**Angular TUtorial**

Contents:

1. **INTRODUCTION To Angular**
2. **Angular SETUP Environment**
3. **Angular Application ARCHITECTURE & Application FLOW**
4. **Angular COMPONENTS**

* **Inline Document**
* **Code Behind Document**

1. **Angular DATA BINDING**

* **Inline Document**
* **Two-way data binding**
* **MVC**
* **MVVM**

1. **CONSUMING DATA from API in Angular**
2. **Angular DIRECTIVES**

* **Content Projection**
* **NgIf**
* **NgSwitch**
* **NgFor**
* **NgClass**
* **NgStyle**

1. **Angular EVENT Binding**

* **Mouse Events**
* **Keyboard Events**
* **Element State Events**
* **Custom Events**

1. **Angular Component Life Cycle HOOKS**
2. **Angular MATERIAL**

* **Component**
* **CDK**

1. **Angular PIPES**

* **Custom Pipes**

1. **Angular SERVICE**

* **HttpClientService**
* **Rxjs**

1. **Angular FORMS**

* **Template Driven Forms**
* **Validations**
* **Form state validation**
* **Input state validation**
* **Errors Object**
* **Model Driven/Reactive Forms**
* **Reactive Form Validation**
* **FormBuilder**

1. **Angular ROUTING**

* **Route Parameter**
* **Child Route**
* **QueryParams**
* **Route Guard**

1. **TESTING of Angular Application**
2. **DEPLOYMENT of Angular Application**

**1. Introduction to Angular**

**- Angular is a developers platform.  
- A developers platform provides end-to-end solution for developer.  
- It provides libraries required for  
      a) Building  
       b) Debugging  
      c) Testing  
       d) Deploying  
- Angular developed by Google and maintained by a large community of developers and organizations.  
  
Versions  
- Google introduced Angular JS in early 2010.  
- Google re-written Angular in 2014.  
- AngularJS  versions are known as 1x, the latest version is 1.8.  
- Angular is an alternative for Angular JS.  
- Angular versions started with 2 and latest version is 15.**

**2. Setup Environment for Angular   
1. Download and Install Node JS [NPM]  
2. Download and Install TypeScript  
3. Download and Install Visual Studio Code Editor  
4. Check the Angular version if it is already installed on your PC  
  
        *C:\>ng version*  
5. Remove the existing version *C:\> npm uninstall @angular/cli -g*         
6. Remove the cached files [Temporary Files] *C:\> npm  cache verify*  
7. Install new version of Angular *C:\> npm  install  -g   @angular/cli*  
Note: CLI is a command line tool provided by angular to create and manage projects.**

***C:\> ng  version***

**Angular versions up to 12 are no longer supported, If you want to install any specific older version then use the following syntax.**

**C:\>*npm  install  -g  @angular/cli@12    [version number]*Create a new Workspace  
  
- A workspace comprises of set of Angular projects.  
- You can maintain a common library, which is accessible across projects.  
  
1. Open any location of your PC in command prompt.  
  
            E:\>  
  
2. Run the following command  
 *E:\>ng  new  angular-workspace  --create-application=false*  
  
3. Open the workspace location is your editor "Visual Studio Code"  
  
       E:\angular-workspace>**

**Angular Workspace File System**

**FAQ: Where the typescript target files are stored in angular project?  
Ans : "./dist/out-tsc"  
  
File/Folder                Description  
  
tsconfig.json:           It comprises of rules defined for typescript used for all projects**

**in workspace.  
  
README.md :     It is a help document created by developers to guide other**

**developers in building, debugging, testing and deploying.**

**package. json :        It comprises of project meta data.  
                            - Version  
                            - Copyrights  
                            - License  
                          - Dependencies  
                          - Author etc...  
  
package-lock. json: It comprises of detailed information related to dependencies**

**used in project.**

**angular. json :         It is project configuration file.**

**.gitignore :               It specifies the resources to ignore while publishing on GIT.**

**.editorconfig :    It is editors configuration file. It sets common rules for all**

**developers working on various editors.  
  
node\_modules :  It comprises of all library files that are installed in your**

**workspace.  
  
Note: Download "Angular Language Service" extention for VS Code that will help you while working with angular library.  
  
Adding a new Project into Workspace  
  
1. Open your workspace location in terminal or command prompt.**

**E:\angular-workspace>  
  
2. Run the following command  
        > ng  generate application  [appName]  
        > ng  generate application  shopping  
  
        ? Would you like to add Angular routing? (y/N) n  
        ? Which stylesheet format would you like to use? CSS  
         
3. Run the project  
 >ng serve --project=shopping or >ng s --project=shopping  
  
  
    Your project starts on a local server with address** [**http://localhost: 4200**](%20%20%20%20%20%20%20%20%20%20%20%20http://localhost:%204200)

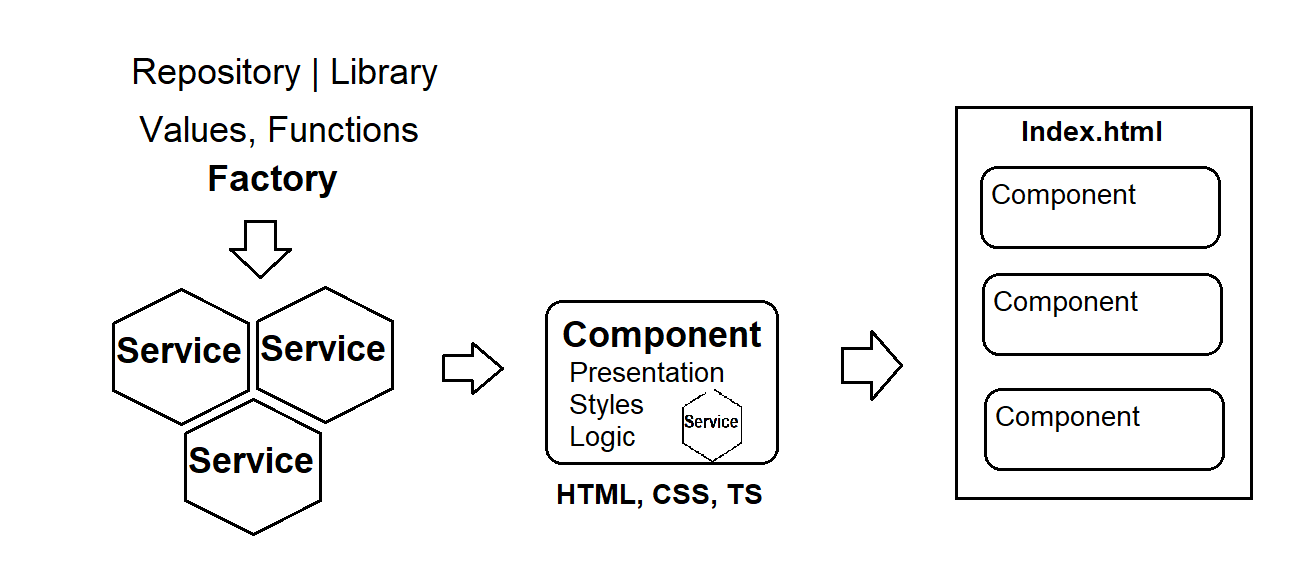
**Project File System**  
 **File/Folder                    Description**  
  
**tsconfig.app.json       It comprises of rules for typescript used in current application.  
  
tsconfig.spec.json     It is a test specification file, that comprises methods which are**

**used for testing typescript rules.  
  
src                             It is a project folder that comprises of all project resources.  
  
src/app                      It comprises of application dynamic resources like  
                                - components  
                                - services  
                              - modules  
                                - pipes etc...  
  
src/assets                   It comprises of static resources like  
                                - images  
                                - documents [docs, pdf]  
                                - multimedia [audio, video] etc..  
  
favicon.ico                   It defines the shortcut icon displayed in browser title bar.  
  
index.html                  It is the startup page.  
  
main.ts                       It is the entry point for application. The execution of application**

**starts from main.  
  
styles.css                 It comprises of global styles used in application.  
src/app/     It imports and exports all library that is required for application.**

**app.module.ts**

**3. Angular Application Architecture & FLOW**

 **1. Factory  
    - It is a set for values and functions.  
    - It uses single call mechanism.  
  - In single call mechanism an object is created every time when function is**

**required.  
  
2. Service  
    - It is a pre-defined business logic.  
    - It comprises of set of factories.  
    - It uses single ton mechanism.  
   - Single ton creates an object for first request and uses the same across all other**

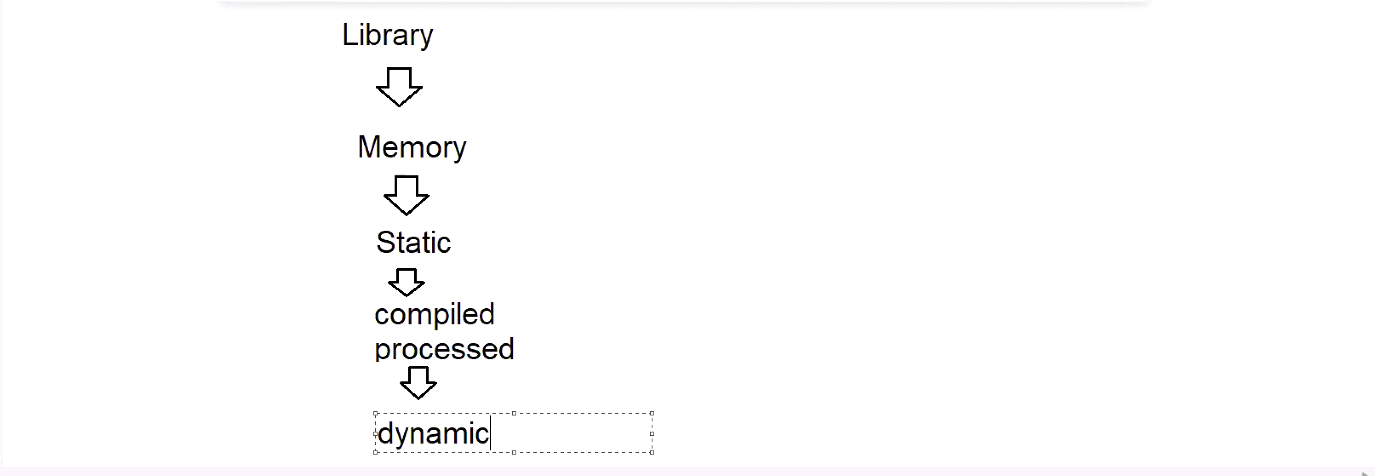
**requests.**

**3. Component  
    - It comprises of presentation, styles and logic.  
    - It is a template for designing and functionality**

**- Component uses services.  
    - Markup is using HTML, styles by using CSS and logic with Typescript.  
  
4. SPA Environment  
    - Index is the start up page in SPA.  
    - Index page uses components for presentation and functionality.  
    - SPA is component based.  
  
Ex:  
1. Go to "app" folder  
2. Customize "app.component.html"  
  
<div class="container">  
  <h1>Angular Shopping</h1>  
</div>  
  
3. Add styles in "app.component.css"  
  
.container {  
    display: flex;  
    justify-content: center;  
    align-items: center;  
    height: 500px;  
}  
h1 {  
    padding: 20px;  
    border:1px solid gray;  
    box-shadow: 2px 2px 3px black;  
    font-family: Arial;  
    border-radius: 20px;  
    background-color: yellow;  
}  
  
4. Index.html uses App Component by its selector name "app-root"  
  
 <body>  
    <app-root> </app-root>  
 </body>  
  
Application Flow:**

**1. User Requests application from browser.**[**"http://localhost:4200/**](%22http://localhost:4200/%224) **2. It invokes "main. ts"**

**3. main. ts imports "AppModule" and bootstraps "AppModule".  
Note: Bootstrapping is the process of converting the static library into dynamic. [Injecting => Compiling => Processing]**

 **4. app.module.ts  imports all library required for project.  
5. app.module.ts will bootstrap the components that you want in index.html.  
6. Index.html uses the bootstrap components by using selectors.  
            <app-login> <app-register>   <app-root>**

**4. ANGUlar components**

**- Components are building blocks for Angular Applications.  
- Every component comprises  
        a) Presentation  
        b) Styles  
        c) Logic  
  
-Presentation is designed by using HTML.  
-Styles are defined by using CSS  
-Logic is defined by using TypeScript, which its trans-compiled into**

**JavaScript.**

**-There are various techniques of designing a component  
    1. You can create a component manually  
    2. You can create a component using CLI commands [Scaffolding]**

**- Component structure can be designed in 2 ways.  
    1. Inline Documentation  
    2. Code Behind Documentation**

**1. Inline Documentation  
- Inline Documentation Technique allows you to configure component design,**

**styles and logic all in one file.  
- Inline technique reduces the number of requests and improves the page load**

**time.  
- Inline technique is not good for reusability, extensibility (regular changes**

**made) and testability.**

**1. Manually adding a component using Inline Technique  
      - Technically component is TypeScript class  
        class  NetflixRegister { }  
  
 - Component class is defined by using a directive or**

**marker   "@Component()"  
  
        @Component()  
        class  NetflixRegister  
        {  
  
        }  
  
FAQ: What is @Component()?  
Ans : It is a directive [marker].  
   
FAQ: What is a directive?  
Ans : Directive comprises of meta data about your component, which is given to compiler.  
  
- @Component directive is present in "@angular/core" library.  
  
Syntax:  
        import  { Component }  from  "@angular/core";  
  
        @Component()  
        class  NetflixRegister  
        {  
        }  
  
- Every component must return markup.  
- Every component must have a selector, which is required to import and use**

**the component in any page.  
  
Syntax:  
   import  { Component } from  "@angular/core";  
  
   @Component({  
        selector: "app-register",  
        template: "<h2>....</h2>",  
        styles: ["h2{'color':'red'}"]  
    })  
    export class  NetflixRegister  
    {  }**

**-Every component must be registered in the "declarations[]" of "app.module.ts".  
    declarations: [  
        NetflixRegister  
    ]  
  
- Every application starts with multiple components, you have to define your**

**component in "bootstrap[]" of app.module.ts  
  
    bootstrap: [  
        NetflixRegister  
     ]  
  
- Go to  "Index.html"  
  
    <body>  
        <app-register> </app-register>  
    </body>  
  
Note: Index.html must have exactly the same components defined in bootstrap of "app.module.ts".  
  
Ex:  
1. Go to "app" folder and add "components" folder  
  
2. Add a new file into components  
  
 netflix-register.component.ts  
  
import { Component } from "@angular/core";  
  
@Component({  
    selector: 'app-register',  
    template: `  
      <div class="register-container">  
       <div>  
       <p>Ready to watch? Enter your email to create or restart your membership.</p>  
        <div>  
            <input type="email" placeholder="Email address">  
            <button>  
                Get Started >  
            </button>  
        </div>  
       </div>  
      </div>  
    `,  
    styles: [".register-container{display:flex; justify-content:center; align-items:center; height: 400px}"]  
})  
export class NetflixRegister  
{  
  
}  
  
3. Go to app.module.ts  
  
     declarations: [  
            NetflixRegister  
        ]  
    bootstrap : [  
            NetflixRegister  
        ]  
  
4. Go to Index.html          
  
        <body>  
            <app-register> </app-register>  
        </body>  
  
Setup Bootstrap for Angular Project  
  
1. Open Your workspace location in terminal  
  
        D:\Angular-Workspace>  
  
2. Run the following commands  
  
        > npm  install bootstrap --save  
        > npm  install bootstrap-icons  --save  
  
3. Go to "styles.css"  
  
@import "../../../node\_modules/bootstrap/dist/css/bootstrap.css";  
@import "../../../node\_modules/bootstrap-icons/font/bootstrap-icons.css";  
  
EX:  
import { Component } from "@angular/core";  
  
@Component({  
    selector: 'app-register',  
    template: `  
      <div class="register-container bg-dark text-white">  
       <div>  
       <p>Ready to watch? Enter your email to create or restart your membership.</p>  
        <div class="input-group input-group-lg">  
            <input type="email" class="form-control" placeholder="Email address">  
            <button class="btn btn-danger">  
                Get Started <span class="bi bi-chevron-right"></span>  
            </button>  
        </div>  
       </div>  
      </div>  
    `,  
    styles: [".register-container{display:flex; justify-content:center; align-items:center; height: 600px}"]  
})  
export class NetflixRegister  
{  
}**

**2. Adding Component using Code Behind Technique  
- In this technique component is defined using 4 files  
  
        1. component.html            Presentation  
        2. component.css            Styles  
        3. component.ts            Logic  
        4. component.spec.ts       Testing  
- Clean separation of presentation, styles and logic.  
- Easy to extend and test.  
- Easy to re-use.  
- The number of requests will increase, hence it will effect the page load time.  
- If you need a component with regular extentions then better use code**

**behind technique.  
Ex:  
1. Go to components folder and add a sub folder "login".  
  
2. Add following files into login folder  
        login.component.ts  
        login.component.html     login.component.css**

**login.component.ts  
  
import { Component } from "@angular/core";  
  
@Component({  
    selector: 'app-login',  
    templateUrl: './login.component.html',  
    styleUrls: ['./login.component.css']  
})  
export class LoginComponent  
{  
  
}  
  
login.component.html  
  
<div class="container-fluid">  
    <div class="login-form">  
        <form>  
            <h2> <span class="bi bi-person-fill"></span> User Login</h2>  
            <dl>  
                <dt>User Name</dt>  
                <dd><input type="text" class="form-control"></dd>  
                <dt>Password</dt>  
                <dd><input type="password" class="form-control"></dd>  
            </dl>  
            <button class="btn btn-primary w-100">Login</button>  
        </form>  
    </div>  
</div>  
  
login.component.css**

**.login-form {  
    display: flex;  
    justify-content: center;  
    align-items: center;  
    height: 500px;  
}  
form {  
    border:1px solid gray;  
    box-shadow: 2px 2px 3px gray;  
    padding: 20px;  
    border-radius: 20px;  
}**

**3. Register the component is app.module.ts**

**declarations : [  LoginComponent ]  
 bootstrap: [ LoginComponent ]  
  
4. Index.html  
        <app-login> </app-login>  
  
Note: In code behind technique @Component() directive will use  
                a) templateUrl : " "  
                b) styleUrls  : [ ]  
        In inline technique  @Component() uses  
                a) template : ""  
                b) styles    : []**

**Adding Components by using CLI commands  
- Angular supports "Scaffolding".  
- Scaffolding is a technique where the code is generated according to the**

**developers requirement.  
- Run the following commands in terminal  [ app\components> ]  
  
Command                               Purpose  
  
 ng generate compo\_name    It generates a new component.  
                    (or)                It adds 4 files  
 ng  g   c  name                      .html, .css, .ts, .spec.ts  
                                          It registers is app.module.ts  
  
--dry-run                           The "--dry-run" option means no changes were**

**made.  
  
--inline-template                     It will not generate .html  
  
--inline-style                           It will not generate .css  
  
--skip-tests                                It will not generate .spec.ts  
  
Syntax:  
 >ng generate component  home  --dry-run  --inline-template --inline-style  --skip-tests  
> ng  g  c  home**

**5. Data Binding in Angular**  
  
**- Data binding is a technique used in web applications, where the data is**

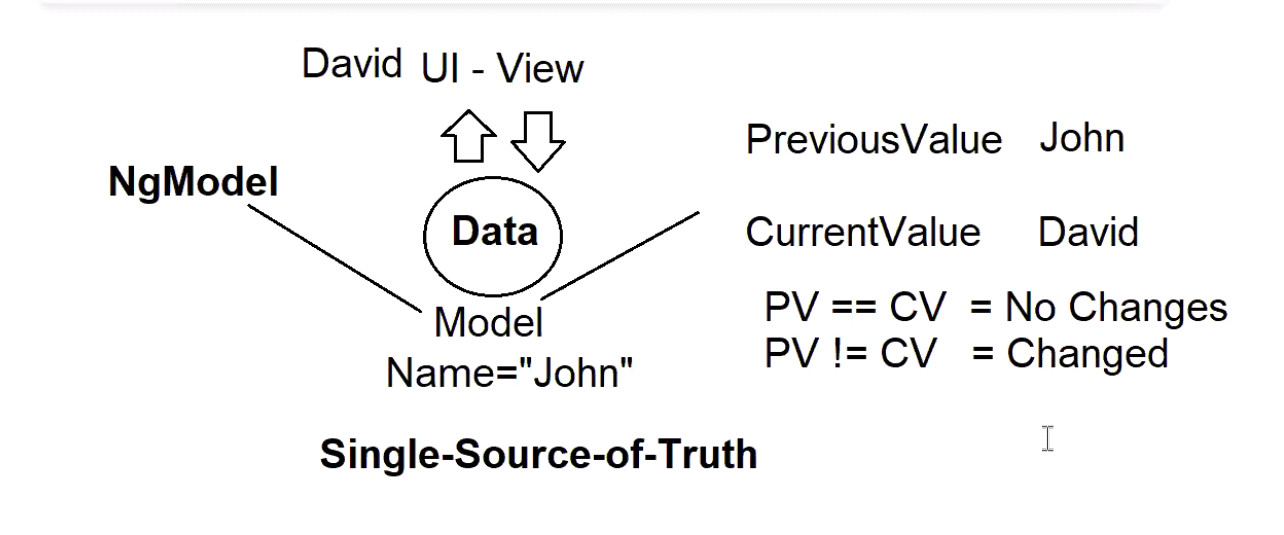
**accessed from source and binded to UI. It identifies the changes in UI and**

**update back to source.  
  
- JavaScript and jQuery handles data binding by using lot of DOM methods**

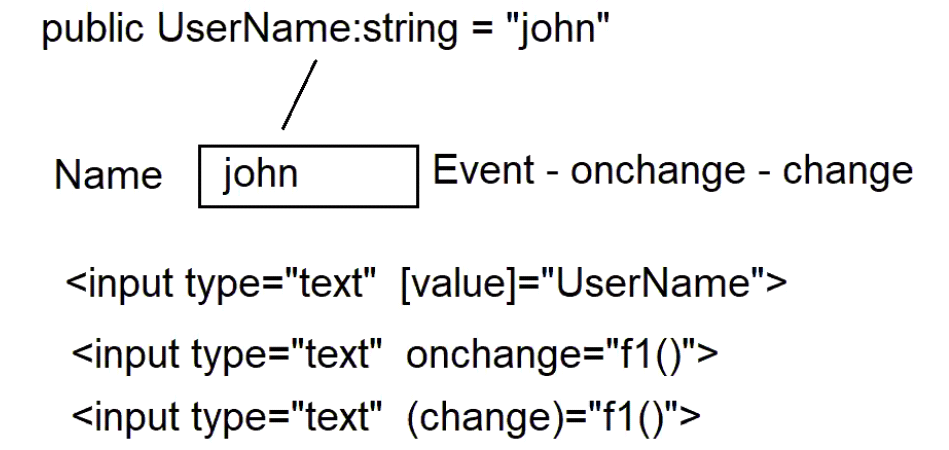
**and Events.  
  
