Introduction

⇒ It is a JavaScript/Type Script Framework to build Client side Applications.

⇒ Specially used to develop Single page Applications.

⇒ Angular provides Modular Approach & hence the application we build has a clear structure.

⇒ By Making use of Components, which is a feature of Angular we have lot of reusability of code.

⇒ Angular has so many features like Validation Routing which makes development quicker and faster.

⇒ Angular even provides the Support for Unit Testing.

⇒ Angular is from Google & uses the Microsoft Type Script language.

Getting Started

→ Pre Requisite

Basics of HTML, CSS and JavaScript

→ Setting up Dev Environment

Node

Npm

Angular CLI

Visual Studio Code

→ Install Node JS

When we install Node, Npm (node package Manager) will also be installed.

→ Once the Node is installed, in the Command Prompt

node -v gives the respective versions

npm -v

→ Install the Angular CLI ie Command line Interface for Angular.

Angular CLI generates the Building blocks of Angular Appn by typing the Commands.

→ It makes the development quicker and easier..
by following Best practices.

→ To install Angular CLI

npm install -g @angular/cli

To check whether the Angular CLI is installed or not.
in the Command prompt

ng -v

→ Install the Visual Studio Code

Hello World App

→ To Create an Hello World Angular App, we make use of Angular CLI

→ Visual Studio Code has an integrated terminal from where we can run the Commands

View → Terminal

→ Make sure we are inside the Angular folder

→ To Create a new Angular project

ng new hello-world (Project Name)

Command takes time to run and once it is completed it create the folder hello world.

→ change the directory to Application

cd hello-world

, to run the application ng serve

→ Once it is Compiled Successfully, it Create the Hello-world folder.

To check localhost:4200

* Flow of Execution :-

→ Angular Apps are Modular in Nature ie Angular Application is a Collection of Many individual Modules.

→ Every Module represents the feature Area in the Application.

eg Users Module
Admin Module

→ Angular Modules are imported and exported

→ Every Angular Application has one Module ie which is the Root Module by Naming Convention it is App Module.

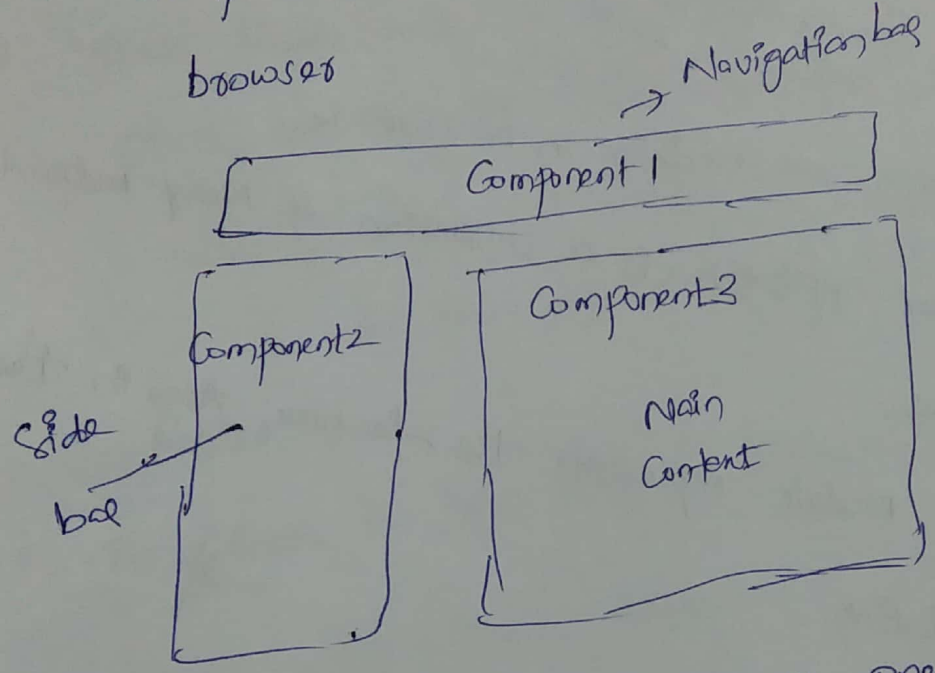
→ Each Module is Intern consists of

Components and

Services

⑥

→ Components controlls a portion of the view on the browser



→ Angular Application has atleast one Component which is the root Content of the Application, is called as App Component

→ All other Components will be nested inside the Component

→ Each Component have

HTML Template

Class

→ HTML Template to represent the view in the browser and Class that controls the logic of the particular view.

→ Module can also have Services, ie basically a class that contains business logic of the Application.

→ In Addition to Components and Services, A Module can have few more peices of the Angular Appn.

