Angular

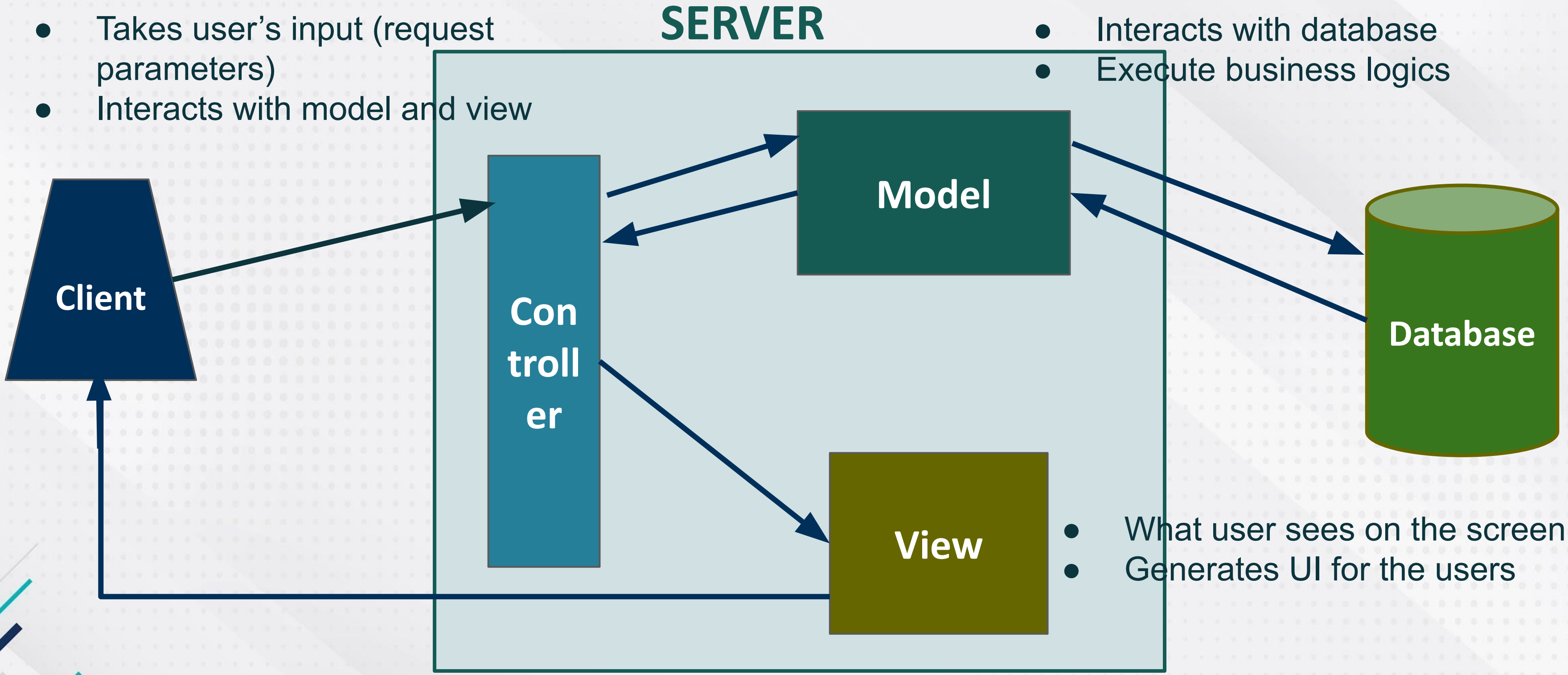
What is MVC?

- MVC stands for **M**odel **V**iew **C**ontroller.
- It is an architectural pattern.
- MVC divides a software application into three parts: Model, View and Controller.
- Each of these three components has some specific responsibility.

MVC Architecture in Web Application

- Takes user's input (request parameters)
- Interacts with model and view

- Interacts with database
- Execute business logics



- What user sees on the screen
- Generates UI for the users

Advantages of MVC

- Codes are easy to maintain and they can be extended easily.
- The components of MVC can be developed simultaneously.
- It reduces complexity by dividing an application into three units. Model, view, and controller.
- This architecture helps to test components independently as all classes and objects are independent of each other

Angular Introduction – What and Why?

What?

- ➢ Angular is a development platform, built on TypeScript.
- ➢ Framework to build client side applications.
- ➢ Great for building Single page applications.

Why?

- ➢ Modular Approach
- ➢ Reusable code
- ➢ Development quicker and easier
- ➢ Unit testable
- ➢ Google + Microsoft

Create and run Angular Project

- Install NODE JS on development machine.
- Install Angular CLI globally
- Create an Angular project
- Compile and run Angular project.

