Angular Reactive Forms Example

6 Comments / 8 minutes of reading / March 9, 2023

← Set Value in Template Driven Forms

Angular Tutorial

Form Builder →

Reactive forms (also known as Model-driven forms) are one of the two ways to build Angular forms. In this tutorial, we will learn how to build a simple Example Reactive Form. To build reactive forms, first, we need to import `ReactiveFormsModule`. We then create the Form Model in component class using Form Group, Form Control & FormArrays. Next, we will create the HTML form template and bind it to the Form Model.

Table of Contents

What are Reactive Forms?

Reactive forms are forms where we define the structure of the form in the component class. i.e. we create the form model with Form Groups, Form Controls, and FormArrays. We also define the validation rules in the component class. Then, we bind it to the HTML form in the template. This is different from the template-driven forms, where we define the logic and controls in the HTML template.

How to use Reactive Forms

1. Import `ReactiveFormsModule`
2. Create Form Model in component class using FormGroup, FormControl & FormArrays
3. Create the HTML Form resembling the Form Model.
4. Bind the HTML Form to the Form Model

Reactive Forms Example Application

Use `ng new` to create a new application

```

ng new mdf  --routing=true --style=css

```

Run `ng serve` and verify if everything is installed correctly.

Import ReactiveFormsModule

To work with Reactive forms, we must import the `ReactiveFormsModule`. We usually import it in root module or in a shared module. The `ReactiveFormsModule` contains all the form directives and constructs for working with angular reactive forms.

```

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { ReactiveFormsModule } from '@angular/forms';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    ReactiveFormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

Model

In the template-driven approach, we used `ngModel` & `ngModelGroup` directive on the HTML elements. The `FormsModule` automatically creates the FormGroup & FormControl instances from the HTML template. This happens behind the scene.

In the Reactive Forms approach, It is our responsibility to build the Model using `FormGroup`, `FormControl` and `FormArray`.

The `FormGroup`, `FormControl` & `FormArray` are the three building blocks of the Angular Forms. We learned about them in the Angular Forms Tutorial.

`FormControl` encapsulates the state of a ***single form element*** in our form. It stores the value and state of the form element and helps us to interact with them using properties & methods.

`FormGroup` represents a collection of form controls. It can also contain other FormGroups and FormArrays. In fact, an angular form is a `FormGroup`.

Let's create the model for our Form.

First, we need to import `FormGroup`, `FormControl` & `Validator` from the `@angular/forms`. Open the `app.component.ts` and the add following import statement.

```

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

```

Creating the Form Model

The FormGroup is created with the following syntax

```

contactForm = new FormGroup({})

```

In the above, we have created an instance of a FormGroup and named it as contactForm . The contactForm is top level FormGroup and is the name of our form model.

The FormGroup takes 3 arguments. a collection of a child controls (which can be FormControl, FormArray, or another FormGroup), a validator , and an asynchronous validator . The validators are optional.

Adding the Child Controls

The next step is to add the child controls to the contactForm .

The first argument to FormGroup is the collection of controls. The Controls can be FormControl , FormArray or another FormGroup . It can be done by creating a new instance for FormControl (or FormGroup or FormArray)

```

contactForm = new FormGroup({
  firstname: new FormControl(),
  lastname: new FormControl(),
  email: new FormControl(),
  gender: new FormControl(),
  isMarried: new FormControl(),
  country: new FormControl()
})

```

In the above example, we have added five `FormControl` instances each representing the properties `firstname`. `lastname`. `email`, `gender`, `ismarried` & `country`.

The Other two arguments to `FormGroup` are `Sync Validator` & `Async Validator`. They are optional.

With this our model is ready.

HTML Form

The next task is to build an HTML form. The following is a regular HTML `form`. We enclose it in a `<form>` tag. We have included two text inputs (FirstName & LastName), an email field, a radio button (gender), a checkbox (isMarried), and a select list (country). These are simple HTML Form elements.

```

<form>

  <p>
    <label for="firstname">First Name </label>
    <input type="text" id="firstname" name="firstname">
  </p>

  <p>
    <label for="lastname">Last Name </label>
    <input type="text" id="lastname" name="lastname">
  </p>

  <p>
    <label for="email">Email </label>
    <input type="text" id="email" name="email">
  </p>

  <p>
    <label for="gender">Geneder </label>
    <input type="radio" value="male" id="gender" name="gender"> Male
    <input type="radio" value="female" id="gender" name="gender"> Female
  </p>

  <p>
    <label for="isMarried">Married </label>
    <input type="checkbox" id="isMarried" name="isMarried">
  </p>


  <p>
    <label for="country">country </label>
    <select id="country" name="country">
      <option value="1">India</option>
      <option value="2">USA</option>
      <option value="3">England</option>
      <option value="4">Singapore</option>
    </select>
  </p>

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

</form>

```

Binding the template to the model

Now we need to associate our model with the above HTML Template. We need to tell angular that we have a model for the form.

This is done using the `formGroup` directive as shown below.

```

<form [formGroup]="contactForm">

```

We use the square bracket (one-way binding) around FormGroup directive and assign our form model (i.e. contactForm name that we gave to our model in the component class) to it.

Next, we need to bind each form field to an instance of the FormControl models. We use the FormControlName directive for this. We add this directive to every form field element in our form. The value is set to the name of the corresponding FormControl instance in the component class.

