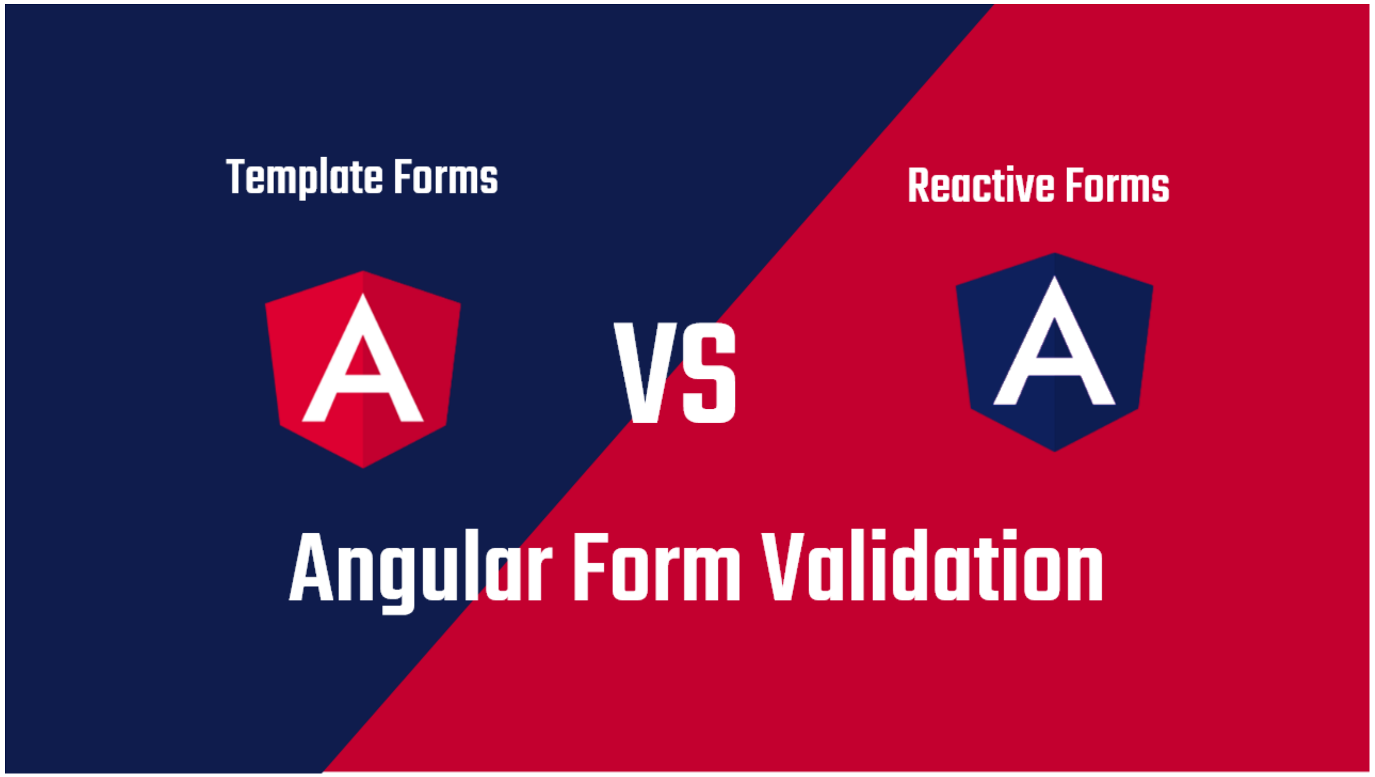


Angular Form Validations

Aurora Solutions Follow
Dec 3, 2018 · 3 min read

Every web-based application that requires data from the user often requires forms to enable user to provide data.. Before we go ahead further, we need to understand about what forms are and the different types of forms supported by Angular.

Angular supports two kinds of forms

Template Driven Forms

As the name suggests, template driven forms are created by writing templates and binding directives and behavior to the templates.

Model Driven (Reactive) Forms

Reactive forms on the other hand utilize underlying APIs instead of directives.

. . .

What Is Form Validation?

