Charles Hurst - Cheikh Ahmadou Bamba Diouf - Meriem NOUIRA - Bilel RAHMOUNI

Projet REST

Outils : Node.js, MySQL Community Server, Postman

Introduction

Le projet **Central Cinema** vise à créer une plateforme moderne pour la gestion et la visualisation des films, leurs horaires et les informations associées, avec une interface utilisateur interactive et intuitive. Le frontend joue un rôle crucial dans ce projet en assurant une expérience utilisateur optimale.

Préparation de l'environnement

Nous faisons les étapes suivantes :

Nous vérifions que Node.js, MySQL et Postman sont installés sur notre machine. Si ce n'est pas le cas, nous installons ces outils :

- ☑ Téléchargez et installez Node.js.
- ☑ Téléchargez et installez MySQL Community Server.
- ☑ Téléchargez et installez Postman.

Nous ouvrons un terminal pour vérifier les installations :

```
node -v       # Vérifie que Node.js est installé
npm -v        # Vérifie que NPM est installé avec Node.js
mysql --version  # Vérifie que MySQL est installé
```

Création de la structure du projet :

Nous créons un répertoire principal pour notre projet :

```
mkdir cinema-project
cd cinema-project
mkdir cinema-backend cinema-frontend
```

Création et configuration de la base de données MySQL

La base de données MySQL constitue le pilier central du projet **Central Cinema**. Elle a été soigneusement configurée pour gérer les films, les programmations et les disponibilités dans différentes salles de cinéma.

Connexion à MySQL

Pour commencer, nous nous connectons à MySQL via le terminal avec la commande suivante :

```
mysql -u root -p
```

Une fois authentifiés, nous pouvons effectuer les opérations nécessaires sur la base de données.

2. Création de la base de données

Nous avons créé la base de données principale en spécifiant un jeu de caractères compatible avec les langues et caractères spéciaux, garantissant ainsi une bonne prise en charge de l'internationalisation :

```
CREATE DATABASE cinema_db CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
USE cinema_db;
```

3. Création des tables

La base de données comprend plusieurs tables soigneusement conçues pour répondre aux besoins fonctionnels du projet :

Table des films :
Cette table contient les informations principales sur les films disponibles dans le système. Elle comprend des colonnes telles que le titre, la durée, le réalisateur, les acteurs, et l'âge minimum requis.

```
CREATE TABLE films (
  id INT AUTO_INCREMENT PRIMARY KEY,
  titre VARCHAR(255) NOT NULL,
  duree INT NOT NULL,
  langue VARCHAR(50),
  sous_titres VARCHAR(50),
```

```
  realisateur VARCHAR(255),
  acteurs TEXT,
  age_minimum INT,
  description TEXT, -- Ajouté pour inclure une description détaillée du
film
  banner VARCHAR(255), -- Ajouté pour stocker l'URL de l'image de la
bannière du film
  trailer VARCHAR(255) -- Ajouté pour inclure l'URL de la bande-annonce
);
```

Table des disponibilités :

Cette table, nommée `cinema_availabilities`, est utilisée pour suivre les horaires spécifiques des films dans différentes salles de cinéma. Chaque disponibilité est liée à un film et inclut le nom du cinéma, la date, et l'heure de la projection.

```
CREATE TABLE cinema_availabilities (
  id INT AUTO_INCREMENT PRIMARY KEY,
  film_id INT NOT NULL,
  cinema_name VARCHAR(255) NOT NULL,
  date DATE NOT NULL,
  time TIME NOT NULL,
  createdAt DATETIME NOT NULL,
  updatedAt DATETIME NOT NULL,
  FOREIGN KEY (film_id) REFERENCES films(id) ON DELETE CASCADE
);
```

4. Modifications et Mises à Jour

Pendant le développement, nous avons effectué les modifications suivantes pour répondre aux besoins du projet :

1. **Ajout de nouvelles colonnes** :
    - Dans la table `films`, nous avons ajouté :
        - `description` : Permet de fournir un synopsis détaillé.
        - `banner` : Stocke l'URL de l'image promotionnelle.
        - `trailer` : Stocke l'URL de la bande-annonce vidéo.
2. **Refonte des noms de tables** :

    - Nous avons corrigé les incohérences dans les noms des tables, en veillant à utiliser `cinema_availabilities` pour toutes les requêtes et structures.
3. **Tests et validations** :
    - Nous avons effectué des requêtes de test pour insérer et récupérer des données, garantissant l'intégrité et la cohérence des relations entre les tables.

