Cinéphoria

Documentation Technique -

Cinéphoria

Architecture logicielle

Introduction à l'architecture logicielle

L'architecture logicielle de Cinéphoria a pour objectif de fournir une base claire, modulaire et sécurisée pour le développement, la maintenance et l'évolution des différentes applications du projet (web, mobile et bureautique).

Cette organisation logicielle a été conçue pour garantir :

- Une séparation claire des responsabilités entre frontend, backend et bases de données.
- Une maintenabilité accrue, grâce à une organisation modulaire (applications indépendantes avec une bibliothèque partagée).
- Une sécurité renforcée (validation, sanitisation, authentification, autorisation, contrôle d'accès, chiffrement)
- Une extensibilité permettant d'ajouter facilement de nouvelles fonctionnalités ou d'autre plateformes clientes.
- Une portabilité et un déploiement fiables avec Docker.

Le projet repose sur une approche moderne basée sur Angular, Express + Node.js en Typescript. La persistance des données est assurée par une approche polyglotte combinant une base de données relationnelle (MariaDB) et une base de données NoSQL (MongoDB).

Frontend :

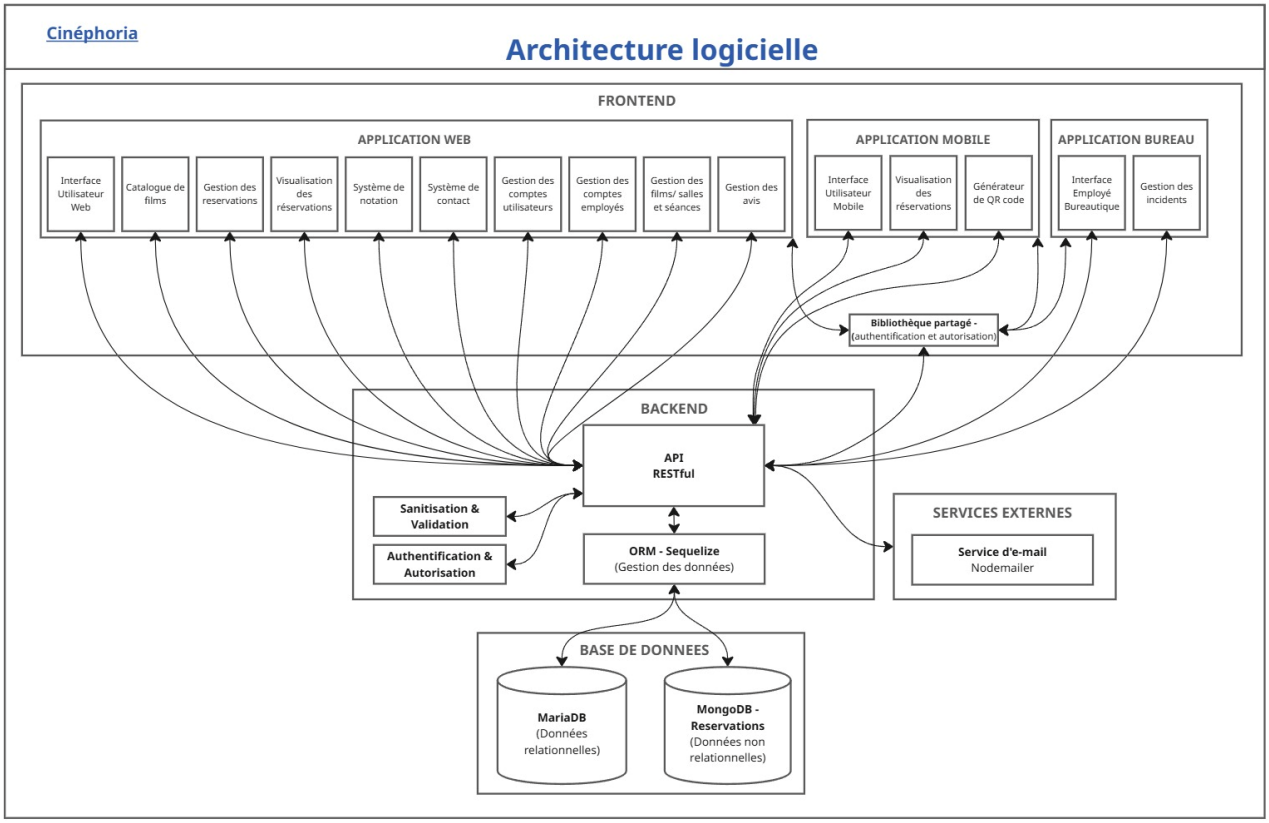
1. Cinéphoria Application Web (CAW) : Angular
2. Cinéphoria Application Mobile (CAM): Angular + Ionic
3. Cinéphoria Application Bureau (CAB) : Angular + Electron
4. Bibliothéque partagée pour les fonctionnalités communes, en particulier l'authentification et l'autorisation : Angular

Choix architecturaux

- Angular monorepo :
  - Permet de centraliser le développement des trois applications et de mutualiser la logique dans une biblithèque partagée.
  - Alternative envisagée : Nx, mais jugé plus complexe et moins nécessaire pour la taille du projet.
- Express et Node.js (Typescript)
  - Simplicité, robustesse et large communauté.
  - Alternative envisagée : NestJs, mais trop lourd pour les besoins actuels.
- Sequelize ORM
  - Facilite la gestion de base de données avec Typescript.
  - Alternative envisagée : Prisma (écartée pour rester sur un ORM plus mature et stable ainsi que mes compétences).

Modélisation de l'architecture

La diagramme d'architecture logicielle ci-dessous illustrent les principaux composants (frontend, backend, bases de données, services externes) et leurs interactions.



Sécurité et conformité

Protection au niveau applicatif :

- *Failles XSS (Cross-Site Scripting)* : utilisation de sanitization et validation coté frontend et backend pour toutes les entrées utilisateurs.
- *Injection SQL :* utilisation de ORM sécurisé (Sequelize) et requêtes paramétrées pour prévenir toute injection SQL.
- *(JWT) Tokens d'authentification* : les Bearer tokens étaient initialement mise en place mais remplacés par HTTPS cookies sécurisé. Verification côté serveur et durée limitée.
- *CRSF* : protection via l'utilisation de cookies SameSite, signé, secure, httpOnly. Validation des requêtes côté serveur.
- *Contrôle d'accès* : mise en place d'une autorisation et authentification RBAC (Role-Based Access Control)
- *Téléversement des fichiers :* limitation des types et tailles de fichiers acceptés.
- *Sécurité HTTP :* utilisation de Helmet pour sécuriser les en-têtes HTTP + Content Security Policy.
- *Chiffrement et hachage :* mots de passe protéges avec bcrypt et salt rounds. Données sensibles chiffrées en transit avec TLS.
- *Auth Guards Angular* et *intercepteurs HTTP* : pour protéger les routes sensibles.
- *Middleware d'authentification :* pour protéger les routes sensibles.

Protection au niveau infrastructure et transport :

- *HTTPS / SSL/TLS :* toutes les communications chiffrées avec certificats Let's Encrypt.
- *Nginx :* utilisé comme reverse-proxy sécurisé pour servir l'application web et appliquer des configurations strictes.
- *Docker Hub / images* : vérification des images pour garantir qu'elles sont à jour et sécurisées.
- *Clés SSH* : utilisées pour les accès sécurisés aux serveurs, sans mots de passe en clair.
- *Configuration générale :* toutes les communications internes et externes passent par HTTPS, aucune transmission de tokens non sécurisés.
- *Utilisateurs non-root + limitation stricte* : Non-root users et contrôle des privilèges.
- *Pare*-feu (UFW) : configuré pour VPS pour restreindre les ports ouverts.
- *Fail2Ban* : installé pour bloquer les tentatives d'intrusion répétées.
- *Audits* :Mise à jour régulière.
- *GitGuardian* : intégré pour détecter les secrets exposés (+.env non commis, .gitignore et .dockerignore pour protégées les données sensitives.

