**First create the service**

**Then do the HTTP observables**

**===============================================**

**Angular Services: Data Sharing & Logic Across Components**

**Introduction**

In this article, we are going to utilize Angular Services.

A Service is a class having certain operations for a specific purpose.

We use services in Angular to share the data among components.

In an application, the service is responsible for providing the data to the component.

So, the sharing of data is also one of the responsibilities of the service.

Another facility that the service provides is application logic. We create the service that has a certain logic and utilizes that in the component so as to make the component directly using the logic/operations without binding that logic to the component individually. We create a service for a certain request that interacts externally to get the data from somewhere and give that to the View. We can perform so many operations by using Services in Angular.

Prerequisites

* HTML, CSS, and JS
* Basic coding knowledge of TypeScript
* [Components](https://www.c-sharpcorner.com/article/overview-of-component-in-angular/)

Let us create a sample TestApp. For this, you should have the below for the development environment installed.

1. Node
2. Npm (comes when you install node)
3. Angular CLI
4. Text Editor.

To create a new application, run the below command on your preferred location.

ng new TestApp

Bash

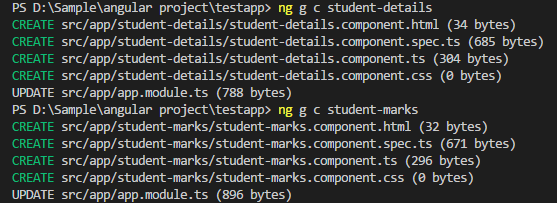
Once your command is completed, you will have a TestApp folder created inside your sample folder.

Note. See my previous article “[Getting Started with Angular CLI](https://www.c-sharpcorner.com/article/getting-started-with-angular-5-with-angular-cli/)” if you want to learn the installation and introduction from the basics and want to get started with the execution of a sample application.

Let us start with a simple application having details of students in two different components. Create two components.

* >ng g c student-details
* >ng g c student-marks

The following files will be added.



Open student-details.component.ts and add the below code.

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

@Component({

selector: 'app-student-details',

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

styleUrls: ['./student-details.component.css']

})

export class StudentDetailsComponent implements OnInit {

public students = [

{"id" : 1001, "name" : "Irshad", "marks" : 90},

{"id" : 1002, "name" : "Imran", "marks" : 80},

{"id" : 1003, "name" : "Rahul", "marks" : 70},

{"id" : 1004, "name" : "Ajay", "marks" : 85},

{"id" : 1005, "name" : "Sunny", "marks" : 60}

];

constructor() { }

ngOnInit() {

}

}

TypeScript

Open student-details.component.html and add the below lines of code.

<h2>Student Details:</h2>

<div \*ngFor="let stud of students">

Id: {{stud.id}}, Name: {{stud.name}}

</div>

Markup

Open student-marks.component.ts and add the below lines.

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

@Component({

selector: 'app-student-marks',

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

styleUrls: ['./student-marks.component.css']

})

export class StudentMarksComponent implements OnInit {

public students = [

{"id" : 1001, "name" : "Irshad", "marks" : 90},

{"id" : 1002, "name" : "Imran", "marks" : 80},

{"id" : 1003, "name" : "Rahul", "marks" : 70},

{"id" : 1004, "name" : "Ajay", "marks" : 85},

{"id" : 1005, "name" : "Sunny", "marks" : 60}

];

constructor() { }

ngOnInit() {

}

}

TypeScript

Open student-marks.component.html and add this code.

<h2>Marks Details:</h2>

<div \*ngFor="let stud of students">

Id : {{stud.id}}, Marks : {{stud.marks}}

</div>

Markup

Open app.component.html and add the following code.

<!-- The content below is only a placeholder and can be replaced. -->

<div style="text-align:center">

<h1>

Welcome to {{ title }}!

</h1>

<app-student-details></app-student-details>

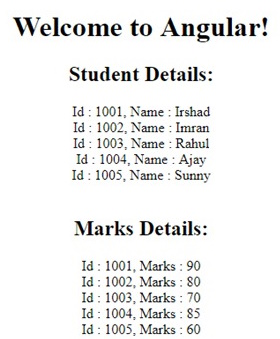
<br>

<app-student-marks></app-student-marks>

</div>

Markup

Run the application.



