esprit
Se former autrement
ONORIS UNITED UNIVERSITIES

Création et validation de formulaires ave l'approche Template driven

United Nations Educational, Scientific and Cultural Organization
uniTwin
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
- Création de formulaires
- Validation de formulaires

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.
- Par exemple si un champ est obligatoire, alors il faut attacher le validateur « required » à ce champ.

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 crées directement au niveau du template.
2. **Reactive Forms** (ou **Model Driven Forms**): Le formulaire ainsi que les validateurs sont crées 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

Les approches de création de formulaire

- La différence majeure entre les deux approches est que les **Reactive forms** sont synchrones tandis que les **Template-driven forms** sont asynchrones

Template Driven Form

Template driven Form - FormControl

- Chaque élément du formulaire possède une valeur, peut avoir plusieurs contraintes et peut passer par plusieurs états.
- Un élément du formulaire est une instance d'une classe prédéfinie appelée **FormControl** qui suit l'état d'un élément du formulaire,
- La classe **FormControl** étend la classe **AbstractControl** qui implémente la plupart des fonctionnalités de base pour accéder à la valeur, au statut de validation, aux interactions utilisateur et aux événements d'un élément du formulaire. [1]

Template driven Form - FormControl

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

Template driven Form - NgModel

- La directive **NgModel** assure le two-way DataBinding. En l'utilisant dans un formulaire, cette directive permet, **de plus**, de suivre l'état de chaque FormControl auquel elle est associée en lui ajoutant des classes CSS Angular

état	Si état est true	Si état est false
Control visité	ng-touched	ng-untouched
Valeur initiale modifiée	ng-dirty	ng-pristine
Valeur valide	ng-valid	ng-invalid

Template driven Form - NgModel

- Pour accéder aux propriétés d'un FormControl moyennant NgModel, il faut exporter la directive NgModel dans une variable locale #variable="ngModel"

Pas d'attribut name

```
<input type="text" [(ngModel)]="model.cin“  #cin="ngModel">
```

```
<form>
<input type="text" [(ngModel)]="model.cin" name="cin" #cin="ngModel">
</form>
```

Dans un formulaire, il faut ajouter l'attribut name pour pouvoir utiliser la directive NgModel sinon c'est une erreur

Template driven Form - NgModelGroup

- La directive **NgModelGroup** regroupe un ensemble d'éléments et surveille les propriétés sur les éléments auquels la directive ngModel est attaché.
- La directive NgModelGroup hérite de la classe **AbstractFormGroupDirective**. [3]

```
<form>
<div ngModelGroup="autres" #sousgroupe="ngModelGroup" >
……..
</div>
</form>
```

Template driven Form - NgForm

- La directive **NgForm** surveille les propriétés de tout le formulaire et/ou d'un ensemble d'éléments du formulaire.
- Tout élément du formulaire **attaché à la directive NgModel**, peut être suivi par la directive NgForm.
- Tout groupe d'éléments **attaché à la directive NgModelGroup**, peut être suivi par la directive NgForm
- Pour accéder aux propriétés du formulaire, il faut alors exporter la directive NgForm dans une variable référence de template

```
<form #formulaire="ngForm">
….
<div ngForm #f="ngForm">
…….
</div>
                </form>
```

Template driven Form - NgForm

- La directive NgForm hérite de la classe **AbstractControlDirective**. [2]
- Les propriétés retournées par NgForm sont les mêmes que FormControl mais qui réfère sur un ensemble d'éléments d'un formulaire.
- La directive NgForm, attaché **implicitement** à la balise form, suit la totalité du formulaire.

```
<form #formulaire="ngForm">
……..
</form>
```

Template driven Form - NgForm

- Les propriétés retournées par NgForm sont les propriétés du groupe d'éléments auquel elle est attachée. Plus précisément les éléments du groupe dont la directive NgModel est attaché. Par exemple:

Propriété	Rôle
Valid/invalid	Si valid = true alors invalid=false => détermine si un formulaire est valide ou non en fonction des validateurs déclarés sur les différents éléments. Un formulaire est valide si tout les éléments du formulaires sont valides.
errors	Si null => pas d'erreur Si non null (objet) => indique les erreurs commises dans les différents éléments du formulaire

Template driven Form - Validation

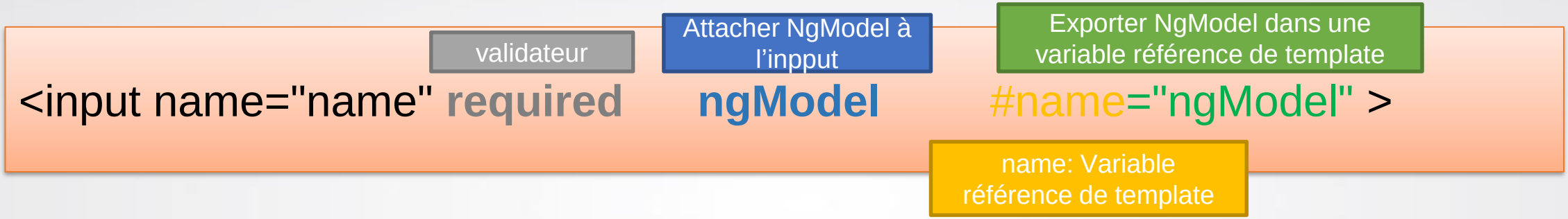
Plusieurs contraintes peuvent être attachés à un élément dans le formulaire:

1. Les attributs HTML de validation : required, minlength, maxlength, min, max, pattern, …
2. Des validateurs personnalisés

- Un élément est valide si la contrainte attachée est respectée.
- Les propriétés **valid**, **invalid** et **errors** de FormControl permettent de détecter si un élément est valide ou non.

Template driven Form – Validation - NgModel

- Exemple1: Un seul validateur sur un élément




```
<div *ngIf="name.valid">* Name is required. </div>
```

Template driven Form – Validation - NgModel

- Exemple 2: Plusieurs validateurs sur un élément

```
<input name="name" required minlength="4"
appForbiddenName="bob" [(ngModel)]="model.name"
#name="ngModel" >
```

```
<div *ngIf="name.errors['required']"> Name is required. </div>
<div *ngIf="name.errors['minlength']"> Name must be at least 4
characters long. </div>
<div *ngIf="name.errors['forbiddenName']"> Name cannot be
Bob. </div>
```

Template driven Form – Validation - NgForm

- Exemple 3:

Pas la peine d'attacher ngForm à la balise form, elle est attaché implictement, mais il faut l'exporter dans une variable référence de template pour accéder aux propriétés par la suite

```
<form #form="ngForm">
<input name="name" required [(ngModel)]="model.name" #name="ngModel" />
<input name="email" pattern="xxxxxx" [(ngModel)]="model.email" #email="ngModel" />
<input name= "tel" pattern="xxxxxx" />
<button [disabled]="form.invalid"> Add </button>
</form>
```

Le formulaire est valide si name est rempli et email est écrit selon le pattern

Le champ tel n'est pas suivi par NgForm car il n'est pas attaché à NgModel, du coup son état n'est tenu en compte

Néthographie

- [1] : https://angular.io/api/forms/FormControl
- [2] : https://angular.io/api/forms/AbstractControlDirective
- [3]: Angular – AbstractFormGroupDirective

Merci de votre attention