

DOCUMENTATION TECHNIQUE - ECORIDE

Projet : Plateforme de covoiturage écologique

Contexte : ECF - Titre Professionnel Développeur Web et Web Mobile (Studi 2025)

Auteur : PODVIN Sandy

Date : Octobre 2025

1. INTRODUCTION

1.1 Présentation du projet

EcoRide est une plateforme web de covoiturage écologique qui vise à réduire l'impact environnemental des déplacements en favorisant le partage de véhicules entre particuliers, avec une attention particulière portée aux véhicules électriques.

Valeurs clés :

- Écologie : promotion des véhicules électriques
- Économie : système de crédits avantageux
- Transparence : système d'avis modéré
- Sécurité : validation des trajets et des utilisateurs

1.2 Objectifs du projet

Objectifs fonctionnels :

- Permettre la recherche et réservation de trajets
- Faciliter la publication de trajets par les conducteurs
- Gérer un système de crédits sécurisé
- Modérer les avis et signalements
- Fournir des outils d'administration

Objectifs techniques :

- Développer une application web responsive
- Implémenter une architecture MVC maintenable
- Sécuriser les transactions et données utilisateurs
- Déployer l'application sur un serveur de production

1.3 Périmètre fonctionnel

Utilisateurs concernés :

- Visiteurs (recherche de trajets)
- Utilisateurs (passagers et/ou chauffeurs)
- Employés (modération)

- Administrateurs (gestion globale)

Fonctionnalités principales :

- Inscription/Connexion avec gestion des rôles
 - Recherche de trajets avec filtres multiples
 - Réservation et paiement par crédits
 - Publication de trajets (chauffeurs)
 - Système d'avis avec validation employé
 - Gestion des signalements
 - Tableau de bord administrateur
 - Système de notifications
 - Formulaire de contact
-

2. RÉFLEXIONS TECHNOLOGIQUES INITIALES

2.1 Stack technique retenue

Frontend : HTML5, CSS3, JavaScript

Backend : PHP 8.2 (architecture MVC)

BDD SQL : MySQL 8.0 / MariaDB 10.4

BDD NoSQL : MongoDB Atlas (logs, notifications)

Serveur : Apache 2.4

Outils : Git, GitHub, Composer, XAMPP

2.2 Justifications des choix techniques

PHP & MySQL :

- Technologie maîtrisée et éprouvée
- Excellente documentation et communauté active
- Hébergement accessible et économique
- Parfaitement adapté aux applications CRUD

Architecture MVC (Model-View-Controller) :

- Séparation claire des responsabilités
- Maintenance et évolution facilitées
- Réutilisabilité du code
- Tests unitaires simplifiés

JavaScript :

- Performance optimale sans frameworks lourds
- Contrôle total du code frontend
- Pas de dépendances externes
- Apprentissage des fondamentaux

MongoDB (NoSQL) :

- Flexibilité pour les logs
 - Structure de données non-rigide
 - Requêtes rapides sur documents complexes
-

3. CONFIGURATION DE L'ENVIRONNEMENT DE TRAVAIL

3.1 Prérequis

Logiciels requis :

- PHP 8.2 ou supérieur
- MySQL 8.0 / MariaDB 10.4+
- Apache 2.4
- Composer
- Git
- Compte MongoDB Atlas (gratuit)

Recommandations :

- XAMPP (Windows/Mac) ou LAMP (Linux)
- Visual Studio Code avec extensions PHP
- Postman pour tester les API

3.2 Installation locale

Étape 1 : Cloner le repository

```
```bash
git clone https://github.com/Ozecult/ECF---Studi.git
cd ECF---Studi
```
```

Étape 2 : Configuration de la base de données

```
```bash
Créer la base de données
mysql -u root -p

CREATE DATABASE ecoride CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci;

Importer le schéma
mysql -u root -p ecoride < database/schema.sql

Importer les données de test
mysql -u root -p ecoride < database/data.sql
EXIT;
````
```

Étape 3 : Configuration

- Copier config/database.example.php vers config/database.php
- Modifier les identifiants de connexion MySQL
- Configurer la connexion MongoDB Atlas

Étape 4 : Démarrage

```
```bash
Démarrer Apache et MySQL via XAMPP
Accéder à http://localhost:8000
````
```

3.3 Structure du projet

```
ecoride/
├── css/
│   └── fonts/
│   └── page/
│       ├── admin.css
│       ├── covoitages.css
│       ├── details.css
│       ├── employe.css
│       ├── forms.css
│       ├── index.css
│       └── utilisateur.css
│   └── animations.css
│   └── barre-recherche.css
│   └── base.css
│   └── components.css
│   └── fonts.css
│   └── reset.css
│   └── style.css
└── database/
    └── data.sql
    └── schema.sql
docs/
    └── documentation_technique.pdf
    └── charte_graphique.pdf
    └── gestion_projet.pdf
    └── manuel_utilisateur.pdf
img/
js/
└── script.js
php/
    └── api/
        └── api-router.php
    └── config/
        ├── config.php
        └── Database.php
        └── mongodb.php
```

```
|- controllers/
|   |- AdminController.php
|   |- AuthController.php
|   |- ContactController.php
|   |- EmployeController.php
|   |- TrajetController.php
|   |- UserController.php
|- helpers/
|   |- EmailSimulator.php
|- models/
|   |- ActivityLog.php
|   |- Avis.php
|   |- MessageContact.php
|   |- Preference.php
|   |- Signalement.php
|   |- Trajet.php
|   |- User.php
|   |- Vehicule.php
|- views/
|   |- admin.php
|   |- barrerecherche.php
|   |- connexion.php
|   |- contact.php
|   |- covoitages.php
|   |- details.php
|   |- employe.php
|   |- footer.php
|   |- header.php
|   |- home.php
|   |- inscription.php
|   |- mdp-oublie.php
|   |- mentionslegales.php
|   |- rechercher.php
|   |- utilisateur.php
|   |- index.php
|- uploads/
|   |- photos/
|- README.md
```

3.4 Alternative : Installation avec Docker (recommandé)

Docker permet de démarrer l'application en 1 commande sans installer XAMPP.

