4. Angular (Latest Stable Version)

- **4. Angular (Latest Stable Version)**
- 4.1. Introduction (Features and Advantage)
- 4.2. Type Script
- 4.3. Modules
- 4.4. Components
- 4.5. Directives, Expression, Filters
- 4.6. Dependency Injection
- 4.7. Services
- 4.8. Routing
- 4.9. SPA (Single Page Application)
- Extra Reading: Data binding, property binding, Event
- Binding, Two-way data binding, String Interpolation.

Java Script

- When JavaScript was created, it initially had another name: “LiveScript”.
- But Java was very popular at that time, so it was decided that positioning a new language as a “younger brother” of Java would help.
- *JavaScript* was initially created to “make web pages alive”.
- It’s one of the core technologies of web development and can be used on both the front-end and the back-end.

- It is used for building client side applications using HTML, CSS and the programming languages like Java Script and Type Script.
- Angular is not a programming language instead it is framework which uses programming languages like java script or type script.

Version	Name	Release Year	
ES1	ECMAScript 1	1997	Initial Release
ES2	ECMAScript 2	1998	Minor Editorial Changes
ES3	ECMAScript 3	1999	Added: Regular Expression, try/catch Exception Handling switch case and do-while
ES4	ECMAScript 4		Abandoned due to conflicts
ES5	ECMAScript 5	2009	Added: •JavaScript "strict mode" •JSON support •JS getters and setters
ES6	ECMAScript 2015	2015	Added: •let and const •Class declaration •import and export •for..of loop •Arrow functions
	CMAScript 2016	2016	Added: •Block scope for variable •async/await •Array includes function •Exponentiation Operator

Version	Name	Release Year	
	ECMAScript 2017	2017	Added: •Object.values •Object.entries •Object.getOwnPropertiesDescriptors
	ECMAScript 2018	2018	Added: •spread operator •rest parameters
	ECMAScript 2019	2019	Added: •Array.flat() •Array.flatMap() •Array.sort is now stable
	ECMAScript 2020	2020	Added: •BigInt primitive type •nullish coalescing operator
	ECMAScript 2021	2021	Added: •String.replaceAll() Method •Promise.any() Method
	ECMAScript 2022	2022	Added: •Top-level await •New class elements •Static block inside classes
ES.Next			Dynamic name for upcoming versions

- **Note:** Older versions of browsers do not support ES6.
- ECMA: European Computer Manufacturers Association

- There are many useful **Javascript frameworks** and libraries available:
- Angular
- React
- jQuery
- Vue.js
- Ext.js
- Ember.js
- Meteor
- Mithril
- Node.js
- Polymer
- Aurelia
- Backbone.js

- **Applications of JavaScript**
- **Client side validation:** to verify any user input before submitting it to the server
- **User Notifications** - You can use Javascript to raise dynamic pop-ups on the webpages to give different types of notifications to your website visitors.
- **Back-end Data Loading** - Javascript provides Ajax library which helps in loading back-end data while you are doing some other processing.

- **Server Applications** - Node JS is built on Chrome's Javascript runtime for building fast and scalable network applications. This is an event based library which helps in developing very sophisticated server applications including Web Servers.

```
<html>
  <body>
    <script language = "javascript" type = "text/javascript">
      <!--
        document.write("Hello World!")
      //-->
    </script>
  </body>
</html>
```

Use strict directive

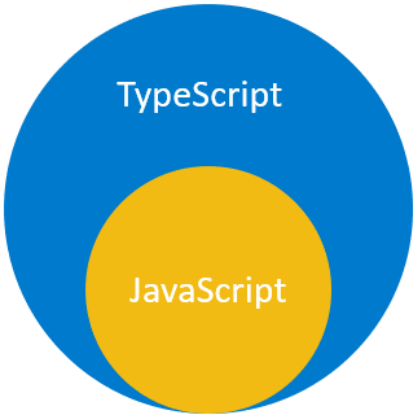
- The purpose of "use strict" is to indicate that the code should be executed in "strict mode".
- With strict mode, you can not, for example, use undeclared variables.
- Strict mode is declared by adding "use strict"; to the beginning of a script or a function.
- Declared at the beginning of a script, it has global scope (all code in the script will execute in strict mode):

- **Example:**
- <!DOCTYPE html>
- <html>
- <body>
- <h2>With "use strict":</h2>
- <h3>Using a variable without declaring it, is not allowed.</h3>
- <script>
- "use strict";
- x = 3.14; // This will cause an error (x is not defined).
- </script>
- </body>
- </html>

Type Script

- Type Script is a types superset of JavaScript developed by Microsoft.
- TypeScript builds on top of JavaScript.
- First, you write the TypeScript code. Then, you compile the TypeScript code into plain JavaScript code using a TypeScript compiler.
- **Angular 2.0** is written in Type Script.
- TypeScript files use the **.ts** extension rather than the **.js** extension of JavaScript files.

- TypeScript uses the JavaScript syntaxes and adds additional syntaxes for supporting Types.
- If you have a JavaScript program that doesn’t have any syntax errors, it is also a TypeScript program.
- It means that all JavaScript programs are TypeScript programs. This is very helpful if you’re migrating an existing JavaScript codebase to TypeScript.



Advantages of TypeScript

- Typescript is superset of javascript.
- Typescript has additional features which do not exist in current version of javascript.
- Typescript is strongly typed.But javascript is dynamically typed.
- Typescript has few object oriented features which do not have in javascript like interfaces, access modifiers, properties etc.
- With typescript we catch errors at compile time instead of run time.

- **Typescript Benefits**
- Intelligence
- Auto completion
- Code navigation
- Strong Typing
- Support ES 2015 features like class, interface and inheritance

Components of TypeScript

- TypeScript is internally divided into 3 main layers -
- 1 Language
- 2 TypeScript Compiler
- 3 TypeScript Language Services
- **1. Language**
- It comprises elements like syntax, keywords, and type annotations.

- **2. The TypeScript Compiler**
- The TypeScript compiler (TSC) transform the TypeScript program equivalent to its JavaScript code.
- It also performs the parsing, and type checking of our TypeScript code to JavaScript code.
- Browser doesn't support the execution of TypeScript code directly.
- So the program written in TypeScript must be re-written in JavaScript equivalent code which supports the execution of code in the browser directly.
- To perform this, TypeScript comes with TypeScript compiler named "tsc."

- We can install the TypeScript compiler by locally, globally, or both with any **npm** package.
- Once installation completes, we can compile the TypeScript file by running "tsc" command on the command line.
- Example: tsc helloworld.ts

- **Compiler Configuration**
- The TypeScript compiler configuration is given in **tsconfig.json file.**

- **tsconfig.json**
- {
- "compilerOptions": {
- "declaration": **true**,
- "emitDecoratorMetadata": **false**,
- "experimentalDecorators": **false**,
- "module": "none",
- "moduleResolution": "node",
- "noFallthroughCasesInSwitch": **false**,
- "noImplicitAny": **false**,
- "noImplicitReturns": **false**,
- "removeComments": **false**,
- "sourceMap": **false**,
- "strictNullChecks": **false**,
- "target": "es3"
- },
- "compileOnSave": **true**
- }

- **3. The TypeScript Language Services:**
- The language service provides information which helps editors and other tools to give better assistance features such as automated refactoring and IntelliSense.
- an additional layer of editor-like applications, such as statement completion, signature help, code formatting, and colorization, among other things.

Steps To execute a TypeScript Program

- 1. Install node js(nodejs.org).
- In command prompt run command: **node -v**
- 2. install typescript
- In command prompt run command:
- **npm install –g typescript** (install typescript globally)
- 3. command : **tsc –v** (know the version of typescript)
- 4. editor to type code(VSCode)

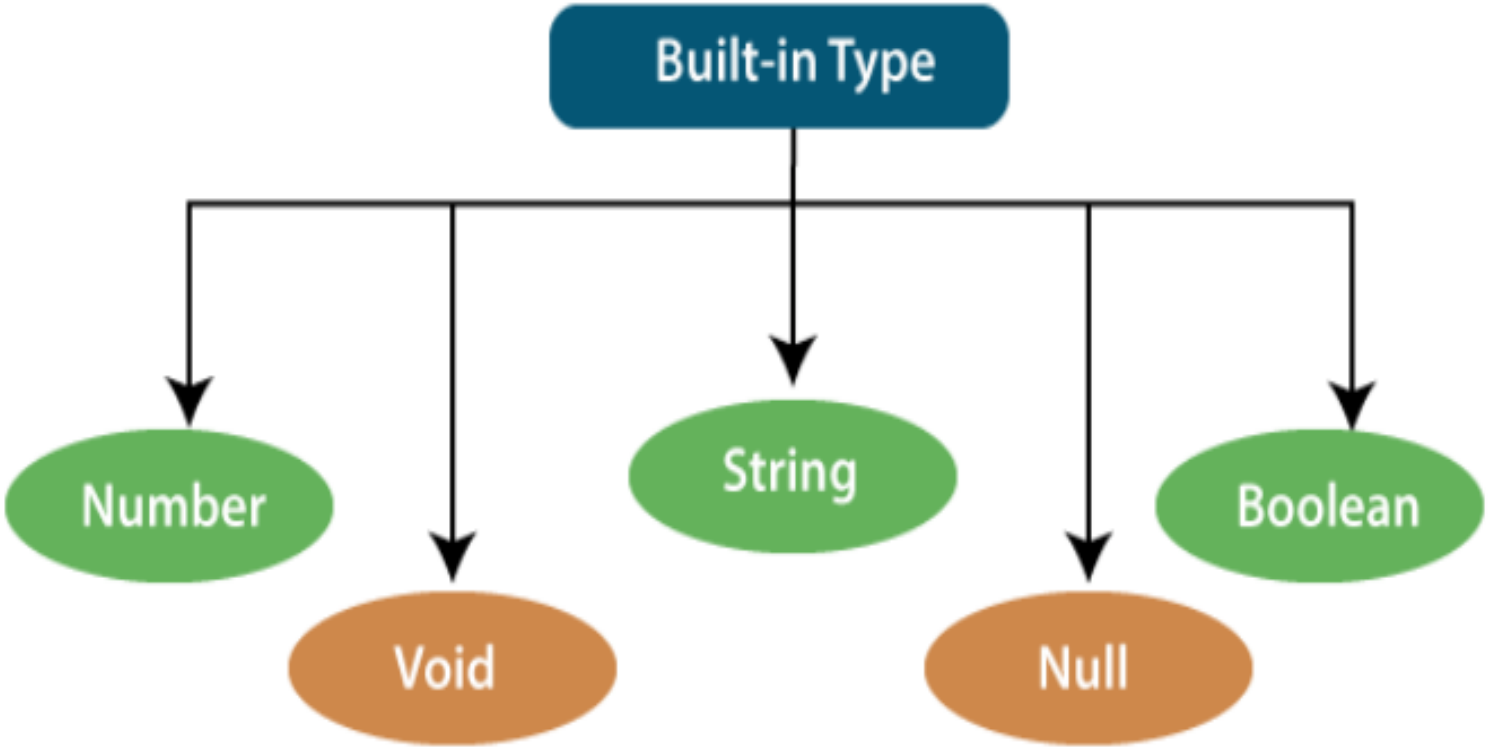
- **5. main.ts**
- let msg='Hello World';
- console.log(msg);
- 6. open terminal and type command
- **tsc main.ts (compilation)**
- 7. To Execute the code type: **node main.js**

