



Spring CRUD JPA

Introduction

The objective of this activity is to create our own Spring CRUD JPA application using a database. The business model I chose for this activity is a game card shop where we offer various types of cards, such as normal cards, rare cards, and super rare cards. As the names suggest, each type of card varies in rarity. In this business, we register users with their first and last names, along with their collection of cards, categorized by type.

1. Entity Definition:

- The application defines an entity class representing a table in the database.
- The entity class uses JPA annotations like `@Entity`, `@Table`, `@Id` to map its fields to the corresponding database columns.

2. DAO Interface Creation:

- A DAO interface is created to define the methods for CRUD operations (e.g., `findAll()`, `findById(int id)`, `save(Usercards usercards)`, `deleteById(int id)`).
- This interface specifies the operations but does not provide the implementation.

3. DAO Implementation Class:

- A DAO implementation class (e.g., `UsercardsDAOJpaImpl`) is created to implement the methods defined in the DAO interface.
- Methods like `findAll()`, `findById(int id)`, `save(Usercards usercards)`, and `deleteById(int id)` are manually implemented using JPA's `EntityManager` methods such as `find()`, `createQuery()`, `merge()`, and `remove()`.

4. EntityManager Injection:

- The `EntityManager` is injected into the DAO implementation class using the `@PersistenceContext` annotation.
- The `EntityManager` provides the API to interact with the persistence context, manage entities, execute queries, and handle transactions.

5. Service Layer:

- A service interface is created (e.g., UsercardsService) that defines the business logic methods.
- A service implementation class (e.g., UsercardsServiceImpl) implements the service interface and delegates the data access operations to the DAO.
- This class is annotated with @Service to indicate that it is a Spring service component.

6. Controller Layer:

- A REST controller class (e.g., UsercardsRestController) is created to handle HTTP requests from clients.
- The controller uses the service layer to perform CRUD operations, receive data from clients, and return responses.

7. Transaction Management:

- Methods in the DAO implementation class or service layer that modify the database (save, deleteById) are annotated with @Transactional.
- This ensures that these methods are executed within a transaction context, providing automatic transaction management (commit or rollback) depending on the success or failure of the operation.

8. Application Configuration:

- The application is configured to use JPA by specifying the data source, JPA properties, and entity manager factory settings in configuration files (application.properties or application.yml).

9. Application Execution:

- The Spring Boot application starts up, initializes the Spring context, configures the JPA settings, and registers beans for DAOs, services, and controllers.

10. Database Interaction:

- JPA (via the EntityManager) interacts with the underlying database to perform CRUD operations.

