Angular Reactive Forms Validation

7 Comments / 7 minutes of reading / March 9, 2023

← ViewProviders | Angular Tutorial | Custom Validator in Reactive Forms →

In this article, will explore how angular reactive forms validation works. One of the common tasks that is performed, while building a form is Validation. We will show you how to build Reactive Forms and apply Built-in validators. In the next tutorial, you will learn how to create custom validators in Reactive Forms.

We looked at how the validation works in the Template-driven Forms tutorial. Most of the concepts explained in that tutorial are equally applicable here.

Table of Contents

Validators in Reactive Forms
- What is a Validator
- How to add a Validator to Reactive Forms
- Built-in Validators

Reactive Forms Validation Example
- Model
- Disabling the Browser validation
- Adding in Built-in Validators

Disable Submit button

Displaying the Validation/Error messages
Dirty & touched
Error message
Final Code
Summary

Validators in Reactive Forms

What is a Validator

A Validator is a function that checks the instance of FormControl , FormGroup or a FormArray and returns a list of errors. If the Validator returns a null means that validation has passed

How to add a Validator to Reactive Forms

We configure the validators as the second and third argument to the FormControl , FormGroup or FormArray in the component class. The second argument is a collection of **sync validators** and the third argument is a collection of an **async validators**.

sync validators runs validations and returns immediately. They either return a list of errors or null if no errors found.

async validators: returns a **Promise** or **Observable**. They either return a list of errors or null if no errors are found.

Built-in Validators

The Angular ReactiveForms Module provides several Built-in validators out of the box. They are required , minlength , maxlength & pattern etc.

Reactive Forms Validation Example

W learned how to create Angular Reactive Forms in the previous tutorial. We will now add some of the built-in validators to that example.

Model

Here is the `contactForm` model from the previous tutorial.

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

Disabling the Browser validation

First, we need to disable browser validator by adding the novalidate attribute to the `<form>` element as shown below. If this attribute is present then the form is not

validated by the built-in HTML5 validation when submitted.

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

Adding in Built-in Validators

The mentioned earlier, the Angular has provided several built-in validators out of the box.

Required Validator

The required validator is a sync validator, which returns true only if the `formcontrol` has a non-empty value entered. The second argument of the FormControl takes the *Sync Validator*.

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

Minlength Validator

Minlength validator requires the control value must not have less number of characters than the value specified in the validator.

For Example, `minlength` validator ensures that the `firstname` value has at least 10 characters.

```

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

```

Maxlength Validator

This Validator requires that the number of characters must not exceed the value specified in the validator.

```

lastname: new FormControl('',[Validators.maxLength(15)]),

```

Pattern Validator

This Validator requires that the control value must match the regex pattern provided in the attribute. For example, the pattern `^[a-zA-Z]+$` ensures that the only letters are allowed (even spaces are not allowed). Let us apply this pattern to the lastName

```

lastname: new FormControl('',[Validators.maxLength(15), Validators.pattern("^[a-zA-Z]+$"

```

Email Validator

This Validator requires that the control value must be a valid email address. We apply this to the email field

```

```

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

After adding all the validators, our final `contactForm` will look like this.

```
contactForm = new FormGroup({
  firstname: new FormControl('',[Validators.required,Validators.minLength(10)]),
  lastname: new FormControl('',[Validators.required, Validators.maxLength(15), Validator
  email:new FormControl('',[Validators.email,Validators.required]),
  gender: new FormControl('',[Validators.required]),
  isMarried: new FormControl('',[Validators.required]),
  country: new FormControl('',[Validators.required]),
  address:new FormGroup({
    city: new FormControl('',[Validators.required]),
    street: new FormControl('',[Validators.required]),
    pincode:new FormControl('',[Validators.required])
  })
})
```

Disable Submit button

We have successfully added the validators. Now, we need to disable the submit button if our form is not valid.

The Angular Forms API exposes the state of the forms through the `FormGroup`, `FormControl` & FormArray instances. The `FormGroup` control has a property `valid`, which is set to `true` if all of its child controls are `valid`.

The `contactForm` represents the top-level `FormGroup`. We use it to set the `disabled` attribute of the submit button.

```
<button type="submit" [disabled]="!contactForm.valid">Submit</button>
```

Displaying the Validation/Error messages

We need to provide a short and meaningful error message to the user. We do that by using the error object returned by the `FormControl` instance

Every form element has a `FormControl` instance associated with it. It exposes the state of form element like `valid`, `dirty`, `touched` etc.

There are two ways in which you can get the reference to the `FormControl`.

One way is to use the `contactForm` variable. We can use `contactForm.controls.firstname.valid` to find out if the `firstname` is valid.

```
<div
  *ngIf="!contactForm.controls.firstname?.valid && (contactForm.controls.firstname?.dirty
  ||contactForm.controls.firstname?.touched)">
    First Name is not valid
</div>
```

The other way to is to define getter function for each `FormControl` instance in the component class.

```
get firstname() {
  return this.contactForm.get('firstname');
}
```

and then use it in the template as follows

```
<div *ngIf="!firstname.valid && (firstname.dirty ||firstname.touched)">
   First Name is not valid
</div>
```

Dirty & touched

Apart from checking `valid` we are also checking for the `dirty` & `touched`. Because we do not want the application to display the error when the form is displayed for the first time. We want to display errors only after the user has attempted to change the value. The `dirty` & `touched` properties help us do that.

