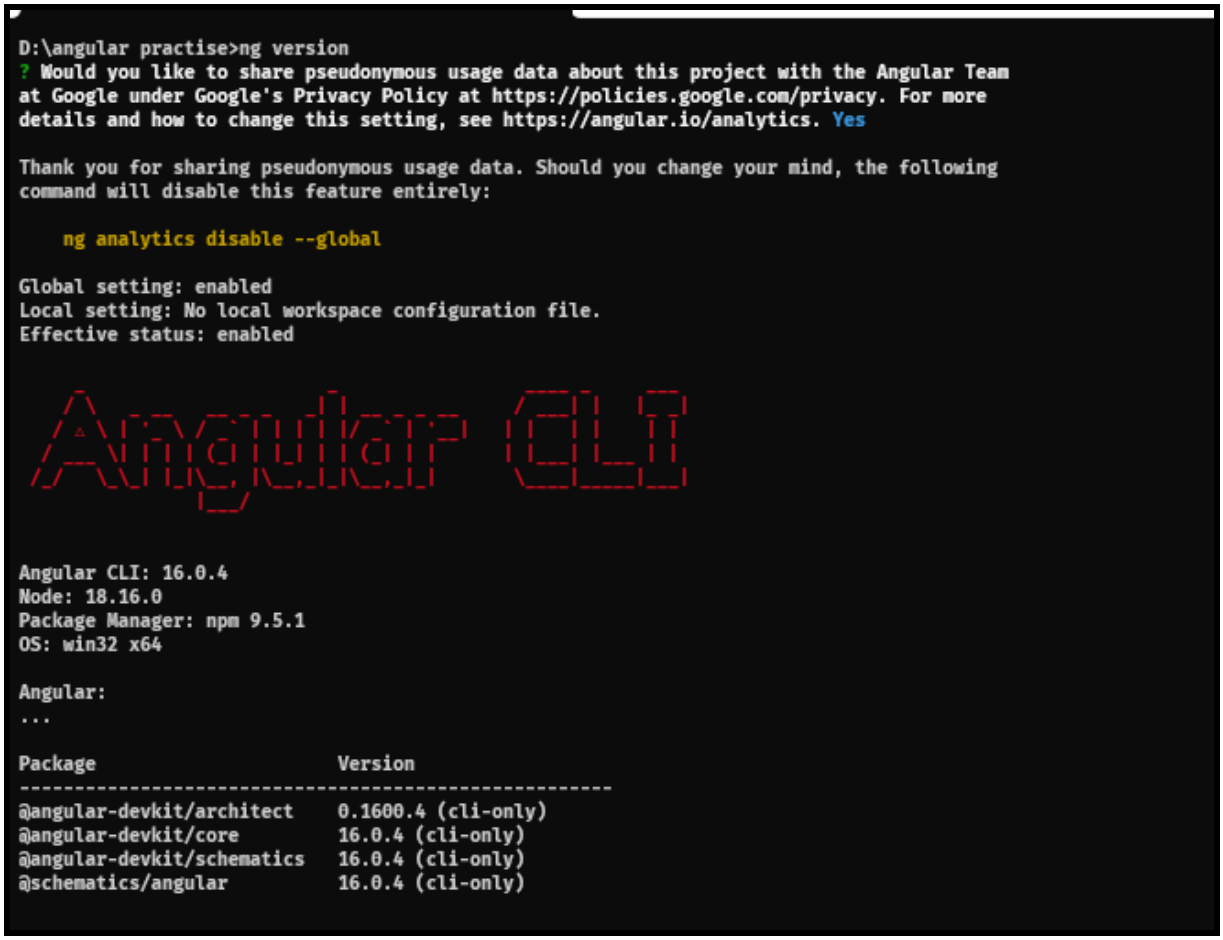
1. npm cache clean --force
2. npm cache verify

As of now we have not installed angular any version.

Let's Learn how to install

```
D:\angular practise>npm install -g @angular/cli
```

Now check the version of angular by using following command:

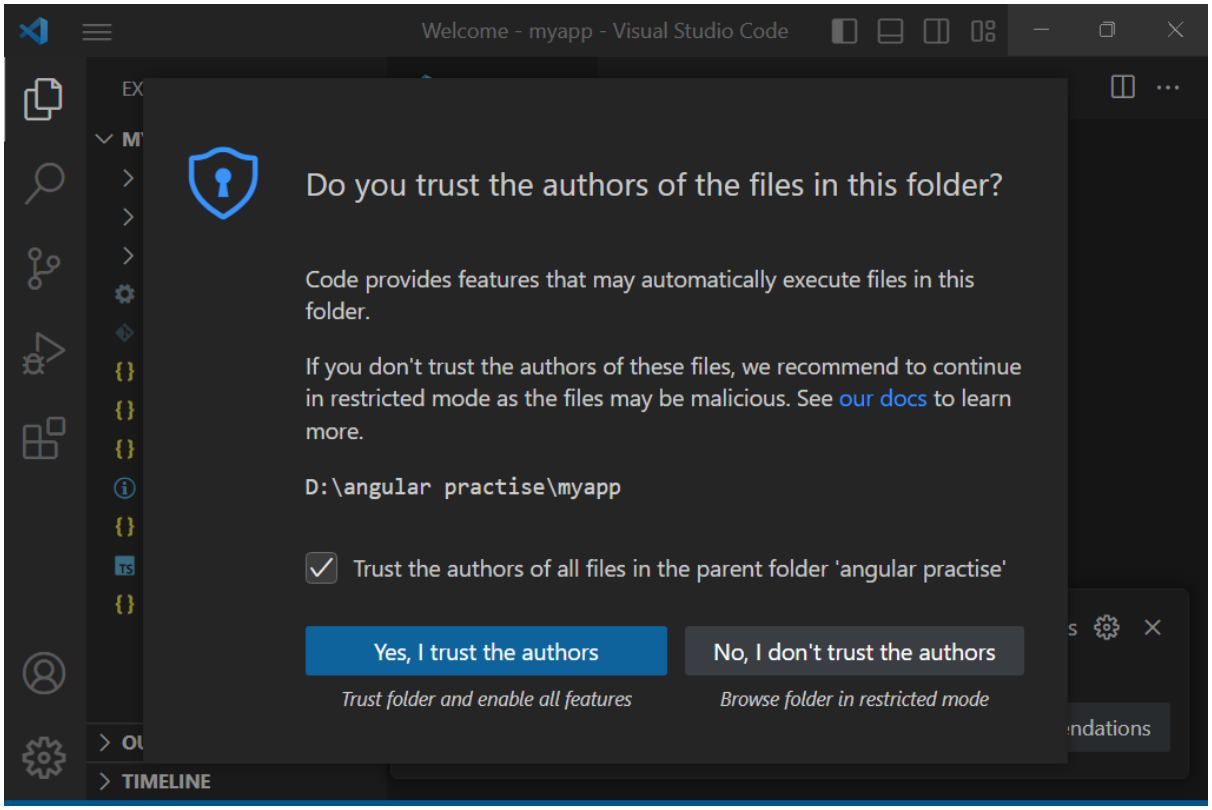


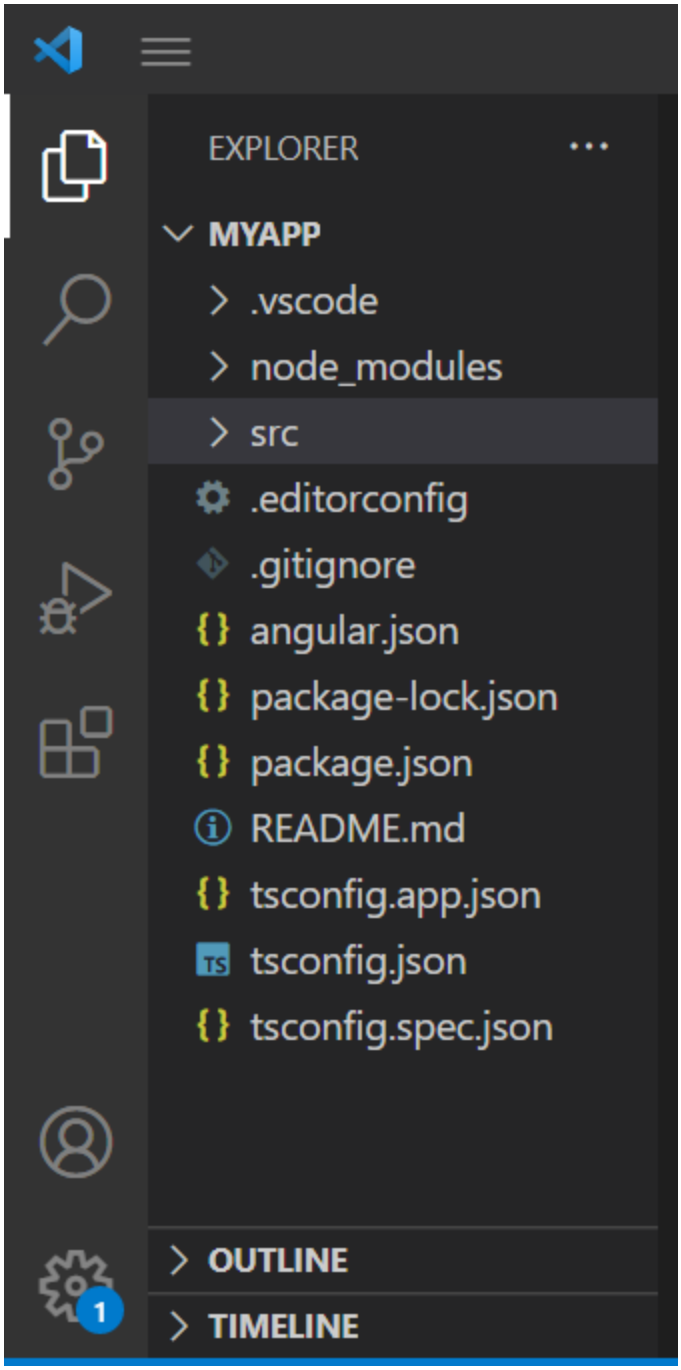
Step4: Create the new angular project i.e. myapp by using below command:

```
D:\angular practise>ng new myapp
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? CSS
CREATE myapp/angular.json (2695 bytes)
CREATE myapp/package.json (1036 bytes)
CREATE myapp/README.md (1059 bytes)
CREATE myapp/tsconfig.json (901 bytes)
CREATE myapp/.editorconfig (274 bytes)
CREATE myapp/.gitignore (548 bytes)
CREATE myapp/tsconfig.app.json (263 bytes)
CREATE myapp/tsconfig.spec.json (273 bytes)
CREATE myapp/.vscode/extensions.json (130 bytes)
CREATE myapp/.vscode/launch.json (470 bytes)
CREATE myapp/.vscode/tasks.json (938 bytes)
CREATE myapp/src/main.ts (214 bytes)
CREATE myapp/src/favicon.ico (948 bytes)
CREATE myapp/src/index.html (291 bytes)
CREATE myapp/src/styles.css (80 bytes)
CREATE myapp/src/app/app.module.ts (314 bytes)
CREATE myapp/src/app/app.component.html (23083 bytes)
CREATE myapp/src/app/app.component.spec.ts (889 bytes)
CREATE myapp/src/app/app.component.ts (209 bytes)
CREATE myapp/src/app/app.component.css (0 bytes)
CREATE myapp/src/assets/.gitkeep (0 bytes)
√ Packages installed successfully.
warning: in the working copy of '.editorconfig', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of '.gitignore', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of '.vscode/extensions.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of '.vscode/launch.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of '.vscode/tasks.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'README.md', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'angular.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'package-lock.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'package.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'src/app/app.component.html', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'src/app/app.component.spec.ts', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'src/app/app.component.ts', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'src/app/app.module.ts', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'src/index.html', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'src/main.ts', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'src/styles.css', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'tsconfig.app.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'tsconfig.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'tsconfig.spec.json', LF will be replaced by CRLF the next time Git touches it
    Successfully initialized git.
```

Step5: Now move to the myapp directory and open the project into the visual studio.

```
D:\angular practise\myapp>code .
```

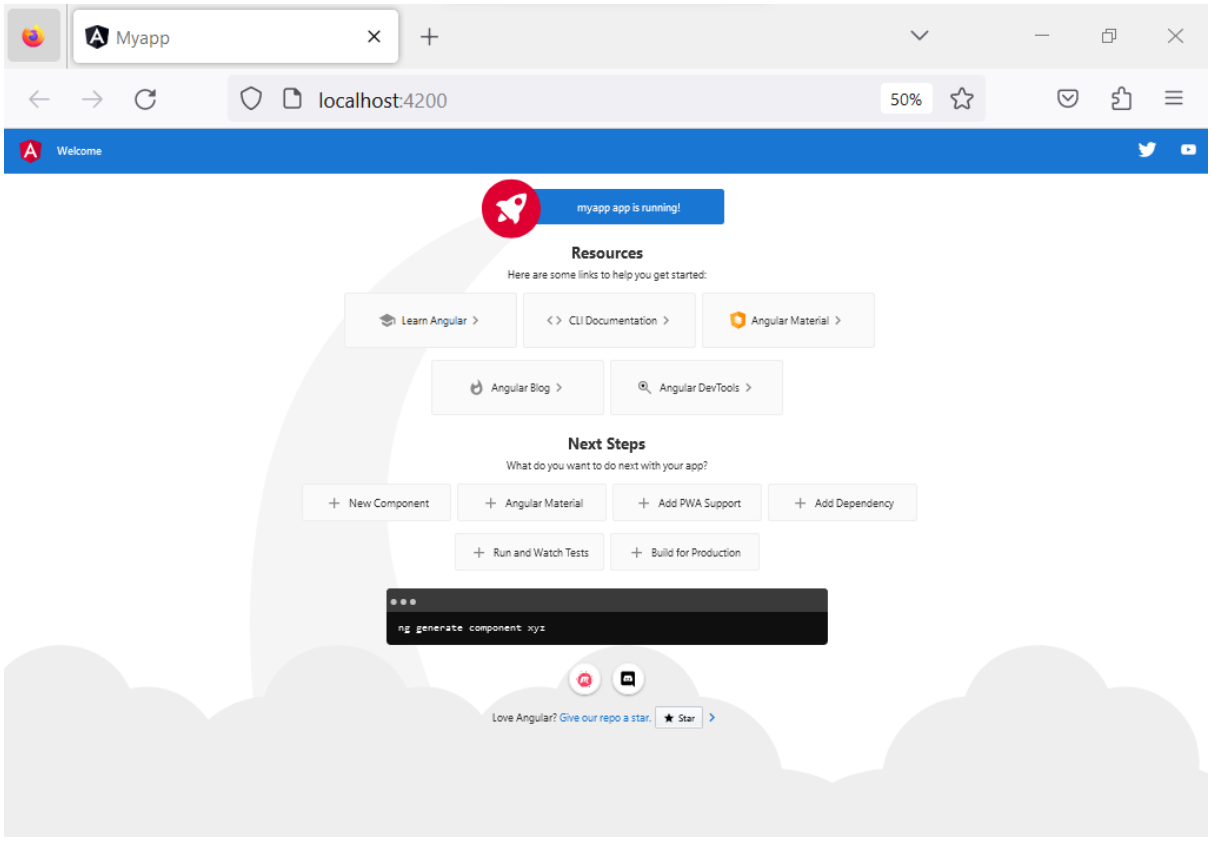

Welcome - myapp - Visual Studio Code
Do you trust the authors of the files in this folder?
Code provides features that may automatically execute files in this folder.
If you don't trust the authors of these files, we recommend to continue in restricted mode as the files may be malicious. See our docs to learn more.
D:\angular practise\myapp
Trust the authors of all files in the parent folder 'angular practise'
Yes, I trust the authors
No, I don't trust the authors
Trust folder and enable all features
Browse folder in restricted mode
TIMELINE



Step6: Open the terminal and start the angular application

```
PS D:\angular practise\myapp> npm start
```

Step7: Open into the browser and test it



DAY-3

Agenda:

- **TypeScript**
  - What is TypeScript?
  - Difference between TypeScript and JavaScript
  - Features of TypeScript
  - Version of TypeScript
  - Steps to Prepare First Example in TypeScript

TypeScript:

- It is programming language, which is developed based on JS.
- It is superset of JS, which adds data types, classes, interfaces and other features.
- It is developed by Microsoft Corporation in 2012.

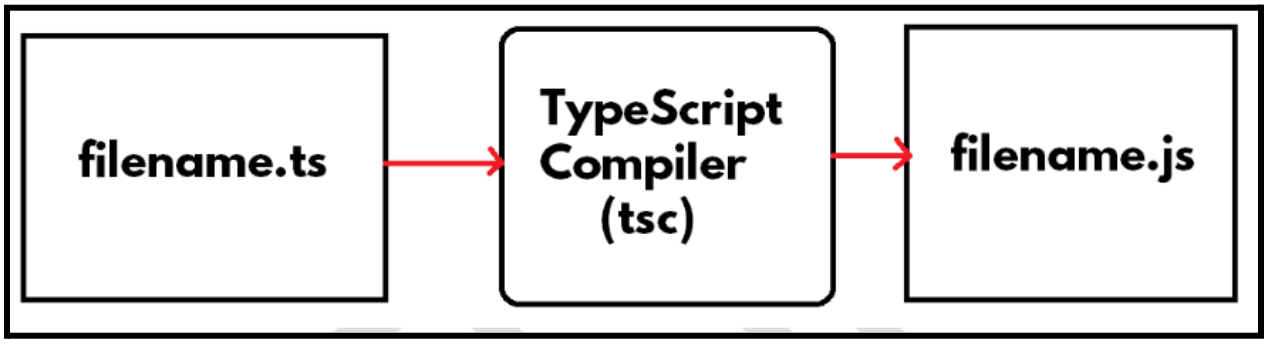
Difference between TypeScript and JavaScript

S.NO	TypeScript	JavaScript
1.	Supports static typing and allows declaring types for variables, function parameters, and return values.	Dynamically typed language where types are determined at runtime.
2.	Requires a compilation step to convert TypeScript code into JavaScript code.	No compilation step required, as JavaScript code runs directly in the browser or on a server.
3.	In Case TS we have features like interfaces, classes, modules, and generics.	It does’nt have build-in support for interfaces, classes, modules.
4.	The extension is .ts	The extension is .js
5.	It supports object-oriented programming concepts like classes, interfaces, inheritance, generics etc..	It just a scripting language.

Features of TypeScript

- **Static Typing :**
    - TypeScript allows you to explicitly declare the types of variables, function parameters, and return values. It helps catch errors during development by ensuring that values are used correctly and consistently.
- **Modules** :
    - TypeScript provides a module system that helps organize code into reusable and independent units. Modules allow you to encapsulate related functionality and control the visibility of variables and functions, making it easier to manage large codebases.
- **Interfaces** :
    - TypeScript introduces interfaces, which define the structure and shape of objects. They enable you to specify the expected properties and methods that an object should have, promoting code clarity and maintainability.
- **Classes**:
    - TypeScript supports object-oriented programming concepts like classes, inheritance, and access modifiers (e.g., public, private, protected). Classes allow you to define blueprints for creating objects with shared properties and behaviors.
- **Generics** :
    - TypeScript supports generics, allowing you to create reusable components that can work with different types. Generics provide a way to write flexible and type-safe code by abstracting over specific data types.
- **Tooling and IDE support :**

    - TypeScript offers excellent tooling and support in popular IDEs like Visual Studio Code. This includes features like autocompletion, type checking, and refactoring tools, which enhance developer productivity and help identify errors early.
- **Browser Compatability :**
    - TypeScript code can be transpiled into JavaScript that is compatible with older browsers. You can write modern JavaScript syntax and features and then convert it into a version that works in a wide range of environments.



Version History

Version	Year
0.8	2012 (first version)
0.9	2013
1.0	2014
2.0	2016
2.6	2017
2.8	2018
…	..
5.0	2023
https://en.wikipedia.org/wiki/TypeScript	

Steps to Prepare First Example in TypeScript

Step1: Install the Node.JS
- https://nodejs.org/en

Step2: Install the TypeScript
- https://www.typescriptlang.org/download

Step3: Install VS Code
Step4 : Create a folder structure
Step5 : Create the TypeScript file and write the code
Step6 : Compile the TypeScript Program
Step7: Execute the TypeScript Program

- In order to install the typescript via npm we can use the below command:
  - npm install typescript -g

```
C:\Users\MOHAMMED IMTIAZ>npm install typescript -g
```

```
C:\Users\MOHAMMED IMTIAZ>tsc -version
Version 5.1.3
```

DAY-4

Agenda

- TypeSrcript Basics
  - Variables
  - DataTypes
- TypeScript OOPS
  - Class
  - Object
  - Constructor
  - Inheritance
  - Access Modifiers
  - Interface
  - Enumeratation
  - Moudles

TypeScript Basics

Variables

- Variable is a named memory location in RAM, to store a value at run time.
  - **Syntax**: var variable : datatype = value;
  - **Example:** var a : number = 100;
- TypeScript supports “optional static typing”. That means it is optional to specify datatype
- for the variable in TypeScript.
  - S**tatic Typing:**
    - If you specify the data type for the variable, it is not possible assign other data type of value into the variable; if you do so, “tsc” compiler will generate coding-time and compile-time errors; but the code will be compiled and executed also, even though it is having errors.

