# **Documentation Technique**

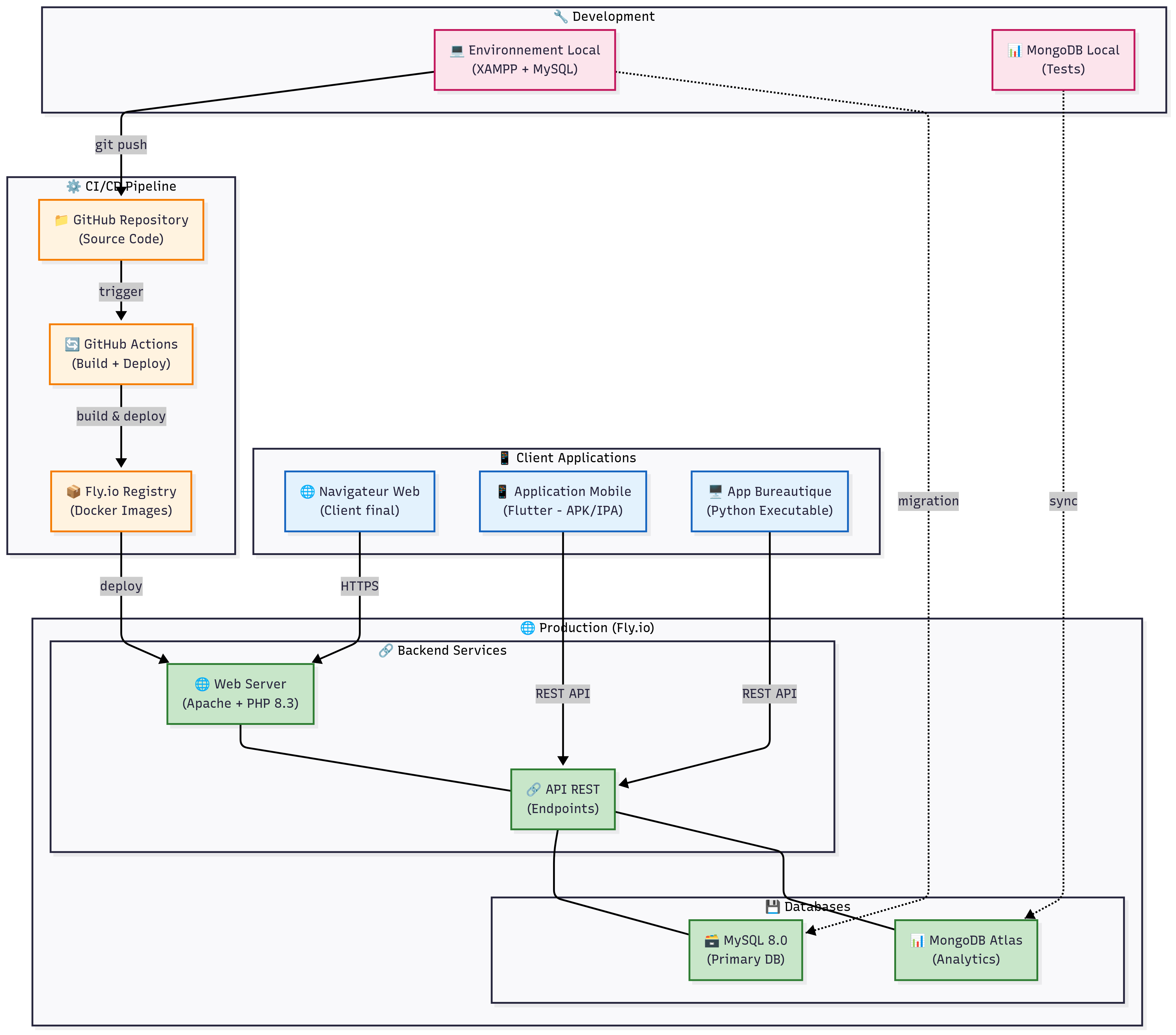
## **Projet Cinephoria CDA**

**Étudiant :** Romain Rémusat  
**Formation :** Concepteur Développeur d'Applications  
**Date :** Juillet 2025

## **Architecture logicielle**

### **Vue d'ensemble du système**

Cinephoria adopte une **architecture distribuée** basée sur le pattern **API-First** permettant l'interopérabilité entre 3 applications distinctes partageant un backend commun.



### **Justification de l'architecture**

**API REST centralisée :** Assure la cohérence des données entre les 3 applications et facilite la maintenance en centralisant la logique métier.

