Template

Interpolation

Pour commencer on va ouvrir notre "app.component.html" et remplacer notre sous-titre :

```
<h2>{{recetteList[0].name}}</h2>
```

par des paragraphes comportant toute nos recettes:

```
<p>{{recetteList[0].name}}</p>
```

ces "**{{}}**" permettent de l'interpolation. grâce à eux, on va pouvoir intégrer directement nos données JS dans notre html.

D'autres façons de faire existent :

```
<a [href]="recetteList[0].name">exemple</a>
<!-- L'attribut entre crochet = la valeur à lui donner. -->
<a [attr.href]="recetteList[0].name">exemple</a>
<!-- On utilise le mot clef attr suivi de l'attribut à modifier. -->
<div [class.super]="boolean">exemple</div>
<!--
    Le mot clef class suivi d'un nom de classe égale à un boolean.
    Permet d'afficher ou non la classe.
-->
<div [style.color]="false?'red':'green'"> exemple</div>
<!--
    le mot clef style suivi d'une propriété css et d'une ternaire.
    Permet de faire varifier le style de l'élément.
-->
```

Il existe d'autres façons de faire qu'on verra plus tard.

Event

Il est possible d'agir depuis le template vers le component entre autre chose avec des évènements.

```
<p (click)="selectRecette(recetteList[0])">{{recetteList[0].name}}</p>
<!-- Au clique on lance la méthode selectRecette -->
```

Retirons cet évènement qui était là pour l'exemple. On va maintenant ajouter un input de type number et lui donner notre méthode.

```
<input type="number" (input)="selectRecette($event)">
```

"$event" indique que l'on souhaite passer à notre méthode, notre évènement. Il nous indique maintenant une erreur car on donne un évènement à notre méthode alors qu'elle attend une recette. Rendons nous dans notre "**app.component.ts**" et changeons notre méthode :

```
// On indique que l'on va recevoir un évènement
selectRecette(event: Event): void{
    // On indique que notre event.target est un HTMLInputElement et on le
transforme en nombre
    const index: number = parseInt((event.target as HTMLInputElement).value);
    // On change notre console.log et on ajoute une condition
    if(this.recetteList[index]){
        console.log(`Vous avez sélectionné ${this.recetteList[index].name}`);
    }
}
```

Il nous restera une dernière erreur, retirons l'appel de la méthode dans le ngOnInit().

```
ngOnInit()
{
    // this.selectRecette(this.recetteList[0])
}
```

Maintenant quand on tape un index dans notre input, il nous indique la recette correspondante.

Mais devoir faire cela reste peu pratique. on est obligé d'indiquer le type de notre event.target, ajoutons à notre input quelques informations :

```
<input #inputRecette type="number" (input)="0">
<!--
    ce "#inputRecette" crée une variable qui va automatiquement contenir notre
input.
    ce ` (input)="0"` indique la présence d'un évènement mais qui n'est pas lié à
une méthode
-->

Puis ajoutons un paragraphes avec l'interpolation de la value de notre input.

```twig
<p>{{ inputRecette.value }}</p>
```
```

On verra alors à chaque touche poussé notre paragraphes se remplir.

Améliorons tout ça pour avoir la recette qui correspond s'afficher: On ajoute une propriété à notre class:

```
receteSelected: Recette|undefined
```

On indique que notre méthode "**selectRecette**" ne reçoit plus un évènement mais un string:

```
selectRecette(receteId: string): void
{
    // On change notre index aussi :
    const index: number = parseInt(receteId);
    // Et dans notre condition, on ajoute la définition de notre recette:
    if(this.receteList[index]){
        console.log(`Vous avez sélectionné ${this.receteList[index].name}`);
        this.receteSelected = this.receteList[index]
    }
}
```

Puis on retourne sur "app.component.html" et on change notre input:

```
<input #inputRecette type="number" (input)="selectRecette(inputRecette.value)">
```

Et enfin on change notre paragraphes par :

```
<p>Vous avez sélectionné : {{ receteSelected?.name }}</p>
<!-- le ? permet de ne pas provoquer d'erreur si jamais notre variable est
indéfini. -->
```

Cela dit, pour un utilisateur lambda, ce n'est pas habituel de chercher un index commençant par 0. Nos recettes ont des ID alors servont nous en pour les selectionner.

```
selectRecette(receteId: string): void
{
    const index: number = parseInt(receteId);
    // La méthode find permet de rechercher dans un tableau selon des critère bien
plus précis
    const recette: Recette|undefined = this.receteList.find(rec=> rec.id ===
index);
    // On remplace ensuite toute notre condition par :
    if(recette){
        this.receteSelected = recette;
        console.log(`Vous avez sélectionné ${recette.name}`);
    }
```

```
    // Et profitons en pour gérer le cas où cela ne correspond à rien:
    else{
        this.recetteSelected = recette;
        console.log(`Aucune recette correspondante`);
    }
}
```

QUESTION : Qu'est ce que l'on pourrait faire pour optimiser un peu ce code? **Réponse :** Sortir "this.recetteSelected = recette;" du if et du else Car apparaissant dans les deux cas, il peut être fait dans tous les cas.

Allons plus loins avec les évènements: Actuellement on fait une recherche à chaque touche entré par notre utilisateur, ce n'est vraiment pas optimisé, on va préféré rechercher que si il appui sur la toucher "entrée". on pourrait vérifier l'évènement "**input**", mais cela nous obligerait à nouveau à changer toute notre méthode. Il s'avère que Angular gère la detection des touches sur les évènements du clavier.

```
<input #inputRecette type="number"
(keyup.enter)="selectRecette(inputRecette.value)">
<!-- l'évènement ne se lancera que si on appui sur la touche entrée. Ce sont les
pseudo-events d'Angular -->
```

Condition Angular

Actuellement notre paragraphe n'affiche rien si aucune recette est selectionné. Nous allons changer cela :

```
@if(recetteSelected)
{
  <p>Vous avez sélectionné : {{ recetteSelected.name }}</p>
}
<!-- Ou bien si on ajoute l'import CommonModule dans nos imports du décorateur : -
->
<p *ngIf="recetteSelected">Vous avez sélectionné : {{ recetteSelected.name }}</p>
<!--
    "*" indique à angular une directive structurelle, on reviendra la dessus plus
tard
    "ngIf" indique d'afficher la balise dans laquelle il est à certaines
conditions
-->
```

On verra d'ailleurs un warning apparaître dans notre terminal, car maintenant que notre message ne s'affiche que si on a trouvé une recette, le "?" est devenu inutile.

Profitions en pour ajouter le cas contraire :

```
@else{
    <p>Aucune recette sélectionnée</p>
```

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

Boucle Angular

Jusqu'ici on a dû copier collé toutes nos recettes une à une, vous avouerez que ce n'est pas le plus pratique on va donc remplacer toute notre liste en paragraphe par :

```
@for (rec of receteList; track $index)
{
  <p>{{rec.name}}</p>
}
<!-- Ou avec le CommonModule : -->
<p *ngFor="let rec of receteList">{{rec.name}}</p>
<!--
    "ngFor" est une directive structurelle qui va boucler sur chaque élément de
notre tableau
     à chaque itération un nouvel élément de mon tableau est rangé dans la
variable "rec"
-->
```

Ces deux directive structurelle sont importé par le commonModule dans vos composants. Ce qui ne sera pas forcément le cas d'autres que l'on verra plus tard.

EXERCICE 2

Consigne dans le fichier "**exercice-2.md**"

Correction

Voir fichiers "**exercice-2.html**" et "**exercice-2.scss**".