MODUL 5 GraphQL

5.1 Deskripsi Singkat

GraphQL adalah bahasa query yang digunakan untuk mengambil data dari server, dan ia telah menjadi pilihan yang populer dalam pengembangan aplikasi web dan mobile. Salah satu aspek utama yang membedakan GraphQL dari pendekatan tradisional seperti REST adalah fleksibilitasnya yang tinggi. Dalam GraphQL, klien dapat meminta data yang spesifik dan hanya mendapatkan informasi yang dibutuhkan, menghindari over-fetching (pengambilan data yang tidak diperlukan) atau under-fetching (tidak mendapatkan cukup data). Ini membuat GraphQL sangat efisien dalam mengoptimalkan kinerja aplikasi.

Pada dasarnya, GraphQL didasarkan pada definisi skema yang ketat. Server GraphQL mendefinisikan tipe-tipe data dan hubungan antara mereka dalam skema yang memetakan struktur data yang tersedia dan cara mengaksesnya. Skema ini berperan sebagai kontrak antara klien dan server, memungkinkan klien untuk menjelajahi tipe data yang tersedia dan membuat query yang sesuai.

GraphQL juga memiliki kemampuan untuk menangani permintaan kompleks dengan baik. Klien dapat menggabungkan beberapa query ke dalam satu permintaan, mengurangi jumlah permintaan ke server dan mengoptimalkan waktu tanggapan. Ini dapat sangat bermanfaat dalam situasi di mana klien memerlukan data dari beberapa sumber atau melibatkan banyak tipe data yang berbeda.

Selain itu, GraphQL memiliki dukungan yang kuat untuk real-time data melalui konsep yang disebut "subscriptions." Ini memungkinkan klien untuk berlangganan perubahan data dan menerima pembaruan secara otomatis saat data yang relevan berubah. Ini cocok untuk aplikasi yang memerlukan tampilan data yang selalu up-to-date, seperti aplikasi obrolan atau pemantauan.

GraphQL juga telah mendapatkan banyak dukungan dari komunitas pengembang, dengan berbagai bahasa pemrograman dan kerangka kerja

yang mendukungnya. Ada alat pengembangan yang kuat seperti GraphQL Playground yang memfasilitasi pengujian dan eksplorasi skema, serta dokumentasi yang secara otomatis dibangun berdasarkan skema, memudahkan pengembang untuk memahami dan berinteraksi dengan API GraphQL.

5.2 Tujuan Praktikum

Tujuan praktikum ini adalah membuat suatu layanan menggunakan GraphQL yang diimplementasikan dengan Spring Framework.

Setelah praktikum ini kompetensi yang akan dicapai adalah mahasiswa mampu memahami konsep teknologi web service dan memahami implementasi web service dengan GraphQL.

5.3 Alokasi Waktu

Alokasi waktu pada praktikum 1 adalah sebagai berikut :

1. Tatap muka di kelas : 50 menit

2. Kerja mandiri (mengerjakan penugasan) : 120 menit

5.4 Material Praktikum

Praktikum ini memerlukan beberapa material sebagai berikut:

1) Java Development Kit (JDK) versi 17,
2) Java IDE: Netbeans, IntelliJ IDEA, atau Spring Tool Suite (STS),
3) Maven 3.5+,
4) MySQL.

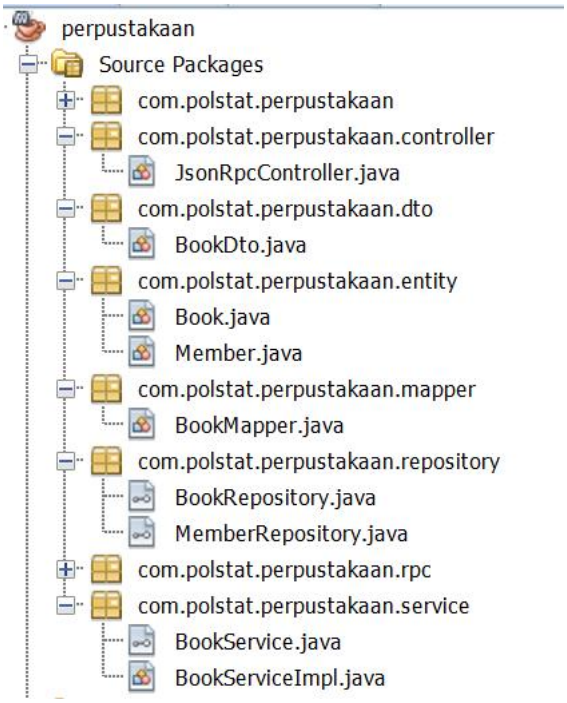
5.5 Kegiatan Praktikum

Pada praktikum ini kita akan memodifikasi aplikasi perpustakaan yang telah dibuat pada praktikum 4 dengan mengimplementasikan GraphQL

untuk mengakses layanan koleksi buku (operasi CRUD). Ikuti langkah-langkahnya sebagai berikut.

Langkah 1: Menyiapkan Proyek

1) Gunakan proyek perpustakaan pada modul praktikum 4 dengan struktur folder awal sebagai berikut.



2) Tambahkan dependency Spring GraphQL pada file pom.xml.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-graphql</artifactId>
</dependency>
```

Langkah 2: Membuat Kode Program

1) Membuat GraphQL Schema

GraphQL schema mendefinisikan struktur dan tipe-tipe data yang dapat diakses oleh klien melalui sebuah API GraphQL. Pada praktikum ini akan dibuat Graph schema yang mendefinisikan operasi CRUD untuk entity Book.

Buatlah folder *src/main/resources/graphql.* Buat file schema.graphqls dan tambahkan schema sebagai berikut.

```
type Book {
   id: ID
   title: String
   author: String
   description: String
}

type Query {
   books:[Book]
   bookById(id: ID): Book
}

type Mutation {
   createBook(title: String!, description: String, author: String!) : Book!
   updateBook(id:String!, title: String!, description: String, author: String!) : Book!
   deleteBook(id:String!): Book
}
```

2) Mengubah Interface BookService

Ubah interface BookService dengan menambahkan metode getBook() dan mengubah return type metode createBook() dan updateBook(). Berikut kode program selengkapnya.

```
package com.polstat.perpustakaan.service;

import com.polstat.perpustakaan.dto.BookDto;
import java.util.List;

public interface BookService {
   public BookDto createBook(BookDto bookDto);
   public BookDto updateBook(BookDto bookDto);
   public void deleteBook(BookDto bookDto);
   public List<BookDto> getBooks();
   public BookDto getBook(Long id);
}
```

3) Mengubah Kelas BookServiceImpl

Ubah BookServiceImpl dengan mengubah return type pada implementasi metode createBook() dan updateBook serta menambahkan implementasi metode getBook(). Berikut kode program selengkapnya.

```
package com.polstat.perpustakaan.service;

import com.polstat.perpustakaan.dto.BookDto;
import com.polstat.perpustakaan.entity.Book;
import com.polstat.perpustakaan.mapper.BookMapper;
import com.polstat.perpustakaan.repository.BookRepository;
import java.util.List;
import java.util.stream.Collectors;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class BookServiceImpl implements BookService{

    @Autowired
    private BookRepository bookRepository;

    @Override
    public BookDto createBook(BookDto bookDto) {
        Book book = bookRepository.save(BookMapper.mapToBook(bookDto));
        return BookMapper.mapToBookDto(book);
    }

    @Override
    public List<BookDto> getBooks() {
        List<Book> books = bookRepository.findAll();
        List<BookDto> bookDtos = books.stream()
                .map((product) -> (BookMapper.mapToBookDto(product)))
                .collect(Collectors.toList());
        return bookDtos;
    }

    @Override
    public BookDto getBook(Long id) {
        Book book = bookRepository.getReferenceById(id);
        return BookMapper.mapToBookDto(book);
    }

    @Override
    public BookDto updateBook(BookDto bookDto) {
        Book book = bookRepository.save(BookMapper.mapToBook(bookDto));
        return BookMapper.mapToBookDto(book);
    }

    @Override
    public void deleteBook(BookDto bookDto) {
        bookRepository.delete(BookMapper.mapToBook(bookDto));
    }
}
```

4) Membuat Kelas BookGraphqlController

Kelas ini digunakan untuk menangani permintaan GraphQL dari klien. Berikut kode program selengkapnya.

```
package com.polstat.perpustakaan.controller;

import com.polstat.perpustakaan.dto.BookDto;
import com.polstat.perpustakaan.service.BookService;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.graphql.data.method.annotation.Argument;
import org.springframework.graphql.data.method.annotation.MutationMapping;
import org.springframework.graphql.data.method.annotation.QueryMapping;
import org.springframework.stereotype.Controller;

@Controller
public class BookGraphqlController {

   @Autowired
   private BookService bookService;

   @QueryMapping
   public List<BookDto> books() {
      return bookService.getBooks();
   }

   @QueryMapping
   public BookDto bookById(@Argument Long id) {
      return bookService.getBook(id);
   }

   @MutationMapping
   public BookDto createBook(@Argument String title, @Argument String
description, @Argument String author) {
      BookDto bookDto = BookDto.builder()
            .title(title)
            .description(description)
            .author(author)
            .build();
      return bookService.createBook(bookDto);
   }

   @MutationMapping
   public BookDto updateBook(@Argument Long id, @Argument String title,
@Argument String description, @Argument String author) {
      BookDto bookDto = bookService.getBook(id);
      bookDto.setTitle(title);
```

```
        bookDto.setAuthor(author);
        bookDto.setDescription(description);
        return bookService.updateBook(bookDto);
    }

