**Angular Hands-on**

**841569**

**Aravind T**

**Angular-T02-HOL\_001:**

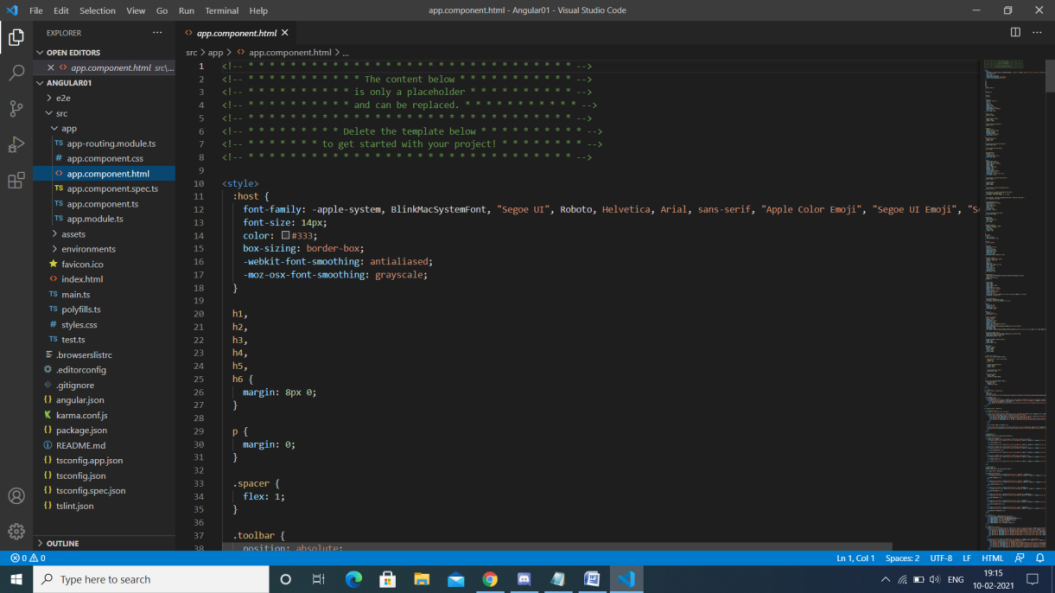
**Task:**

Employ a very basic Angular application using Visual Studio Code editor.

Explanation:

* To work with Angular Project we need to install Angular Package
* Using Angular Cli command (npm install –g @angular/cli)
* Angular new project is creted using (ng new project\_name)
* Visual Studio Code editor is used to do Angular Projects.
* Every time angular creates a new project with a default app component.
* App.module.ts file contains all packages that are imported and used in our project.
* App.component.ts file contains the logical functions
* App.component.html file contains HTML code to view the result in web page.
* Ng serve is the cli command used to run the project.
* It generates a local host with a url.

**Screenshot:**

****

**Angular T03 HOL\_001:**

**Task:**

List various Angular CLI commands.

**Code:**

**App.Component.html**

<app-hello>

</app-hello>

**Hello.component.html**

<H1>This is A T</H1>

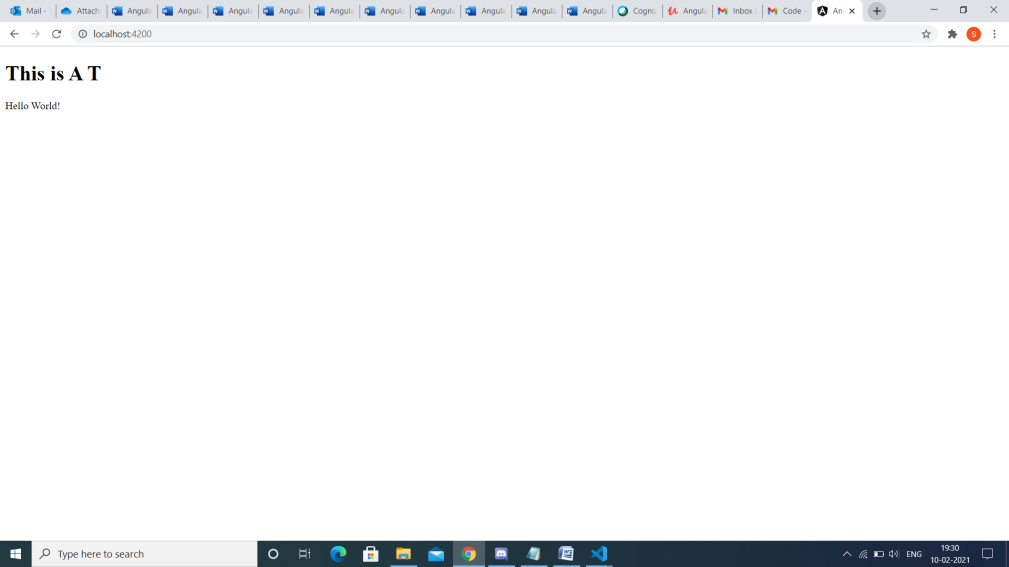
<p>Hello World!</p>

**Explanation:**

**Hello is the new component created using Angular cli command.**

**Ng g c hello**

**Output:**

****

**Angular T04 HOL \_001:**

**Task:**

* Work with various Type Script features like classes and inheritance
* Implement Type Script language features in Angular application

Code:

Department.ts

export interface Department

{

    id:number;

    dname:string;

}

**Employee.ts**

export interface Employee

{

    id:number;

    name:string;

    salary:number;

    permenant:boolean;

}

**Employee-test.ts**

import {Employee} from './Employee';

import {Department} from './Department';

var e1 :Employee=

{

    id:1,

    name:"A T",

    salary:70000,

    permenant:true

}

var d1:Department={

id:1,

dname:'Software'

}

console.log(`ID: ${e1.id}`);

console.log(`Name: ${e1.name}`);

console.log(`Salary:${e1.salary}`);

console.log(`Permenant:${e1.permenant}`);

console.log(`Department ID: ${d1.id}`);

console.log(`Department Name: ${d1.dname}`);

**Employee.js**

"use strict";

exports.\_\_esModule = true;

**Department.ts**

"use strict";

exports.\_\_esModule = true;

**Explanation:**

* Create two type script file and save it with .ts extension.
* Once the type script is executed java script file is automatically created.
* Then run the html code.