**Séparation des responsabilités :** Chaque application a sa spécificité (web pour grand public, mobile pour praticité, bureautique pour employés) tout en partageant les mêmes données.

**Base de données hybride :** MySQL pour les données transactionnelles critiques (réservations) et MongoDB pour les analytics flexibles (dashboard admin).

**Scalabilité :** Architecture permettant l'ajout futur d'autres clients (API publique, applications tierces) sans modification du backend.

## **Choix technologiques**

### **Stack Application Web**

#### **Backend : PHP 8.3**

**Justification :**

* **Maturité éprouvée** pour les applications web de réservation
* **Écosystème riche** avec PDO pour la sécurité des requêtes
* **Performance** adaptée au volume de trafic attendu
* **Facilité de déploiement** sur hébergeurs standards
* **Compatibilité native** avec MySQL

**Avantages spécifiques :**

* Gestion native des sessions pour l'authentification
* Support excellent des transactions SQL
* Bibliothèques de sécurisation (password\_hash, filter\_var)
* Documentation complète et communauté active

#### **Frontend : Bootstrap 5 + JavaScript ES6**

**Justification :**

* **Responsive design** automatique pour mobile-first
* **Rapidité de développement** avec composants pré-construits
* **Cohérence visuelle** garantie sur tous navigateurs
* **Personnalisation** possible via CSS custom

**Alternative écartée :** Framework JS (React/Vue) jugé trop lourd pour un projet de cette envergure.

#### **Base de données : MySQL 8.0**

**Justification :**

* **ACID compliance** essentielle pour les réservations
* **Transactions robustes** pour éviter les sur-réservations
* **Performance** excellente pour les requêtes relationnelles
* **Triggers et procédures** stockées pour la logique métier critique
* **Écosystème mature** avec outils d'administration

**Fonctionnalités utilisées :**

* Contraintes de clés étrangères pour l'intégrité
* Index composites pour optimiser les requêtes fréquentes
* Triggers pour automatiser les mises à jour (places, notes)
* Procédures stockées pour les transactions complexes

### **Stack Application Mobile**

#### **Framework : Flutter 3.16**

**Justification :**

* **Cross-platform** : Une seule codebase pour iOS et Android
* **Performance native** grâce à la compilation AOT
* **Écosystème riche** pour les fonctionnalités requises (QR codes)
* **Hot reload** pour accélérer le développement

**Alternative considérée :** Application web responsive, finalement retenue comme fallback pour simplifier le déploiement.

### **Stack Application Bureautique**

#### **Langage : Python 3.11 + Tkinter**

**Justification :**

* **Simplicité** d'apprentissage et de développement
* **Tkinter inclus** nativement, pas de dépendances externes
* **Intégration API** facile avec requests
* **Déploiement simple** avec PyInstaller

**Alternative écartée :** Electron jugé trop lourd pour une application simple de saisie d'incidents.

### **Déploiement : Fly.io**

**Justification :**

* **Support natif PHP** avec configuration simple
* **Base de données intégrée** MySQL disponible
* **SSL automatique** et CDN inclus
* **Scaling automatique** selon la charge
* **Prix compétitif** pour un projet étudiant

**Avantages :**

* Déploiement git-based simplifié
* Monitoring intégré
* Backup automatique des données
* Support multi-régions

## **Configuration environnement**

### **Environnement de développement**

#### **Prérequis système :**

# Serveur local

XAMPP 8.1+ ou WAMP

PHP 8.3+

MySQL 8.0+

Git 2.30+

# Mobile (optionnel)

Flutter SDK 3.16+

Android Studio / Xcode

# Bureautique

Python 3.11+

Tkinter (inclus)

#### **Configuration PHP :**

# php.ini

memory\_limit = 256M

max\_execution\_time = 300

upload\_max\_filesize = 10M

post\_max\_size = 10M

display\_errors = Off (production)

log\_errors = On

#### **Variables d'environnement :**

# .env

DB\_HOST=localhost

DB\_NAME=cinephoria

DB\_USER=root

DB\_PASS=

MONGODB\_URI=mongodb://localhost:27017/cinephoria\_stats

API\_BASE\_URL=http://localhost:8000/api

JWT\_SECRET=your\_secret\_key\_here

### **Configuration base de données**

#### **MySQL - Structure optimisée :**

-- Configuration InnoDB pour ACID

SET innodb\_flush\_log\_at\_trx\_commit = 1;

SET innodb\_file\_per\_table = ON;

-- Index composites pour performance

CREATE INDEX idx\_seances\_date\_film ON seances(date\_seance, film\_id);

CREATE INDEX idx\_reservations\_user\_date ON reservations(utilisateur\_id, date\_reservation);

#### **MongoDB - Collections :**

// Collection statistiques\_quotidiennes

{

"\_id": ObjectId("..."),

"date": ISODate("2025-12-15"),

"reservations": {

"total": 45,

"confirmees": 38,

"annulees": 7

},

"chiffre\_affaires": {

"total": 456.50,

"par\_salle": {

"Salle 1": 120.00,

"Salle IMAX": 180.50

}

}

}

## **Modèle Conceptuel de Données (MCD)**

### **Entités principales**

Le MCD de Cinephoria illustre la structure relationnelle optimisée pour gérer une chaîne de cinémas. Les entités principales forment le cœur métier :

**UTILISATEURS** (clients, employés, admins)

* Gestion des rôles avec ENUM('utilisateur', 'employe', 'admin')
* Email unique pour authentification
* Champs optionnels (téléphone, date\_naissance) pour personnalisation

**FILMS** (catalogue cinéma)

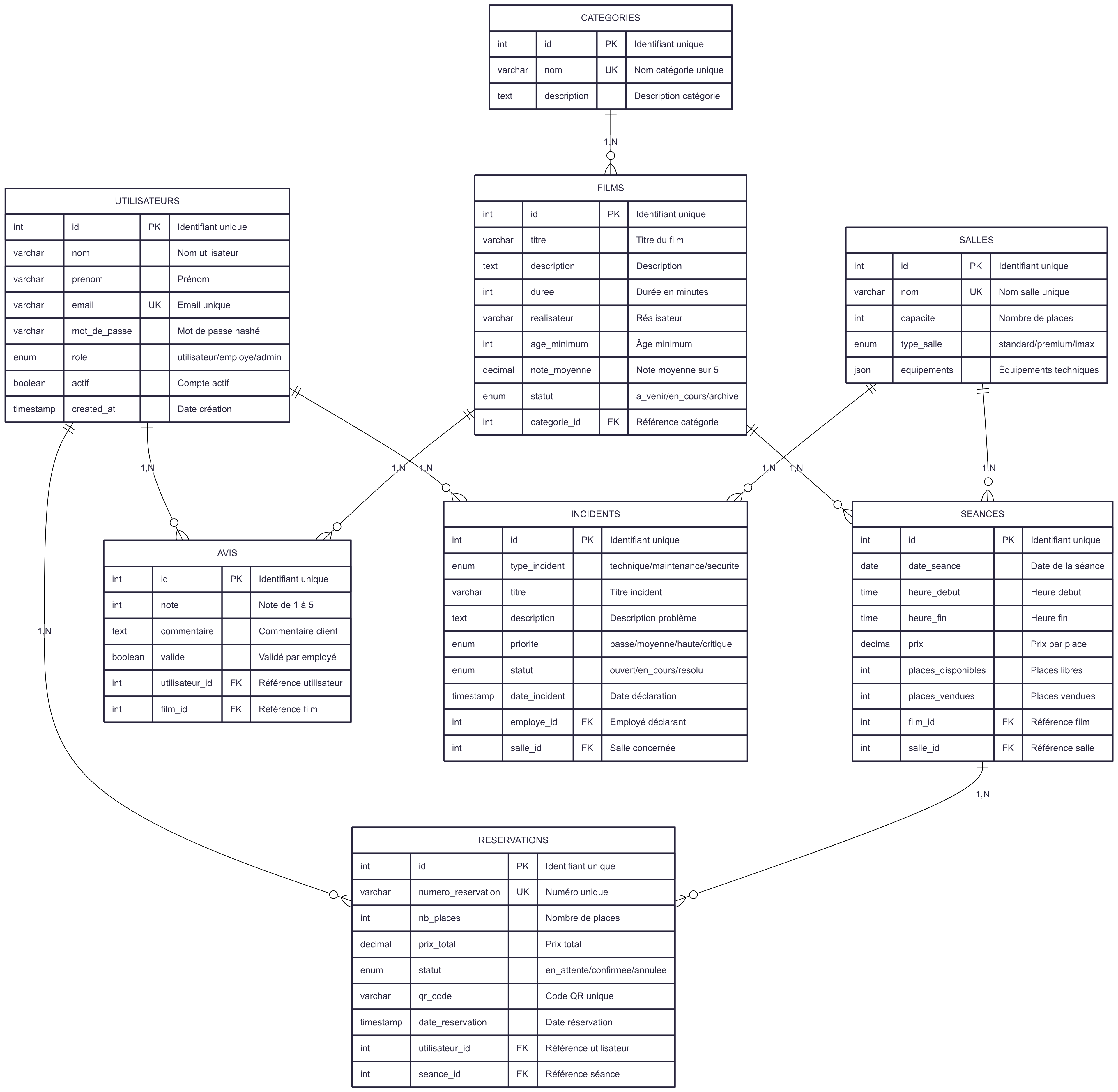
* Relation avec CATEGORIES (1,N)
* Note moyenne calculée automatiquement via triggers
* Statut géré par ENUM pour le cycle de vie

**SEANCES** (programmation)

* Contrainte unique (salle\_id, date\_seance, heure\_debut) évite les chevauchements
* Places disponibles/vendues synchronisées par triggers
* Relation avec FILMS et SALLES

**RESERVATIONS** (transactions clients)

* Numéro unique généré par procédure stockée
* QR code pour validation mobile
* Statuts multiples pour workflow complet



### **Relations critiques**

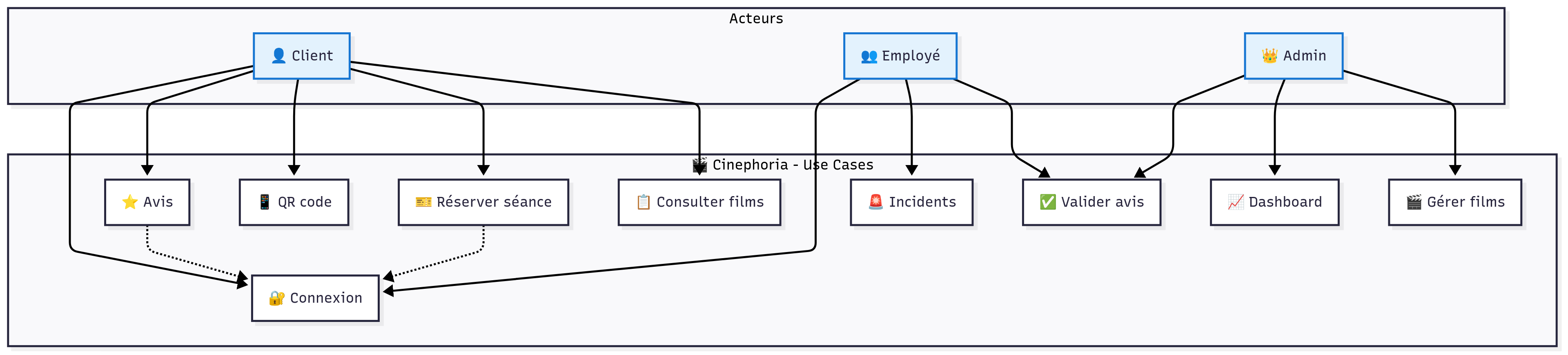
**SEANCES ↔ RESERVATIONS :** Relation 1,N avec gestion atomique des places disponibles via transaction SQL.