    - **Dynamic Typing:** If you don't specify the data type for the variable, we can assign any type of value into the variable.
- In TypeScript, we have "optional static typing". That means it is optional to specify datatype for the variable in TypeScript

Data Types

- **The following are the list of DataTypes in TypeScript:**
    - number : All types of numbers(integer, floating)
    - string : Collection of characters in double quotes or single quotes
    - boolean : true or false
    - any : Any type of value

Object

- Object is the primary concept in OOP (Object Oriented Programming).
- "Object" represents a physical item, that represents a person or a thing.
- Object is a collection of properties (details) and methods (manipulations).
- For example, a student is an "object".
- We can create any no. of objects inside the program.

Property

- Properties are the deails about the object.
- Properties are used to store a value.
- For example, **studentname="Scott"** is a property in the "student object".
- Properties are stored inside the object.
- The value of property can be changed any no. of times

Method

- Methods are the operations / tasks of the object, which manipulates / calculates the data and do some process.
- Method is a function in the object.
- Methods are stored inside the object.

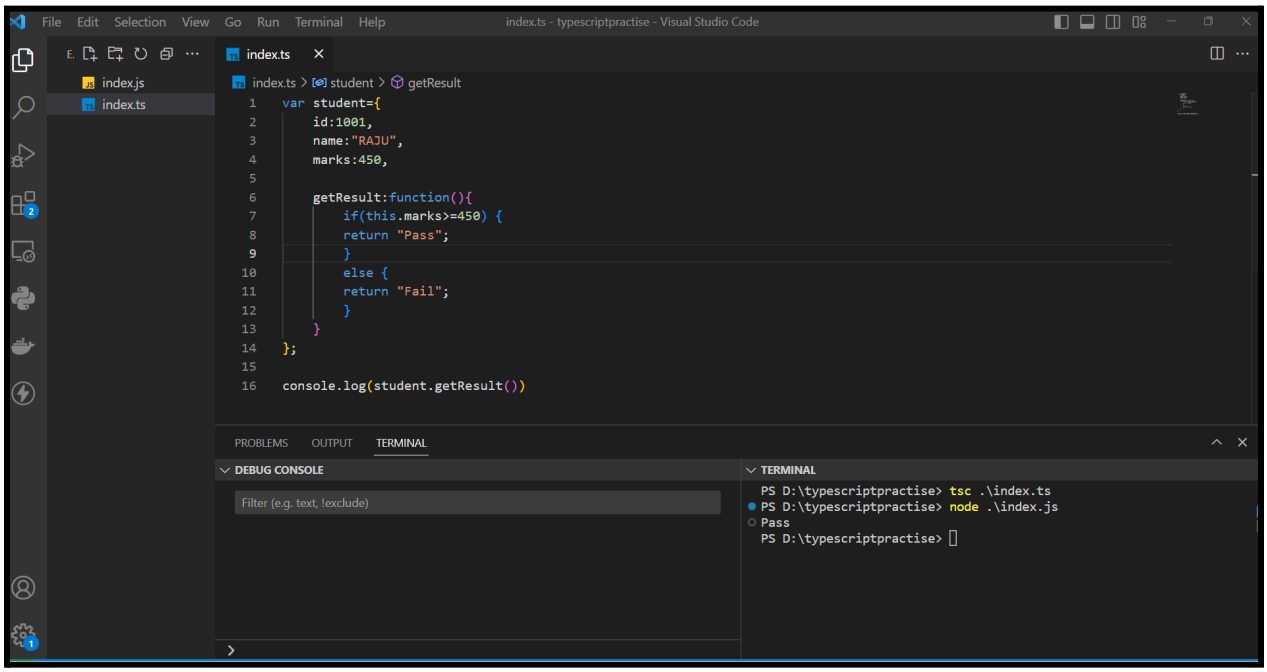
Creating Object in TypeScript

- Object can be created by using "Object Literal Pattern" in TypeScript.
- Object literal" is a collection of properties and methods, that are enclosed within curly braces { }.
- **Syntax** to create Object: { property: value, …, method: function( ) { code here } }

Reference Variable

- The "reference variable" is a variable, that can store the reference of an object.
- We can access all the properties and methods of the object, by using the "reference
- variable".
- Syntax to create Object and store its reference in the "reference variable":
- var referenceVariable = { property: value, …, method: function() { code here } };

Example :

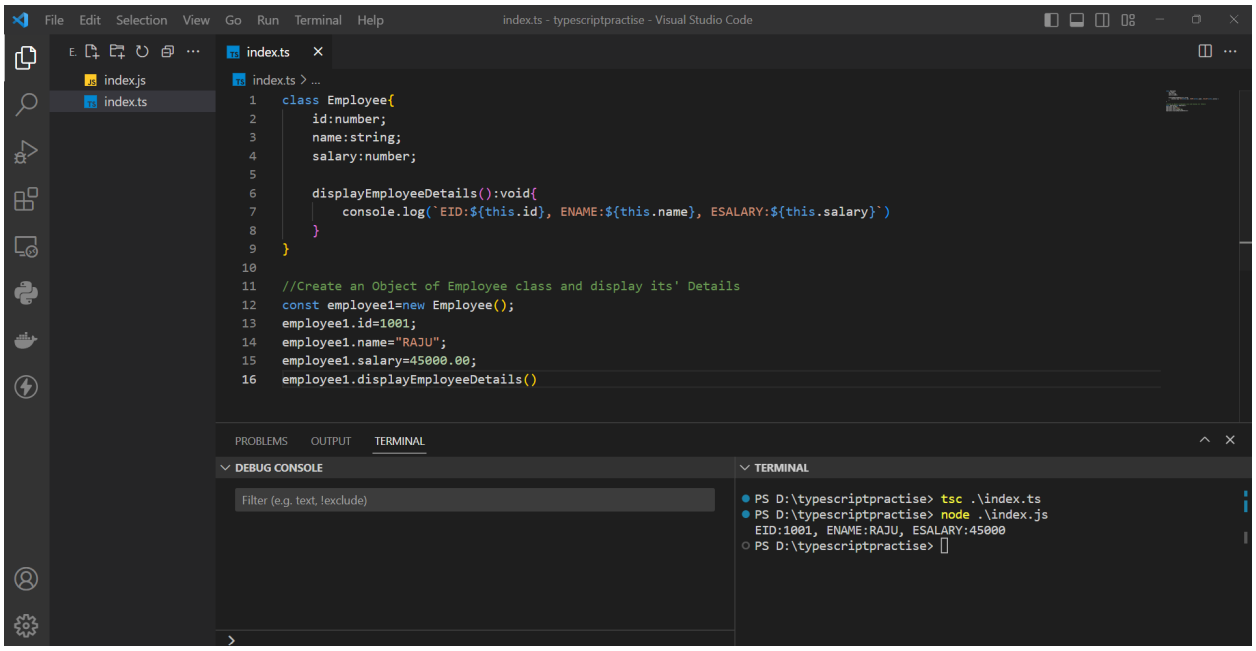


Class

- It is represent a model of the object, which defines list of properties and method of the object.
- Example : " Employee " class represents structures(list of properties and methods) of every Employee object.
- We can create any no of objects based on a class by using a "new" keyword.
- All the objects of a class, shares same set of properties and method of the class.
- We can store the reference of the object in "reference variable"; using which you can access the object.
- Syntax:

```
class className{
  //properties
  //methods
}
```

Example:



==========

Constructor

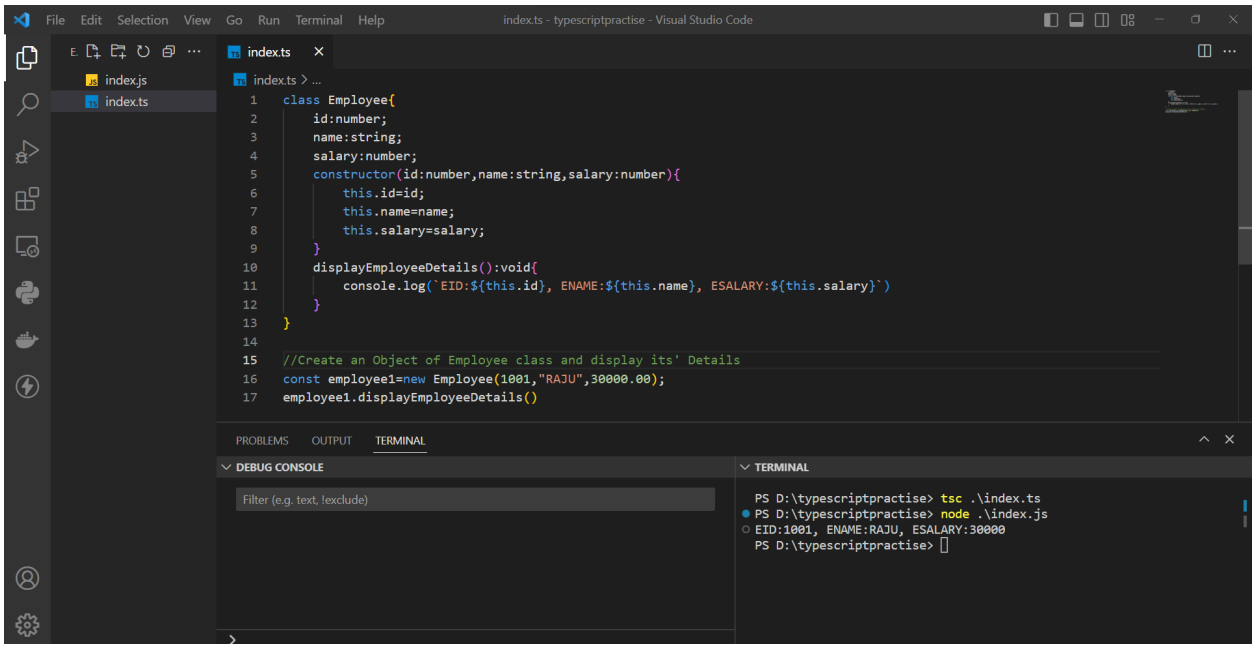
- It is a special function which is a part of the class.
- It will invoked automatically when we create an instace of the class.
    - Note : When we create the multiple instance of the particular class every time it will invoke the constructor.
- It is basically used to initialize the properties of the class.
- We use the name as “constructor”.
- Constructor can take any no of arguments but not the return type.
  Note : We can’t define multiple constructor in TypeScript.

Syntax:

constructor(arg1:dataType,arg2:dataType,arg3:dataTyp,..){

}

Example:



Inheritance

- Acquiring the properties from parent class to child class is called as Inheritance.
- The main advantage of inheritance is code reusability.

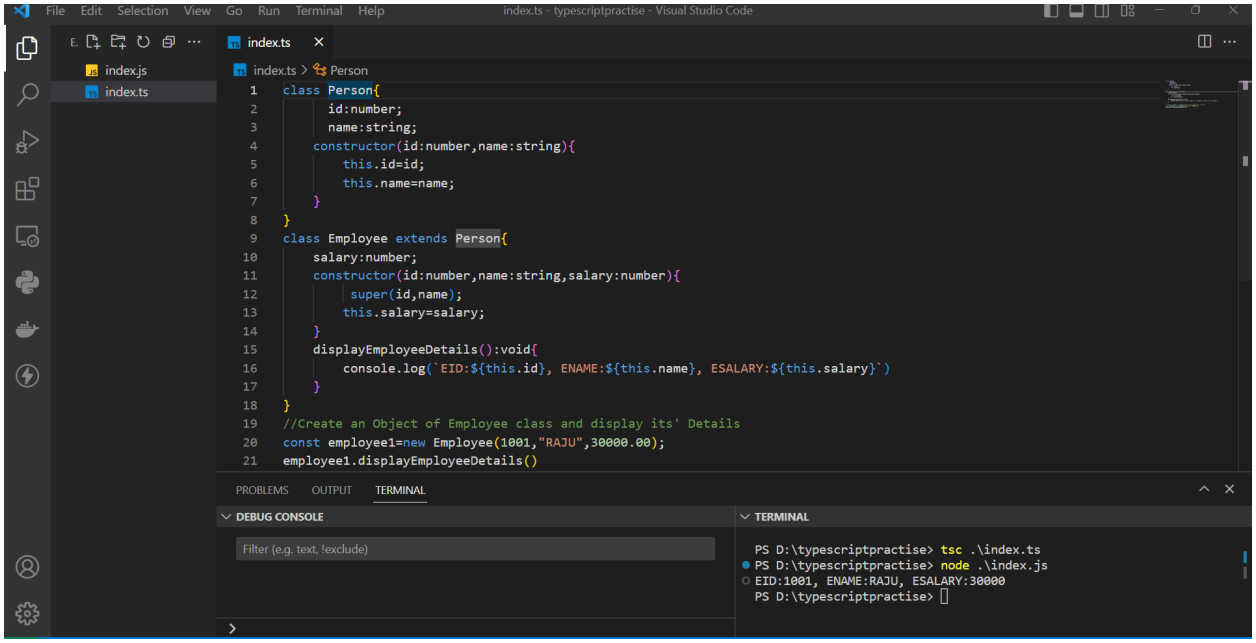
Key Points:

- ☐ The ‘extends’ keyword is used to create inheritance.
- ☐ The ‘super’ keyword is basically used to access the members of parent class from child class.
- ☐ When Parent class has a constructor, then child class has to invoke explicitly by using super(..)

- Example : Employee extends Persone , Batsmen extends Player, ICICIBank extends Bank etc…
- Syntax :
  - class parentClass{
    //parent class members
    }

    class childClass extends parentClass{
    // child class members
    }

Example:



Access Modifiers

- It is used to specify where the member of a class can be accessible. That means it specifies whether the member of a class is accessible outside the class or not.
- Access Modifiers are used to implement "security" in OOP.
- Each member (property / method), we can specify the access modifier separately.
- TypeScript compiler and Visual Studio Code Edtitor shows errors, if a developer try to access the member, which is not accessible.
- "public" is the access modifier for all the members (property / method) in TypeScript class.

  TypeScript supports three access modifiers:

  1. public
  2. private
  3. protected

1. **public**: Public members can be accessed from anywhere, both within the class and outside the class.

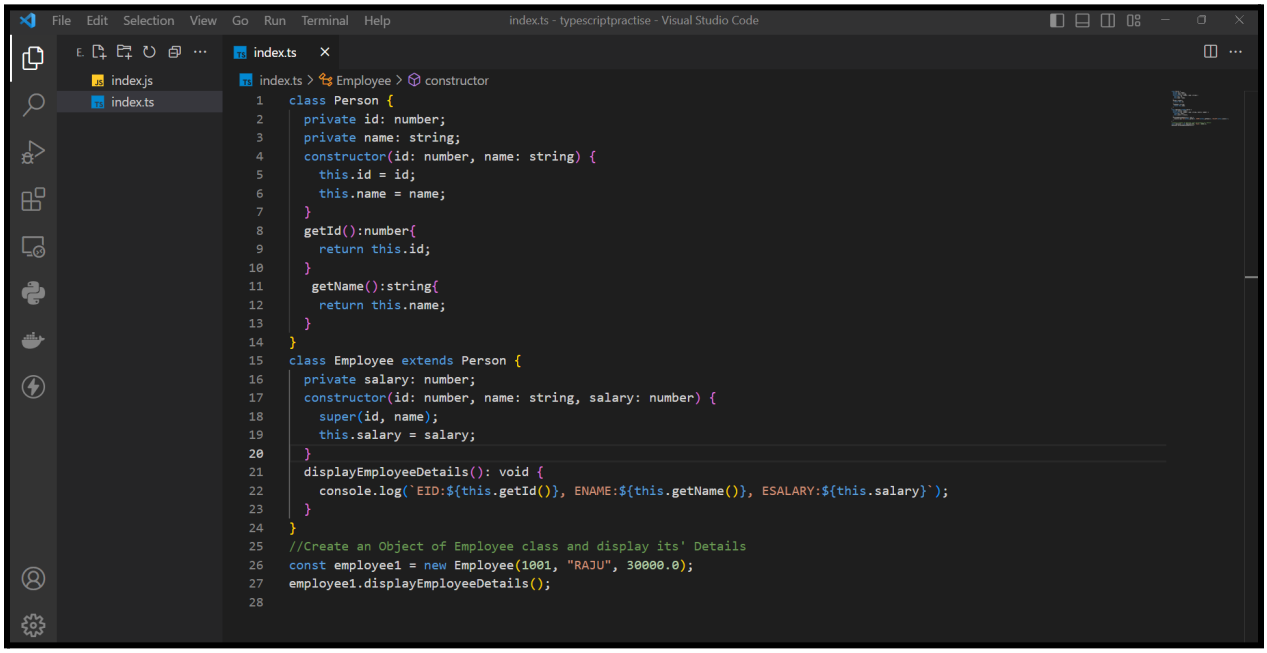
2. **private**: Private members can only be accessed within the class in which they are defined. They are not accessible from outside the class or from derived classes.

3. **protected**: Protected members can be accessed within the class and in derived classes. They are not accessible from outside the class.

Syntax:

```
class className{
    accessModifier property:dataType;

    accessModifier methodName(lis_of_args):returnType{

    }

}
```

Example:



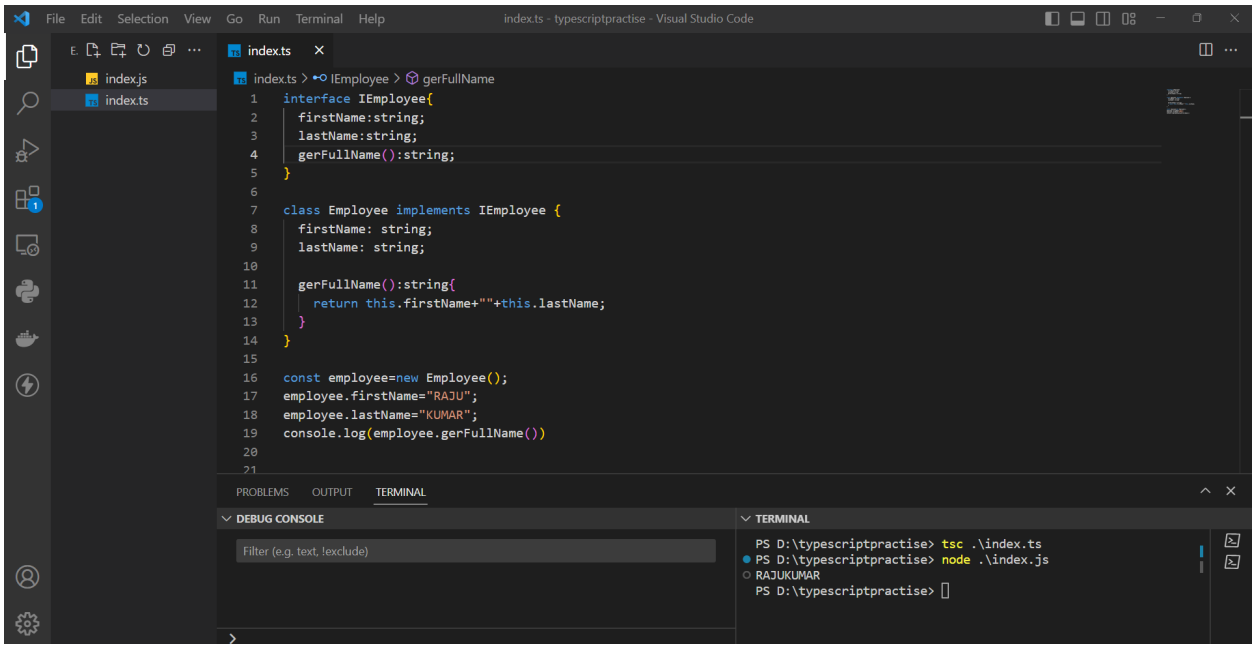
Interfaces

- Interface is the model of a class, which describes the list of properties and methods of a class.
- Interfaces doesn't contain actual code; contains only list of properties and methods.
- Interfaces doesn't contain method implementation (method definition); it contains method declaration only, which defines method access modifier, method name, method arguments and method return type.
- The child class that implements the interface must implement all the methods of the
- interface; if not, compile-time error will be shown at child class. The method name,
- parameters, return type and access modifier must be same in between "interface method (method at the interface)" and "class method (method at the class)".
- Interfaces act as a mediator between two or more developers; one developer implements the interface, other developer creates reference variable for the interface and invokes methods; so interface is common among them.
- The child class can implement the interface with "implements" keyword.
- All the methods of interface is by default, "public".
- One interface can be implemented by multiple classes; One class can implement multiple interfaces.
- We can develop "multiple inheritance" by implementing multiple interfaces at-a-time in the same class.

