Day 4:

Services:

Services:

Services in Angular:

1. Typescript class which helps us to create reusable functionalities.
2. Services can be injected in components this mechanism called as DI.
3. Services are abstraction layer means in service we can implement our business logic.
4. Services also used making http request calls.
5. Services can have methods as well as common data which we will have to share among multiple components

TestAppliction

App:

Components

CustomerRegist

CustomerLogin

Service:

customerService

6. Services are unlike components that we don’t have to include it in module
7. We can have any number of services
8. Services are also used to share data between components

How add/create service

Ng generate service service_name

In case, if youi want to add service in services folder you have to specify name of folder

Ng g s name_folder/name_of_service

Scenario:

Consider we have two components comp1 and comp2.

Create two components by firing given below command:

1. ng g c comp1
2. ng g c comp2

Open comp1.component.html and add given below code.

```
<h1>Component 1</h1>
<button (click)="getMessage()">Get Message</button>
```

Open comp1.component.ts and given below code.

```
 getMessage(){
    alert("Good Morning");
  }
```

Do same for comp2 component.

Now, open app.component.html and add selectors of both components in **app.component.html**

```
<h1>Comp1</h1>
<app-comp1></app-comp1>

<h1>Comp2</h1>
<app-comp2></app-comp2>
```

But here in both components we have implemented same functionality, it is good if we have small implementation but what if we have very complex implementation and we have 20 components in which we have

same functionalities then it’s better to go with service.

To achieve this follow given below steps.

Step 1:

Fire a command ng g s testservice.

This step will create a file as testservice.service.ts file.

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

@Injectable({
  providedIn: 'root'
})
export class TestserviceService {

  constructor() { }
}
```

Step 2: Add given below code in above file

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

@Injectable({
  providedIn: 'root'
})
export class TestserviceService {

  constructor() { }
}
```

Step 3: Open comp1.component.ts and add given below code.

```
import { Component } from '@angular/core';
import { TestserviceService } from '../testservice.service';

@Component({
  selector: 'app-comp1',
  templateUrl: './comp1.component.html',
  styleUrls: ['./comp1.component.css']
})
export class Comp1Component {
  obj = new TestserviceService();

  getMessage(){
    // alert("Good Morning");
    this.obj.getMessage();
  }

}
```

Add same code in comp2.component.ts file

```
import { Component } from '@angular/core';
import { TestserviceService } from '../testservice.service';

@Component({
  selector: 'app-comp2',
  templateUrl: './comp2.component.html',
  styleUrls: ['./comp2.component.css']
})
export class Comp2Component {
  obj = new TestserviceService();
  getMessage(){
    // alert("Good Morning");
    this.obj.getMessage();
  }
}
```

But here we have created object which is not recommended as Angular supports DI.

I hope you are familiar with DI.

But let’s have look on DI design pattern.

What is DI?

Ans: Example of kid getting food from fridge but parents will takes responsibility to get food from fridge and give it to son.

Same way, in programming some frameworks helps us to create objects and inject them.

Can you tell me the reason why DI comes in picture.

There are four main reason because of which DI is implementing by many frameworks.

1. Class won’t be testable
2. Code is not extensible
3. Single responsibility
4. Lifetime of object

Let’s understand it with example

We have a class Car which is depend on other class that is Engine. It means our class require object of engine class.

So, in traditional way we have to create object of engine in class Car.

```
Class Car{
obj = new Engine();
    drive(){
        obj.start();
    }
}
```

So, let’s understand it’s problems one by one.

1. **Class Car is not testable** because car class depends on engine. It means if we will have to write test case for car then engine class must need to be ready otherwise it won’t be tested.

2. **Car class is not extensible**. It means if there will be some modification will be done in engine class then it won’t be integrate in class Car. It means it's not future ready. Consider if engine class will be modified with parameterized constructor then here we will face challenges.
3. **Single Responsibility:** One of the SOLID principle that is Single responsibility means our class should have only single responsibility but here our class is also creating object in it

which is violating single responsibility principle.

