Cahier des charges

Contexte du Projet

Le cinéma Solaris va bientôt ouvrir ses portes et a besoin d'un site web intuitif afin de permettre à ses futurs usagers de consulter films et séances, mais également effectuer une réservation. Aussi, le cinéma souhaite avoir une interface réservée pour la gestion des films, des séances, des salles et des utilisateurs. Pour ce faire, le cinéma a fait appel à un prestataire informatique qui saura répondre à ses besoins.

Objectifs du Projet

- Permettre aux utilisateurs de consulter les films et les séances disponibles.
- Processus complet de réservation de places.
- Offrir aux administrateurs une interface de gestion complète pour les films et les séances.
- Intégrer une API externe (TMDB) pour obtenir les données des films.
- Assurer la sécurité des transactions et des données utilisateurs.

Expression des besoins

Fonctionnalités Utilisateurs

- **Consultation des Films :** Les utilisateurs peuvent parcourir une liste de films disponibles avec des détails tels que le titre, la durée, le synopsis et l'affiche, qui ont étés préalablement récupérés via l'API TMDB.
- **Consultation des Séances :** Accès à l'horaire des séances disponibles pour chaque film, avec possibilité de filtrer par date.
- **Réservation de Places :** Les utilisateurs peuvent réserver une ou plusieurs places pour une séance spécifique, avec visualisation du plan de salle.
- **Notifications :** Envoi de mail pour confirmer les réservations + E-billet.
- **Gestion du compte utilisateur :** Modification des informations personnelles, suppression du compte et consultation des réservations effectuées

Fonctionnalités Administrateurs

- **Gestion des Films :**
  - Ajout d'un film depuis une interface de consultation de l'API TMDB et enregistrement des données utiles en base de donnée (titre, durée, genre, etc).
  - Modification d'un film
  - Suppression de films.
  - Mise en place des films qui seront en tête d'affiche
- **Gestion des Séances :**
  - Consultation des séances sur un planning visuel.
  - Ajout de séances sur un planning visuel.
  - Modification de séances.
  - Suppression de séances.

Description de l'Environnement Technologique

Technologies Utilisées :

- **Framework Backend :** Laravel (PHP) pour gérer la logique d'application, l'authentification, et les interactions avec la base de données.
- **Base de Données :** MySQL/MariaDB pour stocker les données des utilisateurs, films, séances, salles, réservations (et abonnements).
- **API Externe :** TMDB (The Movie Database) pour récupérer les informations et métadonnées des films.
- **Frontend :** Utilisation de Blade (le moteur de template de Laravel), HTML, CSS, et JavaScript pour l'interface utilisateur.
- **Bibliothèques et Frameworks Frontend :** TailwindCSS, AlpineJS
- **Sécurité :** Utilisation de Laravel Breeze pour l'authentification sur le site web, et Laravel Sanctum pour la sécurisation des endpoints API.
- **Paiement en Ligne :** Intégration d'une solution de paiement en ligne comme Stripe ou PayPal.

Infrastructure :

- **Hébergement :** Solution d'hébergement chez un prestataire spécialisé.

Détails de la réalisation

Le framework Laravel

Laravel est un framework PHP fournissant un cadre de travail standardisé et complet pour le développement de sites web en utilisant le design pattern MVC (Model – View – Controller).
De plus en plus populaire, il en est aujourd'hui à sa 12ème version (la 11 à été utilisée pour ce projet).

Le design pattern MVC

Cette architecture de développement repose sur 3 types de composants pour faire fonctionner une application. Le but est de séparer les responsabilités au sein d'une application, et ainsi la rendre plus modulaire et maintenable. Voici leur rôle dans le cadre d'un projet Laravel :

- **Modèle :** Il s'occupe de récupérer et de gérer les données de la base de données, en les instanciant dans des objets PHP prêts à être manipulés. Laravel propose en outre un **ORM Eloquent**, qui permet de s'affranchir du formalisme des requêtes SQL classiques pour une approche plus intuitive et proche d'une syntaxe "Orienté Objet".

