Cours de GraphQL
Introduction en GraphQL

MANGBA Bénédicta & TOYI François Date: January 13, 2025

1 Définition

GraphQL : est un langage de requête pour les API.

2 Installation et Configuration

– Installer Node.js et npm (si ce n'est pas encore fait)
– Installer un framework backend Express, Apollo Server (nous allons utiliser Apollo Server)
– Installation de Prisma (pour la gestion de la base de données)

3 Concepts Fondamentaux de GraphQL

Schéma

Un schéma définit la structure de l'API GraphQL. Il spécifie les types de données, les requêtes et les mutations disponibles. Le schéma est écrit en utilisant le langage de définition de schéma GraphQL.

Types

GraphQL utilise des types pour décrire les données. Les types de base incluent :

- Scalar Types : Types primitifs comme Int, Float, String, Boolean, et ID.

- Object Types : Types définis par l'utilisateur qui contiennent des champs.

- Enum Types : Types qui définissent un ensemble de valeurs possibles.

- Interface Types : Types qui définissent un ensemble de champs que d'autres types doivent implémenter.

- Union Types : Types qui permettent à un champ de retourner plusieurs types différents.

Requêtes (Queries)

Les requêtes sont utilisées pour lire les données. Elles permettent aux clients de spécifier exactement quelles données ils souhaitent récupérer, ce qui évite le problème du "over-fetching" (récupération de trop de données).

Mutations

Les mutations sont utilisées pour modifier les données (ajouter, mettre à jour ou supprimer). Comme les requêtes, les mutations permettent également de spécifier les données à retourner après la modification.

Résolveurs

Les résolveurs permettent d'écrire la logique du schéma.

Résolveurs : Ils définissent comment récupérer les données pour chaque champ dans le schéma.

Logique du schéma : Ils mettent en œuvre la logique de la façon dont les données doivent être obtenues (par exemple, depuis une base de données, une API externe, etc.).

4 Faisons notre premier pas en GraphQL (Affichons 'Hello')

```
import { ApolloServer, gql } from 'apollo-server';

// Définition du schéma GraphQL
const typeDefs = gql`
  type Query {
    hello: String
  }
`;
```

```
// Définition des résolveurs
const resolvers = {
  Query: {
    hello: () => 'Bonjour tout le monde'
  }
};

// Création du serveur Apollo
const server = new ApolloServer({
  typeDefs,
  resolvers
});

// Démarrage du serveur
server.listen().then(({ url }) => {
  console.log(`Server ready at ${url}`);
});
```

5 Réalisons un petit projet de gestion de contact

5.0.1 Préparons la base de données et la structure

La gestion se fera avec prisma

5.0.2 Créons un serveur en GraphQL

```
import 'dotenv/config';
import { ApolloServer, gql } from 'apollo-server';
import fs from 'fs';
import path from 'path';
import resolvers from './resolvers.js';

const typeDefs = gql(
  fs.readFileSync(path.join(process.cwd(), 'src/schema.graphql'),
   { encoding: 'utf-8' })
);

const server = new ApolloServer({
```

```
  typeDefs,
  resolvers
});

server.listen().then(({ url }) => {
  console.log(`Server ready at ${url}`);
});
```

5.1 schema.graphql

```
type Query {
  users: [User]
  user(id: ID!): User
  }

type User {
  id: ID
  name: String
  email: String
  contacts: [Contact]
}
```

5.2 resolver.js

```
//const { PrismaClient } = require('@prisma/client');
import { PrismaClient } from '@prisma/client';
const prisma = new PrismaClient();


// Résolveurs GraphQL
const resolvers = {
  Query: {
    // Récupérer tous les utilisateurs
    users: async () => {
      try {
        return await prisma.user.findMany({
          include: {
```

```
            contacts: true,
          },
        });
      } catch (error) {
        console.error('Erreur lors de la récupération des utilisateurs:', error)
        throw new Error('Erreur lors de la récupération des utilisateurs.');
      }
    },


  user: async (parent, { id }) => {
    try {
      const userIdInt = parseInt(id, 10);
      if (isNaN(userIdInt)) {
        throw new Error("L'ID de l'utilisateur doit être un entier valide.");
      }

      const user = await prisma.user.findUnique({
        where: { id: userIdInt },
        include: {
          contacts: true, // Inclure les contacts associés à cet utilisateur
        },
      });

      if (!user) {
        throw new Error("Utilisateur introuvable.");
      }

      return user;
    } catch (error) {
      console.error('Erreur lors de la récupération de l\'utilisateur:', error);
      throw new Error('Erreur lors de la récupération de l\'utilisateur.');
    }
  },
},
```

Ce code va permettre l'affichage de l'ensemble des utilisateurs et un utilisateur