Data Persistence with Spring Data JPA

Session 2

Session Overview:
Training Approach and Interaction

Session 2

Presentation of Training Materials

Practical Training

Discussion

Session 2 Objective

Participants understand about Relational Database, CRUD Operations, using spring data JPA and JPQL Queries, Relation between entities and cascading operations.

MATERI PELATIHAN

- Introduction to relational databases and CRUD operations.
- Connecting to database with Spring Data JPA and JPA entities.
- Performing CRUD operations with repositories and JPQL queries.
- Implementing relationships between entities and cascading operations.

What is Relational Database?

- Relational Database merupakan cara menyajikan suatu informasi dalam bentuk tabel, baris dan kolom.
- Jenis Database yang memungkinkan membuat suatu hubungan antar informasi dengan menggabungkan beberapa tabel
- Dalam relational database, setiap baris data memiliki ID unik yang disebut sebagai key. Kolom tabel menyimpan atribut data, dan setiap data biasanya memiliki dataa untuk setiap atribut.

CRUD Operations

- **Create:** menambahkan data baru dalam sebuah tabel pada database.
- **Read:** mengembalikan data semua, sebagian tergantung kriteria dalam pencarian data.
- **Update:** mengubah suatu data yang sudah ada di tabel pada database.
- **Delete:** menghapus suatu data yang ada di tabel pada database.

Spring Data JPA

- Spring Data JPA merupakan bagian dari Spring Data yang memudahkan penerapan sebuah repositori berbasis JPA (Java Persistance API)
- JPA menggunakan konsep ORM (Object Relation Mapping) dimana objek dalam Java dimappingkan secara langsung ke dalam tabel database.
- JPA menyediakan spesifikasi untuk menyimpan, membaca, dan mengelola data dari objek java Anda ke tabel relasional di database.

Spring Data JPA

- Spring Data JPA merupakan bagian dari Spring Data yang memudahkan penerapan sebuah repositori berbasis JPA (Java Persistance API)
- JPA menggunakan konsep ORM (Object Relation Mapping) dimana objek dalam Java dimappingkan secara langsung ke dalam tabel database.
- JPA menyediakan spesifikasi untuk menyimpan, membaca, dan mengelola data dari objek java Anda ke tabel relasional di database.

Contoh:

```
7  @Repository
8  public interface UserRepository extends JpaRepository<User, Integer> {
9  }
10
```

JPA Entity

JPA Entity adalah kelas Java yang dapat dipetakan ke tabel database.

Kelas ini menentukan struktur data dan sebagai penjembatan antara objek Java dengan relational database

- @Entity: anotasi yang mendifinisikan suatu kelas yang dimappingkan dengan tabel.
- @Id: anotasi yang menentukan primary key dari sebuah entitas.
- @GeneratedValue: anotasi yang digunakan untuk menentukan teknik dalam pembuatan primary key yang akan digunakan.

Contoh:

```
15  @Entity
16  @Table(name = "tb_users")
17  public class User {
18
19      @Id
20      private int id;
21
22      private String name;
23      private String email;
24  }
25
```

JPQL Queries

Pada Spring Data memungkinkan kita mengekseksuksi suatu perintah SQL pada repositori Spring Data, dengan menganotasi metode @Query — dimana atribut value-nya berisi JPQL atau SQL.

Contoh:

```java
package com.example.demo.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import com.example.demo.model.User;

@Repository
public interface UserRepository extends JpaRepository<User, Integer> {

    @Query("SELECT u FROM User u WHERE u.name = :name")
    User findByName(@Param("name") String name);
}
```

Cascading Operations

Salah satu fitur JPA adalah cascading operation, yang memungkinkan mendefinisikan suatu hubungan antar entitas pada Spring Data JPA. Hal ini memungkinkan proses otomatis, seperti penghapusan, dan perubahan secara langsung terkait entita, yang sangat membantu memastikan konsistensi data.

Contoh:

```java
@Entity
@Table(name = "tb_users")
public class User {

    @Id
    private int id;

    private String name;
    private String email;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Task> tasks;
}
```

```java
@Entity
@Table(name = "tb_tasks")
public class Task {

    @Id
    private int id;
    private String title;
    private String description;
    private String priority;
    private Boolean status;

    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User user;
}
```

Cascading Operations

- **CascadeType.ALL**: Mengaktifkan semua operasi secara otomatis, seperti menyimpan (PERSIST), menggabungkan (MERGE), menghapus (REMOVE), menyegarkan (REFRESH), dan melepaskan (DETACH), ke entitas JPA terkait.
- **CascadeType.PERSIST**: Jika entitas induk disimpan (persist), entitas anaknya juga akan otomatis disimpan.
- **CascadeType.MERGE**: Jika entitas induk diperbarui atau digabungkan (merge), perubahan itu akan otomatis diterapkan ke entitas anaknya.
- **CascadeType.REMOVE**: Jika entitas induk dihapus, entitas anaknya juga akan otomatis ikut dihapus.
- **CascadeType.REFRESH**: Jika entitas induk disegarkan (refresh) untuk mendapatkan data terbaru dari database, entitas anaknya juga otomatis disegarkan.

Connecting Database

- Menambahkan driver MySQL pada project Spring Boot

```xml
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
</dependency>
```

- Setup koneksi ke database Mysql pada file **application.properties**

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=admin12#
spring.datasource.url=jdbc:mysql://localhost:3306/todo_app
```

Exercise

Membuat aplikasi sederhana To-do List App dengan kriteria sebagai berikut:

- Table User: untuk menyimpan data user / pengguna aplikasi
- Table Task: untuk menyimpan data task dari pengguna

Contoh Endpoint:

Endpoint	Method	Deskripsi
/api/users/register	POST	Registrasi pengguna baru.
/api/users	GET	Ambil semua daftar pengguna.
/api/users/:id	GET	Ambil daftar pengguna berdasarkan ID.
/api/users/:id	PATCH	Edit data pengguna berdasarkan ID
/api/users/:id	DELETE	Hapus pengguna berdasarkan ID
/api/todos/user/:id	GET	Ambil daftar tugas pengguna berdasarkan ID pengguna.
/api/todos/:id	GET	Ambil daftar tugas pengguna berdasarkan ID.
/api/todos	POST	Tambah tugas baru.
/api/todos/:id	PATCH	Edit tugas berdasarkan ID.
/api/todos/:id	DELETE	Hapus tugas berdasarkan ID.

Setup database To-do List App

- User

```sql
CREATE TABLE IF NOT EXISTS tb_users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    email VARCHAR(255) NOT NULL
) ENGINE = InnoDB;
```

- Task

```sql
CREATE TABLE IF NOT EXISTS tb_tasks (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    priority ENUM('HIGH', 'MEDIUM', 'LOW') NOT NULL,
    status BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    update_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES tb_users(id) ON DELETE CASCADE
) ENGINE = InnoDB;
```

Model

Pembuatan Model User

```java
package com.example.demo.model;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "tb_users")
public class User {

    @Id
    private int id;

    private String name;
    private String email;
}

```

User model terdiri atas atribut yang sama dengan database yang telah dibuat. Apabila ada perubahan pada database model User dapat disesuaikan

Repository

Pembuatan UserRepository

```java
package com.example.demo.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.example.demo.model.User;

@Repository
public interface UserRepository extends JpaRepository<User, Integer> {
}

```

Repository user bertindak sebagai koneksi data ke database MySQL sesuai dengan entity model yang sudah didefinisikan.

Data

Register User Request

```java
package com.example.demo.model;

import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Size;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class RegisterUserRequest {

    @NotBlank
    @Size(max = 50)
    private String name;

    @NotBlank
    @Size(max = 255)
    private String email;
}

```

Register User Request Data digunakan untuk membuat sebuah model data yang akan digunakan sebagai request data untuk diproses ke akses data layer untuk register pengguna baru. Biasanya banyak digunakan untuk method POST, atau menambahkan sebuah data.

Service

User Service Method Register

```java
@Transactional
    public String register(RegisterUserRequest request) {
        User user = new User();
        user.setName(request.getName());
        user.setEmail(request.getEmail());

        userRepository.save(user);

