CINEPHORIA

Documentation Technique – Cinéphoria

Application Web

1. Architecture Globale

L'application Cinéphoria est construite sur le stack MERN (MongoDB, Express, React, Node.js), avec l'ajout de Prisma comme ORM et MySQL comme base de données principale. Cette architecture a été choisie pour plusieurs raisons :

- **Écosystème JavaScript unifié** : L'utilisation de JavaScript/Node.js côté serveur et React côté client permet une cohérence dans le langage de programmation utilisé à travers toute l'application.
- **Flexibilité et scalabilité** : Le stack MERN offre une grande flexibilité pour le développement d'applications web modernes et est capable de gérer efficacement la montée en charge.
- **Performances** : Node.js est connu pour ses performances élevées, particulièrement pour les applications nécessitant beaucoup d'opérations I/O.
- **Riche écosystème** : L'écosystème npm offre une multitude de packages et d'outils pour accélérer le développement.

Ce stack permet également de réaliser les applications web, mobile et bureautique demandées dans le même écosystème technologique, facilitant ainsi le partage de code et la maintenance.

1.1 Utilisation de Prisma et MySQL

Prisma a été choisi comme ORM (Object-Relational Mapping) pour interagir avec la base de données MySQL. Les raisons de ce choix sont :

- **Type safety** : Prisma génère des types TypeScript basés sur le schéma de la base de données, réduisant ainsi les erreurs liées aux types.
- **Migrations automatisées** : Prisma facilite la gestion des migrations de base de données.
- **Requêtes optimisées** : Prisma génère des requêtes SQL optimisées.

- **Modélisation relationnelle** : MySQL est utilisé pour stocker les données fortement structurées et relationnelles comme les utilisateurs, les cinémas, les films, etc.

1.2 Utilisation de MongoDB

MongoDB est utilisé pour gérer les reviews, bookings et incidents. Ce choix est justifié par :

- **Flexibilité du schéma** : Ces entités peuvent avoir des structures plus flexibles qui évoluent dans le temps.
- **Performances pour les opérations de lecture/écriture** : MongoDB excelle dans les opérations fréquentes de lecture/écriture, ce qui est crucial pour ces fonctionnalités.
- **Scalabilité horizontale** : MongoDB permet une scalabilité horizontale facile, utile si ces fonctionnalités connaissent une forte croissance.

2. Architecture Frontend-Backend

L'application est construite avec une séparation claire entre le frontend et le backend :

- **Frontend** : Développé avec React, géré indépendamment dans le dossier frontend.
- **Backend** : Une API RESTful développée avec Express.js.

Cette séparation offre plusieurs avantages :

- Développement et déploiement indépendants
- Meilleure organisation du code
- Possibilité de scaler indépendamment le frontend et le backend

3. Stratégie de Tests et CI/CD

3.1 Tests

L'application utilise plusieurs niveaux de tests :

- **Tests unitaires** : Pour tester des fonctions ou composants isolés.
- **Tests fonctionnels** : Pour tester des fonctionnalités spécifiques.
- **Tests d'intégration** : Pour tester l'interaction entre différentes parties de l'application.

3.2 CI/CD

Le fichier deploy-feature.yml dans le dossier .github/workflows suggère l'utilisation de GitHub Actions pour l'intégration continue et le déploiement continu. Cela permet d'automatiser les tests et le déploiement à chaque push sur certaines branches.

3.3 Utilisation de Docker

Docker est utilisé pour les tests, offrant plusieurs avantages :

- **Environnement cohérent** : Garantit que les tests s'exécutent dans un environnement identique, quel que soit le système.
- **Isolation** : Chaque test peut s'exécuter dans un conteneur isolé, évitant les conflits.
- **Facilité de configuration** : Les dépendances et configurations nécessaires sont définies dans les Dockerfiles.