- **[Note: To generate tsconfig.json file command**
- **tsc - -init**

- Error in main.ts:

- This happens because file is treated as script rather than a module. A module has its own scope script shares the global scope.
- To get rid of this error add export{} at the top.
- By adding this typescript treat this as module instead of a script.

TypeScript Data Type



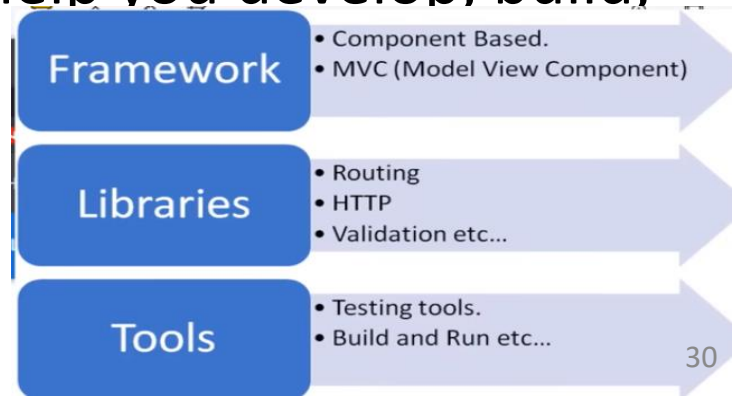
- **Number:** let first:number = 123;
- **String:** let employeeName:string = ‘abc';
- **Boolean:** let isPresent:boolean = true;
- **Void:** is a return type of the functions which do not return any type of value
- function helloUser(): void {
- alert("This is a welcome message");
- }
- **Null:**
- Null represents a variable whose value is undefined.
- let num: number = null;

- **any:** The **any type** in TypeScript is a generic type used when a variable's type is unknown or when the variable's type hasn't yet been defined.
- **Functions**
- A function is the logical blocks of code to organize the program.
- function greet(name: string)
- {
- console.log("Hello, " + name.toUpperCase() + "!!");
- }

TYPESCRIPT	JAVASCRIPT
❑ TypeScript is an object-oriented programming language	❑ JavaScript is a scripting language
❑ TypeScript supports static typing	❑ JavaScript does not support static typing
❑ Errors can be found and corrected during compile time	❑ Errors can be found only during run-time as it is an interpreted language
❑ There is support for ES3, ES4, ES5 and ES6	❑ No support for compiling additional ES3, ES4, ES5 or ES6 features
❑ Functions can have optional parameters	❑ This feature is not possible in JavaScript
❑ Converted into JavaScript code to be understandable for browsers	❑ Can be directly used in browsers
❑ Proper build setup (npm package) is required for static type definitions	❑ No build setup is required

Introduction

- Angular is a development platform, built on TypeScript.
- Platform means it will provide a well structured framework, libraries, tool sets and so on. By which we can create applications.
- As a platform, Angular includes:
  - A component-based framework for building scalable web applications
  - A collection of well-integrated libraries that cover a wide variety of features, including routing, forms management, client-server communication, and more
  - A suite of developer tools to help you develop, build, test, and update your code



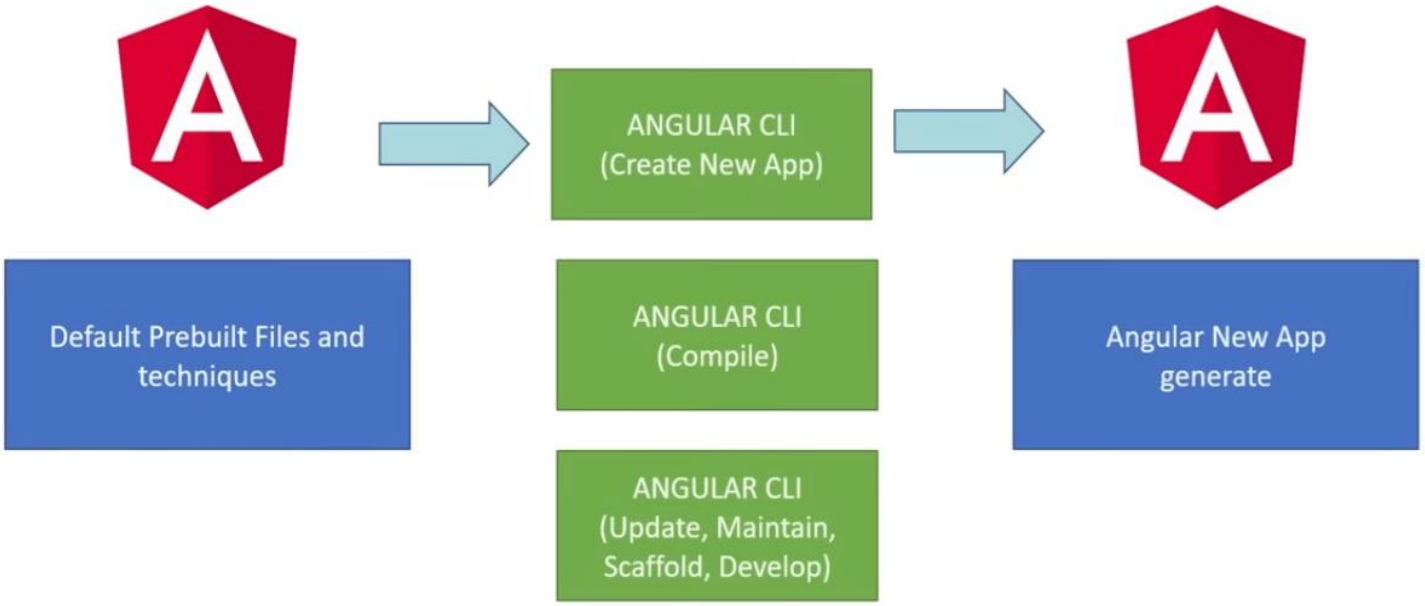
- Angular is one of the most popular JavaScript framework for building client side applications. And it provides lot of reusable code like predefined methods, classes, interfaces etc, which we can use to create dynamic client side applications.
- Angular is a front end development framework to develop single page applications for mobile and desktop.
- Developed by Google.

- **Framework:** It is a platform for developing software applications.
- A framework can have predefined classes and functions that can be reused to add several functionalities which otherwise we would have to write it from scratch by our own.
- **Single Page Application:**
- Single page application is an app that does not need to reload the during its use. It works within a web browser.
- Eg. Facebook, Gmap,Gmail,Twitter, Google Drive, GitHub,Youtube

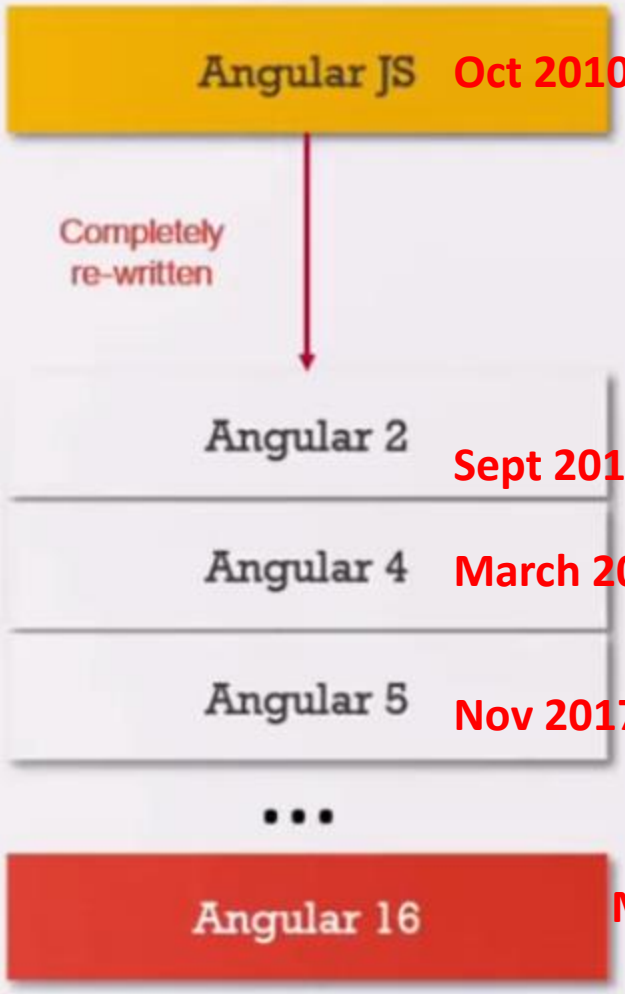
- Most resources of SPA application needs like HTML,CSS and JS unloading at the launch of the app & don’t need to be reloaded during usage.
- The only thing that changes is the data that is transmitted to and from the server as a result the application is very responsive to users queries and doesn’t have to wait for client server communication all the time.
- The best advantages of the SPA is a user experience and a speed. User enjoy natural environment of the app without having to wait for the page reloads and other things. You remain on the page powered by java script.