**UTILISATEURS ↔ AVIS :** Contrainte unique (utilisateur\_id, film\_id) évite les avis multiples.

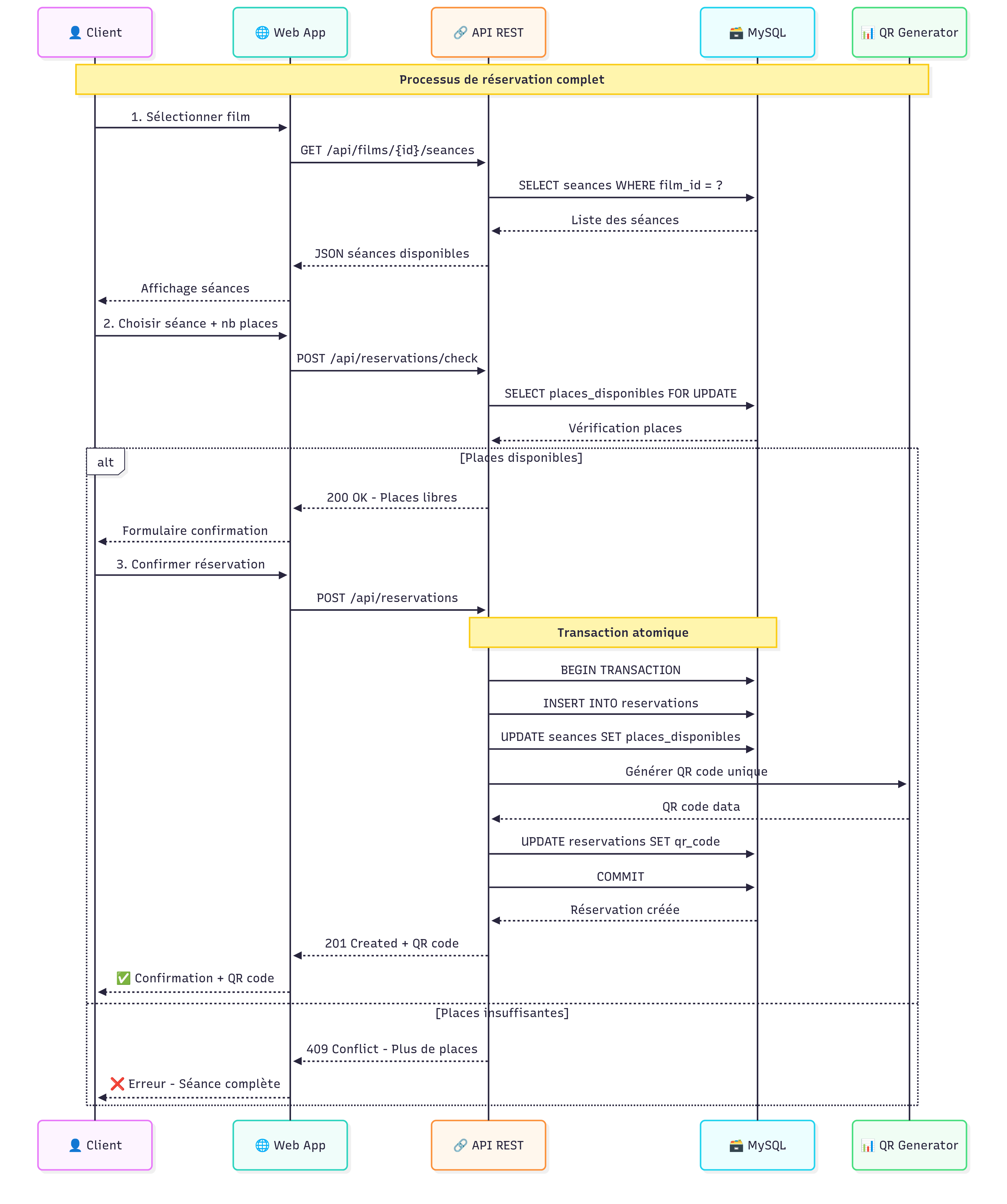
**INCIDENTS ↔ UTILISATEURS :** Double relation (déclarant/résolveur) pour traçabilité complète.

## **Diagrammes UML**

### **Diagramme de cas d'utilisation**



### **Diagramme de séquence - Réservation**



## **Sécurité implémentée**

### **Protection backend**

#### **Injection SQL :**

// Requêtes préparées exclusivement

$stmt = $pdo->prepare("SELECT \* FROM seances WHERE film\_id = ? AND date\_seance >= ?");

$stmt->execute([$film\_id, $date]);

// Validation stricte des paramètres

$film\_id = filter\_var($\_POST['film\_id'], FILTER\_VALIDATE\_INT);

if (!$film\_id) {

throw new InvalidArgumentException("ID film invalide");

}

#### **Authentification sécurisée :**

// Hashage bcrypt

$hash = password\_hash($password, PASSWORD\_DEFAULT);

// Vérification

if (password\_verify($input\_password, $stored\_hash)) {

// Régénération session ID

session\_regenerate\_id(true);

$\_SESSION['user\_id'] = $user['id'];

$\_SESSION['role'] = $user['role'];

}

#### **Gestion des rôles :**

// Middleware autorisation

function requireRole($required\_role) {

if (!isset($\_SESSION['role']) || $\_SESSION['role'] !== $required\_role) {

http\_response\_code(403);

exit('Accès refusé');

}

}

// Usage

requireRole('admin'); // Avant actions administratives

### **Protection frontend**

#### **XSS Prevention :**

// Échappement systématique

echo htmlspecialchars($user\_input, ENT\_QUOTES, 'UTF-8');

// Headers de sécurité

header("Content-Security-Policy: default-src 'self'");

header("X-Frame-Options: DENY");

header("X-Content-Type-Options: nosniff");

#### **CSRF Protection :**

****// Token CSRF

$\_SESSION['csrf\_token'] = bin2hex(random\_bytes(32));

