

Online Store API -Technical Design

1. Overview

The project involves building and testing APIs for three modules: **Products**, **Cart**, and **User**. The APIs support CRUD operations, filtering, sorting, and specific use cases like login, user-specific data, and date-based queries. The design covers both functional requirements and user story-based scenarios.

2. Modules and Functionalities

2.1 Products Module

- **API Endpoints:**
 - GET /products: Retrieve all products.
 - GET /products/{id}: Retrieve a product by ID.
 - GET /products?limit=x: Retrieve a limited number of products.
 - GET /products?sort=asc|desc: Retrieve products in a specific order.
 - GET /products/categories: Retrieve all product categories.
 - GET /products/category/{category}: Retrieve products by category.
 - POST /products: Add a new product.
 - PUT /products/{id}: Update an existing product.
 - PATCH /products/{id}: Partially update a product.
 - DELETE /products/{id}: Delete a product.
- **Data Model:**

```
{
  "id": 1,
  "title": "Product Name",
  "price": 100.0,
  "description": "Product Description",
  "category": "Electronics",
  "image": "url-to-image",
  "rating": {
    "rate": 4.5,
    "count": 120
  }
}
```

- **Functional Requirements:**
 - Support CRUD operations.
 - Enable filtering by category, sorting, and limiting results.
-

- Handle edge cases like invalid IDs, empty categories, or malformed requests.
-

2.2 Cart Module

- **API Endpoints:**

- GET /carts: Retrieve all cart items.
- GET /carts/{id}: Retrieve a cart by ID.
- GET /carts?limit=x: Retrieve a limited number of carts.
- GET /carts?sort=asc|desc: Retrieve carts in a specific order.
- GET /carts?startDate=YYYY-MM-DD&endDate=YYYY-MM-DD: Retrieve carts in a date range.
- GET /carts/user/{userId}: Retrieve carts by user ID.
- POST /carts: Add a new cart.
- PUT /carts/{id}: Update a cart.
- PATCH /carts/{id}: Partially update a cart.
- DELETE /carts/{id}: Delete a cart.

- **Data Model:**

```
{
  "id": 1,
  "userId": 2,
  "date": "2023-01-01",
  "products": [
    {
      "productId": 1,
      "quantity": 2
    },
    {
      "productId": 2,
      "quantity": 1
    }
  ]
}
```

- **Functional Requirements:**

- Support CRUD operations.
 - Filter carts by date range or user ID.
 - Enable sorting and limiting of results.
 - Handle invalid product or user IDs gracefully.
-

2.3 User Module

- **API Endpoints:**
 - POST /auth/login: Log in to generate a token.
 - GET /users/{id}: Retrieve user details by ID.
 - GET /users: Retrieve all users.
 - POST /users: Add a new user.
 - PUT /users/{id}: Update a user.
 - PATCH /users/{id}: Partially update a user.
- **Data Model:**

```
{
  "id": 1,
  "email": "user@example.com",
  "username": "username",
  "password": "hashed_password",
  "name": {
    "firstname": "First",
    "lastname": "Last"
  },
  "address": {
    "city": "City",
    "street": "Street",
    "number": 1,
    "zipcode": "12345",
    "geolocation": {
      "lat": "0.0000",
      "long": "0.0000"
    }
  },
  "phone": "123-456-7890"
}
```

- **Functional Requirements:**
 - Support CRUD operations.
 - Authenticate users and return tokens.
 - Handle edge cases like invalid user IDs or malformed requests.
-

3. Testing Design

3.1 Test Cases

- **Products Module:**
 - Verify retrieval of all products.
 - Verify retrieval of a product by ID.
 - Verify retrieval of products with a limit.
 - Verify retrieval of products with sorting.
 - Verify CRUD operations for products.
 - Validate error handling for invalid product IDs.
- **Cart Module:**
 - Verify retrieval of all cart items.
 - Verify retrieval of a cart by ID.
 - Verify filtering carts by date range or user ID.
 - Verify CRUD operations for carts.
 - Validate error handling for invalid cart or user IDs.
- **User Module:**
 - Verify user login functionality.
 - Verify retrieval of a user by ID.
 - Verify retrieval of all users.
 - Verify CRUD operations for users.
 - Validate error handling for invalid user credentials or IDs.

3.2 Automation Framework

- **Technology Stack:**
 - Programming Language: Java
 - Testing Framework: TestNG or JUnit
 - HTTP Client: REST Assured
 - Reporting: Allure Reports, Extent Reports
 - Data Source: JSON files for test data
 - **Framework Design:**
 - Modular structure with separate classes for each module (Products, Cart, User).
 - Data-driven testing using JSON or Excel files.
-

- Reusable utility methods for API requests (GET, POST, PUT, PATCH, DELETE).
 - Logging and reporting integration.
-

4. Non-Functional Requirements

- **Performance:**
APIs should respond within 200ms for 90% of requests.
 - **Scalability:**
APIs should handle 100 concurrent users without degradation.
 - **Security:**
Implement token-based authentication for protected endpoints.
 - **Error Handling:**
Return meaningful error messages for invalid requests (e.g., 400 Bad Request, 404 Not Found).
-

5. Tools and Technologies

- **API Development:** Node.js, Express.js (or similar backend framework)
 - **Database:** MySQL or MongoDB (depending on use case)
 - **Testing Tools:** REST Assured, Postman, JMeter (for performance testing)
 - **Version Control:** Git
 - **CI/CD:** Jenkins or GitHub Actions for automated testing and deployment
-

6. Deliverables

1. Functional APIs for all endpoints.
 2. Automated test cases for all user stories.
 3. Documentation covering API specs, test cases, and architecture.
-
-