<script>

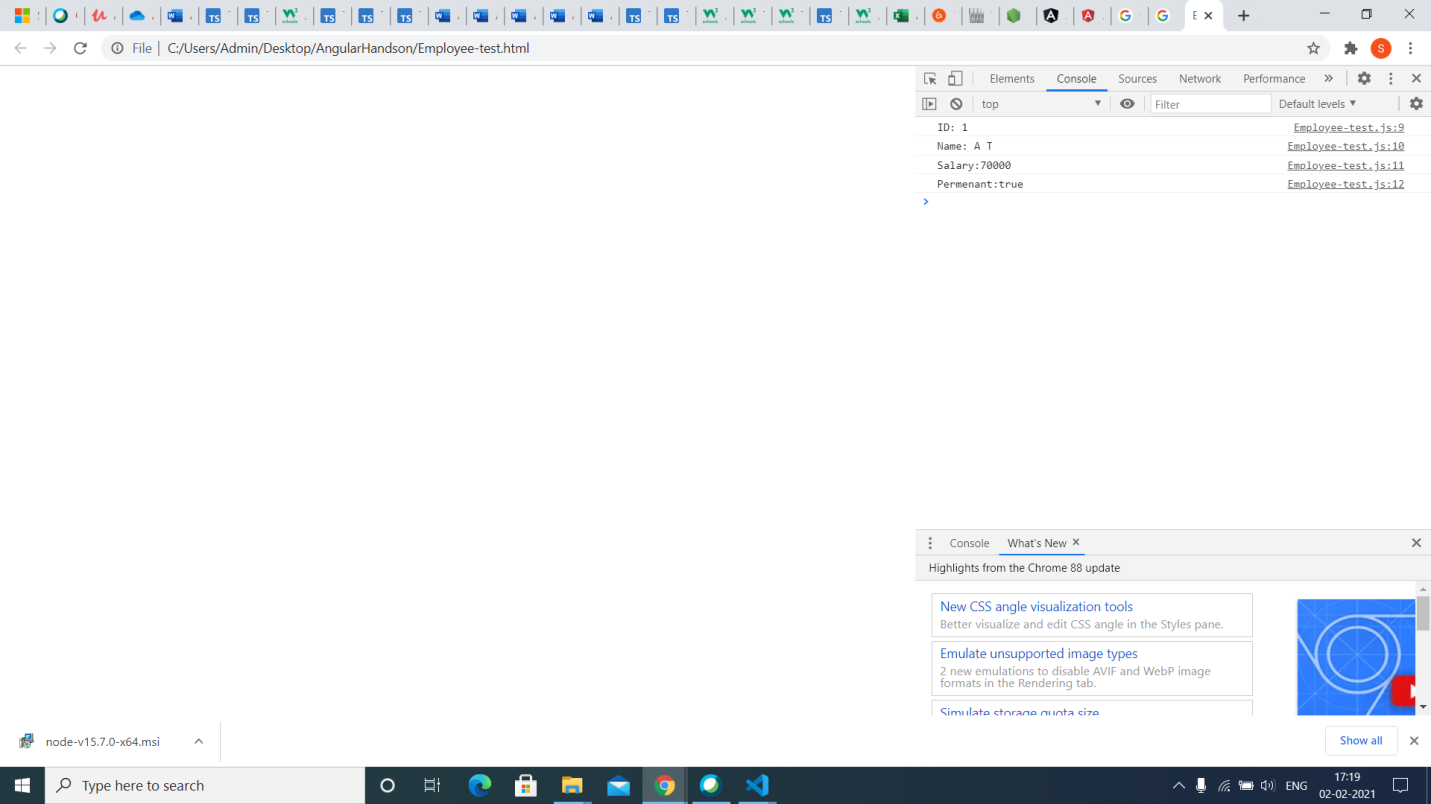
   var exports={};

</script>

<script src="Employee-test.js">

</script>

**Output:**

****

**Angular T05 HOL 001:**

**Task:**

Work with Reactive Forms in Angular.

Code:

Edit-emp-reactive.component.ts

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

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

import { IDepartment } from 'src/app/import/IDepartment';

import {IEmployee} from 'src/app/import/IEmployee'

@Component({

  selector: 'app-edit-emp-reactive',

  templateUrl: './edit-emp-reactive.component.html',

  styleUrls: ['./edit-emp-reactive.component.css']

})

export class EditEmpReactiveComponent implements OnInit {

  constructor() { }

  employeeData:IEmployee = {

    id:0,

    name:'',

    salary:0,

    permenant:true

  };

  deptdata:IDepartment={

    id:0,

    dname:''

  };

  ngOnInit(): void {

  }

  name = new FormControl('');

  employeeForm = new FormGroup({

    id: new FormControl('',Validators.required),

    name: new FormControl('',[Validators.required,Validators.minLength(3)]),

    salary: new FormControl('',Validators.required),

    permanent: new FormControl('',Validators.required)

  });

  departmentForm = new FormGroup({

    id: new FormControl('',Validators.required),

    name: new FormControl('',[Validators.required,Validators.minLength(3)]),

  })

  SubmitEmployeeData(){

  this.employeeData.id=this.employeeForm.get('id')?.value;

  this.employeeData.name=this.employeeForm.get('name')?.value;

  this.employeeData.salary=this.employeeForm.get('salary')?.value;

  this.employeeData.permenant=this.employeeForm.get('permenant')?.value;

  //{return this.employeeForm.get('id');}

  //{return this.employeeForm.get('name');}

 // {return this.employeeForm.get('salary');}

  //{return this.employeeForm.get('permanent');}

}

  depts = ['Payroll','Finance','Sales']

  SubmitDepartmentData(){

    //this.deptdata.id = this.departmentForm.get('id').value;

    //this.deptdata.dname = this.departmentForm.get('dname').value;

    {return this.departmentForm.get('id');}

    {return this.departmentForm.get('dname');}

  }

}

**Edit-emp-reactive.component.html:**

<p>edit-emp-reactive works!</p>

<h3>Reactive Form Component</h3>

<p>edit-emp-reactive works!</p>

Name: <input type="text" [formControl] = 'name'>

The Entered name is {{name.value}}

<hr>

<h3>Fill Up Employee Data</h3>

<form [formGroup]="employeeForm" (ngSubmit)="SubmitEmployeeData()">

  <br>

    <label>

      Id:

      <input type="number" formControlName="id">

    </label><br>

    <br>

    <label>

      Name:

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

    </label><br>

    <br>

    <label>

        Salary:

        <input type="number" formControlName="salary">

      </label><br>

    <br>

      <label>

        Job Type:

        <input type="radio" formControlName="permanent" value="true">Permanent

        <input type="radio" formControlName="permanent" value="false">Temporary

      </label><br>

      <button type="submit">Submit</button>

  </form>

<ul \*ngIf='employeeData.id!=null'>

  <li>{{employeeData.id}}</li>

  <li>{{employeeData.name}}</li>

  <li>{{employeeData.salary}}</li>

  <li>{{employeeData.permenant}}</li>

</ul>

<hr>

<h3>Fill Up Department Data</h3>

<form [formGroup]="departmentForm" (ngSubmit)="SubmitDepartmentData()">

  <br>

    <label>

      Id:

      <input type="number" formControlName="id">

    </label><br>

    <br>

    <label>

      Name:

      <select formControlName="name">

        <option \*ngFor='let dept of depts' ([ngValue])="dept">{{dept}}</option>

      </select>

    </label><br>

    <br>

    <button type="submit">Submit</button>

    </form><br>

    <ul \*ngIf='deptdata!=null'>

      <li>{{deptdata.id}}</li>

      <li>{{deptdata.dname}}</li>

    </ul>

**App.module.ts:**

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

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

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

import { AppRoutingModule } from './app-routing.module';

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

import { EditEmpReactiveComponent } from './edit-emp-reactive/edit-emp-reactive.component';

@NgModule({

  declarations: [

    AppComponent,

    EditEmpReactiveComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule,

    ReactiveFormsModule

  ],

  providers: [],

  bootstrap: [AppComponent]