- Angular is completely de-coupled from DOM manipulations.  
  
- Angular implicitly uses Frameworks like  
        a) MVC [Model-View-Controller]  
        b) MVVM [Model-View View-Model]  
  
- MVC and MVVM framework will handle data binding.  
- Framework is a set of libraries with implicit flow controlled for application.  
  
 What is MVC?  
- It is an software architectural pattern.  
- Architectural pattern can build and control application flow.  
- It is same like design pattern but have broader scope.  
- Trygve introduced MVC in early 1970's with Small Talk.  
- Originally designed for "DeskTop Applications"  
- Now it is used in web applications.  
- Every technology knows and uses MVC  
  
        Java                    Spring MVC  
        Python                Django, Flask, Grok  
        PHP                    Cake PHP, Code Igniter  
        .NET                    ASP.NET MVC  
        Ruby                    Ruby on Rails  
        Perl                    Catalyst, Dancer  
        JavaScript          SPINE, AngularJS, Backbone etc..  
  
MVC Architecture  
- It separates application into 3 components  
        a) Model      
        b) View  
        c) Controller  
- Model Represents Data  
- View Represents UI  
- Controller Represents Application Logic.  
- Angular uses MVC framework for data binding.  
- Data Binding is classified into 2 types  
        a) One Way Binding  
        b) Two Way Binding  
  
One Way Binding  
- It is the process of accessing data from source and binding to UI.  
- It will not update the changes in UI to source.  
- You can binding the value by using various binding techniques  
    a) Attribute Binding  
    b) Property Binding  
    c) Interpolation  
  
- Interpolation is the process of binding a value by using data binding expression "{{  }}"  
  
Syntax:  
            public  UserName: string = "John";  
  
            <input type="text"  value="{{Username}}">    
            <span> {{ UserName }} </span>  
  
- Interpolation is forward only and one time only.  
  
Syntax:  
            public  Stock: boolean = true;  
   
            <input type="checkbox"  checked="{{Stock}}">  
  
 Note: Checked attribute will not update the boolean or assign boolean.  
  
Property Binding  
- It allows to bind any value to element.  
- It keeps the value updated.  
- Property binding is defined by specifying the property name in "[ ]"  
  
        <input type="checkbox" [checked]="Stock">  
  
- Every element don't have all the properties that are available as attributes.  
  
        <table  height="200"  width="300" border="1">  
  
        var table = document. createElement("table");  
        table. width = "300";  
        table. height ="200";       // error : height is not a property**

**- You have to bind directly to attribute by using the prefix "attr"  
  
        public tWidth: number = 300;  
        public tHeight: number = 200;  
  
        <table  [width]="tWidth"  [attr.height]="tHeight" border="1">  
        <table  [width]="tWidth"  [height]="tHeight" border="1">  
            Error: height is not a property of table.  
  
Ex:  
data-binding.component.ts  
  
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-data-binding',  
  templateUrl: './data-binding.component.html',  
  styleUrls: ['./data-binding.component.css']  
})  
export class DataBindingComponent {  
   public Stock: boolean = false;  
   public UserName: string = "John";  
}  
  
data-binding.component.html  
  
<div class="container-fluid">  
   <h2>Interpolation</h2>  
   <dl>  
    <dt>Stock</dt>  
    <dd>  
        <input type="checkbox" [checked]="Stock" > {{(Stock==true)?"Available":"Out of Stock"}}  
    </dd>  
    <dt>Name</dt>  
    <dd>  
        <input type="text" [value]="UserName">  
    </dd>  
   </dl>  
</div>**

**Two Way Data Binding**  
**- Two way refers to bi-directional data transfer.  
- The data present in source is binded to UI.  
- The modifications on data in UI will update to source.  
-  It is managed by "Model" in MVC.  
- Angular model is "NgModel", which is known as a Directive.  
- It refers to "Single-Source-Of-Truth".  
- It tracks and keeps information about the changes in data.**

 **" Model  is  Single-Source-Of-Truth"  
  
- NgModel is a member of "FormsModule" which is defined in  
  "@angular/forms"  
  
- In order to use "NgModel" you have to import in "app.module.ts"  
  
    import  { FormsModule } from  "@angular/forms";  
  
    imports: [  
                FormsModule  
               ]  
  
- NgModel uses property and event binding techniques.  
- NgModel uses [property] binding to get value from source and attach to UI.  
- NgModel uses (event) binding to identify the changes and update back to**

**source.  
  
Syntax:          
            [ ]        => Property Binding [value, checked..]  
            ( )        => Event Binding    [change]  
  
    <input type="text"  [(ngModel)]="SourceRefName">**

 **Ex:  
databinding.component.ts  
  
   public UserName:string = "John";  
  
databinding.component.html  
  
<div class="container-fluid">  
   <h2>Two Way Binding</h2>  
   User Name : <input type="text" [(ngModel)]="UserName">  
   <h4>Hello ! {{UserName}} </h4>  
</div>  
  
Ex: MVC  
- Add contracts folder  
- Add a new file  
        "ProductContract.ts"  
  
export interface ProductContract  
{  
    Name: string;  
    Price: number;  
    Stock: boolean;  
    ShippedTo: string;  
}  
  
Databinding.component.ts  
  
import { Component } from '@angular/core';  
import { ProductContract } from '../../contracts/ProductContract';  
  
@Component({  
  selector: 'app-data-binding',  
  templateUrl: './data-binding.component.html',  
  styleUrls: ['./data-binding.component.css']  
})  
export class DataBindingComponent {  
     public Product: ProductContract = {  
        Name: "",  
        Price: 0,  
        Stock: false,  
        ShippedTo: ""  
     }  
     public UpdatedProduct: ProductContract = {  
        Name: "",  
        Price: 0,  
        Stock: false,  
        ShippedTo: ""  
     }  
     public RegisterClick():void {  
         this. UpdatedProduct = this. Product;  
     }  
}  
  
databinding.component.html  
  
<div class="container-fluid">  
   <div class="row">  
        <div class="col-3">  
            <h2>Register Product</h2>  
            <dl>  
                <dt>Name</dt>  
                <dd><input type="text" [(ngModel)]="Product. Name" class="form-control"></dd>  
                <dt>Price</dt>  
                <dd><input type="text" [(ngModel)]="Product. Price" class="form-control"></dd>  
                <dt>Stock</dt>  
                <dd class="form-switch"> <input [(ngModel)]="Product. Stock" type="checkbox" class="form-check-input"> Available</dd>  
                <dt>Shipped To</dt>  
                <dd>  
                    <select [(ngModel)]="Product.ShippedTo" class="form-select">  
                        <option>Delhi</option>  
                        <option>Hyd</option>  
                    </select>  
                </dd>  
            </dl>  
            <button (click)="RegisterClick()" class="btn btn-primary w-100">Register</button>  
        </div>  
        <div class="col-9">  
           <h2>Details</h2>  
           <dl>  
            <dt>Name</dt>  
            <dd [innerText]="UpdatedProduct.Name"></dd>  
            <dt>Price</dt>  
            <dd>{{UpdatedProduct.Price}}</dd>  
            <dt>Stock</dt>  
            <dd>{{(UpdatedProduct.Stock==true)?"Available":"Out of Stock"}}</dd>  
            <dt>Shipped To</dt>  
            <dd>{{UpdatedProduct.ShippedTo}}</dd>  
           </dl>  
        </div>  
   </div>  
</div>  
  
                                    MVVM  
- It is an software architectural pattern.  
- Model-View  View-Model  
- Model configures memory in View and uses the memory for data.  
- You don't need the use of controller for handling data updates.  
- NgModel will manage MVVM.  
  
Syntax:  
    <input type="text"  ngModel   #Name="ngModel">  
  
- Every ViewModel object contains lot of properties  
  
        Name. value  
        Name. pristine  
        Name. dirty  
        Name. valid  
        Name. invalid  
        Name. touched  
        Name. untouched etc...  
  
        <p> {{Name. value }}  
  
        <input type="text" required minlength maxlength pattern>**

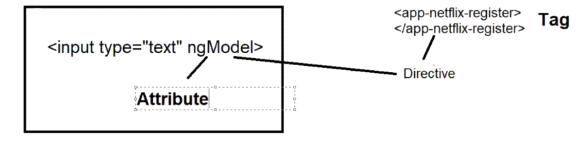
**Ex:  
databinding.component.html  
  
<div class="container-fluid">  
   <div class="row">  
        <div class="col-3">  
            <h2>Register Product</h2>  
            <dl>  
                <dt>Name</dt>  
                <dd><input type="text" ngModel #Name="ngModel" class="form-control"></dd>  
                <dt>Price</dt>  
                <dd><input type="text" ngModel #Price="ngModel" class="form-control"></dd>  
                <dt>Stock</dt>  
                <dd class="form-switch"> <input ngModel #Stock="ngModel" type="checkbox" class="form-check-input"> Available</dd>  
                <dt>Shipped To</dt>  
                <dd>  
                    <select ngModel #ShippedTo="ngModel"  class="form-select">  
                        <option>Delhi</option>  
                        <option>Hyd</option>  
                    </select>  
                </dd>  
            </dl>  
        </div>  
        <div class="col-9">  
           <h2>Details</h2>  
           <dl>  
            <dt>Name</dt>  
            <dd>{{Name.value}}</dd>**

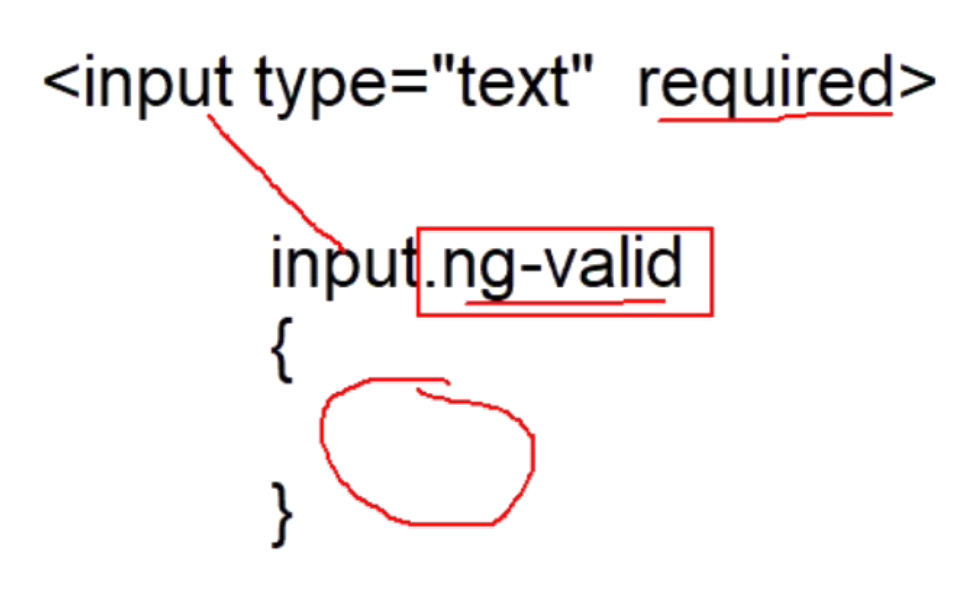
**<dt>Price</dt>  
            <dd>{{Price.value}}</dd>  
            <dt>Stock</dt>  
            <dd>{{(Stock.value==true)?"Available": "Out of Stock"}}</dd>  
            <dt>Shipped To</dt>  
            <dd>{{ShippedTo.value}}</dd>  
           </dl>  
        </div>  
   </div>  
</div>**

**6. Consuming Data from API**  
  
**- There are 3 types of specification for services on Web.  
        a) SOAP  
        b) REST  
        c) JSON  
  
- SOAP  
    Service Oriented Architecture Protocol  
    Consumer sends XML request.  
    Provider sends XML response.  
  
- REST  
    Representational State Transfer  
    Consumer sends a query request.  
    Provider sends XML or JSON response.  
  
- JSON  
    JavaScript Object Notation  
    Consumer sends JSON request.  
    Provider send JSON response.  
  
Syntax: REST  
             
http://sitename.com/products?id=2   
http://sitename.com/products?category=mobiles http://sitename.com/products?category=mobiles&brand=lg "http://sitename.com/products?category=mobiles.**

**[Routing]**[**http://sitename.com/products/mobiles/lg**](http://sitename.com/products/mobiles/lg)**[Routing]  
  
            {  
              Name: "TV",  
              Price: 45000.44  
            }  
  
            <Product>  
               <Name> TV </Name>  
               <Price> 45000.44 </Price>  
            </Product>  
  
- JavaScript provides "fetch()" promise to access data from API.  
  
Syntax:  
        fetch("url")  
        .then(function(response){        => binary format  
           return response. json();  
        })      
        .then(function(data){  
            ...present data..  
        })  
  
Ex: Fakestore API  [Data for ERP]**[**http://fakestoreapi.com**](http://fakestoreapi.com/) **Methods:  
   
GET**[**http://fakestoreapi.com/products**](http://fakestoreapi.com/products)**[ { }, { } ]  
GET** [**http://fakestoreapi.com/products/**](http://fakestoreapi.com/products/%20HYPERLINK%20%20%20%22http://fakestoreapi.com/products/1%22%20HYPERLINK%20%22http://fakestoreapi.com/products/1%22%20HYPERLINK%20%22http://fakestoreapi.com/products/1%221)**1        { }  
  
Ex:  
1. Add a new contract  
         
FakestoreProductContract.ts  
  
export interface FakestoreProductContract  
{  
    id: number;  
    title: string;  
    price: number;  
    description: string;  
    category: string;  
    image: string;  
    rating: {rate: number, count: number}  
}  
  
2.  fakestore.component.ts  
  
import { Component, OnInit } from '@angular/core';  
import { FakestoreProductContract } from '../../contracts/FakestoreProductContract';  
  
@Component({  
  selector: 'app-fakestore',  
  templateUrl: './fakestore.component.html',  
  styleUrls: ['./fakestore.component.css']  
})  
export class FakestoreComponent implements OnInit {  
   constructor(){  
 }  
   public Product: FakestoreProductContract = {  
       id: 0,  
       title: '',  
       price: 0,  
       description: '',  
       category:'',  
       image:'',  
       rating: {rate: 0, count: 0}  
   }  
   public LoadProduct(id: number):void {  
      fetch(`http://fakestoreapi.com/products/${id}`)  
      .then(response=> response. json())  
      .then(data=> {  
         this. Product = data;  
      })  
   }  
   ngOnInit(): void {  
      this. LoadProduct(1);  
   }  
   public Count: number = 1;  
   public NextClick():void {  
      this. Count++;  
      this. LoadProduct(this. Count);  
   }  
   public PreviousClick():void {  
     this. Count--;  
     this. LoadProduct(this. Count);  
   }  
}  
  
3. fakestore.component.html  
  
<div class="container-fluid">  
    <h2>Product Details</h2>  
    <dl>  
        <dt>Title</dt>  
        <dd>{{Product. title}}</dd>  
        <dt>Price</dt>  
        <dd>{{Product. price}}</dd>  
        <dt>Preview</dt>  
        <dd>  
            <img [src]="Product. image" width="200" height="200">  
        </dd>  
        <dt>Rating</dt>  
        <dd><span class="bi bi-star-fill text-success"></span> {{Product. rating. rate}} [{{Product. rating. count}}]</dd>  
    </dl>  
    <button (click)="PreviousClick()" class="btn btn-success me-2">  
        <span class="bi bi-chevron-left"></span>**  
**</button>  
    <button (click)="NextClick()" class="btn btn-success">  
        <span class="bi bi-chevron-right"></span>  
    </button>  
</div>**

**7. Angular Directives**  
**- Directive is a function in angular that can handle various tasks.  
        a) It can return markup.  
        b) It can extend markup.  
        c) It can make the markup more interactive and responsive.  
  
- A directive can be used as  
        a) Element             : to return markup  
        b) Attribute            : to extend markup  
        c) Class               : to make markup more responsive and interactive.**

 **Syntax:  
        <app-netflix-header>                    Element  
        <input type="text" ngModel>        Attribute  
         input.ng-invalid { }                       Class**

 **- Directives in Angular are classified into 3 major groups  
            i) Component Directives  
            ii) Structural Directives  
            iii) Attribute Directives  
  
i. Component Directive:  
- It returns markup.  
- It uses Angular component which renders markup.  
  
Syntax:  
     <app-netflix-register> </app-netflix-register>  
  
ii. Structural Directives:  
- A structural directive is used  
        a) To add elements  
        b) To remove elements  
        c) To repeat elements  
- They can control the DOM structure by adding, remove and repeating**

**elements in page.  
- Angular structural directives are  
        a) NgIf b) NgSwitch c) NgFor  
  
  1. NgIF:  
- It is a structural directive.  
- It can add element to page or remove element from page.  
- It uses a boolean value or expression.  
- Structural directives are added to HTML elements by using "\*".  
  
        *<div  \*ngIf="true/false" (or) BooleanExpression>*  
- Every element can have only one structural directive.  
  
