Angular13

- Angular is an open source, cross platform developers platform.
- A developers platform provides end to end solution for building, debugging, testing and deploying.
- It provides language, framework, tools for end to development.

Language	- TypeScript
Framework	- MVC, MVVM, Cordova, NativeScript, Ionic etc..
Testing Framework	- Jasmine , Karma, Protractor
Template Library	- Materials, Animations etc..

- These tools and languages are used for building Crossplatform mobile and web application. It is suitable for even SPA and Progressive web apps.

Key Features of Angular:

1. Develop Across All Platforms
  *It is open source
  *It uses Cordova, Ionic, NativeScript to develop cross platform mobile application.
  *Application can run on any device
  *For web, distributed, native mobile, native desktop.

2. Speed & Performance
  * It is modular in approach [Application Sepcific Frame Work]
  * It will not use legacy of library
  * It uses asynchronous techniques.
  * Angular JS uses Legacy and Angular uses Modular
    Angular is 10x faster than Angular JS
  * Application can even reach devices with low bandwidth.
  * Angular 9 and above are using AOT compiler [ahead-of-time]. Earlier Angular uses JIT [just-in-time compiler].

JIT Compiler :
- Will compile your code after the page is loaded into browser.
- More time to render

AOT Compiler :

- Converts your Angular HTML and TypeScript code into efficient JavaScript code during building phase before browser downloads and runs that code.
- This provides faster rendering
- Fewer asynchronous requests.
- Smaller angular framework download size
- Template [HTML] errors are detected earlier.

Note: Angular provides a choice for developer to choose between JIT and AOT.

3. Supports Differential loading

Legacy Browser	HTML 4 and ES5
Modern Browser	HTML 5 and ES6

Angular can build library dynamically and load according to browser version.

4. Angular Supports Architectural Frameworks like
   - MVC, MVP, MVVM

M	- Model - Data
V	- View - UI
C	- Controller - Application Logic

Setup Environment for Angular

1. Download and Install Node JS for "NPM"

   https://nodejs.org/en/download/

2. Download and Install Angular CLI, which is an command line tool used to manage your application application.

   https://cli.angular.io/

- Open Your command prompt
- Type the following

```
C:\> npm  install -g  @angular/cli
```

- Check the angular version after installing

```
C:\> ng --version
```

Update your CLI if it is using older version:

```
C:\> npm uninstall  @angular/cli
C:\> npm install @angular/cli
          (or)
C:\> ng update
C:\> ng update @angular/cli @angular/core  --allow-dirty
```

Setup Workspace for Angular Projects

- If you intend to have multiple projects then configure a workspace and add projects into workspace.

- First : Create Workspace without any application

```
C:\> ng new  your-workspace  --createApplication=false
```

WORKSPACE CONFIG – FILES	DESCRIPTION – PURPOSE
node_modules	- It is a folder that comprises of library installed using NPM. - Library can be like Bootstrap, Jquery, Animations, Materials etc. - This library will be shared to all projects in the current workspace.
.editorconfig	- Configuration file for code editors. - Multiple developers working on the same project across various editors and IDE's may have different configuration, which leads to code consistency issues. - EditorConfig file allows to configure common rules for coding. - Various Editors used for Angular are BBEdit, Code Crusader, CodeLite, elementary Code, Builder, Gitea, - GitHub, GitLab, GitBucket, Gogs, Visual Studio Code, Komodo, Kakoune, PyCharm, RubyMine, WebStrom etc..
.gitignore	- Specifies intentionally untracked files that Git should ignore.

angular.json	- CLI Configuration for all project in the workspace. - Configuration in angular.json includes options for build, serve, test tools etc.
package.json	- Configures npm package dependencies that are available to all projects in workspace.
package-lock.json	- Provides the version information of all package that are installed into “node_modules.
README.md	- Help document
tsconfig.json	- TypeScript configuration file. - Allows to configure the target JS version ES5, ES6, output dir
tsLint.json	- It is language analysis tool. - It sets the rules for typescript. - Datatype, Code Block etc.

Update your Angular CLI from Older Version to New

- npm uninstall -g @angular/cli
- npm cache clear --force

Install New CLI

- npm install -g @angular/cli@latest

Create a new Angular Application

1. Open Terminal
2. Change to your workspace folder
3. Type the following command
   C:\projects-workspace> ng generate application your_app_name
   C:\projects-workspace> ng generate application amazon

4. This will prompt you with following questions

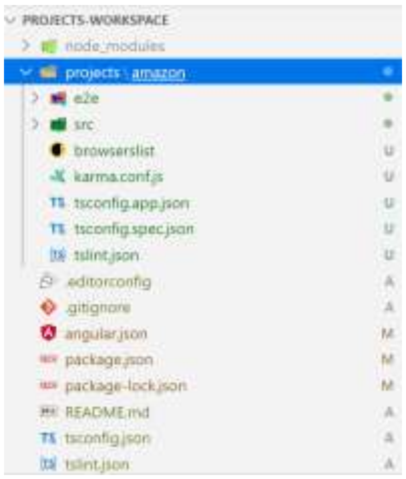
a) Would you like to share anonymous usage data about this project with the Angular Team at

Google under Google's Privacy Policy at https://policies.google.com/privacy? For more

details and how to change this setting, see http://angular.io/analytics. (y/N) N

b) Would you like to add Angular routing? (y/N) N
c) Which stylesheet format would you like to use? CSS

5. This will install all library required for Angular Application
6. Your New Application is in "projects" folder



```
PROJECTS-WORKSPACE
  > node_modules
  v projects \ amazon
      > e2e
      > src
        browserslist          U
        karma.conf.js         U
        tsconfig.app.json     U
        tsconfig.spec.json    U
        tslint.json           U
    .editorconfig             A
    .gitignore                A
    angular.json              M
    package.json              M
    package-lock.json         M
    README.md                 A
    tsconfig.json             A
    tslint.json               A
```

7. Run your project
   - Open Workspace location in Terminal
   - Run the following command
     C:\projects-workspace>ng serve --project=amazon
   - Application starts on Live Server listening on http://localhost:4200
8. Open any browser and request
   http://localhost:4200
9. Angular Application Files and Folder

Angular Application Files & Folder	Description
e2e	- An end-to-end folder - Contains source files for a set of end-to-end tests, which are related to current application.
Src	- Source files for the root level application. - It includes both dynamic and non-dynamic files.
.browserslist	- The config to share target browser and Node.js versions between different front-end tools. [plugins]
Karma.conf.js	- Application specific karma config file - Karma is used for testing.
tsconfig.app.json	- Application specific typescript configuration file.
tsconfig.spec.json	- Typescript configuration for the application tests.
tslint.json	- Application specific TSlint.

Application Source Files

[src – folder]

App Support Files	Purpose
app/	- Contains the component files in which your application logic and data are defined. - You can define components, modules, services, pipes, routing etc.
assets/	- It comprises of non-dynamic files like, images, text, pdf and other static resources.
environments/	- Contains build configuration options for particular target environment, like development, production environment etc.
favicon.ico	- An icon to use for bookmarking your application in browser.
index.html	- Application start with index.html page - This is the first page to be served.
main.ts	- The main entry point for your application. - Compiles the application with the JIT compiler and bootstraps the application. - It creates a Chunk that compiles and converts the static DOM into Dynamic. - You can also use “AOT” compiler. - You can define --aot while using serve.
polyfills.ts	- Provides polyfill scripts for browser support.

	- It uses differential loading technique. - It loads legacy and modern scripts according to browser.
styles.css	- It comprises of global styles. - The CSS code or imports which you can use from any component.
test.ts	- This is main entry point unit testing.

App folder

- It comprises of application components, services, pipes, modules etc.
- By default, every angular application comes with a component called "AppComponent".
- Every component comprises of 4 files:
    - app.component.html : Presentation
    - app.component.ts : Logic
    - app.component.css : Styles
    - app.component.spec.ts : Testing (Test Specification)

  Note: A component can be designed in in-line documentation technique, where only one file is created for component, which contains all html, css, and ts.

  "app.component.ts"

app.module.ts

- Your application comprises of components, modules, pipes, services etc.
- Every modules or library that you want to use in application must be configured in "app.module.ts"
- It uses Dependency Injection which identifies the dependencies required for your components and loads them into memory.
- App module file contain meta data.

- Main.ts is compiling your application, it verifies the dependencies from app.module.ts

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

import { AppComponent } from './app.component';

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

- NgModule({}) : Metadata (information about your application) provided to compiler
- declarations : Registers your components
- imports : Registers your modules
- providers : Registers your services
- bootstrap : Specifies the component to start with

Note: The component you define in bootstrap the same component must be in “index.html”

Angular Component

- Components are Building Blocks for Angular Application.
- A component provides UI and logic for interacting with application.
- A component comprises of styles, which can make presentation interactive and responsive.
- Angular provides a component library called “Angular Material”. You can add the component library and import pre-defined components and use in your application.
- Angular allows to create custom components.

Custom Components:

- Every component in angular comprises of Presentation, Logic and Styles.
- Presentation is defined with HTML.
- Logic is configured using TypeScript, which is transcompiled into JavaScript.
- Styles are defined inline, embed or CSS.
- The component related all properties and methods are derived from “Component” base of “@angular/core” library.
- Technically component is configured as a Typescript file “.ts”
- The component behaviour comes from the “Component” base.
- The component behaviour is given by using component meta data. @Component({ property: value, property: value })
- Its presentation is defined in “.html”.
- Its styles are defined in “.css” or you also define inline and embed.
- You can design a component by using following techniques
  - In-line documentation technique
  - Code behind documentation technique

In-line documentation technique:

- In this technique the presentation, logic and styles all are maintained in a single file.
- Component is configured in only one file ".ts"
- In-line technique is good if your component is having a simple functionality.
- It is good if regular extensions are not required.
- Hard to test.
- Easy extensions are not supported.
- If your configured everything in one file then it will reduces the number of requests and improve the page load time.

Note: @Component() Decorator that marks a class as an Angular component and provides configuration metadata that determines how the component should be processed, instantiated, and used at runtime.

Meta Data comprises of information, which includes what the markup used for presentation, what styles are used for markup, what animations are defined for markup etc.

Metadata Property	Purpose
selector	- Specifies a name used for accessing the component in any another component or from index.html page. - Selector is defined as a Directive, which is used as element. Ex: <app-home> </app-home>
template	- Specifies directly the markup to be rendered when component is requested.
templateUrl	- Specifies the file that contains markup.

	- You can link an HTML file for component.
styles	- It is a collection of style properties and their values. - It is an Array type.
styleUrls	- It can access styles from an external stylesheet.
animations	- It defines a set of CSS keyframes to use for animations.

Ex:

1. Go to "app" folder and add a new file by name

home.component.ts

```
import { Component } from '@angular/core';
@Component({
   selector: 'app-home',
   template: `
   <h2>Amazon Home</h2>
   <div id="offer">{{msg}}</div>
   <p>This is our first component</p>
   `,
   styles: ['h2{text-align:center;color:darkcyan}', '#offer{border:2px solid darkcyan; border-radius:20px; box-shadow:2px 3px 4px darkcyan}']
})

export class HomeComponent {
   public msg = '02-Aug-2020 to 04-Aug-2020 Monsoon Sale 70% Off';
}
```

2. Go to “app.module.ts” and register your component and set in bootstrap.

```
import { HomeComponent } from './home.component';
@NgModule({
  declarations: [
    AppComponent,
    HomeComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [HomeComponent]
})
```

3. Go to “index.html”
   ```
   <body>
   <app-home> </app-home>
   </body>
   ```
4. Start your project
   ```
   > ng serve  - -project=amazon
   ```

Component using Code Behind Technique

- In this technique the code, presentation and styles are maintained in separate files.
- It is easy to extend and test.
- If you have regular extensions to add into your functionality then you can go with “Code Behind” technique.
- However using multiple files will increase the load time.

Ex:

1. Create a new Folder into “app” folder by name “login”
2. Add following files
   - login.component.ts
   - login.component.html
   - login.component.css
3. login.component.ts

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


@Component({
   selector: 'app-login',
   templateUrl: 'login.component.html',
   styleUrls: ['login.component.css']
})


export class LoginComponent {
   public title = 'User Login';
}
```

4. login.component.html

```
<div class="form-login">
    <h2>{{title}}</h2>
    <dl>
        <dt>User Name</dt>
        <dd>
            <input type="text">
        </dd>
        <dt>Password</dt>
        <dd>
            <input type="password">
        </dd>
    </dl>
    <button>Login</button>
</div>
```

5. login.component.css

```
.form-login {
    width: 300px;
    margin:auto;
    align-items: center;
    justify-content: center;
    border:2px solid darkcyan;
    box-shadow: 2px 2px 3px darkcyan;
    border-radius: 10px;
}
```

```
button {
   background-color: darkcyan;
   color:white;
}
```

Installing and Enabling Bootstrap for Angular Project

1. Open your workspace location in Terminal
2. Type the following command
   > npm install bootstrap
3. Bootstrap repository is copied into “node_modules”

```
node_modules
|_bootstrap
   |_dist
      |_css
         |_boostrap.css
```

4. Make bootstrap CSS as global for all components.
   a. Go to “styles.css”
   b. Import bootstrap css

```
@import "../../../node_modules/bootstrap/dist/css/bootstrap.css";
```

Ex:

1. Go to “login.component.html”

```
<div class="container-fluid">
 <h2 class="text-center text-primary">{{title}}</h2>
 <div class="form-group">
  <label>User Name</label>
  <div>
```

```
        <input type="text" class="form-control">
    </div>
   </div>
   <div class="form-group">
    <label>Password</label>
    <div>
        <input type="password" class="form-control">
    </div>
   </div>
   <div class="form-group">
      <button class="btn btn-primary btn-block">Login</button>
   </div>
  </div>
```

2. Go to “login.component.css”

```
.container-fluid {
    width: 300px;
    padding: 30px;
    justify-content: center;
    align-items: center;
    margin:auto;
}
```

Angular CLI Commands to Add a new Component

- CLI is a command line tool that provides commands which can generate components and register in the application.

- CLI have set of commands for all type interactions with application.
- CLI commands are executed for project in “app” folder through a terminal.
- For component the following commands are used
  - ng generate component name : Generate a new component
  - ng g c name : Short hand method
- You can use several attributes to manage the component generation

ng generate component --Attribute	Purpose
--flat=true | false	It will not generate a folder for your component files. All component files are generated into “app” folder.
--dryRun=true | false	It will not actually generate the component. It will just show the preview of files added when it is really generated.
--inlineStyle=true|false	It will not generate a separate style sheet. The styles are maintained in “.ts” file. Ex: styles: [ ]
--inlineTemplate=true|false	It will not generate a separate “html” file. The markup is maintained in “.ts” file. Ex: template: \` \`
--skipTests=true|false	It will not generate a separate “spec.ts” test file.

Note: By default the value for all Boolean attributes is “true”.

Ex:

1. Right Click on “app” folder
2. Select “Open in Integrated Terminal”
   C:\...\app> ng generate component register --skipTests

Note: To run in browser directly after compiling.

- ng serve --project=amazon --open

Software Architectural Patterns used by Angular Client Side

1. MVC [Model View Controller]
2. MVVM [Model View – View Model]

What is MVC?

- MVC is a software architectural pattern.

What is software architectural pattern?

- It is similar to a design pattern but have a broader scope.

What is a design pattern?

- Design patterns are the solution for software design problem that a developer faces in real world application development. They are about creating objects, designing of classes and handling communication.
- The 23 GOF patterns are considered as foundation for all other patterns, which are categorized into 3 groups
    - Creational Patterns
    - Structural Patterns
    - Behavioural Patterns

Software Architectural Patterns

- Building
- Control the application flow [Framework]

MVC

[Model View Controller]

- MVC is a software architectural pattern.

- Software Architectural pattern is similar to Design Pattern but have a broader scope.
- They are responsible for both building the application and controlling the application flow.
- Trygve introduced MVC in the early 1970's and formulated with "Small Talk".
- Code separation and Code reusability concerns.
- MVC separates application into 3 components
  - Model
  - View
  - Controller
- MVC framework can used client side and server side.
- Various technologies are using MVC framework
  - Java: Spring
  - PHP: Cake PHP, Code Igniter
  - Python: Django, Flask
  - Ruby: Ruby on Rails
  - .NET: ASP.NET MVC
  - JavaScript: SPINE, Angular JS

Model:

- It represents the data we are working with as well as the business.
- It defines the rules that specify how data can be stored and manipulation.
- It is similar to a Data Access Layer.
- It is responsible for Data Validations.

Controller:

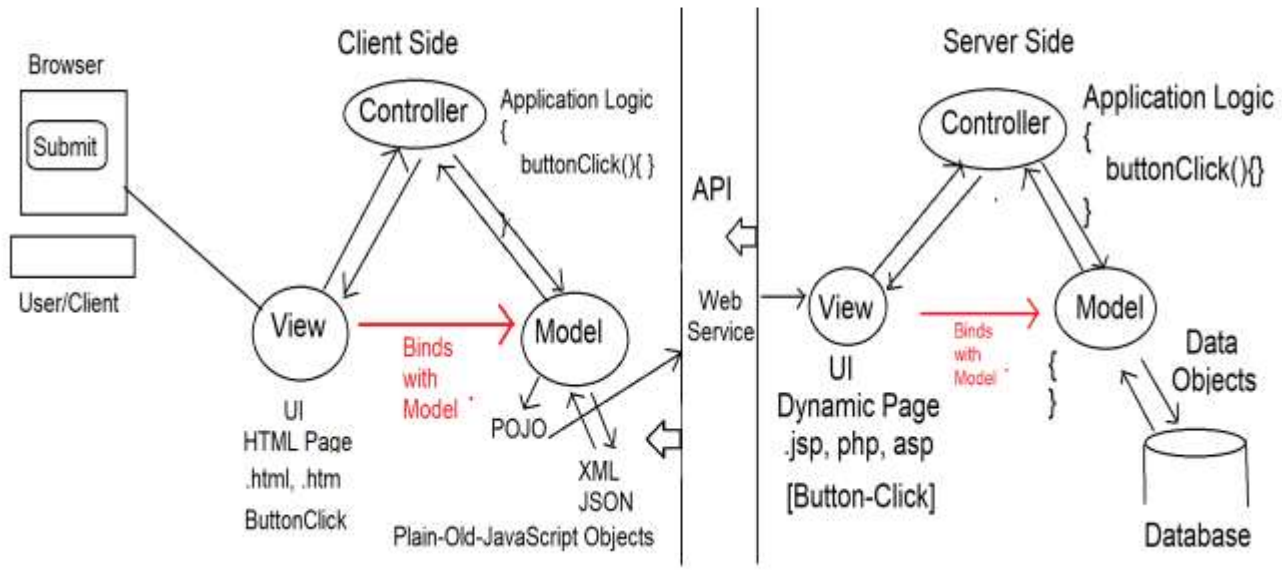
- It is the core component of MVC framework
- It handles overall application flow
- It enables communication between Model and View.
- It comprises of Application Specific Logic.

View:

- View describes the application UI.
- It is a template that renders HTML.

- It is under the control of View Engine
- Angular uses a View Engine called "IVY"

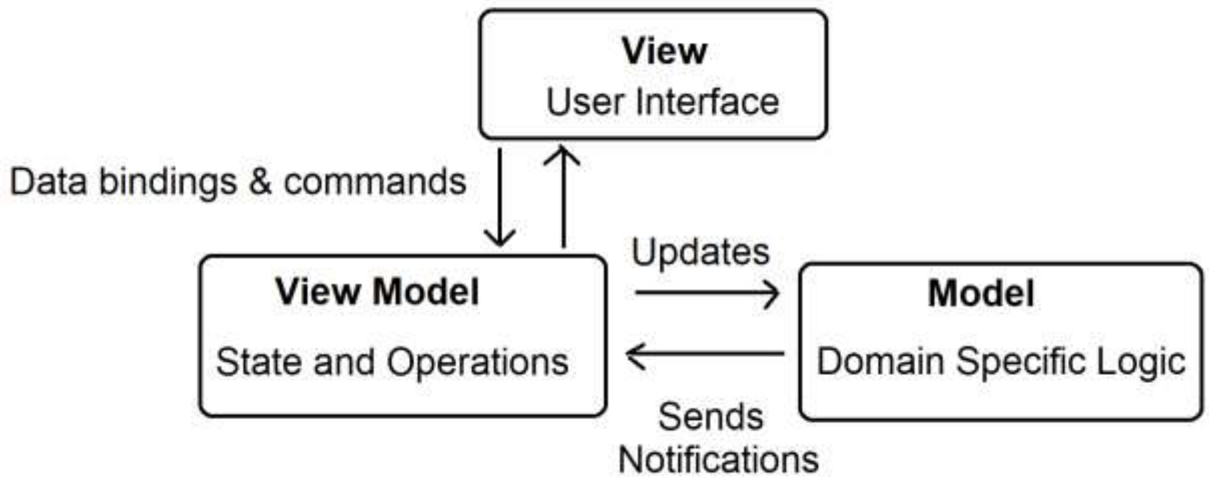
View Engines: Spark, Nahml, Django, Razor, ASPX, IVY etc.



MVVM

(Model View ViewModel)

- It is a software architectural pattern.
- More clear separation of User Interface.
- It separates the UI from Business Logic and Behaviour.
- Designed by Microsoft for use with Windows Presentation Foundation (WPF)
- 2005 by John Grossman [MVP]

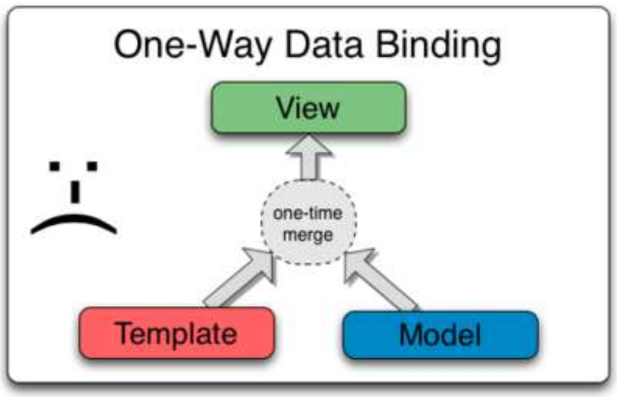


Data Binding in Angular

- The model data is updated to View.
- The model data is bound to the UI.
- Changes in the View are updated back to Model.
- Data Binding is categorized into 2 types
  - One Way Binding
  - Two Way Binding

One Way Binding

- The Model data is accessed and attached to View.
- It is only one direction i.e readonly.
- Any change in View will not update back to the Model.



- Angular can handle one-way binding by using following techniques
  - Interpolation {{ }}
  - Property Binding [ ]
  - Attribute Binding [attr.name]

Interpolation:

- It is a technique used to bind model data to UI by using data binding expression “{{ }}”
- You can use it as a literal [no need to bind with any property or attribute of element].
- You can directly display in any container like: <div>, <span>, <p>, <h2>, <dd>, <td> etc.
- It is one-way binding. Any change in the view will not update back to the model object.

Ex:

1. Add a new component
   > ng g c oneway --skipTests
2. Oneway.component.ts

```
  export class OnewayComponent {
 public product = {
  Name: 'Samsung TV',
  Price: 45000.55
 };
}
```

3. Oneway.component.html

```
<div class="container">
  <h2>Product Details</h2>
  <dl>
```

```
        <dt>Name</dt>
        <dd>{{product.Name}}</dd>
        <dd><input type="text" value="{{product.Name}}"></dd>
        <dt>Price</dt>
        <dd innerHTML="{{product.Price}}"></dd>
      </dl>
    </div>
```

Property and Attribute Binding

- HTML tag is defined with Attributes
  <table width="" height="" bgcolor="" align="">   Attributes [Immutable]
- HTML Elements are defined with Properties
  var table = document.createElement("table");
  table.height = 100;   ]
  table.width = 200;   ]   Properties [Mutable]
  table.align = center;   ]

<table class="form-control">

table.className = "form-control"

table.width = ""

Ex: Attribute Binding

```
public disableButton = true;
<button disabled="{{disableButton}}"> Submit </button>
```

Note: Try changing the boolean true to false and observer both property and attribute binding.

Attribute will not allow to change its state. Property can change the state.

Ex: Property Binding

```
public disableButton = true;
```

```
<button [disabled]=”disableButton”> Submit </button>
```

- Every HTML attribute is not having a relative property to handle dynamically. You have to use attribute as property binding with “attr” as prefix.

  Syntax:
  ```
  <element  [attr.attributeName]=”dynamicReference”>
  ```

  Ex:

1. Onewaydemo.component.ts

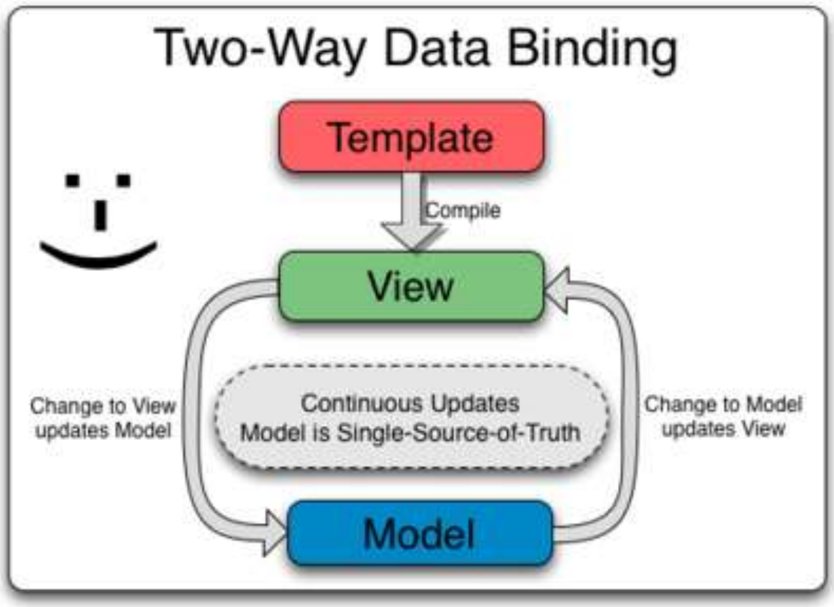
   ```
   public tableWidth = 300;
   public tableHeight = 100;
   ```

2. Onewaydemo.component.html

   ```
   <table border="1" [width]="tableWidth" [attr.height]="tableHeight" >
     <tr>
       <td>Name</td>
       <td>Price</td>
     </tr>
   </table>
   ```

Two Way Data Binding:

- The model data is bound with View.
- Changes in View will update the Model.
- Model to View and Vice Versa.
- Model is referred as “Single-Source-of-Truth”
- It contains information about the value before and after change.


Two-Way Data Binding
Template
Compile
View
Change to View updates Model
Continuous Updates Model is Single-Source-of-Truth
Change to Model updates View
Model

- NgModel is a directive responsible for handling two-way binding.
- It uses "Property and Event" binding technique.
- Property Binding is used to bind the model value.
- Event binding is used to update the changes.
- [ ] defines property binding.
- ( ) defines event binding
- [(ngModel)] two-way binding.
- NgModel is a member of "FormsModel" present in "@angular/forms" library

Ex:

1. Twowaybinding.component.ts

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


@Component({
  selector: 'app-twowaybinding',
  templateUrl: './twowaybinding.component.html',
  styleUrls: ['./twowaybinding.component.css']
})
export class TwowaybindingComponent {
 public name = 'Samsung TV';
 public city = 'Delhi';
 public inStock = true;
}
```

2. Twowaybinding.component.html

```
<div class="container-fluid">
<h2 class="text-center text-primary">Two Way Binding</h2>
<div class="row">
  <div class="col-3">
    <div class="form-group">
        <label>Name</label>
        <div>
            <input [(ngModel)]="name" type="text" class="form-control">
        </div>
    </div>
    <div class="form-group">
      <label>Shipped To</label>
      <div>
          <select [(ngModel)]="city" class="form-control">
            <option>Delhi</option>
            <option>Hyderabad</option>
        </select>
      </div>
     </div>
     <div class="form-group">
       <label>Is In Stock</label>
       <div>
         <input [(ngModel)]="inStock" type="checkbox"> Yes
       </div>
     </div>
    </div>
    <div class="col-9">
     <table class="table table-hover">
       <colgroup span="1" style="background-color: aquamarine;"></colgroup>
       <tr>
         <td>Name</td>
         <td [innerHTML]="name"></td>
       </tr>
       <tr>
         <td>Shipped To</td>
         <td>{{city}}</td>
       </tr>
       <tr>
         <td>Stock Status</td>
         <td>{{(inStock==true)?"Available":"Out of Stock"}}</td>
```

```
            </tr>
          </table>
        </div>
      </div>
      </div>
```

Ex:

1. twowaybinding.component.ts

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

@Component({
  selector: 'app-twowaybinding',
  templateUrl: './twowaybinding.component.html',
  styleUrls: ['./twowaybinding.component.css']
})
export class TwowaybindingComponent {
 public name = 'Samsung TV';
 public city = 'Delhi';
 public inStock = true;
 public product = {
   Name: '',
   City: '',
   InStock: false
 };
 public UpdateClick() {
   this.product = {
     Name: this.name,
     City: this.city,
     InStock: this.inStock
   };
 }
}
```

2. two way binding.component.html

```
<div class="container-fluid">
<h2 class="text-center text-primary">Two Way Binding</h2>
<div class="row">
  <div class="col-3">
    <div class="form-group">
        <label>Name</label>
        <div>
            <input [(ngModel)]="name" type="text" class="form-control">
        </div>
    </div>
    <div class="form-group">
      <label>Shipped To</label>
      <div>
        <select [(ngModel)]="city" class="form-control">
          <option>Delhi</option>
          <option>Hyderabad</option>
        </select>
      </div>
    </div>
    <div class="form-group">
      <label>Is In Stock</label>
      <div>
        <input [(ngModel)]="inStock" type="checkbox"> Yes
      </div>
    </div>
    <div class="form-group">
      <button (click)="UpdateClick()" class="btn btn-primary btn-
block">Update</button>
    </div>
  </div>
  <div class="col-9">
    <table class="table table-hover">
      <colgroup span="1" style="background-color: aquamarine;"></colgroup>
      <tr>
        <td>Name</td>
        <td [innerHTML]="product.Name"></td>
        </tr>
        <tr>
            <td>Shipped To</td>
            <td>{{product.City}}</td>
        </tr>
        <tr>
            <td>Stock Status</td>
            <td>{{(product.InStock==true)?"Available":"Out of Stock"}}</td>
        </tr>
    </table>
  </div>
</div>
</div>
```

Angular Directives

- Directive is technically a function.
- Directive is responsible for converting the static DOM element into dynamic DOM.
- Directive makes HTML more declarative.
- It extends HTML.

