Spring Boot Coding Assignment Week 15

URL to GitHub Repository: https://github.com/eyu-ars/pet-store

URL to Public Link of your Video:
https://www.dropbox.com/scl/fi/kdnjpcfww7kt54borbvsx/week-15-coding-explanation.mp4?dl=0&rlkey=fnwxbikox82rrbop5f58tlgl7

Overview

In prior weeks you worked to create a Spring Boot/Maven project and instructed Spring JPA to create the pet store tables. Then, you added a controller-layer class, a service-layer class, and a data-layer interface. Finally, you added methods to create a pet store row in the pet store table.

In this week's homework you will expand on the prior week's work to add employees and customers to the pet store. Then, you will complete the remaining CRUD operations on the pet store table:

- Retrieve summary data for all pet stores
- Retrieve a pet store by ID, along with all employee and customer data
- Delete a pet store by ID, along with all employee and customer data

Objectives

This assignment has the following objectives:

1. Demonstrate ability to manage Spring JPA relationships by adding child rows in a one-to-many relationship and a many-to-many relationship.
2. Show continued mastery of Spring JPA by coding the retrieve pet store and delete pet store operations.

Output

This section describes the homework needed for Spring Boot week 15. In these exercises you will write code to add an employee and a customer to a pet store, then you will retrieve pet store data along with the customer and employee information. Finally, you will delete a pet store and show that the customer and employee information is also removed.

In the steps below, you can add code into the existing homework project. **You do not need to create separate repositories for each week's work.**

Here are the instructions for the homework for week 15. You will need to watch and understand at least the contents of the week 13 and week 14 videos before attempting these exercises.

Add store employee

In this section, you will write controller- and service-layer methods to add an employee to an existing pet store. Here are the instructions.

Note: if you get stuck, refer to the Week 15 Coding Assignment solution.

Controller class

In this section, you will create a method in the controller class that allows an employee to be added to a pet store.

1. Create a method in the controller that will add an employee to the employee table. The method should be annotated with @PostMapping and @ResponseStatus.
    a. @PostMapping: This must be configured to allow the caller to send an HTTP POST request to "/pet_store/{petStoreId}/employee", where {petStoreId} is the ID of the pet store in which to add the employee (for example, "/pet_store/1/employee"). Remember that the "/pet_store" part of the HTTP URI is defined at the class level in the @RequestMapping annotation.
    b. @ResponseStatus: This should be configured to return a 201 (Created) status code.
2. The method should be public and return a PetStoreEmployee object.
3. The method should accept the pet store ID and the PetStoreEmployee object as parameters. The pet store ID is passed in the URI and the PetStoreEmployee object is passed as JSON in the request body.
4. Log the request.
5. The method should call the saveEmployee method in the pet store service and should return the results of that method call.

DAO interface

In this section, you will create a new DAO interface to manage CRUD operations on the employee table. It is used by the service method to find an existing Employee row.

Note: if you get stuck, refer to the week 15 Coding Assignment solution.

1. Create a new DAO interface named EmployeeDao.
2. The interface should extend JpaRepository. It is very similar to PetStoreDao but it refers to an Employee object in the Generic. Here is an example:

```
package pet.store.dao;

import org.springframework.data.jpa.repository.JpaRepository;

public interface EmployeeDao extends JpaRepository<Employee, Long> {

}
```

