Ecole Marocaine des Sciences de l'ingenieur (EMSI)

Ecole d'ingénieur

RAPPORT DE PROJET DE FIN D'ANNÉE

Microservices & Spring Boot

Système de Réservation de Cinéma

Réalisé par :
TABRANI El Mehdi
OUAZZANE Imade Eddine

Encadré par :
Abdelaziz Ettaoufik

Année Universitaire : 2025 - 2026

Remerciements

Je tiens tout d'abord à remercier mon encadrant, Abdelaziz Ettaoufik, pour son aide précieuse, ses conseils avisés et son suivi tout au long de ce projet.
Je remercie également le corps professoral de [Votre Institut] pour la qualité de l'enseignement qui m'a permis de réaliser ce travail.
Enfin, un grand merci à ma famille et à mes amis pour leur soutien indéfectible.

Résumé

Ce projet consiste en la conception et le développement d'une plateforme de réservation de billets de cinéma en ligne, basée sur une architecture microservices. L'objectif principal est de fournir un système scalable, modulaire et performant capable de gérer les films, les réservations et les paiements de manière indépendante.
L'application est construite autour de Spring Boot pour le backend, utilisant Spring Cloud pour la gestion des microservices (Eureka, API Gateway, Feign Client). Le frontend est développé avec React (Vite) pour offrir une expérience utilisateur moderne et réactive. Les données sont persistées via H2 Database pour le développement, avec une architecture prête pour PostgreSQL en production.
Les fonctionnalités clés incluent la consultation des films, la vérification de la disponibilité, la réservation de sièges et le traitement des paiements.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction Générale

1.1 Contexte du Projet

L'industrie cinématographique a connu une transformation numérique majeure au cours de la dernière décennie. Les systèmes de réservation traditionnels au guichet ne suffisent plus à répondre à la demande croissante des consommateurs pour une accessibilité instantanée et ubiquitaire. Les cinémas modernes nécessitent des systèmes robustes capables de gérer des pics de trafic importants lors des sorties de nouveaux blockbusters. Dans ce contexte, une architecture monolithique traditionnelle peut montrer ses limites en termes de scalabilité et de maintenance. Une approche basée sur les microservices offre une solution élégante, permettant de faire évoluer chaque composant du système (gestion des films, réservations, paiements) indépendamment.

1.2 Problématique

Comment concevoir un système de réservation de cinéma qui soit à la fois :

- **Scalable** : Capable de supporter une augmentation de la charge sur un module spécifique (ex : service de réservation) sans affecter les autres.
- **Maintenable** : Permettant des mises à jour rapides et isolées.
- **Résilient** : Assurant que la panne d'un service (ex : paiement) n'empêche pas la consultation des films.

1.3 Objectifs du Projet

Les objectifs principaux de ce projet sont :

1. Développer une architecture microservices utilisant l'écosystème Spring Cloud.
2. Implémenter la communication inter-services (synchrone et asynchrone).
3. Fournir une interface utilisateur intuitive et réactive.
4. Mettre en œuvre les bonnes pratiques de développement (RESTful API, Clean Code).

1.4 Organisation du Rapport

Ce rapport est structuré comme suit :

— Le chapitre 2 detailles l'analyse des besoins et la spécification fonctionnelle.
— Le chapitre 3 présente l'architecture technique et les choix technologiques.
— Le chapitre 4 décrit la conception détaillée et les modèles de données.
— Le chapitre 5 expose la réalisation et l'implémentation du système.
— Enfin, nous conclurons par les perspectives d'évolution.

Chapitre 2

Analyse et Spécification des Besoins

2.1 Introduction

L'étape d'analyse est cruciale pour définir le périmètre du projet et s'assurer que la solution répond aux attentes des utilisateurs finaux. Nous avons identifié deux acteurs principaux : l'Administrateur et le Client.

2.2 Identification des Acteurs

Client : Utilisateur final qui souhaite consulter les films à l'affiche et réserver des places.

Administrateur : Responsable de la gestion du catalogue de films et du suivi des réservations.

Système Bancaire (Externe) : Service tiers simulé pour la validation des transactions de paiement.

