Ateliers Angular

TP3 :Data Binding et Interpolation

TP 3 Data Binding et Interpolation

I. Objectif :

Maîtriser la data binding (liaison de données) et l'interpolation dans les composants Angular pour développer des applications web interactives et dynamiques.

Objectifs Spécifiques :

- Comprendre et Utiliser l'Interpolation et les Différents Types de Data Binding
- Gérer le Binding des Attributs, des Classes et des Styles :
- Communiquer Efficacement entre les Composants avec @Input et @Output :
- Utiliser les Pipes pour le Formatage des Données dans les Templates :

Ce TP comporte trois types d'activités :
Ateliers pratiques 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.
Développement d'un Projet : À partir de ce TP, on va commencer à développer un projet qui s'étendra sur les prochaines séances.

II. Les ateliers pratiques

Activité 1 : Interpolation de chaînes de caractères

1. Créer un composant "Bienvenue".
2. Ajouter une propriété message dans le composant, initialisée avec la valeur "Bienvenue sur notre site !".
3. Utiliser l'interpolation dans le template pour afficher le message.
4. Intégrer le composant dans l'application principale.

Activité 2 : Property Binding

1. Créer un composant "Produit".
2. Ajouter une propriété imageUrl dans le composant avec l'URL d'une image de produit.
3. Utiliser le property binding dans le template pour afficher l'image.
4. Intégrer le composant dans l'application principale.

Remarque Pour pouvoir afficher une image, il faut changer la configuration dans le fichier angular.json, la section suivante :

```
    assets": [
            {
              "glob": "**/*",
              "input": "src/assets/",
              "output": "/assets/"
            },
```

Activité 3 : Event Binding

1. Ajouter une méthode afficherAlerte() dans le composant "Produit" qui affiche une alerte avec le message "Produit ajouté au panier".
2. Ajouter un bouton dans le template avec un event binding qui déclenche afficherAlerte() lors du clic.
3. Vérifier l'interactivité en testant l'application.

Activité 4 : Two-Way Data Binding avec ngModel

1. Importer FormsModule dans app.module.ts et l'ajouter aux imports du module.
2. Créer un composant "Utilisateur".
3. Ajouter une propriété nom dans le composant, initialisée à une chaîne vide.
4. Utiliser ngModel dans le template pour lier un champ input à la propriété nom et afficher la valeur en temps réel.
5. Intégrer le composant dans l'application principale.

Activité 5 : Binding des Attributs et des Styles

1. Ajouter une propriété booléenne enStock dans le composant "Produit", initialisée à true.
2. Utiliser le class binding dans le template pour appliquer une classe CSS conditionnelle en fonction de enStock.
3. Définir la classe CSS correspondante dans le fichier de styles du composant.
4. Utiliser le style binding pour ajouter un style inline conditionnel.
5. Ajouter un bouton pour basculer la valeur de enStock et observer les changements de style.

Activité 6 : Communication entre Composants avec @Input et @Output

1. Créer un composant "Panier".
2. Utiliser @Output dans le composant "Produit" pour émettre un événement lors de l'ajout au panier.
3. Modifier le bouton dans le template du composant "Produit" pour émettre l'événement lors du clic.
4. Recevoir l'événement dans le composant parent et définir une méthode pour le gérer.

Activité 7 : Utilisation de @Input pour Passer des Données au Composant Enfant

1. Ajouter une propriété nomProduit avec @Input dans le composant "Produit".
2. Passer une valeur à nomProduit depuis le composant parent.
3. Utiliser la propriété nomProduit dans le template du composant "Produit" pour afficher le nom du produit.

Activité 8 : Binding avec les Pipes

1. Ajouter une propriété numérique prix dans le composant "Produit".
2. Utiliser les pipes pour afficher le prix formaté en devise dans le template.

III. Atelier guidé

On se propose dans cet atelier guidé de :

1. Faire lier des Attributs et des Styles

   a. Ajout de propriétés booléennes.
   b. Application conditionnelle de classes CSS et de styles inline.
   c. Interaction utilisateur pour modifier les états visuels.
2. Communiquer entre Composants avec @Input et @Output
   a. Création de composants et utilisation des décorateurs `@Input` et `@Output`.
   b. Gestion des événements entre composants parent et enfant.
3. Utilisation de @Input pour Passer des Données au Composant Enfant
   a. Passage de données du parent vers l'enfant.
   b. Utilisation des propriétés liées pour afficher des informations dynamiques.
4. Binding avec les Pipes
   a. Utilisation des pipes Angular pour formater les données affichées.
   b. Amélioration de la présentation des informations utilisateur.

1. Étape Préliminaire : Création du Projet Angular
2. Créer une Nouvelle Application Angular :

```
ng new gestion-produits
cd gestion-produits
ng serve -o
```

3. Ajouter une propriété booléenne `enStock` dans le composant "Produit", initialisée à `true`

```
// src/app/components/produit/produit.component.ts
import { Component } from '@angular/core';
@Component({
  selector: 'app-produit',
  templateUrl: './produit.component.html',
  styleUrls: ['./produit.component.css']
})
export class ProduitComponent {
  enStock: boolean = true;
}
```

4. Utiliser le class binding dans le template pour appliquer une classe CSS conditionnelle en fonction de `enStock`

```
<!-- src/app/components/produit/produit.component.html -->
<div [ngClass]="{'en-stock': enStock, 'hors-stock': !enStock}">
  <h3>Nom du Produit</h3>
  <p>Description du produit...</p>
</div>
```

5. Définir la classe CSS correspondante dans le fichier de styles du composant

```
/* src/app/components/produit/produit.component.css */
```

```
.en-stock {
  border: 2px solid green;
  background-color: #e0ffe0;
}

.hors-stock {
  border: 2px solid red;
  background-color: #ffe0e0;
}
```

6. Utiliser le style binding pour ajouter un style inline conditionnel

```
<!-- src/app/components/produit/produit.component.html -->
<div [ngClass]="{'en-stock': enStock, 'hors-stock': !enStock}"
     [ngStyle]="{'opacity': enStock ? '1' : '0.5'}">
  <h3>Nom du Produit</h3>
  <p>Description du produit...</p>
</div>
```

7. Ajouter un bouton pour basculer la valeur de `enStock` et observer les changements de style

```
// src/app/components/produit/produit.component.ts
export class ProduitComponent {
  enStock: boolean = true;
  toggleStock(): void {
    this.enStock = !this.enStock;
  }
}
```

8. Communication entre Composants avec @Input et @Output

Utilisez Angular CLI pour générer un nouveau composant nommé `Panier`.

```
ng generate component components/panier
```

9. Utiliser `@Output` dans le composant "Produit" pour émettre un événement lors de l'ajout au panier

```
// src/app/components/produit/produit.component.ts
import { Component, Output, EventEmitter } from '@angular/core';

@Component({
  selector: 'app-produit',
  templateUrl: './produit.component.html',
  styleUrls: ['./produit.component.css']
})
export class ProduitComponent {
  @Output() ajouterAuPanier = new EventEmitter<string>();
  nomProduit: string = 'Produit Exemple';
  enStock: boolean = true;
```

```
  toggleStock(): void {
    this.enStock = !this.enStock;
  }

  ajouterProduit(): void {
    this.ajouterAuPanier.emit(this.nomProduit);
  }
}
```

10. Modifier le bouton dans le template du composant "Produit" pour émettre l'événement lors du clic

```
<!-- src/app/components/produit/produit.component.html -->
<div [ngClass]="{'en-stock': enStock, 'hors-stock': !enStock}"
     [ngStyle]="{'opacity': enStock ? '1' : '0.5'}">
  <h3>{{ nomProduit }}</h3>
  <p>Description du produit...</p>
</div>
<button (click)="toggleStock()">
  {{ enStock ? 'Marquer comme Hors Stock' : 'Marquer comme En Stock' }}
</button>
<button (click)="ajouterProduit()" [disabled]="!enStock">
  Ajouter au Panier
</button>
```

11. Recevoir l'événement dans le composant parent et définir une méthode pour le gérer

```
<!-- src/app/app.component.html -->
<app-produit
  [nomProduit]="'Ordinateur Portable'"
  (ajouterAuPanier)="gererAjoutAuPanier($event)">
</app-produit>
<app-panier [items]="panierItems"></app-panier>
```

```
// src/app/app.component.ts
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  panierItems: string[] = [];

  gererAjoutAuPanier(nomProduit: string): void {
    this.panierItems.push(nomProduit);
```

```
    console.log(`${nomProduit} ajouté au panier.`);
  }
}
```

```
// src/app/components/panier/panier.component.ts
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-panier',
  templateUrl: './panier.component.html',
  styleUrls: ['./panier.component.css']
})
export class PanierComponent {
  @Input() items: string[] = [];
}
<!-- src/app/components/panier/panier.component.html -->
<h2>Panier</h2>
<ul>
  <li *ngFor="let item of items">{{ item }}</li>
</ul>
```

12. Utilisation de @Input pour Passer des Données au Composant Enfant

Ajouter une propriété `nomProduit` avec `@Input` dans le composant "Produit"

```
// src/app/components/produit/produit.component.ts
import { Component, Input, Output, EventEmitter } from '@angular/core';
@Component({
  selector: 'app-produit',
  templateUrl: './produit.component.html',
  styleUrls: ['./produit.component.css']
})
export class ProduitComponent {
  @Input() nomProduit: string = 'Produit Par Défaut';
  @Output() ajouterAuPanier = new EventEmitter<string>();
  enStock: boolean = true;
  toggleStock(): void {
    this.enStock = !this.enStock;
  }
  ajouterProduit(): void {
    this.ajouterAuPanier.emit(this.nomProduit);
  }
}
```

13. Passer une valeur à `nomProduit` depuis le composant parent

<!-- src/app/app.component.html -->

```
<app-produit
  [nomProduit]="'Ordinateur Portable'"
  (ajouterAuPanier)="gererAjoutAuPanier($event)">
</app-produit>
<app-panier [items]="panierItems"></app-panier>
```

14. Utiliser la propriété `nomProduit` dans le template du composant "Produit" pour afficher le nom du produit

<!-- src/app/components/produit/produit.component.html -->

```
<div [ngClass]="{'en-stock': enStock, 'hors-stock': !enStock}"
     [ngStyle]="{'opacity': enStock ? '1' : '0.5'}">
  <h3>{{ nomProduit }}</h3>
  <p>Description du produit...</p>
</div>
<button (click)="toggleStock()">
  {{ enStock ? 'Marquer comme Hors Stock' : 'Marquer comme En Stock' }}
</button>
<button (click)="ajouterProduit()" [disabled]="!enStock">
  Ajouter au Panier
</button>
```

IV. Projet : Cahier des Charges du Projet de Jeu de Quiz

Développer une application web interactive de type quiz où les utilisateurs peuvent répondre à des questions, accumuler des scores, et éventuellement comparer leurs performances.

Le joueur doit répondre à une série de questions. Chaque question a plusieurs options, il doit choisir la bonne réponse. Si la réponse est correcte, le joueur gagne des points ; sinon, il en perd. Les réponses sont capturées via des boutons ou des champs de saisie, et les données sont liées dynamiquement au modèle.

Fonctionnalités Principales (à ce stade) :

1. **Gestion des Questions :**
   - Afficher des questions à choix multiples aux utilisateurs.
   - Les questions peuvent être de différents types : texte, images, ou audio.
   - Implémenter une base de données de questions avec leurs réponses correctes et distracteurs.
2. **Système de Scoring :**
   - Attribuer des points pour chaque réponse correcte.
   - Déduire des points pour les réponses incorrectes (optionnel).
   - Afficher le score actuel après chaque question.

Cette séance, nous allons implémenter la fonctionnalité d'affichage et de gestion des questions du jeu, avec la possibilité pour le joueur de répondre aux questions. Nous allons exploiter le **data binding** pour afficher dynamiquement les données des questions et gérer les réponses via l'**event binding**.

1. Utiliser l'interpolation et le property binding pour afficher dynamiquement les questions.
2. Gérer les événements utilisateurs (réponses aux questions) avec l'event binding.
3. Implémenter le **two-way data binding** pour capturer les réponses des joueurs.

Contenu de cette partie :

1. Nous allons créer les composants principaux suivants :
   - HomeComponent : Composant d'accueil du quiz.
   - GameComponent : Composant qui gère le déroulement du quiz.
   - QuestionComponent : Composant pour afficher une question individuelle.
   - ScoreCompenent : Composant pour afficher le score du joueur.
2. **Étape 1 :** Ajout de Questions dans le Jeu :

Créer un tableau d'objets représentant les questions du jeu avec des propriétés comme question, options (pour les choix multiples), et reponse.

Dans GameComponent, ajouter un tableau de questions :

```
questions = [
    { question: 'Quel est le plus grand océan du monde ?',
      options: ['Pacifique', 'Atlantique', 'Indien', 'Arctique'],
      reponse: 'Pacifique'
    },
    { question: 'Quelle est la capitale de l Algerie ?',
      options: ['Alger', 'Tunis', 'Tanja'], reponse: 'Paris' },
    { question: 'Quelle est la couleur du ciel ?',
      options: ['Bleu', 'Vert', 'Rouge'], reponse: 'Bleu' }
  ];
```

3. Créer une interface dans le template de GameComponent pour afficher chaque question et ses options sous forme de boutons.