3. The new DAO should be in the pet.store.dao package.

Service class

In this section, you will write the saveEmployee method in the service class. This will add code that is essentially similar to the savePetStore method you added last week. Here is the strategy in pseudocode:

```
@Transactional
public PetStoreCustomer saveEmployee(petStoreId, petStoreEmployee) {
  petStore = findPetStoreById(ID) (already written)
  employeeId = petStoreEmployee.getEmployeeId()
  employee = findOrCreateEmployee(petStoreId, employeeId)

  copyEmployeeFields(customer, petStoreEmployee);

  set petStore in employee
  add employee to pet store list of employees

  dbEmployee = save the employee (employeeDao.save)
  return new PetStoreEmployee(dbEmployee)
}

private Employee findOrCreateEmployee(petStoreId, employeeId) {
  if employeeID is null
  then return new Employee
  else return findEmployeeById(petStoreId, employeeId)
}

private Employee findEmployeeById(petStoreId, employeeId) {
  // Note that findById returns an Optional. If the Optional is
  // empty .orElseThrow throws a NoSuchElementException. If the
  // Optional is not empty an Employee is returned.
  employee = employeeDao.findById(employeeId).orElseThrow(msg)

  if employee pet store ID equals petStoreId
  then return employee
  else throw IllegalArgumentException(msg)
}

private void copyEmployeeFields(employee, petStoreEmployee) {
  copy first name, ID, job title, last name and phone
  from petStoreEmployee
  to employee
}
```

Note: if you get stuck, refer to the week 15 coding assignment solution.

1. Inject the EmployeeDao object into the pet store service class using the @Autowired annotation.
2. Add the @Transactional annotation as a method-level annotation from the org.springframework.transaction.annotation package. Set the readOnly attribute to false.

(Note: a method-level annotation means that the annotation goes right above the method declaration.)

```
@Transactional(readOnly = false)
public PetStoreEmployee saveEmployee(Long petStoreId,
    PetStoreEmployee petStoreEmployee) {
```

3. Create a method named findEmployeeById.
    a. It should take the pet store ID and the employee ID as parameters.
    b. Use the employeeDAO method findById() to return the Employee object. If the employee isn't found throw a new NoSuchElementException().
    c. If the pet store ID in the Employee object's PetStore variable does not match the given pet store ID, throw a new IllegalArgumentException.
    d. If everything is OK, the method should return the Employee object.
4. Create a new method findOrCreateEmployee().
    a. This method should take an employee ID as a parameter (this will be null if the employee is being created), as well as the pet store ID. It will return an Employee object if successful.
    b. If the pet store ID is null, it should return a new Employee object.
    c. If the pet store ID is not null, it should call the method, findEmployeeById().
5. Create a new method copyEmployeeFields.
    a. The method should take an Employee as a parameter and a PetStoreEmployee as a parameter.
    b. Copy all matching PetStoreEmployee fields to the Employee object.
6. Add a method named saveEmployee.
    a. This method should take a pet store ID and a PetStoreEmployee object as parameters. It must return a PetStoreEmployee object.
    b. Call findPetStoreById to find the pet store object.
    c. Call findOrCreateEmployee to retrieve an existing employee or to create a new one.
    d. Call copyEmployeeFields to copy the data in the pet store employee parameter (which ultimately came from the JSON in the HTTP POST request payload) to the Employee object.
    e. Set the PetStore object in the Employee object.
    f. Add the Employee object into the Set of Employee objects in the PetStore object.
    g. Save the employee by calling the save method in the employee DAO.
    h. Convert the Employee object returned by the save method to a PetStoreEmployee object and return it.
7. Test it! Use ARC or the REST client of your choice to send an HTTP PUT request to http://localhost:8080/pet_store/{ID}/employee, where {ID} is the valid primary key value of an existing pet store row in the pet_store table. You can find sample JSON to create an employee in the student resource.

Add store customer

In this section, you will add a customer to an existing pet store. Follow the instructions in the "Add store employee" section, above using the pseudocode instructions modified to use customer instead of employee. Modify the instructions to add the customer. Note that the controller and service methods should use a PetStoreCustomer DTO instead of PetStoreEmployee. (You have already created PetStoreCustomer in the pet.store.controller.model package.)

Make sure that the request used to add a customer is an HTTP POST request sent to http://localhost:8080/pet_store/{ID}/customer, where {ID} is the primary key value of an existing pet store record. You can find sample JSON to add a customer in the student resources.

