Formulaires

Animé par Mazen Gharbi

Template Driven Forms

▷ Angular permet de gérer les formulaires de plusieurs façons allant de la plus simple à la plus extensible.

https://stackblitz.com/edit/angular-basic-form

▷ Angular applique implicitement la directive `NgForm` aux éléments `form`.
▷ Avec cette approche, la logique de validation est dans le template au lieu d'être dans le composant.
▷ L'approche est également très peu extensible

Template Driven

app.component.html

```
<form #userForm="ngForm" (submit)="onSubmit()" novalidate>
    <input
            [(ngModel)]="userTmp.firstName"
           name="firstName"
           required
           type="text">
```

app.component.ts

```
export class AppComponent {
    userTmp: User = new User({});
    user: User = new User({});

    onSubmit() {
        this.user = new User(this.userTmp);
    }
}
```

Model Driven Forms

- Angular utilise 3 classes pour gérer les formulaires :
  - Form : gère un champ du formulaire (son état, sa validité, etc...)
  - FormGroup : gère un groupes de champs. Il permet de savoir par exemple si tous les champs du groupe sont valides
  - FormArray : permet de gérer une liste dynamique de `Control`

https://stackblitz.com/edit/angular-model-form

  - Importer « ReactiveFormsModule » !

Model driven

app.component.ts

```
this.firstNameControl = new FormControl('Foo', Validators.compose([Validators.required,
Validators.minLength(3)]));

this.lastNameControl = new FormControl(null, Validators.required);
this.emailControl = new FormControl(null, Validators.pattern(this._emailRegex));
this.wishListControlArray = new FormArray([]);

this.userFormGroup = new FormGroup({
    firstName: this.firstNameControl,
    lastName: this.lastNameControl,
    email: this.emailControl,
    wishList: this.wishListControlArray
});
```

Model driven

app.component.html

```
<form [formGroup]="userFormGroup" (ngSubmit)="onSubmit()" novalidate>
    <input formControlName="firstName" type="text"><br/>
    <input formControlName="lastName" type="text"><br/>
    <input formControlName="email" type="email"><br/>
```

Model Driven Forms

▷ La logique de validation est dans le composant plutôt que dans la vue.
▷ Il est donc possible de factoriser les configurations de "control group" et les réutiliser dans d'autres formulaires

▷ Chaque "control" dispose des propriétés suivantes :
- dirty : indique si le champ a été modifié par l'utilisateur
- pristine : contraire de dirty
- touched : indique si le champ a été touché par l'utilisateur
- valid : indique la validité du champ

Model Driven Forms

▷ Angular applique des classes CSS sur les "inputs" en fonction de leur état afin de pouvoir personnaliser l'affichage en fonction des erreurs.

- ng-pristine, ng-dirty, ng-touched, ng-untouched, ng-valid, ng-invalid

▷ Pour initialiser le formulaire avec les données d'un modèle, on peut utiliser l'attribut d'"input" `ngModel`

```
<input [ngModel]="user.firstName" ...>
```

La classe `FormControl` fournit une propriété `valueChanges` qui est un "observable" permettant de détecter les changements du champ.

```
this.firstNameControl.valueChanges.subscribe((value) => {
    ...
});
```

Custom Validators

https://stackblitz.com/edit/angular-form-custom-validators

Custom Validators

```
class UsersValidator {
    static email () {
        // Je retourne un objet si on a une erreur
        return {
            email: {
                reason: 'Erreur random',
                invalid: true
            }
        };
    }
}
```

app.component.ts

```
this.firstNameControl = new FormControl('Foo',
            Validators.compose([Validators.required, Validators.minLength(3)]));
this.lastNameControl = new FormControl(null, Validators.required);
this.emailControl = new FormControl(null, UsersValidator.email);
this.wishListControlArray = new FormArray([]);
```

Validateurs asynchrones

https://stackblitz.com/edit/angular-form-custom-async-validator

Validateurs asynchrones

```
static emailAsync(control: FormControl): IValidation {
    return new Promise<IValidation>((resolve, reject) => {
        setTimeout(() => {
            if (!UserValidators._EMAIL_REGEX.test(control.value)) {
                /* Error object. */
                resolve({
                    email: {
                        invalid: true,
                        reason: 'I don\'t know'
                    }
                });
            }
            else {
                /* Success. */
                resolve(null);
            }
        }, 1000);
    });
}
```

On resolve soit null soit un Object.
Mais on ne « reject »jamais !

Questions