4. Dépendances Principales

Voici une liste des principales dépendances utilisées dans le projet :

Backend

- Express.js
- Prisma
- MySQL2
- MongoDB
- Mongoose
- Bcryptjs
- Jsonwebtoken
- Cors
- Cookie-parser
- Stripe
- Nodemailer

Frontend

- React
- React Router
- Redux Toolkit
- Axios
- Tailwind CSS
- React-toastify
- @stripe/react-stripe-js
- React-datepicker

Outils de développement

- Vite

- ESLint
- Prettier
- Cypress
- Vitest
- Docker
- Concurrently

Diagramme d’utilisation

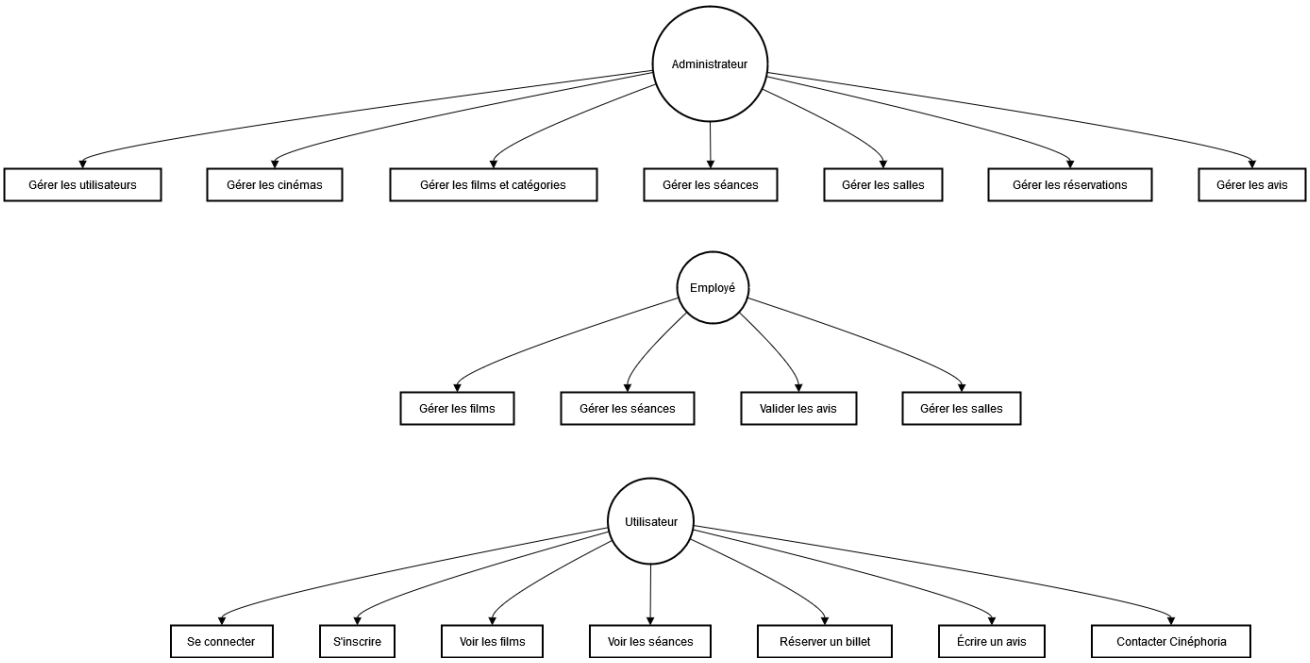
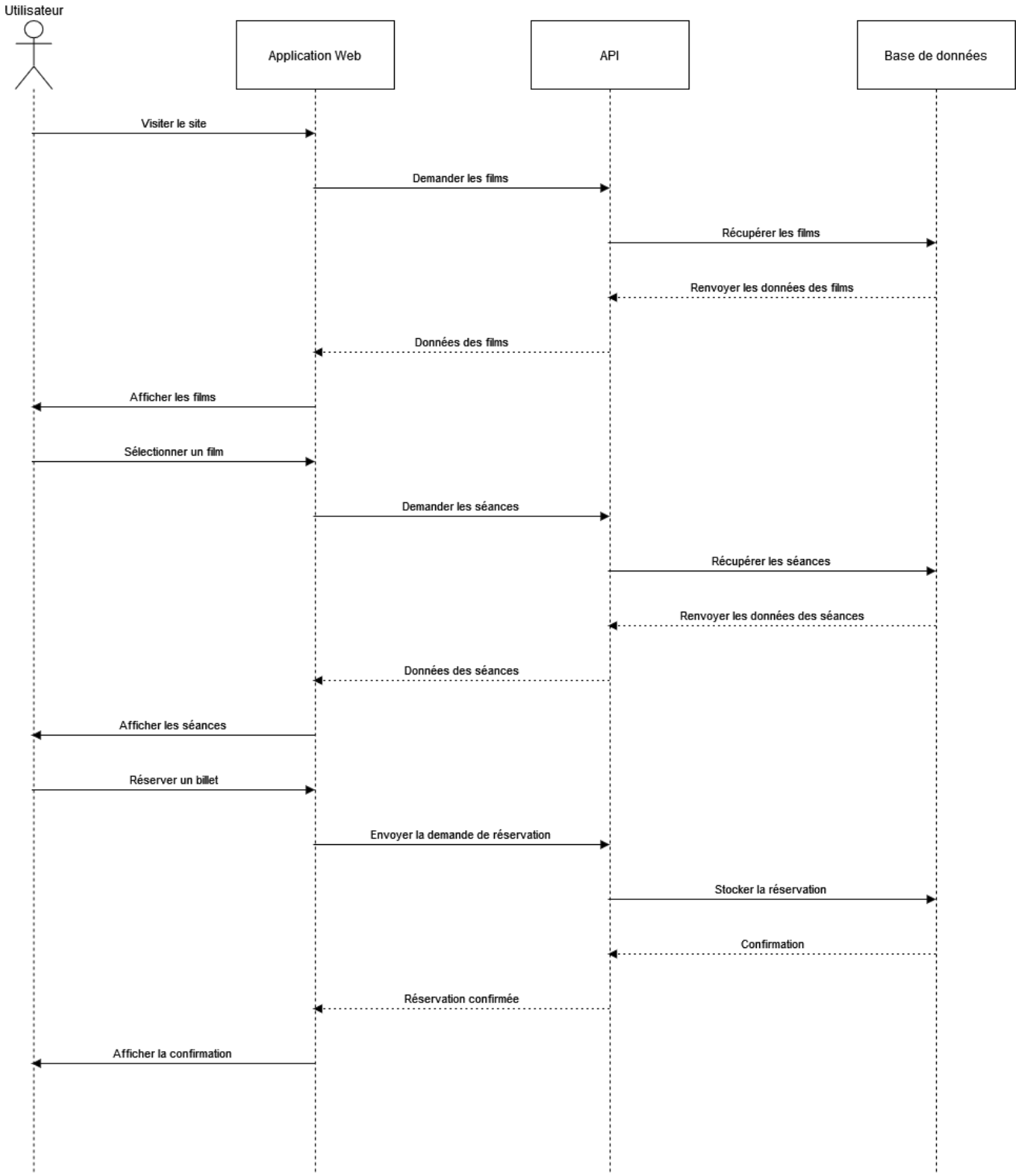


Diagramme de séquence



Application Mobile

5. Architecture Globale

L'application mobile Cinéphoria est développée en utilisant React Native, ce qui permet de partager une grande partie de la logique et du code avec l'application web React. Cette approche offre plusieurs avantages :

- **Réutilisation du code** : Une grande partie de la logique métier peut être partagée entre l'application web et mobile.
- **Cohérence de l'expérience utilisateur** : Facilite la maintenance d'une expérience utilisateur cohérente sur toutes les plateformes.
- **Développement rapide** : L'utilisation de React Native permet un développement rapide et efficace pour iOS et Android avec une seule base de code.

5.1 Fonctionnalités Principales

- Authentification des utilisateurs
- Affichage des séances du jour et à venir
- Génération et affichage de QR codes pour les billets

5.2 Dépendances Principales

- React Native
- Expo
- React Navigation
- Redux Toolkit et React-Redux
- AsyncStorage pour la persistance des données
- react-native-qrcode-svg pour la génération de QR codes

5.3 Partage du Backend

L'application mobile utilise le même serveur backend que l'application web, ce qui permet :

- Une centralisation des données et de la logique métier
- Une maintenance simplifiée du backend
- Une cohérence des données entre les plateformes web et mobile

Diagramme d’utilisation

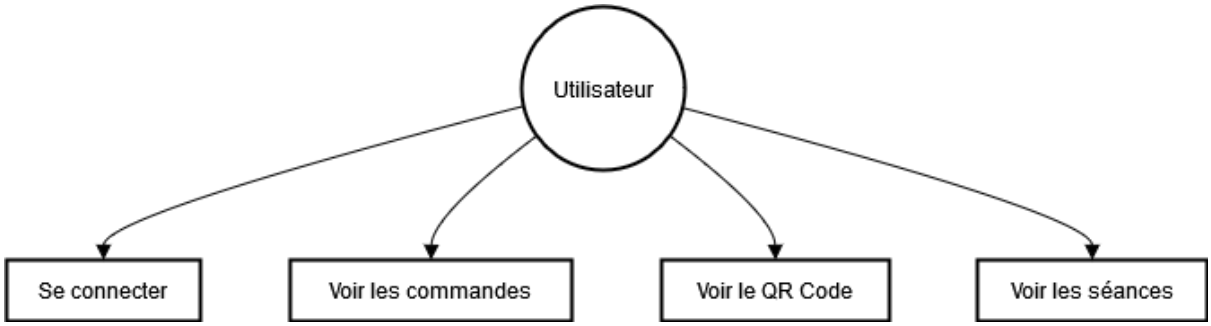
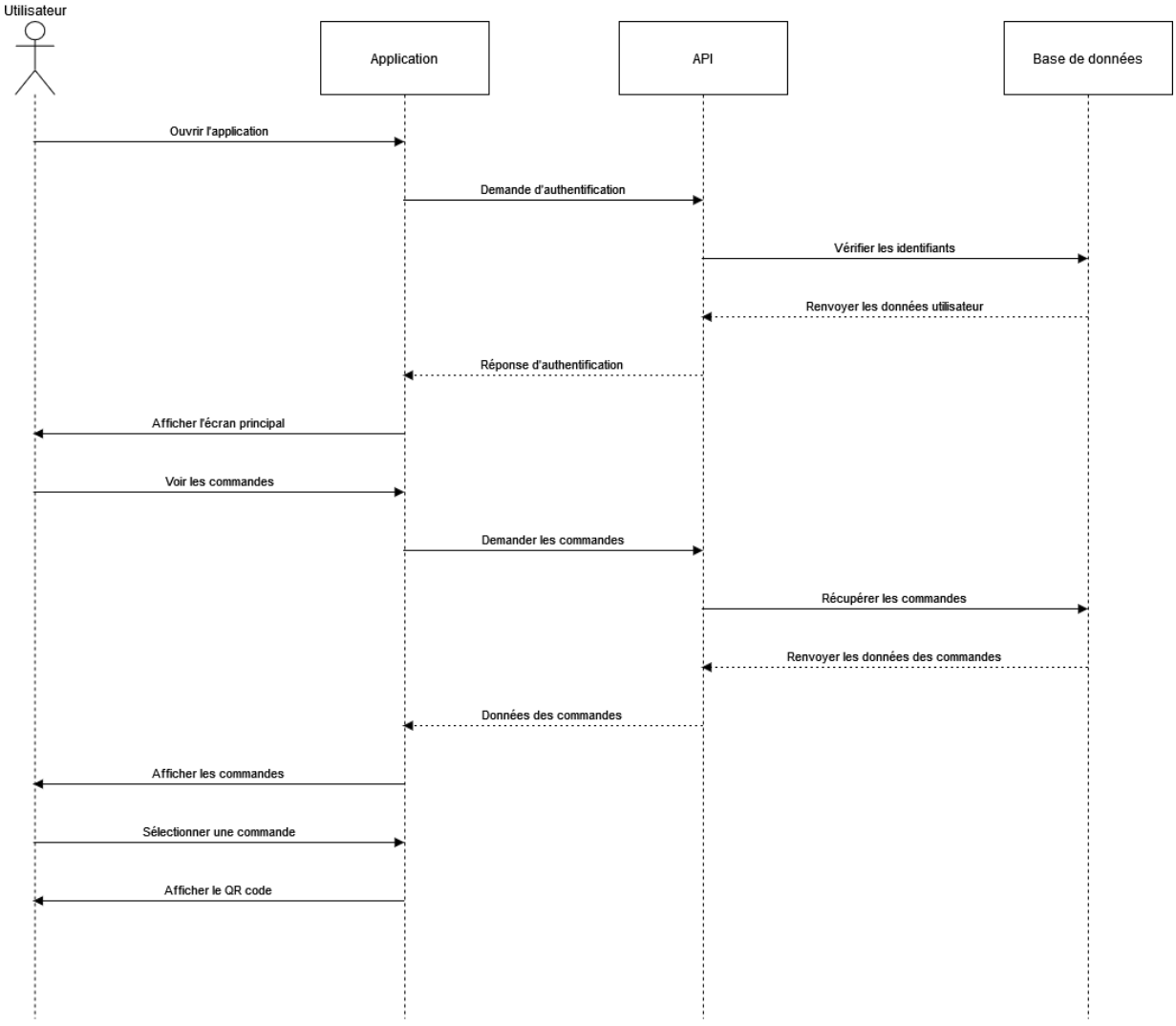