Syntax:

```
interface interfaceName{
    property:dataType;
    method(list_of_arg):returnType;
}
```

Example :



Enumerations:

- Enumeration is a collection of constants.
- Enumeration acts as a data type.
- We can use "enumeration" as a data type for the "enumeration variable" or "enumeration
- property".
- The enumeration variable or enumeration property can store any one of the constants of
- the same enumeration.
- Every constant of enumeration is represented with a number (starts from 0)

#Steps for creating enumeration

Step1: create the enumeration

enum enumeartionName{
  constant1,constant2,....;
}

Step2: Creating the property of enumeration type

class className{
 property:enumerationName;
}

Step3: Create a variable of enumeration type

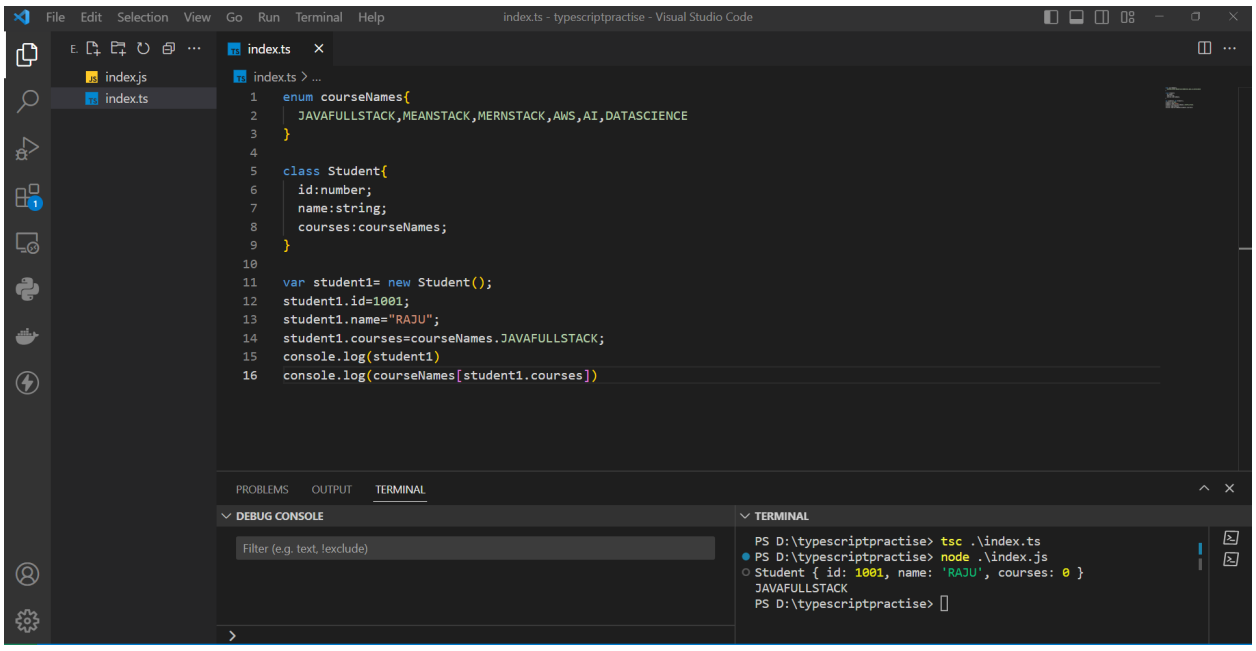
 variableName : enumerationName;

Step4: Assign the value into enumeration variable or enumeration property

enumerationVariable = enumerationName.constant;

this.enumerationProperty=enuermationName.constant;

Example:



DAY-5

Agenda

- Modules
- NodeJS
    - What is NodeJS
    - Working with NodeJS Modules
        - FS(file system)
        - HTTP
    - Working with Express JS
- Understanding of Angular Project Structure

Modules:

- In large scale applications, it is recommended to write each class in a separate file.
- To access the class of one file in other file, we need the concept of "Modules".
- Module is a file (typescript file), which can export one or more classes (or interfaces,
- enumerations etc.) to other files; The other files can import classes (or interfaces,
- enumerations etc.), that are exported by the specific file. We can't import the classes (or
- others) that are not exported.
- We can export the class (or others), by using "export" keyword in the source file.
- We can import the class (or others), by using "import" keyword in the destination file.
- To represent:
    - current folder, we use "./".
    - sub folder in current folder, we use "./subfolder".
    - parent folder, we use "../".

Steps for development of modules

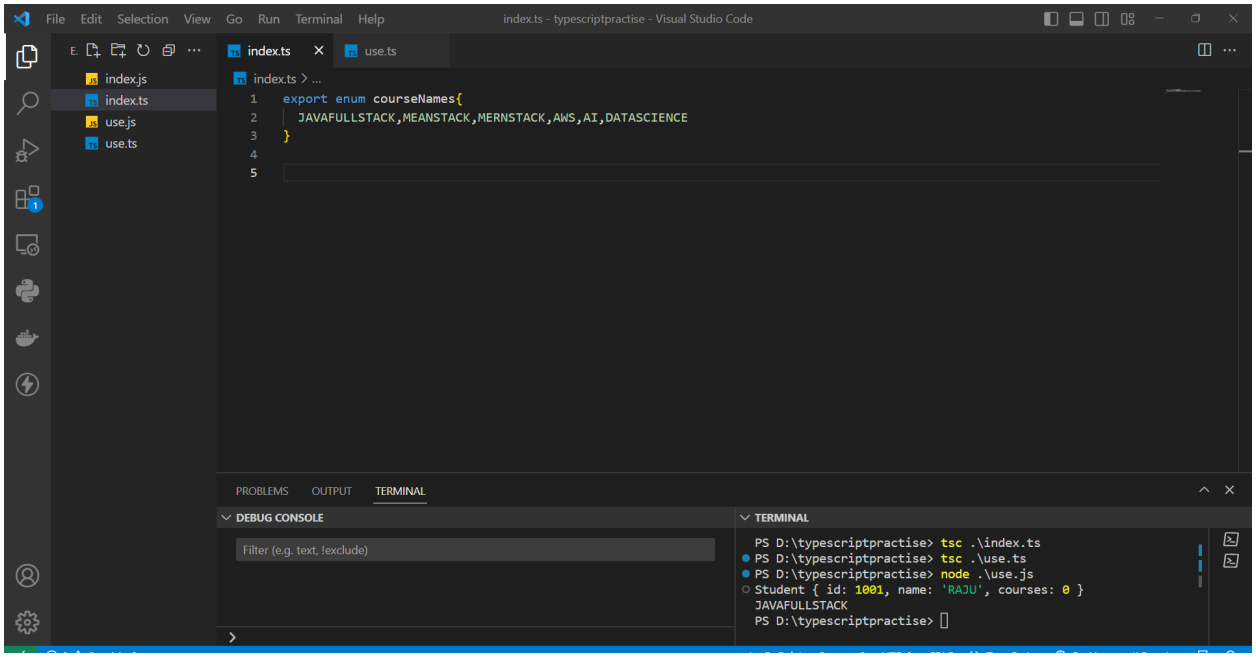
Step1: export a class file1.ts

```
 export class className{
…
}
```

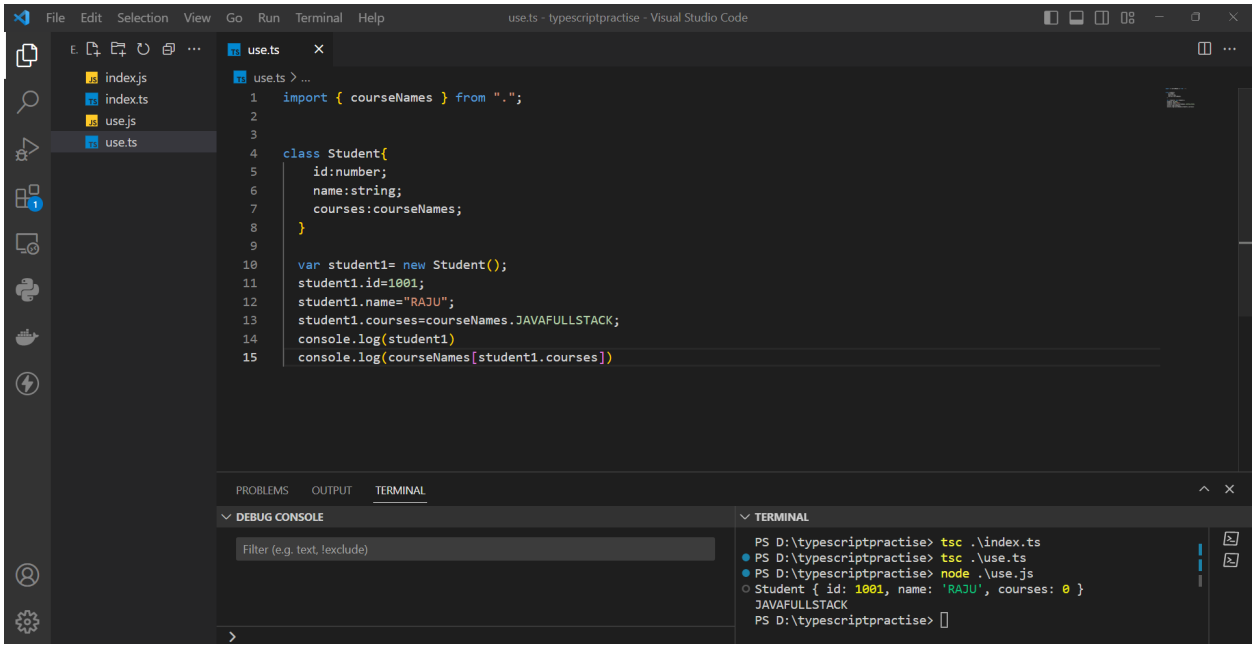
```
 import {className} from ‘./file1.ts’;
class className{
…
}
```

Example

Step1 : index.ts



Step2: use.ts


File Edit Selection View Go Run Terminal Help
use.ts - typescriptpractise - Visual Studio Code
index.js
index.ts
use.js
use.ts
use.ts
use.ts > ...
1 import { courseNames } from ".";
2
3
4 class Student{
5 id:number;
6 name:string;
7 courses:courseNames;
8 }
9
10 var student1= new Student();
11 student1.id=1001;
12 student1.name="RAJU";
13 student1.courses=courseNames.JAVAFULLSTACK;
14 console.log(student1)
15 console.log(courseNames[student1.courses])
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
Filter (e.g. text, !exclude)
TERMINAL
PS D:\typescriptpractise> tsc .\index.ts
PS D:\typescriptpractise> tsc .\use.ts
PS D:\typescriptpractise> node .\use.js
Student { id: 1001, name: 'RAJU', courses: 0 }
JAVAFULLSTACK
PS D:\typescriptpractise>

Node.JS

- NodeJS is a cross platform runtime environment and library for running JS application outside the browser.
- NodeJS is not a framework.
- Basically it is used for creating server-side and networking web applications.
- Many of the basic modules of Node.js is written in JS.

Working with Node.js

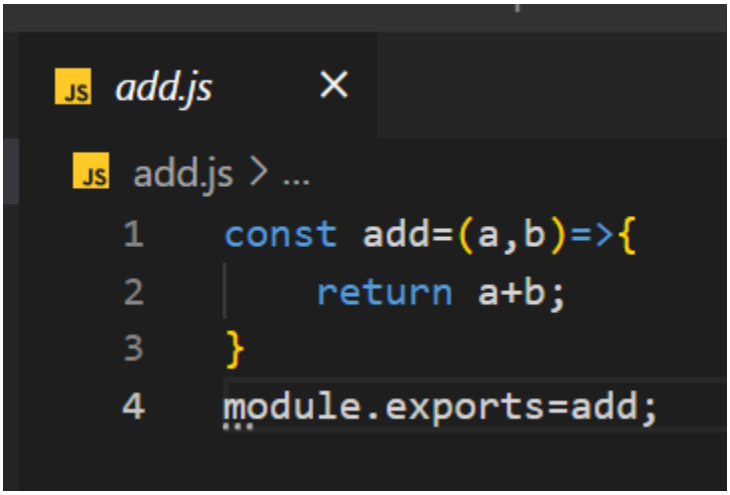
Step1: Install the node.js

https://nodejs.org/en/download

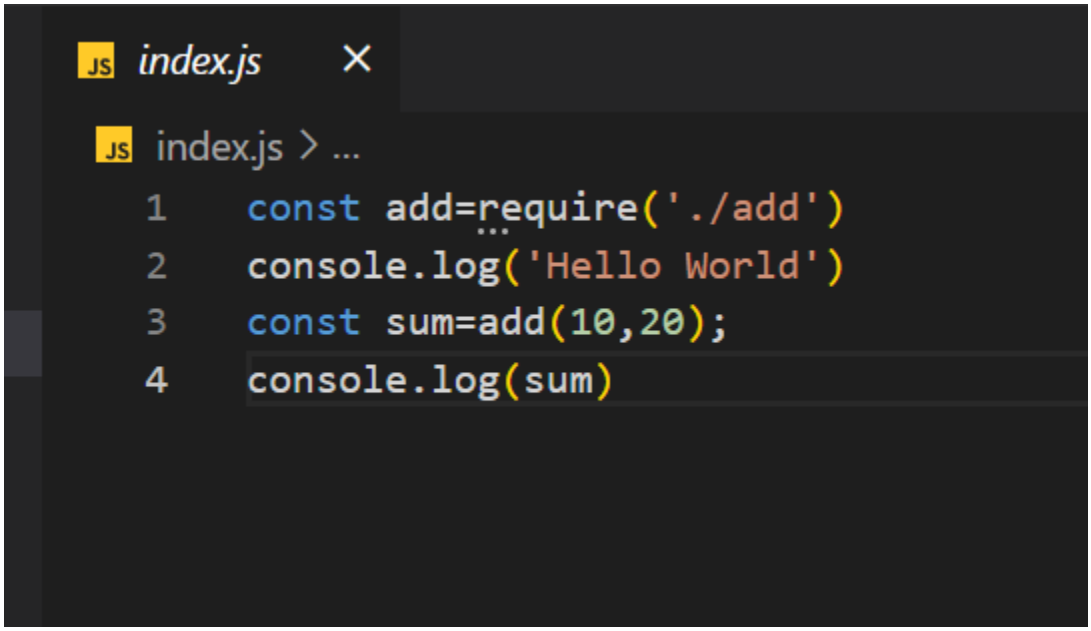
Step2: Create the project structure like a folder and create index.js file

Step3: Inside the index.js file write the below code:

I.add.js



II.index.js



Node.js File System(FS)

- In Node.js, file I/O is provided by simple wrappers around standard POSIX functions. Node File System (fs) module can be imported using following syntax:
- var fs = require('fs');

Note: Each method in fs module has synchronous and asynchronous forms.

1. Example Program by using synchronous

index.js

```
const fs=require('fs')
//1. By using Synchronous
var data = fs.readFileSync('myfile.txt');
console.log(data.toString())
```

2. By using Asynchronous

Step1 : Create the myfile.txt

- Add some text

Step2: Create the index.js file

index.js

```
const fs=require('fs')
//1. By using Synchronous
var data = fs.readFile('myfile.txt',callback)

function callback(err,data){
    if(err)
    console.err('some error')
    else
    console.log(data.toString())
}

```

Node.jS Http

- In order to make HTTP requests in Node.js, there is a built-in module HTTP in Node.js to transfer data over the HTTP. To use the HTTP server in the node, we need to require the HTTP module. The HTTP module creates an HTTP server that listens to server ports and gives a response back to the client.
- syntax:
- const http=require('http')
- Example:

workwithhtttp.js

workwithhtttp.js > server > http.createServer() callback

```
1   const http=require('http')
2   const server=http.createServer((req,res)=>{
3       console.log('http://localhost:5000/');
4       if(req.url=='/home'){
5           res.write("Welcome to Home Page")
6       }else if(req.url=='/') {
7           res.write("WELCOME TO HOME PAGE...");
8       }else{
9           res.write(`
10           <h1>Wrong ULR</h1>
11           <a href='/'>Click Here to Home</a>
12          `)
13      }
14      res.end();
15  })
16  server.listen(5000)
```

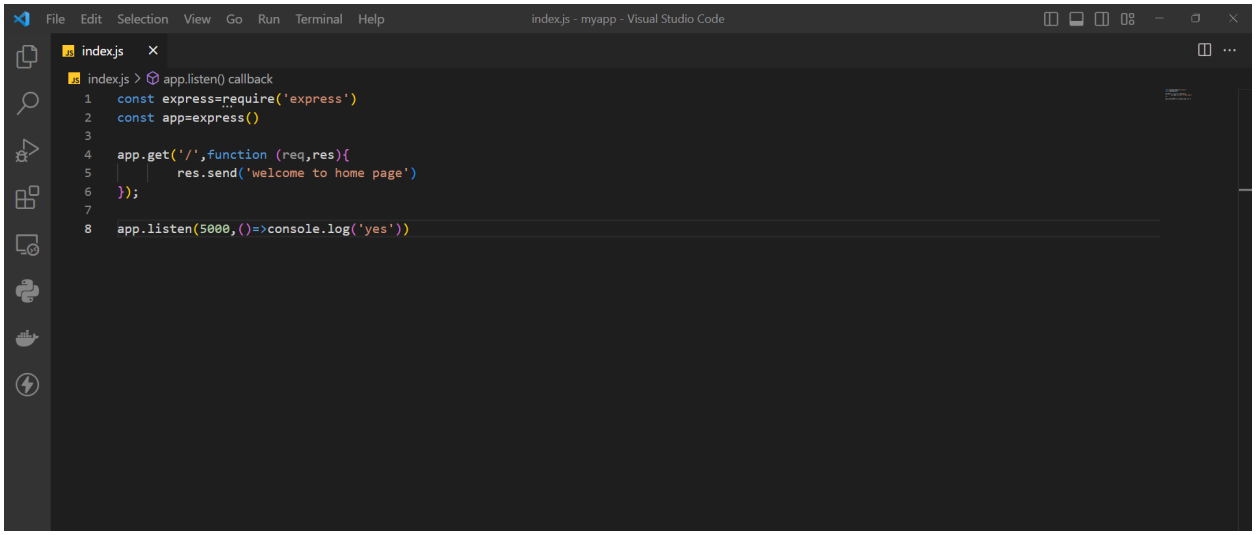
Express.js

- It is a small framework that works on top of Node.js web server functionality to simplify its APIs and add helpful new features. It makes it easier to organize your application's functionality with middleware and routing. It adds helpful utilities to Node.js HTTP objects and facilitates the rendering of dynamic HTTP objects.

Why We need Express.js

- Develops Node.js web applications quickly and easily.
- It's simple to set up and personalise.
- Allows you to define application routes using HTTP methods and URLs.
- Includes a number of middleware modules that can be used to execute additional requests and responses activities.
- Simple to interface with a variety of template engines, including Jade, Vash, and EJS.
- Allows you to specify a middleware for handling errors.

Example :
Step1: Install the express.js
> npm init
> npm i express —save
Step2: Create the index.js file



DAY-6

AGENDA

- Fundamental of Angular
  - **Building Blocks of Angular**
  - **Angular Architecture**
  - **Steps to Prepare First Angular Application.**
    - Creating Application Folder
    - Creating @angular/cli package
    - Creating new angular application
    - Open the angular application in vs code
    - Modify app.component.html
    - Run the application
  - **Folder Structure of Angular Application**
    - package.json
    - packages of angular
    - tsconfig.json
    - protractor.config.js
    - karma.conf.js
    - angular-cli.json
    - src/style.css
    - src/index.html
    - src/main.ts
    - src/app/app.module.ts
    - src/app/app.component.ts
    - src/app/app.component.html
    - src/app/app.component.css
    - src/app/app.component.spec.ts
    - Etc..