Avantages :

- Environnement standardisé et reproduitible
- Aucun conflit de ports ou versions
- MySQL + MongoDB + PhpMyAdmin + Mongo Express inclus

- Installation automatique des dépendances

Prérequis :

- Docker Desktop installé (<https://www.docker.com/products/docker-desktop/>)
- 4 Go de RAM disponibles minimum

Installation rapide :

1. Cloner le repository git clone <https://github.com/Ozecult/ECF---Studi.git>
cd ECF---Studi
2. Démarrer tous les services docker-compose up -d
3. Accéder à l'application
 - Application : <http://localhost:8080/php/index.php>
 - PhpMyAdmin : <http://localhost:8081>
 - Mongo Express : <http://localhost:8082>

Architecture Docker :

Le projet utilise 5 conteneurs orchestrés via Docker Compose :

| Service | Image | Port | Fonction |
|---------------|------------------|-------|--------------------|
| web | PHP 8.2 + Apache | 8080 | Aplication EcoRide |
| mysql | MySQL 8.0 | 3306 | Base de données |
| mongo | MongoDB 7.0 | 27017 | Base NoSQL (logs) |
| phpmyadmin | PhpMyadmin | 8081 | Interface MySQL |
| mongo-express | MongoExpress | 8082 | Interface MongoDB |

Fichiers Docker :

- Dockerfile : Configuration de l'image PHP/Apache personnalisée
- docker-compose.yml : Orchestration des services
- .dockerignore : Exclusions pour optimisation
- .env.example : Variables d'environnement

Commandes utiles :

- Démarrer les services : docker-compose up -d
- Arrêter les services : docker-compose down
- Voir les logs : docker-compose logs -f
- Redémarrer : docker-compose restart

Compatibilité : La configuration config.php détecte automatiquement l'environnement (Docker ou XAMPP) et s'adapte en conséquence. Les deux environnements peuvent coexister sans conflit.

Documentation détaillée : Voir README_DOCKER.md à la racine du projet pour le guide complet d'utilisation.

3.5 Variables d'environnement

Fichier : config/database.php

```
<?php
class Database {
    private static $instance = null;
    private $conn;

    private $host = 'localhost';
    private $db_name = 'ecoride';
    private $username = 'root';
    private $password = '';

    private function __construct() {
        try {
            $dsn = "mysql:host={$this->host};dbname={$this-
>db_name};charset=utf8mb4";
            $options = [
                PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
                PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
                PDO::ATTR_EMULATE_PREPARES => false
            ];
            $this->conn = new PDO($dsn, $this->username, $this-
>password, $options);
        } catch(PDOException $e) {
            error_log("Erreur connexion : " . $e->getMessage());
            die("Erreur de connexion à la base de données");
        }
    }

    public static function getInstance() {
        if ($self::$instance === null) {
            $self::$instance = new Database();
        }
        return $self::$instance;
    }

    public function getConnection() {
        return $this->conn;
    }
}
```

```

## Utilisation

```
```php
$db = Database::getInstance()->getConnection();
$stmt = $db->prepare("SELECT * FROM utilisateurs WHERE email = ?");
````
```

---

## 4. ARCHITECTURE DE L'APPLICATION

### 4.1 Architecture MVC

L'application suit le pattern **Model-View-Controller** :

#### Model (Modèles) :

- Gestion de la logique métier
- Interactions avec la base de données
- Validation des données

#### View (Vues) :

- Affichage des interfaces utilisateur
- Templates HTML/PHP
- Pas de logique métier

#### Controller (Contrôleurs) :

- Traitement des requêtes HTTP
- Orchestration entre Model et View
- Gestion des sessions et permissions

### 4.2 Flux de données

1. L'utilisateur effectue une action (ex: réserver un trajet)  
↓
2. Le routeur (index.php) analyse l'URL  
↓
3. Le contrôleur approprié est appelé  
↓
4. Le contrôleur sollicite le modèle pour les données  
↓
5. Le modèle interroge la base de données  
↓
6. Les données sont renvoyées au contrôleur  
↓
7. Le contrôleur transmet les données à la vue  
↓
8. La vue génère le HTML final  
↓
9. La page est affichée à l'utilisateur

## 4.3 API REST

Endpoints principaux :

```
```php
GET /api/api-router.php?action=get-user-data
POST /api/api-router.php?action=reserver-trajet
POST /api/api-router.php?action=create-trajet
GET /api/api-router.php?action=search-trajets
POST /api/api-router.php?action=soumettre-avis
POST /api/api-router.php?action=recharger-credits
````
```

Format de réponse :

