TD3 : Implémentation du Processus d'Évaluation des Demandes de Prêt en Microservices

Objectif

Ce TD a pour objectif d'implémenter un processus métier de gestion des demandes de prêt immobilier en utilisant des concepts avancés de microservices, de tâches asynchrones, et de messagerie interservices. Les étudiants devront appliquer des notions de **coroutines, files de messages, tâches en arrière-plan et événements** pour assurer une exécution efficace et distribuée du processus.

Exercice 1 : Structuration en Microservices et Communication

Contexte :
Le processus d'évaluation des demandes de prêt comprend plusieurs services (ca dépend de votre model) :

- **Service des demandes de prêt** (création et validation des dossiers)
- **Service de vérification du crédit** (évaluation des antécédents du client)
- **Service d'évaluation du bien** (analyse de la valeur du bien)
- **Service de décision** (approbation ou rejet de la demande)
- **Service de notification** (envoi des résultats aux clients)

Tâche :

1. **Décomposer le processus en microservices** et proposer une architecture.
2. **Modéliser la communication entre les services** en utilisant une **file de messages (RabbitMQ, Kafka)**.
3. Implémenter une **API REST** pour chaque microservice en Python (FastAPI ou Flask).
4. **Définir des événements** pour la transition entre les services (ex. : événement "Crédit vérifié").
5. **Tester l'enchaînement des services** en envoyant des requêtes HTTP et en observant la propagation des événements.

Exercice 2 : Tâches Asynchrones et Traitements en Arrière-Plan

Contexte :
Certaines étapes du processus métier nécessitent du temps (ex. : analyse du crédit, estimation immobilière). Il est inefficace d'attendre la réponse synchrone, ce qui justifie l'utilisation de **tâches asynchrones**.

Tâche :

1. **Utiliser Celery pour exécuter des tâches en arrière-plan** :
    - Vérification du score de crédit
    - Estimation immobilière
    - Génération et envoi de documents (ex. : calendrier de remboursement)
2. **Configurer un worker Celery** et assurer la persistance des tâches.
3. **Implémenter un mécanisme de monitoring** pour suivre l'état des tâches (Flower ou Prometheus).
4. **Gérer les erreurs et la persistance** en cas d'échec d'une tâche.

Exercice 3 : Messagerie et Gestion des Transactions

Contexte :
Les services doivent échanger des informations de manière fiable et atomique. Une transaction ne doit pas être validée si l'une des étapes échoue.

Tâche :

1. **Mettre en place RabbitMQ ou Kafka** pour la gestion des flux d'information entre les services.
2. **Implémenter un mécanisme de compensation** pour garantir la cohérence en cas d'échec partiel (ex. : si l'évaluation du bien échoue, annuler l'évaluation du crédit).
3. **Assurer la durabilité des messages** pour éviter leur perte en cas de panne du système.

Exercice 4 : Notifications et Communication en Temps Réel

Contexte :
Les clients doivent être informés de l'état de leur demande en temps réel (approbation, rejet, demande d'informations complémentaires).

Tâche :

1. **Mettre en place un WebSocket avec FastAPI** pour les notifications en direct.

2. **Gérer les événements en temps réel** en utilisant **Server-Sent Events (SSE)**.
3. **Implémenter un dashboard simple** qui affiche les mises à jour en temps réel du statut des demandes.

Livrables attendus

- **Code source des microservices** et leurs interactions.
- **Fichier de configuration** pour Celery et RabbitMQ/Kafka.
- **Documentation API** (Swagger/OpenAPI).
- **Diagramme d'architecture** du système.
- **Rapport expliquant les choix techniques** et les problèmes rencontrés.

Conseils

- Utilisez **Docker et Docker-Compose / Kubernetes** pour exécuter vos microservices.
- Vérifiez **l'ordonnancement des événements** pour éviter les boucles infinies.
- Exploitez **les logs et outils de monitoring** pour diagnostiquer les erreurs.
- Assurez-vous que votre système est **scalable et tolérant aux pannes**.