esprit
Se former autrement
HONORIS UNITED UNIVERSITIES

Création et validation des formulaires

Année universitaire

2023-2024

United Nations Educational, Scientific and Cultural Organization
UNESCO Chair "Project-based learning" ESPRIT School of engineering, Tunisia
EUR-ACE® Délivrée par la Commission des Titres d'Ingénieur
CONFÉRENCE DES GRANDES ÉCOLES
cdio CONCEIVE DESIGN IMPLEMENT OPERATE

- Introduction
- Reactive Form

Introduction

Introduction

- Les formulaires sont presque toujours présents sur tous les sites Web et applications.
- Ils peuvent être utilisés pour effectuer: l'authentification, la création d'un profil, l'envoie d'une demande, contact, etc.
- Un Formulaire, avant sa soumission, doit respecter un ensemble de règles afin de garantir l'intégralité des données soumises: Il doit être valide
- La validation d'un formulaire se fait moyennant des validateurs.

Les approches de création de formulaire

Pour la création des formulaires, Angular propose deux approches:

1. **Template Driven Forms :** Le formulaire ainsi que les validateurs sont créés directement au niveau du template.
2. **Reactive Forms** (ou **Model Driven Forms**): Le formulaire ainsi que les validateurs sont créés dans la classe du composant et ensuite liés au template grâce au DataBinding.

```
import { ReactiveFormsModule } from '@angular/forms';
import { FormsModule } from '@angular/forms';
@NgModule({
….
imports: [ // other imports ...,
ReactiveFormsModule,
FormsModule ],
….})
export class AppModule { }
```

Reactive Forms

Template Driven Forms

Reactive Forms

Reactive Form - Introduction

- En appliquant l'approche de Reactive Forms, le formulaire ainsi que ses éléments sont créés au niveau de la classe du composant et affichés par la suite au niveau du template
- La création du formulaire et ses différents éléments se fait grâce aux classes **FormControl**, **FormGroup**.
- Avec cette approche, le composant est capable d'observer le changement d'état de chaque élément et d'y réagir=> les mises à jour de valeur et de validité sont toujours synchrones et sous le contrôle.

=> C'est pour qu'on parle du mode synchrone.

Reactive Form - FormControl

class FormControl extends AbstractControl

- La classe FormControl permet de générer un élément de contrôle (input, select, …) au niveau de la classe du composant et lui affecter des validateurs si besoin.
- Elle permet également de suivre l'état de chaque élément de contrôle.

```
username = new FormControl('Toto',
    [Validators.required,Validators.minLength(3)]);
```

Toto: Valeur par défaut

```
username = new FormControl({value:'Toto',disabled:true},
    [Validators.required,Validators.minLength(3)]);
```

Toto: Valeur par défaut
Diasbled: etat par défaut

Reactive Form - FormControl

- Création du formControl dans fichier html en dehors d'un formulaire

```
<input type="text" [formControl]="username">
```

- Création du formControl dans fichier html dans un formulaire

```
<form>
<input type="text" formControlName="username">
</form>
```

Reactive Form - FormGroup

class FormGroup extends AbstractControl

- Permet de créer une collection de **FormControl**.
- Cette classe suit la valeur et l'état de validité de chaque instance **FormControl** qu'elle contient.
- Si un FormControl du FormGroup est invalide alors le FormGroup en entier est invalide.

```
this.user_data = new FormGroup({
username: new FormControl('toto',Validators.required),
city: new FormControl('Ariana', Validators.required)
});
```

Reactive Form - FormGroup

- On peut imbriquer plusieurs FormGroup.
- Le **FormGroup racine** correspond à la totalité du formulaire: <form>
- Le **FormGroup** imbriqué est un sous ensemble d'éléments dans le formulaire.

```
this.user_data = new FormGroup({
username: new FormControl('toto',Validators.required),
city: new FormControl('Ariana') ,
autres : new FormGroup({
    email : new FormControl (''),
    fb: new FormControl ('')     })});
```

FormGroup racine ⇔ <form>

FormGroup imbriqué ⇔ un sous ensemble

Reactive Form - FormGroup

- L'instance crée de la **classe FormGroup racine** est liée à l'élément HTML correspondant grâce à la directive **FormGroupDirective: [formGroup]**

```
<form [formGroup]="user_data"></form>
```

- L'instance crée de classe **FormGroup imbriquée** est liée à un élément HTML avec la directive **FormGroupName : formGroupName**

```
<form [formGroup]="user_data">
<div formGroupName="autres">
……
</div>
</form>
```

Reactive Form - FormGroup

- Chaque FormControl du FormGroup (racine ou imbriqué) est lié à l'élément HTML correspondant grâce à la directive **FormControlName.**

```
<form [formGroup]="user_data">
<input type="text" formControlName="username" />
<input type="text" formControlName="city" />
<div formGroupName="autres">
Email: <input formControlName="email"> <br>
FB: <input formControlName="fb">
</div>
</form>
```

Reactive Form - FormGroup

La classe FormGroup contient un ensemble de méthode qui nous permette de manipuler les éléments (FormControls) d'un FormGroup comme:

- registerControl(): enregistrer un élément dans un FormGroup
- addControl(): ajouter un élément
- removeControl(): supprimer un élément
- setControl(): modifier un élément par un autre
- contains() : vérifier si un élément existe
- setValue(): modifier la valeur d'un élément
- reset(): vider les valeurs dans
- getRawValue() : Récupère toutes les valeurs **quel que soit l'état** (même si disabled)
- value: Récupère les valeurs des éléments actifs (non disabled)

Reactive Form – Classes et directives

```
import { Validators, FormBuilder, FormGroup, FormControl , FormArray }  from '@angular/forms';
```

Classe	Directive	Rôle de la directive
FormControl	FormControlDirective	Synchronise une instance d'un Form control autonome avec un élément html
	FormControlName	Synchronise un FormControl dans un FormGroup avec un élément html
FormGroup	FormGroupDirective	Lie un FormGroup avec un élément du DOM
	FormGroupName	Synchronise un FormGroup imbriqué avec un élément du DOM

Reactive Form - FormBuilder

- **FormBuilder**: est une classe (helper) qui crée pour nous des instances **FormGroup**, **FormControl** et **FormArray**. En gros, cela réduit la répétition et l'encombrement en gérant pour vous les détails de la création du contrôle de formulaire.

```
Userform=this.formBuilder.group({
 username: ['', Validators.required],
 email: ['', Validators.compose([ Validators.required, Validators.pattern('^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+.[a-zA-Z0-9-.]+$') ],
 address: this. formBuilder.group({
        street: '',
        city: ''}) });
```

FormGroup racine

FormGroup fils

Reactive Form - Validation

- Les validateurs sont définis lors de la création d'un FormControl dans la classe du composant.
- Il faut importer **Validators** depuis **@angular/forms**

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

- Angular appelle ces validateurs pour chaque changement de la valeur du FormControl
  - Il y a deux types de validateurs: les validateurs synchrones et les validateurs asynchrones

Reactive Form - Validation

Les classes **FormControl** et **FormGroup** héritent de la même classe **AbstractControl**. Quelques propriétés sont dans le tableau suivant

Propriété	Rôle
Valid/invalid	Si valid = true alors invalid=false => détermine si un élément est valid ou non en fonction des validateurs déclarés dessus
errors	Si null => pas d'erreur Si non null (objet) => indique les erreurs commise
Pristine /dirty	Si pristine = true alors dirty = false => l'élément sa valeur initial n'a pas changé et vis versa.
Touched/untouched	Si touched = true alors untouched = false => l'élément a été visité
Status = VALID ou INVALID	détermine si un élément est valid ou non en fonction des validateurs déclarés dessus

Reactive Form - Validation

Nous pouvons accéder à n'importe quel FormControl à partir du FormGroup parent

Dans le TS:

FormGroup (propriété)

FormControl

```
get name() { return this.userForm.get('name'); }
```

```
get name() { return this.userForm.controls.name; }
```

Dans le HTML:

```
<div *ngIf="userForm.get('name').errors.required"> Name is required. </div>
```

Reactive Form - Validation

1. Définition des validateurs synchrones

```
this.userForm = new FormGroup({
'name': new FormControl(this.model.name, [ Validators.required,
Validators.minLength(4), forbiddenNameValidator(/bob/i) ])})
get name(){return this.userForm.get('name'); }
```

On peut créer des validateurs personnalisés tel que ici forbiddenNameValidator()

2. Détéction des erreurs et affichage des messages

```
<div *ngIf="name?.errors?.['required']"> Name is required. </div>
<div *ngIf="name?.errors?.['minlength']"> 4 characters long. </div>
<div *ngIf="userForm.get('name')?.errors?.['forbiddenName']"> Name cannot
be Bob. </div>
```

Réferences

- [1] : https://angular.io/api/forms/FormControl
- [2] : https://angular.io/api/forms/AbstractControlDirective

Merci de votre attention