Projet ENSTA

Partie 5 : Mise en place de l'interface front

Installation de React

Installation de Node

Pour Linux, il faut simplement mettre à jour le système et installer node via la commande apt install.

sudo apt update
sudo apt install nodejs
sudo apt install npm

Pour Windows, il suffit de télécharger node via l'installateur sur le site https://nodejs.org/en/download/. S'assurer que l'utilitaire npm est présent dans le package lors de la phase d'installation personnalisée.

Mise en place du projet

La mise en place d'un projet react est très rapide grâce à la commande très puissante npx. Dans un premier temps on va initialiser le projet, puis on y injectera les dépendances majeures dont on aura besoin par la suite. Il ne restera plus qu'à lancer le projet pour vérifier que tout fonctionne bien.

2.1. Création du projet

npx create-react-app my-film

Cette commande crée le projet à l'intérieur d'un répertoire my-film/ qu'elle prendra soin de créer s'il n'existe pas déjà.

2.7. Dans un navigateur, entrer l'adresse localhost:3000
Félicitations, le projet React a été initialisé avec succès !

2.2. Ajouter une bibliothèque de composant.

Pour ce projet on va utiliser MUI.

npm install @mui/material @emotion/react @emotion/styled

Header

L'application se présentera sous la forme suivante :
Une page d'accueil contenant la liste des films du backend.
Pour chaque film, un bouton qui redirige vers sa page de détails
Une page qui comprendra les détails d'un film spécifique, ainsi que la liste des autres films de son réalisateur.

3.1. Le premier composant sera le header, qui se présentera sous la forme d'une barre horizontale classique. Dans src/ créer le fichier Header.js. Dans ce fichier, créer votre premier function Component avec le contenu suivant:

```
export default function Header() {
  return (
        <h1>My Header</h1>
  )
}
```

3.2. Supprimer ce qui est retourner par app.js, et le remplacer par la simple balise qui suit :

```
<Header/>
```

Si on réexécute la commande npm start, on doit avoir un simple texte qui dit : "My header".

3.3. Dans le fichier header.js, remplacer le contenu par le code suivant :

```
<AppBar position="static">
 <Toolbar>
  <Typography variant="h6" component="div" sx={{ flexGrow: 1 }}>
        My Films
  </Typography>
 </Toolbar >
</AppBar>
```

Cela créera une barre de navigation qui, pour le moment, ne comporte qu'un titre.

Liste des films

4.1. Créer le composant FilmList

```
export default function FilmList() {
	const films= ["film1", "film2", "film3"];


	return films.map((film)=> {
		return <h1>{film}</h1>
	})
}
```

Dans App.js ajouter sous le Header

```
<FilmList />
```

Maintenant on peut voir l'affichage de film1 film2 et film3

4.5. A la place d'un simple affichage de string, on va plutôt créer des Cards pour afficher les films à l'intérieur.

Créer un composant FilmCard, qui prendra une prop : un objet film.

Voici un template pour votre Card:

```
<Card variant="outlined">
<CardContent>
		<Typography variant="h5" gutterBottom>
			{props.film.titre}
		</Typography>
		<Typography variant="body1">
			{props.film.duree} minutes
		</Typography>
	</CardContent>
</Card>
```

4.6. Maintenant qu'on a le template, on va devoir remplacer la liste de String par les films. On va dans un premier temps mocker cette liste.

Créer un dossier mock et un fichier FilmMock.js

Dans ce fichier créer une liste de film dans un objet qui s'appellera mockFilms.

4.7. Dans FilmList on peut maintenant remplacer la liste de string par notre mock et remplacer la balise h1 par une FilmCard.

```
const films= mockFilms;


return films.map((film)=> {
        return <FilmCard key={film.id} film={film} />
})
```

On peut maintenant voir notre liste de films s'afficher.

Accéder au backend

On a désormais un rendu correct, mais on n'utilise pas de vraies données. Il est temps de faire appel au backend !

6.1. installer axios

```
npm install axios
```

6.2 Créer un dossier api sous src/ et un fichier FilmApi.js dedans.

```
import axios from 'axios';


const FILM_URI = 'http://localhost:8080/film'


export function getAllFilms(){
        return axios.get(FILM_URI);
}
```

La méthode getAllFilms va permettre d'interroger le backend et de récupérer la liste des films

6.3. Dans FilmList on va remplacer le contenu de notre liste par le résultat de la requête si celle-ci est un succès.

```
const [films, setFilms] = useState([]);


useEffect(() => {
  getAllFilms().then(reponse => {
        setFilms(reponse.data);
  }).catch(err => {
        console.log(err);
  })
}, []);
```

On observe que notre backend est bien interrogé et que la liste de film de la réponse est bien affichée.

Créer un nouveau film
La prochaine étape est de pouvoir créer un nouveau film.

6.1
Dans FilmApi créer les méthode d'appel api permettant de créer, éditer et supprimer un film. Elles devront s'appeler respectivement postFilm, putFilm, et deleteFilm.

6.2
Créer un nouveau composant que l'on appellera CreateFilmForm.

Ce composant sera composé de deux TextField pour le titre et la durée, d'un Select pour choisir le réalisateur, et d'un bouton. (https://mui.com/material-ui/react-select/)

Créer un fichier RealisateurApi dans le dossier api et créer la méthode réalisant l'appel getAllRealisateur.

De la même manière que pour récupérer la liste de film, récupérer la liste de réalisateurs.

Vous pourrez créer les valeurs dans votre Select à l'aide du fragment suivant.

```
{
realisateurs.map(realisateur  => {
```

```
return <MenuItem key={realisateur.id} value={realisateur.id}>
        {realisateur.prenom} {realisateur.nom}
</MenuItem>})
}
```

6.3
Créer une méthode qui sera appelée lors de l'appui sur le bouton pour créer un film et appeler la méthode postFilm.

6.4

Dans app.js ajouter votre composant FilmForm.

Vous êtes maintenant en mesure de créer des films, cependant ces derniers ne sont pas ajoutés à la liste. Pour le moment vous devez recharger votre page pour la mettre à jour.

6.5
Dans app.js enlever les balises FilmForm et FilmList.
Créer un composant FilmContainer qui renverra un FilmForm et une FilmList.
Et ajouter la balise FilmContainer dans app.js

6.6
Modifier FilmList et FilmContainer pour que l'appel api soit fait dans FilmContainer. Pour cela vous devez faire en sorte que le composant FilmList utilise une props films qui contiendra la liste de films.

6.7
De la même manière, faite passer l'appel api pour la création du film dans le composant FilmContainer. Pour cela vous devrez faire en sorte que le composant FilmForm utilise une props onSubmit qui contiendra les actions à réaliser lors de l'appui sur le bouton.

Editer un film
7.1. Dans le composant FilmCard, ajouter deux boutons, qui correspondront à l'édition et la suppression d'un film.

```
<IconButton onClick={handleClickOnDeleteButton}>
  <DeleteIcon/>
</IconButton>
<IconButton onClick={handleClickOnEditButton}>
  <EditIcon/>
```

</IconButton>

et ajouter les méthodes suivantes

```
  const handleClickOnDeleteButton = () => {
        console.log("delete");
  }


  const handleClickOnEditButton = () => {
        console.log("edit");
  }
```

Vous pouvez observer que lorsque vous cliquez sur les boutons un message s'affiche dans la console de navigateur.

7.2. On va procéder à l'édition d'un film via une boîte de dialogue.

```
        <Dialog onClose={handleClose} open={open}>
        <DialogTitle>Editer un film</DialogTitle>
        <DialogContent>
        //FilmForm à ajouter
        </DialogContent>
        </Dialog>
```

Réaliser les imports nécessaires

7.3
Modifier le composant FilmForm pour pouvoir lui donner un film en props et que le formulaire soit prérempi.

7.4 Dans FilmList implémenté les fonctions pour soumettre le formulaire

CRUD - Delete et finition

La dernière étape des opérations CRUD est la suppression d'un film. On clique sur le bouton de suppression et le film est supprimé de la liste.

Partie 6 : Utilisation d'un ORM

Jusqu'à maintenant, on utilisait JDBC pour se connecter à la base de données. C'est une solution performante, mais lourde à mettre en œuvre lorsque les entités métiers grossissent et deviennent plus nombreuses.

Une solution alternative consiste à utiliser un ORM (Object Relational Mapping), qui va se charger de faire automatiquement le mapping des données en base avec les entités métiers. Il faut pour cela définir correctement ce mapping dans l'entité via des annotations JPA.

Mise en place du mapping des entités

1.1. Mettre en place le mapping JPA des entités Film et Realisateur.

Mise en place des DAO JPA

2.1. Dans le package com.ensta.myfilmlist.dao.impl, créer une nouvelle classe JpaFilmDAO qui va implémenter l'interface FilmDAO. C'est également un Repository Spring, donc il faut lui ajouter l'annotation correspondante.

2.2. Injecter dans cette classe le Bean EntityManager via l'annotation @PersistenceContext.

```
@PersistenceContext
private EntityManager entityManager;
```

Remarque : ici, on n'utilise pas @Autowired, car il y a en réalité une instance d'entityManager par Thread, donc globalement une instance par requête cliente. Comme l'objet n'est pas thread safe, les requêtes ne peuvent pas partager la même instance. L'annotation @PersistenceContext permet de gérer ce besoin.

2.3. Implémenter les différentes méthodes de l'interface. On utilisera :
entityManager.persist(…) : pour sauvegarder une nouvelle entité
entityManager.merge(...) : pour enregistrer une entité déjà existante
entityManager.find(…) : pour renvoyer une entité par son id
entityManager.remove(…) : pour supprimer une entité
entityManager.createQuery(…) : pour les autres besoins

Remarque : la méthode delete() prend un film en paramètre. Si celui-ci correspond à une entité à l'état « détaché » (c'est-à-dire pas dans le contexte JPA), il faut la rattacher avant de la supprimer. On peut tester ce cas avec entityManager.contains(film). Si cette méthode renvoie false, on peut rattacher l'entité avant de la supprimer via entityManager.merge(film). Il faudra également avoir vérifié auparavant que l'entité existe bien en base de

données, sinon ce n'est pas la peine de faire ce traitement. Ci-dessous le code associé :

```
if (entityManager.find(Film.class, film.getId()) != null) {
    if (!entityManager.contains(film)) {
        film = entityManager.merge(film);
    }
    …
}
```

2.4. Créer une nouvelle classe JpaRealisateurDAO, et effectuer les mêmes opérations.

2.5. Redémarrer le serveur. Vous devriez rencontrer une erreur de type UnsatisfiedDependencyException / NoUniqueBeanDefinitionException.

C'est parce qu'il y a désormais 2 Bean Spring de type FilmDAO, et Spring ne sait donc pas lequel il doit utiliser. Il y a plusieurs façons de résoudre ce problème. Dans le cas présent, on va simplement ajouter une annotation @Primary au-dessus des Bean JPA, pour dire à Spring que ce sont ces Beans-là qui doivent être utilisés en priorité.

2.6. Ajouter l'annotation @Primary sur les classes JpaFilmDAO et JpaRealisateurDAO. Redémarrer le serveur, et tester le bon fonctionnement de l'application.

Remarque : On remarque ici que nous n'avons rien changé d'autre que l'implémentation des DAO. On peut tout à faire revenir aux implémentations JDBC en déplaçant l'annotation @Primary sur les DAO JDBC, et tout fonctionnera correctement. C'est donc un des grands avantages de Spring et de l'utilisation des interfaces : on peut très facilement modifier les implémentations, et donc effectuer des traitements très différents, tout en conservant le même contrat d'interface.

Partie 7 : Améliorer le projet

Les points suivants ne sont que des axes d'amélioration d'une application déjà fonctionnelle, la priorité reste d'avoir toute la partie précédente. N'attaquez donc pas la suite s'il vous manque des choses.

Vous êtes libre de choisir quelle(s) amélioration(s) vous souhaitez apporter, aussi bien celles ci-dessous que d'autres auxquelles vous pourriez penser.

Choisir un réalisateur

Dans la partie 5.9., on a dans notre formulaire d'édition de film un champ pour l'ID du réalisateur. Cependant ce n'est pas très intuitif.

Modifiez votre formulaire pour faire apparaître une liste déroulante (Voir exemple du cours) afin de sélectionner directement le réalisateur.

Il faudra donc faire en sorte de récupérer la liste des réalisateurs depuis le backend.

Ajouter un genre aux films

A tout moment, un projet peut être sujet aux modifications, et les bases de données ne sont pas épargnées.

Modifiez le fichier data.sql du backend et ajoutez une table Genre, qui possédera un id et un nom. Ajoutez une colonne genre_id à la table Film et ajoutez leur valeur dans les inserts.

Modifiez votre code pour intégrer correctement ce nouvel élément dans vos films. Il doit également apparaître dans le frontend.

Opérations CRUD sur les réalisateurs

Pour avoir un gestionnaire complet d'une base de données, il faut avoir la possibilité d'agir sur sa totalité.

Modifiez votre code pour donner la possibilité d'afficher, ajouter, modifier ou supprimer un réalisateur.

Remarque : Lors de la suppression d'un réalisateur, tous les films qu'il a réalisés doivent également être supprimés.

Questions ouvertes :

deleteRealisateur Suppression des films associés Transaction
On va mettre en place un filtre sur le nom du film, et sur le nom d'un réalisateur.
On va mettre en place un order sur l'Id (par defaut), sur le nom d'un film, sur sa durée.
Mettre en place des utilisateurs. On utilisera Spring Security. Un utilisateur peut se connecter.
Un utilisateur a une liste de films qu'il a vue. Il peut ajouter ou enlever un films de sa liste.
Un utilisateur peut donner une note sur 20 (note étant un entier) pour les films.
On affiche les notes des films, note étant calculé en faisant la moyenne des notes des utilisateur.

Question Bonus 1 :
On va faire une pagination pour les clients et les réalisateurs. C'est à dire qu'on va créer un objet Page<T> qui contient les attribues
private int number;
private int size;
private long total;
private List<T> data;
Les webservices renveront donc cet objet plutôt qu'une liste.

Question Bonus 2 :
Rajoutez une colonne genre dans la table des films. Créez une table Genre qui possède un nom. Initialisez avec des genres comme : action, thriller, policier, horreur, comédie, SF, drame, biopic. Les genres étant supposé être fixe (si modification cela sera fait par un admin), on n'a pas besoin de faire de create et de delete.
Faire les fonction getAll, getById pour les genres. Mettre à jours les fonctions des films pour faire apparaître le genre.

Question Bonus 3 :
On va mettre en place Sonar, un outil permettant de vérifier le clean de votre code. Il vous chiffre notamment votre dette technique.