Introduction

Le projet est une application de gestion de produits en ligne
Les fonctionalités disponible:
- récuperer/modifier/ajouter/supprimer des produits
- ajouter/supprimer des produits du panier
- Créer une commande

lancer le projet

installer les dépendances par la commande npm install
lancer le Gateway par la commande node apiGateway.js
lancer le microservice produits par la commande node product_server.js
lancer le microservice panier par la commande node cart_server.js
lancer le microservice commande par la commande node order_server.js

Implémentation des microservices

Microservice Produits (gRPC) : Permet de gérer les produits (récupérer, ajouter, modifier, supprimer)
Microservice Panier (gRPC) : Gère les opérations de panier telles que l'ajout et la suppression des produits.
Microservice Commandes (gRPC) : Permet de récupérer et créer une commande

choix des architectures

L'utilisation de microservices gRPC et de passerelles API REST et GraphQL est une approche courante et efficace dans de nombreux systèmes distribués.
Voici comment cela fonctionne :
Microservices dans gRPC : Nous utilisons gRPC pour la communication entre les microservices.
Chaque microservice expose un ou plusieurs points de terminaison gRPC qui définissent les messages échangés avec ce service et ces clients.
Les microservices peuvent être écrits dans une variété de langages de programmation et peuvent communiquer de manière transparente entre eux via gRPC.
Passerelle API : La passerelle API sert de point d'entrée central dans le système pour les clients.
Il fournit une interface intégrée qui utilise à la fois REST et GraphQL pour permettre aux clients d'interagir avec les microservices.
La passerelle API reçoit les requêtes des clients, les traduit en appels gRPC vers les microservices correspondants, regroupe les réponses selon les besoins et les renvoie au client.
REST : la passerelle API fournit plusieurs points de terminaison REST pour les clients qui préfèrent ce style architectural.
Ces points de terminaison peuvent être conçus pour correspondre aux fonctionnalités fournies par le microservice et fournir une interface familière aux clients REST.
GraphQL : API Gateway fournit également un point de terminaison GraphQL pour les clients qui souhaitent effectuer des requêtes personnalisées plus flexibles.
GraphQL permet aux clients de spécifier exactement les données dont ils ont besoin, évitant ainsi une récupération excessive de données et réduisant la surcharge du réseau.
Communication entre API Gateway et les microservices : API Gateway communique avec les microservices à l'aide du protocole gRPC.
Lorsqu'il reçoit une requête REST ou GraphQL d'un client, il convertit la requête en appel gRPC vers le microservice correspondant.
Le microservice répond avec les données requises et la passerelle API renvoie ces données au client dans le format approprié (JSON pour REST, réponse GraphQL pour GraphQL).
Cette approche présente plusieurs avantages.
Interopérabilité : Les clients ont la possibilité d'interagir avec le système à l'aide de REST ou de GraphQL, leur permettant de choisir le style architectural qui correspond le mieux à leurs besoins.

Séparation des préoccupations : les microservices sont responsables de leurs propres fonctions commerciales, et les passerelles API gèrent les aspects de l'interface utilisateur tels que l'agrégation de données et la transformation des requêtes.
Évolutivité : L'utilisation de gRPC pour la communication entre les microservices bénéficie des hautes performances et de la faible latence de gRPC, ce qui facilite la mise à l'échelle de votre système.
La combinaison de microservices gRPC et de passerelles API REST et GraphQL donne lieu à un système flexible, efficace et évolutif qui répond aux divers besoins de vos clients.

Schéma de données graphql

Ce schéma de données définit les types GraphQL et les opérations disponibles pour interagir avec les produits, les paniers et les commandes dans votre application.

Voici une description de chaque type et opération :

Types GraphQL :
Product : Représente un produit dans votre système. Chaque produit a un identifiant unique (id), un nom (name) et un prix (price).
ProductInput : Type d'entrée utilisé pour créer de nouveaux produits. Il spécifie le nom et le prix du produit.
CartItem : Représente un élément du panier. Il contient l'identifiant du produit (productId) et la quantité (quantity) du produit dans le panier.
Cart : Représente le panier d'un utilisateur. Il contient une liste d'éléments de panier (items), chaque élément étant un CartItem.
ProductToAdd : Type d'entrée utilisé pour ajouter un produit au panier. Il spécifie l'identifiant du produit à ajouter (id) et la quantité (quantity) à ajouter.
ProductToDelete : Type d'entrée utilisé pour supprimer un produit du panier. Il spécifie l'identifiant du produit à supprimer (id).
OrderInput : Type d'entrée utilisé pour créer une commande. Il spécifie l'identifiant (id), le nom (name) et le prix (price) de chaque produit dans la commande.
Order : Représente une commande passée par un utilisateur. Chaque commande a un identifiant unique (id) et un montant total (totalAmount).
Opérations GraphQL :
Query :
readProduct(id: ID!) : Récupère les détails d'un produit spécifique en fonction de son identifiant.
allProducts : Récupère tous les produits disponibles dans le système.
getOrder(id: ID!) : Récupère les détails d'une commande spécifique en fonction de son identifiant.
Mutation :
deleteProduct(id: ID!) : Supprime un produit spécifique en fonction de son identifiant.
createProduct(input: ProductInput!) : Crée un nouveau produit avec les détails spécifiés.
updateProduct(id: ID!, name: String!, price: String!) : Met à jour les détails d'un produit spécifique en fonction de son identifiant.
addProductToCart(input: ProductToAdd!) : Ajoute un produit au panier avec la quantité spécifiée.
deleteProductFromCart(input: ProductToDelete!) : Supprime un produit du panier en fonction de son identifiant.
createOrder(products: [OrderInput]!) : Crée une nouvelle commande avec les produits spécifiés.
Ce schéma de données définit les structures et les opérations nécessaires pour gérer les produits, les paniers et les commandes dans votre application, en utilisant GraphQL comme langage de requête flexible et puissant,

En GraphQL, les requêtes sont divisées en deux types principaux : les requêtes et les mutations. Voici la différence entre les deux :

Query :
Les requêtes (Query) sont utilisées pour récupérer des données depuis le serveur.
Elles sont utilisées lorsque vous souhaitez lire des données, mais pas les modifier.
Les requêtes ne modifient pas l'état du serveur ni des données.
Elles sont similaires à des opérations de lecture dans une API REST.
Exemple de requête : récupérer les détails d'un produit, obtenir la liste de tous les produits, etc.
Mutation :
Les mutations (Mutation) sont utilisées pour modifier ou ajouter des données sur le serveur.
Elles sont utilisées lorsque vous souhaitez effectuer des opérations qui modifient l'état des données.
Les mutations peuvent inclure des opérations telles que la création, la mise à jour ou la suppression de données.
Elles sont similaires à des opérations d'écriture dans une API REST.
Exemple de mutation : créer un nouveau produit, mettre à jour les détails d'un produit, supprimer un produit, etc.
En résumé, les requêtes sont utilisées pour lire des données tandis que les mutations sont utilisées pour modifier des données. Cette distinction claire entre les opérations de lecture et d'écriture permet de structurer les requêtes GraphQL de manière claire et explicite.

résolveurs pour les requêtes GraphQL

Le fichier resolvers.js définit les résolveurs pour les requêtes GraphQL. Les résolveurs sont des fonctions qui sont responsables d'exécuter les requêtes GraphQL et de récupérer ou modifier les données à partir des microservices correspondants.

Chargement des dépendances :
Le fichier commence par l'importation des modules nécessaires, y compris grpc pour la communication gRPC et protoLoader pour charger les fichiers de définition de protocole gRPC.
Chargement des protocoles gRPC :
Les fichiers de protocole gRPC (product.proto, cart.proto, order.proto) sont chargés à l'aide de protoLoader, et les clients gRPC correspondants sont créés pour chaque microservice.
Définition des résolveurs :
Les résolveurs sont des fonctions qui correspondent aux opérations GraphQL définies dans le schéma.
Pour chaque opération de requête ou mutation définie dans le schéma, il y a une fonction résolveur correspondante.
Chaque résolveur utilise les clients gRPC appropriés pour appeler les méthodes des microservices correspondants et récupérer ou modifier les données.
Résolveurs pour les requêtes :
Les résolveurs pour les requêtes (Query) récupèrent les données à partir des microservices.
Par exemple, le résolveur allProducts récupère tous les produits, le résolveur readProduct récupère un produit spécifique par son identifiant, et le résolveur getOrder récupère une commande spécifique par son identifiant.
Résolveurs pour les mutations :
Les résolveurs pour les mutations (Mutation) effectuent des modifications sur les données à partir des microservices.
Par exemple, le résolveur createProduct crée un nouveau produit, le résolveur updateProduct met à jour les détails d'un produit existant, et le résolveur deleteProduct supprime un produit.
Les résolveurs utilisent les clients gRPC correspondants pour appeler les méthodes des microservices et effectuer les opérations nécessaires.
Exportation des résolveurs :
À la fin du fichier, les résolveurs sont exportés pour être utilisés dans le GateWay,

#le GateWay

Le fichier apiGateway.js est responsable de la mise en place de l'API Gateway pour l'application.

Chargement des dépendances :
Le fichier commence par l'importation des modules nécessaires, y compris express pour créer le serveur HTTP, ApolloServer pour la gestion des requêtes GraphQL, ainsi que d'autres modules pour le traitement des requêtes.
Chargement des fichiers proto gRPC :
Les définitions de protocole gRPC pour les microservices sont chargées à l'aide de protoLoader, et les clients gRPC correspondants sont créés pour chaque microservice.
Création de l'instance ApolloServer :
Une instance ApolloServer est créée en utilisant le schéma (typeDefs) et les résolveurs (resolvers) définis précédemment.
Application du middleware ApolloServer à Express :
Le middleware expressMiddleware d'ApolloServer est appliqué à l'application Express pour gérer les requêtes GraphQL.
Définition des routes pour les endpoints REST :
Des routes sont définies pour les endpoints REST correspondant à chaque opération CRUD sur les produits, le panier et les commandes.
Pour chaque opération, une requête HTTP est reçue, puis elle est traduite en un appel gRPC approprié à l'aide du client gRPC correspondant.
Gestion des requêtes GraphQL :
L'API Gateway expose également un endpoint GraphQL qui est géré par l'instance ApolloServer créée précédemment.
Les requêtes GraphQL sont acheminées vers le serveur Apollo pour l'exécution des requêtes et la résolution des données à partir des résolveurs.
Démarrage du serveur :
Enfin, le serveur Express écoute sur un port spécifié (dans cet exemple, le port 3000) pour les requêtes entrantes.
⇒ ce fichier apiGateway.js agit comme une couche intermédiaire entre les clients de l'application et les microservices, en exposant des endpoints REST et GraphQL pour les requêtes des clients et en utilisant gRPC pour communiquer avec les microservices.