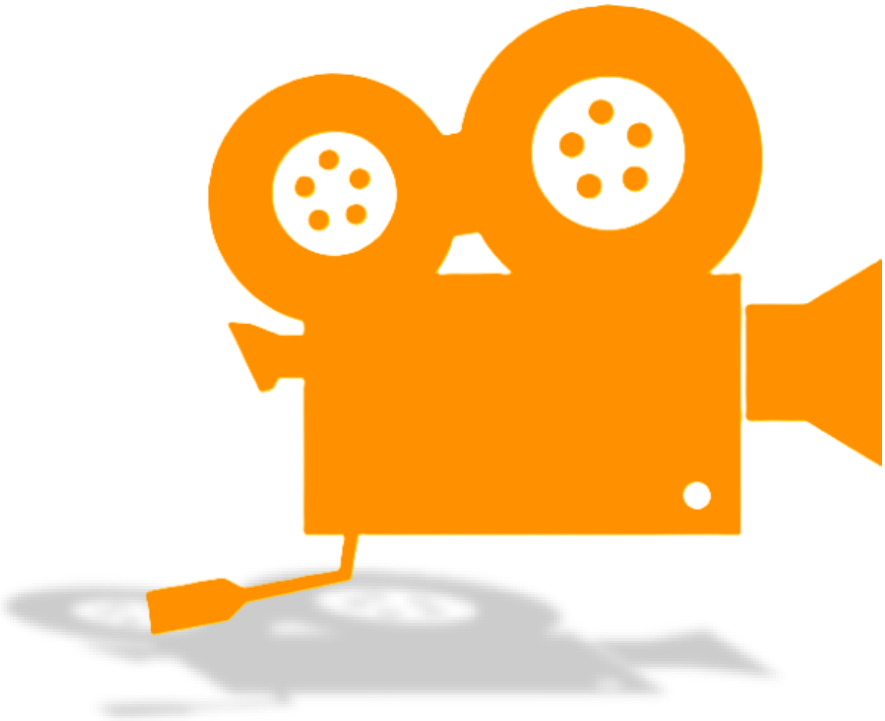
Documentation technique et gestion de projet



CINEPHORIA

1. Table des matières

2. Gestion de projet

1 Méthodologie utilisée

Nous avons suivi une approche agile inspirée de Scrum avec :

- Un backlog produit listant les fonctionnalités (authentification, réservation, gestion des films, espace employé/admin).
- Des sprints courts (1 à 2 semaines) permettant d'avancer par incréments.
- Des réunions régulières pour ajuster les priorités et faire des points d'avancement.

2 Outils de gestion

- Trello pour la planification et le suivi des tâches : https://trello.com/b/TcQIL1xa/projet-cinephoria
- Git/GitHub pour la gestion du code source avec un workflow feature branch + pull request.
  - Cinephoria Front-end -> https://github.com/KZUJL/CinephoriaWeb
  - Cinephoria Back-end -> https://github.com/KZUJL/ApiCinephoria
  - Cinephoria Mobile -> https://github.com/KZUJL/CinephoriaMobileApp
  - Cinephoria Desktop -> https://github.com/KZUJL/CinephoriaDesktop
- Teams pour la communication.

3. Documentation technique

1 Architecture logicielle

L'application suit une architecture **multi-couches** :

- **Frontend mobile & web (Flutter + Vue.js)** : gère l'interface utilisateur et envoie les requêtes API.
- **Backend (ASP.NET Core Web API)** : expose des endpoints REST (/api/Login, /api/Reservation, /api/Movies).
- **Base de données relationnelle (MySQL)** : stockage structuré des utilisateurs, films, séances, réservations.
- **Base de données NoSQL (MongoDB)** : stockage des réservations et notations des films

2 Réflexions initiales technologiques

- **Flutter** a été retenu pour le développement mobile car il permet de créer une application fonctionnant à la fois sur Android et iOS avec une seule base de code. Cela simplifie la maintenance, réduit les coûts de développement et garantit une homogénéité dans l'expérience utilisateur sur les différentes plateformes.

- **ASP.NET Core** a été choisi pour le back-end car c'est un framework moderne, robuste et sécurisé. Il offre une excellente prise en charge des API REST et s'intègre facilement avec des bases de données relationnelles comme MySQL mais aussi avec des solutions NoSQL comme MongoDB. Ce choix garantit à la fois performance et évolutivité.

- **MySQL** a été utilisé pour la gestion des données relationnelles (films, séances, utilisateurs). Sa fiabilité, sa compatibilité avec de nombreux environnements et sa large adoption en font un choix pertinent pour stocker des données structurées nécessitant des relations fortes.

- **MongoDB** complète l'architecture en permettant de gérer des données non structurées ou semi-structurées comme les journaux d'événements, les historiques et certains cas de réservation. Sa flexibilité est un atout dans un projet qui combine des données très structurées et d'autres plus libres.

- **Fly.io** a été retenu comme solution d'hébergement cloud car il facilite le déploiement rapide d'applications via Docker. C'est une plateforme légère qui permet de scaler facilement en fonction des besoins, tout en gérant de manière sécurisée les secrets et les variables d'environnement.

3 Configuration de l'environnement de travail

- **IDE :**

Visual Studio 2022 pour le développement backend en ASP.NET Core.

Visual Studio Code pour le frontend web (Vue.js) et Flutter.

Android Studio pour l'émulation mobile et le build de l'application Flutter.

- **Outils de développement :**
    - *GitHub Desktop* pour la gestion de version et le suivi des commits de manière visuelle.
    - *Git en ligne de commande* pour les commandes Git complexes
    - *Docker* pour le déploiement conteneurisé de l'API sur Fly.io.
    - *Swagger* pour tester les endpoints de l'API en local et en production.

- *MySQL Workbench* pour la gestion et la visualisation des données relationnelles (SQL).
- *MongoDB Compass* pour l'exploration des collections et le debug des données NoSQL.

- **Système d'exploitation :**
  Développement et tests réalisés sous Windows 11

4 Modèle conceptuel de données (MCD)

Voir les documents joints dans le dossier Livrable du GIT :
https://github.com/KZUJL/CinephoriaWeb/tree/main/livrables

5 Plan de test

Afin de garantir la fiabilité de l'application, plusieurs types de tests ont été mis en œuvre, aussi bien sur la partie front-end que sur le back-end.

Tests unitaires (Front-end)

Le projet Vue.js inclut une arborescence src/tests/unit contenant des fichiers de test pour la majorité des composants (ex. MovieManagement.spec.ts, ReservationOverview.spec.ts, LoginModal.spec.ts).
Ces tests, réalisés avec **Vitest** et **Vue Test Utils**, permettent de vérifier le rendu correct des composants ainsi que leur comportement de base (affichage, interactions simples, validations de formulaire).
Exemple : ChangePasswordComponent.spec.ts vérifie que le composant de changement de mot de passe est correctement monté et accessible dans le DOM.

Tests unitaires (Back-end)

Côté API, des tests unitaires ont été développés dans le projet ApiCinephoria.Tests. Ils utilisent **xUnit** et **Moq** pour isoler les contrôleurs et vérifier leurs comportements indépendamment de la base de données.
Exemple : les tests du ReservationController valident que les méthodes Get, Post, Update et Delete renvoient les bons codes HTTP et appellent bien les services associés.

Tests d'intégration (Back-end)

Un projet ApiCinephoria.Tests.Integration a été créé pour exécuter des tests d'intégration complets avec **WebApplicationFactory**.
Ces tests envoient de vraies requêtes HTTP sur l'API et valident le cycle complet : création d'une réservation (POST), modification (PUT), consultation (GET), et

suppression (DELETE).
Ils permettent également de vérifier la robustesse de l'API, par exemple lorsqu'un champ obligatoire est manquant (BadRequest).

Tests fonctionnels (scénarios utilisateurs)

En complément, des scénarios utilisateurs ont été simulés sur le front (connexion, navigation dans les films, réservation de places, confirmation).
Ces tests ont été exécutés manuellement via l'interface web et mobile afin de valider l'expérience utilisateur globale.

Tests de non-régression

Après chaque itération et déploiement sur Fly.io, les fonctionnalités essentielles (authentification, réservation, gestion des films et des salles) ont été revérifiées manuellement afin de s'assurer qu'aucune régression n'était introduite.

UAT

Tests d'acceptation utilisateur : le client a réalisé des tests d'acceptation de l'interface.

6 Déploiement

Le projet a été déployé sur plusieurs environnements :

- **Backend (API ASP.NET Core)** : conteneurisé avec Docker et déployé sur **Fly.io**. Les endpoints REST sont accessibles publiquement et servent de point d'entrée aux frontends (web, mobile, desktop).
- **Base de données relationnelle (MySQL)** : hébergée sur Railway (solution cloud) pour bénéficier d'une gestion simplifiée, d'une disponibilité continue et d'une intégration fluide avec l'API.
- **Base NoSQL (MongoDB)** : déployée sur MongoDB Atlas, plateforme managée qui permet de gérer les données non structurées (incidents, logs) avec un haut niveau de sécurité et de scalabilité.
- **Frontend mobile (Flutter)** : généré en APK via la commande flutter build apk et installé directement sur un terminal Android pour les tests. Ce mode de déploiement a permis de valider l'expérience utilisateur sur mobile.
- **Frontend web (Vue.js)** : Déployé sur Fly.io (conteneurisé avec NGINX).
- **Application desktop (Flutter)** : compilée pour Windows et publiée en local. Cette approche permet de distribuer un exécutable utilisable sans dépendance à un store externe.

7 Intégration et déploiement

Dans le cadre de ce projet réalisé seul, je n'ai pas mis en place une véritable chaîne CI/CD automatisée. Le développement et les tests ont été effectués en local, avec un ensemble de jeux de test pour valider les principales fonctionnalités (connexion, réservation, gestion des films).

Une fois les tests validés, j'ai procédé au déploiement manuel des différentes parties de l'application sur Fly.io (backend et frontend web). Chaque itération a permis de corriger les éventuels bugs rencontrés en production et d'améliorer la stabilité de l'application.

Cette approche manuelle, même si elle n'automatise pas complètement l'intégration et le déploiement, m'a permis de suivre un cycle clair :

- Développement et validation locale,
- Mise en ligne,
- Correction progressive en fonction des retours et anomalies constatées.