```
        <input type="text" [(ngModel)]>
        < Angular >
```

- Directives have different types of functionalities, they can be used as

    a) Element

b) Attribute

c) Class

d) Comment

- Directives are used as elements to return markup

```
<ng-form> </ng-form>
      <router-outlet> </router-outlet>
```

- Directives are used as attributes to extend HTML

```
<input type="text" ngModel #txtName="ngModel">
```

- Directives are used as classes to make the markup more interactive

```
<span class="ng-valid"> </span>
<span class="ng-pristine"> </span>
```

- Directives are uses as Comments to target legacy browser

```
<!-- ngModel:txtName -->
```

- Angular Directives are classified into 3 groups

1. Component Directives
2. Structrual Directives
3. Attribute Directives

Component Directives:

- The component directive is the most common directive in Angular.

- It is a template that comprises logic, presentation and styles.

- It is used for dynamically rendering HTML and handle interaction with user.

Ex:

    login.component.ts - logic
    login.component.html - presentation
    login.component.css - styles

    <app-login> </app-login>

- We can access and use built-in components from a library called "Angular Material"

Structural Directives

=================

- A structural directive is responsible for changing the DOM structure dynamically.

- It can add or remove element, It can iterate over element, it can switch between elements etc..

- Angular structural directives are

  a) ngIf

  b) ngSwitch

  c) ngFor

- Structural directives are added to HTML DOM element by using "*"

    <div *ngIf="">

```
<li *ngFor="">
```

- Every HTML element can use only one structural directive.

```
<li *ngFor=""    *ngIf="">    // invalid
```

NgIf Directive (ngIf)

================

- It is a structural directive uses to add or remove any DOM element dynamically.
- It uses a boolean condition or value to add and remove DOM element.

Hide an Image:

```
<img style="display:none">
```

This will keep the image in your page DOM but hides in the UI. It will not reduce burden on Page.

Syntax:

```
<img *ngIf="booleanValue/Expression">
```

Ex:

1. ifdemo.component.ts

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

```
@Component({
  selector: 'app-ifdemo',
  templateUrl: './ifdemo.component.html',
  styleUrls: ['./ifdemo.component.css']
})
export class IfdemoComponent {
   public product = {
     Name: 'Nike Casuals',
     Price: 4500.55,
     Photo: 'assets/shoe.jpg'
   };
   public showImage = false;
   public btnText = 'Show';
   public TogglePreview() {
      this.showImage = (this.showImage==false)?true:false;
      this.btnText = (this.btnText=='Show')?'Hide':'Show';
   }
}
```

2. ifdemo.component.html

```
<div class="container-fluid">
```

```
<h2>Product Details</h2>
<div class="row">
  <div class="col-3">
      <dl>
          <dt>Name</dt>
          <dd>{{product.Name}}</dd>
          <dt>Price</dt>
          <dd>{{product.Price}}</dd>
      </dl>
      <button (click)="TogglePreview()" class="btn btn-primary btn-sm btn-
block">{{btnText}} Preview</button>
  </div>
  <div class="col-9">
    <div *ngIf="showImage">
        <img [src]="product.Photo" height="200" width="200" >
    </div>
  </div>
</div>
</div>
```

ngIF directive

- ngIf with "then and else" block

```
if condition then
    statement if true;
```

else
  statement if false;
Syntax:
<div *ngIf=”condition; then thenBlockId else elseBlockId”> </div>
<ng-template #thenBlockId>content to display when true </ng-template>
<ng-template #elseBlockId>content to display when false </ng-template>

- Angular Containers can handle your interactions dynamically without effecting the DOM structure.
- Angular provides several containers like
  - <ng-template> </ng-template>
  - <ng-container> </ng-container>
- *ngIf is used with HTML elements.
- You can use property binding technique when you are working with Angular containers.

  <ng-template [ngIf]=”Boolean”> </ng-template>

Ex:

1. Ifdemo.component.ts
   public instock = true;
2. Ifdemo.component.html

```
<div *ngIf="instock; then thenBlock else elseBlock">
  </div>
  <ng-template #thenBlock>
    Available
  </ng-template>
 <ng-template #elseBlock>
    Out of Stock
 </ng-template>
</div>
```

Ex:

1. Ifdemo.component.ts

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

@Component({
  selector: 'app-ifdemo',
  templateUrl: './ifdemo.component.html',
  styleUrls: ['./ifdemo.component.css']
})
export class IfdemoComponent {
   public instock = true;
   public product = {
     Name: 'Nike Casuals',
     Price: 4500.55,
     Photo: 'assets/shoe.jpg'
   };
   public showImage = false;
   public btnText = 'Show';
   public TogglePreview() {
      this.showImage = (this.showImage==false)?true:false;
      this.btnText = (this.btnText=='Show')?'Hide':'Show';
   }
}
```

2. Ifdemo.component.html

```
<div class="container-fluid">
```

```
<h2>Product Details</h2>
<div class="row">
  <div class="col-3">
      <dl>
          <dt>Name</dt>
          <dd>{{product.Name}}</dd>
          <dt>Price</dt>
          <dd>{{product.Price}}</dd>
      </dl>
      <button (click)="TogglePreview()" class="btn btn-primary btn-sm btn-
block">{{btnText}} Preview</button>
  </div>
  <div class="col-9">
    <div *ngIf="showImage; then thenBlock else elseBlock">
    </div>
    <ng-template #thenBlock>
      <img [src]="product.Photo" height="200" width="200">
    </ng-template>
    <ng-template #elseBlock>
      <img alt="Click Preview" height="200" width="200" >
    </ng-template>
  </div>
</div>
</div>
```

NgSwitch Directive

- It is a switch selector in UI.
- It can display only the container that is required for specific situation.
- It can allow to change the display dynamically.
- You can define multiple containers in a page. NgSwitch will select only the container that is required for the situation and removes other containers from DOM.
- Switch block is defined by using “ngSwitch”
- Case block is defined by using “ngSwitchCase”
- Default block is defined by using “ngSwitchDefault”

Syntax:

```
<main-container [ngSwitch]=”value/expression”>
   <child-container  *ngSwitchCase=”1”> </child-container>
   <child-container  *ngSwitchCase=”2”> </child-container>
   <child-container *ngSwitchDefault> </child-container>
</main-container>
```

```
<div [ngSwitch]="3">
    <div *ngSwitchCase="1">
      One
    </div>
    <div *ngSwitchCase="2">
      Two
    </div>
```

```
  <div *ngSwitchDefault>
   Please Enter one or two
  </div>
 </div>
```

Ex:

1. Switchdemo.component.ts

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

@Component({
  selector: 'app-switchdemo',
  templateUrl: './switchdemo.component.html',
  styleUrls: ['./switchdemo.component.css']
})
export class SwitchdemoComponent{
   public product = {
     Name: 'Nike Casuals',
     Price: 4500.55,
     Photo: 'assets/shoe.jpg',
     Description: 'Something about Nike Casuals...'
   };
   public selectedView = 'info';
   public views = ['info', 'preview', 'more'];
  public count = 0;
  public ChangeView(obj) {
    this.selectedView = obj.target.name;
  }
  public NextClick(){
   this.count++;
   this.selectedView = this.views[this.count];
  }
  public PrevClick(){
   this.count--;
   this.selectedView = this.views[this.count];
  }
}
```

2. Switchdemo.component.html

```
<div class="container-fluid">
 <h2 class="text-primary text-center"><span class="fa fa-shopping-cart"></span>Amazon Shopping</h2>
 <div class="btn-toolbar bg-danger justify-content-between">
  <div class="btn-group">
    <button (click)="ChangeView($event)" name="info" class="btn btn-danger">Basic Details</button>
    <button (click)="ChangeView($event)" name="preview" class="btn btn-danger">Preview</button>
    <button (click)="ChangeView($event)" name="more" class="btn btn-danger">More..</button>
  </div>
  <div class="btn-group">
    <button (click)="PrevClick()" class="btn btn-danger"><span class="fa fa-
chevron-circle-left"></span></button>
    <button (click)="NextClick()" class="btn btn-danger"><span class="fa fa-
chevron-circle-right"></span></button>
  </div>
 </div>
 <div class="row" style="margin: 20px;">
    <!--Main Container-->
  <div [ngSwitch]="selectedView">
    <!--info card-->
    <div class="card" *ngSwitchCase="'info'">
        <div class="card-header">
            <h2>{{product.Name}}</h2>
        </div>
        <div class="card-body">
            <h4>{{product.Price}}</h4>
        </div>
    </div>
    <!--Preview card-->
    <div class="card" *ngSwitchCase="'preview'">
        <div class="card-body">
            <img [src]="product.Photo" width="200" height="200" >
        </div>
        </div>
        <!--Description Card-->
        <div class="card" *ngSwitchCase="'more'">
          <div class="card-header">
            <h3>Description</h3>
          </div>
          <div class="card-body">
            <p>{{product.Description}}</p>
          </div>
        </div>
      </div>
     </div>
    </div>
```

NgFor

- It is a repeater.
- It repeats HTML elements based on an Iterator.
- It uses "of" operator to read values from a collection and generate an HTML element for every value.

  Syntax:
  <div *ngFor="let item of collection"> </div>

Ex:

1. fordemo.component.ts

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

@Component({
  selector: 'app-fordemo',
  templateUrl: './fordemo.component.html',
  styleUrls: ['./fordemo.component.css']
})
export class FordemoComponent {
  public categories = ['Electronics', 'Footwear', 'Fashion'];
  public menudata = [
    {Category: 'Electronics', Products: ['Samsung TV', 'JBL Speaker']},
    {Category: 'Footwear', Products: ['Nike Casuals', 'Lee Cooper Boot']}
  ];
}
```

2. fordemo.component.html

```
<div class="container-fluid">
 <div class="row">
  <div class="col-3">
    <h3>Categories</h3>
    <ol>
        <li *ngFor="let item of categories">{{item}}</li>
    </ol>
  </div>
  <div class="col-3">
    <h3>Categories</h3>
    <select class="form-control">
        <option *ngFor="let item of categories">
            {{item}}
        </option>
    </select>
  </div>
  <div class="col-3">
    <h3>Categories</h3>
    <table class="table table-hover">
      <tbody>
          <tr *ngFor="let item of categories">
              <td><a href="#">{{item}}</a></td>
          </tr>
      </tbody>
    </table>
  </div>
 </div>
 <div class="row" style="margin-top: 20px;">
    <div class="col-3">
        <h3>Menu</h3>
        <ol>
            <li *ngFor="let item of menudata">
            {{item.Category}}
            <ol type="a">
               <li *ngFor="let product of item.Products">
                  {{product}}
               </li>
            </ol>
         </li>
      </ol>
   </div>
   <div>
      <h3>Menu</h3>
      <select class="form-control">
         <optgroup *ngFor="let item of menudata" label="{{item.Category}}">
            <option *ngFor="let product of item.Products">
               {{product}}
            </option>
         </optgroup>
      </select>
   </div>
   <div class="col-3">
      <h3>Menu</h3>
      <div *ngFor="let item of menudata">
         <details>
            <summary>{{item.Category}}</summary>
```

```
                <ol>
                    <li *ngFor="let product of item.Products">
                        {{product}}
                    </li>
                </ol>
            </details>
        </div>
    </div>
 </div>
</div>
```

Property	Type	Description
index	Number	Returns the iterator index number. So that you can identify the position of iterating element.
even	Boolean	Returns true if the iterator item is at even occurrence.
odd	Boolean	Returns true if the iterator item is at odd occurrence.
first	Boolean	Returns true if the iterator item is the first item.
last	Boolean	Returns true if the iterator item is the last item.
trackBy	function	It identifies the changes in iterator. So that iterator will be performing iteration over collection only when change occurred.

Syntax:

<div *ngFor=”let item of collection; let i=index; let e=even; let o=odd”> </div>

{{i}} => Returns index number

Download and Configure Fontawesome:

1. https://fontawesome.com/how-to-use/on-the-web/setup/hosting-font-awesome-yourself
2. Download for Web
3. Extract and copy its folders
4. Go to Your workspace “node_modules”
5. Create a new folder by name “fonts” and paste the copied files
6. Import into “styles.css” folder
   @import "../../../node_modules/fonts/css/all.css";

Shoppingcart.component.ts

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


@Component({
  selector: 'app-shoppingcart',
  templateUrl: './shoppingcart.component.html',
  styleUrls: ['./shoppingcart.component.css']
})
export class ShoppingcartComponent {
 public categories = ['Select a Category', 'Electronics', 'Footwear', 'Fashion'];
 public electronics = ['Select Electronics', 'JBL Speaker', 'Earpods'];
 public footwear = ['Select Footwear', 'Nike Casuals', 'Lee Cooper Boot'];
 public fashion = ['Select Fashion', 'Shirt', 'Jeans'];
 public data = [
  {Name: 'JBL Speaker', Price: 4500.55, Photo: 'assets/jblspeaker.jpg'},
  {Name: 'Earpods', Price: 3000.44, Photo: 'assets/earpods.jpg'},
  {Name: 'Nike Casuals', Price: 6000.44, Photo: 'assets/shoe.jpg'},
  {Name: 'Lee Cooper Boot', Price: 2000.44, Photo: 'assets/shoe1.jpg'},
  {Name: 'Shirt', Price: 1000.44, Photo: 'assets/shirt.jpg'},
  {Name: 'Jeans', Price: 4000.44, Photo: 'assets/jeans.jpg'},
 ];
 public products = [];
 public selectedCategoryName = 'Select a Category';
 public selectedProductName;
 public searchResults = [];
 public searchedProduct = {
   Name: '',
   Price: 0,
   Photo: ''
 };
 public cartItems = [];
 public cartItemsCount = 0;
 public showCart = false;
 public GetCartItemsCount(){
   this.cartItemsCount = this.cartItems.length;
 }
 public OnCategoryChange(){
   switch(this.selectedCategoryName)
   {
     case 'Electronics':
       this.products = this.electronics;
       break;
     case 'Footwear':
       this.products = this.footwear;
       break;
     case 'Fashion':
       this.products = this.fashion;
       break;
     default:
       this.products = ['Select a Category'];
       break;
   }
 }
 public onProductChanged(){
     this.searchResults = this.data.filter(x=>x.Name==this.selectedProductName);
     this.searchedProduct = {
       Name: this.searchResults[0].Name,
       Price: this.searchResults[0].Price,
       Photo: this.searchResults[0].Photo
     };
 }
 public AddToCartClick() {
    this.cartItems.push(this.searchedProduct);
```

```
    alert('Item Added to Cart');
    this.GetCartItemsCount();
 }
 public ToggleCartDisplay() {
   this.showCart = (this.showCart==false)?true:false;
 }
 public DeleteCartItem(index){
    let confirmDelete = confirm('Are you sure, want to Delete?');
    if(confirmDelete==true) {
      this.cartItems.splice(index, 1);
      this.GetCartItemsCount();
    }
 }
}
```

Shoppingcart.component.html

```
<div class="container-fluid">
  <h2 class="text-center text-primary"><span class="fa fa-shopping-
cart"></span>Amazon - Shopping</h2>
  <div class="row">
    <div class="col-3">
        <div class="form-group">
            <label>Select a Category</label>
            <div>
```

```
                <select (change)="OnCategoryChange()"
[(ngModel)]="selectedCategoryName" class="form-control">
                    <option *ngFor="let item of categories">
                        {{item}}
                    </option>
                </select>
            </div>
        </div>
        <div class="form-group">
            <label>Select a Product</label>
            <div>
                <select (change)="onProductChanged()"
[(ngModel)]="selectedProductName" class="form-control">
                    <option *ngFor="let item of products">
                        {{item}}
                    </option>
                </select>
            </div>
        </div>
        <div class="form-group">
            <label>Preview</label>
            <div class="card">
                <div class="card-header">
                    <h3>{{searchedProduct.Name}}</h3>
                </div>
                <div class="card-body text-center">
                    <img [src]="searchedProduct.Photo" width="200" height="200">
                </div>
                <div class="card-footer text-center">
                    <h3>{{searchedProduct.Price | currency:'INR'}}</h3>
                    <button (click)="AddToCartClick()" class="btn btn-danger btn-
block"> <span class="fa fa-shopping-cart"></span>Add to Cart</button>
                </div>
            </div>
        </div>
    </div>
    <div class="col-6">
        <div>
            <table *ngIf="showCart" style="margin-top: 100px;" class="table table-
hover">
                <caption>Your Cart Items</caption>
                <thead>
                    <tr>
                        <th>Name</th>
                        <th>Price</th>
                        <th>Preview</th>
                    </tr>
                </thead>
                <tbody>
                    <tr *ngFor="let item of cartItems; let i=index">
                    <td>{{item.Name}}</td>

                    <td>{{item.Price}}</td>

                    <td><img width="50" height="50" [src]="item.Photo"></td>

                    <td>

                       <button (click)="DeleteCartItem(i)" class="btn btn-outline-
danger"> <span class="fa fa-trash"></span> </button>

                    </td>

                 </tr>

              </tbody>

           </table>

        </div>

     </div>

     <div class="col-3">

        <div>

           <button (click)="ToggleCartDisplay()" class="btn btn-danger btn-
block"><span class="fa fa-shopping-cart"></span> [{{cartItemsCount}}] Your
Cart Items</button>

        </div>


     </div>

   </div>

</div>
```

Ex:

1. Likesdemo.component.ts

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

@Component({
  selector: 'app-likesdemo',
  templateUrl: './likesdemo.component.html',
  styleUrls: ['./likesdemo.component.css']
})
export class LikesdemoComponent{
   public products = [
     {Name: 'JBL Speaker', Photo: 'assets/jblspeaker.jpg', Likes: 0, Dislikes: 0},
     {Name: 'Nike Casuals', Photo: 'assets/shoe.jpg', Likes: 0, Dislikes: 0},
     {Name: 'Shirt', Photo: 'assets/shirt.jpg', Likes: 0, Dislikes: 0}
   ];
   public LikesCounter(item){
       item.Likes++;
   }
   public DislikesCounter(item){
        item.Dislikes++;
   }
}
```

2. Likesdemo.component.html

```
<div class="container-fluid">
 <h2>Products Catalog</h2>
 <div class="card-deck">
   <div class="card" *ngFor="let item of products">
      <div class="card-header">
         <h3>{{item.Name}}</h3>
      </div>
      <div class="card-body">
         <img width="200" height="200" [src]="item.Photo" >
      </div>
      <div class="card-footer">
        <div class="btn-group">
         <button (click)="LikesCounter(item)" class="btn btn-
success">[{{item.Likes}}]<span class="fa fa-thumbs-up"></span>
Like(s)</button>
         <button (click)="DislikesCounter(item)" class="btn btn-
danger">[{{item.Dislikes}}]<span class="fa fa-thumbs-
down">Dislike(s)</span></button>
        </div>
      </div>
   </div>
</div>
</div>
```

Iteration even, odd, first, last occurrence

1. Iterationdemo.component.ts

```
public products = [
    {Name: 'JBL Speaker', Photo: 'assets/jblspeaker.jpg', Likes: 0, Dislikes: 0},
    {Name: 'Nike Casuals', Photo: 'assets/shoe.jpg', Likes: 0, Dislikes: 0},
    {Name: 'Shirt', Photo: 'assets/shirt.jpg', Likes: 0, Dislikes: 0},
    {Name: 'Jeans', Photo: 'assets/jeans.jpg', Likes: 0, Dislikes: 0}
  ];
```

2. Iterationdemo.component.html

```
<div class="row">
  <table class="table table-hover">
    <thead>
        <tr>
            <th>Name</th>
            <th>Photo</th>
            <th>Likes</th>
            <th>Dislikes</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 f=first;
let l=last; let e=even; let o=odd">
            <td>{{item.Name}}</td>
            <td><img [src]="item.Photo" width="50" height="50"></td>
            <td>{{item.Likes}}</td>
            <td>{{item.Dislikes}}</td>
            <td>{{f}}</td>
            <td>{{l}}</td>
            <td>{{e}}</td>
            <td>{{o}}</td>
        </tr>
    </tbody>
  </table>
</div>
```

Iteration - TrackBy

Ex:

3. Likesdemo.component.ts

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

@Component({
  selector: 'app-likesdemo',
  templateUrl: './likesdemo.component.html',
  styleUrls: ['./likesdemo.component.css']
})
export class LikesdemoComponent{
   public products = [
     {Name: 'JBL Speaker', Photo: 'assets/jblspeaker.jpg', Likes: 0, Dislikes: 0},
     {Name: 'Nike Casuals', Photo: 'assets/shoe.jpg', Likes: 0, Dislikes: 0},
     {Name: 'Shirt', Photo: 'assets/shirt.jpg', Likes: 0, Dislikes: 0}
   ];
   public LikesCounter(item){
      item.Likes++;
   }
   public DislikesCounter(item){
       item.Dislikes++;
   }
}
```

4. Likesdemo.component.html

```
<div class="container-fluid">
 <h2>Products Catalog</h2>
 <div class="card-deck">
   <div class="card" *ngFor="let item of products">
      <div class="card-header">
         <h3>{{item.Name}}</h3>
      </div>
      <div class="card-body">
         <img width="200" height="200" [src]="item.Photo" >
      </div>
      <div class="card-footer">
        <div class="btn-group">
```

```
            <button (click)="LikesCounter(item)" class="btn btn-
success">[{{item.Likes}}]<span class="fa fa-thumbs-up"></span>
Like(s)</button>

            <button (click)="DislikesCounter(item)" class="btn btn-
danger">[{{item.Dislikes}}]<span class="fa fa-thumbs-
down">Dislike(s)</span></button>

           </div>

        </div>

    </div>

</div>

</div>
```

Iteration even, odd, first, last occurrence

3. Iterationdemo.component.ts

```
public products = [
     {Name: 'JBL Speaker', Photo: 'assets/jblspeaker.jpg', Likes: 0, Dislikes: 0},
     {Name: 'Nike Casuals', Photo: 'assets/shoe.jpg', Likes: 0, Dislikes: 0},
     {Name: 'Shirt', Photo: 'assets/shirt.jpg', Likes: 0, Dislikes: 0},
     {Name: 'Jeans', Photo: 'assets/jeans.jpg', Likes: 0, Dislikes: 0}
   ];
```

4. Iterationdemo.component.html

```
<div class="row">
  <table class="table table-hover">
    <thead>
        <tr>
```

```
            <th>Name</th>
            <th>Photo</th>
            <th>Likes</th>
            <th>Dislikes</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 f=first;
let l=last; let e=even; let o=odd">
            <td>{{item.Name}}</td>
            <td><img [src]="item.Photo" width="50" height="50"></td>
            <td>{{item.Likes}}</td>
            <td>{{item.Dislikes}}</td>
            <td>{{f}}</td>
            <td>{{l}}</td>
            <td>{{e}}</td>
            <td>{{o}}</td>
        </tr>
    </tbody>
  </table>
```

```
</div>
```

Iteration – TrackBy

- Iteration in UI is controlled by “ngFor”
- It performs iteration over all elements every time when ever requested.
- We want “ngFor” to identify the changes occurred in iterations counter and update the changes without iterating all elements again.

Ex: Default Tracking change

1. Trackdemo.component.ts

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

@Component({
  selector: 'app-trackdemo',
  templateUrl: './trackdemo.component.html',
  styleUrls: ['./trackdemo.component.css']
})
export class TrackdemoComponent {
  public products = [
    {ProductId: 1, Name: 'Samsung TV', Price: 45000.44},
    {ProductId: 2, Name: 'Mobile', Price: 15000.44},
    {ProductId: 3, Name: 'Nike Casuals', Price: 5000.44},
  ];

  public productname;
  public productprice;
  public newProduct = {
    ProductId : 0,
    Name: '',
    Price: 0
  };
  public AddProduct() {
    this.newProduct = {
      ProductId: this.products.length + 1,
      Name: this.productname,
      Price: this.productprice
    };
    this.products.push(this.newProduct);
    alert('Record Inserted');
    this.productname = '';
    this.productprice = '';
  }

}
```

2. Trackdemo.component.html

```
<div class="container-fluid">
 <div class="row">
    <div class="col-3">
        <h3>Register Product</h3>
        <div class="form-group">
            <label>Name</label>
            <div>
                <input [(ngModel)]="productname" type="text" class="form-
control">
            </div>
        </div>
        <div class="form-group">
            <label>Price</label>
            <div>
                <input [(ngModel)]="productprice" type="text" class="form-
control">
            </div>
        </div>
        <div class="form-group">
            <button (click)="AddProduct()" class="btn btn-primary btn-block">Add
Product</button>
        </div>
    </div>
    <div class="col-9">
        <h2 class="text-center">Details</h2>
        <table class="table table-hover">
            <thead>
                <tr>
                    <th>Product Id</th>
                    <th>Name</th>
                    <th>Price</th>
                </tr>
            </thead>
            <tbody>
                <tr *ngFor="let product of products">
                    <td>{{product.ProductId}}</td>
                    <td>{{product.Name}}</td>
                    <th>{{product.Price}}</th>
                </tr>
            </tbody>
        </table>
    </div>
   </div>
</div>
```

Ex: TrackBy

Trackdemo.component.ts

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

@Component({
  selector: 'app-trackdemo',
  templateUrl: './trackdemo.component.html',
  styleUrls: ['./trackdemo.component.css']
```

```
})
export class TrackdemoComponent {
  public products = [
    {ProductId: 1, Name: 'Samsung TV', Price: 45000.44},
    {ProductId: 2, Name: 'Mobile', Price: 15000.44},
    {ProductId: 3, Name: 'Nike Casuals', Price: 5000.44},
  ];

  public productname;
  public productprice;
  public newProduct = {
    ProductId : 0,
    Name: '',
    Price: 0
  };
  public AddProduct() {
    this.newProduct = {
      ProductId: this.products.length + 1,
      Name: this.productname,
      Price: this.productprice
    };
    this.products.push(this.newProduct);
    alert('Record Inserted');
    this.productname = '';
```

