ECORIDE

DOCUMENTATION TECHNIQUE

Projet ECF – Développeur Web et Web Mobile

FRANCOIS MONTAIGNE Florence
15/01/2026

1. Présentation du projet

EcoRide est une application web de covoiturage développée dans le cadre de l'ECF du titre **Développeur Web et Web Mobile**.

L'objectif de l'application est de permettre la mise en relation de conducteurs et de passagers pour des trajets partagés, tout en intégrant une gestion de crédits, des avis utilisateurs et une modération par rôles.

Le projet respecte les contraintes fonctionnelles et techniques imposées par l'énoncé ECF.

---

2. Technologies utilisées

Frontend

- HTML5
- CSS3 (design responsive)
- JavaScript (ES6)
- Chart.js (statistiques administrateur)

Backend

- PHP 8.2
- API REST maison
- Authentification JWT

Base de données

- MariaDB / MySQL
- PDO (requêtes préparées)

Outils

- XAMPP (Apache / PHP / MariaDB)
- Visual Studio Code
- phpMyAdmin
- Git / GitHub

---

3. Architecture du projet

Le projet est organisé selon une séparation claire entre le frontend et le backend.

Arborescence principale

```
ecoride/
├─ backend/
│  ├─ public/
│  │  ├─ api/          (endpoints REST)
│  │  └─ frontend/     (pages HTML, JS, CSS)
│  ├─ src/             (logique applicative)
│  │  ├─ Auth.php
│  │  ├─ DB.php
│  │  ├─ Mailer.php
│  │  └─ config.php
│  └─ logs/            (journalisation technique)
└─ README.md
```

Cette organisation s'apparente à un **MVC léger**, facilitant la lisibilité, la maintenance et l'évolution du projet.

Approche orientée objet (POO)

Le backend du projet Ecoride repose sur une approche orientée objet.

Les différentes responsabilités sont réparties dans des classes dédiées :

- DB.php : gestion centralisée de la connexion à la base de données via PDO,

- Auth.php : gestion de l'authentification, des tokens et des rôles,

- Mailer.php : gestion des envois d'emails applicatifs,

- autres classes métier : encapsulation de la logique applicative côté serveur.

Cette organisation permet de séparer les responsabilités, de limiter la duplication

de code et d'améliorer la maintenabilité et l'évolutivité de l'application.

4. Gestion des rôles et droits

Trois rôles sont définis dans l'application :

- **Utilisateur** :
  - rechercher des trajets
  - réserver un covoiturage
  - laisser un avis
- **Employé** :
  - modérer les avis utilisateurs
  - consulter certaines données de gestion
- **Administrateur** :
  - créer des comptes employés
  - consulter les statistiques
  - gérer et auditer les crédits

Les accès sont contrôlés côté backend à l'aide de **JWT** et de vérifications de rôles systématiques dans les endpoints.

5 Base de données relationnelle

La base de données relationnelle est utilisée pour stocker les données métier structurées :

- utilisateurs
- véhicules
- trajets
- participants
- avis
- mouvements de crédits

Les échanges avec la base sont réalisés via PDO avec requêtes préparées afin de prévenir les injections SQL.

La modélisation de la base de données (diagramme de classe UML / MCD)

est fournie dans le document :

docs/Diagrammes_Ecoride.pdf.

5 bis. Stockage NoSQL (MongoDB)

En complément de la base de données relationnelle MariaDB utilisée pour les données métier

(utilisateurs, trajets, réservations, avis), le projet EcoRide intègre un stockage NoSQL de

type document via MongoDB.

MongoDB est utilisé pour stocker des données non transactionnelles et agrégées, notamment

des statistiques globales et des données de reporting. Ces informations sont stockées sous

forme de documents JSON, ce qui permet une structure souple, adaptée à l'analyse et à

l'évolution des besoins.

Ce choix permet de séparer clairement :

- les données métier persistantes (SQL),

- des données analytiques ou calculées (NoSQL),

conformément aux bonnes pratiques d'architecture backend.