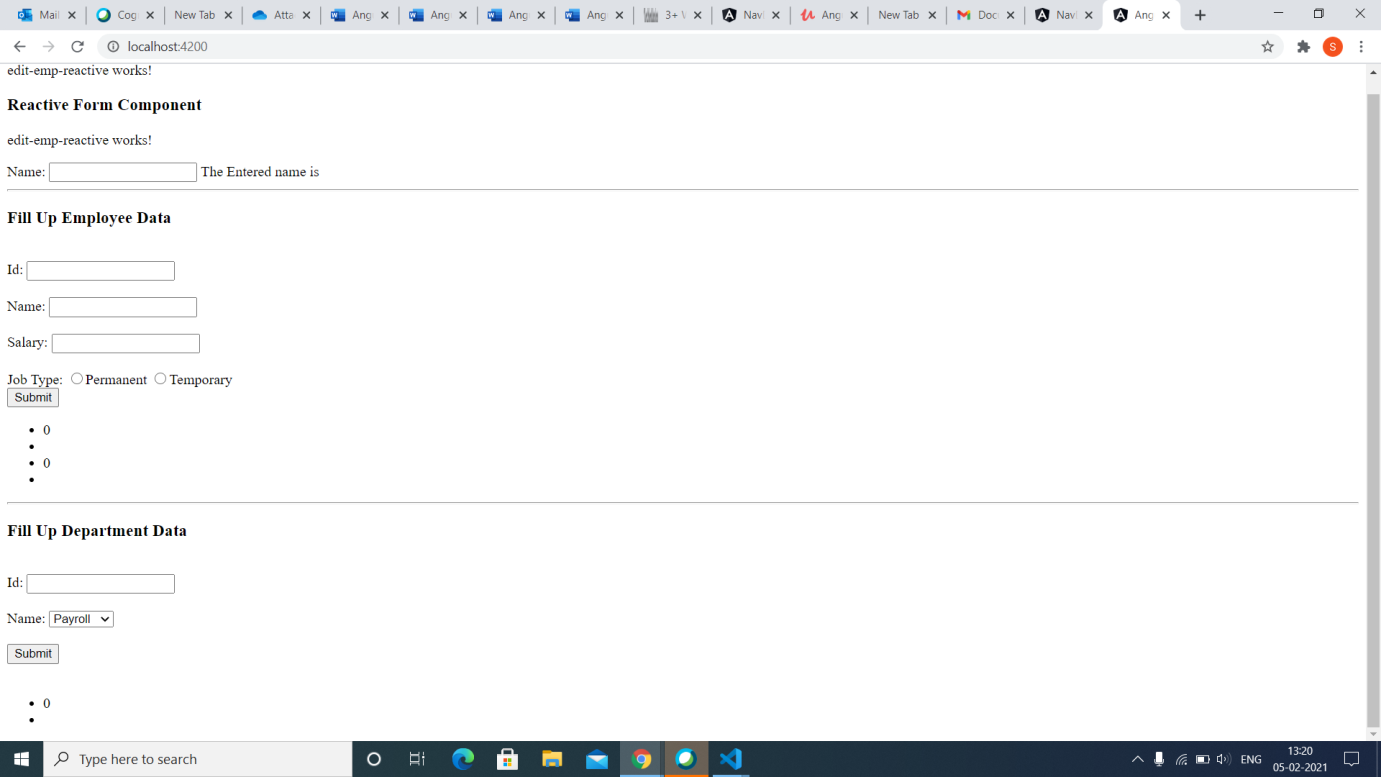
})

export class AppModule { }

**Explanation:**

* To use forms in Angular import FormsModule at app.module.ts
* Create a new component called edit-emp-reactive.
* Using reactive form method the form and its validation is created.

**Output:**

****

**Angular T05 HOL 002**

**Task**

Create a Template-Driven-Forms in Angular.

Code:

App.module.ts

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

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

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

import { AppRoutingModule } from './app-routing.module';

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

import { TemplateDrivenFormComponent } from './template-driven-form/template-driven-form.component';

@NgModule({

  declarations: [

    AppComponent,

    TemplateDrivenFormComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule,

    FormsModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }

**addressModel.ts**

export interface addressModel

{

address: string,

city: string,

state: string,

postcode: number,

country: any[],

aggrement: boolean

}

**Template-driven form.html**

<p>template-driven-form works!</p>

<div class="row">

    <div class="col-md-4 col-md-offset-4" style="margin-top: 50px; border: 1px solid rgb(100, 98, 98); padding: 30px;">

    <form class="form-horizontal" role="form" #f="ngForm" (ngSubmit)="f.form.valid && onFormSubmit()">

    <fieldset>

    <legend>Address Details:

    <strong>Template Driven Form</strong>

    </legend>

    <div class="form-group">

    <label class="col-sm-2 control-label" for="textinput">Address</label>

    <div class="col-sm-10">

    <input type="text" name="address" placeholder="Enter Address" class="form-control">

    </div>

    </div>

    <div class="form-group">

    <label class="col-sm-2 control-label" for="textinput">City</label>

    <div class="col-sm-10">

    <input type="text" name="city" placeholder="Enter City Name" class="form-control">

    </div>

    </div>

    <div class="form-group">

    <label class="col-sm-2 control-label" for="textinput">State</label>

    <div class="col-sm-4">

    <input type="text" name="state" placeholder="State" class="form-control">

    </div>

    <label class="col-sm-2 control-label" for="textinput">Postcode</label>

    <div class="col-sm-4">

    <input type="text" name="postcode" placeholder="Enter Post Code" class="form-control">

    </div>

    </div>

    <div class="form-group">

    <label class="col-sm-2 control-label" for="textinput">Country</label>

    <div class="col-sm-10">

    <select class="form-control" name="country">

    <option>---Select---</option>

    <option \*ngFor="let item of countryData" [value]="item">

    {{item}}

    </option>

    </select>

    </div>

    </div>

    <div class="form-group">

    <div class="col-sm-2 form-check">

    <input type="checkbox" name="aggrement" class="form-check-input">

    </div>

    <label class="col-sm-10 form-check-label">I aggree to Terms & Conditions

    </label>

    </div>

    <div class="form-group">

    <div class="col-sm-12">

    <span style="color: red;">Please Aggree with Terms & Conditions.</span>

    </div>

    </div>

    <div class="form-group">

    <div class="col-sm-offset-2 col-sm-10">

    <div class="pull-right">

    <button type="submit" class="btn btn-primary" style="margin: 4px;">Save</button>

    <button type="reset" class="btn btn-default">Reset</button>

    </div>

    </div>

    </div>

    </fieldset>

    </form>

    </div>

    </div>

**Template driven component.ts**

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

import { addressModel} from 'src/app/data/addressModel';

@Component({

  selector: 'app-template-driven-form',

  templateUrl: './template-driven-form.component.html',

  styleUrls: ['./template-driven-form.component.css']

})

export class TemplateDrivenFormComponent implements OnInit {

  countryData: any[] = ['India', 'US', 'UK'];

model: addressModel = {

address: '',

city: '',

state:'',

postcode: 0,

country:[''] ,

aggrement: false

};

constructor() { }

ngOnInit() {

}

onFormSubmit() {

console.log("Full Address", this.model);

