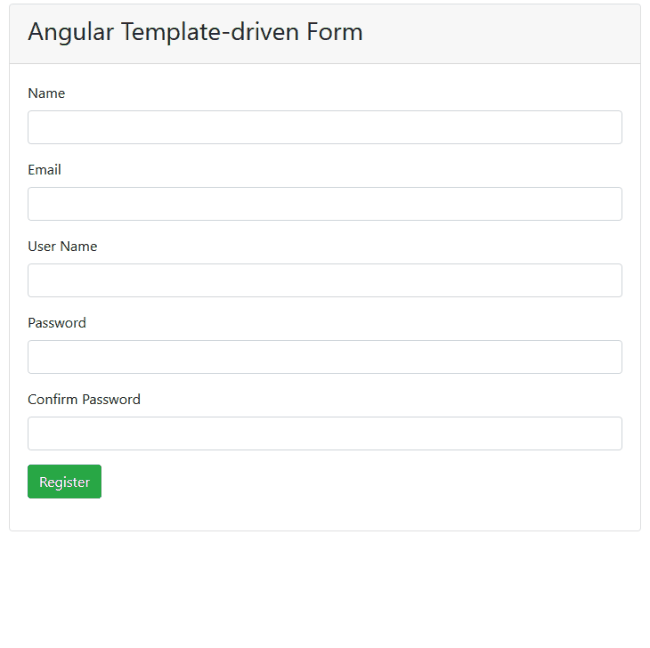
We will create a simple user registration form and implement some inbuilt validations on it. Along with the inbuilt validations, we will also implement some custom validations for the template-driven form.

We will consider the following custom validations for this demo:

* Checking for user name availability
* Password pattern validation
* Matching the password entered in two different fields

Take a look at the application in action.



You can get the source code from [GitHub](https://github.com/AnkitSharma-007/angular-forms-validation).

**Create the Angular app**

Navigate to the folder where you want to create your project file. Open a command window and run the command shown below:

ng new angular-forms-validation --routing=false --style=scss

We are specifying the command to create a new Angular application. The option to create the routing module is set to false and style files extension is set to SCSS. This command will create the Angular project with the name angular-forms-validation.

Change directories to the new project and open the project in VS Code using the set of commands below:

cd angular-forms-validation

code .

**Install Bootstrap**

Run the following command to install Bootstrap:

npm install bootstrap --save

Add the following import definition in the styles.scss file:

@import "~bootstrap/dist/css/bootstrap.css";

**Create the validation service**

Run the following command to create a new service:

ng g s services\customvalidation

This command will create a folder named services that has two files inside it – customvalidation.service.ts and customvalidation.service.spec.ts. Open customvalidation.service.ts and put the following code inside it:

import { Injectable } from '@angular/core';

import { ValidatorFn, AbstractControl } from '@angular/forms';

import { FormGroup } from '@angular/forms';

@Injectable({

providedIn: 'root'

})

export class CustomvalidationService {

patternValidator(): ValidatorFn {

return (control: AbstractControl): { [key: string]: any } => {

if (!control.value) {

return null;

}

const regex = new RegExp('^(?=.\*?[A-Z])(?=.\*?[a-z])(?=.\*?[0-9]).{8,}$');

const valid = regex.test(control.value);

return valid ? null : { invalidPassword: true };

};

}

MatchPassword(password: string, confirmPassword: string) {

return (formGroup: FormGroup) => {

const passwordControl = formGroup.controls[password];

const confirmPasswordControl = formGroup.controls[confirmPassword];

if (!passwordControl || !confirmPasswordControl) {

return null;

}

if (confirmPasswordControl.errors && !confirmPasswordControl.errors.passwordMismatch) {

return null;

}

if (passwordControl.value !== confirmPasswordControl.value) {

confirmPasswordControl.setErrors({ passwordMismatch: true });

} else {

confirmPasswordControl.setErrors(null);

}

}

}

userNameValidator(userControl: AbstractControl) {

return new Promise(resolve => {

setTimeout(() => {

if (this.validateUserName(userControl.value)) {

resolve({ userNameNotAvailable: true });

} else {

resolve(null);

}

}, 1000);

});

}

validateUserName(userName: string) {

const UserList = ['ankit', 'admin', 'user', 'superuser'];

return (UserList.indexOf(userName) > -1);

}

}

The method patternValidator is used to validate the password pattern in our form. The parameter for this method is of type AbstractControl which is a base class for the FormControl.

We will use a regular expression to validate the password. This regular expression will check for the following four conditions in the password:

* The password should be a minimum of eight characters long
* It should have at least one lower case letter
* It should have at least one upper case letter
* It should have at least one number

If the password fails the regex check, we will set the invalidPassword property to true.

The method MatchPassword is used to compare the passwords in two fields. This method will accept two parameters of type string. These parameters represent the name of the fields to be matched. We will get the FormControl for these two fields and then match the values in them. If the values do not match, we will set the passwordMismatch property to true.

