Rubric Final Project

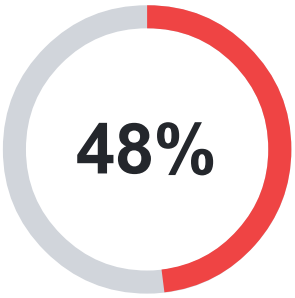
Keep Pushing Forward!

Keep going! Mistakes are opportunities to learn and grow. Review your answers and **try again** to achieve your best result.

Required passing grade: 50%

Status: **Failed**

Final Score: 29 / 60 (48%)



Provide Assignment Feedback

Question 1

Score: **2/4**

Provide a publicly accessible link to the `Customer.java` file from your code repository.

For example: `https://github.com/username/java-database-final/blob/main/back-end/src/main/java/com/project/code/Model/Customer.java`

https://github.com/Skriesten/java-database-final/blob/main/back-end/src/main/java/com/project/code/Model/Customer.java

The submission includes a well-defined Customer class with a proper one-to-many relationship with OrderDetails. The relationship is correctly annotated using @OneToMany with mappedBy set to 'customer', and the fetch type is explicitly set to FetchType.EAGER. The import statements are correctly included, and the class declaration is appropriate. However, the relationship field 'orderDetails' is commented out, which needs to be uncommented to ensure functionality. Additionally, while the basic structure and annotations are correct, ensure that the related OrderDetails class is set up correctly to maintain the bidirectional relationship if needed. Overall, the submission meets the criteria, but make sure to activate the relationship by uncommenting the field.

Question 2

Score: **7/7**

Provide a publicly accessible link to the `OrderDetails.java` file from your code repository.

For example: `https://github.com/username/java-database-final/blob/main/back-end/src/main/java/com/project/code/Model/OrderDetails.java`

---

https://github.com/Skriesten/java-database-final/blob/main/back-end/src/main/java/com/project/code/Model/OrderDetails.java

Accuracy: The URL provided is functional and leads to the correct file, which includes the OrderDetails class. The class contains a store field annotated with both @ManyToOne and @JoinColumn(name = "store_id"), meeting the task requirements.

Functionality: The OrderDetails class is well-implemented, with the store field correctly set up to manage the relationship with the Store entity. The date field is also correctly implemented using LocalDateTime.

Efficiency: The approach taken is efficient, with the necessary fields and annotations correctly defined to ensure proper functionality.

Style: The code is clear and readable, with appropriate use of annotations and consistent formatting. The use of comments to explain the purpose of each field and annotation is helpful.

Practices: The submission adheres to best practices, including the use of JPA annotations for entity relationships and JSON serialization considerations with @JsonManagedReference.

Strengths: The submission demonstrates a solid understanding of JPA entity relationships and the use of annotations to manage them effectively. The inclusion of both no-argument and parameterized constructors adds to the class's robustness. Overall, this is a well-executed implementation that meets all the specified criteria.

Question 3

Score: **4/8**

Provide a publicly accessible link to the `StoreController.java` file from your code repository.

For example: `https://github.com/username/java-database-final/blob/main/back-end/src/main/java/com/project/code/Controller/StoreController.java`

---

https://github.com/Skriesten/java-database-final/blob/main/back-end/src/main/java/com/project/code/Controller/StoreController.java

Accuracy: The URL provided leads to a functional StoreController class in a public GitHub repository, which meets the task requirements. The class exposes a GET endpoint for validating store existence using the store ID, as required by the assignment.

Functionality: The validateStore method uses the existsById method from the repository to check for the store's existence, which is appropriate and functional for this task.

Efficiency: The method efficiently checks the existence of a store by leveraging the repository's existsById method, which is a standard practice for such tasks.

Style: The code is clear and readable, with appropriate use of annotations and method naming conventions. However, consider adding JavaDoc comments to improve maintainability and documentation.

Practices: The use of dependency injection via @Autowired and the separation of concerns between the controller, service, and repository layers demonstrate good adherence to best practices in Spring Boot applications.

Strengths: The submission effectively demonstrates the student's understanding of exposing RESTful endpoints and using repository methods to interact with the database. The code structure and logic are sound and meet the requirements of the assignment.

Question 4

Score: **3/6**

Provide a publicly accessible link to the `ProductController.java` file from your code repository.

For example: `https://github.com/username/java-database-final/blob/main/back-end/src/main/java/com/project/code/Controller/ProductController.java`

---

https://github.com/Skriesten/java-database-final/blob/main/back-end/src/main/java/com/project/code/Controller/ProductController.java

Accuracy: The GET /product/{id} endpoint is implemented correctly. It accepts a product ID, retrieves the product using the repository layer, and returns a well-structured response. Error handling is also present, as it throws a ResponseStatusException if the product is not found.

Functionality: The method works as expected. It retrieves a product by its ID and handles cases where the product is not found by returning a 404 status.

Efficiency: The approach is efficient in terms of using the repository layer to fetch data. However, there is a redundant call to findById(id) which could be optimized by storing the result in a variable.

Style: The code is clear and readable. The use of annotations and structured responses enhances understanding.

Practices: The implementation adheres to best practices, including proper error handling and use of HTTP status codes.

Strengths: The method demonstrates a good understanding of RESTful principles and error handling in Spring Boot applications.

Question 5

Score: **0/8**

Provide a publicly accessible link to the `InventoryController.java` file from your code repository.

For example: `https://github.com/username/java-database-final/blob/main/back-end/src/main/java/com/project/code/Controller/InventoryController.java`

---

https://github.com/Skriesten/java-database-final/blob/main/back-end/src/main/java/com/project/code/Controller/InventoryController.java

The InventoryController class does not expose a GET /filter/{category}/{name}/{storeId} endpoint with conditional logic for filtering. The provided code does not include an endpoint with the specified path or the required filtering logic. Additionally, there is no GET /validate/{quantity}/{storeId}/{productId} endpoint implemented to validate available quantity. The class lacks these specific endpoints, which are necessary to meet the scoring criteria. To improve, the student should implement both endpoints with the required conditional logic and validation checks as per the instructions.

Question 6

Score: **1/7**

Provide a publicly accessible link to the `ReviewController.java` file from your code repository.

For example: `https://github.com/username/java-database-final/blob/main/back-end/src/main/java/com/project/code/Controller/ReviewController.java`

---

https://github.com/Skriesten/java-database-final/blob/main/back-end/src/main/java/com/project/code/Controller/ReviewController.java

The GET /{storeId}/{productId} method is implemented, but it does not dynamically retrieve customer names using the CustomerRepository. The current implementation only fetches reviews based on storeId and productId without including customer names in the response. Additionally, it lacks handling for the 'customer not found' case, which is part of the scoring criteria. To improve, consider using the customerRepository.findById(review.getCustomerId()) to retrieve and include customer names in the response. This enhancement would align with best practices for providing comprehensive review details. Furthermore, the code should handle cases where a customer is not found by providing a default value like 'Unknown'.

Question 7

Score: **4/6**

Provide a publicly accessible link to the `OrderService.java` file from your code repository.

For example: `https://github.com/username/java-database-final/blob/main/back-end/src/main/java/com/project/code/Service/OrderService.java`

---

https://github.com/Skriesten/java-database-final/blob/main/back-end/src/main/java/com/project/code/Service/OrderService.java

Accuracy: The provided URL is correct and leads to the OrderService.java file. The submission meets the task requirements by providing a publicly accessible link.

Functionality: The code within the URL contains methods relevant to the order processing system, including saving order details and handling inventory. However, the createOrderItems method contains incorrect logic for setting the product and inventory, which could lead to runtime errors.

Efficiency: The approach generally follows a standard pattern for handling orders and inventory. However, the createOrderItems method could be optimized by ensuring correct retrieval and update of inventory items.

Style: The code is mostly clear and readable, with comments explaining the purpose of each method. However, some variable names could be more descriptive to enhance clarity, such as renaming store_id to storeId to follow Java naming conventions.

Practices: The code demonstrates good practices by using repository patterns for data access. However, there is a need to ensure that the inventory update logic is correctly implemented and tested.

Strengths: The code effectively uses repositories to manage data persistence and includes exception handling for missing stores. The structure of the class and methods shows a good understanding of object-oriented principles.

Question 8

Score: **3/6**

Provide a publicly accessible link to the `ServiceClass.java` file from your code repository.

For example: `https://github.com/username/java-database-final/blob/main/back-end/src/main/java/com/project/code/Service/ServiceClass.java`

---

https://github.com/Skriesten/java-database-final/blob/main/back-end/src/main/java/com/project/code/Service/ServiceClass.java

The 'validateInventory' method is implemented and checks for an existing inventory entry using both product and store IDs. It retrieves the product and store from their respective repositories and verifies if an inventory

record exists using the 'findByProductIdandStoreId' method. The logic correctly returns 'false' if no inventory record is found and 'true' otherwise. However, there are areas for improvement, such as ensuring that repository methods handle null checks and potential exceptions more gracefully. Additionally, the method could be optimized by directly using repository methods that check for existence rather than retrieving full entities. Overall, the implementation meets the criteria for checking the existence of an inventory entry for a product-store combination effectively.

Question 9

Score: **2/4**

Provide a publicly accessible link to the `InventoryRepository.java` file from your code repository.

For example: `https://github.com/username/java-database-final/blob/main/back-end/src/main/java/com/project/code/Repo/InventoryRepository.java`

---

https://github.com/Skriesten/java-database-final/blob/main/back-end/src/main/java/com/project/code/Repo/InventoryRepository.java

Accuracy: The URL provided is functional and leads to the correct file InventoryRepository.java. The file contains methods relevant to the assignment instructions. However, there is no use of the @Query annotation for fetching inventory by productId and storeId.

Functionality: The method findByProductIdandStoreId attempts to fetch inventory using both productId and storeId, but it does not use the @Query annotation as required by the rubric for the first criterion.

Efficiency: The repository interface extends JpaRepository, which is efficient for CRUD operations. However, the custom query method could be more efficient if it correctly used @Query for complex queries.

Style: The code is clear and follows standard Java naming conventions. The use of comments to indicate sections and instructions is helpful for understanding.

Practices: The interface adheres to best practices by extending JpaRepository and using annotations like @Repository and @Transactional. However, the custom query should use @Query to demonstrate proper use of JPQL or native queries.

Strengths: The repository interface is well-structured and follows good practices for extending JpaRepository. The methods defined are relevant to the inventory management context, showing an understanding of the domain requirements.

Question 10

Score: **3/4**

Provide a publicly accessible link to the `ProductRepository.java` file from your code repository

For example: `https://github.com/username/java-database-final/blob/main/back-end/src/main/java/com/project/code/Repo/ProductRepository.java`

---

https://github.com/Skriesten/java-database-final/blob/main/back-end/src/main/java/com/project/code/Repo/ProductRepository.java

Accuracy: The submission includes a method findByCategory(String category) which correctly filters products by category, fulfilling the criteria for the first rubric. However, there is no method that combines both store ID and category as filters, which affects the second rubric.

Functionality: The findByCategory method should work as expected for filtering products by category. However, the functionality to filter by both store ID and category is missing.

Efficiency: The approach taken for filtering by category is efficient using Spring Data JPA's derived query method. However, to improve efficiency for combined filtering, consider adding a method like findByStoreIdAndCategory.

Style: The code is clear and well-organized, with appropriate use of comments to explain the purpose of each method.

Practices: The use of Spring Data JPA and derived query methods adheres to best practices for maintainability and modularity. However, adding a combined filter method would enhance clarity and functionality.

Strengths: The repository demonstrates a good understanding of using Spring Data JPA for defining repository methods with clear and concise code. The use of derived query methods is a strength in this submission.

Report an issue

Retake Assignment