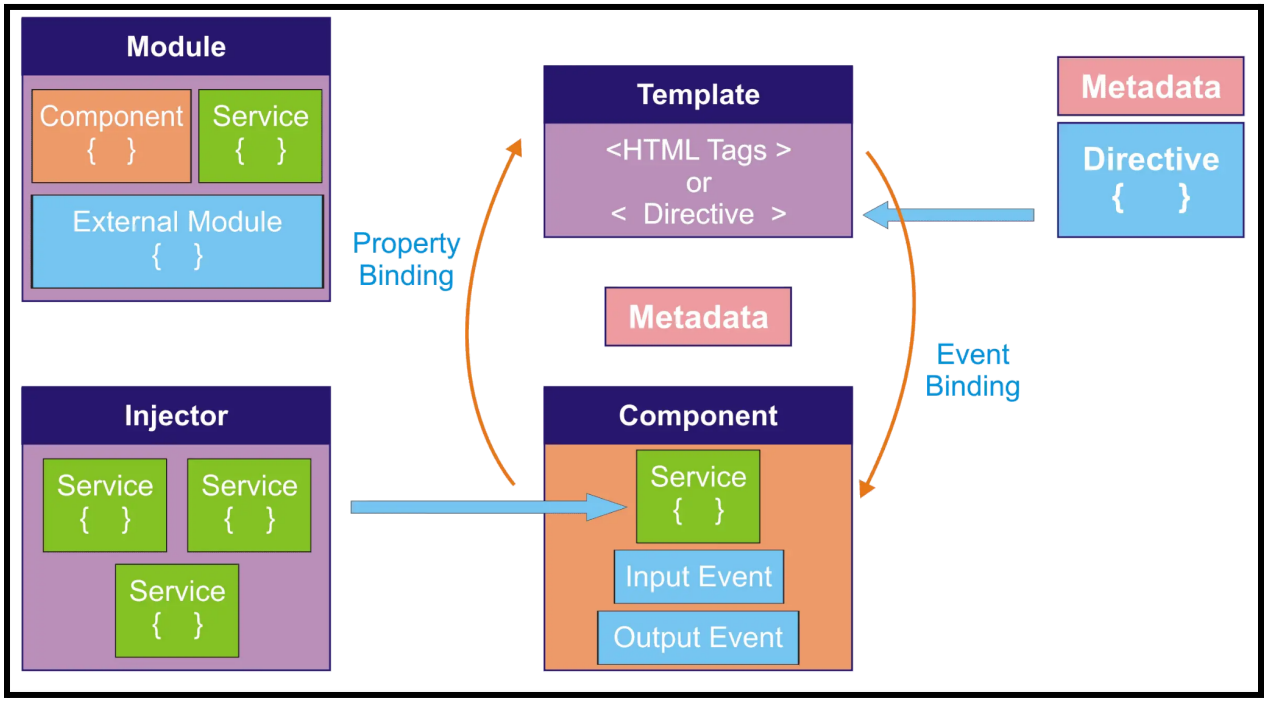
Fundamental of Angular

Building Blocks of Angular

Angular is composed with the following “building blocks”:

S.NO	Building Blocks	Description
1.	Component	Application State + Application Logic.
2.	Metadata	Details about the component / module etc.
3.	Template	Design Logic(HTML)
4.	Data Binding	Connection between HTML element and Component property
5.	Module	Group of Components, Directives, and Pipes.
6.	Service	Reusable Code / Business Logic
7.	DI	Injecting (Loading) Service Objects Into the Components.
8.	Directive	Manipulating DOM elements.
9.	Pipe	Transforming values before displaying

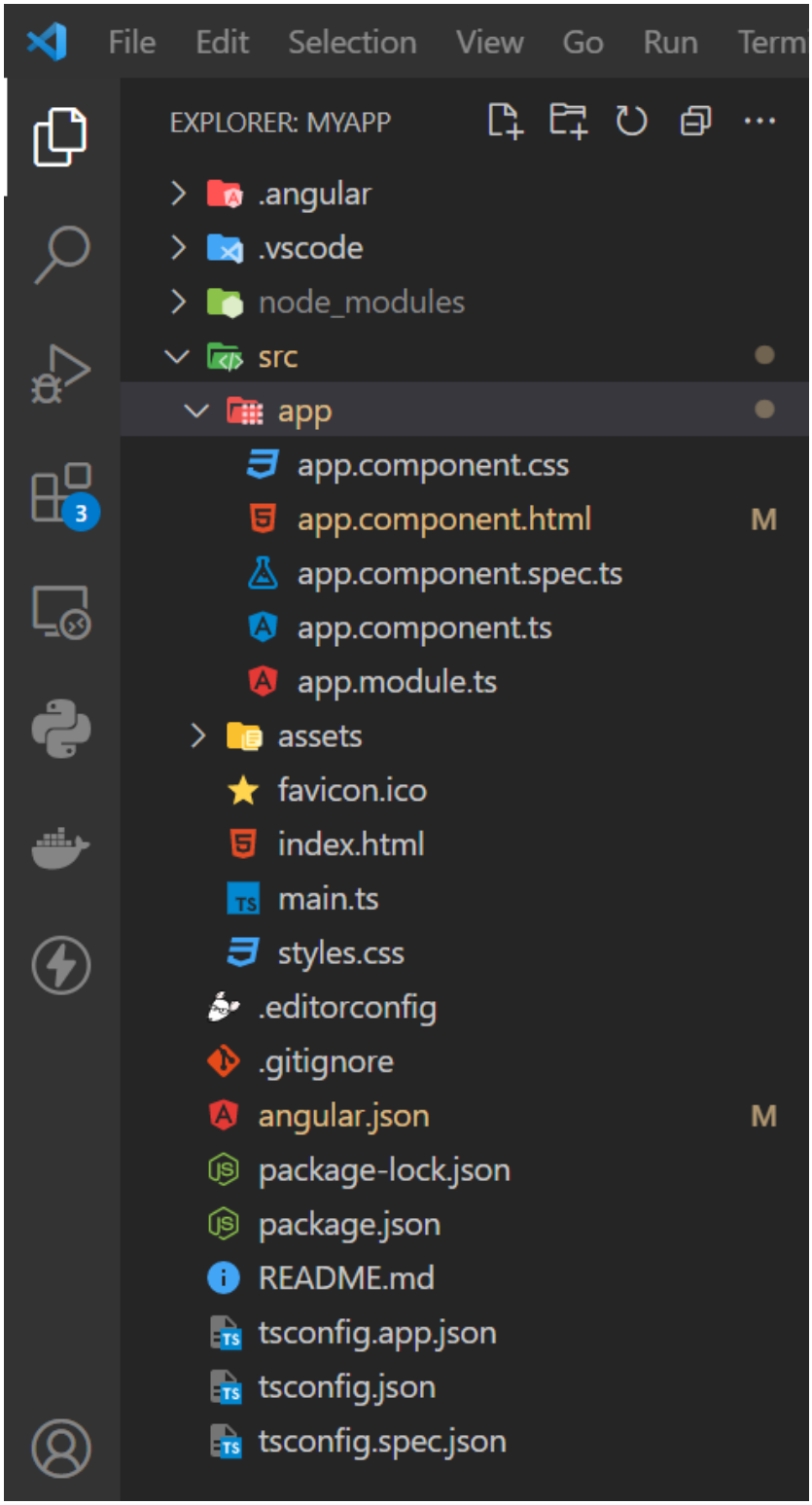
Angular Architecture



Steps To Create the First Angular Application

Step1: Install the NodeJS
Step2: Install the TypeScript
Step3: Install the VS Code
Step4: Create the application folder
Step5: Install the @angular/cli package
Step6: Create the new angular application
Step7: Open the Application in VS Code
Step8: Edit the app.component.html file
Step9: Execute the Code
Step10: Open the browser and test it.

Folder Structure of Angular Application



1. ## package.json

The “package.json” file represents the configuration settings / meta data of the application.
• It specifies package name, version, dependencies etc.
• It is a fixed filename.
• It is must, without which the application is not accepted.
• It is a JSON files, which means it contains key/value pairs. Every key and value must be within
double quotes (“ “) / single quotes (‘ ‘).

Properties of “package.json”

1 name Represents name of the application.

- It can be maximum of 214 characters.
  Non-URL friendly characters such as /, :, @ etc., are not allowed.
  Ex: “name”: “app1”

2 version Represents version of the application.
Ex: 1.0.0

- It should have 3 numbers major version, minor version, subminor version.
  Ex: “version”: “1.0.0”

3. license Represents license of the application.
Ex: MIT, ISC

- MIT:
  MIT stands for “Massachusetts Institute of Technology”.
  MIT license allows to create private applications that can be used either privately within the organization and also can be shared with other known organizations.
  Ex: “license”: “MIT”
- ISC:
  ISC stands for “Internet Systems Consortium”.
  ISC license allows to create public applications that can be used anywhere.
  Ex: “license”: “ISC”

4 scripts : Represents a set of commands that can run on the Command Prompt to run, test the applications using commands.
Ex: “scripts”:

{
“start”: “ng serve”
}

5 private Represents whether the application should be used privately within the same organization or not. If it is true, it can be used only within the same organization. Outside usage is not permitted.

Ex: “private”: true

6 dependencies Represents the list of packages that are to be installed to run the application. These packages will be installed in both developer machine and production server.

Ex: “dependencies”:
{
“@angular/core”: “^5.2.0”
}

7 devDependencies Represents the list of packages that are to be installed to develop the application. These packages will be installed only in the developer machine; not in the production server.

Ex: “devDependencies”:
{
“@angular/cli”: “~1.7.4”
}

Angular Packages:

1 @angular/core

This package provides classes and interfaces that are related to decorators, component life cycle, dependency injection etc., that are needed in every angular application.

This is the mandatory package.

Examples of decorators provided by @angular/core package: @Component, @NgModule, @Input, @Injectable, @Inject etc.

Examples of component life cycle interfaces provided by @angular/core package: OnInit, DoCheck, AfterViewChecked, OnDestroy etc.

This package contains a module called “ApplicationModule”, which contains the set of pre-defined scripts that are needed to run the angular application

2 @angular/common

This package provides common directives and pipes that are needed in most of the angular applications.

This is the mandatory package.

Ex of directives provided by @angular/common package: ngIf, ngFor, ngSwitch, ngClass etc

Ex of pipes provided by @angular/common package: uppercase, lowercase, date, currency etc.

This package contains a module called “CommonModule”, which contains the above specified directives and pipes

@angular/platform-browser

This package imports “ApplicationModule” from “@angular/core”, “CommonModule” from “@angular/common” and re-exports them as “BrowserModule”. Additionally, it provides some runtime services (such as “error handling, history handling” etc.), that are needed while running the application in the browser.

This is the mandatory package.

@angular/compiler
This package is used to compile the “template” into a “javascript code”. We never invoke the angular compiler directly; we will call it indirectly through either “@angular/platform-browser-dynamic” or “@angular/platform-browser”.
This is the mandatory package.

@angular/platform-browser-dynamic
An angular application can have any no. of modules. This package is used to bootstrap (start) executing a module, which execution should be started automatically at application startup.
This is the mandatory package.

@angular/forms
This package is used for creating “two way data bindings” and “validations” in angular applications. This package works based on another package called “zone.js”.
This package has two modules: FormsModule and ReactiveFormsModule.

@angular/router
This package is used to creating “routing” (page navigation) in angular applications.
This package has one module: RouterModule
This is the optional package

@angular/http
This package is used to send ajax requests to server and get ajax response from server. This package works based on another package called “rxjs”.
This package has one module: HttpClientModule

@angular/animations
This package is used to create animations in angular applications.
This package has one module: AnimationsModule

@angular/material
Used to use “angular material design” in angular applications.
This package has several modules: MatButtonMoudle, MatCheckboxModule, MatMenuModule etc

@angular/cdk
Based on this package only, “angular material design” components are developed. So this package must be used while using “@angular/material” package

@angular/cli
This package provides a set of commands to create new angular application and its code items such as components, pipes, directives, services etc.
This is the mandatory package.
This package must be installed globally, with “-g” option.

2. tsconfig.json

Every compiler has some configuration settings.
• This file is used to set configuration settings of the “tsc” (Type Script Compiler).
• The “tsc” compiler automatically reads the “tsconfig.json” file; and then only it compiles the
“.ts” files into “.js” file.
• This is a fixed filename

3. tslint.json
This file contains configuration settings for “tslint” tool, which is used to verify whether the
typescript files are following a set of coding standards or not.
• For example, we can check the maximum length of the line, indentation

4.protractor.conf.js

This file contains configuration settings for “protractor” tool, which is used to perform unit
testing of components.
• The “protractor” tool is used to execute the test cases that are defined using “jasmine

5.karma.conf.js
This file contains configuration settings for “karma” tool, which is used to execute unit test
cases on multiple browsers

6.angular-cli.json
This file contains configuration settings for “@angular/cli” tool, which is used create,
compile and run the application.
It contains settings such as home page (index.html), startup file name (main.ts), css file
name (styles.css) etc

7.polyfills.ts

This file contains configuration settings for importing (loading) polyfills which are needed to
run angular applications on old browsers such as Internet Explorer.

8.src/styles.css
This file contains CSS styles that are applicable for entire application

9.src/index.html
This file is the home page (startup page) for the entire application.
The content of the entire application appears in the samee html file only.
This file invokes the “AppComponent” using <app-root></app-root> tag.

10. src/main.ts
This is the first typescript file that executes in the angular application.

It enables the “Production mode” and specifies startup module:

11.src/app/app.module.ts
This file contains definition of “AppModule”.
• Angular application can has any no. of modules. It should contain atleast one module, that
is called as “AppModule”.
• This file imports “AppComponent” from “app.component.ts” file and bootstraps the same
in “AppModule”.

12.src/app/app.component.ts
• This file contains definition of “AppComponent”.
• Angular application can has any no. of components. It should contain atleast one component, that is called as “AppComponent”

13.src/app/app.component.html

This file contains actual content (html code) of the component.
• Every component should have a template.
• This template content will be rendered into <app-root></app-root> tag at index.html.

14.src/app/app.component.css
This file contains css styles of “AppComponent”.
• One component can have only one css file.
• By default, this file is empty

15. src/app/app.component.spec.ts
• This file contains test cases for “AppCompoent”.
• The test case files should have “spec.ts” file extension

DAY-7

AGENDA

- Modify the app.component.html
    - Adding the google font
    - Adding the bootstrap in angular project
        - Here showing how to use bootstrap components and test it.
    - Understanding about the component
        - Working with app.component.ts

Components

What is Component?

- The component class represents a certain section of the web page. For example, “login form” is represented as a “Login Component”.
- The component class includes “properties” (to store data), “methods” (event handler methods to manipulate data).
- Every “angular application” contains at-least one component, which is called an “app component”. You can create any no. of components in the project.
- The component is invoked (called) through a custom tag (user-defined tag). For example, “login component” is invoked through <login></login> tag. The custom tag is also called as “selector”.
- The component class should have a decorator called “@Component()”, to define that the class is a component class.

Syntax

```
import{Component} from "@angular/core";
@Component(
    meta data
)
class className{
    property:dataType=value;
    method(arguments):returnType{

    }
}
```

MetaData Properties of Component:

---

1. **Selector** : It represents the selector(tag) to invoke the component.
2. **Template** : It represents the template content
3. **templateUrl** : It represents the HTML file that has to be rendered when the component is invoked.
4. **styleUrls** : It represents the list of style sheets(css file) that have to be loaded for the component.
5. **providers** : It represents the list of services to be imported into the component.
6. **animations** : It represents the list of animations to be performed in the component.

Modules

What is Module?

==========

- Module is a part of the project.
- Module is a collection of components, directives and pipes that are related to one specific task of the project.
- Example : “Net banking” project contains modules like “Savings account module”, “Credit cards module” etc.
- Every angular application should contain at least one module, which is called as “root module” or “app module”. The “app component” will be a part of the “app module”.
- Modules can share its components and pipes to other modules.
- Module is a class, with “@**NgModule**” decorator.

Syntax

```
import { NgModule } from “@angular/core”;
@NgModule( meta data )
class classname
{
}
```

Meta Data Properties of Module:

1.**declarations**
Represents the list of components and pipes that are members of the current module

2.**imports**
Represents the list of modules that you want to import into the current module.
You must import “BrowserModule” into the browser, which can be imported from “@angular/platform-browser”.
The “BrowserModule” imports “ApplicationModule” from “@angular/core”, “CommonModule” from “@angular/common” and re-exports them.
The “BrowserModule” must be imported only in the “app module (root module)”; we need not import it in other child modules.

3.**Other modules to import:** FormsModule, ReactiveFormsModule, BrowserAnimationsModule, HttpClientModule, RouterModule etc

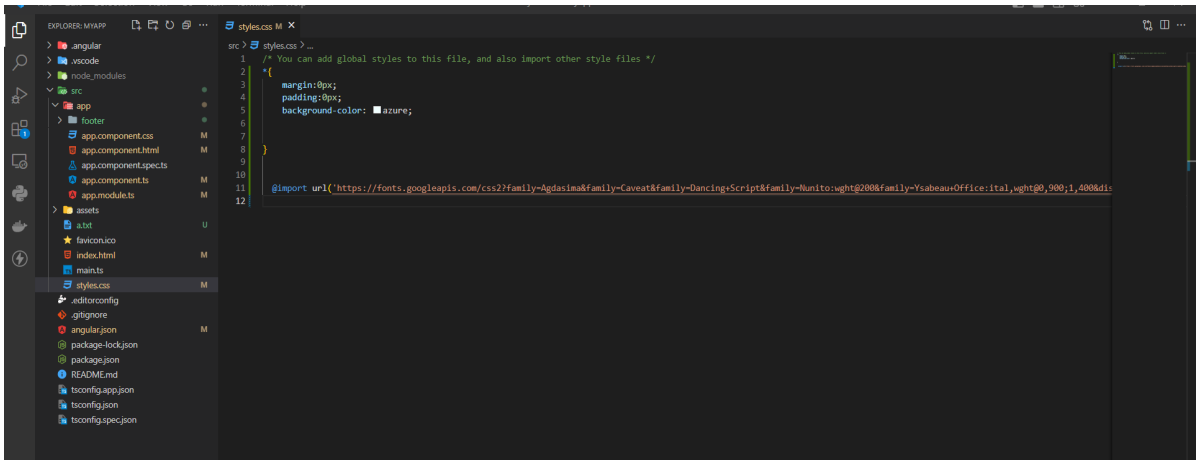
4.**exports:**
Represents the list of components or pipes that are to be exported to other modules.

5.**bootstrap:**
Represents the component that is to be displayed in the web page. Only “app module” has to bootstrap “app component”. Other modules should not bootstrap any component.

6.**providers:**
Represents list of services to be imported into the module.

