**Angular reactive form**

In the Angular reactive form, the developer builds all the control and logic of validation of the form in the component typescript file. The developer has absolute control over how and when the data is synced from the UI to the model and vice versa.

We do not configure Angular form in the template, we are only synchronizing it with the directives, form control name,and formGroup. In this approach, we create a form from the instance of FormGroup. The FormGroup instance allows us to specify form controls and the various validations on the input element of the form. The reactive form is the best choice for creating a complex form, it provides the developer with more control over the form.

**How to use Angular reactive form ?**

We will do this step by step, creating our Angular project, configuring Reactive form setup, and adding with the building block of reactive form. From Form Controls, we will work on various components like Form Groups and Form Builders. Here are abstract steps for creating reactive form.

1. Create Angular project.
2. Import ReactiveFormsModule in our application.
3. Create form model or object in our component and adding form controller class.
4. Create template for reactive form.
5. Add Angular reactive forms validation.

**Import reactive form module in app.module.ts file**

To start using Angular reactive form we need to import ReactiveFormsModule in the app.module.ts file. This module contains all the tools we need now to build our own form.

**Angular reactive form example**

Once we have created our form project and imported ReactiveFormsModule. We will demonstrate the Angular reactive form example and here we have a screenshot of our Angular reactive form demo.

**The core of any reactive form is the FormControl,** which directly represents an individual form element of a form into one FormGroup that represents form. Thus**Angular reactive form is nothing but a set of grouped FormControls**. Grouping form control into a single FormGroup allows us to easily validate, reset and check from states.

The reactive form we should initialize before rendering the template of the component so define form setup in ngOnInit() method.

studentForm = new FormGroup({});

constructor(){}

ngOnInit() {

this.studentForm = this.formBuilder.group({

name : '',

email: '',

..

OR

'number': [null, Validators.required],

'description': [null, [Validators.required, Validators.minLength(10)]],

});

}

In reactive form allow us to group elements into a group and we will group both name and email fields into a FormGroup called Student.  
  
The FormGroup contains FromControls (rendered as key/value pairs) which are used to target each individual input field in our form. The formControls first argument is the default value of the control element and the second argument is validator can be a single or an array of validators we want to apply to this control element. The third argument will be potential asynchronous validators. Let’s add our Angular reactive form example code in the app component controller and template.

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

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

import { MockDepartment } from 'src/assets/data/mockDepartment';

@Component({

selector: 'app-root',

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

styleUrls: ['./app.component.scss']

})

export class AppComponent {

title: string = 'Angular Reactive Form';

**form = new FormGroup({});**

departments: any[] = [];

constructor(private formBuilder: FormBuilder) { }

ngOnInit() {

**this.form = this.formBuilder.group({**

**number : '',**

**description: '',**

**sinceFrom: '',**

**category: '',**

**classType: '',**

**});**

}

submit(form: NgForm) {

console.log(form);

}

}

We have to import the formGroup and FormBuilder from @angular/forms in our component.

**FormGroup**

A FormGroup aggregates the values of each child FormControl into one object, with each control name as the key. It calculates its status by reducing the statuses of its children. For example, if one of the controls in a group is invalid, as a result, the entire group becomes invalid. The FormGroup is useful as a way to group relevant form fields under one group. This gives us the convenience of whether we want to track the form controls individually, or as a group.

**FormBuilder**

Angular has a helper Class called FormBuilder to help us to build a form model with less code. FormBuilder allows us to explicitly declare forms in our components. This allows us to also explicitly list each form control’s validators. FormBuilder has methods as group(), control() and array() that returns FormGroup, FormControl and FormArray respectively. Using FormBuilder we can directly pass the object or array of objects of a class to create the form.

In the app.component.html file add our form HTML code.

<div class="container">

<h4> Angular reactive form Example</h4>

<div class="card">

<div class="card-block">

<form [**formGroup]="form"** (ngSubmit)="submit(form.value)">

<div class="form-group">

<label>Name of students:</label>

<input type="number" class="form-control" formControlName="number" />

</div>

<div class="form-group">

<label>Department:</label>

<select class="form-control" formControlName="category">

<option \*ngFor="let item of departments" value="{{item.name}}">{{item.name}}</option>

</select>

</div>

<div class="form-group">

<label>Department Started on:</label>

<input type="date" class="form-control" formControlName="sinceFrom" />

</div>

<div class="form-group">

<label>Description :</label>

<textarea class="form-control" formControlName="description"></textarea>

</div>

<div class="form-group">

<label>Class on</label>

<input type="radio" name="classType" class="form-control" value="Day class" formControlName="classType">Day Class

<input type="radio" name="classType" class="form-control" value="Evening" formControlName="classType"> Evening Class

</div>

<button class="btn btn-primary btn-block">Submit</button>

</form>

</div>

<p>Form Element : {{ form.value | json}}</p>

</div>

</div><!-- End of container div -->

<router-outlet></router-outlet>

For UI of this form, we have used Bootstrap from ng-bootstrap, if you want to use this UI, then add [bootstrap to our angular project](https://edupala.com/angular-bootstrap-example/). We used the formGroup directive in the reactive form to link with our particular form from component typescript.  
  