    @MutationMapping
    public void deleteBook(@Argument Long id) {
        BookDto bookDto = bookService.getBook(id);
        bookService.deleteBook(bookDto);
    }
}
```

Langkah 3: Menguji Proyek

1) Mengaktifkan GraphiQL Playground

   GraphiQL adalah sebuah alat pengembangan (developer tool) yang digunakan untuk mengelola dan menguji permintaan (query) dan mutasi GraphQL. GraphiQL menyediakan tampilan berbasis web yang interaktif yang dapat digunakan mengeksekusi query GraphQL, menjelajahi skema GraphQL, dan mendokumentasikan jenis-jenis data yang tersedia secara visual.

   Untuk mengaktifkan GraphQL pada proyek Spring, edit file *src/main/resources/application.properties* dan tambahkan pengaturan berikut:

   spring.graphql.graphiql.enabled=true

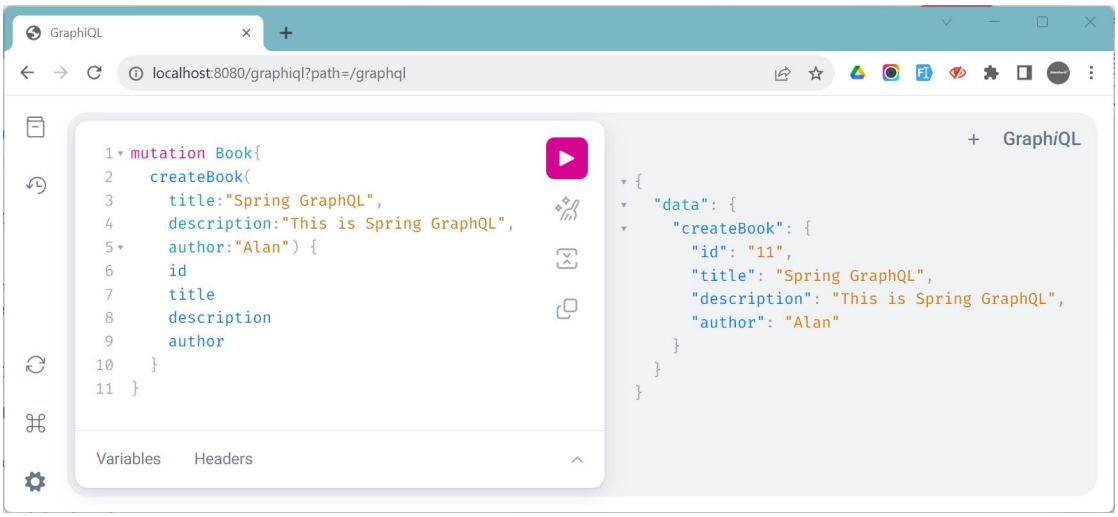
2) Menjalankan Proyek

   Jalankan proyek dengan klik menu Run File pada file *PerpustakaanApplication.java*.

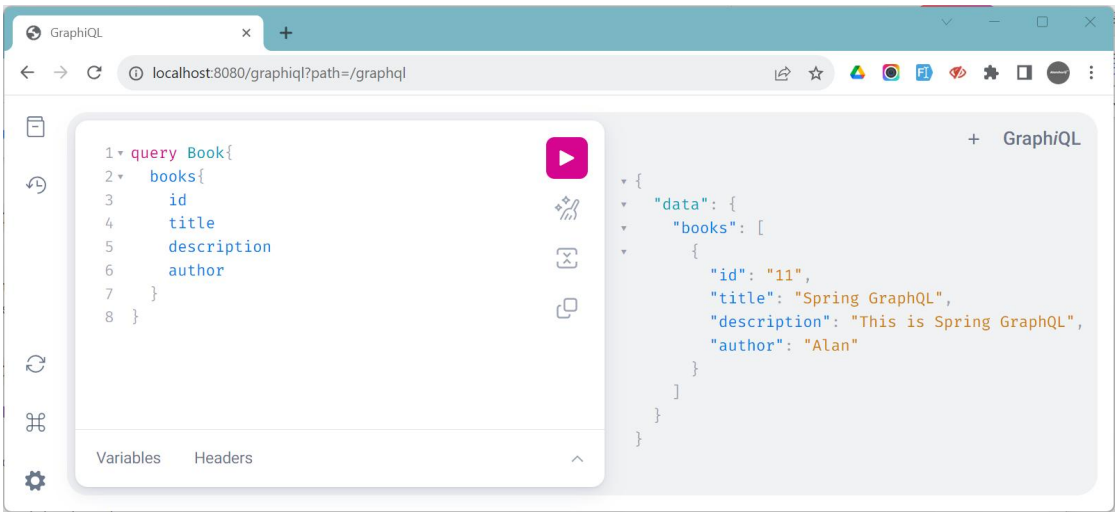
3) Menguji GraphQL

   Untuk menguji GraphQL, buka GraphiQL dengan mengakses URL http://localhost:8080/graphiql?path=/graphql pada browser. Lakukan beberapa pengujian sebagai berikut.

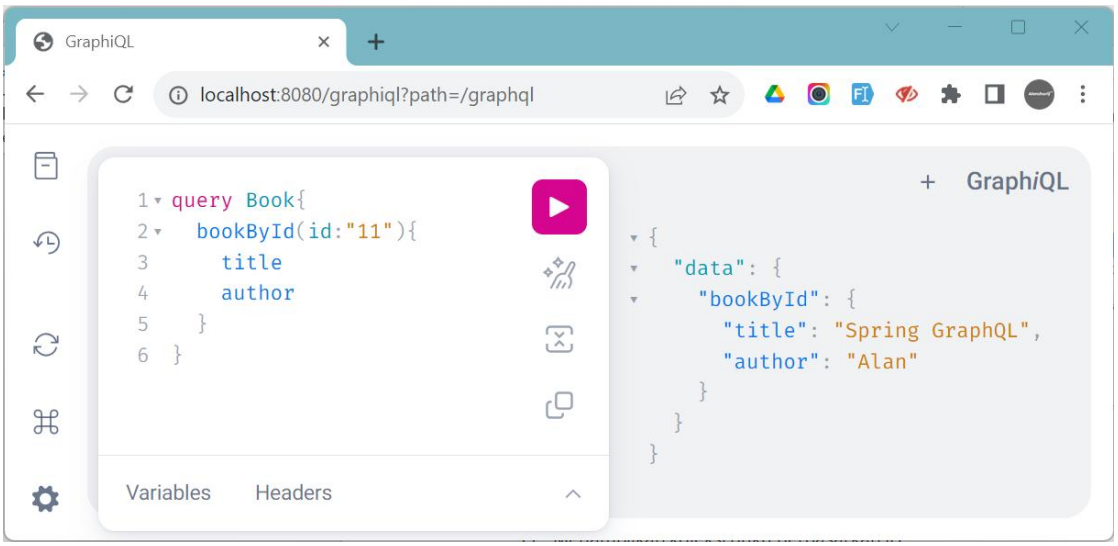
   a) Menambah koleksi buku



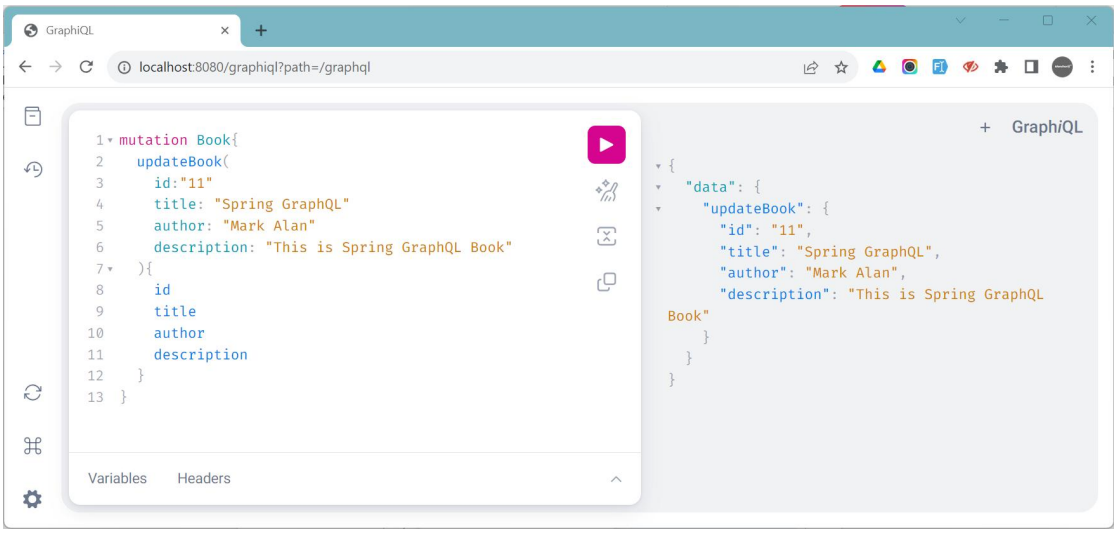
b) Menampilkan semua koleksi buku



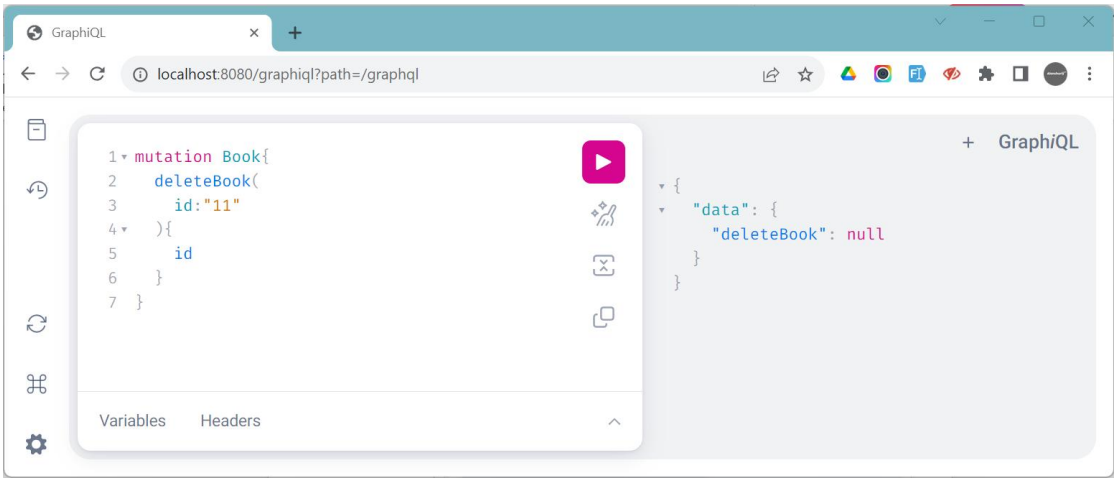
c) Menampilkan koleksi buku berdasarkan ID



d) Mengubah koleksi buku



e) Menghapus koleksi buku



5.6 Penugasan

Lanjutkan proyek perpustakaan praktikum 5 dengan menambahkan operasi CRUD untuk kelas entity Member menggunakan GraphQL.

Buat laporan pekerjaan Anda dengan penjelasan yang meliputi penjelasan kode program yang dibuat, pengujian setiap operasi CRUD beserta *output* responnya.