Page No.
Date

Theory Assignment - 3

Name : Yug P. Parmar

Roll No. : 42

Sem : 7th

Subject : Application Development using Full-Stack (702)

042 - Yug Parmar

Page No. 1
Date

Theory Assignment - 3

☆ Angular Introduction:-

→ Angular is a popular open-source web application framework for building dynamic and single-page applications (SPAs). Developed and maintained by Google, it provides a comprehensive platform for developing web applications with a rich user interface and advanced features. Angular is known for its robust architecture, extensive tooling, and a strong community of developers.

→ Key features of Angular:

- TypeScript Integration:-
Angular is built using Typescript, a statically typed superset of JavaScript. TypeScript provides features like strong typing, interfaces, and code analysis, which make it easier to catch errors during development.

- Componet-Based Architectur:
Angular applications are structured as a collection of components. A component is a self-contained unit that encapsulates both the user interface (view) and the logic (controller) for a specific part of the app.

- Dependency Injection:-
Angular's dependency injection system helps manage the creation and sharing of application services and components.

- Routing:-
Angular provides a provides a powerful routing module that allows you to define routes and navigate between different views or components in your application.

- Forms:-
Angular offers two types of forms : template-driven and ~~reat~~ reactive-forms. These forms provide tools for creating and validating user inputs making it easier to work with complex forms and user interactions.

- Directives:
Angular includes built-in directives like 'ngIf', 'ngFor' and 'ngSwitch' that allow for dynamic manipulation of the Document Object Model (DOM) based on application data.

- Angular is a robust and versatile framework for building web applications of all sizes, from small personal projects to large-scale enterprise solutions. Its powerful features and tools, combined with a strong and active community, make it a top choice for developers looking to create modern and interactive web applications.

★ Application Structure:-

→ Angular applications are structured in a specific way to promote modularity, reusability and maintainibility.

```
- my-app/
    - e2e/
    - node_modules/
    - src/
        - app/
            - components/
            - services/
            - app.module.ts/
        - assets/
        - environments/
        - index.html
        - main.ts
    - angular.json
    - package.json
    - tsconfig.json
```

1) 'e2e/':
- This directory contains end-to-end (E2E) testing configuration and tests. It's used for testing your application's functionality from a user's perspective.

2) 'node_modules/':
This directory stores all the external packages and libraries that your Angular application depends on.

3) 'src/':
- This is main source code directory of your Angular application.
→ 'app/': This directory is the heart of your application includes:
- 'componets/': Angular components are organized here. Each component is stored in its own directory, containing a TypeScript file, an HTML template, a CSS or SCSS file, and a spec file for testing.
- 'services/': Application services, responsible for data retrieval, business logic, or other common tasks, are stored in this directory.
- 'app.module.ts': This file defines the main application module, which is the entry point for the application.

→ 'assets/': This directory holds static assets like images, fonts and other files that are served with your application.

→ 'environments/': Angular allows you to have different environment configurations for development and production.

→ 'index.html': This is the entry point HTML file for your application. It includes essential scripts and styles, and the Angular application is loaded within this HTML

→ 'main.ts': The 'main.ts' file is the application's entry point, where the Angular application is bootstrapped.

4) 'angular.json':
- This configuration file is used to manage various aspects of your Angular project, including build options, paths, and configuration for different environments.

