Angular Reactive Forms Example

Reactive forms ( also known as Model-driven forms) is 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 & Form Arrays. Next, we will create the HTML form template and bind it to the Form Model.

**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 Form Arrays. 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 Form Group, Form Control & Form Arrays
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

|  |  |
| --- | --- |
| 1  2  3 | 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | 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 created the [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/) & [FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/)instances from the template. This happens behind the scene.

[In Reactive](https://www.tektutorialshub.com/angular/angular-formarray-example-in-reactive-forms/) Forms approach, It is our responsibility to build the Model using FormGroup, FormControl and FormArray.

The FormGroup, FormControl & [FormArray](https://www.tektutorialshub.com/angular/angular-formarray-example-in-reactive-forms/) are the three building blocks of the Angular Forms.

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 form groups and form arrays. 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.

|  |  |
| --- | --- |
| 1  2  3 | import { FormGroup, FormControl, Validators } from '@angular/forms' |

**FormGroup**

The FormGroup is created with the following syntax

|  |  |
| --- | --- |
| 1  2  3 | contactForm = new FormGroup({}) |

The FormGroup takes 3 arguments. a collection of a child FormControl, a validator, and an asynchronous validator. The validators are optional.

**FormControl**

The first argument to FormGroup is the collection of FormControl. They are added using the FormControl method as shown below

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | contactForm = new FormGroup({    firstname: new FormControl(),    lastname: new FormControl(),    email: new FormControl(),    gender: new FormControl(),    isMarried: new FormControl(),    country: new FormControl()  }) |

In the above, we have created an instance of a FormGroup and named it as contactForm. contactForm is our top-level FormGroup. Under the contactForm, we have 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.

**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 input (FirstName & LastName), an email field, a radio button (gender), a checkbox (isMarried), and a select list (country). These are Form elements.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | <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 [ngValue]="c.id" \*ngFor="let c of countryList">          {{c.name}}        </option>      </select>    </p>      <p>      <button type="submit">Submit</button>    </p>    </form> |

**Binding the template to the model**

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

This is done using the formGroup directive as shown below.

|  |  |
| --- | --- |
| 1  2  3 | <form [formGroup]="contactForm"> |

We have used the square bracket ([one-way binding](https://www.tektutorialshub.com/angular/angular-data-binding/)) around FormGroup directive and set that equal the model.

Next, we need to bind form fields to 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.

|  |  |
| --- | --- |
| 1  2  3  4 | <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](https://www.tektutorialshub.com/angular/angular-directives/) 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](https://www.tektutorialshub.com/angular/angular-data-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

|  |  |
| --- | --- |
| 1  2  3 | <form [formGroup]="contactForm" (ngSubmit)="onSubmit()"> |

**Final Template**

Our Final Template is as shown below

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | <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="gender"> Male      <input type="radio" value="female" id="gender" name="gender" formControlName="gender"> Female    </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 [ngValue]="c.id" \*ngFor="let c of countryList">          {{c.name}}        </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.

|  |  |
| --- | --- |
| 1  2  3  4  5 | 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.

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

|  |  |
| --- | --- |
| 1  2  3  4 | //Setting Default value as string  firstname= new FormControl(‘Sachin’); |

|  |  |
| --- | --- |
| 1  2  3  4 | //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 with Validator function as shown below.

|  |  |
| --- | --- |
| 1  2  3 | 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.

**Grouping the controls using FormGroup**

We can group various form controls together. For Example fields such as street, city, and Pincode each will have their own form control, but can be grouped together as an address form group

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | contactForm = new FormGroup({    firstname: new FormControl(),    lastname: new FormControl(),    email: new FormControl(),    gender: new FormControl(),    isMarried: new FormControl(),    country: new FormControl(),    address:new FormGroup({      city: new FormControl(),      street: new FormControl(),      pincode:new FormControl()    })  }) |

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

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | <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="pincode">      </div>      </div> |

# Using Angular FormBuilder to build Forms

## What is FormBuilder

The FormBuilder is the helper API to build forms in Angular.  It provides shortcuts to create the instance of the FormControl, [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/) or FormArray. It reduces the code required to write the complex forms.