2.3 Besoins Fonctionnels

2.3.1 Module Gestion des Films

- **Ajouter un film** : Saisir le titre, la description, la durée et le nombre de sièges disponibles.
- **Modifier un film** : Mettre à jour les informations ou le nombre de places.
- **Supprimer un film** : Retirer un film de l'affiche.
- **Lister les films** : Afficher tous les films disponibles avec leurs détails.
- **Rechercher un film** : Filtrer par ID ou par titre.

2.3.2 Module Réservation

- **Vérifier la disponibilité** : S'assurer qu'il reste suffisamment de places avant de confirmer.
- **Créer une réservation** : Enregistrer la demande du client pour un film donné.
- **Annuler une réservation** : Libérer les places réservées.
- **Lister les réservations** : Historique des réservations pour un utilisateur ou global admin.

2.3.3 Module Paiement

— **Traiter le paiement** : Valider le montant et associer le paiement à une réservation.
— **Historique des paiements** : Consulter les transactions passées.

2.4 Besoins Non-Fonctionnels

— **Performance** : Temps de réponse $<$ 200ms pour la consultation.
— **Disponibilité** : Le système doit être résilient aux pannes partielles.
— **Sécurité** : Validation des données entrantes.

2.5 Diagrammes de Cas d'Utilisation

FIGURE 2.1 – Diagramme de Cas d'Utilisation Global

2.5.1 Description des Cas d'Utilisation Prioritaires

CU01 : Réserver une place

1. Le client consulte la liste des films.
2. Il sélectionne un film.
3. Il précise le nombre de places.
4. Le système vérifie la disponibilité (appel synchrone FilmService).
5. Si OK, le système crée la réservation (état "EN ATTENTE").
6. Le système redirige vers le paiement.
7. Une fois payé, la réservation passe à "CONFIRMÉE".

Chapitre 3

Architecture et Conception

3.1 Architecture Microservices

Nous avons opté pour une architecture microservices pour garantir la séparation des responsabilités. Le système est composé des services suivants :