Les fichiers de démonstration NoSQL sont regroupés dans le dossier `nosql/`, incluant :
- un exemple de document JSON (`aggregations.json`),
- une documentation d'installation et d'utilisation (`mongo_setup.md`).
MongoDB est également utilisé pour stocker des données faiblement relationnelles nécessitant une structure souple et un cycle de vie indépendant des données métier SQL, notamment :
- les avis utilisateurs avec workflow de modération,
- les signalements d'incidents,
- les journaux d'audit applicatif.

Ces données sont stockées dans des collections dédiées (reviews, incidents, audit_logs), ce qui permet une gestion plus souple, une meilleure traçabilité et une séparation claire entre données transactionnelles et données événementielles.

Répartition des données

Type de données	Technologie	Rôle
Utilisateurs, trajets	MariaDB	Données métier
Réservations, crédits	MariaDB	Transactions
Avis utilisateurs	MariaDB	Relationnel
Statistiques agrégées	MongoDB	Analyse / reporting
Données calculées (JSON)	MongoDB	Données non relationnelles

6. Journalisation et logs

L'application met en place un système de journalisation côté serveur.

Les logs sont utilisés pour : - simuler l'envoi d'emails (annulation de covoiturage), - conserver une trace des événements techniques, - faciliter le débogage en environnement de développement.

Les fichiers de logs sont stockés dans le dossier :

`backend/logs/`

Ce dossier n'est pas exposé publiquement.

En environnement de production, ce mécanisme pourrait être remplacé par une solution de journalisation centralisée ou un service externe.

7. API REST

L'application repose sur une API REST interne.

Chaque fonctionnalité métier correspond à un endpoint dédié.

Exemples d'endpoints

- `POST /api/login.php` : authentification utilisateur
- `POST /api/book.php` : réservation d'un trajet
- `POST /api/cancel_trip.php` : annulation d'un trajet
- `GET /api/admin_stats.php` : statistiques administrateur
- `GET /api/admin_credits_audit.php` : audit des crédits

Toutes les réponses de l'API sont retournées **exclusivement au format JSON**, y compris en cas d'erreur.

8. Sécurité

Les mesures de sécurité mises en place sont :

- authentification par JWT,
- stockage sécurisé des mots de passe (hashage),
- contrôle des rôles côté serveur,
- requêtes SQL préparées,
- séparation stricte frontend / backend,
- réponses API JSON uniformes.

Les messages techniques détaillés ne sont jamais exposés au frontend et sont réservés aux logs serveur.

---

9. Envoi d'emails (simulation)

Lorsqu'un conducteur annule un covoiturage, les passagers concernés sont notifiés par email.

En environnement de développement, l'envoi d'email est **simulé** via un service interne (`Mailer.php`) qui enregistre les messages dans un fichier de logs.

Cette approche permet de tester la logique métier sans dépendre d'un service SMTP externe.

En environnement de production, ce service pourrait être remplacé par une solution SMTP réelle (PHPMailer, SendGrid, etc.).

---

10. Déploiement

Le projet est conçu pour fonctionner dans un environnement local (XAMPP).

Pour un déploiement en production, les étapes suivantes seraient nécessaires : - configuration d'un serveur Apache ou Nginx, - paramétrage des variables d'environnement, - sécurisation HTTPS, - remplacement des emails simulés par un service SMTP.

Les données stockées dans MongoDB peuvent être exploitées par l'API backend afin

d'alimenter l'espace administrateur (statistiques, graphiques), sans impacter les

performances de la base relationnelle.

---

11. Conclusion

Le projet EcoRide respecte les exigences fonctionnelles et techniques de l'ECF.

Il met en œuvre une architecture claire, une gestion rigoureuse des rôles, une API sécurisée et une logique métier complète.

Les choix techniques effectués garantissent la maintenabilité, la sécurité et l'évolutivité de l'application.