Ateliers Angular

TP4 : Directives Structurelles et Attributs

TP 4 : Directives Structurelles et Attributs

I. Objectif

Utiliser les directives `*ngIf`, `*ngFor`, `[ngClass]`, `[ngStyle]`

Objectifs spécifiques

- Comprendre le fonctionnement des directives *ngIf et *ngFor pour contrôler l'affichage conditionnel et l'itération sur des listes dans Angular.
- Appliquer les directives *ngIf et *ngFor pour afficher dynamiquement les informations des étudiants dans l'application.
- Utiliser [ngClass] et [ngStyle] pour modifier de manière conditionnelle les styles et les classes CSS des composants
- Analyser comment la combinaison des directives *ngIf, *ngFor, [ngClass], [ngStyle] affecte la réactivité et l'ergonomie de l'application.
- Créer une application complète de gestion des étudiants en intégrant efficacement les directives Angular pour améliorer l'interactivité et l'expérience utilisateur.

Ce TP comporte trois types d'activités :

Ateliers en salle : Ces exercices seront réalisés collectivement sous la direction de l'enseignant.

Ateliers guidés : Ces exercices sont accompagnés de solutions détaillées pour vous aider dans votre apprentissage. N'hésitez pas à solliciter votre enseignant à chaque étape si vous rencontrez des difficultés.

Projet : Nous poursuivons le développement du projet et nous appliquons les concepts de ce TP dans le projet.

II. Les ateliers en salle

Activité 1 : Affichage conditionnel avec ngIf

Créez un composant qui affiche un message de bienvenue seulement si l'utilisateur est connecté.

- Dans votre composant, déclarez une propriété isLoggedIn initialisée à false.
- Ajoutez un bouton dans le template qui permet de basculer isLoggedIn entre true et false en utilisant un événement (click).
- Utilisez *ngIf pour afficher le message "Bienvenue, utilisateur!" uniquement lorsque isLoggedIn est true.

Défi :

- `Validation` : Ajouter une condition pour se connecter. Par exemple si le contenu d'un champs texte est égale à votre Prenom
- `Design` : Ajoutez des styles CSS pour rendre l'interface plus agréable

Activité 2 : Liste d'articles avec ngFor

On se propose de créer une liste d'articles et les afficher dans le même projet.

1. Créer un nouveau composante articles.
2. Dans le composant, créez une propriété articles qui est un tableau d'objets, chaque objet ayant des propriétés comme **titre** et **contenu**.
3. Dans le template, utilisez *ngFor pour itérer sur articles et afficher le titre et le contenu de chaque article.
4. Ajoutez un champ de saisie et un bouton pour permettre l'ajout de nouveaux articles à la liste.

Activité 3 : Classes dynamiques avec [ngClass]

On se propose dans cet exercice de mettre en forme les éléments de la liste en fonction de certaines conditions.

1. Dans chaque objet article, ajoutez une propriété importance avec des valeurs possibles : 'élevée', 'moyenne', 'faible'.
2. Utilisez [ngClass] pour appliquer une classe CSS différente en fonction de la valeur de importance.
3. Par exemple, appliquez la classe 'important' si l'importance est 'élevée'.
4. Définissez les styles correspondants dans votre fichier CSS pour chaque classe.

Activité 4 : Styles dynamiques avec [ngStyle]

Ajustez le style des éléments en fonction de données du composant.

1. Créez un composant qui affiche une liste de produits avec leur niveau de stock.
2. Utilisez [ngStyle] pour changer la couleur du texte du niveau de stock :
    1. Vert si le stock est supérieur à 50.
    2. Orange si le stock est entre 20 et 50.
    3. Rouge si le stock est inférieur à 20.
3. Assurez-vous que les valeurs de stock peuvent être mises à jour dynamiquement.

Activité 5 : Combinaison de ngIf, ngFor, [ngClass] et [ngStyle]

Créez une application de gestion de tâches complète.

1. Dans le composant, définissez une propriété taches qui est un tableau d'objets, chaque tâche ayant description, statut (complétée ou non) et priorité (haute, moyenne, basse).
2. Utilisez *ngFor pour afficher la liste des tâches.
3. Utilisez [ngClass] pour barrer le texte des tâches complétées.
4. Utilisez [ngStyle] pour changer la couleur du texte en fonction de la priorité :
    1. **Rouge** pour 'haute'.
    2. **Orange** pour 'moyenne'.

3. **Vert** pour 'basse'.

5. Ajoutez un bouton à chaque tâche pour basculer son statut entre complétée et non complétée.
6. Utilisez *ngIf pour afficher un message "Aucune tâche à afficher" si la liste est vide.

Activité 6 : Affichage conditionnel avancé avec ngIf et else

Améliorez l'interface utilisateur en fournissant des messages alternatifs.

1. Dans le composant de gestion de tâches, utilisez la syntaxe *ngIf...else pour afficher un message différent lorsque la liste des tâches est vide.
2. Créez un template ng-template nommé pasDeTaches qui affiche "Vous n'avez pas de tâches en cours".
3. Utilisez ce template avec *ngIf pour gérer l'affichage alternatif.

