Angular

Introduction

- Angular is a client side framework developed by Google company
- Angular framework is developed using TypeScript
- Angular is mainley used for Single Page Applications Development
- Angular supports multiple devices (Mobiles & Desktops)
- Angular supports multiple browsers
- Angular is free and open source framework
- Angular JS and Angular both are not same
- From Angular 2 version onwards it is called as Angular Framework

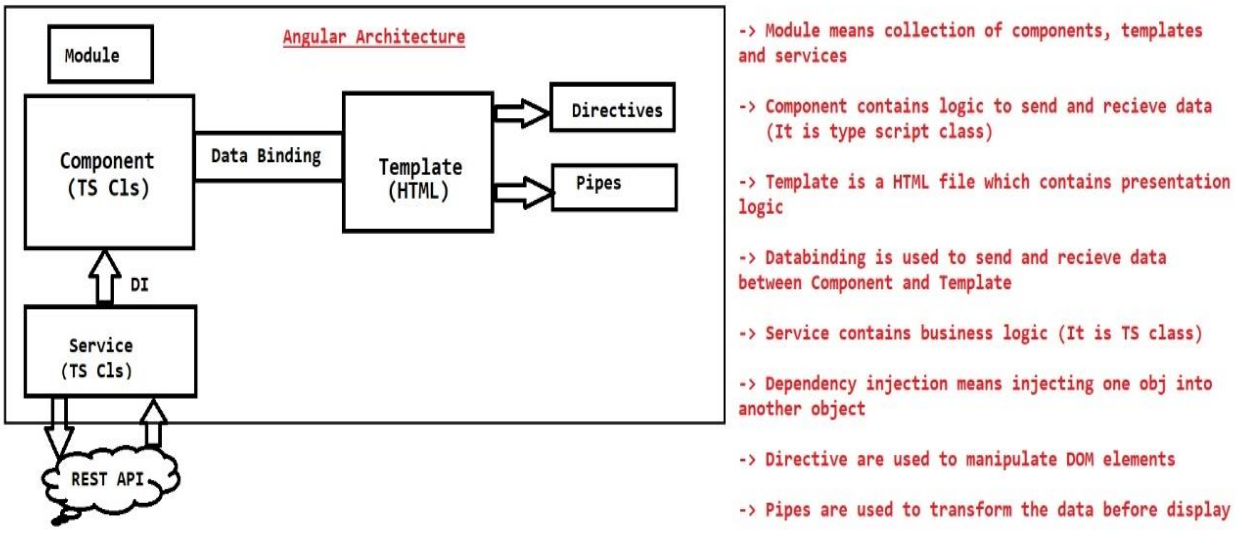
Note: The current version of Angular is 15

Angular Building Blocks

---

1) Components 2) Metadata 3) Template 4) Data Binding 5) Module

6) Service 7) Dependency Injection 8) Directives 9) Pipes



- Angular application is collection of components. In components we will write logic to send data to template and capture data from template. Components are TypeScript classes.
- Metadata nothing but data about the data. It provides information about components and templates.

- Template is a view where we will write our presentation logic. In Angular application template is a HTML file. Every Component contains its own Template.
- Data Binding is the process of binding data between component property and view element in template file.
- Module is a collection of components, directives and pipes
- Service means it contains re-usable business logic. Service classes we will inject into Components using Depdency Injection.
- Dependency Injection is the process of injecting dependent object into target object. In Angular applications services will be injected into components using DI.
- Directives are used to manipulate DOM elements in the Template.

(We can execute presentation logic based on conditions like if-else , loops etc... using directives)

- Pipes are used to transform the data before displaying

(lower case to upper case, INR to USD, dd/mm/yyyy to DD-MMM-YYYY)

Environment Setup For Angular Applications

---

1) Install Node JS

2) Install TypeScript

3) Install Angular CLI

4) Install Visual Studio Code IDE

-> Angular 2+ framework is available as a collection of packages, those packages are available in "Node". To use those packages "npm" (Node Package Manager) is must and should..

URL : https://nodejs.org/en/

-> After installing node software, open cmd and type node -v. It should display node version number then installation is successfull.

-> Angular framework itself is developed based on TypeScript. In Angular applications we will write code using TypeScript only.

-> We can install Typescript using Node Package Manager (npm). Open command prompt and execute below command to install TS.

```
$ npm install -g typescript
```

-> After TypeScript installation got completed we can verify version number using below command in cmd. If it displays version number then installtion is successfull.

```
$ tsc -v
```

-> Install Angular CLI software using below command in TypeScript.

$ npm install @angular/cli -g

-> Check angular installation using below command

$ ng v

-> Download and install VS code IDE

URL : https://code.visualstudio.com/download

Note: We are done with angular setup... lets start building angular applications

==========

-> Create workspace folder in your file system

-> Open command prompt from your workspace folder then type 'code .' then it will open VS CODE IDE from that folder

-> Open terminal in VS CODE IDE to execute our commands ...

-> To create angular application execute below command in command prompt

$ ng new <app-name>

-> Once application got created, navigate into that application folder and execute below command to run angular application.

$ ng serve

-> Angular applications will be deployed into live server which runs on port number 4200. Once application is deployed we can access that using below URL

http://localhost:4200/

-> We can modify default response in app.component.html file

Goto -> Project folder -> src -> app > app.component.html (edit this file)

Angular packages

- **@angular/core** : This pkg provides classes and interfaces that are related to decorators, components and DI. These are mandatory in every Angular 2+ application.
- **@angular/common :** This package provides common directives and pipes which are used in most of the angular applications.
- **@angular/compiler :** This package is used to compile "template" into "java script" code. We never invoke angular compiler directley. We will invoke that indirectly using below 2 packages

  @angular/platform-browser

@angular/platform-browser-dynamic

- **@angular/platform-browser :** This package provides runtime services which are required to run our application in browser.
- **@angular/platform-browser-dynamic :** An angular application can have any no.of modules. This package is used to bootstrap (start) a module, which execution should be started automatically when application started.
- **@angular/forms :** This package is used to create "two way data bindings" , "validations" in angular 2+ applications. This package works based on another package called "zone.js". This package contains below two modules

  a) Forms Module

  b) Reactive Forms Module

- **@angular/router :** This package is used to create routings (page navigations) in Angular 2+ applications.
- **@angular/http :** This package is used to send Ajax request to server and get Ajax response from server.
- **@angular/animations** : This package is used to create animations in angular application.
- **@angular/material :** This package is used for "angular material design" in angular applications.
- **@angular/cdk :** Based on this package only "angular material design" components are developed. So this package must be used while using "@angular/material" package.
- **@angular/cli :** This package provides set of commands to create new angular application, components, pipes, directives, services etc.