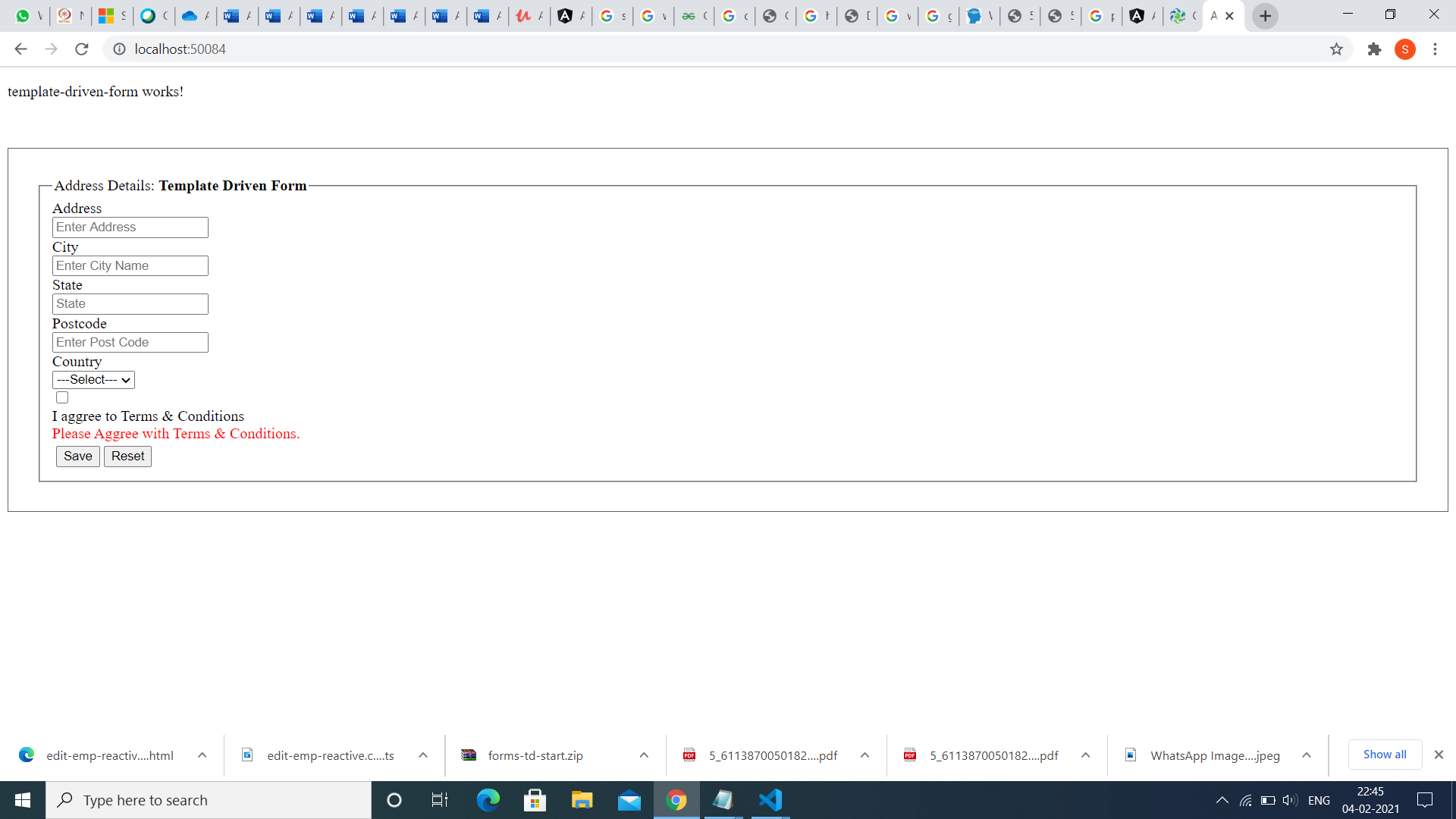
}

}

**Explanation:**

* Create a new component.
* Generate a new addressModel type script file where the filed input name and data types are mentioned.
* Use that type script file by importing inside the template driven component.ts file
* Then using ng serve run the code.

**Output:**

****

**Angular T05 HOL 003**

**Task:**

* Create a Template-Driven-Forms in Angular
* Employee form validation in Angular

Code:

App.module.ts

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

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

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

import { AppRoutingModule } from './app-routing.module';

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

@NgModule({

  declarations: [

    AppComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule,

    FormsModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }

App.component.ts

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

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

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

@Component({

  selector: 'app-root',

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

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  @ViewChild('f')

  signupForm!: NgForm;

  defaultQuestion='India';

  address='';

  title = 'Angulartdform';

  onSubmit()

  {

    console.log(this.signupForm);

  }

}

App.component.html

<h2>Template driven Form With Validation</h2>

<div class="container">

  <div class="row">

    <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">

      <form (ngSubmit)="onSubmit()" #f="ngForm">

        <div id="user-data" ngModelGroup="userData" #userData="ngModelGroup">

          <div class="form-group">

            <label for="username">Username:</label>

            <input type="text" id="username" class="form-control" ngModel name="username" required>

          </div><br>

          <div class="form-group">

            <label for="email">Mail</label>

            <input type="email" id="email" class="form-control" ngModel name="email" required email #email="ngModel">

            <span class="help-block" \*ngIf="!email.valid && email.touched">Enter valid email</span>

          </div>

        </div><br>

        <p \*ngIf="!userData.valid && userData.touched">User Data is not valid</p>

        <div class="form-group">

          <label for="secret">Country</label>

          <select id="secret" class="form-control" [ngModel]="defaultQuestion" name="secret">

            <option value="India">India</option>

            <option value="US">US</option>

            <option value="Pakistan">Pakistan</option>

          </select><br><br>

          <label>Address</label>

          <textarea name="address" rows="2" [(ngModel)]="address">

          </textarea>

          <p>Your Address:{{address}}</p>

        </div><br>

        <div class="form-group">

          <label for="box"></label>

          <input type="checkbox" id="check" name="check" class="form-control" ngModel required #check="ngModel">

          <span class="help-block" \*ngIf="!check.valid && check.touched">Click to submission</span>

          <label>I agree</label>

        </div>

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

      </form>

    </div>

  </div>

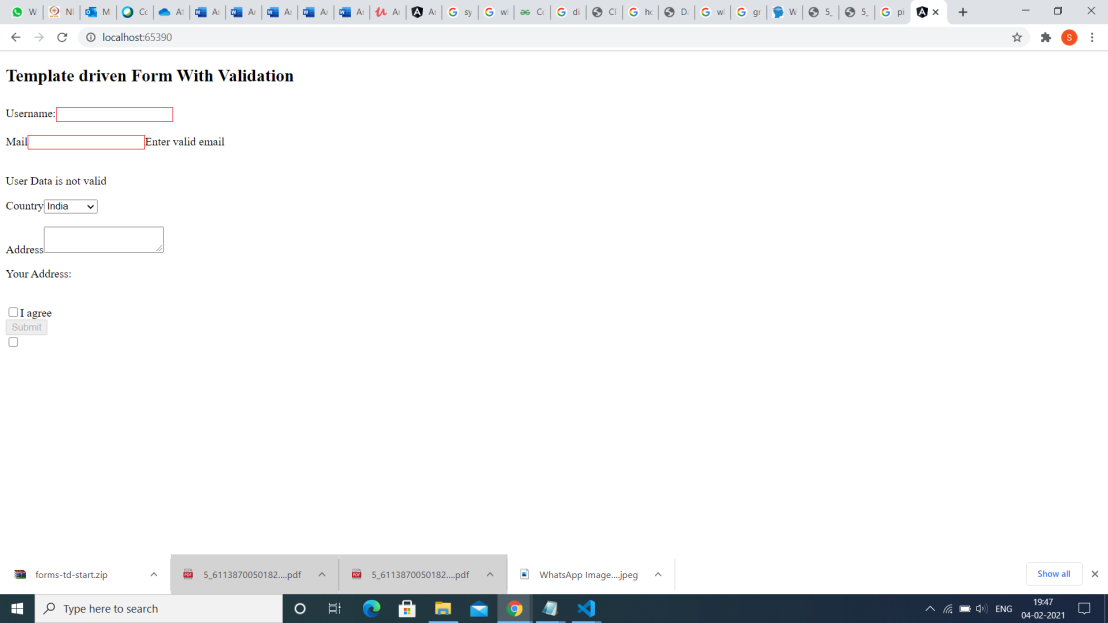
</div>

<div class="col-sm-2 form-check"> <input type="checkbox" >

**Explanation**

* To use forms in Angular import the FormsModeule.
* Template driven is a basic level of developing a form using Angular.
* Form validation is done using template driven method.

