Documentation Technique - EcoRide

Ce document décrit les choix techniques, l'architecture, la configuration de l'environnement, le modèle de données (MCD) et le déploiement de l'application de covoiturage **EcoRide**.

1. Réflexions initiales et Choix Technologiques

Pour répondre au besoin de performance, de portabilité et de maîtrise du code, nous avons fait des choix "natifs" :

- **Langage Back-end : PHP 8 (Natif)**
  - *Pourquoi ?* Pour ne pas dépendre de la lourdeur d'un framework (Symfony/Laravel) sur un projet de cette taille, et pour démontrer une maîtrise des fondamentaux du langage (programmation orientée objet, PDO, espaces de noms).
  - C'est le choix idéal pour un projet étudiant voulant comprendre le web "à la source".
- **Base de données : MySQL / MariaDB**
  - *Pourquoi ?* Standard robuste du web, parfaitement adapté aux données relationnelles (Utilisateurs liés aux Covoiturages).
- **Front-end : HTML5 / CSS3 / JavaScript (Vanilla)**
  - *Pourquoi ?* Pour garantir un chargement rapide des pages et une compatibilité maximale sans nécessiter de compilation (npm/webpack). L'utilisation de templates PHP permet de réutiliser les blocs (`header.php`, `footer.php`).
- **Architecture : Modèle-Vue-Contrôleur (MVC)**
  - Nous avons séparé le code en trois couches pour la maintenabilité :
    - **Contrôleurs (`src/Controllers`)** : Gèrent la logique (inscriptions, recherches). C'est le "cerveau".
    - **Vues (`src/Views`)** : Gèrent l'affichage HTML. C'est la "vitrine".
    - **Configuration (`src/Config` et `src/Models`)** : Gère l'accès aux données.

2. Configuration de l'environnement de travail

Environnement Local Requis :

- **Serveur Web** : XAMPP, WAMP, MAMP ou Serveur interne PHP (`php -S`).
- **Base de Données** : MySQL 8.0+.
- **Version PHP** : 8.1 ou supérieure.
- **Composer** : Requis pour la gestion des dépendances (même minimales) et le déploiement sur Railway.

Structure du Projet :

- `/public` : Racine web (seul dossier accessible publiquement).
  - `index.php` : Point d'entrée unique (routeur).

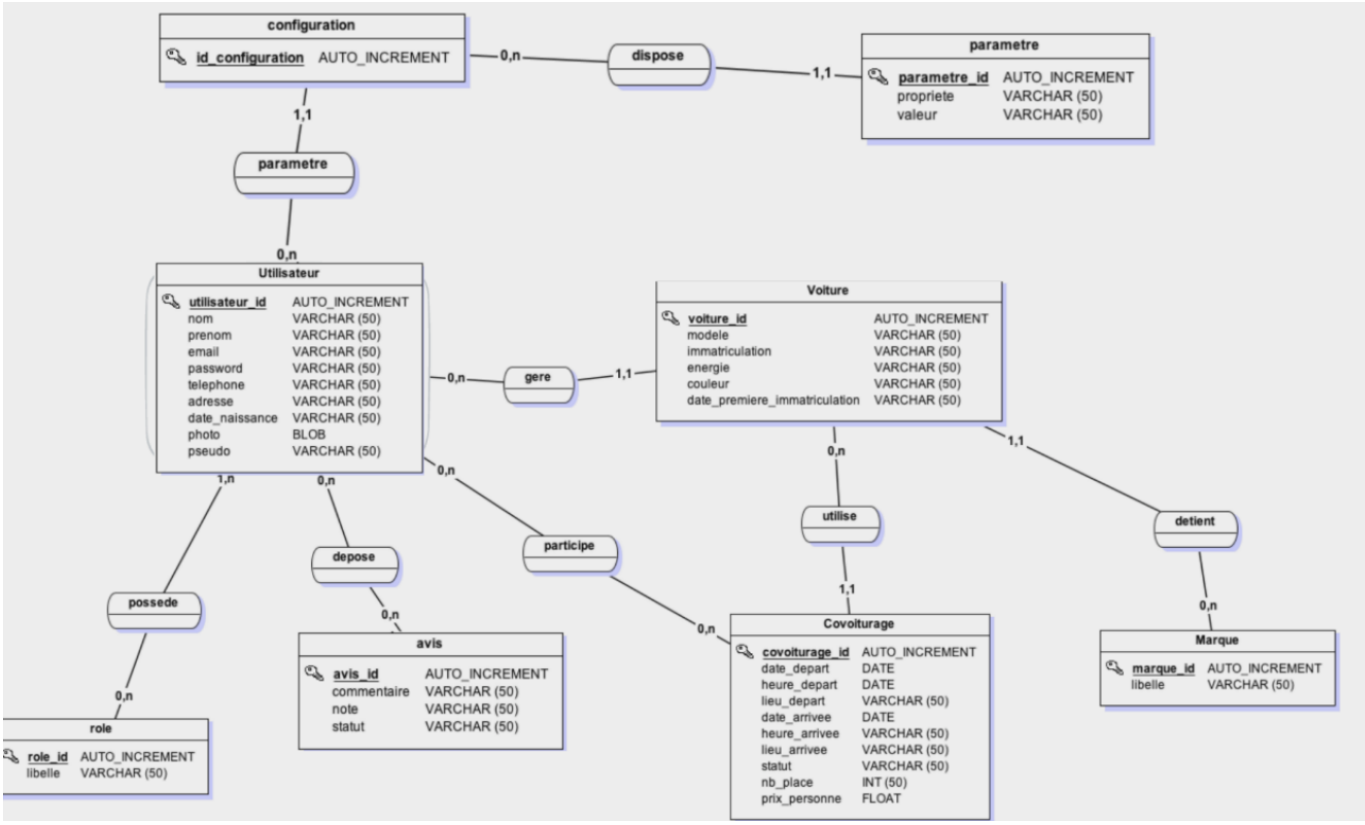
    - `assets/` : Images et CSS.
- `/src` : Code source PHP (protégé).
    - `Controllers/` : Logique métier.
    - `Views/` : Templates HTML.
    - `Config/` : Paramètres de BDD.
- `/sql` : Scripts de base de données (`schema.sql`).
- `/install` : Outils d'initialisation (`reinstall_db.php`, `seed.php`).

3. Modèle Conceptuel de Données (MCD)

La base de données `ecoride` respecte le schéma relationnel suivant (7 tables principales + paramètres) :

- **utilisateur** : Compte client (Email, Password hashé, Pseudo, Crédits, Relation vers Rôle).
- **role** : Droits d'accès (Visiteur, Utilisateur, Employé, Administrateur).
- **voiture** : Véhicules enregistrés, liés à un utilisateur et une marque.
- **marque** : Liste de référence des constructeurs auto (Tesla, Renault...).
- **covoiturage** : Le trajet central (Date départ/arrivée, Prix, Places, Conducteur, Voiture).
- **participation** : Table de liaison (Qui est passager de quel covoiturage ?).
- **avis** : Feedback entre utilisateurs après un trajet.
- **parametre** : Configuration globale du site (ex: Crédits offerts à l'inscription).

Schéma Relationnel (ERD)



4. Diagrammes (UML)

A. Diagramme de Cas d'Utilisation (Use Case)

Les acteurs principaux interagissent ainsi :

- **Visiteur** : Rechercher un trajet, S'inscrire, Se connecter.
- **Utilisateur Connecté** : Publier un trajet, S'inscrire à un trajet, Laisser un avis, Ajouter un véhicule.
- **Employé** : Valider les avis (Modération).
- **Administrateur** : Gérer les comptes (Suspendre), Gérer les crédits, voir les statistiques.

```
usecaseDiagram
    actor "Visiteur" as V
    actor "Utilisateur" as U
    actor "Administrateur" as A

    V --> (Rechercher Trajet)
    V --> (S'inscrire)

    U --> (Publier Trajet)
    U --> (Réserver Place)
    U --> (Gérer Profil/Voiture)

    A --> (Gérer Utilisateurs)
    A --> (Voir Statistiques)
```

B. Diagramme de Séquence (Exemple : Connexion)

1. L'utilisateur remplit le formulaire (Email/Mdp).
2. Le `AuthController` reçoit la demande.
3. Il interroge la `Database` pour trouver l'email.
4. Il vérifie le hash du mot de passe (`password_verify`).
5. Si OK, il crée la `$_SESSION`.
6. Il redirige vers l'accueil.

```
sequenceDiagram
    participant User as Utilisateur
    participant View as Vue (Login)
    participant Ctrl as AuthController
    participant DB as Base de Données

    User->>View: Saisit Email/Password
    View->>Ctrl: POST /login
    Ctrl->>DB: SELECT * FROM utilisateur WHERE email
    DB-->>Ctrl: Retourne User (hash)
    Ctrl->>Ctrl: password_verify()
    alt Mot de passe OK
        Ctrl-->>View: Redirection Accueil (Session Start)
    else Mot de passe Incorrect
        Ctrl-->>View: Afficher Erreur "Identifiants invalides"
    end
```