

Bienvenue dans le tutoriel Angular

DERMANE Fad

February 2025

Contents

1	Installation d'Angular	5
1.1	Prérequis	5
1.2	Nodejs	5
1.3	vérifier la version	5
1.4	NPM	5
1.5	vérifier la version	5
1.6	Un éditeur de code	5
1.7	Installation de Node.js	6
1.8	Installation d'Angular CLI	7
1.9	Désinstaller Angular CLI	7
1.10	Installer Angular CLI globalement	7
1.11	Vérifier l'installation	7
2	Création d'un Projet Angular	8
2.1	Générer un projet appelé first-app avec choix manuel	8
2.2	Résultat de la commande <code>ng new first-app</code>	8
2.3	Se positionner dans le projet	8
2.4	Exécuter le projet	8
2.5	Exécuter et lancer automatiquement l'application dans le navigateur	9
2.6	Résultat	9
3	Structure d'un Projet Angular	10
4	Implémentation de la syntaxe Angular	11
4.1	Data Binding (Liaison de données)	11
4.2	1. Interpolation et les pipe	11
4.3	2. Property Binding (<code>[]</code>)	11
4.4	3. Event Binding (<code>()</code>)	11
4.5	4. Two-Way Binding (<code>[]()</code>)	12
4.6	Directives Angular	12
4.7	1. <code>*ngIf</code>	12
4.8	2. <code>*ngFor</code>	12
5	RxJS	12
5.1	Observable et Abonnement	13
6	Les composant	14
6.1	Définition:	14
6.2	Composant par défaut (<code>AppComponent</code>)	14
6.3	Template du composant (<code>app.component.html</code>)	14
6.4	Styles du composant (<code>app.component.css</code>)	15
6.5	Résultat	18

7	Faire un petit projet crud d'une table livre	19
7.1	Modèle	19
7.2	Interface Livre	19
8	Service	20
8.1	LivreService	20
8.1.1	Modules utilisés	20
8.1.2	Méthodes	20
9	Composant	22
9.1	LivreComponent	22
9.1.1	Modules utilisés	22
9.1.2	Méthodes	22
10	Template HTML	24
10.1	livre.component.html	24
11	Styles CSS	25
11.1	livre.component.css	25
12	projet	26
12.1	Diagramme de classe	26

Introduction

Définition d'Angular

Angular est un framework open-source développé par Google pour créer des applications web dynamiques et interactives en utilisant des langages tels que **TypeScript**, **HTML** et **CSS**.

Il est principalement utilisé pour le développement **frontend**, c'est-à-dire la partie cliente d'une application web.

Rôle d'Angular dans le développement web

Frontend

Angular gère tout ce qui concerne l'interface utilisateur (UI). Il permet de créer des composants réutilisables, de gérer les données dynamiquement.

Backend

Angular ne gère pas directement le backend (la logique serveur, les bases de données, etc.). Cependant, il peut interagir avec un backend via des API (par exemple, en utilisant `HttpClient` pour envoyer des requêtes HTTP (GET, POST, PUT, DELETE)).

Historique d'Angular

AngularJS

Créé en 2010.

Utilise JavaScript et le pattern MVC.

Problèmes de performance sur les grandes applications.

Angular2 (2016)

Réécrit entièrement avec TypeScript.

Utilise le pattern MVVM.

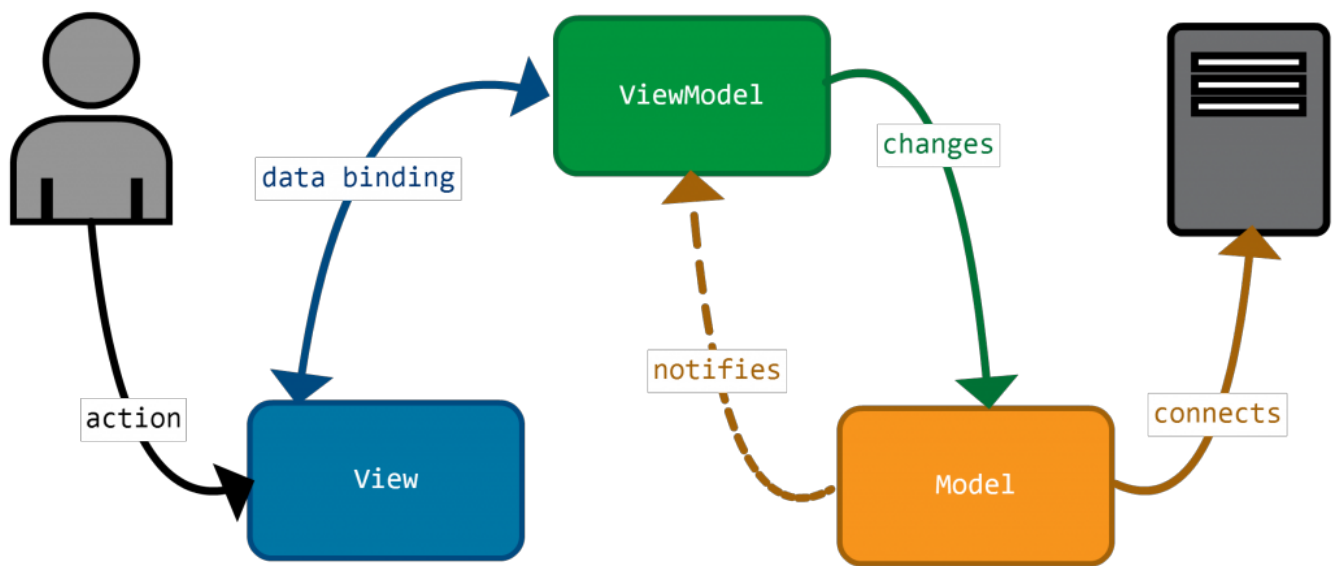
Introduction du concept de composants à la place des contrôleurs. Meilleure gestion des performances et modularité (permet d'organiser et structurer l'application).

Versions suivantes (Angular4 à 18)

Angular CLI : Outil pour générer, tester et déployer plus facilement les projets.

Architecture MVVM

Le MVVM (Model-View-ViewModel) est un modèle architectural couramment utilisé dans le développement d'applications, notamment avec des frameworks comme Angular.



La Vue reçoit toujours les actions de l'utilisateur et interagit seulement avec le ViewModel. Le Modèle communique avec le serveur et notifie le ViewModel de son changement.

Le ViewModel s'occupe de : présenter les données du Model à la Vue, recevoir les changements de données de la Vue, demander au Model de se modifier

1 Installation d'Angular

1.1 Prérequis

Pour commencer, vous devez installer Node.js et npm (Node Package Manager). Ensuite, installez Angular CLI (Command Line Interface) globalement sur votre machine:

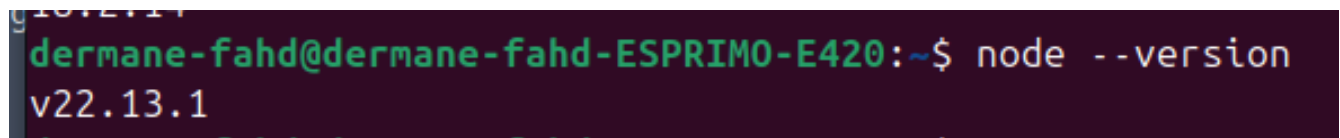
1.2 Nodejs

Nodejs est un environnement d'exécution JavaScript côté serveur. **Nodejs** (version 18 ou supérieure)

1.3 vérifier la version

Pour vérifier la version de Node.js installée sur votre machine, utilisez la commande suivante dans votre terminal:

```
node --version
```



```
dermane-fahd@dermane-fahd-ESPRIMO-E420:~$ node --version
v22.13.1
```

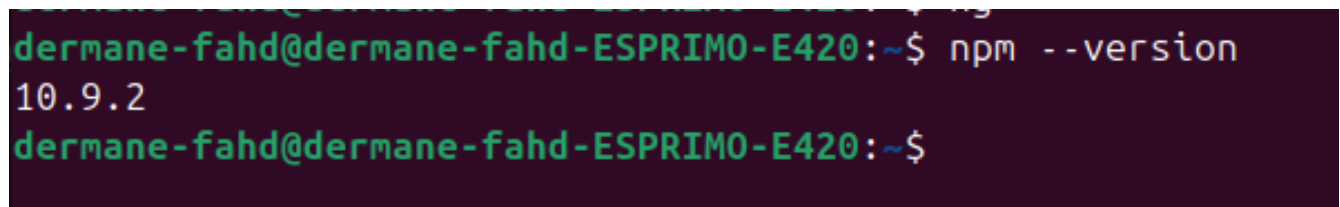
1.4 NPM

NPM (Node Package Manager) est un gestionnaire de paquets pour Node.js. **NB** : Le programme NPM est installé sur votre ordinateur lorsque vous installez Node.js.

1.5 vérifier la version

Pour vérifier la version de NPM installée sur votre machine, utilisez la commande suivante dans votre terminal:

```
npm --version
```



```
dermane-fahd@dermane-fahd-ESPRIMO-E420:~$ npm --version
10.9.2
dermane-fahd@dermane-fahd-ESPRIMO-E420:~$
```

1.6 Un éditeur de code

Un éditeur de code est nécessaire pour développer des applications Angular. Téléchargez Visual Studio Code (VS Code) depuis le lien suivant : <https://code.visualstudio.com/>.



1.7 Installation de Node.js

Si **Node.js** et **npm** ne sont pas installés sur votre machine, téléchargez et installez la version 18 ou supérieure depuis le site officiel : <https://nodejs.org/>.



1.8 Installation d'Angular CLI

Angular CLI: (Command Line Interface) est un outil indispensable pour créer, configurer et gérer des projets Angular. Il fournit des commandes comme **ng new** pour générer un nouveau projet ou **ng serve** pour démarrer un serveur de développement.



1.9 Désinstaller Angular CLI

Si une ancienne version d'Angular CLI est déjà installée, il est recommandé de la désinstaller avant de procéder à une nouvelle installation. Utilisez la commande suivante dans votre terminal :

```
npm uninstall -g @angular/cli
```

1.10 Installer Angular CLI globalement

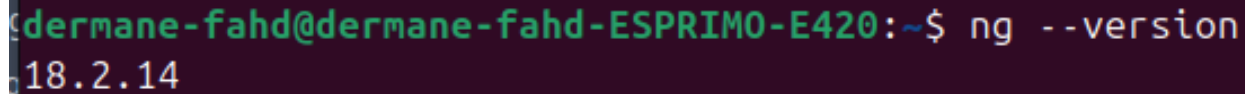
Pour installer **Angular CLI** globalement sur votre machine, utilisez la commande suivante dans votre terminal :

```
npm install -g @angular/cli@18
```

1.11 Vérifier l'installation

Pour vérifier que **Angular CLI** a été correctement installé et afficher la version installée, exécutez la commande suivante :

```
ng --version
```



```
dermane-fahd@dermane-fahd-ESPRIMO-E420:~$ ng --version
18.2.14
```

2 Création d'un Projet Angular

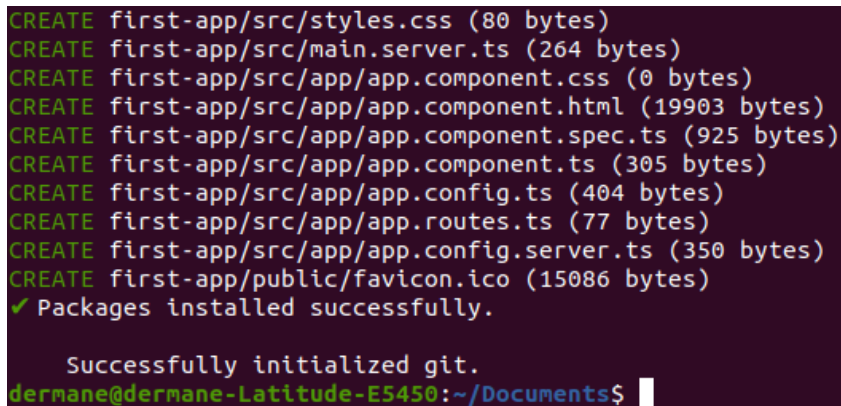
2.1 Générer un projet appelé first-app avec choix manuel

Pour créer un nouveau projet Angular, utilisez la commande suivante dans votre terminal :

```
ng new first-app
```

2.2 Résultat de la commande ng new first-app

Voici une capture d'écran montrant l'exécution de la commande `ng new first-app` dans le terminal :



```
CREATE first-app/src/styles.css (80 bytes)
CREATE first-app/src/main.server.ts (264 bytes)
CREATE first-app/src/app/app.component.css (0 bytes)
CREATE first-app/src/app/app.component.html (19903 bytes)
CREATE first-app/src/app/app.component.spec.ts (925 bytes)
CREATE first-app/src/app/app.component.ts (305 bytes)
CREATE first-app/src/app/app.config.ts (404 bytes)
CREATE first-app/src/app/app.routes.ts (77 bytes)
CREATE first-app/src/app/app.config.server.ts (350 bytes)
CREATE first-app/public/favicon.ico (15086 bytes)
✓ Packages installed successfully.

Successfully initialized git.
dermane@dermane-Latitude-E5450:~/Documents$
```

2.3 Se positionner dans le projet

Une fois le projet créé, accédez au dossier du projet avec la commande suivante :

```
cd first-app
```

2.4 Exécuter le projet

Pour démarrer le serveur de développement et exécuter l'application, utilisez la commande suivante :

```
ng serve
```

ou en version raccourcie :

```
ng s
```

L'application sera accessible à l'adresse : <http://localhost:4200/>.

2.5 Exécuter et lancer automatiquement l'application dans le navigateur

Pour démarrer le serveur de développement et ouvrir automatiquement l'application dans votre navigateur par défaut, utilisez la commande suivante :

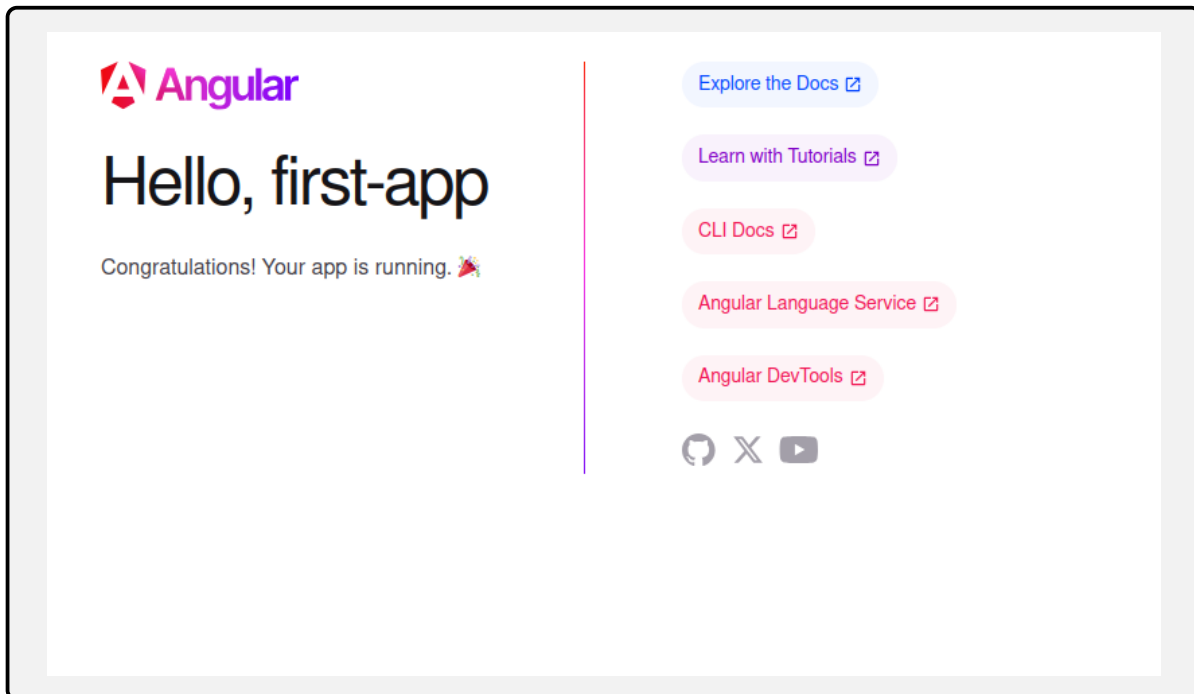
```
ng serve --open
```

ou en version raccourcie :

```
ng s -o
```

2.6 Résultat

Voici une capture d'écran de la page d'accueil de l'application Angular :



3 Structure d'un Projet Angular

Lors de la création d'un projet Angular avec la commande **ng new**, plusieurs dossiers et fichiers sont générés par défaut. Voici les dossiers et fichiers principaux générés :

- **public/** : Contient des fichiers statiques comme des images, des polices ou d'autres ressources.
- **src/** : Contient tout le code source de l'application.
 - **src/app/** : Contient les composants, services et modules spécifiques à l'application Angular.
 - **angular.json** : Contient la configuration principale du projet Angular.
 - **package.json** : Gère les dépendances npm nécessaires au projet Angular.
 - **tsconfig.json** : Contient la configuration du compilateur TypeScript pour l'application.
 - **tsconfig.app.json** : Spécifie les options de compilation pour l'application elle-même.
 - **tsconfig.spec.json** : Spécifie les options de compilation pour les tests unitaires.
 - **main.ts** : Initialise l'application Angular et démarre le module racine.
 - **index.html** : Charge le composant racine `<app-root></app-root>` et sert de point d'entrée pour le navigateur.

4 Implémentation de la syntaxe Angular

4.1 Data Binding (Liaison de données)

Le **data binding** permet de lier des propriétés ou des méthodes du composant à des éléments dans le template HTML. Voici les types de data binding disponibles :

4.2 1. Interpolation et les pipe

L'**interpolation** est utilisée pour afficher une valeur d'une propriété dans le template HTML. Par exemple, pour afficher un message.

Les **pipes** sont utilisés pour appliquer des transformations sur les valeurs affichées, comme formater une date, convertir un texte en majuscules, ou encore afficher des nombres sous forme de pourcentages.

```
<p>{{ message }}</p>

<p>{{ message | uppercase }}</p>

<!-- Affiche: 25% -->
<p>{{ taux | percent }}</p>

<!-- Affiche: 5 000,00 -->
<p>{{ montant | currency:currencyCode }}</p>

<p [ngStyle]="{ 'font-size.px': fontSize, 'color': 'blue' }">
  Texte avec taille dynamique
</p>
```

Dans le composant, définissez la propriété `message` :

```
export class AppComponent {
  message = 'Bonjour Angular !';

  taux: number = 0.25;
  montant: number = 5000;
  currencyCode: string = 'EUR';
  fontSize: number = 20;
}
```

4.3 2. Property Binding ([])

Le **property binding** permet de lier une propriété d'un élément HTML à une propriété d'un composant. Par exemple, pour lier l'attribut `src` d'une image :

```
<img [src]="imageUrl" alt="Image">
```

Dans le composant, définissez la propriété `imageUrl` :

```
export class AppComponent {
  imageUrl = 'https://via.placeholder.com/150';
}
```

4.4 3. Event Binding (())

Le **event binding** permet de lier un événement (comme un clic) à une méthode du composant. Par exemple, pour gérer un clic sur un bouton :

```
<button (click)="onClick()">Cliquez ici</button>
```

Dans le composant, définissez la méthode `onClick` :

```
export class AppComponent {
  onClick() {
    alert('Bouton cliqué !');
  }
}
```

4.5 4. Two-Way Binding ([()])

Le **two-way binding** permet de lier une propriété à un élément HTML tout en écoutant les changements dans l'élément et en les répercutant dans la propriété du composant. Par exemple, pour lier un champ de saisie à une propriété :

```
<input [(ngModel)]="username">
<p>Vous avez saisi : {{ username }}</p>
```

Dans le composant, définissez la propriété `username` :

```
export class AppComponent {
  username = '';
}
```

Remarque : Pour utiliser `ngModel`, vous devez importer `FormsModule` dans votre module.

4.6 Directives Angular

Les directives sont des éléments qui ajoutent de la logique au DOM, comme des boucles ou des conditions.

4.7 1. *ngIf

La directive `*ngIf` affiche un élément si une condition est vraie. Par exemple, pour afficher un message conditionnellement :

```
<div *ngIf="isVisible">Cet élément est visible</div>
```

Dans le composant, définissez la propriété `isVisible` :

```
export class AppComponent {
  isVisible = true;
}
```

4.8 2. *ngFor

La directive `*ngFor` permet de boucler sur une liste d'éléments. Par exemple, pour afficher une liste de noms :

```
<ul>
  <li *ngFor="let name of names">{{ name }}</li>
</ul>
```

Dans le composant, définissez la propriété `names` :

```
export class AppComponent {
  names = ['tamba', 'chabane', 'sarata'];
}
```

5 RxJS

RxJS est une bibliothèque qui facilite la gestion des flux de données asynchrones et des événements dans une application.

5.1 Observable et Abonnement

Un **Observable** est un flux de données qui peut émettre des valeurs (comme des nombres, des objets, des événements utilisateur, etc.) au fil du temps. L'abonnement se fait via la méthode `subscribe()` de l'observable. Une fois que vous vous abonnez à un observable, vous devenez un abonné qui reçoit les valeurs émises par cet observable. Les abonnés peuvent être des fonctions qui réagissent aux événements ou aux données émises par l'observable (par exemple, afficher ces données dans l'interface utilisateur, les transformer, les enregistrer, etc.).

Explication

- **Création d'un Observable** : Un observable est une fonction qui génère des valeurs au fil du temps. Par exemple, un observable peut émettre des valeurs à chaque fois qu'un utilisateur clique sur un bouton, ou à chaque réponse d'une requête HTTP.
- **Souscription à un Observable** : Pour recevoir les valeurs que l'Observable émet, vous devez vous abonner à l'Observable en utilisant la méthode `subscribe()`. Cela permet à l'abonné de recevoir les données ou événements émis par l'Observable.
- **Réception des valeurs** : Une fois que vous vous abonnez à l'Observable, il commence à vous envoyer des valeurs à travers des méthodes définies dans l'abonnement. Ces méthodes sont généralement :
 - `next(value)` : appelée chaque fois qu'une nouvelle valeur est émise par l'Observable.
 - `error(err)` : appelée si une erreur se produit dans l'Observable.
 - `complete()` : appelée lorsque l'Observable a fini d'émettre des valeurs.

Exemple d'utilisation

```
import { Observable } from 'rxjs';

// Cr ation de l'Observable
observable$ = new Observable((observer) => {
  observer.next('Hello');
  observer.next('World');
  observer.next(2);
  observer.next(3);
  observer.complete();
});

// S'abonner à l'Observable
subscription:Subscription = this.observable$.subscribe((valeur) => {
  console.log(valeur); // Affiche chaque valeur mise
});
```

6 Les composant

6.1 Définition:

En Angular, un **composant** est une classe TypeScript associée à un template HTML et à des styles CSS. Il représente une partie de l'interface utilisateur (UI) et est utilisé pour structurer l'application en éléments réutilisables

6.2 Composant par défaut (AppComponent)

Voici un exemple de composant Angular par défaut généré par Angular CLI :

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

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterOutlet, NgFor, NgIf, FormsModule],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'first-app';

  message: string = "hello word";
  taux: number = 0.25;
  montant: number = 5000;
  currencyCode: string = 'EUR';
  fontSize: number = 20;

  images = "./images/1.png";

  names: string[] = ['Alice', 'Bob', 'Charlie'];

  isVisible = true;

  username = '';

  onClick(): void {
    alert('Button clicked');
    console.log('Button clicked');
  }
}
```

6.3 Template du composant (app.component.html)

Le template du composant par défaut peut être modifié pour inclure des exemples de data binding et de directives :

```
<h1>{{ title }}</h1>

<!-- Exemple d'interpolation -->
<p>{{ message }}</p>

<!-- Affiche: 25% -->
<p>{{ taux | percent }}</p>

<!-- Affiche: 5 000,00 -->
<p>{{ montant | currency:currencyCode }}</p>

<p [ngStyle]="{ 'font-size.px': fontSize, 'color': 'blue' }">
  Texte avec taille dynamique
</p>

<!-- Exemple de property binding -->
<img [src]="imageUrl" alt="Image">
```

```

<!-- Exemple d'event binding -->
<button (click)="onClick()">Cliquez ici</button>

<!-- Exemple de two-way binding -->
<input [(ngModel)]="username">
<p>Vous avez saisi : {{ username }}</p>

<!-- Exemple de *ngIf -->
<div *ngIf="isVisible">Cet lment est visible</div>

<!-- Exemple de *ngFor -->
<ul>
  <li *ngFor="let name of names">{{ name }}</li>
</ul>

```

6.4 Styles du composant (app.component.css)

Vous pouvez ajouter des styles personnalisés pour le composant :

```

/* Style pour le titre <h1> */
h1 {
  color: #2c3e50; /* Couleur du texte */
  font-family: 'Arial', sans-serif; /* Police */
  text-align: center; /* Centrer le texte */
  margin-top: 20px; /* Espacement en haut */
}

/* Style pour l'image */
img {
  display: block; /* Pour centrer l'image */
  margin: 0 auto; /* Centrer horizontalement */
  border-radius: 10px; /* Coins arrondis */
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2); /* Ombre */
}

/* Style pour le bouton */
button {
  display: block; /* Pour centrer le bouton */
  margin: 20px auto; /* Centrer horizontalement avec espacement */
  padding: 10px 20px; /* Espacement interne */
  background-color: #3498db; /* Couleur de fond */
  color: white; /* Couleur du texte */
  border: none; /* Supprimer la bordure */
  border-radius: 5px; /* Coins arrondis */
  cursor: pointer; /* Curseur en forme de main */
  font-size: 16px; /* Taille de la police */
  transition: background-color 0.3s ease; /* Animation au survol */
}

/* Effet au survol du bouton */
button:hover {
  background-color: #2980b9; /* Changement de couleur */
}

/* Style pour l'input */
input {
  display: block; /* Pour centrer l'input */
  margin: 20px auto; /* Centrer horizontalement avec espacement */
  padding: 10px; /* Espacement interne */
  width: 80%; /* Largeur de l'input */
  max-width: 300px; /* Largeur maximale */
  border: 1px solid #ccc; /* Bordure */
  border-radius: 5px; /* Coins arrondis */
  font-size: 16px; /* Taille de la police */
}

/* Style pour le paragraphe sous l'input */
p {
  text-align: center; /* Centrer le texte */
}

```

```

    color: #2c3e50; /* Couleur du texte */
    font-size: 18px; /* Taille de la police */
}

/* Style pour la div conditionnelle (*ngIf) */
div {
    text-align: center; /* Centrer le texte */
    margin: 20px auto; /* Espacement */
    padding: 10px; /* Espacement interne */
    background-color: #ecf0f1; /* Couleur de fond */
    border-radius: 5px; /* Coins arrondis */
    width: 80%; /* Largeur */
    max-width: 400px; /* Largeur maximale */
}

/* Style pour la liste (*ngFor) */
ul {
    list-style-type: none; /* Supprimer les puces */
    padding: 0; /* Supprimer le padding par défaut */
    text-align: center; /* Centrer la liste */
}

ul li {
    background-color: #3498db; /* Couleur de fond */
    color: white; /* Couleur du texte */
    margin: 5px auto; /* Espacement entre les éléments */
    padding: 10px; /* Espacement interne */
    border-radius: 5px; /* Coins arrondis */
    width: 80%; /* Largeur */
    max-width: 300px; /* Largeur maximale */
}

/* Style pour la navigation */
nav {
    text-align: center; /* Centrer le texte */
    margin: 20px auto; /* Espacement */
}

nav a {
    color: #3498db; /* Couleur du texte */
    text-decoration: none; /* Supprimer le soulignement */
    font-size: 18px; /* Taille de la police */
    padding: 10px 20px; /* Espacement interne */
    border: 1px solid #3498db; /* Bordure */
    border-radius: 5px; /* Coins arrondis */
    transition: background-color 0.3s ease, color 0.3s ease; /* Animation au survol */
}

/* Effet au survol du lien */
nav a:hover {
    background-color: #3498db; /* Changement de couleur de fond */
    color: white; /* Changement de couleur du texte */
}

/* Style pour le router-outlet */
router-outlet {
    display: block; /* Pour éviter les problèmes de mise en page */
    margin-top: 20px; /* Espacement en haut */
}

```


6.5 Résultat

hello word

HELLO WORD

25%

€5,000.00

Texte avec taille dynamique



Cliquez ici

Vous avez saisi :

Cet élément est visible

tamba

chabane

sarata

7 Faire un petit projet crud d'une table livre

7.1 Modèle

7.2 Interface Livre

Le modèle de données est défini par l'interface **Livre**. Cette interface décrit la structure d'un livre avec les propriétés suivantes :

- **id**: Identifiant unique du livre.
- **titre**: Titre du livre.
- **auteur**: Auteur du livre.
- **disponible**: Statut de disponibilité du livre.

Pour créer une interface avec Angular CLI, vous pouvez utiliser la commande suivante : `ng generate interface models/livre`.

```
export interface Livre {  
  id: number;  
  titre: string;  
  auteur: string;  
  disponible: boolean;  
}
```

8 Service

Un service en Angular : est une classe qui contient des logiques ou des fonctionnalités réutilisables, comme la gestion des données ou des appels API. Il permet de partager des informations et des actions entre plusieurs composants dans une application.

8.1 LivreService

Le service `LivreService` est responsable de la gestion des opérations CRUD (Create, Read, Update, Delete) sur les livres. Il utilise l'interface `Livre` pour manipuler les données.

Pour créer un service en Angular, vous pouvez utiliser la commande suivante : `ng generate service nom-du-service`, ou bien sa version abrégée `ng g s nom-du-service`.

De même, pour créer un composant, utilisez la commande `ng generate component nom-du-composant`, ou l'abréviation `ng g c nom-du-composant`.

8.1.1 Modules utilisés

- `Injectable`: Pour marquer le service comme injectable.
- `Livre`: Interface du modèle de livre.

8.1.2 Méthodes

- `getLivres()`: Retourne la liste des livres.
- `getLivreById(id: number)`: Retourne un livre par son ID.
- `addLivre(livre: Livre)`: Ajoute un nouveau livre.
- `updateLivre(id: number, updatedLivre: Partial<Livre>)`: Met à jour un livre existant.
- `deleteLivre(id: number)`: Supprime un livre par son ID.

```
import { Injectable } from '@angular/core';
import { Livre } from '../models/livre.model';

@Injectable({
  providedIn: 'root',
})
export class LivreService {
  private livres: Livre[] = [
    { id: 1, titre: 'Le Petit Prince', auteur: 'Antoine de Saint-Exupéry', disponible: true },
    { id: 2, titre: '1984', auteur: 'George Orwell', disponible: false },
    { id: 3, titre: 'Harry Potter 1\'' cole des sorciers', auteur: 'J.K. Rowling', disponible: true },
  ];

  constructor() {}

  getLivres(): Livre[] {
    return this.livres;
  }

  getLivreById(id: number): Livre | undefined {
    return this.livres.find((livre) => livre.id === id);
  }

  addLivre(livre: Livre): void {
    this.livres.push(livre);
  }

  updateLivre(id: number, updatedLivre: Partial<Livre>): void {
    const livre = this.getLivreById(id);
    if (livre) {
      Object.assign(livre, updatedLivre);
    }
  }
}
```

```
}  
  
deleteLivre(id: number): void {  
  this.livres = this.livres.filter((livre) => livre.id !== id);  
}  
}
```

9 Composant

9.1 LivreComponent

Le composant LivreComponent est responsable de l’affichage et de la gestion des livres dans l’interface utilisateur. Il utilise le service LivreService pour interagir avec les données.

9.1.1 Modules utilisés

- CommonModule: Pour les directives Angular communes.
- FormsModule: Pour la gestion des formulaires.
- Livre: Interface du modèle de livre.
- LivreService: Service pour les opérations CRUD.

9.1.2 Méthodes

- loadLivres(): Charge la liste des livres.
- addLivre(): Ajoute un nouveau livre.
- selectLivreForUpdate(livre: Livre): Sélectionne un livre pour la mise à jour.
- updateLivre(): Met à jour un livre sélectionné.
- deleteLivre(id: number): Supprime un livre par son ID.

```
import { CommonModule } from '@angular/common';
import { Component } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { Livre } from '../models/livre.model';
import { LivreService } from '../services/livre.service';

@Component({
  selector: 'app-livre',
  standalone: true,
  imports: [CommonModule, FormsModule],
  templateUrl: './livre.component.html',
  styleUrls: ['./livre.component.css']
})
export class LivreComponent {
  livres: Livre[] = [];
  newLivre: Livre = { id: 0, titre: '', auteur: '', disponible: true };
  selectedLivre: Livre | null = null;

  constructor(private livreService: LivreService) {}

  ngOnInit(): void {
    this.loadLivres();
  }

  loadLivres(): void {
    this.livres = this.livreService.getLivres();
  }

  addLivre(): void {
    this.newLivre.id = this.livres.length + 1;
    this.livreService.addLivre(this.newLivre);
    this.newLivre = { id: 0, titre: '', auteur: '', disponible: true };
    this.loadLivres();
  }

  selectLivreForUpdate(livre: Livre): void {
    this.selectedLivre = { ...livre };
  }
}
```

```
updateLivre(): void {
  if (this.selectedLivre) {
    this.livreService.updateLivre(this.selectedLivre.id, this.selectedLivre);
    this.selectedLivre = null;
    this.loadLivres();
  }
}

deleteLivre(id: number): void {
  this.livreService.deleteLivre(id);
  this.loadLivres();
}
}
```

10 Template HTML

10.1 livre.component.html

Le template HTML définit la structure de l'interface utilisateur pour la gestion des livres. Il inclut des formulaires pour ajouter et modifier des livres, ainsi qu'une liste pour afficher les livres existants.

```
<div class="livre-container">
  <h1>Gestion des livres</h1>

  <div class="form-container">
    <h2>Ajouter un livre</h2>
    <input [(ngModel)]="newLivre.titre" placeholder="Titre" />
    <input [(ngModel)]="newLivre.auteur" placeholder="Auteur" />
    <button (click)="addLivre()">Ajouter</button>
  </div>

  <div class="form-container" *ngIf="selectedLivre">
    <h2>Modifier un livre</h2>
    <input [(ngModel)]="selectedLivre.titre" placeholder="Titre" />
    <input [(ngModel)]="selectedLivre.auteur" placeholder="Auteur" />
    <button (click)="updateLivre()">Mettre à jour</button>
    <button (click)="selectedLivre = null">Annuler</button>
  </div>

  <div class="livre-list">
    <h2>Liste des livres</h2>
    <ul>
      <li *ngFor="let livre of livres">
        <span class="titre">{{ livre.titre }}</span> par <span class="auteur">{{ livre.auteur }}</span>
        <button (click)="selectLivreForUpdate(livre)">Modifier</button>
        <button (click)="deleteLivre(livre.id)">Supprimer</button>
      </li>
    </ul>
  </div>
</div>
```


11 Styles CSS

11.1 livre.component.css

Les styles CSS définissent l'apparence de l'interface utilisateur. Ils incluent des styles pour les conteneurs, les formulaires, les boutons, et la liste des livres.

```
.livre-container {
  max-width: 800px;
  margin: 0 auto;
  padding: 20px;
  font-family: Arial, sans-serif;
}

.form-container {
  margin-bottom: 20px;
  padding: 15px;
  border: 1px solid #ddd;
  border-radius: 4px;
  background-color: #f9f9f9;
}

.form-container input {
  padding: 10px;
  margin-right: 10px;
  border: 1px solid #ccc;
  border-radius: 4px;
}

.form-container button {
  padding: 10px 15px;
  margin-right: 10px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

.form-container button:hover {
  opacity: 0.8;
}

.livre-list ul {
  list-style-type: none;
  padding: 0;
}

.livre-list li {
  padding: 10px;
  margin-bottom: 10px;
  border: 1px solid #ddd;
  border-radius: 4px;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.livre-list button {
  padding: 5px 10px;
  margin-left: 10px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

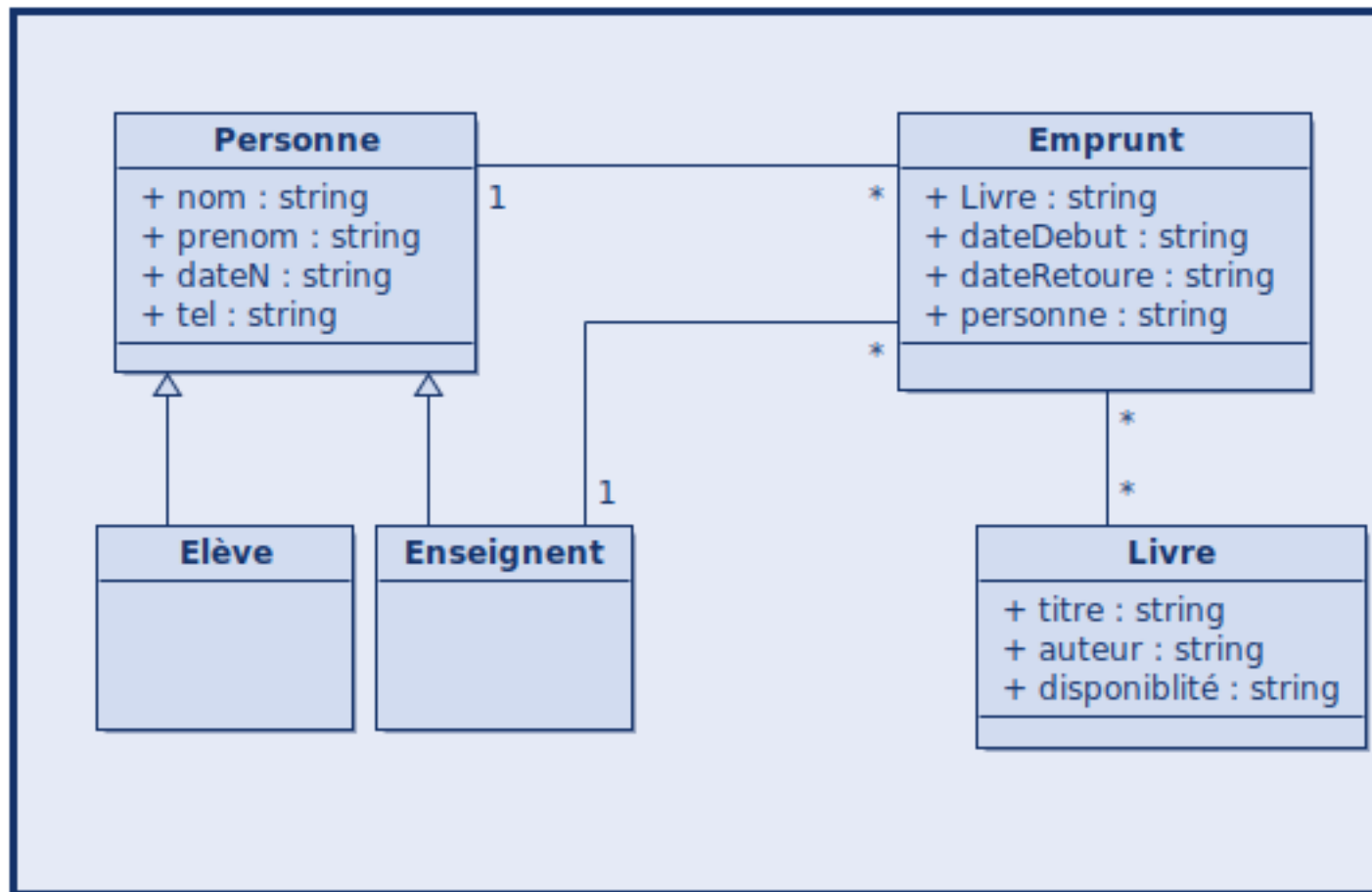
.livre-list button:hover {
  opacity: 0.8;
}
```

12 projet

Introduction

Ce projet a pour objectif de développer un système de gestion de bibliothèque scolaire. Il vise à automatiser les processus liés à l'emprunt, au retour et à l'inventaire des livres, ainsi qu'à la gestion des utilisateurs. L'objectif est de faciliter le travail des bibliothécaires et d'améliorer l'expérience des étudiants.

12.1 Diagramme de classe



Interfaces du système de gestion de bibliothèque

Dans ce projet, nous avons défini plusieurs interfaces pour structurer les données du système de gestion de bibliothèque. Les principales interfaces sont **Livre**, **Emprunt** et **Utilisateur**, chacune représentant des entités différentes de notre modèle de données.

Création des Interfaces dans Angular

Dans ce projet, nous avons créé plusieurs interfaces pour structurer les données de notre application de gestion de bibliothèque. Nous avons utilisé Angular CLI pour générer ces interfaces avec la commande suivante :

```
ng generate interface models/Livre
ng generate interface models/Emprunt
ng generate interface models/personne
```

Interface Livre

L'interface `Livre` définit les propriétés d'un livre dans la bibliothèque :

```
export interface Livre {
  id: number;
  titre: string;
  auteur: string;
  disponible: boolean;
}
```

Interface Emprunt

L'interface `Emprunt` représente un emprunt de livre par un utilisateur, avec des informations sur l'utilisateur et la durée de l'emprunt :

```
export interface Emprunt {
  id: number;
  personneId: number;
  livreId: number;
  dateEmprunt: string;
  dateRetour: string;
}
```

Interface Utilisateur

L'interface `Utilisateur` définit un utilisateur du système, qui peut être un étudiant ou un bibliothécaire :

```
export interface Personne {
  id: number;
  nom: string;
  prenom: string;
  statut: 'élève' | 'enseignant';
}
```

Génération des Services

Nous avons utilisé les commandes Angular CLI suivantes pour générer ces services :

```
ng generate service services/Livre
ng generate service services/Emprunt
ng generate service services/personne
```

PersonneService

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpResponseError } from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';
import { Personne } from '../models/personne.model';

@Injectable({
  providedIn: 'root'
})
export class PersonneService {
  private apiUrl = 'http://localhost:3000/personnes';

  constructor(private http: HttpClient) {}

  getPersonnes(): Observable<Personne[]> {
    return this.http.get<Personne[]>(this.apiUrl).pipe(
```

```

        catchError(this.handleError)
    );
}

getPersonne(id: number): Observable<Personne> {
    return this.http.get<Personne>(`${this.apiUrl}/${id}`).pipe(
        catchError(this.handleError)
    );
}

createPersonne(personne: Personne): Observable<Personne> {
    return this.http.post<Personne>(this.apiUrl, personne).pipe(
        catchError(this.handleError)
    );
}

updatePersonne(id: number, personne: Personne): Observable<Personne> {
    return this.http.put<Personne>(`${this.apiUrl}/${id}`, personne).pipe(
        catchError(this.handleError)
    );
}

deletePersonne(id: number): Observable<void> {
    return this.http.delete<void>(`${this.apiUrl}/${id}`).pipe(
        catchError(this.handleError)
    );
}

private handleError(error: HttpResponse) {
    console.error('Erreur HTTP:', error);
    return throwError('Une erreur s\'est produite. Veuillez r essayer.');
```

LivreService

```

import { Injectable } from '@angular/core';
import { HttpClient, HttpResponse } from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';
import { Livre } from '../models/livre.model';

@Injectable({
    providedIn: 'root'
})
export class LivreService {
    private apiUrl = 'http://localhost:3000/livres';

    constructor(private http: HttpClient) {}

    getLivres(): Observable<Livre[]> {
        return this.http.get<Livre[]>(this.apiUrl).pipe(
            catchError(this.handleError)
        );
    }

    getLivre(id: number): Observable<Livre> {
        return this.http.get<Livre>(`${this.apiUrl}/${id}`).pipe(
            catchError(this.handleError)
        );
    }

    createLivre(livre: Livre): Observable<Livre> {
        return this.http.post<Livre>(this.apiUrl, livre).pipe(
            catchError(this.handleError)
        );
    }
}
```

```

updateLivre(id: number, livre: Livre): Observable<Livre> {
  return this.http.put<Livre>(`${this.apiUrl}/${id}`, livre).pipe(
    catchError(this.handleError)
  );
}

deleteLivre(id: number): Observable<void> {
  return this.http.delete<void>(`${this.apiUrl}/${id}`).pipe(
    catchError(this.handleError)
  );
}

private handleError(error: HttpResponse) {
  console.error('Erreur HTTP:', error);
  return throwError('Une erreur s\'est produite. Veuillez r essayer.');
```

EmpruntService

```

import { Injectable } from '@angular/core';
import { HttpClient, HttpResponse } from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';
import { Emprunt } from '../models/emprunt.model';

@Injectable({
  providedIn: 'root'
})
export class EmpruntService {
  private apiUrl = 'http://localhost:3000/emprunts';

  constructor(private http: HttpClient) {}

  getEmprunts(): Observable<Emprunt[]> {
    return this.http.get<Emprunt[]>(this.apiUrl).pipe(
      catchError(this.handleError)
    );
  }

  getEmprunt(id: number): Observable<Emprunt> {
    return this.http.get<Emprunt>(`${this.apiUrl}/${id}`).pipe(
      catchError(this.handleError)
    );
  }

  createEmprunt(emprunt: Emprunt): Observable<Emprunt> {
    return this.http.post<Emprunt>(this.apiUrl, emprunt).pipe(
      catchError(this.handleError)
    );
  }

  updateEmprunt(id: number, emprunt: Emprunt): Observable<Emprunt> {
    return this.http.put<Emprunt>(`${this.apiUrl}/${id}`, emprunt).pipe(
      catchError(this.handleError)
    );
  }

  deleteEmprunt(id: number): Observable<void> {
    return this.http.delete<void>(`${this.apiUrl}/${id}`).pipe(
      catchError(this.handleError)
    );
  }

  private handleError(error: HttpResponse) {
    console.error('Erreur HTTP:', error);
    return throwError('Une erreur s\'est produite. Veuillez r essayer.');
```

```
}
```

Commandes Angular

Générer les composants de base

```
# Page d'accueil
ng g c components/home

# Barre de navigation
ng g c components/nav

# Pied de page
ng g c components/footer
```

Générer les composants pour les Personnes

```
# Liste des personnes
ng g c components/personnes/personnes-list

# Cr ation d'une personne
ng g c components/personnes/personnes-create

# Mise      jour d'une personne
ng g c components/personnes/personnes-update
```

Générer les composants pour les Livres

```
# Liste des livres
ng g c components/livres/livres-list

# Cr ation d'un livre
ng g c components/livres/livres-create

# Mise      jour d'un livre
ng g c components/livres/livres-update
```

Générer les composants pour les Emprunts

```
# Liste des emprunts
ng g c components/emprunts/emprunts-list

# Cr ation d'un emprunt
ng g c components/emprunts/emprunts-create

# Mise      jour d'un emprunt
ng g c components/emprunts/emprunts-update
```

Générer les services

```
# Service pour les personnes
ng g service services/personne

# Service pour les livres
ng g service services/livre

# Service pour les emprunts
ng g service services/emprunt
```

Composant LivresCreateComponent

TypeScript

```
import { Component } from '@angular/core';
import { LivreService } from '../../../services/livre.service';
import { Livre } from '../../../models/livre.model';
import { Router } from '@angular/router';
import { FormsModule } from '@angular/forms';
import { NgIf } from '@angular/common';

@Component({
  selector: 'app-livres-create',
  standalone: true,
  imports: [FormsModule, NgIf],
  templateUrl: './livres-create.component.html',
  styleUrls: ['./livres-create.component.css']
})
export class LivresCreateComponent {
  livre: Livre = {
    id: 0,
    titre: '',
    auteur: '',
    disponible: true
  };

  constructor(
    private livreService: LivreService,
    private router: Router
  ) {}

  createLivre(): void {
    // Convertir la valeur de 'disponible' en bool en
    this.livre.disponible = this.livre.disponible === true;
    this.livreService.createLivre(this.livre).subscribe({
      next: () => {
        this.router.navigate(['/livres']);
      },
      error: (err) => {
        console.error('Erreur lors de la cr ation du livre', err);
      }
    });
  }
}
```

HTML

```
<div class="container mt-4">
  <h2>Ajouter un Livre</h2>
  <form (ngSubmit)="createLivre()" #livreForm="ngForm">
    <div class="mb-3">
      <label for="titre" class="form-label">Titre</label>
      <input type="text" class="form-control" id="titre" [(ngModel)]="livre.titre" name="titre"
        required #titre="ngModel">
      <div *ngIf="titre.invalid && (titre.touched || titre.dirty)" class="text-danger">
        Le titre est requis.
      </div>
    </div>
    <div class="mb-3">
      <label for="auteur" class="form-label">Auteur</label>
      <input type="text" class="form-control" id="auteur" [(ngModel)]="livre.auteur" name="auteur"
        required #auteur="ngModel">
      <div *ngIf="auteur.invalid && (auteur.touched || auteur.dirty)" class="text-danger">
        L'auteur est requis.
      </div>
    </div>
    <div class="mb-3">
      <label for="disponible" class="form-label">Disponible</label>
```

```

    <select class="form-select" id="disponible" [(ngModel)]="livre.disponible" name="disponible"
    required #disponible="ngModel">
      <option value="true">Oui</option>
      <option value="false">Non</option>
    </select>
    <div *ngIf="disponible.invalid && (disponible.touched || disponible.dirty)" class="text-
    danger">
      La disponibilit  est requise.
    </div>
  </div>
  <button type="submit" class="btn btn-success" [disabled]="livreForm.invalid">Cr er</button>
  <a routerLink="/livres" class="btn btn-secondary">Retour   la liste</a>
</form>
</div>

```

Composant LivresListComponent

TypeScript

```

import { Component, OnInit } from '@angular/core';
import { LivreService } from '../services/livre.service';
import { Livre } from '../models/livre.model';
import { Router, RouterLink } from '@angular/router';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-livres-list',
  standalone: true,
  imports: [CommonModule, RouterLink],
  templateUrl: './livres-list.component.html',
  styleUrls: ['./livres-list.component.css']
})
export class LivresListComponent implements OnInit {
  livres: Livre[] = [];

  constructor(
    private livreService: LivreService,
    private router: Router
  ) {}

  ngOnInit(): void {
    this.loadLivres();
  }

  loadLivres(): void {
    this.livreService.getLivres().subscribe({
      next: (data) => {
        this.livres = data;
      },
      error: (err) => {
        console.error('Erreur lors du chargement des livres', err);
      }
    });
  }

  confirmDelete(id: number): void {
    if (confirm(' tes -vous s r de vouloir supprimer ce livre ?')) {
      this.deleteLivre(id);
    }
  }

  deleteLivre(id: number): void {
    this.livreService.deleteLivre(id).subscribe({
      next: () => {
        this.loadLivres(); // Recharger la liste apr s suppression
      },
      error: (err) => {
        console.error('Erreur lors de la suppression du livre', err);
      }
    });
  }
}

```



```

    }
  });
}
}

```

HTML

```

<div class="container mt-4">
  <h2>Liste des Livres</h2>
  <a routerLink="/livres/create" class="btn btn-primary mb-3">Ajouter un Livre</a>
  <table class="table table-striped">
    <thead>
      <tr>
        <th>Titre</th>
        <th>Auteur</th>
        <th>Disponible</th>
        <th>Actions</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let livre of livres">
        <td>{{ livre.titre }}</td>
        <td>{{ livre.auteur }}</td>
        <td>{{ livre.disponible ? 'Oui' : 'Non' }}</td>
        <td>
          <a [routerLink]="['/livres/update', livre.id]" class="btn btn-warning btn-sm me-2">
            Modifier</a>
          <button (click)="confirmDelete(livre.id)" class="btn btn-danger btn-sm">Supprimer</button>
        </td>
      </tr>
    </tbody>
  </table>
</div>

```

Code Angular pour la mise à jour d'un livre

TypeScript

Voici le code TypeScript pour le composant Angular :

```

import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { LivreService } from '../services/livre.service';
import { Livre } from '../models/livre.model';
import { FormsModule } from '@angular/forms';
import { NgIf } from '@angular/common';

@Component({
  selector: 'app-livres-update',
  standalone: true,
  imports: [FormsModule, NgIf],
  templateUrl: './livres-update.component.html',
  styleUrls: ['./livres-update.component.css']
})
export class LivresUpdateComponent implements OnInit {
  livre: Livre = {
    id: 0,
    titre: '',
    auteur: '',
    disponible: true
  };

  constructor(
    private route: ActivatedRoute,
    private livreService: LivreService,

```

```

    private router: Router
  ) {}

  ngOnInit(): void {
    const id = this.route.snapshot.params['id'];
    this.livreService.getLivre(id).subscribe({
      next: (data) => {
        this.livre = data;
      },
      error: (err) => {
        console.error('Erreur lors du chargement du livre', err);
      }
    });
  }

  updateLivre(): void {
    this.livre.disponible = this.livre.disponible === true;
    this.livreService.updateLivre(this.livre.id, this.livre).subscribe({
      next: () => {
        this.router.navigate(['/livres']);
      },
      error: (err) => {
        console.error('Erreur lors de la mise à jour du livre', err);
      }
    });
  }
}

```

HTML

Voici le code HTML pour le template :

```

<div class="container mt-4">
  <h2>Modifier un Livre</h2>
  <form (ngSubmit)="updateLivre()" #livreForm="ngForm">
    <div class="mb-3">
      <label for="titre" class="form-label">Titre</label>
      <input type="text" class="form-control" id="titre" [(ngModel)]="livre.titre" name="titre"
        required #titre="ngModel">
      <div *ngIf="titre.invalid && (titre.touched || titre.dirty)" class="text-danger">
        Le titre est requis.
      </div>
    </div>
    <div class="mb-3">
      <label for="auteur" class="form-label">Auteur</label>
      <input type="text" class="form-control" id="auteur" [(ngModel)]="livre.auteur" name="auteur"
        required #auteur="ngModel">
      <div *ngIf="auteur.invalid && (auteur.touched || auteur.dirty)" class="text-danger">
        L'auteur est requis.
      </div>
    </div>
    <div class="mb-3">
      <label for="disponible" class="form-label">Disponible</label>
      <select class="form-select" id="disponible" [(ngModel)]="livre.disponible" name="disponible"
        required #disponible="ngModel">
        <option value="true">Oui</option>
        <option value="false">Non</option>
      </select>
      <div *ngIf="disponible.invalid && (disponible.touched || disponible.dirty)" class="text-
        danger">
        La disponibilit est requise.
      </div>
    </div>
    <button type="submit" class="btn btn-warning" [disabled]="livreForm.invalid">Mettre à jour</
    button>
    <a routerLink="/livres" class="btn btn-secondary">Retour à la liste</a>
  </form>
</div>

```

Pour la création d'une personne

TypeScript

Voici le code TypeScript pour le composant Angular :

```
import { Component } from '@angular/core';
import { PersonneService } from '../../../services/personne.service';
import { Personne } from '../../../models/personne.model';
import { Router } from '@angular/router';
import { FormsModule } from '@angular/forms';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-personnes-create',
  standalone: true,
  imports: [FormsModule, CommonModule],
  templateUrl: './personnes-create.component.html',
  styleUrls: ['./personnes-create.component.css']
})
export class PersonnesCreateComponent {
  personne: Personne = {
    id: 0,
    nom: '',
    prenom: '',
    statut: 'lve'
  };

  constructor(
    private personneService: PersonneService,
    private router: Router
  ) {}

  createPersonne(): void {
    this.personneService.createPersonne(this.personne).subscribe({
      next: () => {
        this.router.navigate(['/personnes']);
      },
      error: (err) => {
        console.error('Erreur lors de la création de la personne', err);
      }
    });
  }
}
```

HTML

Voici le code HTML pour le template :

```
<div class="container mt-4">
  <h2>Ajouter une Personne</h2>
  <form (ngSubmit)="createPersonne()" #personneForm="ngForm">
    <div class="mb-3">
      <label for="nom" class="form-label">Nom</label>
      <input type="text" class="form-control" id="nom" [(ngModel)]="personne.nom" name="nom"
        required #nom="ngModel">
      <div *ngIf="nom.invalid && (nom.touched || nom.dirty)" class="text-danger">
        Le nom est requis.
      </div>
    </div>
    <div class="mb-3">
      <label for="prenom" class="form-label">Pr nom</label>
      <input type="text" class="form-control" id="prenom" [(ngModel)]="personne.prenom" name="
        prenom" required #prenom="ngModel">
      <div *ngIf="prenom.invalid && (prenom.touched || prenom.dirty)" class="text-danger">
        Le pr nom est requis.
      </div>
    </div>
    <div class="mb-3">
      <label for="statut" class="form-label">Statut</label>
```

```

    <select class="form-select" id="statut" [(ngModel)]="personne.statut" name="statut" required
    #statut="ngModel">
      <option value=" lve " > lve </option>
      <option value="enseignant">Enseignant</option>
    </select>
    <div *ngIf="statut.invalid && (statut.touched || statut.dirty)" class="text-danger">
      Le statut est requis.
    </div>
  </div>
  <button type="submit" class="btn btn-success" [disabled]="personneForm.invalid">Cr éer</button>
  <a routerLink="/personnes" class="btn btn-secondary">Retour à la liste</a>
</form>
</div>

```

Pour la liste des personnes

TypeScript

Voici le code TypeScript pour le composant Angular :

```

import { Component, OnInit } from '@angular/core';
import { PersonneService } from '../../../services/personne.service';
import { Personne } from '../../../models/personne.model';
import { Router, RouterLink } from '@angular/router';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-personnes-list',
  standalone: true,
  imports: [CommonModule, RouterLink],
  templateUrl: './personnes-list.component.html',
  styleUrls: ['./personnes-list.component.css']
})
export class PersonnesListComponent implements OnInit {
  personnes: Personne[] = [];

  constructor(
    private personneService: PersonneService,
    private router: Router
  ) {}

  ngOnInit(): void {
    this.loadPersonnes();
  }

  loadPersonnes(): void {
    this.personneService.getPersonnes().subscribe({
      next: (data) => {
        this.personnes = data;
      },
      error: (err) => {
        console.error('Erreur lors du chargement des personnes', err);
      }
    });
  }

  confirmDelete(id: number): void {
    if (confirm('Êtes-vous sûr de vouloir supprimer cette personne ?')) {
      this.deletePersonne(id);
    }
  }

  deletePersonne(id: number): void {
    this.personneService.deletePersonne(id).subscribe({
      next: () => {
        this.loadPersonnes(); // Recharger la liste après suppression
      },
      error: (err) => {

```

```

        console.error('Erreur lors de la suppression de la personne', err);
    }
    });
}
}

```

HTML

Voici le code HTML pour le template :

```

<div class="container mt-4">
  <h2>Liste des Personnes</h2>
  <a routerLink="/personnes/create" class="btn btn-primary mb-3">Ajouter une Personne</a>
  <table class="table table-striped">
    <thead>
      <tr>
        <th>Nom</th>
        <th>Pr nom</th>
        <th>Statut</th>
        <th>Actions</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let personne of personnes">
        <td>{{ personne.nom }}</td>
        <td>{{ personne.prenom }}</td>
        <td>{{ personne.statut }}</td>
        <td>
          <a [routerLink]="['/personnes/update', personne.id]" class="btn btn-warning btn-sm me-2">
Modifier</a>
          <button (click)="confirmDelete(personne.id)" class="btn btn-danger btn-sm">Supprimer</button>
        </td>
      </tr>
    </tbody>
  </table>
</div>

```

```

\section*{Pour le update des personnes}
\subsection*{TypeScript}
Voici le code TypeScript pour le composant Angular :
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { PersonneService } from '../../../services/personne.service';
import { Personne } from '../../../models/personne.model';
import { FormsModule } from '@angular/forms';
import { NgIf } from '@angular/common';

@Component({
  selector: 'app-personnes-update',
  standalone: true,
  imports: [FormsModule, NgIf],
  templateUrl: './personnes-update.component.html',
  styleUrls: ['./personnes-update.component.css']
})
export class PersonnesUpdateComponent implements OnInit {
  personne: Personne = {
    id: 0,
    nom: '',
    prenom: '',
    statut: 'lve'
  };

  constructor(
    private route: ActivatedRoute,
    private personneService: PersonneService,
    private router: Router
  ) {}

```

```

ngOnInit(): void {
  const id = this.route.snapshot.params['id'];
  this.personneService.getPersonne(id).subscribe({
    next: (data) => {
      this.personne = data;
    },
    error: (err) => {
      console.error('Erreur lors du chargement de la personne', err);
    }
  });
}

updatePersonne(): void {
  this.personneService.updatePersonne(this.personne.id, this.personne).subscribe({
    next: () => {
      this.router.navigate(['/personnes']);
    },
    error: (err) => {
      console.error('Erreur lors de la mise à jour de la personne', err);
    }
  });
}
}

```

HTML

Voici le code HTML pour le template :

```

<div class="container mt-4">
  <h2>Modifier une Personne</h2>
  <form (ngSubmit)="updatePersonne()" #personneForm="ngForm">
    <div class="mb-3">
      <label for="nom" class="form-label">Nom</label>
      <input type="text" class="form-control" id="nom" [(ngModel)]="personne.nom" name="nom"
        required #nom="ngModel">
      <div *ngIf="nom.invalid && (nom.touched || nom.dirty)" class="text-danger">
        Le nom est requis.
      </div>
    </div>
    <div class="mb-3">
      <label for="prenom" class="form-label">Pr nom</label>
      <input type="text" class="form-control" id="prenom" [(ngModel)]="personne.prenom" name="
        prenom" required #prenom="ngModel">
      <div *ngIf="prenom.invalid && (prenom.touched || prenom.dirty)" class="text-danger">
        Le pr nom est requis.
      </div>
    </div>
    <div class="mb-3">
      <label for="statut" class="form-label">Statut</label>
      <select class="form-select" id="statut" [(ngModel)]="personne.statut" name="statut" required
        #statut="ngModel">
        <option value="lve"> lve </option>
        <option value="enseignant">Enseignant</option>
      </select>
      <div *ngIf="statut.invalid && (statut.touched || statut.dirty)" class="text-danger">
        Le statut est requis.
      </div>
    </div>
    <button type="submit" class="btn btn-warning" [disabled]="personneForm.invalid">Mettre à jour
  </button>
  <a routerLink="/personnes" class="btn btn-secondary">Retour à la liste</a>
  </form>
</div>

```

EmpruntsCreateComponent

TypeScript

```
import { Component } from '@angular/core';
import { EmpruntService } from '../../../services/emprunt.service';
import { Emprunt } from '../../../models/emprunt.model';
import { Router } from '@angular/router';
import { FormsModule, NgForm } from '@angular/forms';
import { NgIf } from '@angular/common';

@Component({
  selector: 'app-emprunts-create',
  standalone: true,
  imports: [FormsModule, NgIf],
  templateUrl: './emprunts-create.component.html',
  styleUrls: ['./emprunts-create.component.css']
})
export class EmpruntsCreateComponent {
  emprunt: Emprunt = {
    id: 0,
    personneId: 0,
    livreId: 0,
    dateEmprunt: '',
    dateRetour: ''
  };

  errorMessage: string = '';

  constructor(
    private empruntService: EmpruntService,
    private router: Router
  ) {}

  createEmprunt(form: NgForm): void {
    if (this.emprunt.dateRetour <= this.emprunt.dateEmprunt) {
      this.errorMessage = 'La date de retour doit être postérieure à la date d\'emprunt.';
      return;
    }

    this.errorMessage = ''; // Réinitialiser le message d'erreur

    this.empruntService.createEmprunt(this.emprunt).subscribe({
      next: () => {
        this.router.navigate(['/emprunts']);
      },
      error: (err) => {
        console.error('Erreur lors de la création de l\'emprunt', err);
        this.errorMessage = 'Une erreur est survenue lors de la création de l\'emprunt. Veuillez réessayer.';
      }
    });
  }
}
```

Template HTML

```
<div class="container mt-4">
  <h2 class="mb-4">Ajouter un Emprunt</h2>

  <!-- Affichage du message d'erreur global -->
  <div *ngIf="errorMessage" class="alert alert-danger">
    {{ errorMessage }}
  </div>

  <form (ngSubmit)="createEmprunt(empruntForm)" #empruntForm="ngForm">
    <!-- Champ ID de la Personne -->
    <div class="mb-3">
```

```

<label for="personneId" class="form-label">ID de la Personne</label>
<input
  type="number"
  class="form-control"
  id="personneId"
  [(ngModel)]="emprunt.personneId"
  name="personneId"
  required
  min="1"
  #personneId="ngModel"
>
<!-- Message d'erreur pour ID de la Personne -->
<div *ngIf="personneId.invalid && (personneId.touched || personneId.dirty)" class="text-
danger">
  <div *ngIf="personneId.errors?.['required']">L'ID de la personne est requis.</div>
  <div *ngIf="personneId.errors?.['min']">L'ID de la personne doit tre sup rieur 0.</
div>
</div>
</div>

<!-- Champ ID du Livre -->
<div class="mb-3">
  <label for="livreId" class="form-label">ID du Livre</label>
  <input
    type="number"
    class="form-control"
    id="livreId"
    [(ngModel)]="emprunt.livreId"
    name="livreId"
    required
    min="1"
    #livreId="ngModel"
  >
  <!-- Message d'erreur pour ID du Livre -->
  <div *ngIf="livreId.invalid && (livreId.touched || livreId.dirty)" class="text-danger">
    <div *ngIf="livreId.errors?.['required']">L'ID du livre est requis.</div>
    <div *ngIf="livreId.errors?.['min']">L'ID du livre doit tre sup rieur 0.</div>
  </div>
</div>

<!-- Champ Date d'Emprunt -->
<div class="mb-3">
  <label for="dateEmprunt" class="form-label">Date d'Emprunt</label>
  <input
    type="date"
    class="form-control"
    id="dateEmprunt"
    [(ngModel)]="emprunt.dateEmprunt"
    name="dateEmprunt"
    required
    #dateEmprunt="ngModel"
  >
  <!-- Message d'erreur pour Date d'Emprunt -->
  <div *ngIf="dateEmprunt.invalid && (dateEmprunt.touched || dateEmprunt.dirty)" class="text-
danger">
    <div *ngIf="dateEmprunt.errors?.['required']">La date d'emprunt est requise.</div>
  </div>
</div>

<!-- Champ Date de Retour -->
<div class="mb-3">
  <label for="dateRetour" class="form-label">Date de Retour</label>
  <input
    type="date"
    class="form-control"
    id="dateRetour"
    [(ngModel)]="emprunt.dateRetour"
    name="dateRetour"
    required
    #dateRetour="ngModel"
  >

```



```

    <!-- Message d'erreur pour Date de Retour -->
    <div *ngIf="dateRetour.invalid && (dateRetour.touched || dateRetour.dirty)" class="text-
danger">
      <div *ngIf="dateRetour.errors?.['required']">La date de retour est requise.</div>
    </div>
  </div>

  <!-- Bouton de Soumission -->
  <button type="submit" class="btn btn-success" [disabled]="empruntForm.invalid">Cr er</button>
  <a routerLink="/emprunts" class="btn btn-secondary">Retour la liste</a>
</form>
</div>

```

EmpruntsListComponent

TypeScript

```

import { Component, OnInit } from '@angular/core';
import { EmpruntService } from '../../services/emprunt.service';
import { Emprunt } from '../../models/emprunt.model';
import { Router, RouterLink } from '@angular/router';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-emprunts-list',
  standalone: true,
  imports: [CommonModule, RouterLink],
  templateUrl: './emprunts-list.component.html',
  styleUrls: ['./emprunts-list.component.css']
})
export class EmpruntsListComponent implements OnInit {
  emprunts: Emprunt[] = [];

  constructor(
    private empruntService: EmpruntService,
    private router: Router
  ) {}

  ngOnInit(): void {
    this.loadEmprunts();
  }

  loadEmprunts(): void {
    this.empruntService.getEmprunts().subscribe({
      next: (data) => {
        this.emprunts = data;
      },
      error: (err) => {
        console.error('Erreur lors du chargement des emprunts', err);
      }
    });
  }

  getPersonneName(personneId: number): string {
    // Impl mentez cette m thode pour retourner le nom de la personne
    return 'Personne ${personneId}'; // Exemple
  }

  getLivreTitle(livreId: number): string {
    // Impl mentez cette m thode pour retourner le titre du livre
    return 'Livre ${livreId}'; // Exemple
  }

  confirmDelete(id: number): void {
    if (confirm('tes -vous s r de vouloir supprimer cet emprunt ?')) {
      this.deleteEmprunt(id);
    }
  }
}

```

```

deleteEmprunt(id: number): void {
  this.empruntService.deleteEmprunt(id).subscribe({
    next: () => {
      this.loadEmprunts(); // Recharger la liste apr s suppression
    },
    error: (err) => {
      console.error('Erreur lors de la suppression de l'emprunt', err);
    }
  });
}
}
}

```

Template HTML

```

<div class="container mt-4">
  <h2>Liste des Emprunts</h2>
  <a routerLink="/emprunts/create" class="btn btn-primary mb-3">Ajouter un Emprunt</a>
  <table class="table table-striped">
    <thead>
      <tr>
        <th>Personne</th>
        <th>Livre</th>
        <th>Date d'Emprunt</th>
        <th>Date de Retour</th>
        <th>Actions</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let emprunt of emprunts">
        <td>{{ getPersonneName(emprunt.personneId) }}</td>
        <td>{{ getLivreTitle(emprunt.livreId) }}</td>
        <td>{{ emprunt.dateEmprunt | date:'dd/MM/yyyy' }}</td>
        <td>{{ emprunt.dateRetour | date:'dd/MM/yyyy' }}</td>
        <td>
          <a [routerLink]="['/emprunts/update', emprunt.id]" class="btn btn-warning btn-sm me-2">
            Modifier</a>
          <button (click)="confirmDelete(emprunt.id)" class="btn btn-danger btn-sm">Supprimer</button>
        </td>
      </tr>
    </tbody>
  </table>
</div>

```

Code Angular : EmpruntsUpdateComponent

TypeScript

```

import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { EmpruntService } from '../services/emprunt.service';
import { Emprunt } from '../models/emprunt.model';
import { FormsModule } from '@angular/forms';

@Component({
  selector: 'app-emprunts-update',
  standalone: true,
  imports: [FormsModule],
  templateUrl: './emprunts-update.component.html',
  styleUrls: ['./emprunts-update.component.css']
})
export class EmpruntsUpdateComponent implements OnInit {
  emprunt: Emprunt = {
    id: 0,
    personneId: 0,

```

```

    livreId: 0,
    dateEmprunt: '',
    dateRetour: ''
  };

  constructor(
    private route: ActivatedRoute,
    private empruntService: EmpruntService,
    private router: Router
  ) {}

  ngOnInit(): void {
    const id = this.route.snapshot.params['id'];
    this.empruntService.getEmprunt(id).subscribe(data => {
      this.emprunt = data;
    });
  }

  updateEmprunt(): void {
    this.empruntService.updateEmprunt(this.emprunt.id, this.emprunt).subscribe(() => {
      this.router.navigate(['/emprunts']);
    });
  }
}

```

Template HTML

```

<div class="container mt-4">
  <h2>Modifier un Emprunt</h2>
  <form (ngSubmit)="updateEmprunt()">
    <div class="mb-3">
      <label for="personneId" class="form-label">ID de la Personne</label>
      <input type="number" class="form-control" id="personneId" [(ngModel)]="emprunt.personneId"
        name="personneId" required>
    </div>
    <div class="mb-3">
      <label for="livreId" class="form-label">ID du Livre</label>
      <input type="number" class="form-control" id="livreId" [(ngModel)]="emprunt.livreId" name="
        livreId" required>
    </div>
    <div class="mb-3">
      <label for="dateEmprunt" class="form-label">Date d'Emprunt</label>
      <input type="date" class="form-control" id="dateEmprunt" [(ngModel)]="emprunt.dateEmprunt"
        name="dateEmprunt" required>
    </div>
    <div class="mb-3">
      <label for="dateRetour" class="form-label">Date de Retour</label>
      <input type="date" class="form-control" id="dateRetour" [(ngModel)]="emprunt.dateRetour"
        name="dateRetour" required>
    </div>
    <button type="submit" class="btn btn-warning">Mettre    jour</button>
    <a routerLink="/emprunts" class="btn btn-secondary">Retour    la liste</a>
  </form>
</div>

```

Liste des Emprunts

Ajouter un Emprunt

Personne	Livre	Date d'Emprunt	Date de Retour
Inconnu	tout laisse	20/03/2025	01/01/1970
dermane	Inconnu	12/03/2025	29/03/2025
dermane	camara lay	04/03/2025	05/03/2025
dermane	camara lay	04/03/2025	05/03/2025
sani	camara lay	03/03/2025	28/03/2025
Inconnu	camara lay	03/03/2025	27/03/2025
Inconnu	Le Petit Prince	12/03/2025	29/03/2025
Inconnu	Le Petit Prince	18/03/2025	28/03/2025

Liste des Livres

Rechercher un livre par titre ou auteur...

Ajouter un Livre

Titre	Auteur	Disponible
Le Petit Prince	Antoine de Saint-Exupéry	Oui
camara lay	enfant noir	Oui
le monde s'effondre	chinua achebe	Oui
tout laisse	soda	Oui
l'argent	rachad	Oui
lfjezj	qzkfjzlr	Oui

Liste des Personnes

Rechercher par nom ou prénom

Ajouter une Personne

Nom	Prénom	Statut	Actions
adom	teouri	élève	 
dermane	koumai	enseignant	 
sani	koumai	élève	 
tamba	riro	élève	 
tete	zgr	enseignant	 
egrg	ggrg	élève	 