- In angular application we can have any no.of modules
- When we run angular application, it starts execution from startup module i.e app-module
- Angular application boot strapping will begin from app module
- AppModule will bootstrap AppComponent
- AppComponent is the default component in Angular application

We can see below files in app module

---

src/app/app.module.ts

src/app/app.component.ts

src/app/app.component.html

src/app/app.component.css

src/app/app.component.spec.ts

src/app/app.module.ts

-> This file contains definition of app module

-> Angular app can have any no.of modules. It should at least one module i.e called as "AppModule"

-> This file imports "AppComponent" from app.component.ts file and bootstraps the same in "AppModule"

```
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 { }
```

src/app/app.component.ts

-> This file contains defintion of "AppComponent"

-> In Angular application we can have many components. It should contain atleast one component that is called as "AppComponent"

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

```
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app1';
}
```

- selector is used to invoke the component
- templateUrl represents presentation logic file (html)
- styleUrls represents css file(s)

----------src/app/app.component.html----------

- This file contains presentation logic of component
- Every component will have its own template
- This template file content will be rendered into

<app-root></app-root> selector tag at index.html

----------src/app/app.component.css----------

- This file contains css syles of "AppComponent"
- By default this file is empty

----------src/app/app.component.spec.ts----------

- This file contains test cases for "AppComponent"
- The test case files should have "spec.ts" file extension

==========

- ⇨ In Angular application "index.html" file will act as wecome file
- ⇨ When we access Angular application URL in Browser it will load index.html file
- ⇨ In index.html file we are using AppComponent selector to invoke AppComponent.

```
<app-root></app-root>
```

- ⇨ We can create Component in the Angular using below command

  $ ng generate component <component-name>

- When we create new component it will generate 4 files like below

CREATE src/app/greet/greet.component.html (20 bytes)

CREATE src/app/greet/greet.component.spec.ts (619 bytes)

CREATE src/app/greet/greet.component.ts (271 bytes)

CREATE src/app/greet/greet.component.css (0 bytes)

UPDATE src/app/app.module.ts (478 bytes)

Components

- The component class represents certain section of web page. For example, "login form" is represented as login component.
- The component class includes "properties" to store the data, "methods" to manipulate the data.
- Every Angular 2+ application contains at-least one component which is called as "app component". We can create any no.of components in angular application.
- A component is invoked through a custom tag called as selector. "app component" selector is "<app-root></app-root>".
- The Component class should have a decorator called "@Component" to define that class as Component class.

Component Class

```
import {Component} from "@angular/core"

@Component (meta-data)

class ClassName{

   property:dataType = value;

   method(args) : returnType {

      //logic

   }

}
```

Metadata properties of @Component

```
@Component({
```

```
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
```

- selector : represents tag which is used to invoke the component
- templateUrl : represents the html file that has to be rendered when the component is invoked
- template represents content of content
- styleUrls : Represents the list of styles (css) that have to be loaded for the component.
- providers : Represents list of services to be imported into the component
- animations : Represents list of animations to be performed in the component.

Note: The components which we are creating will act as child components for AppComponent.

- Child Components we will access from AppComponent using Selector.

- Create Welcome Component & Greet Component
- Invoke Welcome Component & Greet Component in AppComponent like below

----------app.component.html----------

```
<h1> This message from AppComponent</h1>
<app-welcome></app-welcome>
<app-greet></app-greet>
```

Note: Every Component having its own CSS file to apply styles for that component related template logic.

Data Bindings

- The "data binding" is used to establish relation between "component" and "template".

- When the value of "component" is changed, then template will be changed automatically. When the value of "template" is changed, then the "component" will be changed automatically.
- Data Bindings are four Types

1) Interpolation

2) Property Binding

3) Event Binding

4) Two Way Binding

Interpolation

- It is used to display the value of property in template
- If the property value is changed then automatically it will be updated in template

syntax : {{propertyName}}

Property Binding

- Property Binding is used to send the data from component to template and assign the same into an attribute of tag.
- If the property value is changed then automatically it will be updated in template

Syntax: [attribute]=*property

Event Binding

- It is used to pass event notifications from template to component

Syntax : <tag (event) = "method()" > </tag>

Two-way Data Binding

- Two-way data binding is the combination of both property binding and event binding
- When we change the value of the property then automatically it will be updated in HTML element.
- When we change the value of HTML element then automatically it will updated in property.
- "ngModel" is the pre-defined directive which is used to achieve two-way data binding.
- Two data binding is applicable only for <input/> and <select/> tags.
- FormsModule must be imported in order to use Two Way Data Binding

---

Developing Angular application with Two Way Data Binding

---

1) Import FormsModule in app.module.ts file from '@angular/forms' package

2) Declare Variables and function in AppComponent class

3) Write Design Logic in app.component.html file

----------app.module.ts----------

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

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

----------app.component.ts----------

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
    fname:string = "Adam";
    lname:string = "Smith";
```

```
  age:number=25;
  gender:string="Male";
  country:string="India";
  isEmployed:boolean=true;

  handleSubmitBtn(){
    this.fname = "John";
    this.lname="Buttler";
    this.age = 30;
    this.gender= "Male";
    this.country="USA";
    this.isEmployed=false;
  }
}
----------app.component.html----------
<div>
  <h4>Data Bindings Example</h4>
  First Name : {{fname}} <br/>
  Last Name : {{lname}} <br/>
  Age : {{age}} <br/>
  Gender : {{gender}} <br/>
  Country : {{country}} <br/>
  Is Employed : {{isEmployed}} <br/>
  <hr/>
  First Name : <input type="text" [(ngModel)]="fname"/><br/>
  Last Name : <input type="text" [(ngModel)]="lname"/><br/>
  Age : <input type="text" [(ngModel)]="age"/><br/>
  Gender :
    <input type="radio" [(ngModel)]="gender" value="Male">Male
```