5. Vérifications et Résultats

Après la configuration et les ajustements, nous avons vérifié que toutes les tables sont correctement reliées et que les données peuvent être insérées et récupérées sans erreurs. Voici un exemple de requêtes exécutées pour tester la base de données :

- **Insertion d'un film** :

```
INSERT INTO films (titre, duree, langue, realisateur, description,
banner, trailer) VALUES

('Inception', 148, 'Anglais', 'Christopher Nolan', 'Un thriller
psychologique captivant.', 'https://example.com/banner.jpg',
'https://youtube.com/trailer');
```

- **Insertion d'une disponibilité** :

```
INSERT INTO cinema_availabilities (film_id, cinema_name, date, time,
createdAt, updatedAt) VALUES

(1, 'UGC Ciné Cité Les Halles', '2025-01-18', '18:30:00', NOW(), NOW());
```

Développement du backend avec Node.js

Le backend de **Central Cinema** a été développé en utilisant **Node.js** et des bibliothèques modernes pour fournir une API REST performante et évolutive, qui interagit avec la base de données MySQL pour gérer les informations des films et leurs disponibilités dans les cinémas.

1. Initialisation du projet Node.js

Nous avons commencé par initialiser un projet Node.js en créant un fichier `package.json` :

```
cd cinema-backend
npm init -y
```

Cela permet de gérer les dépendances et configurations du projet.

2. Installation des dépendances

Les bibliothèques suivantes ont été installées pour répondre aux besoins spécifiques du backend :

```
npm install express mysql2 sequelize body-parser cors
npm install --save-dev nodemon
```

Express : Utilisé pour créer le serveur et gérer les routes. Nous l'avons choisi pour sa simplicité et sa popularité dans le développement d'API REST.
MySQL2 : Permet d'interagir avec la base de données MySQL. Il est rapide, maintenu et compatible avec Sequelize.
Sequelize : Un ORM (Object-Relational Mapping) qui simplifie la gestion des requêtes SQL et permet de travailler avec des modèles plutôt que des requêtes SQL brutes.
Body-parser : Utilisé pour analyser les corps des requêtes HTTP (ex. JSON).
Cors : Permet de gérer les politiques de partage de ressources entre origines (CORS), nécessaires pour autoriser les requêtes depuis le frontend.
Nodemon : Utilisé en développement pour redémarrer automatiquement le serveur lorsque des modifications sont apportées au code.

3. Initialisation de Sequelize

Nous avons initialisé Sequelize pour structurer notre projet et faciliter la gestion de la base de données :

```
npx sequelize-cli init
```

Cela a généré une architecture organisée avec les dossiers suivants :

- **models** : Pour les modèles Sequelize représentant les tables de la base de données.
- **migrations** : Pour gérer les modifications et la création des tables dans la base de données.
- **config** : Pour configurer les paramètres de connexion à la base de données.

4. Configuration de Sequelize

Nous avons configuré Sequelize en mettant à jour le fichier config/config.json pour se connecter à notre base de données MySQL. La configuration pour l'environnement de développement est la suivante :

```
{
  "development": {
    "username": "root",
    "password": "password",
    "database": "cinema_db",
    "host": "127.0.0.1",
    "dialect": "mysql"
  }
}
```

Nous avons choisi Sequelize car il :

- Simplifie les interactions avec MySQL en utilisant JavaScript.
- Permet de définir des modèles clairs et des associations entre les tables.
- Fournit des outils puissants comme les migrations pour gérer les modifications de la base de données.

