Spring Boot Coding Assignment Week 14

URL to GitHub Repository: https://github.com/aljnas/Week-13-Spring-Boot

URL to Public Link of your Video: https://youtu.be/6wVzJH2XrZ8

Overview

Last week you created an Entity Relationship Diagram (ERD), built the pet store schema entities, added Spring JPA configuration, and created the main application class. Then you ran the application and demonstrated that Spring JPA created the tables in the database schema.

Over the next two weeks you will continue to write code to build a Pet Store REST API. There are three types of CRUD operations that can be performed on the pet store schema:

1. Pet store operations
2. Customer operations
3. Employee operations

In these exercises you will write methods for some pet store operations. These methods are:

- Create a pet store
- Modify a pet store
- Correct the response status with a global error handler

Next week you will complete work on the pet store REST API by coding the read and delete pet store operations, as well as some customer and employee operations.

Objectives

This assignment has the following objectives:

1. Demonstrate knowledge of Spring Boot best practices by writing a multi-layer application including the input/output layer, the service layer, and the data layer.
2. Show mastery of Spring JPA by writing a service-layer method that both inserts and updates pet store data.
3. Understand Spring Boot mechanics by using a Data Transfer Object (DTO) for application data input and output (as opposed to using the JPA entities, which will cause your application to go BOOM without special processing).

Output

This section describes the homework needed for Spring Boot week 14. In these exercises, you will write code for some pet store operations and demonstrate that they work correctly. You will find that between Spring Boot and Spring JPA much of the work has been done for you.

In these steps just add the code into the existing project. You do not need to create separate repositories for each week's work.

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

Create a pet store

Before you can complete the other pet store CRUD operations you must first create a pet store. In this section you will write the code that will save pet store data in the pet store schema tables.

A good plan for a REST application is to create three tiers or layers like this:

- *I/O layer*: This is where the HTTP requests enter the application and HTTP responses leave the application. Data is transformed from JSON into Java objects and vice versa. The Java objects are Data Transfer Objects. The controller is in this layer.
- *Service layer*: This layer is where transactions are managed, and data is transformed from the DTOs into the JPA entities.
- *Data Access Object (DAO or data) layer*: This layer creates the SQL statements that perform CRUD operations on the data. With Spring JPA, this layer is created using interfaces. Spring JPA creates the implementing code so that you typically don't have to write any SQL directly.

In this section you will create the data layer (DAO) interface, the service class, and the controller class. Then you will fill in methods in the reverse order: controller first, then the service class, and finally the DAO interface if necessary. This is so the controller can access the service class as an instance variable and the service can access the DAO as an instance variable. You will create them before using them.

You will also create the DTO that will accept and return the pet store data supplied as JSON.

Follow these instructions to save pet store data.

1. In this step you will create the DTO class that Jackson will use to transform the object to and from JSON.
    a. Create a new package named pet.store.controller.model.
    b. Create a new class in this package named PetStoreData. Copy the fields in the PetStore entity into this class. Do not copy any JPA annotations. Add the @Data and @NoArgsConstructor as class-level annotations. Both are found in the lombok package.
    c. Create the DTO class PetStoreCustomer as either an inner class inside PetStoreData or as a separate class in the pet.store.controller.model package. If you create the class as an inner class, be sure to make the class public and static so that the class can be used as a separate DTO. Copy the fields from the Customer entity into this class. Do not copy the petStores field, which will remove the recursion required by JPA. Add the @Data and @NoArgsConstructor as class-level annotations.
    d. Create the DTO class PetStoreEmployee as either an inner class inside PetStoreData or as a separate class in the pet.store.controller.model package. If you create the class as an inner class, be sure to make the class public and static so that the class can be used as a separate DTO. Copy the fields from the Employee entity into this class. Do not copy the petStore field to avoid recursion. Add the @Data and @NoArgsConstructor as class-level annotations.
    e. In the PetStoreData class, change the data type of the customers field to PetStoreCustomer. Change the data type of the employees field to PetStoreEmployee.
    f. In the PetStoreData class add a constructor that takes a PetStore as a parameter. Set all matching fields in the PetStoreData class to the data in the PetStore class. Also set the

employees and customers fields to the respective PetStoreCustomer and PetStoreEmployee objects. These are Sets so use loops for this.

g. Add the PetStoreCustomer and PetStoreEmployee classes as inner or external classes. For PetStoreCustomer, add a constructor that takes a Customer object. For PetStoreEmployee, add a constructor that takes an Employee object.

2. In this step you will create the data layer interface. This interface extends JpaRepository, which is a Spring JPA-provided interface. JpaRepository is in the org.springframework.data.jpa.repository package.
    a. Create a new package named pet.store.dao.
    b. In this package, create a new interface named PetStoreDao. The interface should extend JpaRepository<PetStore, Long>.
    c. Add the import statement for PetStore.

3. In this step, you will create the service class. You will add methods to this class in a later step.
    a. Create a new package named pet.store.service.
    b. In this package, create a new class named PetStoreService.
    c. Add the class-level annotation @Service from the org.springframework.stereotype package.
    d. Add a PetStoreDao object named petStoreDao as a private instance variable. Annotate the instance variable with @Autowired so that Spring can inject the DAO object into the variable.

4. In this step you will create the pet store controller class in a new package. This will allow Spring to map HTTP requests to specific methods. The URI for every request that is mapped to the controller must start with "/pet_store". You can control the class-level mapping by specifying "/pet_store" as the value inside the @RequestMapping annotation.
    a. Create a controller class named PetStoreController in the pet.store.controller package. This class should have the following class-level annotations:
        i. @RestController – This tells Spring that this class is a REST controller. As such it expects and returns JSON in the request/response bodies. The default response status code is 200 (OK) if you don't specify something different. And finally, this annotation tells Spring to map HTTP requests to class methods. The annotation is in the org.springframework.web.bind.annotation package.
        ii. @RequestMapping("/pet_store") – This tells Spring that the URI for every HTTP request that is mapped to a method in this controller class must start with "/pet_store". This annotation is in the org.springframework.web.bind.annotation package.
        iii. @Slf4j – This is a Lombok annotation that creates an SLF4J logger. It adds the logger as an instance variable named log. Use it like this:

            log.info("This is a log line"):

            This annotation is in the lombok.extern.slf4j package.

    b. Autowire (inject) the PetStoreService as an instance variable.

    c. Create a public method that maps an HTTP POST request to "/pet_store". The response status should be 201 (Created). Pass the contents of the request body as a parameter (type PetStoreData) to the method. (Use @RequestBody.) The method should return a

PetStoreData object. Log the request. Call a method in the service class (savePetStore) that will insert or modify the pet store data.

5. Now we will fill in the save method in the service class. In the service class, the savePetStore method should take a PetStoreData object as a parameter and return a PetStoreData object. The method should:
    a. Call findOrCreatePetStore(petStoreId). This method returns a new PetStore object if the pet store ID is null. If not null, the method should call findPetStoreById, which returns a PetStore object if a pet store with matching ID exists in the database. If no matching pet store is found, the method should throw a NoSuchElementException with an appropriate message.
    b. Call copyPetStoreFields(). This method takes a PetStore object and a PetStoreData object as parameters. Matching fields are copied from the PetStoreData object to the PetStore object. Do not copy the customers or employees fields.
    c. Call the PetStoreDao method save(petStore). Return a new PetStoreData object created from the return value of the save() method.

6. Test the insert operation using the Advanced Rest Client (ARC) or another REST client of your choosing. Send a POST request with pet store data to the application and verify the results using DBeaver. You can find sample JSON used to create a pet store in the student resources.

    Add at least two pet stores to the database.

    Before starting the application, in src/main/resources/application.yaml, set the value of spring.jpa.hibernate.ddl-auto to update. This will prevent the data tables from being recreated each time the application starts.

Modify a pet store

In this section, you will modify one of the pet store objects added in the prior step. This involves sending an HTTP PUT request to the running application. Here are the steps needed to accomplish this:

1. In the pet store controller, add a public method to update the pet store data. This method should:
    a. Have an @PutMapping annotation to map PUT requests to this method. The annotation should specify that a pet store ID is present in the HTTP request URI.
    b. Return a PetStoreData object.
    c. The method parameters are the pet store ID and the pet store data. Don't forget the appropriate annotations to read the store ID from the request URI and the store data from the request body.
    d. Set the pet store ID in the pet store data from the ID parameter.
    e. Log the method call.
    f. Call the savePetStore method in the service class.
2. Test that you can send a PUT request to the application and modify store data using a valid pet store ID. See the student resources for sample JSON to use to modify pet store data.

Write a global error handler

In the previous exercise, you modified an existing pet store using a valid pet store ID. What happens when you try to modify a pet store using an invalid pet store ID? (Try it.) You should see that a 500 (Internal Server Error) status is returned, and the exception is logged with the full stack trace. This is not enough information for the client to make an informed decision as to what action to take. If a 404 (Not Found)

status was returned, the client could display an error message stating that the pet store with the given ID was not found.

In Spring Boot, this task is handled by a global error handler. In this exercise, you will write the global error handler and verify that attempting to modify a pet store with an invalid ID results in the correct 404 (Not Found) status.

In the videos, a global error handler was created that returned a detailed error response. For this exercise, it is not necessary to write code that complex (but you can if you want to). This exercise will have you create a simple error handler. If you want to write one that is similar to the one shown in the videos, feel free!

Here are the instructions. You can ignore them if you are writing a more complex error handler.

1. Create a new package named pet.store.controller.error. In this package, create a new class named GlobalErrorHandler.
    a. Add the class-level annotations @RestControllerAdvice (from the org.springframework.web.bind.annotation package). Also add the @Slf4j annotation (from the lombok.extern.slf4j package).
    b. Add a handler method for NoSuchElementException. The handler method should specify a response status of 404 (Not Found). It should return Map<String, String> and take a NoSuchElementException as a parameter. (Spring will fill in the parameter when it calls the method.)
    c. Log the error using the SLF4J logger. You do not need to log the stack trace. (SLF4J stands for Simple Logging Façade for Java. It implements the Façade Design Pattern so that you can write code conforming to the SLF4J API and use any underlying logging framework that has an SLF4J bridge. By default, Spring Boot uses the Logback logging framework to perform the logging. In this sense, logging is similar to JDBC in which you write code conforming to the JDBC API and the driver class maps SQL requests to the needs of the specific database.)
    d. Return a map with a single entry. The key is "message" (with the quotes) and the value is the result of calling toString() on the exception parameter.
2. Send an HTTP PUT request to the application using an invalid pet store ID. You should see the message as JSON and the result status should be 404 (Not Found).

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 to the data tables by sending an HTTP POST request to the running application.
- Show that you can modify an existing pet store by sending an HTTP PUT request to the running application.
- Demonstrate that an incorrect pet store ID supplied to a modify operation results in a 404 (Not Found) status.