Documentation fonctionnelle

Spécification Technique

Pour aborder le développement de l'application de covoiturage "BOKKO" à un niveau d'architecture plus avancé, il est essentiel de considérer non seulement les choix technologiques mais aussi la manière dont ces technologies s'intègrent dans une architecture globale conçue pour la scalabilité, la performance, la sécurité, et l'évolutivité. Voici une exploration approfondie des spécifications techniques avec un focus sur l'architecture système.

Architecture Globale

L'architecture de "BOKKO" est conçue pour être robuste, évolutive, et sécurisée, en utilisant des technologies modernes et éprouvées à chaque couche de l'application.

Vue d'Ensemble

- **Front-end Mobile :** Flutter
- **Front-end Web :** React
- **Back-end :** Java Spring Boot (Architecture MVC)
- **Base de Données :** MySQL

Architecture Logicielle

1. **Front-end Mobile avec Flutter**

- **Rationalisation :** Flutter est choisi pour sa capacité à compiler en code natif, offrant ainsi d'excellentes performances sur iOS et Android tout en maintenant une expérience utilisateur homogène. Utiliser Flutter permet de bénéficier d'un cycle de développement rapide grâce à son hot reload, facilitant l'itération rapide sur l'UI/UX sans compromettre la qualité ou la performance. L'architecture de Flutter sera basée sur des principes de design réactif, utilisant le State Management pour une gestion efficace de l'état de l'UI à travers Provider ou Riverpod pour une meilleure lisibilité et maintenance du code.

2. **Front-end Web avec React**

- **Rationalisation :** L'utilisation de React pour le front-end web est justifiée par son modèle basé sur les composants, qui permet une réutilisation maximale du code et une gestion efficace de l'état de l'application. React favorise également une approche déclarative pour créer des interfaces utilisateur interactives, rendant le code plus prévisible et plus facile à déboguer. Une architecture SPA (Single Page Application) sera adoptée pour réduire les

chargements de page et améliorer l'expérience utilisateur, avec un routing côté client géré par React Router pour une navigation fluide.

3. **Back-end avec Java Spring Boot**

- **Architecture MVC :** L'adoption du modèle MVC (Modèle-Vue-Contrôleur) avec Spring Boot favorise une séparation claire des responsabilités, améliorant ainsi la modularité et la testabilité de l'application. Spring Boot facilite également l'intégration de dépendances, la configuration automatique, et la mise en place d'une sécurité robuste avec Spring Security pour gérer l'authentification et l'autorisation.
- **Microservices :** Pour soutenir la scalabilité et la maintenance de l'application, l'architecture back-end pourrait évoluer vers une architecture de microservices, où chaque service exécute une fonctionnalité spécifique et opère de manière indépendante. Cela permettrait une meilleure gestion des ressources, une isolation des erreurs, et une facilité de déploiement. Spring Cloud peut être utilisé pour simplifier le développement de ces microservices en gérant la configuration, la découverte de services, et le routage des requêtes.

4. **Base de Données MySQL**

- **Conception de Schéma :** La base de données MySQL sera conçue pour optimiser les performances et la cohérence des données. L'utilisation de clés étrangères pour les relations, d'index pour accélérer les requêtes, et de transactions pour assurer l'intégrité des données est cruciale. Une attention particulière sera portée à la normalisation pour éviter la redondance des données tout en équilibrant avec la dénormalisation là où c'est nécessaire pour les performances de lecture.
- **Haute Disponibilité et Scalabilité :** La mise en place de MySQL en configuration maître-esclave ou en cluster Galera pour MySQL peut offrir une haute disponibilité et une scalabilité horizontale. Cela permet de répliquer les données en temps réel et de basculer automatiquement sur un nœud esclave en cas de défaillance du maître, assurant ainsi une continuité du service.

Sécurité et conformité

Authentification et Autorisation

JWT pour l'Authentification Stateless

- **Implémentation :** JSON Web Tokens (JWT) seront utilisés pour l'authentification des utilisateurs. Lorsqu'un utilisateur se connecte, le système valide ses identifiants et génère un JWT contenant une signature cryptographique et un ensemble de claims (affirmations) qui incluent l'identité de l'utilisateur et les rôles associés. Ce token est envoyé au client et doit être inclus dans les en-têtes HTTP pour les requêtes ultérieures.
- **Avantages :** L'approche stateless de JWT facilite l'évolutivité de l'application en réduisant le besoin de stockage de session côté serveur. Elle permet également une authentification et

une autorisation efficaces sur des services distribués dans une architecture de microservices.

OAuth2 pour l'Intégration avec des Fournisseurs d'Identité Externes

- **Implémentation :** OAuth2 sera utilisé pour permettre aux utilisateurs de s'authentifier via des fournisseurs d'identité externes (comme Google, Facebook, ou un système d'identité d'entreprise). Cela offre une expérience utilisateur simplifiée en permettant aux utilisateurs d'utiliser des comptes existants pour accéder à BOKKO.
- **Gestion des rôles :** Spring Security sera configuré pour gérer les accès basés sur les rôles, en définissant des autorisations spécifiques pour les différents types d'utilisateurs (par exemple, conducteur, passager, administrateur) et en sécurisant les endpoints API en conséquence.

Protection des Données

Chiffrement des Données en Transit et au Repos

- **En Transit :** TLS (Transport Layer Security) sera utilisé pour chiffrer les données en transit entre les clients et le serveur, empêchant ainsi l'écoute clandestine et la modification des données lors de leur transmission sur Internet.
- **Au Repos :** Le chiffrement AES (Advanced Encryption Standard) sera appliqué aux données sensibles stockées, telles que les informations personnelles des utilisateurs et les détails de paiement. Cela assure que même en cas d'accès non autorisé au stockage de données, les informations restent protégées.

Politiques de Gestion des Mots de Passe et Audits de Sécurité

- **Politiques de Gestion des Mots de Passe :** Des politiques strictes seront mises en place pour la création et la gestion des mots de passe, incluant la complexité minimale du mot de passe, la rotation obligatoire des mots de passe, et le stockage sécurisé des mots de passe en utilisant le hachage avec des sels uniques pour chaque utilisateur.
- **Audits de Sécurité :** Des audits de sécurité réguliers seront effectués pour identifier et corriger les vulnérabilités potentielles dans l'application. Cela inclut à la fois des analyses automatiques de code pour détecter les failles de sécurité communes et des tests de pénétration manuels pour évaluer la résilience de l'application contre les attaques.

Conformité

- **Réglementations :** L'application "BOKKO" adhérera aux réglementations locales et internationales pertinentes en matière de protection des données, telles que la GDPR en Europe, pour s'assurer que les pratiques de traitement des données sont conformes aux exigences légales en termes de consentement des utilisateurs, de droits d'accès et de suppression des données.
- **Principes de Sécurité par Conception :** Dès les premières étapes du développement, les principes de sécurité par conception et de minimisation des données seront appliqués,

garantissant que la sécurité est intégrée dans l'architecture de l'application et que seules les données nécessaires sont collectées et traitées.

Sécurité et conformité

- **CI/CD :** L'intégration continue et le déploiement continu seront gérés via Jenkins ou GitHub Actions pour automatiser les tests et le déploiement de l'application, réduisant le risque d'erreurs humaines et accélérant le cycle de mise en marché.
- **Conteneurisation et Orchestration :** Docker sera utilisé pour conteneuriser l'application, facilitant ainsi le déploiement et la scalabilité. Kubernetes pourrait être adopté pour l'orchestration des conteneurs, offrant une gestion efficace des déploiements, une scalabilité automatique, et une haute disponibilité.