Department of Computer Engineering
Academic Term : January - May 2024

Class: B.E. (Computer) Semester VIII Division: B

Subject Name: Distributed Systems CSC 801

Name	**Riya Rajesh Patil**
Roll No.	**9282**
Experiment No.	**4**
Experiment Title	Implement and explore flexibility of Persistent communication using Message Queuing system using RabbitMQ

Aim: Implement and explore flexibility of Persistent communication using Message Queuing system using RabbitMQ

Pom.xml

```
<dependency>
    <groupId>com.rabbitmq</groupId>
    <artifactId>amqp-client</artifactId>
    <version>3.13.1</version>
</dependency>
```

Producer.java

```
import com.rabbitmq.client.*;

import java.io.IOException;
import java.util.concurrent.TimeoutException; // Add this import

public class Producer {
    private final static String QUEUE_NAME = "hello";

    public static void main(String[] argv) throws Exception {
```

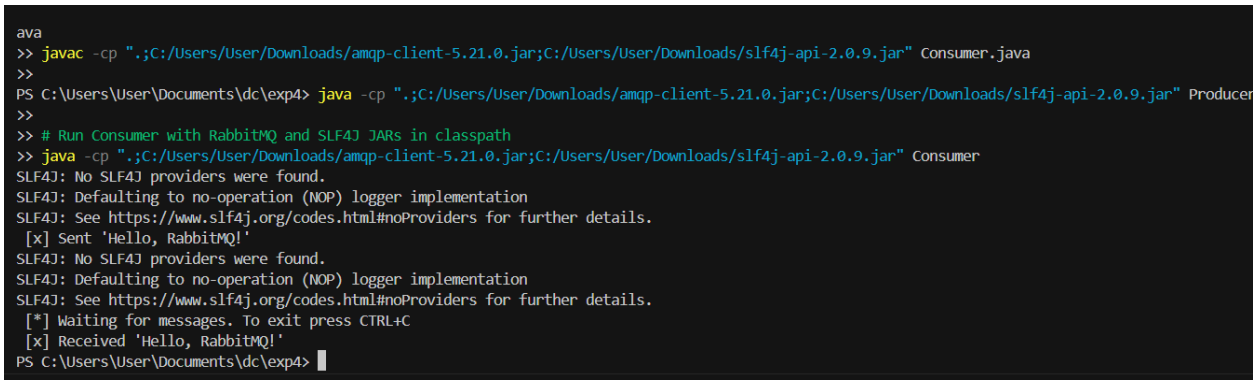
```
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");

        try (Connection connection = factory.newConnection(); Channel channel =
connection.createChannel()) {
            channel.queueDeclare(QUEUE_NAME, false, false, false, null);
            String message = "Hello, RabbitMQ!";
            channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
            System.out.println(" [x] Sent '" + message + "'");
        } catch (IOException | TimeoutException e) {
            e.printStackTrace();
        }
    }
}
```

Consumer.java

```
import com.rabbitmq.client.*;

import java.io.IOException;
import java.util.concurrent.TimeoutException; // Add this import

public class Consumer {
    private final static String QUEUE_NAME = "hello";

    public static void main(String[] argv) throws Exception {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");

        try (Connection connection = factory.newConnection(); Channel channel =
connection.createChannel()) {
            channel.queueDeclare(QUEUE_NAME, false, false, false, null);
            System.out.println(" [*] Waiting for messages. To exit press CTRL+C");

            DeliverCallback deliverCallback = (consumerTag, delivery) -> {
                String message = new String(delivery.getBody(), "UTF-8");
                System.out.println(" [x] Received '" + message + "'");
            };
            channel.basicConsume(QUEUE_NAME, true, deliverCallback, consumerTag -> {
            });
        } catch (IOException | TimeoutException e) {
```

```
            e.printStackTrace();
        }
    }
}
```

Output:



Conclusion:

Message Queuing systems offer a powerful solution for building flexible and scalable communication architectures. By decoupling producers and consumers through message queues, these systems enable asynchronous communication, which enhances system resilience and responsiveness. In this Java program utilizing RabbitMQ, we demonstrated a basic implementation of persistent communication using message queues. The program consists of a producer that sends a message to a queue and a consumer that retrieves and processes messages from the same queue. By leveraging RabbitMQ's capabilities, including message persistence and delivery acknowledgement, the program showcases how message queuing systems can facilitate reliable and robust communication between distributed components in a Java application.