```
      <input type="radio" [(ngModel)]="gender" value="Fe-Male">Fe-Male <br/>

   Country :
      <select [(ngModel)]="country">
         <option>India</option>
         <option>USA</option>
         <option>UK</option>
      </select> <br/>

      Is Employed : <input type="checkbox" [(ngModel)]="isEmployed"> <br/>
   <input type="button" value="Submit" (click)="handleSubmitBtn()" />

</div>
```

Angular application with Registration Form

-> Create Angular application

    $ ng new registration

-> Import FormsModule from '@angular/forms' package in app.module.ts file

-> Write variables and function in component class to deal with registration form

-> Design presentation logic in Template file (app.component.html)

-> Run angular application

          $ ng serve

----------app.module.ts----------

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

```
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule, FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
----------------------app.component.ts-------------------------
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
    username:string="";
    password:string="";
    confirmPassword:string="";
    gender:string="";
    country:string="";
    licenseAgreement:boolean=false;

    msg:string="";

    RegisterClick(){
```

```
      this.msg = "Username : "+this.username+"<br/>Password:"+this.password
      +"<br/>ConfirmPassword:"+this.confirmPassword
      +"<br/>Gender : "+this.gender+"<br/>Country : "+this.country
      +"<br/>License Agreement : "+this.licenseAgreement;
    }
}
```

----------app.component.html----------

```
<h4>Registration</h4>

Username : <input type="text" [(ngModel)]="username"/><br/>
Password : <input type="password" [(ngModel)]="password"/><br/>
Confirm Password : <input type="password" [(ngModel)]="confirmPassword"><br/>
Gender :
      <input type="radio" [(ngModel)]="gender" value="Male">Male
      <input type="radio" [(ngModel)]="gender" value="Fe-Male">Fe-Male <br/>

Country:
     <select [(ngModel)]="country">
       <option>-Select-</option>
       <option>India</option>
       <option>USA</option>
       <option>UK</option>
     </select> <br/>

<input type="checkbox" [(ngModel)]="licenseAgreement"/> I Accept License Agreement
<br/>
<input type="submit" value="Register" (click)="RegisterClick()"/>
<hr/>
<div [innerHTML]="msg"></div>
```

Directives

- Directives are used to perform DOM manipulations

What is DOM?

- DOM stands for Document Object Model
- Webpage content will be represented as a tree i.e DOM
- The HTML DOM is an API (Programming Interface) for JavaScript:
- JavaScript can add/change/remove HTML elements
- JavaScript can add/change/remove HTML attributes
- JavaScript can add/change/remove CSS styles
- JavaScript can react to HTML events
- JavaScript can add/change/remove HTML events

Built-In Directives

style

- It is used to set CSS property value dynamically at runtime.

- When Component property value changed then CSS property value will be changed automatically.

Syntax

<tagname [style.cssproperty]="component-property">

----------app.component.ts----------

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

```
    marks:number=20;
    mycolor:string="";

    constructor(){
       if(this.marks >=35 ){
         this.mycolor="green";
       }else{
         this.mycolor="red";
       }
    }
}
```

----------app.component.html----------

```
<div>
  <h3>Style Directive Example</h3>

 <div [style.color]="mycolor">{{marks}}</div>
</div>
```

---

ngClass

-> IT is used to CSS classname dynamically at run time

-> When the value of Component property is changed then css class will be changed automatically.

-> Use this directive to set styles with multiple properties conditionally.

----------app.component.css----------

```
.class1{
   color:green;
   font-size:30px;
```

```
}

.class2{
   color:red;
   font-size:50px;
}
----------app.component.html----------
<div>
  <h3>ngClass Directive Example</h3>

 <div [ngClass]="myclass">{{marks}}</div>
</div>
----------app.component.ts----------
import { Component } from '@angular/core';

@Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrls: ['./app.component.css']
})
export class AppComponent {
    marks:number=80;
    myclass:string="";

    constructor(){
       if(this.marks >=35 ){
         this.myclass="class1";
       }else{
         this.myclass="class2";
```

```
    }
  }
}
```

ngIf

➔ The ngIf displays the element if condition is true otherwise it will remove element from DOM.

Syntax:

```
-------
<tag *ngIf="condition">
</tag>
```

Note: The ngIf directive must prefix with *

```
-------------------------app.component.ts---------------------
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
    marks:number=80;
    b:boolean;

    constructor(){
      if(this.marks >=35 ){
        this.b=true;
      }else{
        this.b=false;
      }
    }
```

```
}
----------app.component.html----------
<div>
  <h3>ngIf Directive Example</h3>

<div *ngIf="b" style="background-color:blue;">Congratulations...!!</div>

<div *ngIf="!b" style="background-color: red;">Better luck next time..!!</div>
</div>
```

---

ngIf and else

-> The "ngIf and else" displays one element if it is "true" otherwise it displays other element.

syntax:

```
-------
<tag *ngIf="condition; then template1;else template2">
</tag>
<ng-template #template1>
...
</ng-template>
<ng-template #template2>
...
</ng-template>
----------app.component.ts----------
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
```

```
export class AppComponent {
    marks:number=80;
    b:boolean;

    constructor(){
       if(this.marks >=35 ){
         this.b=true;
       }else{
         this.b=false;
       }
    }
}
```

----------app.component.html----------

```
<div>
  <h3>ngIf else Directive Example</h3>

<div *ngIf="b;then template1;else template2">

</div>

<ng-template #template1>
  <div style="background-color:green;">Congratulations...."</div>
</ng-template>

<ng-template #template2>
    <div style="background-color:red">Better luck next time...</div>
</ng-template>
```

---

ngSwitch

- ➔ The "ngSwitch" checks the value of a variable, weather it matches with any one of the cases and displays element when it matches with anyone.
- ➔ Use "ngSwitch" if you want to display some content for every possible value in a variable.

syntax

```
<tag [ngSwitch]="property">

        <tag *ngSwitchCase="'value'"></tag>
        <tag *ngSwitchCase="'value'"></tag>
        <tag *ngSwitchCase="'value'"></tag>
        <tag *ngSwitchDefault></tag>
</tag>
```

----------app.component.ts----------

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
   country:string=null;
}
```

----------app.component.html----------

```
<div>
  <h4>ngSwitch Example</h4>
  <select [(ngModel)]="country">
    <option>India</option>
```

```
    <option>USA</option>
    <option>UK</option>
    <option>Japan</option>
  </select>
  <div [ngSwitch]="country">
    <p *ngSwitchCase="'India'">India Details Here</p>
    <p *ngSwitchCase="'USA'">USA Details Here</p>
    <p *ngSwitchCase="'UK'">UK Details Here</p>
    <p *ngSwitchCase="'Japan'">Japan Details Here</p>
    <p *ngSwitchDefault>Please select country</p>
  </div>

</div>
```

ngFor

- ➔ It is used to repeat the tag once for each element in the array. It generates (repeats) the given content once for one element of the array.
- ➔ We have to use use prefix '*' before "ngFor"

Usecase: Displaying all products available in shopping cart.