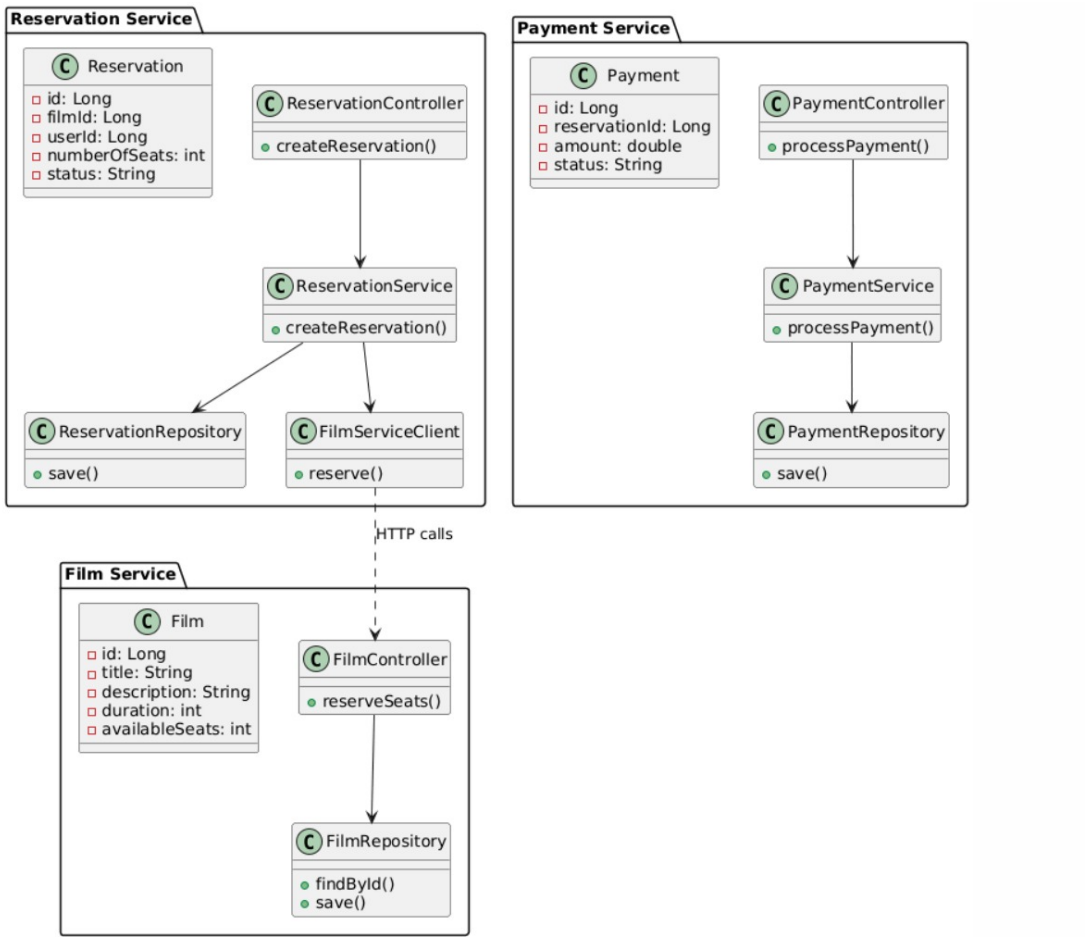

FIGURE 3.1 – Architecture Globale du Système

3.1.1 Composants de l'Architecture

1. **Discovery Server (Eureka)** : Annuaire des services. Chaque microservice s'enregistre ici au démarrage, permettant aux autres de le trouver sans connaître son IP physique.
2. **API Gateway** : Point d'entrée unique pour le frontend. Il route les requêtes vers les bons microservices (ex : /api/films -> film-service).
3. **Film Service** : Gère le domaine "Film".
4. **Reservation Service** : Cœur métier, orchestre la réservation.
5. **Payment Service** : Gère les transactions financières.

3.2 Diagramme de Classes (Modèle de Données)

Puisque nous sommes en microservices, chaque service possède sa propre base de données. Il n'y a pas de base de données monolithique partagée.

3.2.1 Film Service Model

- **Film**
  - id : Long (PK)
  - title : String
  - description : String
  - duration : int (minutes)
  - availableSeats : int

3.2.2 Reservation Service Model

- **Reservation**
  - id : Long (PK)
  - filmId : Long (FK logique vers Film)
  - userId : Long
  - numberOfSeats : int
  - status : Enum (PENDING, CONFIRMED)

3.2.3 Payment Service Model

- **Payment**
  - id : Long (PK)
  - reservationId : Long
  - amount : double
  - status : String

Chapitre 4

Réalisation et Implémentation

4.1 Environnement de Développement

— **Langage** : Java 17

FIGURE 4.1 – JAVA

— **Framework** : Spring Boot 3.2.5

FIGURE 4.2 – spring boot

— **Build Tool** : Maven

— **Frontend** : React.js, Vite, Tailwind CSS

FIGURE 4.3 – Reserve Seat Film

— **IDE** : IntelliJ IDEA / VS Code

FIGURE 4.4 – Visual Studio Code

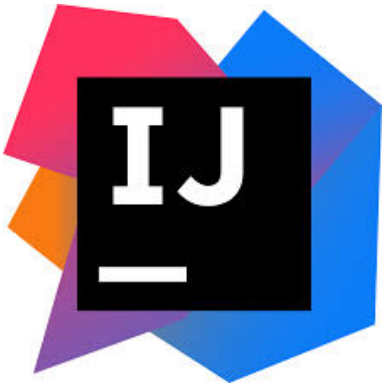

FIGURE 4.5 – intelij

4.2 Implémentation Backend

4.2.1 Service Film (FilmService)

Ce service expose une API REST pour gérer les films. Il utilise `Spring Data REST` pour générer automatiquement les CRUD, et un contrôleur personnalisé pour la logique métier spécifique.

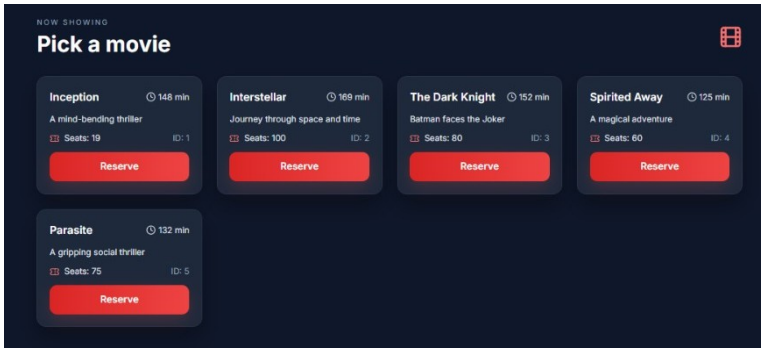

FIGURE 4.6 – Reserve Seat Film

4.2.2 Service Réservation (ReservationService)

Ce service communique avec le FilmService. Nous utilisons OpenFeign pour l'appel distant déclaratif.