The above application is working but it is not a good programming practice. You can see that we have added the same set of data to both the files - student-details.component.ts and in student-marks.component.ts.

public students = [

{"id" : 1001, "name" : "Irshad", "marks" : 90},

{"id" : 1002, "name" : "Imran", "marks" : 80},

{"id" : 1003, "name" : "Rahul", "marks" : 70},

{"id" : 1004, "name" : "Ajay", "marks" : 85},

{"id" : 1005, "name" : "Sunny", "marks" : 60}

];

TypeScript

This same data is being used in both the components, which means the component is not only responsible for displaying or working with the data, it is also taking the responsibility of generating the data, which actually should not be its responsibility. These components should only be responsible for using the given data. This is breaking the single responsibility rule of the programming.

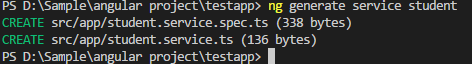
**Here comes Angular Services into the picture**. We should have a z. These components should only perform one responsibility: dealing with how to use the data given by the service.

So, **when we inject the service**, the component will only use the data given by the Service instead of knowing how the data is coming from that service. This makes the service responsible for generating the data and components are responsible for making it display.

Let us start creating a service and use that in your application.

ng generate service student

Bash



Open student.service.ts and add the below code.

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

@Injectable({

providedIn: 'root'

})

export class StudentService {

students = [

{"id" : 1001, "name" : "Irshad", "marks" : 90},

{"id" : 1002, "name" : "Imran", "marks" : 80},

{"id" : 1003, "name" : "Rahul", "marks" : 70},

{"id" : 1004, "name" : "Ajay", "marks" : 85},

{"id" : 1005, "name" : "Sunny", "marks" : 60}

];

constructor() { }

getStudents(){

return this.students;

}

}

TypeScript

Open app.module.ts and add these services to the provider.

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

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

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

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

import { TestComponent } from './test/test.component';

import { StudentDetailsComponent } from './student-details/student-details.component';

import { StudentMarksComponent } from './student-marks/student-marks.component';

@Pipe({

name: 'namePipe'

})

class NamePipe{

transform(value : string, defaultValue : string) : string{

if(value != ""){

return value;

} else {

return defaultValue;

}

}

}

@NgModule({

declarations: [

AppComponent,

TestComponent,

NamePipe,

StudentDetailsComponent,

StudentMarksComponent

],

imports: [

BrowserModule,

FormsModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

TypeScript

Open student-details.component.ts and add the below code.

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

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

@Component({

selector: 'app-student-details',

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

styleUrls: ['./student-details.component.css']

})

export class StudentDetailsComponent implements OnInit {

public students = [];

constructor(private studentService : StudentService) {

this.students = studentService.getStudents();

}

ngOnInit() {

}

}

TypeScript

Open student-marks.component.ts and add this code.

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

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

@Component({

selector: 'app-student-marks',

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

styleUrls: ['./student-marks.component.css']

})

export class StudentMarksComponent implements OnInit {

public students = [];

constructor(private studentService : StudentService) {

this.students = studentService.getStudents();

}

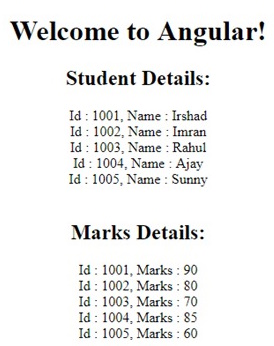
ngOnInit() {

}

}

TypeScript

Run the application.



You have achieved the same thing that we got in our previous example, however, this time, the data is coming from a service and our component is only performing a single functionality, i.e., to render the data.

**HTTP And Observables In Angular**

**Introduction**

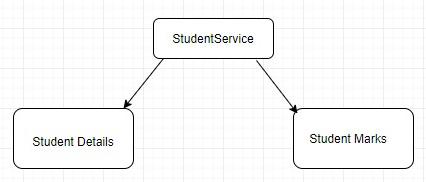
Using HTTP in Angular we are going to fetch data from the web server. We will make a call to the web server that will in return provide data. For a better understanding of this process, we need to know what an HTTP call is, as well as what is observable that is returned from the HTTP call.

Prerequisites

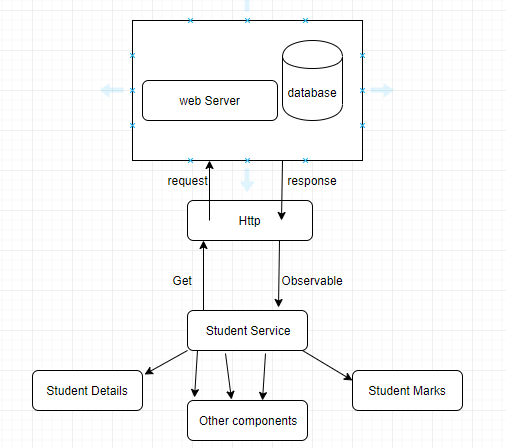
* HTML, CSS, and JS
* Basics of TypeScript.

**Note.**See my previous article “[Services in Angular](https://www.c-sharpcorner.com/article/services-in-angular/)” and create the Application using services.

So far what we have done is to give the data array that has some values directly to the component.



Now, what we need to achieve is to return the same data but with the help of HTTP.



In the above diagram, the studentService will call or make the HTTP request from the HTTP module. This HTTP request will hit the web API or service that will in return fetch the data from the database and send it back as an HTTP response. This response that we are getting from the HTTP is nothing but the observable. Now the observable needs to be cast to the particular type and the studentService will cast this observable data to the array of employees and return it to the subscribed component. HTTP is just a two-way process, the first is to send the request and the second is to receive the response.

The web API/service or the database is only responsible for giving the data and their job is done.

HTTP response is nothing but the Observable returned by HttpClient.get

Components subscribe to the service and receive and operate the data accordingly. Student Details may only display id, and the name of the students, whereas the Student Marks component will display the ID and marks of the students.

How it works,

* Make a request from StudentService.
* Get the observable and cast it.
* Subscribe the observable to the components.
* Store it in the local variables to make it useful in your component.

Let us start using the HTTP functionality.

Open your application.

Open app.module.ts and add the below contents,

Import the HttpClientModule

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

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

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

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

import { StudentDetailsComponent } from './student-details/student-details.component';

import { StudentMarksComponent } from './student-marks/student-marks.component';

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

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

@NgModule({

declarations: [

AppComponent,

StudentDetailsComponent,

StudentMarksComponent

],

imports: [

BrowserModule,

HttpClientModule

],

providers: [StudentService],

bootstrap: [AppComponent]

})

export class AppModule { }

TypeScript

**Create JSON file on location ‘/assets/AppData/students.json’.**

Add the below contents under students.json.

[

{"id": 1001, "name": "Irshad", "marks": 90},

{"id": 1002, "name": "Imran", "marks": 80},

{"id": 1003, "name": "Rahul", "marks": 70},

{"id": 1004, "name": "Ajay", "marks": 85},

{"id": 1005, "name": "Sunny", "marks": 60}

]

JSON

Right now we are using the JSON file to get the data. When the data will come from web API or services then we just need to change the requesting URL.

Add a file (aks interface) student.ts under the app directory and add the below contents.

export interface IStudent {

id: number;

name: string;

marks: number;

}

**TypeScript**

**Open student.service.ts and add the below contents.**

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

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

import { IStudent } from './student';

import { Observable } from 'rxjs';

@Injectable()

export class StudentService {

private url: string = '/assets/AppData/students.json';

constructor(private http: HttpClient) { }

getStudents(): Observable<IStudent[]> {

return this.http.get<IStudent[]>(this.url);

}

}

TypeScript

You can also use ‘any’ in the observable cast.

getStudents(): Observable<any[]> {

return this.http.get<any[]>(this.url);

}

TypeScript

Open student-details.component.ts and add the below contents.

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

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

@Component({

selector: 'app-student-details',

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

styleUrls: ['./student-details.component.css']

})

export class StudentDetailsComponent implements OnInit {

public students = [];

constructor(private studentService: StudentService) { }

ngOnInit() {

this.studentService.getStudents().subscribe(data => this.students = data);

}

}z

**TypeScript**

Open student-marks.component.ts and add the below contents.