```
     this.productprice = '';
    }
    public AddProductUsingApi(){
     this.products = [
       {ProductId: 1, Name: 'Samsung TV', Price: 45000.44},
       {ProductId: 2, Name: 'Mobile', Price: 15000.44},
       {ProductId: 3, Name: 'Nike Casuals', Price: 5000.44},
       {ProductId: 4, Name: 'Watch', Price: 15000.44},
     ];
    }
    public TrackChange(index, product) {
     return product.ProductId;
    }
  }
```

Trackdemo.component.html

```
<div class="container-fluid">
 <div class="row">
    <div class="col-3">
        <h3>Register Product</h3>
        <div class="form-group">
            <label>Name</label>
            <div>
                <input [(ngModel)]="productname" type="text" class="form-control">
            </div>
        </div>
        <div class="form-group">
            <label>Price</label>
            <div>
                <input [(ngModel)]="productprice" type="text" class="form-control">
            </div>
        </div>
        <div class="form-group">
            <button (click)="AddProduct()" class="btn btn-primary btn-block">Add
Product</button>
        </div>
    </div>
    <div class="col-9">
        <h2 class="text-center">Details <button
(click)="AddProductUsingApi()">Add using API</button></h2>
        <table class="table table-hover">
            <thead>
                <tr>
                    <th>Product Id</th>
                    <th>Name</th>
                    <th>Price</th>
                </tr>
            </thead>
            <tbody>
```

```
                <tr *ngFor="let product of products; trackBy: TrackChange">
                    <td>{{product.ProductId}}</td>
                    <td>{{product.Name}}</td>
                    <th>{{product.Price}}</th>
                </tr>
            </tbody>
        </table>
    </div>
 </div>
</div>
```

Iterations and Conditions

- NgFor is for iterations
- NgIf is for conditions
- You can handle conditions within Iterations.
- A single HTML element can’t have multiple template bindings, you have to use angular container [“ng-container”].

EX:

1. Conditions.component.ts

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

@Component({
  selector: 'app-conditions',
  templateUrl: './conditions.component.html',
  styleUrls: ['./conditions.component.css']
})
export class ConditionsComponent {
  public products = [
    {Name: 'Earpods', Price: 4500.55, Photo: 'assets/earpods.jpg', Category:
'Electronics'},
    {Name: 'Nike Casuals', Price: 6000.54, Photo: 'assets/shoe.jpg', Category:
'Footwear'},
    {Name: 'JBL Speaker', Price: 2000.54, Photo: 'assets/jblspeaker.jpg',
Category: 'Electronics'},
    {Name: 'Lee Cooper Boot', Price: 4000.54, Photo: 'assets/shoe1.jpg',
Category: 'Footwear'},
    {Name: 'Shirt', Price: 1000.54, Photo: 'assets/shirt.jpg', Category:
'Fashion'},
    {Name: 'Jeans', Price: 3000.54, Photo: 'assets/jeans.jpg', Category:
'Fashion'},
  ];
  public categories = ['All', 'Electronics', 'Footwear', 'Fashion'];
  public selectedCategoryName = 'All';
}
```

2. Conditions.component.html

```
<div class="container-fluid">
    <div class="row">
        <div class="col-3">
          <div class="form-group">
            <label>Select a Category</label>
            <div>
                <select [(ngModel)]="selectedCategoryName" class="form-
control">
                    <option *ngFor="let item of categories">
                        {{item}}
                    </option>
                </select>
            </div>
          </div>
        </div>
        <div class="col-9">
```

```
            <div class="card-deck">
                <ng-container  *ngFor="let item of products">
                    <div class="card" *ngIf="selectedCategoryName=='All' ||
selectedCategoryName==item.Category">
                        <div class="card-header">
                            <h3>{{item.Name}}</h3>
                        </div>
                        <div class="card-body">
                            <img [src]="item.Photo" width="150" height="150">
                        </div>
                        <div class="card-footer">
                            <h3>{{item.Price | currency:'INR'}}</h3>
                        </div>
                    </div>
                </ng-container>
            </div>
        </div>
    </div>
</div>
```

Using Radios instead of DropDown:

Ex: Individual Radio

```
<div class="form-group">
            <label>Select a Category</label>
            <div>
                <ul style="font-size: 30px;" class="list-unstyled">
                    <li><input [(ngModel)]="selectedCategoryName" type="radio"
value="All" name="categories">All</li>
                    <li><input [(ngModel)]="selectedCategoryName"  type="radio"
value="Electronics" name="categories">Electronics</li>
                    <li><input [(ngModel)]="selectedCategoryName"  type="radio"
value="Footwear" name="categories">Footwear</li>
                    <li><input [(ngModel)]="selectedCategoryName"  type="radio"
value="Fashion" name="categories">Fashion</li>
```

```
                    </ul>
                  </div>
                 </div>
```

Ex: Iteration over radios

```
<div class="form-group">
          <label>Select a Category</label>
          <div>
              <ul style="font-size: 30px;" class="list-unstyled">
                <li *ngFor="let item of categories">
                  <input [(ngModel)]="selectedCategoryName" type="radio"
[value]="item" name="categories"> {{item}}
                </li>
              </ul>
          </div>
        </div>
```

Ex: Buttons Navigation

- Add following method into ".ts"
```
public MenuSelection(e){
      this.selectedCategoryName = e.target.name;
  }
```
- ".html"
```
<div class="form-group">
              <label>Select a Category</label>
              <div>
                <button name="All"
(click)="MenuSelection($event)">All</button>
                <button name="Electronics"
(click)="MenuSelection($event)">Electronics</button>
```

```
                <button name="Footwear"
(click)="MenuSelection($event)">Footwear</button>
                <button name="Fashion"
(click)="MenuSelection($event)">Fashion</button>
              </div>
            </div>
```

Attribute Directives

- An attribute directive allows to extend HTML element.
- It makes HTML more declarative.
- Makes the static DOM element into a dynamic DOM element.
- Angular attribute directives are
    - NgModel
    - NgClass
    - NgStyle

NgModel:

- It is an attribute directive that extends HTML element and configures as a dynamic element.
- NgModel defines a model reference for HTML element so that its value can be stored and used dynamically.

Ex:

```
<input type=”text” [(ngModel)]=”username”>
```

NgClass:

- It is an attribute directive which is used to assign a CSS class dynamically to any element.
- It can change the appearance of HTML element dynamically.

- You can apply any CSS dynamically to HTML element by using 3 reference styles
    - String Reference
    - Array Reference
    - Object Reference
- String Reference
  You can define any one CSS class for element.
  Syntax:
  <div [ngClass]=" 'cssClassName' "> Your Text </div>

Ex:

1. Classbinding.component.css

```
.effects {
  border:2px solid darkcyan;
  box-shadow: 2px 3px 4px darkcyan;
  background-color: yellow;
  text-align: center;
  padding: 10px;
}
```

2. Classbinding.component.html

```
<div class="container-fluid">
 <div>
  <h1 [ngClass]="'effects'">Class Binding in Angular</h1>
 </div>
</div>
```

- Array Reference
  You can define multiple CSS classes for element.

Syntax:
```
<div [ngClass]=”[ ‘class1’, ‘class2’]”> Your Text </div>
```

Ex:

- Classbinding.component.css

```
.borderEffects {
   border:2px solid darkcyan;
   box-shadow: 2px 3px 4px darkcyan;
}
.backgroundEffects {
   background-color: yellow;
}
.textEffects {
   text-align: center;
   padding: 10px;
}
```

- Classbinding.component.html

```
<div class="container-fluid">
  <div>
    <h1 [ngClass]="['borderEffects', 'textEffects', 'backgroundEffects']">Class
Binding in Angular</h1>
  </div>
</div>
```

Ex:

- Classdemo.component.ts

```
export class ClassbindingComponent {
  public effects;
}
```

- Classdemo.component.html

```
<div class="container-fluid">
  <div>
    <div style="margin:10px">
        <input [(ngModel)]="effects" type="text" class="form-control"
placeholder="eg: borderEffects, textEffects, backgroundEffects">
    </div>
    <h1 [ngClass]="effects">Class Binding in Angular</h1>
  </div>
</div>
```

- Classdemo.component.css

```
.borderEffects {
    border:2px solid darkcyan;
    box-shadow: 2px 3px 4px darkcyan;
}
.backgroundEffects {
    background-color: yellow;
}
.textEffects {
    text-align: center;
    padding: 10px;
}
```

Object Reference

- Object defines a set of properties which you can turn ON or OFF dynamically.
- Instead of adding and removing the CSS classes, you can ON or OFF the classes.

  Syntax:
  <div [ngClass]="{className:true/false, className:true/false}"> Your Text </div>

Ex:

- Classdemo.component.ts

```
export class ClassbindingComponent {
  public ONTextEffects = false;
  public ONBorderEffects = false;
  public ONBackgroundEffects = false;
}
```

- Classdemo.component.html

```
<div class="container-fluid">
  <div>
    <div style="margin:10px">
        <ul class="list-unstyled">
            <li><input [(ngModel)]="ONTextEffects" type="checkbox"> Text
Effects</li>
            <li><input [(ngModel)]="ONBackgroundEffects" type="checkbox">
Background Effects</li>
            <li><input [(ngModel)]="ONBorderEffects" type="checkbox">
Border Effects </li>
        </ul>
    </div>
    <h1 [ngClass]="{borderEffects:ONBorderEffects,
textEffects:ONTextEffects,
backgroundEffects:ONBackgroundEffects}">Class Binding in
Angular</h1>
  </div>
</div>
```

- Classdemo.component.css

```
.borderEffects {
border:2px solid darkcyan;
box-shadow: 2px 3px 4px darkcyan;
}
.backgroundEffects {
background-color: yellow;
}
.textEffects {
text-align: center;
padding: 10px;
}
```

Note: The class names in object type reference must be configure as a string if they comprises of special characters.

Ex:

1. Classdemo.component.ts

```
import { Component, OnInit } from '@angular/core';
import { flatten } from '@angular/compiler';
```

```
@Component({
  selector: 'app-classbinding',
  templateUrl: './classbinding.component.html',
  styleUrls: ['./classbinding.component.css']
})
export class ClassbindingComponent {
   public txtEven;
   public applyErrorStyle = false;
   public applyValidStyle = false;
   public OnBlur() {
      if(this.txtEven %2 == 0) {
        this.applyValidStyle = true;
        this.applyErrorStyle = false;
      } else {
        this.applyValidStyle = false;
        this.applyErrorStyle = true;
      }   }}
```

2. Classdemo.component.html

```
<div class="container-fluid">
  <div class="form-group">
    <label>Enter Even Number</label>
    <div>
```

```
        <input [ngClass]="{errorStyle:applyErrorStyle, validStyle:applyValidStyle}"
(blur)="OnBlur()" [(ngModel)]="txtEven" type="text" class="form-control">
    </div>
  </div>
</div>
```

3. Classdemo.component.css

```
.errorStyle {
    border:2px solid red;
    box-shadow: 2px 3px 4px red;
}
.validStyle {
    border:2px solid green;
    box-shadow: 2px 3px 4px green;
}
```

Attribute Directives

- An attribute directive allows to extend HTML element.
- It makes HTML more declarative.
- Makes the static DOM element into a dynamic DOM element.
- Angular attribute directives are
    - NgModel
    - NgClass
    - NgStyle

NgModel:

- It is an attribute directive that extends HTML element and configures as a dynamic element.
- NgModel defines a model reference for HTML element so that its value can be stored and used dynamically.

Ex:
<input type=”text” [(ngModel)]=”username”>

NgClass:

- It is an attribute directive which is used to assign a CSS class dynamically to any element.
- It can change the appearance of HTML element dynamically.
- You can apply any CSS dynamically to HTML element by using 3 reference styles
  - String Reference
  - Array Reference
  - Object Reference
- String Reference
  You can define any one CSS class for element.
  Syntax:
  <div [ngClass]=” ‘cssClassName‘ ”> Your Text </div>

Ex:

3. Classbinding.component.css
   .effects {

   border:2px solid darkcyan;

   box-shadow: 2px 3px 4px darkcyan;

   background-color: yellow;

   text-align: center;

```
    padding: 10px;
  }
```

4. Classbinding.component.html

```
<div class="container-fluid">
  <div>
    <h1 [ngClass]="'effects'">Class Binding in Angular</h1>
  </div>
</div>
```

- Array Reference
  You can define multiple CSS classes for element.
  Syntax:
  <div [ngClass]="['class1', 'class2']"> Your Text </div>

Ex:

- Classbinding.component.css

```
.borderEffects {
    border:2px solid darkcyan;
    box-shadow: 2px 3px 4px darkcyan;
}
.backgroundEffects {
    background-color: yellow;
}
.textEffects {
    text-align: center;
    padding: 10px;
}
```

- Classbinding.component.html

```
<div class="container-fluid">
  <div>
    <h1 [ngClass]="['borderEffects', 'textEffects', 'backgroundEffects']">Class
Binding in Angular</h1>
  </div>
</div>
```

Ex:

- Classdemo.component.ts

```
export class ClassbindingComponent {
  public effects;
}
```

- Classdemo.component.html

```
<div class="container-fluid">
  <div>
    <div style="margin:10px">
        <input [(ngModel)]="effects" type="text" class="form-control"
placeholder="eg: borderEffects, textEffects, backgroundEffects">
    </div>
    <h1 [ngClass]="effects">Class Binding in Angular</h1>
  </div>
</div>
```

- Classdemo.component.css

```
.borderEffects {
    border:2px solid darkcyan;
    box-shadow: 2px 3px 4px darkcyan;
}
.backgroundEffects {
    background-color: yellow;
}
.textEffects {
    text-align: center;
    padding: 10px;
}
```

Object Reference

- Object defines a set of properties which you can turn ON or OFF dynamically.
- Instead of adding and removing the CSS classes, you can ON or OFF the classes.

  Syntax:
  <div [ngClass]="{className:true/false, className:true/false}"> Your Text </div>

Ex:

- Classdemo.component.ts

```
export class ClassbindingComponent {
   public ONTextEffects = false;
   public ONBorderEffects = false;
   public ONBackgroundEffects = false;
}
```

- Classdemo.component.html

```
<div class="container-fluid">
  <div>
    <div style="margin:10px">
        <ul class="list-unstyled">
            <li><input [(ngModel)]="ONTextEffects" type="checkbox"> Text
Effects</li>
            <li><input [(ngModel)]="ONBackgroundEffects" type="checkbox">
Background Effects</li>
            <li><input [(ngModel)]="ONBorderEffects" type="checkbox">
Border Effects </li>
        </ul>
    </div>
    <h1 [ngClass]="{borderEffects:ONBorderEffects,
textEffects:ONTextEffects,
backgroundEffects:ONBackgroundEffects}">Class Binding in
Angular</h1>
  </div>
</div>
```

- Classdemo.component.css

```
.borderEffects {
border:2px solid darkcyan;
box-shadow: 2px 3px 4px darkcyan;
}
.backgroundEffects {
background-color: yellow;
}
.textEffects {
text-align: center;
padding: 10px;
}
```

Note: The class names in object type reference must be configure as a string if they comprises of special characters.

Ex:

4. Classdemo.component.ts

```
import { Component, OnInit } from '@angular/core';
import { flatten } from '@angular/compiler';


@Component({
  selector: 'app-classbinding',
  templateUrl: './classbinding.component.html',
  styleUrls: ['./classbinding.component.css']
})
export class ClassbindingComponent {
   public txtEven;
   public applyErrorStyle = false;
   public applyValidStyle = false;
   public OnBlur() {
      if(this.txtEven %2 == 0) {
        this.applyValidStyle = true;
        this.applyErrorStyle = false;
      } else {
        this.applyValidStyle = false;
        this.applyErrorStyle = true;
      }
   }
}
```

5. Classdemo.component.html

```
<div class="container-fluid">
  <div class="form-group">
    <label>Enter Even Number</label>
    <div>
        <input [ngClass]="{errorStyle:applyErrorStyle, validStyle:applyValidStyle}"
(blur)="OnBlur()" [(ngModel)]="txtEven" type="text" class="form-control">
    </div>
  </div>
</div>
```

6. Classdemo.component.css

```
.errorStyle {
    border:2px solid red;
    box-shadow: 2px 3px 4px red;
}
.validStyle {
    border:2px solid green;
```

```
  box-shadow: 2px 3px 4px green;
}
```

NgStyle

- It defines inline style for HTML element.
- Class binding uses styles from external style sheet, which will increase the number of requests to page and also page load time.
- NgStyle can define styles inline, which reduces the number of requests.
- If you are trying to configure effects dynamically to any specific element, without reusing the style then better go with Style Binding.
- Styles are maintained in a Style Object [TypeScript Object] and applied to the element.

Syntax:

```
<div [ngStyle]="styleObject"> </div>
```

Ex:

1. Styledemo.component.ts

```
export class StyledemoComponent {
  public styleObj = {};
  public onMouseMove(e) {
    this.styleObj = {
      'position': 'fixed',
      'top': e.clientY + 'px',
      'left': e.clientX + 'px'
    };
  }
}
```

2. Styledemo.component.html

```
<div (mousemove)="onMouseMove($event)" class="container-fluid">
    <div style="height:1000px">

    </div>
    <img [ngStyle]="styleObj" src="assets/flag.gif" width="50" height="50">
</div>
```

Ex:

1. Styledemo.component.ts

```
export class StyledemoComponent {
  public styleObj = {};
  public bgcolor;
  public forecolor;
  public align;
  public ApplyStylesClick() {
    this.styleObj = {
      'background-color': this.bgcolor,
      'color': this.forecolor,
      'text-align': this.align
    };
  }
}
```

2. Styledemo.component.html

```
<div class="container-fluid">
    <fieldset>
        <legend>Choose Effects</legend>
        <dl style="width: 300px; align-items: center; margin:auto">
            <dt>Background Color</dt>
            <dd>
                <select [(ngModel)]="bgcolor" class="form-control">
                    <option>Red</option>
                    <option>Yellow</option>
                    <option>Green</option>
                </select>
            </dd>
            <dt>Text Color</dt>
            <dd>
                <select [(ngModel)]="forecolor" class="form-control">
                    <option>White</option>
                    <option>Yellow</option>
                    <option>Red</option>
                </select>
            </dd>
            <dt>Text Align</dt>
            <dd>
                <input [(ngModel)]="align" type="radio" value="left"
name="align"> Left
                <input [(ngModel)]="align" type="radio" value="center"
name="align"> Center
                <input [(ngModel)]="align" type="radio" value="right"
name="align"> Right
            </dd>
            <button (click)="ApplyStylesClick()" class="btn btn-primary btn-
block">Apply Effects</button>
        </dl>

    </fieldset>
    <div class="form-group" style="margin-top: 20px;">
        <h1 [ngStyle]="styleObj">Sample Text</h1>
    </div>
</div>
```

Angular Provides Pre-Defined CSS Classes for Form Interactions

- Angular CSS classes can identify the state of Form and Input element. And can apply effects automatically when state is changed.

Class Name	Purpose
.ng-invalid	Apply effects when form or input field are invalid
.ng-valid	Apply effects when form or input field are valid

Ex:

1. Angularcss.component.css

```
input.ng-invalid {
   border: 2px solid red;
   box-shadow: 2px 3px 3px red;
}
input.ng-valid {
   border: 2px solid green;
   box-shadow: 2px 3px 3px green;
}
```

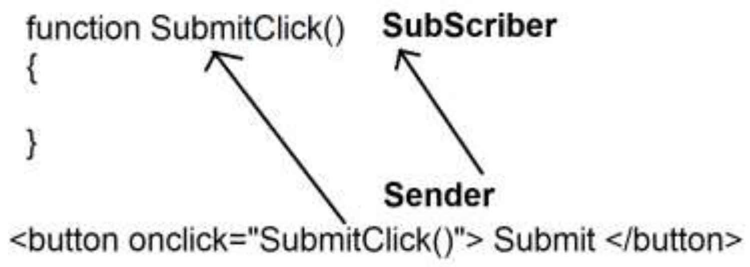
2. Angularcss.component.html

```
<div class="container-fluid">
  <h2>Register</h2>
  <div class="form-group">
    <label>User Name</label>
    <div>
        <input type="text" ngModel class="form-control" #txtName="ngModel"
name="txtName" required>
    </div>
  </div>
  <div class="form-group">
    <label>Email</label>
    <div>
```

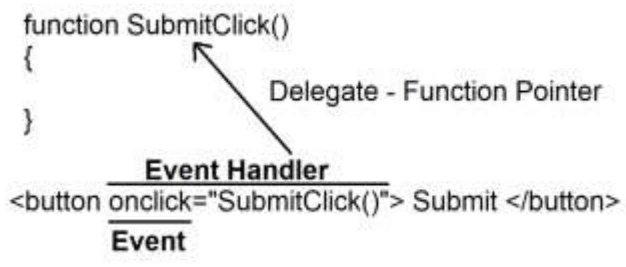
```
        <input type="email" ngModel class="form-control" #txtEmail="ngModel"
name="txtEmail" required>
    </div>
  </div>
</div>
```

Angular Event Binding

- Event is a message sent by sender to its subscriber in order to notify the change.
- Event follows a software design pattern called "Observer", which is a communication pattern. [Behavioural Patterns]



- Event uses an Event Handling that follows "Delegate" [Function Pointer] mechanism.


function SubmitClick()
{
}
Delegate - Function Pointer
Event Handler
<button onclick="SubmitClick()"> Submit </button>
Event

- Event handler requires definition for event arguments, even when you are not passing any argument you have define the memory for empty arguments.
- Event handler of JavaScript events can use 2 arguments
  - this : It can send information about the current object that is name, value, id, etc.
  - event : It can send information about the current event that is X position, which key pressed etc.

Ex:

```
<script>
 function InsertClick(object, e)
 {
   if(e.ctrlKey) {
     window.open("C:/Images/tv.jpg", "TV", "width=100 height=100");
   } else {
   document.write(object.name);
   }
 }
</script>
<p> Use Ctrl + Click to open window </p>
<button name="btnInsert" id="Insert" onclick="InsertClick(this, event)">
Insert </button>
```

- Angular uses all JavaScript Events. However angular can also use its traditional "Ng" Events.
- Angular Events are derived from the base of "EventEmitter" class.
- Angular allows to create custom events by implementing "EventEmitter".
- Angular uses all JavaScript events which are binded by using "( )". Event Binding

  Ex:

  ```
  <button  (click)="method()">
  ```
- Angular supports only "$event" as event argument. "this" is not allowed.
- By using the $event object you can access both object and event properties.

event.target.property (this)	: object properties
event.property	: event property

Angular Events

- Key Events
- Mouse Events
- Timer Events
- Miscellaneous Events

Key Event Binding

- You configure events to handle interactions based on keying of characters.
- The basic events for key binding are

Keyup	Specifies actions to perform when key is release over element.
KeyDown	Specifies action to perform when user hold down a key.
KeyPress	Specifies actions to perform when user finish a key and used another.

- Key Events are regularly used for a set of properties

keyCode	It returns the actual keycode. It is for every key on keyboard Ex: A=65, Z=90
charCode	It returns the character code as per UTF standards. Only for the keys that print some character.
which	It is similar to keycode but can support various keyboard layouts.
shiftKey	Returns true when shift is used.
altKey	Returns true when alt is used.
ctrlKey	Return true when ctrl is used.

Ex:

1. Keybinding.component.ts

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


@Component({
  selector: 'app-keybinding',
  templateUrl: './keybinding.component.html',
  styleUrls: ['./keybinding.component.css']
```

```
})
export class KeybindingComponent{
 public users = [
   {UserName: 'john'},
   {UserName: 'john12'},
   {UserName: 'john_nit'},
   {UserName: 'david'}
 ];
 public userName;
 public userMsg;
 public isUserValid = false;
 public isUserInvalid = false;

 public password;
 public showCapsWarning = false;

 public VerifyUserOnKeyUp(){
   if (this.userName.length < 3) {
     this.userMsg = 'User Name too short..';
     this.isUserInvalid = true;
     this.isUserValid = false;
   } else {
     for(var item of this.users) {
       if(item.UserName == this.userName) {
          this.userMsg = 'User Name Taken - Try Another';
          this.isUserInvalid = true;
          this.isUserValid = false;
          break;
         } else {
          this.userMsg = 'User Name Available';
          this.isUserInvalid = false;
          this.isUserValid = true;
         }
        }
       }
      }
      public VerifyPassword(e){
        if(e.keyCode>=65 && e.keyCode<=90) {
          this.showCapsWarning = true;
        } else {
         this.showCapsWarning = false;
        }
      }
     }
```

2. Keybinding.component.html

```
<div class="container-fluid">
 <h2>Register User</h2>
```

```
  <div class="form-group">
    <label>User Name</label>
    <div>
        <input [ngClass]="{'errorStyle':isUserInvalid, 'validStyle':isUserValid}"
(keyup)="VerifyUserOnKeyUp()" [(ngModel)]="userName" type="text"
class="form-control">
        <span [ngClass]="{'text-success':isUserValid, 'text-
danger':isUserInvalid}">{{userMsg}}</span>
    </div>
  </div>
  <div class="form-group">
    <label>Password</label>
    <div>
        <input (keypress)="VerifyPassword($event)" [(ngModel)]="password"
type="password" class="form-control">
        <div *ngIf="showCapsWarning">
            <span class="fa fa-exclamation-triangle text-warning"></span>
            <span class="text-warning">Warning CAPS is ON</span>
        </div>
    </div>
  </div>
</div>
```

3. Keybinding.component.css

```
.container-fluid {
    margin:auto;
    padding:20px;
    width:300px;
    align-items: center;
    justify-content: center;
}
.errorStyle {
    border:1px solid red;
    box-shadow: 2px 3px 4px red;
}
.validStyle {
    border:1px solid green;
    box-shadow: 2px 3px 4px green;
}
```

Mouse Event Binding

- Angular provides support for actions to perform based on mouse interactions.
- Angular can use all JavaScript mouse events for handling mouse interactions

Event	Description
mouseover	Specifies the actions to perform when mouse pointer is over the HTML element.
mouseout	Specifies the actions to perform when pointer is moved out of the element.
mousedown	Specify actions to perform when mouse button is down over any element.
mouseup	Specifies actions to perform when mouse button is released over any element.

mousemove	Specifies the action to perform while the mouse pointer is moving over any element.

- Mouse events depends on mouse properties like
  - client X – Gets x-axis position
  - clientY – Gets y-axis position

Ex: mouseover & mouseout

1. mousedemo.component.ts

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

@Component({
  selector: 'app-mousedemo',
  templateUrl: './mousedemo.component.html',
  styleUrls: ['./mousedemo.component.css']
})
export class MousedemoComponent {
   public styleObj = {
     'background-color': ''
   };
   public onMouseOver(e) {
      this.styleObj = {
        'background-color': e.target.id
      };
   }
   public onMouseOut() {
     this.styleObj = {
       'background-color': 'black'
     };
   }
}
```

2. mousedemo.component.html

```
<div [ngStyle]="styleObj" class="container-fluid" style="height: 1000px;">
```

```
 <h2>Color Panel</h2>
 <div (mouseover)="onMouseOver($event)" (mouseout)="onMouseOut()"
class="row" style="margin:30px; height: 100px; color:white">
   <div class="col-4" id="red" style="background-color: red;">
     Red
   </div>
   <div class="col-4" id="green" style="background-color: green;">
     Green
   </div>
   <div class="col-4" id="blue" style="background-color: blue;">
     Blue
   </div>
 </div>
 <div>
   <h1 [ngStyle]="styleObj" class="text-center">Sample Text</h1>
 </div>
</div>
```

3. mousedemo.component.css

```
div:hover {
    cursor:grab
}
```

Ex:

1. mousedemo.component.ts

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

@Component({
  selector: 'app-mousedemo',
  templateUrl: './mousedemo.component.html',
  styleUrls: ['./mousedemo.component.css']
})
export class MousedemoComponent {
    public onMouseOver(e){
      e.target.stop();
    }
    public onMouseOut(e){
      e.target.start();
    }
  }
```

2. mousedemo.component.html

```
<div class="container-fluid">
  <div class="row">
    <div class="col-3">
        <h2>Menu</h2>
    </div>
    <div class="col-6">
        <h2>Shopping</h2>
    </div>
    <div class="col-3">
        <h2>Offers</h2>
        <marquee (mouseover)="onMouseOver($event)"
(mouseout)="onMouseOut($event)" scrollamount="10" direction="up">
        <div>
            <img src="assets/jblspeaker.jpg" width="100" height="100">
            <div>
                <span>40% OFF</span>
            </div>
        </div>
        <div>
            <img src="assets/shoe.jpg" width="100" height="100">
            <div>
                <span>60% OFF</span>
            </div>
        </div>
        <div>
            <img src="assets/shirt.jpg" width="100" height="100">
            <div>
              <span>30% OFF</span>
            </div>
          </div>
         </marquee>
       </div>
      </div>
     </div>
```

Ex:

1. mousedemo.component.ts

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

@Component({
  selector: 'app-mousedemo',
  templateUrl: './mousedemo.component.html',
  styleUrls: ['./mousedemo.component.css']
})
export class MousedemoComponent {
   public offerImage = 'assets/giftbox.png';

   public adImage = 'assets/Pepsi.jpg';
   public Ad1() {
      this.adImage = 'assets/rDigital.jpg';
   }
   public Ad2(){
      this.adImage = 'assets/Pepsi.jpg';
   }
   public onMouseOver(e){
      e.target.stop();
   }
   public onMouseOut(e){
      e.target.start();
   }
```

```
    public onMouseDown(){
      this.offerImage = 'assets/offerbox.png';
    }
    public onMouseUp() {
      this.offerImage = 'assets/giftbox.png';
    }
  }
```

2. mousedemo.component.html

```
<div class="container-fluid">
  <div class="row" >
    <div class="text-center" style="margin:auto">
        <img (mouseover)="Ad1()" (mouseout)="Ad2()" [src]="adImage"
width="400" height="100">
    </div>
  </div>
  <div class="row">
    <div class="col-3">
        <h2>Menu</h2>
    </div>
    <div class="col-6">
        <h2>Shopping</h2>
        <div style="margin:auto">
            <img (mousedown)="onMouseDown()" (mouseup)="onMouseUp()"
[src]="offerImage" width="200" height="200">
        </div>
    </div>
    <div class="col-3">
        <h2>Offers</h2>
        <marquee (mouseover)="onMouseOver($event)"
(mouseout)="onMouseOut($event)" scrollamount="10" direction="up">
        <div>
            <img src="assets/jblspeaker.jpg" width="100" height="100">
            <div>
                <span>40% OFF</span>
            </div>
```

```
                </div>
                <div>
                    <img src="assets/shoe.jpg" width="100" height="100">
                    <div>
                        <span>60% OFF</span>
                    </div>
                </div>
                <div>
                    <img src="assets/shirt.jpg" width="100" height="100">
                    <div>
                        <span>30% OFF</span>
                    </div>
                </div>
               </marquee>
            </div>
          </div>
        </div>
```

Other Angular Events

Event	Description
click	Specifies actions to perform when element clicked.
dblclick	Specifies actions on double click.
contextmenu	Specifies actions on right click.
select	Actions when selected
selectstart	Actions while selecting
cut, copy, paste	Actions on cut, copy and paste
change	Actions when value changed.
blur	Actions when element lost focus.
focus	Actions when element gets focus.
submit	Actions on Form submit.

