How to add Validators Dynamically using SetValidators in Angular

7 Comments / 3 minutes of reading / March 9, 2023

← Cross Field Validation Angular Tutorial Angular HTTP Tutorial →

We can add Validators dynamically using the SetValidators or SetAsyncValidators. This method is available to FormControl, FormGroup & FormArray.

There are many use cases where it is required to add/remove validators dynamically to a FormControl or FormGroup. Usually when you have a Form Field, whose value depends on another Form Field.

Table of Contents

Adding the Validators Using the SetValidators

Syntax

The setValidators programmatically adds the sync validators. This method will remove all the previously added sync or async validators.

```
setValidators(newValidator: ValidatorFn | ValidatorFn[]): void
```

Examples:

```

this.myform.controls["mobile"].setValidators(Validators.required);

```

```

this.myform.controls["mobile"].setValidators([Validators.required,Validators.minLength(10)]

```

setAsyncValidators

The `setAsyncValidators` programmatically add the Async validators.

```

setAsyncValidators(newValidator: AsyncValidatorFn | AsyncValidatorFn[]): void

```

`setValidators` overwrites all existing Validators. Hence it is very important to include all the validators that we want in the `setValidators` method

Removing Validators Using clearValidators

There is no option that exists, which can remove an individual validator. Use `clearValidators` to remove all the validators of a control.

```

this.myForm.controls['controlName'].clearValidators()

```

Update Validation Status

Removing or adding the validators does not change the validity status of the form or the control immediately. The Validators run only when we change the value of the field.

We can force angular to run the validations using the `updateValueAndValidity` method.

```

this.myForm.controls['controlName'].updateValueAndValidity()

```

SetValidators Example

The following example, shows how to use the `SetValidators` in Angular

We have two fields `email` & `mobile`.

The user needs to choose, how he wants the system to notify him, using the drop-down field `notifyVia`. The drop-down has two options `email` & `Mobile`.

If the user chooses email, then we need to make the email field as a `Required` field. If he chooses the Mobile, then we must make the mobile field as `Required` field.

We subscribe to the valueChanges event of the `notifyVia` to listen for changes and invoke the `changeValidators` method.

In the changeValidators method, we check the value of notifyVia and add or remove the required validator using the setValidators . We also add the email validator (for email field) or MinLength validator (for mobile field). To remove the validator, we use the method clearValidators()

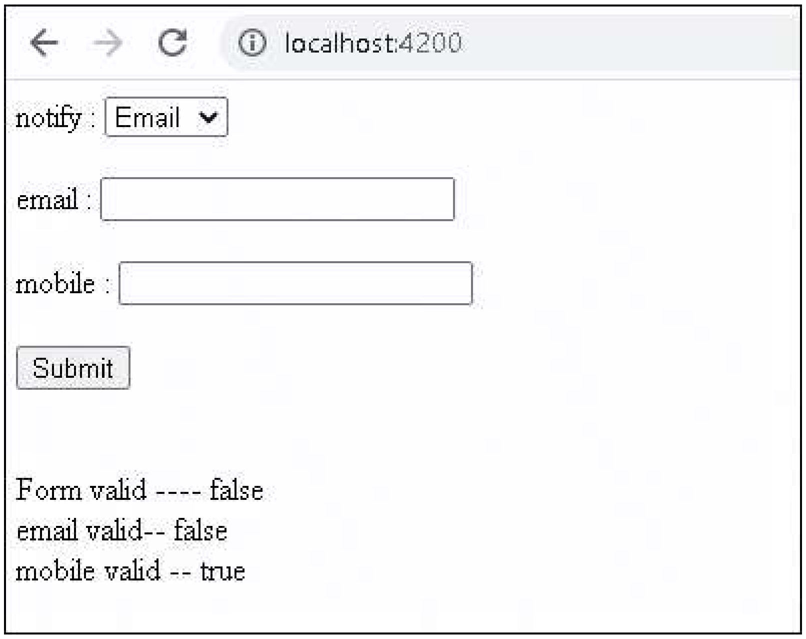
Finally, we use the updateValueAndValidity method, which forces the angular to update the validity status of the control.

```

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

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

  myform:FormGroup;

  notifyOptions = ["Email" ,"SMS"]

  constructor(private fb: FormBuilder) {

    this.myform = this.fb.group({
      email: new FormControl(''),
      mobile: new FormControl(''),
      notifyVia: new FormControl('',Validators.required),
    });

    this.myform.get("notifyVia").valueChanges
      .subscribe(data=> {
        this.changeValidators()
      })
  }


  changeValidators() {

    console.log(this.myform.get("notifyVia").value)

    if (this.myform.get("notifyVia").value=="Email") {
      this.myform.controls["email"].setValidators([Validators.required,Validators.email]);
      this.myform.controls["mobile"].clearValidators();
    } else {
      this.myform.controls["email"].clearValidators();
      this.myform.controls["mobile"].setValidators([Validators.required,Validators.minLength
    }

    this.myform.get("email").updateValueAndValidity();
    this.myform.get("mobile").updateValueAndValidity();


  }
}


```

The component template

```

<form [formGroup]="myform">

  notify :
<select formControlName="notifyVia">
  <option *ngFor="let item of notifyOptions" [ngValue]="item">{{item}}</option>
</select>
<br>
<br>


email :
<input type="text" formControlName= "email"/>
<br>
<br>
mobile :
<input type="text" formControlName= "mobile"/>
<br>
<br>

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

<br>
<br>

Form valid ---- {{myform.valid}}  <br>

```

```
email valid-- {{myform.controls['email'].valid}} <br>

mobile valid -- {{myform.controls['mobile'].valid}} <br>

```



```
Form valid ---- false
email valid-- false
mobile valid -- true
```

References

1. SetValidators
2. SetAsyncValidators
3. ClearValidators
4. UpdateValueAndValidity

Read More

1. **Angular Reactive Forms Validation**
2. **Custom Validator in Angular Reactive Form**
3. **Custom Validator with Parameters in Angular**
4. **Inject Service Into Validator in Angular**
5. **Template-driven form validation in Angular**
6. **Custom Validator in Template Driven Forms in Angular**
7. **Angular Async Validator Example**