Syntax:

```
-------
<tag *ngFor="let variable of arrayname">
</tag>
```

----------app.component.ts----------

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

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

```
export class AppComponent {
    cities:string[] = ["New Delhi", "Mumbai", "Banglore", "Hyderabad"];
}
----------app.component.html----------
<div>
  <h4>ngFor Example</h4>
  <ul>
    <li *ngFor="let city of cities">{{city}}</li>
  </ul>
</div>
```

ngFor with Object Array

- ➔ Using this technique we can print array of object values in web page.
- ➔ First we have to store set of objects inside array then read objects one-by-one using "ngFor" and display the data in table format.

Usecase: Reading Product details (name & price) and displaying them.

syntax to create object array

```
------------------------------
arrayRefVariable:classname[] = [
   new ClassName(),
   new ClassName(),
   new ClassName()
];
```

syntax to use object array using ngFor

```
---------------------------------------
<tag *ngFor="let variable of arrayRefVariable">
      variable.property1
      variable.property2
</tag>
```

ngFor with Object array

----------employee.ts----------

```
export class Employee{
   empId:number;
   empname:string;
   salary:number;

   constructor(a, b, c){
      this.empId = a;
      this.empname = b;
      this.salary = c;
   }
}
```

----------app.component.ts----------

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

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

      employees:Employee[] = [
         new Employee(101, "John", 5000),
         new Employee(102, "Smith", 5000),
         new Employee(103, "Nick", 6000)
```

```
    ];
}
```

----------app.component.html----------

```
<div>
  <h4>ngFor with Object Array Example</h4>
    <table border="1">
        <tr>
          <th>Emp Id</th>
          <th>Emp Name</th>
          <th>Emp Salary</th>
        </tr>
        <tr *ngFor="let emp of employees">
            <td>{{emp.empId}}</td>
            <td>{{emp.empname}}</td>
            <td>{{emp.salary}}</td>
        </tr>
    </table>
  </div>
```

ngFor directive with Add and Remove functionality

➔ We can allow the user to add new records (objects) to existing array. User can also delete existing records.

Adding element to array

```
arrayVariable.push(value);
```

Removing element from array

```
arrayVariable.splice(index, count);
```

----------app.module.ts----------

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

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

import { AppComponent } from './app.component';
import { FormsModule } from '@angular/forms';

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

----------------------employee.ts----------------------------
export class Employee{

    empId:number;
    empname:string;
    salary:number;

    constructor(a, b, c){
        this.empId = a;
        this.empname = b;
        this.salary = c;
    }
```

```
}
----------------------app.component.ts--------------------------

import { Component } from '@angular/core';
import { Employee } from './employee';

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

      employees:Employee[] = [
         new Employee(101, "John", 5000),
         new Employee(102, "Smith", 5000),
         new Employee(103, "Nick", 6000),
         new Employee(104, "Orlen", 8000)
      ];

      newemployee:Employee = new Employee(null,null,null);

      onInsertClick(){
        this.employees.push(new
Employee(this.newemployee.empId,this.newemployee.empname,this.newemployee.salary)
);
         this.newemployee.empId = null;
         this.newemployee.empname = null;
         this.newemployee.salary = null;
      }
```

```
    onDeleteClick(n){
      if(confirm("Are you sure to delete this emp?")){
        this.employees.splice(n,1);
      }
    }
}
----------app.component.html----------
<div>
  <h4>ngFor with Object Array Example</h4>
    <table border="1">
      <tr>
        <th>Emp Id</th>
        <th>Emp Name</th>
        <th>Emp Salary</th>
        <th>Action</th>
      </tr>
      <tr *ngFor="let emp of employees; let i = index">
          <td>{{emp.empId}}</td>
          <td>{{emp.empname}}</td>
          <td>{{emp.salary}}</td>
          <td><input type="button" value="Delete" (click)="onDeleteClick(i)"></td>
      </tr>
      <tr>
        <td><input type="text" [(ngModel)]="newemployee.empId" placeholder="Emp
Id"></td>
        <td><input type="text" [(ngModel)]="newemployee.empname" placeholder="Emp
Name"></td>
        <td><input type="text" [(ngModel)]="newemployee.salary" placeholder="Emp
Salary"></td>
```

```
        <td><input type="button" value="Insert" (click)="onInsertClick()"></td>
      </tr>
    </table>
  </div>
```

Searching & Sorting

- ➔ Array is used to store the data
- ➔ We can search for some content in the array
- ➔ We can sort the data based on specific property
- ➔ We will use "filter" function to search content. The filter function recives a callback function, which gets executed once for each item in the array.
- ➔ We will use "sort" function to sort the data. The sort function recieves a callback function, which gets called for each pair of items in the list.