```
```json
{
  "success": true,
  "message": "Opération réussie",
  "data": {},
  "timestamp": "2025-10-07 14:30:00"
}
````
```

---

## 5. MODÉLISATION DES DONNÉES

### 5.1 Base de données relationnelle (MySQL)

Tables principales :

1. **utilisateurs** : Informations des utilisateurs
2. **vehicules** : Véhicules des conducteurs
3. **trajets** : Trajets proposés
4. **reservations** : Réservations des passagers
5. **avis** : Avis et notes
6. **transactions** : Historique des crédits
7. **signalements** : Signalements de problèmes
8. **employes** : Comptes employés
9. **roles** : Rôles système
10. **preferences\_types** : Types de préférences
11. **preferences\_utilisateurs** : Préférences utilisateurs
12. **messages\_contact** : Contacter EcoRide

## 5.2 Modèle Conceptuel de Données (MCD)

### **Relations principales :**

UTILISATEUR (1,n) ---- possède ----> (0,n) VEHICULE

UTILISATEUR (1,n) ---- conduit ----> (0,n) TRAJET

UTILISATEUR (1,n) ---- réserve ---> (0,n) RESERVATION

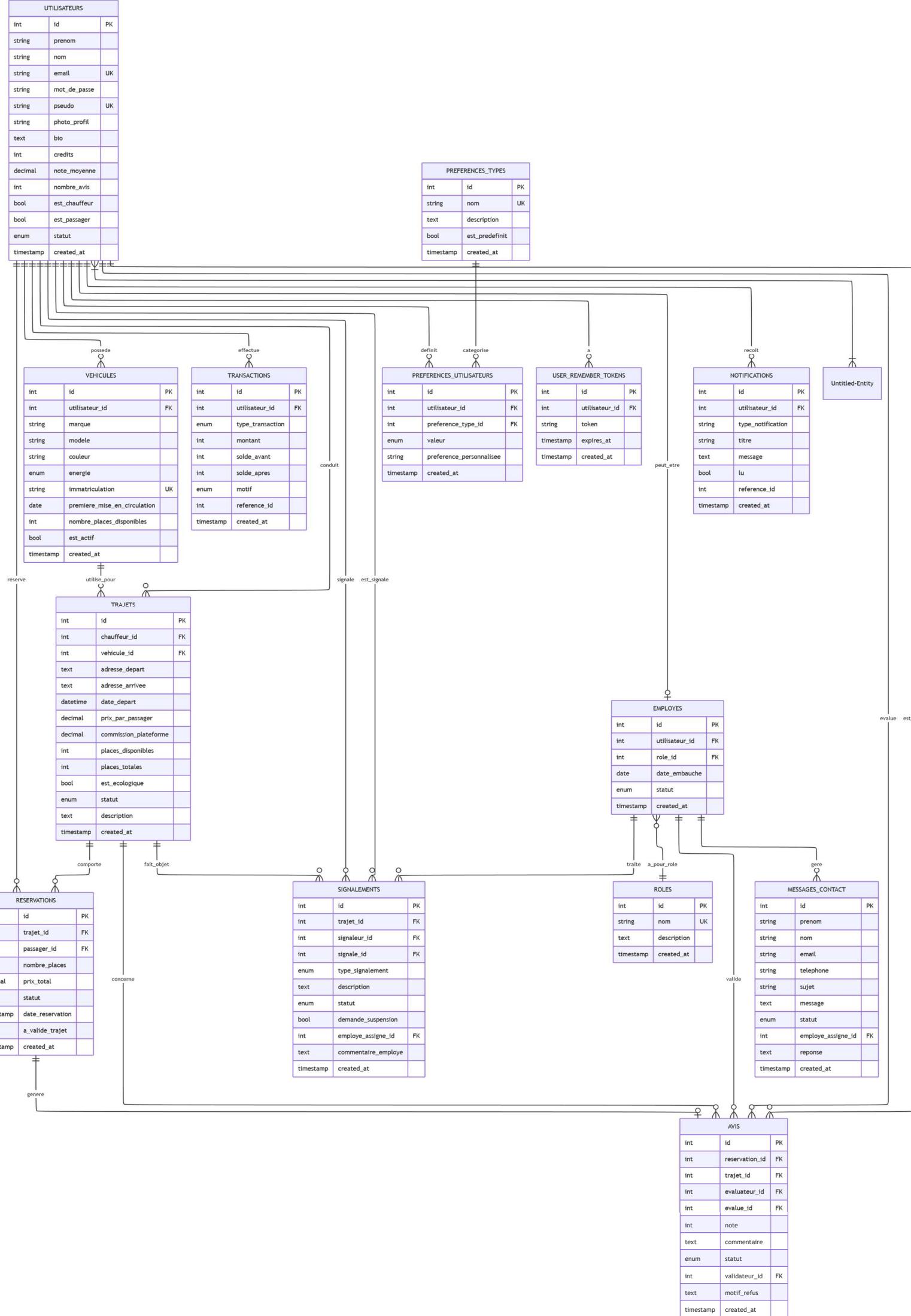
TRAJET (1,1) ----- comporte ----> (0,n) RESERVATION

RESERVATION (1,1) --- génère ----> (0,1) AVIS

UTILISATEUR (1,n) --- effectue ---> (0,n) TRANSACTION

### **Cardinalités importantes :**

- Un utilisateur peut avoir plusieurs véhicules
- Un trajet appartient à un seul conducteur
- Une réservation concerne un trajet et un passager
- Un avis est lié à une réservation



## 5.3 Base de données NoSQL (MongoDB)

Collections :

```
```json
// logs_activite :
{
  "_id": ObjectId,
  "utilisateur_id": 42,
  "action": "reservation_trajet",
  "details": "Réservation trajet #123",
  "ip_address": "192.168.1.1",
  "timestamp": ISODate("2025-10-07T14:30:00Z")
}
```

```

notifications :

```
```json
{
  "_id": ObjectId,
  "utilisateur_id": 42,
  "type": "nouveau_trajet",
  "titre": "Nouvelle réservation",
  "message": "Votre trajet a une nouvelle réservation",
  "lu": false,
  "created_at": ISODate("2025-10-07T14:30:00Z")
}
```

```

## 5.4 Triggers et procédures stockées

Trigger : after\_reservation\_insert

```
```sql
-- Débite les crédits du passager
-- Met à jour les places disponibles
-- Enregistre la transaction
-- Crée une notification
```

```

Trigger : after\_avis\_validated

```
```sql
-- Recalcule la note moyenne de l'utilisateur
-- Met à jour le nombre d'avis
```

```

**Trigger : after\_trajet\_cancelled**

