Les formulaires

Pour créer des formulaires avec angular, deux modules sont disponible "**FormsModule**" et "**ReactiveFormsModule**" On peut obtenir le même résultat avec les deux, le premier est plus centré côté template et le second plus centré côté Typescript. Aucun n'est meilleurs mais "**FormsModule**" sera plus adapté à des petits formulaires et pour les débutants. Ils sont disponible dans la même library "**@angular/forms**"

On va aussi voir deux nouvelles directives "**ngForm**" et "**ngModel**" qui viennent de "**FormsModule**" On placera "**ngForm**" sur nos balises `<form>` et "**ngModel**" sur chacun des champs de celui ci. Il y a aussi "**ngModelGroup**" pour créer des groupes de champs Combiné ces trois vont nous permettre en temps réel de vérifier la validité du formulaire.

Import du module

La première question à se poser, c'est où je vais avoir besoin de formulaire? Dans notre cas, dans le module "**recette**" je vais donc l'ajouter aux imports du module.

```
import { FormsModule } from '@angular/forms';
// Dans l'import du décorateur :
FormsModule,
```

Créer nos Composants

Créons déjà deux nouveaux composants :

```
ng generate component recette/edit-recette --standalone false --module
recette.module
ng generate component recette/recette-form --standalone false --module
recette.module
```

Nous avons ajouté deux options à ces modules, la première **--standalone false** indique que le composant ne sera pas un standalone. La seconde, **--module recette.module** permet de mettre automatiquement à jour le module pour y importer nos nouveaux composants. Prenons de l'avance et ajoutons notre route dans "**recette.routes.ts**":

```
{path: "edit/:id", component: EditRecetteComponent}
```

C'est une route assez précise alors on va la préféré en haut, peu de risque de bloquer une route suivante. On notera que l'on ne fait pas de route pour le component "**recette-form**".

Puis rendons nous dans "**detail-recette.component.ts**" et créons une méthode :

```
goToEditRecette()
{
    this.router.navigate(["recettes/edit", this.recette?.id]);
}
```

Puis dans le HTML :

```
<button *ngIf="recette" (click)="goToEditRecette()">Éditer</button>
```

à côté de notre bouton de retour.

Rendons nous dans notre fichier "**edit-recette.component.ts**" et récupérons notre recette via l'id exactement comme dans "**detail-recette.component.ts**"

```
recette?: Recette;
constructor(private route: ActivatedRoute, private recetteService: RecetteService)
{}
ngOnInit(): void
{
    const recetteId: number =
parseInt(this.route.snapshot.paramMap.get("id")??"");
    this.recette = this.recetteService.getRecetteById(recetteId);
}
```

Puis ajoutons un titre à notre page :

```
<h2>Éditer {{recette?.name}}</h2>
```

Enfin nous allons voir que les composants ne sont pas uniquement des pages à part entière mais parfois juste un morceau de page, par exemple ici on va ajouter notre composant formulaire :

Si on regarde "**recette-form.component.ts**" on verra que son sélecteur se nomme "**app-recette-form**" (On pourrait le changer mais gardons le comme cela). Et bien ce sélecteur peut être utilisé comme balise HTML:

```
<app-recette-form></app-recette-form>
```

Cela ajoutera notre composant "recette-form" à notre composant "edit-recette", mais ajoutons lui deux attributs :

```
<app-recette-form *ngIf="recette" [recette]="recette"></app-recette-form>
<!--
    On affiche notre formulaire uniquement si on a une recette
```

```
    on attribut un élément [recette] à notre formulaire qui est égale à notre
variable "recette"
-->
```

Enfin ajoutons une ligne Dans le cas où il n'y a pas de recette correspondante :

```
<p *ngIf="!recette"> Aucune recette sélectionnée</p>
```

Rendons nous maintenant dans "**recette-form.component.ts**". On va lui indiquer comment récupérer la recette qui lui a été placé en argument :

```
// Ne pas oublier l'import de Input
@Input() recette?: Recette;
// "@input()" nous permet ici de récupérer l'objet donné par `[recette]="recette"
```

Exercice 5

Voir "**exercice-5.md**"

Correction

Voir "**exercice-5.html**" et "**exercice-5.scss**"

Gérer notre formulaire

Déclarations

Maintenant on va se rendre côté "**typescript**" puis ajouter à notre classe trois propriétés et quatre méthodes.

```
types: string[] = [];
ingredientsList: string = "";
stepsList: string = "";

// pensez à implémenter OnInit
ngOnInit(){}
hasType(){}
selectType(){}
onSubmit(){}
```