Form validation is the process in which data entered by user is checked against a set of rules to ensure that it is correct and complete. In the next subsection, we will learn how to proceed with validations in template driven and reactive forms.

Validation In Template Driven Form

In template driven forms, we use html validation attributes like required. Minlength, maxlength etc. Angular recognizes these keywords and runs validation every time the form changes (as a result of user input) and generates a list of validation errors, if any, which results in INVALID status. If there are no validation errors, a null list is generated with VALID status.

Usually, ngModel is exported to a local variable so that control properties like **touched**, **untouched**, **valid**, **invalid**, **pristine**, **dirty**, etc that comes as a part of AbstractControl API can be examined.

- `touched` The field has been touched
- `untouched` The field has not been touched yet

- `valid` The field content is valid
- `invalid` The field content is not valid
- `pristine` The field has not been modified yet
- `dirty` The field has been modified

. . .

Let us consider the following example:

```html
<form name="form" (ngSubmit)="onSubmit()">
   <div class="form-group">
      <label for="email">Email</label>
      <input type="text" class="form-control" name="email" [(ngModel)]="model.email" #em
      <div *ngIf="email.invalid" class="invalid-feedback">
         <div *ngIf="email.errors.required">Email cannot be left blank</div>
         <div *ngIf="email.errors.email">Please enter a valid email address</div>
      </div>
   </div>
</form>
```

angular-template-validation.html hosted with ❤ by **GitHub** **view raw**

This is one field inside a user input form where we specify the email address. Note that we have used two attributes — required and email over here to specify that the field is required and needs to conform to the standard pattern expected for email addresses. The above div can be part of a form which binds ngSubmit to a function (say onSubmit()).

. . .

Reactive Form Validation

In reactive form validation, we add validations to control model in the component class instead of through templates. To enable this, we import Validators class in our component using:

```
Import {Validators} from '@angular/forms';
```

It is possible to write our own custom validation functions or use built in validator functions to perform reactive form validation.

Let us consider an example in which we have a simple form where the user is required to enter email address and password to register to the site.

Sample form code:

```
<form [formGroup]="myForm" (ngSubmit)="onSubmit()">
<div class="form-group">
   <label>Email</label>
   <input type="text" formControlName="email" class="form-control" [ngClass]="{ 'is-inva
   <div *ngIf="submitted && formControls.email.errors" class="invalid-feedback">
      <div *ngIf="formControls.email.errors.required">Email cannot be left blank</div>
      <div *ngIf="formControls.email.errors.email">Please enter a valid email address</d
   </div>
</div>
<div class="form-group">
   <label>Password</label>
   <input type="password" formControlName="password" class="form-control" [ngClass]="{ '
   <div *ngIf="submitted && formControls.password.errors" class="invalid-feedback">
      <div *ngIf="formControls.password.errors.required">Password cannot be left blank</
      <div *ngIf="formControls.password.errors.minlength">Password must be at least 6 ch
      <div *ngIf="formControls.password.errors.maxlength">Password must not be more than
   </div>
</div>
<div class="form-group">
   <button class="btn btn-primary">Register</button>
</div>
</form>
```

angular-reactive-validation.html hosted with ❤ by **GitHub** **view raw**

Note that we have used **formControls** everywhere to get access to controls for each member of the form. This is an easy hack written at component level (a small get function) so that much of the coding is reduced.

Component class would look like the following:

```
export class AppComponent implements OnInit {
  myForm: FormGroup;
  submitted = false;
  constructor(private formBuilder: FormBuilder) {}
  ngOnInit() {
    this.myForm = this.formBuilder.group({
      email: ['', [Validators.required, Validators.email]],
      password: [
        '',
        [Validators.required, Validators.minLength(6), Validators.maxlength(20)]
      ]
    });
  }
  get formControls() {
    return this.myForm.controls;
  }
  onSubmit() {
    this.submitted = true;
    if (this.myForm.invalid) {
      return;
    }
    alert(
      'New user successfully registered \n' + JSON.stringify(this.myForm.value)
    );
  }
}
```

angular-component-class.ts hosted with ❤ by GitHub view raw

Built-in Vs Custom Validations

In the previous example, we use built-in validators like **Required**, **MinLength**, **MaxLength** etc. There are cases in which developers would want to use a custom validator when the validation logic required is a bit deviated from the common validators provided in Validator class.