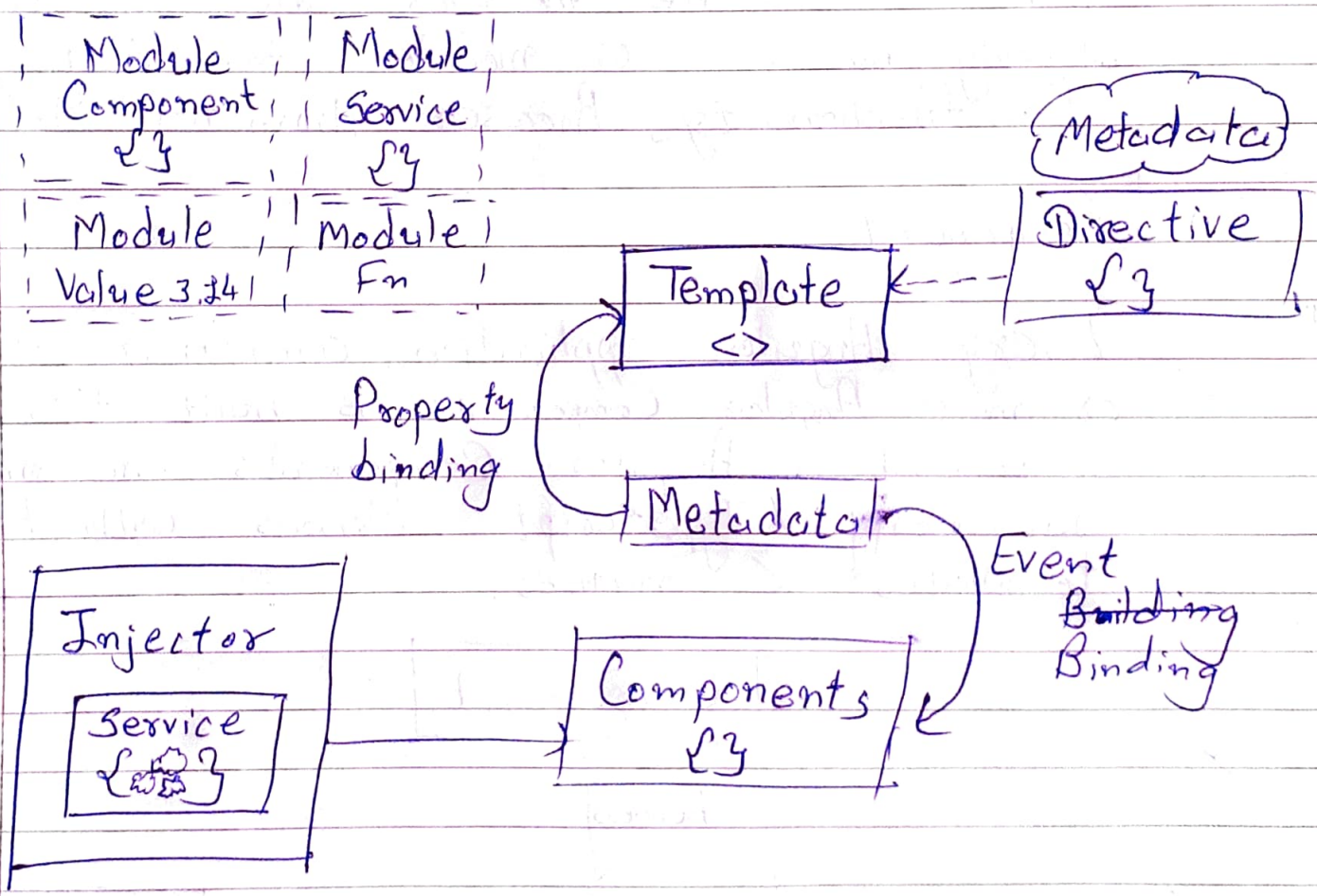
5) 'package.json':
- The 'package.json' file defines the project's dependencies and scripts for building, testing and running your app.

6) 'tsconfig.json':
- The TypeScript configuration file specifies how TypeScript should compile your code.

Page No. 7
Date

✯ Architecture:-

- Angular is an MVC framework that has detailed diagrams on how to create the application & how to pass data between the view & the controller.

