Documentation technique

1. Choix technologiques initiaux

Frontend (HTML, CSS, JavaScript Vanilla, Bootstrap) :

- *HTML/CSS* sont les standards du web, incontournables pour structurer et styliser les pages.
- *Bootstrap* car ses composants prêts à l'emploi représentent un gain de temps considérable, tout en assurant un design responsive simple à mettre en place.
- *JavaScript Vanilla* (sans framework) a été utilisé pour manipuler le DOM et construire une Single Page Application simple. Ce choix m'a permis de montrer ma compréhension des bases du langage tout en continuant à l'appréhender de mieux en mieux sans dépendre d'autres outils.

- Backend (PHP 8.2) :
  - PHP est un langage serveur populaire, facile à héberger (même gratuitement via Alwaysdata).
  - Il bénéficie d'une communauté importante et d'une bonne documentation.
  - En local, j'ai utilisé XAMPP pour héberger l'API.
  - J'ai utilisé PDO pour accéder à la base MySQL : cela apporte une abstraction et protège contre les injections SQL.
- Base de données relationnelle (MySQL/MariaDB) :
  - Ce type de base est exigé dans l'énoncé.
  - Elle convient parfaitement aux données structurées et fortement liées (utilisateurs, trajets, réservations, véhicules, etc.).
  - Les transactions ont été utilisées pour certaines opérations critiques comme la réservation de trajet ou la création d'un employé.
- Base de données NoSQL (MongoDB) :
  - MongoDB répond à une autre exigence de l'énoncé : utiliser une base NoSQL.
  - Je l'ai choisie pour enregistrer des statistiques agrégées (nombre de trajets par jour, crédits gagnés, etc.), utiles pour l'espace admin et demandées dans l'énoncé.

    - MongoDB offre de bonnes performances pour ce type de lecture et permet de stocker les données sans structure rigide.
    - J'ai utilisé MongoDB Atlas pour le déploiement (offre gratuite).
- Outils et bibliothèques complémentaires :
    - *Composer* a été utilisé pour installer la bibliothèque mongodb/mongodb, nécessaire à l'utilisation de MongoDB avec PHP.
    - *Git et GitHub* m'ont permis de versionner le code proprement.
    - *Trello* a servi à organiser les tâches du projet en suivant une méthode Kanban basée sur les User Stories.
    - *PHPMailer* m'a permis d'envoyer des e-mails de façon fiable avec gestion du SMTP (notamment pour la récupération de mot de passe ou l'annulation de trajets).

---

2. Configuration de l'environnement de développement

Pour le développement local, j'ai utilisé l'environnement suivant :

- Logiciels utilisés :
    - XAMPP (Apache, MariaDB, PHP 8.2)
    - VS Code comme éditeur de code
    - Navigateur web avec outils de développement (Chrome/Firefox)
    - Git pour le versioning
    - Composer pour gérer les dépendances PHP
    - MongoDB Community + Compass pour tester localement la base NoSQL
- Paramétrages spécifiques :
    - Création d'un VirtualHost ecoride.local pour faciliter les tests.
    - Activation manuelle des extensions PHP nécessaires (pdo_mysql, mongodb, etc.).
    - Configuration des bases de données via des scripts SQL (01_ecoride_schema.sql et 02_ecoride_data.sql).
    - Installation de la dépendance MongoDB avec Composer (mongodb/mongodb).

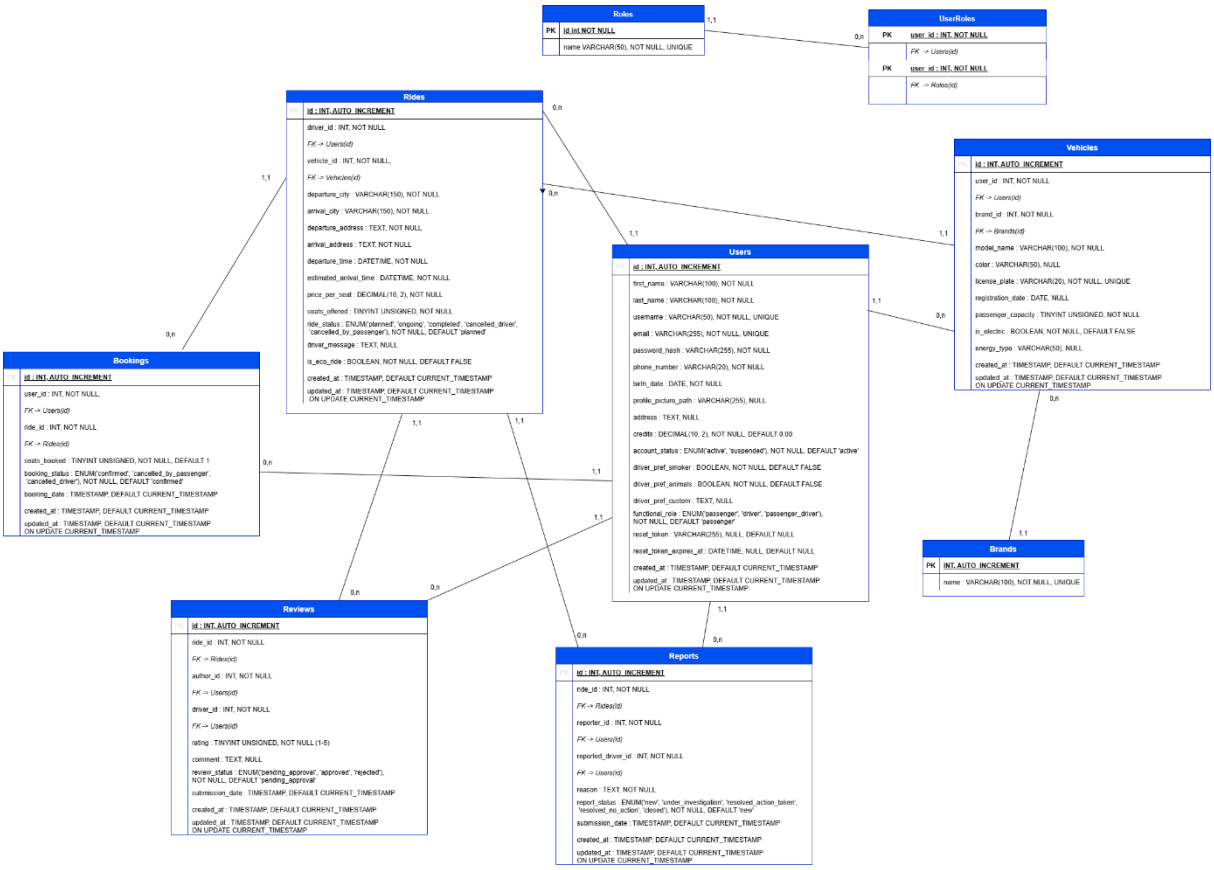
---

3. Modèle conceptuel de données

J’ai utilisé deux types de bases de données, comme demandé dans le cahier des charges : une base relationnelle MySQL et une base NoSQL MongoDB.

La base MySQL a été choisie pour gérer les informations principales de l'application. Elle permet d’organiser les données qui ont des liens clairs entre elles, comme les utilisateurs (Users), leurs véhicules (Vehicles), les trajets qu'ils proposent (Rides), les réservations (Bookings), ainsi que les avis (Reviews) et les signalements (Reports) par exemple. J’ai structuré ces données en tables reliées par des clés primaires et étrangères pour assurer la cohérence. Pour la sécurité, l'accès à MySQL depuis PHP se fait avec PDO et des requêtes préparées pour éviter les injections SQL.

En complément, MongoDB a été utilisé pour une tâche spécifique : afficher les statistiques dans le tableau de bord de l'administrateur. Je l’ai choisie pour sa facilité à enregistrer des événements (comme un trajet terminé ou un gain de crédit pour la plateforme) et à agréger ces informations pour créer les graphiques (nombre de trajets par jour, revenus quotidiens). Cela permet de garder les données d'analyse séparées des données principales de l'application et me semblait être le bon choix pour l’affichage des statistiques demandé dans l’énoncé.



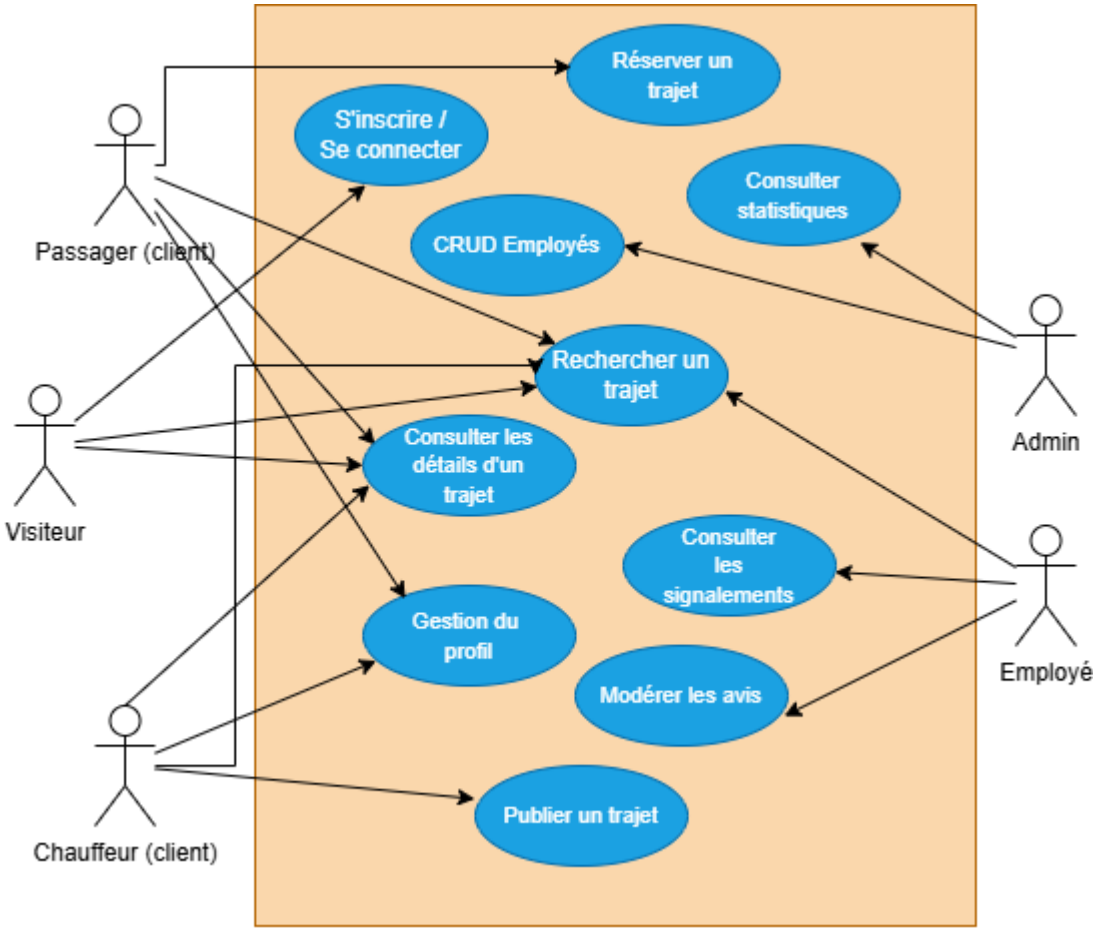
Résumé textuel des relations du schéma MCD :

- Users et Roles (via UserRoles)
    - [Users] (1,1) ---- (role pour) ---- (1,n) [UserRoles]
        - (Un User participe à 1 ou N UserRoles. Un UserRoles est pour 1 et 1 seul User.)
    - [Roles] (1,1) ---- (défini dans) ---- (0,n) [UserRoles]
        - (Un Role est dans 0 ou N UserRoles. Un UserRoles concerne 1 et 1 seul Role.)
- Users et Vehicles
    - [Users] (1,1) ---- (possède) ---- (0,n) [Vehicles]
- Brands et Vehicles
    - [Brands] (1,1) ---- (est marque de) ---- (0,n) [Vehicles]
- Users (Driver) et Rides
    - [Users] (1,1) ---- (conduit) ---- (0,n) [Rides]
- Vehicles et Rides
    - [Vehicles] (1,1) ---- (est utilisé pour) ---- (0,n) [Rides]
- Users (Passenger) et Bookings
    - [Users] (1,1) ---- (effectue) ---- (0,n) [Bookings]
- Rides et Bookings
    - [Rides] (1,1) ---- (a réservations) ---- (0,n) [Bookings]
- Reviews (en relation avec Rides, Users (auteur), Users (chauffeur))
    - [Rides] (1,1) ---- (reçoit avis via) ---- (0,n) [Reviews]
    - [Users] (auteur) (1,1) ---- (écrit) ---- (0,n) [Reviews] (Un auteur peut écrire plusieurs avis, mais un seul par trajet à cause de la contrainte UNIQUE)
    - [Users] (chauffeur) (1,1) ---- (est évalué dans) ---- (0,n) [Reviews]
- Reports (similaire à Reviews)
    - [Rides] (1,1) ---- (est signalé via) ---- (0,n) [Reports]
    - [Users] (signaleur) (1,1) ---- (fait signalement) ---- (0,n) [Reports]
    - [Users] (chauffeur signalé) (1,1) ---- (est signalé dans) ---- (0,n) [Reports]

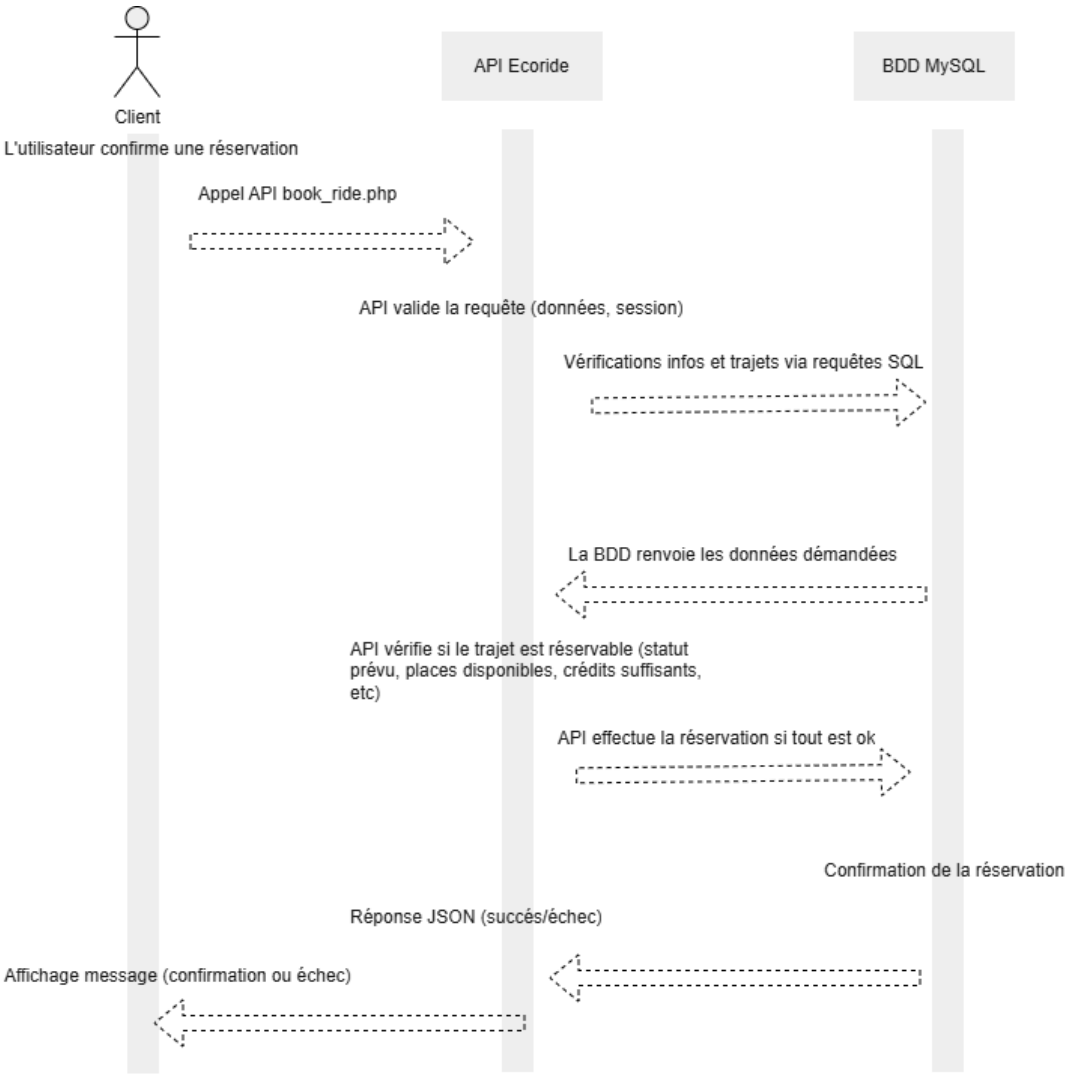
4. Diagrammes UML

- Diagramme de cas d'utilisation :
    - J'ai identifié les acteurs principaux : Visiteur, Utilisateur (Passager et Chauffeur), Employé, Administrateur.

      Concernant le rôle Passager-chauffeur, j'aurais trouvé ça logique d'accorder tous les droits passagers aux chauffeurs, qui pourraient en plus publier des trajets. Mais comme il était explicitement demandé dans l'énoncé d'avoir un rôle combinant les deux, j'ai donc créé 3 rôles clients en tout.
    - Les cas d'utilisation varient selon l'acteur : réservation, publication de trajet, modération, etc. Je n'ai pas voulu surcharger le diagramme pour qu'il reste lisible.



- Diagramme de séquence (exemple : Réservation d'un trajet) :



  - Le diagramme illustre les échanges entre :
    - Navigateur (JS)
    - API PHP
    - Base SQL (pour réserver)
  - Étapes : clic, vérification crédits/places, insertion réservation, réponse API, affichage résultat.

---

5. Documentation du déploiement

- Choix de l'hébergement :
    - Alwaysdata pour PHP/MySQL (plan gratuit, flexibilité pour l'extension PHP mongodb).
    - MongoDB Atlas pour la base NoSQL (cluster M0 gratuit).
- Configuration MongoDB Atlas :
    - Création du cluster, de l'utilisateur de base de données, configuration de l'accès réseau (IPs), récupération de la chaîne de connexion.
- Configuration Alwaysdata :
    - Création du site PHP (cmdecoride.alwaysdata.net).
    - Création de la base de données MySQL et de son utilisateur (note des identifiants : hôte, nom BDD, user, pass).
    - Activation de l'extension PHP mongodb : explication de l'utilisation de ad_install_pecl mongodb en SSH et de l'ajout de la ligne extension=/home/cmdecoride/mongodb8.2.so dans le php.ini personnalisé via le panel.
    - Activation du HTTPS.
- Préparation du code pour la production :
    - Système settings.php / settings.prod.php pour gérer les configurations (BDD, MongoDB, SMTP, URL de base, CORS).
    - Chemins relatifs pour les Workspace JS (/api/...).
    - Nettoyage des console.log.
    - Fichier .gitignore pour settings.prod.php et vendor/.
- Upload des fichiers :
    - Utilisation de FileZilla pour transférer le contenu du dossier Code/ (y compris vendor/ et lib/) vers /www/ sur Alwaysdata.
    - Upload manuel de settings.prod.php dans /www/api/config/ avec les identifiants de production.
- Initialisation de la base de données MySQL de production :
    - Connexion à phpMyAdmin sur Alwaysdata.

    - Exécution des scripts 01_ecoride_schema.sql et 02_ecoride_data.sql.
- Tests post-déploiement : Vérification des fonctionnalités sur l'URL de production.