        <*div  \*ngIf=""    \*ngFor="">     invalid*  
  
- Boolean "false" will remove element from page.  
- Boolean "true" will add element to page. *<div  \*ngIf="false">            => removed  
        </div>  
     
        <div  \*ngIf="true">            => added  
        <div>*  
Ex:  
FakestoreProductContract.ts  
  
export interface FakestoreProductContract  
{  
    id: number;  
    title: string;  
    price: number;  
    description: string;  
    category: string;  
    image: string;  
    rating:{rate: number, count: number}  
}**

**fakestore.componen.ts**

**import { Component, OnInit } from '@angular/core';  
import { FakestoreProductContract } from '../../contracts/FakestoreProductContract';  
  
@Component({  
  selector: 'app-fakestore',  
  templateUrl: './fakestore.component.html',  
  styleUrls: ['./fakestore.component.css']  
})  
export class FakestoreComponent implements OnInit {  
   constructor(){}  
   public Product: FakestoreProductContract = {  
       id: 0,  
       title: '',  
       price: 0,  
       description: '',  
       category:'',  
       image:'',  
       rating: {rate: 0, count: 0}  
   }  
   public LoadProduct(id: number):void {  
      fetch(`http://fakestoreapi.com/products/${id}`)  
      .then(response=> response. json())  
      .then(data=> {  
         this. Product = data;  
      })  
   }  
   ngOnInit(): void {  
      this. LoadProduct(1);  
   }  
   public Count: number = 1;  
   public NextClick():void {  
      this. Count++;  
      this. LoadProduct(this. Count);  
   }  
   public PreviousClick():void {  
     this. Count--;  
     this. LoadProduct(this. Count);  
   }  
}**

**fakestore.component.html  
  
<div class="container-fluid">  
    <div class="description" \*ngIf="toggleDescription. value">  
        <textarea rows="4" cols="40" disabled>  
            {{Product. description}}  
        </textarea>  
    </div>  
    <h2>Product Details</h2>  
    <div class="form-switch">  
       <input type="checkbox" ngModel #toggleDescription="ngModel" class="form-check-input"> View Description  
    </div>  
    <dl>  
        <dt>Title</dt>  
        <dd>{{Product.title}}</dd>  
        <dt>Price</dt>  
        <dd>{{Product.price}}</dd>  
        <dt>Preview</dt>  
        <dd>  
            <img [src]="Product.image" width="200" height="200">  
        </dd>  
        <dt>Rating</dt>  
        <dd><span class="bi bi-star-fill text-success"></span> {{Product.rating.rate}} [{{Product.rating.count}}]</dd>  
    </dl>  
    <button (click)="PreviousClick()" class="btn btn-success me-2">  
        <span class="bi bi-chevron-left"></span>  
    </button>  
    <button (click)="NextClick()" class="btn btn-success">  
        <span class="bi bi-chevron-right"></span>  
    </button>  
</div>  
  
fakestore.component.css  
  
.description {  
    position: fixed;  
    left: 300px;  
    top: 200px;  
}  
  
NgIF with Alternative Block  
- Alternative block refers to container that renders into page when condition**

**is false.  
Syntax:  
         <div \*ngIf="condition; then  ifBlockId else elseBlockId">  
         </div>  
     
- The "then and else" keywords refers to clauses, they can't control HTML**

**blocks.  
- Hence you need to create a dynamic block in Angular by using  
        "<ng-template>"  
  
Syntax:  
    <div  \*ngIf="true; then  ifBlock  else  elseBlock">  </div>  
     
    <ng-template #ifBlock>  
        ... on true ....  
    </ng-template>  
  
    <ng-template #elseBlock>  
        ..on false...  
    </ng-template>  
  
Note: The dynamic templates  <ng-template> must be outside the condition block.**

**CONTENT PROJECTION**

**- It is the process of creating a template explicitly and rendering into the View**

**[UI].**

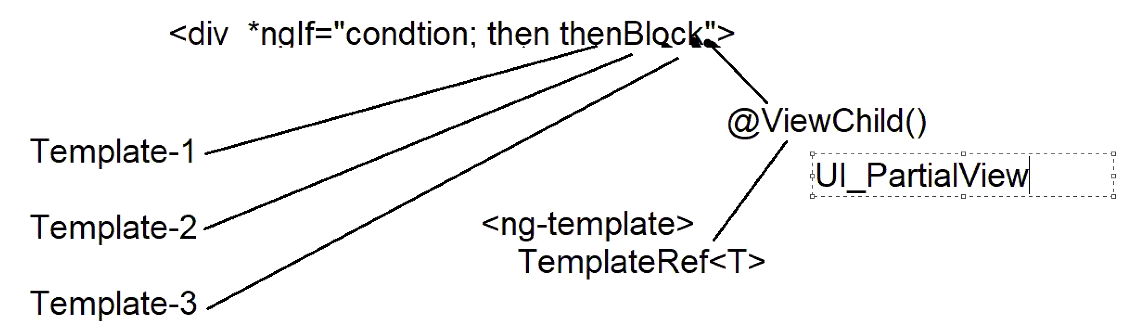
**- Generally it is used for multiple else blocks as like else-if statement.  
- Template is created by using "TemplateRef<T>"**

**- @ViewChild(): It is a directive which is used to rendering template into UI.**

**- Content projection is not suitable if there are many options. In that case it is**

**better to use ngSwitch.  
  
Creating a Template Explicitly:  
  
    @ViewChild()  public  TemplateName:TemplateRef<any>|null;**

**"any" means any type of component.**

**"null" means the template is configured dynamically.  
  
    <ng-template   #TemplateName>  
    </ng-template>**

**Rendering Template in UI:  
         
      <div \*ngIf="condition; thenBlock  TemplateName> </div>  
  
Ex:  
/components> ng g c content-projection  --skip-tests**

**Ex-1:  
content-projection.component.ts  
  
import { Component, OnInit, TemplateRef, ViewChild } from '@angular/core';  
  
@Component({  
  selector: 'app-content-projection',  
  templateUrl: './content-projection.component.html',  
  styleUrls: ['./content-projection.component.css']  
})  
export class ContentProjectionComponent implements OnInit {  
    constructor(){}  
  
    public ThenBlock:TemplateRef<any>|null = null;  
    public TemplateName: string = "Choose Template";**

**@ViewChild('Template1',{static:true}) public Template1:TemplateRef<any>|null = null;  
    @ViewChild('Template2', {static: true}) public Template2:TemplateRef<any>|null = null;  
    @ViewChild('Template3', {static: true}) public Template3:TemplateRef<any>|null = null;  
     
    ngOnInit(): void {  
        this.ThenBlock = this.Template1;  
    }  
    public TemplateChange():void{  
        switch(this. TemplateName)  
        {  
           case "Template1":  
           this. ThenBlock = this.Template1;  
           break;  
           case "Template2":  
           this. ThenBlock = this.Template2;  
           break;  
           case "Template3":  
           this. ThenBlock = this.Template3;  
           break;  
        }   } }**

**content-projection.component.html  
  
<div class="container-fluid">  
    <h2>Content Projection</h2>  
    <div>  
        <select (change)="TemplateChange()" [(ngModel)]="TemplateName">  
            <option>Choose Template</option>  
            <option value="Template1">Tempalte-1</option>  
            <option value="Template2">Tempalte-2</option>  
            <option value="Template3">Tempalte-3</option>  
        </select>  
    </div>  
    <div \*ngIf="true; then ThenBlock"></div>  
    <ng-template #Template1>  
        Template-1  
    </ng-template>  
    <ng-template #Template2>  
        Template-2  
    </ng-template>  
    <ng-template #Template3>  
        Template-3  
    </ng-template>  
</div>  
  
Ex-2: Flipart Product  
mobile-details.component.ts  
  
import { Component, OnInit, TemplateRef, ViewChild } from '@angular/core';  
  
@Component({  
  selector: 'app-mobile-details',  
  templateUrl: './mobile-details.component.html',  
  styleUrls: ['./mobile-details.component.css']  
})  
export class MobileDetailsComponent implements OnInit {  
  constructor(){}  
  
  public ThenBlock:TemplateRef<any>|null = null;  
  
  @ViewChild('View1', {static:true}) public View1:TemplateRef<any>|null = null;  
  @ViewChild('View2', {static:true}) public View2:TemplateRef<any>|null = null;  
  @ViewChild('View3', {static:true}) public View3:TemplateRef<any>|null = null;  
  @ViewChild('View4', {static:true}) public View4:TemplateRef<any>|null = null;  
  @ViewChild('View5', {static:true}) public View5:TemplateRef<any>|null = null;  
  @ViewChild('View6', {static:true}) public View6:TemplateRef<any>|null = null;  
   
  ngOnInit(): void {  
     this.ThenBlock = this.View1;  
  }  
  public ChangeView(viewName:string){  
     switch(viewName){  
        case "View1":  
        this.ThenBlock = this.View1;  
        break;  
        case "View2":  
        this.ThenBlock = this.View2;  
        break;  
        case "View3":  
        this.ThenBlock = this.View3;  
        break;  
        case "View4":  
        this.ThenBlock = this.View4;  
        break;  
        case "View5":  
        this.ThenBlock = this.View5;  
        break;  
        case "View6":  
        this.ThenBlock = this.View6;  
        break;    }  } }**

**mobile-details.component.html  
  
<div class="container-fluid">  
     <div class="row mt-3">  
        <div class="col-2">  
           <nav>  
             <div (mouseover)="ChangeView('View1')">  
                <img src="assets/view1.jpg" width="50" height="50">  
             </div>  
             <div (mouseover)="ChangeView('View2')">  
                <img src=" assets/view2.jpg" width="50" height="50">  
             </div>  
             <div (mouseover)="ChangeView('View3')">  
                <img src="assets/view3.jpg" width="50" height="50">  
             </div>  
             <div (mouseover)="ChangeView('View4')">  
                <img src="assets/view4.jpg" width="50" height="50">  
             </div>  
             <div (mouseover)="ChangeView('View5')">  
                <img src="assets/view5.jpg" width="50" height="50">  
             </div>  
             <div (mouseover)="ChangeView('View6')">  
                <img src="assets/view6.jpg" width="50" height="50">  
             </div>  
           </nav>  
        </div>  
        <div class="col-4">  
            <div \*ngIf="true; then ThenBlock"></div>  
            <ng-template #View1>  
                <img src="assets/view1.jpg" width="300" height="400">  
            </ng-template>  
            <ng-template #View2>  
                <img src="assets/view2.jpg" width="300" height="400">  
            </ng-template>  
            <ng-template #View3>  
                <img src="assets/view3.jpg" width="300" height="400">  
            </ng-template>  
            <ng-template #View4>  
                <img src="assets/view4.jpg" width="300" height="400">  
            </ng-template>  
            <ng-template #View5>  
                <img src="assets/view5.jpg" width="300" height="400">  
            </ng-template>  
            <ng-template #View6>  
                <img src="assets/view6.jpg" width="300" height="400">  
            </ng-template>  
        </div>  
        <div class="col-6">  
            <div>  
                <h3>realme C30 (Lake Blue, 32 GB)  (2 GB RAM)</h3>  
            </div>  
            <div>  
                <span class="bg-success text-white p-1 rounded rounded-3"><span class="bi bi-star-fill text-white"></span> 4.3 </span> <span class="ms-4">74,857 Ratings & 3,981 Reviews</span>  
            </div>  
            <div class="mt-3">  
                <span class="text-success"><b>Extra ₹2750 off</b></span>  
            </div>  
            <div class="mt-2">  
                <h3>₹5,749 </h3> <span><del>₹8,499</del></span> <b><span class="ms-3">32% Off</span></b>  
            </div>  
            <div class="mt-3">  
                <ul class="list-unstyled">  
                    <li> <span class="bi bi-tag-fill text-success"></span> Bank Offer5% Cashback on Flipkart Axis Bank CardT&C </li>  
                    <li> <span class="bi bi-tag-fill text-success"></span> Bank Offer5% Cashback on Flipkart Axis Bank CardT&C </li>  
                    <li> <span class="bi bi-tag-fill text-success"></span> Bank Offer5% Cashback on Flipkart Axis Bank CardT&C </li>  
                    <li> <span class="bi bi-tag-fill text-success"></span> Bank Offer5% Cashback on Flipkart Axis Bank CardT&C </li>  
                    <li> <span class="bi bi-tag-fill text-success"></span> Bank Offer5% Cashback on Flipkart Axis Bank CardT&C </li>  
                </ul>  
            </div>  
     </div>  
</div>  
  
mobile-details.component.css  
  
nav div {  
    border:2px solid blue;  
    margin-bottom: 20px;  
    padding: 5px;  
    width: 65px;  
    height: 65px;  
}  
nav div: hover {  
    cursor: grab;**

**1. NgSwitch:**  
**- Switch is a selector that selects exactly the template that matches your**

**condition and renders into UI.  
  
Syntax:  
        <div  [ngSwitch]="value | expression">  
                <div \*ngSwitchCase=" 'case1' "> </div>  
                <div \*ngSwitchCase=" 'case2' "> </div>  
                <div \*ngSwitchDefault>  </div>  
        </div>  
  
Ex:  
product-details.component.ts  
  
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-product-details',  
  templateUrl: './product-details.component.html',  
  styleUrls: ['./product-details.component.css']  
})  
export class ProductDetailsComponent {  
      public ViewName:string = 'BasicDetails';  
  
      public ViewClick(viewname:string){  
         this.ViewName = viewname;  
      }  
}  
  
product-details.component.html  
  
<div class="container-fluid">  
    <div class="row mt-4">  
        <div class="col-2">  
            <div class="btn-group-vertical">  
                <button (click)="ViewClick('BasicDetails')" class="btn btn-primary w-100 mb-2">Basic Details</button>  
                <button (click)="ViewClick('Preview')" class="btn btn-primary w-100 mb-2">Preview</button>  
                <button (click)="ViewClick('Offers')" class="btn btn-primary w-100">Offers</button>  
            </div>  
            <div class="mt-4">  
                <select [(ngModel)]="ViewName" class="form-select">  
                    <option value="Info">Select Info</option>  
                    <option value="BasicDetails">Basic Details</option>  
                    <option value="Preview">Preview</option>  
                    <option value="Offers">Offers</option>  
                </select>  
            </div>  
        </div>  
        <div class="col-10">  
             <div [ngSwitch]="ViewName">  
                 <div \*ngSwitchCase="'BasicDetails'">  
                    <h4>realme C30 (Lake Blue, 32 GB)  (2 GB RAM)</h4>  
                    <span class="bg-success text-white p-2 rounded rounded-2"><span class="bi bi-star-fill text-white"></span> 4.3 </span> <span style="color:gray" class="ms-4">75,240 Ratings & 4,006 Reviews</span>  
                    <div class="mt-3">  
                        <h2>₹5,999</h2>  
                    </div>  
                 </div>  
  
                 <div \*ngSwitchCase="'Preview'">  
                    <img src="assets/view1.jpg" width="300" height="400">  
                 </div>  
  
                 <div \*ngSwitchCase="'Offers'">  
                    <h3>Available Offers</h3>  
                    <ul class="list-unstyled">  
                        <li class="bi bi-tag-fill text-success"> Bank Offer5% Cashback on Flipkart Axis Bank CardT&C </li>  
                        <li class="bi bi-tag-fill text-success"> Special PriceGet extra ₹2500 off (price inclusive of cashback/coupon)T&C</li>  
                        <li class="bi bi-tag-fill text-success">Partner OfferSign up for Flipkart Pay Later and get Flipkart Gift Card worth up to ₹750\*</li>  
                        <li class="bi bi-tag-fill text-success">Partner OfferPurchase now & get a surprise cashback coupon in February / March 2023</li>  
                    </ul>  
                 </div>  
                 <div \*ngSwitchDefault>  
                     <h4>Please select a View</h4>  
                 </div> </div>  
        </div> </div> </div>**

**1. NgFor:  
- It is a structural directive used to repeat HTML elements. It is a repeater.  
- It uses an iteration pattern to read every element form a collection and**

**repeat markup for every element. for..in    : To read all properties for..of    : To read all values- NgFor uses**

**- ngFor supports "of" operator. It does not support for..in iterator.  
  
Syntax:  
            <li  \*ngFor="var item of collection">**

**Repeated item cannot be shadowing so we cannot use ‘let’ instead of ‘var’.  
  
Ex-1:  
 for-demo.component.ts  
  
    import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-for-demo',  
  templateUrl: './for-demo.component.html',  
  styleUrls: ['./for-demo.component.css']  
})  
export class ForDemoComponent {  
     public Categories: string[] = ["All", "Electronics", "Footwear", "Fashion"];  
}  
  
for-demo.component.html  
  
<div class="container-fluid">  
  <div class="mt-4 row">  
    <div class="col">  
        <ol>  
            <li \*ngFor="let item of Categories">{{item}}</li>  
        </ol>  
    </div>  
    <div class="col">  
        <select>  
            <option \*ngFor="let item of Categories">{{item}}</option>  
        </select>  
    </div>  
    <div class="col">  
        <ul class="list-unstyled">  
            <li \*ngFor="let item of Categories"><input type="checkbox">{{item}}</li>  
        </ul>  
    </div>  
    <div class="col">  
        <div class="btn-group-vertical">  
            <button class="btn btn-primary mb-2" \*ngFor="let item of Categories">  
                {{item}}  
            </button>  
        </div>  
    </div>  
    <div class="col">  
        <div \*ngFor="let item of Categories">  
            <a href="#">{{item}}</a>  
        </div>  
    </div>  
    <div class="col">  
        <table class="table table-hover table-dark">  
            <thead>  
                <tr>  
                    <th>Categories</th>  
                </tr>  
            </thead>  
            <tbody>  
                <tr \*ngFor="let item of Categories">  
                    <td>{{item}}</td>  
                </tr>  
            </tbody>  
        </table>  
    </div>  
  </div>  
</div>  
  
Ex-2:  
for-demo.component.ts  
  
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-for-demo',  
  templateUrl: './for-demo.component.html',  
  styleUrls: ['./for-demo.component.css']  
})  
export class ForDemoComponent {  
    public Menu:{Category: string, Products: string[]}[] = [  
       {  
         Category: "Electronics",  
         Products: ["Televisions", "Mobiles"]  
       },  
       {  
         Category: "Footwear",  
         Products: ["Sneakers", "Boots", "Casuals"]  
       }  
    ];  
    public Products:{Name: string, Price: number}[] = [  
        {Name: "Samsung TV", Price: 45000.44},  
        {Name: "Mobile", Price: 20000.44},  
        {Name: "Nike Casuals", Price: 5300.34}  
    ];  
}  
  
for-demo.component.html  
  
<div class="container-fluid">  
  <div class="mt-4">  
    <table class="table table-hover">  
        <thead>  
            <tr>  
                <th>Name</th>  
                <th>Price</th>  
            </tr>  
        </thead>  
        <tbody>  
            <tr \*ngFor="let item of Products">  
                 <td>{{item.Name}}</td>  
                 <td>{{item.Price}}</td>  
            </tr>  
        </tbody>  
    </table>  
  </div>**

**<div class="mt-4 row">  
     <div class="col">  
        <ol>  
            <li \*ngFor="let item of Menu">  
                {{item.Category}}  
                <ul>  
                    <li \*ngFor="let product of item.Products">  
                        {{product}}  
                    </li>  
                </ul>  
            </li>  
        </ol>  
     </div>**

**<div class="col">  
          <select>  
              <optgroup \*ngFor="let item of Menu" [label]="item.Category">  
                  <option \*ngFor="let product of item.Products">  
                        {{product}}  
                  </option>  
              </optgroup>**

**</select>  
     </div>**

**<div class="col">  
         <details \*ngFor="let item of Menu">  
                <summary>{{item.Category}}</summary>  
                <ul>  
                    <li \*ngFor="let product of item.Products">  
                        {{product}}  
                    </li>  
                </ul>  
         </details>  
     </div>  
  </div>  
</div>**

**NgFor Properties:**  
  
**Property     Type           Description  
  
index     number       It returns the index number of repeating element.  
  
first        boolean     It returns true if the repeating element is the first**

**element.  
  
last        boolean      It returns true if the repeating element is the last**

**element.**

**even     boolean       It returns true if the repeating element is at even**

**occurance.**

**odd        boolean       It returns true if the repeating element is at odd**

**occurance.**

**trackBy   function   It tracks the changes in iterator and updates**

**elements that changed.  
  
Ex-1:  
for-demo.component.css  
  
.odd {  
    background-color: rgb(250, 193, 193);  
}  
.even {  
    background-color: rgb(224, 254, 179); }**

**for-demo.component.ts  
  
import { Component, OnInit } from '@angular/core';  
import { FakestoreProductContract } from '../../contracts/FakestoreProductContract';  
  
@Component({  
  selector: 'app-for-demo',  
  templateUrl: './for-demo.component.html',  
  styleUrls: ['./for-demo.component.css']  
})  
export class ForDemoComponent implements OnInit {  
    public Products: FakestoreProductContract[] = [];  
    constructor(){}  
    ngOnInit(): void {  
        fetch("**[**http://fakestoreapi.com/products**](http://fakestoreapi.com/products%26quot)**”)  
        .then(response=> response. json())  
        .then(data=> {  
           this. Products = data;  
        })  
    } }  
  
for-demo.component.html  
  
<div class="container-fluid">  
   <div class="mt-4">  
        <table class="table table-hover">  
          <thead>  
            <tr>  
               <th>Preview</th>  
               <th>Index</th>  
               <th>First</th>  
               <th>Last</th>  
               <th>Even</th>  
               <th>Odd</th>  
            </tr>  
          </thead>  
          <tbody>  
            <tr [class. even]="e" [class. odd]="o" \*ngFor="let item of Products; let i=index; let f=first; let l=last; let e=even; let o=odd">  
               <td> <img [src]="item. image" width="100" height="100"> </td>**