4. **Lifetime of Object:** Here, We can assume that object on which our class is depend should be reusable it means if I have injected it then and only then we can reuse it.
   So, angular provides mechanism of DI. Let’s see how can achieve it.
   To achieve it we will have to inject object with mechanism of dependency injection. In angular we use constructor to perform DI. So, let’s go back and modify our code.

Open comp1.component.ts file and add given below code:

```
import { Component } from '@angular/core';
import { TestserviceService } from '../testservice.service';

@Component({
  selector: 'app-comp1',
  templateUrl: './comp1.component.html',
  styleUrls: ['./comp1.component.css']
})
export class Comp1Component {
  constructor(private obj:TestserviceService){}
  // obj = new TestserviceService();

  getMessage(){
    // alert("Good Morning");
    this.obj.getMessage();
  }

}
```

Open comp2.component.ts file and add given below code.

```
import { Component } from '@angular/core';
import { TestserviceService } from '../testservice.service';

@Component({
  selector: 'app-comp2',
  templateUrl: './comp2.component.html',
  styleUrls: ['./comp2.component.css']
})
export class Comp2Component {
  // obj = new TestserviceService();
  constructor(private obj:TestserviceService){}
  getMessage(){
    // alert("Good Morning");
    this.obj.getMessage();
  }
}
```

Angular Component:
Read points from slide 49 to 54.

Components:

Start with slide 49

After Slide number 54 start go through below info

Demo 1.

1. Use previous project
2. Add new component child1 which will be by default child of app component.
   ng g c child1

3. Add new component as child2 which we want to make child of child1
   ng g c child1/child2
   So, here hierarchy will become something like

   App
     Child1
       Child2
   Check whether these components will be added or not in their parent component.

Step 1. Open app.component.html and add selector of app-child1

```
<h1>Component Relationship Demo</h1>
<app-child1></app-child1>
```

Step 2: Open child1.component.html and add selector of child2

```
<h1>Child 1 Component</h1>
<app-child2></app-child2>
```

So, all child can be integrate easily with their parent component.

Interpolation:

Open child1.component.ts and add given code:

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

@Component({
  selector: 'app-child1',
  templateUrl: './child1.component.html',
  styleUrls: ['./child1.component.css']
})
export class Child1Component {
  compName="Child 2 Component";
  nameOfUser="BhushanHere24";

  getUserName(){
    return this.nameOfUser;
  }

  getTodaysDate(){
    return new Date();
  }

  number1=100;
  number2=200;

  employees={
    ename:"Bhushan",
    eage:42,
    eaddress:"Mumbai",
    email:"bhushan@icloud.com"
  }

  departments=['HR', 'Finanace','IT', 'Staffing','Delivery']

  siteUrl =window.location.href

}
```

Open child1.component.html and add given below code:

```
<h1>Child 1 Component</h1>
<app-child2></app-child2>
<h1>Interpolation Data</h1>
<h2>{{getUserName()}}</h2>
<h2>{{ getTodaysDate() | date:'medium'}}</h2>
<h2>Sum = {{number1+number2}}</h2>
<h2>{{employees}}</h2>
<h2>{{employees | json}}</h2>
<h2>{{employees.ename}}</h2>
<h2>{{departments}}</h2>
<h2>{{departments[0]}}</h2>
<h2>{{siteUrl}}</h2>
```

Property Binding:

Open child1.component.html

And add given below code.

```
<img [src]="imgUrl" alt="logo">
<br/>
<br/>
<img src={{imgUrl}} alt="logo">
<br/>
<br/>
```

Open child1.component.ts file

```
  // Property Binding
imgUrl = "/assets/img/CITIUSTECH-Logo-600x73.png";
```

(before this paste an image in asset).

Difference between Interpolation and Property binding.

Open child1.component.html and add given below code.

```
<!-- Interpolation and Property Binding -->
<input type="text" [disabled]="disableBox" [value]="placeholder">
<br/>
<br/>
<input type="text" disabled={{disableBox}} [value]="placeholder">
<br/>
<br/>
<button (click)="enableMe()">Enable Me</button>
```

Open child1.component.ts and given below code.

```
disableBox=true
enableMe(){
  this.disableBox=false
  this.placeholder=""
}
```

Event Binding: