Using Angular FormBuilder to build Forms

6 Comments / 5 minutes of reading / March 9, 2023

← Angular Reactive Forms | Angular Tutorial | SetValue & PatchValue →

Angular FormBuilder API makes it easier to build reactive forms. We can easily add the FormGroup, nested FormGroup's, FormArray's & FormControls easily. It also allows us to set up the Validation rules for each of the controls. In this tutorial, we will show you how to create a Form, add validation rules using the FormBuilder.

If you are new to Reactive Forms, we recommend you to learn how to build reactive forms.

Table of Contents

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` 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 import { FormBuilder } from '@angular/forms'
3
```

Next, we need to inject it into our component class

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

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, Form Array, or another Form Group 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.

```
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.

```
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.

```
this.contactForm = this.formBuilder.group({
  firstname: ['', [Validators.required, Validators.minLength(10)]],
  lastname: ['', [Validators.required, Validators.maxLength(15), Validators.pattern("^[a-zA-
  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

We learned how to build reactive forms the Angular Reactive forms tutorial.

app.component.ts

```
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) {


```

```
22
23    // this.contactForm = this.formBuilder.group({
24    //   firstname: [''],
25    //   lastname: [''],
26    //   email: [''],
27    //   gender: [''],
28    //   isMarried: [''],
29    //   country: [''],
30    // });
31
32    // this.contactForm = this.formBuilder.group({
33    //   firstname: [''],
34    //   lastname: [''],
35    //   email: [''],
36    //   gender: [''],
37    //   isMarried: [''],
38    //   country: [''],
39    //   address: this.formBuilder.group({
40    //     city: [''],
41    //     street: [''],
42    //     pincode: ['']
43    //   })
44    // });
45
46    this.contactForm = this.formBuilder.group({
47      firstname: ['', [Validators.required, Validators.minLength(10)]],
48      lastname: ['', [Validators.required, Validators.maxLength(15), Validators.pattern("^[a
49      email: ['', [Validators.required, Validators.email]],
50      gender: ['', [Validators.required]],
51      isMarried: ['', [Validators.required]],
52      country: ['', [Validators.required]],
53      address: this.formBuilder.group({
54        city: ['', [Validators.required]],
55        street: ['', [Validators.required]],
56        pincode: ['', [Validators.required]],
57      })
58    });
59  }
60
61
62  get firstname() {
63    return this.contactForm.get('firstname');
64  }
65
66  get lastname() {
67    return this.contactForm.get('lastname');
68  }
69
70  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

```
<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="ge
    <input type="radio" value="female" id="gender" name="gender" formControlName="g
  </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="isMarrie
  </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">
```

```
89
90      <div class="form-group">
91        <label for="city">City</label>
92        <input type="text" class="form-control" name="city" formControlName="city">
93      </div>
94      <div *ngIf="!city.valid && (city.dirty ||city.touched)">
95        <div [hidden]="!city.errors.required">
96          city is required
97        </div>
98      </div>
99
100
101     <div class="form-group">
102       <label for="street">Street</label>
103       <input type="text" class="form-control" name="street" formControlName="street">
104     </div>
105     <div *ngIf="!street.valid && (street.dirty ||street.touched)">
106       <div [hidden]="!street.errors.required">
107         street is required
108       </div>
109     </div>
110
111     <div class="form-group">
112       <label for="pincode">Pin Code</label>
113       <input type="text" class="form-control" name="pincode" formControlName="pincod
114     </div>
115     <div *ngIf="!pincode.valid && (pincode.dirty ||pincode.touched)">
116       <div [hidden]="!pincode.errors.required">
117         pincode is required
118       </div>
119     </div>
120
121   </div>
122
123   <p>
124     <button type="submit" [disabled]="!contactForm.valid">Submit</button>
125   </p>
126
127 </form>
128
129 </div>
130
131 <div style="float: right; width:50%;">
132
133   <h3>Form Status</h3>
134   <b>valid : </b>{{contactForm.valid}}
135   <b>invalid : </b>{{contactForm.invalid}}
136   <b>touched : </b>{{contactForm.touched}}
137   <b>untouched : </b>{{contactForm.untouched}}
```