En complément des tests classiques, j'ai intégré des tests de sécurité automatisés :

- SAST (quelque fois avec Sonarcloud) pour analyser le code et détecter les vulnérabilités.
- IAST : exécutés via Trivy dans le workflow CI-predeploy.yml, permettant de détecter les failles lors du déploiement.
- DAST : OWASP ZAP comme prochaine étape pour tester dynamiquement l'application et identifier d'éventuelles vulnérabilités exploitables.

Diagramme d’utilisation

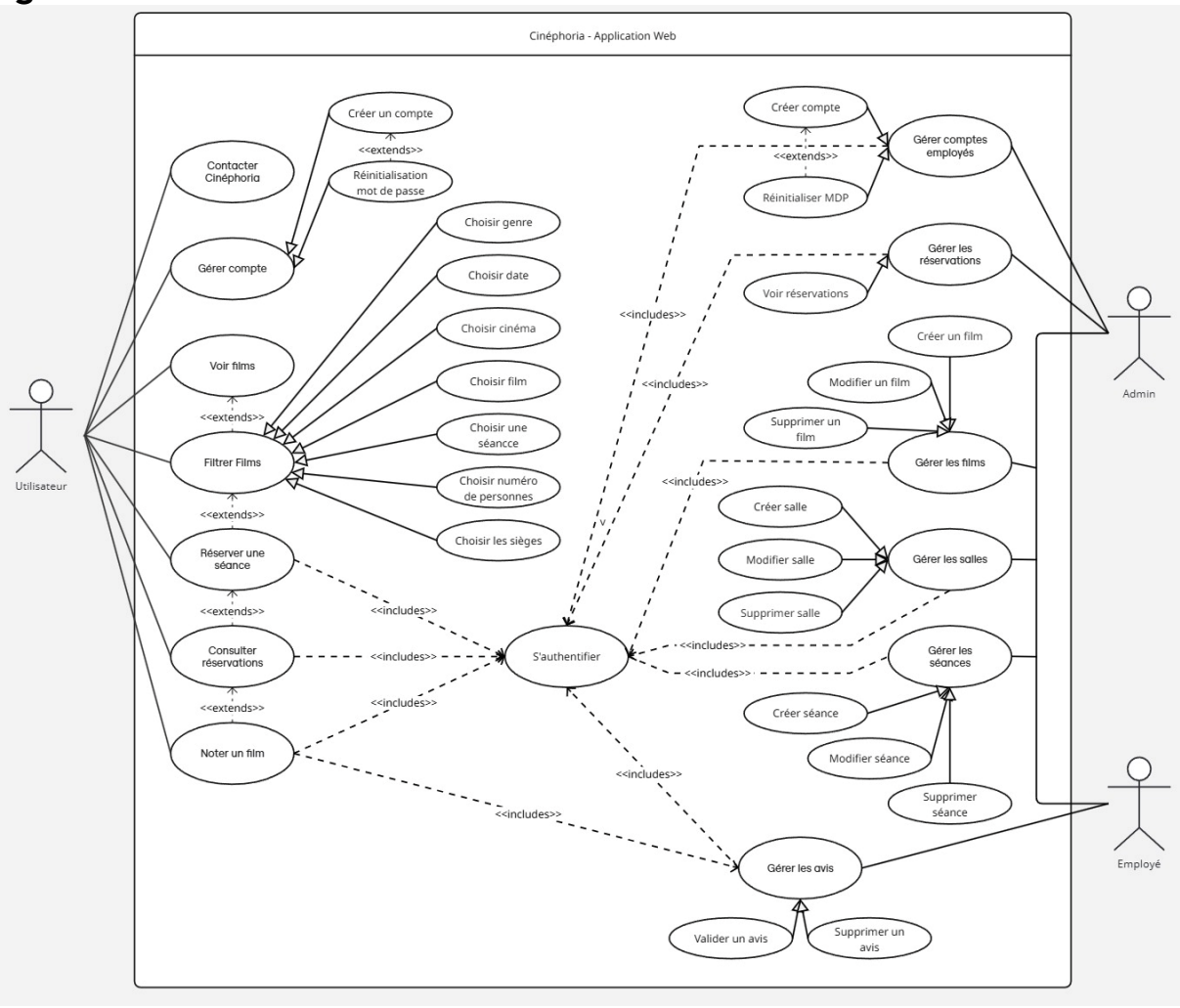
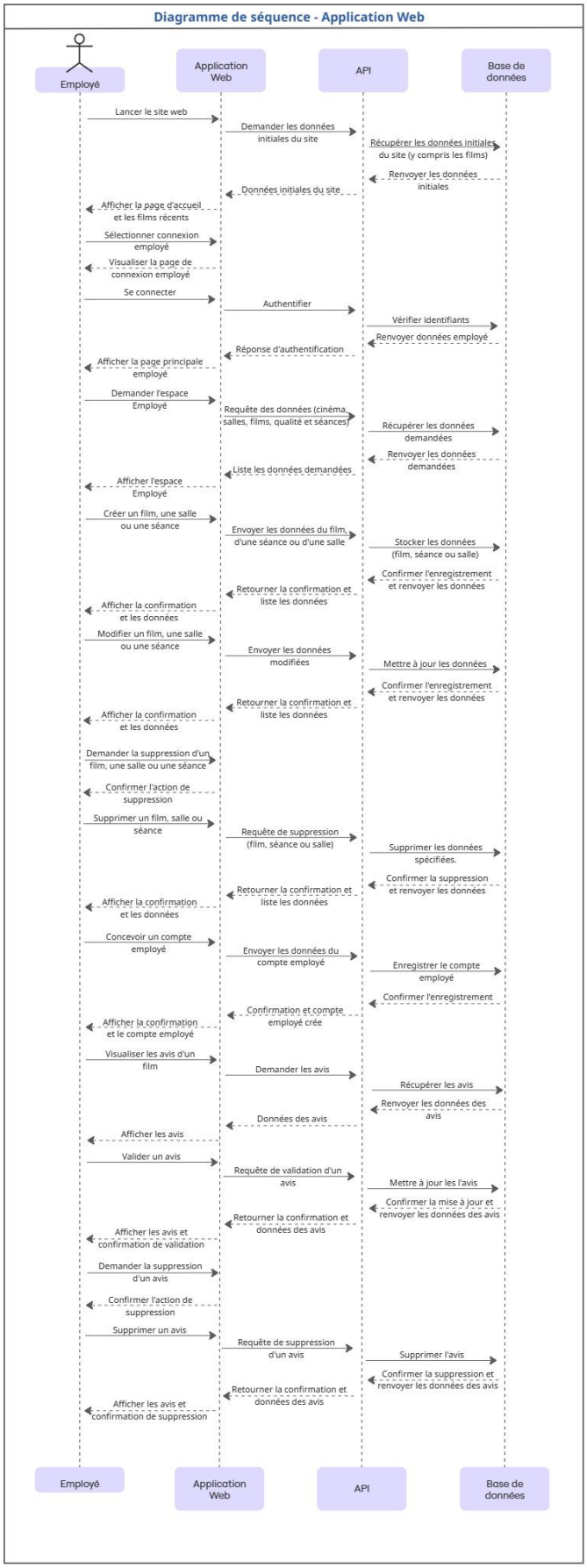
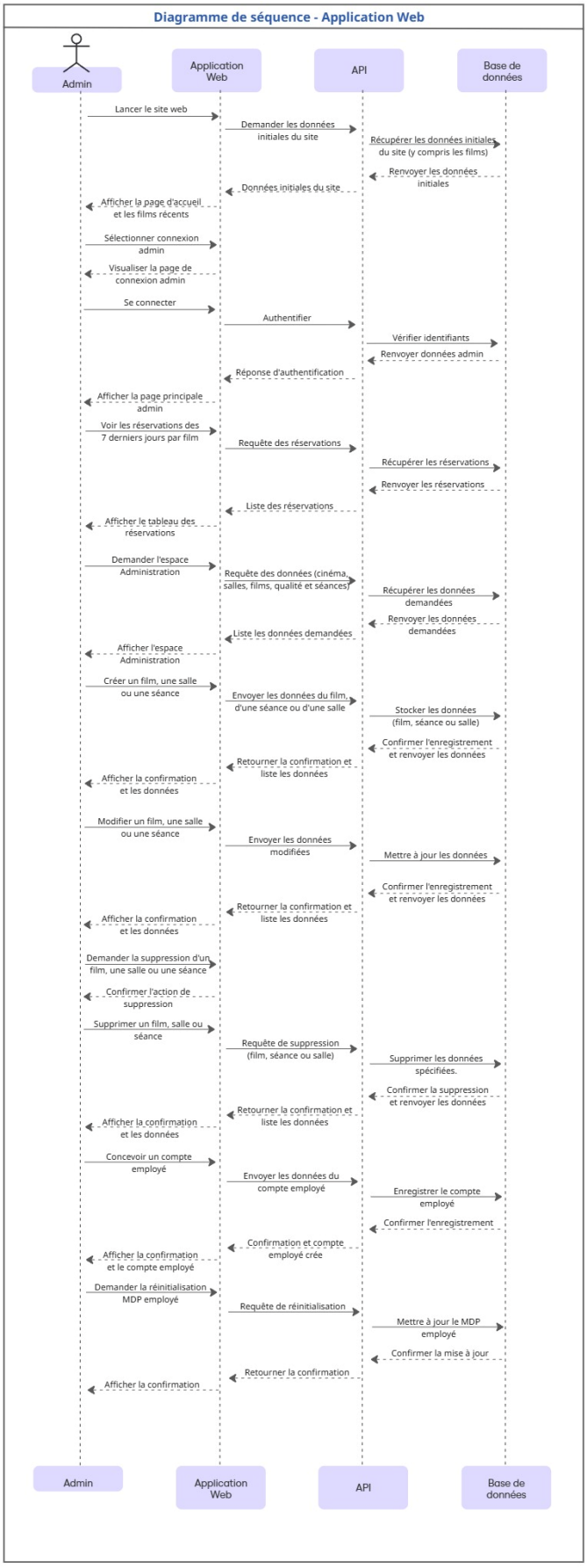


