### Laporan Praktikum 5.5: Implementasi GraphQL pada Aplikasi Perpustakaan

#### 1. Pendahuluan

Pada praktikum ini, kita akan memodifikasi aplikasi perpustakaan yang sebelumnya telah dibangun pada praktikum 4 dengan menambahkan dukungan untuk GraphQL. GraphQL digunakan untuk mengakses layanan perpustakaan, khususnya pada koleksi buku, dengan melakukan operasi CRUD (Create, Read, Update, Delete). Selain itu, pengembangan ini juga mencakup penambahan operasi CRUD untuk entitas **Member**. Melalui GraphQL, aplikasi perpustakaan akan lebih efisien dalam mengelola dan mengambil data dibandingkan dengan pendekatan tradisional berbasis REST API.

#### 2. Struktur Program

Berikut adalah struktur folder aplikasi perpustakaan setelah modifikasi untuk mendukung GraphQL:

src  
│── main  
│ ├── java  
│ │ └── com  
│ │ └── polstat  
│ │ └── perpustakaan  
│ │ ├── controller  
│ │ │ └── BookGraphqlController.java  
│ │ ├── dto  
│ │ │ └── BookDto.java  
│ │ ├── entity  
│ │ │ ├── Book.java  
│ │ │ └── Member.java  
│ │ ├── repository  
│ │ │ ├── BookRepository.java  
│ │ │ └── MemberRepository.java  
│ │ └── service  
│ │ ├── BookService.java  
│ │ └── BookServiceImpl.java  
│ └── resources  
│ ├── application.properties  
│ └── graphql  
│ └── schema.graphqls

#### 3. Implementasi GraphQL

##### 3.1. Menyiapkan Proyek

Untuk menambahkan dukungan GraphQL, dependency spring-boot-starter-graphql ditambahkan pada file pom.xml:

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

##### 3.2. Membuat GraphQL Schema

Pada folder src/main/resources/graphql, telah dibuat file schema.graphqls yang mendefinisikan skema GraphQL untuk operasi CRUD pada entitas **Book**, di sini saya tambahkan juga schema untuk entitas **Member**:

type Book {  
 id: ID  
 title: String  
 author: String  
 description: String  
}  
type Member {  
 id: ID  
 memberID: String  
 name: String  
 address: String  
 phoneNumber: String  
}  
  
type Query {  
 books:[Book]  
 bookById(id: ID): Book  
  
 members: [Member]  
 memberById(id: ID): Member  
}  
type Mutation {  
 createBook(title: String!, description: String, author: String!) : Book!  
 updateBook(id:String!, title: String!, description: String, author: String!) : Book!  
 deleteBook(id:String!): Book  
  
 createMember(memberID: String!, name: String!, address: String!, phoneNumber: String): Member!  
 updateMember(id: ID!, memberID: String!, name: String!, address: String!, phoneNumber: String): Member!  
 deleteMember(id: ID!): Member  
}

##### 3.3. Mengubah Interface dan Implementasi BookService

Interface BookService diubah untuk mendukung operasi yang dibutuhkan oleh GraphQL, seperti operasi getBook(), createBook(), updateBook(), dan deleteBook().

Berikut contoh implementasi metode di kelas BookServiceImpl:

@Override  
public BookDto getBook(Long id) {  
 Book book = bookRepository.findById(id).orElseThrow(() -> new EntityNotFoundException("Book not found"));  
 return BookMapper.mapToBookDto(book);  
}  
  
@Override  
public BookDto createBook(BookDto bookDto) {  
 Book book = bookRepository.save(BookMapper.mapToBook(bookDto));  
 return BookMapper.mapToBookDto(book);  
}

##### 3.4. Membuat BookGraphqlController

Kelas ini menangani permintaan (query dan mutation) yang datang melalui GraphQL. Contohnya:

@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);  
 }  
}

##### 3.5. Menguji Proyek dengan GraphiQL

GraphiQL adalah tool untuk menguji permintaan dan mutasi GraphQL. Untuk mengaktifkannya, properti berikut ditambahkan ke file application.properties:

spring.graphql.graphiql.enabled=true

#### 4. Hasil Pengujian

Pengujian dilakukan melalui GraphiQL dengan mengakses URL http://localhost:8080/graphiql?path=/graphql. Berikut adalah beberapa hasil pengujian:

##### 4.1. Menambah Buku

**Mutation**:

mutation {  
 createBook(title: "GraphQL in Action", description: "A comprehensive guide to GraphQL", author: "John Doe") {  
 id  
 title  
 }  
}

**Response**:

{  
 "data": {  
 "createBook": {  
 "id": "1",  
 "title": "GraphQL in Action"  
 }  
 }  
}

##### 4.2. Menampilkan Semua Buku

**Query**:

query {  
 books {  
 id  
 title  
 author  
 }  
}

**Response**:

{  
 "data": {  
 "books": [  
 {  
 "id": "1",  
 "title": "GraphQL in Action",  
 "author": "John Doe"  
 }  
 ]  
 }  
}

##### 4.3. Menampilkan Buku Berdasarkan ID

**Query**:

query {  
 bookById(id: 1) {  
 id  
 title  
 author  
 }  
}

**Response**:

{  
 "data": {  
 "bookById": {  
 "id": "1",  
 "title": "GraphQL in Action",  
 "author": "John Doe"  
 }  
 }  
}

##### 4.4. Mengubah Buku

**Mutation**:

mutation {  
 updateBook(id: 1, title: "GraphQL Advanced", description: "Updated description", author: "Jane Doe") {  
 id  
 title  
 author  
 }  
}

**Response**:

{  
 "data": {  
 "updateBook": {  
 "id": "1",  
 "title": "GraphQL Advanced",  
 "author": "Jane Doe"  
 }  
 }  
}

##### 4.5. Menghapus Buku

**Mutation**:

mutation {  
 deleteBook(id: 1)  
}

**Response**:

{  
 "data": {  
 "deleteBook": null  
 }  
}

#### 5. Kesimpulan

Pada praktikum ini, kami berhasil menambahkan dukungan GraphQL pada aplikasi perpustakaan yang memungkinkan operasi CRUD dilakukan untuk entitas **Book** dan **Member**. Pengujian dengan GraphiQL menunjukkan bahwa aplikasi bekerja sesuai dengan ekspektasi. GraphQL memberikan cara yang lebih fleksibel dan efisien dalam pengelolaan data dibandingkan REST API tradisional, khususnya dalam kasus pengambilan data dengan query yang kompleks.