Architecture

Explication du choix des technologies ainsi que du fonctionnement global.

Architecture logicielle de l'application Cinephoria

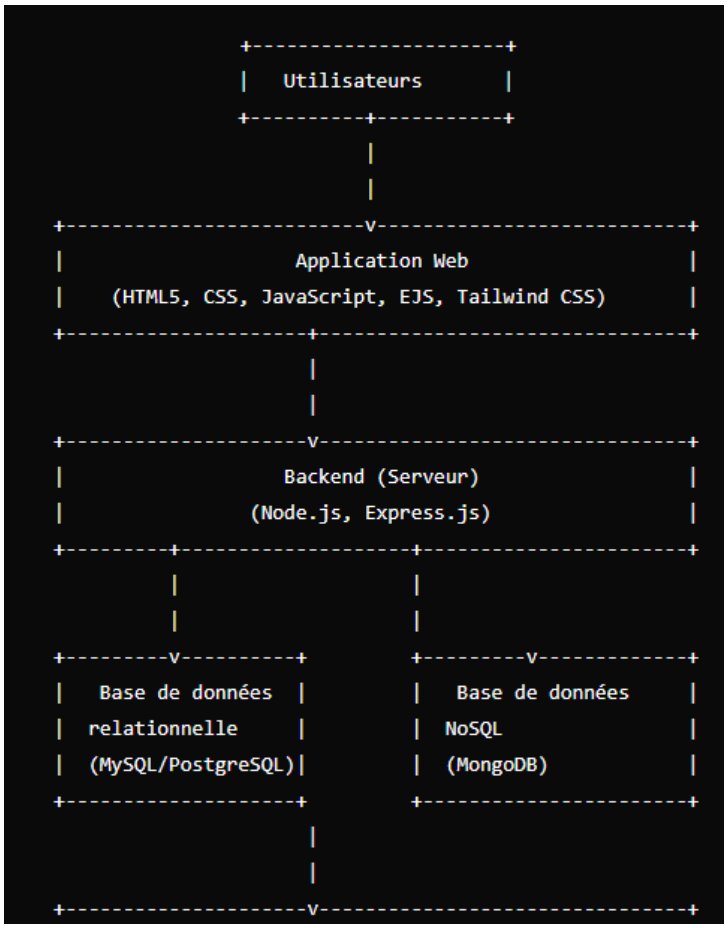
1. Introduction

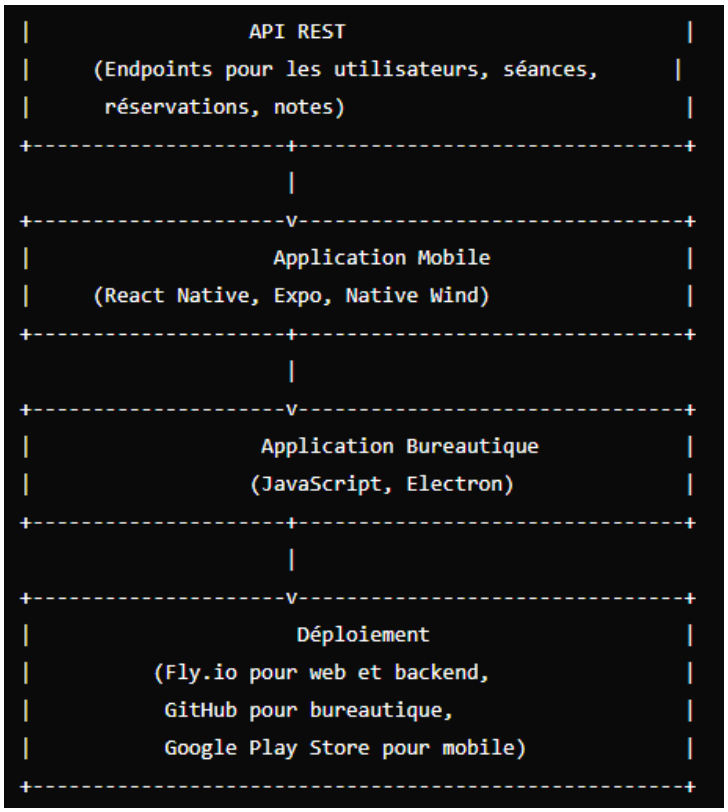
Objectif de l'application : Cinephoria est une application conçue pour améliorer l'expérience des clients des cinémas Cinephoria. L'application permet aux utilisateurs de consulter les séances disponibles, de commander des billets, de noter les séances, et de gérer leurs réservations. Elle inclut également une application bureautique pour les employés afin de saisir des incidents lors des séances.

Public cible : L'application s'adresse principalement aux clients des cinémas Cinephoria ainsi qu'aux employés de l'entreprise. Les clients utiliseront l'application web et mobile pour leurs besoins de réservation et de gestion de billets, tandis que les employés utiliseront l'application bureautique pour la gestion des incidents.

2. Vue d'ensemble de l'architecture

Diagramme d'architecture :





Explications du diagramme:

1. **Utilisateurs** : Point de départ des interactions avec l'application, que ce soit via le web (visiteurs, administrateur, employés), mobile (visiteurs), ou bureautique (employés).