Syntax for searching

```
arrayName.filter(item) => {return true or false}
```

Syntax for sorting

```
arrayName.sort{
   (item1,item2) => {

        if(item1 < item2)
            return -1;
        else if(item1 > item2)
            return 1;
        else
            return 0;
    }
}
```

----------app.module.ts----------

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

import { AppComponent } from './app.component';
import { FormsModule } from '@angular/forms';

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

----------employee.ts----------

```
export class Employee{

    empId:number;
    empname:string;
    salary:number;

    constructor(a, b, c){
        this.empId = a;
        this.empname = b;
        this.salary = c;
```

```
   }
}
-------------------------app.component.ts----------------------
import { Component } from '@angular/core';
import { Employee } from './employee';

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

      originalemployees:Employee[] = [
         new Employee(101, "John", 5000),
         new Employee(102, "Smith", 12000),
         new Employee(103, "Nick", 6000),
         new Employee(104, "Orlen", 8000),
         new Employee(105, "Charles", 9000),
      ];
      employees:Employee[] = [];
      constructor(){
        this.employees = this.originalemployees;
      }
      str:string = "";
      sortcolumn = "empId";
      order = 1;

      onSearchClick(){
```

```
        this.employees = this.originalemployees.filter((emp) =>  {
          return emp.empname.toLowerCase().indexOf(this.str.toLowerCase()) >=0;});
      }

    onSortClick(){
      this.employees = this.originalemployees.sort((emp1,emp2) => {
          var n = 0;
          if(this.sortcolumn=="empId"){
            return (emp1[this.sortcolumn]-emp2[this.sortcolumn]) * this.order;
          }else if(this.sortcolumn=="empname"){
            return (emp1[this.sortcolumn].charCodeAt(0) -
emp2[this.sortcolumn].charCodeAt(0)) * this.order;
          }else{
            return (emp1[this.sortcolumn] - emp2[this.sortcolumn]) * this.order;
          }
      });
    }
  }
----------------------------app.component.html-------------------
<div>
  <h4>ngFor with Searching and Sorting Data</h4>
  <input type="text" placeholder="Search" [(ngModel)]="str">
  <input type="button" value="Search" (click)="onSearchClick()">
  <br/>
  Sort :
  <select [(ngModel)]="sortcolumn">
    <option>empId</option>
    <option>empname</option>
    <option>salary</option>
  </select>
```

```
  <select [(ngModel)]="order">
     <option value="1">Asending Order</option>
     <option value="-1">Descending Order</option>
  </select>
  <input type="button" value="Sort" (click)="onSortClick()">

  <table border="1">
    <tr>
      <th>Emp Id</th>
      <th>Emp Name</th>
      <th>Emp Salary</th>

    </tr>
    <tr *ngFor="let emp of employees">
        <td>{{emp.empId}}</td>
        <td>{{emp.empname}}</td>
        <td>{{emp.salary}}</td>
    </tr>
 </table>
</div>
```

Services

- ➔ The service is a class which contains re-usable business logic (encryption, de-cryption, validations, calculations etc.)
- ➔ Service class logics we can access in one or more components
- ➔ If we keep re-usable set of properties and methods as part of service class, then we can access them from any cmponent and from any other service available in the application.
- ➔ We must declare service class with "@Injectable()" decorator, to make the service can be accessed from any component.
- ➔ We need to import "@Injectable()" decorator from "@angular/core" package.
- ➔ We must use "@Inject()" decorator, to request angular to create an object for the service class. Then angular framework will automatically creates an object for the

service class and passes the object as an argument for our Component class Constructor.

Note: Realtime applications contains logic to access Backend Rest APIs in Service classes.

Syntax

```
import {Injectable} from "@angular/core";
@Injectable()
class ServiceClassName {
    //methods here
}
```

Add Service as a Provider in the Component

```
@Component({..., providers : [ServieClassName] })
class ComponentClassName {
}
```

Get the Instance of Service using Dependency Injection

```
import {Inject} from "@angular/core";
@Component( { } )
class ComponentClassName{

  constructor(@Inject(ServiceClassName) variable:ServiceClsName ){

  }
}
```

- ➔ Service is a class in Angular application which contains re-usable business logic.
- ➔ To represent one class as Service we will use @Injectable decorator
- ➔ We will inject Service class object into Component class using @Inject decorator
- ➔ Service class object we will take as a parameter for Component class Constructor
- ➔ One Service class object can be used in any no.of Component classes.
- ➔ To create Service we will use below command

  $ ng generate service <Service-Name>

➔ Service class looks like below

```
import { Injectable } from '@angular/core';
@Injectable({
  providedIn: 'root'
})
export class LoginService {

  constructor() { }
}
```

Application Development Using Services

1) Create Angular application

$ ng new login

2) Create 'User' class using below command

$ ng g class User

3) Create Service using below command

$ ng g service Login

4) Configure 'LoginService' as provider in "App Module' and import Forms Module

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { LoginService } from './login.service';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule, FormsModule
  ],
```

```
  providers: [LoginService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

➔ Declare properties in User class (user.ts)

```
 export class User {
   username:string;
   password:string;
   constructor(a:string, b:string){
      this.username = a;
      this.password = b;
   }
}
```

➔ Write the business logic in Service class to validate username & password

```
import { Injectable } from '@angular/core';
import { User } from './user';
@Injectable({
  providedIn: 'root'
})
export class LoginService {
  users:User[] = [
    new User("john", "john@123"),
    new User("smith","smith@123"),
    new User("ashok", "ashok@123")
  ];
  constructor() { }
  checkUnameAndPwd(username:string, password:string):boolean{
     var count = 0;
```

```
    for(var i = 0; i<this.users.length;i++){
      if(this.users[i].username==username && this.users[i].password==password){
        count ++;
      }
    }
    if(count ==1){
      return true;
    }else{
      return false;
    }
  }
}
```

➔ Write the logic in Component class to handle login button functionality

```
import { Component, Inject } from '@angular/core';
import { LoginService } from './login.service';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  username:string="";
  password:string="";
  msg:string="";

  constructor(@Inject(LoginService) private loginService:LoginService){
  }
```

```
  CheckLogin(txt1){
    if(this.loginService.checkUnameAndPwd(this.username,this.password)==true){
       this.msg = "Login Successful...";
    }else{
       this.msg = "Invalid Credentials...";
       txt1.focus();
    }
  }
}
```

➔ Develop presentation logic in template file

```
<div>
   <h4>Login Example Using Service</h4>
     Username : <input type="text" [(ngModel)]="username"> <br/>
     Password: <input type="password" [(ngModel)]="password"><br/>
     <input type="submit" value="Login" (click)="CheckLogin()">
     {{msg}}
</div>
```

Pipes

➔ Pipes are used to transform the value into user-expected-format

➔ Pipes are inovked in expresssion (interpolation binding), through pipe (|) symbol.

