Technical Skills Assessment - BackEnd

Instructions

Complete the following projects by following each section's guidelines. This assessment evaluates your abilities across core areas, including Software Architecture, Design Patterns, and Mobile Skills.

Please review and adhere to the following instructions:

1. **Submission Timeline**: You have **1 week** to complete the assessment. Any commits made **after the deadline** will not be reviewed.
2. **Submission Method**: Submit your completed work via the form provided. Ensure all required fields are filled out accurately before submission.
3. **Project Clarity**: Each section outlines a unique project with specific requirements. Aim for clear and concise solutions that demonstrate best practices.
4. **Commit Standards**: Use meaningful commit messages. Ensure each commit is logically separated to show your progression and approach.
5. **Questions and Support**: Please contact your Teach Lead via Slack for clarification.

Project Sections

Each project section below includes an assignment. Review all requirements carefully before you start.

Section 1: Software Architecture

Instructions:	You are tasked with designing the architecture of a simple Inventory Management System following DDD principles. The system will be responsible for tracking products, inventory levels, and orders.
Deliverable:	**Draw.io Link with an Entity-Relationship Diagram**
Scenario:	The company sells physical products, and they need a system that supports the following functionalities: **Product Management** – Add, update, and remove products from the system. **Inventory Tracking** – Manage inventory levels, ensuring the system tracks when products are added or removed from stock. **Order Management** – Track orders placed by customers, updating inventory when orders are completed.

Section 2: Design Patterns

Instructions:	Review the provided code in the following Github Gist: https://gist.github.com/abulnes-designli/78307c879a4b23de58282a616343e3ca Refactor the code by applying a Design Pattern you think is suitable for managing the payment service with different third-party libraries. Ensure your code is more modular and adheres to good practices. You can rename, modify, and refactor the code and logic as needed to complete the task. Ensure your code works (you can add more logs or cases if you consider it)
Deliverable:	**Link with Github Gist Project**
Scenario:	You are integrating a third-party payment service into your ecommerce platform. However, the payment service's API differs from your platform's existing interface. Can you spot how you can modify the following code?

Section 3: BackEnd Technical Skill

Objective

The goal is to build a robust and scalable backend API that manages a simplified e-commerce system. The system should handle product management, user authentication, order processing, and payment simulations using asynchronous processes. The API should follow best practices for RESTful design, handle performance optimizations, ensure data consistency, and provide scalability for high-concurrency scenarios.

Requirements:

API Design & Functionality:

- Implement the following RESTful endpoints:
  - **User Management**:
    - `POST /register`: Register a new user.
    - `POST /login`: Log in a user, returning a JWT for authentication.
  - **Product Management**:
    - `GET /products`: List all products.
    - `GET /products/{id}`: Retrieve details of a specific product.
    - `POST /products`: Add a new product (authentication required).
    - `PUT /products/{id}`: Update product details (authentication required).
    - `DELETE /products/{id}`: Delete a product (authentication required).
  - **Order Management**:
    - `POST /orders`: Place a new order (authentication required).
    - `GET /orders/{id}`: Get details of a specific order (authentication required).
    - `GET /orders`: List all orders for the authenticated user (authentication required).

Authentication & Authorization:

- Use **JWT** (JSON Web Tokens) for user authentication.
- Protect all endpoints for product and order management so only authorized users can perform actions.

Product Catalog:

- Each product should have the following attributes:
  - `id`: Unique identifier.
  - `name`: Name of the product.

    - `description`: Brief description.
    - `price`: Price of the product.
    - `stock`: Number of units available.
    - `createdAt`: Date when the product was added.
    - `updatedAt`: Last update date of the product.

Order Processing & Payment Simulation:

- When an order is placed:
    1. Deduct the ordered quantity from the product stock.
    2. Simulate an asynchronous payment process using a job queue.
    3. Ensure eventual consistency; confirm the order only if the payment succeeds.
    4. Restore stock if the payment fails.

Asynchronous Processing:

- Use a job queue (e.g., Redis, RabbitMQ) for handling asynchronous order processing.
- Implement payment retries with an exponential backoff strategy in case of failures.

Error Handling & Edge Cases:

- Implement proper error handling for scenarios like insufficient stock, unauthorized access, and invalid input.
- Ensure idempotency in order processing so repeated requests don't deduct stock multiple times.

Optimizations:

- Implement caching (e.g., Redis) for frequent GET requests.
- Optimize database queries, especially for product listings and orders, using pagination and indexing.

Bonus Requirements (Optional):

- **WebSocket Integration**: Notify users in real-time when their payment is processed and their order is confirmed.
- **Rate Limiting & Throttling**: Protect against abuse by limiting login attempts and API requests.

User Stories

- **User Registration & Login**:
    - As a new user, I want to register with my email and password so that I can create an account.
    - As a registered user, I want to log in and receive a token, so I can access protected endpoints.
- **Product Management**:
    - As a user, I want to view a list of available products.
    - As an admin, I want to add, update, and delete products to manage the catalog.
- **Order Placement**:
    - As an authenticated user, I want to place an order for products I want to purchase.
    - As an authenticated user, I want to view the details of my past orders.
- **Payment Processing**:
    - As a user, I want my payment to be processed securely and asynchronously.
    - If my payment fails, I want the system to notify me and update my order status.
- **Real-time Notifications (Bonus)**:
    - As a user, I want to receive real-time notifications when my order payment is processed and confirmed.

Technical Details

Stack:

- Use **NestJS** for API development.
- Use **MySql** or **PostgreSQL** as the database.
- Use **Redis** or **RabbitMQ** for job queuing and caching.

Authentication:

- Implement **JWT-based authentication** for secure access to protected endpoints or use Auth0

Asynchronous Processing:

- Use Redis or RabbitMQ as a message queue for simulating payment processing.
- Implement exponential backoff in case of payment failures.

Data Consistency:

- Ensure eventual consistency in the order-processing workflow by using transactions (if the database supports it) or handling compensatory actions.

Optimizations:

- Implement caching with Redis for frequently accessed endpoints, such as product listings.
- Use indexes on database fields for faster queries, especially for products and order-related data.
- Implement pagination for endpoints that return lists of items to handle large datasets efficiently.

Testing:

- Write unit tests for key functionalities (e.g., user authentication, product management).
- Write integration tests for order processing and payment simulation.

Submission

- Create a public GitHub repository for this project.
- Include a README.md file with instructions on how to set up and run the project.
- Share the repository URL once you're finished in the Google Form provided