Angular 2 Labs

Exercise 1: Data-Driven Forms, Validations and Custom Validations

Step 1: Create a Model class as shown in the following code

```
export class Employee{
   constructor(public empNo:number,
            public empName:string,
            public salary:number,
            public deptName:string,
            public designation:string,
            public tax:number,
           pubic experience:number) {

            }}
```

Step 2: In the Component class declare the FormGroup and FormControl as shown in the following code

```
form: FormGroup;
   emp: Employee;
   empNo: FormControl;
   empName: FormControl;
   salary: FormControl;
   deptName: FormControl;
   designation: FormControl;
   tax:FormControl;
   experience:FormControl
```

Step 3: In the Constructor of the Component define the FormGroup and FormControl mapped with the Fields from the model class as shown in the following code

```
this.emp = new Employee(0, '', 0, '', '');
this.form = new FormGroup({
         'empNo': new FormControl(this.emp.empNo),
         'empName': new FormControl(this.emp.empName),
         'salary': new FormControl(this.emp.salary),
         'deptName': new FormControl(this.emp.deptName),
         'designation': new FormControl(this.emp.designation)
'tax': new FormControl(this.emp.tax)
```

```
'experience': new FormControl(this.emp.experience)

      });
```

In the component, add save() method to save the form data into an Array of Employee. Make sure that the this method will add employee record in the employees array only when the all EmpName, Salary, DeptName is entered.

Add a method to set the designation based on following rules of the Experience value.

- Experience from 0 to 2, is Programmer
- Experience from 2.1 to 4 is Software Engg.
- Experience from 4.1 to 7 is Team Lead
- Experience from 7.1 to 10 is Project Lead.
- Experience more than 10.1 is Project Manager

Select the Experience using the Drop-Down List. The Designation field must be disabled.

Step 4: Modify the employee.html as follows

Note: formSubmitted is a Boolean, set to false in constructor of the component. This will be set to false when the submit button is clicked.

```
<div [hidden]="formSubmitted">
   <form (ngSubmit)="save()"
   [formGroup]="form">
   <table class="table table-bordered table-striped">
   <tr>
      <td>EmpNo</td>
      <td>
         <input type="text"
          class="form-control" formControlName="empNo"
         [(ngModel)]="emp.empNo">
      </td>
   </tr>
  ……Repeat table rows for rest of employee fields with formControlName and
ngModel

</table>


<hr>
<table class="table table-bordered table-striped" [hidden]="formSubmitted">
```

```
   <tr>
      <td>
         <input type="button" (click)="clearInputs()" value="Clear"
          class="btn btn-default">
      </td>
      <td>
         <input type="submit"  value="Submit"
         class="btn btn-success" >
      </td>
   </tr>
</table>
</form>
</div>
<hr>
<div [hidden]="!formSubmitted">
   <table class="table table-bordered table-striped">
    …..This table will show Employee Details in Row/Columns
   </table>
   <input type="button" value="Ok"
     class="btn btn-default"
     (click)="showForm()">
</div>
```

The showForm() method will reset formSubmitted flag to display all <input> elements

Standard Validations

Add the following module into the boot.ts

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

Import the above module in the @NgModule

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

Step 1: Modify input elements of employee.html as following

```
<input type="text"
        class="form-control"  required
        name="empNo"
```

```
          [(ngModel)]="emp.empNo"
          formControlName="empNo"
           pattern="[0-9]+">
```

Set MaxLength, MinLength for EmpName and DeptName fields. The EmpName and DeptName must accept string data. Experience, Salary, must be Positive numeric values.

Custom Validator for the Form.

Step 1: Add a new property into the Employee model class of name **email as string**. Modify the FormGroup declaration in the Component class as shown in the following code

```
this.form = new FormGroup({
        'empNo': new FormControl(this.emp.empNo,
Validators.compose([Validators.required, Validators.pattern('[0-9]+')])),
      'empName': new FormControl(this.emp.empName,
Validators.compose([Validators.required, Validators.minLength(2),
Validators.maxLength(16)])),
      'salary': new FormControl(this.emp.salary,
Validators.compose([Validators.required, Validators.pattern('[0-9]+')])),
      'deptName': new FormControl(this.emp.deptName,
Validators.compose([Validators.required, Validators.pattern('[A-Za-z]+')])),
      'designation': new FormControl(this.emp.designation, Validators.required),
    experience: new FormControl(this.emp.experience,
Validators.compose([Validators.required, Validators.pattern('[0-9]+')])),
```

Add a new input element in the employee.html bond with the email property of the model class. Remove Validation specified in the HTML for input elements

Step 2: Add a new typescript code file in the workspace. This code file will contain the logic for the custom validation

```
export class EmailValidator{

   static emailValidator(c) {
      //Validate email address with only lowercase characters
```

```
    var regExp = /^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*(\.[a-z]{2,4})$/;
    if (c.value.match(regExp)) {
      return null;
    } else {
       return { 'incorrectEmail': true };
      }
    }
}
```

Step 3: Import the above class into the component class and in the formgroup add the following line to attach validation rule for field

```
'email': new FormControl(this.emp.email, Validators.compose([Validators.required,
EmailValidator.emailValidator]))
```

Step 4: Set the Email field in the html file as in following code

```
<div *ngIf="form.controls.email.dirty && !form.controls.email.valid" class="alert
alert-danger">
        <p *ngIf="form.controls.email.errors.required">
            Email Address is mandatory
        </p>
        <p *ngIf="!form.controls.email.valid">
            Email must be valid with all characters must be in lower case
        </p>
      </div>
```

Exercise 2: Create a new Custom Validator class for setting the mobile/phone number of the employee in international form.

Working with Angular 2 Http Service

In this lab we will implement the angular 2 Http service. To implement the service, we need to use the @angular/http module. We also need rxjs/observable. This package is used to collect the notifiable responsreceived from the service. Using @angular/http

package we can have access to the Http, Request, Response and RequestOptions classes.

We will implement this lab from scratch. The IDE used for this las is Visual Studio Code (VSCode).

Very Important Note: The Service here can be created using Node+Express OR WEB API

Step 1: Create a new folder of name ServiceDemo, add a app subfolder in it. Open VSCode and using File-Open Folder, open ServiceDemo folder in the VSCode.

Step 2: In the ServiceDemo folder add a new file of name package.json. We need to install following packages

```
{
  "name": "ng2-service-demo",
  "version": "1.0.0",
  "scripts": {
    "start": "concurrently \"tsc -w\" \"node server.js\"",
    "tsc": "tsc",
    "tsc:w": "tsc -w",
    "lite": "lite-server",
    "typings": "typings",
    "postinstall": "typings install"
  },
  "license": "ISC",
  "dependencies": {
    "@angular/common": "2.0.0",
    "@angular/compiler": "2.0.0",
    "@angular/core": "2.0.0",
    "@angular/forms": "2.0.0",
    "@angular/http": "2.0.0",
    "@angular/platform-browser": "2.0.0",
    "@angular/platform-browser-dynamic": "2.0.0",
    "@angular/router": "3.1.0",
    "bootstrap": "3.3.7",
    "co-body": "4.2.0",
    "es6-shim": "0.35.1",
    "koa": "1.2.4",
    "koa-route": "2.4.2",
    "koa-static": "2.0.0",
    "livereload": "0.6.0",
    "reflect-metadata": "0.1.8",
    "rxjs": "5.0.0-rc.1",
    "systemjs": "0.19.39",
    "zone.js": "0.6.25"
  }
}
```

Note that in the **scripts** section we are using **start** key for running server.js. (node server.js)

Step 3: In the ServiceDemo folder add a new file of name index.html. Right-Click on this file and select option **Open in Command Prompt.** This will open the Command prompt. Run the following commands from the Command Prompt.

```
Npm install -g typescript
Npm install -g typings
Npm install -g concurrently
Npm install
```

This will install all required packages. After the successful installation the **node_modules** folder will be created in the project.