Install Nodejs Into System

- ➢ To use angular first we have to install node js into our system.
- ➢ Visit **nodejs.org** to download and install nodejs into our system.
- ➢ To check whether the node is installed into our system, run command **node -v** from the command prompt.
- ➢ As soon as we install nodejs, npm automatically installed into our system.
- ➢ We can check the version of npm using command **npm -v** from command prompt.
- ➢ NPM stands for node package manager.

Install Angular CLI

- Angular CLI is a command line interface for Angular.
- It allows to generate building blocks of Angular application
- It makes development quicker and easier.
- Install Angular CLI using command **npm install -g @angular/cli** from command prompt.
- Here g stands for globally. We have to install angular globally.
- To check Angular CLI is installed or not type the command **ng v** on command prompt.
- To check the version of Angular CLI use the command **ng version or ng –version** on command prompt.

Create First Angular Project

- ➢ To create Angular project open the command prompt.
- ➢ Type command **ng new <project name>** from command prompt.
- ➢ ***Use command ng new <project-name> --standalone=false in the new version otherwise it will not create app.module.ts file.***
- ➢ It will create a new project into the current working directory.
- ➢ Now use command **cd <project-name>** to navigate to the project folder.
- ➢ Use command **ng serve** to run the application.
- ➢ Now goto the browser and use url **localhost:4200**

Questions & Answers(MCQ)

- **What does MVC stand for in Angular?**
  a. Model-View-Connection
  [✓ b. Model-View-Controller]
  c. Module-View-Controller
  d. Module-View-Component
- **Which of the following best describes the "Model" in MVC?**
  a. It handles the user interface and input.
  [✓ b. It is responsible for managing data and business logic.]
  c. It defines the routing in the application.
  d. It is used to decorate components.

Questions & Answers(MCQ)

- **What command is used to install Angular CLI globally?**
  a. npm install -g angular
  b. ng install angular
  c. npm angular-cli
  d. npm install -g @angular/cli ✓
- **Which of the following commands is used to create a new Angular project?**
  a. ng create <project-name>
  b. ng new <project-name> ✓
  c. ng start <project-name>
  d. angular new <project-name>

Questions & Answers(MCQ)

➢ **Which command is used to serve an Angular application on a local development server?**

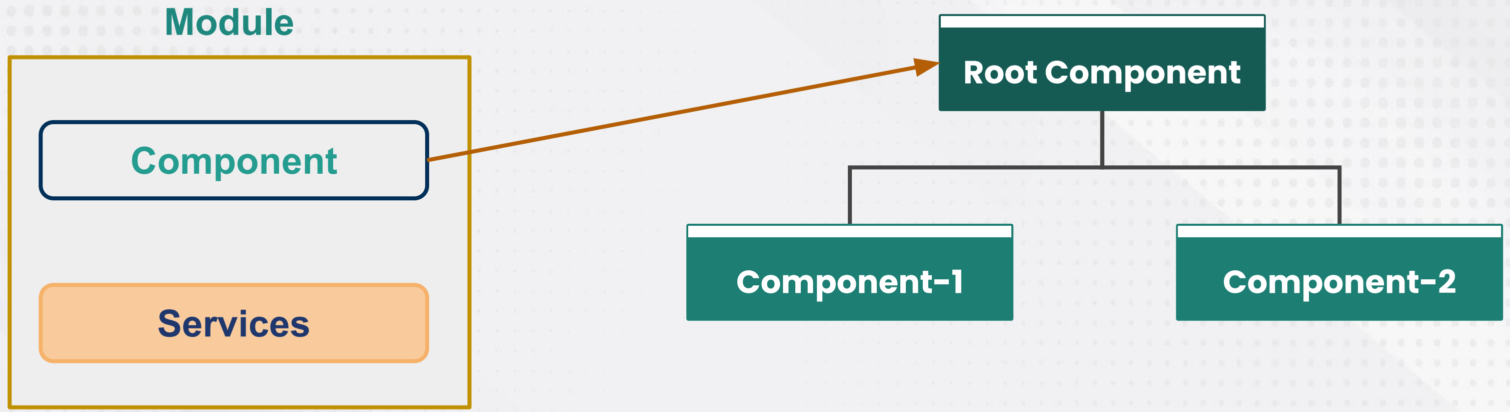
a. ng start
b. [✓ ng serve]
c. ng run
d. npm start

➢ **How do you check the installed version of Angular CLI?**

a. ng version
b. ng --version
c. [✓ Both a and b]
d. angular --version

Architecture of Angular Project

- ➢ Angular app - Contains one or more modules
- ➢ Module - One or more components and services
- ➢ Components
- ➢ Services - Business logic
- ➢ Module interact and ultimately render the view in the browser