Prints

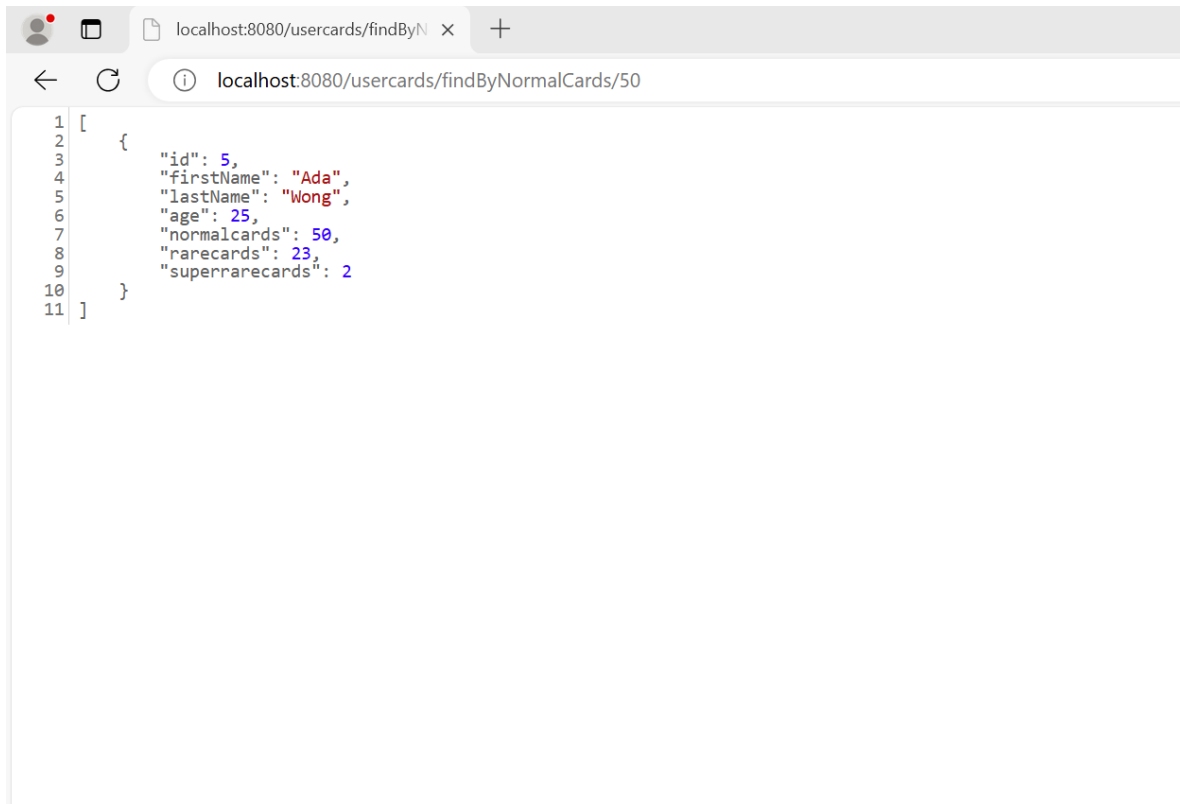
localhost:8080/usercards/all (Everyone on the database)

```
localhost:8080/usercards/all
[
  {
    "id": 1,
    "firstName": "Jose",
    "lastName": "Villarreal",
    "age": 22,
    "normalcards": 33,
    "rarecards": 45,
    "superrarecards": 4
  },
  {
    "id": 2,
    "firstName": "Jack",
    "lastName": "Bauer",
    "age": 44,
    "normalcards": 23,
    "rarecards": 42,
    "superrarecards": 2
  },
  {
    "id": 3,
    "firstName": "Joey",
    "lastName": "Wheeler",
    "age": 17,
    "normalcards": 2,
    "rarecards": 24,
    "superrarecards": 25
  },
  {
    "id": 4,
    "firstName": "Leon",
    "lastName": "Kennedy",
    "age": 25,
    "normalcards": 99,
    "rarecards": 22,
    "superrarecards": 23
  },
  {
    "id": 5,
    "firstName": "Ada",
    "lastName": "Wong",
    "age": 25,
    "normalcards": 50,
    "rarecards": 23,
    "superrarecards": 2
  }
]
```

localhost:8080/usercards/4 (Search by id)

