Etude de Cas Technique - Lead Dev

Ce test est conçu pour être volontairement ouvert afin de vous donner l'opportunité de démontrer votre capacité à aborder un problème complexe de manière autonome, tout en proposant des solutions créatives et robustes. Enfin, nous pensons que les tests techniques ne se suffisent pas à eux-même, et que la curiosité, l'ingéniosité et le savoir-être des candidats priment sur le reste.

Contexte

Vous êtes chargé de développer un module de gestion de transactions pour une plateforme de financement participatif. Cette plateforme permet aux utilisateurs de financer des projets, et vos responsabilités incluent la gestion des paiements, le déclenchement de transactions, la planification de celles-ci, ainsi que la garantie de la résilience du système face aux erreurs lors des transactions.

Votre mission consiste à concevoir et développer un petit service transactionnel, à la fois pour le backend et le frontend. Ce projet doit simuler le processus de transactions, y compris les aspects de planification, de validation et de gestion des erreurs. Vous devez aussi créer une interface utilisateur simple pour visualiser et interagir avec ces transactions.

Objectifs

1. Backend (Nest.js ou Node.js au choix tant que celui ci est argumenté par rapport à nos enjeux & vos préférences / PostgreSQL)
Concevoir une API (REST ou GraphQL au choix et à argumenter) pour gérer des transactions financières simulées. Cela inclut :

- La création de transactions (un montant € allant d'un wallet expéditeur à un wallet récepteur)
- La possibilité de planifier des transactions futures (programmation).
- La gestion des états de transactions (à définir selon votre architecture mais par exemple : pending, scheduled, complete, ...)
- L'implémentation (ou à minima sa conception sur papier) d'un mécanisme de résilience aux erreurs (provenant du code ou de l'API) (voir 3.)
- La persistance des données dans PostgreSQL.

NB : La réalisation concrète d'une transaction est faite par l'appel d'une API externe (typiquement celle d'un prestataire de service de paiement qui "héberge" les wallets des utilisateurs). Vous pouvez donc faire appel à une fonction imaginaire qui serait codée au préalable et requêterait cette API.

2. Frontend (Next.js)
Créer une interface simple permettant de :

- Créer des transactions manuelles et programmées.
- Visualiser l'historique des transactions et leur statut.
- Gérer les erreurs de transaction et éventuellement les réessayer manuellement.

3. Gestion des erreurs et résilience

- Mettre en place une gestion des erreurs
- Gérer les scénarios où une transaction échoue

4. Autonomie et proactivité

- Vous êtes libre de choisir les outils, bibliothèques ou frameworks additionnels que vous estimez utiles.
- Vous devrez également faire des choix d'architecture, d'organisation du code et de bonnes pratiques de développement.
- Proposer des améliorations ou des fonctionnalités additionnelles en lien avec la résilience des transactions ou l'expérience utilisateur sera un plus.

Détails techniques

La langue utilisée dans le code doit être l'anglais

Tests : Il est recommandé d'écrire au moins une partie des tests qui valideraient les principales fonctionnalités. D'autre part, n'hésitez pas à réfléchir à comment tester l'API tierce/externe de réalisation des transactions (à la fois pour tester l'intégration du code avec celle-ci et pour tester l'API elle-même et identifier d'éventuelles régressions). Nous en discuterons oralement.

Bonus : Si le temps vous le permet, vous pouvez également :

- Ajouter un système de notifications pour informer l'utilisateur en temps réel de l'état de ses transactions (ex. : WebSockets).
- Mettre en place des logs et une gestion des erreurs détaillée pour monitorer les transactions.
- Imaginer une architecture permettant l'extensibilité du service transactionnel que vous nous présenterez oralement.

Critères d'évaluation

- Qualité du code : Propreté, organisation et lisibilité du code.
- Gestion des erreurs : Robustesse du système en cas de pannes ou d'échecs de transaction.
- Autonomie et proactivité : Capacité à faire des choix techniques, à anticiper les problèmes et à proposer des solutions.
- Expérience utilisateur : Fluidité et simplicité de l'interface utilisateur.
- Tests : Présence et qualité des tests.

Livrables :

- Le code source du projet (via GitHub ou tout autre repository).
- Une (très) brève documentation sur le fonctionnement du système, expliquant vos choix techniques qui nous donnera matière à en discuter à l'oral.
- Optionnel : un court rapport sur les éventuels axes d'amélioration que vous avez identifiés pour la gestion des transactions (rejoint le point 3 des "Bonus")