

ECORIDE

DOCUMENTATION
TECHNIQUE

| | | |
|----------|--|-----------|
| 1 | INTRODUCTION..... | 3 |
| 1.1 | PRESENTATION DU PROJET ECO RIDE..... | 3 |
| 1.2 | OBJECTIF TECHNIQUE DU PROJET | 3 |
| 2 | REFLEXIONS INITIALES TECHNOLOGIQUES..... | 4 |
| 2.1 | CHOIX DES LANGAGES | 4 |
| 2.2 | TECHNOLOGIES DE DONNEES..... | 4 |
| 2.3 | CHOIX DES FRAMEWORKS/BIBLIOTHEQUES..... | 5 |
| 2.4 | ALTERNATIVES ENVISAGEES | 5 |
| 3 | ENVIRONNEMENT DE DEVELOPPEMENT..... | 6 |
| 3.1 | CONFIGURATION MATERIELLE ET LOGICIELLE | 6 |
| 3.2 | EXTENSIONS ET OUTILS UTILISES..... | 6 |
| 3.3 | ORGANISATION DES DEPOTS | 7 |
| 3.3.1 | <i>Workflow Git</i> | 7 |
| 4 | BASE DE DONNEES..... | 8 |
| 4.1 | SQL | 8 |
| 4.1.1 | <i>Modèle Conceptuel de Données</i> | 8 |
| 4.1.2 | <i>Schéma relationnel (SQL)</i> | 9 |
| 4.1.3 | <i>Exemple d'intégration de données</i> | 10 |
| 4.2 | NoSQL | 10 |
| 4.2.1 | <i>Justification de son utilisation (litiges).....</i> | 10 |
| 4.2.2 | <i>Structure de la collection</i> | 10 |
| 4.2.3 | <i>Connexion via le driver officiel</i> | 11 |
| 5 | DIAGRAMMES UML | 12 |
| 5.1 | DIAGRAMME DE CAS D'UTILISATION | 12 |
| 5.2 | DIAGRAMME DE SEQUENCE | 13 |
| 6 | DEPLOIEMENT DE L'APPLICATION..... | 14 |
| 6.1 | BACKEND : HEBERGEMENT HEROKU | 14 |
| 6.2 | FRONTEND : HEBERGEMENT VERCEL | 14 |
| 6.3 | BASE SQL : CLEVER CLOUD..... | 14 |
| 6.4 | BASE NoSQL : MONGODB ATLAS | 14 |
| 6.5 | STOCKAGE DE FICHIERS : CLOUDINARY..... | 15 |
| 6.6 | GESTION DES ENVIRONNEMENTS (.ENV)..... | 15 |
| 6.7 | PROCEDURE PAS A PAS POUR REPLIER LE DEPLOIEMENT | 16 |
| 7 | CONCLUSION..... | 17 |
| 7.1 | BILAN TECHNIQUE DU PROJET | 17 |
| 7.2 | AMELIORATIONS POSSIBLES | 17 |

1 Introduction

1.1 Présentation du projet EcoRide

EcoRide est une application web de covoiturage écologique destinée à faciliter la mise en relation entre conducteurs et passagers souhaitant partager un trajet. Elle vise à réduire l'empreinte carbone des déplacements tout en offrant une solution économique et pratique aux utilisateurs.

L'application permet de rechercher et réserver des trajets, de proposer un covoiturage en tant que conducteur, et d'évaluer l'expérience après le trajet. Des profils spécifiques existent également pour les employés (validation des avis, gestion des litiges) et les administrateurs (gestion des comptes, accès aux indicateurs).

1.2 Objectif technique du projet

Le projet EcoRide a pour ambition de proposer une solution technique complète permettant de faciliter le covoiturage écologique à travers une application web moderne, sécurisée et responsive.

Cette documentation technique présente l'architecture et les choix technologiques mis en place pour atteindre cet objectif.

Elle détaille notamment :

- *le découpage frontend / backend et les raisons de ce choix,*
- *l'environnement de développement,*
- *la modélisation des données SQL et NoSQL,*
- *les diagrammes UML,*
- *ainsi que la démarche de déploiement.*

Le document se conclut par un bilan technique tourné sur les perspectives d'amélioration.

2 Réflexions initiales technologiques

Dès le départ, l'enjeu technique a été de concevoir une architecture fiable et modulaire, capable de gérer plusieurs profils d'utilisateurs (utilisateurs, employés, administrateurs), tout en garantissant la sécurité des échanges et la maintenabilité du code.

Pour répondre à ces besoins, le projet a été découpé en deux parties distinctes :

Backend : API REST en PHP avec PDO (MySQL), MongoDB pour les litiges, et Cloudinary pour le stockage des photos de profil.