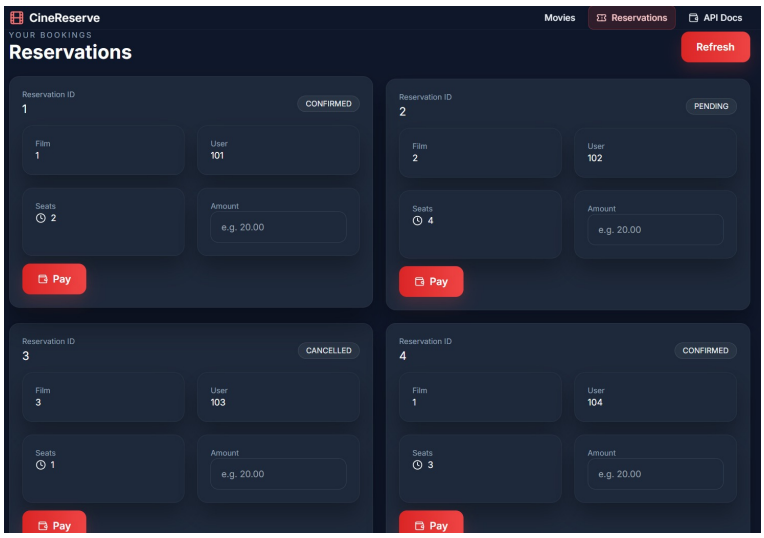

FIGURE 4.7 – Reservations

Chapitre 5

Tests et Déploiement

5.1 Tester avec Swagger UI

Chaque microservice intègre Swagger (OpenAPI) pour la documentation et le test des APIs.

— Film Service : `http://localhost:8181/swagger-ui/`
— Reservation Service : `http://localhost:8182/swagger-ui/`

FIGURE 5.1 – swagger

5.2 Scénarios de Test

5.2.1 Test Nominal : Réservation Réussie

Action : POST sur `/reservations` avec FilmID=1 et Seats=2.
Résultat attendu : Status 200 OK, JSON de la réservation renvoyé, nombre de sièges du film décrémenté de 2.