Ex:

1. Eventsdemo.component.ts

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

@Component({
  selector: 'app-eventsdemo',
  templateUrl: './eventsdemo.component.html',
  styleUrls: ['./eventsdemo.component.css']
})
export class EventsdemoComponent {
    public msg;
    public txtName;
    public OnFocus() {
      this.msg = 'Name in Block Letters';
    }
    public OnBlur() {
      this.msg = '';
      this.txtName = this.txtName.toUpperCase();
    }
    public onCut(){
      this.msg = 'Removed and Placed on Clipboard';
    }
    public onCopy(){
      this.msg = 'Copied to Clipboard';
    }
    public onPaste() {
      this.msg = 'Inserted from Clipboard';
    }
}
```

2. Eventsdemo.component.html

```
<div class="container-fluid">
  <h2>Register</h2>
  <div class="form-group">
      <label>User Name</label>
      <div>
          <input (cut)="onCut()" (copy)="onCopy()" (paste)="onPaste()"
[(ngModel)]="txtName" (focus)="OnFocus()" (blur)="OnBlur()"
class="form-control" type="text" placeholder="Block Letters Only">
          <span>{{msg}}</span>
      </div>
  </div>
</div>
```

Submit Event

- It defines actions to perform when form is submitted.
- Submit event is configured for “<form>” element.
- Submit event fires up only on “Submit” button click.
- Explicitly you have call submit event for other elements.
- HTML provides
    - Generic Button
      Submit, Reset
      <input type=”submit”>
      <input type=”reset”>
      <button type=”submit”>
      <button type=”reset”>
    - Non-Generic
      Ordinary button
      <input type=”button”>
      <button type=”button”>

Ex:

.ts

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

```
@Component({
  selector: 'app-eventsdemo',
  templateUrl: './eventsdemo.component.html',
  styleUrls: ['./eventsdemo.component.css']
})
export class EventsdemoComponent {
   public onSubmit() {
     alert('Form Submitted');
   }
   public onChange(e) {
     if(e.target.value=='submit') {
       this.onSubmit();
     } else {
       alert('Form will Reset');
     }
   }
}
```

.html

```
<div class="container-fluid">
  <h2>Register</h2>
  <form (submit)="onSubmit()">
      <button>Submit By Default</button>
      <button type="button">Ordinary Button</button>
      <select (change)="onChange($event)">
          <option value="reset">Reset</option>
          <option value="submit">Submit</option>
      </select>
  </form>
</div>
```

FAQ: Can Form have multiple submit buttons?

A. Yes.

FAQ: How form submit event knows which button submitted?

Ex:

```
<div class="container-fluid">
  <h2>Register</h2>
  <form (submit)="onSubmit($event)">
     <button id="insert">Insert</button>
     <button id="update">Update</button>
     <button id="delete">Delete</button>
  </form>
</div>
public onSubmit(e) {
     alert(e.target.elements[0].id);
   }
```

Custom Events

- A Component in another component.

- Communication between components.
- Transport data from one component to another.
- Custom Events for component.
- Component Life Cycle Hooks
- Component library provided by Angular. [Angular Material]

Accessing a Component in Another Component

- You can access any component and use in the current component by configuring its selector.

Syntax:

```
<app-parent>
        <app-child> </app-child>
</app-parent>
```

Ex:

1. Add 2 components

```
> ng g c parent
> ng g c child
```

2. child.component.html

```
<div class="container" style="padding:30px; background-color:green;
color:white; text-align:center">

   <h3>Child Component</h3>

</div>
```

3. Parent.component.html

```
<div class="container-fluid" style="background-color:
lightgoldenrodyellow; margin:20px; height: 200px;">
   <h2>Parent Component</h2>
   <app-child></app-child>
</div>
```

Transport Data from Parent to Child

- Child component must have a property to store the value coming from parent.
- You can use that property in child component html page.
- Any property you declared in a component is not accessible to other components. It is local and private for current component.
- If you want any property to access and store value from other components then you have to mark the property by using directive "@Input()"

    @Component()
    @NgModule()
    @Input()
    @Output()
    @Pipe()

- @Input() marker will mark the property as global so that it can be access from another component. And Input refers to a property that can store data.
- You have to import Input directive from "@angular/core" library.

  Syntax:

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

@Input()  propertyName = value;
```

Ex:

1. Child.component.ts

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

@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css']
})
```

```
export class ChildComponent {
    @Input() public msg;
}
```

2. Child.component.html

```
<div class="container" style="padding:30px; background-color:green;
color:white; text-align:center">
   <h3>Child Component</h3>
   <p>{{msg}}</p>
</div>
```

3. Parent.component.ts

```
export class ParentComponent {
  public hello = 'Hello ! from Parent Component';
}
```

4. Parent.component.html

```
<div class="container-fluid" style="background-color:
lightgoldenrodyellow; margin:20px; height: 200px;">
   <h2>Parent Component</h2>
   <app-child [msg]="hello" ></app-child>
</div>
```

Ex: HTML input and output components.

```
<form id="frmdemo" oninput="x.value=parseInt(a.value) + parseInt(b.value)">

  n1 : <input type="text" id="a"  name="a" value="50">

 <br>

  n2 : <input type="text" id="b" name="b" value="30">

 <br>


</form>
```

```
<output form="frmdemo" id="x" name="x" for="a+b">
</output>
```

Add following components

```
> ng g component productsfilter --skipTests=true
> ng g component productsList --skipTests=true
```

Productsfilter.component.ts

```
import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';

@Component({
  selector: 'app-productsfilter',
  templateUrl: './productsfilter.component.html',
  styleUrls: ['./productsfilter.component.css']
})
export class ProductsfilterComponent {
  public selectedCategoryName = 'All';

  @Input() public AllCount = 0;
  @Input() public ElectronicsCount = 0;
  @Input() public FootwearCount = 0;
  @Input() public FashionCount = 0;
  @Output() public EmitCategoryName: EventEmitter<string> = new
EventEmitter<string>();
  public OnCategoryChanged() {
    this.EmitCategoryName.emit(this.selectedCategoryName);
  }
}
```

ProductsFilter.component.html

```
<div style="margin-top: 50px;">
 <h2>Filter Products</h2>
 <div>
     <label>Choose Category</label>
     <div>
         <select (change)="OnCategoryChanged()"
[(ngModel)]="selectedCategoryName" class="form-control">
            <option value="All">All [{{AllCount}}]</option>
            <option value="Electronics">Electronics [{{ElectronicsCount}}]
</option>
            <option value="Footwear">Footwear [{{FootwearCount}}]</option>
            <option value="Fashion">Fashion [{{FashionCount}}]</option>
         </select>
     </div>
 </div>
</div>
```

ProductsList.component.ts

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

@Component({
  selector: 'app-productslist',
  templateUrl: './productslist.component.html',
  styleUrls: ['./productslist.component.css']
})
export class ProductslistComponent {
 public products = [
   {Name: 'JBL Speaker', Price: 45000.55, Photo: 'assets/jblspeaker.jpg', Category: 'Electronics'},
   {Name: 'Earpods', Price: 4000.55, Photo: 'assets/earpods.jpg', Category: 'Electronics'},
   {Name: 'Nike Casuals', Price: 9000.55, Photo: 'assets/shoe.jpg', Category: 'Footwear'},
   {Name: 'Lee Cooper Boot', Price: 3000.55, Photo: 'assets/shoe1.jpg', Category: 'Footwear'},
   {Name: 'Shirt', Price: 2600.55, Photo: 'assets/shirt.jpg', Category: 'Fashion'},
   {Name: 'Jeans', Price: 2000.55, Photo: 'assets/jeans.jpg', Category: 'Fashion'}
 ];
 public ElectronicCount = this.products.filter(x=>x.Category=='Electronics').length;
 public FootwearCount = this.products.filter(x=>x.Category=='Footwear').length;
 public FashionCount = this.products.filter(x=>x.Category=='Fashion').length;
 public selectedCategoryValue = 'All';
 public onFilterCategoryChanged(selectedCategoryName) {
   this.selectedCategoryValue = selectedCategoryName;
 }
}
```

ProductsList.component.html

```
<div class="container-fluid">
  <h2 class="text-center text-primary">Amazon Shopping</h2>
  <div class="row" style="margin-top: 50px;">
    <div class="col-2">
        <app-productsfilter [AllCount]="products.length"
[ElectronicsCount]="ElectronicCount" [FootwearCount]="FootwearCount"
[FashionCount]="FashionCount"
(EmitCategoryName)="onFilterCategoryChanged($event)"></app-productsfilter>
    </div>
    <div class="col-10">
        <div class="card-deck">
            <ng-container *ngFor="let item of products">
                <div class="card" *ngIf="selectedCategoryValue=='All' ||
selectedCategoryValue == item.Category" >
                    <div class="card-header">
                     <h3 class="card-title">{{item.Name}}</h3>
                    </div>
                    <div class="card-body text-center">
                     <img [src]="item.Photo" width="100" height="100">
                    </div>
                    <div class="card-footer">
                     <h3 class="card-subtitle">{{item.Price}}</h3>
                    </div>
                   </div>
            </ng-container>
        </div>
    </div>
  </div>
</div>
```

- Ng g c productscatalog –skipTests
- Ng g c productsfilter –skipTests

Productscatalog.component.ts

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

@Component({
  selector: 'app-productscatalog',
  templateUrl: './productscatalog.component.html',
  styleUrls: ['./productscatalog.component.css']
})
export class ProductscatalogComponent {
  public products = [
    {Name: 'JBL Speaker', Price: 45000.43, Photo: 'assets/jblspeaker.jpg', Category: 
'Electronics'},
    {Name: 'Earpods', Price: 15000.43, Photo: 'assets/earpods.jpg', Category: 
'Electronics'},
    {Name: 'Nike Casuals', Price: 5000.43, Photo: 'assets/shoe.jpg', Category: 
'Footwear'},
    {Name: 'Lee Cooper Boot', Price: 4000.43, Photo: 'assets/shoe1.jpg', Category: 
'Footwear'},
    {Name: 'Shirt', Price: 3000.43, Photo: 'assets/shirt.jpg', Category: 'Fashion'},
    {Name: 'Jeans', Price: 5000.43, Photo: 'assets/jeans.jpg', Category: 'Fashion'},
  ];
  public electronicsCount = 
this.products.filter(x=>x.Category=='Electronics').length;
  public footwearCount = this.products.filter(x=>x.Category=='Footwear').length;
  public fashionCount = this.products.filter(x=>x.Category=='Fashion').length;

  public getCategoryName = 'All';
  public ComponentCategoryChanged(eventValue) {
    this.getCategoryName = eventValue;
  }
}
```

Productscatalog.component.html

```
<div class="container-fluid">
  <h2 class="text-primary text-center">Products Catalog</h2>
  <div class="row">
    <div class="col-2">
        <app-productsfilter [allcount]="products.length"
[electronicscount]="electronicsCount" [footwearcount]="footwearCount"
[fashioncount]="fashionCount"
(emitcategoryname)="ComponentCategoryChanged($event)" ></app-
productsfilter>
    </div>
    <div class="col-10">
        <table class="table table-hover">
            <thead>
                <tr>
                    <th>Name</th>
                    <th>Price</th>
                    <th>Preview</th>
                </tr>
            </thead>
            <tbody>
                <ng-container *ngFor="let item of products">
                    <tr *ngIf="getCategoryName=='All' ||
getCategoryName==item.Category">
                        <td>{{item.Name}}</td>
                        <td>{{item.Price}}</td>
                        <td><img [src]="item.Photo" width="50" height="50"></td>
                    </tr>
                </ng-container>
            </tbody>
        </table>
    </div>
  </div>
</div>
```

Productsfilter.component.ts

```
import { Component, EventEmitter, Input, Output } from '@angular/core';

@Component({
  selector: 'app-productsfilter',
  templateUrl: './productsfilter.component.html',
  styleUrls: ['./productsfilter.component.css']
})
export class ProductsfilterComponent {
   @Input() public allcount = 0;
   @Input() public electronicscount = 0;
   @Input() public footwearcount = 0;
   @Input() public fashioncount = 0;
   public selectedCategoryName = 'All';
   @Output() public emitcategoryname: EventEmitter<string> = new
EventEmitter<string>();

   public onCategoryChanged() {

     this.emitcategoryname.emit(this.selectedCategoryName);

   }

}
```

Productsfilter.component.html

```
<div class="form-group">

  <div>Select a Category</div>

  <ul class="list-unstyled">

      <li>

          <input type="radio" name="filter" value="All"
[(ngModel)]="selectedCategoryName" (change)="onCategoryChanged()"> All
[{{allcount}}]

      </li>

      <li>

        <input type="radio" name="filter" value="Electronics"
[(ngModel)]="selectedCategoryName" (change)="onCategoryChanged()">
Electronics [{{electronicscount}}]

    </li>

    <li>

        <input type="radio" name="filter" value="Footwear"
[(ngModel)]="selectedCategoryName" (change)="onCategoryChanged()">
Footwear [{{footwearcount}}]
```

```
  </li>
  <li>
    <input type="radio" name="filter" value="Fashion"
[(ngModel)]="selectedCategoryName" (change)="onCategoryChanged()">
Fashion [{{fashioncount}}]
  </li>
 </ul>
</div>
```

Angular Material

(Components Library for Angular)

- Angular Material is designed by Google Angular Team for better performance in Angular.
- Material is similar to bootstrap with a set of templates providing rich support of CSS and JQuery.
- Angular provides pre-defined component, which you can inject an use in your angular application.
- Provides modern UI component that work across the web, mobile and desktop.
- Performance of application will improve with material components.
- It is fully tested for modern browsers.

Setup Angular Material for your project:

1. Open your project app folder in terminal
2. Run the following command
   ```
   > ng add @angular/material
   ```
3. This will install angular material library for your project.
4. It will add following components into your project.
   - Angular Material

- Component Dev Kit [CDK]
- Angular Animations

5. It will ask for following Questions:

? Choose a prebuilt theme name, or "custom" for a custom theme: Indigo/Pink

? Set up global Angular Material typography styles? (y/N) y

? Set up browser animations for Angular Material? (Y/n) y

After Installing Material Library for Project:

- Add project dependencies to package.json
- Add the Roboto font to your "index.html"
- Add Material Design Icon font to your "index.html". [similar to Fontawesome]
- Add few Global CSS styles into "styles.css"

Setup a CSS theme for using Angular Material:

- Go to "Styles.css"
  @import '@angular/material/prebuilt-themes/deeppurple-amber.css';

Angular CDK

The Component Dev Kit (CDK) is a set of behavior primitives for building UI components.

Virtual Scrolling in Angular

- Implements Lazy Loading of content in page.
- Lazy Loading is a software design pattern that allows to load the components, modules and lists only when required.

- Eager Loading loads all the components, modules and list at the time of application loading. It occupies more memory. It effects the performance.
- In SPA [Single Page Application] you have to implement virtual scrolling.
- It can load the contents only when required. It will add new contents into page without reloading the complete page.
- Virtual Scrolling allows the user to stay on one page and access all content and display on one page.
- We can avoid pagination.
- Angular can implement "Virtual Scrolling" by using "@angular/cdk" library.
- The module required for implementing virtual scrolling is "ScrollingModule".
- The directive used to implement virtual scrolling for component in a page is "<cdk-virtual-scroll-viewport>".
- We need lazy iterator in CDK view port, which is "cdkVirtualFor"

```
*cdkVirtualFor="let item of collection"
```

Ex:

- Go to "app.module.ts"
- Import and register "ScrollingModule"

```
import { ScrollingModule } from '@angular/cdk/scrolling';

imports: [
    BrowserModule,
    FormsModule,
    BrowserAnimationsModule,
    ScrollingModule
  ],
```

- Add new component
- Scrollingdemo.component.css

```
.example-viewport{
    height: 300px;
```

```
  width: 55%;
  border:1px solid darkcyan;
  margin-bottom: 30px;
}
```

- Scrollingdemo.component.ts

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

@Component({
  selector: 'app-scrollingdemo',
  templateUrl: './scrollingdemo.component.html',
  styleUrls: ['./scrollingdemo.component.css']
})
export class ScrollingdemoComponent{
  public products = [
    {Name: 'Samsung TV', Photo: 'assets/tv.jpg'},
    {Name: 'Speaker', Photo: 'assets/jblspeaker.jpg'},
    {Name: 'Earpods', Photo: 'assets/earpods.jpg'},
    {Name: 'Mobile', Photo: 'assets/mobile.jpg'},
    {Name: 'Nike', Photo: 'assets/shoe.jpg'},
    {Name: 'Lee Boot', Photo: 'assets/shoe1.jpg'},
    {Name: 'Shirt', Photo: 'assets/shirt.jpg'},
    {Name: 'Jeans', Photo: 'assets/jeans.jpg'},
    {Name: 'Samsung TV', Photo: 'assets/tv.jpg'},
    {Name: 'Speaker', Photo: 'assets/jblspeaker.jpg'},
    {Name: 'Earpods', Photo: 'assets/earpods.jpg'},
    {Name: 'Shirt', Photo: 'assets/shirt.jpg'},
    {Name: 'Jeans', Photo: 'assets/jeans.jpg'},
  ];

}
```

- Scrollingdemo.component.html

```
<div class="container-fluid">
  <div class="row">
    <div class="col-4">
```

```
        <h2>Normal List</h2>
        <ol>
          <li *ngFor="let item of products">
            {{item.Name}}
          </li>
        </ol>
      </div>
      <div class="col-8">
        <h2>Virtual Scrolling</h2>
        <cdk-virtual-scroll-viewport itemSize="100" class="example-
viewport">
          <div class="card" *cdkVirtualFor="let product of products">
            <div class="card-body">
              <img [src]="product.Photo" width="50" height="50">
            </div>
            <div class="card-footer">
              <h3>{{product.Name}}</h3>
            </div>
          </div>
        </cdk-virtual-scroll-viewport>
      </div>
    </div>
</div>
```

Angular Material Components

- To implement angular material components
  - You have to identify the Module required for component.
  - You have to identify the Module dependencies.
  - A dependency specifies the required support for the component you are trying implement.
  - You have to verify the “API” provided by Angular Material Library for implementing any component.
  - API gives you a set of directives required for component.

- You must know the hierarchy of components. [In which component the current component fits]
- The basic components required for Material Design
  - Form field
  - Input
  - Button

Form Field

- <mat-form-field> is a component used to wrap several Angular Material Component and apply common Text field styles such as the underline, floating label and hind messages.
- The components that can work inside form field are
  - <input>
  - <textarea>
  - <select>
  - <mat-select>
  - <mat-chip-list>
- The module required for form filed is “MatFormFieldModule” from “@angular/material/form-field”.

Ex:

1. Go to “app.module.ts” and import the required module and its dependencies.

   ```
   import { MatFormFieldModule } from '@angular/material/form-field';
   ```

2. Register the modules required for component.

   ```
   imports: [
    MatFormFieldModule
     ]
   ```

3. You can access and use in any component by using its “directive” in “.html” page

```
 <mat-form-field>
     <mat-label>Enter User Name</mat-label>
     <input matInput placeholder="User Name">
  </mat-form-field>
```

Note: Import "MatInputModule" into "app.module.ts"

- Every Material Component is provided with a set of attributes.
- Form Field uses the attribute "appearance" with following values
  - Legacy
  - Standard
  - Fill
  - Outline

Ex:

1. Formfield.component.ts

```
export class FormfieldComponent {
  public username;
}
```

2. Formfield.component.html

```
<div class="container-fluid">
  <div class="form-group">
    <label>User Name</label>
    <div>
       <input class="form-control" placeholder="User Name">
    </div>
  </div>
  <mat-form-field appearance="outline">
```

```
        <mat-label>Enter User Name</mat-label>
        <input [(ngModel)]="username" matInput placeholder="User Name">
    </mat-form-field>
    <div>
        <h2>Hello ! {{username}}</h2>
    </div>
</div>
```

Date Picker

- Angular 10 provides a new date picker component.
- Angular Material is using 10x version so the date picker is with latest features.
- Date Picker requires the following modules
  - MatDatepickerModule
  - MatNativeDateModule [Dependency for DatePicker – Material Core Module]
- Directives required for date picker
  - <mat-date-picker> [date picker]
  - <mat-date-picker-toggle> [Calendar icon as button]

Ex:

App.module.ts

```
import { MatDatepickerModule} from '@angular/material/datepicker';

import { MatNativeDateModule } from '@angular/material/core';



[MatDatepickerModule,

    MatNativeDateModule

  ]
```

```
<div class="container-fluid">
  <mat-form-field>
     <mat-label>Departure Date</mat-label>
     <input matInput [matDatepicker]="picker">
     <mat-datepicker-toggle [for]="picker" matSuffix ></mat-datepicker-toggle>
     <mat-datepicker #picker></mat-datepicker>
  </mat-form-field>
</div>
```

Material Auto Complete

- Open https://material.angular.io/components/
- Choose any component that you want to implement [Auto Complete]
- You will find 3 Tabs
  - Overview
  - API
  - Examples
- Overview contains summary about the component and its functionality.
- API contains information about the Modules and Dependencies required for component. It also comprises of all properties and methods information used for component. [Syntax]
- Examples is a set of samples for component.
- Go to API and verify
  - Module to Import
    import {MatAutocompleteModule} from '@angular/material/autocomplete';
  - Selector to access
    Selector : mat-autocomplete
    Exported as : matAutocomplete

- To understand the design and functionality of component go to “Examples” and explore the source code of any example by clicking on “< >” view code icon.
- Every component comprises of 3 files
    - Html
    - CSS
    - TS

Ex:

- Go to “app.module.ts”

```
import { MatAutocompleteModule } from
'@angular/material/autocomplete';


imports: [

MatAutocompleteModule

  ],
```

- .ts

```
public selectedCountry;
  public countries = [
    {Name: 'India', Flag: 'assets/india.png', CallCode: '+91'},
    {Name: 'UK', Flag: 'assets/uk.png', CallCode: '+44'}
  ];
```

- .html

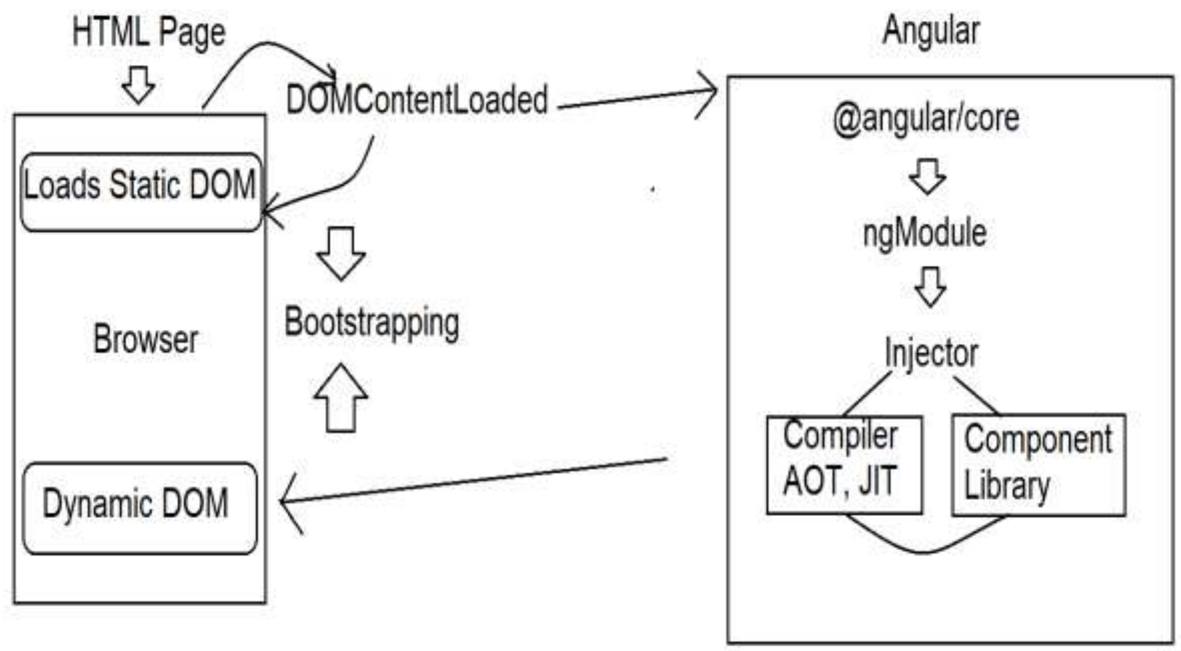
```
<div class="container-fluid">
   <h2>HTML Auto Complete</h2>
   <div class="form-group">
      <input type="text" placeholder="Your country" class="form-control"
list="countries">
      <datalist id="countries">
          <option>India</option>
          <option>US</option>
          <option>UK</option>
        </datalist>
      </div>
      <form name="frmHome" class="example-form">
        <h2>Material Auto Complete</h2>
        <mat-form-field class="example-full-width">
          <mat-label>Choose Your Country</mat-label>
          <input [(ngModel)]="selectedCountry" matInput type="text"
[matAutocomplete]="countriesList" placeholder="Your Country">
          <mat-autocomplete #countriesList="matAutocomplete">
            <mat-option *ngFor="let item of countries" [value]="item.Name">
              <img class="example-option-img" [src]="item.Flag" height="25">
              <span>{{item.Name}}</span> |
              <small>Calling Code : {{item.CallCode}}</small>
            </mat-option>
          </mat-autocomplete>
        </mat-form-field>
      </form>
    </div>
```

- .css

```
.example-form {
    min-width: 150px;
    max-width: 500px;
    width: 100%;
  }

  .example-full-width {
    width: 100%;
  }

  .example-option-img {
    vertical-align: middle;
    margin-right: 8px;
  }

  [dir='rtl'] .example-option-img {
    margin-right: 0;
    margin-left: 8px;
  }
```

Hooking into Component Life Cycle

- A component instance has a lifecycle.
- The lifecycle starts when Angular instantiates the component.
- The lifecycle continues with change detection.
- The lifecycle ends when Angular destroys the component instance and removes from DOM.
- Component Creates, updates and destroys instances.
- All phases of a component are maintained by a sequence of events.
- These events are controlled with a set of methods known as Hook Method.
- Often known as "Life Cycle Hooks".