The method userNameValidator is used to verify if the username is already taken or not. This method will accept a parameter of type AbstractControl.

We will check if the value of this field is present in a static array, UserList. If the value entered by the user is already present, we will set the userNameNotAvailable property to true.

We are using the setTimeout function to invoke this check every two seconds. This will ensure that the error will be thrown after two seconds from the time the user stops typing in the field.

For the sake of simplicity of this article, we are using a static array to search for the availability of user names. Ideally, it should be a service call to the server to search for the value in a database.

**Create the User model**

Create a new folder called models inside the src/app folder. Add a new file called user.ts inside the models folder. Put the following code in the user.ts file.

export class User {

public name: string;

public email: string;

public username: string;

public password: string;

public confirmPassword: string;

}

**Create custom directives**

We will create custom directives to implement custom validators for template-driven forms.

Run the command shown below to create the passwordPattern directive:

ng g d directives\passwordPattern

This command will create a folder named directives that has two files inside it – passwordPattern.directive.ts and passwordPattern.directive.spec.ts. Open passwordPattern.directive.ts and put the following code inside it.

import { Directive } from '@angular/core';

import { NG\_VALIDATORS, Validator, AbstractControl } from '@angular/forms';

import { CustomvalidationService } from '../services/customvalidation.service';

@Directive({

selector: '[appPasswordPattern]',

providers: [{ provide: NG\_VALIDATORS, useExisting: PasswordPatternDirective, multi: true }]

})

export class PasswordPatternDirective implements Validator {

constructor(private customValidator: CustomvalidationService) { }

validate(control: AbstractControl): { [key: string]: any } | null {

return this.customValidator.patternValidator()(control);

}

}

This directive is used to validate the password pattern. We will implement the Validator interface on the class PasswordPatternDirective. We will override the validate method which accepts a parameter of type AbstractControl, that is the control we want to validate. We will then invoke the patternValidator method from the service.

Run the command shown below to create the matchPassword directive:

ng g d directives\matchPassword

Open matchPassword.directive.ts and put the following code inside it:

import { Directive, Input } from '@angular/core';

import { NG\_VALIDATORS, Validator, ValidationErrors, FormGroup } from '@angular/forms';

import { CustomvalidationService } from '../services/customvalidation.service';

@Directive({

selector: '[appMatchPassword]',

providers: [{ provide: NG\_VALIDATORS, useExisting: MatchPasswordDirective, multi: true }]

})

export class MatchPasswordDirective implements Validator {

@Input('appMatchPassword') MatchPassword: string[] = [];

constructor(private customValidator: CustomvalidationService) { }

validate(formGroup: FormGroup): ValidationErrors {

return this.customValidator.MatchPassword(this.MatchPassword[0], this.MatchPassword[1])(formGroup);

}

}

This directive is used to validate if the passwords entered in two fields are matching or not. This directive will accept an input of the type string array, which contains the fields to match. We will override the validate method and pass the parameter of type FormGroup. We will then invoke the MatchPassword method from the service.

Run the command shown below to create the validateUserName directive:

ng g d directives\validateUserName

Open validateUserName.directive.ts and put the following code inside it:

import { Directive, forwardRef } from '@angular/core';

import { Validator, AbstractControl, NG\_ASYNC\_VALIDATORS } from '@angular/forms';

import { CustomvalidationService } from '../services/customvalidation.service';

import { Observable } from 'rxjs';

@Directive({

selector: '[appValidateUserName]',

providers: [{ provide: NG\_ASYNC\_VALIDATORS, useExisting: forwardRef(() => ValidateUserNameDirective), multi: true }]

})

export class ValidateUserNameDirective implements Validator {

constructor(private customValidator: CustomvalidationService) { }

validate(control: AbstractControl): Promise<{ [key: string]: any }> | Observable<{ [key: string]: any }> {

return this.customValidator.userNameValidator(control);

}

}

This directive is used to validate the availability of the user name. We will override the validate method and pass the parameter of type AbstractControl. We will then invoke the userNameValidator method from the service. This method will return a promise.

**Create the template-driven form component**

Run the command shown below to create the template-driven-form component:

ng g c template-driven-form

Open template-driven-form.component.ts and put the following code in it:

import { Component } from '@angular/core';

import { User } from '../models/user';

@Component({

selector: 'app-template-driven-form',

templateUrl: './template-driven-form.component.html',

styleUrls: ['./template-driven-form.component.scss']

})

export class TemplateDrivenFormComponent {

userModal = new User();

constructor() { }

onSubmit() {

alert('Form Submitted succesfully!!!\n Check the values in browser console.');

console.table(this.userModal);

}

}

We have created an object userModal of type User. We will bind the form fields with the property of this object. The onSubmit method will show the success message on the screen and print the contents of the form to the console.