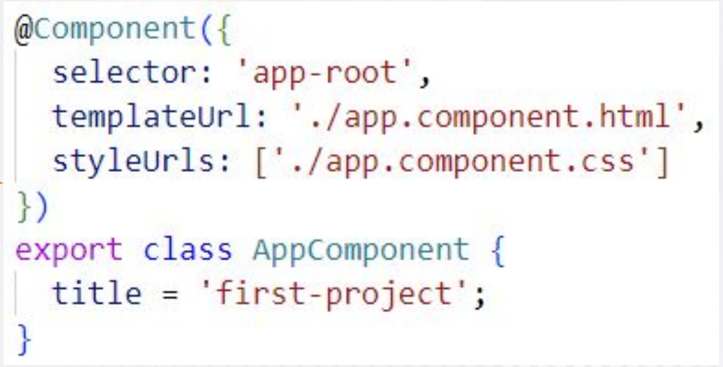
Important Files of Project Folder

- **Package.json:** This file contains the dependencies and devdependencies (Libraries) that are required for angular application.
- **node_modules folder:** All the packages are installed inside the node_modules folder. We do not push this folder into the git repository. To install all dependencies use ***npm install*** command.
- **Src folder:** It contains:
  - **Main.ts:** It is the entry point of angular application
  - **App folder:** It contains **app.module.ts** file which is the root module of the application and **app.component.ts** which is the root component of the application.
- When we runs **ng serve** command it follows the path.

Main.ts → ***app.module.ts*** → ***app.component.ts***

Component

- Components are the building blocks that compose an application.
- A component includes a TypeScript class with a **@Component()** decorator, an **HTML template**, and **styles**.
- The @Component() decorator specifies the following information:
    - **A selector**
    - **An HTML template**
    - **An optional set of CSS styles**



```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'first-project';
}
```

Component

- All data in the **index.html** comes from **<app-root></app-root>** which is placed inside the app folder
- The selecter will be used as a tag to render the html of a component.
- To generate new component use command **ng generate component new-cmp** or **ng g c new-cmp**
- A component contains the following files:
  - app.component.css
  - app.component.html
  - app.component.spec.ts
  - app.component.ts

Component

- ➢ A component is made up of three parts: Template, Class and Metadata
- ➢ **Template:** It is the view that is made up with HTML.
- ➢ **Class:** Code that supports the view. It is typescript code. Contains data and methods.
- ➢ **Metadata:** This is the information that angular needs to decide that a particular class is a component or just a regular class. It is defined using decorator(typescript feature).

Template	+	**Class**	+	**Metadata**
View		Code		Information
HTML		Typescript		Decorator
		Data & Methods		

Manually Generate Component

- ➢ Create a new folder named myContainer inside the app folder.
- ➢ Inside myContainer folder create myContainer.component.ts file and write the following code.

```
import { Component } from "@angular/core";
@Component({
    selector:'app-myContainer',
    templateUrl:'./myContainer.component.html',
    styleUrls: ['./myContainer.component.css']
})
export class MyContainerComponent{
}
```

Manually Generate Component

- ➢ Create myContainer.component.html file and write the following code:

```
<div class="container">
    <app-header></app-header>
</div>
```

- ➢ Create myContainer.component.css file and write the following code:

```
.container{
    background: #f3f3f3;
    width: 800px;
    margin: auto;
    padding: 5px;
}
```

Manually Generate Component

- ➢ Register this MyContainerComponent inside app.module.ts file declarations array.
- ➢ Import the component inside the app.module.ts file.

```
import { MyContainerComponent } from './myContainer/myContainer.component' ;
```

- ➢ Now copy the selector name from myContainer.component.ts file and create a custom tag inside the app.component.html file.

```
<app-myContainer></app-myContainer>
```

- ➢ We can also use template in place of templateUrl. In case of template we have to put the html code inside the ts file.

Generate Component Using CLI

- To generate component from angular CLI use command

  ng generate component <component-name>

  or

  ng g c <component-name>
- All the supporting files will be generated automatically.
- Here the component will be registered automatically by the angular CLI.

