Technical Design Document: E-Commerce Order Management System

This design document provides a high-level overview of the proposed E-Commerce Order Management System, outlining its rationale, terminology, system architecture, data model, interface/API definitions. Detailed implementation specifics and interactions with external services will be covered in separate documents.

1. Rationale:
The E-Commerce Order Management System is designed to streamline and automate the end-to-end lifecycle of orders within the e-commerce platform.

2. Terminology:
Order: A customer's request to purchase one or more products (OrderItems). It keeps information about the Customer, current status, order history and other details.
Customer: Represents the customer, their name, email, and address.
OrderItem: Holds information about ordered product name and its quantity.
OrderStatus: Represents the state of the Order. It can be *Pending* (Order has been created but not yet processed), *Processing* (Payment is being processed), *Paid* (Payment has been successfully processed), *Shipped* (Order has been shipped), *Delivered* (Order has been successfully delivered), *Canceled* (Order has been canceled) or *Refunded* (Payment has been refunded).
OrderHistory: Keeps the history of the Order's status. An Order has a list of OrderHistory. Newly created Order has one OrderHistory with Pending status.

3. System Architecture:
The system consist of one service and relational DB.
Order Service: Manages order creation, modification, keeps track of order history. Manages customer resources.

4. Data Model:
Order: Contains order details such as order id, customer information, current status, ordered items details, the list of order's shipping address, and payment details.
Customer: Stores customer's information, like customer id, name, surname, email, phone number, and the list of customer's all orders.
OrderItem: Keeps order item id, product name, its quantity and the order to which it refers.
OrderStatus: Is an Enum that has Pending, Paid, Shipped, Delivered, Canceled, and Refunded.
OrderHistory: Has id, OrderStatus, order status' modification date and the order to which it refers.

5. Data Validation:
Order: When creating an order customer id must be greater than or equal to 1. Shipping address cannot be null. The list of order items cannot be empty.
OrderItem: The items' product name must not be null and its quantity must be a positive number.
Customer: When creating a customer, its name must not be null. It must have a non-null email and match to this regex: *^(.+)@(.+)$* . Phone number must also be non-null and match *^\+{0,1}[0-9]{8,20}$* regex.

6. Error Handling:
The application employs a centralized error handling mechanism which effectively handles various types of exceptions that might occur during the execution of RESTful services. The following exception types are specifically addressed:
EntityNotFoundException: When an entity is not found, the system responds with an HTTP 404 (Not Found) status with an instance of ApiError containing relevant details, such as the error code and message.

MethodArgumentNotValidException**:**This exception is triggered when method argument validation fails, often due to invalid inputs. system responds with an HTTP 422 (Unprocessable Entity) status. The response is a Map containing field-level error messages
ConstraintViolationException: Handles constraint violations, typically arising from validation annotations on entity fields. The system responds with an HTTP 400 (Bad Request) status. The response is a Map containing a list of error messages.
DataIntegrityViolationException: Specifically caters to exceptions related to data integrity violations. The system responds with an HTTP 400 (Bad Request) status. The response is a List containing a single error message extracted from the exception's cause chain.
General Exception: For any unhandled exceptions, a generic exception handler is used. The system responds with an HTTP 500 (Internal Server Error) status. The response is an instance of ApiError providing details of the error, including the error code and message.
APIError is a uniformed error response, containing the error code, message and a list of error details.
This centralized approach to error handling ensures consistency in responses, simplifies debugging, and provides a clear structure. The logging mechanism further aids in monitoring and identifying potential issues in a timely manner. All error scenarios are logged, capturing relevant details to assist in troubleshooting and diagnosing issues.

7. Interface/API Definitions:
Create Order API: Allows the creation of new order with at least one OrderItem.
Get Order API: Gets the order with its id.
Get All Orders API: Gets all orders with pagination, sorted by order id with either ASC or DESC order.
Update Order API: Enables order modifications, updating of the status, changing the customer, shipping address and payment details, payment, shipped and delivered dates and the whole list of the order items.
Delete Order API: Deletes the order by id, which also cascades the deletion of related order items and order histories.
Get Order History API: Gets the whole history of the order.
Create Customer API: Creates a customer.
Get Customer API: Gets the customer with its id.
Get All Customer API: Gets all customers with pagination, sorted by order id with either ASC or DESC order.
Update Customer API: Enables customer modifications, updating of the name, surname, email and phone number.
Delete Customer API: Deletes the customer by id.