Diagramme de séquence



Application Bureautique

6. Architecture Globale

L'application desktop Cinéphoria est développée avec Electron, ce qui permet d'utiliser les technologies web (HTML, CSS, JavaScript) pour créer une application de bureau cross-platform. Cette approche offre plusieurs avantages :

- **Réutilisation des compétences web** : L'équipe peut utiliser ses compétences en développement web pour créer une application desktop.
- **Cross-platform** : Une seule base de code pour créer des applications pour Windows, macOS et Linux.
- **Intégration avec les API système** : Electron permet d'accéder aux API système natives lorsque nécessaire.

6.1 Fonctionnalités Principales

- Interface pour les employés de Cinéphoria
- Saisie et gestion des incidents dans les salles de cinéma
- Visualisation des informations sur les salles et les séances

6.2 Dépendances Principales

- Electron
- React
- Redux Toolkit et React-Redux
- Electron-store pour le stockage local
- Tailwind CSS pour le styling
- Axios pour les requêtes HTTP

6.3 Partage du Backend

Comme l'application mobile, l'application desktop partage le même backend que l'application web. Cela présente plusieurs avantages :

- Uniformité des données et des processus métier sur toutes les plateformes
- Réduction de la complexité de l'infrastructure backend
- Facilité de mise à jour et de maintenance du système dans son ensemble

7. Avantages de l'Architecture Partagée

L'utilisation d'un backend commun pour les applications web, mobile et desktop offre plusieurs avantages significatifs :

- **Cohérence des données** : Toutes les applications accèdent aux mêmes données, assurant une cohérence à travers les plateformes.

- **Maintenance simplifiée** : Les mises à jour et les corrections de bugs sur le backend sont immédiatement disponibles pour toutes les applications.
- **Développement efficace** : Les développeurs peuvent se concentrer sur l'optimisation d'un seul backend plutôt que de gérer plusieurs systèmes distincts.
- **Scalabilité** : Le backend peut être scalé indépendamment pour répondre aux besoins de toutes les applications.

Cette architecture partagée, combinée à l'utilisation de technologies web modernes sur toutes les plateformes, crée un écosystème cohérent et efficace pour le développement et la maintenance de l'ensemble du système Cinéphoria.