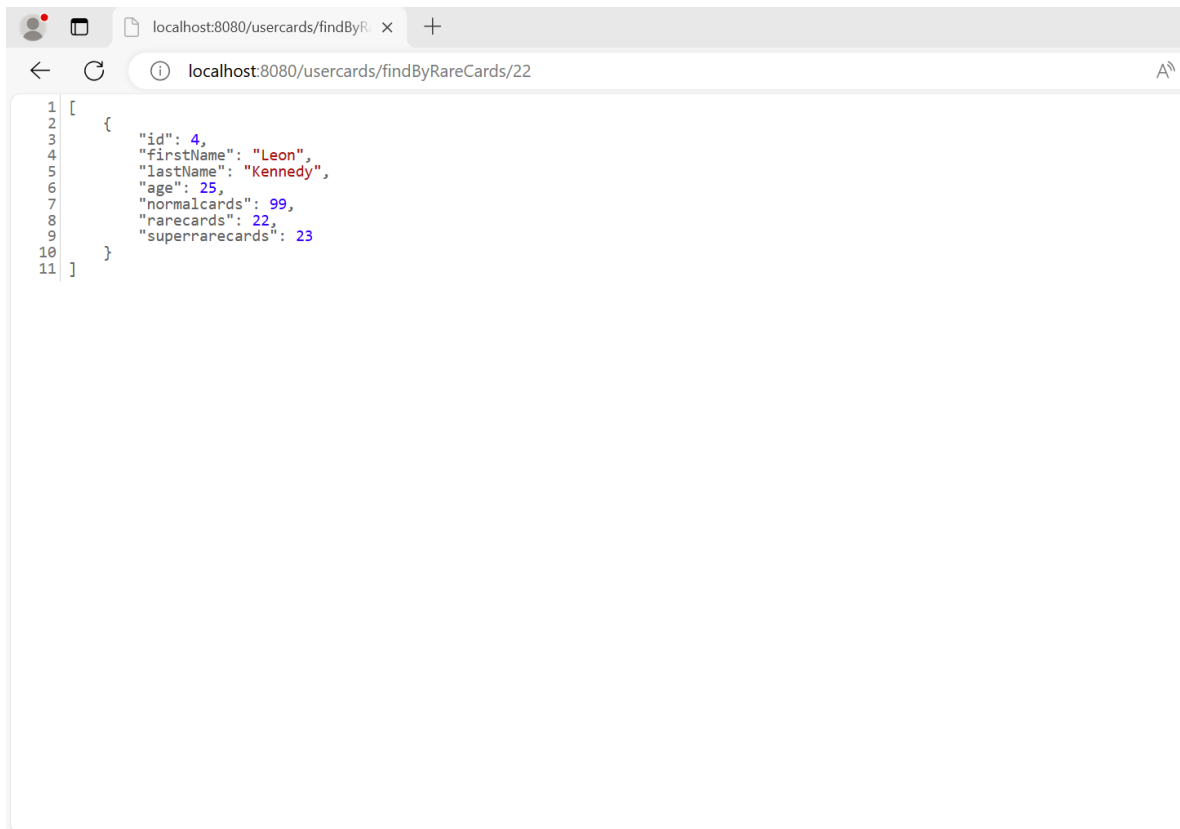
```
localhost:8080/usercards/4
{
  "id": 4,
  "firstName": "Leon",
  "lastName": "Kennedy",
  "age": 25,
  "normalcards": 99,
  "rarecards": 22,
  "superrarecards": 23
}
```

localhost:8080/usercards/findByNormalCards/50 (Find people that have 50 normal cards)



```
1 [
2   {
3     "id": 5,
4     "firstName": "Ada",
5     "lastName": "Wong",
6     "age": 25,
7     "normalcards": 50,
8     "rarecards": 23,
9     "superrarecards": 2
10  }
11 ]
```

