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, permettant un développement rapide tout en assurant sécurité et performance.

- Front-end web :

HTML, CSS, JavaScript et PHP. Ces langages sont classiques, faciles à prendre en main et à déployer sur un serveur Apache. Ils permettent de créer une interface claire et responsive.

- Front-end mobile :

Flutter, qui permet de créer une seule application compatible Android et iOS, avec un développement rapide et un rendu fluide.

- Back-end :

PHP avec une API REST. PHP est bien intégré à MySQL, facile à héberger, et dispose d'une grande communauté.

- Base de données :

MySQL pour les données structurées (films, utilisateurs, réservations) et MongoDB Atlas pour les statistiques et logs, plus flexible pour ce type de données.

- Authentification :

Sessions PHP sur le site web et JWT sur l'application mobile, pour sécuriser les accès selon les rôles (utilisateur, employé, admin).

- Déploiement :

En local avec XAMPP, et en ligne sur un serveur Apache, pour faciliter les tests avant la mise en production.

- Tests :

PHPUnit pour le back-end et Flutter Test pour le mobile, afin de vérifier le bon fonctionnement des principales fonctionnalités.

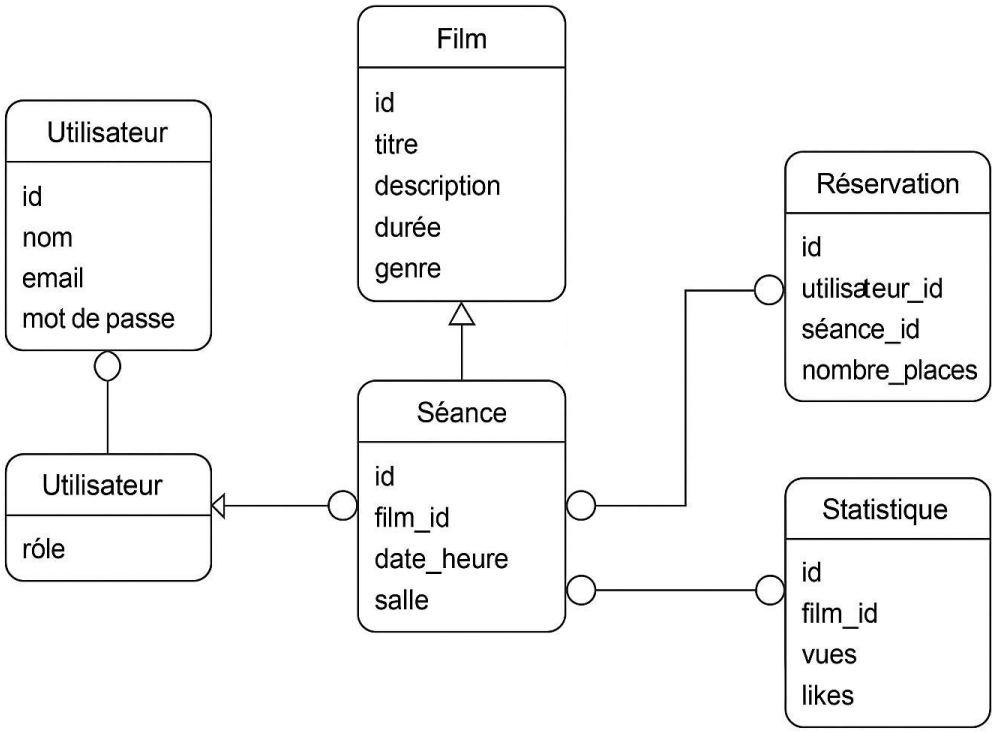
- Gestion du code :

GitHub pour le versioning, avec l'ajout prévu de GitHub Actions pour automatiser les tests et les déploiements.