Questions & Answers(MCQ)

- **What is the purpose of the angular.json file in an Angular project?**
  a. [✓ To configure the Angular CLI build and development tools]
  b. To define the application's components
  c. To manage Node.js dependencies
  d. To configure TypeScript settings

- **Which command adds a new component to an Angular project?**
  a. ng create component <name>
  b. ng add component <name>
  c. [✓ ng generate component <name>]
  d. angular new component <name>

Questions & Answers(MCQ)

➢ **Which of the following files is used to specify the dependencies of an Angular project?**

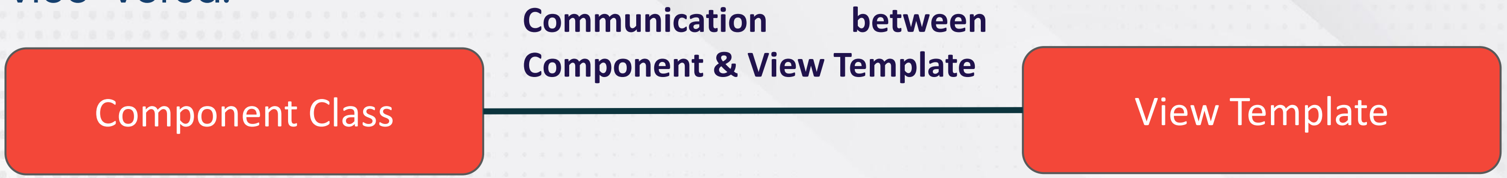
a. angular.json
[✓ b. package.json]
c. tsconfig.json
d. README.md

➢ **By default, on which port does the Angular development server run?**

a. 3000
[✓ b. 4200]
c. 8080
d. 5000

What is Data Binding?

➢ Data binding in Angular allows us to communicate between component class and its corresponding view template and vice-versa.



UI Logic

```
export class DemoComponent {
  title = 'demo component';
  message = 'Message from demo';
  hide = false;
  displayCon() {
    this.hide = true;
  }}
```

HTML

```
<div class="demo">
<div>{{title}}</div>
<div>{{message}}</div>
<button
(click)="displayCon()">Hide</button>
<div [hidden]="hide">Hide this</div>
</div>
```

What is Data Binding?

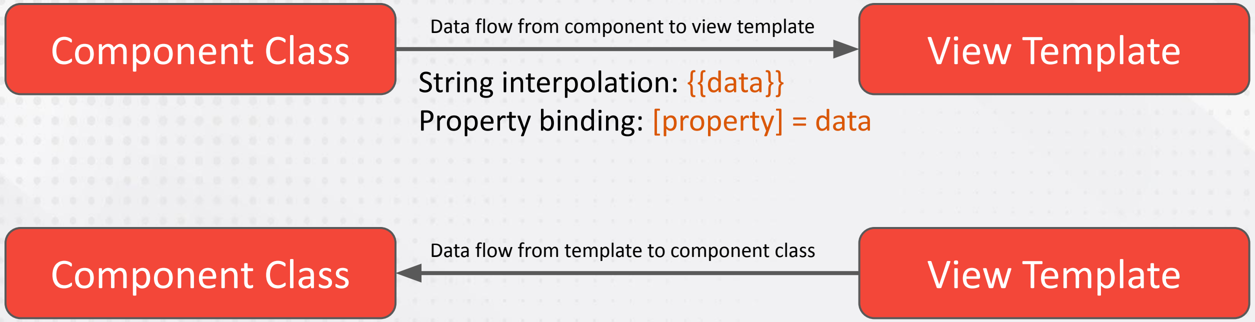
- In the <div> tag the **hidden** property in inside **[]**. It will assign the **value** of the **hide** variable to **hidden** property.
- If we do not wrap the **hidden** property inside the **[]** then it will assign the **“hide”** string to the **hidden**.
- We can also pass the data from the view template to the component class.
- In the previous example we can pass the arguments in `displayCon()` method from the view template.

Types of Data Binding

- ➢ There are two types of data binding in Angular
  - **a. One way data binding**
  - **b. Two way data binding.**
- ➢ One-way data binding is when, data can be access from component class into its corresponding view and vice versa.
- ➢ Two-way data binding binds data from component class to view template and view template to component class. It is a combination of property binding and event binding.

One-way Data Binding

- ➢ One-way data binding is when, data can be access from component class into its corresponding view and vice versa.
- ➢ We can divide the one-way data binding into two parts:
  a. Component to view
  b. View to component


Component Class
Data flow from component to view template
View Template
Component Class
Data flow from template to component class
View Template

String interpolation: {{data}}
Property binding: [property] = data

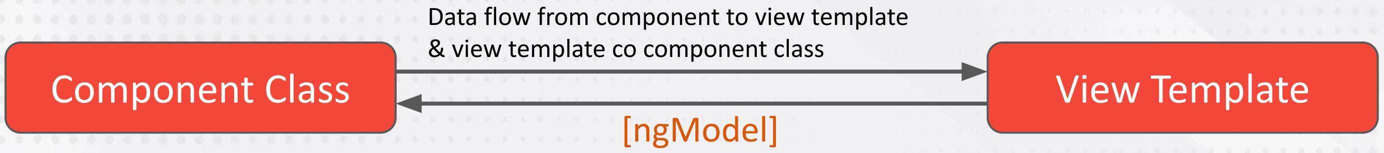
Data flows from component to its view template

Event binding: (data) = “expression”

Data flows from view template to component class

Two-way Data Binding

➢ Two-way data binding binds data from component class to view template and view template to component class. It is a combination of property binding and event binding.