`dirty`: A control is `dirty` if the user has changed the value in the UI.

`touched`: A control is touched if the user has triggered a `blur` event on it.

Error message

The error message “First Name is not valid ” is not helpful. The firstname has two validators. `required` and `minlength`

Any errors generated by the failing validation is updated in the `errors` object. The `errors` object returns the error object or `null` if there are no errors.

```
<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>
```

Final Code

app.component.ts

```
import { Component, ViewChild, ElementRef } 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 = 'Angular Reactive forms';


  contactForm = new FormGroup({
    firstname: new FormControl('',[Validators.required,Validators.minLength(10)]),
    lastname: new FormControl('',[Validators.required, Validators.maxLength(15), Validato
    email:new FormControl('',[Validators.email,Validators.required]),
    gender: new FormControl('',[Validators.required]),
```

```
    isMarried: new FormControl('',[Validators.required]),
    country: new FormControl('',[Validators.required]),
    address:new FormGroup({
      city: new FormControl('',[Validators.required]),
      street: new FormControl('',[Validators.required]),
      pincode:new FormControl('',[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 contact {
  firstname:string;
  lastname:string;
  gender:string;
  isMarried:boolean;
  country:string;
  address: {
    city:string;
    street:string;
    pincode:string;
  }
}


export class country {
  id: string;
  name: string;

  constructor(id: string, name: string) {
    this.id = id;
    this.name = name;
  }
}
```

app.component.html

```

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

```
3
4   <p>
5     <label for="firstname">First Name </label>
6     <input type="text" id="firstname" name="firstname" formControlName="firstname">
7   </p>
8
9   <div
10    *ngIf="!firstname?.valid && (firstname?.dirty ||firstname?.touched)">
11    <div [hidden]="!firstname.errors.required">
12      First Name is required
13    </div>
14    <div [hidden]="!firstname.errors.minlength">
15      Min Length is 10
16    </div>
17  </div>
18
19  <p>
20    <label for="lastname">Last Name </label>
21    <input type="text" id="lastname" name="lastname" formControlName="lastname">
22  </p>
23
24  <div *ngIf="!lastname.valid && (lastname.dirty ||lastname.touched)">
25    <div [hidden]="!lastname.errors.pattern">
26      Only characters are allowed
27    </div>
28    <div [hidden]="!lastname.errors.maxLength">
29      Max length allowed is {{lastname.errors.maxlength?.requiredLength}}
30    </div>
31    <div [hidden]="!lastname.errors.required">
32      Last Name is required
33    </div>
34  </div>
35
36  <p>
37    <label for="email">Email </label>
38    <input type="text" id="email" name="email" formControlName="email">
39  </p>
40  <div *ngIf="!email.valid && (email.dirty ||email.touched)">
41    <div [hidden]="!email.errors.required">
42      email is required
43    </div>
44    <div [hidden]="!email.errors.email">
45      invalid email id
46    </div>
47  </div>
48
49
50  <p>
51    <label for="gender">Geneder </label>
```

```
52    <input type="radio" value="male" id="gender" name="gender" formControlName="ge
53    <input type="radio" value="female" id="gender" name="gender" formControlName="g
54  </p>
55  <div *ngIf="!gender.valid && (gender.dirty ||gender.touched)">
56    <div [hidden]="!gender.errors.required">
57      gender is required
58    </div>
59  </div>
60
61  <p>
62    <label for="isMarried">Married </label>
63    <input type="checkbox" id="isMarried" name="isMarried" formControlName="isMarrie
64  </p>
65  <div *ngIf="!isMarried.valid && (isMarried.dirty ||isMarried.touched)">
66    <div [hidden]="!isMarried.errors.required">
67      isMarried is required
68    </div>
69  </div>
70
71
72  <p>
73    <label for="country">country </label>
74    <select id="country" name="country" formControlName="country">
75      <option [ngValue]="c.id" *ngFor="let c of countryList">
76        {{c.name}}
77      </option>
78    </select>
79  </p>
80  <div *ngIf="!country.valid && (country.dirty ||country.touched)">
81    <div [hidden]="!country.errors.required">
82      country is required
83    </div>
84  </div>
85
86
87
88  <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
```

```
    <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="pincod
    </div>
    <div *ngIf="!pincode.valid && (pincode.dirty ||pincode.touched)">
      <div [hidden]="!pincode.errors.required">
        pincode is required
      </div>
    </div>

  </div>

  <p>{{contactForm.valid}} </p>

  <p>
    <button type="submit" [disabled]="!contactForm.valid">Submit</button>
  </p>


</form>




<!-- <div ngModelGroup="address">

  <p>
    <label for="city">City</label>
    <input type="text" id="city" name="city">
  </p>

  <p>
    <label for="street">Street</label>
    <input type="text" id="street" name="street">
  </p>
  <p>
    <label for="pincode">Pin Code</label>
    <input type="text" id="pincode" name="pincode">
  </p>
```