Diagramme de séquence - Employé et Administrateur





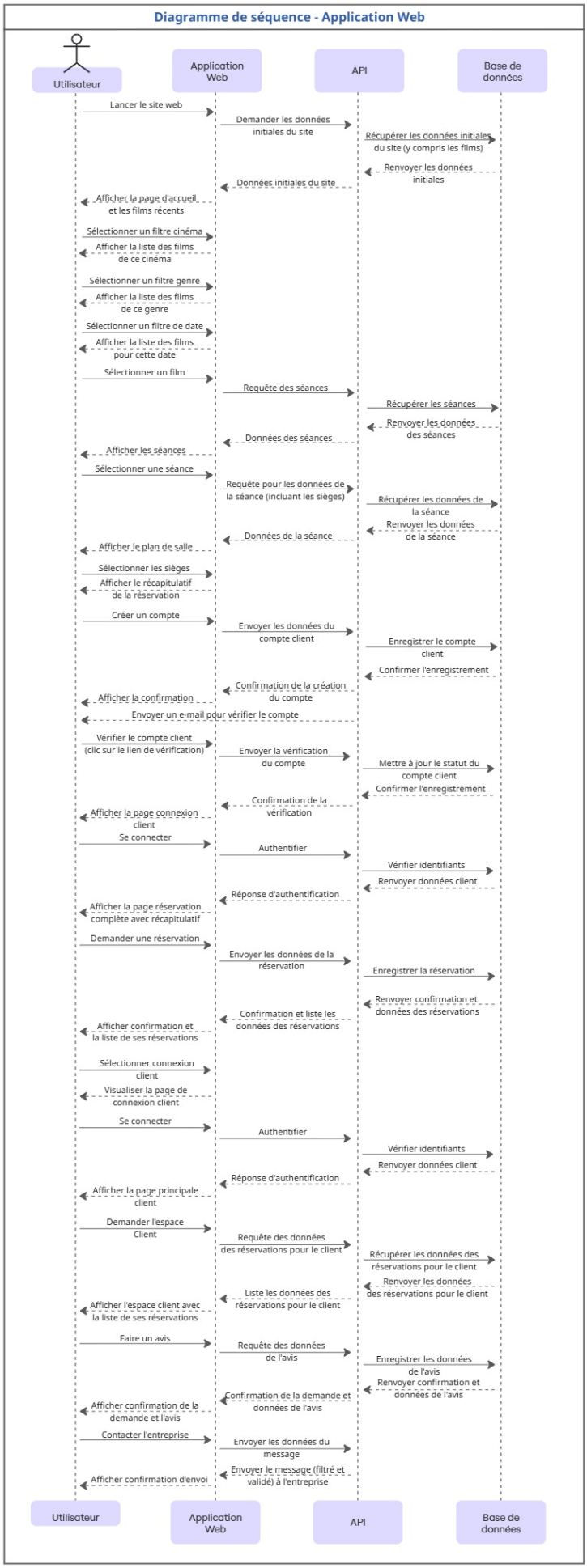
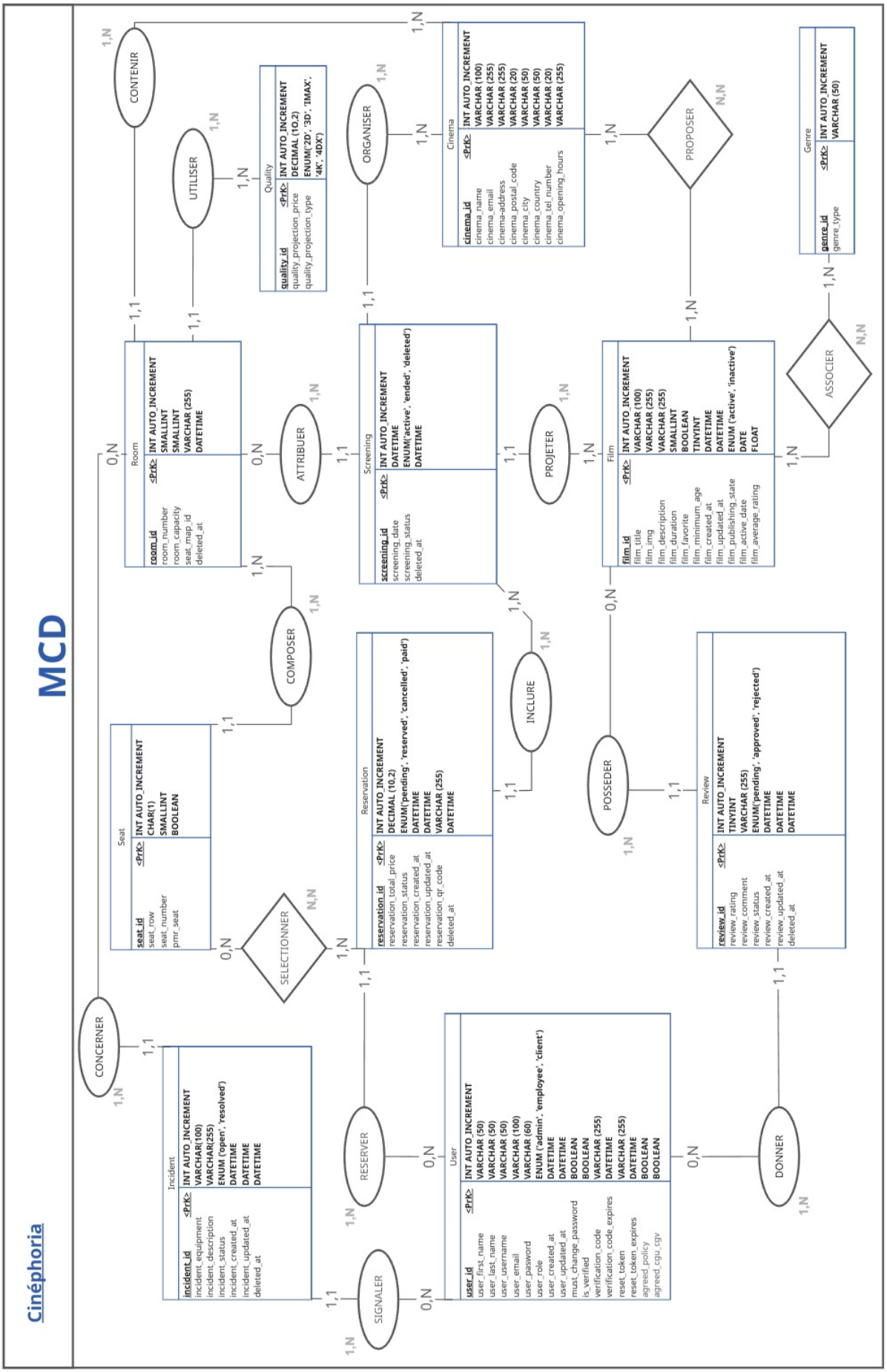
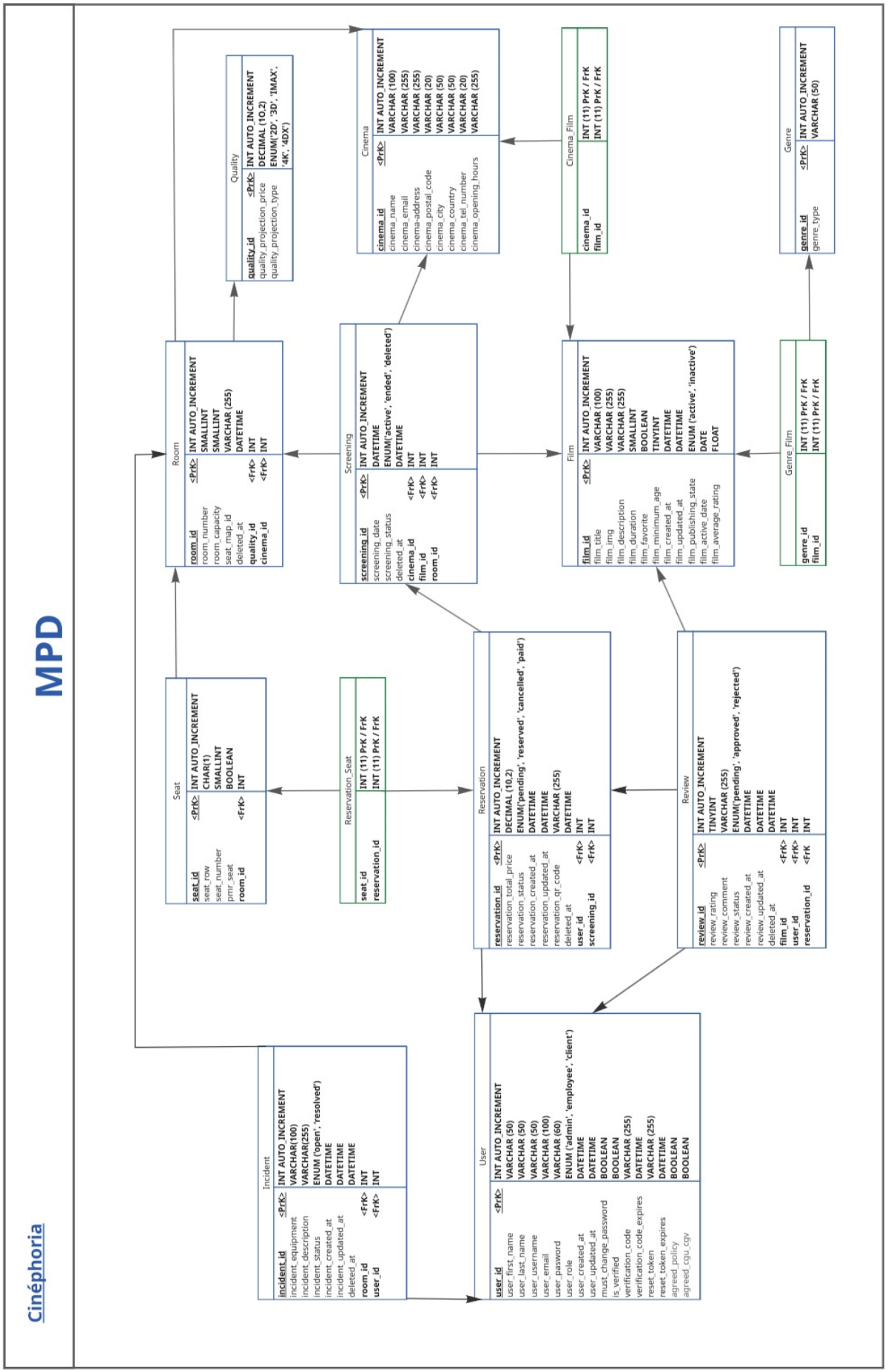


Diagramme de séquence - Application Web
Utilisateur
Application Web
API
Base de données
Lancer le site web
Demander les données initiales du site
Récupérer les données initiales du site (y compris les films)
Renvoyer les données initiales
Données initiales du site
Afficher la page d'accueil et les films récents
Sélectionner un filtre cinéma
Afficher la liste des films de ce cinéma
Sélectionner un filtre genre
Afficher la liste des films de ce genre
Sélectionner un filtre de date
Afficher la liste des films pour cette date
Sélectionner un film
Requête des séances
Récupérer les séances
Renvoyer les données des séances
Données des séances
Afficher les séances
Sélectionner une séance
Requête pour les données de la séance (incluant les sièges)
Récupérer les données de la séance
Renvoyer les données de la séance
Données de la séance
Afficher le plan de salle
Sélectionner les sièges
Afficher le récapitulatif de la réservation
Créer un compte
Envoyer les données du compte client
Enregistrer le compte client
Confirmer l'enregistrement
Confirmation de la création du compte
Afficher la confirmation
Envoyer un e-mail pour vérifier le compte
Vérifier le compte client (clic sur le lien de vérification)
Envoyer la vérification du compte
Mettre à jour le statut du compte client
Confirmer l'enregistrement
Confirmation de la vérification
Afficher la page connexion client
Se connecter
Authentifier
Vérifier identifiants
Renvoyer données client
Réponse d'authentification
Afficher la page réservation complète avec récapitulatif
Demander une réservation
Envoyer les données de la réservation
Enregistrer la réservation
Renvoyer confirmation et données des réservations
Confirmation et liste les données des réservations
Afficher confirmation et la liste de ses réservations
Sélectionner connexion client
Visualiser la page de connexion client
Se connecter
Authentifier
Vérifier identifiants
Renvoyer données client
Réponse d'authentification
Afficher la page principale client
Demander l'espace Client
Requête des données des réservations pour le client
Récupérer les données des réservations pour le client
Renvoyer les données des réservations pour le client
Liste les données des réservations pour le client
Afficher l'espace client avec la liste de ses réservations
Faire un avis
Requête des données de l'avis
Enregistrer les données de l'avis
Renvoyer confirmation et données de l'avis
Confirmation de la demande et données de l'avis
Afficher confirmation de la demande et l'avis
Contacter l'entreprise
Envoyer les données du message
Envoyer le message (filtré et validé) à l'entreprise
Afficher confirmation d'envoi
Utilisateur
Application Web
API
Base de données

Modèle conceptuel de données (MCD)



Modèle Physique de Données (MPD)