1. **Adding Good Fonts in Angular Project**
    - In order to add the Google Fonts in Angular Project we use the below steps:
        - Step1: Go to official link : https://fonts.google.com/
        - Step2: Copy the @import style

- Step3: Paste in styles.css



- Step4 : Use the inside the app.component.css file

app.component.css M

src > app > app.component.css > h2

```css
h2{
text-align: center;
font-size: 45px;
font-family: 'Agdasima', sans-serif;

}
```

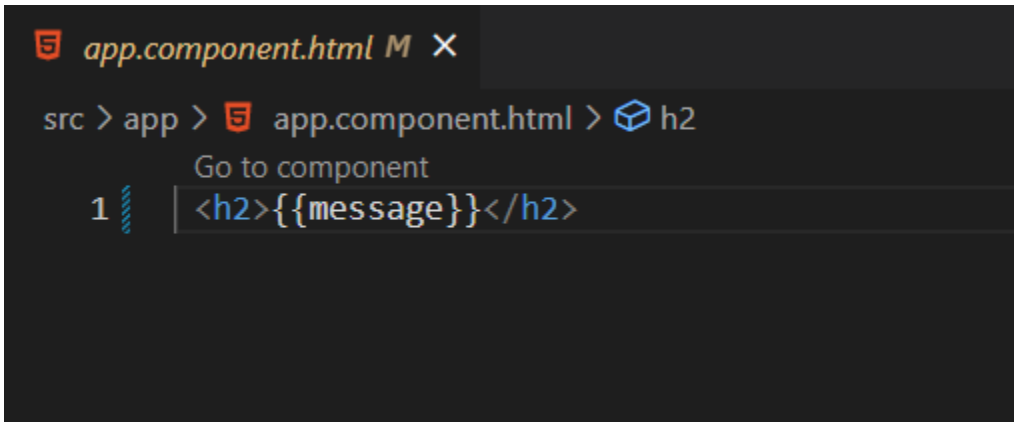
- Step5: Take some property data from app.component.ts

app.component.ts M

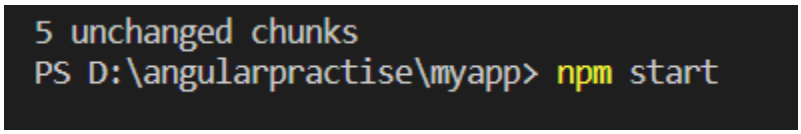
src > app > app.component.ts > AppComponent

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
})
export class AppComponent {
  message: string = 'WELCOME TO HOME PAGE';
}

```

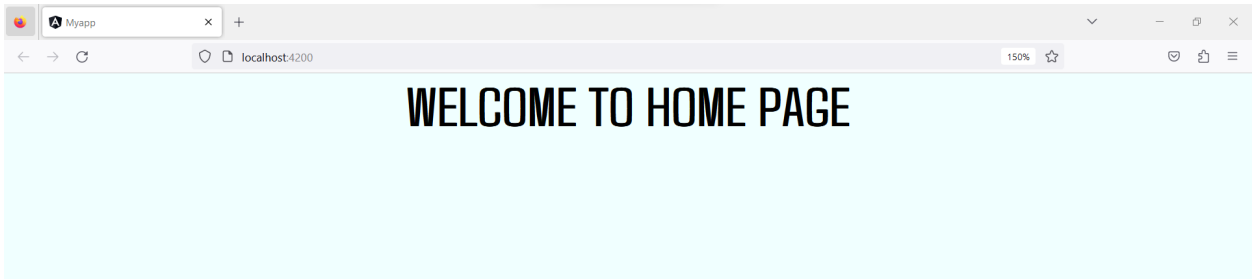
-Step6: app.component.html



-Step8: start the project



- Step9: Open the browser and test it.



-

Bootstrap

- Bootstrap is a powerful framework used for web development. It provides pre-designed and pre-coded components, such as buttons, forms, and navigation menus, as well as a responsive grid system.
- With Bootstrap, developers can create visually appealing and responsive websites or web applications more easily by utilizing its built-in styles, layouts, and interactive features.
- It saves time and effort by providing ready-to-use components and a consistent design framework

They were many ways to add bootstrap in angular application but here I am showing the following two approaches:

Approach1: **By using Bootstrap CDN**

- Bootstrap CDN (Content Delivery Network) is a service that hosts the Bootstrap framework files on servers distributed around the world. It allows developers to include Bootstrap in their web projects by referencing the Bootstrap files directly from the CDN servers, rather than downloading and hosting the files locally.
- CDNs offer several advantages:
- **Speed and Performance:** CDNs have servers located in different geographical regions. When a user accesses your website, the files are delivered from the server closest to their location. This reduces latency and ensures faster loading times.
- **Caching:** CDNs store files in cache, so if multiple websites reference the same Bootstrap files, they can be retrieved from the cache rather than fetching them again from the origin server. This improves efficiency and reduces the load on the server.

- **Scalability:** CDNs are designed to handle high traffic and distribute the load across multiple servers. This helps ensure that the Bootstrap files are delivered quickly and reliably, even during periods of heavy traffic.

Step1: https://getbootstrap.com/docs/4.6/getting-started/introduction/

Step2: Take css, and js files

css :

```
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/css/bootstrap.min.css"
integrity="sha384-xOolHFLEh07PJGoPkLv1IbcEPTNtaed2xpHsD9ESMhqIYd0nLMwNLD69Npy4HI+
N" crossorigin="anonymous">
```

js:

```
<script src="https://cdn.jsdelivr.net/npm/jquery@3.5.1/dist/jquery.slim.min.js"
integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkf
j" crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"
integrity="sha384-9/reFTGAW83EW2RDu2S0VKaIzap3H66lZH81PoYlFhbGU+6BZp6G7niu735Sk7l
N" crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/js/bootstrap.min.js"
integrity="sha384-+sLIOodYLS7CIrQpBjl+C7nPvqq+FbNUBDunl/OZv93DB7Ln/533i8e/mZXLi/P
+" crossorigin="anonymous"></script>
```

Step3: Paste inside the index.html file

Step4: Now use the bootstrap component inside the angular template.

app.component.ts

app.component.ts M

src > app > app.component.ts > ...

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
})
export class AppComponent {
  message: string = 'welcome to home page';
}

```

app.component.html

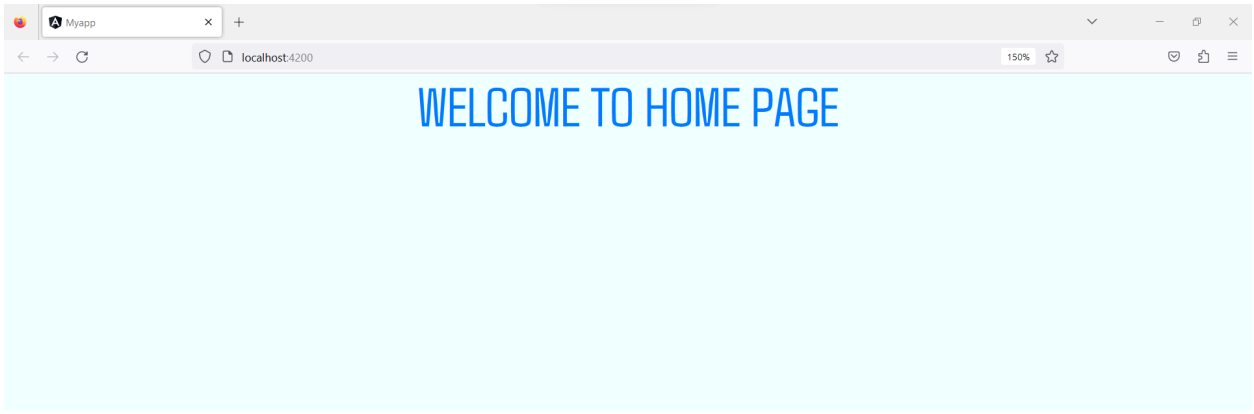
app.component.html M

src > app > app.component.html > h2.text-uppercase.text-primary

Go to component

```html
<h2 class="text-uppercase text-primary">{{message}}</h2>
```

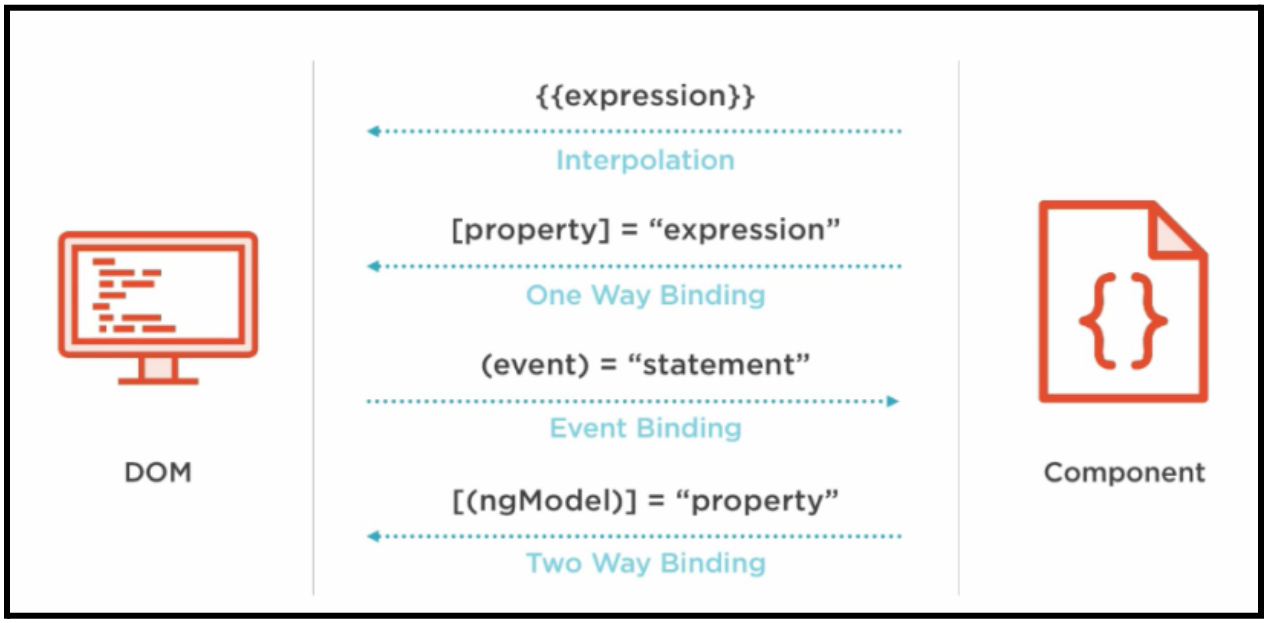
Step5: Open the browser and test it



DAY-8

AGENDA

1. Data Binding
    a. Types of Data Binding
        i. Interpolation Binding
        ii. Property Binding
        iii. Event Binding
        iv. Two ways Binding



Last Session : Angular with Bootstrap

Topic :

1. Adding the Google Fonts in Angular

2. Data-Binding

1. Interpolation:

- Here we communicate from component to template.

- We can use by using following way:

syntax:

{{propertyName}}

Req: Let's create a simple application to print some customer information:

Step1: app.component.ts

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

@Component({
```

```
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
 homePage:string="WELCOME TO ASHOKIT";
 firstName:string="RAJU";
 lastName:string="KUMAR";
 age:number=30;

}
```

Step2: app.component.html

```
<div class="container">
          <div class="headerPart">
         <h2>{{homePage}}</h2>
          <hr color="red"/>
          </div>
          <div class="contentPart">
         <h2>USER INFORMATION</h2>
         FIRSTNAME:{{firstName}}<br/>
```

```
        LastName:{{lastName}}<br/>
         AGE:{{age}}
        </div>
        <hr color="red"/>
        <div class="footerPart">
          Copywrite-2023
        </div>
</div>
```

Step3:app.component.css

```
.headerPart,.footerPart{
        text-align: center;
        font-size: 30px;
        font-family: 'Courier New', Courier, monospace;
}

.contentPart{
        font-size: 30px;
        font-family: 'Courier New', Courier, monospace;
}
```

==========

2. Property Binding:

- In this communiction b/w component to template.

- Here we can use the following way to invoke the value to the html element attribue.

- [attribute]=property;

Example:

- Get the image from local machine and print on the browser.

Step1: Get the image and paste inside the

assets folder

Step2: Create the app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
```

```
})
export class AppComponent {
 homePage:string="WELCOME TO ASHOKIT";
 firstName:string="RAJU";
 lastName:string="KUMAR";
 age:number=30;
 locationofImage:string="../assets/moon1.png";

}
```

Step3:Create the app.component.html file

```
<div class="container">
        <div class="headerPart">
       <h2>{{homePage}}</h2>
        <hr color="red"/>
        </div>
        <div class="contentPart">
       <h2>USER INFORMATION</h2>
       FIRSTNAME:{{firstName}}<br/>
       LastName:{{lastName}}<br/>
        AGE:{{age}}
```

```
        <h3>{{locationofImage}}</h3>
         <img [src]="locationofImage" alt="Still Loading...."/>
         </div>
         <hr color="red"/>
         <div class="footerPart">
           Copywrite-2023
         </div>
</div>
```

==========

3. Event Binding:

- In this case the communication from template to component.
- In order to use the event binding we have use the following way:

```
<element (event)="method()"></element>
```

Example:

Step1: Create the app.component.html

```
<div class="container">
        <div class="headerPart">
        <h2>{{homePage}}</h2>
        <hr color="red"/>
        </div>
        <div class="contentPart">
          <h2>USER INFORMATION</h2>
        FIRSTNAME:{{firstName}}<br/>
        LastName:{{lastName}}<br/>
        AGE:{{age}}

        <h3>{{locationofImage}}</h3>
        <img [src]="locationofImage" alt="Still Loading...."/><br/>
        <button (click)="action1()">On Click Here</button>

        </div>
        <hr color="red"/>
        <div class="footerPart">
          Copywrite-2023
        </div>
</div>
```

Step2: Create the app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
 //property
  homePage:string="WELCOME TO ASHOKIT";
 firstName:string="RAJU";
 lastName:string="KUMAR";
 age:number=30;
 locationofImage:string="../assets/moon1.png";

 //methods
 action1():void{
```

```
  window.alert('Welcome to Ashokit')
 }



}
```

==========

4. Two-Ways Binding:

- It is communication from template to component and vice versa.
- In order to achieve the two-ways binding we can use the following way:

Step1: Add the FormsModule inside the app.module.ts

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

import { AppComponent } from './app.component';
import {FormsModule} from '@angular/forms';

@NgModule({
```

```
  declarations: [
        AppComponent
  ],
  imports: [
    BrowserModule,FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Step2: Use the ngModel directive

```
<div class="container">
        <div class="headerPart">
       <h2>{{homePage}}</h2>
        <hr color="red"/>
        </div>
        <div class="contentPart">
       <h2>USER INFORMATION</h2>
```

```
        FIRSTNAME:{{firstName}}<br/>
        LastName:{{lastName}}<br/>
         AGE:{{age}}

      FIRSTNAME:<input type="text" [(ngModel)]="firstName"/>

        <input type="button" value="ChangeData" (click)="action1()"/>
        </div>
        <hr color="red"/>
        <div class="footerPart">
          Copywrite-2023
        </div>
</div>
```

Step3: Create the app.component.ts file

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
```

```
})
export class AppComponent {
 //property
  homePage:string="WELCOME TO ASHOKIT";
 firstName:string="SUNIL";
 lastName:string="KUMAR";
 age:number=30;
 locationofImage:string="../assets/moon1.png";

 //methods
 action1():void{
   this.firstName="SUDHEER";
   this.lastName="REDDY";
   this.age=35;
  }


}
==========================================================
```

DAY-9

AGENDA

1. Directives
    a. Build in Directives
        i. Style
        ii. ngClass
        iii. ngIf
        iv. ngIf and else
        v. ngSwitch

Directives

- In Angular, a directive is a powerful feature that allows you to extend the behavior of HTML elements or create reusable components.
- Directives are used to manipulate the DOM (Document Object Model), add behavior to elements, and enhance the functionality of your application.

Types of Directives:

- **Attribute Directive:** An attribute directive is used to change the behavior or appearance of an element. It is applied as an attribute to an HTML element and is used to manipulate the element's properties or add custom behavior. Examples of attribute directives in Angular include ngClass, ngStyle, and ngModel.
- **Structural Directive:** Structural directives are used to modify the structure of the DOM by adding or removing elements. They are applied to an element with a specific syntax using an asterisk (*) before the directive name. The most common structural directives in Angular are ngIf, ngFor, and ngSwitch.
- **Component Directive:** Components are the most common type of directive in Angular. A component directive is used to create reusable, self-contained UI components. It consists of a template (HTML), styling (CSS), and behavior (TypeScript). Components can be nested within each other to build complex UI structures.

I. **Style**

—----------

- It is used to set the CSS property value dynamically at run time. When the value of component property is changed, the value of css property will automatically change.
- **Syntax**:
  <tag [style.cssproperty]=”component property”>
  </tag>

II. **ngClass**

—----------

- It is used to set the css class name dynamically at run time. When the value of component property is changed, the css class will be automatically changed.
- Use this directive to set styles with multiple properties, conditionally at runtime.
- **Syntax**:
  <tag [ngClass]=”component property”>
  </tag>

III. ngIF

—----------

- The “ngIf” displays the element if the condition is “true”; otherwise the element will be deleted from DOM.
- **Syntax**:

  <tag *ngIf=”condition”>

  </tag>

  - The “ngIf” must be prefixed with “*”, to mark that it accepts “micro syntax”, which is not just an “expression”, it accepts its own syntax.

- Use “ngIf” if you want to display some content based on the condition. The content appers when the condition is true, it disappears when the condition is false.

IV.ngIf and else

—----------

- The “ngIf and else” displays one element if it is “true”; otherwise it displays another element

- **Syntax**:
  <tag *ngIf=”condition; then template1; else template2”>
  </tag>
  <ng-template #template1>
  …
  </ng-template>
  <ng-template #template2>
  …
  </ng-template>

- The “ng-template” is a container, inside which you can place any no. of tags.
- Use “ngIf and else”, if you want to display one content for the “true” case, another content for the “false” case.

Example:

```
<div>
 <h4>ngIf and else</h4>
  <div *ngIf="b;then template1;else template2">
  </div>
  <ng-template #template1>
    <div style="background-color:green">
          Success
```

```
    </div>
  </ng-template>

<ng-template #template2>
    <div style="background-color:red">
            Better Luck Next Time!
    </div>
  </ng-template>

</div>
```

V. ngSwitch

—----------

- The “ngSwitch” checks the value of a variable, whether it matches with any one of the “cases” and displays the element when it matches with anyone.
- Use “ngSwitch” if you want to display some content for every possible value in a variable.
- **Syntax**:

```
<tag [ngSwitch] =”property”>
<tag *ngSwitchCase=” ‘value‘ ”></tag>
<tag *ngSwitchCase=” ‘value‘ ”></tag>
<tag *ngSwitchCase=” ‘value‘ ”></tag>
…
<tag *ngSwitchDefault></tag>
</tag>
```

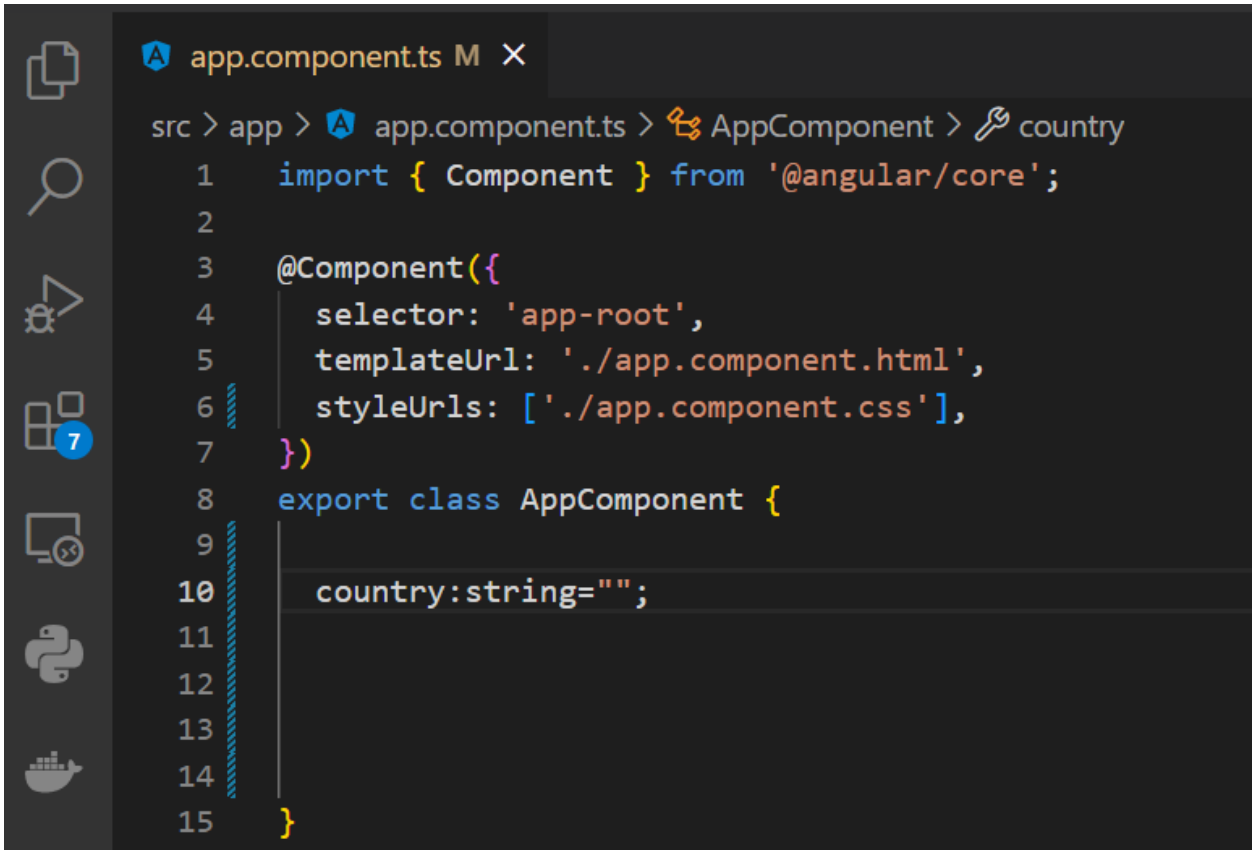
Example:

—---------

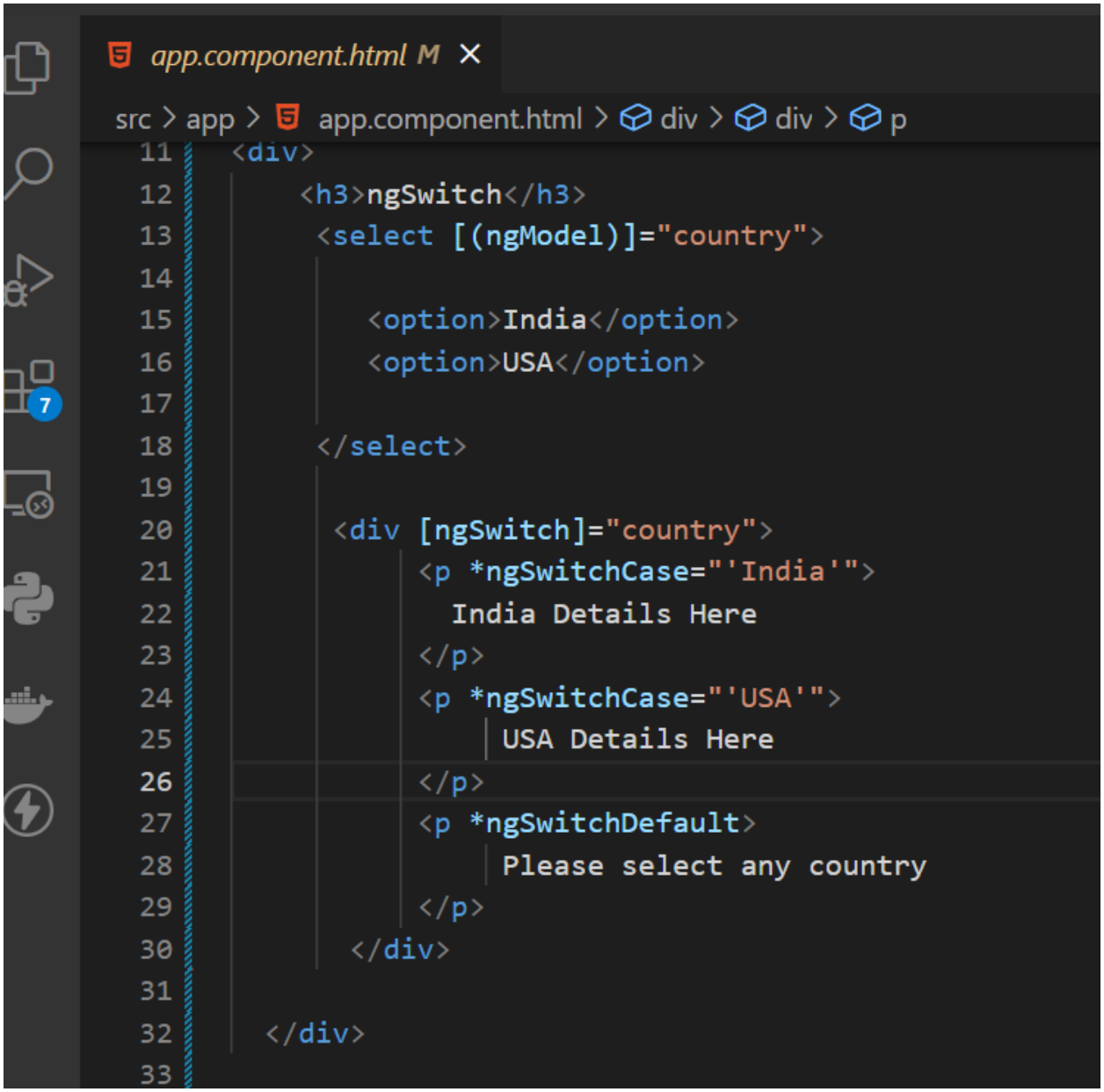
```
<div>
  <h3>ngSwitch</h3>
   <select [(ngModel)]="country">
      <option></option>
      <option>India</option>
      <option>USA</option>

   </select>

    <div [ngSwitch]="country">
        <p *ngSwitchCase="India">
          India Details Here
        </p>
        <p *ngSwitchCase="USA">
            USA Details Here
        </p>
        <p *ngSwitchDefault>
            Please select any country
        </p>
     </div>

</div>
```

==========

Step1: app.component.ts



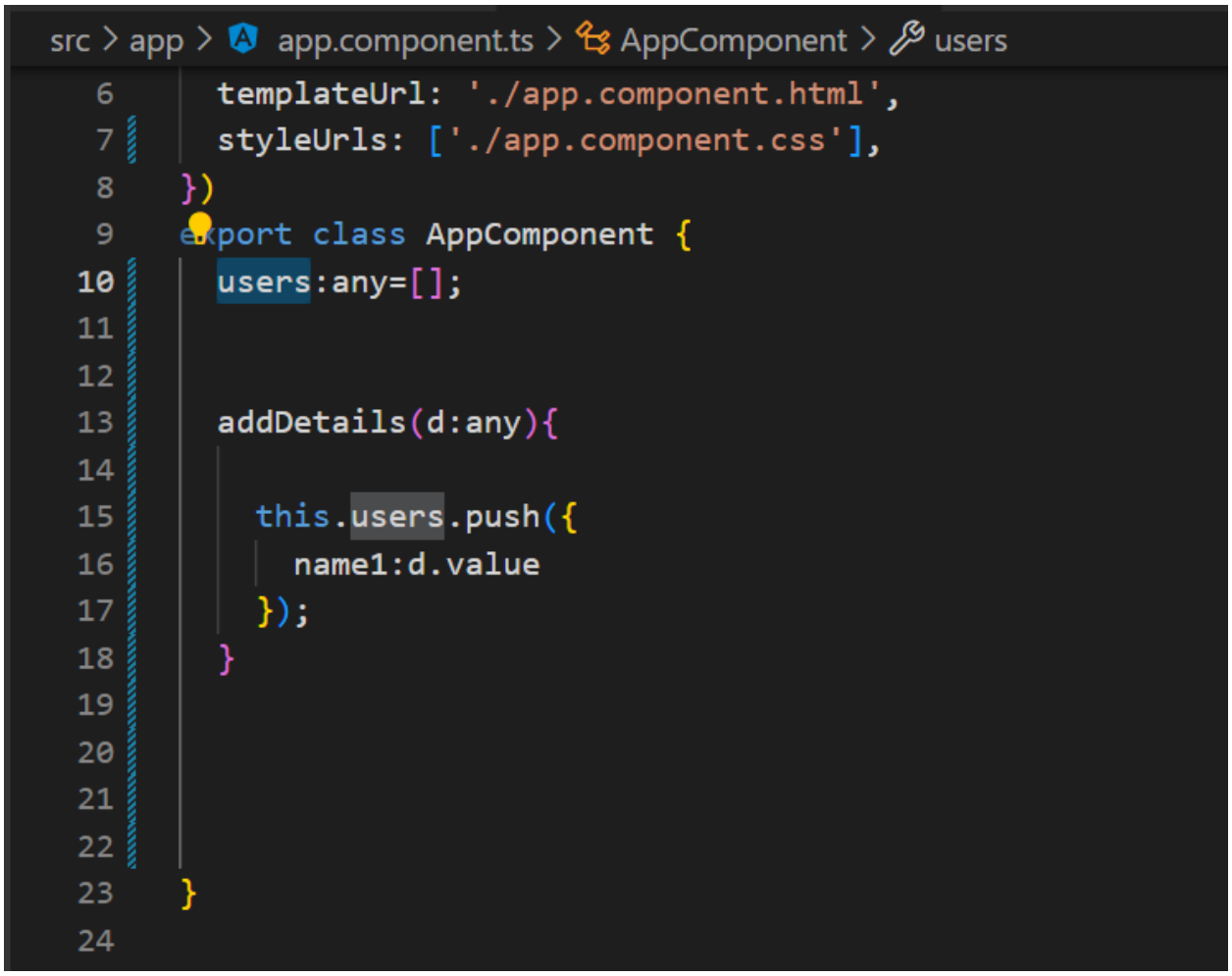
Step2: app.component.html


app.component.html M
src > app > app.component.html > div > div > p

<h3>ngSwitch</h3>
<select [(ngModel)]="country">
<option>India</option>
<option>USA</option>
</select>
<div [ngSwitch]="country">
<p *ngSwitchCase="'India'">
India Details Here
</p>
<p *ngSwitchCase="'USA'">
USA Details Here
</p>
<p *ngSwitchDefault>
Please select any country
</p>

*ngFor

Step1:



Step2:

```
    <h2 [style.backgroundColor]="myBackGroundColor">Second Line</h2>
    <h3 [ngStyle]="{'font-size':30px}">Fourth Line</h3>
    <h4 [ngClass]="myClass">Thrid Line</h4>

</div> -->

<div>
    <input type="text" name="uname" #name1 placeholder="USerName"/>
    <button (click)="addDetails(name1)">ADD</button>
    <table>
     <thead>
         <tr>
             <th>UNAME</th>
         </tr>
     </thead>
     <tbody>
         <tr *ngFor="let n of users">
             <td>{{n.name1}}</td>
         </tr>
     </tbody>
    </table>
  </div>

```

DAY-10

1. *ngFor
    a. Here we learn about the ngFor directive with an application

Drop Down List and Show the Content

Step1: app.component.ts

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

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

Step2:

```
<div class="container">
    <div class="headerPart">
        <h2>Header Part</h2>
    </div>

    <div class="contentPart">
        <div class="selectOption">
            <h3>SELECT COUNTRIES AND DESCRIBE THE DETAILS:</h3>
            <select [(ngModel)]="country">
                <option></option>
                <option>India</option>
                <option>USA</option>
                <option>China</option>

            </select>
```

```
            <div class="details" [ngSwitch]="country">
                <p *ngSwitchCase="'India'">
                    Provide the Details of INDIA
                </p>
                <p *ngSwitchCase="'USA'">
                    Provide the Details of USA
                </p>
                <p *ngSwitchCase="'China'">
                    Provide the Details of China
                </p>
                <p *ngSwitchDefault="">
                       Select the Country
                </p>



            </div>
        </div>

    </div>
    <div class="footerPart">
        Copywrite-2023

    </div>

</div>
```

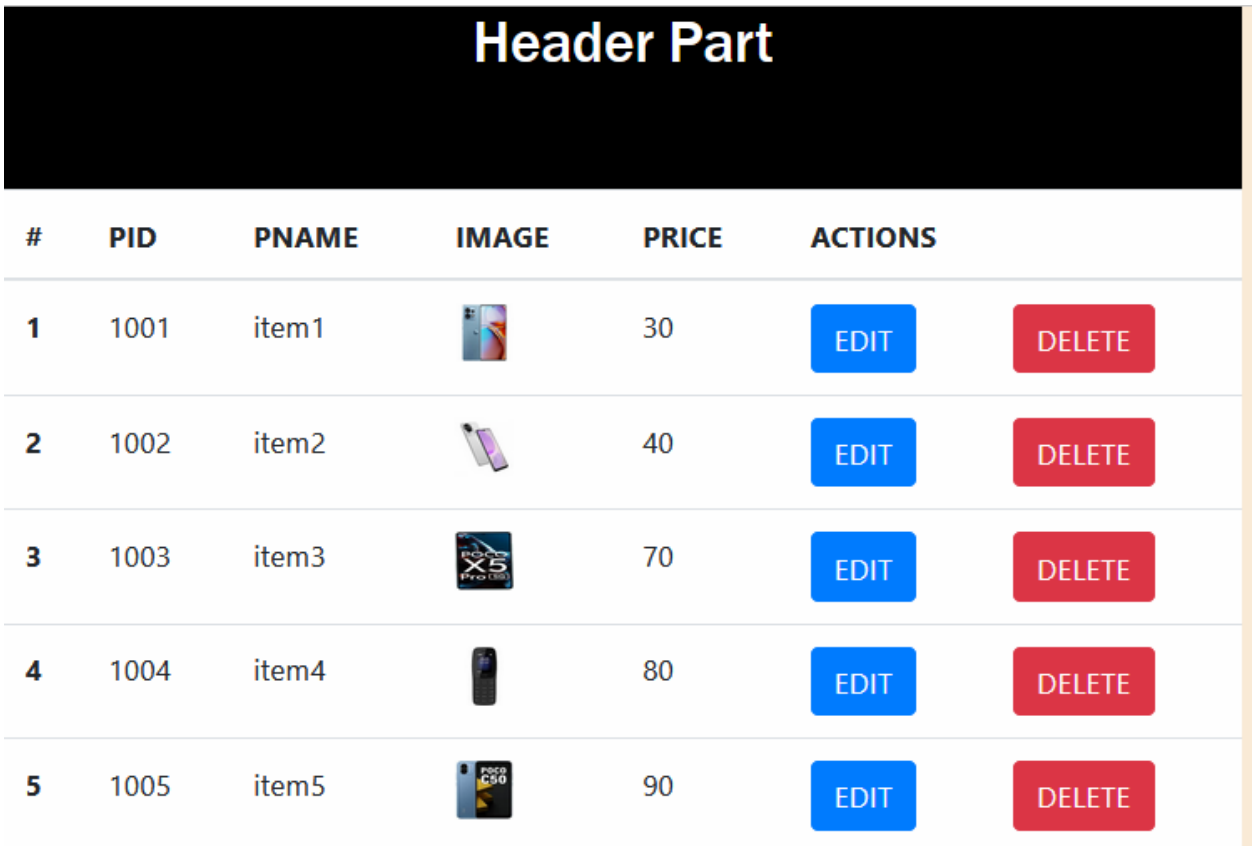
Step3: app.component.css

```
p{
    width:600px;
    height:200px;
    font-size: 1.5rem;
    text-align: center;
    background-color: black;
    color:white;
    border:1px solid red;
}
```

Requirement : Develop the simple application by using the following specification:

1. Show all the product details in the table format.

ScreenShot



In order to implement the above requirement we use the following steps:

Step1: Add the bootstrap inside the index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
```

```
  <title>Myapp</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/css/bootstrap.min.css"
integrity="sha384-xOolHFLEh07PJGoPkLv1IbcEPTNtaed2xpHsD9ESMhqIYd0nLMwNLD69Npy4HI+
N" crossorigin="anonymous">

</head>
<body>
<app-root></app-root>
<script src="https://cdn.jsdelivr.net/npm/jquery@3.5.1/dist/jquery.slim.min.js"
integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkf
j" crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"
integrity="sha384-9/reFTGAW83EW2RDu2S0VKaIzap3H66lZH81PoYlFhbGU+6BZp6G7niu735Sk7l
N" crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/js/bootstrap.min.js"
integrity="sha384-+sLIOodYLS7CIrQpBjl+C7nPvqq+FbNUBDunl/OZv93DB7Ln/533i8e/mZXLi/P
+" crossorigin="anonymous"></script>
</body>
</html>
```

Step2: app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl:'app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
products=[{
  'id':'1001',
```

```
  'name':'item1',
  'img':'../assets/img1.jpg',
  'price':30.0
},
{
'id':'1002',
'name':'item2',
'img':'../assets/img2.jpg',
'price':40.0
},
{
'id':'1003',
'name':'item3',
  'img':'../assets/img3.jpg',
'price':70.0
},
{
  'id':'1004',
  'name':'item4',
  'img':'../assets/img4.jpg',
  'price':80.0
},
{
  'id':'1005',
  'name':'item5',
  'img':'../assets/img5.jpg',
  'price':90.0
  }
];

}
```

Step3: app.component.html

```
<div class="container">
    <div class="headerPart">
```

```
        <h2>Header Part</h2>
    </div>

    <div class="contentPart">
                <div class="workWithNgFor">
                    <table class="table bg-white">
                        <thead>
                          <tr>
                            <th scope="col">#</th>
                            <th scope="col">PID</th>
                            <th scope="col">PNAME</th>
                            <th scope="col">IMAGE</th>
                            <th scope="col">PRICE</th>
                            <th scope="col" colspan="2">ACTIONS</th>
                          </tr>
                        </thead>
                        <tbody>
                          <tr *ngFor="let product of products,index as i">
                             <th scope="row">{{i+1}}</th>
                             <td>{{product.id}}</td>
                             <td>{{product.name}}</td>
                             <td><img src="{{product.img}}"/></td>
                             <td>{{product.price}}</td>
                             <td><button class="btn
btn-primary">EDIT</button></td>
                             <td><button class="btn
btn-danger">DELETE</button></td>
                           </tr>

                        </tbody>
                      </table>

                </div>


    </div>
    <div class="footerPart">
        Copywrite-2023
```

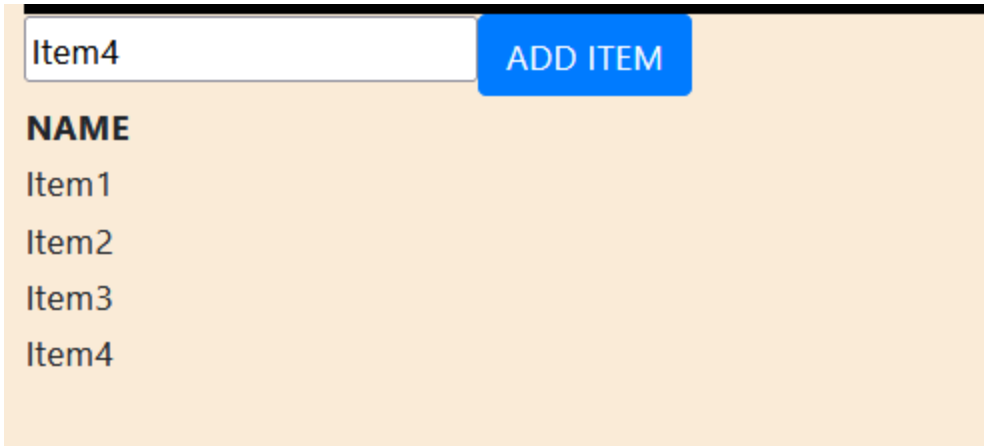
```
    </div>

 </div>
```

Step4: app.component.css

```
p{
    width:600px;
    height:200px;
    font-size: 1.5rem;
    text-align: center;
    background-color: black;
    color:white;
    border:1px solid red;
}

img{
    width:2rem;
    height: 2rem;
}
```

Requirement:

- Develop the simple Application by using following specification:
    - Add one text field of type text
    - Add one button add ADD ITEM
        - Once we click on the ADD button the item has to add to the array and display all the items in table format.



-

Solution:

Step1: app.component.ts

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

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

  addItem(someData:any){
     this.items.push({
       item:someData.value
     });
  }
}
```

Step2: app.component.html

```
<div class="container">
    <div class="headerPart">
        <h2>Header Part</h2>
    </div>
```

```
    <div class="contentPart">
                <div class="myForm">
                    <input type="text" name="item" placeholder="Enter Some Item"
#item>
                    <button class="btn btn-primary" (click)="addItem(item)">ADD
ITEM</button>
                </div>


            <div class="myData">
                <table>
                    <thead>
                    <tr>
                           <th>NAME</th>
                    </tr>
                </thead>
                    <tbody>
                        <tr *ngFor="let it1 of items">
                            <td>{{it1.item}}</td>
                        </tr>
                    </tbody>
                </table>
            </div>
    </div>
    <div class="footerPart">
        Copywrite-2023

    </div>

 </div>
```

DAY-11

AGENDA

- Multiple Components

Req:

Create the following Components and keep inside the root component:

1. IndiaComponent
    a. TS
    b. UP

|-> In order to create the component we use the following command:
> ng g c componentName;

- When we create the respective components it will create the following files and add them into the root module.
    - india.component.ts
    - india.component.html
    - india.component.css
    - india.component.spec.ts–) testing file
    - Updated the child component into the app.module.ts

How we can pass the data from the parent component to child component?
Ans:

- We can pass by using a property binding and child component can receive the data by using @Input() decoarator.

Let's work with data binding between the two components.

Step1: Create the parent component i.e. app.component.ts

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

@Component({
```

```
  selector: 'app-root',
  templateUrl:'app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
    x:number=100;
}
```

Step2: Create the child component i.e child.component.ts

In order to create the child component we use the following command:

>ng g c child

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

@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
    @Input() x:number=0;
}
```

Step3: Inside the app.component.html use the child component selector.