III. Atelier guidé

Activité 1

Créez un nouveau projet Angular en utilisant la commande suivante :

```
ng new tp4 --no-standalone
```

1. Déclaration de la Propriété isLoggedIn

Ouvrez le fichier `welcome.component.ts` et ajoutez la propriété `isLoggedIn` initialisée à `false`

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-welcome',
  templateUrl: './welcome.component.html',
  styleUrls: ['./welcome.component.css']
})
export class WelcomeComponent {
  // Propriété indiquant si l'utilisateur est connecté
  isLoggedIn: boolean = false;

  // Méthode pour basculer l'état de connexion
  toggleLogin() {
    this.isLoggedIn = !this.isLoggedIn;
  }
}
```

2. *Modification du Template HTML*

Ouvrez le fichier `welcome.component.html` et ajoutez un bouton pour basculer l'état de connexion, ainsi que le message de bienvenue conditionne

```html
<!-- Bouton pour basculer l'état de connexion -->
<button (click)="toggleLogin()">
  {{ isLoggedIn ? 'Se déconnecter' : 'Se connecter' }}
</button>


<!-- Message de bienvenue conditionnel -->
<p *ngIf="isLoggedIn">Bienvenue Utilisateur!</p>
```

Activité 2

Étape 1 : Création du Projet Angular

Déjà fait dans l'exercice précèdent.

Étape 2 : Création du Composant `Articles`

Générez un composant nommé `articles` qui sera utilisé pour gérer la liste des articles :

```
ng generate component articles
```

Cette commande créera quatre fichiers pour le composant `articles` :

- `articles.component.ts`
- `articles.component.html`
- `articles.component.css`
- `articles.component.spec.ts`
- UPDATE src/app/app.module.ts (578 bytes)

Étape 3 : Implémentation de la Logique dans le Fichier TypeScript

Ouvrez le fichier `articles.component.ts` et implémentez la logique nécessaire pour gérer la liste des articles :

```typescript
import { Component } from '@angular/core';

@Component({
  selector: 'app-articles',
  templateUrl: './articles.component.html',
  styleUrls: ['./articles.component.css']
})
export class ArticlesComponent {
  // Propriété qui contient la liste des articles
  articles = [
    { titre: 'LapTop Asus', contenu: 'Contenu de l\'article 1' },
    { titre: 'Laptop Gaming', contenu: 'Contenu de l\'article 2' },
    { titre: 'Laptop HP', contenu: 'Contenu de l\'article 3' }
  ];

  // Propriétés pour capturer les valeurs des nouveaux articles
  newTitle = '';
  newContent = '';

  // Méthode pour ajouter un nouvel article
```

```
  addArticle() {
    if (this.newTitle && this.newContent) {
      this.articles.push({
        titre: this.newTitle,
        contenu: this.newContent
      });
      // Réinitialisation des champs
      this.newTitle = '';
      this.newContent = '';
    }
  }
}
```

Étape 4 : Modifiez le Template HTML

Ouvrez le fichier `articles.component.html` et implémentez le formulaire d'ajout et l'affichage de la liste des articles :

<!-- Formulaire pour ajouter un nouvel article -->

```
<div>
  <input type="text" [(ngModel)]="newTitle" placeholder="Titre de l'article" />
  <input type="text" [(ngModel)]="newContent" placeholder="Contenu de l'article" />
  <button (click)="addArticle()">Ajouter l'article</button>
</div>

<!-- Liste des articles -->
<div *ngFor="let article of articles">
  <h3>{{ article.titre }}</h3>
  <p>{{ article.contenu }}</p>
</div>
```

- Le formulaire comporte deux champs de saisie liés aux propriétés `newTitle` et `newContent` à l'aide de `[(ngModel)]`.
- Le bouton appelle la méthode `addArticle()` pour ajouter un nouvel article.
- La directive `*ngFor` est utilisée pour itérer sur la liste `articles` et afficher chaque article.

Étape 5 : Importez `FormsModule`

Pour que `[(ngModel)]` fonctionne, vous devez importer `FormsModule` dans le module Angular approprié. Ouvrez `app.module.ts` et ajoutez `FormsModule` aux imports :

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { WelcomeComponent } from './welcome/welcome.component';
import { ArticlesComponent } from './articles/articles.component';

@NgModule({
  declarations: [
    AppComponent,
    WelcomeComponent,
    ArticlesComponent
  ],
```

```
    imports: [
      BrowserModule,
      FormsModule,
    ],
    providers: [],
    bootstrap: [AppComponent]
  })
  export class AppModule { }
```

Défi :

- `Validation` : Vous pouvez ajouter une validation plus avancée pour éviter les titres vides ou les doublons.
- `Design` : Ajoutez des styles CSS pour rendre l'interface plus agréable.