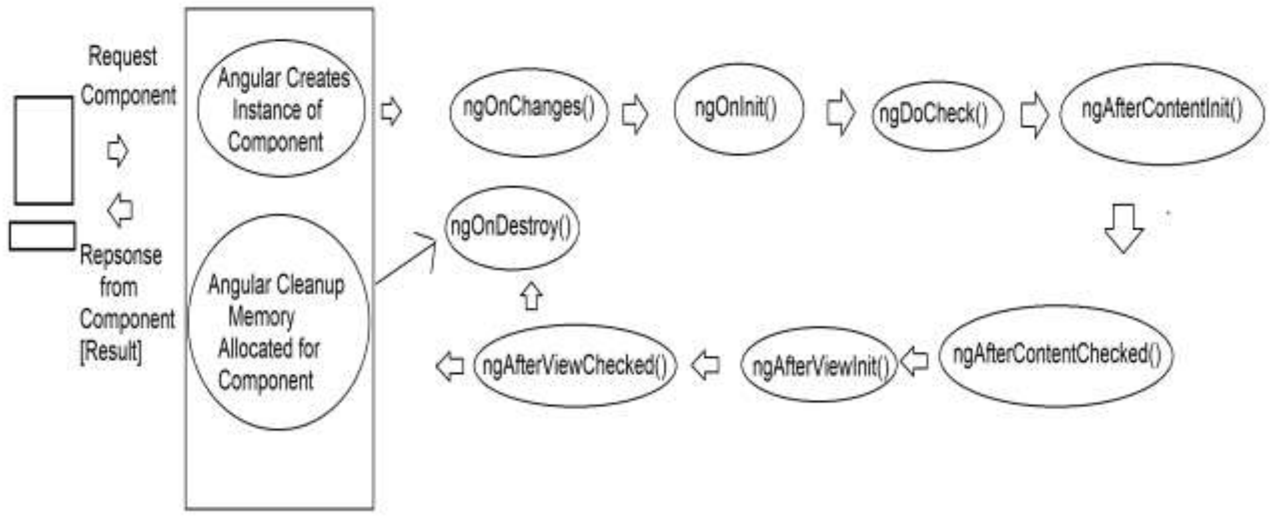


- The lifecycle methods used by "Component"

Hook Method	Purpose
ngOnChanges()	- Angular sets the value. - Binds the values to any property. - It gets notified with the changes in values by using "SimpleChanges" object. - It gets the previous value and current value. - It includes loading component and handling any action (event) preformed in application. - Manages the property and event binding in Angular.
ngOnInit()	- It will be called after the first "ngOnChanges()".

	- Initialize the directive or component after Angular first displays the data-bound properties and sets. - Initialize memory for transporting data across components and bind to parent and child properties. - Memory initialized to handle values between interactions.
ngDoCheck()	- Called immediately after "ngOnChanges() or every change detection and also immediately after "ngOnInit()" on its first run. - It can detect and act upon the changes that Angular can't or won't detect implicitly. - It is very regular while using Custom events with @Input() and @Output(). - Transporting Data between components
ngAfterContentInit()	- Called after "ngOnInit()" - Content into the components View (".html") - It binds the content into dynamic angular containers like "ng-template & ng-container".
ngAfterContentChecked()	- Called after "ngAfterContentInit()" - It is also called after every "ngDoCheck()" - This is responsible for "Content Projection". - It is a way to import HTML content from outside the component and insert that content into the components template at specific location. - After binding data to view on current component. It brings the content from external component or child

	component and renders into current component.
ngAfterViewInit()	- Called after "ngAfterContentChecked()" - Fire up after the initialization of memory for component views and its child views. - It responds to view changes. - It identifies the changes in parent or child views and updates the content. - Input of data from parent to child, Output of data from child to parent these are monitored in "ngAfterViewInit()". - Tracking of changes in data and transporting the changes to update [between parent and child].
ngAfterViewChecked()	- It is called after "ngAfterViewInit()" - It renders the final content to parent and child views.
ngOnDestroy()	- Clean up the memory before destroying the component. - Unsubscribe to methods. - Detach the event handling. - Destroying memory allocated for component. - It is important to handle memory leaks.



FAQ: What is Change Detection?
FAQ: What is Change Projection?
FAQ: What is Content Projection?
FAQ: In Which life cycle hook data is transported from one component to another?
FAQ: Why to destroy component?

Change Detection

- It is managed under "ngOnChanges()"
- Sets the value to property.
- If no value defined into property then no change detected.
- Binds the value to HTML element property.
- If there is a value defined into property and that is bound to element property then change detected.
- Detect the changes in value.
- Update the Changes to Model.
- "ngModel" with property and event binding [Two Way Binding] is one of the live examples for "Change Detection".
- Model is "Single-source-of-truth"
- "SimpleChanges" is the base that identified the changes by accessing
    - CurrentValue

    - PreviousValue
- “SimpleChanges” object identifies the changes in any property and update the changes.
- SimpleChanges uses “SimpleChange” base class that following properties
    - previousValue:any
    - currrentValue:any
    - firstChange:boolean

Ex:

- Add two components
    - ng g c sendvalue –skipTests
    - ng g c displayvalue – skipTests
- displayvalue.component.ts

```
import { Component, Input, OnChanges, SimpleChanges } from
'@angular/core';

@Component({
  selector: 'app-displayvalue',
  templateUrl: './displayvalue.component.html',
  styleUrls: ['./displayvalue.component.css']
})
export class DisplayvalueComponent implements OnChanges{
   @Input() public username;
   public previousvalue;
   public currentvalue;
   public msg;
   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';
       }

   }
}
```

- displayvalue.component.html

```
<div>
   <h2>Child Component</h2>
   Hello ! {{username}}
   <h2>{{msg}}</h2>
   <dl>
      <dt>Previous Value</dt>
      <dd>{{previousvalue}}</dd>
      <dt>Current Value</dt>
      <dd>{{currentvalue}}</dd>
   </dl>
</div>
```

- sendvalue.component.ts

```
export class SendvalueComponent{
  public username = 'John';
}
```

- Sendvalue.component.html

```
<div class="container-fluid">
 <h2>Parent Component</h2>
 <div class="form-group">
   <input type="text" [(ngModel)]="username">
 </div>
 <div class="form-group">
   <app-displayvalue [username]="username"></app-displayvalue>
 </div>
</div>
```

Angular Pipes

- Pipe is used to transform data.
- Data comes to your Angular application from various sources.
- The data type of source provider and the data types supported in TypeScript will not match.
- Hence the data is not displayed in the same format how we are expecting.
- Pipe can transform the data and display in desired format.
- Angular pipes are used for formatting and filtering the data.
- All pipes in Angular are derived from "PipeTransform" base.
- - Every pipe implements a functionality by using "transform()" method.
- Every pipe is defined with a pipe name, which is configured using "@Pipe()" marker
- Angular provides several built-in pipes and also allows to create custom pipes.

Syntax:

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

@Pipe({

  name: "uppercase"

 })

export class UpperCase implements PipeTransform

{

     transform(value) {

           return value;

     }

}
```

- Angular built-in pipes
    - AsyncPipe
    - CurrencyPipe
    - DatePipe

  - DecimalPipe
  - I18PluralPipe
  - I18SelectPipe
  - JsonPipe
  - KeyValuePipe
  - LowerCasePipe
  - UpperCasePipe
  - TitleCasePipe
  - PercentPipe
  - SlicePipe
- Angular pipes are assigned to your data by using “|” pipe symbol.
- Angular Pipes are by default “Pure” pipes. They will not change the value; they just defined a format for value.
- If Pipe can change the state and value then it is “Impure” pipe.

Syntax:

{{ yourData | pipeName:options }}

Pipe	Name	Description

UpperCasePipe	uppercase	It converts all letters into block letters.
		Ex:
		public product = {
		    Name: 'Samsung TV',
		    Price: 45000.50,
		    Mfd: new Date('2020-03-20')
		  };
		{{product.Name | uppercase}}

LowerCasePipe	lowercase	It converts all letters into lowercase letters.
		Ex:
		public product = {
		    Name: 'Samsung TV',
		    Price: 45000.50,
		    Mfd: new Date('2020-03-20')
		  };
		{{product.Name | lowercase}}

TitleCasePipe	titlecase	It capitalizes the first letter of every word. Ex: public product = { Name: 'Samsung TV', Price: 45000.50, Mfd: new Date('2020-03-20') }; {{product.Name | titlecase}}
DecimalPipe	number	It is used to display numeric value with thousands separator and fractions. It comprises of: Minimum-Integer-Digits Minimum-Fraction-Digits Maximum-Fraction-Digits Syntax: {{ data |number }} {{ data | number: {minIntegerDigits}.{minFractionDigits}-{maxFractionDigits} }} Ex: price: 45000.50 {{price}} → 45000.5 {{price|number}} → 45,000.5 {{price|number:’5.4-4’}} → 45,000.5000 {{price|number:’5.2-4’}} → 45,000.50
CurrencyPipe	currency	It is used to display numbers in a currency format. It comprises of thousands separator, fractions and a currency symbol. Syntax: {{data|currency:’currencyFormat’: ‘digitsInfo’}}

		Currency Format: USD, INR, you can use literals.
		Digits Info: {minInteger}.{minFraction}-{maxFraction}
		
		Ex:
		{{product.Price |currency:'INR'}}
		{{product.Price |currency:'&#8377;'}}

DatePipe	date	It is used for displaying the date and time values in various date and time formats. You can use predefined formats for Date or your can define custom format. Pre-Defined Formats: - short - medium - long - full - shortDate - mediumDate - longDate - fullDate - shortTime - mediumTime - longTime - fullTime Syntax: product.Mfd = new Date("2020-03-22"); {{yourDate | date: 'format'}} {{product.Mfd | date:'shortDate'}} Custom Format: MM - 2 Digits Month MMM - Short Month Name

		MMMM - Long Month Name dd - 2 Digits Date d - 1 Digit Date yy - 2 Digits Year yyyy - 4 Digits Year Ex: {{product.Mfd | date:'MMM-dd-yyyy'}}

PercentPipe	percent	Transforms a number into a percentage string.
		
		Syntax:
		{{ value | percent:'digitsInfo'}}
		
		Ex:
		public product = {
		    Name: 'Samsung TV',
		    Price: 45000.50,
		    Mfd: new Date('2020-03-20'),
		    Sales: 0.259
		  };
		{{product.Sales | percent:'2.2-2'}}

SlicePipe	slice	It creates a new Array or string containing a subset (sliced) of the elements. It can extract values based on specified index and returns an array. Syntax: {{collection | slice:startIndex:endIndex }} Ex: public products = ['TV', 'Mobile', 'Shoe', 'Watch']; <ol> <li *ngFor="let item of products | slice:1:3">

		`{{item}}`
		`</li>`
		`</ol>`

JsonPipe	json	- It is used to convert the data into JSON.
		- You can access the form data, convert into JSON and send to any API Service.
		
		Syntax:
		`{{ data | json }}`
		
		Ex:
		`public product = {`
		`    Name: 'Samsung TV',`
		`    Price: 45000.50,`
		`    Mfd: new Date('2020-03-20'),`
		`    Sales: 0.259`
		`  };`
		`<pre>`
		`      {{product | json}}`
		` </pre>`
		
		O/P:
		`{`
		`  "Name": "Samsung TV",`
		`  "Price": 45000.5,`
		`  "Mfd": "2020-03-20T00:00:00.000Z",`
		`  "Sales": 0.259`
		`}`

KeyValuePipe	keyvalue	- It is used to transform an object or map into an array of key and value pairs. - Without key and value pipe we have to use iterator over property using “in” operator. Ex:

		for(var property in object)
		{
		   console.log(property + “:” + object[property]);
		}
		- KeyValue pipe allows to extract the key (property) and value from a map or array.
		Syntax:
		    {{collection | keyvalue}}
		
		- “key” returns the property.
		- “value” returns the value.
		Ex:
		Pipedemo.component.ts
		export class PipedemoComponent {
		  public products = ['TV', 'Mobile', 'Shoe', 'Watch'];
		  public data:{[key:number]:string} = {1:'Samsung TV', 2:'Nike Casuals'};
		}
		Pipedemo.component.html
		<div class="container-fluid">
		   <h2>Products Array </h2>
		   <ol class="list-unstyled">
		    <li *ngFor="let item of products | keyvalue">
		       [{{item.key}}] {{item.value}}
		    </li>
		   </ol>
		   <h2>Products Map</h2>
		   <ol class="list-unstyled">
		      <li *ngFor="let item of data | keyvalue">
		         {{item.key}} - {{item.value}}
		      </li>
		   </ol>
		</div>

I18nSelectPipe	i18Select	- I18 is a community of Angular. - It designed a SelectPipe. - It is a Generic selector that can make decision dynamically according to the state or value, and define the result when the relative condition is matching. - In Early versions we have to depend on lot of iterations and condition. Syntax: {{value_Expression | i18Select: mapping }}
I18nPluralPipe	i18nPlural	- As per coding standards we have to define a plural name for any collection and singular for single object. Ex: product – One, products – Collection - Plural pipe can identify whether the object comprises of single or multiple values and defined a plural name dynamically. - It can get the collection count and display messages according to the count. - It uses a map to verify the values. Syntax: {{collection.length |i18Plural: keyValueCollection}

Ex: Select Pipe

- Pipedemo.component.ts

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

```
@Component({
  selector: 'app-pipedemo',
  templateUrl: './pipedemo.component.html',
  styleUrls: ['./pipedemo.component.css']
})
export class PipedemoComponent {
  public products = [
    {Name: 'Samsung TV', City: 'Goa'},
    {Name: 'Nike Casuals', City: 'Delhi'},
    {Name: 'Mobile', City: 'Hyderabad'},
    {Name: 'Watch', City: 'Mumbai'}
  ];
  public statusMessage = {
    'Hyderabad': 'Delivery in 2 Days',
    'Delhi': 'Delivery in 5 Days',
    'Mumbai': 'Not Deliverable',
    'other': 'Unkown-We Will Update'
  };
}
```

- Pipedemo.component.html

```
<div class="container-fluid">
  <h2>Products Status</h2>
  <table class="table table-hover">
    <thead>
      <tr>
        <th>Name</th>
        <th>City</th>
        <th>Delivery Status</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let item of products">
        <td>{{item.Name}}</td>
        <td>{{item.City}}</td>
```

```
                <td>{{item.City|i18nSelect:statusMessage}}</td>
              </tr>
            </tbody>
          </table>
        </div>
```

Ex: Plural Pipe –

- Import material components : MatIcon, MatBadge

PipeDemo.component.ts

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

@Component({
  selector: 'app-pipedemo',
  templateUrl: './pipedemo.component.html',
  styleUrls: ['./pipedemo.component.css']
})
export class PipedemoComponent {
  public notifications = [];
  public notificaitonsMap:{[key:string]:string} = {
    '=0': 'No Missed Calls', '=1': 'One Missed Call', 'other': '# Missed Calls'
  };
  public display = false;
  public GameClick(){
    this.notifications.push('Game Installed');
    alert('Installed Successfully..');
  }
  public FBClick(){
    this.notifications.push('FaceBook Updated');
    alert('Updated Successfully..');
  }
  public ClearAll(){
    this.notifications = [];
  }
  public showNotificaitons(){
    this.display = (this.display==true)?false:true;
  }
}
```

Pipedemo.component.html

```
<div class="container">
      <h2>Plural Demo</h2>
      <div class="form-group">
        <div class="btn-toolbar bg-danger justify-content-between">
          <div class="btn-group">
            <button (click)="GameClick()" class="btn btn-danger">Install
Game</button>
            <button (click)="FBClick()" class="btn btn-danger">Update FB</button>
            <button class="btn btn-danger">Clean Memory</button>
          </div>
          <div class="btn-group">
            <button (click)="ClearAll()" class="btn btn-danger">
              <span class="fa fa-trash">Clear Notifications</span>
            </button>
          </div>
        </div>
      </div>
      <div class="form-group">
        <button (click)="showNotificaitons()" class="btn">
          <mat-icon  matBadge="{{notifications.length}}">phone</mat-icon>
        </button>
      <div>
        {{notifications.length | i18nPlural:notificaitonsMap}}
      </div>
      </div>
      <div *ngIf="display" class="form-group">
        <ol>
          <li *ngFor="let item of notifications">
             {{item}}
          </li>
        </ol>
      </div>
</div>
```

Custom Pipe

- Angular allows to configure and create your own pipe that can serve any specific situation in your application.
- Pipe is a class that implement "PipeTransform" base class defined in "@angular/core".
- It configures a functionality for pipe by using "transform()" method.
- The "transform()" method must return a transformation.
- Pipe is marked by using "@Pipe()" marker defined in Pipe base of "@angular/core".
- @Pipe() it specifies the name and meta data for pipe.
- Pipes are registered in "app.module.ts" declarations.

Ex:

1. Add a new file into "app" folder
2. Sentencecase.pipe.ts

```
import { PipeTransform, Pipe } from '@angular/core';

@Pipe({
   name: 'sentencecase'
})

export class SentenceCasePipe implements PipeTransform
{
   transform(str){
      let firstChar = str.charAt(0);
      let restChars = str.substring(1);
      let sentence = firstChar.toUpperCase() + restChars.toLowerCase();
      return sentence;
   }
}
```

3. Go to "app.module.ts"

```
declarations: [
        SentenceCasePipe
    ]
```

4. You can apply to any content

```
public msg = 'WelCOMe TO ANgular';
{{msg | sentencecase}}
```

Task: Create a pipe that can sort the collection and display in ascending order.

```
{{ collection | pipe }}
```

The process of creating reusable, maintainable, and testable components for our application and injecting into other components to use their functionality is known as "Dependency Injection".

Which pipe you will name as "impure pipe"

- I18nSelect
- I18nPlural
- Slice

Angular Service

- Service is a pre-defined business logic which can be reused in the application by injecting into any component.
- Service is a collection of Factories.
- Factory is a collection of related type of functions.
- You can inject a factory directly into any component in order to use the functionality.
- Factory uses "Single Call Mechanism". Every time when you want to use a function you need an object to create. [Disconnected and Discrete]
- Service uses a "Single Ton Mechanism". Object is created only for the first request and the same object is used across any number of requests. [Connected and Continuous]

- Angular Service uses "Dependency Injection" to inject a service into any component constructor. Instead of creating the service object using "new" operator.
- Dependency Injection is an "Application Design Pattern".
- Angular has its own DI framework.
- Angular uses DI framework to increase the application efficiency and modularity.
- Dependencies are services or objects that a class needs to perform its function.
- DI is a coding pattern in which a class asks for dependencies from external sources rather than creating them itself.
- It allows to share information between classes.
- In Angular, the DI framework provides declared dependencies to a class when that class is instantiated.
- The DI framework lets you to supply data to a component from an injectable service class.
- Technically service in Angular is a class, which comprises of a set of service methods.

  Syntax
  ```
  import { Injectable } from '@angular/core'
  @Injectable()
  export class DemoService
  {
  }
  ```
- Angular can't inject any service into component until you configure and Angular dependency injector with a provider.
- @Injectable is an Injector

What is Injector?

- It is an object in Angular "dependency-injection" system.
- It can find a names dependency in its cache or create a dependency using a configured provider.
- Injectors are created for "NgModules" automatically as part of the "bootstrap" process.

- Injector provides a singleton instance of a dependency and can inject the same instance into multiple components.
- The injector provides a hierarchy so that the content can be uses for the parent and child components.
- We can configure injector with different providers that can provide different implementations of the same dependency.

What is Provider?

- Provider is an object that implements one of the “Provider” interfaces.
- Provider defines how to obtain an injectable dependency associated with a DI token.
- Injector uses a provider to create a new instance of dependency for a class.
- Angular registers its own providers with every injector for services.
- Angular provides different type of providers
  - ValueProvider
  - ClassProvider
  - TypeProvider
  - ConstructorProvider
  - FactoryProvider etc..

Syntax:

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

@Injectable(

  {

    providedIn: 'root'

  }

)

export class SampleService {

  constructor() { }

}
```

Injecting Services:

- You have to inject the service into any component.
- We have to make sure that service is injected into component rather that creating a new instance.
- You can tell angular to inject dependency in a components constructor by specifying a constructor parameter with the dependency type (service type).
- Every parameter defined in constructor is accessible only within the constructor.
- You can define an “Access Modifier” to specify the scope of parameter as “private, public or protected”.

Syntax:

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

import { SampleService } from '../sample.service';


@Component({
  selector: 'app-sampleconsume',
  templateUrl: './sampleconsume.component.html',
  styleUrls: ['./sampleconsume.component.css']
})
export class SampleconsumeComponent implements OnInit {
  constructor(private sampleservice: SampleService) {


   }


  ngOnInit(): void {
    this.sampleservice.GetData();
  }
```

}

- Service uses singleton pattern when configured with “provider” in the service marker.

Ex:

```
@Injectable(
  {
    providedIn: 'root'
  }
```

- Service can be implemented without singleton by configuring in the “app.module.ts” Providers
  providers: [ServiceName]
- When you defined service in “Providers” of app.module.ts you can configure your service without “providedIn”.

Ex:

- Create a new folder by name “CustomServices” and add into “app” folder
- Add a new file
  captcha.service.ts

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


@Injectable({
  providedIn: 'root'
})
export class CaptchaService
{
  public GenerateCode() {
    let a = Math.random() * 10;

    let b = Math.random() * 10;

    let c = Math.random() * 10;

    let d = Math.random() * 10;

    let e = Math.random() * 10;

    let f = Math.random() * 10;

    let code = `${Math.round(a)} ${Math.round(b)}
${Math.round(c)} ${Math.round(d)} ${Math.round(e)}
${Math.round(f)}`;

    return code;

  }

}
```

- Go to "Login" component
- Login.component.ts

```
import { Component } from '@angular/core';
import { CaptchaService } from '../CustomServices/captcha.service';

@Component({
  selector: 'app-login',
  templateUrl: 'login.component.html',
  styleUrls: ['login.component.css']
})

export class LoginComponent {
  constructor(private captcha: CaptchaService){}
  public code = this.captcha.GenerateCode();
  public refreshClick() {
    this.code = this.captcha.GenerateCode();
  }
}
```

- Login.component.html

```
<div class="container-fluid">
  <div class="form-group">
  <label>User Name</label>
  <div>
      <input type="text" class="form-control">
  </div>
 </div>
 <div class="form-group">
  <label>Password</label>
  <div>
      <input type="password" class="form-control">
  </div>
 </div>
 <div class="form-group">
    <label>Verify Code</label>
    <div>
        {{code}} <button (click)="refreshClick()" class="btn"><span class="fa
fa-sync"></span></button>
    </div>
    <div>
        <input type="text" class="form-control">
    </div>
 </div>
```

```
        <div class="form-group">
          <button class="btn btn-primary btn-block">Login</button>
        </div>
      </div>
```

Ex: Service for providing data to various components.

- Open “CustomServices” folder in terminal
- Run the following command
  > ng g service data --skipTests

data.service.ts

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

@Injectable({
  providedIn: 'root'
})
export class DataService {
  public GetProducts(){
      return [
        {Name: 'JBL Speaker', Price: 45000.43, Photo: 'assets/jblspeaker.jpg', Category: 'Electronics'},
        {Name: 'Earpods', Price: 15000.43, Photo: 'assets/earpods.jpg', Category: 'Electronics'},
        {Name: 'Nike Casuals', Price: 5000.43, Photo: 'assets/shoe.jpg', Category: 'Footwear'},
        {Name: 'Lee Cooper Boot', Price: 4000.43, Photo: 'assets/shoe1.jpg', Category: 'Footwear'},
      {Name: 'Shirt', Price: 3000.43, Photo: 'assets/shirt.jpg', Category: 'Fashion'},
      {Name: 'Jeans', Price: 5000.43, Photo: 'assets/jeans.jpg', Category: 'Fashion'},
    ];
  }
}
```

- Go to ProductsComponent
  Products.component.ts

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

export class ProductscatalogComponent implements OnChanges, OnInit {
  constructor(private data: DataService){}
  public products = [];
  ngOnChanges(){
      // you can't bind the data for products as it is accessed by provider from
cache.
  }
  ngOnInit() {
    this.products = this.data.GetProducts();
  }
}
```

Angular Forms

- Form is a container that comprises of set of elements, which allow interaction with our application.
- Form provides an UI from where user can input, edit, delete, view data.
- HTML form comprises of elements like button, textbox, checkbox, radio, listbox etc.
- Angular makes the static HTML form into dynamic.

- HTML presents the form and Angular makes it interactive to handle client-side interactions.
- Based on where Angular is handling interactions the forms in angular are classified into 2 types
    - Template Driven Forms
    - Model Driven Form / Reactive Form

Template Driven Forms

- A template driven form configures and handles all interactions at View Level (HTML)
- Configuration of a form and its manipulation both handled in HTML template.
- Very optimized controller level interaction. All interactions are at view level.
- It reduces the number of requests to a component.
- It improves the page load time.
- It is good for forms designed in "in-line" technique.
- Template drive form is heavy on page. Slow in handling interactions and rendering.
- Hard to test and extend the form.
- Separation issues. Not loosely coupled.
- You can use template driven forms when you are designing an UI that doesn't require regular extensions.
- The directives that are used to configure "Form and Form Elements" in template driven approach.
    - NgForm
    - NgModel
- NgForm: It provides a set of properties and methods that are used to configure and handle <form> element.
- NgModel: It provides a set of properties and methods that are used to configure and handle a form control like button, textbox, checkbox, radio, dropdown list, etc.
- The library for "NgForm and NgModel" is "@angular/forms"

- The module is “FormsModule”

Configuring Form:

```
<form #formName=”ngForm”>

</form>
```

- NgForm provides set of attributes
    - Value
    - pristine
    - dirty
    - valid
    - invalid
    - etc..

Syntax:

```
formName.value

formName.pristine

formName.dirty
```

Configuring a Form Element

- “NgModel” is used to make a static form control into dynamic.