Diagramme d’utilisation

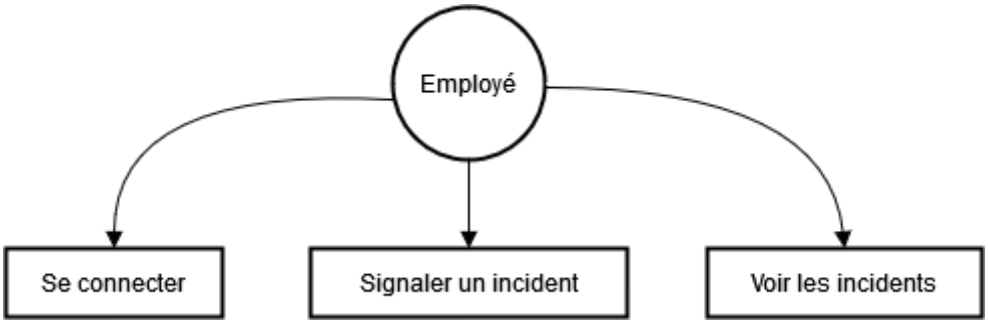
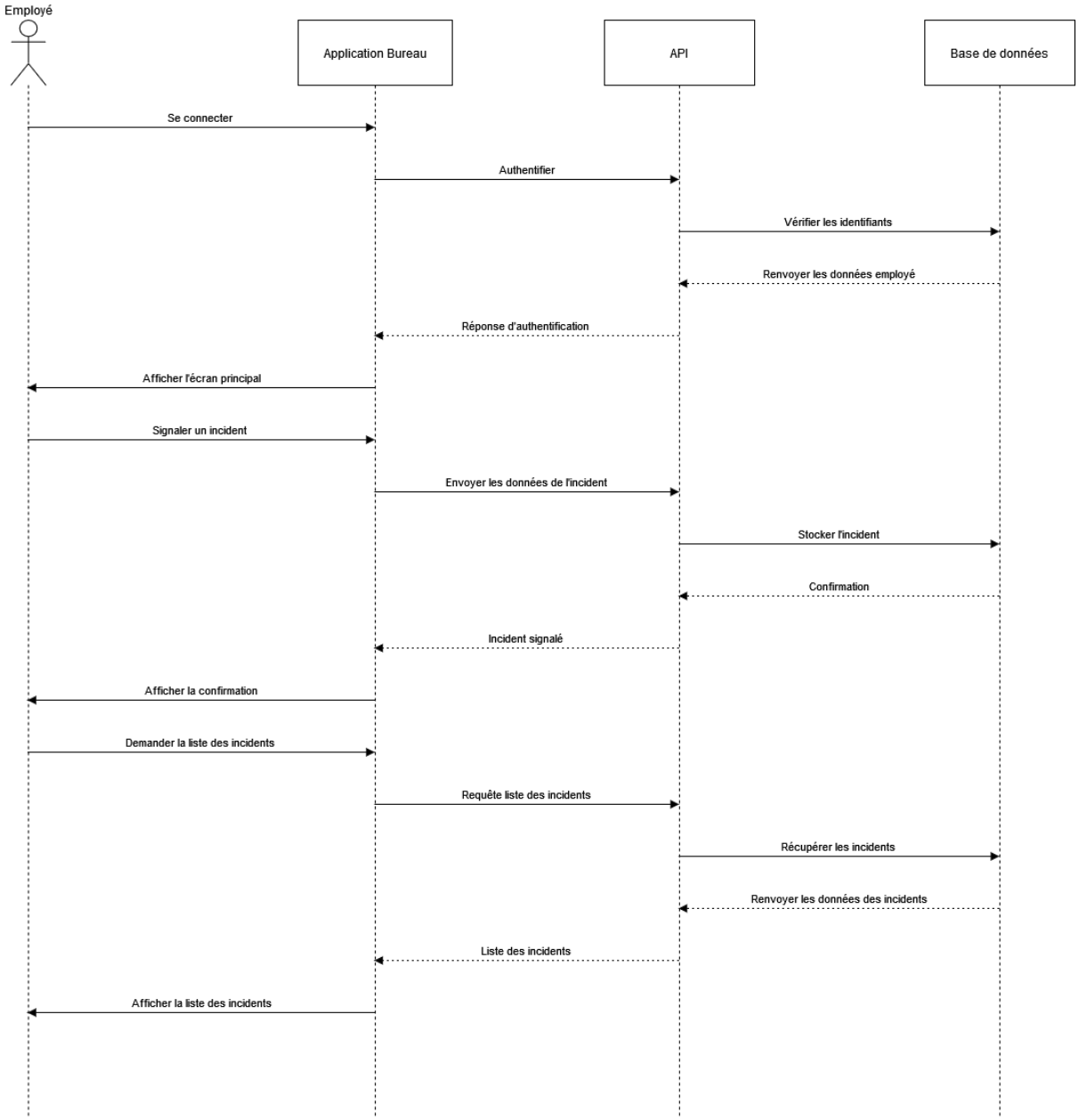
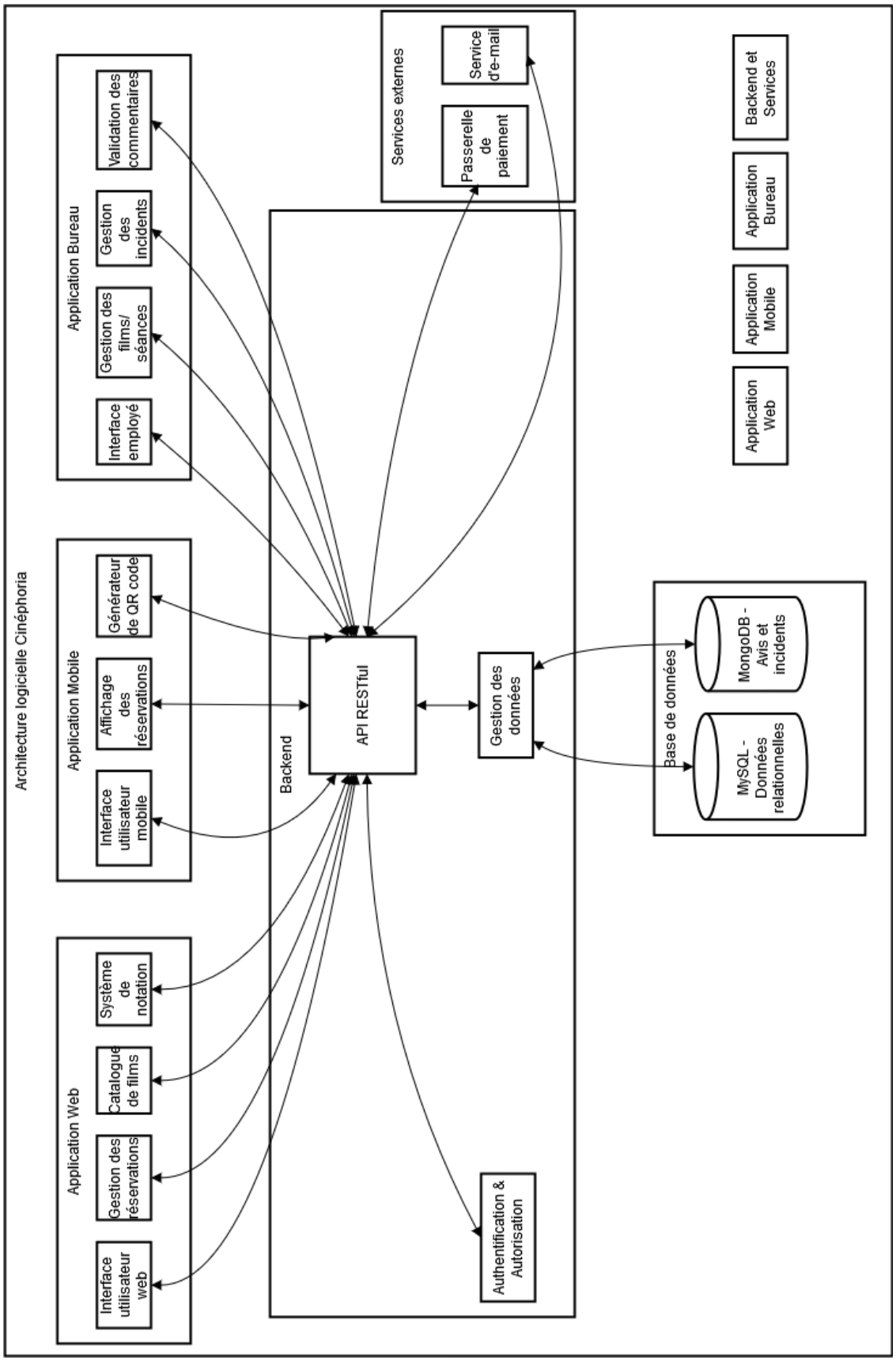