Modules support and import the code as and when required and render the view to the browser.

File Structure :-

Package.json :-

→ This Package Contains the dependencies and dev dependencies which are nothing but libraries and Modules that are required for Angular Application to work.

→ The Packages listed in the devDependencies are installed when we run the Command

ng new Hello-world.

→ All the packages get installed in the node_modules

→ We also have some of the scripts operated

ie ng Serve

Used to run the Application

Src folder

→ In the Src folder we have main.ts which is the entry point for our Angular Application

→ We also have app folder, that contains app.module.ts which is the root module of our Application

→ We also have app.component.ts in the app folder which is the root Component for our Application

Flow of Execution

When we run the Command ng tseeve to start the Angular Application, The execution comes to main.ts file

→ In the main.ts file it bootstrap the App Module

→ In the App Module, it inteen bootstrap the App Component

→ App Component has two things

HTML Template to represent the View in the Browser

Class to control the view logic of that particular view.

Source Code

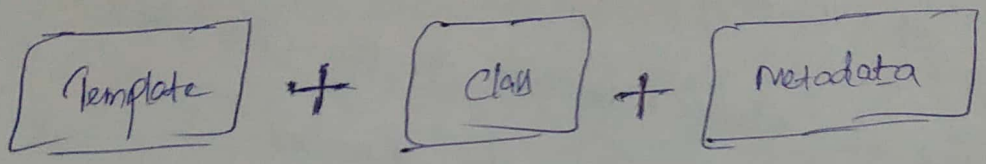
```
import {Component} from '@angular/core';

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

→ At run time HTML property gets replaced by class property. (11)

→ When we change the value in the class, it's Automatically changes in the Browser.

Note :- Component renders different portions of the view in the browser.

Component :-



View
HTML

Code
Typescript
Data and
Methods

Information
Decorator

⇒ Template represents the View of it is Created by HTML & will be the User Interface for the Applications

⇒ Class is Nothing but Code that Supports the View & it is Created by using Typescript.

Class like Any other Programming language Contains the data property and Methods to Control the logic of the View.

Example :- We can have a method to show and or hide the element based on the value of the property

→ A Component can also have Metadata Attached to it.

This is the information that the angular needs to decide whether the particular class is a Angular Component or Normal class

The Meta data is defined using decorator ↑ is a feature in typescript

Decorator is just a function that provides information about the class attached to it.

And for Component we use Component Decorator.

Example :-

In the app.component.ts we have one class Named AppComponent & it contains one data property

```
export class AppComponent
{
  title = 'Code Evolution'
}
```

→ To this class we leave the Metadata Attached in the form of Component Decorator.

→ Component Decorator function is a that attaches the Metadata to the class right below. It

→ this decorator i.e @Component tells the Angular that this is not the plain class & it is Angular Component.

⇒ Component decorator pattern contains both the Meta data and the Template which represents the View.

(15)

As part of Metadata we have

Selector
templateUrl
StyleUrls

⇒ Selector is a custom HTML tag that can be used to represent this Angular Component in the HTML

when we specify the Selector in the HTML. Angular renders the Components template in its place.

Example In the index.html it is used as custom HTML tag

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

→ But what exactly here is the template for this Component & Can be Specified by using templateUrl.

→ The Template Url points to the HTML file that represents the Component.

In app.Component.html we have title & title is the property in the Class app.Component.ts.

#

Creating a New Component

→ To Create a new component, from the Angular CLI

```
ng g c test
```

⇒ When the above Component got executed.
four new files will be created and update of the App Module.

→ In the app folder test folder has been created & it Contains html, CSS, ts and spec.ts file will be created.

→ When we Create any new Component, our application Should aware of this is

In the App Module file, we have imported the Test Component and added to the declarations array.

→ The declarations array contains all the components used by the application. (18)

→ To include this component in the HTML, we just add a custom HTML tag that represents the selector

ie Test.Component.ts

```
@Component(
{
  selector : 'app-test'
  templateUrl : './test.Component.html',
  styleUrl :['./test.Component.ts']
}
)
export class TestComponent implements OnInit
{
  constructor() { }
  ngOnInit() { }
}
```

→ app.component.html which represents the View for app Component & which is the root Component of our application

Include the custom Component

<app-test> </app-test>

⇒ Changes that can be made to any of the Component :-

There are 3 ways to specify the selector

① Selector : 'app-test'

And use it as custom HTML tag

<app-test> </app-test>

② Begin the selector with . and it can be used as a class name in the custom tag in HTML.

Selector : '.app-test'

And in the HTML

```
<div class = 'app-test '>

</div>
```

(3) We can also take the selector between the square brackets

Selector : '[app-test]'

and in the HTML, we need to use app-test as an attribute ie