```
```sql
-- Rembourse tous les passagers
-- Annule toutes les réservations
-- Envoie des notifications
```

```

## 6. DIAGRAMMES UML

### 6.1 Diagramme de cas d'utilisation

**Acteurs :**

- Visiteur
- Utilisateur (Passager/Chauffeur)
- Employé
- Administrateur

**Cas d'utilisation principaux :**

**Visiteur :**

- Consulter les trajets disponibles
- Rechercher un trajet
- Créer un compte

**Utilisateur :**

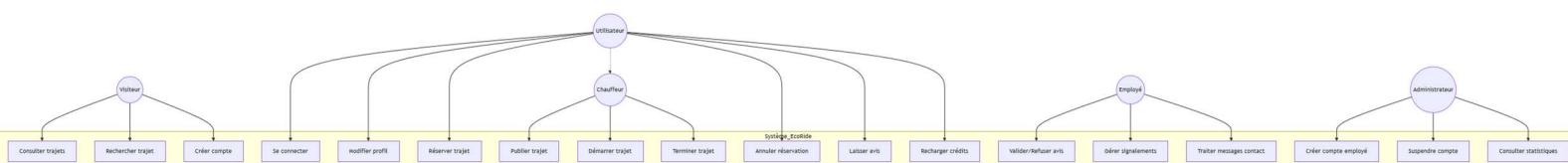
- Se connecter / Se déconnecter
- Modifier son profil
- Rechercher et réserver un trajet
- Publier un trajet (chauffeur)
- Démarrer/terminer un trajet (chauffeur)
- Annuler une réservation
- Laisser un avis
- Recharger des crédits

**Employé :**

- Valider/refuser des avis
- Gérer les signalements
- Traiter les messages contact

**Administrateur :**

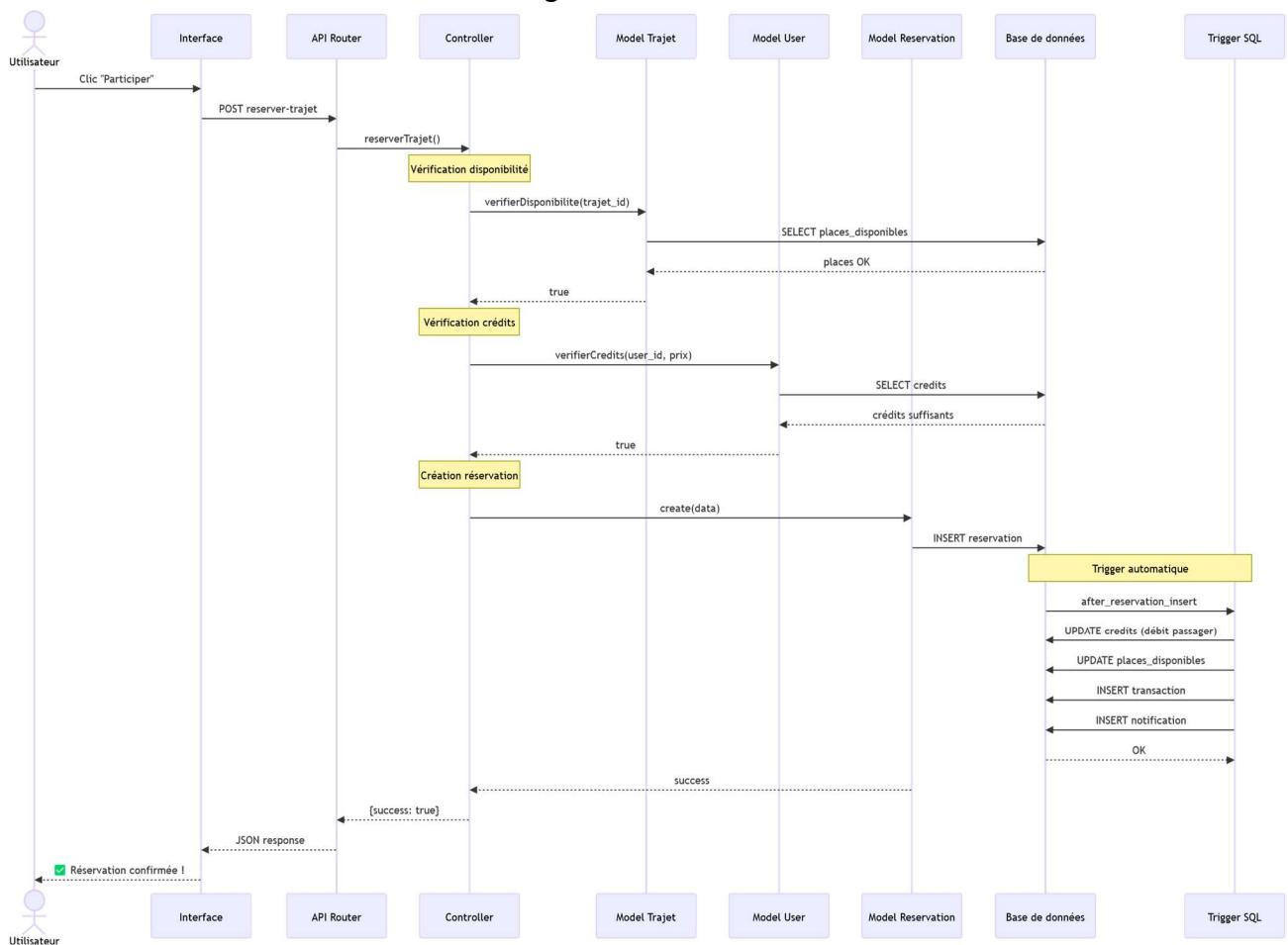
- Créer des comptes employés
- Suspendre/réactiver des comptes
- Consulter les statistiques
- Gérer les demandes de suspension



## 6.2 Diagrammes de séquence

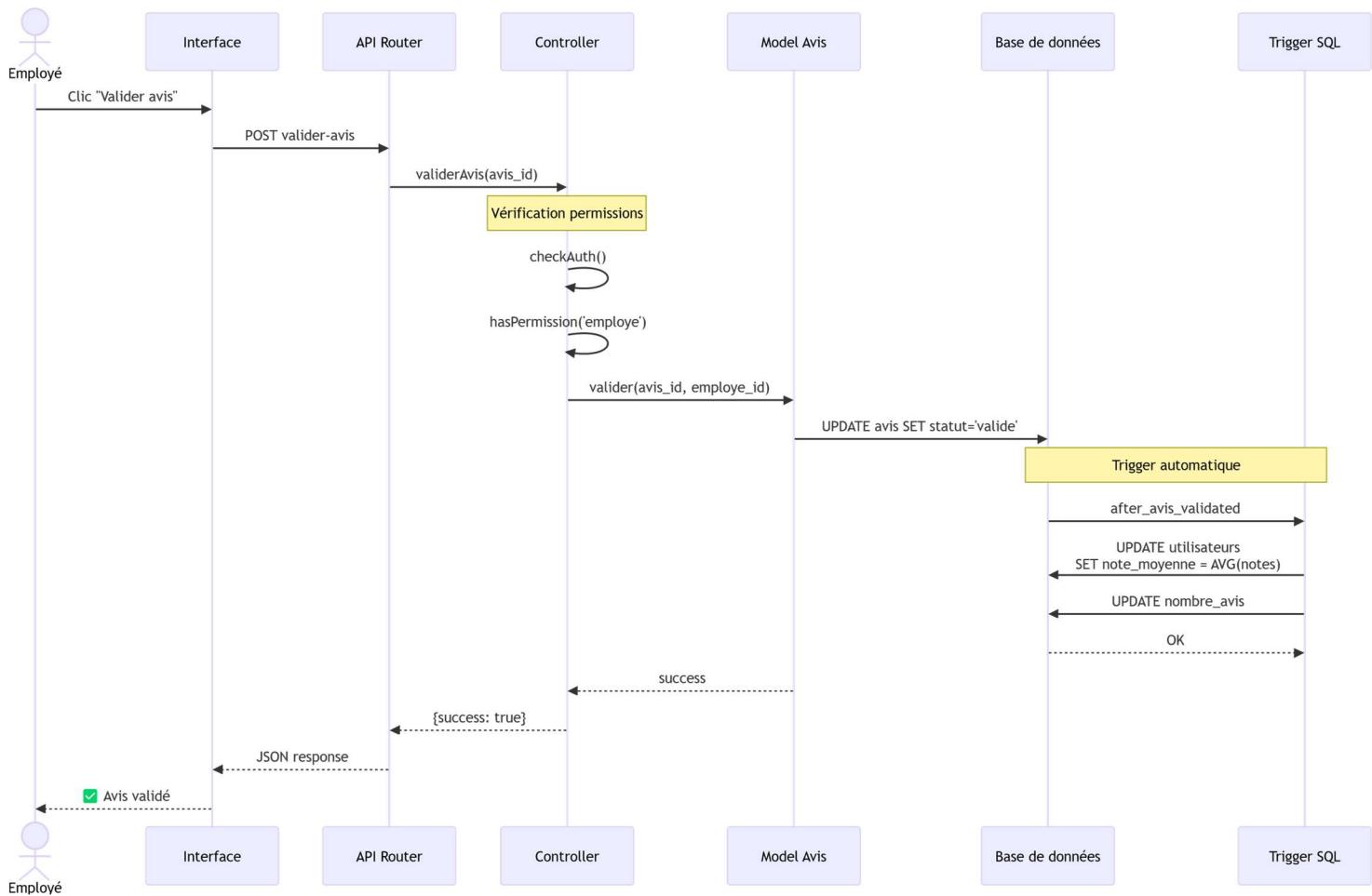
### Séquence 1 : Réservation d'un trajet

Utilisateur -> Interface : Clic "Participer"  
 Interface -> API : POST reserver-trajet  
 API -> Controller : reserverTrajet()  
 Controller -> Model Trajet : verifierDisponibilite()  
 Model Trajet -> BDD : SELECT places\_disponibles  
 BDD -> Model Trajet : places OK  
 Controller -> Model User : verifierCredits()  
 Model User -> BDD : SELECT credits  
 BDD -> Model User : crédits OK  
 Controller -> Model Reservation : create()  
 Model Reservation -> BDD : INSERT reservation  
 BDD -> Trigger : after\_reservation\_insert  
 Trigger -> BDD : UPDATE credits, places  
 BDD -> Trigger : OK  
 Model Reservation -> Controller : success  
 Controller -> API : JSON {success: true}  
 API -> Interface : Confirmation  
 Interface -> Utilisateur : Message succès