Frontend : HTML 5, CSS avec Bootstrap, SPA en JavaScript vanilla et un routeur personnalisé.

2.1 Choix des langages

- **PHP** : choisi pour le backend car maîtrisé dans le cadre de la formation DWWM, adapté aux bases SQL et facilement déployable (Heroku, Clever Cloud).
- **JavaScript** : indispensable pour le frontend (SPA, logique dynamique, appels fetch vers l'API).

2.2 Technologies de données

- **MySQL (relationnel)** : utilisé pour gérer les entités principales de l'application (utilisateurs, trajets, véhicules, réservations, avis). Ce choix garantit une structure solide et cohérente grâce aux relations entre les tables, avec des clés étrangères pour maintenir l'intégrité des données.
- **MongoDB (NoSQL)** : réservé au stockage des litiges. Son approche flexible permet de gérer des informations moins structurées (messages de suivi, statuts) sans rigidité de schéma, tout en illustrant l'usage complémentaire d'une base NoSQL.
- **Cloudinary (stockage externe)** : utilisé pour l'hébergement et la gestion des photos de profil. Ce service offre une solution sécurisée pour le stockage des fichiers, en évitant la contrainte du stockage local côté serveur. Les URLs générées sont directement exploitées par le frontend.

2.3 Choix des frameworks/bibliothèques.

- **Bootstrap 5** : adopté pour accélérer le développement de l'interface utilisateur, assurer un rendu responsive et harmoniser la présentation.
- **Routeur SPA personnalisé** : créé en JavaScript vanilla afin de gérer la navigation sans rechargement complet des pages, garantissant une meilleure fluidité.
- **Dotenv** : pour la gestion des variables d'environnement (.env).
- **PHP Mailer** : pour l'envoi de mails (notifications).
- **Cloudinary** : pour le stockage et la gestion des photos de profil des utilisateurs.
- **JWT (JSON Web Token)** : pour la gestion de l'authentification et des rôles.
- **Fetch API native** : privilégiée pour la communication avec le backend, sans recourir à des bibliothèques tierces comme Axios, afin de garder le projet simple et pédagogique.

2.4 Alternatives envisagées

Backend : L'utilisation du framework Symfony a été envisagé, bien que très pertinent pour la création d'une API backend le choix de ne pas l'utiliser afin de développer chaque partie du backend manuellement et de montrer la logique mise en œuvre et la maîtrise du backend brut.

Frontend : un framework moderne comme Angular, React ou Vue aurait pu être utilisé pour structurer l'application, mais le choix d'un routeur en JavaScript vanilla a été préféré pour bien maîtriser les fondamentaux.

3 Environnement de développement

3.1 Configuration matérielle et logicielle

Le projet EcoRide a été développé sur un Mac mini M4 Pro doté de 24 Go de RAM, sous MacOs Sequoia. Lors des déplacements professionnels sur MacBook M1 Pro MacOs Sequoia également.

L'éditeur principal a été Visual Studio Code, choisi pour sa légèreté, sa facilité de prise en main et ses nombreuses extensions.

La base de données MySQL a d'abord été hébergée en local via XAMPP, avec une gestion simplifiée grâce à phpMyAdmin pour la création et l'administration des tables. Lors du passage en préproduction, elle a été migrée sur CleverCloud, et administrée principalement via MySQL Workbench, mieux adapté à un environnement distant.

Le backend a été exécuté localement avec le serveur PHP CLI, puis testé en préproduction sur Heroku.

Le frontend a été exécuté en local avec http-server-spa, avant d'être déployé et testé en préproduction sur Vercel.

3.2 Extensions et outils utilisés

Le développement de l'application a été facilité par l'usage d'outils et d'extensions adaptés, permettant d'améliorer la qualité du code et le suivi du projet :

- **Git** : gestion des versions et organisation des branches (main, develop, preprod, feat/*, fix/*).
- **Composer** : gestionnaire de dépendances PHP (ex. PHP Mailer, JWT, Cloudinary, etc..).
- **npm** : gestion des dépendances côté frontend (ex. Bootstrap).
- **npx** : exécution ponctuelle de paquets comme http-server-spa pour lancer le frontend en local.
- **Postman** : test manuel des endpoints de l'API.
- **phpMyAdmin et MySQL Workbench** : administration des bases MySQL (local et distant).
- **Trello** : suivi et gestion de projet (méthodologie Kanban).
- **Figma** : réalisation des maquettes et de la charte graphique.

Extensions VS Code utilisées :

- **ESLint** : détection et correction des erreurs de style en JavaScript.
- **Prettier** : formatage automatique du code pour assurer une homogénéité.
- **PHP Intelephense** : autocomplétions, analyse et linting pour PHP.
- **PHP Server** : lancement rapide d'un serveur PHP local.
- **Git Graph** : visualisation des branches et de l'historique Git.