4. Interpolation et Property Binding pour l'Affichage des Questions :

Utilisez l'interpolation pour afficher le texte des questions dans le template et le property binding pour les options (boutons de réponses). Utilisez *ngFor pour itérer sur la liste des questions et afficher chaque question avec ses options.

5. Implémenter le système de choix des réponses via des boutons.

Utiliser l'Event Binding pour Gérer les Réponses du Joueur :

Utiliser l'**event binding** pour capter les clics des joueurs sur les boutons de réponse.
Ajouter une méthode onSelectOption(option, question) dans GameComponent pour traiter la

6. Modifier game.component.html pour gérer l'état des boutons :

Gérer la mise à jour du score dans le composant parent en fonction des réponses correctes ou incorrectes. **Two-Way Data Binding pour Capturer la Réponse :**

Introduire le **two-way data binding** pour capturer et afficher les réponses sélectionnées par l'étudiant dans un champ de texte (ou un autre contrôle).

Ajouter un champ de saisie pour permettre au joueur d'entrer une réponse directe :

Mise en Pratique - Développement du Jeu :

Scénario du Jeu :

L'étudiant doit répondre à une série de questions. Chaque question a plusieurs options, et l'étudiant doit choisir la bonne réponse. Si la réponse est correcte, l'étudiant gagne des points ; sinon, il en perd. Les réponses sont capturées via des boutons ou des champs de saisie selon le type de question, et les données sont liées dynamiquement au modèle.

A. **Défi 1 :** Ajouter une fonctionnalité qui bloque les réponses après la sélection, afin que l'étudiant ne puisse plus répondre deux fois à la même question.
B. **Défi 2 :** Implémenter une interface qui affiche en temps réel le nombre de bonnes réponses et de mauvaises réponses, en plus du score.
C. **Défi 3 :** Ajouter un chronomètre pour chaque question, avec un compte à rebours. Si l'étudiant ne répond pas dans le temps imparti, la réponse est automatiquement comptée comme incorrecte.

V. Validation des acquis

QCM 1 : Interpolation de Données

1. Quelle syntaxe est utilisée pour afficher une propriété du composant dans le template en utilisant l'interpolation ?

   a) [property]="value"

   b) {{ value }}

   c) (event)="handler()"

   d) *ngIf="condition"

QCM 2 : Property Binding

2. Lequel des éléments suivants est un exemple correct de property binding en Angular ?

   a) <img src="{{ imageUrl }}" alt="Image">

   b) <img [src]="imageUrl" alt="Image">

c) <img (src)="imageUrl" alt="Image">

d) <img *src="imageUrl" alt="Image">

QCM 3 : Event Binding

3. Quelle est la syntaxe correcte pour lier un événement de clic sur un bouton à une méthode du composant ?

   a) <button [click]="maMethode()">Cliquez-moi</button>

   b) <button (click)="maMethode()">Cliquez-moi</button>

   c) <button *click="maMethode()">Cliquez-moi</button>

   d) <button {{ click="maMethode()" }}>Cliquez-moi</button>

QCM 4 : Two-Way Data Binding avec ngModel

4. Pour utiliser la liaison de données bidirectionnelle avec ngModel, quelle syntaxe devez-vous employer dans le template ?

   a) <input [ngModel]="nom">

   b) <input [(ngModel)]="nom">

   c) <input {{ ngModel="nom" }}>

   d) <input (ngModel)="nom">

QCM 5 : Importation de FormsModule

5. Pour que ngModel fonctionne correctement, quel module devez-vous importer dans votre module principal (app.module.ts) ?

   a) BrowserModule

   b) HttpClientModule

   c) FormsModule

   d) ReactiveFormsModule

QCM 6 : Utilisation des Pipes

6. Quel est le rôle des pipes dans Angular ?

   a) Ils permettent de lier les propriétés du composant aux attributs des éléments HTML.

   b) Ils sont utilisés pour transformer et formater les données directement dans le template.

   c) Ils gèrent les interactions utilisateur en appelant des méthodes du composant.

   d) Ils permettent de répéter une partie du template pour chaque élément d'une collection.

Question Ouverte

Veuillez répondre de manière détaillée à la question suivante pour démontrer votre compréhension approfondie du data binding et de l'interpolation dans Angular.