# **📘 Documentation Technique** **Cinéphoria**

**1. Architecture logicielle**

Le projet **Cinéphoria** repose sur une architecture en couches :

* **Front-end** :
  + Application mobile développée avec **Flutter**.
  + Application web en **HTML/CSS/JavaScript**.
* **Back-end** :
  + Développé avec **Flask** (sans ORM comme SQLAlchemy pour un meilleur contrôle des requêtes SQL).
* **Base de données** :
  + **MySQL** pour les données relationnelles (films, utilisateurs, réservations).
  + **MongoDB Atlas** pour les données non structurées (statistiques, logs).
* **API REST** : utilisée pour la communication entre les interfaces front-end et le back-end.
* **Modularité** : chaque module (authentification, gestion des films, réservations, statistiques) est indépendant, facilitant la maintenance et l’évolutivité.

## **2. Choix technologiques**

Pour Cinéphoria, j’ai choisi des technologies simples, efficaces et bien connues, qui me permettent de développer rapidement tout en gardant un bon niveau de sécurité et de performance.

Pour le **front-end web**, j’utilise **HTML, CSS, JavaScript et PHP**. Ce sont des langages classiques, faciles à prendre en main et à déployer sur un serveur Apache. Ils me permettent de créer une interface claire et responsive.

Pour le **front-end mobile**, j’ai opté pour **Flutter**, qui permet de créer une seule application compatible Android et iOS. C’est rapide à développer et le rendu est très fluide.

Le **back-end** est en **PHP**, avec une **API REST**. PHP est bien intégré à MySQL et facile à héberger. Il a une grande communauté, ce qui aide en cas de problème.

Côté base de données, j’utilise **MySQL** pour tout ce qui est structuré (films, utilisateurs, réservations), et **MongoDB Atlas** pour les statistiques et les logs, car c’est plus flexible pour ce type de données.

Pour l’**authentification**, j’utilise les **sessions PHP** sur le site web et des **JWT** sur l’application mobile, ce qui permet de sécuriser les accès selon les rôles (utilisateur, employé, admin).

Le **déploiement** se fait en local avec **XAMPP**, et en ligne sur un **serveur Apache**, ce qui me permet de tester facilement avant de mettre en production.

Pour les **tests**, j’utilise **PHPUnit** pour le back-end et **Flutter Test** pour le mobile, afin de vérifier que les fonctionnalités principales marchent bien.

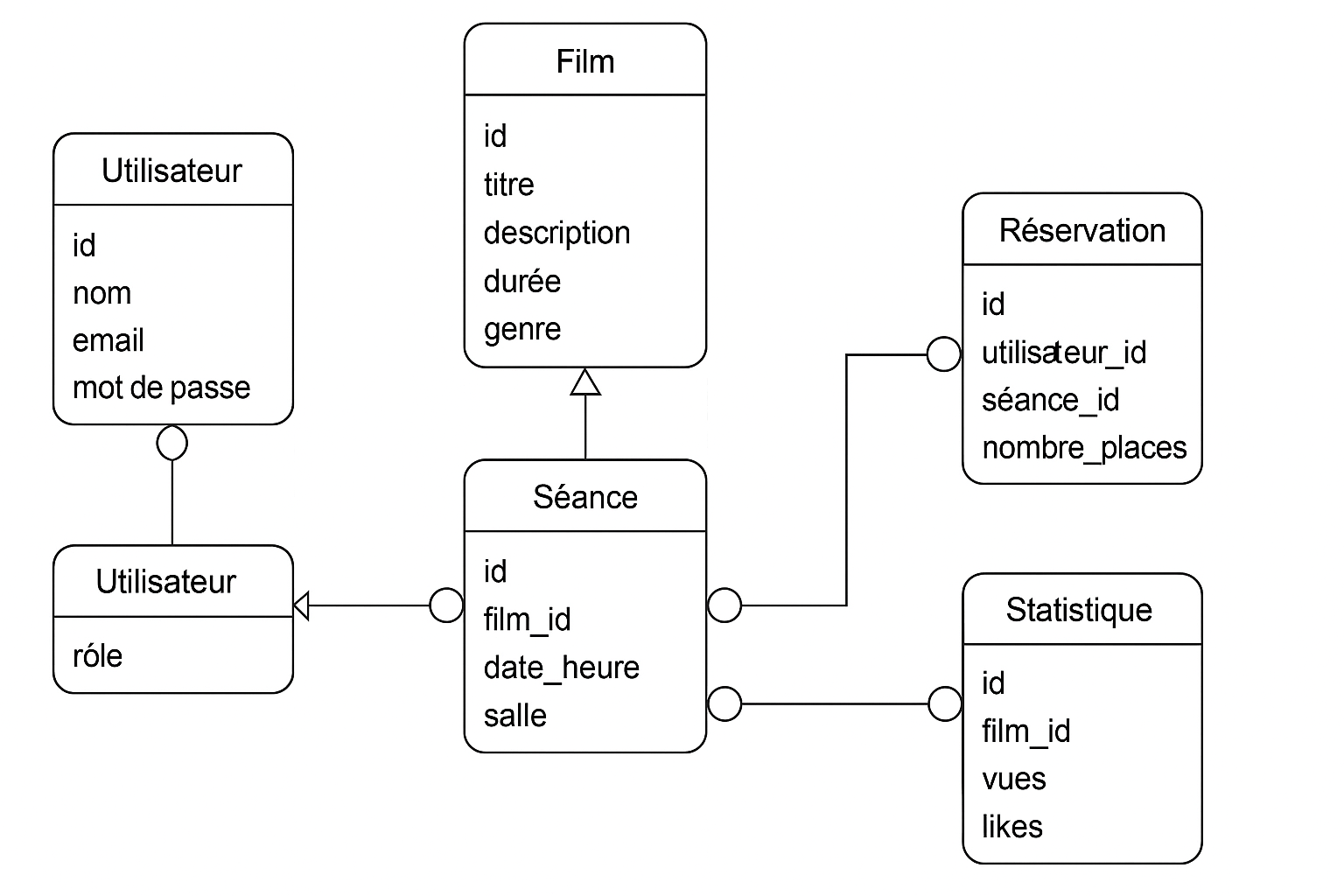
Enfin, j’utilise **GitHub** pour gérer le code, et je prévois d’ajouter **GitHub Actions** pour automatiser les tests et les déploiements plus tard.

