Réflexions initiale technologique sur le sujet

Base de données relationnelle : Mysql car plus simple à mettre en place, très simple à déployer, avec une large compatibilité applicative, optimisée pour des lectures intensives et doté d'un écosystème mature et d'un support commercial robuste.
La base de données NoSQL, j'ai choisi MongoDB car c'est la plus répandue. Elle est aussi simple d'utilisation que sleekdb par exemple, mais la documentation et la communauté sont plus importantes pour MongoDB.

Rôle de MySQL

- **Intégrité et cohérence des données** MySQL offre des transactions ACID (atomicité, cohérence, isolation, durabilité) essentielles pour des opérations critiques telles que la gestion des comptes utilisateurs, les réservations de covoiturages ou encore la gestion financière (crédits et débits). Par exemple, lorsqu'un utilisateur réserve une place, MySQL garantit la cohérence de la mise à jour du nombre de places disponibles et du solde de crédits.
- **Modélisation structurée et relations complexes** Les données telles que le profil des utilisateurs, les informations des véhicules, les itinéraires et les historiques de covoiturage possèdent un schéma bien défini et des relations interdépendantes (jointures entre utilisateurs, véhicules et réservations). MySQL permet de représenter ces relations avec précision grâce aux contraintes d'intégrité référentielle, garantissant ainsi un modèle de données robuste et fiable pour les opérations transactionnelles.

Rôle de MongoDB

- **Souplesse du schéma et agilité** MongoDB, en tant que base de données orientée document, permet de stocker des données dont la structure peut varier ou évoluer rapidement. Par exemple, les avis des utilisateurs, les logs d'activité, les préférences personnalisées ou même certains caches de résultats de recherche peuvent être stockés sous forme de documents JSON. Cette flexibilité est particulièrement intéressante dans un environnement où le modèle de données peut être amené à changer en fonction de l'évolution des fonctionnalités de l'application.
- **Scalabilité et performance en lecture** Pour des fonctionnalités nécessitant une lecture rapide et en temps réel—comme l'affichage instantané des covoiturages disponibles, le filtrage par critères (éco, prix, durée, note), ou l'agrégation de logs d'utilisation—MongoDB offre une excellente performance grâce à son modèle de données dénormalisé et sa capacité à scaler horizontalement. Cela permet à l'application de gérer des volumes élevés de requêtes et de données non structurées sans sacrifier la réactivité.

Complémentarité dans le contexte de l'application

- **Polyglot Persistence pour un meilleur compromis** En combinant MySQL et MongoDB, l'application de covoiturage pourrait utiliser MySQL pour les opérations transactionnelles et les données sensibles à l'intégrité (gestion des comptes, gestion des réservations, paiements, etc.), alors que MongoDB serait mis à contribution pour gérer des données moins formelles

et nécessitant une grande flexibilité (avis des utilisateurs, logs, préférences personnalisées, ou même un cache de recherche d'itinéraires permettant d'optimiser les performances).

- **Optimisation des requêtes** Par exemple, la logique métier qui demande une recherche par ville et par date pourrait d'abord interroger MongoDB pour récupérer rapidement des itinéraires potentiellement correspondants grâce à des index adaptés, puis consulter MySQL pour valider l'état de la réservation et assurer la cohérence transactionnelle lors de l'inscription définitive du covoiturage dans le système.

En résumé, l'association d'une base relationnelle comme MySQL à une base non relationnelle comme MongoDB permet de combiner la rigueur des transactions et des schémas bien définis avec la flexibilité, l'agilité de développement et la scalabilité en lecture des données hétérogènes. Cette approche permet à l'application de covoiturage d'être à la fois robuste et réactive, tout en facilitant son évolution face aux besoins utilisateurs changeants.

Configuration de l'environnement de travail

back et front en docker avec mysql, mongodb, php, apache ainsi que phpmyadmin et mongo-express pour avoir une interface graphique plus facile pour contrôler rapidement ce qui se passe en BDD.
Installation de PHP apache via docker car tout était déjà installé sur ma machine.
Installation de composer, puis dans le dossier dédié, j'ai exécuté la commande *composer create-project symfony/skeleton .*

Depuis mon IDE, j'ai créé le dépôt git et publié sur github.
J'utilise PHPstorm ou vscode. Draw.io pour les diagrammes et figma pour les maquettes. Postman pour tester l'API.

Documentation du déploiement de votre application expliquant votre démarche ainsi que les différentes étapes.

Tous les hébergements testés sont des versions gratuites car il n'y a pas de budget alloué pour cet exercice.
Ticket en cours chez alwaysdata pour connaître la raison du timeout pour déploiement automatique depuis github.
Fly.io, la machine mysql crash quand je lui envoie les migrations doctrine. Puis le docker symfony crash quand je fais plusieurs requêtes rapprochées depuis le front, donc je n'ai pas retenu cette solution qui est pourtant bien plus simple pour le déploiement.
Vercel est bien pour le front mais je ne trouve pas très adapté pour le back. J'ai donc regroupé au même endroit le back et le front. MongoDB est hébergé sur mongodb.com,

En attendant le retour du ticket alwaysdata, il faut utiliser un client FTP pour mettre à jour les fichiers. Puis se connecter en SSH pour exécuter composer update et bin/console doctrine:migrations:migrate afin de mettre à jour les différents éléments.