```
<input type=”text” ngModel   #txtName=”ngModel” name=”txtName”>

txtName.value

txtName.valid
```

Key Note:

- A Form Reference must implement “NgForm” to handle the form behaviour. <form #frmRegister=”ngForm”> [ngForm is of type NgForm]
- NgForm makes the form dynamic.
- NgForm is a member of “FormsModule” in “@angular/forms”.

- Form can’t access and submit data of any control without a Name. Every control defined in a form must have a “name” defined.
  <input type=”text” name=”txtName”>
- Angular can’t recognize any form element dynamically. The static form element must transform into dynamic form element.
- “NgModel” is a directive that makes the Static Form element into Dynamic.
  <input type=”text” name=”txtName” ngModel>
- Every dynamic element must have a reference name, which is used as a model name to store its value dynamically.
- Every form element must implement “NgModel”.
- Every control is configured “ngModel”. So that it can have a reference name dynamically.
  <input type=”text” name=”txtName” ngModel #txtName=”ngModel”>

  Form related properties are derived from “NgForm”
  Control related properties are derived from “NgModel”

Accessing the values from Template Form:

  Accessing all Values
- You can use “value” property of “NgForm”.
- The “value” property returns an object that contains collection of key value pairs.
- The key/value pairs refer to Control Name and the Control Value.
- All form control and their values can be accessed by using “formName.value” property.
  Syntax:
  frmRegister.value → Returns collection of all form elements and their values as an object
  Accessing any specific value
- You can access the value of any control by using “value” property of “NgModel”
  Syntax:
  txtName.value → Returns the value of any specific element.

Ex:

- Templateform.component.ts

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

@Component({
  selector: 'app-templateform',
  templateUrl: './templateform.component.html',
  styleUrls: ['./templateform.component.css']
})
export class TemplateformComponent {
   public product;
   public onFormSubmit(obj){
      this.product = obj;
      alert(this.product.txtName);
   }
}
```

- Templateform.component.html

```
<div class="container-fluid">
  <div class="row">
    <div class="col-3">
        <h2>Register Product</h2>
        <form #frmRegister="ngForm"
(submit)="onFormSubmit(frmRegister.value)" >
            <div class="form-group">
                <label>Name</label>
                <div>
                    <input ngModel #txtName="ngModel" name="txtName"
type="text" class="form-control">
                </div>
            </div>
            <div class="form-group">
```

```
                <label>Price</label>
                <div>
                    <input ngModel #txtPrice="ngModel" name="txtPrice"
type="text" class="form-control">
                </div>
            </div>
            <div class="form-group">
                <label>Quantity</label>
                <div>
                    <input ngModel #txtQty="ngModel" name="txtQty"
type="text" class="form-control">
                </div>
            </div>
            <div class="form-group">
                <label>Shipped To</label>
                <div>
                   <select ngModel #lstShippedTo="ngModel"
name="lstShippedTo" class="form-control">
                       <option>Delhi</option>
                       <option>Hyderabad</option>
                   </select>
                </div>
            </div>
            <div class="form-group">
                <label>In Stock</label>
                <div>
                    <input ngModel #optStock="ngModel" name="optStock"
type="checkbox"> Yes
                </div>
            </div>
            <div class="form-group">
                <button class="btn btn-primary btn-block">Register</button>
            </div>
        </form>
    </div>
```

```
        <div class="col-4">
          <h2>Product Object</h2>
          <pre>
            {{frmRegister.value | json}}
          </pre>
        </div>
        <div class="col-5">
          <h2>Product Details</h2>
          <dl>
            <dt>Name</dt>
            <dd>{{frmRegister.value.txtName}}</dd>
            <dt>Price</dt>
            <dd>{{txtPrice.value}}</dd>
            <dt>Qty</dt>
            <dd>{{txtQty.value}}</dd>
            <dt>Total</dt>
            <dd>{{txtQty.value * txtPrice.value}}</dd>
            <dt>Shipped To</dt>
            <dd>{{lstShippedTo.value}}</dd>
            <dt>Stock Status</dt>
            <dd>{{(optStock.value==true)?"Available":"Out of Stock"}}</dd>
          </dl>
        </div>
      </div>
    </div>
```

Validation in Template Driven Forms

- Validation is the process of verifying user input.
- Validation is required to ensure that contradictory and un-authorized data is not stored into the data source.
- Angular can handle validations client side by using a set of validation services.
- Angular validation services are categorized into 2 groups
    - Form State Validation Services
    - Input State Validation Services

Form State Validation Services

- Form state validation services verifies all fields in the form simultaneously at the same time.
- Angular verifies all fields in the forms before submitting and report errors.

Service Name	Property	Description
NgPristine	pristine	- It returns Boolean true when form is untouched. - All fields loaded but no modification identified.
NgDirty	dirty	- It returns Boolean true when form is modified. - At least one field in the form modified then entire form is recognized as dirty.
NgValid	valid	- It returns true when all fields in the form are in valid state.
NgInvalid	invalid	- It returns true when any one form field state is recognized as invalid.
NgSubmitted	submitted	- It returns true on form submit.

- All angular validation services return Boolean value

Syntax:

formName.propertyName

frmRegister.invalid

frmRegister.pristine

Ex:

formvalidation.component.html

```
<div class="container-fluid">
  <div class="row">
    <div class="col-3">
        <form #frmRegister="ngForm">
            <dl>
                <dt>Name</dt>
                <dd>
                    <input type="text" name="txtName" ngModel #txtName="ngModel"
required minlength="4">
                </dd>
                <dt>Mobile</dt>
                <dd>
                    <input type="text" name="txtMobile" ngModel
#txtMobile="ngModel" required pattern="\+91[0-9]{10}">
                </dd>
            </dl>
            <button [disabled]="frmRegister.invalid" class="btn btn-primary btn-
block">Submit</button>
        </form>
    </div>
    <div class="col-9">
        <h2>Form State Services</h2>
        <table class="table table-hover">
            <thead>
                <tr>
                    <th>Service Name</th>
                    <th>Value</th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td>Pristine</td>
                    <td>{{frmRegister.pristine}}</td>
                </tr>
                <tr>
                    <td>Dirty</td>
                    <td>{{frmRegister.dirty}}</td>
                </tr>
                <tr>
                    <td>Invalid</td>
                    <td>{{frmRegister.invalid}}</td>
                </tr>
                <tr>
                    <td>Valid</td>
                    <td>{{frmRegister.valid}}</td>
                </tr>
                <tr>
                    <td>Submited</td>
                    <td>{{frmRegister.submitted}}</td>
                </tr>
              </tbody>
            </table>
        </div>
      </div>
    </div>
```

Input State Validation Services

- Input state validation is verifying every form element individually.
- You can validate every form field and identify the issues.

Service	Property	Description
NgPristine	pristine	It returns true when any specific element is not yet modified.
NgDirty	dirty	It returns true when the value of form element is modified.
NgTouched	touched	It returns true when element gets focus and blurred.
NgUntouched	untouched	It returns true if element never touched.
NgValid	valid	It returns true if all input validations are valid.
NgInvalid	invalid	It returns true if any one validation property returns invalid.
NgErrors	errors	It is an object that collects all errors of input field. - Required - Minlength - Maxlength - Pattern - Email etc.

Syntax:

txtName.pristine

txtName.invalid

Ex: Form State and Input State validation

formvalidation.component.html

```
<div class="container-fluid">

  <form #frmRegister="ngForm">
    <dl>
        <h2>Register User</h2>
        <dt>User Name</dt>
        <dd>
            <input type="text" name="txtName" ngModel #txtName="ngModel"
class="form-control" required>
            <span *ngIf="frmRegister.submitted && txtName.invalid ||
txtName.touched && txtName.invalid" class="text-danger">Name
Required</span>
        </dd>
        <dt>Mobile</dt>
        <dd>
            <input type="text" name="txtMobile" ngModel #txtMobile="ngModel"
class="form-control" required>
            <span *ngIf="frmRegister.submitted && txtMobile.invalid ||
txtMobile.touched && txtMobile.invalid" class="text-danger">Mobile
Required</span>
        </dd>
        <button class="btn btn-primary btn-block">Register</button>
    </dl>
  </form>
</div>
```

Note: You can't use "invalid or valid" properties for a field that comprises of multiple validations. You can use "errors" object to identify the specific error in input field.

Ex: Errors object to handle multiple errors

```
<div class="container-fluid">


  <form #frmRegister="ngForm">
    <dl>
        <h2>Register User</h2>
        <dt>User Name</dt>
        <dd>
            <input type="text" name="txtName" ngModel #txtName="ngModel" 
class="form-control" required minlength="4">
            <div *ngIf="txtName.touched && txtName.invalid" class="text-danger">
                <span *ngIf="txtName.errors.required" >Name Required</span>
                <span *ngIf="txtName.errors.minlength">Name too short..</span>
            </div>
        </dd>
        <dt>Mobile</dt>
        <dd>
            <input type="text" name="txtMobile" ngModel #txtMobile="ngModel" 
class="form-control" required pattern="\+91\d{10}">
```

```
        <div *ngIf="txtMobile.touched && txtMobile.invalid" class="text-
danger">
          <span *ngIf="txtMobile.errors.required">Mobile Required</span>
          <span *ngIf="txtMobile.errors.pattern">Invalid Mobile</span>
        </div>
      </dd>
      <button class="btn btn-primary btn-block">Register</button>
    </dl>
  </form>
</div>
```

Can we use required property for radio button, dropdown list, checkbox to handle required validation?

A. No. You have to define "Custom Validations"

Custom Validation

- Every input validation can't be handled by using HTML validation properties.
- You have to write custom methods to verify the input value as per your requirements and report an error explicitly.

Ex:

Formvalidation.component.ts

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

@Component({
  selector: 'app-formvalidation',
  templateUrl: './formvalidation.component.html',
  styleUrls: ['./formvalidation.component.css']
})
export class FormvalidationComponent{
  public displayCityError = false;
  public displayEvenError = false;
  public VerifyCity(val){
    if(val=='nocity') {
      this.displayCityError = true;
    } else {
      this.displayCityError = false;
    }
  }
  public VerifyEven(val) {
     if(val % 2 == 0) {
       this.displayEvenError = false;
     } else {
       this.displayEvenError = true;
     }
  }
}
```

Formvalidation.component.html

```
<div class="container-fluid">


 <form #frmRegister="ngForm">
   <dl>
       <h2>Register User</h2>
       <dt>User Name</dt>
       <dd>
           <input type="text" name="txtName" ngModel #txtName="ngModel"
class="form-control" required minlength="4">
           <div *ngIf="txtName.touched && txtName.invalid" class="text-danger">
               <span *ngIf="txtName.errors.required" >Name Required</span>
               <span *ngIf="txtName.errors.minlength">Name too short..</span>
           </div>
       </dd>
       <dt>Mobile</dt>
       <dd>
           <input type="text" name="txtMobile" ngModel #txtMobile="ngModel"
class="form-control" required pattern="\+91\d{10}">
           <div *ngIf="txtMobile.touched && txtMobile.invalid" class="text-
danger">
               <span *ngIf="txtMobile.errors.required">Mobile Required</span>
               <span *ngIf="txtMobile.errors.pattern">Invalid Mobile</span>
           </div>
       </dd>
       <dt>Gender</dt>
```

```
        <dd>
            <input type="radio" name="optGender" value="none" ngModel
#optGender="ngModel"> None
            <input type="radio" name="optGender" value="male" ngModel
#optGender="ngModel"> Male
            <input type="radio" name="optGender" value="female" ngModel
#optGender="ngModel"> Female
        </dd>
        <dt>Select Your City</dt>
        <dd>
            <select (change)="VerifyCity(lstCities.value)" name="lstCities" ngModel
#lstCities="ngModel" class="form-control" >
                <option value="nocity">Select City</option>
                <option value="Delhi">Delhi</option>
                <option value="Hyd">Hyd</option>
            </select>
            <span *ngIf="displayCityError" class="text-danger">Please Select Your
City</span>
        </dd>
        <dt>Enter an Even Number</dt>
        <dd>
            <input (blur)="VerifyEven(txtEven.value)" type="text" class="form-
control" name="txtEven" ngModel #txtEven="ngModel">
            <span *ngIf="displayEvenError" class="text-danger">Not an Even
Number</span>
        </dd>
        <button class="btn btn-primary btn-block">Register</button>
    </dl>
  </form>
</div>
```

CSS Effects for Elements using Validation Properties

Formvalidation.component.css

```
dl {
    width:400px;
    margin:auto;
    justify-items: center;
    align-items: center;
}
.validStyle {
    border:2px solid green;
    box-shadow: 2px 2px 3px green;
}
.invalidStyle {
    border:2px solid red;
    box-shadow: 2px 2px 3px red;
}
```

Formvalidation.component.html

```
<dd>
```

```
        <input [ngClass]="{validStyle:txtName.valid,
invalidStyle:txtName.invalid}" type="text" name="txtName" ngModel
#txtName="ngModel" class="form-control" required minlength="4">
        <div *ngIf="txtName.touched && txtName.invalid" class="text-danger">
          <span *ngIf="txtName.errors.required" >Name Required</span>
          <span *ngIf="txtName.errors.minlength">Name too short..</span>
        </div>
      </dd>
```

Angular Provides CSS classes for Validation

- Angular provides a set of CSS validation classes.
- These classes can identity the validation state of form or input element and application effects dynamically.

Class Name	Description
.ng-invalid	It contains a set of style attributes that are applied to any input element of form when its state is returned as “invalid”.
.ng-valid	It contains a set of style attributes that are applied to any input element of form when its state is returned as “valid”.
.ng-pristine	It contains a set of style attributes that are applied to any input element of form when its state is returned as “pristine”.
.ng-dirty	It contains a set of style attributes that are applied to any input element of form when its state is returned as “dirty”.

.ng-touched	It contains a set of style attributes that are applied to any input element of form when its state is returned as “touched”.
.ng-untouched	It contains a set of style attributes that are applied to any input element of form when its state is returned as “untouched”.

Form CSS Classes are applied by using form selector or any reference for form

form.ng-valid { }
form.ng-pristine{ }

Input CSS classes are applied by using input selector or any reference for input.

input.ng-valid { }
input.ng-touched { }

Ex:

Formvalidation.component.css

```
dl {
   width:400px;
   margin:auto;
   justify-items: center;
   align-items: center;
}
input.ng-invalid {
   border: 1px solid red;
   box-shadow: 2px 2px 3px red;
}
```

```
input.ng-valid {
   border: 1px solid green;
   box-shadow: 2px 2px 3px green;
}
form.ng-invalid {
   background-color: lightcoral;
}
form.ng-valid {
   background-color:lightgreen;
}
```

Formvalidation.component.html

```
<div class="container-fluid">

  <form #frmRegister="ngForm">
   <dl>
      <h2>Register User</h2>
      <dt>User Name</dt>
      <dd>
         <input type="text" name="txtName" ngModel
#txtName="ngModel" class="form-control" required
minlength="4">
         <div *ngIf="txtName.touched && txtName.invalid"
class="text-danger">
            <span *ngIf="txtName.errors.required" >Name
Required</span>
            <span *ngIf="txtName.errors.minlength">Name too
short..</span>
         </div>
```

```
        </dd>
        <dt>Mobile</dt>
        <dd>
          <input type="text" name="txtMobile" ngModel
#txtMobile="ngModel" class="form-control" required
pattern="\+91\d{10}">
          <div *ngIf="txtMobile.touched && txtMobile.invalid"
class="text-danger">
            <span *ngIf="txtMobile.errors.required">Mobile
Required</span>
            <span *ngIf="txtMobile.errors.pattern">Invalid
Mobile</span>
          </div>
        </dd>
        <dt>Gender</dt>
        <dd>
          <input type="radio" name="optGender" value="none"
ngModel #optGender="ngModel"> None
          <input type="radio" name="optGender" value="male"
ngModel #optGender="ngModel"> Male
          <input type="radio" name="optGender" value="female"
ngModel #optGender="ngModel"> Female
        </dd>
        <dt>Select Your City</dt>
        <dd>
          <select (change)="VerifyCity(lstCities.value)"
name="lstCities" ngModel #lstCities="ngModel" class="form-
control" >
```

```
            <option value="nocity">Select City</option>
            <option value="Delhi">Delhi</option>
            <option value="Hyd">Hyd</option>
          </select>
          <span *ngIf="displayCityError" class="text-danger">Please
Select Your City</span>
        </dd>
        <dt>Enter an Even Number</dt>
        <dd>
          <input (blur)="VerifyEven(txtEven.value)" type="text"
class="form-control" name="txtEven" ngModel
#txtEven="ngModel">
          <span *ngIf="displayEvenError" class="text-danger">Not an
Even Number</span>
        </dd>
        <button class="btn btn-primary btn-
block">Register</button>
    </dl>

  </form>
</div>
```

Reactive Forms or Model Driven Forms

- Reactive forms provide Model Driven approach.
- They are bound to model so that any change in model will update the view.
- A model driven approach binds the view with data structure.

- Configuration of forms and controls are defined at application logic level. (controller)
- Easy to extend and loosely coupled.
- Easy to test.
- Clean separation of functionality and presentation (Implementation and Design)
- Reactive forms are asynchronous, they allow to submit only a specific portion of form.
- Type support partial updates.
- You can dynamically add or remove controls from form.
- The library required for configuration and implementation of reactive forms is
  "@angular/forms"
- The classes required for configure forms and controls dynamically
  - ReactiveFormsModule
  - FormsModule

Configure a Form Control

- The form elements like textbox, checkbox, radio, dropdown etc. are configured by using "FormControl" base.
- Any form element can be configured by using "**FormControl**"

  Syntax:
  public elementName = new FormControl("value", options);

- The form control is bound to any element in the view by using "[formControl]" property. It is the member of

"**ReactiveFormsModule**". Import the module into "app.module.ts"

Syntax:
<input type="text" [formControl]="elementName">

- You can dynamically set value or update value into form control by using the functions
    - setValue()
    - patchValue()

    Syntax:
    this.elementName.setValue("somevalue");

EX:

- Go to "app.module.ts"
  import { FormsModule, ReactiveFormsModule } from '@angular/forms';

  imports: [
  ReactiveFormsModule
    ],
- Add a new component
- **Reactivedemo.component.ts**

  import { Component, OnInit } from '@angular/core';
  import { FormControl } from '@angular/forms';

```
@Component({
  selector: 'app-reactivedemo',
  templateUrl: './reactivedemo.component.html',
  styleUrls: ['./reactivedemo.component.css']
})
export class ReactivedemoComponent {
  public txtName = new FormControl('');
  public lstCities = new FormControl('');

  public UpdateClick() {
    this.txtName.setValue('Samsung TV');
    this.lstCities.setValue('Hyderabad');
  }
}
```

- **Reactivedemo.component.html**

```
<div class="container-fluid">

    <div class="row">
        <div class="col-3">
            <h2>Register</h2>
            <div class="form-group">
                <label>Name</label>
                <div>
                    <input [formControl]="txtName" type="text"
class="form-control">
                </div>
            </div>
            <div class="form-group">
                <label>Shipped To</label>
                <div>
                    <select [formControl]="lstCities" class="form-control">
                        <option>Delhi</option>
                        <option>Hyderabad</option>
                    </select>
                </div>
            </div>
            <div class="form-group">
                <button (click)="UpdateClick()" class="btn btn-info btn-
block">Update</button>
            </div>
        </div>
        <div class="col-9">
            <h2>Product Details</h2>
            <dl>
                <dt>Name</dt>
                <dd>{{txtName.value}}</dd>
                <dt>Shipped To</dt>
                <dd>{{lstCities.value}}</dd>
            </dl>
        </div>
    </div>
</div>
```

Summary: **FormControl - to create a control, [formControl] – to bind with element in UI.**

Create and Configure Form and Nested Forms

- You can dynamically create and configure forms.
- It allows to extend the form and make it more asynchronous.
- You can create a form by using “FormGroup” base.
- “FormGroup” is a collection of FormControls.

Syntax:

```
public parentForm = new FormGroup({
        controlName: new FromControl(‘’),
        controlName: new FormControl(‘’),
        childForm: new FormGroup({
                controlName: new FormControl(‘’)
        })
})
```

- To bind a form and nested form you have to use the properties
    - [formGroup] – Parent Form
    - [formGroupName] – Child Form

Syntax:

```
<form [formGroup]=”parentForm”>

        <div [formGroupName]=”childForm”>

        </div>

</form>
```

- If you are defining a control in form group the control is bound to element by using the attribute
  "formControlName"

  Syntax:

  <input type="text" formControlName="controlName">

- The method used to set and patch values are
  - setValue()
  - patchValue()

Ex:

Reactivedemo.component.ts

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

import { FormControl, FormGroup } from '@angular/forms';


@Component({

  selector: 'app-reactivedemo',

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

  styleUrls: ['./reactivedemo.component.css']

})

export class ReactivedemoComponent {

  public frmRegister = new FormGroup({

    Name: new FormControl(''),
    Price: new FormControl(''),
    frmDetails: new FormGroup({
      City: new FormControl(''),
      InStock: new FormControl('')
    })
  });
  public UpdatePartial() {
    this.frmRegister.patchValue({
      Name: 'Samsung TV',
      frmDetails: {
        City: 'Delhi',
        InStock: true
      }
    });
  }
}
```

Reactivedemo.component.html

```
<div class="container-fluid">

    <div class="row">
    <div class="col-3">
      <h2>Register Product</h2>
       <form [formGroup]="frmRegister">
         <fieldset>
           <legend>Basic Info</legend>
           <dl>
             <dt>Name</dt>
             <dd>
               <input formControlName="Name" class="form-control" 
type="text">
             </dd>
             <dt>Price</dt>
             <dd>
               <input formControlName="Price" type="text" 
class="form-control">
             </dd>
           </dl>
         </fieldset>
         <fieldset>
           <legend>Stock Details</legend>
           <div formGroupName="frmDetails">
```

```
                    <dl>
                        <dt>City</dt>
                        <dd>
                            <select formControlName="City" class="form-
control">
                                <option>Delhi</option>
                                <option>Hyd</option>
                            </select>
                        </dd>
                        <dt>In Stock</dt>
                        <dd>
                            <input formControlName="InStock"
type="checkbox">
                        </dd>
                    </dl>
                    <button class="btn btn-primary btn-block"
(click)="UpdatePartial()">Update Details</button>
                </div>
            </fieldset>
        </form>
    </div>
    <div class="col-9">
```

```
      <h2>Product Details</h2>
      <dl>
        <dt>Name</dt>
        <dd>{{frmRegister.value.Name}}</dd>
        <dt>Price</dt>
        <dd>{{frmRegister.value.Price}}</dd>
        <dt>City</dt>
        <dd>
          {{frmRegister.value.frmDetails.City}}
        </dd>
        <dt>Stock</dt>
        <dd>
{{(frmRegister.value.frmDetails.InStock)==true?"Available":"Out of
Stock"}}
        </dd>
      </dl>
    </div>
  </div>
</div>
```

Form Builder in Reactive Form

- FormBuilder is a service provided by Angular to configure forms and its elements dynamically.
- FormBuilder uses singleton pattern.
- FormBuilder provides methods
    - group()
    - control()
    - array()
- group() configures a form group with a set of elements initialized. <form>
- control() configures a form control <input> <select> etc.
- array() configures a collection of form controls. You can add or remove controls dynamically.
- Form builder uses the following properties to bind with element is UI.
    - formGroup : Parent Form
    - formGroupName : Child Form
    - formControlName : Form Elements
- **FormBuilder** is a member of "@angular/forms"

Ex:

Builderdemo.component.ts

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

import { FormBuilder } from '@angular/forms';


@Component({

  selector: 'app-builderdemo',
  templateUrl: './builderdemo.component.html',
  styleUrls: ['./builderdemo.component.css']
})
export class BuilderdemoComponent implements OnInit {

  constructor(private fb: FormBuilder) { }

  public frmRegister = this.fb.group({
    Name: [''],
    Price: [''],
    frmDetails: this.fb.group({
      City: [''],
      InStock: ['']
    })
  });

  ngOnInit(): void {
  }

}
```

Builderdemo.component.html

```
<div class="container-fluid">
 <div class="row">
  <div class="col-3">
    <h3>Register Product</h3>
    <form [formGroup]="frmRegister">
        <fieldset>
            <legend>Basic Info</legend>
            <dl>
                <dt>Name</dt>
                <dd>
                    <input formControlName="Name" type="text" class="form-
control">
                </dd>
                <dt>Price</dt>
                <dd>
                    <input formControlName="Price" type="text" class="form-
control" >
                </dd>
            </dl>
        </fieldset>
        <fieldset>
```

```
      <legend>Stock Details</legend>
      <div formGroupName="frmDetails">
        <dl>
          <dt>City</dt>
          <dd>
            <select formControlName="City" class="form-control">
              <option>Delhi</option>
              <option>Hyd</option>
            </select>
          </dd>
          <dt>InStock</dt>
          <dd>
            <input formControlName="InStock" type="checkbox">
          </dd>
        </dl>
      </div>
    </fieldset>
  </form>
</div>
<div class="col-9">
  <h3>Details</h3>
```

```
    <dl>
        <dt>Name</dt>
        <dd>{{frmRegister.value.Name}}</dd>
        <dt>Price</dt>
        <dd>{{frmRegister.value.Price}}</dd>
        <dt>City</dt>
        <dd>{{frmRegister.value.frmDetails.City}}</dd>
        <dt>Stock</dt>
        <dd>
            {{(frmRegister.value.frmDetails.InStock==true)?"Available":"Out
of Stock"}}
        </dd>
    </dl>
  </div>
 </div>
</div>
```

Form Array & Form Control

- Form Array allows to “add or remove” any form element dynamically.
- It is configured by using “array()” method of “FormBuilder” service.

- Form Array represents TypeScript array and can make use of all array functions:
    - push()
    - pop()
    - removeAt()
    - shift() etc.

Ex:

Builderdemo.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormArray, FormBuilder } from '@angular/forms';

@Component({
  selector: 'app-builderdemo',
  templateUrl: './builderdemo.component.html',
  styleUrls: ['./builderdemo.component.css']
})
export class BuilderdemoComponent implements OnInit {

  constructor(private fb: FormBuilder) { }

  public frmRegister = this.fb.group({
    Name: [''],
    Price: [''],
    frmDetails: this.fb.group({
      City: [''],
      InStock: ['']
    }),
    newControls: this.fb.array([this.fb.control('')])
  });

  get newControls(){
    return this.frmRegister.get('newControls') as FormArray;
  }

  public AddPhoto() {
    this.newControls.push(this.fb.control(''));
  }

  public RemovePhoto(i){
    this.newControls.removeAt(i);
  }

  ngOnInit(): void {
 }

}
```

Builderdemo.component.html

```
<div class="container-fluid">
 <div class="row">
  <div class="col-3">
    <h3>Register Product</h3>
    <form [formGroup]="frmRegister">
        <fieldset>
            <legend>Basic Info</legend>
            <dl>
                <dt>Name</dt>
                <dd>
                    <input formControlName="Name" type="text" class="form-control">
                </dd>
                <dt>Price</dt>
                <dd>
                    <input formControlName="Price" type="text" class="form-control" >
                </dd>
            </dl>
        </fieldset>
        <fieldset>
            <legend>Stock Details</legend>
            <div formGroupName="frmDetails">
                <dl>
                    <dt>City</dt>
                    <dd>
                        <select formControlName="City" class="form-control">
                            <option>Delhi</option>
                            <option>Hyd</option>
                        </select>
                    </dd>
                    <dt>InStock</dt>
                    <dd>
                        <input formControlName="InStock" type="checkbox">
                    </dd>
                </dl>
            </div>
            <div>
            <h2>Upload Photo <button (click)="AddPhoto()" class="btn
btn-link">Add More</button> </h2>
        <div *ngFor="let control of newControls.controls; let i=index"
style="margin-top: 20px;" >
          <input type="file" formControlName="i" >
          <button (click)="RemovePhoto(i)" class="btn btn-
link">Remove</button>
        </div>
      </div>
    </fieldset>
  </form>
 </div>
 <div class="col-9">
  <h3>Details</h3>
  <dl>
    <dt>Name</dt>
    <dd>{{frmRegister.value.Name}}</dd>
    <dt>Price</dt>
    <dd>{{frmRegister.value.Price}}</dd>
    <dt>City</dt>
    <dd>{{frmRegister.value.frmDetails.City}}</dd>
    <dt>Stock</dt>
```

```
        <dd>
            {{(frmRegister.value.frmDetails.InStock==true)?"Available":"Out of Stock"}}
        </dd>
    </dl>
  </div>
 </div>
</div>
```

Validating Input in reactive forms

- In a reactive form component class is “source of truth”.
- Instead of adding validator through attribute in template. You can configure them in controller class.
- Angular will call the validator functions whenever the value changes.
- It verifies that the input value is according to the validator defined and returns Boolean true or false.
- Angular provides a set of built-in validator and also allows to configure custom validator functions.
- The built-in validators are derived from “Validators” class.
- The commonly used validator functions are:
    - min()
    - max()
    - required()
    - requiredTrue()
    - email()

  - minLength()
  - maxLength()
  - pattern()
  - nullValidator()
  - compose()
  - composeAsync()

FAQ: What are Sync and Async Validators?