```
<div class="container">
       <app-child [x]="x"></app-child>
</div>
```

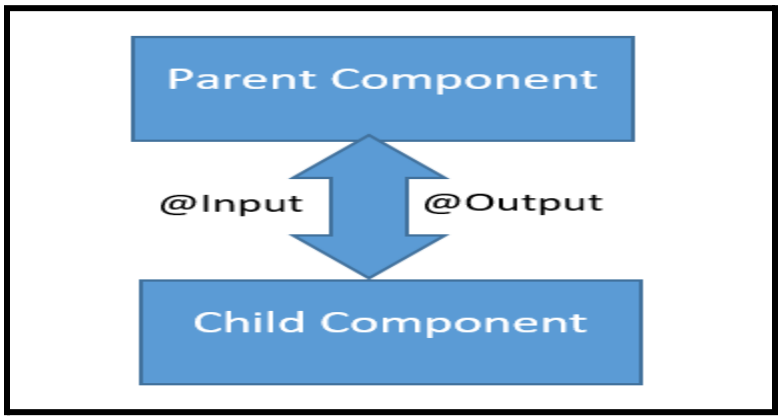
Step4: Display the value of x which is received from the parent component.

```
<p>child works!</p>



{{x}}
```

DAY-12

AGENDA

- **Sharing Data Between the Components**
  - @Input()
  - @Output()



@Output:

- This decorator is used to send the data from child component to parent component.
- In order to work with Output decorate following the below steps:

  Step1: Create the child component

  Step2: Create the parent component

  I. Inside the child.component.html

```
<button (click)="sendData()">Sending Message From Child</button>

Message From Parent:{{y}}
```

  II. Inside the child.component.ts

```
import { Component, EventEmitter, Input, Output } from
'@angular/core';



@Component({

  selector: 'app-child',

  templateUrl: './child.component.html',

  styleUrls: ['./child.component.css']

})

export class ChildComponent {

  @Output() myEvent:EventEmitter<string>=new EventEmitter();

   @Input('y') y:string='';



   childMessage:string="Hi, I am fine!";




   sendData():void{

      this.myEvent.emit(this.childMessage);

   }

}
```

III. Inside the parent.component.html

```
 <hr/>



<app-child [y]="messageFromParent"
(myEvent)="receiveData($event)"></app-child>

<hr/>

Recevive Data From Child:{{receiveDataFromChild}}

<hr/>
```

IV. Inside the parent.component.ts

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



@Component({

  selector: 'app-parent',

  templateUrl: './parent.component.html',

  styleUrls: ['./parent.component.css']

})

export class ParentComponent {

  messageFromParent = 'Hello Child How are You?';

  receiveDataFromChild:string='';

  receiveData(data:any):void{

    this.receiveDataFromChild=data;
```

```
  }

}
```

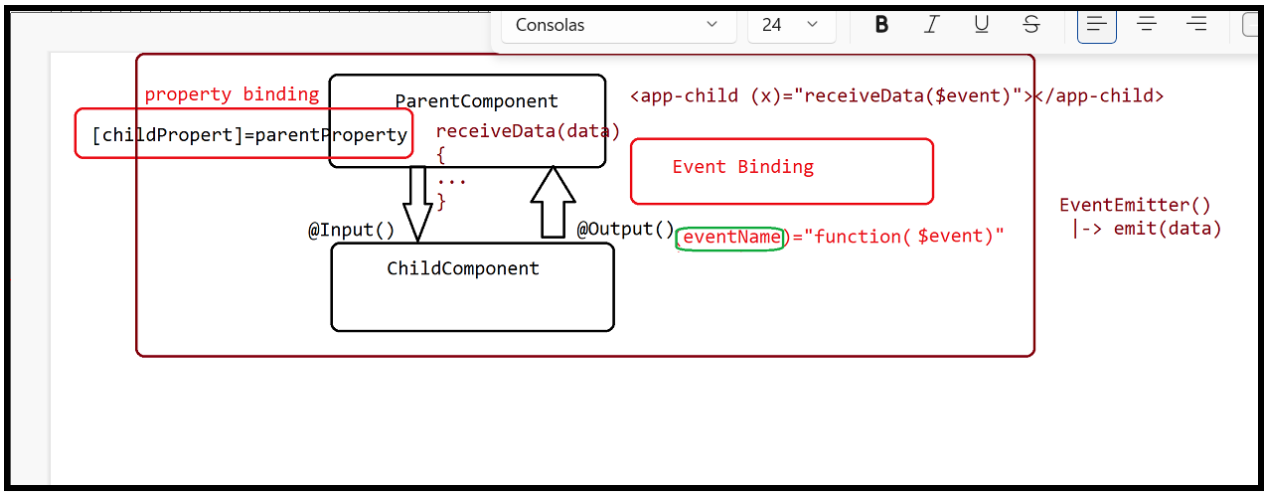
V. Inside the app.compoenent.html

```
<div>

<h2>Welcome to Home Page...</h2>

 <app-parent></app-parent>

</div>
```

VI. Inside the app.component.ts

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



@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})
```

```
export class AppComponent {



}
```



DAY-13

Part-2

AGENDA

- **Sharing Data Between the Components**
    - ViewChild
        - ViewChildren
    - ChildContent
        - ContentChildren

ViewChild

- The “ViewChild” represents an element, which is a child of the view (template) of the component.
- ViewChild is used to access an element, that is present in the view (template) of the component.
- ViewChild can contain a child element of a specific type (class).
- ViewChild is used to access properties / methods of the child.

Steps:

- import “viewchild”

  import {ViewChild} from “@angular/core”;

- Create viewChild property
  - class partentComponent{

    @ViewChild(classname) propertyName:className;

    }

- Access properties / methods of the child element, using ViewChild’s property:
  - this.propertyname.property
  - this.propertyname.method( )

  |-> Create two components parent and child

  Step1:

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

```
@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css']
})
export class ChildComponent {
     name:string="from child";
}
```

Step2:Inside the parent.component.html

```
<div>
    <app-child></app-child>
    <button (click)="show()">Click</button>
</div>

{{pname}}
```

Step3: Inside the parent.component.ts

```
import { Component, ViewChild } from '@angular/core';
import { ChildComponent } from '../child/child.component';
```

```
@Component({
  selector: 'app-parent',
  templateUrl: './parent.component.html',
  styleUrls: ['./parent.component.css']
})
export class ParentComponent {
  @ViewChild(ChildComponent) cref=ChildComponent;
   pname:string='';

  show():void{
    console.log(this.cref)
    console.log(this.cref.name);
    this.pname=this.cref.name;
  }
}
```

II. **ContentChild**

- The “ContentChild” represents an element, which is a child of the content of the
- component.
- ContentChild is used to access an element, that is present in the content of the component.
- ContentChild can contain a child element of a specific type (class).
- ContentChild is used to access properties / methods of the child.

Create ContentChild property:

class parentcomponent

{

@ContentChild(classname) propertyname: classname;

…

}

Step1: app.component.html

```
<div>

<h2>Welcome to Home Page...</h2>

<app-parent>

    <app-child></app-child>
```

```
</app-parent>

</div>
```

Step2: Inside the parent.component.ts

```
import { Component, ViewChild,QueryList,ContentChild } from '@angular/core';

import { ChildComponent } from '../child/child.component';



@Component({

  selector: 'app-parent',

  templateUrl: './parent.component.html',

  styleUrls: ['./parent.component.css']

})

export class ParentComponent {

  //@ViewChild(ChildComponent) cref=ChildComponent;

    @ContentChild(ChildComponent) cref=ChildComponent;

   pname:string='';



  show():void{

    console.log(this.cref)

    console.log(this.cref.name)

  }
```

```
}
```

Step3: Inside the parent.component.html

```
<div>


    <button (click)="show()">Click</button>

    <ng-content></ng-content>

</div>


{{pname}}
```

Step4: Inside the child.component.ts

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


@Component({

  selector: 'app-child',

  templateUrl: './child.component.html',

  styleUrls: ['./child.component.css']

})

export class ChildComponent {
```

```
    name:string="from child";

}
```

Customized Directives:

- In order to work with customized directive following the below steps:

  #Step1: Create the directive as below command:

  > ng g d customizedstyledirective

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


@Directive({
  selector: '[appCustomizedstyledirective]'
})
export class CustomizedstyledirectiveDirective implements OnInit {


  constructor(private e:ElementRef) {


   }
  ngOnInit(): void {
```

```
    this.e.nativeElement.style.backgroundColor='cyan';

    this.e.nativeElement.style.color='red';

  }

}
```

#Step2: Use the directive inside the template

```
 <div class="container">

            <div class="row">

                  <div class="col-md-12">

                        <h2> From the App Component</h2>

                  </div>



            </div>

             <div class="row">

                  <div class="col-md-12 myClass"
appCustomizedstyledirective>

                        <h2> Learn About Directives</h2>

                  </div>



            </div>
```

```
</div>
```

Working with Events in Customized Directives

Step1: Create the directive

```
import { Directive, ElementRef, Renderer2, HostListener, OnInit } from
'@angular/core';



@Directive({

  selector: '[appMyeventdirective]'

})

export class MyeventdirectiveDirective implements OnInit {



  constructor(private e:ElementRef,private render:Renderer2) { }



  ngOnInit(): void {

    this.e.nativeElement.style.color='brown';

  }




  @HostListener('mouseover') method1(event:Event){

        this.e.nativeElement.style.color='red';
```

```
  }


 @HostListener('mouseout') method2(event:Event){
    //this.e.nativeElement.style.color='yellow';
    this.render.setStyle(this.e.nativeElement,'backgroundColor','black');
    this.render.setStyle(this.e.nativeElement,'color','white');


}
 message():void{
   alert('hi')
 }


}
```

Step2: Use the directive

Inside the app.component.html

```
<div class="container">
          <div class="row">
                <div class="col-md-12">
```

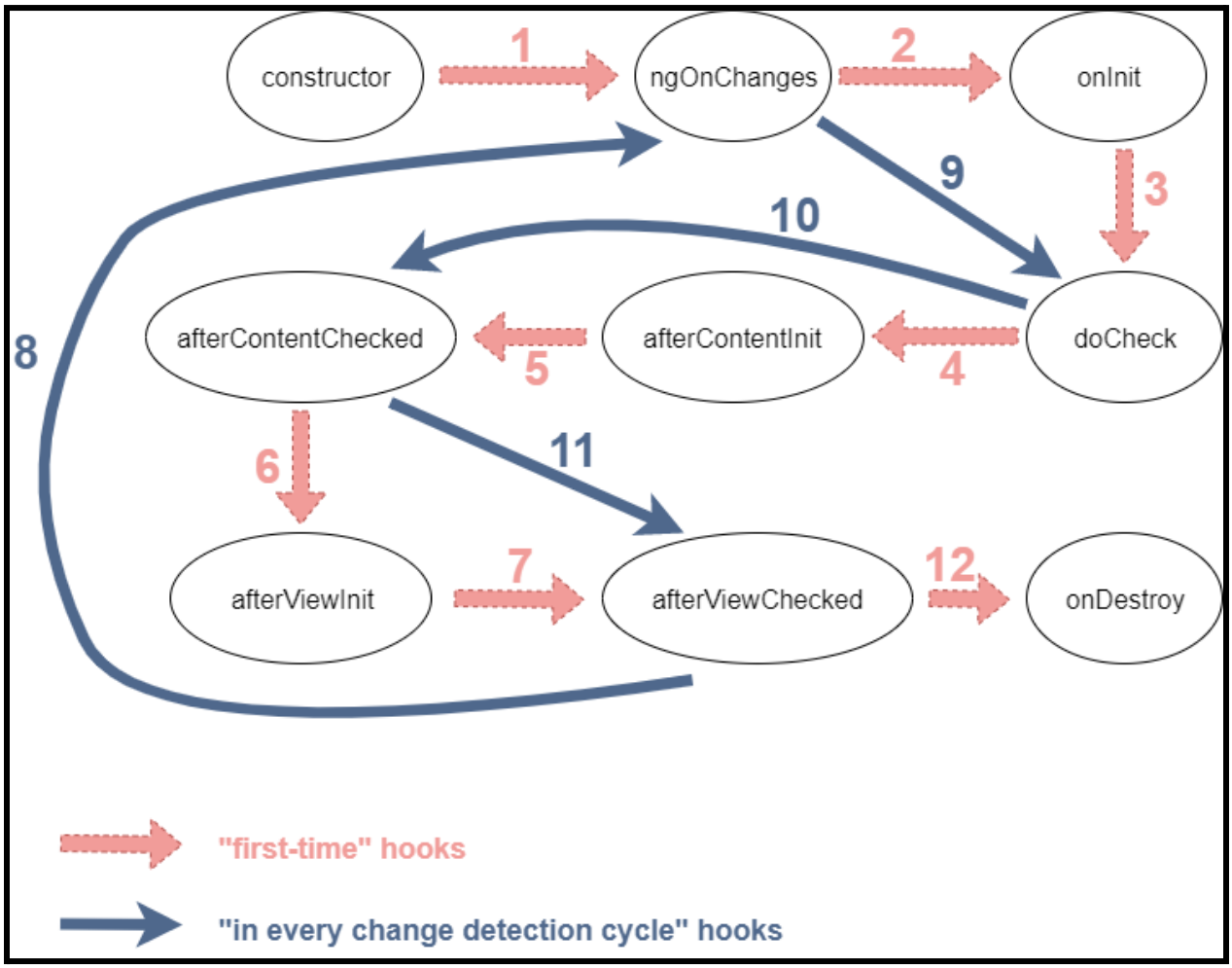
```
                        <h2> From the App Component</h2>
                    </div>


            </div>
             <div class="row">
                    <div class="col-md-12 myClass"
appCustomizedstyledirective>
                        <h2> Learn About Directives</h2>
                    </div>
                    <hr/>
                    <div class="col-md-12 myClass"
appRenderCustomizedstyledirective>
                        <h2>Directive with Renderer</h2>
                    </div>
                   <hr/>
                   <div class="col-md-12 myClass"  appMyeventdirective>
                    <h2>Directive with Events</h2>
            </div>
            <div>
                    <hr/> 
            </div>
            <div class="col-md-12 myClass">
                    <button  appMydirective1>Directive with Events</button>
```

```
            </div>

            </div>

</div>
```

Step3: app.component.css

```
.container{

   width:1000px;

   height:1000px;

   background-color: azure;

}



.myClass{

  border:2px solid red;

  background-color: antiquewhite;

  color:black;

  font-size:30px;

  text-align: center;

}
```

Life Cycle Hooks



Life Cycle Hooks

- Components have a life cycle , which is managed by angular.
- Angular creates it, renders it, creates and renders it children, checks it when its properties changed, and destroys it before removing it from the DOM.
- Angular offers lifecycle hooks that provide visibility into these key life moments and the ability to act when they occur.
- The life cycle events will execute automatically at different stages, while executing the component.
- Directive has a similar life cycle , as angular creates,updates and destroys instances in the course of execution.
- Angular applications can use lifecycle hook methods to tap into key events in the lifecycle of a component or directive to initialize new instances, initiate change detection when needed, respond to updates during change detection, and clean up before the deletion of instances.

Execution Process

Angular calls these hook methods in the following order:

- **First Component Object :**
    - Angular will create the object for the component class i.e. property and methods/function of component class , will be stored in the component object.
- **Constructor :**
    - After an object is created for the component class immediately angular will invoke the constructor.
    - We use the constructor to provide the default values to any properties of the component and also we can inject the services into the component.
- **OnChanges.ngOnChanges :**
    - After the constructor it invokes the ngOnChanges method.
    - When the value of an input property changes, the ngOnChanges method is called if it exists in the component.

- **OnInit.ngOnInit:**
    - **ngOnInit** method of **OnInit** interface will be executed. Use this method to call services to get data from a database or any other data source.

- **DoCheck.ngDoCheck( ) :**
    - **ngDoCheck** method of **DoCheck** interface will execute.
    - This method executes when an event occurs, such as clicking, typing some key in the board etc.
    - Use this method to identify whether the "**change detection**" process occurs or not
- **OnDestroy.ngOnDestroy( ):** This method executes when the component is deleted from memory (when we close the web page in the browser).

In order to work with constructor, OnChanges use the following steps:

#Step1: create the child component:

> ng g c child

>ng g c child2

#Step2: Inside the app.component.ts the following code:

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

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent  {
  title = '';
  msg:string='Default value';
  constructor(){
    //Here we write the code of the following things:
     //1. It is used to initalize the some value into the property.
     //2. Inside the constructor we also write the code
     // to invoke the services.
     this.title='Life Cycle Hooks'
     console.log('--inside the constructor---')
  }
  someChange(){
    this.msg="Ashok It";
  }

}
```

#Step3: app.component.html

```
<div class="container">

      <div class="row">

          <div class="col-md-12">

            <h2>{{title}}</h2>

           <hr/>



          </div>

      </div><!--For Parent-->

      <div class="row">

        <div class="col-md-12">

          <!-- <button (click)="someChange()">Click</button> -->

          <input type="text" name="msg" [(ngModel)]="msg"
placeholder="Enter Some Data"/>

          <hr/>

        </div>

      </div>

       <div class="row">

        <div class="col-md-12">

          <app-child [msg]="msg"></app-child><br/>

        </div>
```

```
      </div><!--For Child-->
      <hr/>
      <div class="row">
       <div class="col-md-12">
        <app-child2 [msg]="msg"></app-child2>
       </div>
      </div>
      <hr/>
</div>
```

#step4: child.component.ts

```
import { Component, ElementRef, Input, OnChanges } from '@angular/core';


@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css']
})
export class ChildComponent implements OnChanges {
  @Input('msg')title:string='';
```

```
  constructor(private e:ElementRef){

    console.log('--inside the child constructor--')

  }

  ngOnChanges(){

    console.log('--inside the ngOnChanges--')

    this.title=this.title+"from child";

    this.e.nativeElement.style.color='red';

  }

}
```

#Step5: child.component.html

```
<div class="container">

    <div class="row">

        <div class="col-md-12">

            {{title}}

        </div>

    </div>

</div>
```

#Step6: app.component.css

```
.container {

    width:600;

    height:600px;

    text-align:center;

    font-size:30px;

    background-color: antiquewhite;

}
```

- **OnInit.ngOnInit:**
    - **ngOnInit** method of **OnInit** interface will be executed. Use this method to call services to get data from a database or any other data source.

- The following are the steps involve to work with OnInit

Step1: Create the parent and child component

>ng g c parent

>ng g c child

Step2: app.component.ts

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



@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

export class AppComponent   {



  constructor(){

    console.log('--inside the appcomponent const--')

  }

}
```

Step3: app.component.html

```
<div class="container">

   <div class="row">

     <div class="col-md-12">

        <h2>WELCOME TO HOME PAGE</h2>

     </div>

   </div>



   <div class="row">

     <div class="col-md-12">

      <app-parent></app-parent>

     </div>

   </div>

</div>
```

Step4: parent.component.ts

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



@Component({

  selector: 'app-parent',

  templateUrl: './parent.component.html',
```

```
  styleUrls: ['./parent.component.css']

})

export class ParentComponent implements OnInit{

  flag:any=false;

  constructor(){

     console.log('--inside the parent constructor--')

  }



ngOnInit(): void {

   console.log('--inside the ngOnInit--:from parent')

}

performSomeAction():void{

  this.flag=true;

}



}
```

Step5: parent.component.html

```
<div class="container">

    <div class="row">

        <div class="col-md-12">
```

```
            <h2>Parent Component</h2>

            <button (click)="performSomeAction()">Change Component</button>

        </div>

    </div>

    <div class="row">

        <div class="col-md-12">

            <app-child *ngIf="flag"></app-child>

        </div>

    </div>

</div>
```

Step6 : child.component.ts

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



@Component({

  selector: 'app-child',

  templateUrl: './child.component.html',

  styleUrls: ['./child.component.css']

})

export class ChildComponent implements OnInit {



  constructor(){
```

```
    console.log('--inside the child constructor--')

  }

  ngOnInit(): void {

    console.log('--inside the child component--:ngOnInit')

  }

}
```

Step7: child.component.html

```
<p>child works!</p>
```

- **OnDestroy.ngOnDestroy( ):** This method executes when the component is deleted from memory (when we close the web page in the browser).

In order to work with ngOnDestory() method we use the following steps:

Step1: parent.component.ts

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

```
@Component({
  selector: 'app-parent',
  templateUrl: './parent.component.html',
  styleUrls: ['./parent.component.css']
})
export class ParentComponent implements OnInit{
  flag:any=false;
  constructor(){
     console.log('--inside the parent constructor--')
  }

ngOnInit(): void {
   console.log('--inside the ngOnInit--:from parent')
}
performSomeAction():void{
  this.flag=!this.flag;
}

}
```

Step2: parent.component.html

```
<div class="container">

    <div class="row">

        <div class="col-md-12">

           <h2>Parent Component</h2>

          <button (click)="performSomeAction()">Change Component</button>

        </div>

    </div>

    <div class="row">

        <div class="col-md-12">

            <app-child *ngIf="flag"></app-child>

        </div>

    </div>

</div>
```

Step3: child.component.ts

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



@Component({

  selector: 'app-child',

  templateUrl: './child.component.html',

  styleUrls: ['./child.component.css']
```

```
})
export class ChildComponent implements OnInit,OnDestroy {


  constructor(){
    console.log('--inside the child constructor--')
  }
  ngOnInit(): void {
    console.log('--inside the child component--:ngOnInit')
  }
  ngOnDestroy(): void {
   console.log('--from child ngOnDestory --')
  }
}
```

Step4: child.component.html

```
<p>child works!</p>
```

Services

In Angular, a service is a class that **encapsulates** a specific functionality or provides a common data source or utility to be shared across components.