POST LAB QUESTIONS

1. **What is persistent communication? Give an example.**

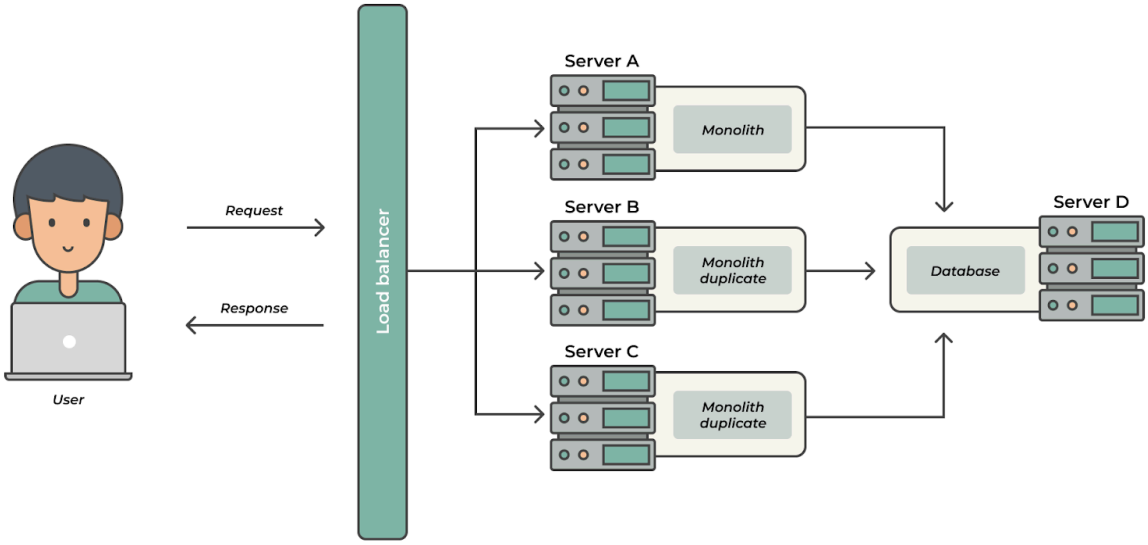
Ans: Persistent communication refers to communication where messages are stored even if the recipient is offline or unavailable. An example is email communication, where emails are queued and delivered once the recipient is online.

2. **Which data structure is required to implement persistent communication?**

Ans: A message queue is typically used to implement persistent communication. It's a data structure that stores messages until they are consumed by the recipient.

3. **Draw a sample message queuing architecture.**

Ans: The architecture of a message queue is simple — applications called the producers create messages and add them to the message queue. Other applications called the consumers pick up these messages and process them. Some examples of message queues are Apache Kafka, RabbitMQ, and LavinMQ, among others



2. Which data structure is required to implement persistent communication?

To implement persistent communication, a data structure commonly used is a queue. Queues are well-suited for managing persistent communication because they follow the First-In-First-Out (FIFO) principle, ensuring that messages are delivered in the order they were added. This is essential for maintaining the sequence of communication between sender and receiver, especially in scenarios where messages need to be stored temporarily if the recipient is unavailable or busy.

Queues can be implemented using various data structures such as arrays, linked lists, or other specialized data structures optimized for efficient insertion and removal of elements. Additionally, persistent queues can be implemented using durable storage mechanisms like databases or distributed file systems, ensuring that messages are not lost even in the event of system failures or restarts.

Overall, queues provide a reliable and efficient means of implementing persistent communication by ensuring message delivery order, managing message persistence, and enabling asynchronous communication between components in distributed systems.

3.Draw a sample message queuing architecture.