Cinéma Le Cinéphoria

Document Technique Projet Cinéphoria

Studi

Joël DERMONT
27/01/2025

Sommaire

1 Introduction 4
1.1 Présentation générale 4
2 Fonctionnement global 4
2.1 Vue d'ensemble 4
2.2 Fonctionnement par type d'utilisateur 4
2.2.1 Visiteurs (non authentifiés) 4
2.2.2 Utilisateurs (authentifiés) 5
2.3 Employés 5
2.4 Administrateurs 5
3 Choix Technologiques 6
3.1 Application Web (Symfony) 6
3.2 Application Mobile (Flutter) 6
4 Réflexions initiales technologiques sur le sujet 7
5 Configuration de l'environnement de travail pour l'application Web 7
5.1 Installation de XAMPP 7
5.2 Installation de PhpStorm 7
5.3 Installation des outils nécessaires 7
5.3.1 Installer Composer 7
5.3.2 Installer Symfony CLI 8
5.3.3 Installer Node.js et npm 8
5.3.4 Installer Git 8
5.3.5 Création et configuration d'un projet Symfony 8
5.3.6 Configuration de Git 9
5.3.7 Lancement du Projet Symfony 10
6 Configuration d'un Environnement de Travail pour l'application mobile 10
6.1 Installation de Android Studio 10
6.2 Installation des outils nécessaires 11
6.2.1 Installer Flutter 11
6.2.2 Installer les SDKs et outils Android 11
6.2.3 Installer les extensions Flutter et Dart 11

6.2.4 Configuration du projet Flutter 12
6.2.5 Tester l'application sur un appareil physique 12
6.2.6 Configurer le build Android 12
7 Modèle conceptuel de données (MCD) 13
8 Diagramme d'utilisation 13
9 Explication de la transaction SQL 14
9.1 Démarrage de la transaction 14
9.2 Insertion des cinémas 14
9.3 Insertion des genres de films 14
9.4 Insertion des salles de cinéma 15
9.5 Insertion des films 15
9.6 Insertion des séances dans les salles 16
9.7 Association des séances avec les cinémas 17
9.8 Validation de la transaction 18
10 Explication du plan de test et du déploiement 18
10.1 Plan de test 18
10.1.1 Objectifs des tests 18
10.1.2 Types de tests 18
11 Déploiement application Web 22
11.1 Préparation du VPS OVH 22
11.1.1 Commander et configurer le VPS chez OVH 22
11.1.2 Installer les prérequis 22
11.1.3 Déploiement de l'application avec Git 22
12 Déploiement de l'application Mobile (Android) 26
12.1 Configuration des URLs de l'API 26
12.2 Génération des builds avec Android Studio 26
12.2.1 Ouvrir le projet Flutter dans Android Studio 26
12.2.2 Configurer le fichier build.gradle 26
12.2.3 Générer le build en mode release 27
12.3 Signature et Publication sur le Google Play Store 27
12.3.1 Générer une clé de signature 27

12.3.2 Ajouter la signature à gradle.properties 27
12.3.3 Modifier android/app/build.gradle pour utiliser la signature 28
12.3.4 Vérifier et publier sur Google Play 28
13 Explication de la démarche afin de proposer un déploiement continu 29
13.1 Application web 29
13.1.1 Travailler en local 29
13.1.2 Commit & Push vers Git 29
13.1.3 Se connecter au serveur en production 29
13.1.4 Aller dans le dossier du projet et pull les modifications 29
13.1.5 Mettre à jour les dépendances 29
13.1.6 Mettre à jour la base de données 30
13.1.7 Vider le cache et redémarrer les services 30
13.2 Application mobile 30
13.2.1 Développer et tester en local 30
13.2.2 Commit & Push vers Git 30
13.2.3 Générer un build APK pour Android 30
13.2.4 Générer un AAB pour publier sur le Play Store 31
13.2.5 Signer l'AAB pour Google Play 31
13.2.6 Publier la mise à jour sur le Play Store 32

1 Introduction

1.1 Présentation générale

Cinéphoria est une plateforme complète permettant de gérer et d'améliorer l'expérience des spectateurs dans des cinémas répartis en France et en Belgique. Le projet se divise en deux applications :

- Application Web : Gestion des films, séances, réservations, et compte utilisateur.
- Application Mobile : Consultation des séances et des billets via QR code.

2 Fonctionnement global

2.1 Vue d'ensemble

Cinéphoria est une solution complète conçue pour répondre aux besoins des visiteurs, employés, et administrateurs des cinémas. Elle repose sur deux applications distinctes mais interconnectées :

- Une application web Symfony pour la gestion des films, séances, réservations, et comptes utilisateurs.
- Une application mobile Flutter permettant aux utilisateurs finaux d’accéder à leurs billets et d’interagir avec leurs commandes.
- Les deux applications sont unifiées par un backend centralisé basé sur Symfony et communiquent via une API REST sécurisée.

2.2 Fonctionnement par type d’utilisateur

2.2.1 Visiteurs (non authentifiés)

Accès Web :

- Les visiteurs peuvent consulter la liste des films disponibles et explorer leurs horaires, leurs descriptions et les séances proposées.
- Ils peuvent préparer une réservation en sélectionnant un film, une séance et un siège. Cependant, la confirmation de la réservation nécessite de créer un compte et de s’authentifier.

- Les informations pratiques (adresse, horaires, contact) sont accessibles sur toutes les pages grâce au pied de page.

2.2.2 Utilisateurs (authentifiés)

Accès Web :

Une fois connectés, les utilisateurs peuvent :

- Finaliser leurs réservations et voir leurs commandes passées ou en attente.
- Noter les films qu'ils ont visionnés en laissant un avis après la séance.

Accès Mobile :

Les utilisateurs peuvent :

- Accéder à leurs billets sous forme de QR codes, scannables lors de leur arrivée en salle.
- Consulter les séances auxquelles ils sont inscrits pour le jour courant et les jours à venir.

Interactions avec l'API :

- Les données des utilisateurs sont synchronisées entre les applications web et mobile via l'API Symfony, garantissant une expérience cohérente.

2.3 Employés

Accès Web :

Les employés peuvent :

- Gérer les avis laissés par les utilisateurs (validation ou suppression).
- Ajouter, modifier ou supprimer des films et séances selon les besoins.

2.4 Administrateurs

Accès Web :

Les administrateurs disposent d'un espace dédié pour :

- Créer et gérer les comptes des employés.

- Gérer les films, séances et salles avec des fonctionnalités avancées.
- Accéder à un tableau de bord basé sur une base NoSQL (MongoDB) pour suivre les réservations et analyser les performances sur une période donnée.
- Réinitialiser les mots de passe des employés si nécessaire.

3 Choix Technologiques

3.1 Application Web (Symfony)

- Framework Symfony : Fournit une structure robuste et modulaire pour développer rapidement des applications scalables.
- Langage PHP : Choisi pour sa compatibilité avec Symfony et son écosystème mature.
- Base de données relationnelle (MySQL) : Optimisée pour la gestion des relations complexes (films, séances, utilisateurs).
- Base de données NoSQL (MongoDB) : Pour stocker les statistiques et tableaux de bord.
- Frontend : HTML5, CSS (Bootstrap), et JavaScript (jQuery/Vanilla JS).
- Sécurité : Symfony Security avec formulaire de connexion sécurisé (form_login) , Protection CSRF, token JSON Web (JWT) pour l'API de l'appication mobile, contrôle des routes (access_control).

3.2 Application Mobile (Flutter)

- Framework Flutter : Permet un développement multiplateforme (iOS et Android) avec une seule base de code.
- Langage Dart : Efficace et performant pour des interfaces utilisateur fluides.
- API RESTful : Communication avec le back-end Symfony pour la gestion des données.

4 Réflexions initiales technologiques sur le sujet

Les choix technologiques retenus sont basés sur les besoins fonctionnels du projet, les contraintes techniques identifiées et mon expérience avec les outils sélectionnés. Symfony pour le backend et Flutter pour le mobile permettent d'assurer une forte cohérence dans le développement et un déploiement efficace, tout en offrant une architecture flexible pour répondre à l'évolution future du projet.

5 Configuration de l'environnement de travail pour l'application Web

5.1 Installation de XAMPP

- Téléchargez et installez XAMPP depuis le site officiel. Assurez-vous d'inclure Apache, MySQL, et PHP (version 8.x ou supérieure).
- Installez XAMPP dans un répertoire simple (par exemple, C:\xampp) et démarrez les services Apache et MySQL via le panneau de contrôle.
- Activez le module mod_rewrite dans les paramètres d'Apache pour gérer les routes Symfony.

5.2 Installation de PhpStorm

- Téléchargez et installez PhpStorm depuis le site officiel.
- Configurez PhpStorm pour utiliser le fichier PHP de XAMPP (C:\xampp\php\php.exe) comme interpréteur PHP.
- Installez le plugin Symfony Support pour une meilleure intégration avec Symfony (optionnel mais recommandé).

5.3 Installation des outils nécessaires

5.3.1 Installer Composer

- Téléchargez Composer depuis https://getcomposer.org et suivez les instructions.

- Vérifiez l'installation avec composer -v.

5.3.2 Installer Symfony CLI

- Téléchargez Symfony CLI depuis https://symfony.com/download.
- Vérifiez l'installation avec symfony -v.

5.3.3 Installer Node.js et npm

- Téléchargez Node.js depuis https://nodejs.org (version LTS recommandée).
- Node.js inclut npm, le gestionnaire de dépendances front-end.

  Vérifiez les installations avec :

  node -v (pour Node.js)

  npm -v (pour npm)

5.3.4 Installer Git

- Téléchargez Git depuis https://git-scm.com et installez-le.
- Pendant l'installation, configurez Git pour s'intégrer avec votre terminal préféré (par exemple, Git Bash).
- Vérifiez l'installation avec git --version.

5.3.5 Création et configuration d'un projet Symfony

5.3.5.1 Créer le projet Symfony

- Dans le répertoire de XAMPP (C:\xampp\htdocs), créez un projet Symfony avec Symfony CLI :

  symfony new nom_projet –webapp

5.3.5.2 Configurer MySQL

- Accédez à PHPMyAdmin (http://localhost/phpmyadmin) et créez une base de données.
- Configurez la connexion dans le fichier .env de votre projet Symfony.

5.3.5.3 Installer Webpack Encore (pour le front-end)

- Installez Webpack Encore et les dépendances nécessaires :

  composer require symfony/webpack-encore-bundle

  npm install

  npm install --save-dev @symfony/webpack-encore

- Compilez les assets avec :

  npm run dev

5.3.6 Configuration de Git

5.3.6.1 Initialiser un dépôt Git

- Dans le répertoire de votre projet Symfony, initialisez un dépôt Git :

  git init

5.3.6.2 Créer un fichier .gitignore

- Ajoutez un fichier .gitignore pour exclure les fichiers inutiles (par exemple, node_modules, vendor, etc.). Exemple de contenu :

  /node_modules/

  /vendor/

  .env

5.3.6.3 Faire le premier commit

- Ajoutez les fichiers et effectuez votre premier commit :

  git add .

  git commit -m "Initial commit"

5.3.6.4 Configurer un dépôt distant

- Si vous utilisez GitHub, créez un dépôt en ligne et liez-le à votre projet :

  git remote add origin https://github.com/votre-utilisateur/votre-repo.git

  git branch -M main

  git push -u origin main

5.3.7 Lancement du Projet Symfony

- Lancez le serveur Symfony

  symfony server:start

- Accédez à l'application via votre navigateur (par défaut : http://127.0.0.1:8000).

6 Configuration d'un Environnement de Travail pour l'application mobile

6.1 Installation de Android Studio

- Télécharge Android Studio depuis : https://developer.android.com/studio
- Installe Android Studio avec les options par défaut.
- Ouvre **Android Studio** et installe :
  - Le **SDK Android** (dernière version stable).

- Les outils de compilation (NDK, build-tools, etc.).

6.2 Installation des outils nécessaires

6.2.1 Installer Flutter

- Télécharge Flutter depuis le site officiel : https://flutter.dev/docs/get-started/install
- Décompresse le dossier Flutter dans un répertoire sans espace, par exemple :
  - C:\flutter
- Ajoute Flutter au PATH de Windows :
  - Dans Variables d'environnement, ajoute C:\flutter\bin au PATH.
- Vérifie l'installation avec :

  flutter doctor
- Ce test indique les dépendances manquantes.

6.2.2 Installer les SDKs et outils Android

- Ouvre Android Studio → Paramètres → Appearance & Behavior → System Settings → Android SDK.
- Vérifie que Android SDK et Android SDK Command-line Tools sont installés.
- Installe la dernière version du SDK avec Android SDK Platform.
- Vérifie que adb est accessible avec :

  adb –version

6.2.3 Installer les extensions Flutter et Dart

- Ouvre Android Studio.

- Va dans Plugins et installe :
    - Flutter Plugin
    - Dart Plugin
- Redémarre Android Studio.

6.2.4 Configuration du projet Flutter

- Via Android Studio :
    - Ouvre Android Studio → New Flutter Project.
    - Sélectionne Flutter Application.
    - Définis le chemin de Flutter (C:\flutter).
    - Configure le projet et clique sur Finish.

6.2.5 Tester l'application sur un appareil physique

- Active le mode développeur sur ton téléphone.
- Active le débogage USB (dans les options développeur).
- Branche l'appareil et vérifie qu'il est détecté :

    flutter devices
- Lance l'application

    flutter run

6.2.6 Configurer le build Android

- Ouvre android/app/build.gradle et mets à jour :

    defaultConfig {

    minSdkVersion 21

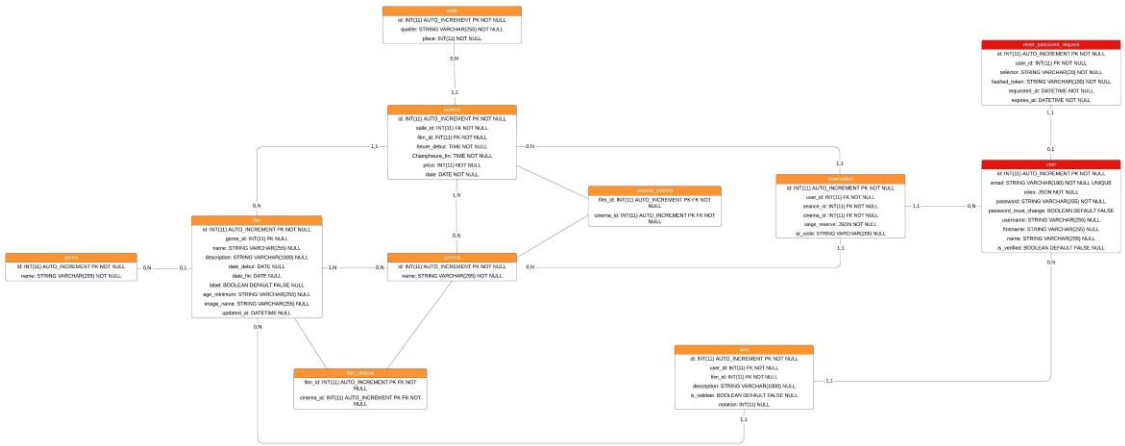
    targetSdkVersion 33

}

- distributionUrl=https\://services.gradle.org/distributions/gradle-8.3-all.zip
- Installe les dépendances :

  flutter pub get

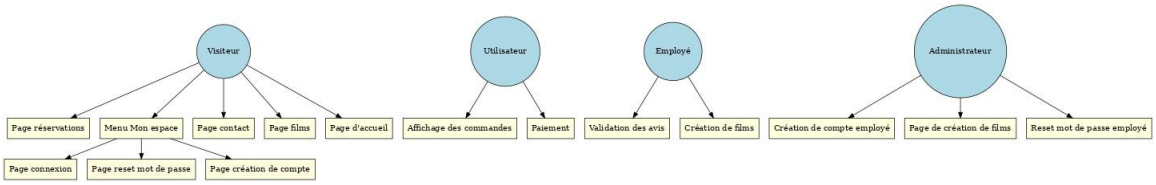
7 Modèle conceptuel de données (MCD)



Lien Google Drive:

https://drive.google.com/file/d/1quv7RQLs7J3P8z7IqO2XC72o0Itku7Qd/view?usp=sharing

8 Diagramme d'utilisation



Lien Google Drive:

https://drive.google.com/file/d/1Li2mAsExDOuJOmyGYnjDHW1h72j8XYzl/view?usp=sharing

9 Explication de la transaction SQL

9.1 Démarrage de la transaction

Cette instruction explique une transaction. Une transaction permet d'exécuter plusieurs opérations sur la base de données de manière atomique : soit toutes les modifications sont appliquées avec succès, soit aucune modification n'est enregistrée si une erreur se produit. Si une erreur survient après cette commande, il est possible de rétablir la base de données dans son état initial avec la commande ROLLBACK sans affecter les modifications précédentes.

La transaction ajoute tous les éléments nécessaires pour visualier les films des cinémas avec les séances

Code SQL :

```
START TRANSACTION;
```

9.2 Insertion des cinémas

Cette requête insère plusieurs cinémas dans la table cinema. Chaque ligne représente un cinéma avec un id (clé primaire) et un nom. Par exemple, le premier cinéma a l'ID 1 et le nom 'Toulouse'.

Code SQL :

```
INSERT INTO cinema (id, name) VALUES
(1, 'Toulouse'),
(2, 'Nantes'),
(3, 'Bordeaux'),
(4, 'Lille'),
(5, 'Charleroi'),
(6, 'Liège'),
(7, 'Paris');
```

9.3 Insertion des genres de films

Cette requête insère plusieurs genres de films dans la table genre. Chaque genre a un id et un nom. Exemple : Le genre 'Action' est assigné à l'ID 1.

Code SQL :

```
INSERT INTO genre (id, name) VALUES
(1, 'Action'),
(2, 'Comédie'),
(3, 'Horreur'),
(4, 'Science-fiction'),
(5, 'Romance'),
(6, 'Thriller'),
(7, 'Drame'),
(8, 'Animation');
```

9.4 Insertion des salles de cinéma

Les salles de cinéma sont insérées dans la table salle. Chaque salle est associée à un id, une qualité (comme 3DX, 4DX, etc.) et un nombre de places. Exemple : La salle avec l'ID 1 est une salle 3DX avec 100 places.

Code SQL :

```
INSERT INTO salle (id, qualite, places) VALUES
(1, '3DX', 100),
(2, '4DX', 100),
(3, 'IMAX', 100),
(4, 'Dolby', 100);
```

9.5 Insertion des films

Les films sont insérés dans la table film. Chaque film a :
- Un id (clé primaire),
- Un genre_id (clé étrangère, associée au genre dans la table genre),
- Un nom de film, une description,
- Une date de début (date_debut) et une date de fin (date_fin),
- Un label (évalué en termes de contenu pour un public, ici 0 ou 1 pour les films),
- Un âge minimum recommandé pour visionner le film,
- Le nom du fichier image associé au film,
- Un timestamp pour la mise à jour du film (updated_at).

Code SQL :

```
INSERT INTO film (id, genre_id, name, description, date_debut, date_fin,
```

label, age_minimum, image_name, updated_at) VALUES
(1, 2, 'Ghosted', 'Lorsque Cole tombe follement amoureux...', CURDATE(), DATE_ADD(CURDATE(), INTERVAL 6 DAY), 0, '12', 'ghosted.jpg', NOW()),
(2, 1, 'Avatar : la voie de l'eau', 'Se déroulant plus d'une décennie après...', CURDATE(), DATE_ADD(CURDATE(), INTERVAL 6 DAY), 1, '12', 'avatar.jpg', NOW());

9.6 Insertion des séances dans les salles

Ce processus est réalisé par plusieurs commandes similaires, où pour chaque film, des séances sont créées dans les salles. Prenons l'exemple de la salle 1 (3DX) :

La requête suivante crée des séances pour les films dans la salle 1 (3DX) avec un horaire de début et de fin :
'14:00:00' comme heure de début et '16:00:00' comme heure de fin.
- price : Le prix de la séance est fixé à 10.
- La jointure croisée (CROSS JOIN) génère les dates des séances (sur 7 jours à partir de la date actuelle) et les films.
- La condition 'NOT EXISTS' garantit qu'une séance n'est pas créée si elle existe déjà pour ce film et cette salle.

Cette requète utilise plusieurs techniques pour générer les séances et s'assurer qu'il n'y a pas de doublon.

Voici les principales étapes :

1. **Jointure croisée (CROSS JOIN)** : Cette opération génère toutes les combinaisons possibles entre les films et les jours. Cela permet de créer des séances pour chaque film sur une période de 7 jours à partir de la date actuelle.
2. **Génération de dates** : La liste de jours (de 0 à 6) est utilisée pour générer des dates pour chaque film. Chaque film sera associé à une date différente, sur 7 jours consécutifs.
3. **Prix de la séance** : Chaque séance se voit attribuer un prix (ici, 10).
4. **Condition 'NOT EXISTS'** : Cette condition permet de vérifier si une séance pour un film, dans une salle et à une date donnée existe déjà dans la base de données. Si une telle séance existe déjà, elle ne sera pas insérée, garantissant ainsi qu'il n'y ait pas de doublon.

5. **Création des séances** : Si aucune séance dupliquée n'est trouvée, la séance est insérée dans la table 'seance'.

Code SQL :

```
INSERT INTO seance (heure_debut, heure_fin, price, salle_id, date, film_id)
SELECT
    '14:00:00' AS heure_debut,
    '16:00:00' AS heure_fin,
    10 AS price,  -- Prix par séance
    1 AS salle_id,  -- Salle 1 (3DX)
    DATE_ADD(CURDATE(), INTERVAL days.day_offset DAY) AS date,
    films.film_id AS film_id
FROM
    (SELECT 1 AS film_id UNION ALL SELECT 2 UNION ALL SELECT 3 ...) AS
films
CROSS JOIN
    (SELECT 0 AS day_offset UNION ALL SELECT 1 UNION ALL SELECT 2 ...)
AS days
WHERE
    NOT EXISTS (
        SELECT 1
        FROM seance
        WHERE seance.film_id = films.film_id
        AND seance.date = DATE_ADD(CURDATE(), INTERVAL days.day_offset
DAY)
        AND seance.salle_id = 1
    );
```

9.7 Association des séances avec les cinémas

Les séances sont ensuite associées à plusieurs cinémas en utilisant une jointure croisée entre la table 'seance' et une liste de cinémas. Cela permet de distribuer les séances dans plusieurs cinémas (identifiés par leur cinema_id).

Code SQL :

```
INSERT INTO seance_cinema (seance_id, cinema_id)
SELECT
    seances.id AS seance_id,
    cinemas.cinema_id
```

```
FROM
    (SELECT id FROM seance) AS seances
CROSS JOIN
    (SELECT 1 AS cinema_id UNION ALL SELECT 2 UNION ALL ...) AS
cinemas;
```

9.8 Validation de la transaction

Enfin, la transaction est validée avec la commande 'COMMIT;', ce qui signifie que toutes les modifications (cinémas, genres, films, salles, séances, associations) sont appliquées à la base de données de manière permanente. Si une erreur se produit avant ce point, les modifications peuvent être annulées en utilisant 'ROLLBACK'.

Code SQL :

```
COMMIT;
```

10 Explication du plan de test et du déploiement

10.1 Plan de test

10.1.1 Objectifs des tests

- Assurer la qualité des applications (Symfony et mobile) en validant leur bon fonctionnement.
- Identifier et corriger les bugs ou dysfonctionnements avant la mise en production.
- Garantir que les fonctionnalités répondent aux besoins des utilisateurs.

10.1.2 Types de tests

10.1.2.1 Application Symfony (Back-End et Web)

10.1.2.1.1 Tests Unitaires

Vérifient des fonctionnalités spécifiques et isolées PHPUnit :

- Validation des données (mot de passe, email, etc.).
- Calcul du prix des réservations selon le format des séances.

10.1.2.1.2 Tests Fonctionnels

Vérifient le bon fonctionnement de parcours utilisateur complet :

- Création de compte et réservation.
- Authentification et création des avis.

10.1.2.1.3 Tests d'Intégration

Testent la communication entre différentes parties de l'application (API, base de données) avec Behat et Postman:

- Appels API pour récupérer les films et séances.
- Interaction avec MySQL et MongoDB.

10.1.2.1.4 Tests End-to-End (E2E)

Vérifient le bon fonctionnement du parcours visiteurs et utilisateurs complets :

Création de compte et authentification :

- Vérification des étapes de création d'un compte utilisateur, y compris la validation des emails et des contraintes de mot de passe.
- Validation du processus de connexion avec redirection vers la fonctionnalité souhaitée en cas d'accès restreint.

Parcours de réservation :

- Sélection d'un cinéma, d'un film, et d'une séance.
- Choix des sièges disponibles (en tenant compte des contraintes pour les personnes à mobilité réduite).
- Vérification du calcul du prix en fonction du format et du nombre de places.
- Validation de la réservation après authentification.

Consultation des films et des séances :

- Affichage des films disponibles avec leurs affiches, descriptions, âges minimums, notes, et labels "coup de cœur".
- Filtrage des films par cinéma, genre, et date.
- Consultation des séances associées à un film, avec indication du prix selon la qualité.

Gestion des commandes :

- Vérification de l'affichage des réservations dans l'espace utilisateur, y compris les détails (film, séance, sièges, etc.).
- Validation du QR code généré pour chaque billet.

Notation des films :

- Vérification de la possibilité pour l'utilisateur de noter un film et de laisser un avis après une séance passée.
- Validation que les avis sont soumis à modération avant publication.

Fonctionnalité "Mot de passe oublié" :

- Simulation du processus de récupération de mot de passe et vérification de l'obligation de le changer après réception.

Contact avec le cinéma :

- Validation de l'envoi d'un message via le formulaire de contact, avec ou sans authentification.

Vérifient le bon fonctionnement du parcours administrateurs et employés complets :

- Création et gestion des comptes utilisateurs et employés :
  Vérification des étapes de création des comptes employés par les administrateurs et reset du mot de passe.
- Authentification et gestion des droits d'accès :
  S'assurer que les administrateurs et employés accèdent uniquement aux fonctionnalités autorisées selon leurs rôles.

- Gestion des films et séances :
  Validation de la création, modification et suppression des films et des séances, en respectant les contraintes définies.
- Gestion des avis utilisateurs :
  Vérification que les employés peuvent valider ou supprimer les avis soumis par les utilisateurs.
- Consultation des statistiques (administrateurs uniquement) :
  Validation de l'affichage des tableaux de bord, incluant les réservations par film sur une période donnée.

10.1.2.2 Application Mobile (Flutter)

10.1.2.2.1 Tests Unitaires

Vérifient les composants isolés (widgets, logique métier) avec flutter test :

- Connexion via la page de login.
- Affichage des séances.
- Génération du QR code.

10.1.2.2.2 Tests d'Interface Utilisateur

Assurent que les widgets Flutter affichent correctement les informations :

- Navigation entre les écrans (films, billets, etc.).
- Affichage des billets.

10.1.2.2.3 Tests Fonctionnels

Vérifient l'interaction avec l'API Symfony flutter driver:

- Récupération des données utilisateur via JWT.
- Consultation des billets.

11 Déploiement application Web

11.1 Préparation du VPS OVH

11.1.1 Commander et configurer le VPS chez OVH

- Accédez à votre espace client OVH et commandez un VPS.
- Connectez-vous via SSH avec les identifiants fournis :

  ssh root@IP_DU_VPS

- Mettez à jour le serveur :

  apt update && apt upgrade -y

11.1.2 Installer les prérequis

- Installez les logiciels nécessaires :

  apt install apache2 php8.2 mysql-server git unzip curl -y

- Installez Composer (gestionnaire de dépendances PHP) et Node.js (pour les assets)

  curl -sS https://getcomposer.org/installer | php

  mv composer.phar /usr/local/bin/composer

  apt install nodejs npm -y

11.1.3 Déploiement de l'application avec Git

11.1.3.1 Cloner le projet depuis le dépôt Git

- Accédez au répertoire où vous souhaitez déployer l’application

  cd /var/www

  git clone https://github.com/votre-utilisateur/votre-repo.git cinephoria

11.1.3.2 Configurer les permissions

- Donnez les droits nécessaires au répertoire de l'application :

  chown -R www-data:www-data /var/www/cinephoria chmod -R 755 /var/www/cinephoria

11.1.3.3 Configurer les variables d'environnement

- Copiez le fichier .env en .env.local

  cp .env .env.local

- Modifiez le fichier .env.local avec vos paramètres de production (base de données, environnement) :

  APP_ENV=prod

  APP_SECRET=cle_secrete_unique

  DATABASE_URL=mysql://cinephoria_user:mot_de_passe_securise@127.0.0.1:3306/cinephoria

  MAILER_DSN=smtp://email@domaine.com:mot_de_passe@smtp:port

  MONGODB_URL=mongodb://127.0.0.1:27017
  MONGODB_DB=cinephoria

11.1.3.4 Installer les dépendances PHP

- Depuis le répertoire du projet, exécutez

  composer install --no-dev --optimize-autoloader

11.1.3.5 Compiler les assets (CSS/JS)

- Installez les dépendances front-end et compilez les fichiers :

  npm install

  npm run build

11.1.3.6 Appliquer les migrations de la base de données :

- Mettez à jour la base de données avec Doctrine en mode Dev :

  symfony console doctrine:database:create

  symfony console make:migration

  php bin/console doctrine:migrations:migrate

11.1.3.7 Installation de MongoDB :

- Importer la clé publique MongoDB :

  curl -fsSL https://pgp.mongodb.com/server-6.0.asc | sudo gpg --dearmor -o /usr/share/keyrings/mongodb-server-6.0.gpg

- Ajouter le dépôt MongoDB :

  echo "deb [signed-by=/usr/share/keyrings/mongodb-server-6.0.gpg] https://repo.mongodb.org/apt/debian bookworm/mongodb-org/6.0 main" | sudo tee /etc/apt/sources.list.d/mongodb-org-6.0.list

- Installer MongoDB :

  apt update

  apt install -y mongodb-org

- Démarrer et activer MongoDB au démarrage :

  systemctl start mongod

  systemctl enable mongod

11.1.3.8 Configurer le VirtualHost Apache :

- Ajoutez un fichier de configuration pour votre application :

  nano /etc/apache2/sites-available/cinephoria.conf

- Contenu du fichier :

```
<VirtualHost *:443>
    ServerName votre-domaine.com
    DocumentRoot /var/www/cinephoria/public
SSLEngine on
    SSLCertificateFile /etc/letsencrypt/live/nom_dossier/fullchain.pem
    SSLCertificateKeyFile /etc/letsencrypt/live/nom_dossier/privkey.pem
    <Directory /var/www/cinephoria/public>
        AllowOverride All
        Require all granted
                FallbackResource /index.php
    </Directory>
</VirtualHost>
```

- Activez la configuration et redémarrez Apache :

```
a2ensite cinephoria.conf
systemctl restart apache2
```

11.1.3.9 Sécuriser le serveur :

- Configurez un certificat SSL avec Let's Encrypt :

```
apt install certbot python3-certbot-apache -y
certbot –apache
```

12 Déploiement de l'application Mobile (Android)

12.1 Configuration des URLs de l'API

- Dans le code de votre application Flutter, remplacez les URLs de l'API pour pointer vers votre serveur de production (par exemple : https://votre-domaine.com/api).
- Dans les fichiers où vous effectuez des requêtes HTTP (par exemple api_service.dart), remplacez les URLs locales (http://127.0.0.1:8000/api) par celles du serveur déployé :

```
final String baseUrl = "https://votre-domaine.com/api";
```

- Vérifiez les variables d'environnement si vous utilisez flutter_dotenv pour gérer différentes configurations (.env pour dev et prod).

12.2 Génération des builds avec Android Studio

12.2.1 Ouvrir le projet Flutter dans Android Studio

- Lancez Android Studio et ouvrez votre projet Flutter.
- Assurez-vous que le SDK Android et Flutter sont bien configurés.

