**Angular learning course from codevolution**

3 ways of components usage

1. Selector: “app-root”…<app-root></app-root>
2. Selector : “.app-root” <div class=”app-root”>
3. Selector : [app-root] <div app-root><style/div>

Inline templates:

Template : ‘app.component.html’

`<div>inline templates</div>

Styles: [`color: red;`]

Interpolation : {{variable}}

Template usage in variables for interpolation:  
template : `<div> hello {{name}}</div>

Export class appcomponent(){

Public name=”srihari”;

}

Assignments are not available

Templates donot have access to global ones

**May 23**

**Attribute vs Property**

* Attributes and Properties are not the same.
* Attributes - HTML
* Properties - DOM (Document Object Model)
* Attributes initialize DOM properties and then they are done. Attribute values cannot change
* once they are initialized.
* Property values however can change.
* Interpolation only works with string values
* It doesn’t cover Boolean values and conditions
* <input [disabled]=”false” value="Vishwas">
* <input bind-disabled=”false” value="Vishwas">

**Class binding**

1. class binding has the preference
2. error testing [class.danger]=”condition value”
3. directive: a custom html attribute that html provides
4. Multiple classes can also be applied by using object parameter ng class directive

**Style binding**

**Event binding**

1. (click)=”onclick()”
2. Onclick(){  
   clg(“hello”;
3. }

**Template reference variable:**

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

@Component({

  selector: 'app-root',

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

  template: `

  <div class="container">

  <h1 class="bg-warningi">Hello world</h1>

    <div class="container">

    <div [ngClass]="succ">Hello world</div>

    <p [class.danger]="errorornot">error test</p>

    <p [ngClass]="mul">How r u</p>

    <p [style.color]="'orange'">oye</p>

    <p [style.color]="errorornot?'red':'green'">Error check using style binding</p>

    </div>

    <!-- event binding -->

    <div class="container">

        <button (click)="onclick($event)">{{dtext}}</button>

        <p [style.color]="display?'green':'red'">Clicked on button event binding with click function and text style display</p>

        <p>{{texter}}</p>

        <button (click)="wish='cliked'">click here</button>

        {{wish}}

    </div>

    <div class="templatereferencevariable">

      <input type="text" #mytext>

      <button (click)="loger(mytext.value)">Log</button>

    </div>

</div>

    `,

  // styleUrl: './app.component.css',

  styles: [`

    .bg-warningi {

      background-color: yellow;

    }

    .danger{

      color:red;

    }

  `]

})

export class AppComponent {

  public succ = "bg-warningi"

  public errorornot = true;

  clicked = true

  dtext = "change to green"

  mul = {

    "danger": this.errorornot,

    "success": !this.errorornot,

    "special": true

  }

  texter = ""

  wish = ""

  onclick(e: any) {

    this.display = !this.display;

    this.dtext = this.display ? "change to red" : "change to green"

    console.log(e.type)

  }

  public display = false;

  loger(texter:any){

    console.log(texter)

  }

  title = 'Tutorial';

}

**Two way binding:**

1. Ng model
2. Structural directives
3. Ngfor
4. Ngswitch
5. Ngif
6. Ngif is similar to display:none property in css
7. Ng template ng if then block and elseblock
8. Ngswitch case now
9. Ngswitch ngswitchcase
10. Ng for loop
    1. Properties
       1. Index as i
       2. First as f
       3. Last as l
       4. Even as e
       5. Odd as o

**Component Interaction**

1. Data sent from parent component to child component
2. Parameters inside input decorator
3. Child to parent also data being sent here
4. Onyly using events child to parent data is being sent

**Pipes**

Only to change views not properties

1. Pipe operator |
2. Lowercase
3. Uppercase
4. Title case
5. Slice : 3:5
6. Json
   1. All these are string pipes
7. number 1.2-3
8. percent
9. currency
10. currency : ‘IN’
11. date
    1. date: ‘short’
    2. shortDate
    3. shortTime
    4. medium
    5. mediumShort
    6. mediumLong

**Services:**

Don’t violate principles like:

1. dry – do not repeat yourself
2. single responsibility principles

a class with a specific purpose

services are for:

* share data
* logic implementation
* external interactions (using dependency injection)

**Dependency Injection**

1. code without di and drawbacks
2. di as design pattern
3. di as a framework angular provides

when dependencies changes, errors occurs

* dependencies changes because of evolution of classes
* when a component uses another component and old component is changes, then new component gets error

With Dl:

class Car{

engine;

tires;

constructor(engine, tires)

this.engine = engine;

}

using variables in place of direct access, is called dependency injection.

Its simple like, parameters are used inside class for inheriting,

var oldEngine = new Engine(oldparameter);

var oldTires = new Tires(oldparameter);

var oldCar= new Car(01dEngine, oldtires);

var newEngine = new Engine(newparameter);

var newTires = new Tires(newparameter);

var newcar= new Car(newEngine, newtires);

Di framework has an injector;

Dl as a framework contd

1) Define the EmployeeService class

2) Register with Injector

3) Declare as dependency in EmpList and EmpDetail

**HTTP and observables:**

HTTP, Observables and RxJS

* 1. HTTP Get request from EmpService
  2. Receive the bbservable and cast it into an employee array
  3. Subscribe to the observable from EmpList and EmpDetail
  4. Assign the employee array to a local variable

RxJS

- Reactive Extensions for Javascript

- External library to work with Observables

Note the difference between httpclientmodule and httpclient because of their usage.

Subscribing to observable services and

Error handling to rxjs

**Routing and routing components:**

1. Routeroutlet
2. Routerlink directive
3. Routerlinkactive

**Wildcard routing and redirecting routes**

1. Wild card route be last route in the paths folder always
2. \*\*
3. Path matching is done from top always
4. Redirectto:’/’ pathmatch:”prefix” and pathmatch=”full”

**Route parameters:**

1. Reading route parameters and using like /departments/it
2. /depts/cse
3. Import router from angualar
4. Private router of type router router:Router
5. onSelect (department){

this . router . navigate( [ ' /departments ' ,department . id] ) ;

**Console application chatgpt code:**

Sure! Here's a simple example of a console application written in C# that prompts the user to enter their name and then greets them with a personalized message:

```csharp

using System;

class Program

{

static void Main()

{

// Prompt the user to enter their name

Console.Write("Enter your name: ");

// Read the user's input

string name = Console.ReadLine();

// Greet the user with a personalized message

Console.WriteLine($"Hello, {name}! Welcome to the C# console application.");

// Wait for the user to press any key before exiting

Console.WriteLine("Press any key to exit...");

Console.ReadKey();

}

}

```

Here's what this program does:

1. It prompts the user to enter their name using `Console.Write("Enter your name: ")`.

2. It reads the user's input using `Console.ReadLine()` and stores it in the variable `name`.

3. It greets the user with a personalized message using `Console.WriteLine($"Hello, {name}! Welcome to the C# console application.")`.

4. It displays a message asking the user to press any key to exit using `Console.WriteLine("Press any key to exit...")`.

5. It waits for the user to press any key to exit the program using `Console.ReadKey()`.

When you run this program, it will display the prompt to enter your name in the console window. After you enter your name and press Enter, it will greet you with a message like "Hello, [Your Name]! Welcome to the C# console application." Then, it will wait for you to press any key to exit the program.

C#

Xaml

http get and post in asp.net web api

api authentications

responses

naming conventions

payloads

positive responses and negative responses

authentication errors

ui and api call

encrypt url

containers

dependency injections

circular dependency injections

syntax and coding

monolith and microservices

razor server

https://www.youtube.com/watch?v=Xvukp6FpUVU