SSGMCE/FRM/32-B

SSGMCE	SHRI SANT GAJANAN MAHARAJ COLLEGE OF ENGG.			**LABORATORY MANUAL**	
	PRACTICAL EXPERIMENT INSTRUCTION SHEET				
	EXPERIMENT TITLE: : DESIGNING AND WORKING WITH TEMPLATE DRIVEN FORMS IN ANGULAR.				
EXPERIMENT NO.: **SSGMCE-WI-**IT-5IT09-07		ISSUE NO.: 00		ISSUE DATE: 30-07-2023	
REV. DATE:	REV. NO.:	DEPTT.: INFORMATION TECHNOLOGY			
LABORATORY: Computer Laboratory- 01				SEMESTER: V	PAGE :1 OF 3

1.0) AIM: To study and implement **Reactive Forms** in Angular 20 using the FormGroup, FormControl, and FormBuilder APIs for handling form input, validation, and submission.

2.0) SCOPE:

In Angular, "Template Driven Forms" is one of the two approaches to working with forms. It's designed to simplify form handling by allowing you to define form controls directly in the template.

3.0) FACILITIES/ APPARATUS:

i)Hardware: I3 Processor, 8GB RAM, HD Monitor, Windows 10, Internet connectivity

ii) Software: Updated Web browser, Node js, Angular CLI, VS Code.

4.0) THEORY:

In Angular, there are two main approaches to creating forms: **Template-Driven Forms** and **Reactive Forms**.
Reactive Forms (also called **Model-Driven Forms**) provide a **structured and programmatic approach** to handling forms. Unlike template-driven forms, where logic resides in templates, reactive forms use explicit form models defined in the component class.

Key Features of Reactive Forms:

1. **FormGroup & FormControl:**
    - FormControl tracks the value and state of a single input field.
    - FormGroup is a collection of form controls that tracks their combined value and validation status.
2. **FormBuilder:**

    A service that simplifies form model creation with concise syntax.

PREPARED BY:	APPROVED BY:(H.O.D.)

3. **Synchronous Access:**
   - Reactive forms provide synchronous access to the form model, which makes them predictable and testable.
4. **Validation:**
   - Angular provides built-in validators (required, minLength, pattern) and also allows custom validators.
5. **Observables Support:**
   - Form controls are based on observables, which allow reacting to value changes in real time.

5.0) PROCEDURE:

1. **Setup:**

   Import **ReactiveFormsModule** from `@angular/forms` in your app module.

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

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

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

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


@NgModule({

  declarations: [AppComponent],

  imports: [BrowserModule, ReactiveFormsModule],

  bootstrap: [AppComponent]

})

export class AppModule {}
```

2. Create Form Model in Component:

```
import { Component } from '@angular/core';import { FormGroup, FormControl, Validators,
FormBuilder } from '@angular/forms';
@Component({
  selector: 'app-reactive-form',
  templateUrl: './reactive-form.component.html',
  styleUrls: ['./reactive-form.component.css']
})
export class ReactiveFormComponent {
  registrationForm: FormGroup;
  constructor(private fb: FormBuilder) {
    // Using FormBuilder
    this.registrationForm = this.fb.group({
      name: ['', [Validators.required, Validators.minLength(3)]],
      email: ['', [Validators.required, Validators.email]],
      phone: ['', [Validators.required, Validators.pattern('^[0-9]{10}$')]],
      password: ['', [Validators.required, Validators.minLength(6)]],
      confirmPassword: ['', Validators.required]
    });
  }
  onSubmit() {
    console.log(this.registrationForm.value);
    alert('Form Submitted Successfully!');
  }
}
```

3. Design Template with FormGroup Binding:

```
<div class="container mt-4">
  <h3 class="text-primary">Reactive Form Example</h3>
  <form [formGroup]="registrationForm" (ngSubmit)="onSubmit()">

    <!-- Name -->
    <div class="mb-3">
      <label>Name</label>
      <input type="text" formControlName="name" class="form-control" />
      <div class="text-danger" *ngIf="registrationForm.get('name')?.invalid &&
registrationForm.get('name')?.touched">
        Name is required (min 3 chars).
      </div>
    </div>

    <!-- Email -->
    <div class="mb-3">
      <label>Email</label>
      <input type="email" formControlName="email" class="form-control" />
      <div class="text-danger" *ngIf="registrationForm.get('email')?.invalid &&
registrationForm.get('email')?.touched">
```

```
        Enter a valid email.
      </div>
    </div>

    <!-- Phone -->
    <div class="mb-3">
      <label>Phone</label>
      <input type="text" formControlName="phone" class="form-control" />
      <div class="text-danger" *ngIf="registrationForm.get('phone')?.invalid &&
registrationForm.get('phone')?.touched">
        Enter a valid 10-digit phone number.
      </div>
    </div>

    <!-- Password -->
    <div class="mb-3">
      <label>Password</label>
      <input type="password" formControlName="password" class="form-control" />
    </div>

    <!-- Confirm Password -->
    <div class="mb-3">
      <label>Confirm Password</label>
      <input type="password" formControlName="confirmPassword" class="form-control" />
    </div>


<!-- Submit -->
    <button type="submit" class="btn btn-success"
[disabled]="registrationForm.invalid">Submit</button>
  </form>
</div>
```

4. **Then run the project through**
   ng serve
   output will be listen at port **https://localhost:4200**

5.0) Conclusion:

Reactive Forms provide a **structured, scalable, and maintainable approach** to handling forms in Angular applications. With synchronous form state management, powerful validation, and observable-based value tracking, Reactive Forms are more suitable for complex and dynamic form handling compared to template-driven forms.

PREPARED BY:	APPROVED BY:(H.O.D.)