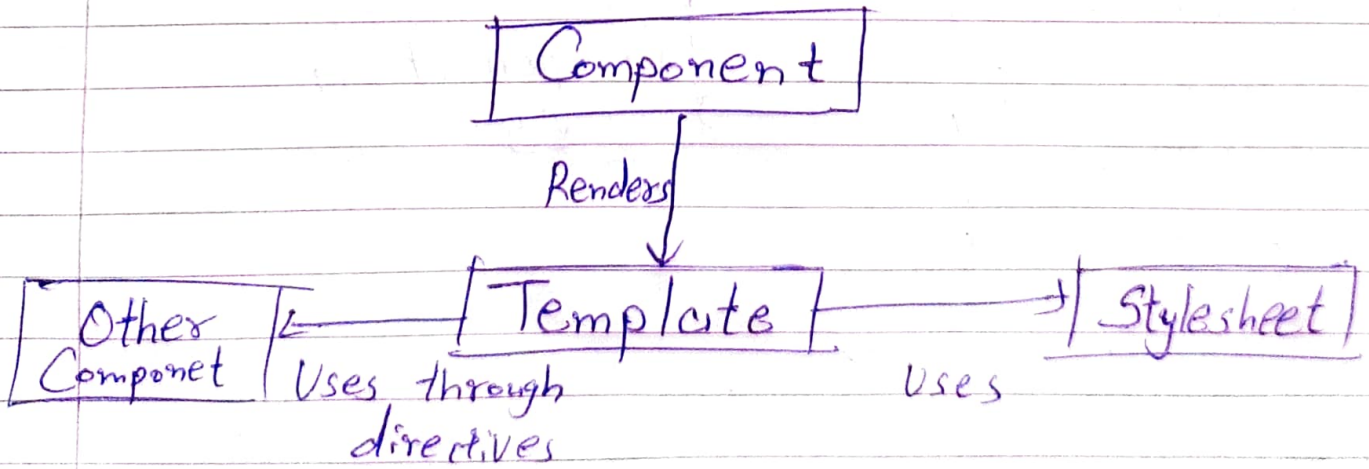


→ Modules :-

- A component service within an Angular Module context is grouped together into a single component or service. An animation-related functionality can be split into multiple modules and Angular already provides a module for that type of functionality, Browser Animation Module.

→ Component :-

- Every Angular application, consisting of one or more Angular Components, is built from Components. Angular Components are simple JavaScript / TypeScript classes with HTML templates & names.



Page No. 9
Date

→ Templates:-

The template is an HTML super set. It includes all the features of HTML, as well as additional functionality that binds component data into the HTML and generates HTML DOM elements dynamically.

→ Metadata:-

Angular uses metadata to inform the framework on how to handle a class. It is used to stipulate how a class should be used.

→ Services:-

A very specific functionality is provided by a service. It does one thing and does it well. Service composition is meant to reduce repetition. Instead of developing a utility function inside a component, separating it into a service will be useful in another component as well.

Page No. 10
Date

☆ TypeScript Introduction:-

→ TypeScript is a statically typed superset of JavaScript that brings enhanced developer productivity and code reliability to modern web and application development.
- Angular is a framework that lets us create interactive web frontends for users.

→ Variable Data Types:-
- One of the basic typescript features that we use regularly in our app is variable declarations.
- To declare variable with datatype annotations with TypeScript,

```
const a: number = 1;
const b: string = "foo";
```

→ Function Data Types:-
- We can add datatype annotations and return values of functions with TypeScript

```
const add = (a: number, b: number): number => a+b;
```

→ Object Data Types:-
- In addition to defining data types for primitive values, we can define datatype annotations for objects.

1)
```
interface Person {
    firstname: string;
    lastname: string;
}
const person: Person = { firstname: "abc",
                         lastname: "xyz" }
```

2)
```
type Person = {
    firstname: string;
    lastname: string;
}
Cost person: Person = { firstname: "abc", lastname: "xyz" }
```

→ Union and Intersection Types:-

- We can combine multiple types into one. One way is to define an union of two types. And the other is to define an intersection of two types.

-
```
type Foo = {
    foo: String;
};
type Bar = {
    bar: ~~string~~ number;
};
type Baz = Foo | Bar;
const x: Baz = { Foo: "foo" };
const y: Baz = { bar: 23 };
const z: Baz = { Foo: "foo", bar: 23 };
```