3.3 Organisation des dépôts

Le projet **EcoRide** est réparti en deux dépôts distincts afin de bien séparer les responsabilités :

- **Ecoride_back** : API REST en PHP, modèles et contrôleurs, configuration des bases de données, ainsi que les fichiers SQL (ecoride.sql et ecoride_seed.sql).
- **Ecoride_front** : application frontend, accompagnée des livrables (documentation, manuel utilisateur, charte graphique).

3.3.1 Workflow Git

Au début du projet, Git n'était pas encore complètement maîtrisé : une seule branche `develop` était utilisée, et tous les commits (fonctionnalités, correctifs, tests) y étaient directement intégrés, sans respect de conventions claires.

En cours de projet, une montée en compétence, à travers l'étude de la documentation (notamment *Conventional Commits*) et les cours suivis dans le cadre de la formation, a permis de mettre en place une organisation plus professionnelle :

- **main** : branche principale, dédiée à la production.
- **develop** : branche de développement, centralise les merges des fonctionnalités validées.
- **preprod** : branche intermédiaire pour tester avant passage en production.
- **feat/usXX-nom_fonctionnalité** : branches par fonctionnalité (issues de `develop`).
- **fix/...** : branches de correction ponctuelle.

Cette évolution illustre la progression dans l'utilisation de Git : passer d'une gestion simple et peu structurée à un workflow se rapprochant des bonnes pratiques de développement collaboratif.