```

<input type="text" id="firstname" name="firstname" formControlName="firstname">
<input type="text" id="lastname" name="lastname" formControlName="lastname">

```

Submit form

We submit the form data to the component using the Angular directive named ngSubmit. Note that we already have a submit button in our form. The ngSubmit directive binds itself to the click event of the submit button. We are using event binding (parentheses) to bind ngSubmit to OnSubmit method. When the user clicks on the submit button ngSubmit invokes the OnSubmit method on the Component class

```

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

```

We are yet to add the onSubmit() method in the component class. Hence the Angular will throw an error here.

Final Template

Our Final Template is as shown below

```

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

  <p>
    <label for="firstname">First Name </label>
    <input type="text" id="firstname" name="firstname" formControlName="firstname">
  </p>

  <p>
    <label for="lastname">Last Name </label>
    <input type="text" id="lastname" name="lastname" formControlName="lastname">
  </p>

  <p>
    <label for="email">Email </label>
    <input type="text" id="email" name="email" formControlName="email">
  </p>

  <p>
    <label for="gender">Geneder </label>
    <input type="radio" value="male" id="gender" name="gender" formControlName="gen
    <input type="radio" value="female" id="gender" name="gender" formControlName="ge
  </p>

  <p>
    <label for="isMarried">Married </label>
    <input type="checkbox" id="isMarried" name="isMarried" formControlName="isMarried'
  </p>


  <p>
    <label for="country">country </label>
    <select id="country" name="country" formControlName="country">
      <option value="1">India</option>
      <option value="2">USA</option>
      <option value="3">England</option>
      <option value="4">Singapore</option>
    </select>
  </p>


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

</form>

```

Receive the data in the Component class

The last step is to receive the form data in the component class. All we need to do is to create the onSubmit method in our component class.

```

onSubmit() {
  console.log(this.contactForm.value);
}

```

We are using the console.log(this.contactForm.value) to send the value of our form data to the console window.

The final component class is shown below.

```

import { Component } from '@angular/core';
import { FormGroup, FormControl, Validators } from '@angular/forms'
```

```

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'mdf';

  contactForm = new FormGroup({
    firstname: new FormControl(),
    lastname: new FormControl(),
    email: new FormControl(),
    gender: new FormControl(),
    isMarried: new FormControl(),
    country: new FormControl()
  })


  onSubmit() {
    console.log(this.contactForm.value);
  }
}


```

Test the form

Now you can run the app and see the result. Open the developer console and see the value returned by the `contactForm`.value. The values of the form are returned as `JSON` object, which you can pass it your backend API to persist the information to the database.

FormControl

A `FormControl` takes 3 arguments. a default value, a validator, and an asynchronous validator. All of them are optional.

Default Value

You can pass a default value either as a string or as an object of key-value pair. When you pass an object you can set both the value and the whether or not the control is disabled.

```

//Setting Default value as string
firstname= new FormControl('Sachin');

```

```

//Setting Default value & disabled state as object
firstname: new FormControl({value: 'Rahul', disabled: true}),

```

Sync Validator

The second parameter is an array of sync Validators. Angular has some built-in Validators such as `required` and `minLength` etc.

You can pass the Validator function as shown below.

```

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

```

Asynchronous validator

The third argument is the Async Validator. The syntax of Async Validators is similar to Sync Validators.

More on validation in our next tutorial Validations in Reactive forms.

Grouping the controls using FormGroup

We can group various FormControls together. For Example fields such as street, city, country and Pincode each will have their own FormControl but can be grouped together as an address FormGroup

```

contactForm = new FormGroup({
  firstname: new FormControl(),
  lastname: new FormControl(),
  email: new FormControl(),
  gender: new FormControl(),
  isMarried: new FormControl(),
  address:new FormGroup({
    city: new FormControl(),
    street: new FormControl(),
    pincode:new FormControl(),
    country: new FormControl(),
  })
})

```

In the code above, we have created new `FormGroup` Address and added three form controls i.e `city`, `street`, `country` & `Pincode`

In the template use the `formGroupName` directive to enclose the control using a `div` element as shown below

```

```

```
  <div formGroupName="address">

    <div class="form-group">
      <label for="city">City</label>
      <input type="text" class="form-control" name="city" formControlName="city" >
    </div>

    <div class="form-group">
      <label for="street">Street</label>
      <input type="text" class="form-control" name="street" formControlName="street" >
    </div>

    <div class="form-group">
      <label for="pincode">Pin Code</label>
      <input type="text" class="form-control" name="pincode" formControlName="pincod
    </div>

    <p>
      <label for="country">country </label>
      <select id="country" name="country" formControlName="country">
        <option value="1">India</option>
        <option value="2">USA</option>
        <option value="3">England</option>
        <option value="4">Singapore</option>
      </select>
    </p>

  </div>

```

Summary

We learned how to build Angular Reactive Forms in this tutorial. In the next tutorial, we will add validation rules to our application.

1. Angular Forms Tutorial: Fundamental & Concepts
2. Template Driven Forms in Angular
3. Set Value in Template Driven forms in Angular
4. Reactive Forms in Angular
5. FormBuilder in Reactive Forms
6. SetValue & PatchValue in Angular