Ensuite j'injecte mon service de recette dans mon constructeur :

```
import { Router } from '@angular/router';
import { RecetteService } from '../recette.service';
```

```
/* ... */
constructor(
    private recetteService: RecetteService,
    private router: Router
    ) {}
```

Définitions

Dans mon ngOnInit :

```
// Je récupère la liste de mes types
this.types = this.recetteService.getRecetteTypeList();
if(!this.recette) return;
// Je transforme mes array en string.
this.ingredientsList = this.recette.ingredients.join("\n");
this.stepsList = this.recette.steps.join("\n");
```

Penchons nous ensuite sur notre méthode "**hasType(){}**" :

```
hasType(type: string): boolean
{
    if(!this.recette) return false;
    return this.recette.type.includes(type);
    // On return true si notre recette est de ce type, et false si il ne l'est
pas.
}
```

Puis voyons "**selectType(){}**":

```
selectType($event: Event, type:string): void
{
    if(!this.recette) return;
    const isChecked: boolean = ($event.target as HTMLInputElement).checked;
    this.recette.type = isChecked? type:"";
}
```

Ensuite penchons nous sur "**onSubmit(){}**":

```
onSubmit():void
{
    if(this.recette){
        // On retransforme nos chaînes de caractères en tableau.
        this.recette.ingredients = this.ingredientsList.split("\n");
        this.recette.steps = this.stepsList.split("\n");
    }
```

```
    this.router.navigate(["/recettes", this.recette?.id])
    // Attention de bien avoir importé le router dans le constructor.
}
```

Quand on soumet le formulaire, on va rediriger vers la page de la recette correspondante.

Events

On va maintenant ajouter des évènements à notre formulaire; Dans notre balise form on va placer :

```
<form (ngSubmit)="onSubmit()" #recetteForm="ngForm"></form>
<!--
    à la soumission on lance notre méthode onSubmit()
    On crée une propriété "recetteForm" qui va contenir notre objet
"HTMLFormElement" mais amélioré via la directive "ngForm"
-->
```

Cela va nous permettre d'avoir un objet formulaire avec de nouvelles fonctionnalités.

Dans la balise input pour le nom :

```
<!-- En plus des attributs déjà ajoutés qui ne sont pas affichés ici (name,
required, pattern...) -->
<input [(ngModel)]="recette.name" #name="ngModel">
```

On retrouve ici à la fois les **()** et les **[]**.

- **[]** indique que l'on va transmettre des données depuis TS jusqu'au HTML (comme via nos attributs).
- **()** indique un évènements donc qu'on va transmettre depuis HTML jusqu'à TS.

Pareil qu'avec "**ngForm**", on va récupérer une version améliorer de notre élément HTML grâce à "**ngModel**"
On va ajouter au message d'erreur suivant :

```
<div class="error" [hidden]="name.valid || name.pristine">
```

On remarquera ceci `[hidden]="name.valid || name.pristine"`

- L'attribut hidden pour cacher l'élément dans le HTML. Puis on utilise notre variable boosté par "ngModel" pour obtenir les propriétés suivantes :
    - "**valid**" vérifie juste la validité du champs (selon son type et pattern)
    - "**pristine**" vérifie si le champs a déjà été utilisé par l'utilisateur. (On ne voudrait pas de message d'erreur si l'utilisateur n'a pas encore touché au champ.)

Je vais recommencer l'opération pour les champs "**duration**", "**description**", "**duration**", "**ingredients**" et "**étapes**" en changeant à chaque fois le nom de la variable.

> attention pour les étapes ou ingrédient on utilisera "**ingredientsList**" ou "**stepsList**" que l'on a défini dans notre classe et non pas "**recette.ingredients**" ou "**recette.steps**"

Pour ce qui est du type, on va remplacer notre code par :

```
<label for="types">Type :</label>
<label *ngFor="let t of types" [attr.for]="t">
    <input type="radio"
    name="type"
    id="{{t}}"
    [value]="t"
    [checked]="hasType(t)"
    (change)="selectType($event, t)"
    required>
    <span style="background-color:{{t| typeColor}};">{{t}}</span>
</label>
```

Enfin on va ajouter un dernier attribut à notre bouton submit:

```
<!-- On désactive notre bouton tant que notre formulaire n'est pas valide. -->
<button type="submit" [disabled]="!recetteForm.form.valid">Sauvegarder</button>
```

Allons ajouter un peu de css, car angular peut aussi jouer avec cela:

```
.ng-valid[required]
{
    border: 2px solid green;
}
.ng-invalid:not(form)
{
    border: 2px solid red;
}
```

Angular place une classe `.ng-invalid` sur les éléments du formulaire qui ne sont pas valide, et une classe `.ng-valid` sur ceux qui sont valide.