**<td> {{i}} </td>  
               <td>{{f}}</td>  
               <td>{{l}}</td>  
               <td>{{e}}</td>  
               <td>{{o}}</td>  
            </tr>  
          </tbody>  
        </table>  
   </div>  
</div>  
  
Ex-2:Track By  
  
for-demo.component.ts  
  
import { Component, OnInit } from '@angular/core';  
import { FakestoreProductContract } from '../../contracts/FakestoreProductContract';  
  
@Component({  
  selector: 'app-for-demo',  
  templateUrl: './for-demo.component.html',  
  styleUrls: ['./for-demo.component.css']  
})  
export class ForDemoComponent implements OnInit {  
    public Products: any[] = [  
      {Name: "TV", Price: 45000.44},  
      {Name: "Mobile", Price:13400.44},  
      {Name: "Shoe", Price: 5000.33}  
    ]  
    constructor(){}  
    ngOnInit(): void {      
    }  
    public UpdateClick(){  
       alert('Update Clicked');  
       this. Products = [  
        {Name: "TV", Price: 45000.44},  
        {Name: "Mobile", Price:13400.44},  
        {Name: "Shoe", Price: 5000.33},  
        {Name: "Watch", Price: 2300.44}  
       ]  
    }  
    public TrackChanges(index: number){  
        return index;  } }**

**for-demo.component.html  
  
<div class="container-fluid">  
   <div class="mt-4">  
      <div class="mb-2">  
         <button (click)="UpdateClick()" class="btn btn-primary">Update</button>  
      </div>  
       <table class="table table-hover">  
          <thead>  
            <tr>  
               <th>Name</th>  
               <th>Price</th>  
            </tr>  
          </thead>  
          <tbody>  
            <tr \*ngFor="let item of Products; trackBy:TrackChanges">  
               <td>{{item. Name}}</td>  
               <td>{{item. Price}}</td>  
            </tr>  
          </tbody>  
       </table>  
   </div>  
</div>  
  
Ex-3: Table api.nasa.gov  
  
for-demo.component.ts  
  
import { Component, OnInit } from '@angular/core';  
import { FakestoreProductContract } from '../../contracts/FakestoreProductContract';  
  
@Component({  
  selector: 'app-for-demo',  
  templateUrl: './for-demo.component.html',  
  styleUrls: ['./for-demo.component.css']  
})  
export class ForDemoComponent implements OnInit {  
    public MarsObject: any = {};  
  
    constructor(){}  
    ngOnInit(): void {  
        fetch(*Mars api link*)  
        .then(response=> response.json())  
        .then(data=>{  
           this.MarsObject = data;  
        })  
    } }**

**for-demo.component.html  
  
<div class="container-fluid">  
   <div class="mt-4">  
       <table class="table table-hover">  
         <thead>  
            <tr>  
               <th>Photo Id</th>  
               <th>Preview</th>  
               <th>Camera</th>  
               <th>Rover</th>  
            </tr>  
         </thead>  
         <tbody>  
            <tr \*ngFor="let item of MarsObject.photos">  
                <td>{{item.id}}</td>  
                <td>  
                  <a [href]="item.img\_src" target="\_blank">  
                     <img width="100" height="100" [src]="item.img\_src">  
                  </a>  
                </td>  
                <td>  
                  {{item.camera.full\_name}}  
                </td>  
                <td>  
                  {{item.rover.name}}  
                </td>  
            </tr>  
         </tbody>  
       </table>  
   </div>  
</div>  
  
Ex-4: Cards  
  
<div class="container-fluid">  
   <div class="mt-4 d-flex flex-wrap">  
      <div \*ngFor="let item of MarsObject.photos" class="card p-2 m-2" style="width:200px">**

**<img [src]="item.img\_src" class="card-img-top" height="180">  
         <div class="card-header">  
            <h3>{{item.id}}</h3>  
         </div>  
         <div class="card-body">  
            <dl>  
               <dt>Camera Name</dt>  
               <dd>{{item.camera.full\_name}}</dd>  
               <dt>Rover Name</dt>  
               <dd>{{item.rover.name}}</dd>  
            </dl>  
         </div>  
      </div>  
   </div>  
</div>**

[www.fakestoreapi.com](http://www.fakestoreapi.com/)  
**[Data for ERP application](FakeStoreRealTime)**  
  
**Request                                    Data  
GET    /products                        It returns an [] of products.  
GET    /products/1                       It returns specific product by id { }  
GET    /products/categories           It returns all categories string[ ]  
GET /products/category/jewelery It returns specific category products[]  
  
shopper-template.component.ts   
  
import { Component, OnInit } from '@angular/core';  
import { FakestoreProductContract } from '../../contracts/FakestoreProductContract';  
  
@Component({  
  selector: 'app-shopper-template',  
  templateUrl: './shopper-template.component.html',  
  styleUrls: ['./shopper-template.component.css']  
})  
export class ShopperTemplateComponent implements OnInit {  
  constructor(){}  
  
  public Categories: string[] = [];  
  public Products: FakestoreProductContract[] = [];  
  public CartItems:FakestoreProductContract[]= [];  
  public CartItemsCount: number = 0;  
  public isCartVisible: boolean =false;  
  
  public LoadCategories():void {  
      fetch('**[**http://fakestoreapi.com/products/categories**](http://fakestoreapi.com/products/categories%26)**’)  
      .then(response=> response. json())  
      .then(data=> {  
        data.unshift("all");  
        this.Categories = data;  
      })  
  }  
  public LoadProducts(url:string):void {  
      fetch(url)  
      .then(response => response. json())  
      .then(data=>{  
         this. Products = data;  
      })  
  }  
  public GetCartItemsCount():void {  
      this.CartItemsCount = this.CartItems.length;  
  }  
  
  ngOnInit(): void {  
    this.LoadCategories();  
    this.LoadProducts(`http://fakestoreapi.com/products`);  
    this.GetCartItemsCount();  
  }  
  public CategoryChanged(categoryName:string):void {  
      if(categoryName=="all"){  
        this.LoadProducts(`http://fakestoreapi.com/products`);  
      } else {  
        this.LoadProducts(`http://fakestoreapi.com/products/category/${categoryName}`);  
      }   
  }  
  
  public AddToCartClick(id: number):void {  
       fetch(`http://fakestoreapi.com/products/${id}`)  
       .then(response => response.json())  
       .then(data=>{  
           this.CartItems.push(data);  
           this.GetCartItemsCount();  
           alert(`${data.title} \nAdded to Cart`);  
       })  
  }  
  public ToggleCart():void {  
      this.isCartVisible = (this.isCartVisible==false)?true:false;  
  }  
  public RemoveClick(index:number):void {  
      var flag = confirm('Are you sure want to delete?');  
      if(flag==true) {  
        this.CartItems.splice(index,1);  
        this.GetCartItemsCount();  
      }  
  }  
}**

**shopper-template.component.html  
  
<div class="container-fluid">  
    <header class="d-flex  p-2  justify-content-between">  
        <div><h2>Shopper.</h2></div>  
        <nav>  
            <button (click)="CategoryChanged('all')" class="btn">Home</button>  
            <button (click)="CategoryChanged('electronics')" class="btn">Electronics</button>  
            <button class="btn">Men's Fashion</button>  
            <button class="btn">Women's Fashion</button>  
            <button (click)="CategoryChanged('jewelery')" class="btn">Jewelery</button>  
        </nav>  
        <div>  
            <button class="btn"><span class="bi bi-search"></span></button>  
            <button class="btn"> <span class="bi bi-heart"></span></button>  
            <button class="btn"><span class="bi bi-person"></span></button>  
            <button (click)="ToggleCart()" class="btn position-relative"><span class="bi bi-cart"></span> <span class="badge rounded rounded-circle bg-danger text-white position-absolute">{{CartItemsCount}}</span> </button>  
        </div>  
    </header>  
    <div class="bg-dark text-white text-center p-2 mt-2">  
        ⚡️ HAPPY HOLIDAY DEALS ON EVERYTHING ⚡️  
    </div>  
    <section class="mt-3 row">  
        <nav class="col-3">  
          <div>  
            <label class="form-label"><b>Select Category</b></label>  
            <div>  
                <select (change)="CategoryChanged(category. value)" ngModel #category="ngModel" class="form-select">  
                    <option \*ngFor="let item of Categories" [value]="item">  
                        {{item | uppercase}}  
                    </option>  
                </select>  
            </div>  
          </div>  
          <div class="mt-2" \*ngIf="isCartVisible">  
            <h4>Your Cart</h4>  
            <table class="table table-hover">  
                <thead>  
                    <tr>  
                        <th>Title</th>  
                        <th>Preview</th>  
                    </tr>  
                </thead>  
                <tbody>  
                    <tr \*ngFor="let item of CartItems; let i = index">  
                        <td>{{item.title}}</td>  
                        <td>  
                            <img [src]="item.image" width="50" height="50">  
                        </td>  
                        <td>  
                            <button (click)="RemoveClick(i)" class="btn btn-danger">  
                                <span class="bi bi-trash-fill"></span>  
                            </button>  
                        </td>  
                    </tr>  
                </tbody>  
            </table>  
          </div>  
        </nav>  
        <main class="col-9 d-flex flex-wrap overflow-auto" style="height: 450px;">  
            <div \*ngFor="let product of Products" class="card p-2 m-2" style="width: 200px;">  
                <img [src]="product.image" class="card-img-top" height="160">  
                <div class="card-header" style="height:140px">  
                    <p>{{product.title}}</p>  
                </div>  
                <div class="card-body">  
                    <dl>  
                        <dt>Price</dt>  
                        <dd>{{product. price}}</dd>  
                        <dt>Rating</dt>  
                        <dd>  
                            <span class="bi bi-star-fill text-success"></span> {{product.rating.rate}} [{{product.rating.count}}]  
                        </dd>  
                    </dl>  
                </div>  
                <div class="card-footer">  
    <button (click)="AddToCartClick(product.id)" class="btn btn-danger w-100">  
                        <span class="bi bi-cart4"></span> Add to Cart  
                    </button>  
                </div>  
            </div>  
        </main>  
    </section>  
</div>**

**iii. Attribute Directive:**

**- Attribute directives are responsible for making the markup more interactive**

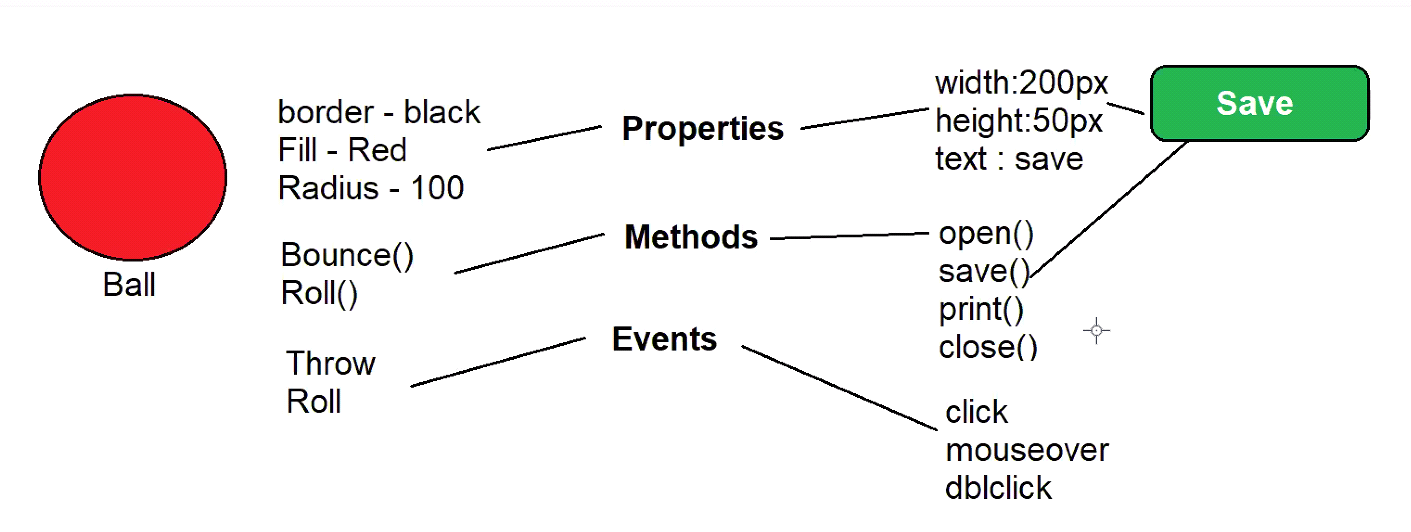
**and responsive.  
- Angular provides 2 types attributes directives  
            a) ngClass  
            b) ngStyle  
  
a)NgClass  
- It is used to apply a CSS class dynamically to any element.  
- So that you can change the appearance of element dynamically.  
  
             <div class="text-danger"> Valid | Invalid </div>  
  
- This is often called as "Class Binding" in Angular.  
- CSS class can be applied to any element using NgClass directive in 3 ways  
  
                a) Using a string reference  
                b) Using an Array reference  
                c) Using an Object reference  
  
Syntax:  
            <div  [ngClass]=" 'className' ">  
            <div  [ngClass]="[ 'class1', 'class2']">  
            <div  [ngClass]="{ 'class1':true, 'class2':false}">  
  
- String reference will allow to set and change one CSS class.  
- Array reference will allow to set or change multiple CSS classes.  
- Object reference will allow to turn ON or OFF the classes using boolean**

**value. [true, false]  
  
Ex:  
 class-binding.component.css  
  
 .bg-style {  
    background-color: yellow;  
}  
.text-style {  
    color:red;  
    text-align: center;  
    padding: 10px;  
}  
.border-style {  
    border:2px solid red;  
}  
  
 class-binding.component.html  
  
<div class="container-fluid">  
    <h2>Class Binding</h2>  
    <ul class="list-unstyled">  
        <li><input ngModel #border="ngModel" type="checkbox">Border</li>  
        <li><input ngModel #background="ngModel" type="checkbox">Background</li>  
        <li><input ngModel #text="ngModel" type="checkbox">Text Effect</li>  
    </ul>  
    <h1 [ngClass]="'bg-style'">Sample Text</h1>  
    <h1 [ngClass]="['bg-style','text-style','border-style']">Sample Text</h1>  
    <h1 [ngClass]="{'bg-style':background.value, 'text-style':text.value, 'border-style':border.value}">Sample Text</h1>  
</div>  
  
Ex:  
class-binding.component.css  
  
.dark-theme {  
    background-color: black;  
    color: white;  
}  
  
class-binding.component.html  
  
<div class="container-fluid d-flex justify-content-center align-items-center" style="height:400px">  
  
   <div>  
    <div class="form-switch">  
        <input type="checkbox" ngModel #darkTheme="ngModel" class="form-check-input"> <label class="form-check-label"> Dark Theme </label>  
     </div>  
     <form class="border border-2 p-2" [ngClass]="{'dark-theme':darkTheme.value}">  
        <h2><span class="bi bi-person-fill"></span>User Login</h2>  
        <dl>  
           <dt>User Name</dt>  
           <dd><input type="text" [ngClass]="{'bg-danger':UserName.invalid}" name="UserName" required minlength="4" ngModel #UserName="ngModel" class="form-control"></dd>  
           <dt>Password</dt>  
           <dd><input type="password" ngModel #Password="ngModel" class="form-control"></dd>  
        </dl>  
      <button class="btn btn-dark w-100">Login</button>  
      </form>  
    </div>  
 </div>  
  
b)NgStyle  
- It defines inline styles for HTML element.  
- Inline styles are individual for any element and are not accessible to other**

**elements.  
- They are faster in rendering as they are configured native to element.  
- However you can't re-use the inline styles.  
- NgStyle can apply and change inline styles dynamically.  
- It uses a style object with style properties and values.  
  
Syntax:  
            <div  [ngStyle]="{attribute:value, attribute:value}">  
  
- Applying inline style dynamically to any element in Angular is known as "Style Binding".  
  
Ex:  
class-binding.component.ts  
  
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-class-binding',  
  templateUrl: './class-binding.component.html',  
  styleUrls: ['./class-binding.component.css']  
})  
export class ClassBindingComponent {**

**public bgStyle:string = 'red';  
      public textStyle:string = 'white';  
      public padding:string = "0px";  
      public fontSize:string = "0px";  
  
      public styleObject:{'background-color':string, 'color':string, 'padding':string, 'font-size':string} = {  
          "background-color" : "red",  
          "color" : "white",  
          "padding": "10px",  
          "font-size": "10px",  
      }  
      public ApplyClick(){  
         this.styleObject = {  
             "background-color": this.bgStyle,  
             "color": this.textStyle,  
            "padding": this.padding + "px",  
            "font-size": this.fontSize + "px"  
         }  
      }  
      public FontChange(){  
        this.styleObject = {  
          "background-color": this.bgStyle,  
          "color": this.textStyle,  
           "padding": this.padding + "px",  
           "font-size": this.fontSize + "px"  
          }  
      }  
}  
  
class-binding.component.html  
  
<div class="container">  
    <h3>Select Styles</h3>  
    <dl>  
        <dt>Background</dt>  
        <dd>  
            <select [(ngModel)]="bgStyle">  
                <option value="red">Red</option>  
                <option value="green">Green</option>  
                <option value="blue">Blue</option>  
                <option value="yellow">Yellow</option>  
            </select>  
        </dd>  
        <dt>Text Color</dt>  
        <dd>  
            <select [(ngModel)]="textStyle">  
                <option value="red">Red</option>  
                <option value="black">Black</option>  
                <option value="white">White</option>  
                <option value="yellow">Yellow</option>  
            </select>  
        </dd>  
        <dt>Padding</dt>  
        <dd>  
            1px <input [(ngModel)]="padding" type="range" min="1" value="1" max="50"> 50px  
        </dd>  
        <dt>Font Size</dt>  
        <dd>  
            10px <input (change)="FontChange()" [(ngModel)]="fontSize" type="range" min="10" max="100" value="10">  
        </dd>  
        <dd><button (click)="ApplyClick()" class="btn btn-primary">Apply</button></dd>  
    </dl>  
    <div class="container-fluid d-flex justify-content-center align-items-center" style="height:400px">  
        <h1 [ngStyle]="styleObject">Welcome to Angular</h1>  
    </div>  
</div>  
  
Summary(Angular Components)  
- Component Directive  
- Structural Directive  
- Attribute Directive  
- Data Binding  
- Style Binding  
- Class Binding  
  
 8. Event Binding**  
  
**- Event is a message sent by sender to its subscriber in order to notify the**

**change.  
  
- Event uses a delegate mechanism [function pointer].  
  
                function  InsertClick()        => Subscriber  
                {  
                }  
                <button onclick="InsertClick()">      => Sender**

 **- Subscriber defines actions to perform.  
- Sender notifies the changes. It specifies when the actions need to perform.  
- Event uses a software design pattern called "Observer", which is a**

**communication pattern under "Behavioural Patterns".  
- Angular uses all JavaScript browser events.  
- Angular Events are configure by "EventEmitter" class of "@angular/core"**

**library.  
  
    Angular EventEmitter        JavaScript Browser Event  
    click                                     onclick  
    change                                 onchange  
  
FAQ: Why angular can't use JavaScript events directly?  
Ans : Angular is "de-coupled" from  DOM  
  
- Angular events are categorized into various groups  
  
1. Mouse Events 5. Cliboard Events  
      mouseover cut  
      mouseout copy   
      mousedown paste  
      mouseup  
      mousemove  
  
2. Keyboard Events 6.  Touch Events**

**keyup Touchstart  
      keydown Touchend  
      keypress Touchmove  
  
3. Button Events 7.  Form Events  
      click submit  
      dblclick ngSubmit  
      contextmenu reset  
   
4. Element State Events 8.  Timer Events  
      blur setTimeout  
      focus     clearTimeout  
      change setInterval  
      selectstart clearInterval  
  
- Angular events are binded to any HTML element by using "( )".  
  
Syntax:  
            <select (change)="Method()">  
            <button (click)="Method()">  
  
Ex:  
            <buton  (click)="InsertClick()">  
  
             (click)                           => Event  
             (click)="InsertClick()"            => Event Handler  
  
- Every event handler can have one default argument "$event".  
  