Open template-driven-form.component.html and put the following code in it:

<div class="container">

<div class="row">

<div class="col-md-8 mx-auto">

<div class="card">

<div class="card-header">

<h3>Angular Template-driven Form</h3>

</div>

<div class="card-body">

<form class="form" #registerForm="ngForm" [appMatchPassword]="['password', 'confirmPassword']"

(ngSubmit)="registerForm.form.valid && onSubmit()" novalidate>

<div class=" form-group">

<label>Name</label>

<input type="text" class="form-control" [(ngModel)]="userModal.name" name="name"

#name="ngModel" required>

<span class="text-danger"

\*ngIf="(name.touched || registerForm.submitted) && name.errors?.required">

Name is required

</span>

</div>

<div class="form-group">

<label>Email</label>

<input type="text" class="form-control" [(ngModel)]="userModal.email" name="email"

#email="ngModel" required email>

<span class="text-danger"

\*ngIf="(email.touched || registerForm.submitted) && email.errors?.required">

Email is required

</span>

<span class="text-danger" \*ngIf="email.touched && email.errors?.email">

Enter a valid email address

</span>

</div>

<div class="form-group">

<label>User Name</label>

<input type="text" class="form-control" [(ngModel)]="userModal.username" name="username"

#username="ngModel" appValidateUserName required>

<span class="text-danger"

\*ngIf="(username.touched || registerForm.submitted) && username.errors?.required">

User Name is required

</span>

<span class="text-danger" \*ngIf="username.touched && username.errors?.userNameNotAvailable">

User Name not available

</span>

</div>

<div class="form-group">

<label>Password</label>

<input type="password" class="form-control" [(ngModel)]="userModal.password" name="password"

#password="ngModel" appPasswordPattern required>

<span class="text-danger"

\*ngIf="(password.touched || registerForm.submitted) && password.errors?.required">

Password is required

</span>

<span class="text-danger" \*ngIf="password.touched && password.errors?.invalidPassword">

Password should have minimum 8 characters, at least 1 uppercase letter, 1 lowercase

letter and 1 number

</span>

</div>

<div class="form-group">

<label>Confirm Password</label>

<input type="password" class="form-control" [(ngModel)]="userModal.confirmPassword"

name="confirmPassword" #confirmPassword="ngModel" required>

<span class="text-danger"

\*ngIf="(confirmPassword.touched || registerForm.submitted) && confirmPassword.errors?.required">

Confirm Password is required

</span>

<span class="text-danger"

\*ngIf="confirmPassword.touched && confirmPassword.errors?.passwordMismatch">

Passwords doesnot match

</span>

</div>

<div class="form-group">

<button type="submit" class="btn btn-success">Register</button>

</div>

</form>

</div>

</div>

</div>

</div>

</div>

We will create a template-driven form and use the Bootstrap card for styling. The card header will contain a title whereas the card body will have the form fields.

We will use the appMatchPassword directive on our form and pass the password and confirmPassword fields for validation. The ngModel property is used to bind the form control to the model.

For validating the user name availability we will use the appValidateUserName directive on the username field. Similarly, we will use the appPasswordPattern directive on the password field to validate the password pattern.

We will check for the errors in the form controls and then display the appropriate validation error message on the screen.

**Create the nav-bar component**

Run the command shown below to create the nav-bar component:

ng g c nav-bar

Open nav-bar.component.html and put the following code in it:

<nav class="navbar navbar-expand-sm navbar-dark bg-dark fixed-top">

<a class="navbar-brand" [routerLink]='["/"]'>Form Validation Demo</a>

<div class="collapse navbar-collapse">

<ul class="navbar-nav mr-auto">

<li class="nav-item">

<a class="nav-link" [routerLink]='["/template-form"]'>Template Form</a>

</li>

</ul>

</div>

</nav>

Here we are adding the navigation link to the template-driven form component.

**Update the app component**

Open the app.component.html file and put the following code in it:

<app-nav-bar></app-nav-bar>

<div class="container">

<router-outlet></router-outlet>

</div>

**Update the App module**

We will import the forms module and also set up the routing for our application in the app module. Add the following code in the app.module.ts file. You can refer to [GitHub](https://github.com/AnkitSharma-007/angular-forms-validation/blob/master/src/app/app.module.ts) for the complete source code of this file:

import { RouterModule } from '@angular/router';

import { FormsModule } from '@angular/forms';

@NgModule({

...

imports: [

...

FormsModule,

RouterModule.forRoot([

{ path: '', component: TemplateDrivenFormComponent },

{ path: 'template-form', component: TemplateDrivenFormComponent }

]),

],

})

**Execution demo**

Use the following command to start the webserver:

ng serve -o

This command will launch the application in your default browser at http://localhost:4200/. You can perform all the form validations which we have discussed here.

This application is also hosted at <https://ng-forms-validation.herokuapp.com/>. Navigate to the link and play around with it for a better understanding.

**Summary**

We have created a sample user registration form using the template-driven form approach in Angular. We have implemented the inbuilt validations as well as custom validations to the form. The Bootstrap library is used to style the form.

Get the source code from [GitHub](https://github.com/AnkitSharma-007/angular-forms-validation) and play around with it for a better understanding.