☆ Angular CLI Commands :-

→ To get the npm version : npm -v

- To get the node version : node -v
- To get the Angular version : ng v
- To install Angular CLI : npm install @angular/cli
- To get help in the terminal : ng help
- To create a new project in angular :
  ng new project-name
- To skip external dependencies while creating a new project :
  ng ~~generate comp~~ new project-name --skip-tests
- To create a new Component in the Angular project :
  ng generate component component-name
- To create a build : ng build
- To run test cases : ng test
- For Angular documentation : ng doc

- To create a directive in Angular:
  ng g d directive-name
- To create a service in Angular:
  ng g s service-name
- To create a class:
  ng g cl class-name
- To create an interface:
  ng g i interface-name
- Serve the application locally:
  ng serve

☆ Component Lifecycle Hooks:-

→ Angular components have a well-defined lifecycle, which allows you to execute code at specific points in the component's existence.

- These lifecycle hooks provide opportunities to perform tasks like initialization, data retrieval and cleanup.

1) ngOnChanges:-

- ngOnChanges triggers following the modification of @Input bound class members.
- Data bound by the @Input() decorator come from an external source.
- When the external source alters that data in a detectable manner, it passes through the @Input property again.

2) ngOnInit:-

- This hook is called after Angular has initialized a component and set its input properties. It is commonly used for initialization tasks, such as fetching data from a service or setting default values.

3) ngDoCheck :-
- This hook is called whenever Angular checks for changes in the component. It allows you to implement custom change detection logic, but it's used less frequently than other hooks.

4) ngAfterContentInit :-
- After Angular projects, external content into a componetnt's view or react view that has a directive.
- Called once after the first ngDoCheck.

5) ngOnDestroy :-
- This hook is called just before Angular destroys the component. It's good place to clean up resources, unsubscribe from observables, or perform any necessary cleanup.

→ LifeCycle Hook log :-

- ngOnChanges
- ngOnInit
- ngDoCheck
- ngAfterContentInit
- ngAfterContentChecked
- ngAfterViewInit
- ngAfterViewChecked
- ngDoCheck
- ngAfterContentChecked
- ngAfterViewChecked
- ngOnDestroy

Page No. 16
Date

✯ Components, Templates

→ Components are building block of Angular application.

- The main job of Angular Component is to generate a section of web page called view.
- Every component will have an associated template and it will be used to generate views.

→ Add a Component :-

- Create a new Component :-
  ng generate component express-entry
- Express-Entry component is created under src/app/express-entry folder.
- Component class, Template & Stylesheet are created.
- AppModule is updated with new component.

⇒ Templates :-

- The integral part of Angular Component is Template.
- It is used to generate the HTML content.
- Templates are plain HTML with additional functionality

→ Attach a template :-

- Template can be attached to Angular Component using @Component decorator's metadata.
- Angular provides two metadata to attach template to components.

→ Template URL :-

- We already know how to use templateURL.
- It expects the relative path of the template file.
- For example, AppComponent set its template as app.component.html

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

→ Template :-

- template enables to place the HTML string inside the component itself.
- If the ~~compo~~ template content is Minimal, then it will be easy to have it Component

Page No. 18
Date

class itself for easy tracking

- Example :-

- c1.component.ts :-

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

@ Component ({
    selector : 'app-c1',
    templateURL : './c1.component.html',
    styleURL : ['./c1.component.css']
})
export class c1 {
    name = 'ABC';
}
```

- c1.component.html :-