Activité 3

pour ajouter une mise en forme conditionnelle aux éléments de la liste d'articles en fonction de leur niveau d'importance :

Étape 1 : Ajouter la Propriété `importance` aux Articles

- Modifiez la liste d'articles dans le fichier `articles.component.ts` pour inclure une propriété `importance` à chaque article. Les valeurs possibles peuvent être : `'élevée'`, `'moyenne'`, `'faible'`.
- Ajoutez également une propriété pour capturer l'importance lors de l'ajout d'un nouvel article.

Code modifié dans `articles.component.ts` :

```
articles = [
  { titre: 'Article 1', contenu: 'Contenu de l\'article 1', importance: 'élevée' },
  { titre: 'Article 2', contenu: 'Contenu de l\'article 2', importance: 'moyenne' },
  { titre: 'Article 3', contenu: 'Contenu de l\'article 3', importance: 'faible' }
];
```

- Ajoutez la propriété `newImportance = 'moyenne';` pour capturer l'importance des nouveaux articles.

- Étape 2 : Mettre à Jour le Formulaire pour Sélectionner l'Importance

- Dans `articles.component.html`, ajoutez un `<select>` au formulaire d'ajout d'article pour choisir l'importance de l'article.

Code modifié dans `articles.component.html` :

```
<input type="text" [(ngModel)]="newTitle" placeholder="Titre de l'article" />
<input type="text" [(ngModel)]="newContent" placeholder="Contenu de l'article" />
<select [(ngModel)]="newImportance">
  <option value="élevée">Élevée</option>
  <option value="moyenne">Moyenne</option>
  <option value="faible">Faible</option>
```

```
</select>
<button (click)="addArticle()">Ajouter l'article</button>
```

Étape 3 : Utiliser `ngClass` pour Appliquer une Mise en Forme Conditionnelle

- Utilisez `[ngClass]` dans la boucle `*ngFor` pour appliquer des classes CSS différentes en fonction de la valeur de la propriété `importance`.

Code modifié dans `articles.component.html` :

```
<div *ngFor="let article of articles" [ngClass]="{
    'important': article.importance === 'élevée',
    'moyen': article.importance === 'moyenne',
    'faible': article.importance === 'faible'
  }">
  <h3>{{ article.titre }}</h3>
  <p>{{ article.contenu }}</p>
</div>
```

- Ici, `ngClass` applique la classe `'important'`, `'moyen'`, ou `'faible'` selon l'importance de chaque article.

- Étape 4 : Définir les Styles CSS

- Ouvrez le fichier `articles.component.css` et ajoutez les classes CSS pour chaque niveau d'importance.

Code CSS dans `articles.component.css` :

```
.important {
  background-color: #ffcccc;
  border: 1px solid red;
}

.moyen {
  background-color: #fff2cc;
  border: 1px solid orange;
}

.faible {
  background-color: #ccffcc;
  border: 1px solid green;
}
```

- Ces styles CSS définissent une couleur de fond et une bordure différentes pour chaque niveau d'importance (élevée, moyenne, faible).

- Étape 5 : Vérification

- Enregistrez tous les fichiers et relancez l'application Angular avec la commande suivante :

```
ng serve
```

Ces étapes vous permettront de mettre en forme vos articles en fonction de leur importance, et de rendre votre interface plus intuitive visuellement. N'hésitez pas à personnaliser les couleurs et les styles pour mieux répondre à vos besoins !

```
<-- Bouton pour basculer l'état de connexion -->
<button (click)="toggleLogin()">
  {{ isLoggedIn ? 'Se déconnecter' : 'Se connecter' }}
</button>

<!-- Message de bienvenue conditionnel -->
<p *ngIf="isLoggedIn">Bienvenue, utili<!sateur!</p>
```

Activité 4

Dans cette partie, il est question de développer un composant de l'application ListeEtudiants pour afficher une liste des étudiants et permet de sélectionner un étudiant et afficher ses détails.

Dans le fichier TypeStudent.ts on va définir un tableau statique des étudiants. Il est préférable de déclarer ce tableau dans un fichier à part pour pouvoir le manipuler sans toucher le code.

1) Définissez une constante `Students` comme un tableau de 10 étudiants. Le fichier devrait ressembler à ceci.

```
export interface Student {
  id: number;
  name: string;
  classe ?:string
}
export const Students: Student[] = [
  { id: 1, name: 'Asma' },
  { id: 2, name: 'Oumaima' },
  { id: 3, name: 'Raouf' },
  { id: 4, name: 'Ibrahim' },
  { id: 5, name: 'Nour' },
  { id: 6, name: 'Rihem' },
  { id: 7, name: 'Dyama' },
  { id: 8, name: 'Dr IQ' },
  { id: 9, name: 'Howa' },
  { id: 10, name: 'Hiya' }
];
```

2) Pour pouvoir utiliser cette liste dans le composant etudiant, il faut déclarer et initialisez un attribut `etuds` de type `Etudiants` dans la classe `etudiantComponent`.

```
export class EtudiantComponent implements OnInit {
  etuds = Students;
}
```

L'application contient toujours des erreurs, pourquoi ?

……

……

3) On se proposer maintenant de lister les étudiants avec `*ngFor` qui est une directive répéteur permettant de boucler et d'injecter les éléments dans le DOM. Il répète l'élément hôte pour chaque élément d'une liste.