5. Création des modèles

Nous avons créé deux modèles principaux pour représenter les tables films et cinema_availabilities.

5.1 Modèle Film

La table films contient les informations principales sur chaque film. Nous avons utilisé Sequelize pour générer ce modèle :

```
npx sequelize-cli model:generate --name Film --attributes
titre:string,duree:integer,langue:string,sous_titres:string,realisateur:str
ing,acteurs:text,age_minimum:integer
```

Le modèle généré est ensuite ajusté pour inclure des validations supplémentaires si nécessaire.

5.2 Modèle CinemaAvailability

La table cinema_availabilities gère les disponibilités des films dans les cinémas, incluant le nom du cinéma, la date, et l'heure. Elle remplace l'ancienne table programmation. Voici la commande utilisée pour générer le modèle :

```
npx sequelize-cli model:generate --name CinemaAvailability --attributes
film_id:integer,cinema_name:string,date:date,time:time
```

Nous avons ensuite ajouté les relations nécessaires dans les modèles pour lier `CinemaAvailability` à `Film` via une clé étrangère (`film_id`).

6. Migration de la base de données

Une fois les modèles définis, nous avons créé les tables correspondantes dans la base de données en exécutant les migrations :

```
npx sequelize-cli db:migrate
```

7. Configuration du serveur Express

Nous avons créé un serveur avec Express pour gérer les requêtes du frontend. Voici le fichier server.js :

```
const express = require('express');
```

```
const bodyParser = require('body-parser');
const cors = require('cors');
const { sequelize } = require('./models');

const app = express();
app.use(bodyParser.json());
app.use(cors());

// Ajout des routes pour les films et disponibilités
const filmRoutes = require('./routes/films'); app.use('/films',
filmRoutes);

const PORT = 5000;
app.listen(PORT, async () => {
  console.log(`Server running on http://localhost:${PORT}`);
  try {
    await sequelize.authenticate();
    console.log('Database connected!');
  } catch (err) {
    console.error('Database connection failed:', err);
  }
});
```

Nous avons choisi Express pour sa simplicité et sa compatibilité avec Sequelize. Il permet de créer rapidement des API REST performantes.

Nous testons en exécutant le serveur :

```
nodemon server.js
```

8. Pourquoi ces technologies ?

Technologie	Raison du choix
Node.js	Haute performance pour les applications backend grâce à son architecture non-bloquante.
Express	Framework léger et flexible pour créer des API REST.
Sequelize	Gestion facile des bases de données relationnelles avec des modèles en JavaScript.

MySQL	Base de données robuste et bien adaptée aux applications nécessitant des relations complexes.
Postman	Outil convivial pour tester et déboguer les API.

Développement des routes API

1. Création des routes pour les films

Nous créons un répertoire `routes` et un fichier `films.js` :

```
mkdir routes
touch routes/films.js
```

Nous ajoutons les routes suivantes dans `routes/films.js` :

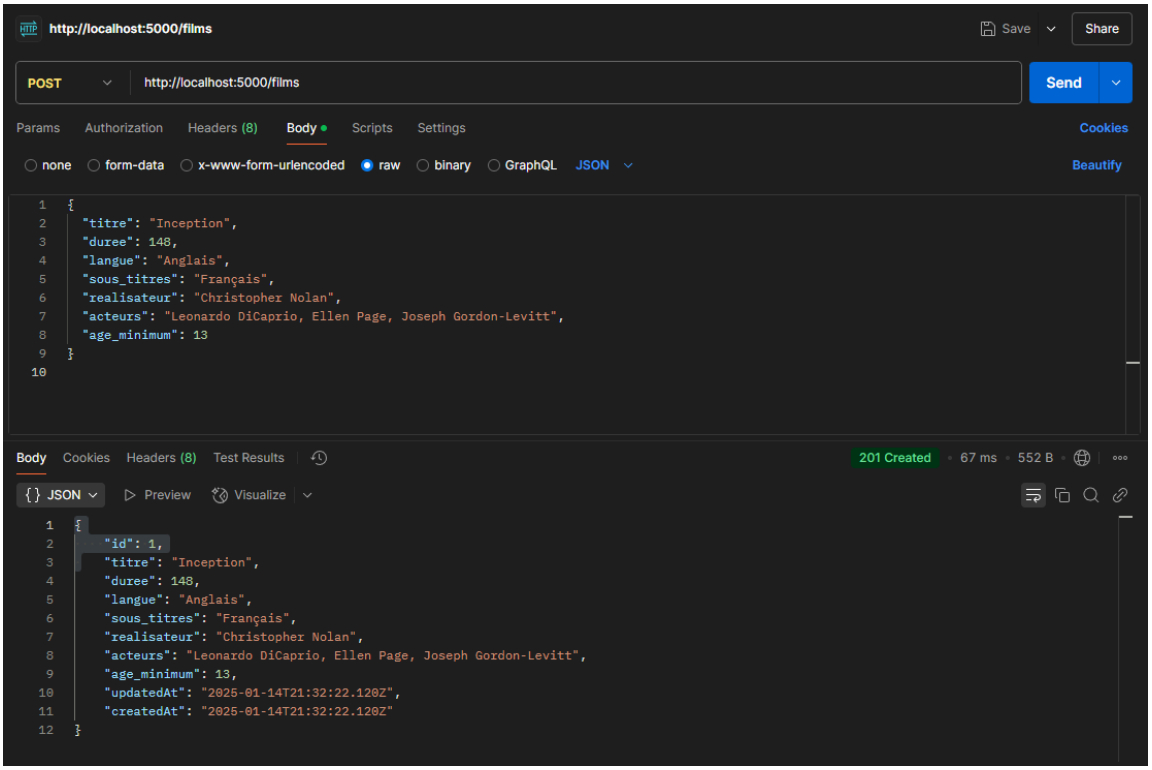
```
const express = require('express');
const router = express.Router();
const { Film } = require('../models');

// Ajouter un film
router.post('/', async (req, res) => {
  try {
    const film = await Film.create(req.body);
    res.status(201).json(film);
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});

// Lister les films
router.get('/', async (req, res) => {
  try {
    const films = await Film.findAll();
    res.status(200).json(films);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

module.exports = router;
```

Nous connectons les routes dans `server.js` :

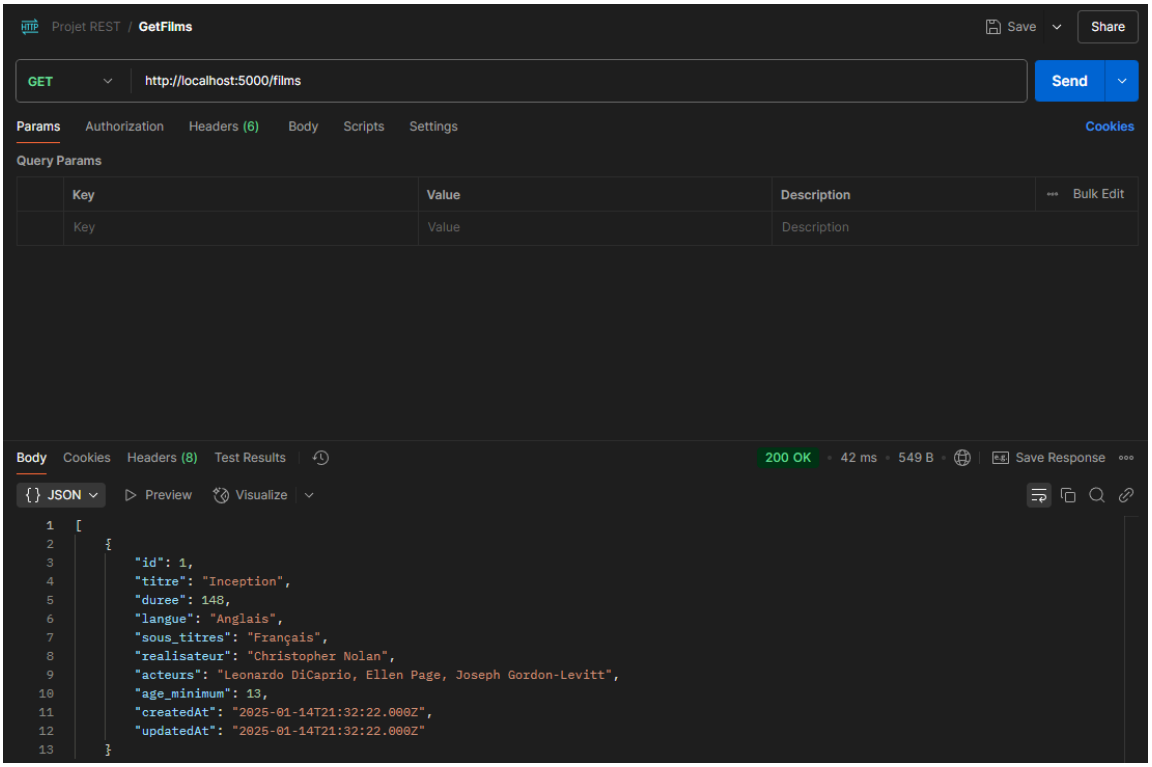
```
const filmRoutes = require('./routes/films');
app.use('/films', filmRoutes);
```

Nous testons avec Postman :

- **POST** `http://localhost:5000/films` : Permet d'ajouter un film.



- **GET** `http://localhost:5000/films` : Permet de lister les films.



Développement du frontend avec React

1. Initialisation du projet React

Nous passons dans le répertoire frontend et créons un projet React :

```
cd ../cinema-frontend
npx create-react-app .
npm install axios
```

2. Développement des composants

Formulaire d'ajout de film (fichier `AddFilm.js`) :

```
import React, { useState } from 'react';
import axios from 'axios';

function AddFilm() {
  const [film, setFilm] = useState({ titre: '', duree: '', langue: '',
realisateur: '' });

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      await axios.post('http://localhost:5000/films', film);
      alert('Film ajouté avec succès !');
    } catch (err) {
      alert('Erreur : ' + err.message);
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <input placeholder="Titre" onChange={(e) => setFilm({ ...film, titre:
e.target.value })} />
      <button type="submit">Ajouter</button>
    </form>
  );
}
```

```
export default AddFilm;
```

Liste des films (fichier `FilmList.js`) :

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';

function FilmList() {
  const [films, setFilms] = useState([]);

  useEffect(() => {
    const fetchFilms = async () => {
      const response = await axios.get('http://localhost:5000/films');
      setFilms(response.data);
    };
    fetchFilms();
  }, []);

  return (
    <div>
      {films.map((film) => (
        <div key={film.id}>
          <h2>{film.titre}</h2>
          <p>Durée : {film.duree} minutes</p>
        </div>
      ))}
    </div>
  );
}

export default FilmList;
```

3. Architecture du Frontend

3.1 Structure de l'application

L'application frontend a été structurée en plusieurs composants React, permettant une réutilisation et une organisation efficace du code. Voici les principaux composants :

- **HomePage** : Affiche les films disponibles, un carrousel de films récents et une navigation conviviale.
- **FilmList** : Gère la liste des films affichés avec des options de recherche.
- **MovieDetails** : Affiche les détails d'un film sélectionné, tels que le synopsis, les horaires et la bande-annonce.
- **AddFilm** : Fournit un formulaire pour ajouter de nouveaux films dans la base de données.
- **Navigation** : Ajout d'un bouton ou icône permettant de retourner à la page d'accueil depuis la page de détails.

Chaque composant est lié à une route spécifique grâce à **React Router**, garantissant une navigation fluide entre les différentes pages.

4. Technologies Utilisées

4.1 React

- **Pourquoi React ?** React est une bibliothèque JavaScript populaire qui permet de créer des interfaces utilisateur réactives et dynamiques. Elle a été choisie pour ses avantages tels que :
    - **Composants réutilisables** : Facilite le développement et la maintenance.
    - **Performance** : Grâce à son DOM virtuel, React optimise les mises à jour et le rendu.
    - **Écosystème riche** : De nombreuses bibliothèques et outils s'intègrent facilement (ex. React Router).
- **Fonctionnalités clés implémentées avec React :**
    - Gestion des états locaux via `useState`.
    - Effets secondaires pour les appels API via `useEffect`.
    - Gestion des routes avec `react-router-dom`.

4.2 Axios

- **Pourquoi Axios ?** Axios est utilisé pour effectuer des requêtes HTTP vers le backend. Nous l'avons choisi pour sa simplicité et ses fonctionnalités avancées :
    - Support des requêtes `GET`, `POST`, etc.
    - Gestion simplifiée des erreurs.

    - Configuration facile des en-têtes (par ex. pour l'authentification si nécessaire).
- **Exemple d'utilisation :**
    - Requêtes pour récupérer les détails d'un film (`GET`).
    - Envoi des données lors de l'ajout d'un film (`POST`).

4.3 Font Awesome

- **Pourquoi Font Awesome ?** Font Awesome est utilisé pour les icônes (ex. l'icône maison pour "Retour à l'accueil"). Cette bibliothèque a été choisie pour :
    - Sa richesse en icônes prêtes à l'emploi.
    - Son intégration facile avec React.

5. Interface Utilisateur (UI/UX)

5.1 Caractéristiques principales

- **Design moderne et intuitif** :
    - Utilisation de CSS pour un rendu visuel soigné et adapté aux besoins (ex. cartes de films, horaires sous forme de boîtes cliquables).
    - Ajout d'une **palette de couleurs cinématographiques** (noir, or, rouge).
- **Réactivité** :
    - Disposition fluide avec des grilles flexibles (`flexbox`) et ajustement dynamique des tailles.
- **Accessibilité** :
    - Textes clairs et boutons bien visibles pour une expérience utilisateur optimale.

5.2 Composants stylisés

Les composants ont été stylisés avec un fichier CSS dédié à chaque composant (ex. `MovieDetails.css`). Cela permet une séparation claire entre la logique et la présentation.

6. Fonctionnalités Frontend

Voici les fonctionnalités principales implémentées sur le frontend :

1. **Page d'accueil :**
    - Affichage des films les plus récents sous forme de carrousel.

    - Recherche de films via une barre de recherche dynamique.
2. **Page de détails :**
    - Affichage des informations d'un film (titre, synopsis, durée, réalisateur, bande-annonce, horaires).
    - Groupement des horaires par cinéma, avec un style moderne (boîtes cliquables).
    - Bouton de retour à la page d'accueil (icône maison).
3. **Ajout de films :**
    - Formulaire pour soumettre de nouveaux films avec leurs informations.
    - Validation des champs avant l'envoi.

7. Raisons du Choix des Technologies

Technologie	Raison du choix
React	Création d'interfaces dynamiques et réactives.
Axios	Gestion des requêtes HTTP avec simplicité et efficacité.
React Router	Navigation fluide entre les pages.
Font Awesome	Utilisation d'icônes modernes pour améliorer l'expérience utilisateur.
CSS	Stylisation et personnalisation des composants pour un rendu professionnel.

Conclusion

Le projet **Central Cinema** est un système complet et bien structuré qui offre :

- Une interface moderne et agréable pour les utilisateurs.
- Une architecture backend robuste et évolutive.
- Une base de données relationnelle bien conçue pour garantir l'intégrité et la performance.

Grâce à l'utilisation de technologies modernes et à une intégration fluide entre les différentes parties (frontend, backend, base de données), **Central Cinema** est prêt à répondre aux besoins actuels tout en offrant une base solide pour des améliorations futures.