            JavaScript Event Args  
   
            onclick="InsertClick(event, this)"  
                 
            this => sends information about current object  
                      [id, value, name, className etc..]  
  
            event => sends information about current event  
                          [clientX, clientY, keyCode, charCode, which...]  
  
- Angular allows only one default argument "$event", which send information**

**about both object and event.  
  
            public InsertClick(e)  
            {  
            }  
  
            <button (click)="InsertClick($event)">  
  
- You can access object related information by using "e.target"  
            e.target.id  
            e.target.name  
            e.target.className  
  
- You can access event related information by using "e"  
            e.clientX  
            e.clientY  
            e.ctrlKey  
            e.keyCode etc..  
  
Ex:  
event-binding.component.ts  
  
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-event-binding',  
  templateUrl: './event-binding.component.html',  
  styleUrls: ['./event-binding.component.css']  
})  
export class EventBindingComponent {  
    public InsertClick(e:any):void {  
       document.write(`  
         Button Id    : ${e.target.id} <br>  
         Button Name  : ${e.target.name} <br>  
         Button Class : ${e.target.className} <br>  
         X Position   : ${e.clientX} <br>  
         Ctrl Key     : ${e.ctrlKey}  
       `);  
    }  
}  
  
event-binding.component.html  
  
<div class="container-fluid">  
    <h2>Event Binding</h2>  
    <button (click)="InsertClick($event)" id="btnInsert" name="InsertButton" class="btn btn-primary">Insert</button>  
</div>  
  
  
- Angular and JavaScript allows custom event arguments, you can define**

**custom args individually or along with default argument.  
  
        <button (click)="InsertClick('Record Inserted')">  
        <button (click)="InsertClick(['TV', 'Mobile'])">  
        <button (click)="InsertClick({id:1, name="TV"})>  
        <button (click)="InsertClick(1, "TV", true)">  
        <button (click)="InsertClick($event, "Record Inserted")">  
  
  
1.Mouse Events  
  
Ex:  
event-binding.component.ts  
  
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-event-binding',  
  templateUrl: './event-binding.component.html',  
  styleUrls: ['./event-binding.component.css']  
})  
export class EventBindingComponent {  
    public img\_src:string = "assets/shoe.jpg";  
  
    public styleObject:{position:string, left:string, top:string} = {  
       position: '',  
       left: '',  
       top: ''  
    };  
  
    public SetColor(colorName:string){  
         switch(colorName){  
            case "black":  
             this.img\_src = "assets/blackshoe.jpg";  
             break;  
             case "green":  
             this.img\_src = "assets/greenshoe.jpg";  
             break;  
             case "blue":  
             this.img\_src = "assets/blueshoe.jpg";  
             break;  
         }  
    }  
    public MouseMove(e:any){  
        this.styleObject = {  
           position : 'fixed',  
           left: e.clientX + 'px',  
           top: e.clientY + 'px'  
        }  
    }  
}  
event-binding.component.html  
  
<div class="container-fluid" (mousemove)="MouseMove($event)">  
    <h2>Shoe</h2>  
    <img width="200" [src]="img\_src" height="100">  
    <div class="mt-2">  
        <div class="btn-group">  
            <button (mouseover)="SetColor('black')" name="black" class="text-white" style="background-color:black">Black</button>  
            <button (mouseover)="SetColor('green')" name="green" class="text-white" style="background-color:green">Green</button>  
            <button (mouseover)="SetColor('blue')" name="blue" class="text-white" style="background-color:blue">Blue</button>  
        </div>  
    </div>  
    <div style="height:1000px"></div>  
    <img src="assets/flag.gif" [ngStyle]="styleObject" width="50" height="50">  
</div>**

**1. What is Event?  
2. Which design pattern it follow?  
3. Event Handler  
4. Delegate  
5. EventEmitter  
6. Event Args - $event  
7. Custom Args  
8. Various Categories of Events  
  
2.Keyboard Events  
keyup, keydown, keypress  
                
- If you are handling any logic with regard to key or char then use the event "keyup and keydown".  
  
- If you are handling any logic with regard to key or char codes then use "keypress".   
                        A = 65   Z = 90  
- Keyup and Keydown can't return the keycode or charcode, usually they are returned only with keypress.  
  
Ex:  
event-binding.component.ts  
  
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-event-binding',  
  templateUrl: './event-binding.component.html',  
  styleUrls: ['./event-binding.component.css']  
})  
export class EventBindingComponent {  
   public users:any[] = [  
    {UserName:'john'},  
    {UserName:'john12'},  
    {UserName:'john\_nit'},  
    {UserName:'david'}  
   ];**

**public UserError:string = '';  
   public isInValid:boolean = false;  
   public isPwdWarnVisible = false;  
  
   public VerifyUser(e:any){  
        for(var user of this.users)  
        {  
            if(user.UserName==e.target.value){  
                this.UserError = 'User Name Taken - Try Another';  
                this.isInValid = true;  
                break;  
            } else {  
                this.UserError = 'User Name Available';  
                this.isInValid = false;  
            }  
        }  
   }  
   public VerifyPassword(e:any){  
         if(e.keyCode>=65 && e.keyCode<=90){  
             this.isPwdWarnVisible = true;  
         } else {  
            this.isPwdWarnVisible = false;  
         }  
   }  
}  
event-binding.component.html  
  
<div class="container-fluid">  
   <h2>Regiser User</h2>  
   <dl>  
     <dt>User Name</dt>  
     <dd><input type="text" (keyup)="VerifyUser($event)"></dd>  
     <dd [ngClass]="{'text-danger':isInValid, 'text-success':true}">{{UserError}}</dd>  
     <dt>Password</dt>  
     <dd><input type="password" (keypress)="VerifyPassword($event)"></dd>  
     <dd class="text-warning" \*ngIf="isPwdWarnVisible">  
         <span class="bi bi-exclamation-triangle-fill"></span> Caps ON  
     </dd>  
   </dl>  
</div>  
  
3.Element State Events  
focus, blur, change, selectstart  
  
Ex:  
 event-binding.component.ts  
  
 import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-event-binding',  
  templateUrl: './event-binding.component.html',  
  styleUrls: ['./event-binding.component.css']  
})  
export class EventBindingComponent {  
   public users:any[] = [  
    {UserName:'john'},  
    {UserName:'john12'},  
    {UserName:'john\_nit'},  
    {UserName:'david'}  
   ];  
   public UserError:string = '';  
   public isInValid:boolean = false;  
   public isPwdWarnVisible = false;  
   public PwdError:string = '';  
  
   public VerifyUser(e:any){  
        for(var user of this.users)  
        {  
            if(user.UserName==e.target.value){  
                this.UserError = 'User Name Taken - Try Another';  
                this.isInValid = true;  
                break;  
            } else {  
                this.UserError = 'User Name Available';  
                this.isInValid = false;  
            }  
        }  
   }  
   public VerifyPassword(e:any){  
         if(e.keyCode>=65 && e.keyCode<=90){  
             this.isPwdWarnVisible = true;  
         } else {  
            this.isPwdWarnVisible = false;  
         }  
   }  
   public VerifyRequired(e:any){  
       if(e.target.value=="") {  
          this.UserError = "User Name Required";  
          this.isInValid = true;  
       } else {  
          this.UserError = "";  
       }  
   }  
   public PasswordFocus(){  
       this.PwdError = 'Caps Not Allowed';  
   }  
   public PasswordBlur(){  
    this.PwdError = '';  
   }  
}  
event-binding.component.html  
  
<div class="container-fluid">  
   <h2>Regiser User</h2>  
   <dl>  
     <dt>User Name</dt>  
     <dd><input type="text" (blur)="VerifyRequired($event)" (keyup)="VerifyUser($event)"></dd>  
     <dd [ngClass]="{'text-danger':isInValid, 'text-success':true}">{{UserError}}</dd>  
     <dt>Password</dt>  
     <dd><input type="password" (focus)="PasswordFocus()" (blur)="PasswordBlur()" (keypress)="VerifyPassword($event)"></dd>  
     <dd class="text-warning" \*ngIf="isPwdWarnVisible">  
         <span class="bi bi-exclamation-triangle-fill"></span> Caps ON  
     </dd>  
     <dd>{{PwdError}}</dd>  
   </dl>  
</div>  
  
Ex:  
emi-component.ts  
  
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-emi',  
  templateUrl: './emi.component.html',  
  styleUrls: ['./emi.component.css']  
})  
export class EmiComponent {  
    public P:number = 10000;  
    public r:number = 10.25;  
    public n:number = 1;  
    public EMI:number = 0;  
  
    public AmountChange(e:any) {  
       this.P = e.target.value;  
    }  
    public YearChange(e:any){  
      this.n = e.target.value;  
    }  
    public RateChange(e:any){  
      this.r = e.target.value;  
    }  
    public CalculateClick(){  
        var n = this.n \* 12;  
        var r = this.r / 12 / 100;  
        this.EMI = this.P \* r \* Math.pow(1+r, n) / Math.pow(1+r,n) - 1;  
    }  
}  
  
  
emi-component.html  
  
<div class="container-fluid">  
    <h2 class="text-center">Personal Loan EMI Calculator</h2>  
    <div class="border border-2 border-dark p-3">  
        <div class="row">  
            <div class="col">  
                Amount you need &#8377;<input type="text" [value]="P" size="10">  
            </div>  
            <div class="col">  
                For <input type="text" [value]="n" size="4"> years  
            </div>  
            <div class="col">  
                Interest rate <input [value]="r" type="text" size="4"> %  
            </div>  
        </div>  
        <div class="row mt-4">  
            <div class="col">  
                &#8377; 10,000 <input type="range" (change)="AmountChange($event)" value="10000" min="10000" max="1000000"> &#8377; 10,00,000  
            </div>  
            <div class="col">  
                1 <input type="range" value="1" (change)="YearChange($event)" min="1" max="5"> 5  
            </div>  
            <div class="col">  
                10.25% <input min="10.25" value="10.25" (change)="RateChange($event)" step="0.1" max="18.45" type="range"> 18.45%  
            </div>  
        </div>  
        <div class="row mt-4">  
            <div class="col text-center">  
                <button (click)="CalculateClick()" class="btn btn-primary">Calculate</button>  
            </div>  
        </div>  
    </div>  
    <h1 class="text-center mt-4">  
        Your monthy installed is <span class="text-primary"> {{EMI | currency:'INR'}} </span> for {{n}} years.  
    </h1>  
</div>**

**BMI Calculator:**  
**bmi.component.ts  
  
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-bmi',  
  templateUrl: './bmi.component.html',  
  styleUrls: ['./bmi.component.css']  
})  
export class BmiComponent {  
    public BMIStatus:{'margin-left':string} = {  
        "margin-left" : '550px'  
    }  
}  
bmi.component.html  
  
<div class="container-fluid">  
    <h2>BMI Status</h2>  
    <div class="progress">  
        <div class="progress-bar bg-dark" style="width:400px">  
             < 52 low weight  
        </div>  
        <div class="progress-bar bg-primary" style="width:400px">  
            53 - 72 normal weight  
       </div>  
       <div class="progress-bar bg-dark" style="width:400px">  
         > 73 over weight  
        </div>  
    </div>  
    <div class="bi bi-triangle-fill" [ngStyle]="BMIStatus">  
  
    </div>  
</div>  
  
  
                      Custom Events  
- You can create a custom component.  
- Custom components are designed for resuability.  
- You can reuse in the same way how it is designed or you can customize**

**according to requirements.  
- If you want a component to customize according requirements then you**

**have to configure with custom events.  
  
Challenges:  
1. Accessing a component in another.  
    [Parent and Child]  
  
    Syntax:  
            <app-parent>  
                    <app-child> </app-child>  
            </app-parent>  
  
2. Transporting data from parent to child.  
  
3. Transporting data from child to parent.  
  
Transport data from parent to child:**

**Syntax:   
- Create a property in child component that can accept input from parent.  
  
    import  { Input }  from  '@angular/core';  
  
    @Input()  public  MsgFromParent:string=" ";  
  
    Note: @Input() is a marker or directive which is used to mark any  
              property that is expecting data from parent component.  
  
- Send data into the component using "Property Binding Technique".  
  
    <app-child  [MsgFromParent]=" 'message' "> </app-child>**

**Transport data from child to parent:  
Syntax:   
  
- You have to create a custom event**

**Note: Custom event is to trigger or emit value as output.  
  import { Output, EventEmitter } from '@angular/core';  
  
  @Output()  public  ChildClick:EventEmitter<string> = new**

**EventEmitter<string>();  
  
- Emit the value by using any built-in event  
  
    public  OnButtonClick()  
    {  
         this.ChildClick.emit( 'message' );  
    }  
  
- Child component uses custom event to emit value  
  
    <app-child  (ChildClick)="GetMessageFromChild($event)">  
  
- Parent component have to collect the event arguments and use in its**

**context.  
  
     public  GetMessageFromChild(e:string)  
    {  
         this.messageFromChild = e;  
    }  
  
    <p> {{messageFromChild}} </p>  
  
Ex:  
 child.component.ts  
  
 import { Component, Input, EventEmitter, Output } from '@angular/core';  
  
@Component({  
  selector: 'app-child',  
  templateUrl: './child.component.html',  
  styleUrls: ['./child.component.css']  
})  
export class ChildComponent {  
   @Input() public MsgFromParent:string = '';  
  
   @Output() public ChildClick:EventEmitter<string> = new EventEmitter<string>();  
  
   public ChildButtonClick(){  
        this.ChildClick.emit('Hello ! from child');  
   }  
}  
child.component.html  
  
<div class="container p-4 bg-light text-dark" style="height:200px">  
    <h3>Child Container <button (click)="ChildButtonClick()">Msg to Parent</button> </h3>  
    <p>{{MsgFromParent}}</p><br><br>  
</div>  
  
parent-component.ts  
  
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-parent',  
  templateUrl: './parent.component.html',  
  styleUrls: ['./parent.component.css']  
})  
export class ParentComponent {  
    public msg:string = '';  
    public msgFromChild:string = '';  
    public SendClick(){  
       this.msg = 'Hello ! from Parent';  
    }  
    public GetMessageFromChild(e:string){  
        this.msgFromChild = e;  
    }  
}  
  
parent.component.html  
  
<div class="container-fluid p-4 m-3 bg-dark text-white" style="height:400px">  
    <h2>Parent Container <button (click)="SendClick()">Send Msg to Child</button> </h2>  
    <p>{{msgFromChild}}</p>  
    <app-child [MsgFromParent]="msg" (ChildClick)="GetMessageFromChild($event)" ></app-child>  
</div>**

**Custom Event Ex:**

**filter.component.ts  
  
import { Component, Input, Output, EventEmitter } from '@angular/core';  
  
@Component({  
  selector: 'app-filter',  
  templateUrl: './filter.component.html',  
  styleUrls: ['./filter.component.css']  
})  
export class FilterComponent {  
   @Input() public ElectronicsCount:number = 0;  
   @Input() public JeweleryCount:number = 0;  
   @Input() public MensCount:number = 0;  
   @Input() public WomensCount:number = 0;  
   @Input() public AllCount:number = 0;  
   public CategoryName:string = 'all';  
  
   @Output() public CategoryChanged:EventEmitter<string> = new EventEmitter<string>();  
   @Output() public SendClick:EventEmitter<any> = new EventEmitter<any>();  
  
   public SendCategoryName(e:any) {  
       this.CategoryChanged.emit(e.target.value);  
   }  
   public SendButtonClick(){  
      this.SendClick.emit({'Name':'TV', 'Price':56000});  
   }  
}  
filter.component.html  
  
<h3>Filter Products</h3>  
<ul class="list-unstyled">  
    <li class="d-flex justify-content-between mb-2"><span> <input type="radio" value="all" (change)="SendCategoryName($event)" class="form-check-input" name="category">  All </span> <span class="badge bg dark">{{AllCount}}</span> </li>  
    <li class="d-flex justify-content-between mb-2"><span> <input type="radio" value="electronics" (change)="SendCategoryName($event)" class="form-check-input" name="category">  Electronics</span> <span class="badge bg-dark"> {{ElectronicsCount}}</span> </li>  
    <li class="d-flex justify-content-between mb-2"> <span> <input type="radio" value="jewelery" (change)="SendCategoryName($event)"  class="form-check-input"  name="category"> Jewelery</span> <span class="badge bg-dark">{{JeweleryCount}}</span> </li>  
    <li class="d-flex justify-content-between mb-2"> <span> <input type="radio" value="men's clothing" (change)="SendCategoryName($event)" class="form-check-input"  name="category"> Men's Clothing</span> <span class="badge bg-dark">{{MensCount}}</span> </li>  
    <li class="d-flex justify-content-between mb-2"> <span> <input type="radio" value="women's clothing" (change)="SendCategoryName($event)" class="form-check-input"  name="category"> Women's Clothing</span> <span class="badge bg-dark">{{WomensCount}}</span> </li>  
  
</ul>  
<button (click)="SendButtonClick()">Send</button>  
  
  
products-data.component.ts  
  
import { Component, OnInit } from '@angular/core';  
import { FakestoreProductContract } from '../../contracts/FakestoreProductContract';  
  
@Component({  
  selector: 'app-products-data',  
  templateUrl: './products-data.component.html',  
  styleUrls: ['./products-data.component.css']  
})  
export class ProductsDataComponent implements OnInit {  
    public Products:FakestoreProductContract[] = [];  
    public AllCount:number = 0;  
    public ElectronicsCount:number = 0;  
    public JeweleryCount:number = 0;  
    public MensCount:number = 0;  
    public WomensCount:number = 0;  
  
    ngOnInit(): void {  
       fetch('http://fakestoreapi.com/products’)  
       .then(response=>response.json())  
       .then(data=>{  
        this.Products = data;  
        this.AllCount = data.length;  
        this.ElectronicsCount = data.filter((product:any)=> product.category=='electronics').length;  
        this.JeweleryCount = data.filter((product:any)=> product.category=='jewelery').length;  
        this.MensCount = data.filter((product:any)=> product.category=="men's**

**clothing").length;  
        this.WomensCount = data.filter((product:any)=> product.category=="women's clothing").length;  
       });  
    }  
    public GetCategoryName(e:any){   
        if(e=="all") {  
          fetch(`http://fakestoreapi.com/products`)  
          .then(response=> response.json())  
          .then(data=>{  
            this.Products = data;  
          })  
        }  
        else {  
           fetch(`http://fakestoreapi.com/products/category/${e}`)  
          .then(response=> response.json())  
          .then(data=>{  
            this.Products = data;  
          })  
        }  
         
    }  
    public product:any = {};  
    public GetProduct(e:any){  
        this.product = e;  
    }  
}  
products-data.component.html  
  
<div class="container-fluid">  
    <header class="bg-dark text-white p-1 text-center">  
        <h2> <span class="bi bi-cart3"></span> Shopping - Online</h2>  
    </header>  
    <section class="row">  
        <nav class="col-3">  
            <app-filter [AllCount]="AllCount" [ElectronicsCount]="ElectronicsCount" [JeweleryCount]="JeweleryCount" [MensCount]="MensCount" [WomensCount]="WomensCount" (CategoryChanged)="GetCategoryName($event)" (SendClick)="GetProduct($event)" ></app-filter>  
        </nav>  
        <main class="col-9 d-flex flex-wrap overflow-auto" style="height:500px">  
            <pre>  
                {{product.Name}}  
                {{product.Price}}  
            </pre>  
            <div \*ngFor="let item of Products" class="card m-2 p-2" style="width:200px">  
                <img [src]="item.image" class="card-img-top" height="150">  
                <div class="card-header" style="height:150px">  
                    <p>{{item.title}}</p>  
                </div>  
            </div>  
        </main>  
    </section>  
</div>**

**9. Angular Component Life Cycle Hooks**  
  
**Angular Application Flow:  
1. Client makes a request from browser** [**http://localhost:**](http://localhost:4200/login)**4200/login  
  
2. The request to be processed in "HttpServer" by using "HttpHandler" inside**

**"HttpProcessingPipeline".  
  
3. HttpHandler uses a "Routing Engine".  
  
4. Routing Engine gets the URL request and checks with "RouteTable"  
  
5. Route Table comprises of information about the request and component.  
  
6. If requested path is not matching with Route path in Table then it returns**

**"404" - Not Found.  
  
            2xx         Success  
            3xx         Redirection(Navigating from 1 location to another location)  
            4xx         Client Side Error  
            5xx         Server Side Error/problems(Server is not responding, server is**

**taking more time, server is disconnected)  
   
7. If requested path is matching, then the component is created and loaded**

**into page. [index.html]  
  
8. The component in Angular will undergo various phases from start to end,**

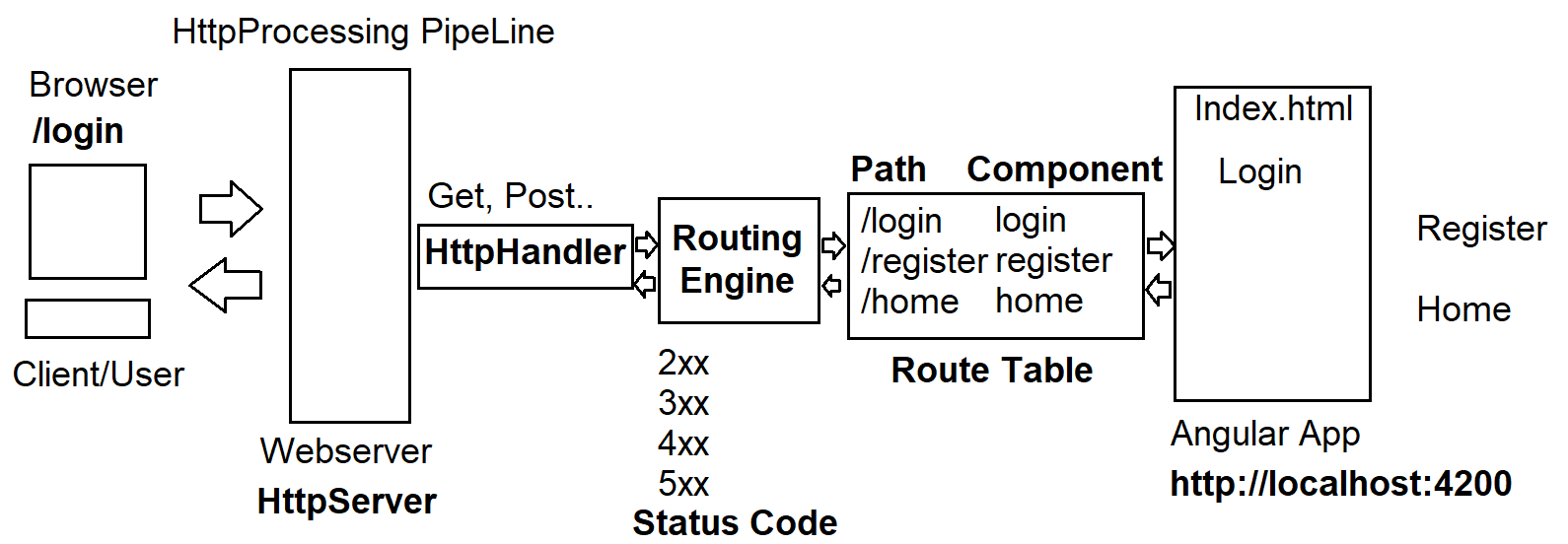
**which are known as "Life Cycle Phases".  
  