Let us consider the case in which we have a new field called phone number. The numbers shall be separated using two hyphens like this: 161–202–1111 . In this case, we would need a custom validator.

The valid regex would be: **-?(\d{3})(\-\d{3})(\-\d{4})**

Custom validator function would look like the following:

```typescript
import { AbstractControl } from '@angular/forms';
// setup simple regex to check for phone number format.
const phoneNumRegex = /[^-?(\d{3})(\-\d{3})(\-\d{4})]/;
export function validatePhoneNumber(control: AbstractControl) {
  if (control.value.match(phoneNumRegex)) {
    return { validNum: true };
  }
  return null;
}
```

angular-custom-validator.ts hosted with ❤ by **GitHub** **view raw**

Now, this function can be used in the reactive form in following manner:

```typescript
  ngOnInit() {
     this.myForm = this.formBuilder.group({
       email: ['', [Validators.required, Validators.email]],
       password: [
         '',
         [Validators.required, Validators.minLength(6), Validators.maxlength(20)]
       ],
       phoneNumber: ['', [Validators.required, validatePhoneNumber]]
     });
   }
```

angular-custom-validator-usage.ts hosted with ❤ by **GitHub** **view raw**

. . .

Conclusion

In this tutorial, we learnt about form validations. Angular supports two kinds of forms — template driven and reactive forms. We saw how validations can be performed in these forms using built in and custom validations.

Angular | Front End Development | Web Development

14 claps

WRITTEN BY

Aurora Solutions

Follow

https://aurorasolutions.io/

Aurora Solutions

Follow

Reliable Web, Mobile and Devsecops development

Write the first response

More From Medium

More from Aurora Solutions

Running Automated Tests on AWS Devicefarm in Custom Environment

Kashif Ali in Aurora Solutions
Jan 23, 2019 · 5 min read

65

More from Aurora Solutions

Smart Application Logging For Production Grade Systems

Aurora Solutions in Aurora Solutions
Apr 10, 2019 · 5 min read ★

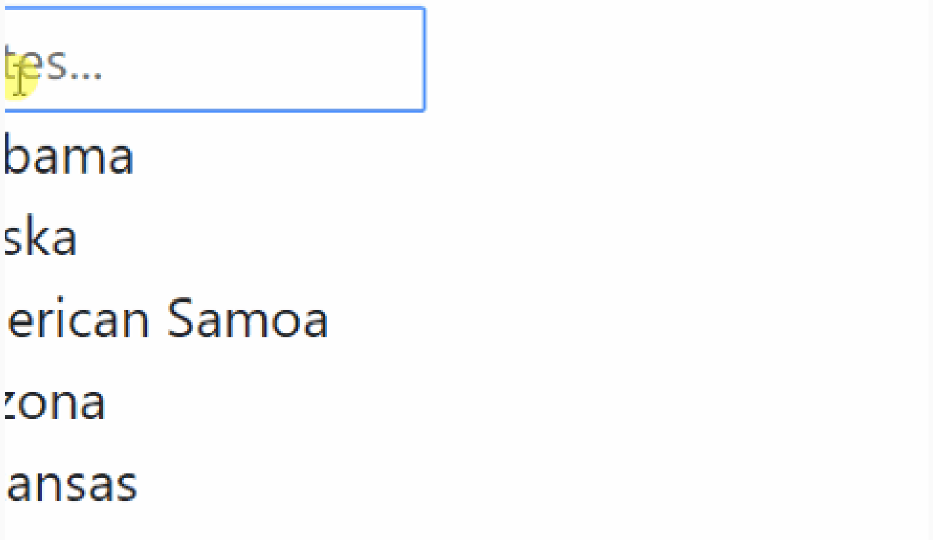
24

Related reads

Dynamic filtering with RxJs and Angular forms — A tutorial

Alain Chautard in Angular Training
Feb 18, 2019 · 4 min read ★

754


tes...
bama
ska
erican Samoa
zona
ansas

ifornia