## **3. MCD (Modèle Conceptuel de Données)**

Le MCD comprend les entités suivantes :

* **Utilisateur** (id, nom, email, mot de passe, rôle)
* **Film** (id, titre, description, durée, genre)
* **Séance** (id, film\_id, date\_heure, salle)
* **Réservation** (id, utilisateur\_id, séance\_id, nombre\_places)
* **Statistique** (id, film\_id, vues, likes)

Les relations sont conçues pour garantir l’intégrité des données et optimiser les requêtes.



## **4. Diagrammes UML**

* **Diagrammes de cas d’utilisation** :
  + Visiteur : consulter les films
  + Utilisateur : réserver, noter un film
  + Employé : valider les billets
  + Admin : gérer les films, utilisateurs, statistiques
* **Diagrammes de séquence** :
  + Réservation d’un billet
  + Génération et scan d’un QR code
  + Ajout d’un film par un administrateur

*(Je peux te générer ces diagrammes en image si tu veux les insérer dans ton document.)*

## **5. Plan de test**

* **Tests unitaires** :
  + Fonctions critiques : génération de QR code, validation de réservation, authentification.
* **Tests fonctionnels** :
  + Parcours utilisateur complet : inscription, connexion, réservation, annulation.
* **Environnement de test** :
  + Base de données dédiée
  + Utilisation de mocks pour les appels API

## **6. Déploiement**

* **Web** : hébergement local via XAMPP ou sur un serveur distant.
* **Mobile** : génération d’un APK via Flutter, possibilité de publication sur le Play Store.
* **Base de données** :
  + **MySQL** : hébergé localement ou sur un serveur distant.
  + **MongoDB Atlas** : hébergement cloud sécurisé.

## **7. CI/CD**

* **Intégration continue** :
  + Utilisation possible de **GitHub Actions** ou **GitLab CI** pour automatiser les tests et la génération d’APK.
* **Déploiement continu** :
  + Mise à jour automatique de l’application après validation des tests.
* **Avantages** :
  + Réduction des erreurs humaines
  + Déploiement plus rapide et plus fiable

## **8. Sécurité**

* + Pour sécuriser Cinéphoria, j’ai mis en place plusieurs protections à différents niveaux.
  + Côté **front-end**, je fais attention à bien valider les données saisies par les utilisateurs pour éviter les failles comme les attaques XSS. Ça permet d’éviter que du code malveillant soit injecté dans les pages.
  + Côté **back-end**, j’utilise des **JWT** pour sécuriser les échanges avec l’application mobile, et je vérifie les rôles des utilisateurs pour limiter l’accès aux fonctionnalités sensibles. Les requêtes SQL sont protégées contre les injections grâce à des requêtes préparées.
  + Dans la **base de données**, les mots de passe sont **hashés** (avec bcrypt) pour éviter qu’ils soient lisibles en cas de fuite. L’accès aux données sensibles est limité, et des **sauvegardes régulières** sont prévues pour éviter toute perte.