Diagramme de séquence



Architecture globale



Transaction SQL

La transaction SQL a pour objectif d'ajouter un nouveau film dans le système de gestion de cinéma et de programmer simultanément des séances pour ce film dans plusieurs cinémas. C'est un scénario courant lorsqu'un nouveau film est sur le point d'être diffusé et doit être intégré dans le système de réservation.

Fonctionnement de la transaction :

Début de la transaction :

La transaction commence par définir toutes les informations nécessaires pour le nouveau film (titre, description, date de sortie, etc.).

Insertion du film :

Le nouveau film est inséré dans la table Movie avec toutes ses informations.

Association des catégories :

Les catégories spécifiées sont associées au film dans la table de jonction _CategoryToMovie.

Programmation des séances :

Des séances sont programmées pour le nouveau film dans plusieurs cinémas. Pour cet exemple, seules les salles standards sont sélectionnées. Les séances sont programmées à partir de la date de sortie du film, une par jour.

Création des plages horaires :

Pour chaque séance programmée, une plage horaire correspondante est créée dans la table TimeRange.

Vérification :

La transaction vérifie ensuite que toutes les insertions ont été effectuées correctement :

Le film a bien été ajouté

Les catégories ont été associées

Les séances ont été programmées

Les plages horaires ont été créées

Validation ou annulation :

Si toutes les vérifications sont positives, la transaction est validée (commit).

En cas de problème à n'importe quelle étape, toute la transaction est annulée (rollback).

L'intérêt principal de cette approche par transaction est de garantir l'intégrité des données. Soit toutes les opérations réussissent et le nouveau film est complètement intégré au système avec ses séances, soit aucune modification n'est apportée à la base de données. Cela évite les situations où un film serait ajouté sans séances, ou des séances créées sans plages horaires, ce qui pourrait causer des incohérences dans le système de réservation.

