Angular
Gérer les formulaires

Sur une application web nous avons besoin de gérer les entrées des utilisateurs.

Angular offre une série de class qui permet de gérer à la fois la saisie, le pré-remplissage, la validation des champs et la validation du formulaire.

Introduction

- Template driven form ou ***reactive form*** ?
- Les class de formulaires

1 Mettre en place un **reactiveForm**

2 Les class **FormControl** et **FormGroup**

3 Gérer la validation : la class **Validators**

3.1 propriétés et méthodes pour gérer la validation

4 Afficher les erreurs dans le HTML

INTRO - TDFORM OU REACTIVE FORM

	REACTIVE	TEMPLATE-DRIVEN
Setup of form model	Explicit, created in component class	Implicit, created by directives
Data model	Structured and immutable	Unstructured and mutable
Predictability	Synchronous	Asynchronous
Form validation	Functions	Directives

https://angular.io/guide/forms-overview#key-differences

CHOISIR UNE APPROCHE (Intro suite)

Les **reactive forms** ou les **formulaires gérés depuis le template** traitent les données des formulaires différemment. Chaque approche offre des avantages différents.

Les reactive forms fournissent un accès direct et explicite au modèle d'objet de formulaires sous-jacent.

Ils sont plus évolutifs, réutilisables et testables. Si les formulaires sont un élément clé de votre demande, utilisez des formulaires réactifs.

Les formulaires gérés depuis le template s'appuient sur les directives HTML pour créer et manipuler le modèle objet.

Ils sont utiles pour ajouter un formulaire simple à une application, comme un formulaire d'inscription.

https://angular.io/guide/forms-overview#key-differences

LES CLASS DES FORMULAIRES

Les reactive forms et les template-driven forms sont construits sur les class suivantes.

- FormControl suit la valeur et la validation d'un control de formulaire (champ de formulaire)
- FormGroup suit la valeur et la validation d'une collection de form controls
- FormArray suit la valeur et la validation d'un array de form controls
- ControlValueAccessor crée un pont entre des instances Angular Form Control et les éléments de formulaire du DOM

https://angular.io/guide/forms-overview#common-form-foundation-classes

1 METTRE EN PLACE UN REACTIVE FORM (3 étapes)

1

app.module.ts

```
imports: [
        …,
        ReactiveFormsModule
]
```

Déclarer **ReactiveFormsModule** dans l'application.

Cela ajoutera à nos formulaires, les directives formGroup et formControlName permettant contrôler un formulaire depuis son modèle que nous déclarons dans la class du component

2

app.component

```
<form [formGroup]=”myForm”
        ngSubmit=”onSubmit()”>
    <input formControlName=’firstname’ />
    <input formControlName=’email’ />
    <button>Envoyer</button>
<form>
```

3

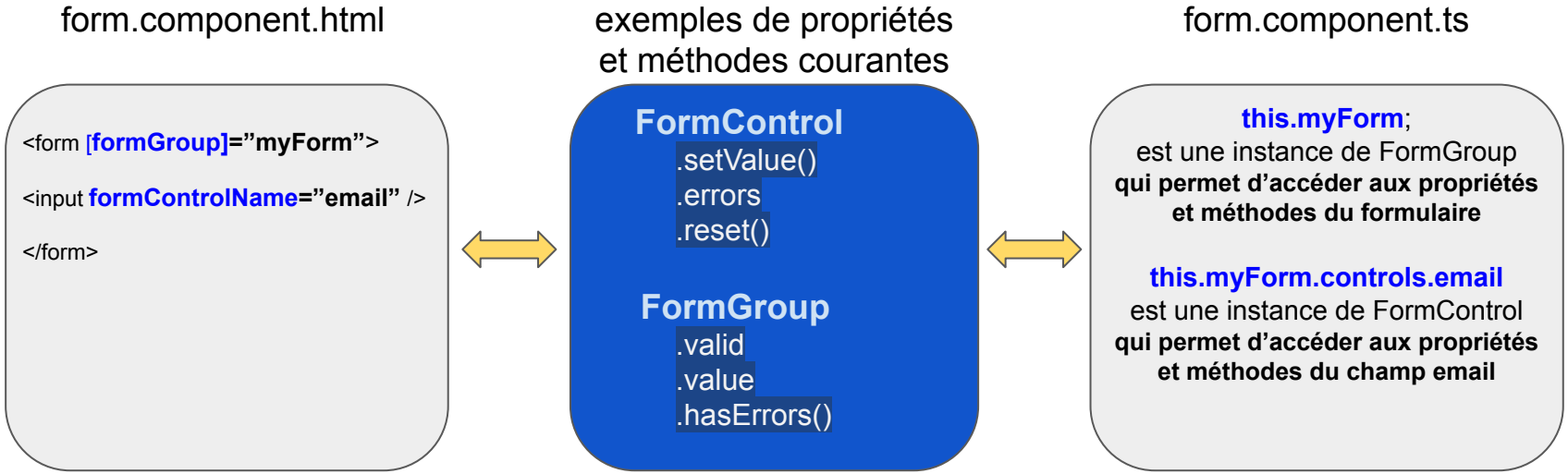
```
constructor(private fb:FormBuilder) {}

myForm = this.fb.group({
        firstname: ['', Validators.required],
        email: ['', Validators.email],
})
```

NB : la methode .group() de FormBuilder est juste un sucre syntaxique pour créer des instances de FormGroup

https://angular.io/guide/reactive-forms#using-the-formbuilder-service-to-generate-controls

2 LES CLASS FormGroup et FormControl



Les class **FormGroup** et **FormControl**
permettent à Angular de lier un formulaire HTML à une instance
et au développeur de le contrôler via les propriétés et méthodes offertes par ces 2 class

https://angular.io/api/forms/FormControl
https://angular.io/api/forms/FormGroup

3 GÉRER LA VALIDATION DES CHAMPS AVEC Validators

app.component.html

```
<form [formGroup]=”myForm”
        ngSubmit=”onSubmit()”>
    <input formControlName=’firstname’ />
    <input formControlName=’email’ />
    <button>Envoyer</button>
<form>
```

app.component.ts

```
constructor(private fb:FormBuilder) {}

myForm = this.fb.group({
        firstname: ['', Validators.required],
        email: ['', Validators.email],
})
```

La Class Validators offre des règles de validation utilisables directement
.required, .minLength(), .maxLength(), .email, .min(), .max()

Vous pouvez spécifier vos propres règles de validation sur un champ avec .pattern() en spécifiant une expression régulière en paramètre

https://angular.io/api/forms/Validators

3.1 FormGroup ET FormControl : ***exemples*** de propriétés et méthodes

Le formulaire *(class FormGroup)*

```
this.myForm.value; // valeur de l'objet myForm

this.myForm.controls; // object-collection champs

this.myForm.valid; // true ou false

this.myForm.hasErrors('email'); // true or false

this.myForm.touched; // true or false

this.myForm.dirty; // true or false
```

Les champs *(class FormControl)*

```
name = new FormControl('Bill', Validators.required);

name.value;  // 'Bill'
name.status;  // valid

name.setValue('Steve');
// set la valeur du champ name

name.email.reset();
// efface la valeur du champ email
```

Les propriétés et méthodes de **FormControl** et **FormGroup**
permettent au développeur de gérer la validation des champs et du formulaire

exemple: `if( this.myForm.valid ) { // le formulaire ne comporte pas d'erreurs }`

4 AFFICHER LES ERREURS DANS LA VUE

app.component.html

```
...
        <input formControlName='firstname' />
        <p *ngIf="myForm.controls.firstname.errors?.required">
                Prénom requis
        </p>

        <input formControlName='email' />
        <p *ngIf="myForm.controls.email.errors?.email">
                Saisissez un email valide
        </p>
```

app.component.ts

```
constructor(private fb:FormBuilder) {}

myForm = this.fb.group({
        firstname: ['', Validators.required],
        email: ['', Validators.email],
})
```

Pour afficher les erreurs dans la vue HTML il suffit d'utiliser les conditions apportées par les propriétés **errors**, **valid**, **touched**, **dirty**, **ou d'autres méthodes de FormControl** si besoin.

https://angular.io/guide/form-validation#built-in-validator-functions

REACTIVE FORM / A RETENIR

LES CLASS

- ➔ **FormControl** permet de créer une instance de champ de formulaire
- ➔ **FormGroup** permet de créer une instance de modèle du formulaire
- ➔ **FormBuilder** permet de créer une instance de FormGroup **plus facilement**
- ➔ **Validators** permet d'ajouter une ou plusieur règles de validation sur un FormControl

Une instance de FormGroup est liée à la vue grâce à la directive <form [formGroup]=””></form>
Une instance de FormControl est liée à la vue grâce à la directive <input formControlName=””>