Que fait la directive *ngFor : donner au moins trois exemples d'utilisations

……

……

……

……

……

……

Ouvrez le fichier de modèle `Etudiant.Component.html` et apportez les modifications suivantes :

a. Ajoute titre <`h2>` en début,
b. La liste des étudiants sera affichée avec une balise <ul>. En dessous, ajoutez une liste HTML non ordonnée (`<ul>`)
c. La balise `<li>` doit être encadrée avec la directive `ngFor` pour affiche les propriétés d'un `etudiant`.
d. Saupoudrez quelques classes CSS pour le style (on va ajouter les styles CSS sous peu).

Faites en sorte que cela ressemble à ceci :

```
<h2>Liste des Etudiants</h2>
<ul class="etudiants">
  <li *ngFor="let etudiant of etudiants">
    <span  class="badge">{{etudiant.id}}</span> -{{etudiant.name}}
  <li>
</ul>
```

N'oubliez pas l'astérisque (*) devant `ngFor`. C'est une partie essentielle de la syntaxe pour la directive structurelle. Une fois le navigateur actualisé, la liste des étudiants apparaît.

4) Importez CommonModule pour pouvoir utiliser ngFor

```
import  {CommonModule} from '@angular/common';
```

5) La liste des étudiants doit être attrayante et doit répondre visuellement lorsque l'utilisateur survolent et sélectionnent un étudiant dans la liste.

Les styles de base pour l'ensemble de l'application dans `styles.css`. Cette feuille de style n'incluait pas les styles de cette liste d'étudiants. On peut ajouter plus de styles à `styles.css` et continuer à la développer à mesure qu'on ajouter des composants. Mais il est préférable plutôt de définir des styles privés pour un composant spécifique et conserver tout ce dont un composant a besoin - le code, le HTML et le CSS - en un seul endroit.

Cette approche facilite la réutilisation du composant ailleurs et offre l'apparence prévue du composant même si les styles globaux sont différents. Pour cela, on va définir des styles privés soit en ligne dans le tableau `@Component.styles`, soit en tant que fichier (s) de feuille de style identifié dans le tableau `@Component.styleUrls`.

Lorsque Angular a généré le `Etudiant.Component`, il a créé une feuille de style `etudiant. component.css` vide l'a pointé dans `styleUrls` comme ceci. On va utiliser ce fichier pour définir les styles du composant étudiants.

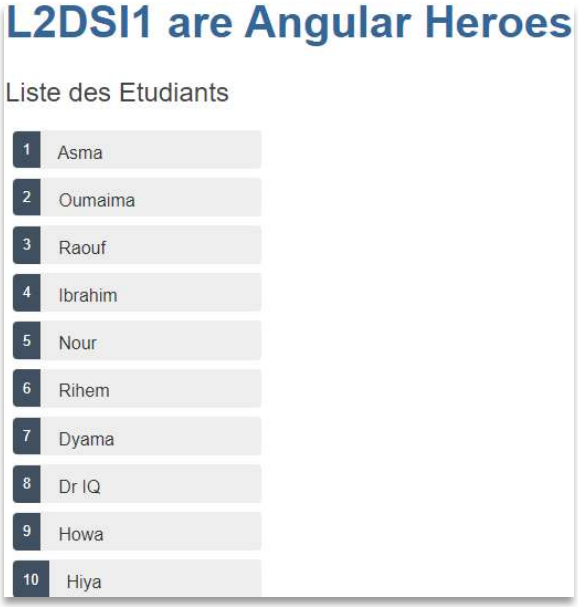
```
@Component({
  selector: 'app-etudiant',
  templateUrl: './etudiant.component.html',
  styleUrls: ['./etudiant.component.css']
})
```

Ouvrez le fichier `etudiant.component.css` et tapez ce code css dans les styles CSS privés pour `EtudiantsComponent`.

```
.etudiants {
  margin: 0 0 2em 0;
  list-style-type: none;
  padding: 0;
  width: 15em;
}
.etudiants li {
  cursor: pointer;
  position: relative;
  left: 0;
  background-color: #EEE;
  margin: .5em;
  padding: .3em 0;
  height: 1.6em;
  border-radius: 4px;
}
.etudiants li:hover {
  color: #607D8B;
  background-color: #DDD;
  left: .1em;
}
```

```
.etudiants li.selected {
  background-color: #CFD8DC;
  color: white;
}
.etudiants li.selected:hover {
  background-color: #BBD8DC;
  color: white;
}
.etudiants .badge {
  display: inline-block;
  font-size: small;
  color: white;
  padding: 0.8em 0.7em 0 0.7em;
  background-color: #405061;
  line-height: 1em;
  position: relative;
  left: -1px;
  top: -4px;
  height: 1.8em;
  margin-right: .8em;
  border-radius: 4px 0 0 4px;
}
```

Les styles et les feuilles de style identifiés dans les métadonnées `@Component` sont limités à ce composant spécifique. Les styles `etudiants.component.css` s'appliquent uniquement au `EtudiantsComponent` et n'affectent pas le HTML externe ou le HTML d'un autre composant.



Bonus

Donner Le style correspondant à :

- `Etudiants li`
- `Etudiants li.hover`
- `Etudiants li.selected`

A ce stade l'application doit être comme la capture ci-contre.

Partie 5 : Affichage Maître / Détail

On se propose maintenant d'avoir deux vues :

- La vue **maître** peut afficher une liste des étudiants.
- La vue **détail** affiche typiquement les informations détaillées d'un étudiant.

Généralement lorsque l'utilisateur clique sur un étudiant dans la liste maître, le composant détail doit afficher l'étudiant sélectionné en bas de la page. Il faut alors définir un écouter de l'événement de clic sur l'élément de l'étudiant et mettre à jour les détails en fonction.

1) On va ajouter le traitement de l'événement de clic. Dans le fichier `etudiants.component. html` ajoutez à la balise `<li>` :

```
    <li *ngFor="let e1 of etuds" (click)="onSelect(e1)">
```

Les parenthèses autour de click indiquent à Angular d'écouter l'événement click de l'élément <li>. Lorsque l'utilisateur clique dans la ligne, Angular exécute l'expression onSelect(etudiant).

Donc il faut définir une méthode `onSelect()` dans `etudiant.component.ts` pour afficher l'étudiant correspondant.

2) Dans ce fichier, ajouter un attribut `selectedEtudiant` a la classe sans l'initialiser dans la classe EtudiantComponent. Et ajoutez la méthode `onSelect()`, qui attribue l'étudiant sur lequel vous avez cliqué du modèle au `selectedEtudiant` du composant.

```
selectedEtudiant !: Student;

onSelect(e: Student): void {
 this.selectedStudent = e;
}
```

Ajouter l'import nécessaire pour corriger l'erreur.

A ce stade le fichier `src/app/etudiant/etudiant.component.ts` est comme suit :

```
import { Component } from '@angular/core';
import  {CommonModule} from '@angular/common';
import {Student, Students} from './TypeStudent' ;
import { FormsModule } from '@angular/forms';  // <-- Importer FormsModule

@Component({
  selector: 'app-etudiant',
  standalone: true,
  imports: [FormsModule, CommonModule],
  templateUrl: './etudiant.component.html',
  styleUrl: './etudiant.component.css'
})
export class EtudiantComponent {
  etuds=Students;
  selectedEtudiant!: Student ;

  onSelect(etudiant: Student): void {
    this.selectedEtudiant = etudiant;
  }
}
```

3) Actuellement, vous avez une liste d'étudiants dans le modèle. On se propose maintenant d'afficher les détails d'un étudiant si on cliquer dessus. Donc on a besoin d'une section pour que les détails soient rendus dans le modèle.

Ajoutez le code pour afficher les détails d'un étudiant sous la section de la liste (à la fin du fichier `etudiants.component.html`) :

```
<h2>Liste des Etudiants</h2>
<ul class="etudiants">
  <li *ngFor="let etudiant of etuds" (click)="onSelect(etudiant)">
    <span  class="badge">{{etudiant.id}}</span> {{etudiant.name}}
  <li>
</ul>

<h2>Info de {{selectedEtudiant.name | uppercase}} </h2>
<div> <span>id: </span> {{selectedEtudiant.id}}</div>
  <div>
    <label> <strong>name</strong>
      <input [(ngModel)]="selectedEtudiant.name" placeholder="name">
      </label>
  </div>
```

Une fois le navigateur actualisé, l'application affiche des lignes vides.

Pour connaitre l'origine de ce vide, une erreur certainement, on va utiliser l'outils de débogage du navigateur. Si vous ouvrez les outils de développement du navigateur (par le menu ou la touche F12), dans la console un message d'erreur comme celui-ci :

```
core.js:5967 ERROR TypeError: Cannot read property 'name' of undefined
```

En effet, lorsque l'application démarre, l'attribut `selectedEtudiant` n'est pas défini. Alors les expressions de liaison dans le modèle qui font référence aux propriétés de cet attribut (telles que `{{selectedEtudiant.name}}`) - *doivent provoque une erreur* car aucun étudiant n'est sélectionné.

4) Pour corriger cette erreur, il faut masquer les détails non initialisés avec ** ngIf*. Cette partie du code ne doit fonctionner et afficher les détails de l'étudiant sélectionné que si l'attribut `selectedEtudiant` est initialisé.

On va encadrer le code HTML des détails de l'étudiant dans un `<div>` avec le la directive `ngIf` d'Angular. Ajoutez `*ngIf` à la balise `<div>` et définissez-la sur `selectedEtudiant`.

N'oubliez pas l'astérisque (*) devant `ngIf`. C'est une partie essentielle de la syntaxe.

Ci-après le fichier `src/app/etudiant/etudiants.component.html`

```
<h2>Liste des Etudiants</h2>
<ul class="etudiants">
  <li *ngFor="let e1 of etuds" (click)="onSelect(e1)">
    <span  class="badge">{{e1.id}}</span> {{e1.name}}
  <li>
</ul>

<div *ngIf="selectedEtudiant">
  <h2>Info de {{selectedEtudiant.name | uppercase}} </h2>
  <div> <span>id: </span> {{selectedEtudiant.id}}</div>
  <div>
    <label>
      <input [(ngModel)]="selectedEtudiant.name" placeholder="name">
      </label>

  </div>
</div>
```

Une fois le navigateur actualisé, la liste des noms réapparaît. La zone de détails est vide. Cliquez sur un étudiant dans la liste des Etudiants et ses détails s'affichent. L'application fonctionne à nouveau.

5) Il est difficile d'identifier l'étudiant *sélectionné* dans la liste lorsque tous les éléments `<li>` se ressemblent. Si l'utilisateur clique sur un étudiant, ce dernier doit être rendu avec une couleur d'arrière-plan distinctive mais subtile.

Cette coloration de l'étudiant sélectionné est le travail de la classe CSS `selected` dans les styles que vous avez ajoutés précédemment. Il vous suffit d'appliquer la classe `selected` au `<li>` lorsque l'utilisateur clique dessus.

La liaison de classe Angular facilite l'ajout et la suppression d'une classe CSS de manière conditionnelle. Ajoutez simplement

```
[class.selected]="etudiant === selectedEtudiant"
```

Le balise `<li>` fini, ressemble à ceci :

```
<h2>Liste des Etudiants</h2>
<ul class="etudiants">
  <li *ngFor="let e1 of etuds" [class.selected]="e1 === selectedEtudiant"
                (click)="onSelect(e1)">
    <span  class="badge">{{e1.id}}</span> {{e1.name}}
  <li>
</ul>
```

Défi :

1. Pourquoi on a utilisé une affectation avec « === » ?
2. créer une liste plus riche avec d autre données sur l'étudiant.

IV. La partie Projet:

Notre objectif pour le projet, au cours de cette séance, est d'utiliser les **directives structurelles** d'Angular pour gérer l'affichage des questions du jeu, masquer ou afficher des éléments en fonction des actions du joueur (réponses correctes ou incorrectes), et gérer des listes d'éléments dynamiques (comme une liste de scores, d'historiques de réponses, etc.).

Objectifs Spécifiques :

- Apprendre à utiliser les **directives structurelles** en Angular (*ngIf, *ngFor) pour contrôler dynamiquement l'affichage des éléments en fonction des actions des joueurs dans le jeu.
- Apprendre à utiliser les **directives attributs** pour appliquer des styles et des classes conditionnelles en fonction des actions du joueur, en créant un feedback visuel dans l'interface du jeu.

Les directives structurelles

1. **Affichage Conditionnel avec *ngIf :**

- Utiliser la directive structurelle *ngIf pour afficher ou masquer des éléments en fonction des actions du joueur, comme une bonne ou mauvaise réponse.
- **Étape 1 :** Ajouter un message conditionnel qui s'affiche si la réponse est correcte ou incorrecte.

```
<div *ngIf="isReponse; else incorrectMessage">
  <p>Bonne réponse ! Vous gagnez des points.</p>
</div>
<ng-template #incorrectMessage>
  <p>Mauvaise réponse. Essayez encore !</p>
</ng-template>
```

Code TypeScript :

```
isReponse: boolean = false;
checkAnswer(playerAnswer: string, question: any) {
  this.isReponse = playerAnswer === question.reponse;
  if (this.isReponse) {
    this.score += 10;
  } else {
    this.score -= 5;
  }
}
```

2. Ajouter un feedback dynamique basé sur la réponse du joueur. Le message doit apparaître ou disparaître selon la réponse choisie.

Itération Dynamique avec *ngFor :

- Utiliser *ngFor pour itérer sur des listes d'éléments dynamiques, comme l'affichage de plusieurs questions ou d'un historique des scores.
- **Étape 2 :** Créer une liste de questions et les afficher en boucle avec *ngFor :

```
<div *ngFor="let question of questions">
  <h3>{{ question.question }}</h3>
  <button *ngFor="let option of question.options"
(click)="onSelectOption(option, question)">
    {{ option }}
  </button>
</div>
```

-

3. Utiliser *ngFor pour afficher un historique des réponses du joueur (bonnes et mauvaises réponses).

Combinaison de *ngIf et *ngFor :

- Créer une interface conditionnelle où les questions ne s'affichent que si elles n'ont pas encore été répondues, et l'historique des réponses s'affiche au fur et à mesure.

- **Étape 3 :** Modifier le code pour masquer une question une fois qu'elle a été répondue :

```
<div *ngFor="let question of questions">
  <div *ngIf="!question.answered">
    <h3>{{ question.question }}</h3>
    <button *ngFor="let option of question.options" (click)="onSelectOption(option,
question)">
      {{ option }}
    </button>
  </div>
</div>
```

- **Code TypeScript :**

```
onSelectOption(selectedOption: string, question: any) {
  question.answered = true; // Marquer la question comme répondue
  if (selectedOption === question.reponse) {
    this.score += 10;
  } else {
    this.score -= 5;
  }
}
```

Mise en Pratique - Développement du Jeu :

Scénario du Jeu :
L'étudiant répond à une série de questions, et à chaque réponse, l'affichage est mis à jour dynamiquement. Une question disparaît une fois qu'elle est répondue, et un message s'affiche pour informer l'étudiant si sa réponse est correcte ou incorrecte. De plus, un historique des réponses peut être affiché à la fin de chaque question pour permettre au joueur de suivre ses progrès.

- **Défi 1 :** Ajouter une fonctionnalité où les questions s'affichent une à la fois, et la suivante apparaît seulement après avoir répondu à la précédente.
- **Défi 2 :** Implémenter un tableau qui affiche toutes les questions déjà répondues avec leur statut (correcte ou incorrecte) en utilisant *ngFor.
- **Défi 3 :** Créer un bouton "Terminer le jeu" qui n'apparaît qu'une fois que toutes les questions ont été répondues, en utilisant *ngIf.

Les directives attributs

Les directives attributs ([ngClass] et [ngStyle]) pour modifier dynamiquement les **styles** et les **classes CSS** des éléments du jeu en fonction des réponses du joueur (bonne ou mauvaise réponse). Cela permettra d'améliorer l'expérience utilisateur en ajoutant un feedback visuel immédiat.

1. Utilisation de [ngClass] pour Modifier les Classes CSS :
   - Utiliser [ngClass] pour appliquer différentes classes CSS en fonction des réponses correctes ou incorrectes du joueur.
   - **Étape 1 :** Ajouter des classes CSS pour les bonnes et mauvaises réponses :

```
.correct-answer {
  background-color: green;
  color: white;
}
```

```
.incorrect-answer {
  background-color: red;
  color: white;
}
```

- **Étape 2 :** Dans le template Angular, utiliser [ngClass] pour appliquer dynamiquement ces classes en fonction de la réponse :

```
<div *ngFor="let question of questions">
  <button *ngFor="let option of question.options"
          [ngClass]="{'correct-answer': isCorrect(option, question), 'incorrect-answer':
!isCorrect(option, question)}"
          (click)="onSelectOption(option, question)">
    {{ option }}
  </button>
</div>
```

- Code TypeScript pour la méthode isCorrect :

```
isCorrect(option: string, question: any): boolean {
  return option === question.reponse;
}
```

2. Implémenter un feedback visuel pour indiquer immédiatement si l'option sélectionnée est correcte ou incorrecte en appliquant une classe CSS dynamique.

Utilisation de [ngStyle] pour Modifier les Styles en Ligne :

- Utiliser [ngStyle] pour appliquer des styles spécifiques en fonction de l'état du jeu, comme modifier la taille, la couleur ou l'opacité des éléments.
- **Étape 3 :** Ajouter un effet visuel pour mettre en évidence la réponse choisie par l'étudiant, par exemple en changeant la couleur du bouton au survol :

```
<button *ngFor="let option of question.options"
        [ngStyle]="{'background-color': isSelected(option) ? 'blue' : 'grey'}"
        (click)="onSelectOption(option, question)">
  {{ option }}
</button>
```

3. Mettre à jour l'apparence des options avec [ngStyle] lorsque l'étudiant survole un bouton ou clique sur une réponse, pour une meilleure interaction visuelle.

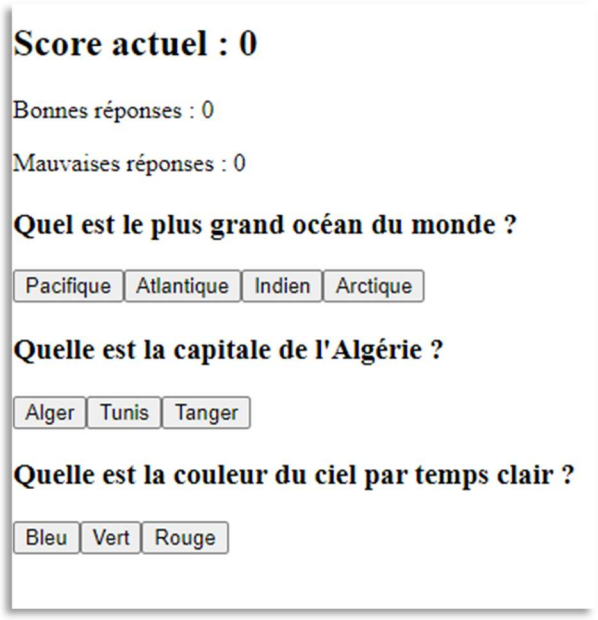
Combinaison de [ngClass] et [ngStyle] pour le Feedback Visuel :

- Combiner [ngClass] et [ngStyle] pour appliquer à la fois des classes CSS et des styles en ligne afin de gérer plusieurs effets visuels simultanément.
- **Étape 4 :** Appliquer une classe dynamique en fonction de l'état du jeu (bonne ou mauvaise réponse) et utiliser [ngStyle] pour ajuster des styles spécifiques comme l'opacité ou la taille des boutons de réponse :

```
<button *ngFor="let option of question.options"
        [ngClass]="{'correct-answer': isCorrect(option, question)}"
```

```
        [ngStyle]="{'opacity': question.answered ? '0.5' : '1'}"
        (click)="onSelectOption(option, question)">
  {{ option }}
</button>
```

- **Exercice :** Utiliser [ngClass] et [ngStyle] pour ajouter un feedback visuel clair lorsque l'étudiant répond correctement ou incorrectement, avec des effets de style visibles (changement de couleur, opacité réduite des boutons déjà cliqués, etc.).



Mise en Pratique – Aspects visuels avec ngClass:

1. Ajouter un effet visuel pour indiquer au joueur que le bouton qu'il survole est activé, et appliquer des classes dynamiques basées sur les réponses correctes et incorrectes.
2. Utiliser [ngClass] pour changer la taille ou l'apparence des éléments du jeu, comme rendre les boutons plus grands lorsqu'ils sont cliqués.

Défi 3 :

Mettre en œuvre un système de « verrouillage » visuel qui modifie l'apparence des boutons après que l'étudiant a répondu, afin qu'il sache quelles options sont déjà choisies.

V. Validation des acquis

Question 1 : Fonctionnement de *ngIf

Quelle est la principale fonctionnalité de la directive `*ngIf` dans Angular ?

A) Itérer sur une liste d'éléments

B) Appliquer des styles conditionnels

C) Contrôler l'affichage conditionnel des éléments

D) Lier des données bidirectionnellement

Question 2 : Fonctionnement de *ngFor

Quelle directive Angular est utilisée pour itérer sur une collection et afficher chaque élément ?

A) *ngIf

B) *ngFor

C) [ngClass]

D) [ngStyle]

Question 3 : Application de *ngIf

Comment la directive `*ngIf` contribue-t-elle à la réactivité d'une application Angular ?

A) En permettant de modifier les styles d'un élément

B) En contrôlant dynamiquement l'affichage des éléments en fonction des données

C) En facilitant la liaison bidirectionnelle des données

D) En itérant sur des collections de données

Question 4 : Utilisation de [ngClass]

Comment `[ngClass]` améliore-t-il la gestion des styles dans une application Angular ?

A) En itérant sur des listes d'éléments

B) En appliquant des classes CSS de manière conditionnelle

C) En liant des données bidirectionnellement

D) En contrôlant l'affichage des éléments

Question 5 : Utilisation de [ngStyle]

Dans quel scénario préféreriez-vous utiliser `[ngStyle]` plutôt que `[ngClass]` ?

A) Lorsque vous souhaitez itérer sur une liste d'éléments

B) Lorsque vous avez besoin d'appliquer plusieurs classes conditionnellement

C) Lorsque vous devez appliquer des styles CSS spécifiques en fonction des données

D) Lorsque vous voulez contrôler l'affichage d'un élément

Question 6 : Combinaison de Directives

Quel est l'avantage principal de combiner `*ngIf` et `*ngFor` dans une application Angular ?

A) Améliorer la gestion des événements utilisateur

B) Contrôler l'affichage conditionnel tout en itérant sur des collections de données

C) Appliquer des styles dynamiques

D) Faciliter la liaison bidirectionnelle des données

Question 7 : Impact sur la Réactivité

Comment l'utilisation combinée de `*ngIf`, `*ngFor`, `[ngClass]` et `[ngStyle]` affecte-t-elle la réactivité de l'application ?

A) Elle ralentit l'application en augmentant le nombre de directives

B) Elle améliore la réactivité en permettant des mises à jour dynamiques et conditionnelles de l'interface utilisateur

C) Elle n'a aucun impact significatif sur la réactivité

D) Elle complique le code sans bénéfices réels

Question 8 : Gestion des Classes Dynamiques

Quelle directive permet d'appliquer une classe CSS spécifique à un élément uniquement si une condition est remplie ?

A) *ngIf

B) *ngFor

C) [ngClass]

D) [ngStyle]

Question 9 : Modification des Styles Dynamique

Quelle directive Angular vous permet de modifier dynamiquement les propriétés CSS telles que la couleur, la taille ou l'opacité d'un élément ?

A) *ngIf

B) *ngFor

C) [ngClass]

D) [ngStyle]

Question 10 : Création d'une Application Complète

Quelle approche est la plus appropriée pour intégrer efficacement les directives `*ngIf`, `*ngFor`, `[ngClass]` et `[ngStyle]` dans une application complète de gestion des étudiants afin d'améliorer l'interactivité et l'expérience utilisateur ?

A. Utiliser uniquement *ngFor pour toutes les itérations et ignorer les autres directives

B. Combiner *ngIf et *ngFor pour contrôler l'affichage et l'itération, tout en utilisant [ngClass] et [ngStyle] pour appliquer des styles conditionnels en fonction des données des étudiants

C. Appliquer des styles statiques dans les templates sans utiliser les directives Angular

D. Utiliser *ngIf pour tout le contrôle d'affichage sans combiner avec d'autres directives