List of built-in pipes in Angular 2+

---

1. uppercase

2. lowercase

3. slice

4. number

5. currency

6. percent

7. date

8. json etc.

➔ Create Angular application using below command

$ ng new app-name

➔ Import FormsModule in "AppModule"

➔ Configure below Styles in app/styles.css file

```
.class1{
   border: 2px solid red;
   margin: 20px;
}
```

➔ Declare properties in App Component ts file

```
import { Component } from '@angular/core';
@Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrls: ['./app.component.css']
})
export class AppComponent {
  city:string = "Hyderabad";
  salary:number= 752487500;
  n:number=0.72;
  person:object = {firstname:"Adam",lastname:"Smith"};
  dt:Date = new Date();
}
```

➔ Write below presentation logic in app component template file using Pipes

```
<div class="class1">
    <h4>Pipes Example</h4>
    City : {{city}} <br/>
    Salary: {{salary}}<br/>
```

n: {{n}}

Current Time : {{dt}}

Person : {{person}}

<hr/>

City In Uppercase:: {{city | uppercase}}

City In Lowercase : {{city | lowercase}}

Slice : {{city | slice : 2:6}}

Currency in USD: {{salary | currency:"USD"}}

Currency in INR : {{salary | currency:"INR"}}

Short Date : {{dt | date : "shortDate"}}

Medium Date: {{dt | date: "mediumDate"}}

Medium: {{dt | date: "medium"}} >

Formatted Date : {{dt | date:"d/M/y"}}

:

➔ Run the application using below command

$ ng serve

Forms and Validations

➔ Developing forms is very common requirement in every web applications
➔ Forms are used to capture data from end user

Ex: Login form, registration form, search form etc...

➔ In Angular we can develop forms in 2 ways

1) Reactive Forms

2) Template-Driven Forms

Choosing an approach

- Reactive forms and template-driven forms process and manage form data differently. Each approach offers different advantages.
- Reactive forms provide direct, explicit access to the underlying forms object model. Compared to template-driven forms, they are more robust: they're more scalable, reusable, and testable. If forms are a key part of your application, or you're already using reactive patterns for building your application, use reactive forms.

- Template-driven forms rely on directives in the template to create and manipulate the underlying object model. They are useful for adding a simple form to an app, such as an email list signup form. They're easy to add to an app, but they don't scale as well as reactive forms. If you have very basic form requirements and logic that can be managed solely in the template, template-driven forms could be a good fit.

Template Driven Forms

- ➔ Template driven forms are suitable for development of Simple forms with limited no.of fields and simple validations.
- ➔ In these forms, each field is represented as a property in the component class.
- ➔ Validation rules are defined in the template using "html5" attributes. Validation messages are displayed using "validation properties" of angular.
- ➔ "FormsModule" should be imported from "@angular/forms" package.

HTML5 attributes for Validation

---

required="required" : This field is mandatory

minlength="n" : Minimum no.of characters

pattern="regexp" : Regular expression

Requirement : Develop a form using Template-Driven Approach with below fields in the form with form validations.

-> Create Angular application

```
$ ng new app10
```

-> Write CSS in app/styles.css file

```
/* You can add global styles to this file, and also import other style files */
input.ng-invalid.ng-touched{
   border: 2px solid red;
}
input.ng-valid.ng-touched{
   border: 2px solid green;
}
.error{
   color: red;
}
```

-> Import FormsModule in "AppModule"

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { FormsModule } from '@angular/forms';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule, FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

-> Declare properties in "AppComponent" for form binding

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  submit(form:any){
    var firstName = form.firstName;
```

```
    console.log(firstName);
    var lastName = form.lastName;
    console.log(lastName);

    var comment = form.comment;
    console.log(comment);
  }
}
```

-> Write the presentation logic in "App Component"

----------app.component.html----------

```
<h1>Template Driven Forms Validation in Angular 13 Example</h1>

<form #contactForm="ngForm" (ngSubmit)="submit(contactForm.value)">

  <div class="form-group">
    <label for="firstName">First Name</label>
    <input required minlength="3" maxlength="10" ngModel name="firstName" type="text" #firstName="ngModel" class="form-control" id="firstName">
    <div class="alert alert-danger" *ngIf="firstName.touched && !firstName.valid">
      <div *ngIf="firstName.errors && firstName.errors['required']">First Name is required.</div>
      <div *ngIf="firstName.errors && firstName.errors['minlength']">First Name is minimum {{ firstName.errors && firstName.errors['minlength'].requiredLength }} character.</div>
      <div *ngIf="firstName.errors && firstName.errors['maxlength']">First Name is maximum 10 character.</div>
    </div>
  </div>

  <div class="form-group">
    <label for="lastName">Last Name</label>
```

```
    <input required ngModel name="lastName" type="text" #lastName="ngModel"
class="form-control" id="lastName">
    <div class="alert alert-danger" *ngIf="lastName.touched && !lastName.valid">
      Last Name is required.
    </div>
  </div>

  <div class="form-group">
    <label for="comment">Comment</label>
    <input required ngModel #comment="ngModel" name="comment" id="comment"
class="form-control">
    <div class="alert alert-danger" *ngIf="comment.touched && !comment.valid">
      Comment is required.
    </div>
  </div>

  <button class="btn btn-primary" type="submit"
[class.disabled]="!contactForm.valid">Submit</button>

</form>
```

Reactive Forms

- ➔ Reactive Forms are also called as Model Driven Forms
- ➔ Reactive forms are new types of forms in angular which are suitable for creating large forms with many fields and complex validations
- ➔ In these forms, each field is represented as "FormControl" and group of controls is represented as "FormGroup"
- ➔ "ReactiveFormsModule" should be imported from "@angular/forms" package.
- ➔ Validation rules are defined in the component using "Validators" object of angular and validation messages are dsiplayed in the template using "validation properties" of angular.

Validations in Reactive Forms

Validators.required

Validators.minLength

Validators.maxLength

Validators.pattern

Validation Properties

---

untouched

touched

pristine

dirty

valid

invalid

errors

Requirement : Develop a form using Reactive forms with below fields

First Name

Last Name

Email

Amount

Gender

Country

---

1) Create Angular application (new new app-name)

2) Write Styles in src/styles.css for ng-touched and ng-untouched

3) Import "ReactiveFormsModule" in "app module"

4) Write logic in "App Component"

- Form validations in Constructor using Validators
- Method to handle form submission

5) Write Presentation logic template

6) Run the application and test it

/* You can add global styles to this file, and also import other style files */

/* You can add global styles to this file, and also import other style files */