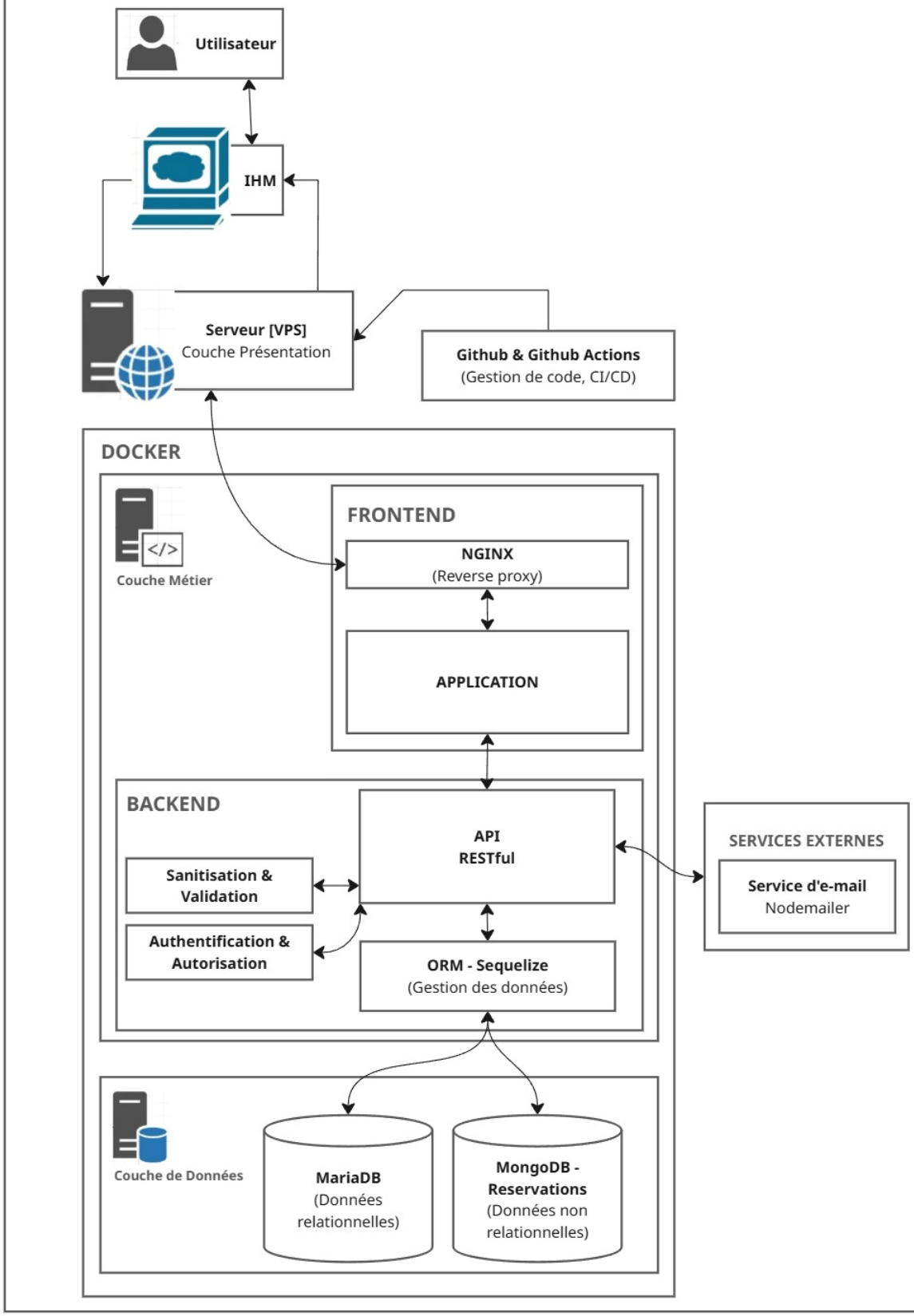


Architecture globale

L'application web Cinéphoria est déployée sur un VPS OVHCloud et orchestrée par Docker Compose pour garantir une isolation, une portabilité et une reproductibilité optimales des différents environnements. L'architecture suit un modèle multicouches avec séparation claire entre le frontend, le backend et les bases de données.

- **Docker et Docker Compose :** assurent l'isolation des services et facilitent le déploiement et la maintenance des environnements.
- **Nginx comme reverse proxy :** gestion des certificats SSL/TLS, compression gzip, en-têtes de sécurité et limitation de débit pour protéger l'application.
- **Security by design :**
  - Frontend Angular : Sanitisation et validation des données, intercepter, initialiser et guard pour l'authentification et l'autorisation.
  - Backend : Sanitisation et validation des données, CORS configuré, Helmet pour les headers de sécurité, parsers limités, middleware d'authentification, hachage sécurisé des mots des passe avec bcrypt (salt rounds et expiration), cookies HTTPS pour les tokens.
  - Bases de données : gestion stricte des utilisateurs, limitation des privilèges/permissions, désactivation des comptes par défaut (ex. root pour MarriaDB).
- **Persistance Polygotte :** MariaDB est utilisée comme base principale et MongoDB pour un usage complémentaire avec une collection pour les statistiques des réservations.
- **Services externes :** Nodemailer pour la communications avec les utilisateurs.
- **Intégration Continue et Déploiement Continue (CI/CD) :**
  - Intégration continue : workflow sécurisé avec les secrets stockés dans GitHub Actions pour la vérification du code via linting, tests et scans de sécurité pour détecter l'exposition des secrets (GitGuardian).
  - Déploiement continue : après succès de l'intégration sur la branche main, le déploiement vers le VPS OVHCloud se fait automatiquement via SSH sécurisé.

Architecture globale



Stratégie de Tests et CI/CD

Tests

Pour le projet Cinéphoria, différents types de tests ont été mis en place afin de garantir la qualité, la fiabilité et la sécurité de l'application :

- Tests unitaires : ils vérifient le comportement isolé des composants et fonctions, tant pour le frontend que pour le backend.
- Tests d'intégration : ils assurent que les différents modules de l'application communiquent correctement entre eux.
- Tests fonctionnels : ils simulent des scénarios utilisateurs pour vérifier que les fonctionnalités principales répondent aux besoins spécifiés dans les User Stories.
- Tests end-to-end (E2E) avec Cypress : ils reproduisent le parcours complet d'un utilisateur, du front au back, pour garantir que l'application fonctionne correctement dans son ensemble.

Ces tests peuvent être exécutés localement ou via Docker, avec des variables d'environnement adaptées (DOCKER-DB LOCAL DEV pour local, DOCKER-TEST pour Docker). Des scripts automatisés permettent de lancer tous les tests :

- Localement : en suivant les instructions d'installation, en insérant les utilisateurs de test via npm run backend:seeder, puis en exécutant npm run test:all pour l'ensemble des tests ou des commandes spécifiques pour unitaires, intégration ou fonctionnels.
- Avec Docker : en utilisant docker-compose-test pour créer un environnement isolé, puis en exécutant les tests de la même manière à l'intérieur des conteneurs. Les tests E2E peuvent également être lancés en mode headless pour automatisation.
- CI/CD : les workflows Github Actions intègrent docker-compose-test afin de tester automatiquement l'application à chaque push ou pull request, garantissant cohérence, reproductibilité et fiabilité.

Pour plus de détails sur la configuration et l'exécution des tests, les commandes et instructions complètes sont disponibles dans le fichier README.md du projet.

Intégration continue/Déploiement continu (CI/CD)

Le déploiement est automatisé grâce à une approche **CI/CD avec GitHub Actions** :

- **CI** : chaque push sur une branche feature ou development déclenche un pipeline qui exécute les tests, l'analyse de code (linting, sécurité, dépendances) et vérifie que le projet est stable.
- **CD** : lors d'un merge sur main, un workflow de déploiement s'exécute automatiquement après un workflow qui exécute les tests. Les fichiers sont copiés sur le VPS via SSH sécurisé, puis déployés avec Docker Compose et servis par Nginx.
- Les secrets (clés SSH, variables d'environnement, certificats TLS/SSL via Let's Encrypt) sont gérés de façon sécurisée et ne sont jamais exposés dans le code.