- **Vue :** La vue sert à l'affichage des données, elle va, dans le cadre d'un projet web Laravel, contenir le code HTML. Laravel propose le moteur de template **Blade** pour gérer ces vues, ce dernier offre la possibilité de modulariser les vues (layouts, composants…), et d'intégrer du code php pour conditionner ou formater l'affichage des données dynamiques envoyées par le contrôleur.

- **Contrôleur :** Le contrôleur à pour tâche de faire l'intermédiaire entre les Modèles et les Vues. C'est lui qui va faire appel aux classes Modèles pour récupérer efficacement les données et les préparer/formater pour leur utilisation dans une Vue. C'est aussi lui qui va, lorsque l'utilisateur va effectuer une requête au travers de la vue, effectuer la logique de traitement des données reçues.

- **Les routes :** Même si leur mention n'est pas faite dans l'acronyme MVC, leur importance est capitale pour comprendre le fonctionnement de cette architecture dans Laravel. Les routes vont permettre de définir, au travers d'adresse HTTP et de méthodes (GET,POST…) l'accès aux différentes pages de l'application, en précisant quelle contrôleur et quelle fonction de ce dernier utiliser.

Points clés du projet

La conception du site web a été faite en tentant de suivre les règles de base du développement avec Laravel.
La base de donnée MySQL a donc été créée en utilisant le système de migration proposé par le framework (définition des tables et des colonnes en utilisant l'ORM Eloquent), des modèles ont étés créés pour chacune de ces tables, en utilisant encore une fois les propriétés "implicites" de Laravel pour établir les relations entre les différentes tables (voir Schéma MCD plus haut). Le développement s'est ensuite fait en respectant le pattern MVC présenté ci-dessus.

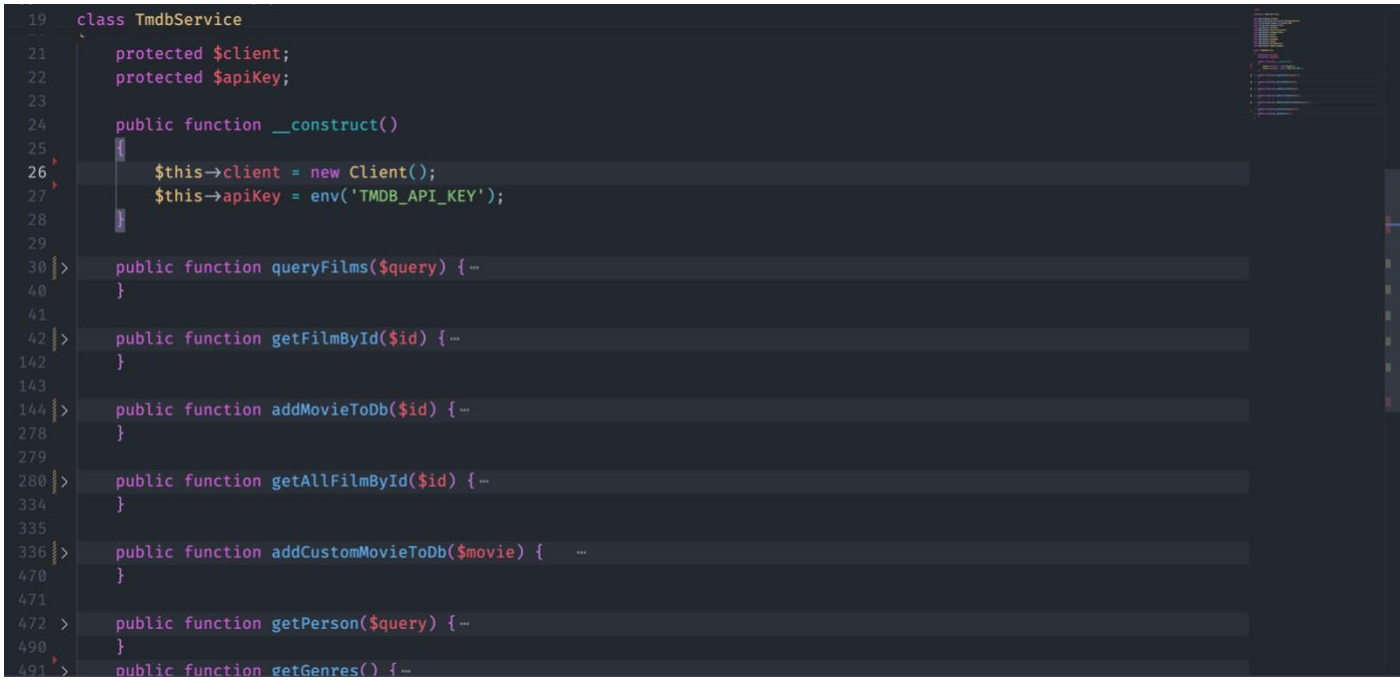
Néanmoins, certaines fonctionnalités spécifiques ont dû être développé pour répondre aux impératifs du projet.

