German International University
Faculty of Engineering
Department of Media Engineering and Technology

Architecture of Massively Scalable Applications, Spring 2025
Lab Manual 3

PostgreSQL and Spring Boot

1 Introduction

This lab manual implements a simple Spring Boot application with PostgreSQL. The goal is to create models for **User, Product, Transaction, and Cart** with appropriate relationships.

2 Project Setup

2.1 Spring Boot Initialization

Create a new Spring Boot project either through IntelliJ, or `https://start.spring.io/`, and add the following dependencies:

a) Spring Web

b) Rest Repositories

c) Spring Data JPA

d) PostgresSQL Driver

e) Spring Boot Dev Tools

2.2 Running Postgres through Docker

In the terminal write:

```
docker pull postgres
```

Then run the container using:

```
docker run -d -p 5432:5432 -e POSTGRES_PASSWORD=<your_password> --name <container_name> postgres
```

Then inside the container terminal:

```
bash
psql -U postgres
CREATE DATABASE <your_database>;
\c <your_database>
```

2.3 Configuring `application.properties`

```
# PostgreSQL Database Configuration
spring.datasource.url=jdbc:postgresql://localhost:5432/<your_database>
spring.datasource.username=postgres
spring.datasource.password=<your_password>
spring.datasource.driver-class-name=org.postgresql.Driver
# JPA & Hibernate Configuration
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.generate-ddl=true
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.use_sql_comments=true
```

3 Models

3.1 User Model

```
public class User {
    private UUID id;
    private String name;
    private String email;
    private int age;
    // Add relations

    //Constructors and Getters and Setters
```

Add annotations to the class to enforce the following:

a) The model is saved in the `users` table.

b) `id` is the primary key, and is an auto-generated unique identifier.

3.2 Product Model

```
public class Product {
    private int id;
    private String name;
    private double price;

    //Add relations
    // Constructor and Getters and Setters
```

Add annotations to the class to enforce the following:

a) The model is saved in the `products` table.

b) `id` is the primary key, and is an auto-incremented integer.

3.3 Transaction Model

```
public class Transaction {
    private int id;
```

```
    private String description;
    private double amount;
    private String date;

   //Add relations

    // Constructor and Getters and Setters
```

Add annotations to the class to enforce the following:

a) The model is saved in the `transcations` table.

b) `id` is the primary key, and is an auto-incremented integer.

3.4 Cart Model

```
public class Cart {
    private int id;
    private double totalPrice;

    //Add relations

    //Constructor and Getters and Setters
```

Add annotations to the class to enforce the following:

a) The model is saved in the `carts` table.

b) `id` is the primary key, and is an auto-incremented integer.

3.5 Relations

Add the following relations to the previous models. All relations should be bidirectional.

3.5.1 One-to-Many between Users and Transactions:

- Each `User` can have multiple `Transaction` records.
- Each `Transaction` is associated with only one `User`.
- `Transcation` is the owning entity.
- `User`'s `Transactions` are only loaded when needed.

3.5.2 One-to-One between User and Cart

- Each `User` has exactly one `Cart`.
- Each `Cart` belongs to exactly one `User`.
- `Cart` is the owning entity.

3.5.3 Many-to-Many between Products and Cart

- A `Cart` can contain multiple `Product` items.
- A `Product` can be in multiple `Cart` instances.
- `Cart` is the owning entity.
- The pivot table is called `cart_product`.

4 Repository

4.1 UserRepository

```
public interface UserRepository extends JpaRepository<User, UUID> {}
```

4.1.1 Creating a Custom ORM Method

Add a custom ORM method using the naming convention, which returns a user given their email.

4.1.2 Calling a Stored Procedure

To create the stored procedure inside the PostgreSQL container, execute the following SQL command:

```
CREATE OR REPLACE PROCEDURE delete_user(IN userId UUID)
LANGUAGE plpgsql
AS $$
BEGIN
    DELETE FROM users WHERE id = userId;
END;
$$;
```

Add the following method, which should call the `delete_user` stored procedure. Add all necessary annotations.

```
public void deleteUser(UUID userId);
```

4.1.3 Calling a Custom Query

Add the following method which uses the following query

```
UPDATE "users" SET name=:name, email=:email, age=:age WHERE id=:userId
```

to update the fields of the user. Add all necessary annotations.

```
public void updateUser(String name, String email, int age, UUID userId);
```

4.1.4 Calling a Predefined ORM method

We can directly use predefined ORM methods without manually creating them. Thus, none will be added.

5 Service

5.1 UserService

In this class we will create some methods that calls the repository class

- `saveUser(User user)`: Saves a new user entity into the database.
- `findAllUsers()`: Fetches all users stored in the database.
- `findUserById(UUID id)`: Retrieves a user by their unique identifier. If the user does not exist, an exception is thrown.
- `findUserByEmail(String email)`: Retrieves a user by their email.
- `updateUser(String name, String email, int age, UUID userId)`: Updates an existing user details, returning a success or failure message.
- `deleteUser(UUID userId)`: Deletes a user from the database based on their unique identifier.

6 UserController

```
@RestController
@RequestMapping("/users")
public class UserController {}
```

- **Create a User** This method creates a new user by accepting a JSON request body and storing it in the database.

```
@PostMapping("/")
public User createUser(@RequestBody User user) {}
```

- **Get All Users** Retrieves a list of all users stored in the database.

```
@GetMapping("/")
public List<User> getUsers() {}
```

- **Get User by ID** Finds and returns a user based on the provided unique identifier.

```
@GetMapping("/{userId}")
public User getUserById(@PathVariable UUID userId) {}
```

- **Get User by Email** Finds and returns a user based on the provided email.

```
@GetMapping("")
public User getUserByEmail(@Param("email) String email) {}
```

- **Update User** Updates the details of an existing user by modifying their name, email, and age.

```
@PutMapping("/{userId}")
public String updateUser(@PathVariable UUID userId,
                         @RequestBody User user) {}
```

- **Delete User** Removes a user from the database based on their unique identifier.

```
    @DeleteMapping("/{userId}")
    public void deleteUser(@PathVariable UUID userId) {}
```

7 Testing with PostgreSQL

Run the application and use **Postman** or **curl** to test API endpoints. Ensure the database reflects the correct relationships.

8 Solution

You can find the full example solution on `https://github.com/Scalable2025/Lab3`