Application Mobile

Diagramme d'utilisation

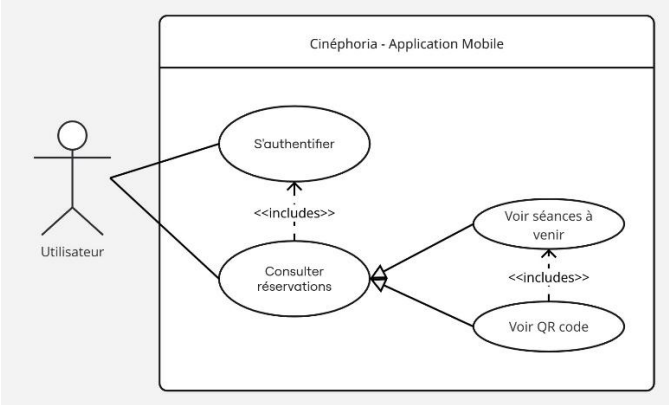
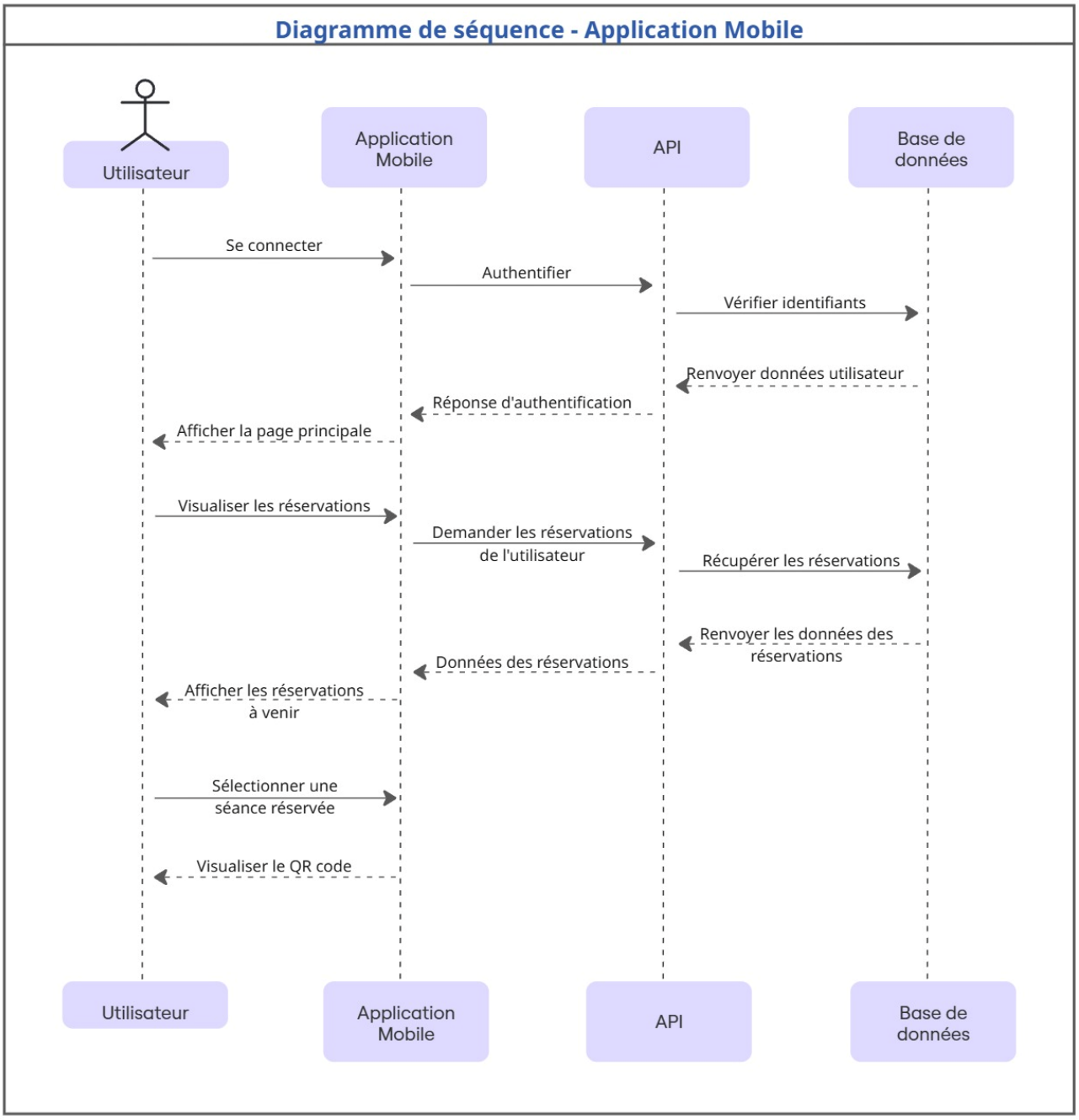


Diagramme de séquence



Application Bureautique

Diagramme d'utilisation

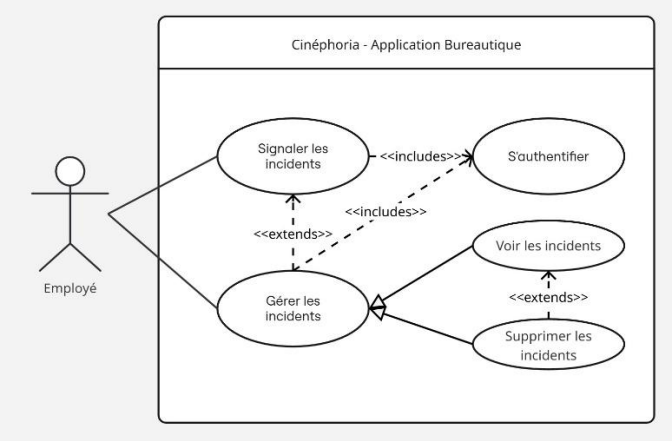
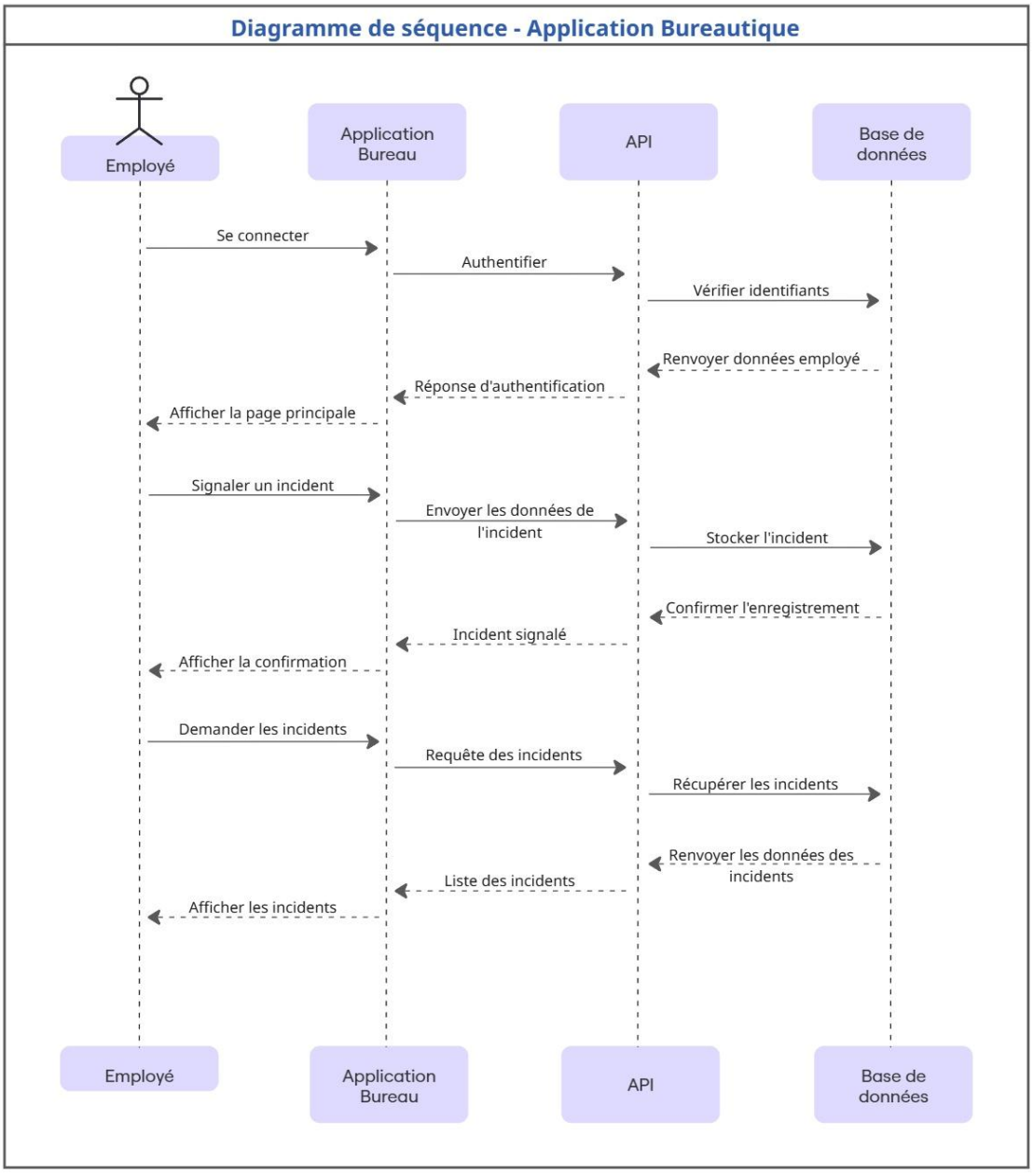