Intégration de l'API TMDB

Pour pouvoir récupérer efficacement et de manière fiable des données relatives à des films, l'usage d'un service tiers a semblé être la meilleure solution. The Movie Database est une base de données recensant une grande quantité d'informations sur les films et séries, et qui propose un accès API gratuit dans le cadre d'un usage à but non lucratif. Il faut cependant s'inscrire sur leur site internet pour récupérer une "Clé API" permettant l'usage du service.

Une classe "TmdbService" a donc été créée pour gérer les requêtes API vers le service, elle comporte un ensemble de méthodes servant à la récupération, au filtrage et au formatage des données en vue de leur insertion dans les différentes tables de la base de données, tout en respectant les relations entre ces tables.

Cette classe service est utilisée dans le contrôleur relatif à la gestion (ajout/modifications) des films, et également dans le Seeder approprié, pour générer une liste de films lors des phases de tests.



```php
class TmdbService
{
    protected $client;
    protected $apiKey;

    public function __construct()
    {
        $this->client = new Client();
        $this->apiKey = env('TMDB_API_KEY');
    }

    public function queryFilms($query) {…
    }

    public function getFilmById($id) {…
    }

    public function addMovieToDb($id) {…
    }

    public function getAllFilmById($id) {…
    }

    public function addCustomMovieToDb($movie) {…
    }

    public function getPerson($query) {…
    }
    public function getGenres() {…
```

Intégration de FullCalendar

Pour la gestion des séances, l'ambition a été de mettre à disposition des employés une interface intuitive pour consulter, modifier et définir le planning des séances du cinéma. Le choix a été fait d'utiliser la bibliothèque JavaScript FullCalendar pour arriver à ces fins.

Cahier des Charges

1. Présentation Générale

1.1. Contexte

Le cinéma Solaris souhaite développer une application mobile permettant la validation/le compostage des réservations clients, lors de leur arrivée au cinéma. Cette application visera exclusivement les appareils Android. Elle interagira avec la partie Back-end du site web du cinéma (**cinemasolaris.fr**), qui gèrera l'interaction directe avec la base de données MySQL.

1.2. Objectif

L'application doit permettre aux agents du cinéma de **vérifier l'intégrité des réservations client** via un **scan de QR code** (que le client aura reçu au moment de sa réservation), et de les valider si tout est en ordre, permettant ainsi au client d'accéder à la séance. Son usage devra être restreint aux employés du cinéma, nécessitant donc la mise en place d'une authentification fiable.

1.3. Fonctionnalités principales

- Scan de QR Code de réservation.
- Connexion sécurisée via une API qui contactera le back-end du site cinemasolaris.fr
- Vérification et mise à jour de la réservation via cette même API.
- Affichage des résultats du scan, en fonction du degré de validité du QR code soumis.

2. Spécifications Fonctionnelles

L'application sera épurée et comportera 3 pages :

- Page de connexion (point d'entrée initial de l'application)
- Page de Scan (point d'entrée de l'application une fois l'utilisateur authentifié)
- Page de résultat

2.1. Page de connexion

- Champs : **Login** et **Mot de passe, ces identifiants seront identiques à ceux requis pour la connexion sur le site web cinemasolaris.fr.**

- Envoi des informations via **requête HTTP POST** au backend Laravel.
- Mot de passe chiffré avant envoi.
- Le serveur vérifiera l'existence de l'utilisateur et ses droits.
- Retour : **Token d'authentification** à utiliser pour les requêtes suivantes et comme témoin de connexion.

