Ecommerce Application Design

Microservice Modules

- Product Catalog Service – product-catalog-service
    1. Manages the product information.
    2. Provides a REST API to retrieve product details.
- Order Service – ecom-order-service
    1. Handles the order creation ,inventory check initiation, and order fulfillment, retrial mechanism if failed.
    2. Uses Kafka to communicate with other services (e.g., notify inventory service when an order is placed).
    3. ntegrates with Redis for caching order information.
- Customer Service – ecom-customer
    1. Manages customer information.
    2. Provides endpoints for user registration, profile management,
- Inventory Service – ecom-inventory-service
    1. Manages inventory levels.
    2. Updates inventory quantities when an order is placed.
    3. Communicates with the Payment Processing Service via Kafka.
- Payment Service - ecom-payment-service
    1. Handles payment processing (simulated for the purpose of the demo).
    2. Integrates with the Order Service using Kafka to confirm or rollback order.
- API Compositor Service - ecom-compositor-service
- Analytic Service – ecom-analytics-service
    1. handles all anylytic requirements using kafka messaging
- Gateway Server - gateway-server
- Config Server - config-server
- Eureka Server - eureka-server

GitHub URL : https://github.com/topics/microservices-project

Distributed patterns

- **API-Compositor Pattern** – Aggregates data from multiple services.
  Implemented **Order details fetch** in ecom-compositor-service by querying Order , Product Catalog and Payment microservice.
    - Using WebFlux (Non-blocking) to make async requests to other microservices (instead of Feign).
    - Redis for storing cookies and session data.
    - getAllOrderDetails_APICompositorWithErrorHandling : http://localhost:8072/ecom-compositor-service/api/v1/compositor/allOrders
    - fetchAllOrderDetails_ApiCompositor_WithoutErrorHandling : http://localhost:8072/ecom-compositor-service/api/v1/compositor/allOrderDetails

- **Saga Pattern –** Manages long-running transactions with compensating actions. Implemented **Order placing** by
    - **Step 1**: Customer places an order by Creating Order in order micro service
    - **Step 2**: Checking Inventory for stock and confirm inventory if available in inventory micro service
    - **Step 3**: Payment is processed in Payment microservice
    - **Step 4**: If any service fails, a compensating transaction (like rolling back inventory or refunding the payment) is triggered.
    - **Kafka** will be used to send events between services to signal the success/failure of each step.
    - Used **Redis** for storing cookies and session data.
    - Implemented **Retry mechanism** and collected **Interim and Final Responses** via Postman
    - http://localhost:8072/ecom-order-service/api/v1/orders/placeorder
    - Json Body:

        -
          ```
          {
          "orderId": null,
          "customerId": "C001",
          "productId": "P001",
          "quantity": 20,
          "orderStatus": null
          }
          ```

Tools and Technologies

- **Spring Boot:** For building the microservices.
- **Zipkin and Jaeger** – For distributed tracing requests across different services
- **Loki & Grafana –** For log aggregation and monitoring. Loki is used for collecting logs from the services and Grafana is used to visualize them.
- **Kafka** - used to send events between services to signal the success/failure
- **Separate Schemas** for Each Microservice
- **API gateway** for service discovery to route traffic to the appropriate services
- **Eureka Server** for client side discovery
- **Config Server** to manage externalized configuration for distributed systems

Database and Caching Design

- **Databases**: Each microservice will have its own database or use a shared database for simplicity. For the demo, a simple in-memory database (H2) can be used.
- **Redis**: Used to cache product and inventory details, reducing load on databases and improving response times.

Postman collection and DB scripts

- These artefacts are available in the following url : https://github.com/dinnupaul/ecom-microservices-artefacts

Conclusion:

This design ensures that this e-commerce platform is both scalable and resilient, with minimal latency and high availability. This implementation using Java microservices with Spring Boot is structured across separate services, with Redis used for session management and caching, Kafka for messaging between services, and Saga Pattern for managing long-running processes.