Diagramme de séquence



Technologies utilisés

Application Web

- *Gestion des versions :* Git, GitHub, GitHub Actions (CI/CD).
- *Environnement de développement (IDE) :* Visual Studio Code (VSCode) avec extensions adaptées (Angular, Docker, ESLint).
- *Node.js avec NPM* comme gestionnaire de paquets pour le frontend et le backend.
- *Express.js (*avec express-rate-limit) pour la création d'API RESTful sécurisées et performantes.
- *TypeScript* pour un typage strict et une meilleure maintenabilité.
- *Angular v18* **LTS** avec *Angular CLI* pour le frontend, offrant un framework robuste et structuré.
- *Angular Material* (customisé en SCSS) pour une UI moderne et responsive.
- *ESLint et Prettier* pour l'analyse statique et la qualité du code.
- *Docker & Docker Compose* pour la conteneurisation et la gestion d'environnements isolés.
- *MariaDB* pour la gestion des données relationnelles (SQL). [MySQL workspace local]
- *MongoDB* pour la gestion des données NoSQL. [MongoDB Compass]
- *Mongoose* pour l'abstraction et la validation des données NoSQL.
- *Sequelize (ORM)* pour sécuriser les accès SQL et simplifier les requêtes.
- *Cypress et Jest* pour les tests automatisés (unitaires, intégration et e2e).
- *Nginx* en reverse proxy sécurisé, avec SSL/TLS via Let's Encrypt.

Application Mobile

- *PWA avec Ionic Angular*, pour permettre un déploiement cross-platform léger.

Application Desktop

- *Electron.js avec Angular,* permettant de packager l'application en version desktop multi-OS.

Pour développer Cinéphoria, j'ai choisi un écosystème moderne et cohérent : Angular v18 LTS et TypeScript pour un frontend robuste et maintenable, Node.js avec Express.js pour un backend léger et performant, ainsi que MariaDB et MongoDB pour gérer à la fois les données relationnelles et non structurées. La conteneurisation avec Docker assure non seulement un déploiement fiable et sécurisé, mais aussi une compatibilité étendue sur différents environnements. Le reverse-proxy Nginx sécurisé avec SSL/TLS garantissent un déploiement fiable et des communications chiffrées. Les outils de qualité et de tests comme ESLint, Prettier, Jest et Cypress, ainsi que GitHub Actions et GitGuardian, assurent la fiabilité et la sécurité de l'application. Ionic et Electron permettent d'étendre l'expérience sur mobile et desktop sans compromis. Ces choix technologiques ont été retenus pour répondre efficacement aux besoins fonctionnels et sécuritaires du projet, tout en facilitant la maintenabilité, la cohérence et la robustesse du système global.

- **Configuration matérielle :**
  - Ordinateur portable (Windows 11) avec l'utilisation des bureaux et un écran externe afin d'améliorer productivité, organisation et une meilleure visualisation du code.

Gestion de projet avec Trello et méthode Agile

Pour le projet Cinéphoria, j'ai utilisé la méthode Agile Scrum avec une organisation Kanban via Trello. Cette approche m'a permis de gérer efficacement le développement, de prioriser les fonctionnalités importantes et de suivre facilement l'avancement du projet.

Chaque User story (avec ID unique) est définie avec la technique des 3 W's (Who, What, Why), une difficulté Fibonacci, des priorités et des critères d'acceptation clairs. Cela m'a permis également de structurer le travail de manière transparente et de m'assurer que chaque fonctionnalité correspond aux besoins des utilisateurs finaux (tout en facilitant la traçabilité et la qualité du code).

L'utilisation de Trello a apporté un avantage concret : je pouvais visualiser rapidement l'état de chaque tâche grâce aux colonnes (Backlog, prochain sprint, sprint backlog, en cours, révision, test, terminé, mergé, déployé).

Elle a également offert la flexibilité nécessaire pour s'adapter lorsque certaines tâches prenaient plus de temps que prévu, tout en restant essentielles pour garantir la sécurité de l'application et la qualité du code. Une méthode particulièrement pratique pour une progression constante, organisation clair et amélioration continue.

Transaction SQL

Une transaction SQL permet de regrouper plusieurs requêtes comme une unité : toutes les opérations sont exécutées avec succès ou aucune ne l'est en cas d'erreur.

Avantages principaux:

- **Intégrité des données :** évite l'insertion partielle ou incohérente.
- **Fiabilité et cohérence :** garantit la synchronisation des relations entre tables.

Cette transaction est un exemple d'ajout d'un nouveau film, associer ses genres et programmer ses séances, tout en s'assurant qu'aucune étape ne soit appliquée en cas de problème.

Description de la procédure :

La procédure stockée transaction_example_add_new_film a pour objectif d'ajouter un nouveau film dans la base de données SQL.

La procédure commence par vérifier si un film portant le même titre existe déjà. Si c'est le cas, la transaction est annulée afin d'éviter tout doublon. Cette étape garantit la cohérence et l'intégrité des données.

Si le film n'existe pas, ses informations principales (titre, description, image, durée, indicateur "coup de cœur", âge minimum et date d'activation) sont insérées dans la table Film. L'identifiant généré est récupéré pour créer les relations avec les autres tables.

Les genres correspondant au film sont liés à l'aide de la table Genre_Film. Cette étape garantit que le film sera correctement catégorisé pour les filtres et l'affichage dans l'application.

Les séances sont planifiées pour chaque salle des cinémas concernés, avec un statut actif par défaut.