**ngSubmit:**  
The ngSubmit binding handles the form submits an event.

We have learned template forms are mutable, on another hand Angular reactive forms are immutable. Every change to the form creates a new state of the form and maintains the integrity of the model in the component.

**Angular reactive forms validation**

The FormBuilder class allows us to initialize and add a validator for each FormControl. Here is a simple demo example of how we can different validators on the reactive form control elements.

ngOnInit() {

this.form = this.fb.group({

name: [null, [Validators.required, Validators.minLength(5)]],

dob: [null, [Validators.required]],

email: [null, [Validators.required,

Validators.pattern("^[a-z0-9.\_%+-]+@[a-z0-9.-]+\\.[a-z]{2,4}$")]],

password: [null, [Validators.required, Validators.minLength(6)]],

});

Here is a screenshot of our Angular reactive forms validation.

Let edit the app.component.ts file to add validation code for our example.

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

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

import { MockDepartment } from 'src/assets/data/mockDepartment';

@Component({

selector: 'app-root',

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

styleUrls: ['./app.component.scss']

})

export class AppComponent {

title: string = 'Angular Reactive Form';

form = new FormGroup({});

departments: any[] = [];

constructor(private formBuilder: FormBuilder) { }

ngOnInit() {

this.form = this.formBuilder.group({

'number': [null, Validators.required],

'description': [null, [Validators.required,

Validators.minLength(10), Validators.maxLength(15)]],

'sinceFrom': [''],

'category': [null, Validators.required],

'classType': [],

});

this.departments = MockDepartment;

}

submit(form: NgForm) {

console.log(form);

}

}

**Displaying error message for form element in the template:**

The FormArray class is a way to manage the collection of Form controls in Angular. The controls can be a FormGroup, a FormControl, or another FormArray. Access the FormArray control, we can access form control two ways in the component template.

1. First by using get() method we can easily access the form array instance.
2. Using form control to check form element validation. If any error occurs then display error message for each form element error related to validation. An example to check email form control validation.

<span \*ngIf="!form.get('email').valid && form.get('email').touched">

Please enter a valid email!

</span>

Now lets, edit our previous form in the app.component.html to add a check and add validation message.

<div class="container">

<h4> Angular reactive form Example</h4>

<div class="card">

<div class="card-block">

<form [formGroup]="form" (ngSubmit)="submit(form.value)">

<div class="form-group">

<label>Name of students:</label>

<input type="number" class="form-control" formControlName="number" />

<div class="alert alert-danger" \*ngIf="!form.controls['number'].valid && form.controls['number'].touched">

Total number of student is required.

</div>

</div>

<div class="form-group">

<label>Department:</label>

<select class="form-control" formControlName="category">

<option \*ngFor="let item of departments" value="{{item.name}}">{{item.name}}</option>

</select>

<div class="alert alert-danger"

\*ngIf="!form.controls['category'].valid && form.controls['category'].touched">

Department is required

</div>

</div>

<div class="form-group">

<label>Department Started on:</label>

<input type="date" class="form-control" formControlName="sinceFrom" />

</div>

<div class="form-group">

<label>Description :</label>

<textarea class="form-control" formControlName="description"></textarea>

<div class="alert alert-danger"

\*ngIf="!form.controls['description'].valid && form.controls['description'].touched">

Description should be of minimum 10 and maximun 50 characters

</div>

</div>

<div class="form-group">

<label>Class on</label>

<input type="radio" name="classType" class="form-control" value="Day class" formControlName="classType">Day

Class

<input type="radio" name="classType" class="form-control" value="Evening" formControlName="classType"> Evening

Class

</div>

<button class="btn btn-primary btn-block">Submit</button>

</form>

</div>

<p>Form Element : {{ form.value | json}}</p>

</div>

</div>

<router-outlet></router-outlet>

**The Built-in Angular Validation Attributes for Angular reactive form validation.**

1. **required:**This attribute is used to specify a value that must be provided.
2. **minlength:** This attribute is used to specify a minimum number of characters.
3. **maxlength:** This attribute is used to specify a maximum number of characters. This type of validation cannot be applied directly to form elements because it conflicts with the HTML5 the attribute of the same name. It can be used with model-based forms.
4. **pattern:** This attribute is used to specify a regular expression that the value provided by the user must match.

**Styling Elements Using Validation Classes in angular reactive form validation**

The underlying Angular framework detects the validation state of the input field and applies its own specific classes to reflect those states. The classes to which an input element is assigned provide details of its validation state as.

1. **ng-dirty:** Input field had interacted with.
2. **ng-touched:** Input field has received focus.
3. **ng-valid:** Data entry has passed validation.
4. **ng-invalid:** Data entry has not passed validation)
5. **ng-pristine:** An element if its contents have not been changed.

We can apply the red border on the input of the invalid form element by adding the following CSS style in the app.component.scss as.

input.ng-invalid.ng-touched {

border: 1px solid red;

}

We can also assign our custom validation methods from our own custom Validator service.