E-commerce Service (Main Application)

A. Problem Statement:

Design and implement the e-commerce service application to handle product management, including categories, cart functionality, and order processing. The e-commerce service will provide comprehensive features for managing product catalogues, allowing seamless organisation and categorisation of products. Users can browse and add products to their cart, with the service facilitating smooth cart management and updates. Additionally, the e-commerce service will handle the processing of orders.

B. Features of the User Application:

The application should allow to:

- **Implement Product Operations:** Allow admin users to create, update and delete products and create APIs to view all and individual products.
- **Create product categories:** Design a category-based system where every product belongs to a particular category, making categorisation easier.
- **Cart and Cart management:** Enable users to have their carts in which they add and remove products and purchases.
- **Implement Order Placement:** Enable users to place orders and generate order summaries.
- **Authorisation and Access Control:** Ensure users can only access features and resources based on their assigned roles and permissions.
- **Error Handling and Exception Management:** Develop robust error handling mechanisms and implement exception management to handle unexpected scenarios gracefully.

C. Steps to Create the Project:

Step 1: Setup a Spring Boot Project:

1. Create a new Spring Boot project using your preferred IDE or start.spring.io.
2. Include the necessary dependencies, such as Spring Web, Spring Data JPA, and Spring Security.

Step 2: Defining entities:

1. **Product:** Create a Product model class with attributes like id, name, price and description.
2. **Category:** Create an Order class with attributes such as ID, user ID, products, total price, shipping address, etc.
3. **Cart and CartItem:** Create a Cart and CartItem class with attributes like ID, user ID, list of products, etc.

   *(Please refer to **"Structure of the cart and cart item"** pdf for more details on cart and cartItem structure)*
4. **Order and OrderItem:** Create an Order and OrderItems model class with attributes like id, customer, price, etc.

   *(Please refer to **"Structure of the order and order item"** pdf for more details on cart and cartItem structure)*
5. **Address**: Create an Address model class with attributes such as ID, city, street, userId, firstName, lastName, etc.
6. **Product:** Create Product and ProductData classes.
7. **DataObject:** Create a data object which will help to populate product data for a specific category.

Step 3: Mapping for entities:

1. Map the relationship between the **Cart** and **Product** entities, as a cart can contain multiple products.
2. Map the relationship between **Product** and **Category** entities, as a product will belong to a category.
3. Map the relationship between **Cart** and **Order** entities, as an order will be placed for the products belonging to the cart.

Step 4: APIs - You need to ensure the following APIs are exposed.

1. **Product** endpoints:
    a. GET ***/api/products*:** Retrieve all the products
    b. GET ***/api/products/{id}*:** Retrieve a specific product. Provide the product ID as a path variable.
    c. GET ***/api/*products/{categoryId}:** Retrieve all the specific products belonging to a particular category by categoryId.
    d. GET ***/api/*products/{categoryName}:** Retrieve all the specific products belonging to a particular category by category name.
    e. POST ***/api/products*:** Create a new product
    f. PUT ***/api/products/{id}*:** Update an existing product.
    g. DELETE ***/api/products/{id}*:** Delete a product. Provide the product ID as a path variable.

2. **Category** endpoints:
    a. POST ***/categories*:** Create a new category.
    b. GET ***/categories/{id}*:** Retrieve a category by ID.
    c. GET ***/categories*:** Retrieve all categories.
    d. PUT ***/categories/{id}*:** Update a category.
    e. DELETE ***/categories/{id}*:** Delete a category.

3. **Cart** endpoints:
    a. POST ***/cart/add*:** Create a new cart.

b. GET ***/cart/get:*** Retrieve a cart by ID.
c. PUT ***/cart/update/{cartItemId}*****:** Update a cart.
d. DELETE ***/cart/delete/{cartItemId}*****:** Delete a cart.

4. **Order** Endpoints:
    a. POST ***/api/orders*****:** Place an order.
    b. GET ***/api/orders/get:*** It returns the list of all orders.
    c. GET ***/api/orders/get/{id}:*** It returns an order for the given id.
    d. GET ***/api/orders/{id}/summary:*** Generate order summary. *(This is an optional API which the learner can create)*

5. **Home** Endpoints:
    a. GET **/home:** Get all the products sorted by their categories in a Hashmap.
    b. GET **/home/search:** (String **"Pattern"** as a QueryParam): To Search products by their name and return them sorted by their categories in a Hashmap.

6. **Data** Controller:
    a. POST **/data:** Populates data for a particular category.
7. **User** Controller: You need to create APIs as per project requirements.

Note: Students have to identify which API access has to be given with a pre-authorise role

Step 5: Services and Repository

1. Define Services for each entity (Product, Cart, Order) to handle business logic and data manipulation operations.
2. Specify methods for all API operations that are mentioned above.

3. Define repository interfaces for each entity (Product, Cart, Order) to handle data access operations.
4. Implement CRUD methods, custom queries, and other operations in the respective repositories.

Step 6: Exception handling

1. Implement global exception handling to handle and return meaningful error responses for various exceptions during user operations.
2. Customise error responses to provide informative error messages and appropriate HTTP status codes.

Step 7: Creating test cases

1. Use testing frameworks to write unit tests for the services, repositories, and API controllers.
2. Verify that the APIs are functioning as expected and returning the correct responses.
3. Test various scenarios, including edge cases and error handling.

Note: *Please refer to the* ***"Writing unit test cases"*** *and* ***"Writing integration test cases"*** *PDFs to understand how to write test cases*.