```
<div app-test>

</div>
```

Template :-

We have a template URL property, which points to the file that contains the HTML.

→ In Any Component it is possible to specify the template Inline in the same type script file & for that we use template property.

template : ' Inline Template '

Case (2) Some times our inline template might span couple of lines & for that we make use of ` Symbol

```
template : `<div>
           Inline Template
           </div>`;
```

// is the case with styleUrl and we use Styles

6. Property Binding

→ Before understanding property Binding it is important to understand the difference between HTML attribute and Dom property

Eg <input type = "text" value = 'vishwas'>

inspect the element in the browser

Step ① $0.getAttribute('value') → vishwas

Step ② $0.value => Vishwas

If I change the text from vishwas to CodeEvolution in the input element

Step ① value will remain same

Step ② will change it returns CodeEvolution

→ Attribute property did not change, where as the value property changes

Points

→ Attributes & properties are not same

→ Attributes are defined by HTML, but properties are defined by using document object Model

→ Attributes initialises Dom properties and then they are done. Attribute values cannot change once they are initialized.

→ Property values however can change.

→ With property binding in Angular, we are Actually binding the property of Dom Element

public class TestComponent implements OnInit

{

public myId = "testId";

}

→ we need to bind this id to the HTML input property element.

→ we enclose id within [ ] brackets

< input [id] = "myId" type = "text" value = "vishwas" >

In the Console

→ Step ③ < input type = "text" value = "vishwas" id = "testId" >

→ we can also use interpolation for property binding ie

< input type = "text" id = {{ myId }} value = "vishwas" >

↳ In the Console we get same as Step ③

→ The problem with interpolation is it only works with string values.

→ There are some boolean HTML properties that we may need to bind to.

Example

<input disabled "id = {{myId}} type = "text"

by default true value = "vishwas">

in the browser it is disabled

(or) {{false}}

<input disabled = "false" -- >

still in the browser it is disabled

i.e This is the problem with boolean attribute such as disabled.

Here interpolation doesn't works & we use property binding

<input [disabled] = "false" -- >

Here in the browser input is not disabled

→ lets Create a property in the class, then bind it to the boolean HTML element.

→ public isdisabled = true;

in the HTML element

<input [disabled] = "isdisabled" id = "{{myid}}"

value = "vishwas" >

→ we can also use bind- instead of [] for property

binding is

<input bind-disabled = "isdisabled" id = {{myid}}

value = "vishwas" >

7. Class Binding :

→ Here we discuss Binding Classes to HTML elements in Angular

→ In the test.component.ts

```
styles : [ `
    .test-Sucess {
        Color: green ;
    }
    .test-danger {
        Color: red ;
    }
    .test-Special {
        font-Style: italic ;
    }
` ]
```

In the HTML Template I have

<h2> Code Evolution </h2>

The regular way is

regular <h2 class = "tpt-success"> Code Evolution </h2>

To assign the class binding in the class

public successclass = "tpt-success";

In the Template

class binding <h2 [class]="successclass"> CodeEvolution </h2>

→ If we take both regular class attribute & class binding then regular class Attribute becomes dummy.

→ we have to use Any one but not both

→ Another syntax for class binding is based on expression apply the class binding

Lets declare a variable in the class ie

```
public hasError = true
```

In the HTML Template

```
<h2 [class.text-danger]="hasError">CodeEvolution</h2>
```

→ The above code works fine when we apply condition for a single class

→ If we want to conditionally apply multiple classes, Angular provides ngClass

→ A Directive is nothing but custom HTML attribute that Angular provides.

lets declare a variable in the class ie

```
public SuccessClass = "text-success";
public hasError = false;
public isSpecial = true;

public messageClasses = {
    "text-success" = !this.hasError,
    "text-danger" = this.hasError,
    "text-special" = this.isSpecial.
}
```

(32)

→ in the HTML Template

```
<h2 [ngClass]="messageClasses"> Code Evolution </h2>
```

→ Class binding is very useful because it allows us to add or remove class ~~elements~~ to HTML elements based on user interaction or State of Application.

→ We can change the classes being applied by changing in the Component class.

8. Style Binding

→ In Angular Style binding is used to apply inline style to HTML elements.

→ Style binding is similar to class binding

```
<h2> Style binding </h2>
```

To apply style binding

```
<h2 [style.color]="'orange'"> Style binding </h2>
```

→ To use the conditional operator to the css property

```
<h2 [style.color]="hasError ? 'red' : 'green'"> Binding </h2>
```

→ We can also apply component class properties to style binding ie