Angular CLI

- Angular CLI is a command line interface that used to initialize, develop, and maintain angular applications directly from a command shell.

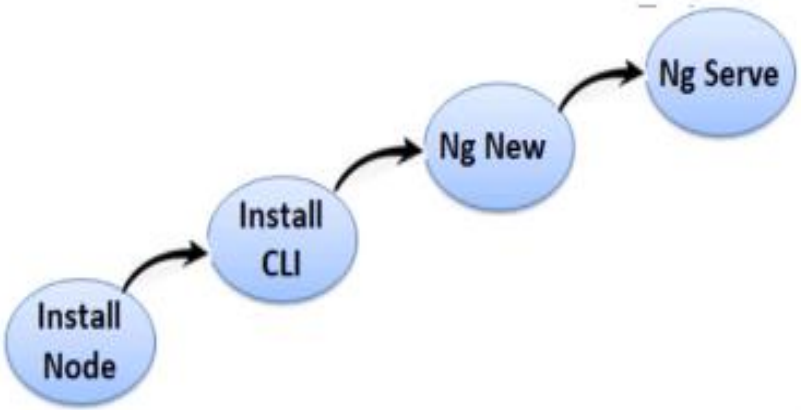


Versions of Angular



1. Angular JS was not designed with the need of todays applications.
2. Angular 2 was completely re-written from ground up.
3. It fixed all the issues which Angular 1 had.
4. There had not been major changes after the release of Angular 2.

- **Getting Started with Angular**
- 1.Install Node.JS
- (website: nodejs.org)
- Command:

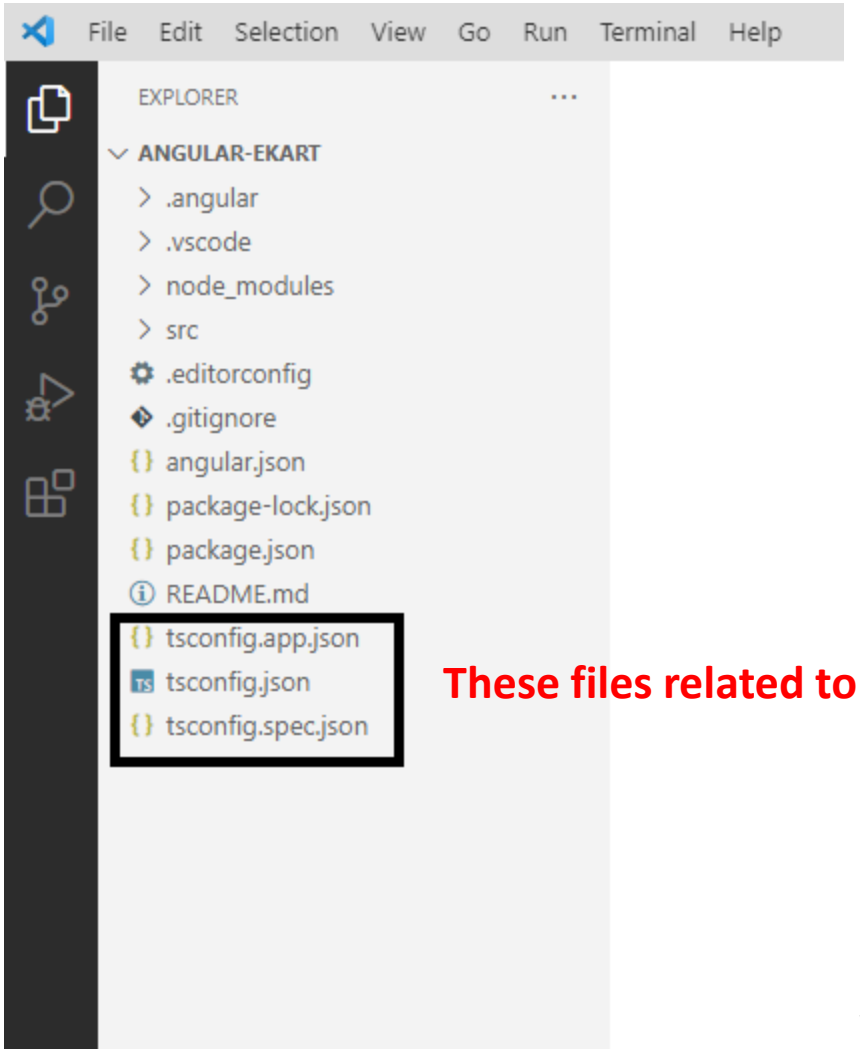


- **node --version or node –v (to see node version)**
- **npm –v (to see npm version)**
- 2. Install Angular CLI (It is command line interface which we use to create new angular projects.
- Commands:
- **npm install –g @angular/cli@latest**
- **ng version (To see version of CLI)**
- **(Note: npm: node package manager helps to install java script package in your computer)**

- **3.Create angular Project**
- First create the folder (eg Angular Programs).
- To create a new angular project move to the folder where you wanted to create the project using command prompt or terminal and type following command.
- **Syntax: ng new project-name**
- **Eg. ng new angular-ekart**
- **(Questions asked)**
- ?Would you like to add Angular routing? No
- ? Which stylesheet format would you like to use? CSS
- (now go to the folder and check)

- Angular Routing: It helps you to navigate, helps to define a navigation from one screen to another screen

- Angular is using Typescript instead of java script for better maintainable bug free code.



These files related to typescript

- **Compile and Run**
- To run the angular project move to the project folder using command prompt/terminal and type the following command:
- **ng serve**
- (compile the angular project and it will generate bundles for java script and css and then open a live development server on which we can run angular project)
- 1. **cd angular-ekart**
- 2. **ng serve** (use VSCode Terminal)
- 3. Now we can open angular project on localhost:4200
- 4. copy the url http://localhost:4200/ and past on browser. (angular project is running)

- ng serve command first compile our project and creates some bundles like runtime.js, polyfills.js,styles.js etc. And then inject these bundles in index.html file.

- <app-root> in index.html is basically rendering a component.

- **Files:**
- package.json: for configuring npm, the dependancies, the references
- tsconfig.json: to configure the typescript
- angular.json: to configure the angular
- tsconfig.spec.json: related to unit testing.
- node_module folder: has all packages.all packages listed package.json file you will find that in this folder.
- src folder: where we write our source code.

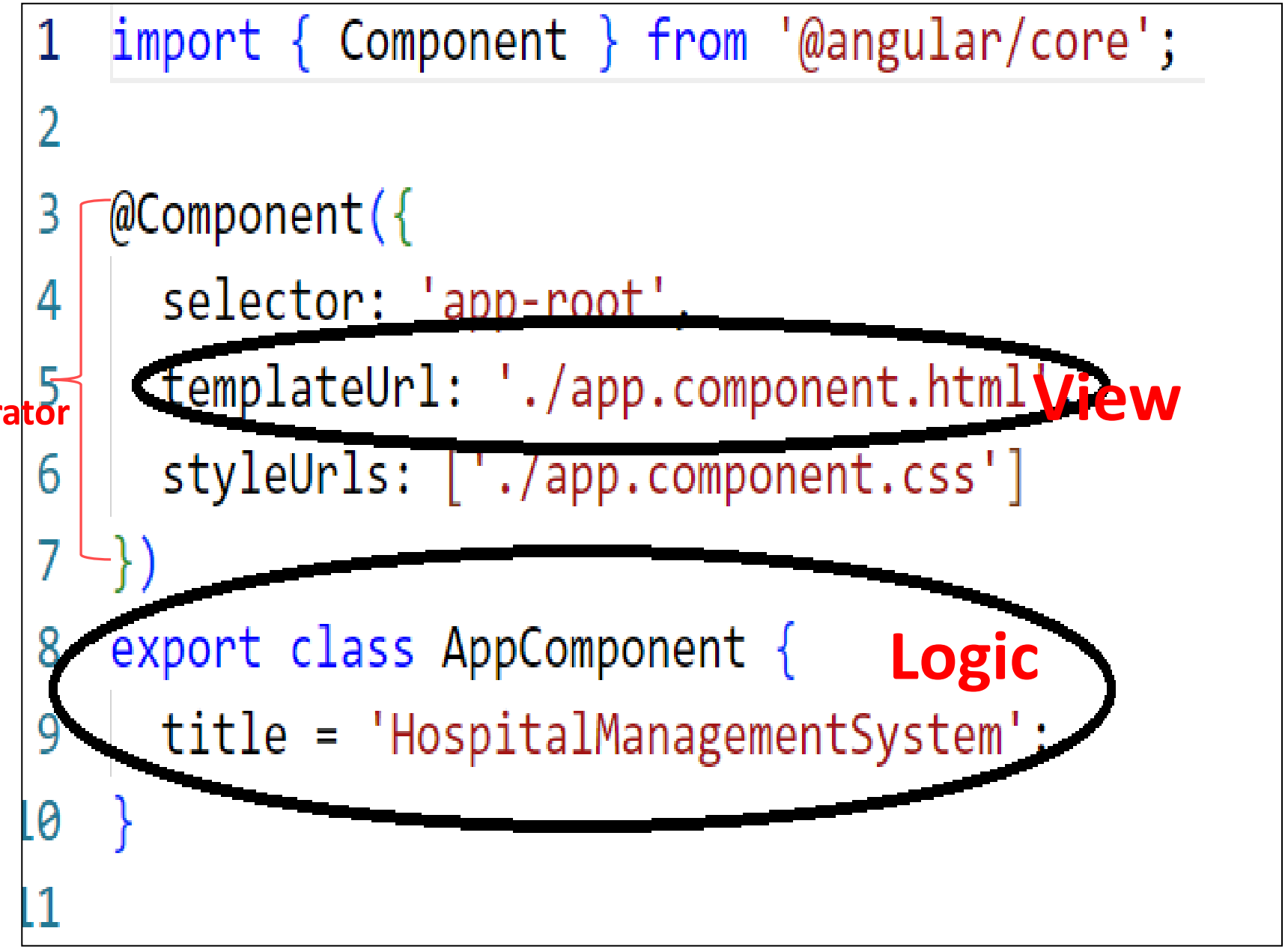
Component

- Component is a typescript class decorated with **@Component** decorator and it contains methods and properties which we can use in HTML.
- Root component can have several nested components.