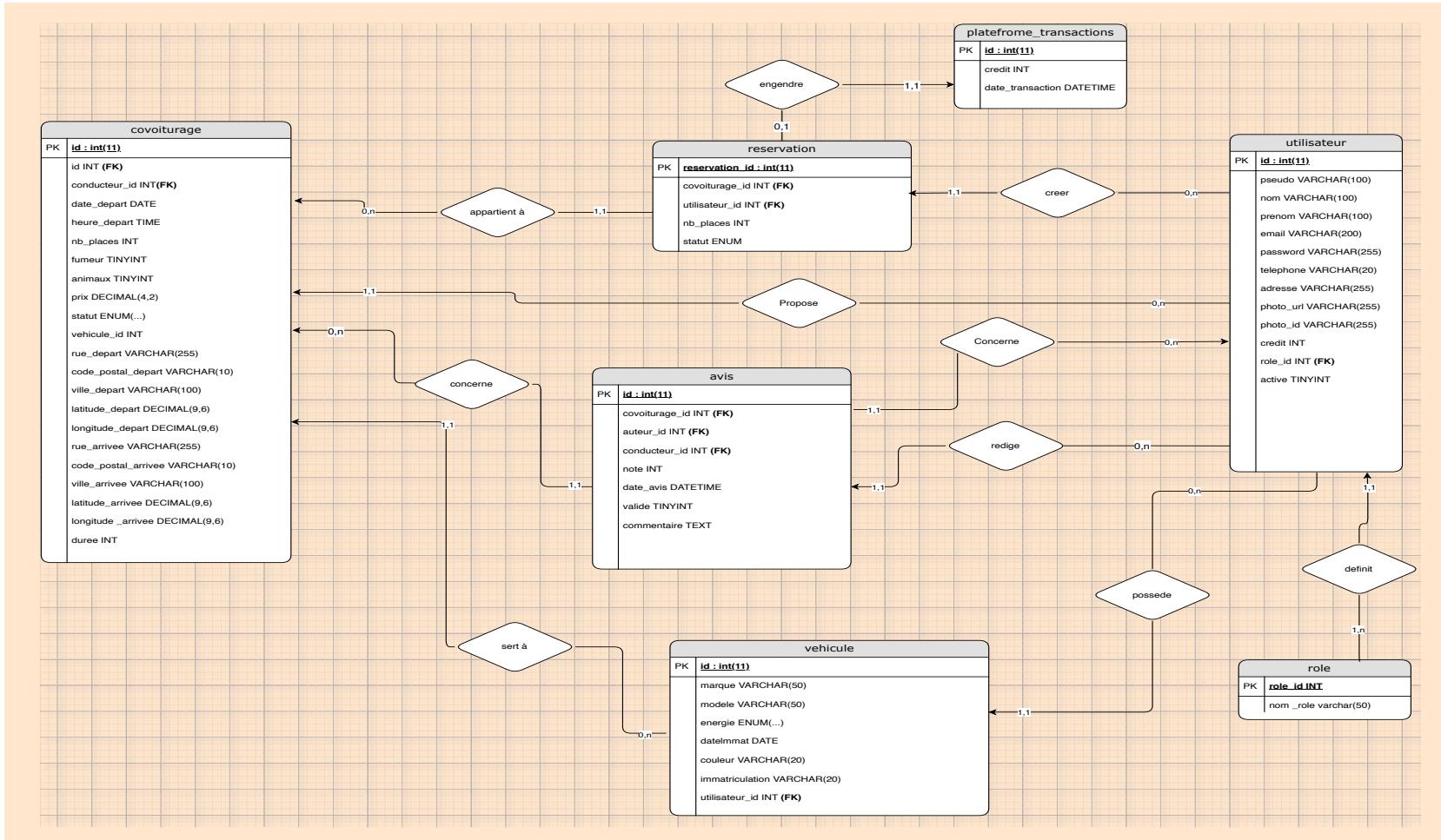
4 Base de données

4.1 SQL

4.1.1 Modèle Conceptuel de Données

Le Modèle Conceptuel de Données (MCD) présente les entités principales de l'application EcoRide (utilisateurs, covoiturages, réservations, véhicules, avis, litiges) ainsi que les relations entre elles.

Il permet de visualiser de manière la structure des données, sans considération technique, et constitue la première étape de la modélisation

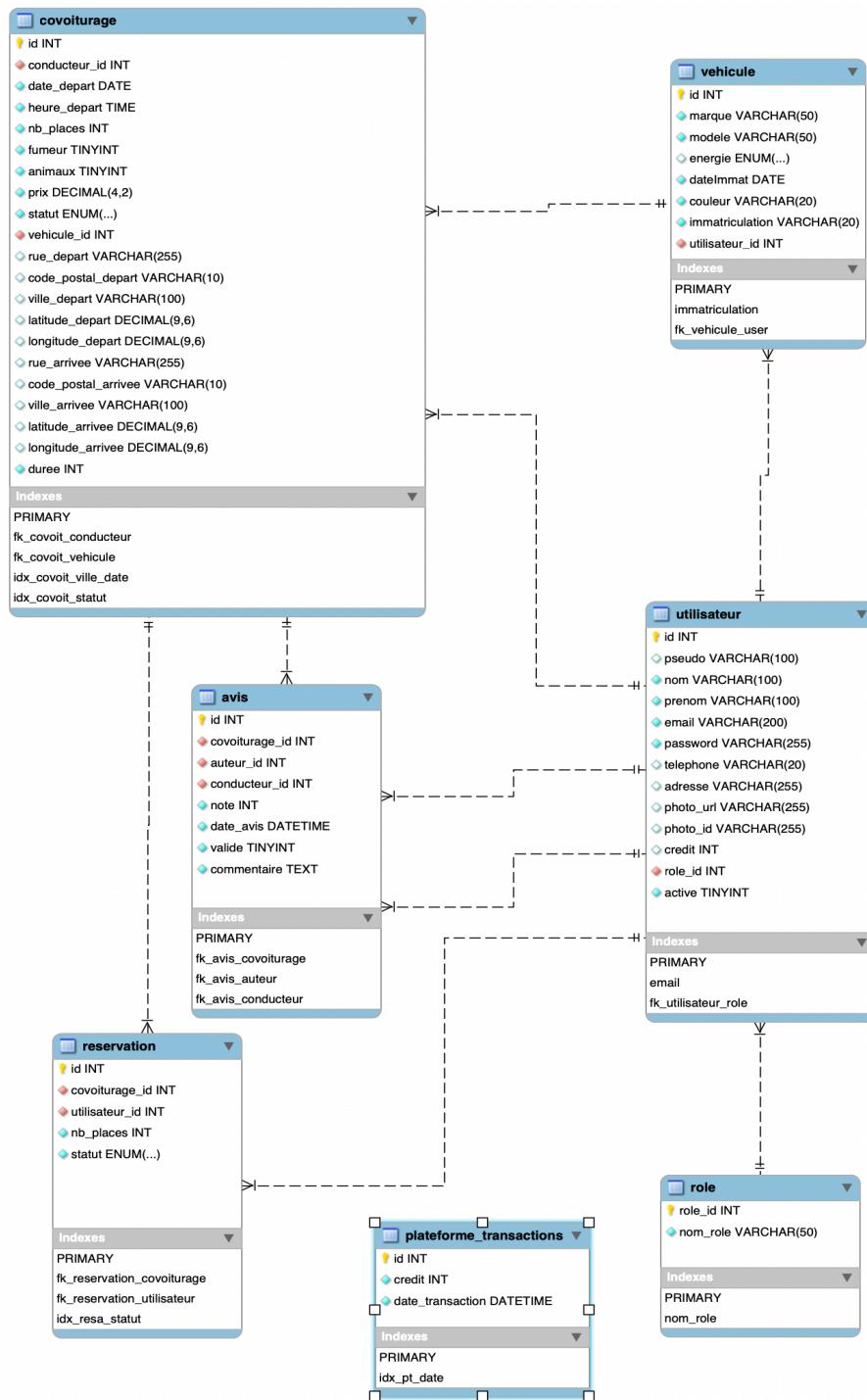


4.1.2 Schéma relationnel (SQL)

Le schéma relationnel ci-dessous est la traduction du MCD en tables SQL.