9. The lifecycle phases are managed by various methods, which are known as**

**"Life Cycle Hooks".**

**High level Application flow:**

**Component Life Cycle-**

* **It represents the phases of component from the time it is requested to the time it is destroyed.**
* **A component is successfully requested when routing engine returns 200 status code [OK].**
* **A component is destroyed when another component is requested or application ends.**
* **A component have to handle various phases from Instantation to destroying.**
* **These phases are managed by a set of methods, known as "Life Cycle Hooks"**



**Note: The Engine used for processing the request and response with route path is "Ivy". Http is a stateless protocol. Stateless protocol means the http protocol cannot remember the data from one page to another page. once the page is accessed then the data is deleted.  
  
10. The life hooks comprises of following methods,**

|  |  |
| --- | --- |
| **LifeCycle Hook Methods** | **LifeCycle Hook Methods** |
| **ngOnChanges()** | **ngAfterViewInit()** |
| **ngOnInit()** | **ngAfterViewChecked()** |
| **ngDoCheck()** | **ngOnDestroy()** |
| **ngAfterContentChecked()** | **ngAfterContentInit()** |

**1. ngOnChanges()   
- It is the first phase that starts after the component object is created.**

**- It is calles when any @Input() property changed including its first change.**

**i.e If your component is receiving data from a parent, and that data changes.  
- Memory is allocated for references in component.**

**- The memory references for storing various values are initialized.**

**public Price:number;**

**Public Price:number=undefined=45000.45**

**Previous value=undefined**

**Current value=45000.45**

**- The property value is binded to UI**

**<input type=”text” [value]=”Price”>**

**- It defines the changes in value by using default events**

**<input type=”text” [(ngModel)]=”Price”>**

**<input type=”text” (change)=”PriceChange()” [value]=Price>**

**- It updated the new value into memory reference.**

**- It is also about changes in CSSOM[Object Model]**

**- It includes**

**a) Class Binding**

**b) Style Binding**

**Summary:**

**- Initialization - Event Binding - Style Binding**

**- Property Binding - Class Binding**

**- The changes are detected and managed by "SimpleChanges" object, which is**

**a member of "OnChanges".  
  
- SimpleChanges object provides an alogirithm with 2 values  
                PreviousValue == CurrentValue  => No Change Detected  
                PreviousValue != CurrentValue   => Change Detected  
  
Syntax:  
            public Name:string = "john";  
            PreviousValue        Name = undefined  
            CurrentValue        Name = "john"  
            PreviousValue ! = CurrentValue => Change Detected**

**2. ngOnInit()**

**- It runs once, after the component is created, and after Angular sets all the**

**@Input() properties.  
- It allocates memory for current component, which is known as context**

**memory.  
- You can store values in context memory, which are accessible to other**

**components that run within the context of a component.  
- Parent component can send data into child component.  
  
3. ngDoCheck()**

**- Called during every change detection run. Used for custom change detection.  
- The data of child component is not accessible to parent.  
- The context memory of child is not accessible to its parent.  
- It is the method that is executed to transport data explicitly from child**

**component to parent.  
- If any action can't handle implicity then its defined explicilty by using**

**ngDoCheck.  
- Custom events are created by using  
        @Output()  
        EventEmitter  
- Event Args are emitted from child to parent.  
  
4. ngAfterContentInit()**

**- It executes once after something is inserted from parent component to child**

**component using <ng-content>.  
- It initializes the memory for creating layout.  
- It configures the memory for rendering the layout.  
- The output is ready.  
- Implicit templates are used.  
        ngIf, ngFor, ngSwitch  
  
5. ngAfterContentChecked()**

**- It runs everytime that Angular checks the content that was put inside a**

**component using <ng-container>.  
- In this phase the explicit templates are created.  
- The explicit templates are added to page dynamically.  
- It handles "Content Projection".  
- Content Projection is managed by  
        TemplateRef<T>  
        @ViewChild()  
  
6. ngAfterViewInit()  
- It is responsible for handling initial Painiting.**

**- It runs once, after Angular finishes rendering your component’s view (the**

**HTML + child components inside it).**

**i.e Your component has its own HTML and may also include other child**

**components. Angular builds this entire view. Once it’s fully built and visible,**

**it calls ngAfterViewInit().  
  
7. ngAfterViewChecked()  
- It is responsible for generating final output.**

**- It runs every time Angular checks your component’s view (its HTML and**

**child components), to see if anything changed. i.e**

**Angular builds your view (the HTML + child components).**

**Then, every time something might have changed — like user typing, button**

**click, or data update — Angular checks everything again. when it checks**

**your component's view, it runs ngAfterViewChecked().  
- The changes in UI are update in this phase.  
- The final output is ready.  
  
8. ngOnDestroy()  
- It is called when you change from one component to another.  
- The memory allocated for current component is destroyed.  
- It will unsubscribe all the methods.  
- It disconnects all actions.  
- Cleans up the memory allocated for component.  
  
Ex: Change Detection  
  
child.component.ts  
  
import { Component, Input, EventEmitter, Output, OnChanges, SimpleChanges } from '@angular/core';  
  
@Component({  
  selector: 'app-child',  
  templateUrl: './child.component.html',  
  styleUrls: ['./child.component.css']  
})  
export class ChildComponent implements OnChanges {  
    @Input() public UserName:string|null = null;  
    public currentValue:string | null = null;  
    public previousValue:string | null = null;  
    public msg:string|null = null;  
     
    ngOnChanges(changes : SimpleChanges){  
        for(var property in changes)  
        {  
            let change = changes[property];  
            this.currentValue = change.currentValue;  
            this.previousValue = change.previousValue;  
        }  
        if(this.currentValue==this.previousValue) {  
            this.msg = "No Change Detected";  
        } else {  
            this.msg = "Change Detected";  
        }  
      }}  
  
child.component.html  
  
<div class="container p-4 bg-light text-dark" style="height:200px">  
    <h3>Child Component</h3>  
    <p>Hello ! {{UserName}}</p>  
    <h3>{{msg}}</h3>  
    Previous Value: {{(previousValue==undefined)?"Undefined":previousValue}} <br>  
    Current Value : {{currentValue}}  
</div>  
  
parent.component.ts  
  
import { Component, OnChanges, OnDestroy, SimpleChanges } from '@angular/core';  
  
@Component({  
  selector: 'app-parent',  
  templateUrl: './parent.component.html',  
  styleUrls: ['./parent.component.css']  
})  
export class ParentComponent  {  
    public UserName:string|null = "John";     
}  
  
parent.component.html  
  
<div class="container-fluid p-4 m-3 bg-dark text-white" style="height:400px">  
  <h2>Parent Component</h2>  
  <dl>  
    <dt>User Name</dt>  
    <dd><input type="text" [(ngModel)]="UserName"></dd>  
  </dl>  
  <app-child [UserName]="UserName" ></app-child>  
</div>**

**10. Angular MATERIAL**  
 **- Angular Material is a component library.  
- A component library provides several pre-defined components, which you**

**can customize and implement in your project.  
- It saves your development time and improves the performance of**

**application.  
- Angular Material library provides  
        a) CDK  
        b) Components  
  
- Component is a template with design, logic and styles.  
- CDK is Component Development Kit, which provides performance and**

**behaviour for components.**[***https://material.angular.io/***](https://material.angular.io/) **Setup Angular Material for Project:  
  
1. Open Terminal in your project workspace.  
  
2. Run the following command  
  
        >ng add  @angular/material                [installs latest version]  
        >ng add  @angular/material@v14        [install 14 version]  
  
Q: Choose a prebuilt theme name, or "custom" for a custom theme:  
        Indigo-Pink  
   
Q:  Set up global Angular Material typography styles? (y/N)  y  
  
Q:  Include the Angular animations module? Include and Enable  
  
FAQ: What are the changes identified in a project after installing angular material?  
Ans:  
        1. Animation Modules in imported into "app.module.ts"  
        2. angular.json is configure with "css" theme.  
        3. index.html links typography files to page.  
        4. styles.css is included with global styles.  
  
Ex:  
1. Go to "app.module.ts" and import the modules  
  
import { MatFormFieldModule  } from '@angular/material/form-field';  
import {MatInputModule} from '@angular/material/input';  
  
imports : [  
        MatFormFieldModule,  
        MatInputModule  
     ]  
  
2. Add a new component  
        > ng g c material-demo --skip-tests  
  
3. material-demo.component.html  
  
<div class="container-fluid">  
    <h2>Bootstrap Input</h2>  
    <div class="w-25">  
        <label class="form-label">User Name</label>  
        <div>  
            <input type="text" placeholder="Enter Name" class="form-control">  
        </div>  
    </div>  
    <h2>Material Input</h2>  
    <mat-form-field appearance="fill" >  
        <mat-label>User Name</mat-label>  
        <input type="text" ngModel #UserName="ngModel" matInput placeholder="Enter Name">  
    </mat-form-field>  
  
    <h2>Hello ! {{UserName.value}} </h2>  
</div>**

**Ex: Date Picker  
  
app.module.ts  
  
import { MatFormFieldModule  } from '@angular/material/form-field';  
import {MatInputModule} from '@angular/material/input';  
import { MatDatepickerModule } from '@angular/material/datepicker';  
import { MatNativeDateModule } from '@angular/material/core';  
  
  
imports: [  
    BrowserModule,  
    FormsModule,  
    BrowserAnimationsModule,  
    MatFormFieldModule,  
    MatInputModule,  
    MatDatepickerModule,  
    MatNativeDateModule  
  ],  
  
  
material-demo.component.html  
  
<div class="container-fluid">  
    <h2>Date Picker</h2>  
    <mat-form-field appearance="outline">  
        <mat-label>Departure Date</mat-label>  
        <mat-datepicker-toggle matSuffix [for]="departure"></mat-datepicker-toggle>  
        <input type="text" matInput [matDatepicker]="departure">  
        <mat-datepicker #departure></mat-datepicker>  
    </mat-form-field>  
</div>**

**Material CDK  
- CDK is component development kit.  
- It provides performance and behaviour for components.  
  
Ex: Lazy Loading  
  
1. Go to app.module.ts  
  
import { ScrollingModule } from '@angular/cdk/scrolling';  
  
imports: [  
 ScrollingModule  
]  
  
2. material-demo.component.ts  
  
import { Component, OnInit } from '@angular/core';  
import { FakestoreProductContract } from '../../contracts/FakestoreProductContract';  
  
@Component({  
  selector: 'app-material-demo',  
  templateUrl: './material-demo.component.html',  
  styleUrls: ['./material-demo.component.css']  
})  
export class MaterialDemoComponent implements OnInit {  
    public Products:FakestoreProductContract[] = [];  
    ngOnInit(){  
        fetch('**[**http://fakestoreapi.com/products**](http://fakestoreapi.com/products%26)**')  
        .then(res=>res.json())  
        .then(data=>{  
           this.Products = data;  
        })  
    }  
}  
  
3. material-demo.component.html  
  
<div class="container-fluid">  
   <h2>Without Lazy Loading</h2>  
    <div class="overflow-auto" style="height:300px; border:1px solid black; padding:5px; width:250px">  
      <div \*ngFor="let item of Products">  
         <img [src]="item.image" width="150" height="150">  
      </div>  
    </div>  
   <h2>With Lazy Loading</h2>  
   <cdk-virtual-scroll-viewport itemSize="100" class="overflow-auto" style="height:300px; border:1px solid black; padding:5px; width:250px">  
        <div \*cdkVirtualFor="let item of Products">  
            <img [src]="item.image" width="150" height="150">  
        </div>  
   </cdk-virtual-scroll-viewport>  
</div>**

**11. Angular PIPEs  
- Pipe is used to transform data.  
- It allows to change the format of data and present.  
- Pipe is a class that implements "PipeTransform" base.  
- Pipe class container "transform()" method that can transform and return**

**your data.  
- Pipe transforms data in a user friendly format.  
- Angular provides several bulit-in Pipes and also allows to create Custom**

**pipes.  
- Angular built-in pipes are  
            uppercase number json i18nPlural  
            lowercase currency slice async  
            titlecase date i18nSelect percent etc..**

**- Pipe is attached to your data by using "|" symbol.  
- Pipe can have parameters which are defined using ":"  
  
Syntax:  
            data | pipeName:param1:param2  
  
  
1. uppercase                : Changes all to CAPS  
2. lowercase                : Sets all chars to lowercase  
3. titlecase                    : Sets every first char in word to CAPS.  
  
       public Name:string = "welcome";  
  
      {{ Name | uppercase }}  
  
4. number                    : It is used to display numeric values with  
                                  decimal places. It class is "Decimal"**

**Syntax:  
 {{ value | number: 'minIntegerDigits.minFractionDigits-maxFractionDigits' }}**

**minIntegerDigits: Minimum number of digits before the decimal point**

**(default is 1)**

**minFractionDigits: Minimum number of digits after the decimal point**

**(default is 0)**

**maxFractionDigits: Maximum number of digits after the decimal point**

**(default is 3)**

**public  Price:number = 34000.40;  
  
      {{ Price | number: '0.2' }}  
  
5. currency                   : It is similar to number but can have currency symbol.  
  
    {{Product.Price | currency:'INR'}}   
  
6. date                        : It can display date in various formats.  
  
             short shortDate dd    - day         MMM      - short month   
             long longDate MM  - month MMMM   - long month  
            full fullDate yy     - year yyyy  
                                                                  
Syntax:  
         public  Mfd:Date  = new Date("2023-01-22");  
         
        {{Product.Mfd | date : 'dd-MMM-yyyy'}}**

**7. percent                      : It converts a number in precent  
 - It is an impure pipe.  
 - Impure pipe can change the value.  
 - Pure pipe will not change the value, it just changes**

**the format of value.  
Syntax:  
            public  Rate:number = 0.245;  
             
            {{ Rate | precent: '2.2' }} // 0.245 × 100 = 24.5%**

**Formatting with '2.2' means always 2 digits after decimal   
  
8. json : It converts the data into JSON format.  
 - JavaScript uses a method JSON.stringfy(data)  
Syntax:  
            public Product:{Name:string, Price:number} = {  
                Name : "TV",  
                Price: 45000  
            };  
  
            {{ Product | json }}**

**9. slice                     : It is an array method that reads and returns the values**

**between specified index.  
Syntax:  
        public  Msg:string = "Welcome to JavaScript"  
         
        {{ Msg | slice:0:7 }} // Welcome**

**10. keyvalue : Angular have iterator "ngFor"  
 - ngFor uses "of" operator.  
 - "of" operator can't read properties, it can read only vals.   
Syntax:  
         <li  \*ngFor="let item of  categories | keyvalue">  
                {{ item.key }}  -  {{ item.value }}  
          </li>**

**11. i18Select : It uses an object to verify the matching keys and selects**

**a suitable value to display.  
  
Syntax:  
           public  Products:[] = { }  
           public  object:{}  = { }  
  
           {{ item | i18nSelect : object }}  
  
Ex:  
  pipe-demo.component.ts  
   
  import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-pipe-demo',  
  templateUrl: './pipe-demo.component.html',  
  styleUrls: ['./pipe-demo.component.css']  
})  
export class PipeDemoComponent implements OnInit {  
  
    public Products : {Name:string, ShippedTo:string}[] = [  
       {Name:"Samsung TV", ShippedTo: "Delhi"},  
       {Name:"Mobile", ShippedTo: "Hyd"},  
       {Name:"Watch", ShippedTo: "Goa"},  
       {Name:"Shirt", ShippedTo: "Mumbai"}  
    ];  
  
    public Status : any = {  
         'Delhi' : 'Delivery in 2 Days',  
         'Hyd'   : 'Same day Delivery',  
         'Mumbai': 'One Day Delivery',  
         'other' : 'Usually dispatched in 3-4 working days'  
    };**

**ngOnInit(): void {  
         
    }  
}  
  
pipe-demo.component.html  
  
<div class="container-fluid">  
   <h2>Your Cart</h2>  
   <table class="table table-hover">  
        <thead>  
            <tr>  
                <th>Name</th>  
                <th>City</th>  
                <th>Delivery Status</th>  
            </tr>  
        </thead>  
        <tbody>  
            <tr \*ngFor="let product of Products">  
                <td>{{product.Name}}</td>  
                <td>{{product.ShippedTo}}</td>  
                <td>  
                    {{product.ShippedTo | i18nSelect : Status}}  
                </td>  
            </tr>  
        </tbody>  
   </table>  
</div>  
  
11. i18Plural - In programming traditions developers use "plural"**

**name for a collection of values. [more than one].  
 - Plural pipe can get the length and count of elements**

**in a collection and display suitable messages.  
  
Syntax:  
             public  Map = {  
                    '=0' :  "No Value",  
                    '=1' :  "One value",  
                    'other' : "#  values"  
              }  
            public  Collection:string[] = [];  
  
            {{  Collection.length | i18nPlural : Map }}  
  
Ex:  
 pipe-demo.component.ts  
   
 import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-pipe-demo',  
  templateUrl: './pipe-demo.component.html',  
  styleUrls: ['./pipe-demo.component.css']  
})  
export class PipeDemoComponent implements OnInit {  
    public Messages : string[] = [];  
    public Count : number = 0;  
    public Message : string = '';  
    public ShowMessages : boolean = false;  
  
    public NotificationMap = {  
       '=0' : 'No New Messages',  
       '=1' : 'One New Message',  
       'other': '# Messages'  
    }  
  
    ngOnInit(): void {  
        this.Count = this.Messages.length;  
    }  
    public SendClick(){  
        var now = new Date();  
        this.Messages.push(this.Message + "-" + now.toLocaleTimeString());  
        this.Count = this.Messages.length;  
        alert("Message Sent");  
        this.Message = "";  
    }  
    public ShowClick(){  
       this.ShowMessages = (this.ShowMessages==true)?false:true;  
    }  
}  
  
pipe-demo.component.html  
  
<div class="container-fluid">  
   <h2>Plural Demo</h2>  
   <div class="mb-3 input-group w-25">  
      <input type="text" [(ngModel)]="Message" class="form-control"> <button (click)="SendClick()" class="btn btn-primary"> <span class="bi bi-send"></span> </button>  
   </div>  
   <button (click)="ShowClick()" class="btn btn-primary position-relative w-25">  
      <span class="bi bi-bell"></span> {{Messages.length | i18nPlural:NotificationMap}}  
      <span class="badge bg-dark position-absolute end-0 top-0 rounded rounded-circle"> {{Count}} </span>  
   </button>  
   <div class="mt-4" \*ngIf="ShowMessages">  
       <ul class="list-unstyled">  
           <h3>Your Messages</h3>  
           <li \*ngFor="let item of Messages">  
                {{item}}  
           </li>  
       </ul>  
   </div>  
</div>**

**Custom Pipes  
- Pipe is a class.  
- Pipe class implements  "PipeTransform" base  
- Pipe uses a directive "@Pipe()"  
- Pipe comprises of "transform()" which is used to transform and return the**

**data.  
- Pipe can be parameter less or parameterized.  
  
 *> ng  g   pipe   sentence  --skip-tests*  
1. Add a folder "pipes"  
  
2. Add a new file  
        "sentence.pipe.ts"  
  
import { PipeTransform, Pipe } from '@angular/core';  
  
@Pipe({  
    name: 'sentence'  
})  
export class SentencePipe implements PipeTransform  
{  
     transform(sentence:string) {  
         var firstChar = sentence.charAt(0);  
         var restChars = sentence.substring(1);  
         var sentence = firstChar.toUpperCase() + restChars.toLocaleLowerCase();  
         return sentence;  
     }  
}  
  
3. Register in app.module.ts  
  
 declarations : [  
        SentencePipe  
  ]  
  
 4.  Apply  
  
        public  title:string = "weLCOMe to AngULAr";  
  
        {{ title | sentence }}**

**Sorting Pipe - Custom Pipe with Parameters  
  
> ng  generate  pipe  sorting --skip-tests  
  
Ex:  
import { Pipe, PipeTransform } from '@angular/core';  
  
@Pipe({  
  name: 'sorting'  
})  
export class SortingPipe implements PipeTransform {  
  
  transform(collection:string[], reverse?:boolean) {  
     
       if(reverse==true){  
          collection.sort();  
          collection.reverse();  
          return collection;  
       } else {  
           return collection.sort();  
       }    
  }  
}**

**FAQ's  
1. What is a Pipe? What is the role of Pipe in Angular?  
A. Pipe is a class, which transforms the data.  
  
2. What is Parameterized Pipe?  
A. Pipe can be parameter less, which perform same functionality every time.  
    A Parameterized pipe can change the functionality according to situation.  
  