```
<p> <h1> Hello, {{name}} </h1>
```

Page No. 19
Date

☆ Data Binding, Event Binding, Expressions, Interpolation.

⇒ Data Binding:-

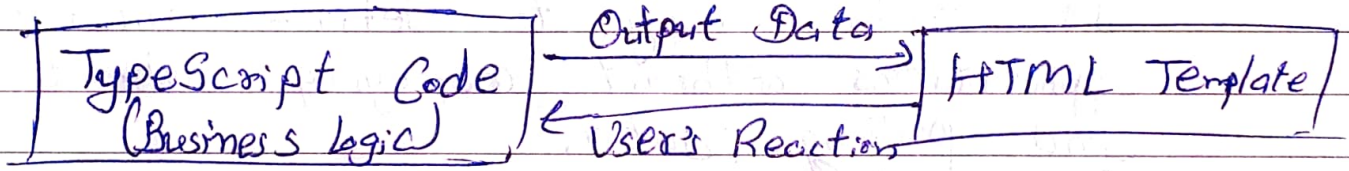
- Data Binding is the core concept of Angular and is used to define the communication between the component and the DOM.
- It is a technique to link your data to your view layer.
- It makes easy to define interactive applications without worrying about pushing and pulling data.
- Data binding can be either one-way or two-way.

→ One-Way data binding:

- It is a simple one way communication where HTML template is changed when we make changes in TypeScript code.
- ~~In one-way data binding, the value~~
- Angular Interpolation / String interpolation, Property Binding and Event Binding are the example of one-way data binding.

→ Two-way databinding :-

- In Two-way databinding, automatic synchronization of data happens between the Model and the View. Here, change is reflected in both components.
- Whenever you make changes in the model, it will be reflected in the view and when you make changes in view, it will be reflected in model.



⇒ Event Binding :-

- In Angular, Event Binding is used to handle the events raised from the DOM like button click, mouse ~~hover~~ move etc.
- When Dom event happens, it calls the specified method in the component.

→ Example :-

- app.component.ts :-

Page No. 27
Date

```
import {Component} from '@angular/core';
@Component ({
    selector : 'app-root',
    templateURL : './app.component.html',
    StyleVRL : ['./app.component.css']
})
export class AppComponent {
    onSave ($event) {
        console.log ("Save button clicked", $event);
    }
}
```

- app.component.html :-

```
<button (click) = "onSubmit ($event)"> Save
                                    </button>
```

⇒ Expression, Interpolation :-

- String Interpolation is a one-way data binding technique which is used to output the data from a TypeScript code to HTML template.
- It uses the template expression in double curly braces to display the data from the component to the view.
- String interpolation adds the value of a property from a component.

→ Example :-

```
export class AppComponent {

    title = 'It's Me';
    N1 : number = 10;
    N2 : number = 20;
    addTwoNumbers () {
        return this.N1 + this.N2
    }
}
```

☆ Routing :-

- Routing in Angular allows you to create single-page applications by navigation between different views within your application without a full page reload.
- It's an essential part of building complex web applications.

→ Configure Routing :-

- Angular CLI provides complete support to setup routing during the application creation process as well as during working on application.

- `ng new routing-app --routing`

```
import {NgModule} from '@angular/core';
import {Routes, RouterModule} from '@angular/router';

const routes : Routes = [];

@NgModule ({
  imports : [RouterModule.forRoot(routes)],
  exports : [RouterModule]
})
export class AppRoutingModule {}
```

Page No. 24
Date

→ Creating routes:-

- Creating a route is simple and easy.
- The basic information to create a route is given below:
  - Target component to be called.
  - The path to access the target component

```
const routes : Routes = [
    { path : 'ct', component : CtComponent }
];
```

→ Accessing Routes:-

- Accessing the route is a two-step process.
- Include router-outlet tag in the root component template.

```
<router-outlet> </router-outlet>

<a routerLink = "/ct"> First Component </a>
```

✡ Modules (Class) :-

→ Module in Angular refers to a place where you can group the components, directives, pipes and services, which are related to the application.

- In case you are developing a website, the header, footer, left, center and the right section become part of a module.
- To define module, we can use the ngModule

```
import {BrowserModule} from '@angular/platform-browser'
import {NgModule} from '@angular/core';
import {AppComponent} from './app.component';

@NgModule ({
    declarations : [
        AppComponent
    ],
    imports : [
        BrowserModule
    ],
    providers : [],
    bootstrap : [AppComponent]
})
export class AppModule {}
```

Page No. 26
Date

→ Import:-
- It is an array of modules required to be used in the app.
- It can also be used by the components in the declaration array.
- Ex: right now in the @NgModule we see the Browser Module imported. In case your application needs forms, you can include the module as follows:-