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



4. Diagrammes UML

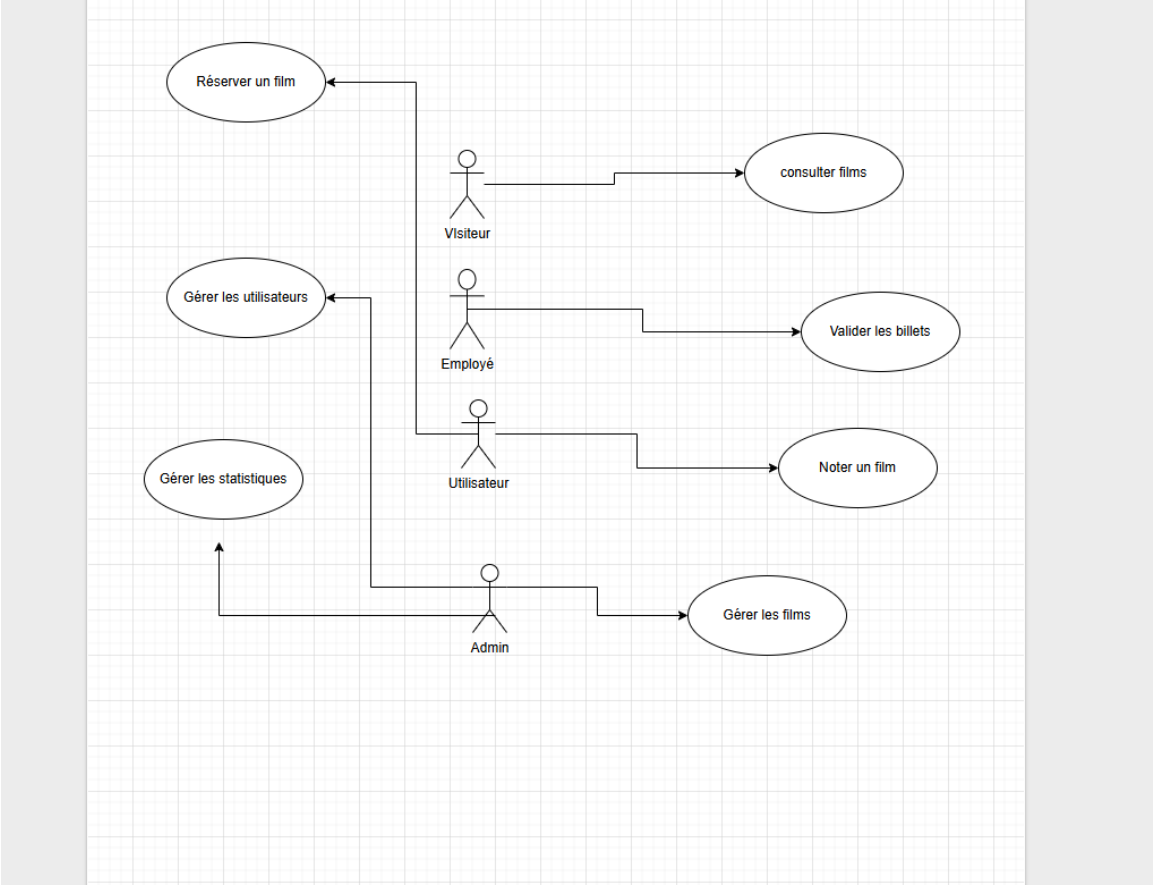
- Diagrammes de cas d'utilisation :

- Visiteur : consulter les films

- Utilisateur : réserver, noter un film

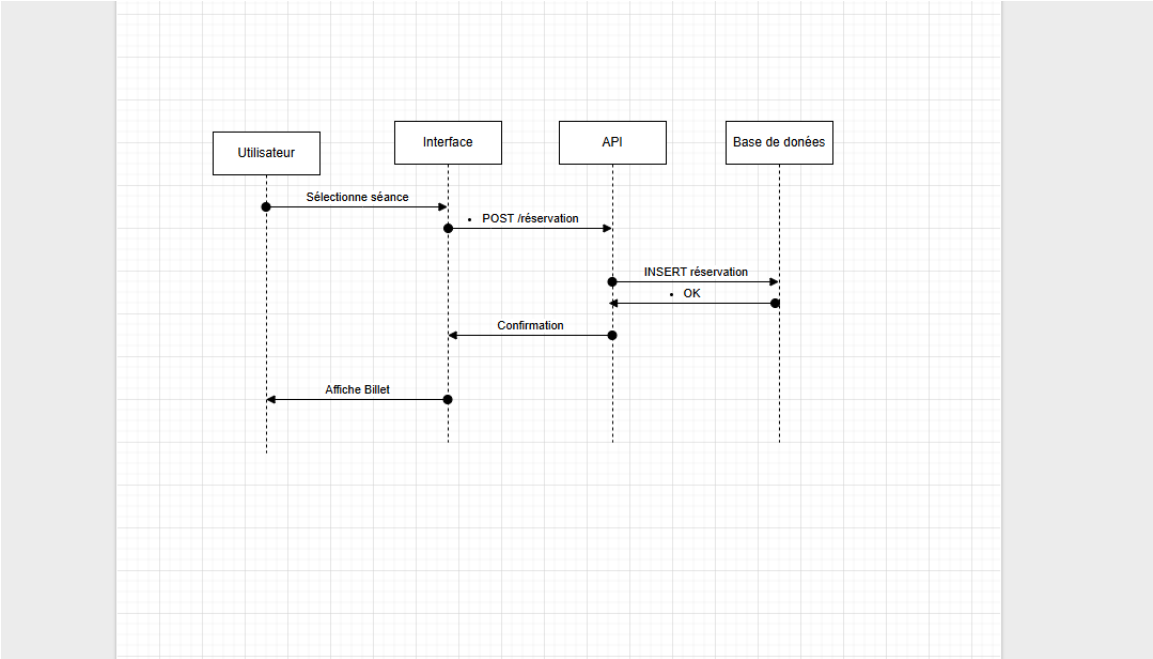
- Employé : valider les billets

- Admin : gérer les films, utilisateurs, statistiques

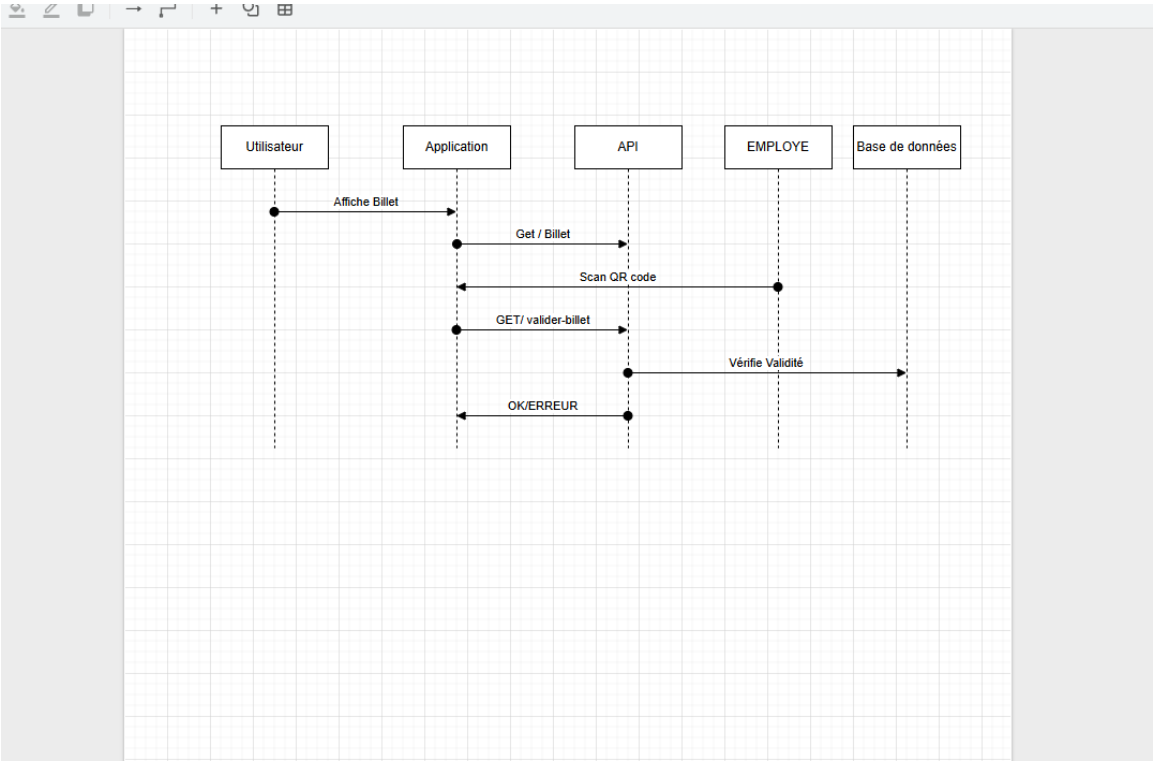


- Diagrammes de séquence :