## Séquence 2 : Validation d'un avis (Employé)

Employé -> Interface : Clic "Valider avis"  
 Interface -> API : POST valider-avis  
 API -> Controller : validerAvis()  
 Controller -> Model Avis : valider(avis\_id)  
 Model Avis -> BDD : UPDATE avis SET statut='valide'  
 BDD -> Trigger : after\_avis\_validated  
 Trigger -> BDD : Recalcul note\_moyenne  
 BDD -> Trigger : OK  
 Model Avis -> Controller : success  
 Controller -> API : JSON {success: true}  
 API -> Interface : Confirmation  
 Interface -> Employé : Avis validé



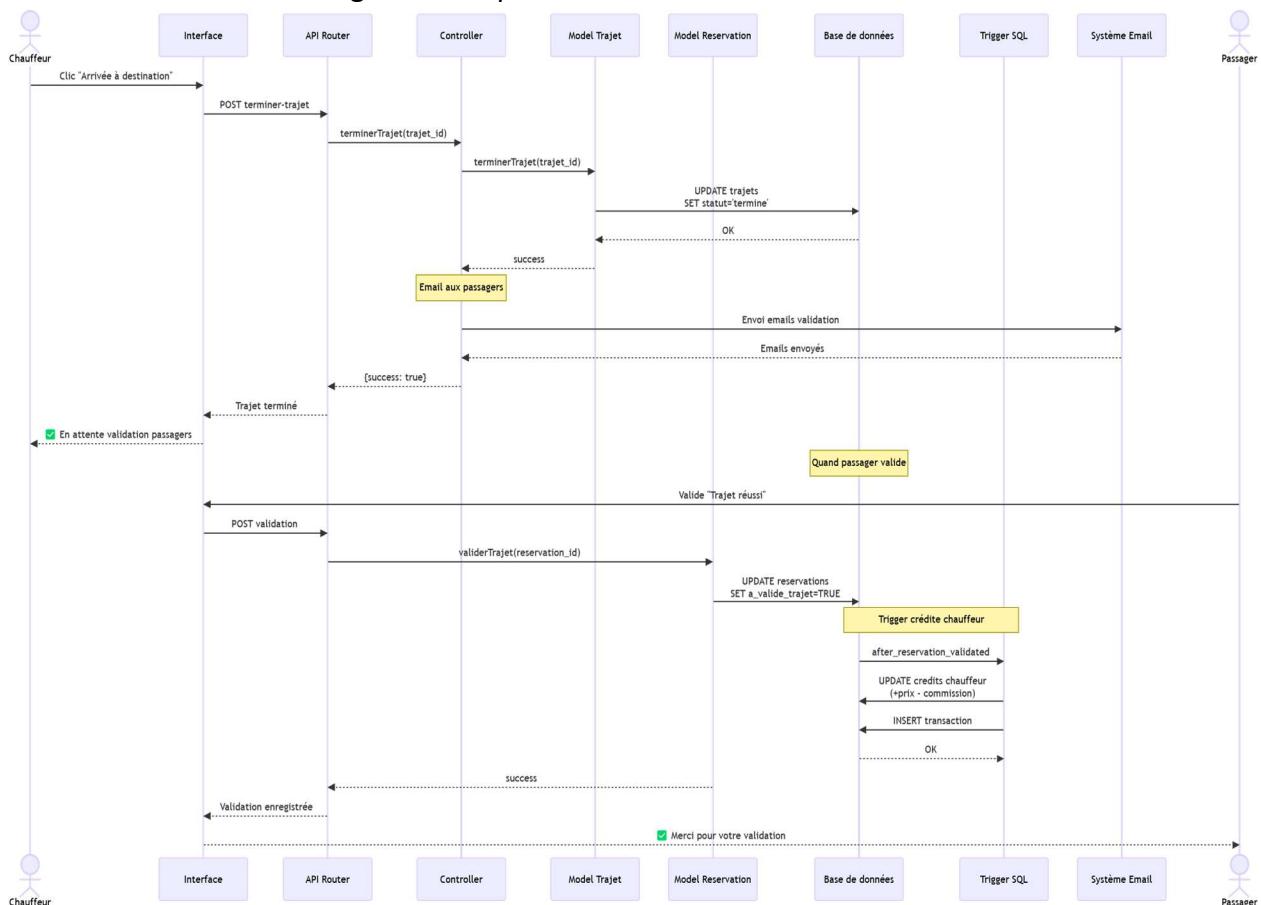
## Séquence 3 : Terminer un trajet et créditer le chauffeur

Chauffeur -> Interface : Clic "Arrivée à destination"  
 Interface -> API : POST terminer-trajet  
 API -> Controller : terminerTrajet(trajet\_id)  
 Controller -> Model Trajet : terminerTrajet(trajet\_id)

Model Trajet -> BDD : UPDATE trajets SET statut='terminé'  
 BDD -> Model Trajet : OK  
 Model Trajet -> Controller : success  
 Controller -> Système Email : Envoi emails validation  
 Système Email -> Controller : Emails envoyés  
 Controller -> API : JSON {success: true}  
 API -> Interface : Trajet terminé  
 Interface -> Chauffeur : En attente validation passagers

Note : Quand passager valide

Passager -> Interface : Valide "Trajet réussi"  
 Interface -> API : POST validation  
 API -> Model Reservation : validerTrajet(reservation\_id)  
 Model Reservation -> BDD : UPDATE reservations SET a\_valide\_trajet=TRUE  
 BDD -> Trigger : after\_reservation\_validated  
 Trigger -> BDD : UPDATE credits chauffeur (+prix - commission)  
 Trigger -> BDD : INSERT transaction  
 BDD -> Trigger : OK  
 Model Reservation -> API : success  
 API -> Interface : Validation enregistrée  
 Interface -> Passager : Merci pour votre validation



## 6.3 Diagramme de classes

### Classe User