// Vérification

if (!hash\_equals($\_SESSION['csrf\_token'], $\_POST['csrf\_token'])) {

throw new Exception("Token CSRF invalide");

}

### **Sécurité base de données**

#### **Contraintes métier :**

-- Éviter les données incohérentes

ALTER TABLE avis ADD CONSTRAINT chk\_note CHECK (note >= 1 AND note <= 5);

ALTER TABLE seances ADD CONSTRAINT chk\_places CHECK (places\_disponibles >= 0);

-- Contrainte unique pour éviter doubles réservations

ALTER TABLE avis ADD CONSTRAINT uk\_user\_film UNIQUE (utilisateur\_id, film\_id);

#### **Triggers de sécurité :**

-- Audit trail automatique

CREATE TRIGGER audit\_reservations

AFTER UPDATE ON reservations

FOR EACH ROW

INSERT INTO audit\_log (table\_name, action, old\_values, new\_values, timestamp)

VALUES ('reservations', 'UPDATE', JSON\_OBJECT(...), JSON\_OBJECT(...), NOW());



## **Plan de test**

### **Stratégie de test globale**

#### **Tests unitaires (30% effort)**

**Objectif :** Valider le comportement des fonctions critiques

**Fonctions testées :**

// Tests authentification

testValidLogin() // Connexion valide

testInvalidCredentials() // Échec connexion

testPasswordHashing() // Sécurité mot de passe

// Tests réservation

testCreateReservation() // Création normale

testOverbooking() // Prévention survente

testConcurrentAccess() // Accès simultané

// Tests validation

testEmailValidation() // Format email

testInputSanitization() // Nettoyage données

**Framework :** PHPUnit avec coverage 80% du code métier

#### **Tests fonctionnels (40% effort)**

**Objectif :** Valider les parcours utilisateur complets

**Scénarios prioritaires :**

1. **Parcours réservation complet**
   * Inscription → Connexion → Sélection film → Réservation → QR code
   * Vérification email confirmation
   * Test sur mobile et desktop
2. **Gestion administrative**
   * Connexion admin → Ajout film → Création séances
   * Validation avis → Export statistiques
   * Gestion employés et permissions
3. **Application bureautique**
   * Connexion employé → Saisie incident
   * Suivi résolution → Export rapport

#### **Tests d'intégration (20% effort)**

**Objectif :** Valider la communication entre composants

**Tests API :**

# Tests automatisés avec curl

curl -X POST /api/reservations \

-H "Content-Type: application/json" \

-d '{"seance\_id":1,"nb\_places":2}'

# Tests charge avec Apache Bench

ab -n 1000 -c 10 http://localhost/api/films

**Tests base de données :**

* Intégrité référentielle
* Performance des requêtes complexes
* Comportement des triggers en charge

#### **Tests sécurité (10% effort)**

**Objectif :** Valider la robustesse sécuritaire

**Tests automatisés :**

// Test injection SQL

$malicious\_input = "'; DROP TABLE users; --";

testSqlInjection($malicious\_input); // Doit échouer

// Test XSS

$xss\_payload = "<script>alert('XSS')</script>";

testXSSPrevention($xss\_payload); // Doit être échappé

// Test autorisation

testUnauthorizedAccess(); // Accès admin sans droits

### **Critères de réussite**

**Fonctionnel :**

* 100% des User Stories validées
* Parcours critique sans erreur
* Performance < 2s sur actions principales

**Technique :**

* 0 vulnérabilité critique
* Couverture tests > 70%
* Disponibilité > 99% en production

**Qualité :**

* Code respectant PSR-12
* Documentation à jour
* Logs structurés et exploitables

## **Déploiement et CI/CD**

### **Stratégie de déploiement**

#### **Environnements :**

**Local :** XAMPP + fichiers de configuration development

http://localhost/cinephoria/

MySQL : localhost:3306

MongoDB : localhost:27017

**Staging :** Environnement de test sur Fly.io

https://cinephoria-staging.fly.dev/

Base de données de test avec données fictives

**Production :** Déploiement final sur Fly.io

https://cinephoria.fly.dev/

Base de données production avec sauvegardes

#### **Configuration Fly.io :**

# fly.toml

app = "cinephoria"

[env]

APP\_ENV = "production"

PHP\_VERSION = "8.3"

[http\_service]

internal\_port = 8080

force\_https = true

[[services]]

http\_checks = []

internal\_port = 8080

processes = ["app"]

protocol = "tcp"

[deploy]

release\_command = "php artisan migrate --force"