## How to use FormBuilder

### Import & inject FormBuilder API

To use the FormBuilder, first, we need to import it in our component

|  |  |
| --- | --- |
| 1  2  3 | import { FormBuilder } from '@angular/forms' |

Next, we need to inject it into our component class

|  |  |
| --- | --- |
| 1  2  3  4 | constructor(private formBuilder: FormBuilder) {  } |

Finally, use the group, array & control methods to build the FormModel

### FormGroup

We use the group method to build the Form Group. We pass the list of [Form Controls](https://www.tektutorialshub.com/angular/formcontrol-in-angular/), [Form Array](https://www.tektutorialshub.com/angular/angular-formarray-example-in-reactive-forms/), or another [Form Group](https://www.tektutorialshub.com/angular/formgroup-in-angular/) to the group method as key-value pair. Where the key is the name of the FormControl, FormGroup or FormArray. The value is the configuration of the control.

In the following example, we have added six form controls. The First argument to the FormControl is the initial value, which we set to empty string.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | this.contactForm = this.formBuilder.group({    firstname: [''],    lastname: [''],    email: [''],    gender: [''],    isMarried: [''],    country: [''],  }); |

### Nested FormGroup

Creating a Nested FormGroup is just as easy. use the formbuilder.group method, as shown below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | this.contactForm = this.formBuilder.group({    firstname: [''],    lastname: [''],    email: [''],    gender: [''],    isMarried: [''],    country: [''],    address: this.formBuilder.group({      city: [''],      street: [''],      pincode: ['']    })  }) |

### Validations

The second argument to the FormControl is the list of sync validators. The following example shows how to add validators.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | this.contactForm = this.formBuilder.group({    firstname: ['', [Validators.required, Validators.minLength(10)]],    lastname: ['', [Validators.required, Validators.maxLength(15), Validators.pattern("^[a-zA-Z]+$")]],    email: ['', [Validators.required, Validators.email]],    gender: ['', [Validators.required]],    isMarried: ['', [Validators.required]],    country: ['', [Validators.required]],    address: this.formBuilder.group({      city: ['', [Validators.required]],      street: ['', [Validators.required]],      pincode: ['', [Validators.required]],    })  }); |

## FormBuilder Example

**app.component.ts**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123 | import { Component, ViewChild, ElementRef } from '@angular/core';  import { FormGroup, FormControl, Validators } from '@angular/forms'  import { FormBuilder } from '@angular/forms'  import { groupBy } from 'rxjs/internal/operators/groupBy';      @Component({    selector: 'app-root',    templateUrl: './app.component.html',    styleUrls: ['./app.component.css']  })  export class AppComponent {      title = 'Angular Reactive forms';      contactForm;      constructor(private formBuilder: FormBuilder) {            // this.contactForm = this.formBuilder.group({      //   firstname: [''],      //   lastname: [''],      //   email: [''],      //   gender: [''],      //   isMarried: [''],      //   country: [''],      // });        // this.contactForm = this.formBuilder.group({      //   firstname: [''],      //   lastname: [''],      //   email: [''],      //   gender: [''],      //   isMarried: [''],      //   country: [''],      //   address: this.formBuilder.group({      //     city: [''],      //     street: [''],      //     pincode: ['']      //   })      // });        this.contactForm = this.formBuilder.group({        firstname: ['', [Validators.required, Validators.minLength(10)]],        lastname: ['', [Validators.required, Validators.maxLength(15), Validators.pattern("^[a-zA-Z]+$")]],        email: ['', [Validators.required, Validators.email]],        gender: ['', [Validators.required]],        isMarried: ['', [Validators.required]],        country: ['', [Validators.required]],        address: this.formBuilder.group({          city: ['', [Validators.required]],          street: ['', [Validators.required]],          pincode: ['', [Validators.required]],        })      });    }        get firstname() {      return this.contactForm.get('firstname');    }      get lastname() {      return this.contactForm.get('lastname');    }      get email() {      return this.contactForm.get('email');    }      get gender() {      return this.contactForm.get('gender');    }      get isMarried() {      return this.contactForm.get('isMarried');    }      get country() {      return this.contactForm.get('country');    }      get city() {      return this.contactForm.get("address").get('city');    }      get street() {      return this.contactForm.get("address").get('street');    }      get pincode() {      return this.contactForm.get("address").get('pincode');    }        countryList: country[] = [      new country("1", "India"),      new country('2', 'USA'),      new country('3', 'England')    ];      onSubmit() {      console.log(this.contactForm.value);    }    }      export class country {    id: string;    name: string;      constructor(id: string, name: string) {      this.id = id;      this.name = name;    }  } |