```
User

- id: int
- prenom: string
- email: string
- mot_de_passe: string
- credits: int
- est_chauffeur: bool
- est_passager: bool

+ getUserById(id): User
+ updateUser(id, data): bool
+ verifierCredits(id, montant): bool
+ addCredits(id, montant): bool
```

### Classe Trajet

```
Trajet

- id: int
- chauffeur_id: int
- vehicule_id: int
- adresse_depart: string
- adresse_arrivee: string
- date_depart: datetime
- prix_par_passager: int
- places_disponibles: int

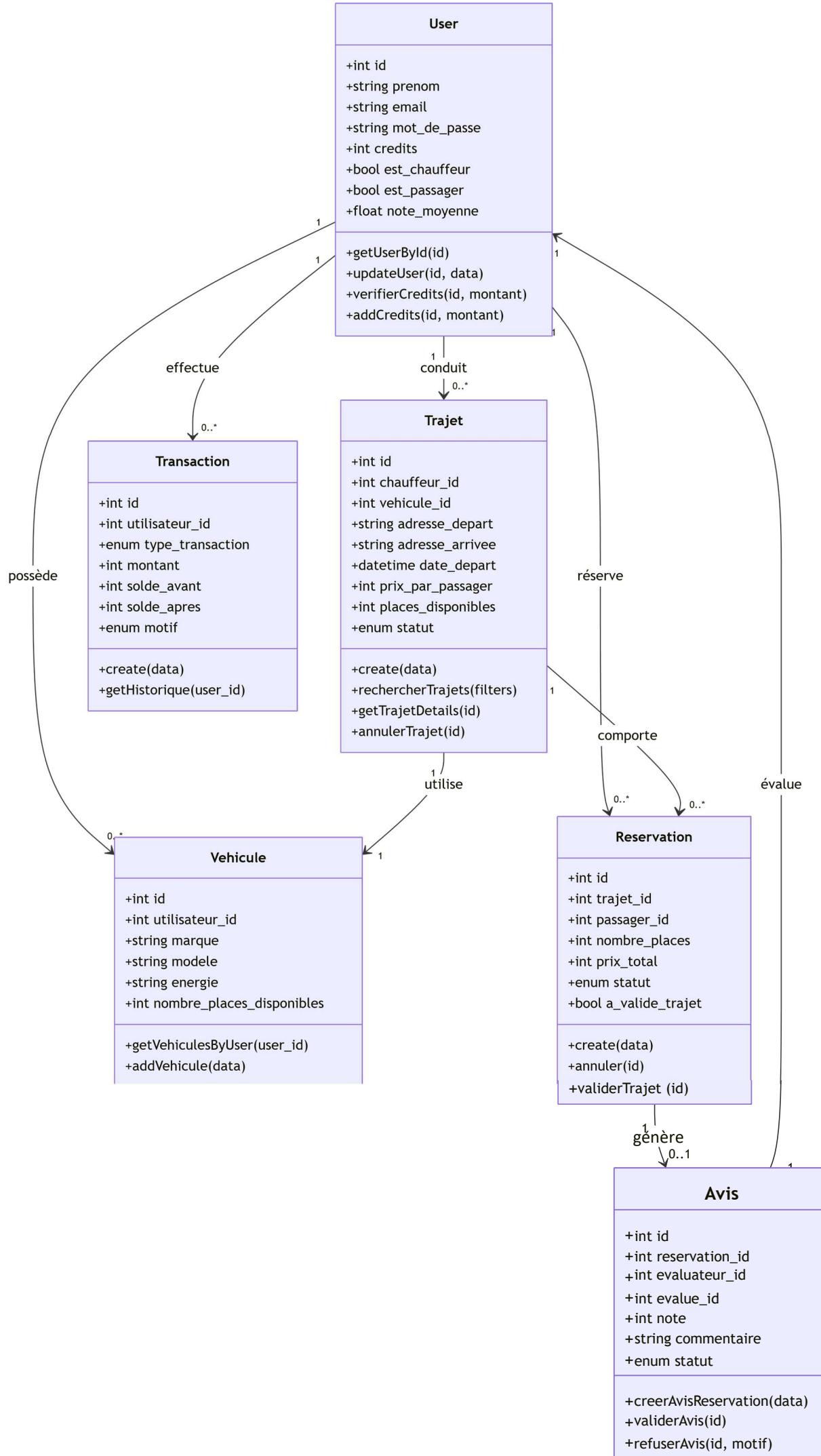
+ create(data): int
+ rechercherTrajets(filters): array
+ getTrajetDetails(id): array
+ annulerTrajet(id): bool
```

### Classe Reservation

