**Angular Framework**

**It is framework based on HTML, CSS and JS and it is used to develop the web applications that run inside the browser.**

**Why to use the angular framework?**

**It gives our application a clear structure.**

**It has the re-usable codes.**

**Make our application more testable.**

**Angular follows component-based design for developing the web application**

**Components**

**A Component is group of data, html template, logic for the area of the screen which user see. E.g navbar, header, footer, rating system, course section everything is considered as the component.**

**Module**

**A Module is group of related component.**

**Dependency Injection**



**Ng serve**

**Ng Serve will use webpack tool to bundle all the files into single .js, .jpg, .png, .css,… files.**

**Property Binding Syntax (Only for known properties of elements)**

**@Component({**

**template = `<img [src]= “imageUrl/>”` 🡪 property binding**

**})**

**Export class CoursesComponent{**

**imageUrl = “”**

**}**

**Attribute Binding(Those attribute not known to elements)**

**Use attr.attribute-name**

**@Component({**

**template = `<table>**

**<tr>**

**<td [attr.colSpan]=""></td>🡪 attribute binding**

**</tr>**

**</table>`**

**})**

**Export class CoursesComponent{}**

**Class Binding**

**Use: class.class-name**

**@Component({**

**Template = `<button [class.activeClass]]= “isActive”> Save </button>`**

**})**

**Export class CourseComponent{**

**isActive: Boolean = true/false; 🡪 if true class will be applied or else not.**

**}**

**Style Binding**

**Use: [style.style-property]**

**@Component({**

**Template = `<h1 [style.backgroundColor]]= “red” > Save </h1>`**

**})**

**Export class CourseComponent{}**

**Event Binding**

**Use = (click)= “function-name()”**

**@Component({**

**Template = `<button (click)= “onSave()”> Save </button>`**

**})**

**Export class CourseComponent{**

**onSave(){}**

**}**

**Event Filtering**

**Filter key code from key events.**

@Component({

Template = `<input type="text"(keyup.enter)="onKeyDownEnter()"/>`

})

Export class CourseComponent{

onKeyDownEnter(){}

}

**Template Variable**

**Use: #variable-name in element to declare the template variable.**

**Example:**

**@Component({**

**Template: `<input #email (key.enter)= “onKeyEnter(email.value)”`>**

**})**

**Export class CourseComponent{**

**onKeyEnter(emailValue: string){**

**console.log(emailValue);**

**}**

**}**

**Two Way Binding (Component to View & View to Component)**

**Method 1:**

@Component({

Template = `

<input type="text" [value]='email'

(key.enter)="email= '$event.target.value';onKeyEnter()"/>

`

})

Export class CourseComponent{

Email: string;

}

**Method 2:**

**ngModel is the directive used to do two-way binding.**

**<input [(ngModel)]="email" (keyup.enter)="onKeyEnter()"/>**

**The change in input field will be detected in the variable email of the component class.**

**PIPES**

**It is used to format the data.**

**Build-in Pipes: Uppercase, LowerCase, Currency, Decimal, Percentage**

**Example:**

**@Component({**

**selector: 'courses',**

**template: `{{course.title | uppercase}}<br>**

**{{course.students | number}}**

**`**

**})**

**Course.title will pass through uppercase pipe and result will be in uppercase format.**

**Course.students value will be displayed in number format.**

**Decimal Format:**

**number: ‘no\_of\_digits.min\_decimal – max\_decimal’**

**Example:**

**Number: 4.975**

{{course.rating | number:'1.2-2'}}

**// means number displayed as 1 digit and minimum, maximum 2 decimal places i.e. 4.97**

**Currency Format:**

**Example:**

**Currency: ‘country code’: ‘apply symbol’: ‘decimal format’**

**{{ course.price | currency:'INR':true: '3.2-2'}}**

**Custom Pipe**

**Make a file pipe-name.pipe.ts**

**Exmaple: pipe-name is summary.**

**Import Pipe and PipeTransform**

**In Pipe decorator add field name and mention pipe-name.**

**Implement PipeTransform Interface and implement transform function.**

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

@Pipe({

    name: 'summary'

})

export class SummaryPipe implements PipeTransform{

    transform(value: any, limit?: number) {

        if(!value)

            return null;

            let actualLimit = (limit)?limit: 50;

        return value.substr(0, actualLimit) + "...";

    }

}

**Lastly within modules add new Pipe Class name in declarations array.**

**Directives**

**Directives are classes that additional behavior to our elements.**

**Directives are of two types:**

**Structural: Modify structure of the DOM**

**Attributes: Modify attributes of the elements.**

1. **Ng-if**

**Approach 1:**

<div \*ngIf="courses.length > 0">

    List Of Courses

</div>

<div \*ngIf="courses.length == 0">

    No Courses Yet

</div>

**Approach 2:**

<div \*ngIf="courses.length > 0; else noCourses">

    List Of Courses

</div>

<ng-template #noCourses>

    No Courses Yet

</ng-template>

**Approach 3:**

<div \*ngIf="courses.length > 0;then courseList else noCourses"></div>

<ng-template #courseList>

    List Of Courses

</ng-template>

<ng-template #noCourses>

    No Courses Yet

</ng-template>

**Hidden Property**

<div [hidden]="courses.length == 0">

    List Of Courses

</div>

<div [hidden]="courses.length > 0">

    No Courses Yet

</div>

**When we use hidden property the element exist in DOM but it is just hidden and while using ngIf the element is removed from DOM.**

**ngFor**

**Example 1:**

**html file**

<ul>

    <li \*ngFor="let course of courses">

       {{course.id}} - {{course.name}}

    </li>

</ul>

**Example 2:**

**html file: exported values aliased to local variables.**

<ul>

    <li \*ngFor="let course of courses; index as i; even as isEven">

       {{i}} - {{course.name}} - <span \*ngIf="isEven;">even</span>

    </li>

</ul>

**Class file**

courses = [

    {id: 1, name: 'Course1'},

    {id: 2, name: 'Course2'},

    {id: 3, name: 'Course3'}

  ];

The following exported values can be aliased to local variables:

* $implicit: T: The value of the individual items in the iterable ([ngForOf](https://angular.io/api/common/NgForOf)).
* [ngForOf](https://angular.io/api/common/NgForOf): [NgIterable](https://angular.io/api/core/NgIterable)<T>: The value of the iterable expression. Useful when the expression is more complex then a property access, for example when using the async pipe (userStreams | [async](https://angular.io/api/common/AsyncPipe)).
* index: number: The index of the current item in the iterable.
* count: number: The length of the iterable.
* first: boolean: True when the item is the first item in the iterable.
* last: boolean: True when the item is the last item in the iterable.
* even: boolean: True when the item has an even index in the iterable.
* odd: boolean: True when the item has an odd index in the iterable.

**ngFor to Change Detection**

<ul>

    <li \*ngFor="let course of courses">

       {{course.id}} - {{course.name}}

       <button class="btn btn-primary" (click)="onChange(course)">Update</button>

    </li>

</ul>

**Class file**

  onChange(course: courseType){

    course.name = "updated";

  }

**“SPA” Application**

**Patterns to study:**

**Dependency Injection Pattern**

**Factory Pattern**