Il met en évidence les tables de la base de données MySQL, les clés primaires et étrangères, ainsi que les cardinalités implémentées.

Ce schéma reflète la structure réellement utilisée dans le projet et sert de référence pour la création et la gestion de la base.



4.1.3 Exemple d'intégration de données

Afin de faciliter les tests et la validation de l'application, un fichier de seed.sql a été préparé. Il contient un jeu de données représentatif (utilisateurs, trajets, réservations, avis...) permettant de simuler rapidement un environnement fonctionnel.

L'intégralité de ce jeu de données est disponible dans le fichier :

- /sql/ecoride_seed.sql

4.2 NoSQL

4.2.1 Justification de son utilisation (litiges)

Afin de stocker les litiges des utilisateurs, une base de données non relationnelle a été retenue. En effet, ce type de données ne nécessite pas de relations complexes avec les autres tables relationnelles (utilisateurs, trajets, réservations, etc.).

L'utilisation de MongoDB permet ainsi de gérer les litiges sous forme de documents flexibles (messages de suivi, statuts évolutifs, pièces associées), sans contrainte de schéma strict, tout en illustrant l'usage complémentaire d'une base NoSQL dans le projet. La collection litiges est organisée de la manière suivante :

4.2.2 Structure de la collection

La collection litiges est organisée de la manière suivante :

- **_id** → *Identifiant unique du litige (généré par MongoDB).*
- **reservation** → *Identifiant de la réservation concernée par le litige.*
- **redacteur** → *Identifiant de l'utilisateur ayant rédigé le litige.*
- **conducteur** → *Identifiant du conducteur impliqué.*
- **covoiturage** → *Identifiant du covoiturage associé.*
- **message** → *Description du litige faite par l'utilisateur.*
- **status** → *Statut du litige (clos, en attente, en cours de traitement).*
- **suivi** → *Tableau des notes de suivi ajoutées par les employés.*
- **createdAt** → *Date de création du litige.*
- **updatedAt** → *Date de la dernière mise à jour.*
- **closedAt** → *Date de clôture du litige (si applicable).*

4.2.3 Connexion via le driver officiel

La connexion à la base NoSQL MongoDB est réalisée à l'aide du driver officiel mongodb/mongodb installé via Composer.

Pour centraliser cette connexion, une factory a été mise en place dans le fichier config/mongo.php.

Cette classe MongoClientFactory lit les informations de connexion (URI et nom de la base) dans le fichier .env puis instancie un client MongoDB\Client. Elle renvoie ensuite une collection MongoDB prête à être utilisée par les modèles.

Extrait class MongoClientFactory :

```
<?php
require_once __DIR__ . '/../vendor/autoload.php';
use MongoDB\Client;
use MongoDB\Collection;
final class MongoClientFactory
{
    private function __construct() {}
    public static function getCollection(string $name): ?Collection
    {
        try {
            $uri      = $_ENV[ 'MONGO_URI' ] ?? null;
            $dbName   = $_ENV[ 'MONGO_DB' ] ?? 'ecoride';
            if (!$uri) {
                throw new \RuntimeException( 'MONGO_URI manquant dans .env' );
            }
            static $client = null;
            static $db     = null;

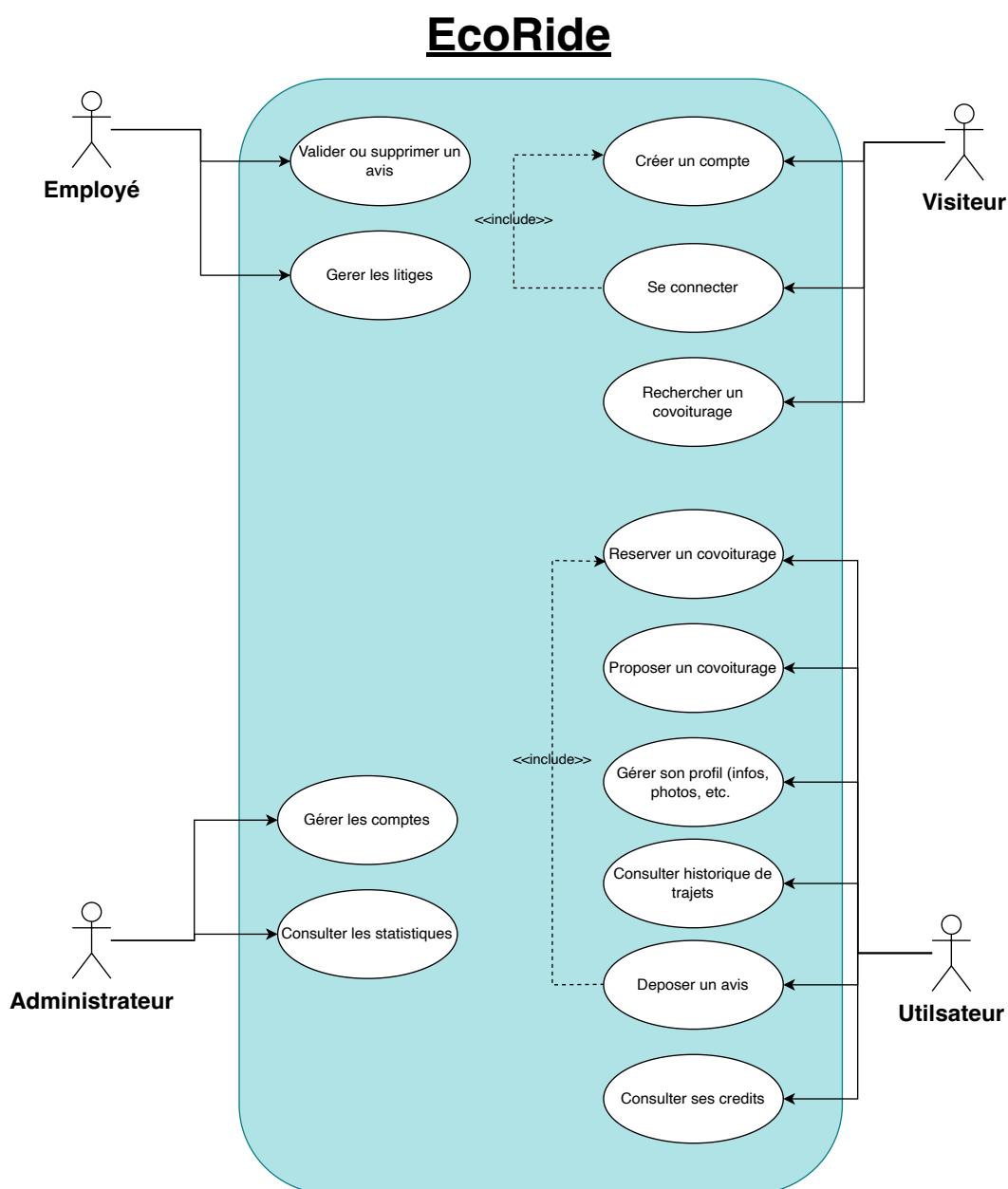
            if (!$client) {
                $client = new Client($uri);
                $db    = $client->selectDatabase($dbName);
            }
            return $db->selectCollection($name);
        } catch (\Throwable $e) {
            error_log( 'Mongo error: ' . $e->getMessage());
            return null;
        }
    }
}
```

Cette classe garantit l'uniformité de la connexion MongoDB dans tout le projet, en évitant la duplication de code.