- Example

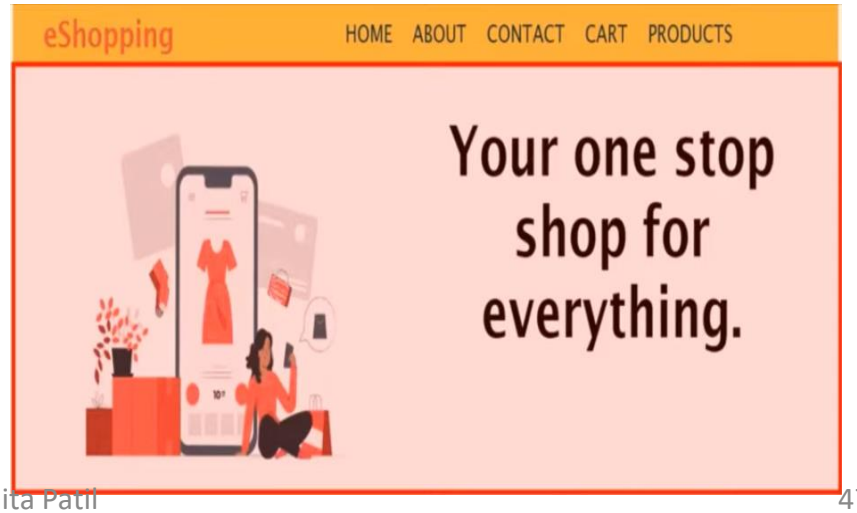

WEBSITE HEADER
HOME
ABOUT
CONTACT
PRODUCTS
HOME
Lorem ipsum dolor sit amet. Sed quibusdam voluptates aut voluptas quibusdam vel earum facere et voluptatibus ratione.
FOOTER

- For a component:
- A **class**: it contains the code required for view template. That means it contains the UI logic.
- A **View Template**: it defines the user interface. It contains HTML directive and data binding.
- A **decorator**: it adds metadata to a class making it a component.

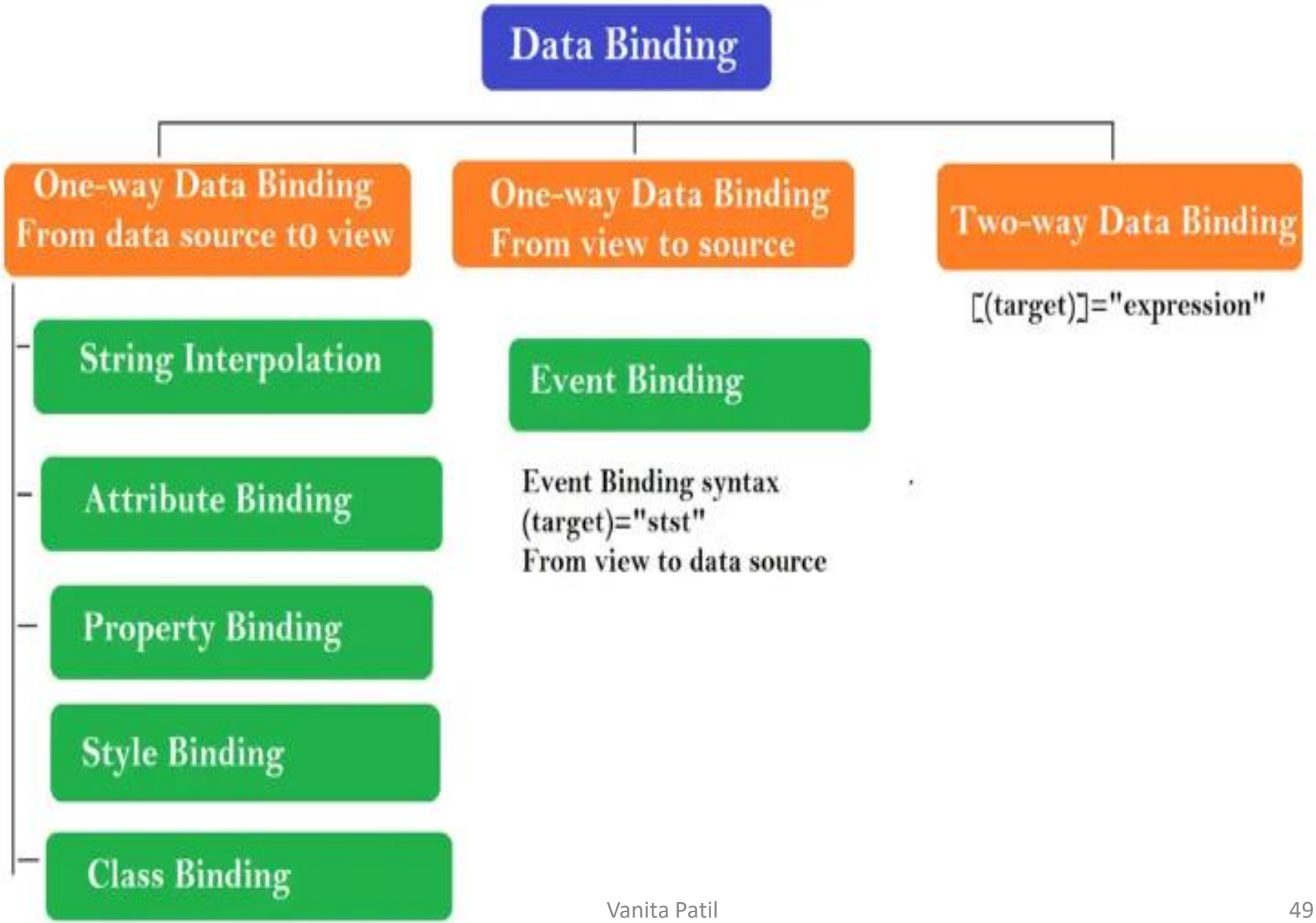

1 import { Component } from '@angular/core';
2
3 @Component({
4 selector: 'app-root',
5 templateUrl: './app.component.html',
6 styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9 title = 'HospitalManagementSystem';
10 }
11
Decorator
View
Logic

Creating a custom component

- To design this we need three components.
- Componet1: which contain header and navbar.
- Component2: for navbar
- Componet3: for header
- Then we will place this container component in AppComponent.

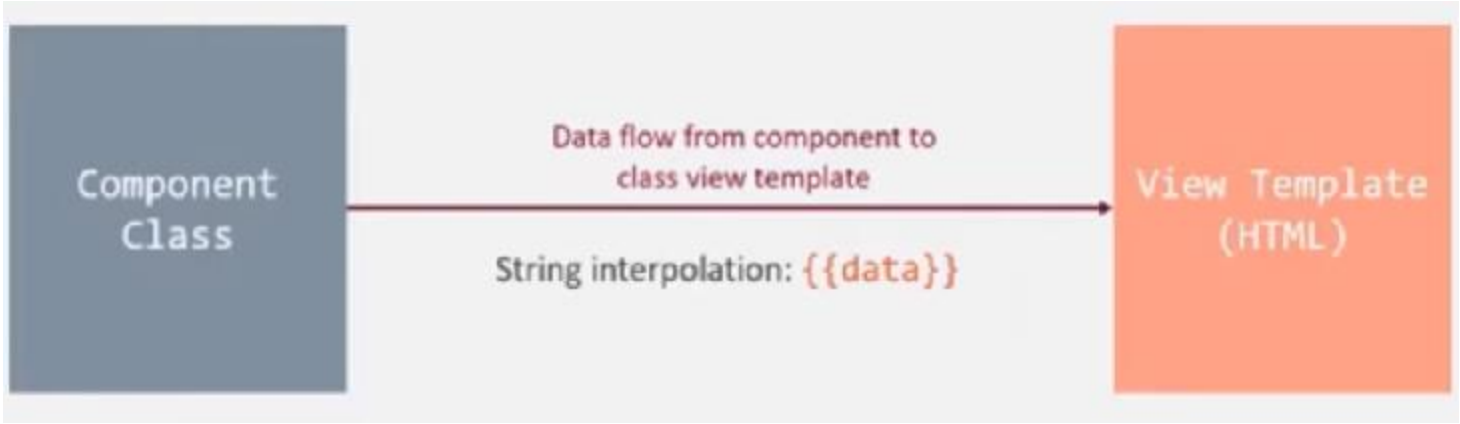


- **To create a component:**
- ng generate component demo


Data Binding
One-way Data Binding From data source t0 view
One-way Data Binding From view to source
Two-way Data Binding
[(target)]="expression"
String Interpolation
Attribute Binding
Property Binding
Style Binding
Class Binding
Event Binding
Event Binding syntax
(target)="stst"
From view to data source

String Interpolation

- String interpolation is angular is used to bind data from component class to view template.
- String interpolation used for one way data binding.