- **Sync validators** are Synchronous functions that take a control instance (object) and immediately return a set of validation errors or null.
- **Async validators** are Asynchronous functions that take a control instance and return a Observable which emits the result later as per the situation.
- **Angular by default uses "Async" validators.**

Syntax:

public txtName = new FormControl('Value', [Validators])

<input validator>

Ex:

Reactivevalidation.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';

@Component({
  selector: 'app-reactivevalidation',
  templateUrl: './reactivevalidation.component.html',
  styleUrls: ['./reactivevalidation.component.css']
})
export class ReactivevalidationComponent implements OnInit {

  public frmRegister: FormGroup;
  public submitted = false;
  constructor(private fb: FormBuilder) { }

  ngOnInit(): void {
    this.frmRegister = this.fb.group({
      UserName: ['', [Validators.required, Validators.minLength(4)]],
      Mobile: ['', Validators.required],
      Email: ['', [Validators.required, Validators.email]],
    });
  }
  get frm() {
    return this.frmRegister.controls;
  }
  public OnSubmit() {
    this.submitted = true;
    if (this.frmRegister.invalid) {
      return;
    }
    alert('Registered Successfully..');
  }
}
```

Reactivevalidation.component.html

```
<div class="container-fluid">
  <h2>Register User</h2>
  <form [formGroup]="frmRegister" (ngSubmit)="OnSubmit()" >
    <div class="form-group">
        <label>User Name</label>
        <div>
            <input type="text" formControlName="UserName" class="form-
control">
            <div *ngIf="submitted && frm.UserName.errors" class="text-
danger">
              <span *ngIf="frm.UserName.errors.required">User Name
Required</span>
              <span *ngIf="frm.UserName.errors.minlength">Name too
Short..</span>
```

```
            </div>
        </div>
    </div>
    <div class="form-group">
        <label>Mobile</label>
        <div>
            <input type="text" formControlName="Mobile" class="form-
control">
            <div *ngIf="submitted && frm.Mobile.errors" class="text-
danger">
              <span *ngIf="frm.Mobile.errors.required"> Mobile
Required</span>
            </div>
        </div>
    </div>
    <div class="form-group">
        <label>Email</label>
        <div>
            <input type="text" formControlName="Email" class="form-
control">
            <div *ngIf="submitted && frm.Email.errors" class="text-
danger">
```

```
            <span *ngIf="frm.Email.errors.required">Email
Required</span>
            <span *ngIf="frm.Email.errors.email">Invalid Email</span>
          </div>
        </div>
    </div>
    <div class="form-group">
        <button class="btn btn-primary btn-block">Register</button>
    </div>
  </form>
</div>
```

Routing in Angular

- Routing is a technique introduced into Web Application development.
- Routing make URL more SEO and User friendly.
- SEO friendly URL can exactly identify the user location and prompt the suggestions based on the browsing history.
- User friendly URL allows the user to reach any topic just with one click.
- Routing can load new contents into page without reloading the complete page.
- SPA and Progressive web applications require routing.

- User will stay on one page and gets access to everything on to the page.
- Routing is implemented both server-side and client-side.
- Angular implements routing in client-side application.

Angular Routing Library

- @angular/router is a package that provides the following modules for configuring the routes and exporting the routes for application.
  - RouterModule
  - Routes
- The routes for Angular application are defined in “app-routing.module.ts”

Routes	The routes object is responsible for configuring routes for your application. It is a collection of routes for your application.
RouterModule	It imports the routes and exports the routes into a route table. When ever the client request comes it verifies from route table and renders.

Syntax:

“app-routing.module.ts”

```
 import { NgModule } from ‘@angular/core’;
 import { Routes, RouterModule } from ‘@angular/router’;
const routes: Routes = [ { }, { } ];
@NgModule({
        Imports: [RouterModule.forRoot(routes)],
         Exports: [RouterModule]
   })
 export class AppRoutingModule { }
```

- Route Collection comprises a set of route objects.
- Every route object comprises of following properties

Property	Description
Path	- It is the request path used to access any resource in application. It specifies the request name. Syntax: { path: 'home' } Request: http://localhost:4200/home - Path can be configured with wild card routes. {path: ' ' } – If no component is requested then what to do? {path: '**'} – If there is no component path that you requested. The component you requested is not found then what to do?
component	It refers the component to be rendered when any specific path is requested by client.

	Syntax: { path: 'home', component: HomeComponent }
redirectTo	It specifies the path for redirection. It can use an existing path and redirect to that path when the condition or situation matches.
pathMatch	It specifies the URL to access when requested path is similar to existing path. No need to re-define the path. - Full: to match all details defined - Prefix: to match only component.
children[]	It is a collection child routes.

- The hyperlinks used for route navigation in UI are defined by using "routerLink" property
  Syntax:
  <a routerLink="pathname"> Link Text / Image </a>
- The target location where you want to render the component is defined by using <router-outlet>.
- Outlet defines the location where the resulting component should be rendered.
- Every page can use only one <router-outlet>

Ex:

- Create a new Application
  > ng generate application flipkart
- Ignore routing module by selecting "No"
- Add following components
  - ng g c home

  - ng g c electronics
  - ng g c footwear
  - ng g c fashion
  - ng g c notfound

- home.component.html

```
<div>
   <h2>Flipkart - Home</h2>
   <p>Something about flipkart</p>
</div>
```

- electronics.component.html

```
<div>
   <h2>Electronics Home</h2>
   <img src="assets/tv.jpg" width="100" height="100">
   <img src="assets/mobile.jpg" width="100" height="100">
</div>
```

- Similarly design fashion, footwear components.
- notfound-component.html

```
<div>
   <h2>Page you requested - Not Found</h2>
</div>
```

- Go to “app” folder and add a new file by name app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes  } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { ElectronicsComponent } from
'./electronics/electronics.component';
```

```
import { FashionComponent } from
'./fashion/fashion.component';
import { FootwearComponent } from
'./footwear/footwear.component';

const routes: Routes = [
  {path: 'home', component: HomeComponent},
  {path: 'electronics', component: ElectronicsComponent},
  {path: 'footwear', component: FootwearComponent},
  {path: 'fashion', component: FashionComponent},
  {path: '', redirectTo: 'home', pathMatch: 'full'},
  {path: '**', component: NotfoundComponent }
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

- Register the routing module in “app.module.ts”

```
imports: [
  AppRoutingModule
  ],
```

- Go to “app.component.html”

```
<div class="container-fluid">
 <h2 class="text-center text-primary"> <span class="fa fa-
shopping-cart"></span> Flipkart Shopping</h2>
 <div class="row" style="margin-top: 50px;">
```

```
  <div class="col-3">
     <ul class="list-unstyled">
       <li><a routerLink="home" class="btn btn-danger btn-block">
<span class="fa fa-home"></span> Home</a></li>
       <li><a routerLink="electronics" class="btn btn-danger btn-
block"> <span class="fa fa-tv"></span> Electornics</a></li>
       <li><a routerLink="footwear" class="btn btn-danger btn-
block"> <span class="fa fa-shoe-prints"></span>
Footwear</a></li>
       <li><a routerLink="fashion" class="btn btn-danger btn-block">
<span class="fa fa-tshirt"></span> Fashion</a></li>
     </ul>
  </div>
  <div class="col-9">
     <router-outlet></router-outlet>
  </div>
 </div>
</div>
```

Route Parameters

- Web Application use “Query String” to transport data across requests.

  Page? Name=value & password=value

- Query String is replaced with Route Parameters.
- Route Parameters are safe when compared to query string.

- Route parameters provide an easy technique to query any content directly from URL.

  Query String
  Shopping.html?category=electronics&subcategory=mobile&model=Samsung

  Route Parameters
  Shopping/electronics/mobile/Samsung

- Route parameters are configured in route path

  {path: ‘electronics/:param1/:param2..’}

- Actual values are passed into Route parameters from the URL

  URL: http://localhost:4200/electronics/value1/value2

  param1=value1

  param2=value2

- You can access and use the parameters in any component by using an object derived from “ActivatedRoute”

Syntax:

```
private route: ActivatedRoute
this.route.snapshot.paramMap.get("routeParameterName");
```

Summary:

- Configure Route Parameters
  Syntax:
  {path: 'search/:id/:name/:price', component: SearchComponent},
- Pass values into parameters
  http://localhost:4200/search/1/tv/23000
  <a routerLink="search/1/tv/23000> Search </a>
- Access and use parameters
  public id = this.route.snapshot.paramMap.get('id')

Ex: Route Parameters

- Add a new service into project
  > ng g service data –skipTests
- data.service.component.ts

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

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

  constructor() { }
  public GetCategories(){
    return [
      {CategoryId: 1, CategoryName: 'Electronics'},
      {CategoryId: 2, CategoryName: 'Footwear'}
    ];
  }
  public GetProducts(){
    return [
      {ProductId: 1, Name: 'JBL Speaker', Price: 4500.44, CategoryId:
1},
      {ProductId: 2, Name: 'Earpods', Price: 2500.44, CategoryId: 1},
      {ProductId: 3, Name: 'Nike Casuals', Price: 5500.44,
CategoryId: 2},
      {ProductId: 4, Name: 'Lee Cooper Boot', Price: 6500.44,
CategoryId: 2},
    ];
  }
}
```

- Add following components

```
> ng g c categorieslist --skipTests
> ng g c productslist --skipTests
```

- Categorieslist.component.ts

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


@Component({
  selector: 'app-categorieslist',
  templateUrl: './categorieslist.component.html',
  styleUrls: ['./categorieslist.component.css']
})
export class CategorieslistComponent implements OnInit {

  public categories = [];
  constructor(private data: DataService ) { }

  ngOnInit(): void {
    this.categories = this.data.GetCategories();
  }

}
```

- Categorieslist.component.html

```
<div>
    <h2>Categories List</h2>
    <ol>
        <li *ngFor="let item of categories">
            <a
routerLink="{{item.CategoryId}}">{{item.CategoryName}}</a>
        </li>
    </ol>
</div>
```

- Productslist.component.ts

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

```
@Component({
  selector: 'app-productslist',
  templateUrl: './productslist.component.html',
  styleUrls: ['./productslist.component.css']
})
export class ProductslistComponent implements OnInit {

  public categoryId;
  public products = [];
  constructor(private data: DataService, private route:
ActivatedRoute) { }

  ngOnInit(): void {
    this.categoryId = this.route.snapshot.paramMap.get('id');
    this.products =
this.data.GetProducts().filter(x=>x.CategoryId==this.categoryId);
  }

}
```

- Productslist.component.html

```
<div>
    <h2>Products List</h2>
    <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>
          <a routerLink="/categories">Back to Categories</a>
        </div>
      </div>
```

- App-routing.module.ts

```
  {path: 'categories', component: CategorieslistComponent},
  {path: 'categories/:id', component: ProductslistComponent},
```

- App.component.html

```
<li><a routerLink="categories" class="btn btn-danger btn-block"
>Categories</a></li>
```

Configuring Child Routes

- Routes by default handle navigation between component.
- Child routes are used to define navigation within the component.
- A route collection can be maintained within the context of existing route.
- The child routes are defined by using “children” collection.

Syntax:

```
const routes: Routes = [
        {
         path: ‘parent’, component: ParentComponent,
                    children: [
                         {path: ‘child’, component: ChildComponent}
                    ]
        }]
```

- Redirection within component is configured by using “navigate()” or “RouterModule”.

  Syntax:
  ```
  private router: Router;
  this.router.navigate([‘path’, parameters], { relativeTo: this.route })
  ```

  relativeTo: this.route: Specifies navigation within the parent.

Ex:

- Go to “data.service.ts”
  ```
  public GetProducts(){
    return [
      {ProductId: 1, Name: 'JBL Speaker', Price: 4500.44, CategoryId: 1, Photo: 'assets/speaker.jpg', Description: 'Something about JBL Speaker and Its Features'},
      {ProductId: 2, Name: 'Earpods', Price: 2500.44, CategoryId: 1, Photo: 'assets/earpods.jpg', Description: 'Something about Earpods and Its Features'},
  ```

```
    {ProductId: 3, Name: 'Nike Casuals', Price: 5500.44, CategoryId: 2, Photo: 'assets/shoe.jpg', Description: 'Something about Nike Casuals and Its Features'},
    {ProductId: 4, Name: 'Lee Cooper Boot', Price: 6500.44, CategoryId: 2, Photo: 'assets/shoe1.jpg', Description: 'Something about Lee Boot and Its Features'},
   ];
  }
```

- Add a new component

  > ng g c productdetails

- Go to “app-routing.module.ts”

```
{path: 'categories/:id', component: ProductslistComponent,
      children: [
        {path: 'details/:id', component: ProductdetailsComponent}
      ]
   },
```

- Go to “productslist.component.ts”

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router, RouterModule } from
'@angular/router';
import { DataService } from '../data.service';

@Component({
  selector: 'app-productslist',
  templateUrl: './productslist.component.html',
  styleUrls: ['./productslist.component.css']
})
```

```
export class ProductslistComponent implements OnInit {

  public categoryId;
  public products = [];
  constructor(private data: DataService, private route:
ActivatedRoute, private router: Router) { }

  ngOnInit(): void {
    this.categoryId = this.route.snapshot.paramMap.get('id');
    this.products =
this.data.GetProducts().filter(x=>x.CategoryId==this.categoryId);
  }
  public DetailsClick(item){
    this.router.navigate(['details', item.ProductId],{relativeTo:
this.route});
  }
}
```

- Go to “productslist.component.html”

```
<div>
   <div class="row">
     <div class="col-6">
        <h2>Products List</h2>
        <table class="table table-hover">
            <thead>
                <tr>
                    <th>Name</th>
                    <th>Price</th>
                    <th>View Details</th>
```

```
          </tr>
        </thead>
        <tbody>
          <tr *ngFor="let item of products">
            <td>{{item.Name}}</td>
            <td>{{item.Price}}</td>
            <td><button (click)="DetailsClick(item)" class="btn btn-
link">Details</button>
              <a routerLink="details/{{item.ProductId}}">Details
Link</a>
            </td>
          </tr>
        </tbody>
      </table>
      <div>
        <a routerLink="/categories">Back to Categories</a>
      </div>
    </div>
    <div class="col-6">
      <router-outlet></router-outlet>
    </div>
  </div>
</div>
```

- Go to “productdetails.component.ts”

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

```
@Component({
  selector: 'app-productdetails',
  templateUrl: './productdetails.component.html',
  styleUrls: ['./productdetails.component.css']
})
export class ProductdetailsComponent implements OnInit {

  public productid;
  public searchedproduct;
  constructor(private data: DataService, private route:
ActivatedRoute) { }

  ngOnInit(): void {
    this.productid = this.route.snapshot.paramMap.get('id');
    this.searchedproduct =
this.data.GetProducts().find(x=>x.ProductId==this.productid);
  }

}
```

- Go to “productdetails.component.html”

```
<div class="card">
    <h2>{{searchedproduct.Name}}</h2>
    <div class="card-img-top">
        <img [src]="searchedproduct.Photo" width="100">
    </div>
    <div class="card-body">
        <p>
```

```
            {{searchedproduct.Description}}
        </p>
    </div>
</div>
```

Lazy Loading of Routes

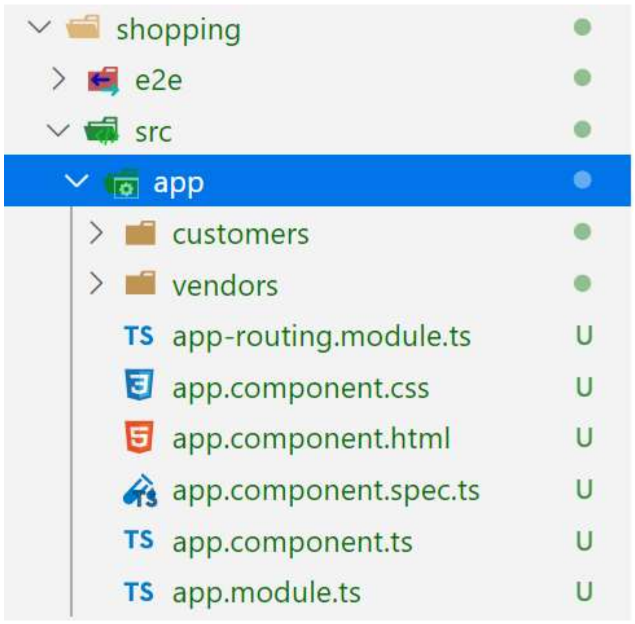
- Eager Loading
- Lazy Loading
- All Angular modules “NgModules” of application are eagerly loaded.
- Modules are loaded along with application.
- It makes application heavy on browser.
- It makes page rendering slow.
- Even when you are not using any specific module, it is loaded into memory.
- Lazy loading is design pattern.
- Lazy loading allows to load “NgModules” only when they are required.
- It keeps initial bundle size smaller.
- It improves the render time.
- It decreases the page load time.
- It is more light weight on browser.
- It can reach broad range of devices. [Mobile to Browser].

Ex:

- Create a new application
  ```
  > ng g application shopping --routing
  ```

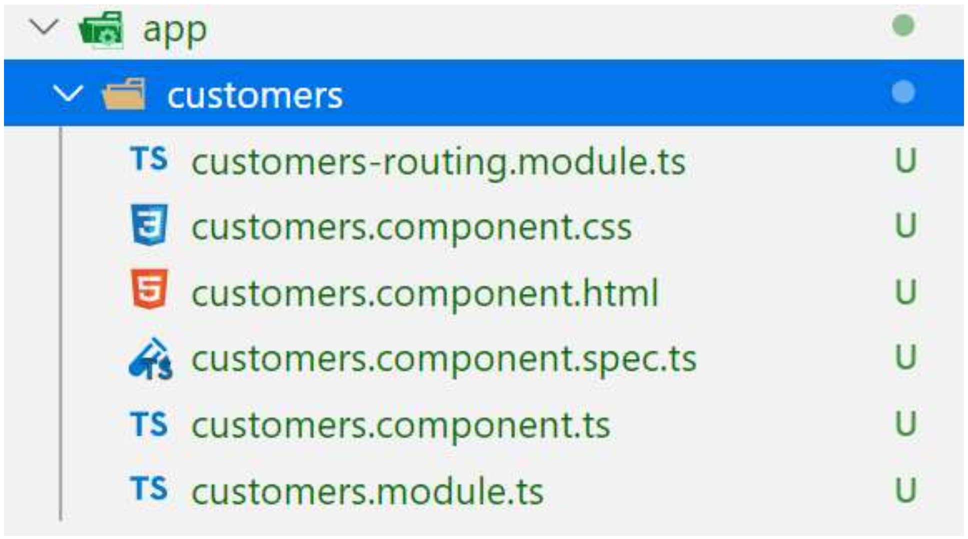
- Open “shopping – app” folder in integrated terminal
- Generate the following modules

```
> ng g module vendors --route vendors --module app.module
> ng g module customers --route vendors --module app.module
```



```
shopping
  e2e
  src
    app
      customers
      vendors
      app-routing.module.ts        U
      app.component.css            U
      app.component.html           U
      app.component.spec.ts        U
      app.component.ts             U
      app.module.ts                U
```

- Every module comprises of following files



```
app
  customers
    customers-routing.module.ts        U
    customers.component.css            U
    customers.component.html           U
    customers.component.spec.ts        U
    customers.component.ts             U
    customers.module.ts                U
```

Note: Every module act as an application within the root application

- Every module comprises of routes loaded forChild().
  Syntax:
  customers-routing.module.ts

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

import { CustomersComponent } from './customers.component';

const routes: Routes = [{ path: '', component:
CustomersComponent }];

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

Note: The lazy routes for modules are configured by using "loadChildren()" in the route configuration.

Syntax:

{path: 'moduleName', loadChildren: (import the module).then(load the module) }

Ex:

app-routing.module.ts

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

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

```
const routes: Routes = [
{ path: 'vendors', loadChildren: () =>
import('./vendors/vendors.module').then(m => m.VendorsModule) },
{ path: 'customer', loadChildren: () =>
import('./customers/customers.module').then(m =>
m.CustomersModule) }
];


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

- Go to “app.component.html”

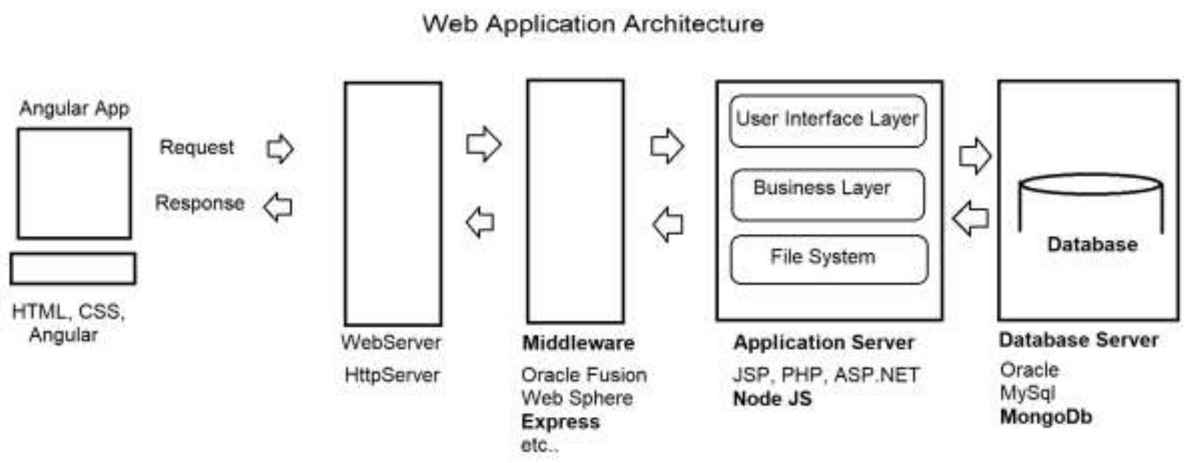
```
<h2>
  Amazon Shopping
</h2>
<div>
  <button routerLink="">Home</button>
  <button routerLink="/customer">Customers</button>
  <button routerLink="/vendors">Vendors</button>
```

```
</div>
<div style="margin-top: 50px;">
  <router-outlet></router-outlet>
</div>
```

You can implement lazy loading for components by configuring lazy route:

```
{path: 'electronics', loadChildren: ()=>
import('./electronics/electronics.component').then(c =>
c.ElectronicsComponent) },
```

MEAN Stack – for End to End Application



Database

(MongoDb)

- Mongo is a cloud service provider.

- MongoDb document-oriented database.
- JavaScript Based
- Non-Sql Database
- Non-RDBMS
- Open Source
- Cross Platform

RDBMS	MongoDb
Database	Database
Table	Collection
Row	Document
Column/Field	Field
Join	Embedded Document

Setup Environment for MongoDb

- Download and install MongoDb Database Server.
  https://www.mongodb.com/try/download/community
- Start MongoDB Database Server
  - Go to “services.msc” in window run option
  - Right Click on “MongoDB Database Server” and “start”
- Open MongoDB Client to configure, create and manipulate database.
  - Open Command Prompt
  - Change to the following location and execute the command
    C:\Program Files\MongoDB\Server\4.0\bin> mongo.exe
  - This will connect to MongoDB server
    mongodb://127.0.0.1:27017

MongoDB Commands [Case Sensitive]

Command	Description
show dbs	To view the list of all databases.
db	To view the active database.
use	To switch to existing database or to create a new database. Syntax: > use database_name > use angular10db
db.createCollection()	To create a new database table. Syntax: > db.createCollection("tableName", options) > db.createCollection("tblproducts") > db.createCollection("tblcategories")
show collections	To View the list of tables
db.collectionName.insert()	To insert records (documents) into database table. Syntax: > db.collectionName.insert({}) one record > db.collectionName.insert([{}, {}]) multiple The datatypes for MongoDB are similar to JavaScript datatype. Ex: db.tblcategories.insert({CategoryId:1, CategoryName:"Electronics"})

	db.tblcategories.insert([{CategoryId:2, CategoryName:"Footwear"}, {CategoryId:3, CategoryName:"Fashion"}])
	
	Ex:
	db.tblproducts.insert([{Name: "Samsung TV", Price: 34000.55, InStock:true, Photo: "assets/tv.jpg", CategoryId:1}, {Name: "JBL Speaker", Price: 4000.55, InStock:true, Photo: "assets/speaker.jpg", CategoryId:1}, {Name: "Nike Casuals", Price: 3000.55, InStock:true, Photo: "assets/shoe.jpg", CategoryId:2}, {Name: "Shirt", Price: 1000.55, InStock:true, Photo: "assets/shirt.jpg", CategoryId:3}])

db.collectionName.find({})	To view the records in database table. > db.tblproducts.find() > db.tblproducts.find().pretty()

Server Side – Node JS

- Server-side scripting is a technique used in web development, where scripts are employed on server in order generate a response customized to every client request.

- Node JS is an open source, cross platform JavaScript based server-side scripting.
- Asynchronous
- No Buffering
- Single Threaded
- Node JS uses “JavaScript Programs” server-side.
- Server-Side Node JS programs will have extension “.js”

Nodes JS Server Side Program Approach

- Install the library required.
  Syntax:
  > npm install packageName
- Import the library into program by using “require()”.
  Syntax:
  var ref = require(“moduleName”);
- Access the library members by creating a reference.
- Implement the functionality.

Node JS connecting with MongoDB

- Install MongoDB library for handling communication with MongoDB database.
  C:\projects-workspace> npm install mongodb
- Create a new folder “server-side” in your workspace.
- Add a new JavaScript file into folder
  testconnection.js
  //Import MongoDB Client library
  var mongoClient = require("mongodb").MongoClient;

```
//MongoDB Connection String
var url = "mongodb://127.0.0.1:27017";

//Connecting with MongoDB server
mongoClient.connect(url, function(err,clientObj){
   if(!err){
      console.log(`Connected...`);
   } else {
      console.log(err);
   }
})
```

- Compile and run form terminal or command prompt

```
> node testconnection.js
```

Connecting and Reading Documents from MongoDB Collection

Ex: testconnection.js

```
//Import MongoDB Client library

var mongoClient = require("mongodb").MongoClient;


//MongoDB Connection String

var url = "mongodb://127.0.0.1:27017";


//Connecting with MongoDB server

mongoClient.connect(url, function(err,clientObj){
  if(!err){
    var database = clientObj.db("angular10db");
    database.collection("tblproducts").find().toArray(function(err,
documents){
      if(!err){
        console.log(documents);
      } else {
        console.log(err);
      }
    })
  } else {
    console.log(err);
  }
})
```

Connecting and Inserting Documents into MongoDB Collection

Ex:

```
var mongoClient = require("mongodb").MongoClient;
var url = "mongodb://127.0.0.1:27017";

mongoClient.connect(url, function(err, clientObj){
  if(!err){
        var database = clientObj.db("angular10db");
        var data = {
            Name: "Fossil Watch",
            Price: 45000.55,
            InStock: true,
            Photo: 'assets/fossil.jpg',
            CategoryId:1
        };
        database.collection("tblproducts").insertOne(data, function(err, result){
            if(!err) {
                console.log("Record Inserted");
            }
        })
    } else {
        console.log(err);
    }
})
```

API and Service

- Distributed computing Architecture
- A distributed computing Architecture allows two applications running or two different machines to share information. (or) Two

applications running on same machine but on different process can share information.

- Various Technologies
  - CORBA
  - DCOM
  - RMI
  - EJB
  - Web Services
  - Remoting etc.
- Create an application that can reach broad range of devices i.e from a browser to mobile.
- Enable communication between client and server application for sharing information.
- API and Web Service Specifications
  - SOAP
  - REST
  - JSON
- SOAP [Service Oriented Architecture Protocol]
  - Client Request will be in XML
  - Server Response will be in XML
- REST [Representational State Transfer]
  - Client Request will be a simple query [Query String]
  - Response will be in XML or JSON
- JSON [JavaScript Object Notation]
  - Client Request will be in JSON
  - Server Response will be JSON
- XML & JSON
  - Work Offline

    - Transport data without COM Marshalling [Converting Binary to Object -Object to Binary]
    - XML cross platform
    - JSON is cross platform, Native to browser.

Middleware

- Express.js or Express
- It is a back-end web application framework for Node.js.
- Open Source, Cross platform.
- Allows communication between client and server-side application.
- It is responsible for parsing the data, locating the static files requested by client and serving the files to client, handling CORS etc.
- Express listens from server and process requests made by client.
- Client request can be
    - GET – To fetch resources from server
    - POST- To submit resource to server
    - PUT – To modify the resources on server
    - DELETE – To remove the resource from server
- Server-Script scripting handling "Server-side Objects"
    - Request Object:
      It provides a set of properties and methods that are used by server that accept request from client and fetch the resource data from server.
      Server uses request object to fetch the client details, like query string, cookie, form body.

- Response Object:
  It provides a set of properties and methods that are used by server in order to send response to client. [HTML, File, JSON, XML etc]
- Application Object
- Session Object
- Cookie Object

Creating Web API using Express

- Install “express” library for your project.
  ```
  > npm install express
  ```
- Add a new JavaScript file “api.js”

```
var express = require("express");

var app = express();

app.get("/", function(request, response){
  response.send("Welcome to API");
});
app.get("/getproducts", function(req, res){
   res.send([{"Name":"TV"},{"Name":"Mobile"}]);
});
app.get("/getdetails/tv", function(req, res){
   res.send({"Name":"TV", Price:23500.44});
});
app.post("/addproduct", function(req, res){
   res.send("POST - This is request for data submit");
});
```