**Output**

****

**Angular T06 HOL 001:**

**Task**

* Explain about component interaction in an Angular application
* Explain about various ways of sharing data between components
* Parent to Child: Sharing Data via Input
* Child to Parent: Sharing Data via ViewChild
* Child to Parent: Sharing Data via Output() and EventEmitter

Code:

Implementing Parent to Child: Sharing Data via Input:

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

@Component({

selector: 'app-parent',

template: `

<app-child [childMessage]="parentMessage"></app-child>

`,

styleUrls: ['./parent.component.css']

})

export class ParentComponent{

parentMessage = "message from parent"

constructor() { }

}

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

@Component({

selector: 'app-child',

template: `

Say {{ message }}

`,

styleUrls: ['./child.component.css']

})

export class ChildComponent {

@Input() childMessage: string;

constructor() { }

}

2)Child to Parent: Sharing Data via View Child

import { Component, ViewChild, AfterViewInit } from '@angular/core';

import { ChildComponent } from "../child/child.component";

@Component({

selector: 'app-parent',

template: `

Message: {{ message }}

<app-child></app-child>

`,

styleUrls: ['./parent.component.css']

})

export class ParentComponent implements AfterViewInit {

@ViewChild(ChildComponent) child;

constructor() { }

message:string;

ngAfterViewInit() {

this.message = this.child.message

}

}

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

@Component({

selector: 'app-child',

template: `

`,

styleUrls: ['./child.component.css']

})

export class ChildComponent {

message = 'Hello World!';

constructor() { }

}

3) Child to Parent: Sharing Data via Output() and Event Emitter:

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

@Component({

selector: 'app-parent',

template: `

Message: {{message}}

<app-child (messageEvent)="receiveMessage($event)"></app-child>

`,

styleUrls: ['./parent.component.css']

})

export class ParentComponent {

constructor() { }

message:string;

receiveMessage($event) {

this.message = $event

}

}

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

@Component({

selector: 'app-child',

template: `

<button (click)="sendMessage()">Send Message</button>

`,

styleUrls: ['./child.component.css']

})

export class ChildComponent {

message: string = "Hello World!"

@Output() messageEvent = new EventEmitter<string>();

constructor() { }

sendMessage() {

this.messageEvent.emit(this.message)

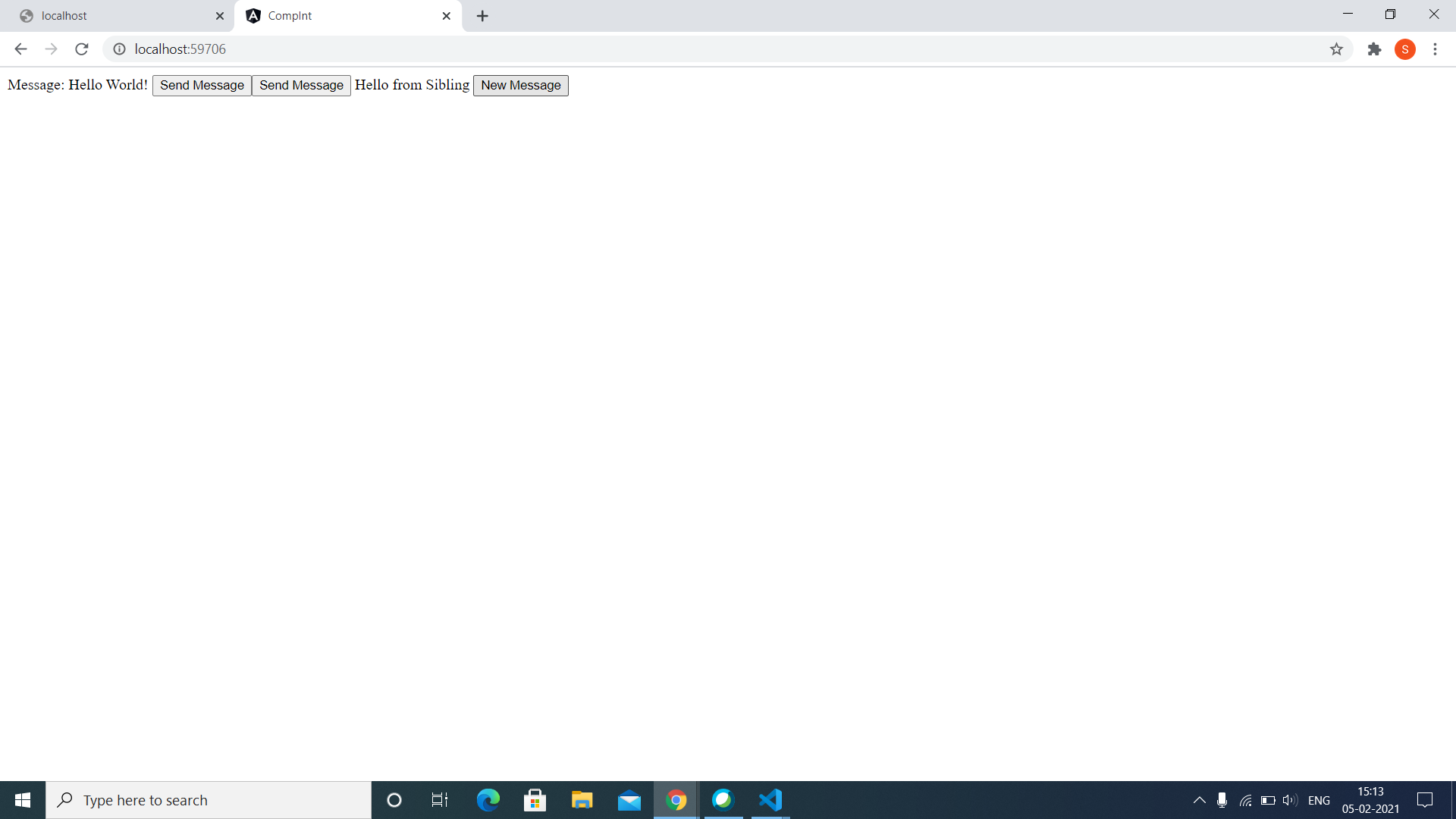
}

}

**Explanation:**

* To implement the following code 3 components parent,child,sibling must be generated using cli command.
* The parent to child data sharing done using Input.
* It is the most common and straightforward method of sharing data.
* It works by using the @Input() decorator to allow data to be passed via the template.
* The child to parent data sharing is done using View Child.
* View Child allows a one component to be injected into another, giving the parent access to its attributes and functions.
* Also, we need to implement the AfterViewInit lifecycle hook to receive the data from the child.
* The Child to Parent data sharing is done via Output() and Even Emitter.
* In the child, we declare a message Event variable with the Output decorator and set it equal to a new event emitter.
* Then we create a function named send Message that calls emit on this event with the message we want to send.

