Course Project

Project Title: Building a Microservices Prototype for Online Retail Services

Objective

To design and prototype an online retail service system using Microservices Architecture, integrating software design patterns to enhance the system's scalability, independence, and resilience. The project is focused on architectural structuring and application of patterns rather than full-scale functionality.

Project Description

Students will develop a prototype for an online retail service, akin to platforms like Amazon or eBay, utilizing Microservices Architecture. This project will concentrate on designing and demonstrating individual microservices, each representing a core aspect of the system, and applying relevant design patterns to optimize microservice functionality and interaction.

Key Learning Outcomes

- Mastery of Microservices Architecture principles.
- Implementation of software design patterns in a distributed, microservices context.
- Understanding how architecture and patterns affect quality attributes like scalability, fault tolerance, and maintainability.

Prototype Requirements

Core Microservices

- *Product Catalog Service*: Managing product listings.
- *Customer Service*: Handling customer data and profiles.
- *Order Service*: Processing and tracking orders.
- *Payment Service*: Managing payment transactions.

Integrated Design Patterns

Please use at least four of the following patterns:

1. Product Catalog Service:
    - **Factory Method Pattern**: Use this for creating instances of complex product objects, allowing for flexibility in extending product types in the future.
    - **Singleton Pattern**: Manage a single, shared connection to the product catalog database.
2. Customer Service:
    - **Strategy Pattern**: Implement different algorithms for customer segmentation or personalized marketing strategies that can be switched at runtime.
    - **Decorator Pattern**: Use this for adding additional responsibilities to customer objects dynamically (like premium or loyalty program attributes).
3. Order Service:
    - **Command Pattern**: Handle order processing operations (like place order, cancel order) that can be encapsulated as objects, allowing for queuing, logging, and undo operations.
    - **Observer Pattern**: For notifying other services (like Product Catalog or Customer Service) when an order is placed or updated.
4. Payment Service:
    - **State Pattern**: Manage state changes in payment processing (like initiated, pending, completed, failed) in an organized manner.
    - **Chain of Responsibility Pattern**: Process various payment methods (credit card, PayPal, etc.) where each method represents a segment in the chain.
5. General Integration Across Services:
    - **Facade Pattern**: Simplify interactions between microservices by providing a unified interface to a set of interfaces in the microservices system, especially useful in the context of the API Gateway Pattern.
    - **Adapter Pattern**: Useful for making the interfaces of different microservices or external systems compatible with each other.
6. Data Access and Repository:
    - **Proxy Pattern**: Implement lazy loading or access control for database interactions.
7. Event-Driven and Asynchronous Communication:
    - **Mediator Pattern**: Manage complex communications between different microservices by introducing a mediator object that encapsulates how objects interact.
8. Error Handling and Resilience:

    Memento Pattern: Use for saving the state of an object (like an order or a transaction) to provide rollback mechanisms in case of failures.

Quality Attributes Focus

Analyze how the microservices architecture and the chosen design patterns contribute to the system's overall performance, resilience, and modularity.

Technical Stack

- *Languages and Frameworks*: Suitable choices for each microservice, potentially differing across services.
- *Databases*: Independent database for each service (SQL or NoSQL as appropriate).
- *Messaging and Communication*: Utilize RESTful services, message brokers, or event streams for inter-service communication.

Milestones

- *Designing Microservices*: Outlining the structure and responsibilities of each microservice and selecting appropriate design patterns.
- *Developing and Testing Microservices*: Implementing and testing each microservice individually, focusing on pattern integration.
- *Integration and System Testing*: Ensuring the microservices work cohesively.
- *Documentation and Final Presentation*: Documenting the architecture, design decisions, and preparing a comprehensive presentation.

Submission

- *Prototype Code*: Hosted on GitHub, with separate repositories or modules for each microservice.
- *Documentation*: Detailed coverage of microservices design, applied patterns, and quality attributes analysis.
- *Presentation*: A thorough presentation of the prototype, focusing on microservices and design patterns.