Cette transaction permet donc d'automatiser et de sécuriser le processus d'ajout d'un nouveau film au catalogue, en s'assurant que toutes les informations et programmations nécessaires sont en place avant de le rendre disponible pour les réservations.

8. Réflexions initiales technologiques

Lors de la phase de conception du projet Cinéphoria, plusieurs réflexions technologiques ont guidé mes choix :

1. **Choix du stack MERN** :
    - La popularité et la maturité de l'écosystème JavaScript/Node.js
    - La flexibilité de MongoDB pour certaines données (reviews, bookings, incidents)
    - La puissance de React pour créer des interfaces utilisateur dynamiques
    - La performance de Node.js pour le backend
2. **Intégration de MySQL avec Prisma** :
    - Besoin d'une base de données relationnelle pour les données structurées (utilisateurs, cinémas, films)
    - Prisma choisi pour sa type safety et ses migrations automatisées
3. **Approche multi-plateforme** :
    - Utilisation de React Native pour l'application mobile afin de maximiser la réutilisation du code
    - Choix d'Electron pour l'application desktop pour tirer parti des compétences web de l'équipe
4. **Architecture API-first** :
    - Conception d'une API RESTful robuste pour servir toutes les plateformes
    - Facilite l'évolution future et l'ajout potentiel de nouvelles interfaces
5. **Sécurité et authentification** :
    - Implémentation de JWT pour une authentification sécurisée et sans état
    - Utilisation de bcrypt pour le hachage sécurisé des mots de passe
6. **Performance et scalabilité** :
    - Choix de technologies permettant une mise à l'échelle horizontale (Node.js, MongoDB)
    - Utilisation de Redis envisagée pour le caching (non implémenté dans la version actuelle)

9. Gestion de projet avec Notion et méthode Agile

Pour la gestion de ce projet, j'ai opté pour l'utilisation de Notion en combinaison avec la méthode Agile. Voici les raisons de ces choix :

1. **Notion comme outil de gestion** :
    - Interface intuitive et flexible
    - Capacité à créer des tableaux Kanban, des bases de données, et des pages de documentation
    - Facilité de collaboration en temps réel
    - Intégration possible avec d'autres outils (GitHub, Slack)
2. **Méthode Agile** :
    - Adaptabilité aux changements fréquents des exigences du projet
    - Livraisons incrémentales permettant des retours rapides
    - Amélioration continue du produit et du processus
    - Meilleure collaboration entre les membres de l'équipe
3. **Mise en œuvre** :
    - Sprints de deux semaines
    - Daily stand-ups virtuels
    - Revues de sprint et rétrospectives régulières
    - Backlog produit et backlog de sprint gérés dans Notion
4. **Avantages constatés** :
    - Meilleure visibilité sur l'avancement du projet
    - Communication améliorée au sein de l'équipe
    - Capacité à s'adapter rapidement aux feedbacks et aux changements de priorités
    - Documentation du projet centralisée et facilement accessible

10. Configuration de l'environnement de travail

L'environnement de développement a été soigneusement configuré pour optimiser la productivité :

1. **IDE** : Visual Studio Code
    - Extensions principales :
        - Prettier pour le formatage du code
        - ES7+ React/Redux/React-Native snippets
        - GitHub Copilot pour l'assistance IA
        - Simple React Snippets
        - Tree Exporter pour la visualisation de la structure du projet
2. **Navigateur** : Firefox Developer Edition
    - Choisi pour ses outils de développement puissants, particulièrement utiles pour le débogage React
3. **Outils de développement** :
    - Postman pour tester et documenter l'API
    - MySQL Workbench pour la gestion de la base de données MySQL
    - MongoDB Compass pour l'exploration et la manipulation des données MongoDB
    - Docker pour la conteneurisation et l'isolation des environnements
    - Git Bash et Windows PowerShell pour les opérations en ligne de commande
4. **Design** : Affinity Designer pour l'édition et la création d'assets graphiques
5. **Configuration matérielle** :

    - Utilisation de deux écrans externes en orientation verticale, en plus de l'écran du PC portable
    - Cette configuration permet une meilleure visualisation du code et une multitâche efficace

Cette configuration de l'environnement de travail a été choisie pour maximiser l'efficacité du développement, faciliter la collaboration, et assurer une qualité de code élevée tout au long du projet Cinéphoria.