Note that customer and pet store have a many-to-many relationship. This means that a Customer object has a list of PetStore objects. This means that, in the method findCustomerById, you will need to loop through the list of PetStore objects looking for the pet store with the given pet store ID. If not found, throw an IllegalArgumentException.

Note: if you get stuck, refer to the week 15 Coding Assignment solution.

List all pet stores

In this section you will write the methods to list all pet stores. This method will return summary data for the pet stores. In other words, it will return the pet store data but not the list of customers or employees. Here are the instructions.

Note: if you get stuck, refer to the week 15 Coding Assignment solution.

1. Add a method to the controller that returns List<PetStoreData>. Remember to add the @GetMapping annotation. This annotation does not take a value (i.e., no ID) and the method takes no parameters. Create/call the retrieveAllPetStores() method in the service class.
2. In the service class method retrieveAllPetStores():
    a. Add the @Transactional annotation.
    b. Call the findAll() method in the pet store DAO. Convert the List of PetStore objects to a List of PetStoreData objects.
    c. Remove all customer and employee objects in each PetStoreData object. Here's how to remove the customer and employee objects in a loop if you need a hint.

```
List<PetStoreData> result = new LinkedList<>();

for(PetStore petStore : petStores) {
  PetStoreData psd = new PetStoreData(petStore)

  psd.getCustomers().clear();
  psd.getEmployees().clear();

  result.add(psd);
}
```

    d. Return the summary list.

3. Test the new method by sending a GET request to "/pet_store". You should see all the pet stores returned without customers and employees.

Retrieve a pet store by store ID

In this section you will write the controller and service methods to retrieve a pet store by its ID. You will test this with a valid ID.

Note: if you get stuck, refer to the week 15 Coding Assignment solution.

Follow these instructions:

1. Add a controller method to retrieve a single pet store given the pet store ID. It will be very similar to the retrieve all pet stores method except that the @GetMapping annotation will take the pet store ID that is passed to the method as a parameter. Create/call the method in the service class.
2. In the service class method, add an @Transactional annotation. Call the find by ID method written earlier and convert the results to a PetStoreData object.
3. Test using a valid pet store ID. You should see the pet store data along with all customers and employees.

Delete pet store

In this section, you will delete a pet store. You should see all employees of the pet store deleted as well. In addition, all customer join table records for the pet store should be deleted as well.

Note: if you get stuck, refer to the week 15 Coding Assignment solution.

1. Add the deletePetStoreById() method in the controller.
    a. It should take the pet store ID as a parameter (remember to use @PathVariable).
    b. Add the @DeleteMapping annotation. This should specify that the pet store ID is part of the URI. For example, the URI should be http://localhost:8080/pet_store/{ID}, where {ID} is the ID of the pet store to delete.
    c. Log the request.
    d. Call the deletePetStoreById() method in the service, passing the pet store ID as a parameter.
    e. Return a Map<String, String> where the key is "message" and the value is a deletion successful message.
2. Add the deletePetStoreById() method in the service class.
    a. Call findPetStoreById() to retrieve the PetStore entity.
    b. Call the delete() method in the PetStoreDao interface, passing in the PetStore entity.
3. Test it. When you delete a pet store…
    a. All employees of that pet store should be removed as well.
    b. All customer join table records should be removed but not the actual customer records. If the customer records are removed as well, make sure that the @ManyToMany annotation in the PetStore class has the attribute cascade set to CascadeType.PERSIST and not CascadeType.ALL.

PROMINEO TECH

Observe

Follow the instructions for making a video submission for this coding assignment. Your video should, at a minimum, do the following:

- Show that you can add a pet store employee to the employee table.
- Show that you can add a pet store customer to the customer table.
- Demonstrate that you can retrieve pet store summary information for all pet stores.
- Show that you can retrieve pet store data including employees and customers for a single pet store.
- Demonstrate that you can delete a pet store row, along with the child employee rows and customer join table rows without deleting the customer rows.