        return "Create success";
    }
```

User Service Method Register digunakan sebagai method untuk register data user baru sesuai dengan Data Register User Request yang dikirimkan.

Controller

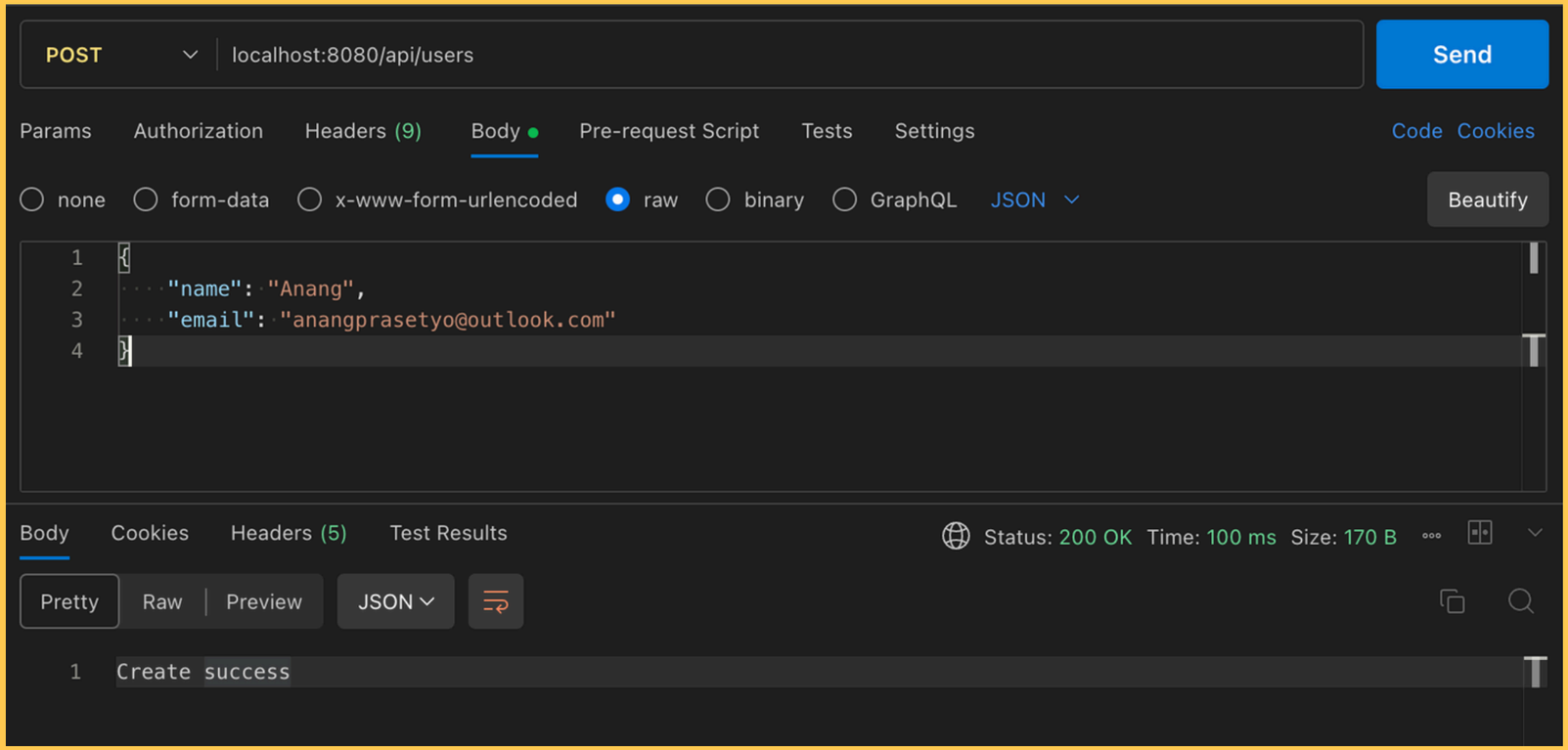
User Controller Method Register

```
@PostMapping(
        path = "/api/users",
        consumes = MediaType.APPLICATION_JSON_VALUE,
        produces = MediaType.APPLICATION_JSON_VALUE
    )
    public String register(@RequestBody RegisterUserRequest request) {
        return userService.register(request);
    }
```

User Controller Method Register digunakan sebagai method untuk menangani request POST untuk register user baru.

Testing

Endpoint Register User



Service

User Service Method Get All User