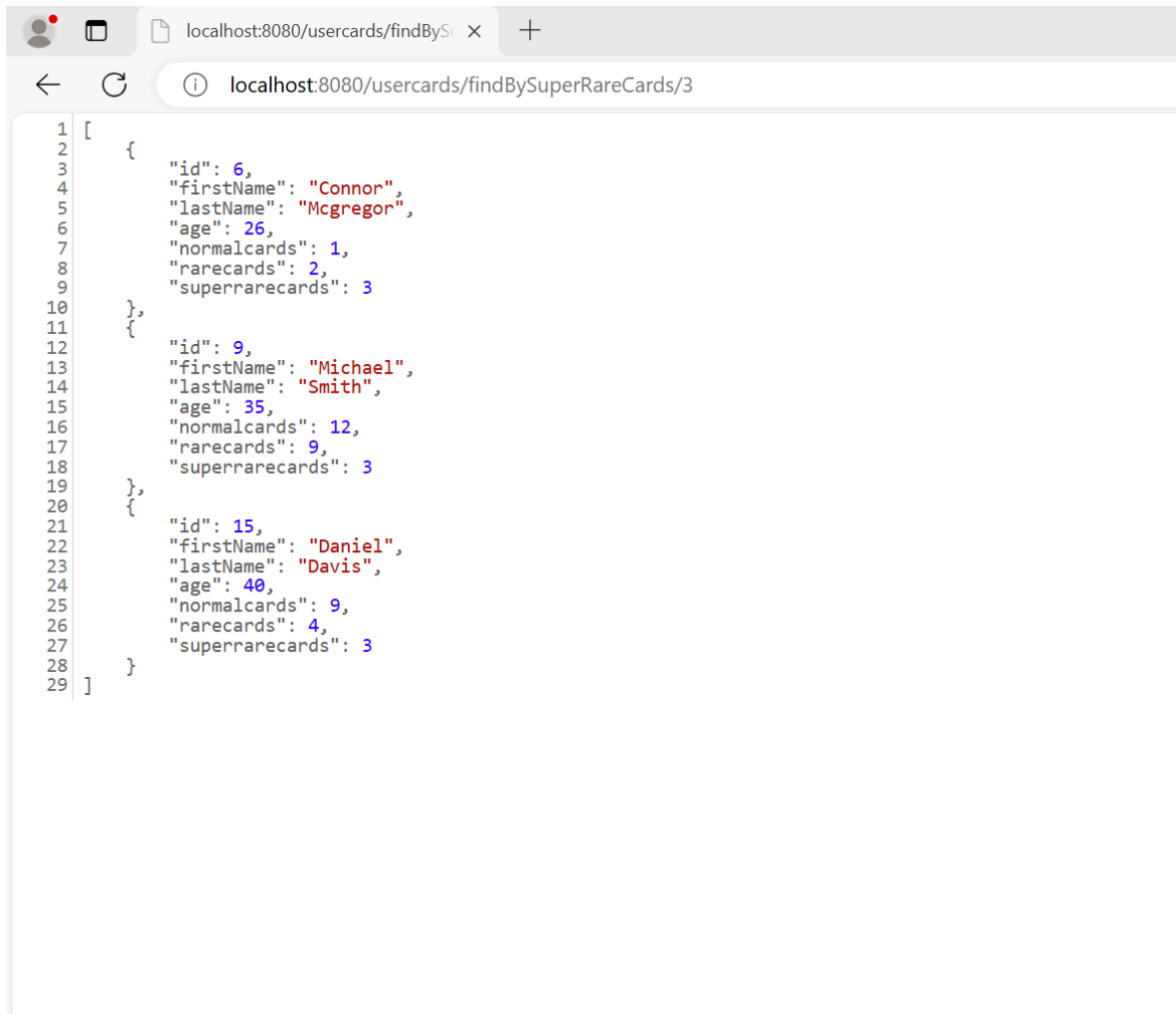
localhost:8080/usercards/findByRareCards/22 (Find people with a specific number of rare cards)



The screenshot shows a web browser window with a single tab titled 'localhost:8080/usercards/findByR...'. The address bar displays 'localhost:8080/usercards/findByRareCards/22'. The main content area shows a JSON array with one object, representing a user. The JSON is formatted with line numbers 1 through 11 on the left. The object contains the following fields: 'id' (4), 'firstName' ('Leon'), 'lastName' ('Kennedy'), 'age' (25), 'normalcards' (99), 'rarecards' (22), and 'superrarecards' (23).

```
1 [
2   {
3     "id": 4,
4     "firstName": "Leon",
5     "lastName": "Kennedy",
6     "age": 25,
7     "normalcards": 99,
8     "rarecards": 22,
9     "superrarecards": 23
10  }
11 ]
```

localhost:8080/usercards/findBySuperRareCards/3 (Find people with a specific amount of super rare cards)



```
1 [
2   {
3     "id": 6,
4     "firstName": "Connor",
5     "lastName": "Mcgregor",
6     "age": 26,
7     "normalcards": 1,
8     "rarecards": 2,
9     "superrarecards": 3
10  },
11  {
12    "id": 9,
13    "firstName": "Michael",
14    "lastName": "Smith",
15    "age": 35,
16    "normalcards": 12,
17    "rarecards": 9,
18    "superrarecards": 3
19  },
20  {
21    "id": 15,
22    "firstName": "Daniel",
23    "lastName": "Davis",
24    "age": 40,
25    "normalcards": 9,
26    "rarecards": 4,
27    "superrarecards": 3
28  }
29 ]
```