Step 4: To create a server we need to create a server application using Koa package (note: http-server can also be used here instead of Koa)

```
var koa = require("koa");
var serve = require("koa-static");
var livereload = require("livereload");

var koaapp = koa();
var server = livereload.createServer();

server.watch(__dirname + "/app/*.js");

koaapp.use(serve("."));

akoaappp.listen(3030);
```

This will host the Http server on port 3030.

Step 5: We need to define the system configuration for the application. To implement it in the ServiceDemo folder add a new file of name systemjs.config.js we need to add code as shown following

```
var map = {
       "rxjs": "node_modules/rxjs",
       "@angular/common": "node_modules/@angular/common",
       "@angular/compiler": "node_modules/@angular/compiler",
       "@angular/core": "node_modules/@angular/core",
       "@angular/platform-browser": "node_modules/@angular/platform-browser",
       "@angular/platform-browser-dynamic": "node_modules/@angular/platform-browser-
dynamic",
       "@angular/http": "node_modules/@angular/http",
       "@angular/forms": "node_modules/@angular/forms",
       "@angular/router": "node_modules/@angular/router"
};
var packages = {
       "rxjs": { "defaultExtension": "js" },
       "@angular/common": { "main": "bundles/common.umd.js", "defaultExtension":
"js" },
```

```
      "@angular/compiler": { "main": "bundles/compiler.umd.js", "defaultExtension":
"js" },
      "@angular/core": { "main": "bundles/core.umd.js", "defaultExtension": "js" },
      "@angular/platform-browser": { "main": "bundles/platform-browser.umd.js",
"defaultExtension": "js" },
      "@angular/platform-browser-dynamic": { "main": "bundles/platform-browser-
dynamic.umd.js", "defaultExtension": "js" },
      "@angular/http": { "main": "bundles/http.umd.js", "defaultExtension": "js" },
      "@angular/forms": { "main": "bundles/forms.umd.js", "defaultExtension": "js"
},
      "@angular/router": { "main": "bundles/router.umd.js", "defaultExtension":
"js" },
      "app": {
            format: 'register',
            defaultExtension: 'js'
      }
};

var config = {
      map: map,
      packages: packages
};

System.config(config);
```

Step 6: In the app folder add a new file of name Employee.ts. This file will contains the Employee class as shown in the following code

```
export class Employee{
    EmpNo:number;
                EmpName:string;
                Salary:number;
                Designation:string;
                DeptName:string;
}
```

Step 7: We need to create a service to make call to external service. In the app folder add a new file of name EmployeeInfoService.ts. Add the following code in it

```
//1.
import {Injectable} from '@angular/core';
import {Http,Response,RequestOptions,Headers} from '@angular/http';
import {Employee} from './Employee';
import {Observable} from 'rxjs/observable';
//2.
@Injectable()
export class EmployeeService {

//3.
    constructor(private http:Http) { }
    //4.
    private serviceUrl = 'http://localhost:8516/api/EmployeeInfoeAPI';

//5.
    getEmployees():Observable<Response>{
           return  this.http.get(this.serviceUrl);
```

```
    }
    //6.
    addEmployee(emp:Employee): Observable<Response>{
          let headers = new Headers({ 'Content-Type': 'application/json' });
         let options = new RequestOptions({ headers: headers });
        return this.http.post(this.serviceUrl,JSON.stringify(emp),options);
    }
    //7.
    editEmployee(emp: Employee): Observable<Response> {
        let headers = new Headers({ 'Content-Type': 'application/json' });
        let options = new RequestOptions({ headers: headers });
        let id=emp.EmpNo;
             return this.http.put(`${this.serviceUrl}/${id}`,
JSON.stringify(emp),options);
       }

//8.
       deleteEmployee(id: number): Observable<Response> {
             return this.http.delete(`${this.serviceUrl}/${id}`);
       }
}
```

The above code has following specifications (Note: The comment numbers of the above code map with the line numbers provided in following points.)