**Output:**



**Angular T06 HOL 002**

**Task:**

To Share Data with a Service

Code:

Data.service.ts

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

import { BehaviorSubject } from 'rxjs';

import { Subscription} from 'rxjs'

@Injectable()

export class DataService {

private messageSource = new BehaviorSubject('default message');

currentMessage = this.messageSource.asObservable();

constructor() { }

changeMessage(message: string) {

this.messageSource.next(message)

}

}

**Parent.component.ts**

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

import { DataService } from "../data.service";

import { Subscription } from 'rxjs';

@Component({

selector: 'app-parent',

template: `

{{message}}

<button (click)="newMessage()">New Message</button>

`,

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

})

export class ParentComponent implements OnInit, OnDestroy {

message:string="Hello World";

subscription: Subscription | undefined;

constructor(private data: DataService) { }

ngOnInit() {

this.subscription = this.data.currentMessage.subscribe(message => this.message = message)

}

ngOnDestroy() {

  this.subscription.unsubscribe();

  }

}

**Sibling.Component.ts:**

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

import { DataService } from "../data.service";

import { Subscription } from 'rxjs';

@Component({

selector: 'app-sibling',

template: `

{{message}}

<button (click)="newMessage()">New Message</button>

`,

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

})

export class SiblingComponent implements OnInit, OnDestroy {

  message: string="Hello World";

  message2:string="Hello World";

  subscription!: Subscription;

constructor(private data: DataService) { }

ngOnInit() {

this.subscription = this.data.currentMessage.subscribe(message => this.message = message)

}

ngOnDestroy() {

this.subscription.unsubscribe();

}

newMessage() {

this.data.changeMessage("Hello from Sibling")

}

}

**Explanation:**

* Create a new service using cli command (ng g service data)
* In the service, you will create a private BehaviorSubject that will hold the current value of the message.
* You define a currentMessage variable which handle this data stream as an observable that will be used by the components.
* Finally, you will create function that calls next on the BehaviorSubject to change its value.

Output:

**Angular T07 HOL 001**

**Task:**

To Implement navigation in an Angular application using Router.

Code:

Menu.component.html

<p>menu works!</p>

<div class="container">

    <nav class="navbar navbar-inverse">

    <div class="container-fluid">

    <div class="navbar-header">

    <a class="navbar-brand"

    href="https://www.justcompile.com/">Just Compile</a>

    </div>

    <ul class="nav navbar-nav">

    <li><a routerLink="home">Home</a></li>

    <li><a routerLink="about">About</a></li>

    <li><a routerLink="login">Login</a></li>

    </ul>

    </div>

    </nav>

    </div>

RouterConfig.ts

import { Routes } from '@angular/router';

import { HomeComponent } from './component/home/home.component';

import { AboutComponent } from './component/about/about.component';

import { LoginComponent } from './component/login/login.component';

export const appRoutes: Routes = [

{ path: 'home', component: HomeComponent },

{ path: 'about',component: AboutComponent},

{ path: 'login',component: LoginComponent}]

App.module.ts

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

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

import { RouterModule } from '@angular/router';

import {  appRoutes }   from 'src/app/routerConfig';

import { AppRoutingModule } from './app-routing.module';

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

import { HomeComponent } from './component/home/home.component';

import { AboutComponent } from './component/about/about.component';

import { LoginComponent } from './component/login/login.component';

import { MenuComponent } from './menu/menu.component';

@NgModule({

  declarations: [

    AppComponent,

    HomeComponent,

    AboutComponent,

    LoginComponent,

    MenuComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule,

    RouterModule.forRoot(appRoutes)

  ],

  providers: [],

  bootstrap: [AppComponent]

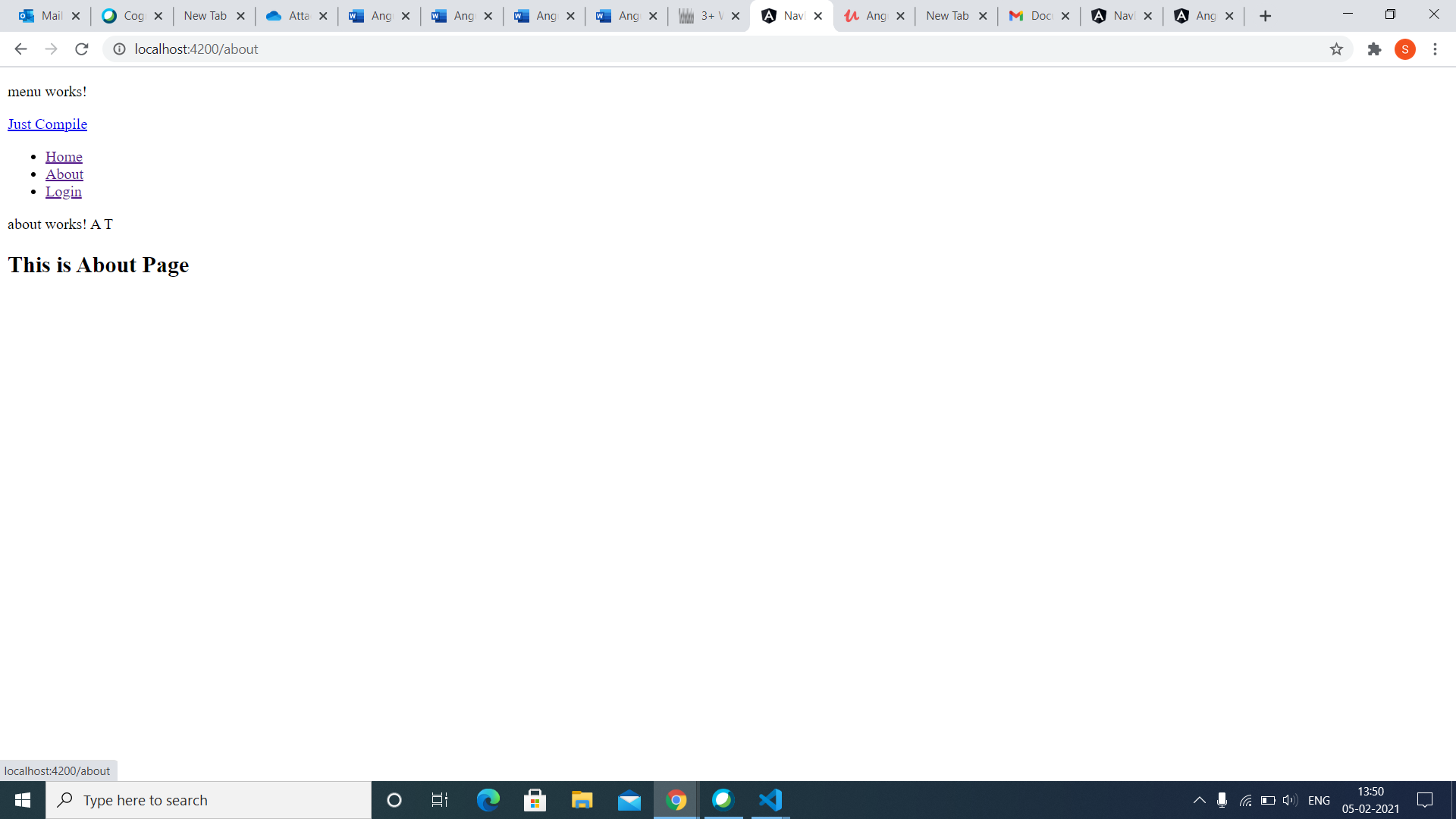
})

export class AppModule { }

Explanation:

* We need to create 3 components for this project.
* Firstly, create a folder inside /src called components.
* Secondly, generate three components inside the components directory as following.
* ng g c component/home
* ng g c component/about
* ng g c component/login
* Create menu component.
* We need to add the Angular Router by importing library inside our app.module.ts file.
* Angular router enables to switch between the nav-bar options.