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

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

@Component({

selector: 'app-student-marks',

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

styleUrls: ['./student-marks.component.css']

})

export class StudentMarksComponent implements OnInit {

public students = [];

constructor(private studentService: StudentService) { }

ngOnInit() {

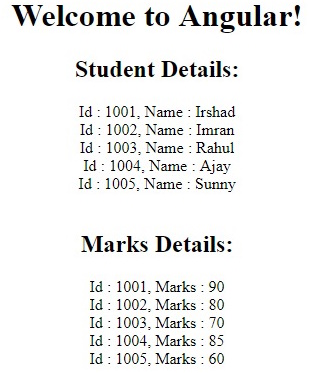
this.studentService.getStudents().subscribe(data => this.students = data);

}

}

TypeScript

Run the application,



What we have done,

1. Added HttpClientModule in our application.
2. Injected this as a dependency injection in our student services.
3. Invoked the get method of from instance HTTP with proper URL. The get method returned the observable that we cast to the IStudent array type.
4. We have a method getStudents() that will return the observable.
5. The observable will not return any data until some component subscribes to it.
6. When we subscribe we get the data assign it to the property in that component class and bind it to the HTML view.

**Building Template-Driven Form in Angular**

In Template Driven Forms we specify behaviors/validations using directives and attributes in our template and let it work behind the scenes.

All things happen in Templates hence very little code is required in the component class. This is different from the reactive forms, where we define the logic and controls in the component class.

1. The form is set up using ngForm directive.
2. Controls are set up using the ngModel directive.
3. ngModel provides the two-way data binding.
4. The Validations are configured in the template via directives.

**Steps To Build Template-Deiven Form**

**Step 1:** Create a root directory called Form using the following command:

mkdir Form

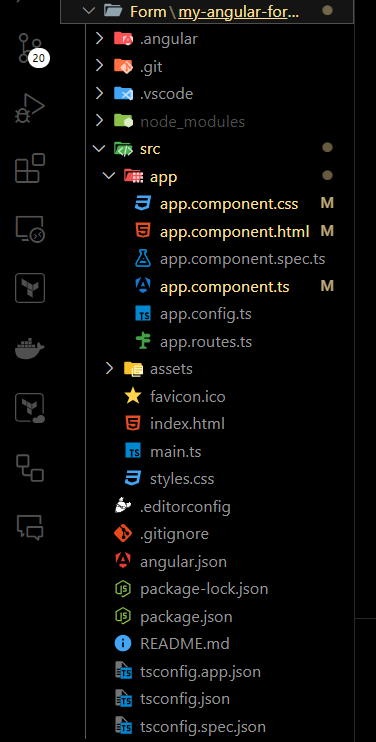
**Step 2:**Navigate to the root directory. Run the following command to initialize a new Angular app:

npm install -g @angular/cli  
cd From  
ng new my-angular-form-project

**Dependencies**

"dependencies": {  
 "@angular/animations": "^17.3.0",  
 "@angular/common": "^17.3.0",  
 "@angular/compiler": "^17.3.0",  
 "@angular/core": "^17.3.0",  
 "@angular/forms": "^17.3.0",  
 "@angular/platform-browser": "^17.3.0",  
 "@angular/platform-browser-dynamic": "^17.3.0",  
 "@angular/router": "^17.3.0",  
 "rxjs": "~7.8.0",  
 "tslib": "^2.3.0",  
 "zone.js": "~0.14.3"  
 }