```
input.ng-invalid.ng-touched{
   border: 2px solid red;
}
input.ng-valid.ng-touched{
   border: 2px solid green;
}
.error{
   color: red;
}
```

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { ReactiveFormsModule } from '@angular/forms';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule, ReactiveFormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

```
import { Component } from '@angular/core';
import { FormControl, FormGroup, Validators } from '@angular/forms';
@Component({
  selector: 'app-root',
```

```
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'reactiveformapp';

  form = new FormGroup({
    name: new FormControl('', [Validators.required, Validators.minLength(3)]),
    email: new FormControl('', [Validators.required, Validators.email]),
    body: new FormControl('', Validators.required)
  });

  get f(){
    return this.form.controls;
  }

  submit(){
    console.log(this.form.value);
  }
}
```

```
<h1>Angular 13 Reactive Forms Validation Example</h1>
<form [formGroup]="form" (ngSubmit)="submit()">
  <div class="form-group">
      <label for="name">Name</label>
      <input
        formControlName="name"
        id="name"
        type="text"
```

```
        class="form-control">
      <div *ngIf="f['name'].touched && f['name'].invalid" class="alert alert-danger">
        <div *ngIf="f['name'].errors && f['name'].errors['required']">Name is required.</div>
        <div *ngIf="f['name'].errors && f['name'].errors['minlength']">Name should be 3 character.</div>
      </div>
    </div>

    <div class="form-group">
      <label for="email">Email</label>
      <input
        formControlName="email"
        id="email"
        type="text"
        class="form-control">
      <div *ngIf="f['email'].touched && f['email'].invalid" class="alert alert-danger">
        <div *ngIf="f['email'].errors && f['email'].errors['required']">Email is required.</div>
        <div *ngIf="f['email'].errors && f['email'].errors['email']">Please, enter valid email address.</div>
      </div>
    </div>

    <div class="form-group">
      <label for="body">Body</label>
      <textarea
        formControlName="body"
        id="body"
        type="text"
        class="form-control">
      </textarea>
```

```
      <div *ngIf="f['body'].touched && f['body'].invalid" class="alert alert-danger">
        <div *ngIf="f['body'].errors && f['body'].errors['required']">Body is required.</div>
      </div>
    </div>

    <button class="btn btn-primary" [disabled]="form.invalid"
type="submit">Submit</button>
</form>
```

Steps to install Boostrap In Angular Application

---

1) Execute below commands in terminal

$ npm install bootstrap --save

$ npm install jquery --save

$ npm install popper.js --save

2) Add below styles and scripts in "angular.json" file

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

Routing

- ➔ The routing concept is used to create page navigations in angular2+ applications
- ➔ "Routing" is the process of mapping between the "route(url)" and corresponding component

eX:

http://localhost:8080/home --> HomeComponent

http://localhost:8080/aboutus --> AboutUsComponent

http://localhost:8080/services --> ServicesComponent

➔ The "@angular/router" package provides essential api to create routing.

➔ Angular2+ supports two types of routing

a) Hash-less routing Ex : /home

b) Hash routing Ex : #/home

Steps for working with Routing

➔ Import "@angular/router" package in "package.json" file

```
"dependencies":{
        "@angular/router": "latest"
 }
```

➔ Set the base location of the application on server:

```
<base href="/">
```

➔ Import "Router" from "@angular/router" package

```
import {Router} from "@angular/router";
```

➔ Create Routes

```
var variable1: Routes = [
        {path:"path-here", component:ComponentClsName},
        {path:"path-here", component:ComponentClsName}
]
```

➔ Import "RouterModule" from "@angular/router" package

```
import  {RoutesModule} from "@angular/router"
```

➔ Combine Your Routes & RouterModule

```
var variable2 =
        RouterModule.forRoot(variable1,useHash:true/false});
```

➔ Import both "routes" and "RouterModule" in AppModule

```
@NgModule({..., imports : [..., variable2]})
class AppModule{
}
```

➔ Create Hyperlink to route

```
<a routerLink="/path">Link Here</a>
```

➔ Create Place Holder to display route content

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

➔ Create angular application

```
$ ng new routeapp
```

➔ Create Components in Angular application

```
$ ng g c Home
$ ng g c About
$ ng g c Contact
```

➔ Configure Routes and Combine them with RouterModule in "AppModule"

----------app.module.ts----------

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';
import { ContactComponent } from './contact/contact.component';
import { AboutComponent } from './about/about.component';
import { RouterModule, Routes } from '@angular/router';
import { FormsModule } from '@angular/forms';
```

```
var myRoutes:Routes = [
   {path:"", component:HomeComponent},
   {path:"home", component:HomeComponent},
   {path:"about",component: AboutComponent},
   {path:"contact", component: ContactComponent}
];

var myRoutes2 = RouterModule.forRoot(myRoutes);

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    ContactComponent,
    AboutComponent
  ],
  imports: [
    BrowserModule, FormsModule, myRoutes2
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

➔ Configure Hyperlinks in app.component.html file

```
<div class="class1">
  <h4>Routing</h4>
  <a routerLink="home">Home</a>  
  <a routerLink="about">About</a> 
```

```
 <a routerLink="contact">Contact</a>

  <div id="container">
     <router-outlet>
     </router-outlet>
  </div>
</div>
```

➔ Create Presentation logics in component tempaltes

```
--------------------home.component.html--------------------------------
<div class="class1">
    <h5>I am from HomeComponent</h5>
</div>
----------------------contact.component.html-------------------------
<div class="class1">
    <h5>I am from Contact Component</h5>
</div>
-------------------- about.component.html-------------------------------
<div class="class1">
    <h5>I am from About Component</h5>
</div>
-----------------------------------------------------------------------
```

➔ Create Styles in global styles.css file

```
/* You can add global styles to this file, and also import other style files */
#container{
   background-color: #ccccff;
```

```
    margin: 5px;
    padding: 5px;
    border-radius: 5px;
    height: 200px;
}
```

-> Run the angular application

```
$ ng serve
```

- ➔ We can pass parameters to the route
- ➔ Route parameter is represented as ":paramName"
- ➔ We can get the value of the parameter in the compoment class using "ActivatedRoute" service.
- ➔ ActivatedRoute service should be injected into Component class in which we have to read router parameter value.

Import ActivatedRoute

```
import {ActivatedRoute} from "@angular/router"
```

Inject ActivatedRoute service in Component

```
constructor(@Inject(ActivatedRoute) private route:ActivatedRoute){
}
```

Get The value of parameter

```
this.route.snaphost.params("parametername");
```

Get The value of parameter with updates

```
this.route.params.subscribe(params => {
        params["parameter-name];
});
```

AJAX

- ➔ AJAX stands for Asynchronus Java Script and XML
- ➔ AJAX is not a language but it is a concept which is used to send request from browser to server and also get response from server to browser without refreshing(reloading) the web page in browser.
- ➔ AJAX allows us to interact with the server and get some data from server without refreshing full web page.

  Ex: Country, State and City dropdowns functionality

Advantages of Ajax

1) Executes faster

2) Less burden on browser and server

3) Better user experience

Types of Ajax Requests

GET : Used to retrieve/search data from server

POST : Used to insert data to server

PUT : Used to update data on server

DELETE : Used to delete data from server

- ➔ '@angular/common/http' package provided necessary services to send AJAX request to server and get AJAX response from server
- ➔ If we want to send AJAX request we will import and inject HttpClient. Using this HttpClient we can send AJAX request to server.

Sending GET request to server

```
this.http.get<ModelClsname>(
                        "url",
                         {responseType: "json|text"})
                        .subscribe(this.successCallbackFunction, this.errorCallBackFunction);
```

Sending POST request to server

```
this.http.post(
        "url",
    {data},
        {responseType:"json|text"}).
        subscribe(this.successCallBack, this.errorCallBack);
```

Sending PUT request to server

```
this.http.put(
        "url",
    {data},
        {responseType:"json|text"}).
        subscribe(this.successCallBack, this.errorCallBack);
```

Sending DELETE request to server

```
this.http.delete(
        "url",
        {responseType:"json|text"}).
        subscribe(this.successCallBack, this.errorCallBack);
```

Define Success Callback function

```
successcallback = (response) =>
{

        //do action with response
}
```

Define error callback function

```
errorcallback = (error) =>
{
        //do action with error
}
```

REST api integration with Angular UI

Steps to access REST API in Angular

1) Create Angular application

```
$ ng new app15
```

2) import HttpClientModule in angular application

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';

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

@NgModule({
  declarations: [
    AppComponent
  ],
```

```
  imports: [
    BrowserModule, FormsModule, HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

3) Inject HttpClient into app component and access rest api

```
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Component, Inject } from '@angular/core';
import { Observable } from 'rxjs';

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

  message:string="";

  constructor(@Inject(HttpClient)private http:HttpClient){}

  getData(){
    this.http.get("http://localhost:9090/welcome", {responseType : 'text'})
```

```
    .subscribe(data => {
      this.message = data;
    });
  }

}
```

4) Develop presentation logic in app component template

```
<div>
      <h4>Spring Boot REST API + Angular Integration</h4>

      <input type="button" value="Get Data" (click)="onGetDataClick()"/>
      <div>{{message}}</div>
</div>
```

HTTP Client Examples

API Details : http://localhost:4040/SB-Rest-App/swagger-ui.html

Angular App Devlopment For GET Request

-> Create Angular Application

```
        $ ng new app16
```

-> Create Book class to represent json response in object format

```
        $ ng generate class Book
```

```
export class Book {
   bookId:number;
   bookName:string;
```

```
  bookPrice:number;

   constructor(a:number,b:string,c:number){
     this.bookId = a;
     this.bookName = b;
     this.bookPrice = c;
  }
}
```

Import HttpClientModule & FormsModule in AppModule

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

import { AppComponent } from './app.component';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';

@NgModule({
 declarations: [
  AppComponent
 ],
 imports: [
  BrowserModule, FormsModule, HttpClientModule
 ],
 providers: [],
```

```
  bootstrap: [AppComponent]
})
export class AppModule { }
```

➔ Write REST Call logic in AppComponent

```
import { HttpClient } from '@angular/common/http';
import { Component, Inject } from '@angular/core';
import { Book } from './book';

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

  books:Book[] = [];

  constructor(@Inject(HttpClient)private http:HttpClient){}

 getData(){
    this.http.get<Book[]>("http://localhost:9090/books", {responseType : 'json'})
    .subscribe(data => {
      this.books = data;
    });
  }
}
```

➔ Write presentation logic in template

```
<div>
<h3>Book Details</h3>
   <input type="button" value="Get Data" (click)="getData()"/>

   <table border="1">
      <tr>
         <th>Book Id</th>
         <th>Book Name</th>
         <th>Book ISBN</th>
         <th>Book Price</th>
      </tr>
      <tr *ngFor="let book of books">
         <td>{{book.bookId}}</td>
         <td>{{book.bookName}}</td>
         <td>{{book.isbn}}</td>
         <td>{{book.bookPrice}}</td>
      </tr>
   </table>
</div>
```

HTTP POST Request Example

----------AppComponent.ts----------

```
import { HttpClient } from '@angular/common/http';
import { Component, Inject } from '@angular/core';
import { Book } from './book';
```

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

  msg:string;
  book:Book = new Book(null,null,null,null);

  constructor(@Inject(HttpClient)private http:HttpClient){}

  onInsertClick(){
    this.http.post("api-url", this.book, {responseType:"text"})
            .subscribe(data => {
              this.msg = data;
    );
  }

}
```

----------app.component.html----------

```
<div>
   <h3>Angular UI + Boot REST API</h3>
   <form>
   Book ID : <input type="text" name="bookId" [(ngModel)]="book.bookId"/><br/>
   Book Name : <input type="text" name="bookName"
[(ngModel)]="book.bookName"/><br/>
   Book Price : <input type="text" name="bookPrice" [(ngModel)]="book.bookPrice"/><br/>
   <input type="submit" value="Save Book" (click)="onInsertClick()"/><br/>
```

```
    {{msg}}
    </form>
</div>
```