PROBLEM STATEMENT – 3

General Guidelines for submission:

1. **Create a GitHub Repository.**
2. **Naming convention: <SRN1>_<SRN2>_<SRN3>_<SRN4>_Project title (srn in ascending order).**
3. **Weekly progress according to the problem statement assigned must be pushed to the repository and this will be considered carefully while evaluating.**
4. **Each team's weekly progress update in the repository must be shown to the teachers in class.**

Problem statement: Microservices communication using RabbitMQ

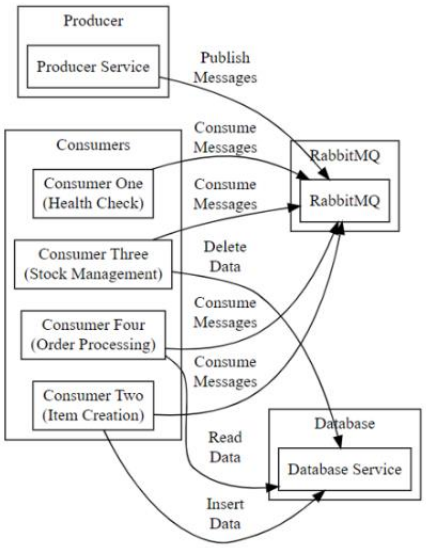
Description:

Build an ***inventory management system***. The inventory management system aims to efficiently manage inventory items, track stock levels, and handle orders through a microservices architecture. The system will utilize RabbitMQ for inter-service communication and Docker for containerisation, ensuring scalability, modularity, and ease of deployment.

Objectives:

1. Scalable Architecture: Design a microservices architecture that allows independent development, deployment, and scaling of each component.
2. Robust Communication: Implement communication between microservices using RabbitMQ queues to ensure reliable message passing.
3. Functional Microservices: Develop microservices to handle specific tasks such as health checks, item creation, stock management, and order processing.
4. Database Integration: Integrate a database service to store inventory data and ensure proper CRUD operations.
5. Containerization: Containerize the application using Docker to provide consistency and portability across different environments.
6. Testing and Documentation: Conduct thorough testing to ensure the system functions as expected and provide comprehensive documentation for setup, usage, and maintenance.

Sample Flow diagram:



Deliverables:

Week 1: Setup and Basic Implementation

- Setup Docker Environment:
  - Create Dockerfiles for each microservice.
  - Write a docker-compose.yml file to define the services and network configurations.
  - Test Docker setup locally to ensure containers can communicate with each other.
- Implement RabbitMQ Communication:
  - Set up RabbitMQ container and network.
  - Implement basic RabbitMQ communication between producer and consumer services.
  - Test message passing between microservices via RabbitMQ queues.
- Develop HTTP Servers:
  - Choose a suitable web framework (e.g., Flask for Python) for implementing HTTP servers.
  - Implement HTTP endpoints for health checks, and CRUD operations related to inventory management.
  - Test HTTP servers locally to ensure they are functional.

Week 2: Microservices Development

- Producer Service:
    - Develop the producer service responsible for constructing queues/exchanges and transferring data to consumers.
    - Implement the HTTP server to listen to health_check and CRUD requests.
    - Test the producer service to ensure it interacts correctly with RabbitMQ and handles HTTP requests.
- Consumer Services:
    - Implement consumer services (consumer_one to consumer_four) to handle specific tasks like health checks, item creation, stock management, and order processing.
    - Ensure each consumer service can communicate with RabbitMQ and perform its designated actions.
    - Test each consumer service individually to verify its functionality.

Week 3: Integration and Testing

- Integration Testing:
    - Integrate all microservices into the Docker environment.
    - Perform end-to-end testing to ensure seamless communication between microservices via RabbitMQ queues.
    - Test various scenarios to ensure the proper functioning of the inventory management system.

Sample file structure: