Documentation technique de l'application Cinéphoria

Benjamin BALET

Table des matières

Introduction

Ce document est une documentation technique de votre application Cinéphoria, il contient :

- Architecture logicielle de l'application : explication du choix des technologies ainsi que du fonctionnement global.
- Réflexions initiales technologiques sur le sujet.
- Configuration de l'environnement de travail.
- Modèle conceptuel de données (MCD).
- Diagramme d'utilisation, diagramme de séquence.
- Explication de votre plan de test ainsi que de votre déploiement.
- Explication de la démarche que vous avez eue afin de proposer un déploiement continu (CI/CD).
- Explication de la transaction SQL

Architecture logicielle

La figure ci-dessous montre les différents composants du système Cinéphoria :

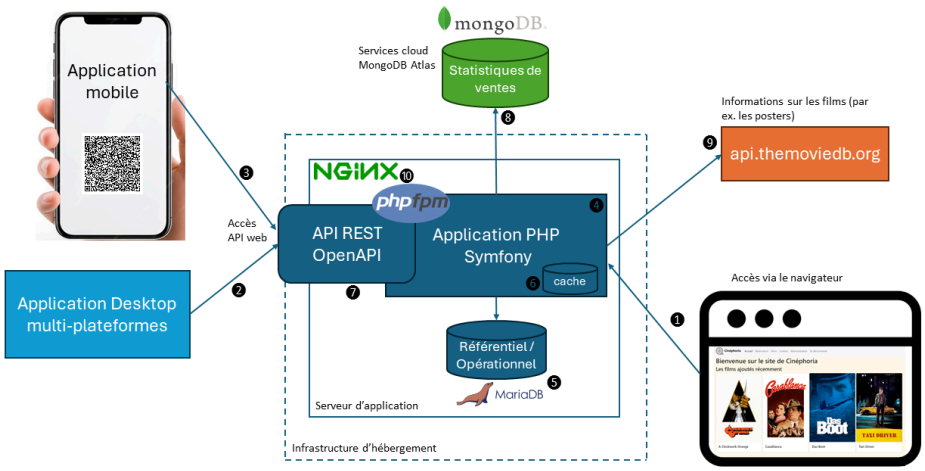

Figure 1: Schéma d'architecture logicielle.

Voici la description du système et de son fonctionnement :

1. Un navigateur permet d'accéder à l'application web avec une connexion sécurisée. Le frontend est développé avec la bibliothèque Stimulus qui donne l'illusion d'une application SPA lorsque l'on navigue entre les pages (car seulement une partie est rechargée). Cette application permet de faire de la réservation de tickets et de gérér les données référentielles et opérationnelles du système.
2. Une application multiplateformes accède à l'application via une API REST (avec une connexion sécurisée). Un mécanisme de jetons JWT permet de transmettre l'identité de l'utlisateur entre les appels REST. Cette application permet de gérer les incidents liées aux salles des cinémas.
3. Une application Android accède à l'application via une API REST (avec une connexion sécurisée). Un mécanisme de jetons JWT permet de transmettre l'identité de l'utlisateur entre les appels REST. Cette application permet aux clients de voir leurs réservations (notamment les séances du jour et futures) et d'afficher le QR Code de leurs tickets. Cette même application peut être utilisée par un employé afin de scanner ces tickets et de vérifier leur validité.
4. L'application principale du système est développée pour PHP8.2+ avec le framework MVC Symfony 7.0. Elle est compatible avec les principales bases de données relationnelles, car elle s'appuie sur un ORM. Elle s'appuie sur différentes bibliothèques PHP par ex. pour générer un QR Code, gérer les jetons JWT, ou créer une API REST avec OpenAPI.

5. La base de données relationnelle recommandée est MariaDB. Elle stocke les données référentielles et opérationnelles du système.
6. Un cache sur le système de fichier du serveur web permet de réduire les appels à l'API themoviedb en stockant les affiches des films et évitant de les redemander une fois en cache.
7. Une API REST sécurisée (avec authentification, permissions) permet aux applications tierces d'accéder à certaines fonctionnalités du système.
8. Une base de données NoSQL (mongoDB hébergée sur le cloud) stocke les statistiques de vente de tickets. Le schéma ne contient qu'une collection "Booking" destinée à stocker des documents contenant les réservations. Cette collection est optimisés pour les requêtes de type timeseries (car nous souhaitons des statistiques dans le temps sur les ventes).
9. Pour obtenir des informations sur les films (principalement les posters) nous nous appuyons sur une API publique tierce déjà existante.
10. L'application web est hébergée sur un VPS. Sur lequel est installé nginx (pour les ressources statiques et en *reverse proxy* pour les pages dynamiques). Le code PHP est exécuté par un serveur PHP-FPM (sur lequel opcache est activé).

Réflexions initiales technologiques

Introduction

Pour le projet Cinéphoria, il est essentiel de choisir une architecture et des technologies qui répondent aux besoins spécifiques du projet, tout en garantissant la performance, la sécurité et la maintenabilité de l'application. Cette section détaille les réflexions initiales sur les choix technologiques, en tenant compte des exigences fonctionnelles, des contraintes techniques et des meilleures pratiques du développement logiciel.

Choix de l'Architecture

L'architecture choisie pour ce projet est une architecture monolithique modulaire. Cette approche permet de développer l'application en un seul bloc tout en séparant les différentes fonctionnalités en modules bien définis. Cela facilite la gestion du projet et l'intégration des différentes parties de l'application.

Avantages de l'Architecture Monolithique Modulaire

- **Simplicité :** Une architecture monolithique est plus simple à mettre en œuvre et à gérer pour de petits projets.
- **Performances :** Les appels internes entre modules sont plus rapides car ils se font dans le même espace de mémoire.
- **Déploiement unique :** Toutes les parties de l'application sont déployées en une seule fois, ce qui simplifie le processus de déploiement.

Choix des Technologies

Front-end

Pour le front-end, nous avons choisi d'utiliser les technologies suivantes :

- **HTML5, CSS3 et Bootstrap :** Pour une structure sémantique et un design responsive.
- **Stimulus :** Pour simuler le comportement d'une application SPA (Single Page Application). Stimulus permet d'enrichir les pages HTML avec des comportements dynamiques sans complexité excessive.

Back-end

Le back-end sera développé en utilisant :

- **Symfony (PHP) :** Symfony est un framework PHP robuste et flexible, idéal pour développer des applications web sécurisées et performantes. Il intègre une vaste gamme de fonctionnalités et de bibliothèques pour faciliter le développement.
- **MariaDB :** Pour les données structurées nécessitant des transactions complexes. MariaDB est fiable, robuste et bien supporté. Il supporte les transactions.
- **API Platform :** Pour développer une API REST basée sur le standard OpenAPI et en s'intégrant dans l'écosystème de Symfony.
- **PHP FPM :** Pour le serveur PHP.
- **nginx :** Pour le serveur web en reverse proxy derrière PHP FPM pour les pages dynamiques.

Services cloud

Le backend interagira avec les services Cloud publics suivants:

- **mongoDB Atlas :** https://www.mongodb.com/products/platform/atlas-database pour le stockage des statistiques de ventes dans une base NoSQL.
- **themoviedb :** api.themoviedb.org API pour obtenir des informations sur les films, les posters notamment.

Mobile

Pour l'application mobile, nous utiliserons :

- **Android (Java) :** Java est le langage de programmation principal pour le développement d'applications Android. Il est stable, bien documenté (encore plus que Kotlin) et largement utilisé dans l'industrie. • Zxing pour le scan des tickets (le QRCode) : intent et décodage, • Retofit2 (et OK http) pour les appels API, • Android SVG pour afficher l'image du ticket à scanner (après obtention de son code SVG), • Auth0/JWT pour décoder le token JWT et savoir s'il a expiré.

Bureautique

Pour l'application bureautique, nous avons choisi :

- **Go avec Fyne.io :** Go est un langage de programmation moderne, rapide et efficace. Fyne.io est une bibliothèque graphique multiplateforme qui permet de créer des applications de bureau avec des interfaces utilisateur élégantes et responsive.

Sécurité

La sécurité est une priorité dans le développement de l'application. Les mesures suivantes seront mises en place :

- **Authentification et Autorisation :** Utilisation de JWT (JSON Web Tokens) pour sécuriser les API et vérifier l'identité des utilisateurs.
- **Chiffrement :** Toutes les communications entre les clients et le serveur seront chiffrées via HTTPS.
- **Validation des Entrées :** Toutes les entrées des utilisateurs seront validées et nettoyées pour prévenir les attaques par injection SQL et les attaques XSS (Cross-Site Scripting).

Configuration de l'environnement de travail

Pour le développement web

L'environnement de travail se base sur Windows, WSL et Docker. WSL permet d'exécuter des commandes Linux ce qui peut faciliter l'emploi d'utilitaires diffcilement disponibles sous Windows. Concernant Docker, nous nous basons sur le focntionnement de Symfony sans configuration particulière pour les *dev containers*.

Pour le développement desktop

L'environnement de travail se base sur WSL et Docker pour réaliser la cross-compilation. Un environnement Windows est utilisé afin de tester la version win_64 du produit. Le projet utilise l'API déployée sur l'environnement cible (i.e. https://cinephoria.jorani.org/api/docs).

Pour le développement mobile

L'environnement de travail se base sur Android Studio sans autre dépendance locale. Le projet utilise l'API déployée sur l'environnement cible (i.e. https://cinephoria.jorani.org/api/docs).

Modèle conceptuel de données

Le schéma ci-dessous est le MCD de l'application :

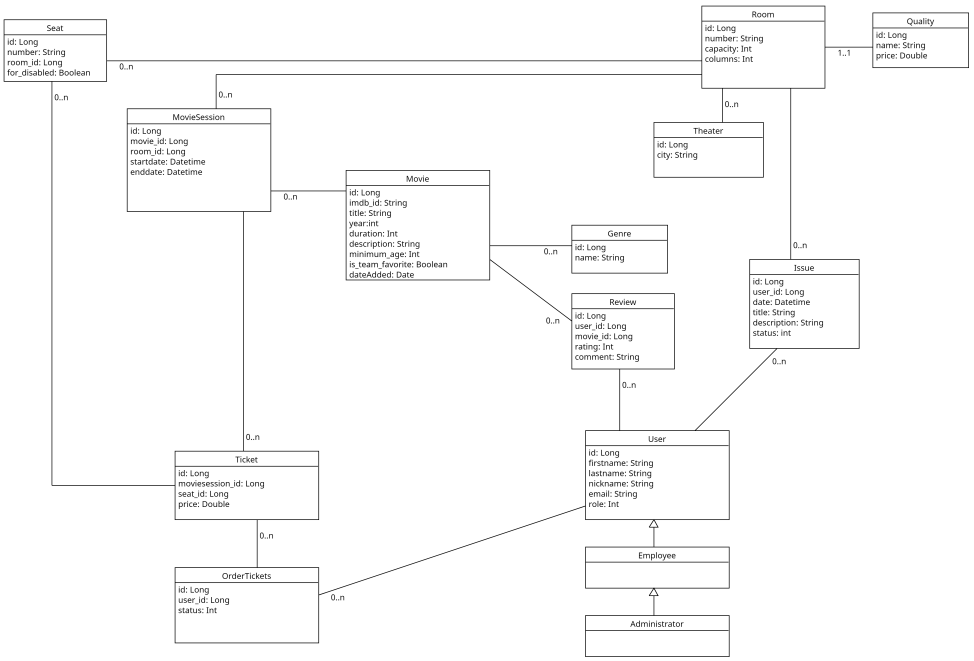

Figure 2: Modèle Conceptuel des Données.

Base de données NoSQL

Concenrnant la base de données NoSQL, nous avons opté pour une base de données orientée document. Nous avons défini la collection Booking comportant le document structuré comme dans cet exemple :

```
timestamp: 2024-02-24T20:46:21.227+00:00
movieTitle: "The Godfather"
_id: 65da559dd187bb9ebd221cf0
tickets: 4
```

Un index est postionné sur le champ puisque les requêtes concernent des statistiques chronologiques.

Diagramme d'utilisation et d'activité

Le diagramme d'uilisation est le suivant:

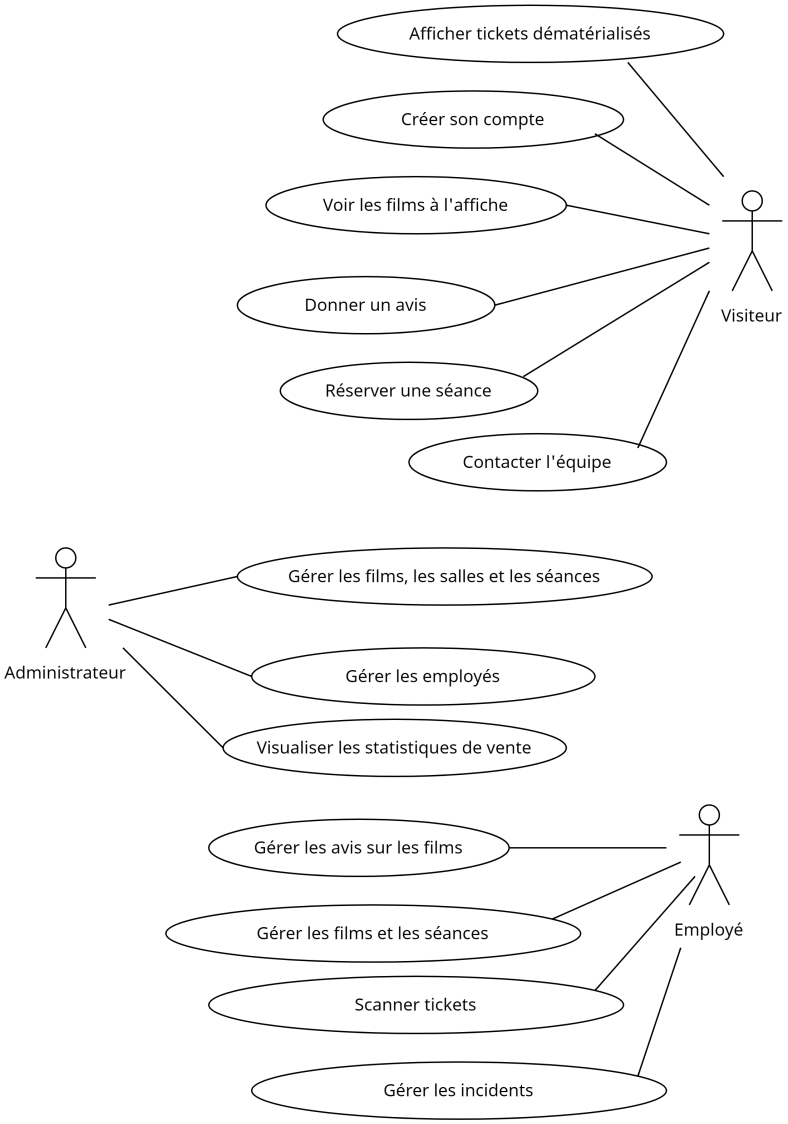

Figure 3: Use Case Diagram.

Concernant les diagrammes d’activité, seul le principe de fonctionnement de la réservation est intéressant à modéliser :

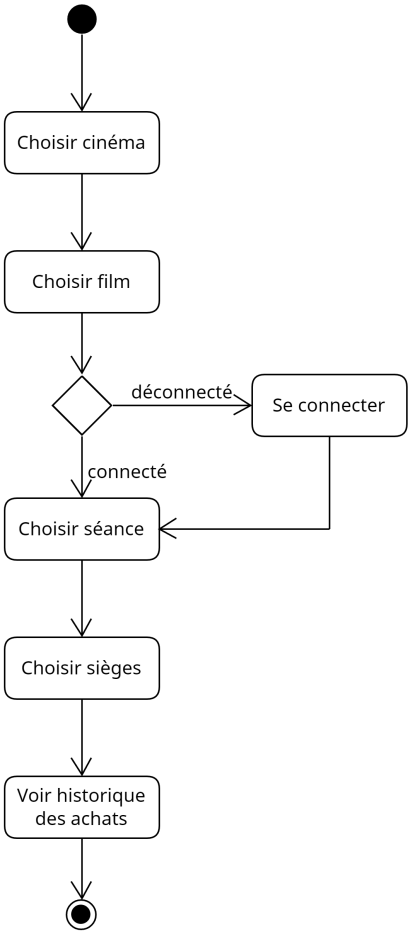

Figure 4: Diagramme de séquence de la réservation.

Plan de test et de déploiement

Plan de test

Le plan de test a consisté à détailler les User Stories de l'application en une liste détaillées d'exigences (voir le document de gestion du projet)). À chaque itération je vérifiais que l'exigence était atteinte avec un ou plusieurs tests manuels. À la livraison finale, j'éffectuais un test complet et de bout en bout de l'application sur l'ensemble des exigences. Une attention particulière a été apportée à l'API REST qui a été testée avec des tests fonctionnels automatisés et un objectif de couverture de 100%. En effet l'API REST est le point d'entrée des applications desktop et mobile.

Plan de déploiement

Objectifs et Portée

- Objectifs du déploiement : livrer en continu les nouvelles fonctionnalités
- Portée du déploiement : intégralité de l'application à chaque déploiement

Environnement de Déploiement

- Environnements cibles : serveur de test (qui jouera le rôle du serveur de production)
- Configuration matérielle et logicielle : VPS 2vCPU et 2 Go de RAM, 20Go de disque SDD

Ressources Nécessaires

- Équipe de déploiement : développeur de l'application

- Outils et technologies : GitHub Action, rsync+SSH sur le serveur cible. Serveur installé et pré-configuré.

Calendrier de Déploiement
- Dates clés : à chaque itération. Livraison finale le 22/07/2024.
- Phases de déploiement : tests sur l'enviornnement local du développeur, un fois validé merge sur la branche main provoque le déploiement.

Procédures de Déploiement
- Étapes détaillées : lancer manuellement l'action GitHub de déploiement ou faire un push sur la branche main.
- Instructions spécifiques : tester l'accès à l'application après le déploiement.

Gestion des Risques
- Identification des risques : liaison avec MongoDB Atlas non fonctionnelle.
- Plans de mitigation : tester l'accès à l'application après le déploiement.

Tests et Validation
- Plan de test : cf. § Plan de test.
- Critères d'acceptation : tout tests passants.

Communication
- Plan de communication : N/A.
- Notification des utilisateurs : messages aux utilisateurs inscrits avec une description de la nouvelle version.

Suivi Post-Déploiement
- Support et maintenance : par le développeur du projet.
- Surveillance et reporting : Suivi avec le logiciel DataDog et alarming sur indisponibilité et problèmes de capacité.

Documentation et Formation
- Documentation : documentation technique.
- Formation : N/A.

Déploiement continu (CI/CD)
La chaîne CI/CD repose sur des actions GitHub.

Intégration continue
L'intégration continue consiste en une action GitHub exécutée à chaque push Git sur le repository. L'environnement de test est alors préparé sur un runner avec une préparation de l'application et des fixtures pour charger le jeu de données. Source de l'action GitHub d'intégration continue.

Déploiement continu
La démarche a consisté à préparer un serveur de déploiement avec un VPS de la société OVH.
Il a fallut commencer par préparer les prérequis :
- Installation et durcissement de l'OS.
- Installation de PHP, des extansions (par ex. mongoDB, mySQL, etc.) et de FPM.
- Installation et configuration de nginx.
- Création d'une entrée DNS.
- Obtention de certificats SSL avec Let's Encrypt.

- Installation et sécuritsation de MariaDB
- Ajout de l'IP du VPS dans la liste blanche du cloud de MongoDB (Atlas).
- Installation et configuration de rsyncd qui pointe sur le répertoire d'installation du site web.

On créé ensuite un utilisateur Linux dédié au déploiement et à l'exécution du site web.

On prépare des clés SSH privées/publiques qui serviront à la connexion SSH sans mot de passe (par échange de clés) entre la forge logicielle (GitHub) et notre serveur de déploiement.

Le déploiement proprement dit se déroule ainsi:

- Un paquet logiciel est préparé sur la forge logicielle GitHub (sur un runner): • Téléchargement et compilation des dépendances JS. • Téléchargement des dépendances PHP.
- Une fois, prêt le paquet est déployé par synchronisation de fichier entre le runner et le serveur cible.
- Une fois déployé, des opérations de post-installation sont lancées

Comme nous ne nous sommes pas dans le cas d'une application réelle, la basse de données est effacées à chaque déploiement.

Source de l'action GitHub de déploiement continu.

Transaction SQL

La transaction SQL présente sur le dépôt GitHub (/copy-theater.sql) a pour but de dupliquer un cinéma. Cette duplication est dans une transaction, car la copie de la structure du cinéma (salles, sièges) peut échouer à une des étapes.

Pour l'utiliser, on indique le nom du cinéma qui sera la source de la copie (par ex. "Nantes") et le nom du nouveau cinéma (par ex. "Marseille"). Le script copie alors les salles et leur disposition de sièges à l'indentique

Mode d'emploi :

1. Remplacer les valeurs entre crochets en début de script par les valeurs correspondantes
2. Exécuter le fichier dans un client SQL (phpMyAdmin, etc.) ou via la ligne de commande
3. Vérifier que la transaction s'est bien déroulée

Exemple d'utilisation (app est la base de données cible) : sudo mysql -u root -D app < copy-theater.sql

Annexes

Table des illustrations