12.2.2 Configurer le fichier build.gradle

- Accédez au fichier android/app/build.gradle et configurez les options suivantes :

```
defaultConfig {
    minSdkVersion 21
    targetSdkVersion 33
}
buildTypes {
    release {
        shrinkResources false
```

```
        minifyEnabled false

        proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'

    }

}
```

12.2.3 Générer le build en mode release

- Générer un APK

  flutter build apk –release

  Le fichier généré se trouve dans build/app/outputs/flutter-apk/app-release.apk.

- Générer un AAB (Android App Bundle) pour le Play Store :

  flutter build appbundle –release

  Le fichier se trouve dans build/app/outputs/bundle/release/app-release.aab.

12.3 Signature et Publication sur le Google Play Store

12.3.1 Générer une clé de signature

- Si ce n'est pas encore fait, générez un fichier .keystore pour signer l'application :

  keytool -genkey -v -keystore keystore.jks -keyalg RSA -keysize 2048 -validity 10000 -alias nom_alias

  Placez le fichier keystore.jks dans android/app/.

12.3.2 Ajouter la signature à gradle.properties

- Dans android/gradle.properties, ajoutez :

  storeFile=keystore.jks

storePassword=VOTRE_MOT_DE_PASSE

keyPassword=VOTRE_MOT_DE_PASSE

keyAlias=nom_alias

12.3.3 Modifier android/app/build.gradle pour utiliser la signature

- Ajoutez ces lignes dans signingConfigs :

```
signingConfigs {
    release {
        storeFile file("keystore.jks")
        storePassword System.getenv("storePassword") ?: "VOTRE_MOT_DE_PASSE"
        keyAlias System.getenv("keyAlias") ?: "nom_alias"
        keyPassword System.getenv("keyPassword") ?: "VOTRE_MOT_DE_PASSE"
    }
}
```

- Puis associez cette signature au build release :

```
buildTypes {
    release {
        signingConfig signingConfigs.release
    }
}
```

12.3.4 Vérifier et publier sur Google Play

- Testez la signature :

jarsigner -verify -verbose -certs build/app/outputs/flutter-apk/app-release.apk

- Uploadez l'AAB sur Google Play Console pour publication.

13 Explication de la démarche afin de proposer un déploiement continu

13.1 Application web

13.1.1 Travailler en local

- Modifier le code sur la machine locale.
- Tester les modifications en local pour s'assurer que tout fonctionne.

13.1.2 Commit & Push vers Git

git add .

git commit -m "Description des modifications"

git push origin main # Ou une autre branche

13.1.3 Se connecter au serveur en production

ssh user@ton_serveur

13.1.4 Aller dans le dossier du projet et pull les modifications

cd /chemin/de/ton/projet

git pull origin main # Ou la branche correspondante

13.1.5 Mettre à jour les dépendances

composer install --no-dev --optimize-autoloader

npm install && npm run build

13.1.6 Mettre à jour la base de données

php bin/console doctrine:migrations:migrate

13.1.7 Vider le cache et redémarrer les services

php bin/console cache:clear

systemctl restart apache2

13.2 Application mobile

13.2.1 Développer et tester en local

- Modifier le code sur ta machine locale.
- Tester l'application sur le téléphone :

  flutter run
- Vérifier les erreurs potentielles :

  flutter analyze

  flutter test

13.2.2 Commit & Push vers Git

git add .

git commit -m "Mise à jour Flutter"

git push origin main # Ou une autre branche

13.2.3 Générer un build APK pour Android

- Générer un APK pour test hors store :

  flutter build apk

13.2.4 Générer un AAB pour publier sur le Play Store

flutter build appbundle

13.2.5 Signer l'AAB pour Google Play

- Générer une clé Keystore :

  keytool -genkey -v -keystore ~/keystore.jks -keyalg RSA -keysize 2048 -validity 10000 -alias key-alias

- Configurer la signature dans android/app/build.gradle :

```
android {
    signingConfigs {
        release {
            storeFile file("chemin/vers/keystore.jks")
            storePassword "ton_mot_de_passe"
            keyAlias "key-alias"
            keyPassword "ton_mot_de_passe"
        }
    }
    buildTypes {
        release {
            signingConfig signingConfigs.release
        }
    }
}
```

- Générer à nouveau le build signé :

  flutter build appbundle

13.2.6 Publier la mise à jour sur le Play Store

- Aller sur Google Play Console : play.google.com/console
- Sélectionner l’application.
- Aller dans Gestion des versions > Créer une nouvelle version.
- Uploader le fichier AAB (app-release.aab).
- Ajouter les notes de version et soumettre la mise à jour.