```
1   @Transactional
2       public List<User> getAll() {
3           return userRepository.findAll();
4       }
```

User Service Method Get All User digunakan sebagai method untuk mengambil semua data pengguna yang sudah teregister.

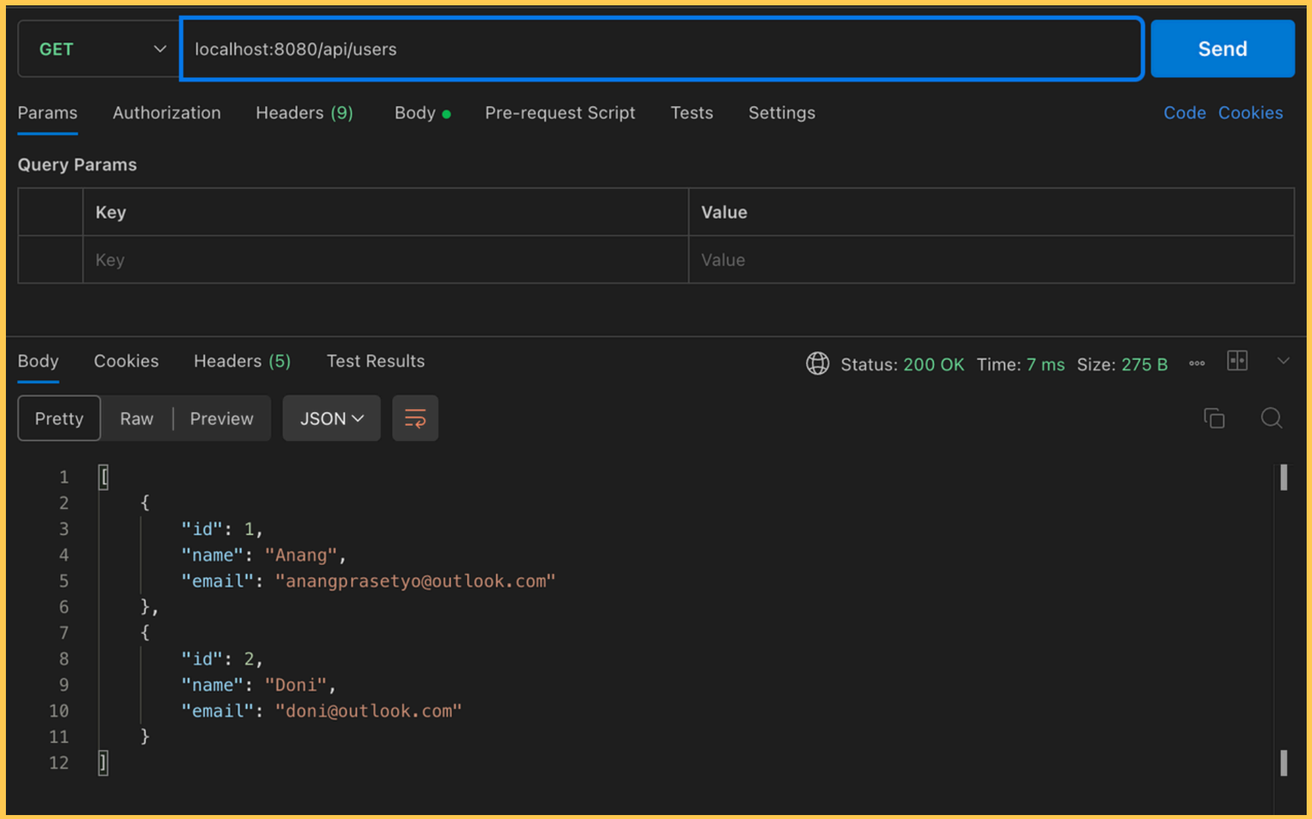
User Controller Method Get All User

```
1   @GetMapping("/api/users")
2       public List<User> getAll() {
3           return userService.getAll();
4       }
```

User Controller Get All User digunakan sebagai method untuk menangani request GET untuk mengambil semua data pengguna.

Testing

Endpoint Get All User



Service

User Service Method Get By Id