Data flow from component to view template
& view template co component class
Component Class
View Template
[ngModel]

Data flows from component to its view template & at the same time, from view template to component class

String Interpolation

- Interpolation refers to embedding expressions into marked up text. By default, interpolation uses the double curly braces **{{ and }}** as delimiters.
- Consider an Angular component that contains a currentCustomer variable

```
currentCustomer = 'Jhon';
```

- Use interpolation to display the value of this variable in the corresponding component template.

```
<h3>Current customer: {{ currentCustomer }}</h3>
```

- Angular replaces currentCustomer with the string value of the corresponding component property. In this case, the value is **Jhon**
- In interpolation you can also use inbuilt javascript methods like currentCustomer.toUpperCase().

String Interpolation

- We can also call the method in string interpolation syntax.

```
export class DemoComponent {
  product = {
    pImage: './assets/images/samsung.jpg',
    brand: 'Samsung',
    price: 20000,
    model: 'Galaxy S-20',
    discount: 15.5,
  };
  calculateDiscount() {
    return (
      this.product.price -
(this.product.price *
this.product.discount) / 100
    );
  }
}
```

```
<div>
    <h2>Product Information</h2>
    <img [src]="product.pImage"
alt="mobile image">
    <p>Brand: {{product.brand}}</p>
    <p>Price: {{'
₹'+product.price}}</p>
    <p>Model: {{product.model}}</p>
    <p>Discount: {{product.discount +
'%'}}</p>
        <p>Discounted Price: {{'
₹'+calculateDiscount()}}</p>
</div>
```

String Interpolation

➢ We can also use ternary operator inside the string interpolation syntax.

```
export class DemoComponent {
  product = {
    pImage: './assets/images/samsung.jpg' ,
    brand: 'Samsung',
    price: 20000,
    model: 'Galaxy S-20',
    discount: 15.5,
    inStock: 10,
  };
  calculateDiscount () {
    return (
      this.product.price - (this.product.price
* this.product.discount) / 100
    );
  }
}
```

```
<div>
    <h2>Product Information</h2>
    <img [src]="product.pImage"
alt="mobile image">
    <p>Brand: {{product.brand}}</p>
    <p>Price: {{'₹'+product.price}}</p>
    <p>Model: {{product.model}}</p>
    <p>Discount: {{product.discount +
'%'}}</p>
    <p>{{product.inStock>0?'Only '+
product.inStock+' items left' :'Out of
stock'}}</p>
    <p>Discounted Price: {{'
₹'+calculateDiscount()}}</p>
</div>
```

Property Binding

- Property binding is used to bind dynamic property into the html attribute.
- The difference between property binding and interpolation is that:
  - **Interpolation** is used to just display a piece of data in HTML, such as displaying a title or a name.
  - **Property binding** lets us bind a property of a DOM object, for example the ***hidden*** property to some data value. This can let us show or hide a DOM element or manipulate the DOM in some other way.

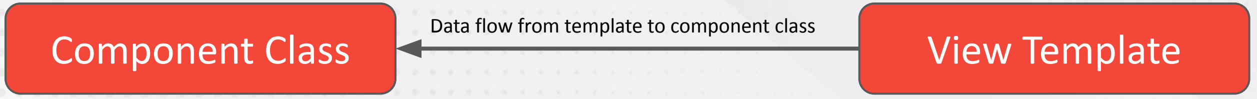
Property Binding Example

```
export class DemoComponent {
  product = {
    pImage:
'./assets/images/samsung.jpg',
    brand: 'Samsung',
    price: 20000,
    model: 'Galaxy S-20',
    discount: 15.5,
    inStock: 10,
  };
  calculateDiscount() {
    return (
      this.product.price -
(this.product.price *
this.product.discount) / 100
    );
  }
}
```

```
<div>
    <h2>Product Information</h2>
    <img [src]="product.pImage"
alt="mobile image">
    <p>Brand: {{product.brand}}</p>
    <p>Price: {{'₹'+product.price}}</p>
    <p>Model: {{product.model}}</p>
    <p>Discount: {{product.discount +
'%'}}</p>
    <p>{{product.inStock>0?'Only '+
product.inStock+' items left' :'Out of
stock'}}</p>
    <p>Discounted Price: {{'
₹'+calculateDiscount()}}</p>
    <button
[disabled]="product.inStock<=0">Buy
Now</button>
</div>
```

Event Binding

- We use event binding to bind the data from view template to component class.



