Title: Building an E-commerce Application using Spring MVC with ORM for CRUD Operations

Introduction:

In this assignment, you will learn how to develop an E-commerce application using the Spring MVC framework and Object-Relational Mapping (ORM) for performing CRUD (Create, Read, Update, Delete) operations. The assignment will guide you through the implementation of various features of the application, including managing products, categories, and customer orders.

Assignment Tasks:

Task 1: Setting up the Project

Create a new Spring MVC project: Set up a new project using a build tool like Maven or Gradle and configure it as a Spring MVC project.

Configure dependencies: Add the necessary dependencies to your project's build file, such as Spring MVC, Hibernate (or any other ORM framework of your choice), and any additional libraries required for database connectivity or UI development.

Set up the database connection: Configure the database connection properties in a configuration file (e.g., application.properties or application.yml) to establish the connection between your application and the database.

Task 2: Database Design and Mapping

Design the database schema: Create a well-defined database schema for your E-commerce application, considering entities like products, categories, and customer orders. Identify the attributes and relationships between these entities.

Implement entity classes: Create Java entity classes using ORM annotations (e.g., JPA annotations for Hibernate) to represent the database tables. Map each entity's attributes to their respective columns in the database.

Establish relationships between entities: Use ORM associations (e.g., @OneToMany, @ManyToOne, @ManyToMany) to define relationships between entities based on their cardinality (one-to-one, one-to-many, many-to-many). Add appropriate annotations to the entity classes to reflect these relationships.

Task 3: Product Management

Create a controller: Implement a controller class responsible for handling requests related to product management.

Implement CRUD operations: Create methods within the controller for handling CRUD operations on products, using Spring MVC annotations like @GetMapping, @PostMapping, @PutMapping, and @DeleteMapping. These methods will handle requests for creating, reading, updating, and deleting products.

Utilize ORM techniques: Use ORM techniques (such as Hibernate's SessionFactory or EntityManager) within the controller methods to interact with the database and perform the necessary CRUD operations on product data.

Task 4: Category Management

Create a controller: Set up a separate controller class for managing categories.

Implement CRUD operations: Define methods within the category controller to handle CRUD operations on categories, following the same approach used for product management.

Apply ORM techniques: Utilize ORM techniques to interact with the database and perform CRUD operations on category data. Establish a relationship between products and categories using ORM associations, ensuring proper mapping and cascade options.

Task 5: Customer Order Management

Create a controller: Develop a controller class for managing customer orders.

Implement CRUD operations: Implement methods within the order controller to handle CRUD operations on customer orders.

Use ORM techniques: Utilize ORM techniques to interact with the database and perform CRUD operations on customer order data. Establish relationships between orders, products, and customers using ORM associations, ensuring proper mapping and cascade options. This allows you to retrieve orders along with associated products and customer information.

Task 6: User Interface

Design user interfaces: Create user interface templates using technologies like JSP (JavaServer Pages) or Thymeleaf. Design separate pages for managing products, categories, and customer orders.

Implement UI logic: Implement the necessary logic in the user interface to perform CRUD operations. Use HTML forms to capture user input and send requests to the corresponding controller methods.

Implement validation and error handling: Incorporate validation on the user interface to ensure data integrity. Handle any errors or exceptions gracefully by displaying appropriate error messages to the user.

Integrate UI with controllers: Connect the user interface templates with the respective controller methods by using appropriate action mappings and form submissions.

Task 7: Testing and Deployment

Write unit tests: Create unit tests using testing frameworks like JUnit or TestNG to ensure the functionality of your CRUD operations for products, categories, and customer orders. Test various scenarios such as creating new records, updating existing records, and deleting records.

Perform integration testing: Conduct integration testing to validate the interaction between the controllers, service layer (if applicable), and the database. Verify that the data is correctly persisted and retrieved from the database.

Deploy the application: Deploy the application on a local server, such as Apache Tomcat or any other server of your choice. Ensure that the application runs smoothly and all CRUD operations can be performed successfully.

Bonus Task (optional):

If you choose to attempt the bonus task, you can implement additional features to enhance your E-commerce application. Some ideas include:

User authentication and authorization: Implement a login and registration system to secure access to the application. Differentiate user roles and restrict certain operations based on user permissions.

Shopping cart management: Allow users to add products to a shopping cart, modify quantities, and proceed to checkout.

Payment integration: Integrate a payment gateway or implement a mock payment system to handle transactions and process payments.

Order tracking and status management: Implement functionality to track the status of customer orders and update their progress.