```
@Transactional
    public User get(int id) {
        return userRepository.findById(id).get();
    }
```

User Service Method Get By Id digunakan sebagai method untuk mengambil data pengguna berdasarkan ID.

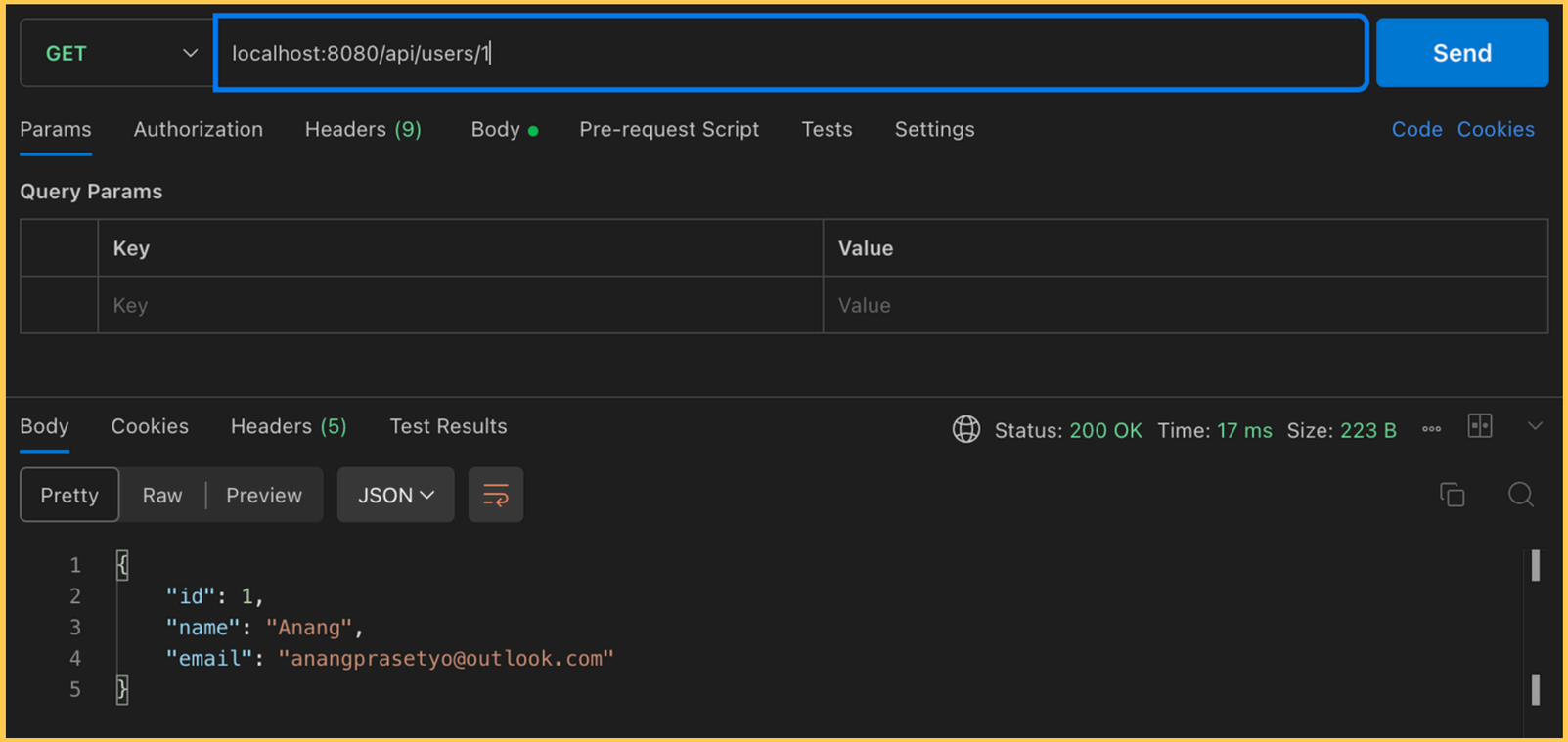
User Controller Method Get By Id

```
@GetMapping("/api/users/{id}")
    public User get(@PathVariable int id) {
        return userService.get(id);
    }
```

User Controller Get By Id digunakan sebagai method untuk menangani request GET untuk mengambil data pengguna berdasarkan ID.

Testing

Endpoint Get By Id



Data

Update User Request

```java
package com.example.demo.model;

import jakarta.validation.constraints.Size;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class UpdateUserRequest {

    @Size(max = 50)
    private String name;

    @Size(max = 255)
    private String email;
}

```

Update User Request Data digunakan untuk membuat sebuah model data yang akan digunakan sebagai request data untuk diproses ke akses data layer untuk update user / pengguna.

Service

User Service Method Update

```java
@Transactional
    public UserResponse update(int id, UpdateUserRequest request) {
        User user = userRepository.getReferenceById(id);

        if (Objects.nonNull(request.getName())) {
            user.setName(request.getName());
        }

        if (Objects.nonNull(request.getEmail())) {
            user.setEmail(request.getEmail());
        }

        return toUserResponse(user);
    }
```

User Service Method Update digunakan sebagai method untuk update data user sesuai dengan Data Update User Request yang dikirimkan.

Controller

User Controller Method Update