```
Reservation

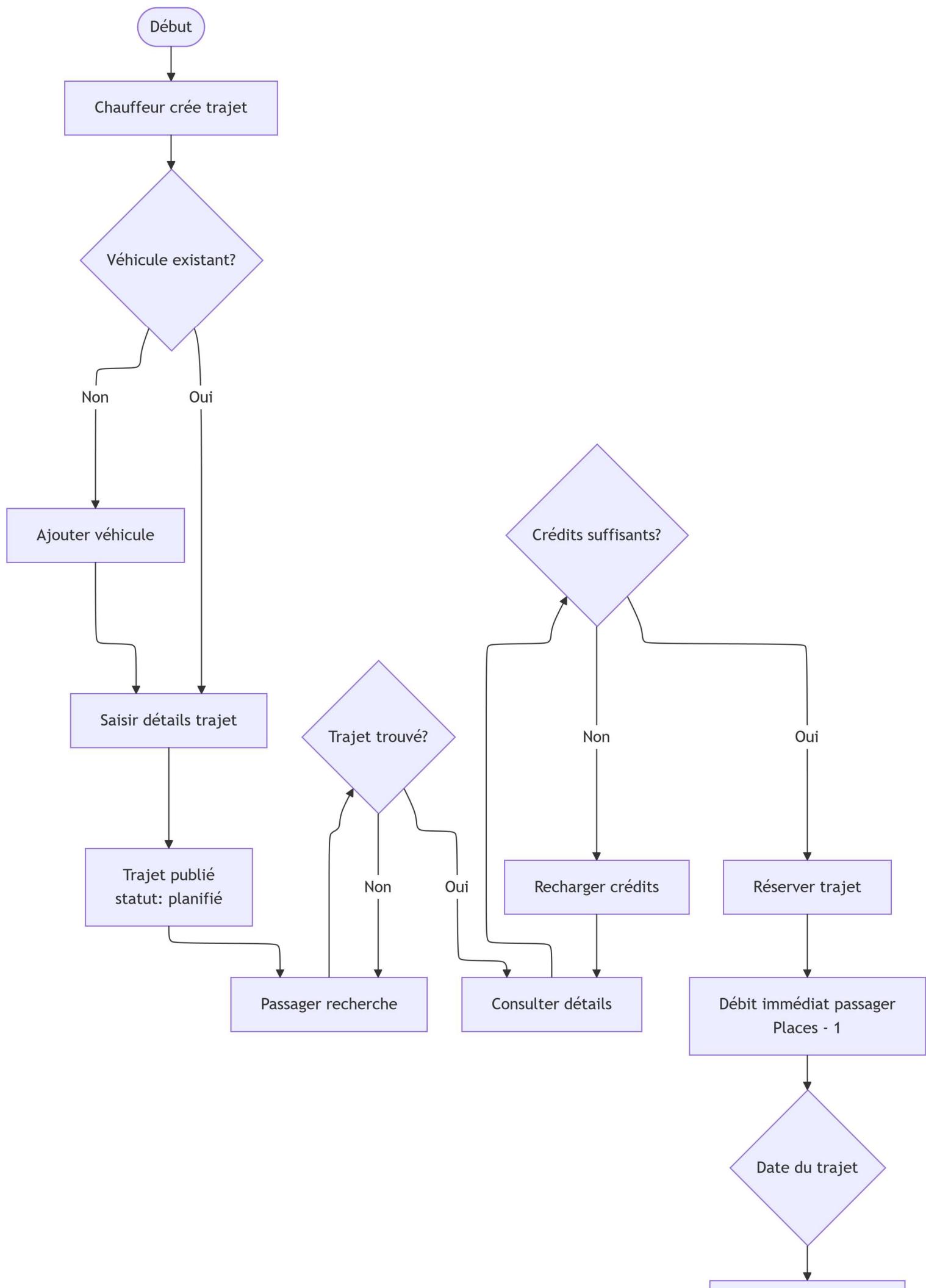
- id: int
- trajet_id: int
- passager_id: int
- nombre_places: int
- prix_total: int
- statut: enum

+ create(data): bool
+ annuler(id): bool
+ validerTrajet(id): bool
```



## 6.4 Diagramme d'activité : Processus complet d'un trajet

```
Début
 ↓
Chauffeur crée trajet → Trajet publié
 ↓
Passager recherche et réserve → Débit crédits passager
 ↓
Jour J : Chauffeur démarre trajet
 ↓
Chauffeur termine trajet → Email envoyé aux passagers
 ↓
Passager valide ?
 └── OUI → Crédit chauffeur + Avis (si souhaité)
 └── NON (problème) → Signalement → Employé traite
 ↓
Employé valide avis → Note mise à jour
 ↓
Fin
```



Chauffeur démarre  
statut: en\_cours

Trajet en cours...

Chauffeur termine  
statut: terminé

Email aux passagers

Passager valide?

Trajet réussi

Problème

Validation OK

Signalement créé

Employé traite

Résolution

Crédit chauffeur  
+prix - commission

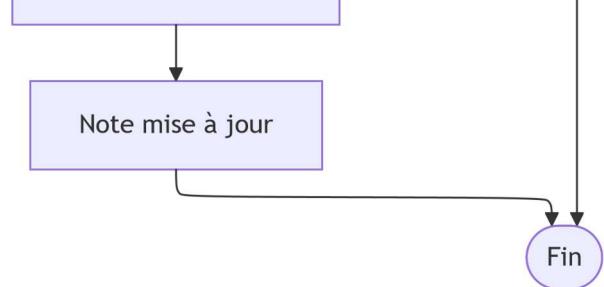
OK

Suspension

Passager laisse avis

Employé valide avis

Admin suspend compte



## 7. SÉCURITÉ

### 7.1 Authentification

```
```php
// Lors de l'inscription
$hashedPassword = password_hash($password, PASSWORD_BCRYPT);

// Lors de la connexion
if (password_verify($inputPassword, $storedHash)) {
    // OK
}
```

Politique de mots de passe :

- Minimum 8 caractères
- Au moins 1 majuscule
- Au moins 1 chiffre
- Au moins 1 caractère spécial

Gestion des sessions :

```
```php
session_start([
 'cookie_lifetime' => 0,
 'cookie_httponly' => true,
 'cookie_secure' => true, // HTTPS uniquement
 'use_strict_mode' => true
]);
```

### 7.2 Protection contre les injections SQL

#### Utilisation de requêtes préparées (PDO) :

```
```php
$stmt = $db->prepare("SELECT * FROM utilisateurs WHERE email = ?");
$stmt->execute([$email]);
```
```

**Validation des entrées :**

