Custom Validator in Angular Reactive Form

5 Comments / 6 minutes of reading / May 18, 2023

← Validation in Reactive Forms

Angular Tutorial

Passing Parameter to Custom Validator →

Learn how to build a custom validator in Angular Reactive form. A data entry form can contain a large no of fields. The Angular forms module makes it easier to create, manage, and validate the form fields. There are two ways in which you can create forms in Angular. One is Reactive forms & the other one is template-driven forms. Learn how to create reactive forms & how to create template-driven forms.

Table of Contents

Built-in Validators

Validating the Forms is very important, otherwise, we will end up having invalid data in our database. The Angular Forms Module provides a few built-in validators to help us to validate the form. They are listed below.

1. Required validator
2. Min length Validator
3. Max length Validator
4. Pattern Validator
5. Email Validator

We covered them in the validation in reactive forms & validation in template-driven forms tutorial.

Custom Validator in Angular Reactive Form

Built-in validators are useful but do not cover all use cases. This is where we use the custom validator. It is very easy to create a custom validator in Angular.

How to Build Custom Validator

Building a custom Validator is as easy as creating a `Validator` function. It is a function, which must implement ValidatorFn Interface.

ValidatorFn

The `ValidatorFn` is an Interface, which defines the signature of the Validator function.

```
1
2 interface ValidatorFn {
3   (control: AbstractControl): ValidationErrors | null
```

```
}

```

The function takes the AbstractControl. This is the base class for FormControl , FormGroup , and FormArray . The validator function must return a list of errors i.e ValidationErrors or null if the validation has passed

Custom Validator Example

Create a new angular application. Add the following code in app.component.html

```
<h1>Custom Validator in Angular</h1>

<h2>Reactive Form</h2>

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

  <div>
    <label for="numVal">Number :</label>
    <input type="text" id="numVal" name="numVal" formControlName="numVal">
  </div>

  <p>Is Form Valid : {{myForm.valid}} </p>

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

</form>

```

Source Code

Our example app has numVal form field. We want it to be greater than 10.

Angular does not have any built-in validator for that. Hence let us build a custom Validator gte

Create a new file gte.validator.ts under the app folder.

```

import { AbstractControl, ValidationErrors } from '@angular/forms'

export function gte(control: AbstractControl): ValidationErrors | null {

    const v=+control.value;

    if (isNaN(v)) {
      return { 'gte': true, 'requiredValue': 10 }
    }

    if (v <= 10) {
      return { 'gte': true, 'requiredValue': 10 }
    }

    return null

}

```

Source Code

First, import the AbstractControl and ValidationErrors from the @angular/forms

```

import { AbstractControl, ValidationErrors } from '@angular/forms'

```

The validator function must adhere to the ValidatorFn Interface. It should receive the AbstractControl as its parameter. It can be FormControl, FormGroup or FormArray.

The function must validate the control value and return ValidationErrors if any errors are found otherwise null.

```
1
2 export function gte(control: AbstractControl): ValidationErrors | null {
3
```

The ValidationErrors is a key-value pair object of type [key: string]: any and it defines the broken rule. The key is the string and should contain the name of the broken rule. The value can be anything, but usually set to true.

The validation logic is very simple. Check if the value of the control is a number using the isNaN method. Also, check if the value is less than or equal to 10. If both the rules are valid and then return null

```
1
2   const v=+control.value;
3
4   if (isNaN(v)) {
5     return { 'gte': true, 'requiredValue': 10 }
6   }
7
8   if (v <= 10) {
9     return { 'gte': true, 'requiredValue': 10 }
10  }
11
12  return null
13
```

If the validation fails then return the ValidationErrors. You can use anything for the key, but it is advisable to use the name of the validator i.e gte as the key. Also, assign true as value. You can as well assign a string value.

```

return { 'gte': true, 'requiredValue': 10 }

```

You can return more than one `key-value` pair as shown in the above example. The second key `requiredValue` returns the value 10. We use this in the template to show the error message.

Using the Custom Validator

To use this validator first, import it in the component class.

```

import { gte } from './gte.validator';

```

Add the validator to the Validator collection of the `FormControl` as shown below.

```

  myForm = new FormGroup({
    numVal: new FormControl('', [gte]),
  })


```

The complete `app.component.ts` is shown below.

```

import { Component } from '@angular/core';
import { FormGroup, FormControl, AbstractControl, ValidationErrors } from '@angular/form
import { gte } from './gte.validator';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
```

```
export class AppComponent {

  constructor() {
  }

  myForm = new FormGroup({
    numVal: new FormControl('', [gte]),
  })

  get numVal() {
    return this.myForm.get('numVal');
  }

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

```

Source Code

Accessing the Errors from Custom Validator

We need to provide a meaningful error message to the user.

Validators return ValidationErrors . They are added to the control's errors collection of the control. The valid property of the control is set to false .

Hence we check if the valid property. We also check the dirty and touched property. Because we do not want to display the error message when the form is displayed for

the first time.

```
<div>
  <label for="numVal">Number :</label>
  <input type="text" id="numVal" name="numVal" formControlName="numVal">
  <div *ngIf="!numVal.valid && (numVal.dirty ||numVal.touched)">
    <div *ngIf="numVal.errors.gte">
      The number should be greater than {{numVal.errors.requiredValue}}
    </div>
  </div>

</div>
```

We check if the `gte` is true and display the error message. Note that `gte` is the name of the `key` we used while creating the validator.

We also make use of the `requiredValue` to show a meaningful message to the user.

```
<div *ngIf="numVal.errors.gte">
  The number should be greater than {{numVal.errors.requiredValue}}
</div>
```

Now, we have successfully built a custom validator in reactive forms.

Next Steps

Our Custom validator is simple, but it can be improved.

Instead of the hardcoded value of 10, we want it set it while defining the validator. i.e we want to send the parameter to the validator function.