Application E-commerce basée sur l'Architecture Microservices

1. Introduction

Notre application e-commerce est une plateforme moderne conçue pour gérer des produits et traiter des commandes client. Elle adopte l'architecture microservices, une approche de développement qui privilégie la modularité et l'indépendance des composants.

2. L'Architecture Microservices

L'architecture microservices est une approche de développement d'applications dans laquelle chaque fonctionnalité est traitée comme un service autonome. Ces services sont :
- Autonomes : Chaque service fonctionne indépendamment des autres.
- Légers : Ils communiquent entre eux via des protocoles simples, souvent HTTP.
- Conteneurisés : Pour faciliter leur déploiement et leur scalabilité.

3. Structure de notre Application

Notre application est segmentée en plusieurs services indépendants, principalement :
1. Produits-service : Gère tout ce qui concerne les produits.
2. Commandes-service : Traite des commandes passées par les clients.

Chaque service est responsable de sa propre base de données, ce qui garantit l'isolation et la sécurité des données.

Structure Générale :
1. common : C'est un dossier qui contient du code qui est partagé entre différents services. Sa fonction principale est d'éviter la redondance du code.

- config : Ce sous-dossier contient des fichiers de configuration, comme les paramètres de connexion à la base de données.

2. produits-service : Ce dossier englobe tout ce qui concerne la gestion des produits.

- controllers : Ce dossier contient les fonctions qui répondent aux différentes routes API. Par exemple, `produit.controller.js` aurait des fonctions pour ajouter, supprimer, mettre à jour et récupérer des produits.

- models : Ici, vous trouverez les schémas de la base de données et la logique ORM (Object-Relational Mapping) pour les produits. Le fichier `produit.model.js` décrit la structure d'un produit dans la base de données.

- routes : Les fichiers de ce dossier définissent les routes API pour les produits, par exemple : ajouter un produit, récupérer tous les produits, etc.

- server.js : C'est le point d'entrée de notre service produit. Il initialise le serveur, connecte la base de données et écoute les requêtes.

commandes-service : Similaire à `produits-service`, ce dossier s'occupe de la gestion des commandes.

- controllers : Fonctions pour traiter les routes API des commandes.

- models : Schémas de la base de données et logique ORM pour les commandes.

- routes : Définit les routes API pour les commandes.

- server.js : Point d'entrée du service de commandes.

4. Fonctionnement du Produits-service

Le service de produits est la première étape de l'expérience utilisateur. Il permet de :
- Afficher une liste de tous les produits disponibles.
- Ajouter de nouveaux produits au catalogue.
- Mettre à jour les détails d'un produit existant.
- Supprimer des produits qui ne sont plus en stock ou qui ne sont plus vendus.

5. Fonctionnement du Commandes-service

Après avoir consulté les produits, un client peut passer une commande. Le service de commandes permet de :
- Passer une nouvelle commande pour un produit spécifique.
- Voir toutes les commandes existantes.
- Mettre à jour une commande, par exemple pour changer la quantité.
- Annuler une commande.

6. Communication entre les Services

L'un des aspects clés de l'architecture microservices est la communication entre les services. Dans notre cas, ils utilisent le protocole HTTP en suivant l'approche RESTful.

Lorsqu'une commande est passée, le service "Commandes" doit vérifier la disponibilité du produit commandé. Pour ce faire, il communique avec le service "Produits" pour obtenir ces informations. Cette interaction garantit que nous ne passons pas de commandes pour des produits non disponibles.

7. Problèmes rencontrés et solutions

Au cours de notre discussion et de la mise en œuvre, nous avons rencontré plusieurs problèmes :

- **Problème de CORS** : Lors de l'essai initial, nous avons rencontré des problèmes liés à la politique de partage de ressources cross-origin (CORS). Cela a empêché notre frontend de communiquer avec nos services backend.

  Solution : Nous avons introduit un middleware pour gérer les headers CORS et autoriser les requêtes de sources spécifiques.

- Défis de communication entre services** : Garantir une communication fluide et fiable entre les services a été un défi, notamment en ce qui concerne la gestion des erreurs .

8. travail en cours

Communication entre les Microservices :

Dans une architecture microservices, la communication entre les différents services est essentielle. Elle doit être efficace, fiable et sécurisée pour garantir le bon fonctionnement de l'ensemble du système. Dans notre application e-commerce, voici comment nous avons géré la communication entre les services "Produits" et "Commandes".

Protocole HTTP et Architecture RESTful

Nous avons adopté le protocole HTTP pour la communication entre nos microservices, en suivant une architecture RESTful. Cette approche est basée sur des conventions

standardisées pour effectuer des requêtes et recevoir des réponses, en utilisant des méthodes comme GET, POST, PUT, et DELETE.

HTTP RESTful: Par exemple, lorsque le service commandes reçoit une commande, il pourrait envoyer une requête HTTP au service produits pour mettre à jour la quantité du produit.
gRPC: Une alternative plus performante pour la communication entre microservices. gRPC est basé sur HTTP/2 pour les appels de transport et utilise Protocol Buffers pour la sérialisation.
Mise à jour du service commandes:

Lorsqu'une commande est passée, le service commandes doit:

Vérifier la disponibilité du produit dans le service produits.
Si le produit est disponible, il faut alors décrémenter la quantité du produit dans le service produits.
Mise à jour du service produits:

Le service produits doit avoir une route pour mettre à jour la quantité d'un produit. Cette route doit être sécurisée pour s'assurer qu'elle ne soit accessible que par les services autorisés (dans ce cas, uniquement le service commandes).