Services play a crucial role in building scalable and maintainable applications by promoting code reuse, separation of concerns, and modular architecture.

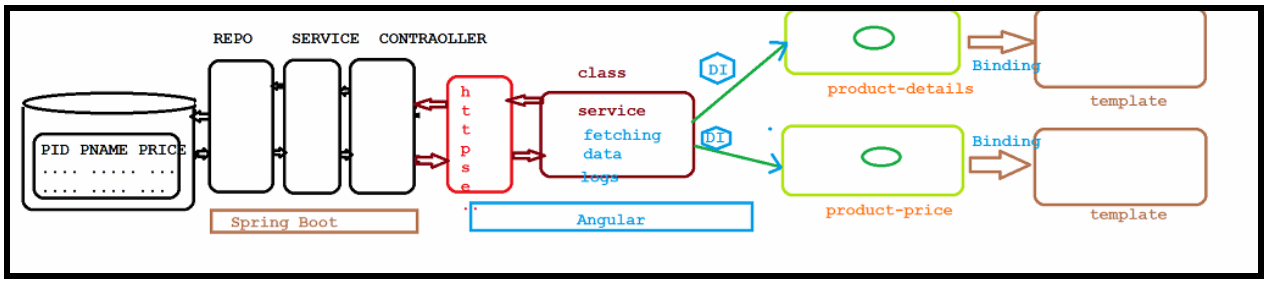
Here are key characteristics and purposes of services in Angular:

Code Organization: Services help to organize and structure the application's codebase by separating different concerns into distinct services. This promotes a clean and maintainable code architecture.

Data Sharing: Services act as a central hub for sharing data between components. They can store and manage data that needs to be accessed by multiple components, ensuring data consistency and avoiding code duplication.

Business Logic: Services encapsulate complex business logic and operations. They provide a dedicated place to implement and manage functions related to data manipulation, calculations, API interactions, and more.

Reusability: Services are designed to be reusable components that can be used in different parts of the application. They can be easily shared and imported into different modules and components, promoting code reuse and reducing duplication.



Observable

-In Angular, the Observable is a built-in class provided by the **RxJS** library. It is used for handling **asynchronous operations** and managing streams of data. Observables are widely used in Angular for handling HTTP requests, event handling, and other asynchronous operations.

- Observables represent a collection of values over time. They can emit multiple values asynchronously and can be subscribed to by observers who want to receive these values. Observables provide a way to handle data asynchronously and react to changes when they occur.
- The key characteristics of Observables in Angular are as follows:
    - - Asynchronous: Observables can handle asynchronous operations such as HTTP requests, timers, and user events.

    - Multiple Values: Observables can emit multiple values over time. Each emitted value is called a "next" value.

- **Stream-based:** Observables are streams of data that can be subscribed to by observers. Observers can react to changes in the stream and perform actions accordingly.
- **Cancelable:** Observables can be canceled or unsubscribed from. This helps in managing resources and preventing memory leaks.

- To use Observables in Angular, you need to import the Observable class from the RxJS library and create an instance of it. You can then subscribe to the Observable to receive the emitted values and perform actions based on those values.

HTTP

- In Angular, HttpClient is a built-in service provided by the Angular framework. It is a part of the @angular/common/http module and is used for making HTTP requests to a server or external API endpoints.
- The HttpClient service simplifies the process of sending HTTP requests and receiving responses by providing a high-level API. It supports various HTTP methods such as GET, POST, PUT, DELETE, etc., and also allows you to set headers, handle request parameters, and process response data.
- Here's a brief overview of some of the key features and functionality provided by HttpClient in Angular:
- **Sending HTTP Requests:** HttpClient allows you to send HTTP requests to a server or API endpoints using methods like get(), post(), put(), delete(), etc. You can specify the URL, request headers, request body, and other parameters as needed.
- **Handling Response:** The HttpClient methods return an Observable that you can subscribe to in order to receive the response data. You can use operators provided by the RxJS library (e.g., map(), filter(), catchError()) to transform and process the response data.

- **Request Interception and Headers:** You can intercept and modify outgoing requests using the HttpInterceptor interface. It allows you to add headers, modify the request URL, handle authentication, and perform other pre-request or post-request operations.

- **Error Handling:** HttpClient provides mechanisms to handle errors in HTTP requests. You can handle errors using the catchError() operator or by specifying an error handler function in the subscribe() method.
- **Request Progress Tracking:** HttpClient provides the ability to track the progress of an HTTP request, such as the percentage of data downloaded or the total number of bytes transferred. This is useful when working with large files or long-running requests.
- To use HttpClient in Angular, you need to import it from the @angular/common/http module and inject it into your component or service using dependency injection. Once injected, you can use the HttpClient instance to make HTTP requests and handle responses.

The following are the below steps to work with Services:

#Step1: Create the db.json file inside the ./app/src/assets/db.json

```
[

{

    "pid":1001,

    "pname":"Mobile",

    "price":10000
```

```
},
{
    "pid":1002,
    "pname":"Mobile1",
    "price":20000
},

{
    "pid":1003,
    "pname":"Mobile3",
    "price":40000
},
{
    "pid":1003,
    "pname":"Mobile4",
    "price":50000
}
]
```

#**Step2**: Create a service as below

> ng g s productservice

productservice.service.ts

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

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

import { Observable } from 'rxjs';

import {Product} from './product';

@Injectable({

  providedIn: 'root'

})

//@Injectable()

export class ProductserviceService {

 private url:any='../assets/db.json';

  constructor(private http:HttpClient) {

    console.log('--inside the ProductService')




  }

  greetingMessage():string{

    return "From Product-Service";

  }



  getData():Observable<Product[]>{

    return this.http.get<Product[]>(this.url)
```

```
  }
}
```

#Step3: Create the component

app.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ProductserviceService } from './productservice.service';
import {Product} from './product';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
})

export class AppComponent implements OnInit {
  message = '';
  products:Product[]=[];
  constructor(private productService:ProductserviceService){}
  ngOnInit(): void {
```

```
    this.message = this.productService.greetingMessage();

     this.productService.getData().subscribe((data)=>{

         console.log(data)

         this.products=data;

     })

  }

}
```

#Step4: Present the data inside the app.component.html

```
<div class="container">

    <div class="row">

        <div class="col-md-12">

            <h1> From App Component</h1>

            <hr/>

               Message:{{message}}

            <hr/>



        </div>

    </div>

    <div class="row">

        <div class="col-md-12">

            <table class="table">
```

```
                <thead>
                  <tr>
                    <th scope="col">#</th>
                    <th scope="col">PID</th>
                    <th scope="col">PNAME</th>
                    <th scope="col">PRICE</th>
                  </tr>
                </thead>
                <tbody>
                  <tr *ngFor="let product of products;let i=index">
                    <th scope="row">{{i+1}}</th>
                    <td>{{product.pid}}</td>
                    <td>{{product.pname}}</td>
                    <td>{{product.price}}</td>
                  </tr>

                </tbody>
              </table>
        </div>
    </div>
</div>
```

PIPES

- Pipes will transform the value into user expected format.
- Pipes invoked in expression (interpolation binding) , through the pipe (|).
- **Syntax** : {{property | pipe:value}}
- Below are the predefined Pipes in angular.
    - 1. uppercase
    - 2. lowercase
    - 3. slice
    - 4.currency
    - 5.date
    - 6.json
    - etc…

Example : app.component.html

```
<div class="container">

    <div class="row">

        <div class="col-md-12">

            <h1> From App Component</h1>

            <hr/>

               Message:{{message}}

            <hr/>
```

```
    </div>
</div>
<div class="row">
    <div class="col-md-12">
        <table class="table">
            <thead>
              <tr>
                <th scope="col">#</th>
                <th scope="col">PID</th>
                <th scope="col">PNAME</th>
                <th scope="col">PRICE</th>
                <th scrop="col">DISCOUNT</th>
                <th scop="col">Date</th>
              </tr>
            </thead>
            <tbody>
              <tr *ngFor="let product of products;let i=index">
                <th scope="row">{{i+1}}</th>
                <td>{{product.pid}}</td>
                <td>{{product.pname | uppercase}}</td>
                <td>{{product.price | currency:'USD'}}</td>
                <td>{{product.discount | percent}}</td>
                <td>{{dt|date:'y/M/d H:mm'}}</td>
```

```
                </tr>



              </tbody>

            </table>

        </div>

    </div>

</div>
```

https://angular.io/guide/pipes

Creating pipes for custom data transformations

- Custom pipes are the user-defined pipes.
- This pipe will be decorated with @Pipe() and implements the "PipeTransform" interface.
    - And override the method transform() inside this method we can write the logic to use your own pipe.
    - Now we have to use the pipe with symbol : {{property | userDefinePipe}}
    - The following are the steps involved to create the pipe.

Step1: Create the Pipe by using following command:

> ng g pipe sqr

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

```
@Pipe({
  name: 'sqr'
})
export class SqrPipe implements PipeTransform {

  transform(value: unknown, ...args: unknown[]): unknown {
    return null;
  }

}
```

Step2: Define the property and assign some value

app.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ProductserviceService } from './productservice.service';
import {Product} from './product';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
```

```
  styleUrls: ['./app.component.css'],
})


export class AppComponent implements OnInit {
  message = '';
  products:Product[]=[];
  dt:Date=new Date();
  input:any=10;
  constructor(private productService:ProductserviceService){}
  ngOnInit(): void {
    this.message = this.productService.greetingMessage();
     this.productService.getData().subscribe((data)=>{
         console.log(data)
         this.products=data;
     })
  }
}
```

Step3: app.component.html

```
<div class="container">

    <div class="row">

        <div class="col-md-12">

            <h1> From App Component</h1>

            <hr/>

               Message:{{message}}

            <hr/>



        </div>

    </div>

    <div class="row">

        <div class="col-md-12">

            <table class="table">

                <thead>

                  <tr>

                    <th scope="col">#</th>

                    <th scope="col">PID</th>

                    <th scope="col">PNAME</th>

                    <th scope="col">PRICE</th>

                    <th scrop="col">DISCOUNT</th>

                    <th scop="col">Date</th>

                  </tr>
```

```
            </thead>
            <tbody>
              <tr *ngFor="let product of products;let i=index">
                <th scope="row">{{i+1}}</th>
                <td>{{product.pid}}</td>
                <td>{{product.pname | uppercase}}</td>
                <td>{{product.price | currency:'USD'}}</td>
                <td>{{product.discount | percent}}</td>
                <td>{{dt|date:'y/M/d H:mm'}}</td>
              </tr>

            </tbody>
          </table>
    </div>
</div>
<hr/>
<div class="row">
  <div class="col-md-12">
    <h2>Customized Pipes</h2>
    <hr/>
   SQUARE VALUE: {{input|sqr}}
  </div>
</div>
```

```
</div>
```

Forms And Validations

Agenda

- Forms And Validations
  - Template Driven Form
  - Reactive Form

Template Driven Form

- **Template Driven Forms** are suitable for development of simple forms with limited no. of fields and simple validations.
- In these forms, each field is represented as a **property** in the component class.
- **Validation rules** are defined in the template, using "**HTML 5**" attributes. Validation messages are displayed using "validation properties" of angular.
- **FormsModule** should be imported from "@angular/forms" package

Validation

- **required = required** : The fields are mandatory.
- **minlength = n** : The minimum no of characters.
- **pattern = req_exp** : The regular expression

Regular expression

- Regular expression is a sequence of patterns that defines a string. It is used to denote regular languages.
- It is also used to match character combinations in strings. String searching algorithm used this pattern to find the operations on string.
- In regular expression, y* means zero or more occurrence of y. It can generate {e, y, yy, yyy,……}
- In regular expression, y+ means one or more occurrences of y. It can generate {y, yy,yyy,.....}

Metacharacters	Description	Example
^	This character is used to match an expression to its right at the start of a string.	^a is an expression match to the string which starts with 'a' such as "aab", "a9c", "apr", "aaaaab", etc.
$	The $sign is used to match an expression to its left at the end of a string.	r$ is an expression match to a string which ends with r such as "aaabr", "ar", "r", "aannn9r", etc.
.	This character is used to match any single character in a string except the line terminator, i.e. /n.	b.x is an expression that match strings such as "bax", "b9x", "bar".
|	It is used to match a particular character or a group of characters on either side. If the character on the left side is matched, then the right side's character is ignored.	A|b is an expression which gives various strings, but each string contains either a or b.
\	It is used to escape a	

	special character after this sign in a string.	
A	**It is used to match the character 'A' in the string.**	**This expression matches those strings in which at least one-time A is present. Such strings are "Amcx", "mnAr", "mnopAx4".**
Ab	**It is used to match the substring 'ab' in the string.**	**This expression matches those strings in which 'Ab' is present at least one time. Such strings are "Abcx", "mnAb", "mnopAbx4".**

Quantifiers

- The quantifiers are used in the regular expression for specifying the number of occurrences of a character.

Characters	Description	Example
+	This character specifies an expression to its left for one or more times.	**s+** is an expression which gives **"s", "ss",** "**sss**", and so on.

?	This character specifies an expression to its left for 0 (Zero) or 1 (one)times.	**aS?** is an expression which gives either "**a**" or "**as", but not "ass".**
*****	This character specifies an expression to its left for 0 or more times	**Br*** is an expression which gives **"B", "Br", "Brr", "Brrr"**, and so on…
{x}	It specifies an expression to its left for only x times.	**Mab{5}** is an expression which gives the following string which contains 5 b's: "**Mabbbbb**"
{x, }	It specifies an expression to its left for x or more times.	**Xb{3, }** is an expression which gives various strings containing at least 3 b's. Such strings are "**Xbbb", "Xbbbb**", and so on.
{x,y}	It specifies an expression to its left, at least x times but less than y times.	**Pr{3,6}a** is an expression which provides two strings. Both strings are as follows: **"Prrrr" and "Prrrrr"**

Groups and Ranges

- The groups and ranges in the regular expression define the collection of characters enclosed in the brackets.

Characters	Description	Example
( )	It is used to match everything which is in the simple bracket.	A(xy) is an expression which matches with the following string: "Axy"
{ }	It is used to match a particular number of occurrences defined in the curly bracket for its left string.	xz{4,6} is an expression which matches with the following string: "xzzzzz"
[ ]	It is used to match any character from a range of characters defined in the square bracket.	xz[atp]r is an expression which matches with the following strings: "xzar", "xztr", and "xzpr"
[pqr]	It matches p, q, or r individually.	Following strings are matched with this expression:

		"p", "q", and "r".
[pqr][xy]	It matches p, q, or r, followed by either x or y.	Following strings are matched with this expression: "px", "qx", and "rx", "py", "qy", and "ry".
(?: …)	It is used for matching a non-capturing group.	A(?:nt|pple) is an expression which matches to the following string: "Apple"
[^…..]	It matches a character which is not defined in the square bracket.	Suppose, Ab[^pqr] is an expression which matches only the following string: "Ab"
[a-z]	It matches letters of a small case from a to z.	This expression matches the strings such as: "a", "python", "good".
[A-Z]	It matches letters of an upper case from A to Z.	This expression matches the strings such as: "EXCELLENT", "NATURE".
^[a-zA-Z]	It is used to match the	This expression matches

	string, which is either starts with a small case or upper-case letter.	**the strings such as: "A854xb", "pv4fv", "cdux".**
[0-9]	**It matches a digit from 0 to 9.**	**This expression matches the strings such as: "9845", "54455"**
[aeiou]	**This square bracket only matches the small case vowels.**	**-**
[AEIOU]	**This square bracket only matches the upper-case vowels.**	**-**
ab[^4-9]	**It matches those digits or characters which are not defined in the square bracket.**	**This expression matches those strings which do not contain 5, 6, 7, and 8.**

Escape Characters or Character Classes

Characters	Description
\s	It is used to match a one white space character.
\S	It is used to match one non-white space character.
\0	It is used to match a NULL character.
\a	It is used to match a bell or alarm.
\d	It is used to match one decimal digit, which means from 0 to 9.
\D	It is used to match any non-decimal digit.
\n It helps a user to match a new line.	
\w	It is used to match the alphanumeric [0-9a-zA-Z] characters.
\W	It is used to match one non-word character
\b	It is used to match a word boundary.

Validation Properties:

S.NO	Validation Properties	value
1	**touched**	**true:** Field is focus
		false : Field is not focus
2.	**untouched**	**true:** Field is not focused
		false : Field is focused.
3.	**dirty**	**true:** Field is modified by user.
		false : Field is not modified by user.
4.	**pristine**	**true :** Field is not modified by user.
		false : Field is modified by user.
5.	**valid**	**true :** Field value is valid.
		false : Field value is not valid.
6.	**invalid**	**true :** Field value is invalid
		false : Field value is valid.
7.	**errors**	**required : true/false** **minlength : true/false** **pattern : true/false** **number : true/false** **email : true/false** **url : true/false**
Digits only : ^[0-9]*$** **Alphabets only : ^[a-zA-Z]*$ **Mobile Number : ^[89]\d{9}$** **etc..**		

- In order to implement we use the following steps:

Step1: Create the new myfrom-application

> ng new myform-app

Step2: Open in the vs code

Step3: Modify the app.module.ts

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

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

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

import {FormsModule} from '@angular/forms';

@NgModule({

  declarations: [

    AppComponent

  ],

  imports: [

    BrowserModule,FormsModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }
```

Step3: Modify the app.component.ts

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



@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  fname:string='';

  pwd:string='';

  data1:any='';




  valid(input:any):void{

      if(input.valid){

       this.data1=(JSON.stringify(this.data1));



      }else{

          alert('some errors');

      }
```

```
  }

}
```

Step4: Modify the app.component.html file

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>



 <div class="container">



    <div class="row">

       <div class="col-md-12">

            <h1>WELCOME TO TEMPLATE DRIVE FORM</h1>

       </div>

       <hr/>
```

```
    </div>
     <div class="row">
       <div class="col-md-12" style="text-align:center">
            <h2>FORM</h2>
            <hr/>
               <form style="margin-left:250px;" #myFrom="ngForm">
                   <table>
                   <tr>
                       <td><label for="uname">USERNAME:</label></td>
                       <td><input type="text" name="fname" required="required"
minlength="3" [(ngModel)]="fname" #c1="ngModel"/></td>
                       <td>
                        <span *ngIf="c1.touched && c1.invalid &&
c1.errors?.['required']" class="errors">Field is Madatory*</span>
                        </td>
                         <td>
                            <span *ngIf="c1.touched && c1.invalid &&
c1.errors?.['minlength']" class="errors">Minimum Length Should be 3</span>
                        </td>
                    </tr>
                   <tr>
                       <td><label for="pwd">PASSWORD:</label></td>
```

```
                    <td><input type="password" name="pwd" required="required"
pattern="^[a-zA-Z0-9]*$" [(ngModel)]="pwd" #c2="ngModel"/></td>

                    <td>

                     <span *ngIf="c2.touched && c2.invalid &&
c2.errors?.['required']" class="errors">Field is Madatory*</span>

                     </td>

                      <td>

                         <span *ngIf="c2.touched && c2.invalid &&
c2.errors?.['pattern']" class="errors">It should be combination of followings:


                             <ul>

                                 <li>It is Combination of a-z</li>

                                 <li>It is Combination of A-Z</li>

                                 <li>It is Combination of 0-9</li>


                             </ul>

                         </span>

                      </td>

                </tr>



               <tr>

                   <td></td>

                   <td>
```

```
                        <input type="submit" value="LOGIN"
(click)="valid(myFrom)">

                    </td>

                </tr>

            </table>

            </form>



          <hr/>

      </div>

    </div>



    <div class="row">

      <div class="col-md-12" [innerHTML]="data1">



      </div>

    </div>

</div>

</body>

</html>
```