```
app.put("/updateproduct", function(res, res){
   res.send("PUT - This is request to modify data");
});
app.delete("/deleteproduct", function(res, res){
   res.send("DELETE - This is request to Delete data");
});
app.listen(8080);
console.log("Server Started and Listening on : http://127.0.0.1:8080");
```

- Test from browser or any web debugger like: fiddler, postman etc.
  http://127.0.0.1:8080/getproducts

Ex:

Api.js

```
const { response } = require("express");

var express = require("express");

var mongoClient = require("mongodb").MongoClient;


var url = "mongodb://127.0.0.1:27017";


var app = express();


app.get("/", function(request, response){

  response.send("Welcome to API");

});
```

```
app.get("/getproducts", function(req, res){
   mongoClient.connect(url, function(err,clientObj){
      if(!err){
         var database = clientObj.db("angular10db");
         database.collection("tblproducts").find().toArray(function(err,
documents){
            if(!err){
               res.send(documents);
            } else {
               console.log(err);
            }
         })
      }else {
         console.log(err);
      }
   })
});
app.get("/getdetails/tv", function(req, res){
   res.send({"Name":"TV", Price:23500.44});
});
app.post("/addproduct", function(req, res){
    res.send("POST - This is request for data submit");
});
app.put("/updateproduct", function(res, res){
    res.send("PUT - This is request to modify data");
});
app.delete("/deleteproduct", function(res, res){
    res.send("DELETE - This is request to Delete data");
});
app.listen(8080);
console.log("Server Started and Listening on : http://127.0.0.1:8080");
```

- Request for accessing data from MongoDB
  http://127.0.0.1:8080/getproducts

Body Parser

- It is a library required to parse the data into JSON.
- It uses URL encoding and converts the form data into JSON in order to submit to the server.
- Download and Install body parser for your server-side application.
  > npm install body-parser

Syntax:

```
var bodyParser  = require("body-parser");
```

```
app.use(bodyParser.urlencoded({
        extendend:true
}))
app.use(bodyParser.json());
```

CORS Configuration

- Cross Origin Resource Sharing
- Server Application and Client Application are running of different port numbers.
- Sharing information across the origin will be blocked.
- You have to allow sharing across origin for different types of requests.
- You can manage by configure CORS in your middleware.
- Express CORS configuration comprises of following details

Syntax:

```
app.use(function(req, res, next){
    res.header("Access-Control-All-Origin", "*");
    res.header("Access-Control-Allow-Headers", "Origin, x-
Requested-With, Content-Type, Accept");
    res.header("Access-Control-Allow-Method", "GET", "POST",
"PUT", "DELETE");
    next();
})
```

Create Business Layer to Handle Communication Between Client Side and Server-Side Application

- You have to install the following library
  - MongoDB
    > npm install mongodb
  - Express
    > npm install express
  - Body Parser
    > npm install body-parser
- Create a server-side Business Logic to handle interaction with database in backend and client-side application.

RESTapi.js

```
const { RequiredValidator } = require("@angular/forms");


//import library

var mongoClient = require("mongodb").MongoClient;

var express = require("express");

var bodyParser = require("body-parser");


//Configure MongoDB Connection String

var url = "mongodb://127.0.0.1:27017";
```

```
//Configure Middleware
var app = express();

//Configure Body Parser
app.use(bodyParser.urlencoded({
   extended:true
}))
app.use(bodyParser.json());

//Configure CORS
app.use(function(req, res, next){
   res.header("Access-Control-All-Origin","*");
   res.header("Access-Control-Allow-Headers","Origin, x-Requested-
With, Content-Type, Accept");
   res.header("Access-Control-Allow-
Methods","GET","POST","PUT","DELETE");
   next();
});

app.get("/getproducts", function(req, res){
   mongoClient.connect(url, function(err, clientObj){
        if(!err){
            var database = clientObj.db("angular10db");
            database.collection("tblproducts").find().toArray(function(err,
documents){
                if(!err){
                    res.send(documents);
                } else {
                    console.log(err);
                }
            })
        }else {
            console.log(err);
        }
    })
});
app.post("/addproducts", function(req, res){
    mongoClient.connect(url, function(err, clientObj){
        if(!err) {
            var database = clientObj.db("angular10db");
            var data = {
                Name:req.body.Name,
            Price:req.body.Price,
            InStock:req.body.InStock,
            Photo: req.body.Photo,
            CategoryId: req.body.CategoryId
         }
         database.collection("tblproducts").insertOne(data, function(err,
result){
            if(!err){
               console.log("Record Inserted");
            } else {
               console.log(err);
            }
         })
      } else {
         console.log(err);
      }
   })
});
app.listen(8080);
console.log("Server Listening : http://127.0.0.1:8080");
```

Consume in Angular

- Angular application uses a JavaScript Library "RxJS" for handling asynchronous events.
- Angular application uses "HttpClient" module that provides the properties and methods required for handling requests from API.
- Angular is integrated with "RxJS" library

RxJS

- RxJS is a JavaScript library.
- It is used for configuring "asynchronous" events.
- Asynchronous in RxJS is a collection of events that are executed with non-blocking technique.
- RectiveX uses software design patterns like "Observer" and "Iterator".
- Observer is a software design pattern, it is a communication pattern under "Behavioural Patterns". [Event uses Observer pattern].
- The "RxJS" components required for "async" programming

Component	Description
Observable	- It is a collection of events or values. - These events are intended to use on asynchronous calls. - Events are invoked using async techniques. - Values are accessed by using async techniques.
Observer	- It is a collection of call backs.

	- These callbacks knows how to listen to the values returned by observable. - Notified with the changes. - It knows about the type of values delivered by observable.
Subscription	- It is responsible for executing the observable. - It is also responsible for cancelling the execution.
Operators	- They are representing "Iterator" pattern. - This is used in handling collections in application. - Operations over collection is managed by operators. Ex: map, filter etc.
Subject	- It is like Event Emitter. - It emits the value on specific event.
Schedulers	- They are used to schedules the tasks. - They uses "setTimeout()" - It can keep the task waiting for specific duration of time.

Syntax:

```
Observable().subscribe(function(result){
        Reference = result;
})
```

HttpClient Library

- Angular is provided with “http” library.
- You can access by using “HttpClientModule”.
- HttpClientModule provides a set of properties and methods that are used by client-side application to handle communication with server side “API”.
- It internally uses “XmlHttpRequest” object of JavaScript.
- JQuery uses “Ajax(), getJSON() etc” [xhr]
- Angular JS uses “$http()” [xhr]
- Angular uses “HttpClientModule”
- It is defined in “@angular/common/http”

  Syntax:

  ```
  private http: HttpClient;
  http.get();
  http.post();
  ```

Accessing Data from API and Consuming in Angular Application

- Create a new API on Server

  “ServerSide/restapi.js”

  ```
  const { RequiredValidator } = require("@angular/forms");

  //import library
  var mongoClient = require("mongodb").MongoClient;
  var express = require("express");
  var bodyParser = require("body-parser");

  //Configure MongoDB Connection String
  var url = "mongodb://127.0.0.1:27017";
  ```

```
//Configure Middleware
var app = express();

//Configure Body Parser
app.use(bodyParser.urlencoded({
   extended:true
}))
app.use(bodyParser.json());

//Configure CORS
app.use(function(req, res, next){
   res.header("Access-Control-All-Origin","*");
   res.header("Access-Control-Allow-Headers","Origin, x-
Requested-With, Content-Type, Accept");
   res.header("Access-Control-Allow-
Methods","GET","POST","PUT","DELETE");
   next();
});

app.get("/getproducts", function(req, res){
   mongoClient.connect(url, function(err, clientObj){
      if(!err){
         var database = clientObj.db("angular10db");

database.collection("tblproducts").find().toArray(function(err,
documents){
            if(!err){
               res.send(documents);
```

```
                } else {
                    console.log(err);
                }
            })
        }else {
            console.log(err);
        }
    })
});
app.post("/addproducts", function(req, res){
    mongoClient.connect(url, function(err, clientObj){
        if(!err) {
            var database = clientObj.db("angular10db");
            var data = {
                Name:req.body.Name,
                Price:req.body.Price,
                InStock:req.body.InStock,
                Photo: req.body.Photo,
                CategoryId: req.body.CategoryId
            }
            database.collection("tblproducts").insertOne(data,
function(err, result){
                if(!err){
                    console.log("Record Inserted");
                } else {
                    console.log(err);
                }
            })
        } else {
            console.log(err);
        }
    })
});
app.listen(8080);
console.log("Server Listening : http://127.0.0.1:8080");
```

- Create a new Angular Application

```
> ng g application demoapp
```

- Create “Contracts” folder in “app” folder
- Add following files

IProduct.ts

```
interface IProduct {
    Name: string;
    Price: number;
    InStock: boolean;
    Photo: string;
    CategoryId: number;
}
```

ICategory.ts

```
interface ICategory {
    CategoryId: number;
    CategoryName: string;
}
```

- Add “Services” folder into “app” folder
- Add a new Service file

apidata.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable, throwError  } from 'rxjs';
import { catchError } from 'rxjs/operators';

@Injectable({
   providedIn: 'root'
})
export class ApidataService {
 constructor(private http: HttpClient){}
 public productsGetUrl = 'http://127.0.0.1:8080/getproducts';

 public GetProducts(): Observable<IProduct[]>{
    return this.http.get<IProduct[]>(this.productsGetUrl);
 }
}
```

- Add a new component
  "productslist.component.ts"

```
import { Component, OnInit } from '@angular/core';
import { ApidataService } from '../Services/apidata.service';

@Component({
  selector: 'app-productslist',
  templateUrl: './productslist.component.html',
  styleUrls: ['./productslist.component.css']
})
export class ProductslistComponent implements OnInit {

  public products = [];
  constructor(private data: ApidataService) { }

  ngOnInit(): void {

this.data.GetProducts().subscribe(product=>this.products=produc
t);
  }

}
```

Productslist.component.html

```
<div class="container-fluid">
    <h2>Products List</h2>
    <table class="table table-hover">
        <thead>
            <tr>
                <th>Name</th>
                <th>Price</th>
                <th>Stock</th>
            </tr>
        </thead>
        <tbody>
            <tr *ngFor="let item of products">
                <td>{{item.Name}}</td>
                <td>{{item.Price}}</td>
                <td>{{item.InStock}}</td>
            </tr>
        </tbody>
    </table>
```

```
</div>
```

- Go to “app.module.ts”

```
import { HttpClientModule } from '@angular/common/http';
imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule
  ],
```

Execution:

- Make sure that your api started
  C:\Projects\ServerSide> node restapi.js
- Start your angular application
- Make sure that browser is allowing CORS.

Inserting Record:

RestApi.js

```
const { RequiredValidator } = require("@angular/forms");


//import library

var mongoClient = require("mongodb").MongoClient;

var express = require("express");

var bodyParser = require("body-parser");


//Configure MongoDB Connection String
```

```
var url = "mongodb://127.0.0.1:27017";

//Configure Middleware
var app = express();

//Configure Body Parser
app.use(bodyParser.urlencoded({
   extended:true
}))
app.use(bodyParser.json());

//Configure CORS
app.use(function(req, res, next){
   res.header("Access-Control-All-Origin","*");
   res.header("Access-Control-Allow-Headers","Origin, x-Requested-
With, Content-Type, Accept");
   res.header("Access-Control-Allow-
Methods","GET","POST","PUT","DELETE");
   next();
});
```

```
app.get("/getproducts", function(req, res){
   mongoClient.connect(url, function(err, clientObj){
      if(!err){
         var database = clientObj.db("angular10db");
         database.collection("tblproducts").find().toArray(function(err,
documents){
            if(!err){
               res.send(documents);
            } else {
               console.log(err);
            }
         })
      }else {
         console.log(err);
      }
   })
});

app.get("/getcategories", function(req, res){
   mongoClient.connect(url, function(err, clientObj){
      if(!err){
            var database = clientObj.db("angular10db");
            database.collection("tblcategories").find().toArray(function(err,
documents){
                if(!err){
                    res.send(documents);
                } else {
                    console.log(err);
                }
            })
        }else {
            console.log(err);
        }
    })
});

app.post("/addproducts", function(req, res){
    mongoClient.connect(url, function(err, clientObj){
        if(!err) {
            var database = clientObj.db("angular10db");
            var data = {
                Name:req.body.Name,
                Price:req.body.Price,
                InStock:req.body.InStock,
                Photo: req.body.Photo,
                CategoryId: req.body.CategoryId
            }
            database.collection("tblproducts").insertOne(data, function(err,
result){
                if(!err){
                    console.log("Record Inserted");
                } else {
                    console.log(err);
                }
            })
        } else {
            console.log(err);
        }
    })
});
app.listen(8080);
console.log("Server Listening : http://127.0.0.1:8080");
```

Apidata.service.ts

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

import { HttpClient, HttpErrorResponse } from
'@angular/common/http';

import { Observable, throwError  } from 'rxjs';

import { catchError } from 'rxjs/operators';


@Injectable({

   providedIn: 'root'

})

export class ApidataService {

 constructor(private http: HttpClient){}

 public productsGetUrl = 'http://127.0.0.1:8080/getproducts';

 public productsPostUrl = 'http://127.0.0.1:8080/addproducts';

 public categoriesGetUrl = 'http://127.0.0.1:8080/getcategories';


 public GetCategories(): Observable<ICategory[]>{

     return this.http.get<ICategory[]>(this.categoriesGetUrl);

 }


 public GetProducts(): Observable<IProduct[]>{
    return this.http.get<IProduct[]>(this.productsGetUrl);
 }
 public AddProduct(data){
    return this.http.post<any>(this.productsPostUrl,
data).pipe(catchError(this.CatchError));
 }
 public CatchError(error: HttpErrorResponse){
     return throwError(error.statusText);
 }
}
```

Productslist.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ApidataService } from '../Services/apidata.service';

@Component({
  selector: 'app-productslist',
  templateUrl: './productslist.component.html',
  styleUrls: ['./productslist.component.css']
})
export class ProductslistComponent implements OnInit {
  public products = [];
  public categories = [];
  constructor(private data: ApidataService) { }

  ngOnInit(): void {
    this.data.GetProducts().subscribe(product=>this.products=product);

this.data.GetCategories().subscribe(category=>this.categories=category
);
  }
  public onFormSubmit(data){
    this.data.AddProduct(data).subscribe(error=>console.log('Something
Went Wrong'));
    alert('Record Inserted');
    location.reload();
  }
}
```

Productslist.component.html

```
<div class="container-fluid">
    <div class="row">
        <div class="col-3">
            <h2>Register Product</h2>
```

```
      <form #frmRegister="ngForm" method="POST"
(submit)="onFormSubmit(frmRegister.value)">
        <dl>
          <dt>Name</dt>
          <dd>
            <input type="text" name="Name" ngModel
#Name="ngModel" class="form-control">
          </dd>
          <dt>Price</dt>
          <dd>
            <input type="text" name="Price" ngModel
#Price="ngModel" class="form-control">
          </dd>
          <dt>In Stock</dt>
          <dd>
            <input type="checkbox" name="InStock" ngModel
#InStock="ngModel"> Yes
          </dd>
          <dt>Photo</dt>
          <dd>
            <input type="text" name="Photo" ngModel
#Photo="ngModel" class="form-control">
```

```
                </dd>
                <dt>Category</dt>
                <dd>
                    <select name="CategoryId" ngModel
#CategoryId="ngModel" class="form-control">
                        <option *ngFor="let item of categories"
[value]="item.CategoryId">
                            {{item.CategoryName}}
                        </option>
                    </select>
                </dd>
            </dl>
            <button class="btn btn-primary btn-block">Register</button>
        </form>
    </div>
    <div class="col-9">
        <h2>Products List</h2>
        <table class="table table-hover">
            <thead>
                <tr>
                    <th>Name</th>
                    <th>Price</th>
                    <th>Stock</th>
                </tr>
            </thead>
            <tbody>
                <tr *ngFor="let item of products">
                    <td>{{item.Name}}</td>
                    <td>{{item.Price}}</td>
                    <td>{{(item.InStock)==true?"Available":"Out of
Stock"}}</td>
                </tr>
            </tbody>
        </table>
    </div>
  </div>
</div>
```

Angular Animations

- Angular provides a library for configuring CSS animation dynamically.
- Angular can implement CSS 2D and 3D animations.
- Angular uses CSS
    - Transition
    - Transform
    - Animate
- Angular animation library is configured with “BrowserAnimationsModule”
- It is provided with “@angular/animations”
- Angular animation methods

Animation Method	Description
trigger()	It is used to configuration animations effects in a component.
state()	It is used to configure animation state. Angular animations have 2 states - Initial State - Final State

	In initial state we define effects for element to apply before transformation. It is represented by using “[void=>*]” In final state we define effects for element to apply after transformation. It is represented by using “[*=>void]” Note: CSS uses them as “@keyframes”
style()	It is used to define the effects to apply. You can use all CSS attributes.
transition()	It configures the state and time for animation.
animation()	It configures the animation duration.

- Animations for any component are configured in the component meta data by using “animations[]”

  Syntax:
  ```
  @Component({
          selector: ‘’,
          templateUrl: ‘’,
          styleUrls: [],
          animations:[]
  })
  ```
  Animations collection uses animation functions.
  Syntax:
  ```
  animations:[
          trigger(‘EffectName’,[
  ```

```
                state(‘initial’, styles({attribute:value, attribute:value})),
                state(‘final’, styles({attribute:value, attribute:value)),
                transition(‘initial=>final/void=>*’, animate(timeInterval)),
                transition(‘final=>initial/*=>void’, animate(timeInterval))
            ]) // Trigger End
        ]
```

You have to apply trigger to any HTML element

<div [@TriggerName/EffectName]> </div>

Ex:

- Add Angular Material to your application
- Import BrowserAnimations Module in “app.module.ts”
- Add a new component

Animationdemo.component.ts

```
import { animate, state, style, transition, trigger } from
'@angular/animations';
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-animationdemo',
  templateUrl: './animationdemo.component.html',
  styleUrls: ['./animationdemo.component.css'],
  animations: [
    trigger('ZoomEffect', [
      state('initial', style({
        width: '200px',
        height: '200px',
        transform: 'rotate(0deg)'
      })),
      state('final', style({
        width: '400px',
        height: '400px',
        transform: 'rotate(360deg)'
      })),
      transition('initial=>final', animate('3000ms')),
      transition('final=>initial', animate('3000ms'))
    ])
  ]
})
export class AnimationdemoComponent implements OnInit {

  public animationState = 'initial';
  public zoomText = 'Zoom In';
  constructor() { }

  ngOnInit(): void {
  }
  public ZoomClick(){
    this.animationState =
(this.animationState=='initial')?'final':'initial';
    this.zoomText = (this.zoomText=='Zoom In')?'Zoom Out':'Zoom
In';
  }
}
```

Animationsdemo.component.html

```
<div class="container-fluid">
```

```
  <h2>Animations</h2>
  <div class="form-group text-center">
    <button (click)="ZoomClick()" class="btn btn-
primary">{{zoomText}}</button>
  </div>
  <div class="form-group">
    <div (mouseover)="ZoomClick()" style="margin:auto; justify-
content: center; align-items: center;">
        <img [@ZoomEffect]="animationState"
src="assets/shirt.jpg">
    </div>
  </div>
</div>
```

Angular Security

- Reporting vulnerabilities
- Preventing XSS [Cross-Site-Scripting] Attacks
- Preventing Request Forgery
- Angular provides built-in protections against web application attacks.
- Angular also provides options to authenticate and authorize users to give access to resources in application.

Preventing Cross-Site-Scripting Attacks (XSS)

- HTML is used when interpreting a value as HTML component. innerHTML = “<b>Hello !</b>”
- Styles is used when binding to CSS into style property.
- URL is used for url properties such as “<a href=’’>”.

- Resources URL as URL that will be loaded when code is executed. <script src="">
- Angular uses "DomSanitizer.Sanitize()" method for handling XSS.
- Angular also provides methods to by-pass the security XSS
    - bypassSecurityTrustHTML
    - bypassSecurityTrustScript
    - bypassSecurityTrustUrl
    - bypassSecurityTrustResourceUrl
    - bypassSecurityTrustStyleUrl

Ex:

Securitydemo.component.ts

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

import { DomSanitizer } from '@angular/platform-browser';

import { timeStamp } from 'console';


@Component({

  selector: 'app-securitydemo',

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

  styleUrls: ['./securitydemo.component.css']

})

export class SecuritydemoComponent implements OnInit {


  public xssUrl = 'javascript:alert("Hello !")';
  public trustedUrl;
  public xssVideoUrl =
'https://www.youtube.com/embed/xDwNeVEIOeU';
  public trustedVideoUrl;
  constructor(private sanitizer: DomSanitizer) { }

  ngOnInit(): void {
    this.trustedUrl = this.sanitizer.bypassSecurityTrustUrl(this.xssUrl);
    this.trustedVideoUrl =
this.sanitizer.bypassSecurityTrustResourceUrl(this.xssVideoUrl);
  }

}
```

Securitydemo.component.html

```
<div class="container-fluid">
    <h2>XSS URL - Don't Trust</h2>
    <a [href]="xssUrl">Click Here - Don't Trust</a>
    <h2>XSS URL - Trust Me</h2>
    <a [href]="trustedUrl">Click Here - Trust Me</a>
    <h2>YouTube Video</h2>
    <iframe [src]="trustedVideoUrl" width="400"
height="300"></iframe>
```

</div>

Authorization

- Authorization is the process restricting access to the resources in application.
- You can configure components so that they are accessible only to the user when authentication is successful.
- Authentication is the process of verifying user credentials, security token etc.
- You can restrict access to any component by using “Route Guards”.
- It prevents users from navigating to any specific location without proper authentication.
- You can secure the route path.
- A route guard allows to configure custom logic and functionality, where we can verify the user credentials and allow or restrict access.
- For any component if you want a restricted access then you can generate a route guard.
- Angular provides various route guards:

CanActivate	It restricts access to specific route.
CanActivateChild	It restricts access to child route.
CanDeactivate	It is used to restrict the user to exit the route.
Resolve	It is used to access data from any API before route activation.
CanLoad	It is authorizing lazy routes.

- All route guards and return different types of values. Usually a boolean value is used to confirm user authentication.
- Route Guards are generated for every component by using
  > ng generate guard <guard-name>

Ex:

- Go to your project enabled with routing
- Add a new component
  > ng g c manager-home
- Generate a guard for component
  > ng generate guard manager-guard
  Manager-guard.guard.ts

```
import { Injectable } from '@angular/core';
import { Router, CanActivate, ActivatedRouteSnapshot,
RouterStateSnapshot, UrlTree } from '@angular/router';
import { Observable } from 'rxjs';
import { DataService } from './data.service';

@Injectable({
  providedIn: 'root'
})
export class ManagerGuardGuard implements CanActivate {
  constructor(private data: DataService, private router: Router){}
  public users = [];
  public username = 'David';
  public password = 'mng12';

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): boolean {
     // your authentication logic here..
      this.users = this.data.GetUsers();
      for(var user of this.users) {
        if(user.Name==this.username && user.pwd==this.password)
{
          return true;
        } else {
          this.router.navigate(['login']);
        }
      }
      return false;
  }
}
(or)
this.users = this.data.GetUsers();
      for(var user of this.users) {
        if(user.Name!==this.username &&
user.pwd!==this.password) {
          this.router.navigate(['login']);
          return false;
        }
      }
      return true;
```

- Maintain data in “data.service.ts”

```
public GetUsers(){
    return [
      {Name: 'John', Role: 'Admin', pwd: 'admin12'},
    {Name: 'David', Role: 'Manager', pwd: 'mng12'}
   ];
  }
```

- Go to app-routing.module.ts
  {path: 'manager', component: ManagerHomeComponent, canActivate: [ManagerGuardGuard]},
- Try changing the user name and password if it is mismatching it will restrict.

Unit Testing
Building and Deploying

- Angular supports frameworks like MVC and MVVM.
- MVC and MVVM enable Unit Testing.
- They support Test Driven Development [TDD].
- Unit Testing include testing every function that you write for component, pipe, service etc.
- Testing verifies whether the expected values and return value are same and will report bugs.
- Angular is integrated with Jasmine & Karma, which are used for unit testing. [Protractor, xUnit, etc]

Setup Framework for Testing [Manually]

- Create a new folder in your workspace by name “TestingDemo”
- Install Jasmine Framework for your workspace
  > npm install jasmine-core
- Create a new HTML document and add into project and link the following files
  Home.html

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="../node_modules/jasmine-
core/lib/jasmine-core/jasmine.css">
    <script src="../node_modules/jasmine-core/lib/jasmine-
core/jasmine.js"></script>
    <script src="../node_modules/jasmine-core/lib/jasmine-
core/jasmine-html.js"></script>
    <script src="../node_modules/jasmine-core/lib/jasmine-
core/boot.js"></script>
  </head>
</html>
```

- Testing every component, service, pipe or module function include 3 stages
  - Arrange
  - Act
  - Assert
- Arrange: It is describing the test. [describe()]
- Act: It specifies the function to test. [it()]
- Assert: Verifying the expected and return value. Report the test result pass or fail. [expect()]

Syntax:

```
describe("Test Description", function(){

    it("Test Function and It Purpose", function(){

        expect(returnedValue).toBe(ExpectedValue);

    })
```

})

Ex:

- Add a new folder “Components”
- Add following files

```
Home.component.js
function addition(a, b)
{
   return a + b;
}
function hello(str)
{
   return str;
}
Home.component.spec.js
describe("Math Component Test", function(){
   it("Addition Test", function(){
      expect(addition(20,20)).toBe(40);
   })
   it("Hello Test", function(){
      expect(hello("john")).toBe("johns");
   })
})
```

```
Home.html

<!DOCTYPE html>

<html>

   <head>
```

```
        <link rel="stylesheet" href="../node_modules/jasmine-core/lib/jasmine-core/jasmine.css">
        <script src="../node_modules/jasmine-core/lib/jasmine-core/jasmine.js"></script>
        <script src="../node_modules/jasmine-core/lib/jasmine-core/jasmine-html.js"></script>
        <script src="../node_modules/jasmine-core/lib/jasmine-core/boot.js"></script>
        <script src="Components/math.component.js"></script>
        <script src="Components/math.component.spec.js"></script>
    </head>
</html>
```

- Go to “karma.config” to configure the live server and browser for testing.
- In your angular workspace run the command

```
> ng test --project=yourProjectName
```

Building and Deploying

- Building is the process of compiling Angular application.
- Checking the syntax errors, Dependencies, Directives, Meta data, Services etc.
- After checking trans compiling Typescript into JavaScript.
- Copying all compile JavaScript file into “output directory”.

- Final build information is copied into "dist" folder.
- "dist" is used for deploying.
- Earlier you were using "ng serve" which is implicitly using a "web pack" for building while serving.
- Angular project command for building application

Command	Description
ng build	It builds complete angular application with defaults. - Default output directory is "dist" - Default startup page is "index.html" - Default base href is "/"
ng build --outputPath=folderName	It build the output int a new directory instead of "dist"
ng build --baseHref=http://locahost:8080	- To change the base href.
ng build --index=login	- To change the startup page

Deploying Angular Application

- You can deploy angular application on Local Servers
  - XAMPP
  - IIS
  - WAMP
  - MAMP etc.
- You can deploy angular application on cloud servers
  - Firebase

  - Azure
  - AWS
  - Now
  - Netify
  - GitHub Page
  - NPM etc.
- Angular can use manual and automated deployment tools
- Always recommended to use automated deployment tools
  - @angular/fire- firebase
  - @azure/ng-deploy
  - @zeit/ng-deploy
  - Angular-cli-ghpages [Git]
  - Ngx-deploy-npm

Deploying on Firebase:

- Create a firebase account by using your Google account
  Firebase.google.com
- Create a new Project: smart-shop
- Go to your project workspace in VS code
- Install firebase tools for your PC
  > npm install -g firebase-tools
- Login into firebase account
  > firebase login
- Build your project for production
  > ng build --project=flipkart –prod
- This will generate “dist” folder with your project repository
- Add angular firebase plugin for your project, which will take care about the deployment process.
  > ng add @angular/fire --project=flipkart

Select your firebase project to deploy: smart-shop

- Deploy entire repository into live application

  > ng deploy --project=flipkart

Note: Deployment finished then go to “hosting” and access the domain