Output:



**Angular T08 HOL 003:**

**Task:**

Implement asynchronous or callback-based code in an Angular Application using RxJS

Code:

App.component.ts

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

import { Observable } from 'rxjs';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

  title(title: any) {

    throw new Error('Method not implemented.');

  }

data: Observable<number> | undefined;

numArray: number[] = [];

  errors: boolean = false;

  finished!: boolean;

fetchData() {

this.data = new Observable(observer => {

setTimeout(() => { observer.next(10); }, 1000),

setTimeout(() => { observer.next(20); }, 2000),

setTimeout(() => { observer.complete(); }, 3000);

});

const sub = this.data.subscribe((value) => this. numArray.push(value),

error => this.errors = true,

() => this.finished = true);

}

}

App.component.html

<b>Observables Demo</b>

<h1>Async Demo</h1>

<h6 style="margin-bottom: 0">VALUES:</h6>

<div \*ngFor="let value of numArray"> {{ value }}</div>

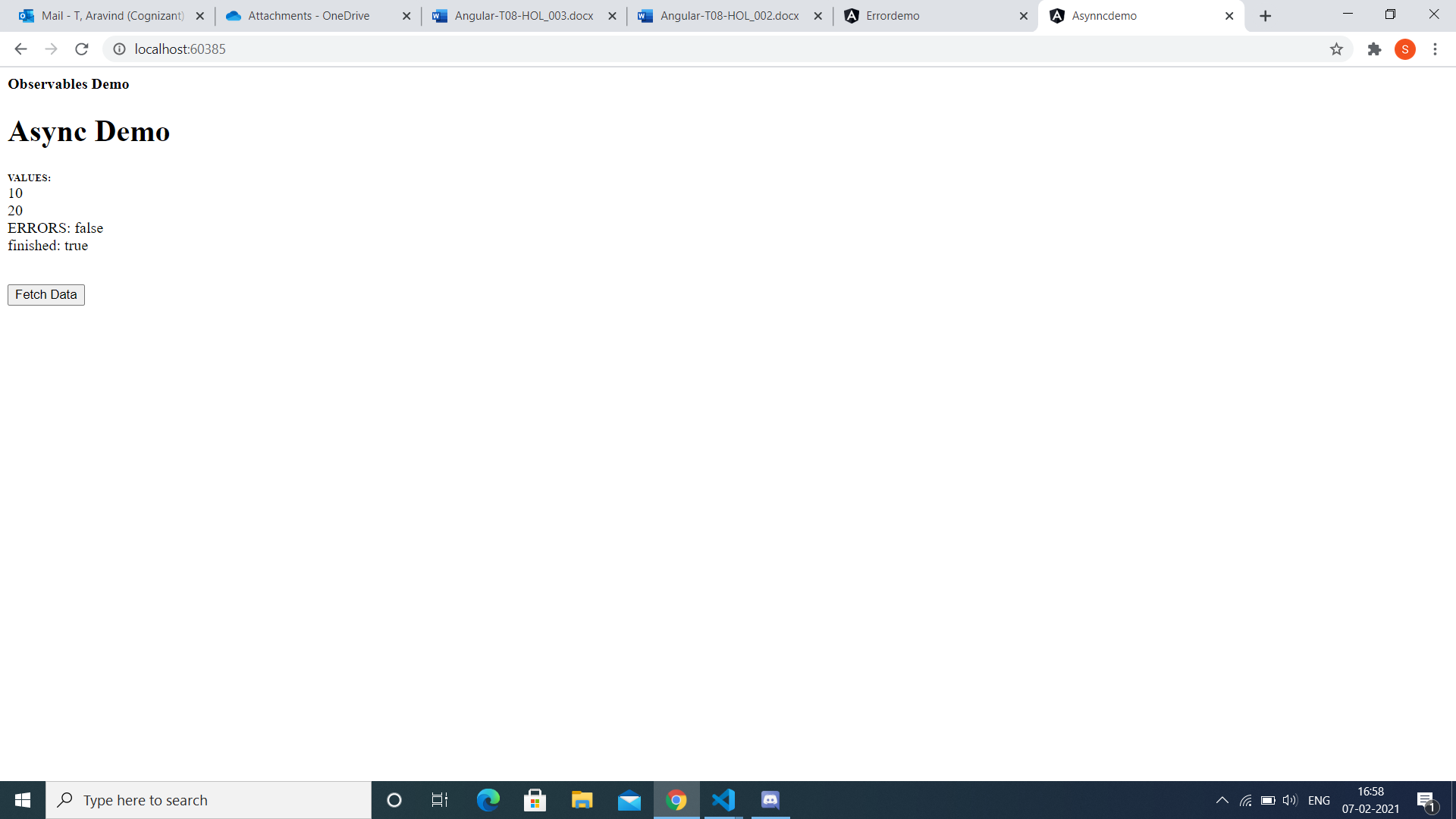
<div style="margin-bottom: 0">ERRORS: {{ errors }}</div>

<div style="margin-bottom: 0">finished: {{ finished }}</div>

<button style="margin-top: 2rem;" (click)="fetchData()">Fetch Data</button>

* Explanation:
* Create a new angular project
* In app.component.ts imports Observable class from rxjs module
* data is of type Observable which holds numeric values
* fetchData() is invoked on click of a button
* A new Observable is created and stored in the variable data
* next() method of Observable sends the given data through the stream.
* With a delay of 1, 2 and 3 seconds, a stream of numeric values will be sent. Complete() method completes the Observable stream i.e., closes the stream.
* In app.component.html ngFor loop is iterated on numArray, which will display the values on the page.
* {{ errors }} will render the value of errors property if any Displays finished property value when complete() method of Observable is executed
* Button click event is bound with fetchData() method which is invoked and creates an observable with a stream of numeric values.
* With the time set limit the value will be displayed when the button is clicked.

Output:



**Angular T08 HOL 002:**

**Task:**

To Implement error handling in Angular application using Http Error Response.

**Code:**

**Api.service.ts**

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

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

import { throwError } from 'rxjs';

import { catchError } from 'rxjs/operators'

@Injectable({

providedIn: 'root'

})

export class ApiService {

  private SERVER = "http://server.com/api/products";

  constructor(private httpClient: HttpClient) { }

  handleError(error: HttpErrorResponse) {

  let errorMessage = 'Unknown error!';

  if (error.error instanceof ErrorEvent) {

  // Client-side errors

  errorMessage = `Error: ${error.error.message}`;

  } else {

  // Server-side errors

  errorMessage = `Error Code: ${error.status}\nMessage: ${error.message}`;

  }

  window.alert(errorMessage);

  return throwError(errorMessage);

  }

  public fetchData(){

    return this.httpClient.get(this.SERVER).pipe(catchError(this.handleError));

    }

  }

App.module.ts

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

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

import { HttpClientModule } from '@angular/common/http';

import { AppRoutingModule } from './app-routing.module';

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

@NgModule({

  declarations: [

    AppComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule,

    HttpClientModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }

Explanation:

* We need to generate a new service called api using cli command (ng g service api)
* Place the file inside the src folder followed by app folder.
* Import the catchError() and throwError() to implement error handling in angular.