Lorsque la connexion d'un utilisateur est réussi, le token d'authentification sera stockée dans la mémoire sécurisée de l'appareil, évitant d'avoir à se reconnecter à chaque réouverture de l'application, **dans un laps de temps précis à déterminer.**

2.2 Page de Scan

- Interface de caméra épurée avec un **bouton central** pour activer le scan.
- Deux options activables/désactivables envisagées :
  - **Activation du flash**.
  - **Activation de l'autoscan** (scan en continu sans appui sur le bouton central).
- **Redirection vers la page de résultat lorsqu'un scan est effectué, avec la « valeur » du QR code scannée.**

2.3 Page de Résultat du Scan

- Envoi d'une requête HTTP POST avec :
  - **Token d'authentification dans le header de requête**.
  - **ID de réservation** (« valeur » du scan transmise par la page de scan).
  - **Date/heure du scan**.
- Le serveur devra effectuer les vérifications d'intégrité de la séance, en tenant compte des cas d'erreurs suivants :
  - La réservation existe-t-elle ?
  - Est-elle toujours valide ?
  - La séance est-elle proche du moment du scan ?
  - La réservation a-t-elle déjà été scannée ?
- Le serveur renvoie les informations suivantes si la réservation est valide (ou un message d'erreur adaptée à l'erreur rencontrée) :
  - **Titre du film**.
  - **Numéro de salle (pour pouvoir orienter les clients dans le cinéma)**.
  - **Nombre de places réservées (pour vérifier que le nombre de personnes correspond aux nombre de sièges réservés)**.
- Affichage des informations ou d'un message d'erreur en cas de problème.

3. Spécifications Techniques

3.1. Technologies utilisées

- **Application Mobile** : .NET MAUI (C#)
- **Back-end côté serveur** : Laravel (PHP)
- **Communication Application/Serveur** : API REST
- **Sécurisation de l'API** : Laravel Sanctum
- **Base de données** : MySQL
- **Scan de QR Code** : Bibliothèque .NET MAUI **BarcodeScanning.**

3.2. Sécurité

- Authentification via identifiants et Bearer **Token**.
- Chiffrement du mot de passe avant envoi.
- Contrôle des droits d'accès et/ou intégrité du token sur Laravel.

4. Validation et Tests

- Tests via Postman pour assurer le bon fonctionnement des requêtes API.
- Tests UI sur appareil android pour vérifier l'ergonomie et la fluidité de l'application.

Documentation API

Authentification

URL : https://cinemasolaris.fr/api/login

- **Méthode** : POST
- **Exemple body**:

```
{
        "name" : "username",
        "password" : "passwordCrypte"
```

- **Réponse** :

```
{
        "success" : true,
        "user" : {
                "token" : "authToken",
```

Validation de réservation

URL : https://cinemasolaris.fr/api/reservation/check

- **Méthode** : POST
- **Header requis** :

```
"Authorization" : "Bearer mvAuthToken"
```

- **Exemple body :**

```
{
        "reference" : "QRcodeContent",
```

- **Réponse en cas de succès** :

```
{
    "responseCode": 2000,
    "message": "Réservation Validée",
    "data": {
        "film": {
            "titre": "Nom du film"
        },
        "salle": "Salle 1",
        "places": 2,
```

- **Réponse en cas d'erreur** :

```
{
        "responseCode" : 1XXX
```

Différents codes d'erreurs personnalisés possibles pour chaque cas de figure :

- 1000 : Token invalide ou absent
- 1001 : Réservation introuvable/inexistante
- 1003 : Réservation annulée par le client
- 1005 : Réservation expirée (date et/ou heure de séance dépassé OU réservation déjà validée par l'application)
- 1004 : Réservation scannée trop tôt (même contenu que la réponse de succès, pour estimer quand effectuer une nouvelle tentative).
- 1 : Erreur Serveur (avec message générique)