1. Imports all required classes from the installed packages.
2. The Service class with @Injectable decorator.
3. The Constructor with **Http** object passed to it.
4. The Service Url of the external service.
5. The getEmployees() method to receive all employees in the observable.
6. The addEmployee() method is used to add a new employee
7. The editEmployee() method to edit employee.
8. The deleteEmployee() method to delete employee.

Step 8: In the app folder add a new file of name AppComponent.ts. This will contains angular 2 component as shown in the following code

```
//1.
import {Component, OnInit} from '@angular/core';
import { FormsModule } from '@angular/forms';
import {Response,HttpModule} from "@angular/http";
import {Observable} from "rxjs/observable";
import {Employee} from './Employee';
import {EmployeeService} from './EmployeeInfoService';
//2.
@Component({
    selector: 'emp-form',
    templateUrl: 'app/Employee.html',
    providers:[EmployeeService]
})
export class EmployeeComponent implements OnInit {
    public Employees:Employee[];
    public emp:Employee;
    isEdit:boolean;
```

```
    //3.
    constructor(private empSvc:EmployeeService) {
        this.emp = new Employee();
        this.isEdit = false;
     }

//4.
    ngOnInit() {
           this.loadEmployees();
    }
    //5.
    private loadEmployees(){
        this.empSvc.getEmployees()
                    .subscribe((response:Response)=>{
                     this.Employees=response.json();
                    })
    }
//6.
    addEmployee(){
        if (!this.isEdit) {
        this.empSvc.addEmployee(this.emp)
                    .subscribe((response:Response)=>{
                        this.emp = response.json();
                        this.loadEmployees();
                    });
        }else{
            this.empSvc.editEmployee(this.emp)
            .subscribe(()=>{
                        this.loadEmployees();
                    });
                    this.emp = new Employee();
           this.isEdit = false;
        }
    }
    //7.
    editEmp(emp){
      this.emp = emp;
      this.isEdit = true;
    }
    //8.
    deleteEmp(){
        this.empSvc.deleteEmployee(this.emp.EmpNo)
                    .subscribe((response:Response)=>{
                        this.emp = response.json();
                        this.loadEmployees();
                    });
    }

}
```

The above code has following specifications (Note: The comment numbers of the above code map with the line numbers provided in following points.)

1. Imports all required packages to access necessary classes for the components.

2. The Anguar 2 component of name EmployeeComponent. This defines selector ‘emp-form’, templateUrl, providers properties to be used by components.
3. Constructor injected EmployeeService object.
4. ngOnInit method call the loadEmployees() method.
5. The loadEmployees() method which call the getEmployees() method of the Service call.
6. The addEmployee() method is used to either add a new record or edit record.
7. The editEmp() method set the edit flag.
8. The deleteEmployee() method to delete employee.