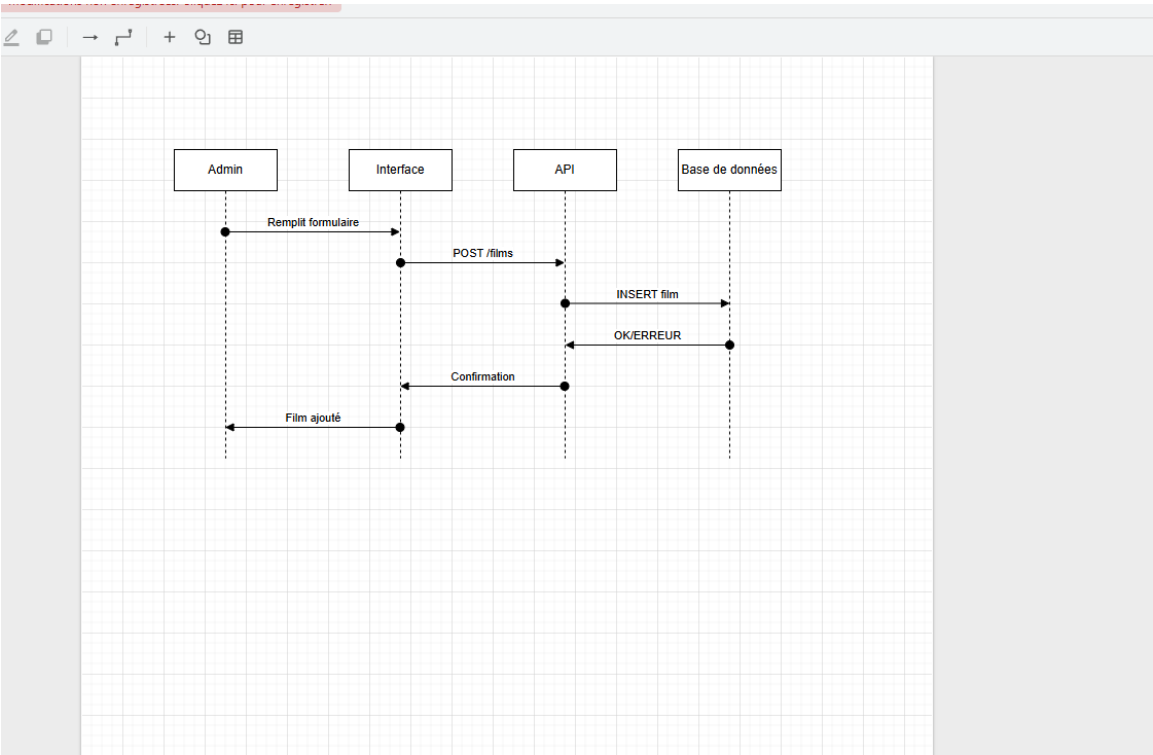
- Réservation d’un billet



- Génération et scan d’un QR code



- Ajout d’un film par un administrateur



5. Plan de test

- Tests unitaires :

- test vérifie si une date est valide au format AAAA-MM-JJ. Il utilise des assertions pour confirmer le bon fonctionnement de la fonction `is_valid_date`.

- Tests fonctionnels :

- test fonctionnel vérifie que l’on peut créer un incident, le récupérer, puis le supprimer correctement via l’API.

- 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, plusieurs protections sont mises en place à différents niveaux :

- Front-end : validation des données saisies pour éviter les failles XSS (injection de code malveillant).

- Back-end : utilisation de JWT pour sécuriser les échanges avec l'application mobile, vérification des rôles pour limiter l'accès aux fonctionnalités sensibles, requêtes SQL préparées pour éviter les injections.

- Base de données : mots de passe hashés avec bcrypt, accès limité aux données sensibles, sauvegardes régulières pour éviter toute perte.

9. Transaction SQL

Pour garantir la cohérence des données lors de la création d'une réservation et l'attribution des sièges, Cinéphoria utilise des transactions SQL. Cela permet de s'assurer que toutes les opérations liées à une réservation sont réalisées ensemble : si l'une échoue, aucune modification n'est appliquée à la base.

Exemple de transaction pour la création d'une réservation et la réservation de sièges :

```
START TRANSACTION;

-- Création d'une réservation pour l'utilisateur 1 à la séance 1
INSERT INTO reservations (utilisateur_id, seance_id, nombre_personnes,
prix_total)
VALUES (1, 1, 2, 20.00);

-- Réservation des sièges 5 et 6
INSERT INTO places_reservees (reservation_id, numero_place, mobilite_reduite)
VALUES (LAST_INSERT_ID(), 5, 0),
       (LAST_INSERT_ID(), 6, 0);

COMMIT;
```

Explications :

- START TRANSACTION; démarre la transaction.

- On insère la réservation dans la table reservations.

- On réserve ensuite les sièges associés à cette réservation dans la table places_reservees, en utilisant LAST_INSERT_ID() pour récupérer l'identifiant de la réservation créée.

- COMMIT; valide l'ensemble des opérations. Si une étape échoue, un ROLLBACK; peut être utilisé pour annuler toutes les modifications.

Cette approche garantit l’intégrité des données et évite les incohérences (par exemple, des sièges réservés sans réservation associée).