```
```php
$email = filter_var($_POST['email'], FILTER_VALIDATE_EMAIL);
$prix = filter_var($_POST['prix'], FILTER_VALIDATE_INT);
````
```

## 7.3 Protection XSS

**Échappement des sorties :**

```
```php
echo htmlspecialchars($userInput, ENT_QUOTES, 'UTF-8');
````
```

**Content Security Policy (headers) :**

```
```php
header("Content-Security-Policy: default-src 'self'");
header("X-Content-Type-Options: nosniff");
header("X-Frame-Options: DENY");
````
```

## 7.4 Protection CSRF

```
```php
// Génération
$_SESSION['csrf_token'] = bin2hex(random_bytes(32));

// Vérification
if ($_POST['csrf_token'] !== $_SESSION['csrf_token']) {
    die('Token CSRF invalide');
}
````
```

## 7.5 Gestion des permissions

**Système de rôles :**

```
```php
function checkAuth() {
    if (!isset($_SESSION['user_id'])) {
        http_response_code(401);
        exit;
    }
}

function hasPermission($role) {
    return $_SESSION['role'] === $role;
}
````
```

~~~

## 7.6 Upload de fichiers sécurisé

**Validation des photos de profil :**

```
```php
// Vérification du type MIME
$allowedTypes = ['image/jpeg', 'image/png', 'image/webp'];
if (!in_array($_FILES['photo']['type'], $allowedTypes)) {
    throw new Exception('Type de fichier non autorisé');
}

// Limite de taille (2 MB)
if ($_FILES['photo']['size'] > 2 * 1024 * 1024) {
    throw new Exception('Fichier trop volumineux');
}

// Génération d'un nom unique
$filename = $userId . '_' . time() . '.' . $extension;
````
```

---

## 8. DÉPLOIEMENT

### 8.1 Stratégie de déploiement

**Environnements :**

- **Local** (développement) : XAMPP
- **Production** : <https://www.alwaysdata.com>

**Outils de versioning :**

- Git pour le contrôle de version
- GitHub pour le repository distant
- Branches : main, develop, feat/\*

### 8.2 Checklist de déploiement

**Avant le déploiement :**

- Tests unitaires passent
- Code review effectué
- Base de données sauvegardée
- Variables d'environnement configurées
- Fichiers sensibles exclus (.gitignore)

#### Déploiement :

- Upload des fichiers via FTP/Git
- Import du schéma de base de données
- Configuration des permissions fichiers
- Vérification des chemins absolus
- Test des fonctionnalités critiques

#### Après le déploiement :

- Vérification SSL/HTTPS
- Test de performance
- Monitoring des erreurs
- Sauvegarde automatique activée

### 8.3 Configuration serveur

#### Prérequis serveur :

```
PHP >= 8.2
MySQL >= 8.0 ou MariaDB >= 10.4
Apache >= 2.4 avec mod_rewrite
HTTPS (certificat SSL)
```

#### Configuration .htaccess :

```
RewriteEngine On
RewriteCond %{HTTPS} off
RewriteRule ^(.*)$ https:// %{HTTP_HOST}%{REQUEST_URI} [L,R=301]

Protection des fichiers sensibles
<FilesMatch "\.\.">
 Order allow,deny
 Deny from all
</FilesMatch>
```

#### Configuration php.ini :

```
```ini
upload_max_filesize = 2M
post_max_size = 2M
session.cookie_httponly = 1
session.cookie_secure = 1
display_errors = Off
log_errors = On
````
```

### 8.4 Commandes de déploiement

#### Déploiement via Git :

```
```bash
git pull origin main
composer install --no-dev
```
```

### Sauvegarde de la base de données :

```
```bash
mysqldump -u user -p > backup_$(date +%Y%m%d).sql
````
```

---

## 9. TESTS

### 9.1 Tests fonctionnels réalisés

#### Scénarios testés :

##### 1. Inscription/Connexion

- Création compte avec mot de passe valide ✓
- Connexion avec identifiants corrects ✓
- Rejet mot de passe faible ✓
- Gestion "Se souvenir de moi" ✓

##### 2. Recherche de trajets

- Recherche avec filtres multiples ✓
- Affichage des résultats corrects ✓
- Gestion absence de résultats ✓

##### 3. Réservation

- Réservation avec crédits suffisants ✓
- Blocage si crédits insuffisants ✓
- Mise à jour places disponibles ✓
- Débit immédiat des crédits ✓

##### 4. Publication trajet (chauffeur)

- Ajout véhicule ✓
- Création trajet valide ✓
- Validation formulaire ✓

##### 5. Système de crédits

- Recharge de crédits ✓
- Transfert au chauffeur après validation ✓
- Remboursement en cas d'annulation ✓

##### 6. Avis et signalements

- Soumission avis ✓

- Validation employé ✓
- Signalement avec prise en charge ✓

## 7. Administration

- Création compte employé ✓
- Suspension compte utilisateur ✓
- Affichage statistiques ✓

## 9.2 Tests de performance

### Résultats :

- Tests de performance effectués sur environnement local.
  - Temps de réponse acceptables pour une application de cette taille.
- 

# 10. MAINTENANCE ET ÉVOLUTIONS

## 10.1 Bugs connus

### Mineurs :

- Calcul heure d'arrivée approximatif (dépend API externe)
- Rafraîchissement manuel nécessaire après recharge crédits

### En cours de correction :

- Aucun bug critique identifié

## 10.2 Améliorations futures

### Court terme :

- Système de messagerie interne entre utilisateurs
- Notifications en temps réel (WebSockets)
- Application mobile (React Native / Flutter)

### Moyen terme :

- Intégration paiement réel (Stripe)
- Géolocalisation en temps réel des trajets
- Système de badges et récompenses

### Long terme :

- Intelligence artificielle pour suggestions de trajets
  - API publique pour partenaires
  - Internationalisation (multilingue)
- 

# CONCLUSION

EcoRide est une application web complète qui répond aux exigences du Titre Professionnel Développeur Web et Web Mobile. L'architecture MVC, la sécurité renforcée et l'utilisation de bases de données SQL et NoSQL sont intégrés dans l'application.

Le projet est évolutif et prêt pour une mise en production, avec une base solide permettant l'ajout de fonctionnalités futures.

---

#### **Liens utiles :**

- Repository GitHub : <https://github.com/Ozecult/ECF---Studi>
- Application déployée : <https://ecoride-ecologique.alwaysdata.net/php/index.php?page=home>
- Trello : <https://trello.com/b/sWHWiAEd/ecoride>