**Folder Structure**

Folder Structure

**Example:**In this example we are creating a basic Template-Driven Form

*<!-- src/app/app.component.html -->*

<**form** #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)">

<**p**>

<**label** for="firstname">First Name</**label**>

<**input** type="text" name="firstname" ngModel #fname="ngModel">

</**p**>

<**p**>

<**label** for="lastname">Last Name</**label**>

<**input** type="text" name="lastname" ngModel>

</**p**>

<**p**>

<**label** for="email">Email </**label**>

<**input** type="text" id="email" name="email" ngModel>

</**p**>

<**p**>

<**label** for="gender">Gender</**label**>

<**input** type="radio" value="male" name="gender" ngModel> Male

<**input** type="radio" value="female" name="gender" ngModel> Female

</**p**>

<**p**>

<**label** for="isMarried">Married</**label**>

<**input** type="checkbox" name="isMarried" ngModel>

</**p**>

<**p**>

<**label** for="country">Country</**label**>

<**select** name="country" ngModel>

<**option** [ngValue]="c.id" \*ngFor="let c of countryList">

{{c.name}}

</**option**>

</**select**>

</**p**>

<**p**>

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

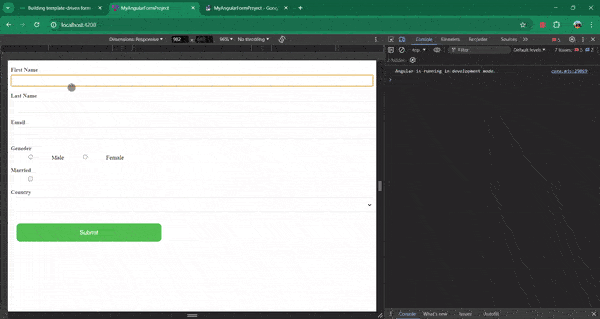
</**p**>

</**form**>

**To run the application, type the following command:**

ng serve

**Output**

Building Template-Driven Form in Angular

**Benefits of Template-Driven Forms**

* **Simplicity:** Template-driven forms are simple to set up and use, making them ideal for smaller or less complex forms. They allow you to define your form in the template, which can be more intuitive for developers familiar with HTML.
* **Two-Way Data Binding:**With Angular’s two-way data binding, template-driven forms can automatically update the model as the user interacts with the form. This feature reduces boilerplate code and makes it easier to manage form data.
* **Built-In Validation:** Angular’s built-in validators can be easily applied to form controls using attributes like required, minlength, maxlength, and more. Custom validators can also be added if needed.
* **Integration with Angular Directives:**Template-driven forms uses Angular’s powerful directives, such as ngModel, ngSubmit, and ngForm, to create and manage forms efficiently.

**How To Use Reactive Forms in Angular?**

In Angular, the forms are an important part of handling user input and validation. Reactive Forms offers a model-driven approach that provides greater control and flexibility by managing form controls, validation, and data binding directly within the component class.

**Core Components**

1. **FormGroup:**Represents a collection of FormControl instances. It is used to group related form controls, such as those in a form.
2. **FormControl:** Represents a single form control. It manages the value and validation status of an individual input field.
3. **FormArray:** Manages an array of FormControl or FormGroup instances, allowing for dynamic and repeatable form controls.
4. **Validators:**Provides built-in validation functions, such as required, minLength, email, and min, which can be used to validate form controls.

**Approach**

* Import ReactiveFormsModule in the AppModule.
* We will create FormGroup and FormControl instances within the component's ngOnInit method, defining form controls and their validations.
* We then bind the form model to the template using [formGroup] and formControlName and handle form submission with a method that processes the form data.

**Steps to Use Reactive Forms in Angular**

**Step 1: Install Angular CLI**

If you haven’t installed Angular CLI yet, install it using the following command

npm install -g @angular/cli

**Step 2: Create a New Angular Project**

ng new form-app --no-standalone  
cd form-app

**Step 3: Create Standalone Component**

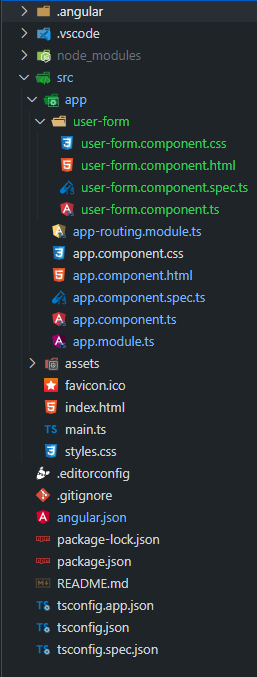
Create a standalone component. You can generate a standalone component using the Angular CLI:

ng generate component user-form

**Dependencies**

"dependencies": {  
 "@angular/animations": "^18.1.4",  
 "@angular/common": "^18.1.4",  
 "@angular/compiler": "^18.1.4",  
 "@angular/core": "^18.1.4",  
 "@angular/forms": "^18.1.4",  
 "@angular/platform-browser": "^18.1.4",  
 "@angular/platform-browser-dynamic": "^18.1.4",  
 "@angular/router": "^18.1.4",  
 "rxjs": "~7.8.0",  
 "tslib": "^2.3.0",  
 "zone.js": "~0.14.10"  
}

**Project Structure**

Folder Structure

**Example:**In this example, we are using Reactive Forms in Angular to create a user form with fields for name, email, and age. The form includes validation rules and displays error messages if the inputs are invalid. After submission, if the form is valid, the form data is shown centered below the submit button. This approach uses Angular’s form controls and validation mechanisms to handle user inputs and feedback effectively.

*<!-- src/app/user-form/user-form.component.html -->*

<**h1** class="header">GeeksforGeeks</**h1**>

<**h3**>Reactive Forms in Angular</**h3**>

<**form** [formGroup]="userForm" (ngSubmit)="onSubmit()">

*<!-- Existing Fields -->*

<**label** for="name">Name:</**label**>

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

<**div** \*ngIf="userForm.get('name')!.invalid && userForm.get('name')!.touched">

Name is required and must be at least 3 characters long.

</**div**>

<**label** for="email">Email:</**label**>

<**input** id="email" type="email" formControlName="email">

<**div** \*ngIf="userForm.get('email')!.invalid && userForm.get('email')!.touched">

Enter a valid email address.

</**div**>

<**label** for="age">Age:</**label**>

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

<**div** \*ngIf="userForm.get('age')!.invalid && userForm.get('age')!.touched">

Age is required and must be 18 or older.

</**div**>

*<!-- New Fields -->*

<**label** for="phoneNumber">Phone Number:</**label**>

<**input** id="phoneNumber" type="text" formControlName="phoneNumber">

<**div** \*ngIf="userForm.get('phoneNumber')!.invalid &&

userForm.get('phoneNumber')!.touched">

Phone number is required and must be 10 digits.

</**div**>

<**label** for="address">Address:</**label**>

<**input** id="address" type="text" formControlName="address">

<**div** \*ngIf="userForm.get('address')!.invalid && userForm.get('address')!.touched">

Address is required.

</**div**>

<**label** for="country">Country:</**label**>

<**select** id="country" formControlName="country">

<**option** value="" disabled>Select your country</**option**>

<**option** \*ngFor="let country of countries" [value]="country">{{ country }}</**option**>

</**select**>

<**div** \*ngIf="userForm.get('country')!.invalid && userForm.get('country')!.touched">

Country is required.

</**div**>

<**label** for="dateOfBirth">Date of Birth:</**label**>

<**input** id="dateOfBirth" type="date" formControlName="dateOfBirth">

<**div** \*ngIf="userForm.get('dateOfBirth')!.invalid &&

userForm.get('dateOfBirth')!.touched">

Date of Birth is required.

</**div**>

<**label** for="password">Password:</**label**>

<**input** id="password" type="password" formControlName="password">

<**div** \*ngIf="userForm.get('password')!.invalid && userForm.get('password')!.touched">

Password is required and must be at least 6 characters long.

</**div**>

<**label** for="confirmPassword">Confirm Password:</**label**>

<**input** id="confirmPassword" type="password" formControlName="confirmPassword">

<**div** \*ngIf="userForm.get('confirmPassword')!.invalid &&

userForm.get('confirmPassword')!.touched">

Confirm Password is required.

</**div**>

<**div** \*ngIf="userForm.errors?.['passwordsMismatch']">

Passwords do not match.

</**div**>

<**button** type="submit" [disabled]="userForm.invalid">Submit</**button**>

</**form**>

<**div** \*ngIf="userForm.valid" class="form-data">

<**pre**>{{ userForm.value | json }}</**pre**>

</**div**>

**Steps to run this Project**

ng serve --open