3. What is chaining of Pipes?  
A. It is the process of defining multiple pipes to data.  
      \*ngFor="let item of collection | keyvalue | slice:0:2 | uppercase"  
  
4. What is a Pure and Impure Pipe?  
A. A pure pipe will not change the value.  
           Ex: uppercase, number, slice  
    An Impure pipe can change the value.  
           Ex: percent, i18nSelect, i18nPlural  
  
5. What is the base class for Pipe?  
A. PipeTransform  
  
6. What is the marker for Pipe?  
A. @Pipe()**

**12. Angular SERVICE**

**- Service is a pre-defined bussiness logic, which you can implement and**

**customize according to requirements.  
- You can use a service to extend the functionality of your application without**

**writing the functionality.  
- Service enables  
        a) Reusability  
        b) Maintainability  
        c) Testability  
        d) Security  
        e) Extensibility  
- Services is a set of factories.  
- Factory is a set of values and functions.  
- Factory uses single call mechanism, where an object is created every time to**

**access a function.  
- Service uses a "single ton" mechanism, where object is create for very first**

**request and same object is accross requests.  
- Service uses DI [Dependency Injection].  
- DI comprises of 2 components  
        a) Provider  
        b) Injector   
- Provider is responsible for locating the values or functions from memory.  
- Injector is responsible for Injecting the value into component.  
- Angular Provides several built-in services and also allows to create custom**

**services.  
- Angular Built-in Services  
        HttpClient Service  
        Routing Services  
        Security Services etc...  
  
Ex:  
1. Add a new folder in app folder  
        "services"  
  
2. Generate new service  
         
        > ng  g service  captcha  
  
import { Injectable } from '@angular/core';  
  
@Injectable({  
  providedIn : "root"  
})  
export class CaptchaService {  
  
  public GenerateCode():string {  
      var a = Math.random() \* 10;  
      var b = Math.random() \* 10;  
      var c = Math.random() \* 10;  
      var d = Math.random() \* 10;  
      var e = Math.random() \* 10;  
      var f = Math.random() \* 10;  
      var code = `${Math.round(a)} ${Math.round(b)} ${Math.round(c)} ${Math.round(d)} ${Math.round(e)} ${Math.round(f)}`;  
      return code;  
  }  
  constructor() { }  
}  
  
4. login.component.ts  
  
import { Component, OnInit } from "@angular/core";  
import { CaptchaService } from "../../services/captcha.service";  
  
@Component({  
    selector: 'app-login',  
    templateUrl: './login.component.html',  
    styleUrls: ['./login.component.css']  
})  
export class LoginComponent implements OnInit  
{  
    public code:string = '';  
    constructor(private captcha: CaptchaService){  
  
    }  
    ngOnInit(): void {  
        this.code = this.captcha.GenerateCode();  
    }  
    public NewCode():void {  
        this.code = this.captcha.GenerateCode();  
    }  
}  
  
5. login.component.html  
  
<div class="container-fluid">  
    <div class="login-form">  
        <form>  
            <h2> <span class="bi bi-person-fill"></span> User Login</h2>  
            <dl>  
                <dt>User Name</dt>  
                <dd><input type="text" class="form-control"></dd>  
                <dt>Password</dt>  
                <dd><input type="password" class="form-control"></dd>  
                <dt>Verify Code <button class="btn" (click)="NewCode()"> <span class="bi bi-arrow-clockwise"></span> </button> </dt>  
                <dd>{{code}}</dd>  
            </dl>  
            <button class="btn btn-primary w-100">Login</button>  
        </form>  
    </div>  
</div>**

**FAQ: What are the various provider locations for service?  
Ans:  
        a) root : refers to current application level.  
        b) platform : refers to current module level.  
        c) any : refers to workspace level  
  
FAQ: Where to configure the provider?  
Ans :  
        a) At service level  
        b) At application module level  
  
Syntax: Service Level  
  
        @Injectable({  
             providedIn : "root" | "any" | "platform"  
        })  
  
Syntax:  app.module.ts  
  
        providers: [ ServiceName ]  
  
FAQ:  What is difference between provider configured at service level and at module level?  
Ans :  Module Level is "Eager Loading"  
        Service Level is "Lazy Loading"  
  
FAQ: What is difference between a Factory and Service?  
Ans  : Factory uses "Single Call" where an object is created every time when function is requested where as Service  uses "Single Ton" pattern where object is created for first request and the same object is used accross requests.  
  
FAQ: Why to inject a service reference in constructor and not to create an object for service class?  
Ans : To Implement "single ton".  
        constructor(private ref : ServiceName) {  }  
  
FAQ: Why to define an access modifier for parameter in constructor method?  
Ans : To configure the parameter at class level, so that it is accessible to all methods in class.  
  
  
                              HttpClient Service  
                                
- It provides a set of functions that are responsibile for handling interactions**

**with API. We can call it as a class or service.  
  
FAQ: What is difference between fetch and HttpClient?  
Ans:  
          FETCH                             HTTPCLIENT  
           
         Returns data in Binary            Returns in JSON  
  
         Parsing of data is required      No parsing required  
  
         Fetch is not safe                    HttpClient is secured  
  
         Poor in errors                        Good error handling features  
  
         It is in Sync                         It uses Async [RxJS]  
  
Step-1: Import the service from library  
 import  { HttpClient }  from  "@angular/common/http"; [component]  
 import  { HttpClientModule}  from [appModule]**

**"@angular/common/http";                  
  
Step-2: Inject the library into any component [service library]  
           constructor(private http : HttpClient) {  }  
  
Step-3:  Access by using "http" methods  
            this.http.get()  
            this.http.post()**

**RxJS  
- Reactive Extentions Library for JavaScript.  
- It is used to create "asynchronous" events using "Observables".  
- "Asyncrhonous" uses  "un-blocking" technique, which executes the events**

**without blocking other requests.  
- RxJS provides various components  
        a) Observable  
        b) Subscriber  
  
- Observable configures asynchronous event.  
- Subscriber executes the asynchronous event.  
  
Ex:  
1. Go to "app.module.ts"  
  
import { HttpClientModule } from '@angular/common/http';  
  
imports : [  
             HttpClientModule  
            ]  
  
2. Go to services folder and add a new service  
  
    "fakestore.service.ts"  
  
import { Injectable } from '@angular/core';  
import { HttpClient } from '@angular/common/http';  
import { Observable } from 'rxjs';  
import { FakestoreProductContract } from '../contracts/FakestoreProductContract';  
  
@Injectable({  
    providedIn: "root"  
})  
export class FakestoreService  
{  
    constructor(private http: HttpClient){  
  
    }  
    public GetProducts():Observable<FakestoreProductContract[]>{  
        return this.http.get<FakestoreProductContract[]>("**[**http://fakestoreapi.com/products"**](http://fakestoreapi.com/products%26quot)**);  
    }  
  
    public GetCategories():Observable<string[]>{  
        return this.http.get<string[]>("**[**http://fakestoreapi.com/products/categories**](http://fakestoreapi.com/products/categories%26quot)**")  
    }  
}  
  
3. Add a new component  
         
"service-demo.component.ts"  
  
import { Component, OnInit } from '@angular/core';  
import { FakestoreProductContract } from '../../contracts/FakestoreProductContract';  
import { FakestoreService } from '../../services/fakestore.service';  
  
@Component({  
  selector: 'app-service-demo',  
  templateUrl: './service-demo.component.html',  
  styleUrls: ['./service-demo.component.css']  
})  
export class ServiceDemoComponent implements OnInit {  
    public Categories:string[] = [];  
    public Products:FakestoreProductContract[] = [];  
  
    constructor(private fakestore: FakestoreService){  
  
    }  
    ngOnInit(): void {  
      this.fakestore.GetCategories().subscribe(data => this.Categories = data );  
      this.fakestore.GetProducts().subscribe(data=> this.Products = data);  
    }  
}  
  
Service-demo.component.html  
  
<div class="container-fluid">  
    <h2 class="text-center">Fakestore API</h2>  
    <select>  
        <option \*ngFor="let category of Categories">{{category}}</option>  
    </select>  
    <hr>  
    <table class="table table-hover">  
        <thead>  
            <tr>  
                <th>Title</th>  
                <th>Preview</th>  
            </tr>  
        </thead>  
        <tbody>  
            <tr \*ngFor="let item of Products">  
                <td>{{item.title}}</td>  
                <td>  
                    <img [src]="item.image" width="100" height="100">  
                </td>  
            </tr>  
        </tbody>  
    </table>  
</div>**

**13. Angular FORMS  
- Angular Forms are classified into 2 types  
        a) Template Driven Forms  
        b) Model Driven Forms or Reactive Forms  
  
 Template Driven Form  
- Forms are configured and handled at View level. [ in UI ]  
- It uses MVVM [Model-View  View-Model].  
- The library required for template driven form is defined in "FormsModule"**

**of "@angular/forms".  
- The <form> element is defined by using "ngForm".  
- The other input and form elements are defined by using "ngModel".  
  
Syntax:  
        <form  #frmRegister="ngForm">  
            <input type="text" ngModel #userName="ngModel">  
            <select ngModel  #City="ngModel"> </select>  
        </form>  
  
        frmRegister.value       object { key : value }  
                                        Element Name is Key  
                                        Element Value is its value  
- It reduces the number of requests.  
- It improves the load time.  
- Separation, Reusability and Extensibility issues.  
- UI is heavy and slow in rendering.  
  
Note: Always use template forms when you want a form with stable functionality and doesn't require regular extentions.  
  
Ex:  
template-form.component.ts  
  
 import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-template-form',  
  templateUrl: './template-form.component.html',  
  styleUrls: ['./template-form.component.css']  
})  
export class TemplateFormComponent {  
    public SubmitClick(data:any){  
        alert(JSON.stringify(data));  
    }  
}  
   
template-form.component.html  
  
<div class="container-fluid">  
  <form #frmRegister="ngForm" (submit)="SubmitClick(frmRegister.value)">  
    <h2>Register User</h2>  
    <dl>  
        <dt>User Name</dt>  
        <dd><input type="text" name="UserName" ngModel #UserName="ngModel"></dd>  
        <dt>Age</dt>  
        <dd><input type="number" name="Age" ngModel #Age="ngModel"></dd>  
        <dt>City</dt>  
        <dd>  
         <select name="City" ngModel #City="ngModel">  
            <option>Delhi</option>  
            <option>Hyd</option>  
        </select>  
        </dd>  
    </dl>  
    <button>Register</button>  
  </form>  
</div>  
  
        frmRegister.value                             { }  
        frmRegister.value.UserName         only user name " "  
         
- You can directly access any specific element value  
         UserName.value  
         Age.value  
  
                    Validating Template Driven Form  
- Validation is the process of verifying user input.  
- Validation is required to ensure that contradicationary and unauthorized**

**data is not get stored into database.  
- Validations can be handled  
        a) Client Side  
        b) Server Side  
- Client Side validations in JavaScript and jQuery are handled by using lot of**

**functions. [string functions, math functions, array function etc..]  
- Angular is provided with "Validation Services".  
- These validation service can use HTML validations and verify values without**

**writing much logic.  
- HTML validations are  
        required pattern number  
        minlength min email  
        maxlength max url  
          
- Validation service is buit-in with validation functions.  
- The validation properties of Angular validation service are classified into 2**

**groups,  
                a) Form State Validation  
                b) Input State Validation**

**Form State Validation Services:  
- Form State validation services are used to verify all fields in a form.  
- You can access from "ngForm" reference  
  
ServiceName        Property            Type         Description  
  
NgPristine            pristine           boolean     It returns true if no field in the**

**form is modified.  
  
NgDirty                 dirty                 boolean     It returns true if any one field in**

**the form is modified.  
  
NgValid                valid                boolean     It returns true when all fields in**

**the form are valid.  
  
NgInvalid            invalid             boolean   It returns true when any one field**

**in invalid.  
  
NgSubmitted      submitted        boolean     It returns true on form submit.**

**Syntax:  
            <form  #frmRegister="ngForm">  
             frmRegister.valid  
             frmRegister.pristine  
             frmRegister.invalid etc...  
  
Ex:  
 template-form.component.html  
  
<div class="container-fluid">  
  <form #frmRegister="ngForm" class="p-2" [ngClass]="{'valid-style':frmRegister.valid, 'invalid-style':frmRegister.invalid}" (submit)="SubmitClick(frmRegister.value)">  
    <h2>Register User</h2>  
    <dl>  
        <dt>User Name</dt>  
        <dd><input type="text"  required minlength="4" maxlength="10" name="UserName" ngModel #UserName="ngModel"></dd>  
        <dt>Age</dt>  
        <dd><input type="number" required name="Age" ngModel #Age="ngModel"></dd>  
    </dl>  
    <button [disabled]="frmRegister.invalid">Register</button>  
    <button \*ngIf="frmRegister.dirty">Save Draft</button>  
  </form>  
  <hr>  
    Pristine : {{frmRegister.pristine}} <br>  
    Dirty     : {{frmRegister.dirty}} <br>  
    Valid    : {{frmRegister.valid}} <br>  
    invalid  : {{frmRegister.invalid}} <br>  
    Submitted : {{frmRegister.submitted}}  
</div>  
  
template-form.component.css  
  
.invalid-style {  
     background-color: rgb(247, 194, 194);  
}  
.valid-style {  
    background-color: rgb(196, 247, 196);  
}**

**Input State Validation Services:  
- It verifies every field in the form inidividually and reports the error.  
- The input state validation servies are  
  
NgPristing          pristine NgTouched touched  
NgDirty               dirty NgUnTouched untouched  
NgValid               valid NgErrors    errors              
NgInvalid           invalid  
  
Syntax:  
        <input type="text" required  #UserName="ngModel" ngModel>  
  
         UserName.invalid  
         UserName.touched etc  
Ex:  
template-form.component.html  
  
<div class="container-fluid">  
  <form #frmRegister="ngForm" class="p-2">  
    <h2>Register User</h2>  
    <dl>  
        <dt>User Name</dt>  
        <dd><input type="text"  required minlength="4" name="UserName" ngModel #UserName="ngModel"></dd>  
        <dd class="text-danger" \*ngIf="(frmRegister.submitted && UserName.invalid)||(UserName.touched && UserName.invalid)">  
             User Name Required  
        </dd>  
        <dt>Age</dt>  
        <dd><input type="number" required name="Age" ngModel #Age="ngModel"></dd>  
        <dd class="text-danger" \*ngIf="(frmRegister.submitted && Age.invalid)||(Age.touched && Age.invalid)">  
            Age Required  
        </dd>  
    </dl>  
    <button>Register</button>  
  </form>  
</div>**

**Angular Errors Object  
- It can identify specific validation error in input element and report the**

**relative error message.  
- Upto Angular 13 , it is just a JavaScript with validation properities  
                errors.required  
                errors.minlength  
                errors.pattern ...  
- From Angular 14 and in Angular 15, errors object changed into an array of**

**error validations, which is nullable type.  
                errors?.['required']  
                errors?.['pattern']  
                errors?.['minlength'] ...  
  
- Several HTML elements can't use HTML5 validations, then you have to**

**configure validations using custom functions. [Custom Validations]  
  
- Custom Validations require  EventBinding and JavaScript functions  
  
          <select (change)="VerifyCity($event)">  
              <option value="-1"> Choose </option>  
              <option value="Delhi"> Delhi </option>  
          </select>  
  
           public  VerifyCity(e:any) {  
                 if(e.target.value=="-1") {  
                      // set error  
                 } else {  
                    // remove error  
                 }  
            }  
  
- Angular also provides pre-defined validation CSS classes.  
            .ng-invalid  
            .ng-valid  
            .ng-pristine  
            .ng-dirty  
  
        form.ng-invalid {  }  
        input.ng-invalid { }  
  
Ex:  
template-form.component.css  
  
input.ng-invalid {  
    border:1px solid red;  
}  
input.ng-valid {  
    border:1px solid green;  
    box-shadow: 2px 2px 2px green;  
}  
form.ng-invalid {  
    background-color: rgb(253, 200, 200);  
}  
  
template-form.component.ts  
  
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-template-form',  
  templateUrl: './template-form.component.html',  
  styleUrls: ['./template-form.component.css']  
})  
export class TemplateFormComponent {  
     public CityError:string = '';  
  
     public CityChanged(e:any){  
           if(e.target.value=="-1") {  
              this.CityError = "Pleas select your city";  
           } else {  
               this.CityError = "";  
           }  
     }  
}  
  
template-form.component.html  
  
<div class="container-fluid">  
   <form class="mt-2 p-4" #frmRegister="ngForm" >  
       <h2>Register User</h2>  
       <dl>  
          <dt>User Name</dt>  
          <dd><input type="text"  required  minlength="4" maxlength="10" name="UserName" ngModel #UserName="ngModel"></dd>  
          <dd class="text-danger" \*ngIf="frmRegister.submitted && UserName.invalid">  
              <div \*ngIf="UserName.errors?.['required']">User Name Required</div>  
              <div \*ngIf="UserName.errors?.['minlength']">User Name too short min 4 chars required</div>  
              <div \*ngIf="UserName.errors?.['maxlength']">User Name too long max 10 chars only</div>  
          </dd>  
          <dt>Password</dt>  
          <dd><input type="password" name="Password" ngModel #Password="ngModel" required pattern="(?=.\*[A-Z])\w{4,10}"></dd>  
          <dd class="text-danger" \*ngIf="(frmRegister.submitted && Password.invalid)||(Password.touched && Password.invalid)">  
              <div \*ngIf="Password.errors?.['required']">Password Required</div>  
              <div \*ngIf="Password.errors?.['pattern']" >Password 4 to 15 chars with atleast 1 uppercase letter.</div>  
          </dd>  
          <dt>Mobile</dt>  
          <dd><input type="text" required pattern="\+91\d{10}" ngModel #Mobile="ngModel" name="Mobile"></dd>  
          <dd class="text-danger" \*ngIf="frmRegister.submitted && Mobile.invalid">  
              <div \*ngIf="Mobile.errors?.['required']">Mobile Required</div>  
              <div \*ngIf="Mobile.errors?.['pattern']">Invalid Mobile +91 and 10 digits</div>  
          </dd>  
          <dt>City</dt>  
          <dd>  
             <select (change)="CityChanged($event)" ngModel #City="ngModel" name="City">  
                <option value="-1">Select Your City</option>  
                <option value="Delhi">Delhi</option>  
                <option value="Hyd">Hyd</option>  
             </select>  
          </dd>  
          <dd class="text-danger">  
             {{CityError}}  
          </dd>  
       </dl>  
       <button class="btn btn-primary">Register</button>  
   </form>  
</div>  
  
                        Model Driven or Reactive Forms  
                          
- Model Driven specifies that configurating and manipulations are handled at**

**"Controller" level.  
- It have clean separation between code and UI.  
- UI is light weight and faster in rendering.  
- It is good for extensibility and testability.  
- It is good for dynamic form, which can change according to situation.  
- It increases the number of requests and also page initial load time.  
- It is not good for inline documentation.  
- They are Asynchronous, faster in interactions.  
- They internally uses Ajax.  
- They enable "Patial PostBack".  
- They enable "Async Validations".  
- Reactive forms require  "ReactiveFormModule"  from @angular/forms  
- To create a form and its element it provides various classes  
  
            ClassName             Purpose**

**FormGroup             To create <form> element  
            FormControl           To create <input> <select> <textarea> form elements.  
            FormBuilder           It is a form service.  
  