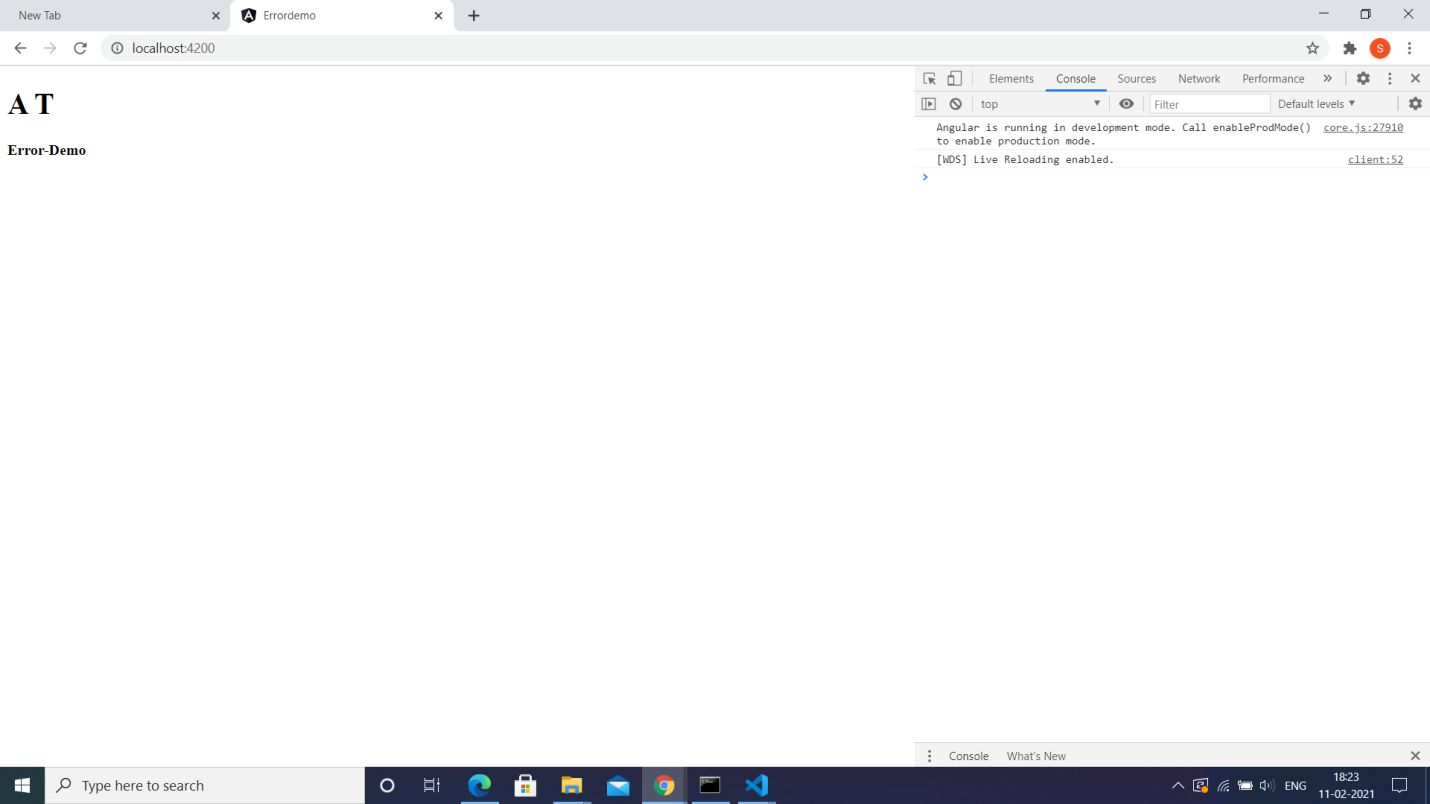
CatchError()

* As usual and like with any RxJs Operator, catchError is simply a function that takes in an input Observable, and outputs an Output Observable.
* With each call to catchError, we need to pass it a function which we will call the error handling function.
* The catchError operator takes as input an Observable that might error out, and starts emitting the values of the input Observable in its output Observable.
* If no error occurs, the output Observable produced by catchError works exactly the same way as the input Observable.

throwError()

* if an error occurs, then the catchError logic is going to kick in. The catchError operator is going to take the error and pass it to the error handling function.
* That function is expected to return an Observable which is going to be a replacement Observable for the stream that just errored out.
* Let's remember that the input stream of catchError has errored out, so according to the Observable contract we cannot use it anymore.
* This replacement Observable is then going to be subscribed to and its values are going to be used in place of the errored out input Observable.

Output:



**Angular T08 HOL 001**

**Task:**

To Create an Angular service that consumes a RESTful service.

Code:

StudentApi.json

{

  "students":[

    {"id":1001,"sname":"A T","course":".NET","fee":10000},

    {"id":1002,"sname":"ABD","course":".NET core","fee":20000}

  ]

}

Student.service.ts

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

import { Student }  from './student';

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

@Injectable({

  providedIn: 'root'

})

export class StudentService {

 // [x: string]: any;

  url:string='http://localhost:3000/students'

  students:Student[] | undefined;

 // \_http: any;

  constructor(private \_http:HttpClient) {

  }

  getStudentsFromAPI()

  {

    return this.\_http.get<Student[]>(this.url);

  }

}

Student.ts

export class Student {

    id:number;

    sname:  string;

    course:string;

    fee:number;

    constructor(id:number, sname:string, course:string, fee:number)

    {

        this.id=id;

        this.sname=sname;

        this.course=course;

        this.fee=fee;

    }

}

App.component.html

<app-student>

</app-student>

Student.component.ts:

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

import {Student } from '../student';

import {StudentService } from '../student.service';

@Component({

  selector: 'app-student',

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

  styleUrls: ['./student.component.css'],

  providers:[StudentService]

})

export class StudentComponent implements OnInit {

  students:Student[] | undefined;

  private \_ss: any;

  constructor(private\_ss:StudentService) {

  }

   getStudentsFromService()

   {

     this.\_ss.getStudentsFromAPI()

         .subscribe((response: Student[] | undefined) =>this.students = response);

   }

  ngOnInit(): void {

  }

}

Student.component.html:

<p>student works!</p>

<h2>List of students</h2>

<button (click) ='getStudentsFromService()'>Get students</button>

<table border="1">

    <tr>

        <th>ID</th>  <th>Name</th> <th>Course</th> <th>Fee</th>

    </tr>

    <tr \*ngfor='let s of students'>

        <td>{{s.id}}</td>  <td>{{s.sname}}</td> <td>{{s.course}}</td>  <td>{{s.fee}}</td>

    </tr>

</table>

App.module.ts

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

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

import {  HttpClientModule } from  '@angular/common/http' ;

import { RouterModule } from '@angular/router';

import { AppRoutingModule } from './app-routing.module';

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

import { StudentComponent } from './student/student.component';

@NgModule({

  declarations: [

    AppComponent,

    StudentComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule,

    HttpClientModule,

    RouterModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }

Explanation:

* the RESTful Web Service available in URL (<https://reqres.in/api/users?page=2>) provides the users list, using HttpClient and display the data.
* Create a json server StudentApi where you enter the details of the student.
* Using the command (json-server –watch StudentApi.json)
* Import the Http Client Module in app.module.ts file .
* Create a Student service type script file using the command ng g s student.
* Create student type script file using ng g cl student.
* Create a new component called Student.
* By clicking the button the student details from thejson server will be displayed in the app component html page.

Output: