Hands-On: Hibernate Log Configuration, DDL-Auto Configuration, and JPA Repository Operations with Spring Boot

Prerequisites

- Java 8 or higher installed on your machine
- A Java IDE like IntelliJ IDEA or Eclipse
- A MySQL or PostgreSQL database installed and running

Step 1: Create a Spring Boot Project

First, let's create a new Spring Boot project using the Spring Initializr.

1. Go to the Spring Initializr website.
2. Choose Maven or Gradle as your project type.
3. Enter a Group and Artifact name for your project.
4. Choose the latest stable version of Spring Boot.
5. Add the following dependencies:
    - Spring Data JPA
    - MySQL Connector/J or PostgreSQL Driver (depending on your database)
6. Generate the project and unzip the downloaded archive.

Step 2: Configure Hibernate Logging

To configure Hibernate logging, add the following lines to the `**application.properties**` file:

```
logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE
```

This configuration will enable logging of SQL queries and their parameter bindings.

Step 3: Configure DDL-Auto

To configure DDL-Auto, add the following line to the `**application.properties**` file:

```
spring.jpa.hibernate.ddl-auto=update
```

This configuration will update the database schema automatically based on the entities defined in your application.

Step 4: Create the User Entity

Create a new package called `**com.example.demo.entity**` and create a new class called `**User**`. Add the following code to the class:

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

```
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String firstName;
    private String lastName;
    private String email;
    // getters and setters
}
```

This class defines a User entity with an ID, first name, last name, and email.

Step 5: Create the UserRepository

Create a new package called **`com.example.demo.repository`** and create a new interface called **`UserRepository`**. Add the following code to the interface:

```
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findById(Long id);
    List<User> findByFirstName(String firstName);
}
```

This interface extends the **`JpaRepository`** interface and defines two methods: **`findById()`** and **`findByFirstName()`**. The **`findById()`** method finds a user by its ID, and the **`findByFirstName()`** method finds all users whose first name matches the given value.

Step 6: Create the UserController

Create a new package called **`com.example.demo.controller`** and create a new class called **`UserController`**. Add the following code to the class:

```
@RestController
@RequestMapping("/users")
public class UserController {
    @Autowired
    private UserRepository userRepository;
    @GetMapping("/{id}")
    public ResponseEntity<User> getUserById(@PathVariable Long id) {
        Optional<User> user = userRepository.findById(id);
        if (user.isPresent()) {
            return ResponseEntity.ok(user.get());
        } else {
            return ResponseEntity.notFound().build();
        }
    }
    @PostMapping
    public User createUser(@RequestBody User user) {
        return userRepository.save(user);
    }
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteUserById(@PathVariable Long id) {
        userRepository.deleteById(id);
        return ResponseEntity.noContent().build();
```

```
    }
}
```

This class defines a REST API for the **`User`** entity. The **`getUserById()`** method uses the **`findById()`** method of the **`UserRepository`** interface to find a user by its ID. The **`createUser()`** method uses the **`save()`** method of the **`UserRepository`** interface to save a new user to the database. The **`deleteUserById()`** method uses the **`deleteById()`** method of the **`UserRepository`** interface to delete a user from the database by its ID.

Step 7: Test the API

Start your Spring Boot application and open your browser or a tool like Postman. Try accessing the following endpoints:

- GET **`/users/{id}`**: returns the user with the given ID, if it exists
- POST **`/users`**: creates a new user with the given details
- DELETE **`/users/{id}`**: deletes the user with the given ID, if it exists

You can also try accessing the following endpoints to see the logging of SQL queries:

- GET **`/users`**: returns all users
- GET **`/users?firstName={firstName}`**: returns all users whose first name matches the given value

Conclusion

In this lab, we learned how to configure Hibernate logging, DDL-Auto, and how to use JPA Repository **`findById()`**, defining Query Methods, JPA Repository **`save()`**, and JpaRepository **`deleteById()`** to interact with a database using Spring Boot.