- **header.component.ts**
- export class HeaderComponent {
- slogan:string="Your one stop shop for everything";
- getSlogan()
- {
- return "new slogan for web page using function"
- }
- }

- **header.component.html**
-
-
- <img src="/assets/shopping.jpg" height="240" width="320">
-
-
- <h2>{{getSlogan()}}</h2>
-
-
-

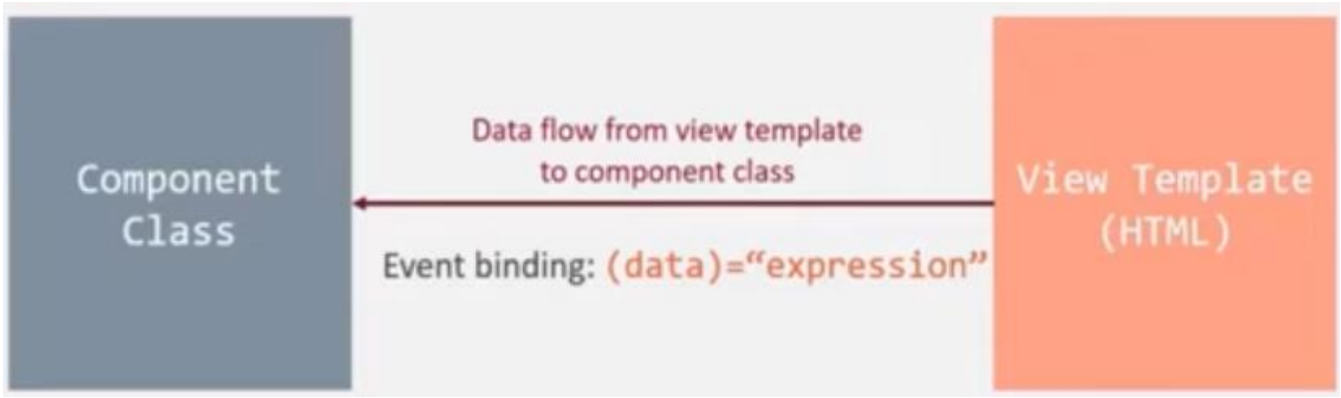
- <!-- Normal HTML -->
- <input placeholder="some text">

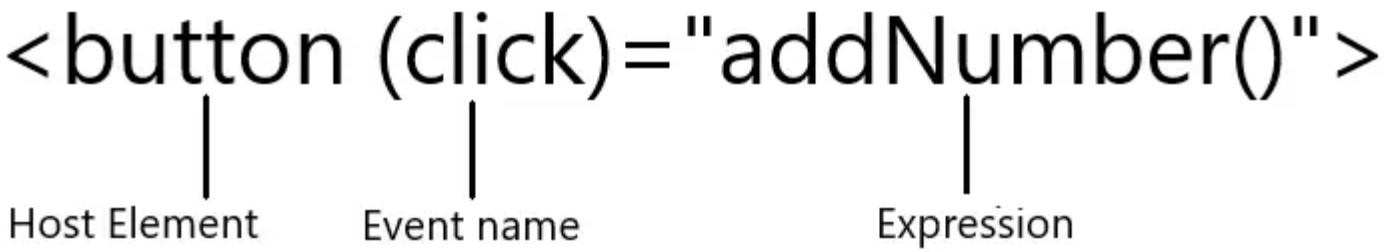
- <!-- Interpolation -->
- <input placeholder="{{ variable }}">

- <!-- Property Binding -->
- <input [placeholder]="variable">

Event Binding

- Event binding allows us to bind web page events to a components property or a methods.
- Using event binding we can pass data from view to component.
- Event binding lets you listen for and respond to user actions such as keystrokes, mouse movements, clicks, and touches.





Angular event binding syntax

- Angular event binding has the following parts:
  - The host element is the source of events for the binding.
  - The event name is what type of event to bind to the host element
  - Expression is evaluated when the event is triggered, it can be an expression or method with and without parameters.

- **Types of Angular event listener or Angular events list**

Event name	Description
(click)	The click event occurs when an element is clicked.
(change)	The change event is triggered when change occurs on the binding elements, like select, Textarea, input, and other elements.
(dblclick)	The double-click event occurs when an element is clicked two times.
(blur)	The *blur event* fires when an element has lost focus.
(submit)	The submit event fire when clicking on the button or inputting with type submit.
(focus)	The *focus event* fires when an element has received *focus*
(scroll)	The scroll event fires when the document view has been scrolled.
(keyup)	When a user presses and releases a key, an event occurs and is mostly used with input fields. It is one of the keyboard events.
(keydown)	*The keydown event* is fired when a key is pressed. It is one of the keyboard events.
(keypress)	The *keypress event* is fired when a key that produces a character value is pressed down. It is one of the keyboard events.

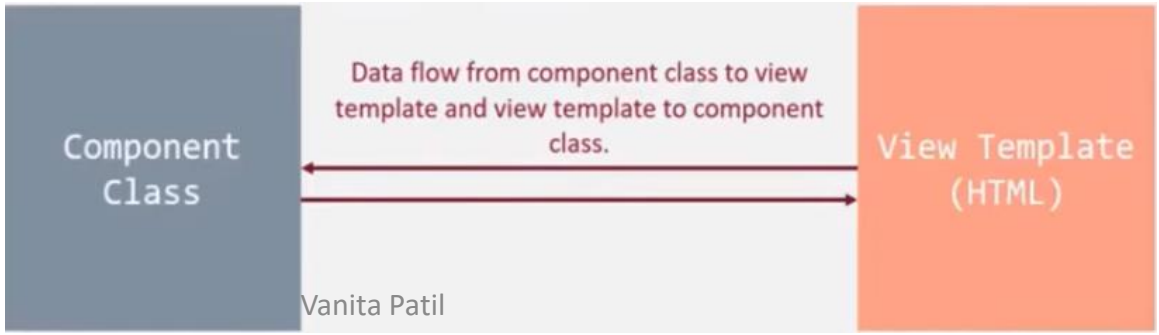
(mousedown)	The *mousedown event* is fired at an Element when a pointing device button is pressed while the pointer is inside the element and is a mouse event.
(mouseup)	The mouseup *event* occurs when a user releases a mouse button over an element and is a mouse event.
(mouseenter)	The mouseenter *event* occurs when the mouse pointer is moved onto an element and is a mouse event.
(mouseleave)	The mouseleave *event* occurs when the mouse pointer is moved out of an element and is a mouse event.
(mousemove)	The mousemove *event* occurs when the pointer is moving while it is over an element and is a mouse event.
(mouseover)	The *mouseover event* occurs when the mouse pointer is over the selected element and is a mouse event.
(mouseup)	The mouseup *event* occurs when a user releases a mouse button over an element and is a mouse event.
(copy)	The copy *event* occurs when the user *copies* the content of an element.
(paste)	The past *event* occurs when the user pastes the content of an element.
(cut)	The cut *event* occurs when the user *cuts* the content of an element.
(drag)	The drag *event* occurs when an element or text selection is being *dragged*
(drop)	The drop event occurs when dragged data is *dropped*.
(dragover)	The dragover *event* occurs when a draggable element or text selection is being dragged over a valid drop target.
(input)	The input *event* occurs when an element gets user *input*.

- **Handling events**
- A common way to handle events is to pass the event object, **$event**, to the method handling the event.
- The **$event** object often contains information the method needs, such as a user's name or an image URL.
  Example:
- <input type="text" (input)="changeSearchValue($event)">

- **Example:**
- **search.component.html**
-
- Search:
- <input type="text" (input)="changeSearchValue($event)">
- You search for {{searchValue}}
-
-
-

- **search.component.ts**
- import { Component } from '@angular/core';
  @Component({
- selector: 'app-search',
- templateUrl: './search.component.html',
- styleUrls: ['./search.component.css']
- })
- export class SearchComponent
- {
- searchValue:string='';
- changeSearchValue(eventData:Event)
- {
- //console.log((<HTMLInputElement>eventData.target).value);
- this.searchValue=(<HTMLInputElement>eventData.target).value;
- }
- }
-

- **Two way data binding:**
- Using two way data binding binds data from component class to view template and view template to component class.
- This is a combination of property binding and event binding.
- For property binding use square brackets [ ] and for event binding use parenthesis ( ). And use directive **ngModel**.



- ngModel: directive to bind the value in the <input> element and allow two-way data binding.
- To use **ngModel** we have to import **FormModule** in **app.module.ts** file.
- **app.module.ts file:**
- import { FormsModule } from '@angular/forms';

- **Also register this FormModule inside imports:**

- imports: [
- BrowserModule,
- FormsModule
-],

- **search.component.html**
-
- Search:
- <input type="text" [(ngModel)]="searchValue">
- You search for {{searchValue}}
-

- **seach.comonent.ts**
- export class SearchComponent
- {
- searchValue:string='iphone';
- changeSearchValue(eventData:Event)
- {
- this.searchValue=(<HTMLInputElement>eventData.target).value;
- }
- }

