Microservices Assignment (E-Commerce)

Overview:

The system is built around five core microservices, each dedicated to a distinct aspect of the E-commerce business flow, and considerations for cross-cutting concerns have been addressed.

Services:

1. **Products Service:**
   - **Functionality**:
     - Manages the core product information such as name, description, and category.
     - Handles operations like adding or removing products from the inventory.
   - **Reasoning:**
     - Separation of concerns: Focusing on products allows for independent scaling and updates.
     - Reusability: Other services can consume product-related information without handling the underlying complexities.
   - **Assumptions:**
     - Each product has a unique identifier.
     - Product details are static for simplicity; dynamic details like price and availability are handled by other services.

2. **Inventory Service:**
   - **Functionality**:
     - Maintains the inventory details, including size, price, and design variations.
     - Enables admin to add/remove product details
   - **Reasoning:**
     - Decoupling product management from inventory management for flexibility and scalability.
     - Supports the ability to manage different variations of products efficiently.
   - **Assumptions:**
     - Inventory data includes size, price, and design.
     - Inventory details may vary for each product.

3. Inventory Service:

- **Functionality**:
    - Handles operations related to the user's shopping cart.
    - Allows adding products to the cart and initiating the checkout process.
- **Reasoning**:
    - Focuses on user-specific operations, ensuring personalized experiences.
    - Independence: Cart operations are isolated for ease of management and scaling.
- **Assumptions**:
    - Cart information is associated with a user ID.
    - The cart does not persist beyond a single session for simplicity.

4. Order Service:

- **Functionality**:
    - Manages the order lifecycle, including order creation, payment processing, and fulfillment.
    - Coordinates with the Cart Service to convert a cart into an order.
- **Reasoning:**
    - Decouples order management from cart operations for modularity.
    - Facilitates better scalability and handling of complex order-related tasks.
- **Assumptions**:
    - Orders are associated with a user ID.
    - Payment processing is handled by external services for simplicity.

5. Authentication Service:

Used Identity
\Server 4 for the authentication task.

- **Functionality**:
    - Manages user authentication and authorization.
    - Generates tokens for secure API access.
- **Reasoning**:
    - Security: Isolates authentication logic for centralized security measures.
    - Reusability: Allows other services to implement secure access control without redundant code.
- **Assumptions**:
    - Token-based authentication is used.
    - Basic user information (username, password) is stored securely.

Communication

In the microservices assignment, asynchronous communication using message queues, specifically **RabbitMQ**, is employed to enhance system scalability, flexibility, and resilience. Here's how communication is facilitated:

1. Decoupling Services:

Asynchronous communication allows for the decoupling of microservices. Instead of direct synchronous communication, where one service directly calls another, services interact through message queues. This decoupling promotes independence between services, enabling them to evolve and scale independently without direct dependencies.

2. Message Queues (RabbitMQ):

RabbitMQ serves as the message broker, facilitating communication between microservices. When a service needs to communicate with another, it sends a message to the appropriate queue in RabbitMQ. The receiving service listens to its designated queue, processes the message asynchronously, and responds accordingly.

3. Event-Driven Architecture:

The use of asynchronous communication introduces an event-driven architecture. Services communicate by emitting events (messages) to the message queue. Other services subscribe to specific events of interest, allowing them to react to changes or updates in the system. This event-driven approach enhances the responsiveness and modularity of the system.

4. Advantages of Asynchronous Communication:

Scalability: Asynchronous communication supports better scalability by allowing services to handle messages at their own pace. It prevents cascading failures that can occur in synchronous communication.

Fault Tolerance: If a service is temporarily unavailable or experiencing high load, messages are stored in the queue until the service becomes available, ensuring fault tolerance and message persistence.

Loose Coupling: Services are loosely coupled, reducing the impact of changes in one service on others. This enhances flexibility and maintainability.

5. Message Formats:

Messages exchanged between services typically follow a predefined format, commonly using JSON or another lightweight data interchange format. This standardization ensures consistency in message structure, making it easier for services to interpret and process messages correctly.

How to Run on Local Environment

Requirements:

1. docker
2. docker compose

Steps:

1. Open the root directory in linux shell/wsl (or powershell if docker desktop is installed)
2. Change the directory to **Ecom.Infra/** folder.
   Command: **cd Ecom.Infra/**
3. Run the following command to build all the images
   Command: **docker compose build**
4. Run the following command to run the microservices
   Command: **docker compose up**

Components of Assignment:

1. **Microservices**
2. **Databases** →
   a. MongoDb to store simple data like products etc.
   b. Mysql to store User identification info along with Identity server 4
3. **RabbitMQ** → Used for the asynchronous communication between different services, making our system eventual consistent and hence by increasing consistency.
4. **Eureka** → Used for the Service discovery, enhancing the workflow for horizontal scaling easier and fast and remove the need of manual configuration for new instances of services.
5. **Ocelot** → Used as an API Gateway which acts as a reverse proxy and secure our servers and add an extra layer of abstraction between client and servers.

DOCKER HUB PROFILE WITH UPLOADED IMAGES:
https://hub.docker.com/u/beastmonarch