Bienvenu sur ecoriDE 🍁

EcoRide a pour mission de réduire l’impact environnemental des trajets grâce au covoiturage.

Favre-Bonvin Yoann

# RESUME DU PROJET

**EcoRide** est une jeune startup française fondée avec une ambition claire : **réduire l’impact environnemental des déplacements quotidiens** en promouvant une solution de **covoiturage écologique**. Consciente des enjeux climatiques actuels, l’entreprise souhaite offrir une **alternative durable et économique** à l’usage individuel de la voiture.

La problématique centrale à laquelle EcoRide souhaite répondre est :

* **Comment rendre le covoiturage plus accessible, sécurisé et attractif pour les usagers soucieux de l’environnement ?**

Pour répondre à cette problématique, EcoRide a exprimé le besoin de **développer une application web moderne**, intuitive et éco-conçue, permettant de :

* Rechercher, proposer et réserver des trajets en covoiturage. Offrir une expérience fluide tant pour les passagers que les conducteurs.
* Filtrer les trajets selon des critères environnementaux (véhicule électrique principalement)
* Gérer les profils utilisateurs, les crédits, et la participation aux trajets.
* Mettre en avant une **identité visuelle alignée avec les valeurs écologiques de la marque** (couleurs, icônes, ergonomie).

Le projet est piloté par **José**, directeur technique, qui assure la supervision fonctionnelle et technique. Il souhaite une plateforme robuste, en prenant en compte les besoins fonctionnels détaillés (interfaces, filtres, espace utilisateur, espace administrateur, etc.).

Solution mise en œuvre :

* **Frontend** développé en HTML/CSS/JS avec Bootstrap, en responsive design.
* **Backend** en PHP structuré en **programmation orientée objet (POO)**, séparé en contrôleurs, services et repositories.
* Base **relationnelle (MySQL)** pour les trajets, utilisateurs, avis…
* Base **NoSQL (MongoDB)** pour enregistrer les logs de recherche.

# INTRODUCTION

Ce projet, nommé **EcoRide**, s’inscrit dans le cadre de ma reconversion professionnelle vers le développement web. Après près de vingt années passées dans le secteur du commerce, j’ai fait le choix de me tourner vers un métier plus technique, en lien avec mes valeurs, notamment l’écologie et le numérique responsable.

**EcoRide** est le premier projet complet que j’ai mené dans ce nouveau domaine. Il m’a permis de mettre en pratique l’ensemble des compétences acquises durant ma formation, que ce soit sur le plan technique, organisationnel, ou encore graphique.

Ce projet revêt donc une importance particulière : il marque le passage entre un ancien parcours professionnel et un nouveau métier dans lequel je souhaite m’investir à long terme.

Le projet répond à une problématique réelle : comment réduire l’impact environnemental des trajets du quotidien grâce à une solution numérique accessible et durable ?

L’enjeu n’était pas seulement de créer un site fonctionnel, mais de concevoir une plateforme cohérente, intuitive, sécurisée et éco-responsable, tant dans sa conception que dans sa finalité.

En tant que débutant, j’ai naturellement rencontré plusieurs obstacles, notamment :

* **La gestion du temps et des priorités** : sans méthode au départ, j’ai souvent travaillé dans le désordre, ce qui a conduit à des pertes de temps évitables.
* **Le manque d’organisation en phase de démarrage** : l'absence de plan clair a entraîné des allers-retours inutiles entre la conception et le développement.
* **L’adaptation à un nouvel environnement technique** : j’ai dû apprendre à utiliser des outils comme Docker, Composer, GitHub, ou encore MongoDB, parfois de manière autodidacte.

# ANALYSE

**Lecture du cahier des charges**

Dès réception du brief projet, ma première étape a été de comprendre le besoin exprimé par le directeur technique « José ». L’objectif était clair : créer une plateforme de covoiturage orientée écologie, permettant de :

* Proposer et rechercher des trajets,
* Réserver des places en tant que passager,
* Promouvoir les véhicules à faibles émissions (électriques notamment),
* Offrir une interface fluide, moderne et responsive.

**Comment ai-je abordé ce projet ?**

J’ai commencé par identifier les différents profils d’utilisateurs, afin d’adapter l’ergonomie et les fonctionnalités aux usages concrets. Les trois personas principaux sont :

* **Conducteur écoresponsable** : souhaite proposer des trajets, valoriser son véhicule électrique, et arrondir ses fins de mois.
* **Passager soucieux de l’environnement** : cherche une alternative à la voiture individuelle, sans sacrifier confort ou sécurité.
* **Administrateur** : supervise l’activité, modère les contenus, vérifie les véhicules.

Le diagramme de cas d’utilisation (annexe 1) m’a permis de visualiser les interactions entre les utilisateurs et le système. Ce schéma a été fondamental pour ne rien oublier dans la conception

**Comment ai-je convertie les besoins « non techniques » en « techniques » ?**

Pour transformer les besoins métiers en une solution technique exploitable, j’ai mobilisé plusieurs outils d’analyse :

* Diagramme de classe (Annexe 2)

Le diagramme de classe a permis de :

* Déterminer les **entités principales** : Utilisateur, Trajet, Véhicule, Avis, Participation.
* Définir les **relations entre les tables**.
* Structurer la future base de données relationnelle.
* Diagramme de séquence (Annexe 2 bis)

J’ai également modélisé **le processus d'une recherche de trajet**, du clic de l’utilisateur jusqu’à l’affichage des résultats. Ce diagramme a été utile pour :

* Visualiser les **étapes techniques** (requêtes, réponses),
* Préparer l’**interaction AJAX** entre le front et le back.
* Schéma MVC (Annexe 2 ter)

Enfin, le schéma MVC m’a permis de poser une base pour :

* **Organiser le code** de manière modulaire (Controller, Service, Repository),
* Assurer une **maintenabilité** et **extensibilité** du projet,
* Respecter les **bonnes pratiques de séparation des responsabilités**.

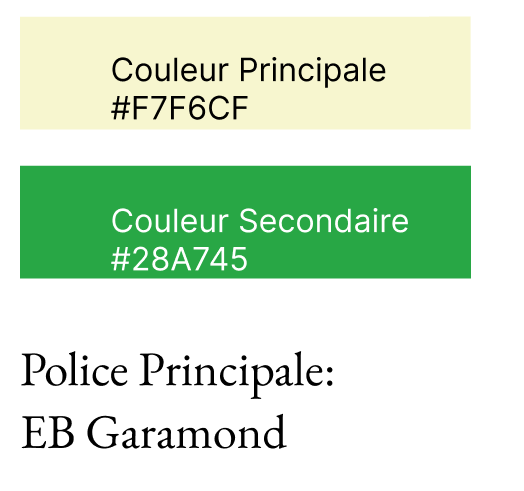
Une fois ces étapes faites, il est temps de se plonger dans la conception de charte graphique puis par la suite les wireframes et finir par les mockups. Le tout en version desktop et mobile (responsive).

* Comment ai-je défini ma charte graphique ?

Pour répondre à cette question, il faut reprendre la description du projet par le directeur technique. Afin de se différencier de ces concurrents (blablacar par exemple), il souhaite avoir une approche écologique du covoiturage en mettant en avant les trajets écologiques (voitures électriques 🌿 Éco).

Je suis donc partie sur les axes principaux qui définissent le caractère « écologique » dans la société, à savoir : les couleurs et la typographie.

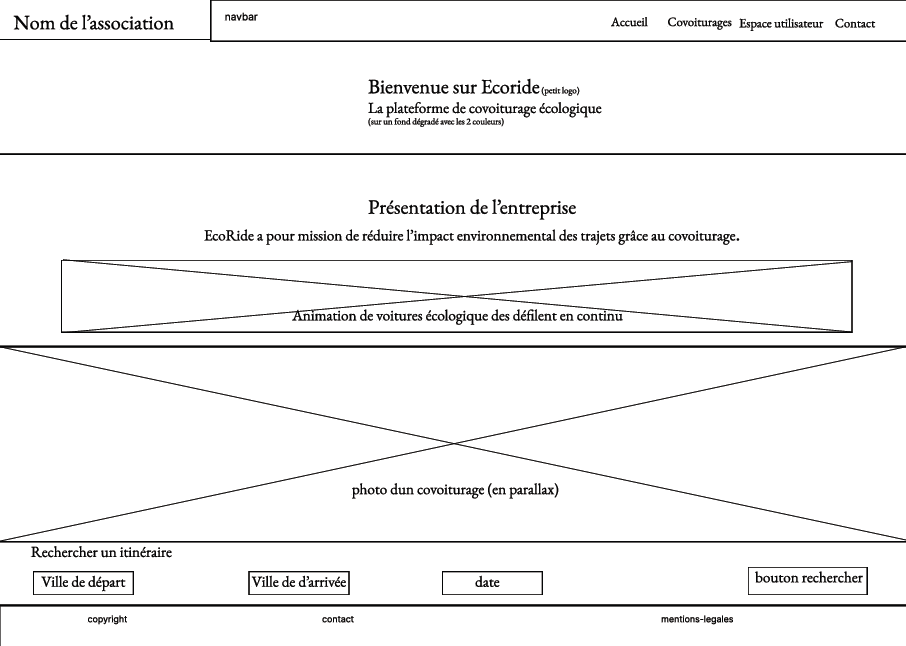
* Pour les couleurs, l’écologie est très souvent associée à la couleur verte (qui rappelle la nature), le beige et le marron : j’ai décidé d’utiliser principalement le vert et le beige pour des soucis de contraste (notamment pour les daltoniens qui peuvent être gênés avec le marron et le vert), j’ai quand même mis quelques touches de marron à certains endroits. L’avantage aussi du beige est que cela permet de moins consommer d’énergie sur les écrans
* Pour la typographie, je suis partie sur le GARAMOND qui permet de réduire de 24% la quantité d’encre (en cas d’impression).



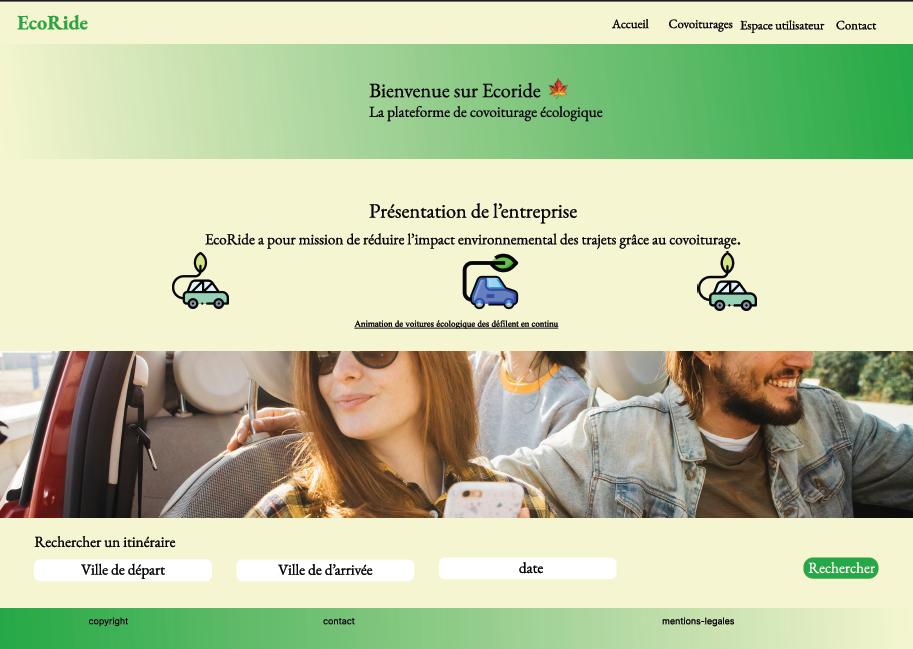
* Partie maquettage au format desktop et responsive (mobile) est à retrouver en intégralité en annexe.

Pour cette partie, il fallait répondre aux cahiers des charges fournis par le directeur technique, à savoir :

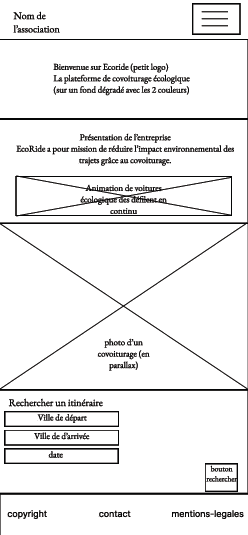
* La page d’accueil :
* une navbar avec les éléments suivants « accueil », « covoiturage », « connexion » (que j’ai renommé « espace utilisateur » qui me paraissait plus explicite) et « contact ».
* une présentation rapide de l’entreprise avec des photos.
* Une barre de recherche d’un itinéraire
* Un footer avec le mail et un lien vers les mentions légales.



wireframe accueil



mockup accueil



wireframe responsive accueil

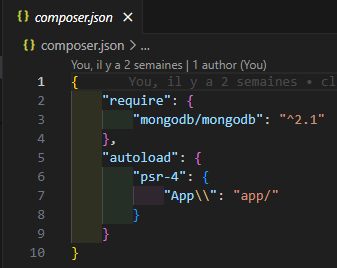
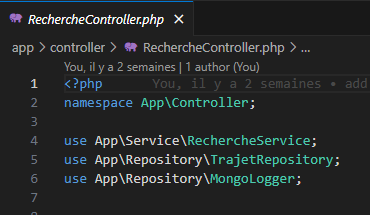
mockup responsive accueil



# CONCEPTION

Pour la réalisation de ce projet, il a fallu définir l’environnement de travail, à savoir :

* Les outils de développement :
* Visual Studio Code version 1.99.3 : éditeur de code léger et gratuit. Il permet l’utilisation de plusieurs langages (HTML, CSS, JS, PHP, …). Il est équipé d’une interface moderne avec auto-complétion, debug et terminal intégré. Il est léger et rapide et se lie facile avec le repos GitHub. (Voir annexe n°4)
* GitHub : Suivi de version du code. Dépôt hébergé sur GitHub, branches pour le développement, commits réguliers. (Voir annexe n°5)
* Composer version 2.X : Gestionnaire de dépendances PHP pour autoloader les classes (PSR-4 : plus besoin de « require »)



* Docker version 24.X : Conteneurisation de l’environnement pour faciliter le déploiement
* Heroku : Hébergement du site relié au repo GitHub.
* Les outils techniques pour la partie Front-end :
* HTML / CSS : Pour la structure et le style des pages (Voir annexe n°4)
* Bootstrap 5.X : Framework CSS pour accélérer la création d’une interface responsive (Voir annexe n°4)
* JavaScript (Vanilla) version ES6+ : Pour dynamiser le front avec des appels AJAX (Fetch API) sans rechargement de page.
* AOS.js : pour les animations d’apparition au scroll.
* Les outils techniques pour la partie Back-end :
* PHP version 8.X : Langage côté serveur principal, utilisé pour gérer la logique métier et orientée objet (POO) (Voir annexe n°6).
* Architecture MVC custom : séparation stricte des responsabilités :
* Controllers/ : gèrent les requêtes HTTP.
* Services/ : contiennent la logique métier.
* Repositories/ : encapsulent l’accès aux bases de données.
* Autoload PSR-4 via Composer pour charger dynamiquement les classes PHP
* Les outils pour la Base De Données :
* MySQL version 8.X : Base de données relationnelle pour stocker les utilisateurs et les trajets
* MongoDB version 7.X : Base NoSQL pour enregistrer les recherches des utilisateurs (log / veille)
* Conteneurisation : (voir annexe n°10 et 10bis)

L’ensemble du projet est conteneurisé avec Docker :

* Un service php-apache : exécute le serveur web avec PHP intégré.
* Un service mysql avec initialisation automatique de la base *(init.sql)*
* Un service mongo pour les logs des recherches utilisateurs NoSQL qui les stocke
* Le fichier *docker-compose.yml* définit l’ensemble de la configuration, garantissant : une mise en place en un seul *docker-compose up* ; aucune dépendance externe nécessaire sur la machine hôte ; un environnement identique en local et en production.

Il suffit, en « invite de commande », de démarrer le logiciel par le biais du code : « docker-compose up –build ».

Une image contenant capture d’écran, texte

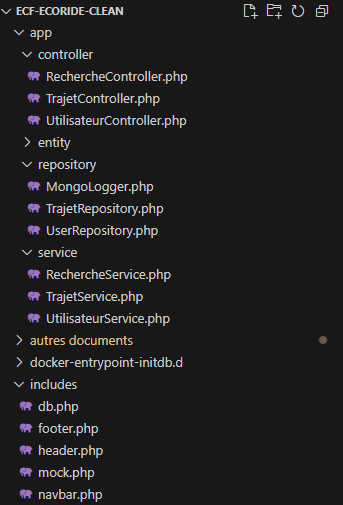
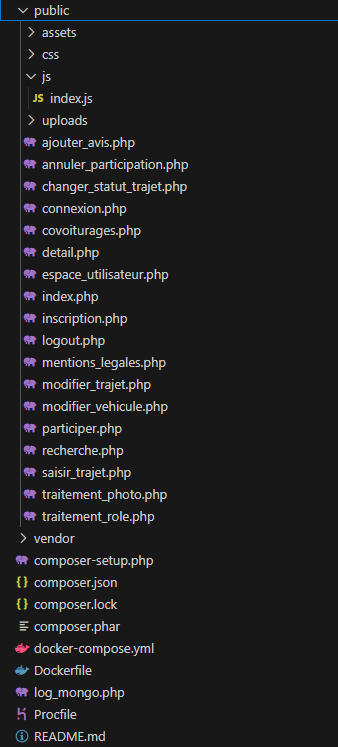
Le contenu généré par l’IA peut être incorrect.

Cela garantit une portabilité du projet sur tout environnement compatible Docker, sans conflits de version ou dépendances.

Docker permet de créer un environnement complet (serveur web, base de données, etc.) **isolé, portable et cohérent**. Grâce à la commande docker-compose up, tout se lance automatiquement sans configuration manuelle locale. Cela garantit un **environnement de développement identique en local et en production.**

* Structure du projet :

L’arborescence du projet est pensée pour la lisibilité et la maintenabilité :



En conclusion, la conception technique s’est appuyée sur plusieurs livrables préparatoires :

* Schéma MVC : organisation du code ne couches (Controller, Service, Repository)
* Diagramme de classes : définition des entités principales et de leurs attributs
* MCD (modèle conceptuel de données) : modélisation de la base MySQL
* Diagramme de séquence : représentation du flux d’une recherche de trajet
* Wireframes et Mockups : définition de l’expérience utilisateur sur desktop et mobile.

# DEVELOPPEMENT

3.1 Données – Modélisation, scripts et jeux de données

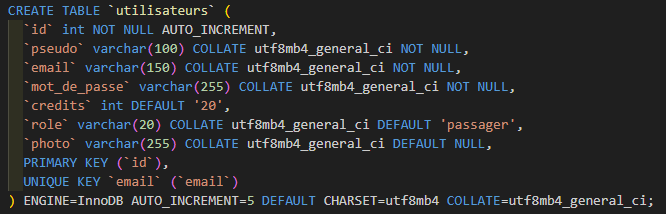
* Mise en place de la base de données relationnelle (MySQL) (annexe n°3)

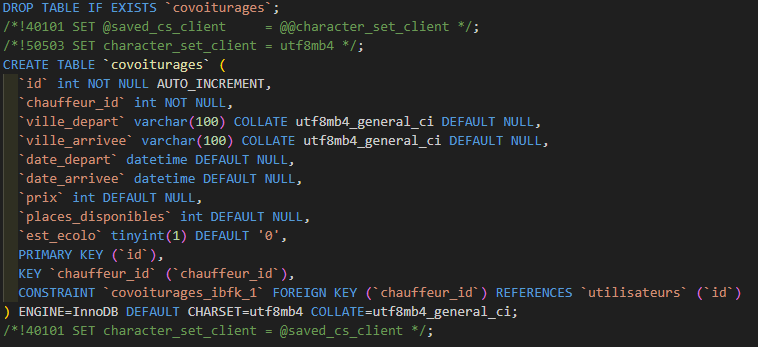
L’application repose sur une base de données relationnelle MySQL nommée EcoRide, contenant les tables suivantes nécessaire à la gestion de l’application :

* AVIS : enregistre les retours des passagers sur les trajets
* PARTICIPATIONS : permet d’enregistrer les demandes de réservation
* TRAJETS : représente les trajets proposés ou réservés
* UTILISATEURS : stocke les informations des utilisateurs inscrits
* VEHICULES : permet de stocker les différents véhicules

La création des tables MySQL a été automatisée grâce à un fichier SQL appelé *init.sql*, conçu manuellement au préalable.

Ce fichier contient toutes les instructions nécessaires à la construction de la base, par exemple :





Une fois ce script prêt, il est placé dans un dossier particulier reconnu automatiquement par le conteneur MySQL utilisé dans Docker :

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Lors du 1er démarrage du conteneur, Docker exécute ce fichier et crée la base, les tables, et éventuellement les données de test, sans aucune action manuelle. Cela permet de gagner du temps, d’éviter les erreurs et de garantir que chaque environnement (dev, prod, test) démarre avec une structure commune.

* Jeu de données utilisateur (test)

Pour tester l’application, des utilisateurs fictifs ont été insérés automatiquement via la formule suivante :

INSERT INTO utilisateurs (pseudo, email, mot\_de\_passe, crédit, rôle, photo) VALUES



* Intégration de MongoDB (NoSQL)

Une base MongoDB complémentaire a été intégrée au projet pour enregistrer les recherches effectuées par les utilisateurs, sous forme de documents JSON.

Chaque recherche utilisateur est stockée avec l’IP, la date et les termes de recherche :

Une image contenant texte, capture d’écran, logiciel

Le contenu généré par l’IA peut être incorrect.

* Objectifs atteints grâce à cette architecture
* Organisation claire et normalisée des données relationnelles
* Enregistrement des logs non structurés via MongoDB
* Complémentarité SQL + NoSQL dans une même application, via Docker

3.2 Développement Front-end / Back-end

* Front-end – Interface utilisateur

Le front-end a été conçu avec une attention particulière portée à la performance, à l’ergonomie et à l’accessibilité. Il a également été conçu en respectant l’expérience utilisateur que ce soit sur desktop ou mobile.

Technologies utilisées :

* **HTML5 / CSS3** pour la structure et la mise en forme. (annexe n°4)
* Une image contenant texte, capture d’écran, affichage

  Le contenu généré par l’IA peut être incorrect.**JavaScript Vanilla (ES6)** pour les interactions dynamiques. UN fichier « index.js » créé et intégré dans « index.php » (ex : page d’accueil, les voitures défilent en continu de gauche à droite)

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

* **Bootstrap 5** pour un design responsive adapté desktop / mobile. (annexe n°4)
* **AOS.js** pour des animations légères (fade, slide...). (ex : page d’accueil, le texte sur la présentation de l’entreprise apparait comme si on écrivait à la machine à écrire) également intégré dans le fichier « index.php »

****

Principales caractéristiques :

* **Barre de navigation fixe** avec pages publiques (accueil, inscription, connexion)
* **Formulaire de recherche** dynamique via **JavaScript + AJAX (Fetch)** sans rechargement de la page.
* **Barre de recherche et affichage** des trajets disponibles.
* Design en lien avec la thématique **écoresponsable** (vert, beige, icône voiture électrique)



* Communication asynchrone (AJAX) (voir annexe n°7, 7bis et 7ter) :

Le formulaire de recherche utilise **Fetch API** pour envoyer les données au serveur et afficher les résultats dynamiquement :

1. L’utilisateur renseigne départ/arrivée/date.
2. Le JS intercepte l’événement *submit*, empêche le rechargement.
3. Une requête POST est envoyée à *recherche.php.*
4. Le serveur renvoie un JSON contenant les trajets disponibles.
5. Le JS les affiche dans le DOM.

Cela garantit une expérience fluide et moderne, sans latence ni rafraichissement inutile de la page.

* Back-end – Programmation orientée objet (POO)

L’organisation suit une architecture MVC **Controller – Service – Repository** :

* Controller : Reçoit les requêtes, déclenche les actions à effectuer
* Service : Contient la logique métier (ex : validation, traitement)
* Repository : Gère les requêtes SQL et l’accès aux base de données.

Cette architecture facilite la maintenance, les tests et la réutilisabilité du code.

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

3.3 Sécurité

* Authentification sécurisée

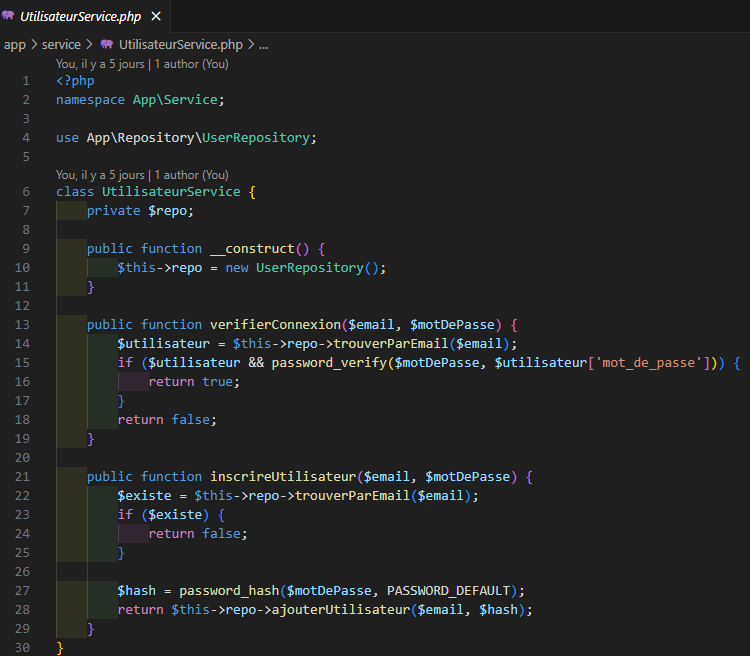
L’application gère les connexions via des sessions PHP. Lorsqu’un utilisateur s’authentifie :

* La session est ouverte avec *session\_start()* dès le début du script.
* L’identifiant utilisateur est enregistré dans *$\_SESSION['user\_id'],* ce qui permet de sécuriser l’accès aux pages privées et il est conditionné à l’existence de cette connexion.
* Pour renforcer la sécurité sur les sessions, on pourrait rajouter un « timeout » d’inactivité ce qui permettrait de stopper la session au bout d’un certain laps de temps, très utile surtout en version mobile où l’on ne ferme pas systématiquement nos pages d’application.

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

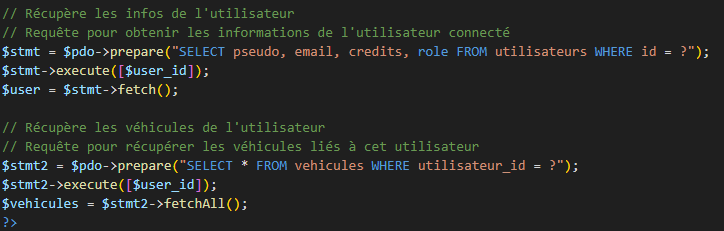
* Les mots de passe sont **hachés avec *password\_hash()*** lors de l'inscription. Ils ne sont donc jamais stockés en clair avec cette fonction native de PHP. Ce qui garantit en cas de vol de la base de donnée que les données sont difficilement récupérables
* Lors de la connexion, la vérification se fait avec *password\_verify().*



Ces pratiques évitent le stockage de mots de passe en clair et renforcent la sécurité contre les attaques.

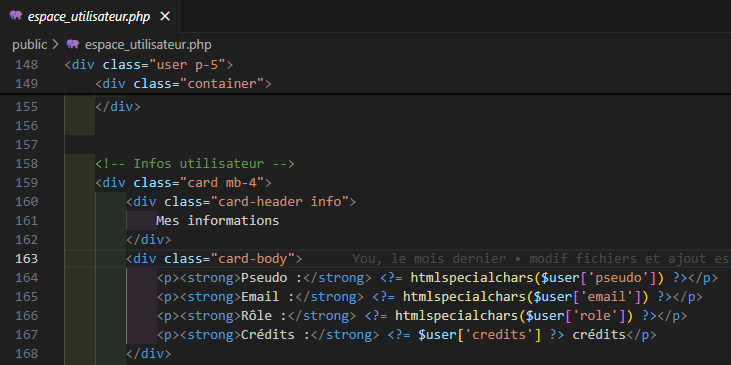
* Protection contre les injections SQL

L’accès à la base de données utilise PDO avec des requêtes préparées :



* Validation des données
* Les formulaires HTML valident les champs côté client *(ex : Required, type=email)*
* Le back-end vérifie également les données avant de les utiliser : il vérifie les champs vides, filtre avec *htmlspecialchars()* pour éviter les attaques XSS, vérifie du format des emails.
* Les mots de passe et configurations ne sont pas exposés directement dans les scripts publics.

Tout ceci contribue à conserver des données propres et évite les attaques les plus fréquentes.



Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.



* Sécurité MongoDB

Les interactions avec MongoDB sont limitées à : des insertions (*insertOne()*) et des tests de lecture sécurisés (*find()*). Les logs MongoDB ne contiennent aucune donnée sensible.

* Afin de sécuriser encore l’application, nous pourrions rajouter l’utilisation de « TOKEN CSRF », cela serait très utile dans ce genre de projet. Voici les explications :

a. Comprendre la menace

Une attaque CSRF (*Cross-Site Request Forgery*) consiste à tromper un utilisateur connecté pour qu’il exécute à son insu une action non désirée sur le site (ex. : réserver un trajet, modifier son profil…).

Exemple : un lien malveillant déclenche une requête sur EcoRide pendant que la session est active. Le serveur ne sait pas que ce n’est pas une action volontaire de l’utilisateur.

b. Mise en place d’un token CSRF

Pour protéger l’application, j’ai prévu l’intégration d’un token CSRF, c’est-à-dire une clé secrète ajoutée à chaque formulaire sensible.

Fonctionnement prévu :

1. À l’ouverture d’un formulaire (dans un fichier PHP), un token unique est généré et stocké en session :

*$\_SESSION['csrf\_token'] = bin2hex(random\_bytes(32));*

1. Ce token est injecté dans un champ caché (dans le fichier HTML) :

*<input type="hidden" name="csrf\_token" value="<?= $\_SESSION['csrf\_token'] ?>">*

1. Lors de la soumission du formulaire (dans le fichier PHP), le serveur vérifie que le token reçu correspond à celui de la session :

*if ($\_POST['csrf\_token'] !== $\_SESSION['csrf\_token']) {*

*die("Requête non autorisée.");*

*}*

c. Résultat

* Cela empêche toute soumission externe frauduleuse, car l’attaquant ne peut pas connaître le token unique généré côté serveur.
* Cette mesure renforce la sécurité de l’application, notamment pour les formulaires sensibles (connexion, modification de profil, réservation…).

TESTS, DEPLOIEMENT ET BILAN

4.1 Test & Validation :

Même si le projet a été mené en autonomie, j’ai essayé d’appliquer une stratégie de tests manuels pour garantir la fiabilité et la fluidité de l’application.

* Tests fonctionnels réalisés :

|  |  |  |
| --- | --- | --- |
| Fonctionnalité | Objectif | Résultat |
| Inscription / Connexion | Vérifier que l’utilisateur peut s’inscrire et se connecter avec gestions des rôles | * Fonctionne avec sessions |
| Recherche de trajet | Tester le formulaire AJAX, l’envoi au back-end et l’affichage dynamique | * Données récupérées sans rechargement de la page |
| Réservation / Participation | Vérifier qu’un passager peut réserver un trajet et qu’il est lié à l’utilisateur | * Réservation enregistrée en BDD |
| Ajout d’un véhicule | Vérifier qu’un conducteur peut enregistrer un véhicule (électrique ou non) | * Données validées et stockées |
| Ajout d’avis | Vérifier que l’avis est bien lié au trajet et au passager | * Insertion correcte dans la table *avis* |
| Affichage de logs MongoDB | Contrôler que chaque recherche est enregistrée dans MongoDB | * Données bien stockées en JSON |

* Tests de validation des données :
* Champs requis vérifiés côté client (HTML : *« required », type= »email »…)*
* Sécurisation côté serveur avec *htmlspecialchars()*
* Protection contre : les champs vides ; les injections SQL (via PDO) ; les XSS (filtrage de sorties)
* Déploiement :

Le déploiement a été automatisé grâce à Github et la plateforme Heroku :

1. Code versionné et maintenu sur Github.
2. Repository lié à une application Heroku (voir annexe n°9)
3. Déploiement automatique à chaque « push » sur la branche « main » du repo (voir annexe n°8)
4. Logs accessibles via « heroku logs » pour surveiller le comportement en ligne

L’image Docker a permis de packager l’application dans un environnement stable (PHP + Apache + MySQL + Mongo). Des scripts « build » et « init.sql » assurent la préparation de la base sans intervention manuelle.

Dans le cadre de ce projet et travaillant seul je ne me suis pas rendu compte au début de l’importance des différentes branches dans mon repo GitHub. En effet je n’ai travaillé que sur la branche « Main » et « Devop » ce qui est après coup une erreur. Je me suis rendu compte de l’importance de ce point que sur la fin de ce projet. Mais j’ai compris et appris de cette erreur et voilà comment j’aurai dû procéder et structurer mes dépôts :

1. Construire une branche « Main » relié à la plateforme Heroku pour le déploiement final de l’application. (FAIT)
2. Branche « Devop » qui est la branche de travail pour le développement. Toutes les nouvelles fonctionnalités sont fusionnées ici avant d’aller sur la branche « Main ». Elle sert à tester les ajouts sans risquer la stabilité. (FAIT)
3. Création d’une branche « Features» qui permet l’ajout de chaque fonctionnalité comme par exemple : (NON FAIT)

* Features/auth : branche qui permet de tester la connexion et l’authentification
* Features/trajet : branche qui permet créer, rechercher ou afficher un trajet
* Features/Ajax : intégration du système Ajax coté front pour les formulaires

Cette branche est essentielle car à chaque création d’une fonctionnalité, on peut bien l’authentifier et tester afin d’éviter la perte de temps lors des débugs.

1. Branche « Fix » qui sert lors de la correction de bug. (NON FAIT)
2. Branche « Design » qui sert uniquement à la partie visuelle et qui sont récurrentes dans l’application comme par exemple le « header », le « footer ». (NON FAIT)

Voici une procédure que j’aurai dû suivre dans ce projet (même en travaillant seul) pour respecter les bonnes pratiques du développement, à savoir :

* Travaille sur la branche « features »
* « push » régulièrement sur le repos GitHub
* Quand la fonctionnalité est terminé et stable, « merge »  sur la branche « Devop »
* Une fois que tout est stable sur la branche « devop », merge sur la branche « Main ».
* Résultats obtenus :

Ce projet m’a permis de livrer une application :

* Fonctionnelle : toutes les fonctionnalités attendues sont en place (inscription, trajet, avis, recherche, …)
* Responsive : utilisable aussi bien sur desktop que mobile grâce à Bootstrap
* Ecoresponsable dans sa logique : identité visuelle écologique, valorisation des trajets propres
* Techniquement robuste : POO, MVC, base mixte SQL / NoSQL et sécurisation de base.
* Bilan personnel :

Ce projet a été une étape clé dans ma reconversion professionnelle, il m’a permis de confronter mes acquis à un projet concret et complet ; apprendre à organiser une structure logicielle MVC, gérer Docker, Composer, AJAX, … ; progresser en autonomie sur des outils complexes ; prendre conscience de l’importance de la préparation (analyse, maquettage, test et organisation de travail).

Les points forts de ce projet sont : une capacité d’adaptation sur des outils techniques nouveaux et différents ; la mise en place d’une application complète (front-end, back-end, BDD, sécurité et déploiement) ; une cohérence entre l’objectif et son identité graphique et fonctionnelle.

Les axes d’amélioration sont :

* + Mieux structurer les phases initiales (planning, organisation de travail et de l’environnement de travail)
  + Documenter davantage le code pour un travail en équipe.

Pour résumer mon bilan voici un tableau sous le modèle KISS

|  |  |
| --- | --- |
| Keep – ce que je continuerai à faire | * Structurer mes projets avec une architecture MVC claire et modulaire. * Utiliser Docker pour garantir un environnement stable et reproductible. * Travailler avec GitHub et des branches propres pour versionner et sécuriser mon code. * Créer des maquettes et une **charte graphique cohérente** avant d’entamer le développement. |
| Improve – ce que je dois améliorer | * Planifier davantage en amont (étapes, rétroplanning) pour éviter de me disperser. * Être plus rigoureux sur la documentation du code et des fichiers de configuration. |
| Start – ce que je vais commencer à faire | * Intégrer des tests automatisés dès le début d’un projet. * Travailler avec des outils de gestion de projet (ex. : Trello, Notion) pour mieux suivre l’avancement. * Explorer des frameworks PHP modernes (Laravel, Symfony) pour gagner en productivité. |
| Stop – ce que je vais arrêter de faire | * Me lancer dans le développement sans avoir clairement défini les besoins ni validé les parcours utilisateurs. * Gérer tout manuellement quand des outils comme Docker ou Composer permettent d’automatiser. |

# ANNEXES

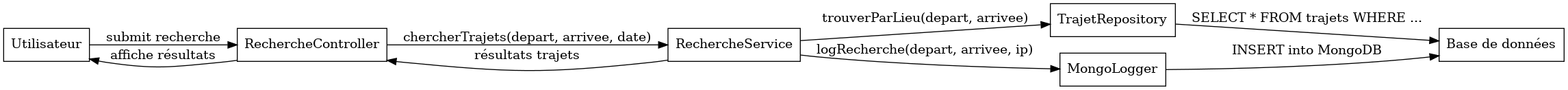
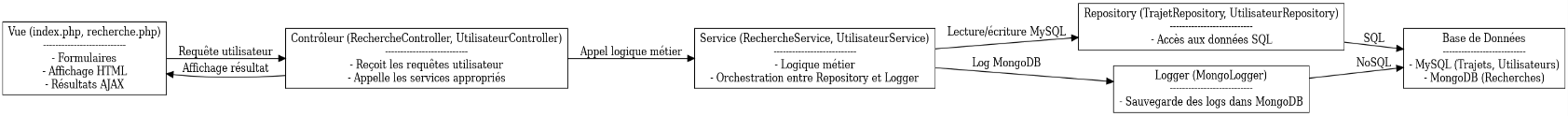
Une image contenant texte, capture d’écran, diagramme, ligne

Le contenu généré par l’IA peut être incorrect.

Annexe 1: Schéma du diagramme d’utilisation

Une image contenant texte, diagramme, cercle, dessin

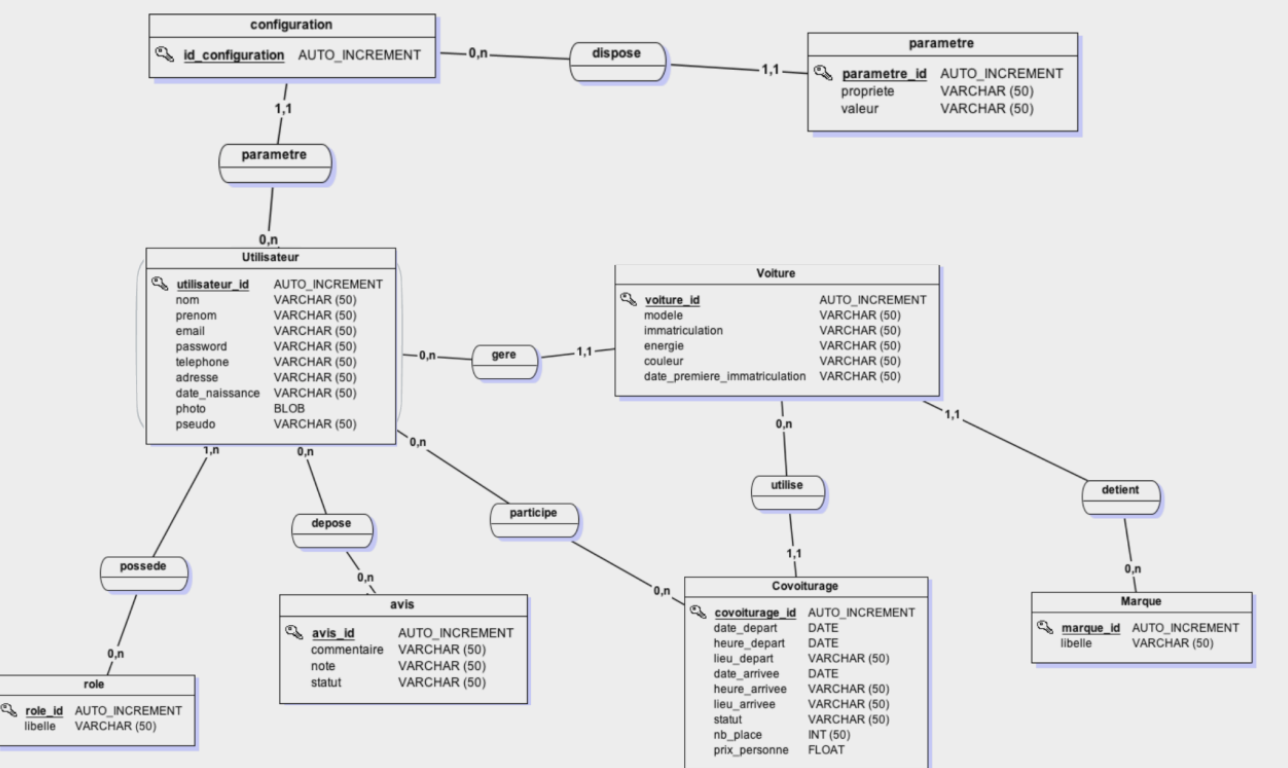
Le contenu généré par l’IA peut être incorrect.



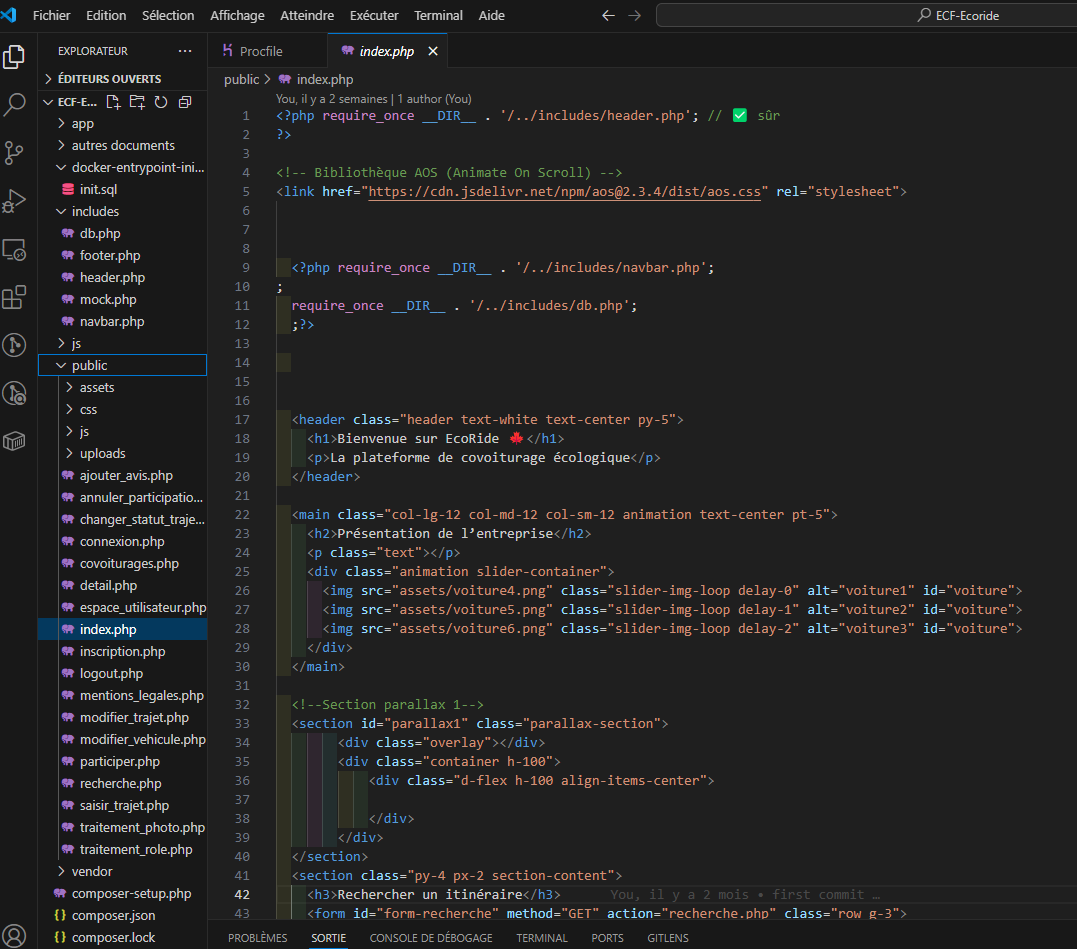
Annexe 2 : Diagramme de classe

Annexe 2 ter : diagramme MVC

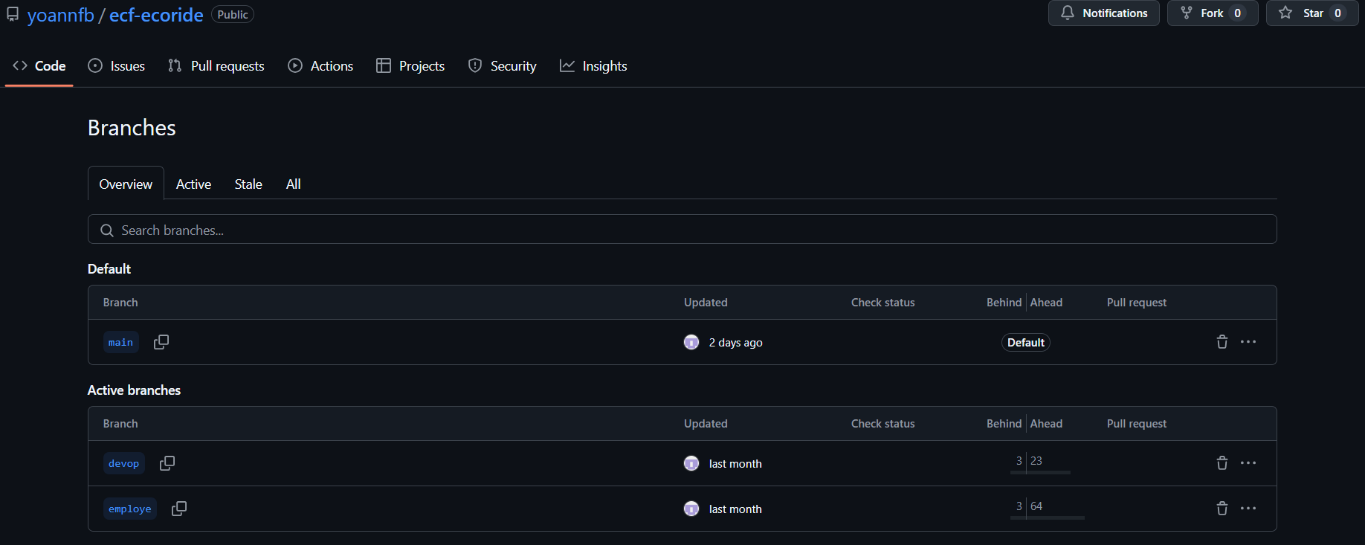
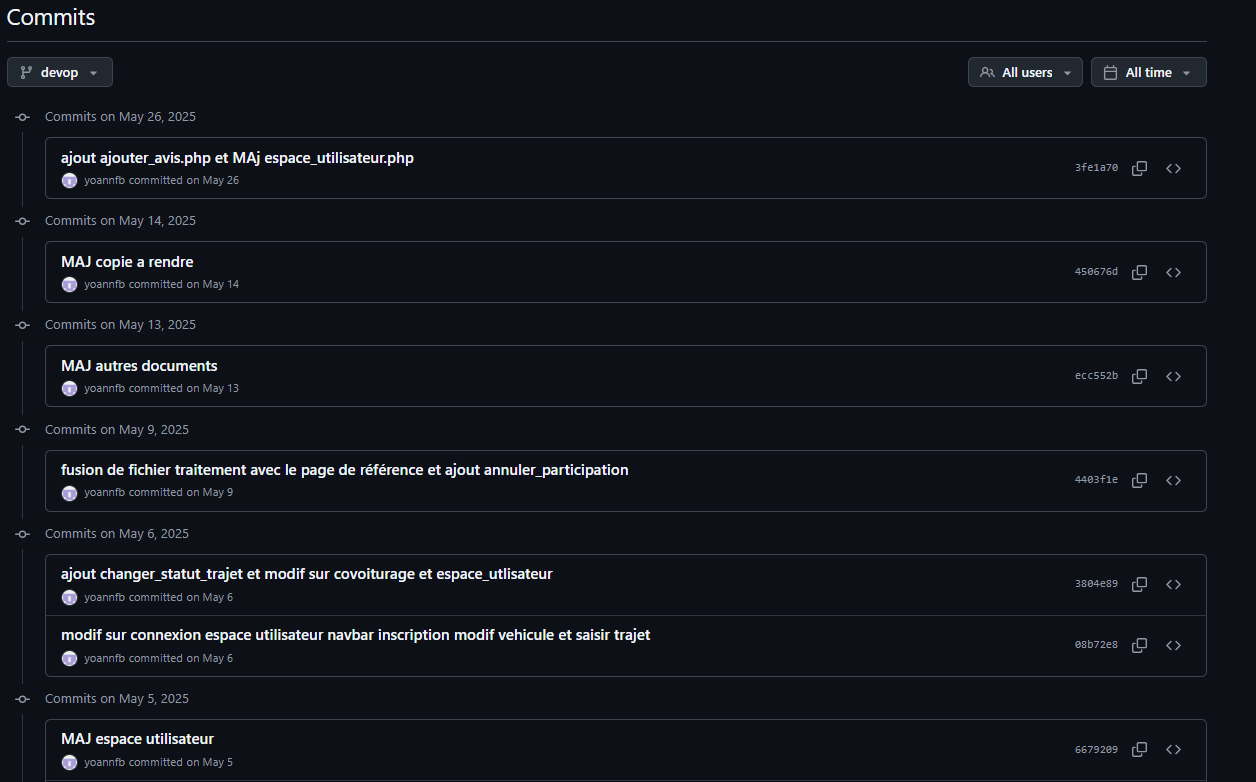
Annexe 2 bis : diagramme de séquence recherche



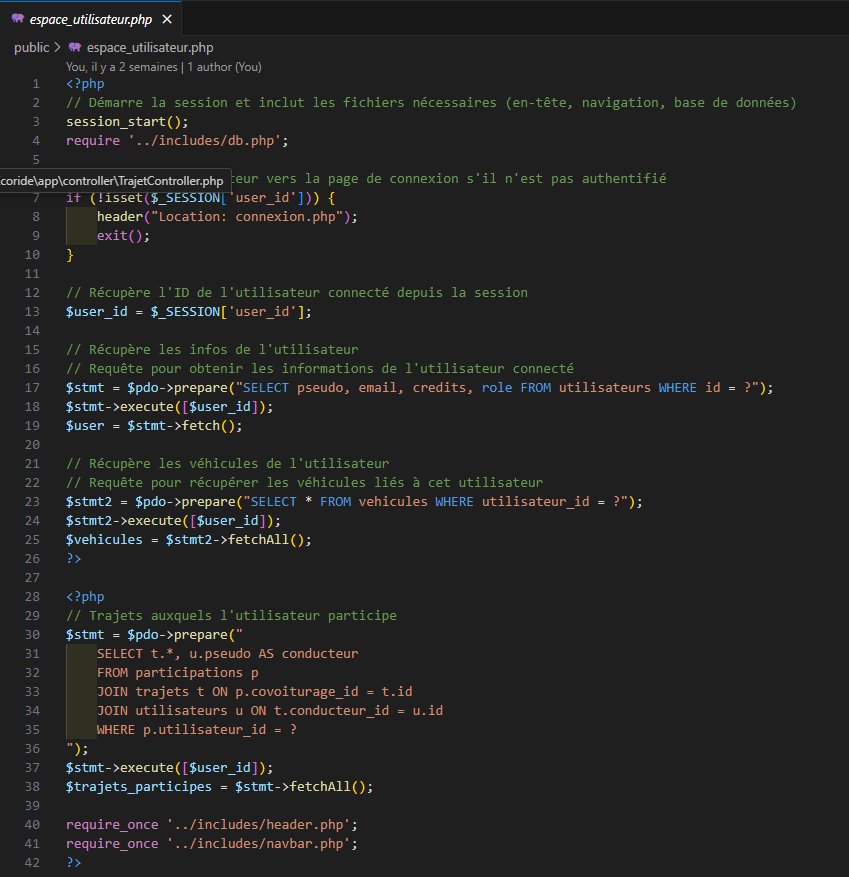
Annexe 3: Schéma MCD de la base de données relationnelle



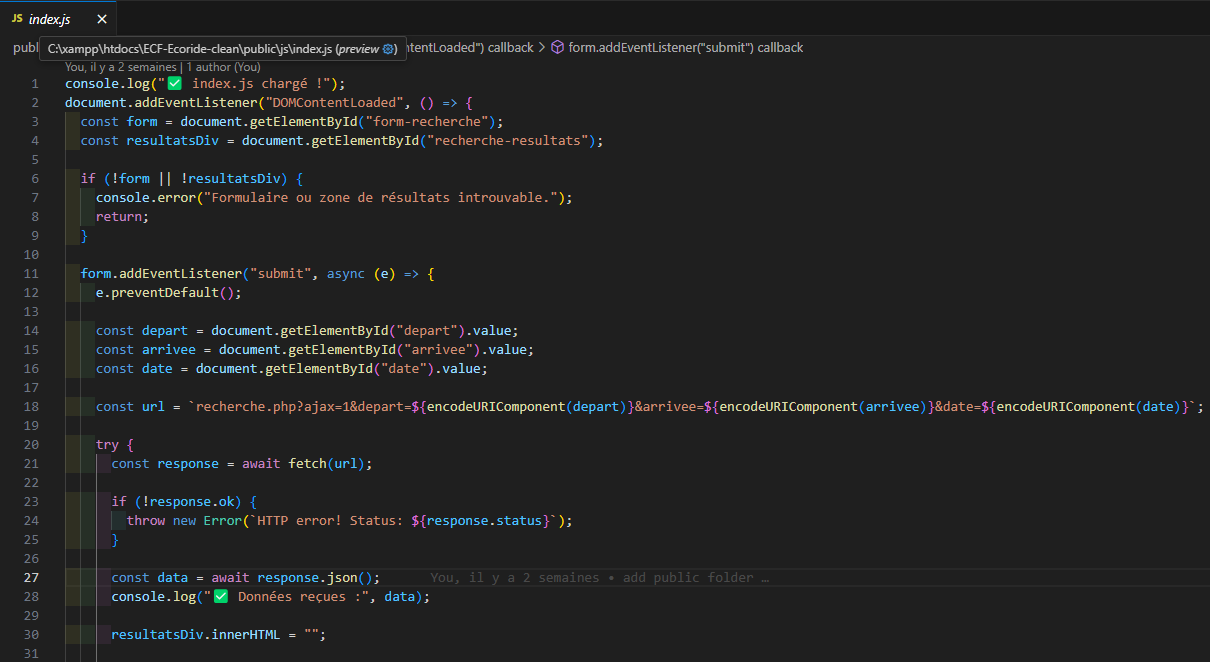
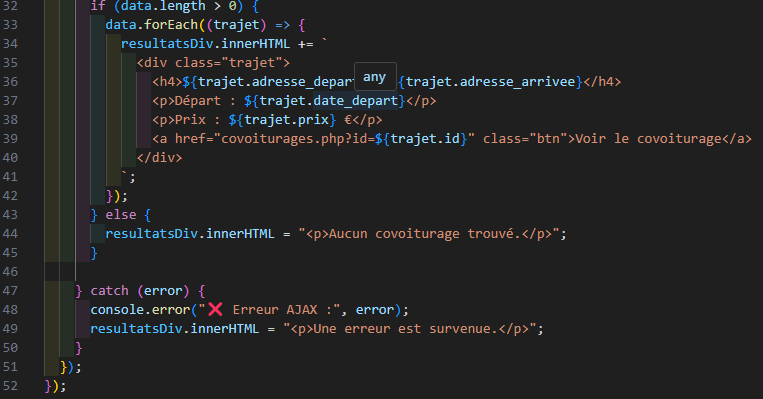
Annexe 4: structure du projet



Annexe 5: repository Github



Annexe 5: exemple code PHP



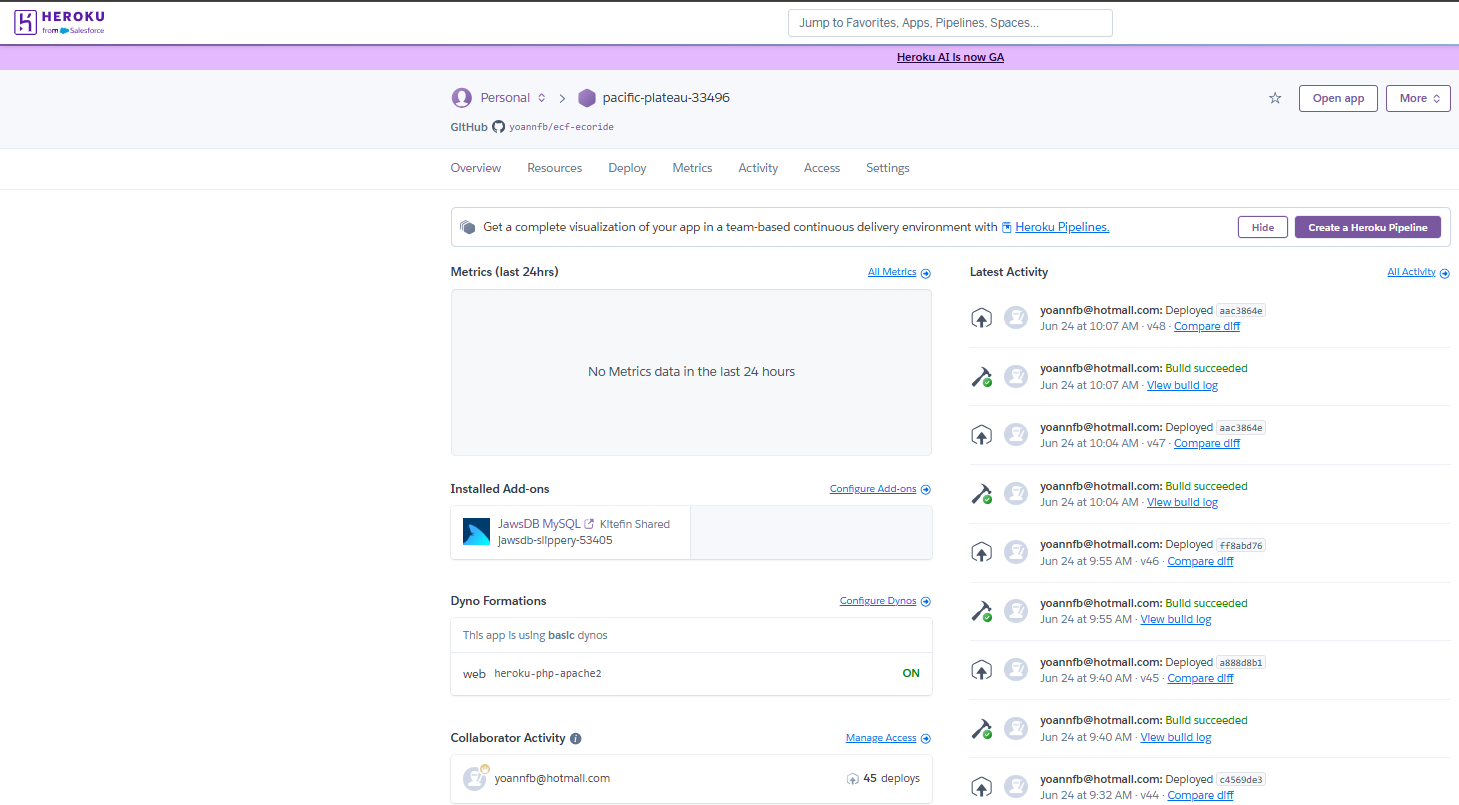
Annexe 7: fonctionne asynchrone AJAX



Annexe 7ter: fichier recherche.php

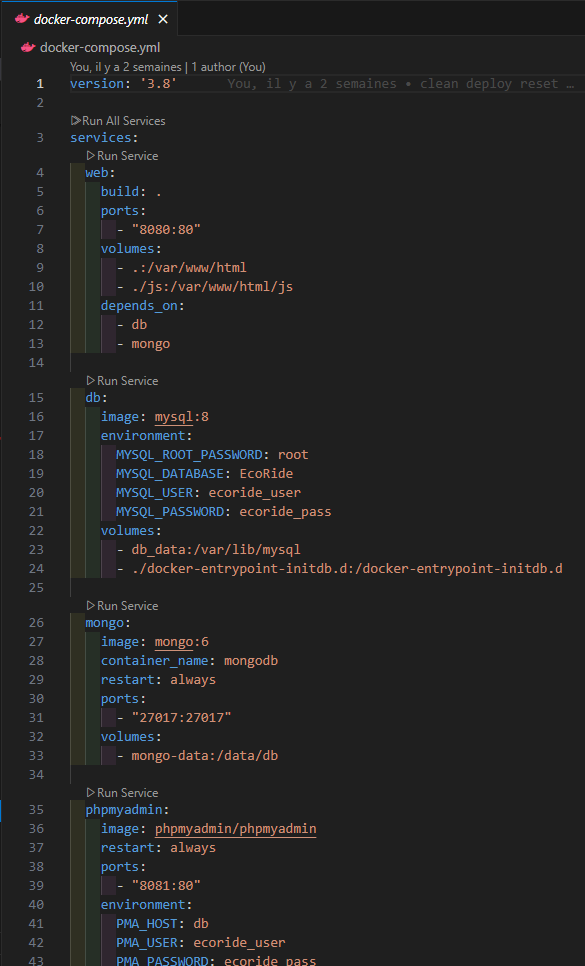


Annexe 7bis: formulaire recherche dans index.php

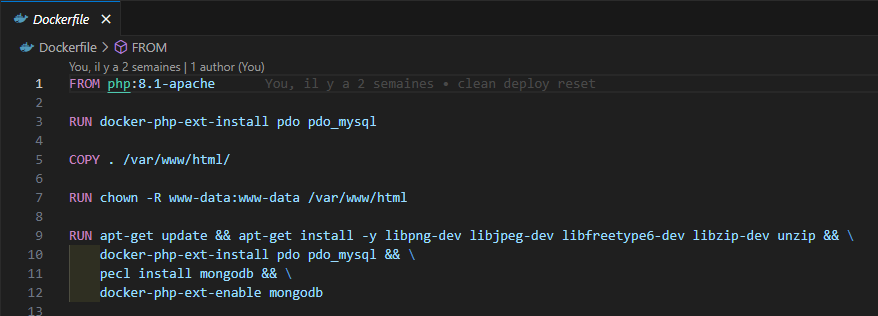


Annexe 9: Déploiement sur Heroku

Annexe 8: exemple du repo GitHub



Annexe 10: fichier docker-compose.yml



Annexe 10bis : fichier Dockerfile