Step 9: In the app folder add a new file of name Employee.html with the following markup

```
<h2>The Employee Information System</h2>
<form>
<table class="table table-bordered table-stripped">
    <tr>
        <td>
            <table class="table table-bordered table-stripped">
                <tr>
                    <td>EmpNo</td>
                    <td>
                      <input type="text"
                       [(ngModel)]="emp.EmpNo"
                      class="form-control"
                      [ngModelOptions]="{standalone: true}">
                    </td>
                </tr>
                <tr>
                    <td>EmpName</td>
                    <td>
                        <input type="text"  [(ngModel)]="emp.EmpName"
                        class="form-control"
                        [ngModelOptions]="{standalone: true}">
                    </td>
                </tr>
                <tr>
                    <td>Salary</td>
                    <td>
                        <input type="text"  [(ngModel)]="emp.Salary"
                         class="form-control"
                         [ngModelOptions]="{standalone: true}">
                    </td>
                </tr>
                <tr>
                    <td>Designation</td>
                    <td>
                        <input type="text"  [(ngModel)]="emp.Designation"
                         class="form-control"
                         [ngModelOptions]="{standalone: true}">
                    </td>
                </tr>
                <tr>
                    <td>DeptName</td>
                    <td>
                        <input type="text"  [(ngModel)]="emp.DeptName"
```

```
                        class="form-control"
                        [ngModelOptions]="{standalone: true}">
                    </td>
                </tr>
                <tr>
                    <td colspan="4">
                        <input type="reset"
                        value="New">
                        <input type="button"
                        value="Save"
                         (click)="addEmployee()">
                        <input type="button"
                        value="Delete"
                        (click)="deleteEmp()">
                    </td>
                </tr>
            </table>
        </td>
    </tr>
    <tr>
        <td>
            <div style="overflow:scroll;height:200px">
            <table class="table table-bordered table-stripped">
                <tr>
                    <td>EmpNo</td>
                    <td>EmpName</td>
                    <td>Salary</td>
                    <td>Designation</td>
                    <td>DeptName</td>
                    <td></td>
                </tr>
                <tbody>
                    <tr *ngFor = "let Emp of Employees">
                        <td>{{Emp.EmpNo}}</td>
                        <td>{{Emp.EmpName}}</td>
                        <td>{{Emp.Salary}}</td>
                        <td>{{Emp.Designation}}</td>
                        <td>{{Emp.DeptName}}</td>
                        <td>
                            <input type="button" value="Edit"
                             (click)="editEmp(Emp)">
                        </td>
                    </tr>
                </tbody>
            </table>
            </div>

        </td>
    </tr>
</table>
</form>
```

The above markup contains <input> elements which are bind with the properties from the Employee class using ngModel.

Step 10: To define an entry point for the application we need to add define @NgModule. To implement that, in the app folder add a new file of name boot.js and add the following code in it

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

```
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import {Response,HttpModule} from "@angular/http";
import { EmployeeComponent }   from './AppComponent';
import {EmployeeService} from './EmployeeInfoService';

@NgModule({
    imports: [FormsModule,BrowserModule,HttpModule],
    declarations: [EmployeeComponent],
    providers: [EmployeeService],
    bootstrap: [EmployeeComponent]
})
export class NameModule { }
platformBrowserDynamic().bootstrapModule(NameModule);
```

The imports FormsModule, BrowserModule and HttpModule. These are used for two-way binding, ngModelOptions and http service calls respectively. The declarations use the EmployeeComponent to load it in NgModule. The providers property is used to inject the EmployeeService. The bootstrap property specifies that EmployeeComponent is the first component.

Step 11: In the index.html add the following code

```
<!DOCTYPE html>
<html>

<head>
  <base href="/" />
  <title>Angular 2 setup</title>
  <script>
        document.write('<script src="http://' + (location.host ||
'localhost').split(':')[0] +
        ':35729/livereload.js?snipver=1"></' + 'script>')
    </script>
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.css" />
  <style>
    .selected-order {
      background-color: #CFD8DC;
    }
  </style>
</head>

<body>
  <div class="container">
    <app>Loading...</app>
  </div>
  <emp-form></emp-form>
  <script src="node_modules/es6-shim/es6-shim.min.js"></script>
  <script src="node_modules/reflect-metadata/Reflect.js"></script>
  <script src="node_modules/systemjs/dist/system.src.js"></script>
  <script src="node_modules/zone.js/dist/zone.js"></script>
  <script src="systemjs.config.js"></script>
  <script>
        System.import('./app/boot')
            .then(null, console.error.bind(console));
    </script>
</body>

</html>
```

Run the External service (WEB API OR Node+Express OR other REST Service)

From the command prompt run the following command

```
npm run start
```

Open the browser and enter the following address

http://localhost:3030

This will show Employee Information form, Test add, Edit and delete functionalities.