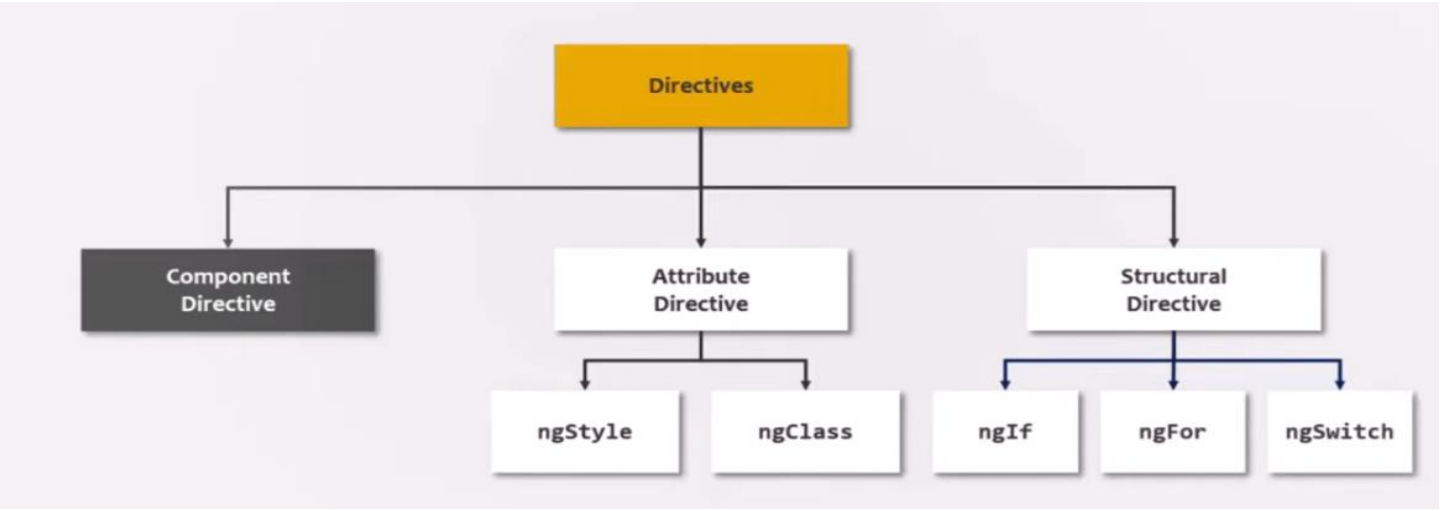
- **Add a table row dynamically using two way data binding:**
- **app.component.html**
- <input type="text" [(ngModel)]="someValue">
- <input type="button" value="Click" (click)="CallSomeLogic()"/>
- {{someValue}}
- <table border="1">
- <tr>
- <td>Name</td>
- </tr>
- <tr *ngFor="let temp of someValues">
- <td>
- {{temp}}
- </td>
- </tr>
- </table>

- **app.component.ts**
- import { Component } from '@angular/core';
  @Component({
- selector: 'app-root',
- templateUrl: './app.component.html',
- styleUrls: ['./app.component.css']
- })
- export class AppComponent
- {
- title = 'HospitalManagementSystem';
- someValue:string="";
- someValues:Array<string>=new Array<string>();
- CallSomeLogic()
- {
- //alert("Hello");
- this.someValues.push(this.someValue);
- this.someValue="";
- }
- }

Directive

- Directive is an instruction to the DOM.
- Using directive we tell angular how DOM elements should behave and look. And also which DOM element should add to the page and which one to not add.
- We can change DOM elements appearance, behavior and layout using directives.

- Directives are classified in 3 ways:



- Component directive is nothing but a angular components. components are also kinds of instruction to the DOM. Where ever we use component there we instruct angular to render **view template** of component.
- Components are directives but directives with the template.

- **Structural Directive:**
- Change the DOM layout by adding and removing DOM elements from webpage.
- Whenever we use structural directive before it we use *(astrick) eg <div *ngIf> .
- It tells angular we are using structural directive.

- **ngFor Directive:**
- ngFor directive iterates over collection of data like an array, list etc. and creates an html element for each of the items from an HTML template.
- ngFor directive is structural directive.
- It manipulate the DOM by adding or removing elements.
- **Syntax**
- <div *ngFor="let item of elements; index as i">
- ...
-

- **Example:1**
- <div *ngFor ="let item of [2,3,4,5,6]; let i=index">
- {{i+1}}: Current element of array is {{item}}
-
- **Example2:**
- **TS file:**
- @Component({
- selector: 'my-app',
- templateUrl: './app.component.html',
- })
- export class AppComponent {
- fruits = ['apple', 'pear', 'banana', 'coconut'];
- }
- **HTML file:**
-
- <li *ngFor="let fruit of fruits">{{ fruit }}
-

- **Example 3:**

```
export class AppComponent {
  title = 'angular-project';
  friendslist = [
    {
      name: 'Nishant',
      age: 25
    },
    {
      name: 'Shailesh',
      age: 45
    },
    {
      name: 'Abhishek',
      age: 36
    },
    {
      name: 'Akshay',
      age: 65
    },
    {
      name: 'Ashish',
      age: 12
    },
    {
      name: 'Uday',
      age: 31
    },
    {
      name: 'Mayank',
      age: 45
    },
]   }
```

- HTML:
-
- <li *ngFor="let item of friendslist">
- {{ item.name }} {{ item.age }}
-
-

- ***ngIf:**
- it is structural directive.
- It is used to add or remove element from the web page based on given condition.
- If condition returns true, it will add the element, otherwise if the condition is false it will remove the element from the web page.
- **Syntax:**
- <li *ngIf='condition'>
- **Example: search component:**
- <span *ngIf="searchValue!=''">You search for {{searchValue}}

- The ng-template directive allows us to run a set of HTML tag blocks based on ngIf|else condition.
- This directive is useful for showing or hiding a section of the component.

- <ng-container ***ngIf="condition; else #conditionFalse"**>
- Render content when condition is true
- </ng-container>
- <ng-template **#conditionFalse**>
- Render content when condition is false
- </ng-template>

- **Program: angular ngIf directive to check odd and even numbers.**
- evenodd.component.html
- <div *ngFor="let n of numbers">
- <div *ngIf="n % 2 == 0; else showOdd">
- Even number {{ n }}
-
- <ng-template #showOdd>
- Odd number {{ n }}
- </ng-template>
-

- evenodd.component.ts
- export class EvenoddComponent
- {
- numbers: number[] = [1, 2, 3, 4, 5, 6, 7, 8];
- }

- evenodd.component.css
- .even{
- background:pink;
- padding:2px;
- }
- .odd{
- background:yellowgreen;
- padding:2px;
- }
- div{
- margin-bottom:10px;
- }

- **app.component.html**
- <h4>NgIf</h4>
- <ul *ngFor="let person of people">
- <li *ngIf="person.age < 30"> (1)
- {{ person.name }} ({{ person.age }})
-
-

- **app.component.ts**
- export class AppComponent {
- title = 'project-name';
- people: any[] = [
- {
- "name": “aaa",
- "age": 35
- },
- {
- "name": “bbb",
- "age": 32
- },
- {
- "name": “ccc",
- "age": 21
- },
- {
- "name": “ddd",
- "age": 34
- },
- {
- "name": “eee",
- "age": 32
- }
-]; }

- **Example 2:**
- @Component({
- selector: 'my-app',
- template: `
- <div *ngIf="userLoggedIn; else userloggedOut">
- Hello User
-
- <ng-template #userloggedOut>
- Hello User, Login
- </ng-template>
- `,
- })
- export class AppComponent {
- userLoggedIn = false;
- userloggedOut = true;
- }

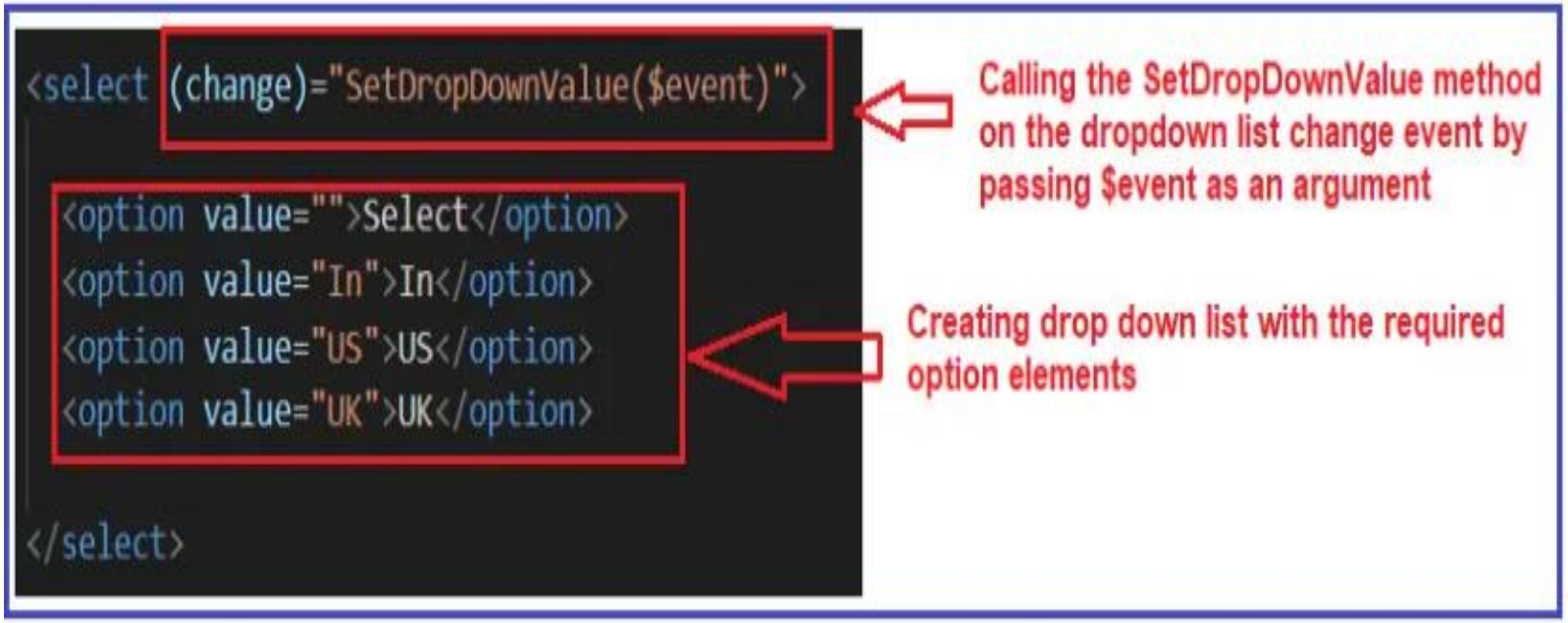
- ngSwitch:
- The **[ngSwitch]** directive on a container specifies an expression to match against. The expressions to match are provided by **ngSwitchCase** directives on views within the container.
- **Syntax:**
- **<container-element [ngSwitch]="switch_expression">**
- **<element *ngSwitchCase="condition1">Case 1 content </element>**
- **<element *ngSwitchCase="condition2">Case 2 content </element>**
- **....**
- **<element *ngSwitchDefault>Default content</element>**
- **</container-element>**

- Angular *ngSwitch directive has the following sub-elements.
- **ngSwitch:** is a structural directive holding all expression bodies and it holds a variable that compares with ngSwitchCase.
- ***ngSwitchCase**: have an expression for each matching condition and will render the corresponding template if its condition matches that of the NgSwitch have.
- ***ngSwitchDefault**: element is optional and it will render when any of the match conditions is failed.

- **app.compoenet.ts**

```
export class AppComponent {
  title = 'project-name';
  public dropDownValue = "";
     SetDropDownValue(drpValue : any)
    {
      this.dropDownValue = drpValue.target.value;
    }
}
```

- **app.component.html**
- <h2>Select Country</h2>
  <select (change)="SetDropDownValue($event)">
- <option value="">Select</option>
- <option value="In">IN</option>
- <option value="US">US</option>
- <option value="UK">UK</option>
- </select>
-
  <h2>You Have Selected</h2>
  <div [ngSwitch] = 'dropDownValue'>
- <h3 *ngSwitchCase="'In'">India</h3>
- <h3 *ngSwitchCase="'US'">United State</h3>
- <h3 *ngSwitchCase="'UK'">United Kingdom</h3>
- <h3 *ngSwitchDefault="">You have not selected any country</h3>
-


<select (change)="SetDropDownValue($event)">
<option value="">Select</option>
<option value="In">In</option>
<option value="US">US</option>
<option value="UK">UK</option>
</select>
Calling the SetDropDownValue method on the dropdown list change event by passing $event as an argument
Creating drop down list with the required option elements


<div [ngSwitch] = 'dropDownValue'>
<h3 *ngSwitchCase="'In'">India</h3>
<h3 *ngSwitchCase="'US'">United State</h3>
<h3 *ngSwitchCase="'UK'">United Kingdom</h3>
<h3 *ngSwitchDefault="">You have not selected any country</h3>

Switch Statement
Cases
Default case

- **Attribute Directive**:
- Attribute directives **allow us to change the appearance or behavior of an element**, component, or another directive. Like changing element color, background, and more.
- Attribute directive does not add or remove the elements from the web page.

ngStyle Directive

- It is an attribute directive which changes look and behavior of an HTML element.
- It is used to set CSS style dynamically for an HTML element based on a given typescript expression.
- **Example1:**
- **<div [ngStyle]="{'background-color':'green'}"></ **

- ngStyle becomes much more useful when the value is *dynamic*.
- The *values* in the object literal that we assign to ngStyle can be JavaScript expressions which are evaluated and the result of that expression is used as the value of the CSS property, like this:

- **Example2:**
- **Change a color of available products:**
- <div class="details" [ngStyle]="{color:p.available==='Available'?'Green':'Red'}"> {{p.available}}

- **Change background color of available and not available Products:**
- <div class="product-container"[ngStyle]="{backgroundColor:p.available==='Available'?'#D5F5E3':'#FADBD8'}">

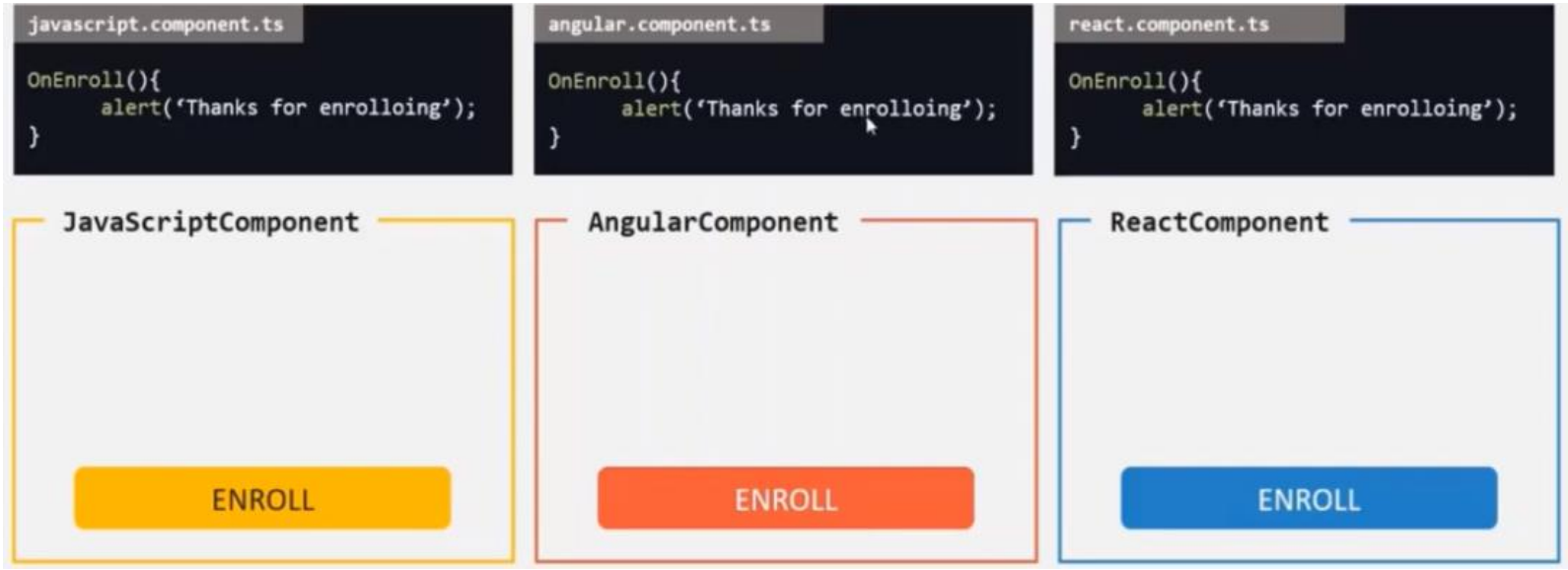
ngClass Directive

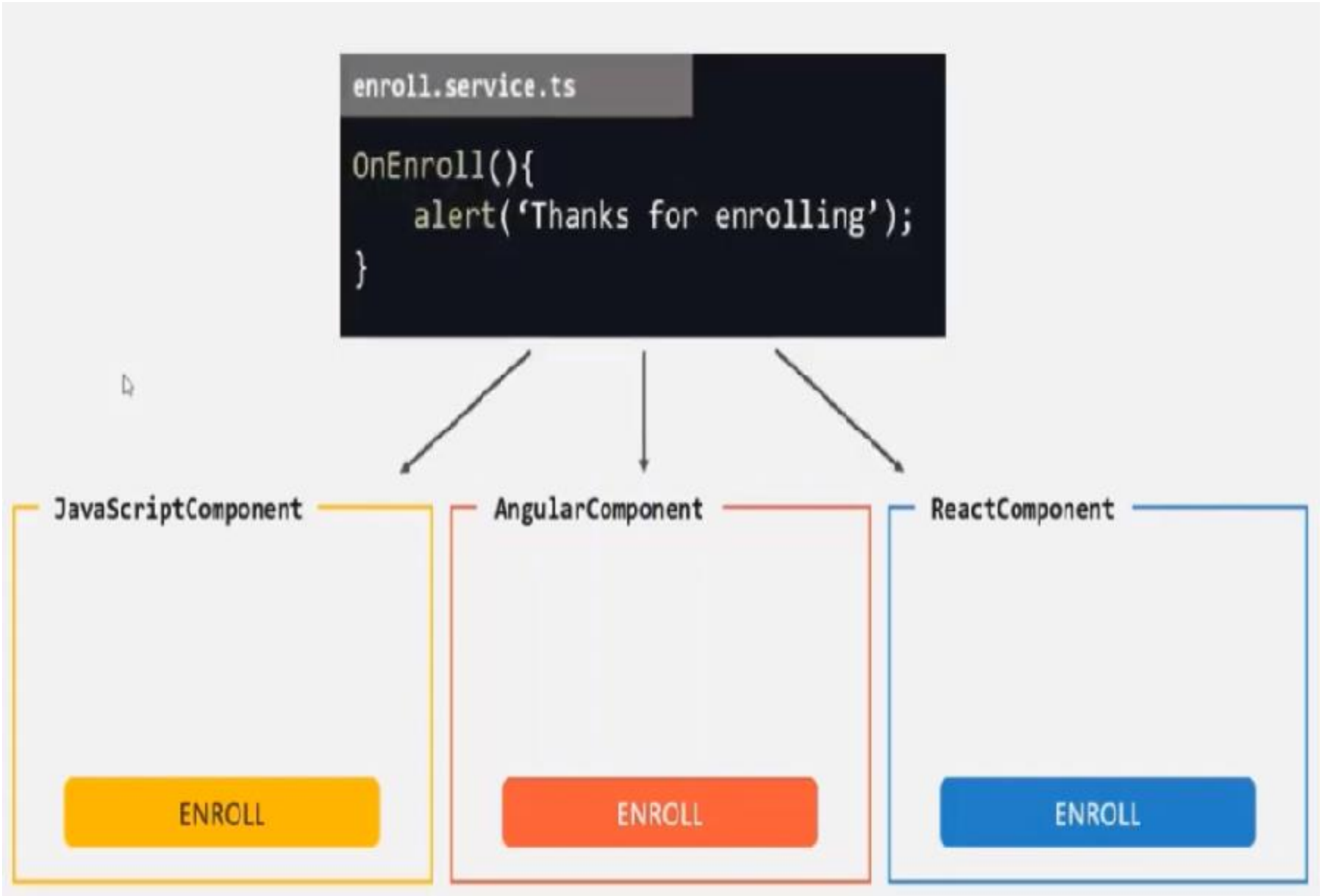
- It is an attribute directive.
- Used to add CSS class dynamically to the web page element.
- Example:
- Whenever user enter something in textbox background color of search div should change to orange otherwise it should be transparent.

- **search.component.css:**
- .changeBackground{
- background-color:#FAD7A0;
- }
- **search.component.html:**
- <div class="search-div" [ngClass]="{changeBackground:searchValue!=''}">
-

Services and Dependency Injection in Angular

- A service is a reusable typescript class that can be used in multiple components across your angular application.




enroll.service.ts
OnEnroll(){
alert('Thanks for enrolling');
}
JavaScriptComponent
ENROLL
AngularComponent
ENROLL
ReactComponent
ENROLL

- Example 2:

ComponentA.component.ts

```
Products = [
 {id: 101, name: 'Laptop', price: 789, available: true},
 {id: 202, name: 'Mobile', price: 349, available: false},
 {id: 303, name: 'Desktop', price: 529, available: true}
]
```

ComponentB.component.ts

```
Products = [
 {id: 101, name: 'Laptop', price: 789, available: true},
 {id: 202, name: 'Mobile', price: 349, available: false},
 {id: 303, name: 'Desktop', price: 529, available: true}
]
```

ComponentA

ID	Name	Price	Availability
101	Laptop	$789	Available
202	Mobile	$349	Not Available
303	Desktop	$529	Available

ComponentB

ID	Name	Price	Availability
101	Laptop	$789	Available
202	Mobile	$349	Not Available
303	Desktop	$529	Available

products.service.ts

```
Products = [
 {id: 101, name: 'Laptop', price: 789, available: true},
 {id: 202, name: 'Mobile', price: 349, available: false},
 {id: 303, name: 'Desktop', price: 529, available: true}
]
```

ComponentA

ID	Name	Price	Availability
101	Laptop	$789	Available
202	Mobile	$349	Not Available
303	Desktop	$529	Available

ComponentB

ID	Name	Price	Availability
101	Laptop	$789	Available
202	Mobile	$349	Not Available
303	Desktop	$529	Available

- **Advantages of services:**
- Services are easy to debug and test.
- Services provides reusability of the code.
- With services we can communicate across different components.

- angular/javascript.component.html
-
- <img src="assets/js.png" style="width:240px;
- height:180px;">
-
- <h3>{{title}}</h3>
-
-
- <button (click)="OnEnroll()">Enroll</button>
-
-

- angular/javascript.component.ts
- export class JavascriptComponent
- {
- title="JavaScript";
- OnEnroll(){
- alert("Thank You for enrolling to“ +this.title + “ course.");
- }

- **To create a services:**
- 1. Create a **Services** folder inside app folder.
- 2. create a file say **enroll.service.ts**
- 3. create a **service class**. (service class is a simple class without decorator).
- export class EnrollService{
- OnEnrollClicked(title:string){
- alert("Thank You for enrolling to"+title+"course.");
- }
- }
- 4.In **javascript.component.ts** file create one method and create an instance of class EnrollService.

- OnEnroll(){
- const enrollService=new EnrollService(); //instance of class
- }

- 5. import a class in component file.
- import { EnrollService } from '../Services/enroll.service';
- 6. on the instance call a method.
- OnEnroll(){
- const enrollService=new EnrollService();
- **enrollService.OnEnrollClicked(this.title);**
- }
- 8. write a click event of button.
- <button (click)="OnEnroll()">Enroll</button>

Dependency Injection

- We are using **EnrollService** class from .services in javascript as well as angular component.
- For javascript as well as angular component **EnrollService** class is dependancy because we want to use **OnEnrollClicked()** method in both.
- So to use this we need to create an instance of a class and on that instance we call the method.
- **enrollService.OnEnrollClicked(this.title);**
- So instead of creating an instance of a class angular offers tool to access our services that is **angular dependency injector.**
- **dependency injector simply inject the dependency so we do not have to create an instance of EnrollService class by our own that instance will be provided by angular.**

- Definition: Dependency injection is technique in which class receives dependencies from external sources rather than creating them itself.
- Remove a line
- const enrollService=new EnrollService(); //instance
- **Now we need to inform angular that we need instance of EnrollService class in both components.**
- **For that we need to**
- 1. create a constructor for java script and angular class.

- constructor(private enrollService:EnrollService){
-
  }

- **2. specify providers property for component decorator.**
- @Component({
- selector: 'app-angular',
- templateUrl: './angular.component.html',
- styleUrls: ['./angular.component.css'],
- providers:[EnrollService]
- })
- 3. OnEnroll(){
- this.enrollService.OnEnrollClicked(this.title);
- }

Routing

- In Angular, routing plays a vital role since it is essentially used to create Single Page Applications.
- Outing allows us to navigate from one part of application to another part or from view of one component to view of another component.
- The Router is a separate module in Angular.
- It is in its own library package, **@angular/router**.
- The Angular Router provides the necessary service providers and directives for navigating through application views.

Routing Components

- **Router**
- An **Angular Router** is a service (Angular Router API) that enables navigation from one component to the next component as users perform application tasks like clicking on menus links, and buttons, or clicking on the back/forward button on the browser.
- **Route**
- Route tells the Angular Router which view to display when a user clicks a link or pastes a URL into the browser address bar. **Every Route consists of a path and a component it is mapped to**.

- **Routes**
- Routes is an array of Route objects our application supports
- **RouterLink**
- The RouterLink is a directive that binds the HTML element to a Route. Clicking on the HTML element, which is bound to a RouterLink, will result in navigation to the Route. The RouterLink may contain parameters to be passed to the route's component.

- **How to configure Angular Router**
- 1. **Define the routes**
- create an array of route objects. Each route maps the path (URL Segment) to the component.
- const appRoutes={ path: 'product', component: ProductComponent }
- Where
- **path:** The URL path segment of the route. We will use this value to refer to this route elsewhere in the app
- **component:** The component to be loaded.
- This route tells angular to render ProductComponent when the user navigates to the URL “/product”

- **2. Register the Routes**
- Import the Angular Router from **@angular/router** library in the root module of the application.
- import { RouterModule } from '@angular/router';
- Then, install the routes using the RouterModule.forRoot method, passing the routes as the argument in the imports array
- imports: [RouterModule.forRoot(appRoutes)],
- **3. Map Action to Routes**
- we need to bind the click event of the link, image, or button to a route. This is done using the routerlink directive.
- <a [routerLink]="['product']">Product

- **Default Route**
- { path: '', redirectTo: 'home', pathMatch: 'full' },
- The path is empty, indicating the default route.
- The default route is redirected to the home path using the RedirectTo argument.
- This route means that, when you navigate to the root of your application /, you are redirected to the home path (/home), which in turn displays the HomeComponent.
- Note, that we have pathMatch argument set to ‘full’.
- The pathMatch tells the Router how to match the URL.
- When it is set to full, the path is matched to the entire URL
- Every route ends in an empty space for ex: /contact/’’. If pathMatch is not set to full then the router will apply the redirect, which results in the error.

- **Wild Card Route**
- { path: '**', component: ErrorComponent }
- The “**” matches every URL. The Router will display the ErrorComponent.

- Instead of displaying the view of each of these components in the page we want display only that view for which the user has clicked the link.

- Creating a routing in angular:
- Create route in separate typescript file or inside the app.module.
- 1. create a constant and import
- import { Routes } from '@angular/router';
- const appRoute:Routes

- 2. It is an array.
- Inside this array we specify the routes objects.
- const appRoute:Routes=[
- {path:'Home',component:HomeComponent},
- {path:'About',component:AboutComponent},
- {path:'Contact',component:ContactComponent},
- {path:'Courses',component:CoursesComponent}
-]

- **3. import RouterModule in imports[].**
- imports: [
- BrowserModule,
- AppRoutingModule,
- RouterModule.forRoot(appRoute)
-]
- With this angular application knows about the routes.
- **4.** Tell angular where to display a view when one of these paths are typed in URL.
- Display a view inside the app.componet.html
- **<router-outlet></router-outlet>**

- <router-outlet> is a directive that tells angular where in our page we want to display a view.
- This directive is imported when we import router module in app.module file.

Pipes in Angular

- Pipes in angular are used to transform a data before displaying it in the view.
- Angular pipes takes data as input and format or transform the data to display in template.
- **1. DatePipe**
- Formats a date value according to locale rules.
- Examples:
- {{ dateObj | date }} // output is 'Jun 15, 2015'
- {{ dateObj | date:'medium' }} // output is 'Jun 15, 2015, 9:43:11 PM'
- {{ dateObj | date:'shortTime' }} // output is '9:43 PM'
- {{ dateObj | date:'mm:ss' }} // output is '43:11'

- **2. PercentPipe**
- Transforms a number to a percentage string.
- <!--output '26%'-->
- A: {{a | percent}}
- **3. UpperCasePipe**
- Transforms text to all upper case.
- **4. LowerCasePipe**
- Transforms text to all lower case.
- **5. TitleCasePipe**
- Transforms text to title case. Capitalizes the first letter of each word and transforms the rest of the word to lower case.
- **Example:**
- In lowercase: <pre>'{{value | lowercase}}'</pre>
- In uppercase: <pre>'{{value | uppercase}}'</pre**>**
- {{'some string' | titlecase}}

- **6. CurrencyPipe**
- Transforms a number to a currency string.
- <!--output ₹10,000.00'-->
- A: {{a | currency:'INR'}}

Angular Filter

- Filters can be added in Angular to format data to display on UI without changing the original format.
- Filters can be added to an expression or directives using the pipe | symbol.
- **Syntax:**
- {{expression | filterName:parameter }}

- **Angular Filters:** Angular provides filters to transform data of different data types.
- The following table shows the significant filters:
- **currency**: It is used to convert a number into a currency format.
- **date**: It is used to convert a date into a specified format.
- **filter**: It is used to filter the array and object elements and return the filtered items.
- **json**: To convert a JavaScript object into JSON.

- **limitTo**: It is used to return an array or a string that contains a specified number of elements.
- **lowercase**: It is used to convert a string into lowercase letters.
- **uppercase**: It is used to convert a string into uppercase letters.
- **number**: It is used to convert a number into a string or text.
- **orderBy**: It sorts an array based on specified predicate expressions.

- END