Documentation technique

1. Réflexion technologique initiale

Lors de la conception d'EcoRide, plusieurs objectifs ont guidé mes choix technologiques :

- **Facilité de développement** : je maîtrise JavaScript, ce qui m'a permis de travailler efficacement avec les technologies modernes.

- **Performance côté client** : React avec TypeScript offre un rendu rapide, une bonne gestion d'état et une expérience utilisateur fluide.

- **Sécurité et scalabilité côté serveur** : Express.js avec Node.js permet de créer des APIs robustes, sécurisées et évolutives.

- **Stockage structuré vs non structuré** : j'ai opté pour **MySQL** pour les données fortement liées (utilisateurs, trajets, réservations) et **MongoDB** pour les incidents, qui sont plus flexibles et moins relationnels.

- **Apprentissage** : TypeScript et MySQL étaient des outils que je voulais approfondir, et ce projet en a été l'occasion.

2. Configuration de l'environnement de travail

- **Visual Studio Code** : éditeur principal avec extensions ESLint, Prettier…
- **Postman** : pour tester les routes backend.
- **DataGrip** : gestion des bases de données MySQL et MongoDB.
- **Chrome DevTools** : débogage du frontend.
- **Terminal & npm scripts** : pour lancer les serveurs, exécuter les builds, etc.

3. Documentation du déploiement

Backend

- **API déployée sur Heroku**, avec configuration d'un dyno web et d'un buildpack Node.js.

- **Base de données MySQL** hébergée via **JawsDB** (add-on Heroku).
- **Base de données MongoDB** hébergée sur **Atlas**.

Pour le déploiement de l'application, une fois la branche *dev* fusionnée avec la branche *main* de mon dépôt GitHub, j'ai commencé par le **backend**, en mettant en place les **bases de données**. Mon objectif était de m'assurer que les connexions fonctionnaient correctement via mon **API en environnement de développement**. Il s'agissait ensuite principalement de s'assurer de l'exactitude et la bonne configuration des variables d'environnement.

Pour la base de données relationnelle, j'ai utilisé **JawsDB**, un add-on Heroku qui fournit un accès à une instance **MySQL** distante. Cette instance est automatiquement provisionnée avec un nom d'utilisateur, un mot de passe, un hôte, un port et une base, que j'ai récupérés via les variables d'environnement de mon projet.

Pour la base non relationnelle, j'ai utilisé **MongoDB Atlas**, où j'ai créé un cluster hébergeant une base dédiée aux incidents. Cette base est indépendante du reste du système et accessible via une **URI sécurisée** définie dans les variables d'environnement.

J'ai également dû configurer **Heroku** afin qu'il puisse gérer un **monorepo**, c'est-à-dire un dépôt unique contenant à la fois le backend et le frontend dans des dossiers séparés. Pour cela, j'ai notamment eu à spécifier dans le fichier **Procfile** la commande suivante : *web: cd server && npm install && npm run build && npm run start*. Cette commande permet à **Heroku** de se positionner dans le dossier backend (server), d'installer les dépendances, de compiler le projet, puis de lancer le serveur, en ignorant ainsi le frontend.

Une fois le serveur et les bases de données déployés, j'ai exécuté le script d'initialisation de la base **MySQL** en production avec la commande suivante : *heroku run "npm run db:init:mysql"*. Ce script, présent dans mon projet, permet de créer automatiquement les tables nécessaires ainsi que d'insérer les données de départ (seeds) dans la base **JawsDB**.

Frontend

- Déployé sur **Vercel**.

Le déploiement de la partie frontend a été beaucoup plus simple et rapide que celle du backend. Il a suffi d'indiquer à Vercel le dossier contenant le client dans le monorepo, puis de renseigner les variables d'environnements nécessaires.