Syntax:  
            public  frmRegister = new FormGroup({  
                    FieldName : new FormControl("value", Validations[]),  
                    FieldName : new FormControl("value", Validations[])  
            })  
  
- You have to bind the form group and controls to UI.  
  
            <form   [formGroup]="frmRegister">  
                <input type="text"  formControlName="FieldName">  
                <select formControlName="FieldName"> </select>  
            </form>  
  
Ex: Basic Reactive Form with simple group and controls  
  
reactive-form.component.ts  
  
import { Component } from '@angular/core';  
import  { FormGroup, FormControl, Validators } from '@angular/forms';  
  
@Component({  
  selector: 'app-reactive-form',  
  templateUrl: './reactive-form.component.html',  
  styleUrls: ['./reactive-form.component.css']  
})  
export class ReactiveFormComponent {  
    public frmRegister = new FormGroup({  
        UserName: new FormControl('Your Name'),  
        Age: new FormControl(0),  
        Mobile: new FormControl('')  
    })  
    public SubmitClick(data:any){  
        alert(JSON.stringify(data));  
    }  
}  
  
reactive-form.component.html  
  
<div class="container-fluid">  
    <h2>Register User</h2>  
    <form [formGroup]="frmRegister" (submit)="SubmitClick(frmRegister.value)">  
        <dl>  
            <dt>User Name</dt>  
            <dd><input type="text" formControlName="UserName" ></dd>  
            <dt>Age</dt>  
            <dd><input type="number" formControlName="Age"></dd>  
            <dt>Mobile</dt>  
            <dd><input type="text" formControlName="Mobile"></dd>  
        </dl>  
        <button>Register</button>  
    </form>  
</div>  
  
- You can configure nested forms with Async approach  
  
Syntax:  
         frmRegister = new FormGroup({  
                Name : new FormControl(" "),  
                Price  : new FormControl(0),  
                frmVendor: new FormGroup({  
                      VendorName: new FormControl("");  
                      VendorRating: new FormControl(0);  
                })  
        })  
  
        <form  [formGroup] = "frmRegister">  
  
             <input type="text"  formControlName="Name">  
             <input type="text"  formControlName="Price">  
         
              <div formGroupName="frmVendor">  
                <input type="text" formControlName="VendorName">  
                <input type="range" formControlName="VendorRating">  
                 <button> Save</button>  
              </div>  
             <button>Submit</button>  
        </form>  
  
        this.frmVendor.patchValue( { VendorRating : 4.3} );  
        this.frmVendor.update({})**

**Note: You have to access any nested control from a "Form" by using accessors.  
  
Syntax: Angular upto 14 from 11  
  
            get  VendorName()  
            {  
             return  this.frmRegister.get("VendorName") as FormControl;  
            }  
  
Syntax: Angular 15  
            get  VendorName()  
            {  
             return  this.frmRegister.get("VendorName");  
            }  
  
\* Angular 15 introduced strongly typed Reactive Froms.  
  
Ex:  
reactive-form.component.ts  
  
import { Component } from '@angular/core';  
import  { FormGroup, FormControl, Validators } from '@angular/forms';  
  
@Component({  
  selector: 'app-reactive-form',  
  templateUrl: './reactive-form.component.html',  
  styleUrls: ['./reactive-form.component.css']  
})  
export class ReactiveFormComponent {  
     public frmRegister = new FormGroup({  
          Name: new FormControl(''),  
          Price: new FormControl(0),  
          frmVendor: new FormGroup({  
              VendorName: new FormControl(''),  
              VendorRating: new FormControl(0)  
          })  
     })  
  
     get VendorName(){  
        return this.frmRegister.get("VendorName");  
     }  
  
     get VendorRating(){  
        return this.frmRegister.get("VendorRating");  
     }  
  
     public RegisterClick(data:any) {  
         alert(JSON.stringify(data));  
     }  
     public UpdateClick(){  
         this.frmRegister.patchValue({  
             Price : 70000.40,  
             frmVendor: {  
                VendorRating: 4.1  
             }  
         })  
     }  
}  
  
reactive-form.component.html  
  
<div class="container-fluid">  
    <h2>Product Register</h2>  
    <form [formGroup]="frmRegister" (submit)="RegisterClick(frmRegister.value)">  
        <dl>  
            <h3>Product Details</h3>  
            <dt>Name</dt>  
            <dd><input type="text" formControlName="Name"></dd>  
            <dt>Price</dt>  
            <dd><input type="text" formControlName="Price"></dd>  
        </dl>  
        <div formGroupName="frmVendor">  
            <h3>Vendor Details</h3>  
            <dl>  
                <dt>Vendor Name</dt>  
                <dd><input type="text" formControlName="VendorName" ></dd>  
                <dt>Vendor Rating</dt>  
                <dd><input type="number" formControlName="VendorRating"></dd>  
            </dl>  
        </div>  
        <button>Register</button>  
        <button type="button" (click)="UpdateClick()">Update</button>  
    </form>  
</div>  
  
FAQ: What is the issue with FormGroup and FormControl?  
Ans : They use dynamic memory, which refers to "Single Call".  
        Better to use "FormBuilder".  
   
                                FormBuilder       
- It is a service.  
- It uses "Single Ton" pattern  
- It is good for forms that have to use continuous memory.  
- You have to inject FormBuilder into component  
          constructor(private  fb : FormBuilder) {  }  
  
- FormBuilder service provides various methods  
     
         group()             It is used to configure a form or child form.  
         control()          It is used to configure a control.  
         array()             It is used to configure a form array with set of controls.**

**Ex:  
reactive-form.component.ts  
  
import { Component } from '@angular/core';  
import  { FormGroup, FormControl, Validators, FormBuilder } from '@angular/forms';  
  
@Component({  
  selector: 'app-reactive-form',  
  templateUrl: './reactive-form.component.html',  
  styleUrls: ['./reactive-form.component.css']  
})  
export class ReactiveFormComponent {  
  
    constructor(private fb: FormBuilder){  
    }  
    public frmRegister = this.fb.group({  
        Name : this.fb.control(''),  
        Price : this.fb.control(0),  
        frmVendor: this.fb.group({  
            VendorName: this.fb.control(''),  
            VendorRating: this.fb.control(0)  
        })  
    })  
  
     get VendorName(){  
        return this.frmRegister.get("VendorName");  
     }  
  
     get VendorRating(){  
        return this.frmRegister.get("VendorRating");  
     }  
  
     public RegisterClick(data:any) {  
         alert(JSON.stringify(data));  
     }  
     public UpdateClick(){  
         this.frmRegister.patchValue({  
             Price : 70000.40,  
             frmVendor: {  
                VendorRating: 4.1  
             }  
         })  
     }  
}  
  
a) group()            It configures <form> element  
b) control()            It configures <input> <select> <textarea> etc..  
c) array()                It allows to add, remove elements dynamically.  
  
Syntax:  
        FormArray. push()  
        FormArray. removeAt(index)  
  
Note: Accessor for Form Array from Angular 15 requires the type to define  
  
        get  Photos(): FormArray {  
            return this.frmRegister.get("Photos") as FormArray;  
        }  
  
Ex:  
reactive-form.component.ts  
  
import { Component } from '@angular/core';  
import  { FormGroup, FormControl, Validators, FormBuilder, FormArray } from '@angular/forms';  
  
@Component({  
  selector: 'app-reactive-form',  
  templateUrl: './reactive-form.component.html',  
  styleUrls: ['./reactive-form.component.css']  
})  
export class ReactiveFormComponent {  
  
    constructor(private fb: FormBuilder){   
    }  
  
    public frmRegister = this.fb.group({  
        Name : this.fb.control(''),  
        Price : this.fb.control(0),  
        frmVendor: this.fb.group({  
            VendorName: this.fb.control(''),  
            VendorRating: this.fb.control(0)  
        }),  
        Photos: this.fb.array([this.fb.control('')])  
    })  
  
     get VendorName(){  
        return this.frmRegister.get("VendorName");  
     }  
  
     get VendorRating(){  
        return this.frmRegister.get("VendorRating");  
     }  
  
     get Photos():FormArray{  
        return this.frmRegister.get("Photos") as FormArray;  
     }  
  
     public RegisterClick(data:any) {  
         alert(JSON.stringify(data));  
     }  
     public UpdateClick(){  
         this.frmRegister.patchValue({  
             Price : 70000.40,  
             frmVendor: {  
                VendorRating: 4.1  
             }  
         })  
     }  
     public AddMoreClick(){  
        this.Photos.push(this.fb.control(''));  
     }  
     public RemoveClick(index:number) {  
        this.Photos.removeAt(index);  
     }  
}  
  
reactive-form.component.html  
  
<div class="container-fluid">  
    <h2>Product Register</h2>  
    <form [formGroup]="frmRegister" (submit)="RegisterClick(frmRegister.value)">  
        <dl>  
            <h3>Product Details</h3>  
            <dt>Name</dt>  
            <dd><input type="text" formControlName="Name"></dd>  
            <dt>Price</dt>  
            <dd><input type="text" formControlName="Price"></dd>  
        </dl>  
        <div class="mb-3 mt-2">  
            <h3>Upload Photos <button type="button" (click)="AddMoreClick()" class="btn btn-link">Add More</button></h3>  
            <div class="mt-2 mb-2" \*ngFor="let item of Photos.controls; let i=index">  
                <input type="file" formControlName="i"> <button (click)="RemoveClick(i)" class="btn btn-danger"> <span class="bi bi-trash"></span> </button>  
            </div>  
        </div>  
        <div formGroupName="frmVendor">  
            <h3>Vendor Details</h3>  
            <dl>  
                <dt>Vendor Name</dt>  
                <dd><input type="text" formControlName="VendorName" ></dd>  
                <dt>Vendor Rating</dt>  
                <dd><input type="number" formControlName="VendorRating"></dd>  
            </dl>  
        </div>  
         
        <button>Register</button>  
        <button type="button" (click)="UpdateClick()">Update</button>  
    </form>  
</div>  
  
  
                         Reactive Form Validation  
- Validations are defined for controls in "controller".  
- UI elements will just use the validations, no need to configure validation at**

**UI level.  
- Reactive Form uses "Validators" class that provides a set of validators like  
                required  
                minlength  
                maxlength  
                pattern etc...  
  
Syntax:  
Name : fb.control(' ', Validators[])  
  
Ex:  
Name : fb.control(' ', [Validators.required, Validators.minlength(4)]  
  
Ex:  
reactive-validation.component.ts  
  
import { Component } from '@angular/core';  
import { FormBuilder, FormControl, Validators } from '@angular/forms';  
  
@Component({  
  selector: 'app-reactive-validation',  
  templateUrl: './reactive-validation.component.html',  
  styleUrls: ['./reactive-validation.component.css']  
})  
export class ReactiveValidationComponent {  
    constructor(private fb: FormBuilder){  
  
    }  
    public frmRegister = this.fb.group({  
         Name : this.fb.control('',[Validators.required, Validators.minLength(4)]),  
         Mobile: this.fb.control('', [Validators.required, Validators.pattern(/\+91\d{10}/)])  
    })  
    get Name():FormControl{  
      return this.frmRegister.get("Name") as FormControl;  
    }  
    get Mobile():FormControl{  
      return this.frmRegister.get("Mobile") as FormControl;  
    }  
}  
  
reactive-validation.component.html  
  
<div class="container-fluid">  
    <form [formGroup]="frmRegister">  
        <h2>Register</h2>  
        <dl>  
            <dt>Name</dt>  
            <dd><input type="text" formControlName="Name"></dd>  
            <dd class="text-danger" \*ngIf="Name.touched && Name.invalid">  
                <div \*ngIf="Name.errors?.['required']">Name Required</div>  
                <div \*ngIf="Name.errors?.['minlength']">Name too short</div>  
            </dd>  
            <dt>Mobile</dt>  
            <dd>  
                <input type="text" formControlName="Mobile">  
            </dd>  
            <dd class="text-danger" \*ngIf="Mobile.touched && Mobile.invalid">  
                <div \*ngIf="Mobile.errors?.['required']">Mobile Required</div>  
                <div \*ngIf="Mobile.errors?.['pattern']">Invalid Mobile +91 and 10 digits</div>  
            </dd>  
        </dl>  
        <button>Register</button>  
    </form>  
</div>**

**14. Angular ROUTING  
- It is a technique used in web applications to handle navigation.  
- A hyper link also configures navigation, but it navigates the user from one**

**page to another or to any named location in page.  
                        <a href="page.html>  
  
- Routing uses "Ajax" calls.  
- User can stay on one page and can get access to everything from the page.  
- User will not navigate from one page to another. Eveything comes on to**

**single page.  
- It is often reffered as "SPA" [Single Page Application]  
- Routing can load new details into page, without reloading page.  
- Routing also enables "User Friendly" and "SEO friendly URL's".  
  
Ex: Without Routing**

**http://www.amazon.in/electronics.jsp?category=mobiles http://www.amazon.in/electronics.jsp?category=mobiles&brand=realme   
  
Ex: With routing**[**http://www.amazon.in/electronics/mobiles/realme**](http://www.amazon.in/electronics/mobiles/realme) **- Routing is configured  
        a) Server Side  [.NET, NodeJS, JSP, PHP] - API  
        b) Client Side   [Angular, React, Vue..]  
  
- Angular Routing is handled by "@angular/router" library  
  
- Angular Routing requires following library  
        RouterModule Routes  
        ActivatedRoute Route  
        RouterOutlet Router  
        RouterLink  
  
Ex:  
1. Create a new project in workspace  
  
        > ng generate  application  amazon  
           would you like to use routing? yes  
  
2. A file is added into "app" folder  
          app-routing.module.ts**

**\*One Zip file is there named amazon.zip**

**Types of Paths:  
- Route Paths are configure in 2 ways  
        a) Relative Path  
                <a routerLink="details"> Details </a>  
        b) Absolute Path  
                <a routerLink="/details"> Details </a>  
  
- Absolute path removes the existing and adds the current.  
- Relative path adds to existing path.  
  
Route Parameters:  
- Every route can use parameters.  
- Route parameters are used to transport data from one component to**

**another.  
- Parameters are defined in path  
  { path: "details/:id/:name/:price", component : DetailsComponent}  
  
- Pameters are passed in URL** [**http://localhost:4200/details/1/tv/5000**](http://localhost:4200/details/1/tv/5000) **[Every parameter is mandatory]  
  
                map { id:1, name:"tv", price:5000 }  
  
- Route parameters can be accessed and used in any component by using**

**"ActivatedRoute" service.  
  
            constructor(private route: ActivatedRoute){  
            }  
            this.route.snapShot.paramMap.get("id");  
  
- Child Routes  
  
        { path: "parent" ,  component : ParentComponent,  
             children : [  
                { path: "child", component: ChildComponent }  
            ]  
        }  
  
- Route Guards will not allows every user to access the route.  
  You can set authentication for routes.  
        >ng g guard  amazon-admin   --skip-tests  
  
         
Deployment, Testing**

**\*One Zip file is there named amazon.zip**

**15. Angular Testing  
                                 
- Testing is the process of verifying  
        "AS-IS  and TO-BE"  
  
- AS-IS refers to what developer have designed.  
- TO-BE refers to what client wants.  
  
        AS-IS        ===    TO-BE                Test Pass  
        AS-IS        !==    TO-BE                Test Fail  
  
- Angular Inbuilt Provides "Jasmine-Karma" framework.  
  
Step-1  
- You create your component and add methods.  
  
Syntax:      home.component.ts  
  
Step-2  
- You have to add a test file for component.  
  
Syntax:    home.component.spec.ts  
  
    >ng   g   c   home    --skip-tests [Don't use this flag if you want a test file]  
  
Step-3  
- You have to write the test methods in "spec.ts"  
- Jasmine Framework methods that match various testing phases  
  
        Test Phases                  
        a) Arrange            Configure  
        b) Act                    Define the functionality to test  
        c) Assert                Report the result  
  
        Arrange           describe()  
        Act                    it()  
        Assert               expect()  
  
Ex:  
1. Generate a new component  
  
> ng g c  amazon-cart  
  
2. Add methods  
  
amazon.cart.component.ts  
  
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-amazon-cart',  
  templateUrl: './amazon-cart.component.html',  
  styleUrls: ['./amazon-cart.component.css']  
})  
export class AmazonCartComponent {  
    public Title():string {  
        return "Amazon Cart";  
    }  
    public Total(qty:number, price:number):number {  
        return qty \* price;  
    }  
}  
  
3. Go to "amazon-cart.component.spec.ts"  
  
import { ComponentFixture, TestBed } from '@angular/core/testing';  
  
import { AmazonCartComponent } from './amazon-cart.component';  
  
describe('AmazonCartComponent', () => {  
  let component: AmazonCartComponent;  
  let fixture: ComponentFixture<AmazonCartComponent>;  
  
  beforeEach(async () => {  
    await TestBed.configureTestingModule({  
      declarations: [ AmazonCartComponent ]  
    })  
    .compileComponents();  
  
    fixture = TestBed.createComponent(AmazonCartComponent);  
    component = fixture.componentInstance;  
    fixture.detectChanges();  
  });  
  
  it('should create', () => {  
    expect(component).toBeTruthy();  
  });  
  
  // Write a test for Title() - It must return  "Amazon Shopping Cart"  
  
  describe("Cart Title Test",()=> {  
      it("Expecting Amazon Shopping Cart as Title",()=>{  
          expect(component.Title()).toBe("Amazon Shopping Cart");  
      })  
  });  
  
  // Write a test for Total() - It must return 40000  
  
  describe("Total Test", ()=>{  
      it("Total must be 40000",()=> {  
          expect(component.Total(2,20000)).toBe(40000);  
      })  
  });  
  
});  
  
4. Run test  
  
    >ng  test   --project=amazon**

**16. Angular APPLICATION DEPLOYMENT  
- Build and Deploy  
- Building is the process of checking  
        Syntax Errors  
        Code Snippets  
        Dependencies  
        Providers  
        Injectors  
  
    >ng build    
  
- All components logic is bundlend into "dist" folder  
- You can deploy of servers  
        a) Local Servers  
        b) Cloud Servers  
- FireBase  [Google]  
- Azure  
- AWS  
- GIT Hub Pages  
- NPM  
  
1. You have to build project  
  
        >ng build --project=amazon  
  
2. Install Firebase tools on your PC  
  
        > npm install -g  firebase-tools  
  
3. Add Firebase library to Angular Application  
     
        > ng   add  @angular/fire --project=amazon  
  
4. Go to the Firebase website and create a new project**[**https://firebase.google.com/**](https://firebase.google.com/)**=> console  
  
          > ng   add  @angular/fire --project=amazon  
  
        (\*) ng deploy -- hosting**[**https://angular-smart-shop.web.app**](https://angular-smart-shop.web.app/) **5. Run the command**