5 Diagrammes UML

5.1 Diagramme de cas d'utilisation

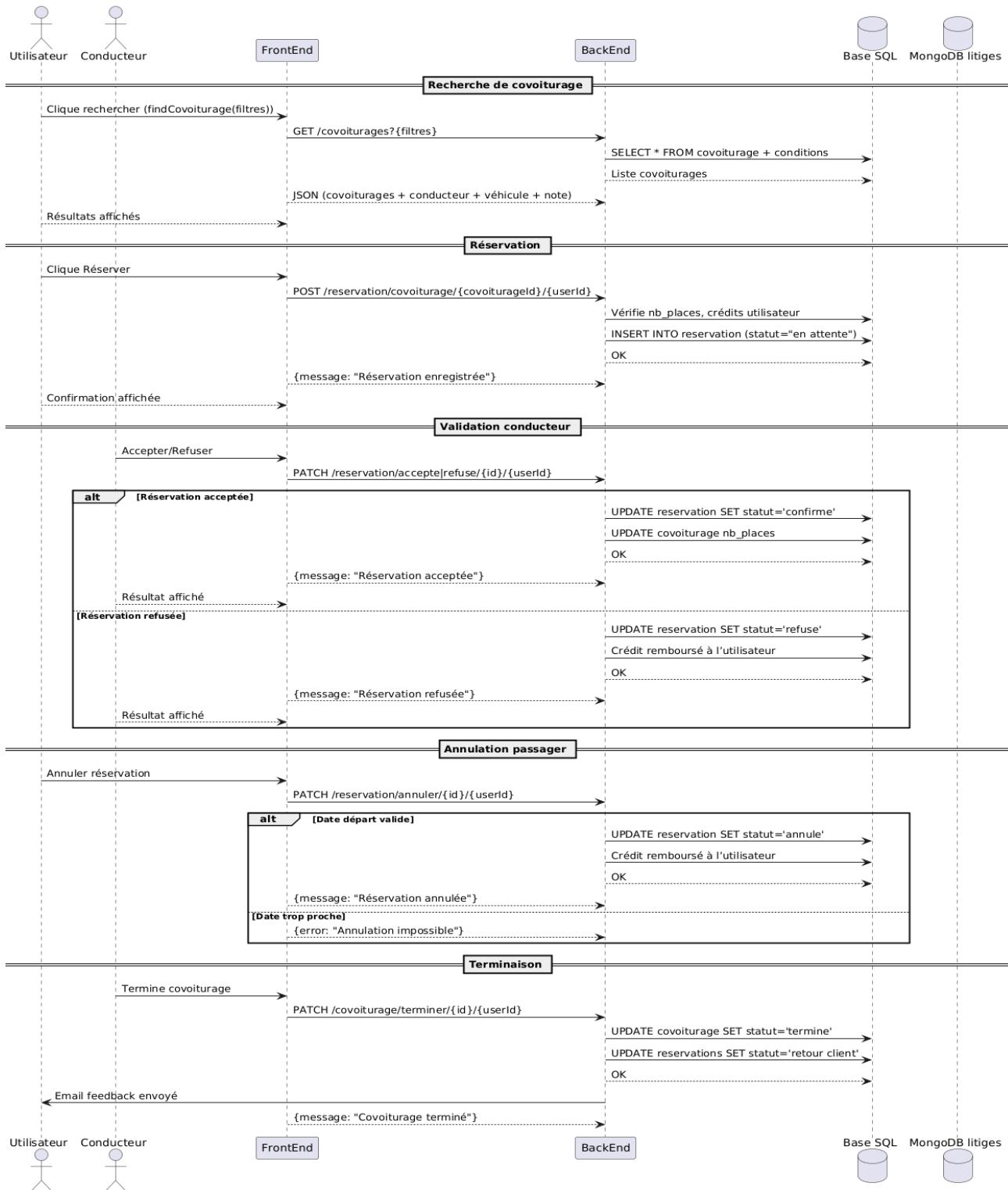
Ce diagramme illustre les actions principales des différents acteurs du système (utilisateur, visiteur, employé, administrateur) et les fonctionnalités de l'application (inscription, réservation, gestion de litiges, validation d'avis, etc.).



5.2 Diagramme de séquence

Le diagramme illustre les interactions entre passager, conducteur, FrontEnd, BackEnd et base de données lors de la recherche, réservation, validation/refus, annulation et terminaison d'un covoiturage.

Il montre les étapes clés : recherche avec filtres, réservation et contrôle des crédits/places, validation/refus par le conducteur, annulation par le passager et terminaison du covoiturage avec mise à jour des statuts et envoi d'un email.



6 Déploiement de l'application

Le déploiement de l'application EcoRide s'appuie sur plusieurs services distincts, chacun répondant à un besoin spécifique : backend, frontend, bases de données et stockage de fichiers.

6.1 Backend : hébergement Heroku

Le backend PHP a été déployé sur Heroku.

Heroku gère automatiquement l'exécution du projet grâce au fichier composer.json présent dans le dépôt.

Un fichier .htaccess est utilisé pour rediriger toutes les requêtes vers index.php, ce qui permet au routeur de l'API de fonctionner correctement.

Les informations sensibles (base de données, Cloudinary, MongoDB, clés JWT) sont stockées sous forme de variables d'environnement configurées directement dans Heroku.

6.2 Frontend : hébergement Vercel

Le frontend (HTML, CSS, JavaScript, Bootstrap) est hébergé sur Vercel.

Vercel déploie automatiquement le site à chaque mise à jour de la branche main du dépôt GitHub. La redirection côté Vercel permet d'assurer le bon fonctionnement du site en mode SPA (Single Page Application).

6.3 Base SQL : Clever Cloud

La base MySQL est hébergée sur Clever Cloud.

Initialement, le développement a été fait avec XAMPP en local (MySQL + phpMyAdmin).

En préproduction et production, la base a été migrée vers Clever Cloud.

Administration : l'outil MySQL Workbench est utilisé pour gérer les tables et les données à distance.

Les fichiers SQL (ecoride.sql et ecoride_seed.sql) permettent de recréer facilement la structure et d'injecter les données de test.