2. **Application Web** : Développée avec EJS, JavaScript, et Tailwind CSS. Fournit l'interface utilisateur pour les clients.
3. **Backend (Serveur)** : Développé avec Node.js et Express.js. Gère les requêtes HTTP, l'authentification, et les interactions avec les bases de données.
4. **Base de données relationnelle (MySQL/PostgreSQL)** : Stocke les données transactionnelles et relationnelles critiques.
5. **Base de données NoSQL (MongoDB)** : Stocke les données flexibles et non structurées, comme les statistiques de réservations hebdomadaire des utilisateurs.
6. **API REST** : Fournit des endpoints pour les opérations sur les utilisateurs, les séances, les réservations, et les notes.
7. **Application Mobile** : Développée avec React Native, Expo, et Native Wind. Permet aux utilisateurs d'accéder à leur réservations via leur smartphone.
8. **Application Bureautique** : Développée avec JavaScript et Electron. Permet aux employés de gérer les incidents dans les salles du cinéma.
9. **Déploiement** : Utilisation de Fly.io pour l'hébergement web et backend, GitHub pour la gestion et le déploiement de l'application bureautique, et Google Play Store pour la distribution de l'application mobile (disponible également sur Github.

3. Composants principaux

Backend (Serveur)

- **Technologies utilisées :** Node.js, Express.js.
- **Structure des API :**Les principales routes incluent des endpoints pour gérer les utilisateurs, les séances, les réservations, les salles, les avis... Les méthodes HTTP utilisées sont principalement GET, POST, PUT, DELETE.
  - **Exemples :**
    - **GET /cinemas :** Récupère la liste des cinémas.
    - **POST /reservations :** Crée une nouvelle réservation.
    - **PUT /seances/**
    - **:** Met à jour les informations d'une séance.
    - **DELETE /seances/**
    - **:** Supprime une séance.
- **Gestion des données :** MySQL/PostgreSQL sont utilisés pour les transactions relationnelles et critiques, tandis que MongoDB stocke les données non structurées.

Frontend (Client)

- **Technologies utilisées :** CSS (Tailwind CSS), JavaScript, EJS pour le web; React Native et Expo pour le mobile.
- **Structure des composants :** Les composants UI sont structurés de manière modulaire, facilitant la gestion de l'état global et des données côté client.
- **Interaction avec le backend :** Les applications front-end communiquent avec le backend via des appels API REST, sécurisés et optimisés pour la performance.

4. Sécurité

- **Authentification et Autorisation :** Utilisation de JSON Web Tokens (JWT) pour l'authentification sécurisée. Gestion des autorisations pour restreindre l'accès aux ressources.
- **Protection des données :**Chiffrement des mots de passe avec Bcrypt, échappement des entrées pour éviter les injections SQL et les failles XSS.
  - **Validation et échappement des entrées :** Validation côté client et serveur pour s'assurer que les données sont conformes aux attentes. Échappement des données pour prévenir les injections SQL.
  - **Requêtes préparées :** Utilisation de requêtes préparées pour sécuriser les interactions avec la base de données.
  - **Contrôles d'accès et gestion des sessions :** Mise en place de contrôles stricts sur les sessions utilisateurs et de gestion des autorisations (Sécurité).

5. Gestion des dépendances

- **Modules et Bibliothèques :** Utilisation d'Express.js, Tailwind CSS, React Native, et d'autres bibliothèques pertinentes pour le développement.
- **Gestion des versions :** Utilisation de npm pour gérer les versions des dépendances et les mises à jour, en veillant à la compatibilité et à la stabilité du projet.

6. Déploiement et Infrastructure

- **Environnements de déploiement :** Développement, test, et production. Chaque environnement dispose de configurations spécifiques pour assurer une transition en douceur.
- **Technologies de déploiement :** Utilisation de Fly.io pour l'hébergement, GitHub pour la gestion du code et des versions, Expo pour le déploiement mobile.

7. Tests

- **Types de tests :** Tests unitaires et tests d'intégration.
- **Outils de test :** Utilisation de Jest pour les tests unitaires et les tests d'intégration.
  - **Tests d'API :** Vérification du bon fonctionnement des routes de l'API, gestion des erreurs et interactions avec la base de données.
  - **Configuration des tests :** Initialisation des données avant les tests, nettoyage après les tests, et réinitialisation après chaque test pour maintenir un état cohérent de la base de données.
  - **Objectifs des tests :** Assurer le bon fonctionnement des fonctionnalités de l'API, protéger les données, et garantir la qualité globale de l'application.

Besoin : Assurer la qualité et la fiabilité de l'application à travers des tests complets.

- **Jest**
  - **Besoin :** Framework de test complet et rapide pour les tests unitaires et d'intégration.
  - **Solution :** Utilisation de Jest pour les tests unitaires et d'intégration.
  - **Avantages :**
    - Framework de test complet et rapide.
    - Facilité d'intégration avec les projets JavaScript.

        - Supporte les tests de snapshot pour les composants React.
    - **Inconvénient :**
        - Peut être complexe à configurer pour les tests d'intégration avancés.

8. Conclusion

Résumé : L'architecture de Cinephoria repose sur des technologies modernes et robustes, assurant performance, scalabilité, et sécurité. L'utilisation de Node.js, React Native, et des bases de données adaptées permet de répondre efficacement aux besoins des utilisateurs.

Ainsi, l'architecture de Cinephoria garantit une expérience utilisateur enrichie et sécurisée.