ANGULAR - FORMS

June 22nd, 2023

Objective

- Introduction to Angular Forms
- Difference between Template Driven & Reactive Forms
- Template Driven – Validation and Data Binding
- Reactive Forms – Validation and Data Binding
- Custom Validations

Angular Forms

Forms are basically used to collect the data from the user, it can various fields based on the business requirements

Angular forms comes with inbuild features like

- **Initialize the forms fields**
- **Binds the data from form to Angular Component**
- **Track changes made to the form fields**
- **Validate the inputs**
- **Display helpful errors to the user**

Building Blocks of Angular Forms

- **FormControl**
  - It represents a single input field in an Angular form
- **FormGroup**
  - It is a collection of FormControls . Each FormControl is a property in a FormGroup
- **FormArray**
  - It is a array of FormControls similar to FormGroup but this can dynamically insert and remove controls from form array.

Difference - Template Driven & Reactive Forms

Template Driven	Reactive Form
Easy to use	More flexible, but needs a lot of practice
Suitable for simple scenarios	Handles any complex scenarios
Logic is driven from the template, minimal component	Logic resides mainly in the component or typescript code, less HTML markup
Unit testing is tedious	Easier unit testing

Template Driven Form

- First step is to import FormsModule in App Module

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

- **FormControl -** create template references for form fields, We can then use these references to access the value or validity of these fields.

- **ngModel –** it will use the name attribute to create the FormControl instance for each of the Form field it is attached

```
<input type="text" name="firstname" ngModel>
```

- **Submit Form**

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

Template Driven Form continued..

```
<form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)">
  <p>
    <label for="firstname">First Name</label>
    <input type="text" id="firstname" name="firstname" ngModel required>
  </p>
  <p>
    <label for="lastname">Last Name</label>
    <input type="text" id="lastname" name="lastname" ngModel minlength="5">
  </p>
  <p>
    <label for="email">Email </label>
    <input type="text" id="email" name="email" ngModel email required>
  </p>

  <p>
    <button type="submit" [disabled]="!contactForm.form.valid">Submit</button>
  </p>
  <pre>Value : {{contactForm.value | json }} </pre>
  <pre>Valid : {{contactForm.valid}} </pre>
</form>
```

FIS

Reactive Forms

- **In Reactive forms we define the structure of the form in the component class**
- **Steps:**
  - Import ReactiveFormsModule - **import { ReactiveFormsModule } from '@angular/forms';**
  - Create Form Model using FormGroup & Form Control –

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

public customeForm:FormGroup;

this.customerForm = new FormGroup({
    firstname: new FormControl('', [Validators.required,
Validators.minLength(5),Validators.maxLength(20),Validators.pattern("^[a-zA-Z]+$")]),
    lastname: new FormControl('', [Validators.required, Validators.minLength(5),Validators.maxLength(20)]),
    email: new FormControl('',[Validators.email]),
    gender:new FormControl(''),
    country:new FormControl('')
  })

onSubmit() {
  console.log(this.customerForm.value);
 }
```

Reactive Form continued..

- **Bind the HTML Form to the Form Model**

```
<form [formGroup]="customerForm" (ngSubmit)="onSubmit()">
  <p>
    <label for="firstname">First Name </label>
    <input type="text" id="firstname" name="firstname" formControlName="firstname">
    <span
      *ngIf="((customerForm.controls['firstname']?.invalid) &&(customerForm.controls['firstname']?.dirty ||
customerForm.controls['firstname']?.touched ))“>
    First Name is not valid
    </span>

  </p>
  <p>
    <label for="lastname">Last Name </label>
    <input type="text" id="lastname" name="lastname" formControlName="lastname">
    <span
      *ngIf="((customerForm.controls['lastname']?.invalid) &&(customerForm.controls['lastname']?.dirty ||
customerForm.controls['lastname']?.touched ))“>
      Last Name is not valid
    </span>
  </p>
    <button type="submit" [disabled]="!customerForm.valid">Submit</button>
  </p>
</form>
```

Validations

- **Built-in Validatiors**
  - required, minlength, maxlength, pattern etc.

```
firstname: new FormControl('',[Validators.required,Validators.minLength(10)]),

lastname: new FormControl('',[Validators.required, Validators.pattern("^[a-zA-Z]+$")]),

email:new FormControl('',[Validators.email,Validators.required]),
```

- **Show Error messages**

```
<span
        *ngIf="((customerForm.controls['firstname']?.invalid)
&&(customerForm.controls['firstname']?.dirty || customerForm.controls['firstname']?.touched ))“>
      First Name is not valid
      </span>
```

Custom Validations

- **Custom validations can be built in Angular, it must implement ValidatorFn interface**
- **The validator function must return a list of errors i.e ValidationErrors or null if the validation has passed**
- **Create a new file for Custom Validations – customerValidation**
- Import **AbstractControl** & **ValidationErrors** from angular forms

```
import { AbstractControl, ValidationErrors } from '@angular/forms'
```

- Validation method must receive AbstractControl as parameter, validate the control value and return response

```
export function ageCheck(control: AbstractControl): ValidationErrors | null {
```

```
export function ageCheck (control: AbstractControl) {
    if (control.value < 18) {
      return { valResult: true };
    }
    return null;
}
```

Consuming Custom Validation

- Import the custom validation in the Component

```
import { ageCheck } from './customerValidation.validator';
```

- Add Custom Validator to the FormControl

```
myForm = new FormGroup({
    age: new FormControl('', [ageCheck]),
  })
```

- In component HTML using the variable which is returned in the validation method

```
<div *ngIf="age.errors.valResult">
        The Age should be greater than 18
      </div>
```