Event binding: (data) = “expression”

Data flows from view template to component class

Event Binding Example

```
export class DemoComponent {
  name = '';
  handelChange(event: any) {
    this.name =
event.target.value;
  }
}
```

```
<div>
    <h2>Event Binding Demo</h2>
    <input type="text"
(input)="handelChange($event)">
    <p>The value in the textbox is:
{{name}}</p>
</div>
```

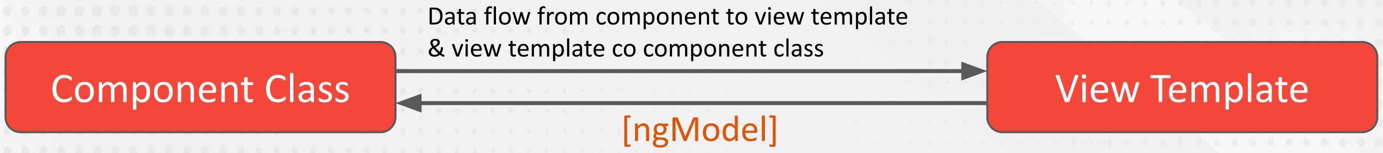
Event Binding Example

```
export class DemoComponent {
  quantity = 0;
  product = {
    pImage:
'./assets/images/samsung.jpg' ,
    brand: 'Samsung',
    price: 20000,
    model: 'Galaxy S-20',
    discount: 15.5,
    inStock: 10,
  };
  decrementCount() {
    if (this.quantity > 0) {
      this.quantity--;
    }  }
  incrementCount() {
    if (this.quantity <
this.product.inStock) {
      this.quantity++;
    }  }
}
```

```
<div>
    <h2>Product Information</h2>
    <img [src]="product.pImage"
alt="mobile image">
    <p>Brand: {{product.brand}}</p>
    <p>Price: {{'₹'+product.price}}</p>
    <p>Model: {{product.model}}</p>
    <p>Discount: {{product.discount +
'%'}}</p>
    <p>{{product.inStock>0?'Only '+
product.inStock+' items left' :'Out of
stock'}}</p>
    <button
(click)="decrementCount()">-</button>
    {{quantity}} <button
(click)="incrementCount()">+</button>
</div>
```

Two-way Data Binding

- In two-way data binding data flows from component class to its view template & at the same time, from view template to component class
- Basically two-way data binding is a combination of property binding and event binding.


Data flow from component to view template
& view template co component class
Component Class
View Template
[ngModel]

Two-way Data Binding Example

```
export class SearchComponent {
  searchData = 'default';
  handelInput(event: any) {
    this.searchData =
event.target.value;
   }
}
```

```
<div class="ekart--search--product" >
    <input type="text"
class="ekart-search-product-input"
[value]="searchData"
(input)="handelInput($event)" >
    <button class="btn
btn-search">Search</button>
</div>
<div class="ekart-search-result-for" >
    <p><strong>Search result for:
</strong>{{searchData}} </p>
</div>
```

```
.ekart--search--product {
  margin: 20px auto;
  text-align: center;
}
.ekart-search-product-input {
  width: 40%;
  height: 5.5vh;
  padding: 1px 10px;
  outline: none;
  border-radius: 2px;
  background-color: #efefef;
  border: #28282b 1px solid;
}
.btn {
  padding: 5px 10px;
  border: none;
}
.btn-search {
  padding: 12px 30px;
  background-color: #28282b;
  margin-left: -5px;
  color: #fff;
}
.ekart-search-result-for {
  margin: 20px auto;
  padding: 10px 140px;
}
```

Two-way Data Binding

- ➢ Instead of using property binding and event binding combination we can use built in directive ***ngModel***.

```
<input type="text" class="ekart-search-product-input"
[(ngModel)]="searchData">
```

- ➢ Inside ***app.module.ts*** file import ***FormsModule***

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

- ➢ Register `FormsModule` inside `imports` array.

```
imports: [BrowserModule, AppRoutingModule, FormsModule]
```

Questions & Answers(MCQ)

- **What is Data Binding in Angular?**
  a. [✓ A process of connecting HTML UI with TypeScript code]
  b. A method for debugging Angular apps
  c. A feature for routing in Angular
  d. A tool for styling Angular components

- **Which of the following is NOT a type of data binding in Angular?**
  a. One-way Binding
  b. Two-way Binding
  c. Event Binding
  d. [✓ Three-way Binding]

Questions & Answers(MCQ)

- **What symbol is used for property binding in Angular?**
  a. [✓ []]
  b. ()
  c. [()]
  d. {}

- **Which syntax is used for event binding in Angular?**
  a. []
  b. [✓ ()]
  c. [()]
  d. {{}}

Questions & Answers(MCQ)

- **What does [()] represent in Angular?**
  a. Property Binding
  b. Event Binding
  [✓ c. Two-way Binding]
  d. Interpolation

- **How do we use interpolation in Angular?**
  [✓ a. {{ expression }}]
  b. [[expression]]
  c. (expression)
  d. < expression >

Angular Directives

What are Directives?