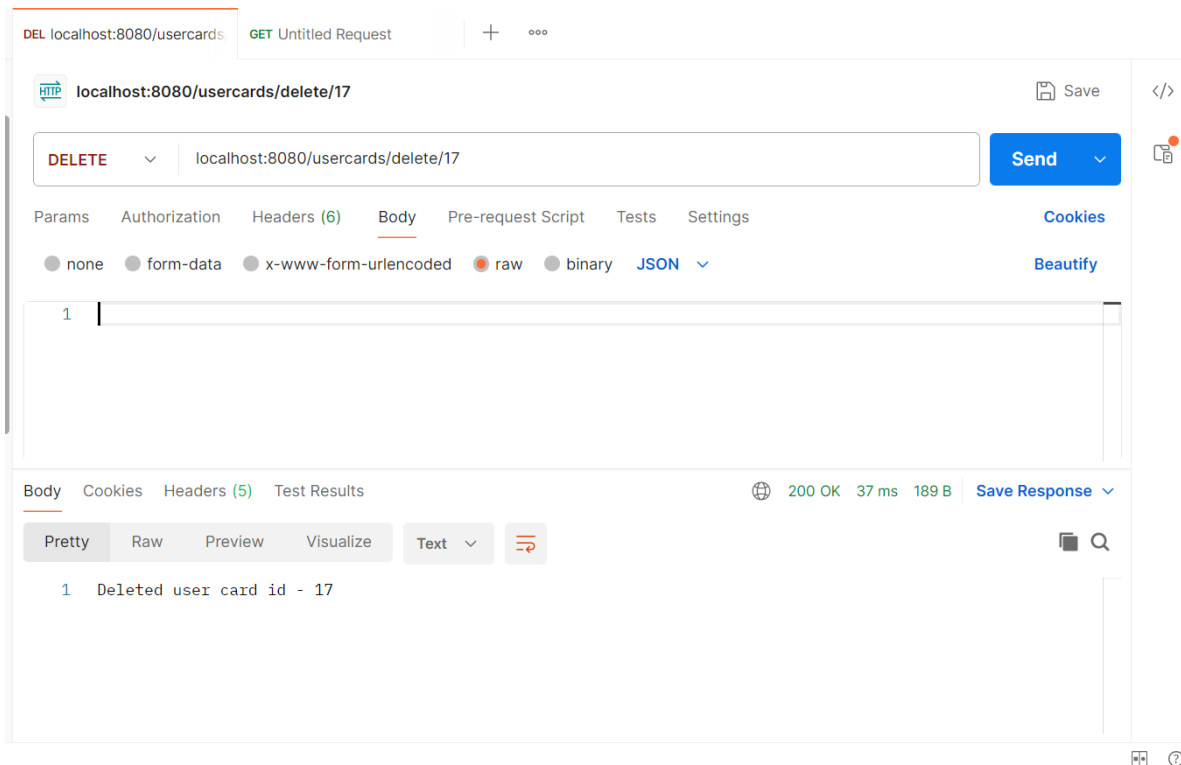
localhost:8080/usercards/save (Add a user to the database with its corresponding id)

The screenshot shows a REST client interface with a POST request to `localhost:8080/usercards/save`. The request body is a JSON object with the following fields:

```
{
  "id": 0,
  "firstName": "Carl",
  "lastName": "Lloyd",
  "age": 23,
  "normalcards": 4,
  "rarecards": 55
}
```