6.4 Base NoSQL : MongoDB Atlas

Les litiges sont stockés dans une base MongoDB hébergée sur MongoDB Atlas.

Connexion réalisée via le driver officiel mongodb/mongodb installé avec Composer.

Une factory (config/mongo.php) centralise la connexion et fournit des collections utilisables par les modèles.

Les variables de connexion (MONGO_URI, MONGO_DB) sont stockées dans le .env.

6.5 Stockage de fichiers : Cloudinary

Les photos de profil des utilisateurs ne sont pas stockées en local mais directement sur Cloudinary.

Intégration via le kit de développement (Software Development Kit) officiel PHP Cloudinary (installé avec Composer).

Une classe CloudinaryClient centralise la configuration et fournit des méthodes pour uploader/supprimer les images.

L'URL sécurisée renvoyée par Cloudinary est directement stockée en base de données (colonne photo_url).

Le public_id est également conservé pour permettre la suppression ou la mise à jour des images.

6.6 Gestion des environnements (.env)

Un fichier .env à la racine du projet centralise les variables sensibles :

```
DB_HOST=.....  
DB_NAME= ecoride  
DB_USER=.....  
DB_PASS=.....  
JWT_SECRET=.....  
EMAIL= .....  
PASSWORD_MAIL= .....  
MONGO_URI= .....  
MONGO_DB= .....  
CLOUDINARY_CLOUD_NAME=.....  
CLOUDINARY_API_KEY=.....  
CLOUDINARY_API_SECRET=.....
```

En production, ces valeurs ne sont pas stockées dans le code source mais définies directement dans les dashboards des hébergeurs (Heroku, Clever Cloud, Vercel).

6.7 Procédure pas à pas pour répliquer le déploiement

a) **Cloner les dépôts :**

- git clone du backend
- git clone du frontend

b) **Installer les dépendances :**

- Backend : composer install
- Frontend : npm install si besoin de Bootstrap en local

c) **Configurer les services externes :**

- Créer une base MySQL sur Clever Cloud et importer ecoride.sql (+ ecoride_seed.sql si besoin).
- Créer un cluster MongoDB Atlas et renseigner son URI dans le .env.
- Créer un compte Cloudinary et ajouter les clés d'API dans le .env.

d) **Configurer les variables d'environnement :**

- Backend (Heroku) : définir les variables dans l'interface Heroku.
- Frontend (Vercel) : si besoin, définir les URLs de l'API backend.

e) **Déployer :**

- Backend : push sur la branche main du dépôt lié à Heroku → déploiement automatique.
- Frontend : push sur la branche main du dépôt lié à Vercel → déploiement automatique.

f) **Vérification :**

- Tester les endpoints API avec Postman.
- Vérifier l'affichage du frontend en production.
- Vérifier les interactions (réservations, avis, litiges, upload photo).

Les procédures pour chaque dépôt sont également décrites en détails dans leurs README.md respectifs

7 Conclusion

7.1 Bilan technique du projet

Le projet **EcoRide** a permis de mettre en place une architecture complète et fonctionnelle, combinant :

- un **backend PHP** structuré (API REST, PDO pour MySQL, MongoDB pour les litiges, Cloudinary pour les photos),
- un **frontend JavaScript** en mode SPA avec un routeur personnalisé et une intégration responsive grâce à Bootstrap,
- une organisation Git progressive (main, develop, preprod, feat/, fix/) qui illustre la montée en compétences en cours de projet,
- une gestion claire des environnements avec .env et des services externes (Heroku, Clever Cloud, MongoDB Atlas, Vercel).

Ce projet a donc permis de valider plusieurs compétences essentielles :

- conception et modélisation (MCD, schéma SQL, diagrammes UML),
- mise en œuvre d'un backend sécurisé (authentification JWT, validation des données, protection XSS/SQL injection),
- intégration de services externes (API OSRM, API adresse.gouv, Cloudinary, MongoDB Atlas),
- déploiement sur plusieurs services (Heroku, Clever Cloud, Vercel).

7.2 Améliorations possibles

Bien que fonctionnelle, l'application peut encore évoluer techniquement pour se rapprocher des standards industriels :

- **Dockerisation** : conteneuriser le backend (PHP + MySQL + MongoDB) afin d'unifier les environnements de développement, préproduction et production.
- **CI/CD** : mettre en place une intégration continue (GitHub Actions) avec déploiement automatisé vers Heroku/Vercel après validation des tests.
- **Tests automatisés** : ajout des tests unitaires et des tests fonctionnels afin de fiabiliser les évolutions.
- **Évolutions fonctionnelles** : paiement en ligne, tableau de bord administrateur enrichi, notifications temps réel.