```
@PatchMapping(
        path = "/api/users/{id}",
        consumes = MediaType.APPLICATION_JSON_VALUE,
        produces = MediaType.APPLICATION_JSON_VALUE
    )
    public User update(@PathVariable int id, @RequestBody UpdateUserRequest request) {
        return userService.update(id, request);
    }
```

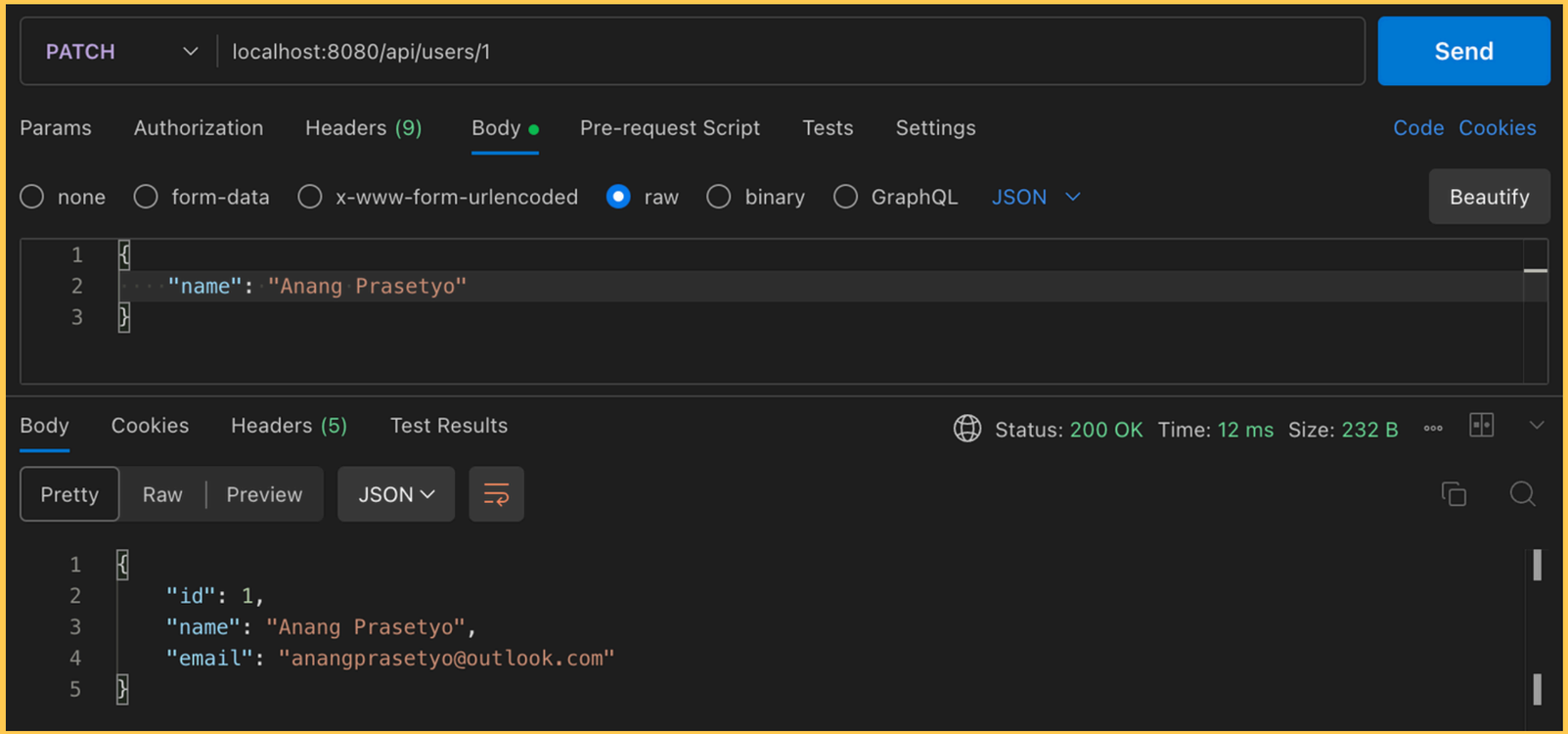
User Controller Method Update digunakan sebagai method untuk menangani request PATCH untuk update data pengguna.

Perbedaan penggunaan method PUT vs PATCH:

- **PUT:** untuk memperbarui seluruh data yang ada pada database.
- **PATCH**: untuk memperbarui sebagian data yang ada pada database.

Testing

Endpoint Update User



Service

User Service Method Delete