**app.component.html**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147 | <div style="float: left; width:50%;">    <form [formGroup]="contactForm" (ngSubmit)="onSubmit()" novalidate>      <p>      <label for="firstname">First Name </label>      <input type="text" id="firstname" name="firstname" formControlName="firstname">    </p>      <div      \*ngIf="!firstname?.valid && (firstname?.dirty ||firstname?.touched)">      <div [hidden]="!firstname.errors.required">        First Name is required      </div>      <div [hidden]="!firstname.errors.minlength">        Min Length is 10      </div>    </div>      <p>      <label for="lastname">Last Name </label>      <input type="text" id="lastname" name="lastname" formControlName="lastname">    </p>      <div \*ngIf="!lastname.valid && (lastname.dirty ||lastname.touched)">      <div [hidden]="!lastname.errors.pattern">        Only characters are allowed      </div>      <div [hidden]="!lastname.errors.maxLength">        Max length allowed is {{lastname.errors.maxlength?.requiredLength}}      </div>      <div [hidden]="!lastname.errors.required">        Last Name is required      </div>    </div>      <p>      <label for="email">Email </label>      <input type="text" id="email" name="email" formControlName="email">    </p>    <div \*ngIf="!email.valid && (email.dirty ||email.touched)">      <div [hidden]="!email.errors.required">        email is required      </div>      <div [hidden]="!email.errors.email">        invalid email id      </div>    </div>        <p>      <label for="gender">Geneder </label>      <input type="radio" value="male" id="gender" name="gender" formControlName="gender"> Male      <input type="radio" value="female" id="gender" name="gender" formControlName="gender"> Female    </p>    <div \*ngIf="!gender.valid && (gender.dirty ||gender.touched)">      <div [hidden]="!gender.errors.required">        gender is required      </div>    </div>      <p>      <label for="isMarried">Married </label>      <input type="checkbox" id="isMarried" name="isMarried" formControlName="isMarried">    </p>    <div \*ngIf="!isMarried.valid && (isMarried.dirty ||isMarried.touched)">      <div [hidden]="!isMarried.errors.required">        isMarried is required      </div>    </div>        <p>      <label for="country">country </label>      <select id="country" name="country" formControlName="country">        <option [ngValue]="c.id" \*ngFor="let c of countryList">          {{c.name}}        </option>      </select>    </p>    <div \*ngIf="!country.valid && (country.dirty ||country.touched)">      <div [hidden]="!country.errors.required">        country is required      </div>    </div>      <div formGroupName="address">        <div class="form-group">        <label for="city">City</label>        <input type="text" class="form-control" name="city" formControlName="city">      </div>      <div \*ngIf="!city.valid && (city.dirty ||city.touched)">        <div [hidden]="!city.errors.required">          city is required        </div>      </div>          <div class="form-group">        <label for="street">Street</label>        <input type="text" class="form-control" name="street" formControlName="street">      </div>      <div \*ngIf="!street.valid && (street.dirty ||street.touched)">        <div [hidden]="!street.errors.required">          street is required        </div>      </div>        <div class="form-group">        <label for="pincode">Pin Code</label>        <input type="text" class="form-control" name="pincode" formControlName="pincode">      </div>      <div \*ngIf="!pincode.valid && (pincode.dirty ||pincode.touched)">        <div [hidden]="!pincode.errors.required">          pincode is required        </div>      </div>      </div>      <p>      <button type="submit" [disabled]="!contactForm.valid">Submit</button>    </p>    </form>    </div>    <div style="float: right; width:50%;">      <h3>Form Status</h3>    <b>valid : </b>{{contactForm.valid}}    <b>invalid : </b>{{contactForm.invalid}}    <b>touched : </b>{{contactForm.touched}}    <b>untouched : </b>{{contactForm.untouched}}    <b>pristine : </b>{{contactForm.pristine}}    <b>dirty : </b>{{contactForm.dirty}}    <b>disabled : </b>{{contactForm.disabled}}    <b>enabled : </b>{{contactForm.enabled}}      <h3>Form Value</h3>    {{contactForm.value |json}}    </div> |