The response is a 200 OK status with a response time of 195 ms and a body size of 270 B. The response body is displayed in the 'Body' tab, showing the same JSON object as the request body.

localhost:8080/usercards/delete/17 (Deletes a user by its id)



Conclusion

This Spring JPA CRUD project demonstrates the effective use of Java Persistence API to manage database operations in a structured and scalable manner. By implementing a game card shop business model, the project showcases how to create, read, update, and delete user data and their associated card collections. It emphasizes the importance of separating concerns between the data access, business logic, and presentation layers, making the application modular, maintainable, and easier to extend. Overall, it provides a solid foundation for understanding database management in a Spring Boot environment.

Appendix (Code)

UsercardsApplication.java

```
package com.example.usercards;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

//Main application to run
```



```

@SpringBootApplication
public class UsercardsApplication {

    public static void main(String[] args) {
        SpringApplication.run(UsercardsApplication.class, args);
    }

}

```

UsercardsRestController.java

```

package com.example.usercards.rest;
import com.example.usercards.entity.Usercards;
import com.example.usercards.service.UsercardsService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/usercards") // Prefix for route
public class UsercardsRestController {

    private final UsercardsService usercardsService;

    // Inject the service using autowired
    @Autowired
    public UsercardsRestController(UsercardsService usercardsService) {
        this.usercardsService = usercardsService;
    }

    // Endpoint to get all user
    @GetMapping("/all")
    public List<Usercards> findAll() {
        return usercardsService.findAll();
    }

    // Endpoint to get user by id
    @GetMapping("/{id}")
    public Usercards findById(@PathVariable int id) {
        return usercardsService.findById(id);
    }

    // Endpoint to get save an user or update an existing one
    @PostMapping("/save")
    public Usercards save(@RequestBody Usercards usercards) {
        return usercardsService.save(usercards);
    }

    // Endpoint to eliminate an existing user by id
    @DeleteMapping("/delete/{id}")
    public String deleteById(@PathVariable int id) {
        usercardsService.deleteById(id);
        return "Deleted user card id - " + id;
    }
}

```

```

    }

    // Endpoint to search usercards by their amount of normal cards
    @GetMapping("/findByNormalCards/{normalCards}")
    public List<Usercards> findByNormalCards(@PathVariable int normalCards) {
        return usercardsService.findByNormalCards(normalCards);
    }

    // Endpoint to search usercards by their amount of rare cards
    @GetMapping("/findByRareCards/{rareCards}")
    public List<Usercards> findByRareCards(@PathVariable int rareCards) {
        return usercardsService.findByRareCards(rareCards);
    }

    // Endpoint to search usercards by their amount of rare cards
    @GetMapping("/findBySuperRareCards/{superRareCards}")
    public List<Usercards> findBySuperRareCards(@PathVariable int
superRareCards) {
        return usercardsService.findBySuperRareCards(superRareCards);
    }
}

```

UsercardsDAO.java

```

package com.example.usercards.dao;

import java.util.List;
import com.example.usercards.entity.Usercards;

public interface UsercardsDAO {

    List<Usercards> findAll(); // List of usercards to find
    Usercards findById(int id);
    List<Usercards> findByNormalCards(int normalCards); // Returns a list if
exists
    List<Usercards> findByRareCards(int rareCards); // Returns a list
    List<Usercards> findBySuperRareCards(int superRareCards); // Returns a list
    Usercards save(Usercards usercards); // Save or update the user
    void deleteById(int id); // Delete by id

}

```

UsercardsDAOImpl.java

```

package com.example.usercards.dao;

import jakarta.persistence.EntityManager;
import jakarta.persistence.PersistenceContext;
import jakarta.persistence.TypedQuery;
import jakarta.transaction.Transactional;
import org.springframework.stereotype.Repository;
import com.example.usercards.entity.Usercards;
import java.util.List;

```

```

@Repository
public class UserscardsDAOJpaImpl implements UserscardsDAO {

    @PersistenceContext
    private EntityManager entityManager;

    @Override
    public List<Userscards> findAll() {
        // Use JPQL to obtain all data from userscards
        TypedQuery<Userscards> query = entityManager.createQuery("from
Userscards", Userscards.class);
        return query.getResultList();
    }

    @Override
    public Userscards findById(int id) {
        // Use EntityManager to search for id
        return entityManager.find(Userscards.class, id);
    }

    @Override
    public List<Userscards> findByNormalCards(int normalCards) {
        // Use JPQL to look up with the specified number of normal cards
        TypedQuery<Userscards> query = entityManager.createQuery(
            "from Userscards where normalcards = :normalCards",
Userscards.class);
        query.setParameter("normalCards", normalCards);
        return query.getResultList();
    }

    @Override
    public List<Userscards> findByRareCards(int rareCards) {
        // Use JPQL to look up with the specified number of rare cards
        TypedQuery<Userscards> query = entityManager.createQuery(
            "from Userscards where rarecards = :rareCards", Userscards.class);
        query.setParameter("rareCards", rareCards);
        return query.getResultList();
    }

    @Override
    public List<Userscards> findBySuperRareCards(int superRareCards) {
        // Use JPQL to look up with the specified number of super rare cards
        TypedQuery<Userscards> query = entityManager.createQuery(
            "from Userscards where superrarecards = :superRareCards",
Userscards.class);
        query.setParameter("superRareCards", superRareCards);
        return query.getResultList();
    }

    @Override
    @Transactional
    public Userscards save(Userscards userscards) {
        // Use EntityManager to save or update userscards (users)
        return entityManager.merge(userscards);
    }
}

```

```

    }

    @Override
    @Transactional
    public void deleteById(int id) {
        // Search the user by id and eliminate it
        Userscards usercards = entityManager.find(Userscards.class, id);
        if (usercards != null) {
            entityManager.remove(usercards);
        }
    }
}

```

Application.properties

```

spring.application.name=userscards
#
# JDBC properties
#
spring.datasource.url=jdbc:mysql://localhost:3306/users_cards
spring.datasource.username=jdbcdaniel
spring.datasource.password=xideral

```

UserscardsService.java

```

package com.example.userscards.service;

import com.example.userscards.entity.Userscards;
import java.util.List;

public interface UserscardsService {
    List<Userscards> findAll();
    Userscards findById(int id);
    List<Userscards> findByNormalCards(int normalCards);
    List<Userscards> findByRareCards(int rareCards);
    List<Userscards> findBySuperRareCards(int superRareCards);
    Userscards save(Userscards usercards);
    void deleteById(int id);
}

```

UserscardsServiceImpl.java

```

package com.example.userscards.service;

import com.example.userscards.dao.UserscardsDAO;
import com.example.userscards.entity.Userscards;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import jakarta.transaction.Transactional;
import java.util.List;

@Service

```

```

public class UserscardsServiceImpl implements UserscardsService {

    private final UserscardsDAO userscardsDAO;

    @Autowired
    public UserscardsServiceImpl(UserscardsDAO userscardsDAO) {
        this.userscardsDAO = userscardsDAO;
    }

    @Override
    public List<Userscards> findAll() {
        return userscardsDAO.findAll();
    }

    @Override
    public Userscards findById(int id) {
        return userscardsDAO.findById(id);
    }

    @Override
    public List<Userscards> findByNormalCards(int normalCards) {
        return userscardsDAO.findByNormalCards(normalCards);
    }

    @Override
    public List<Userscards> findByRareCards(int rareCards) {
        return userscardsDAO.findByRareCards(rareCards);
    }

    @Override
    public List<Userscards> findBySuperRareCards(int superRareCards) {
        return userscardsDAO.findBySuperRareCards(superRareCards);
    }

    @Override
    @Transactional
    public Userscards save(Userscards usercards) {
        return userscardsDAO.save(usercards);
    }

    @Override
    @Transactional
    public void deleteById(int id) {
        userscardsDAO.deleteById(id);
    }
}

```