```
@Transactional
    public String delete(int id) {
        User user = userRepository.findById(id).get();

        userRepository.delete(user);
        return "Delete sucess";
    }
```

User Service Method Delete digunakan sebagai method untuk menghapus data pengguna berdasarkan ID.

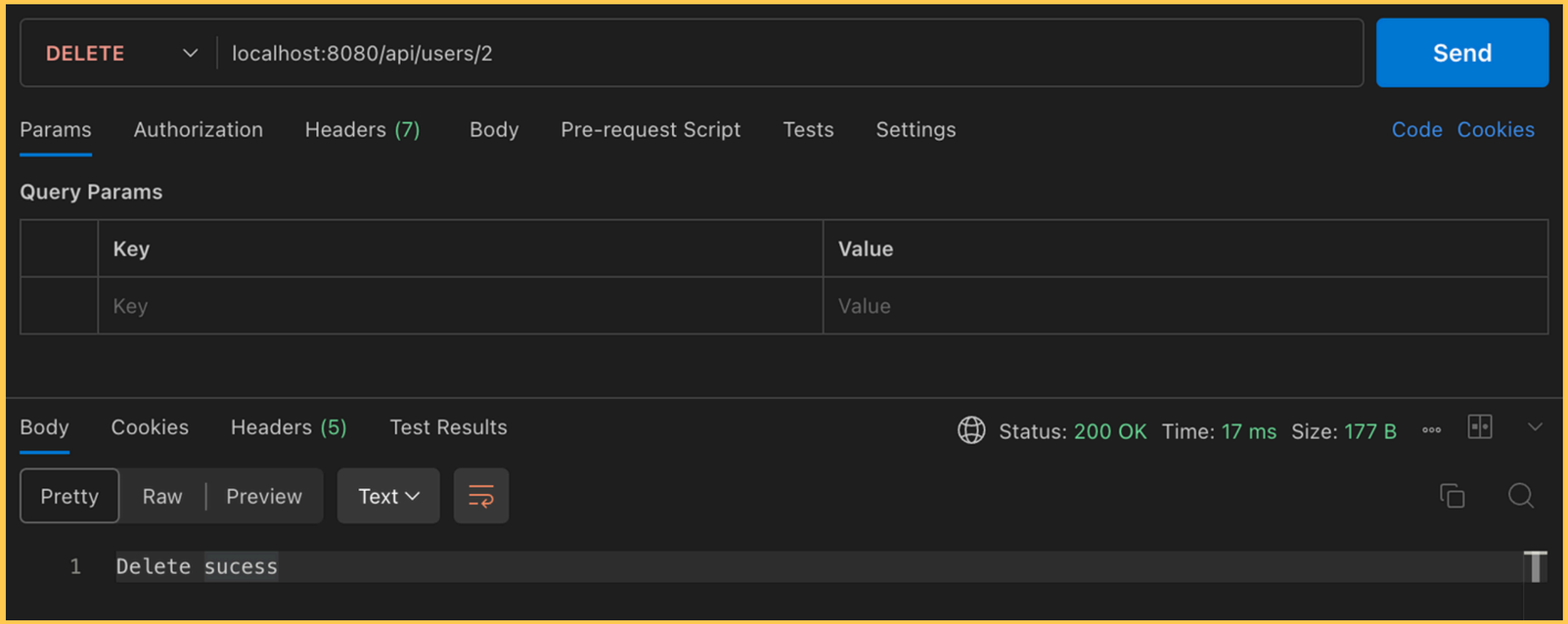
User Controller Method Delete

```
@DeleteMapping("/api/users/{id}")
    public String delete(@PathVariable int id) {
        return userService.delete(id);
    }
```

User Controller Get By Id digunakan sebagai method untuk menangani request DELETE untuk menghapus data pengguna berdasarkan ID.

Testing

Endpoint Delete User



Challenge

Lanjutkan untuk Endpoint Task

REFERENCES

- What is a relational database (RDBMS)? | Google Cloud. (n.d.). Google Cloud. https://cloud.google.com/learn/what-is-a-relational-database
- Spring Data JPA Query. Baeldung. https://www.baeldung.com/spring-data-jpa-query
- Spring Data JPA Query. Baeldung. https://www.geeksforgeeks.org/jpa-cascading-operations/

BINUS CENTER

THANK YOU

Thank you for following session 2 to the end.
See you at the next meeting