PARTIE 7: LES FORMULAIRES ET ÉLÉMENTS DE FORMULAIRE

TROIS TYPES D'ÉCRITURE

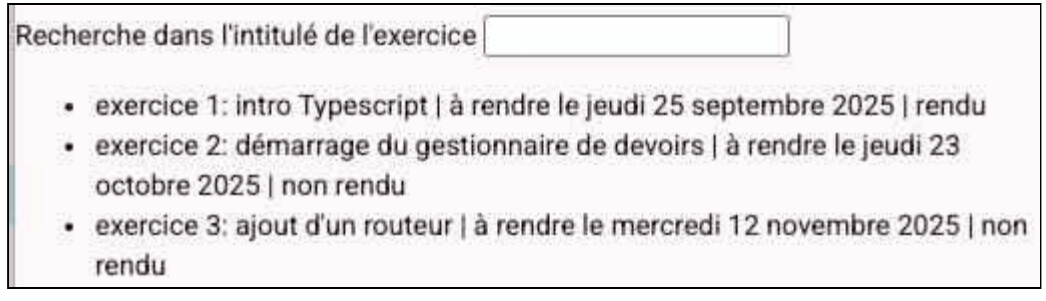
- les formulaires pilotés par le *template* pour des formulaires simples. Après import de **`FormsModule`**, les directives `ngForm` et `ngModel` peuvent être utilisées.
- les formulaires pilotés par le code plus adaptés dès que le formulaire devient plus complexe (validations, champs dynamiques). Le module **`ReactiveFormsModule`** sera utilisé.
- Depuis Angular 22, les formulaires basés sur les signaux sont disponibles. Intéressant.

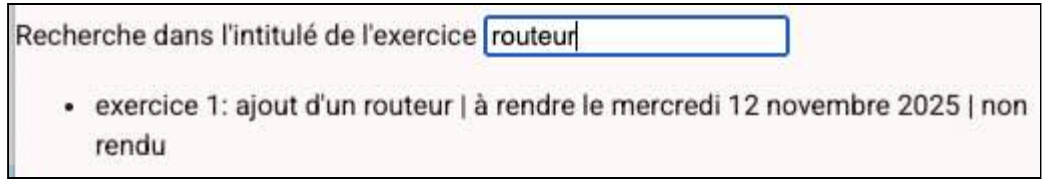
Pour ce cours, nous avons fait le choix des `reactive forms` = pilotés par le code.

CLASSE DE BASE ASSOCIÉE À UN ÉLÉMENT DE FORMULAIRE

- Chaque élément de formulaire (input, select,...) sera associé à une abstraction côté typescript, un `FormControl`, qui permettra de suivre sa valeur et son état.
- Quand un élément de formulaire est vidé avec la méthode `reset` du `FormControl`, la valeur associée au `FormControl` est `null`.
- La méthode `valueChanges` d'un `FormControl` retourne un observable qui émet des valeurs lorsque l'utilisateur modifie le contenu de l'élément associé.

UTILISATION D'UN FORMCONTROL (1/3)





UTILISATION D'UN FORMCONTROL (2/3)

- Créez un nouveau composant `Search` dans votre projet et ajoutez `ReactiveFormsModule` dans ses `imports`.
- Le *template* de ce composant est très simple:

```
<label>
  Recherche dans l'intitulé de l'exercice
  <input [formControl]="searchControl" />
</label>
```

- L'attribut formControl permet de relier explicitement le champ de saisie à un `FormControl` qui doit être créé, propriété du composant nommée `searchControl` dans l'exemple.

```
protected readonly searchControl  = new FormControl<string | null>(null);
```

UTILISATION D'UN FORMCONTROL (3/3)

- Utilisez la méthode `valueChanges` du contrôle pour émettre un événement porteur du texte à rechercher uniquement si:
    - la valeur ne change pas pendant au moins 1/2 seconde,
    - la valeur est distincte de la précédente émise.
- Intégrez ce composant dans le *template* de votre composant `Exercises`. Il affichera dorénavant les exercices en tenant compte du texte à rechercher à partir d'un signal `displayedExercises` à calculer à partir du tableau d'exercices complet et du texte à rechercher.

CLASSE DE BASE ASSOCIÉE À UN FORMULAIRE

Tous les `FormControl` d'un formulaire peuvent être regroupés dans un `FormGroup`.

Il peut y avoir aussi des sous-`FormGroup` pour regrouper une partie du formulaire qui a des règles de validation communes.

LES ABSTRACTIONS

- `FormBuilder` est un service qu'on peut injecter pour créer facilement les `FormControl` et les `FormGroup`

```
export class CreateExercise {
  private fb = inject(FormBuilder);
  private startDate = new Date(Date.now());

  protected  titleFC = this.fb.control('');
  protected  dueDateFC = this.fb.control(formatDate(this.startDate,
    'yyyy-MM-dd', 'en'));
  protected  submittedFC = this.fb.control(false)

  // un FormGroup qui regroupe les FormControl
  protected readonly exerciseGroup = this.fb.group({
    title: this.titleFC,
    dueDate: this.dueDateFC,
    submitted: this.submittedFC
  });
```

LE FORMULAIRE

```
<!-- association du formulaire au FormGroup-->
<!-- l'événement de soumission d'un formulaire
 s'appelle ngSubmit dans Angular  -->
<form (ngSubmit)="onSubmit()" [formGroup]="exerciseGroup">
  <label>Intitulé du devoir (de 3 à 50 caractères)
    <!-- association d'un élément de formulaire
    avec le FormControl adéquat du FormGroup -->
    <input formControlName="title" />
  </label>
  <label>Date limite du rendu
    <input  type="date" formControlName="dueDate">
  </label>
  <label>
    <input type="checkbox" formControlName="submitted" />
    rendu
  </label>
  <button type="submit">Ajouter l'exercice</button>
</form>
```

LE FORMULAIRE

```
<!-- association du formulaire au FormGroup-->
<!-- l'événement de soumission d'un formulaire
 s'appelle ngSubmit dans Angular  -->
<form (ngSubmit)="onSubmit()" [formGroup]="exerciseGroup">
  <label>Intitulé du devoir (de 3 à 50 caractères)
    <!-- association d'un élément de formulaire
    avec le FormControl adéquat du FormGroup -->
    <input formControlName="title" />
  </label>
  <label>Date limite du rendu
    <input  type="date" formControlName="dueDate">
  </label>
  <label>
    <input type="checkbox" formControlName="submitted" />
    rendu
  </label>
  <button type="submit">Ajouter l'exercice</button>
</form>
```

LE FORMULAIRE

```
<!-- association du formulaire au FormGroup-->
<!-- l'événement de soumission d'un formulaire
 s'appelle ngSubmit dans Angular  -->
<form (ngSubmit)="onSubmit()" [formGroup]="exerciseGroup">
  <label>Intitulé du devoir (de 3 à 50 caractères)
    <!-- association d'un élément de formulaire
    avec le FormControl adéquat du FormGroup -->
    <input formControlName="title" />
  </label>
  <label>Date limite du rendu
    <input  type="date" formControlName="dueDate">
  </label>
  <label>
    <input type="checkbox" formControlName="submitted" />
    rendu
  </label>
  <button type="submit">Ajouter l'exercice</button>
</form>
```

UN PEU DE VALIDATION

- Angular fournit des validations qui rappelle les attributs ou type des éléments de formulaires en HTML :
  - `Validators.required`
  - `Validators.minLength(n)`
  - `Validators.maxLength(n)`
  - `Validators.min(n)`
  - `Validators.max(n)`
  - `Validators.pattern`
  - `Validators.email`
- Il est possible d'associer ces validations aux `FormControl`

```
this.fb.control('', Validators.required);
this.fb.control('', [Validators.required, Validators.minLength(3), Validators.max
```

QUELQUES PROPRIÉTÉS ET UNE MÉTHODE DES `FormControl`

- `value` la valeur bien sûr
- `valid` à `true` lorsque toutes les validations sont passées
- `invalid` l'inverse
- `dirty` sa valeur a été changée
- `errors` qui contient un objet avec des précisions sur les non-validations
- `hasError(nameError)` une méthode qui permet de tester la présence d'une non validation

QUELQUES PROPRIÉTÉS DES `FormGroup`

- on retrouve les attributs `valid`, `invalid`, `errors` et `value` mais la valeur de retour concerne l'ensemble des `FormControl` du `FormGroup`.

Les éléments correspondant aux `FormControl` et `FormGroup` se voient de plus affecter des classes reflétant leur état (`ng-dirty`, `ng-invalid`, ...) pour leur donner des styles adaptés.

On peut créer des validations personnalisées à l'aide de fonction (`ValidatorFunction` et `AsyncValidatorFunction`) .

UTILISATION DES VALIDATIONS DANS UN *TEMPLATE*

```
<!-- désactiver le bouton de soumission tant qu'au moins un des
 champs est en erreur -->
    <button type="submit"
      [disabled]="exerciseGroup.invalid">
        Ajouter l'exercice
    </button>
```

```
<!-- Afficher une précision sur l'erreur concernant un champ -->
@if (titleFC.hasError('minlength')) {
  <p class="error">Au moins 3 caractères</p>
}
```

UTILISATION DES VALEURS DANS LE TYPESCRIPT

```
this.exercisesService.create({
  id: '',
  title: this.titleFC.value!,
  dueDate: new Date(this.dueDateFC.value!),
  submitted: this.submittedFC.value!
});
```

Il peut être nécessaire d'utiliser ! si la valeur souhaitée est une `string` car Angular prend en compte le fait qu'un champs peut être `disabled` et renvoie donc un type `string | undefined`

EXERCICE

- Rendre fonctionnelle la page de création d'un nouvel exercice qui permet d'ajouter un exercice.
- Une création sans erreur redirige sur la page d'accueil du projet.

ENTRAINEMENT : MODIFIER L'EXERCICE (1/3)

- Ajouter dans le service `ExercisesService` deux méthodes:
  - une méthode pour récupérer un exercice à partir de son `id` en se servant du tableau où les exercices sont déjà stockés

```
get(id: string): ExerciseModel | undefined { ... }
```

  - une méthode pour modifier un exercice existant dans le tableau et au niveau de l'API

```
update(exercise: ExerciseModel): void { ... }
```

ENTRAINEMENT : MODIFIER L'EXERCICE (2/3)

- Le composant `EditExercise`:
  - récupère l'exercice qui correspond à l'`id` qui est dans l'URL. Si l'exercice n'existe pas, provoquer une alerte et une redirection sur la page d'accueil,
  - crée des `FormControl` similaires à ceux créés dans le composant `CreateExercises`, excepté que leur valeur doit correspondre aux propriétés de l'exercice récupéré,
  - a dans son *template* un formulaire,
  - la soumission du formulaire doit être possible uniquement si au moins un champ de l'exercice est modifié,

ENTRAINEMENT : MODIFIER L'EXERCICE (3/3)

- Le formulaire pour modifier un exercice est très similaire à celui qui permet d'en créer un => créer un composant `ExerciseForm` que les composants `CreateExercise` et `EditExercise` utiliseront.
    - Il prend en entrée un `ExerciseModel` quand il s'agit d'une modification,
    - A la soumission du formulaire le composant émet un événement `submit`, porteur d'un `NewExerciseModel` (création) ou d'un `ExerciseModel` (modification),
- refactorer `CreateExercise` et `EditExercise` pour utiliser le nouveau composant `ExerciseForm`.