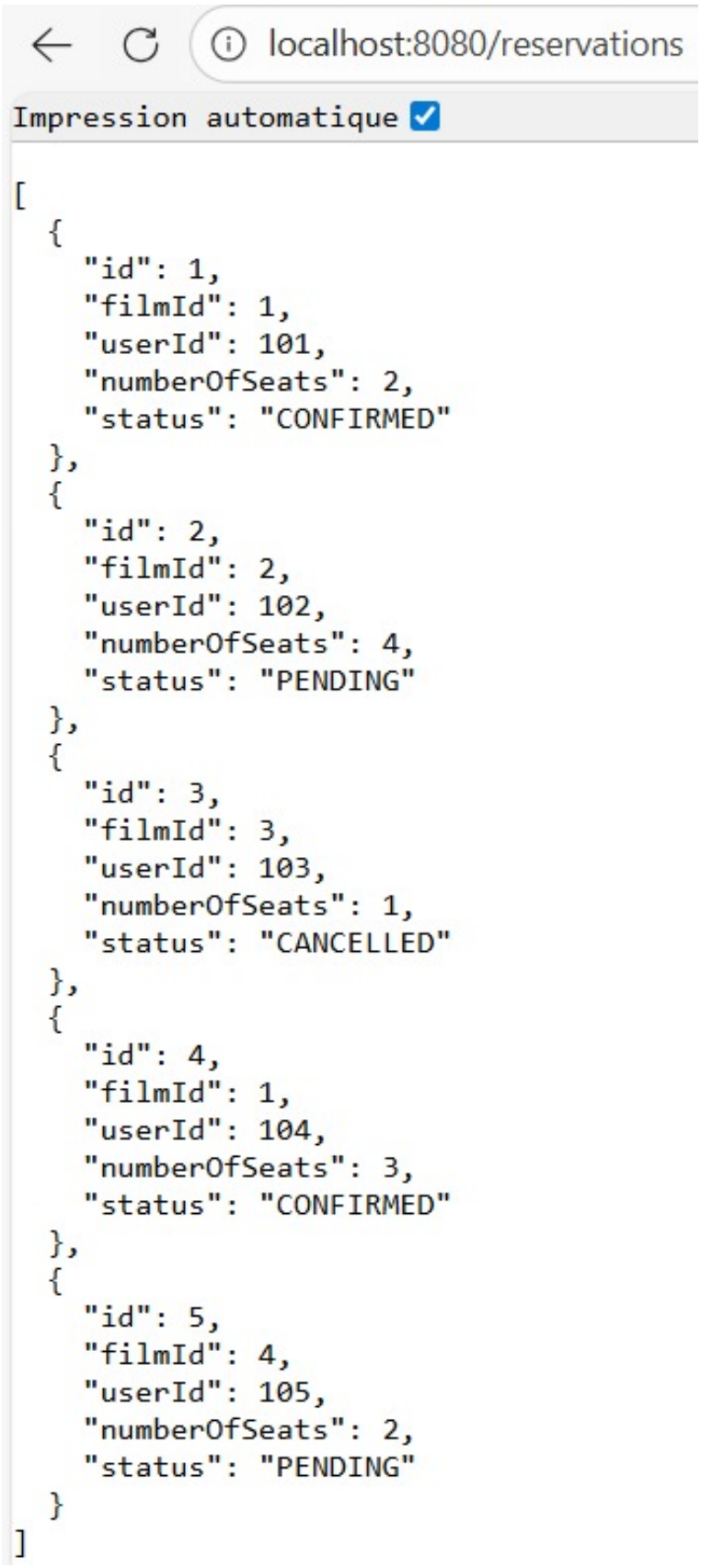

FIGURE 5.2 – Reservation

```
{
  "_embedded" : {
    "films" : [ {
      "title" : "Inception",
      "description" : "A mind-bending thriller",
      "duration" : 148,
      "availableSeats" : 120,
      "_links" : {
        "self" : {
          "href" : "http://192.168.149.1:8181/films/1"
        },
        "film" : {
          "href" : "http://192.168.149.1:8181/films/1"
        }
      }
    }, {
      "title" : "Interstellar",
      "description" : "Journey through space and time",
      "duration" : 169,
      "availableSeats" : 100,
      "_links" : {
        "self" : {
          "href" : "http://192.168.149.1:8181/films/2"
        },
        "film" : {
          "href" : "http://192.168.149.1:8181/films/2"
        }
      }
    }, {
      "title" : "The Dark Knight",
      "description" : "Batman faces the Joker",
      "duration" : 152,
      "availableSeats" : 80,
      "_links" : {
        "self" : {
          "href" : "http://192.168.149.1:8181/films/3"
        },
        "film" : {
          "href" : "http://192.168.149.1:8181/films/3"
        }
      }
    }, {
      "title" : "Spirited Away",
      "description" : "A magical adventure",
      "duration" : 125,
      "availableSeats" : 60,
      "_links" : {
        "self" : {
          "href" : "http://192.168.149.1:8181/films/4"
        },
        "film" : {
          "href" : "http://192.168.149.1:8181/films/4"
```


FIGURE 5.3 – Films

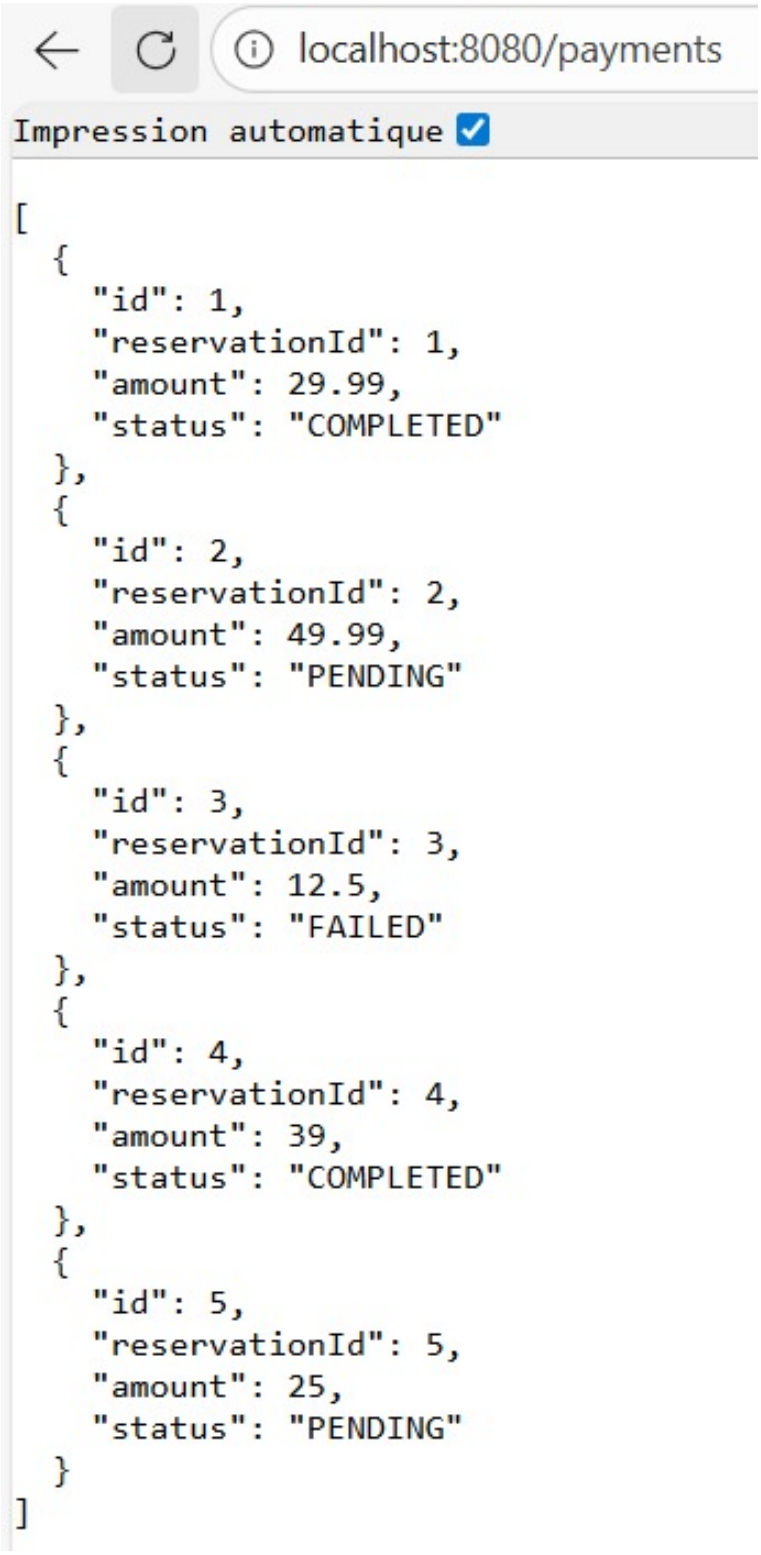

FIGURE 5.4 – PAYMENT

5.2.2 Test d'Erreur : Sièges Insuffisants

Action : POST sur `/reservations` avec demande de 1000 sièges.
Résultat attendu : Status 400 Bad Request, message "Not enough seats".

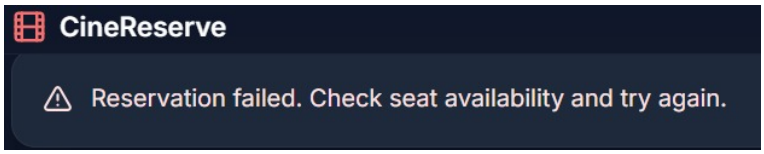

FIGURE 5.5 – Seat insuffisent

5.2.3 Page d'Accueil

La page d'accueil affiche la liste des films sous forme de cartes.

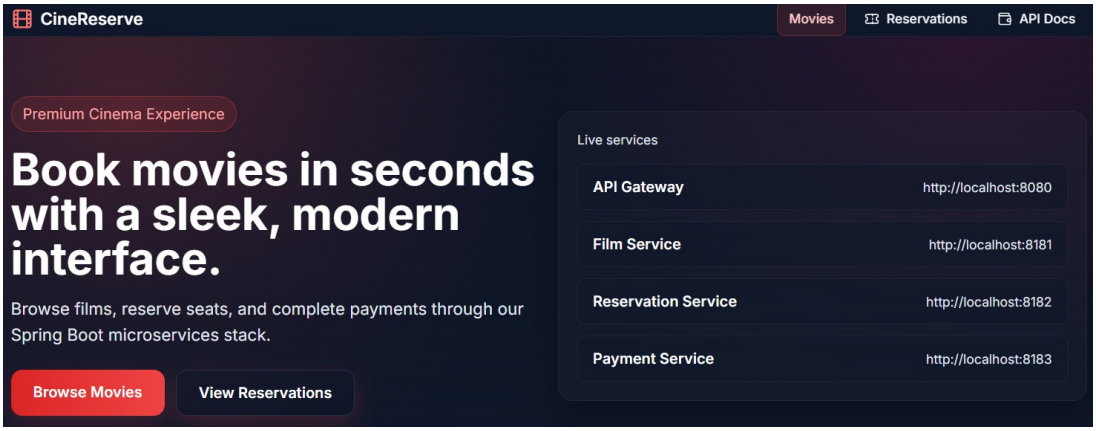

FIGURE 5.6 – Page d'Accueil et Routes

5.2.4 Formulaire de Réservation

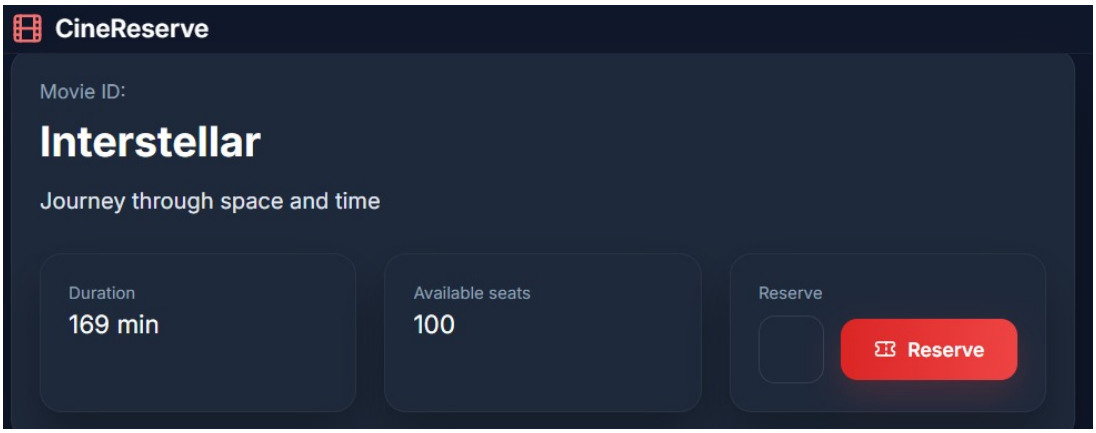

FIGURE 5.7 – Interface de Réservation

Chapitre 6

Conclusion et Perspectives

Ce projet nous a permis de mettre en pratique les concepts avancés de l'architecture logicielle distribuée. L'utilisation de Spring Boot et Spring Cloud a grandement facilité la mise en place de l'infrastructure microservices.

6.1 Bilan Technique

Nous avons réussi à implémenter :

— 3 microservices fonctionnels et indépendants.
— Une communication inter-service robuste avec Feign.
— Un frontend moderne communiquant avec le backend.

6.2 Limites et Améliorations Futures

Pour aller plus loin, nous pourrions :

— Ajouter une couche de sécurité avec **Keycloak** ou **Spring Security (JWT)**.
— Mettre en place un **Config Server** pour centraliser la configuration.
— Utiliser **Docker** et **Kubernetes** pour l'orchestration des conteneurs.
— Ajouter du monitoring avec **Prometheus** et **Grafana**.