```
public lightlight-Color = "red";
```

<h2> [Style.color] = "highlightColor" > Style Binding </h2>

→ finally to apply multiple styles, we need to make use of

ngStyle directive

ie lets create an object of all the styles

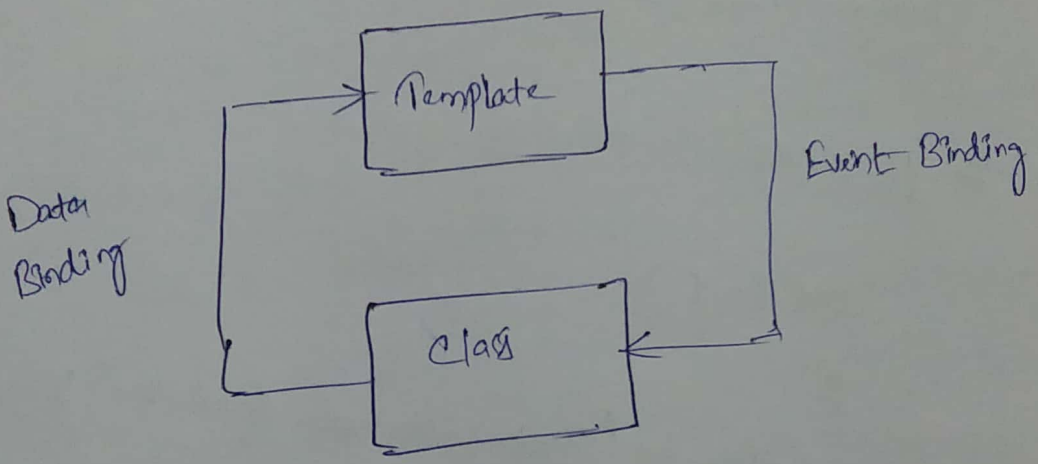
```
public titleStyles = [
    color : "blue",
    fontStyle : "italic"
]
```

<h2 [ngStyle] = "titleStyles" > Style Binding </h2>

9. Event Binding :-

→ Till now we have seen the data flow from Component class to HTML Template. ie

Any update to the class property will reflect to the HTML Template.



→ Sometimes to respond to user events like mouse click, keyboard events, we need the data flow in the other direction as well ie from Template to class.

→ To Capture events we use event binding.

→ <button> Greet <button>

When the user clicks on the button, we want to display Hello user

→ We need to listen the Click event on this button. ie

<button (click) = "onClick()" > Greet </button>

→ In the class

```
onClick ()
{
  Console.log ("Welcome to Code Evolution")
}
```

→ It is also possible to set the properties to the events.

```
public greeting = "";

onClick()
{ this.greeting = "Welcome User"
}
```

In the HTML Template

<h> <button (onclick) = "onclick()" > Greet </button>

{{greet}}

→ Sometimes we may want to know about the information of the event, for we use $event is

<button (onclick) = "onclick ($event)" > Greet </button>

→ $event gives all the information about the event that was raised.

```
onclick (event)
{
  Console.log (event)
}
```

→ Sometimes we can also have Template Statements instead of Event handler. In that case

<button (onclick) = "Welcome vishwas" > Greet </button>

10. Template Reference Variables

⇒ When there is user interaction, we might want some data to flow from view to the class to perform some operation.

→ eg: we may require a value from input field to perform validations.

To easily Access Dom elements and their properties, Angular provides Template Reference Variables.

eg:

```
<input type="text">
<button> Log </button>
```

Requirement is: when the user clicks on the log the text has to displayed on the console

for this we use Template Reference Variable

<input #myInput type="text">

<button (click)="logMessage(myInput.value)">log</button>

In the class

```
logMessage(value)
{
  console.log(value)
}
```

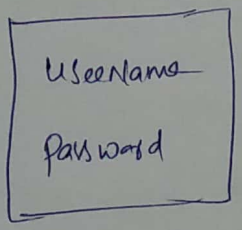
→ Reference variable can be used to map all the HTML elements & its values

10. Two Way Data Binding

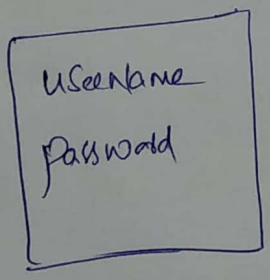
→ When we work with Angular forms, Model and View are in Sync. Otherwise data might not be Consistent.

Eg:

View



Class



→ Whenever Username is updated, Corresponding class Variable has to be updated.

→ We can acheive two way binding by using ngmodel directive

Example

In the class

public name = " "

In the HTML

<input type = "text">

{{ name }}

p To Make Two way data binding, we use ngmodel

<input [(ngModel)] = "name" type = "text">

{{name}}

p We need to import ngModel directive