```
Import { FormsModule } from '@angular/forms';
```

- The import in the @NgModule will be like:-

```
Imports : [
    BrowserModule,
    FormsModule
  ]
```

→ Providers:-
- This will include the services created.

☆ Pipes :-

→ Pipes are a useful feature in angular
- These are the simple way to transform values in an angular template.
- It takes the integers, strings, array and dates as input separated with | to be converted in the format as required and display same as in the browser.

→ A pipe holds in data as input and transforming it into the desired output.

→ Syntax :

```
{{ title | uppercase}}
```

→ Built-In Angular Pipes :-

- Angular has a stock of pipes such as Date pipe, Uppercase pipe, Lowercase pipe and percent pipe.
- They are available for use in any angular template.
- Angular doesn't have the filter pipe or OrderBy pipe.
- Pipes are an excellent way to encapsulate and share a collective display value information.

Page No. 28
Date

- AsyncPipe
- Datapipe
- Jsonpipe
- Lowercasepipe
- Uppercasepipe
- CurrencyPipe
- Decimal pipe
- Percentpipe
- Slicepipe
- Titlecase pipe

→ Parameterizing a pipe in Angular:-

- We can move on a parameter to the pipe; we can use the HTML code to pass the parameter.

{{ birthday | date : "dd/mm/yyyy" }}

→ Chaining pipes:-

- We can chain pipes together and creates useful combinations and also can use the lowercase and uppercase pipe.

{{ birthday | date | uppercase }}

- By default, pipes of angular are pure.
- Every pipe we have seen are pure & builtin pipes.
- We can make the pipe impure by setting the pure flag into false.

✯ Directives :-

→ The Angular directives are used to manipulate the DOM.

- By using Angular directives, you can change the appearance, behaviour or a layout of a DOM element.
- Angular directives can be classified in 3 categories based on how they behave:
  - Component Directives
  - Structural Directives
  - Attribute Directives.

→ Component Directives :-

- Component directives are used in main class.
- They contain the detail of how the

- *ngSwitch Directive: The ngSwitch allows us to Add/Remove DOM element.

- *ngFor Directive: The ngFor directive is used to repeat a portion of HTML template once per each item from an iterable list.

→ Attribute Directives:-

- Attribute directives are used to change the look and behaviour of the DOM elements.

- ngClass Directive: The ngClass Directive is used to add or remove CSS classes to an HTML element.

- ngStyle Directive: The ngStyle Directive facilitate you to modify the style of an HTML element using the expression.

⇒ Example:-

→ ngIf:-

```
<div *ngIf="showElement">This is if directive.</div>
```

→ ngFor :

```
<ul>
  <li *ngFor="let i of array"> {{i}} </li>
</ul>
```

→ ngSwitch :

```
<div [ngSwitch] = "condition">
    <p *ngSwitchCase = "'case1'"> Case 1 Content </p>
    <p *ngSwitchCase = "'case2'"> Case 2 Content </p>
    <p *ngSwitchDefault> Default Content </p>
</div>
```

☆ Forms :-

→ Forms are used to handle user input data.

- Angular supports two types of forms. They are Template driven forms and Reactive Forms.

⇒ Template driven forms :-

- Template driven form is created using directives in the template. It is mainly used for creating a simple form app.

→ Create the form template :-

```
<form #myform = "ngform" (ngSubmit) = "onSubmit
                                (myForm.value)">
  <div class = "form-group" >
    Name : <input type = "text" id = "name"
              name = "name" ngModel>
  </div>
  <div class = "form-group" >
    Email : <input type = "text" id = "email"
              name = "email" ngModel >
  </div>
  <button type = "submit" > Submit </button>
</form>
```

→ Component Class :-

```
import {Component} from '@angular/core'
export class MyFormComponent {
    onSubmit (formData : any) {
        console.log (formData);
    }
}
```

⇒ Reactive Forms :-

- Reactive Forms is created inside component class so it is also referred as model driven forms.
- Every form control will have an object in the component and this provides greater control and flexibility in the form programming.
- Reactive Form is based on Structured data Model.

→ Import ReactiveModule :

```
import {ReactiveFormsModule} from '@angular/forms'
@ngModule ({
    imports : [ReactiveFormsModule]
})
```

→ Create the Form in the Component:

```
import {Component} from '@angular/core';
import {FormBuilder, FormGroup, Validators} From
                    '@angular/Forms';
@ Component ({
      selector: 'app-my-form',
      templateUrl: './my-form.component.html'
})
export class MyFormComponent {
      myForm: FormGroup;

  constructor (private fb: FormBuilder)
  {
      this.myForm = this.fb.group ({
        name: ['', Validators.required],
        email: ['', [Validators, require.Validation
                    ~~Validat~~.email]]
      });
  }

onSubmit () {
      console.log (this.myForm.value);
}
}
```

→ Bind the form to the template

```
<form [formGroup] = "myForm" (ngSubmit)
                    = "onSubmit()">
  Name: <input type = "text" id = "name"
          formControlName = "name" >
  Email : <input type = "text" id = "email"
          formControlName = "email" >
<button type = "submit" > Submit </button>
</form>
```

★ Services :-

- Services are a crucial part of the architecture and play a significant role in managing data, providing shared functionality and promoting code reusability.
- Services are often used for tasks like fetching data from an API, sharing data between components, or encapsulating common functionality.

→ Create a Service :-

```
ng generat service my-service
```

→ Define a Service

```
import {Injectable} from '@angular/core'
@Injectable ({
    providedIn : 'root'
})
export class MyService{
    constructor(){}
    getData (): string{
        return 'Data from the service';
    }
}
```

→ Inject the service :-

```
import {Component} from '@angular/core';
import {MyService} from '@./my-services-servi
@component ({
    selector: 'app-my-component',
    template: '<p> {{data}} </p>
})
export class MyComponent {
    data: string,
    constructor (private myService: MyService)
    {
        this.data = this.MyService.getData()
    }
}
```

→ Asynchronous services :-

```
import {Injectable} from '@angular/core'
import {HttpClient} from '@angular/common/http'
import {Observable} from 'rxjs',
@Injectable ({
    providedIn : 'root'
})
export class DataServices {
    private apiURL = 'https://localhost: 5800/getbook',
    constructor (private http: HttpClient) {}
    fetchData (): Observable <any> {
        return this.http.get (this.apiURL())
    }
}
```

☆ HttpClient :-

→ The HttpClient module is used to make HTTP requests to interact with external APIs, retrieve data, and perform CRUD operations on a server.

- It is an essential part of building web applications that communicate with back-end services.

→ Import the HttpClient Module :-

```
import {HttpClientModule} from '@angular
@NgModel ({                                /common/http';
    imports: [HttpClientModule]
})
```

→ Create a service :-

```
ng generate service data
```

→ Define the services :-

```
import {Injectable} from '@angular/core';
import {HttpClient} from '@angular/common/
import {Observable} from 'rxjs'                http';
@Injectable ({
    providedIn: 'root'
})
```

Page No. 39
Date

```
export class DataService {
  private apiUrl = "https://localhost:8800/book";
  constructor (private http: HttpClient) {}
    fetchData (): observable<any> {
        return this.http.get (this.apiURL);
    }
    postData (@data: any): observable<any> {
        return this.http.post (this.apiURL, data);
    }
}
```

→ Inject the service;

→ Use the service:

```
import {Component, OnInit} from '@angular/core';
import {DataService} from './dat.service';

@Component ({
    selector: 'app-my-component',
    template: `
    <button (click) = "FetchData ()" > Fetch </button>
    <div> {{ response | json }} </div>
    `
})
export class MyComponent {
    response: any;
```

```
constructor (private dataservice : DataService){}

fetchData (){
  this.dataservice.FetchData().subscribe(data=>
    {
      this.response = data;
    });
}
```

- Using 'HttpClient' in Angular allows you to perform a wide range of HTTP operations, From Simple GET requests to more complex operations like POST, PUT, DELETE and handling query parameters and headers