Deux vues sont générées pour faciliter la consultation des données : l'une regroupant les détails du film et ses genres, l'autre listant les séances avec le cinéma et la salle associée. Ces vues simplifient l'affichage et la récupération des informations sans modifier les données d'origine. Elles peuvent être recréées ou mises à jour à chaque ajout de film

```sql
-- • La base de données à utiliser pour la transaction est définie.
USE cinephoriasqldb;

-- • La procédure est supprimée si elle existe déjà, afin de permettre une création propre.
DROP PROCEDURE IF EXISTS transaction_example_add_new_film;

-- • Un délimiteur est utilisé pour encadrer la définition de la procédure stockée.
DELIMITER //

-- • Une procédure stockée est créée pour ajouter un nouveau film.
CREATE PROCEDURE transaction_example_add_new_film(
    IN p_film_title VARCHAR(100),
    IN p_film_description VARCHAR(255),
    IN p_film_img VARCHAR(255),
    IN p_film_duration SMALLINT,
    IN p_film_favorite BOOLEAN,
    IN p_film_minimum_age TINYINT,
    IN p_film_active_date DATE
)

BEGIN
proc_transaction_example_add_new_film: BEGIN
    DECLARE film_count INT;
    DECLARE genre_count INT;
    DECLARE screening_count INT;
    DECLARE new_film_id INT;
    -- • Des variables sont déclarées pour compter les films, genres et séances, et pour stocker l'identifiant du nouveau film.
    START TRANSACTION;

    -- • Avant l'insertion, la procédure vérifie si un film portant le même titre existe. Si c'est le cas,
    --   la transaction est annulée et la procédure est quittée avec ROLLBACK.
    IF EXISTS (SELECT 1 FROM Film WHERE film_title = p_film_title) THEN
        ROLLBACK;
        SELECT 'Transaction rollback - Film already exists' AS result;
        LEAVE proc_transaction_example_add_new_film;
    END IF;

    -- • Les informations du film sont ensuite définies à partir des paramètres fournis.
    SET @film_title = p_film_title;
    SET @film_description = p_film_description;
    SET @film_img = p_film_img;
    SET @film_duration = p_film_duration;
    SET @film_favorite = p_film_favorite;
    SET @film_minimum_age = p_film_minimum_age;
    SET @film_active_date = p_film_active_date;

    -- • Le nouveau film est inséré avec le statut actif par défaut et sans notes initiales.
    INSERT INTO Film (film_title, film_description, film_img, film_duration, film_favorite, film_minimum_age, film_active_date, film_publishing_state, film_average_rating)
    VALUES (@film_title, @film_description, @film_img, @film_duration, @film_favorite, @film_minimum_age, @film_active_date, 'active', 0);
    -- • L'identifiant du dernier film inséré est récupéré pour les relations suivantes.
    SET @new_film_id = LAST_INSERT_ID();

    -- • Les genres sont associés au film en fonction des catégories existantes.
    INSERT INTO Genre_Film (genre_id, film_id)
    SELECT genre_id, @new_film_id
    FROM Genre
    WHERE genre_type IN ('Fantastique', 'Aventure'); -- • Supposer que ces genres existent (modifier en conséquence)

    -- • Les séances sont programmées pour le nouveau film dans les salles disponibles.
    INSERT INTO Screening (screening_date, screening_status, cinema_id, film_id, room_id)
    SELECT
        DATE_ADD(NOW(), INTERVAL 1 DAY) + INTERVAL (ROW_NUMBER() OVER (PARTITION BY Cinema.cinema_id ORDER BY Room.room_id) - 1) DAY AS screening_date,
        'active' AS screening_status,
        Cinema.cinema_id,
        @new_film_id,
        Room.room_id
    FROM Cinema
    JOIN Room ON Cinema.cinema_id = Room.cinema_id
    LIMIT 10; -- • Limiter à 10 projections pour la démonstration

    -- • Une vue est créée pour les détails du nouveau film.
    CREATE OR REPLACE VIEW v_new_film_details AS
    SELECT f.film_id, f.film_title, GROUP_CONCAT(g.genre_type) AS genres
    FROM Film f
    LEFT JOIN Genre_Film gf ON f.film_id = gf.film_id
    LEFT JOIN Genre g ON gf.genre_id = g.genre_id
    GROUP BY f.film_id;
    -- • Le résultat de cette vue peut être interrogé avec SELECT * FROM v_new_film_details;

    -- • Une vue est créée pour les projections du nouveau film.
    CREATE OR REPLACE VIEW v_new_film_screenings AS
    SELECT f.film_id, f.film_title, s.screening_id, s.screening_date, c.cinema_name, r.room_number
    FROM Film f
    LEFT JOIN Screening s ON f.film_id = s.film_id
    LEFT JOIN Cinema c ON s.cinema_id = c.cinema_id
    LEFT JOIN Room r ON s.room_id = r.room_id;
    -- • Le résultat de cette vue peut être interrogé avec SELECT * FROM v_new_film_screenings;

    -- • Les vues peuvent être consultées via des requêtes simples ou pour supprimer, décommenter les 2 lignes suivantes
    -- DROP VIEW IF EXISTS v_new_film_details;
    -- DROP VIEW IF EXISTS v_new_film_screenings;

    -- • Des compteurs sont utilisés pour valider que toutes les étapes de la transaction ont réussi.
    SET @film_count = (SELECT COUNT(*) FROM Film WHERE film_id = @new_film_id);
    SET @genre_count = (SELECT COUNT(*) FROM Genre_Film WHERE film_id = @new_film_id);
    SET @screening_count = (SELECT COUNT(*) FROM Screening WHERE film_id = @new_film_id);

    -- • La transaction est validée si toutes les opérations sont réussies, sinon elle est annulée avec ROLLBACK.
    IF @film_count = 1 AND @genre_count > 0 AND @screening_count > 0 THEN
        COMMIT;
        SELECT 'Transaction completed - New film and screenings successfully added' AS result;
    ELSE
        ROLLBACK;
        SELECT 'Transaction failed - The new film and screenings were not added' AS result;
    END IF;

    END proc_transaction_example_add_new_film;
    -- • Les variables temporaires sont nettoyées à la fin de la procédure.
    SET @film_title = NULL;
    SET @film_description = NULL;
    SET @film_img = NULL;
    SET @film_duration = NULL;
    SET @film_favorite = NULL;
    SET @film_minimum_age = NULL;
    SET @film_active_date = NULL;
END //

DELIMITER ;
```

```sql
-- • La procédure et transaction peut être appelée indépendamment avec différents paramètres pour ajouter d'autres films.
-- • Exemple d'appel de la procédure avec des détails de film spécifiques.
CALL transaction_example_add_new_film(
    'La Légende de la mers',
    'Une aventure épique au fond des océans pour toute la famille.',
    'assets/img/la_legende_de_la_mer.webp',
    95,
    FALSE,
    0,
    '2025-08-27'
);
```

Les étapes

- La base de données à utiliser pour la transaction est définie.
- La procédure est supprimée si elle existe déjà, afin de permettre une création propre.
- Un délimiteur est utilisé pour encadrer la définition de la procédure stockée.
- Une procédure stockée est créée pour ajouter un nouveau film.
- La transaction regroupe l'ajout du film, l'association de ses genres et la programmation de ses séances.
- Des variables sont déclarées pour compter les films, genres et séances, et pour stocker l'identifiant du nouveau film.
- Avant l'insertion, la procédure vérifie si un film portant le même titre existe. Si c'est le cas, la transaction est annulée et la procédure est quittée.
- Les informations du film sont ensuite définies à partir des paramètres fournis.
- Le nouveau film est inséré avec le statut actif par défaut et sans notes initiales.
- L'identifiant du dernier film inséré est récupéré pour les relations suivantes.
- Les genres sont associés au film en fonction des catégories existantes.
- Les séances sont programmées pour le nouveau film dans les salles disponibles.
- Les vues sont créées pour faciliter la consultation des détails du film et de ses séances.
- Les vues peuvent être consultées via des requêtes simples ou supprimées si nécessaire.
- Des compteurs sont utilisés pour valider que toutes les étapes de la transaction ont réussi.
- La transaction est validée si toutes les opérations sont réussies, sinon elle est annulée.
- Les variables temporaires sont nettoyées à la fin de la procédure.
- La procédure et transaction peut être appelée indépendamment avec différents paramètres pour ajouter d'autres films.