➢ A directive is an instruction to the DOM. We use directives to manipulate the DOM.

Manipulate DOM

Change Behaviour

Add/Remove DOM Elements

Types of Directives

- Directive in angular can be divided into three categories:
  - **Component Directive:** A component directive is the angular component. It is a directive with a template.
  - **Attribute Directive:** Attribute directive is used to change the appearance or behavior of a DOM element. **Ex-** ***ngClass, ngStyle***
  - **Structural Directive**: Structural directive is used to add or remove a DOM element on the Webpage. We always use * before any structural directive. **Ex-** ***ngIf, ngFor, ngSwitch***

ngFor Directive

- ➢ Angular ngFor directive iterates over a collection of data like array, list etc and creates an HTML element for each of the items for an HTML template.
- ➢ It is used to repeat a portion of HTML template once per each item for an iterable list.
- ➢ It is a structural directive. It manipulates the DOM by adding/removing elements from the DOM.

```
<div *ngFor="let item of [10,20,30,40,50]">
    <p>The item value is {{item}}</p>
</div>
```

ngFor Directive Example

```
export class DemoComponent {
  employees = [{
      name: 'Emp-1',
      salary: 85000,
      designation: 'Developer',
    },
    {
      name: 'Emp-2',
      salary: 64000,
      designation: 'Tester',
    },
    {
      name: 'Emp-3',
      salary: 72000,
      designation: 'Manager',
    },  ];
}
```

```
<table id="employees">
    <tr>
        <th>Employee Name</th>
        <th>Designation</th>
        <th>Salary</th>
    </tr>
    <tr *ngFor="let emp of
employees">
        <td>{{emp.name}}</td>
        <td>{{emp.designation}}</td>
        <td>{{emp.salary}}</td>
    </tr>
</table>
```

ngIf Directive

➢ Angular ngIf directive is a structural directive that is used to completely add or remove a DOM element from the webpage based on a given condition.

```
export class DemoComponent {
  name = '';
}
```

```
<div>
    <input type="text"
[(ngModel)]="name">
    <p *ngIf="name!=''"><b>You
Entered: </b> {{name}}</p>
</div>
```

Using Bootstrap in Angular

- ➢ Install the bootstrap into your project using command **npm i --save bootstrap**
- ➢ Open the angular.json file and register the bootstrap in styles array.

```
"styles": [
          "node_modules/bootstrap/dist/css/bootstrap.min.css",
          "src/styles.css"
        ],
```

- ➢ Now you can use bootstrap classes inside your project.

Services in Angular

Need of Service in Angular

- Suppose we have a button ***Subscribe Now*** in five components. Then in the normal case we have to write the subscription logic in all five components.
- **In this approach there are following disadvantages:**
  - Same code is repeated in five different components. We are violating the DRY principle.
  - Component class should only be responsible for representing UI to the user.
  - Presentation logic must be separated from business logic to make components more maintainable.
- So to overcome these problems we can write the subscription logic in a separate file.

What is Service in Angular?

- A service in Angular is a reusable class that can be used in multiple components across our Angular application.
- We can use the services to communicate between two non-related components in Angular.

Advantages of Angular Service

- Services allows us to reuse a piece of code in multiple components, wherever it is required. In this way we avoid repeating a piece of code.
- It allows us to separate business logic from UI logic. We write UI logic in component class and business logic in a service class.
- We can unit test the business logic written in a service class separately without creating a component. Testing and debugging is easier with service.

Creating & Using Service

- ➢ To create a service in your project, create a folder ***services*** inside the app folder.
- ➢ Now create a ***ts*** file named ***subscribe.service.ts*** inside this ***services*** folder.
- ➢ Create a class named ***SubscribeService*** inside this ts file and write the code that you want to reuse in different components.

```
export class SubscribeService {
  onSubscribe() {
    // Write the code to fetch the data from db
    alert('Subscribe service method called');
  }
}
```

Creating & Using Service

➢ To Now goto the component ts file and create the instance of ***SubscribeService*** class.

```
export class OneComponent {
  subscribeClick() {
    let subscribe = new SubscribeService();
    subscribe.onSubscribe();
  }
}
```

➢ Now on button click event you can call this method.

```
<button type="button" class="btn btn-success"
(click)="subscribeClick()">Subscribe Now</button>
```

➢ You can do the same task for all components where you want to use this service.

Creating & Using Service

- There are some problems with this approach.
- In this approach the component class and service class are tightly coupled.
- If we change something in service class then we have to make the necessary changes in the component class also.
- Suppose we define a parameterized constructor in service class then we have create the instance of service class with this new parameterised constructor.

```
constructor(name: string) {}
```

- To solve this problem we will use the dependency injection.

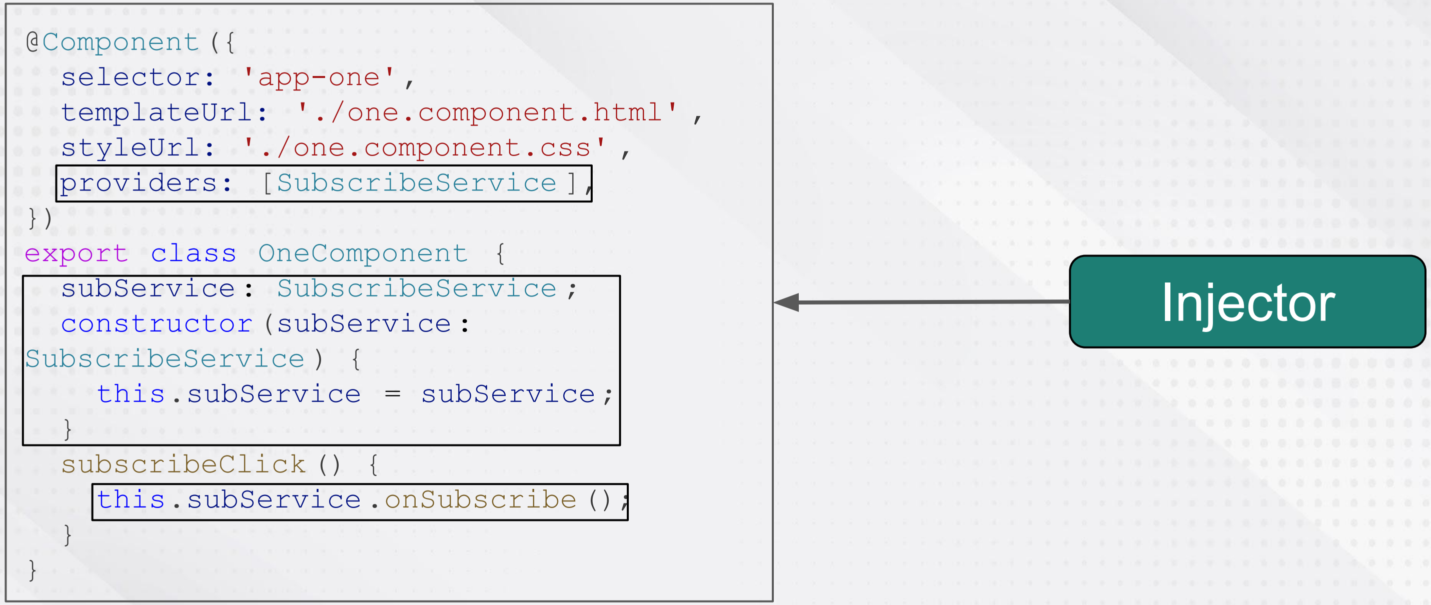
Dependency Injection

What is Dependency Injection?

- A **dependency** is a relationship between two software components where one component relies on the other component to work properly.
- **Dependency injection** (DI) is a technique (design pattern) using which a class receives its dependencies from an external source rather than creating itself.

How to Inject Dependency?

➢ In dependency injection, angular create the instance of the dependency and inject it. To do this angular has a tool called ***injector***.



```
@Component({
  selector: 'app-one',
  templateUrl: './one.component.html' ,
  styleUrl: './one.component.css' ,
  providers: [SubscribeService],
})
export class OneComponent {
  subService: SubscribeService;
  constructor(subService:
SubscribeService) {
    this.subService = subService;
  }
  subscribeClick() {
    this.subService.onSubscribe();
  }
}
```

Disadvantages of not Using Dependency Injection

- ➢ Without dependency injection, a class is tightly coupled with its dependency. This makes a class non-flexible. Any change in dependency forces us to change the class implementation.
- ➢ It makes testing of class difficult. Because if the dependency changes, the class has to change. And when the class changes, the unit test mock code also has to change.

Advantages of Dependency Injection

- ➢ Dependency injection keeps the code flexible, testable, and mutable.
- ➢ Classes can inherit external logic without knowing how to create it.
- ➢ Dependency injection benefits components, directives and pipes.