

Universidad de Las Américas  
Facultad de Ingenierías y Ciencias Agropecuarias  
*Ingeniería De Software*  
Progreso 1

**Nombres: Enrique Merizalde**

**Fecha: 11/06/2025**

## TALLER EN CLASE RABBIT MQ

### 1. Objetivo General

Aplicar el patrón de mensajería asincrónica para demostrar el desacoplamiento entre productores y consumidores.

Configurar un broker de mensajería RabbitMQ y conectar productores y consumidores de mensajes usando Apache Camel.

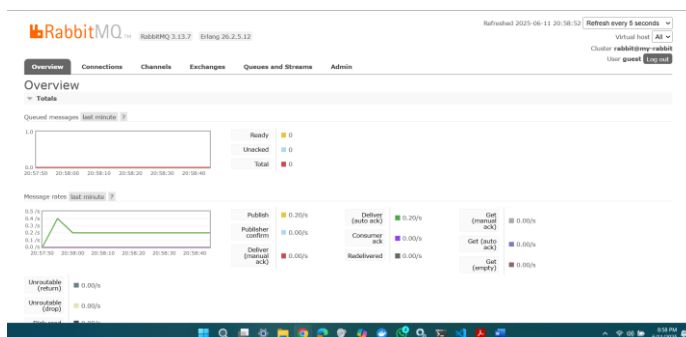
### 1. Instalación RabbitMQ en Docker

Search

Only show running containers

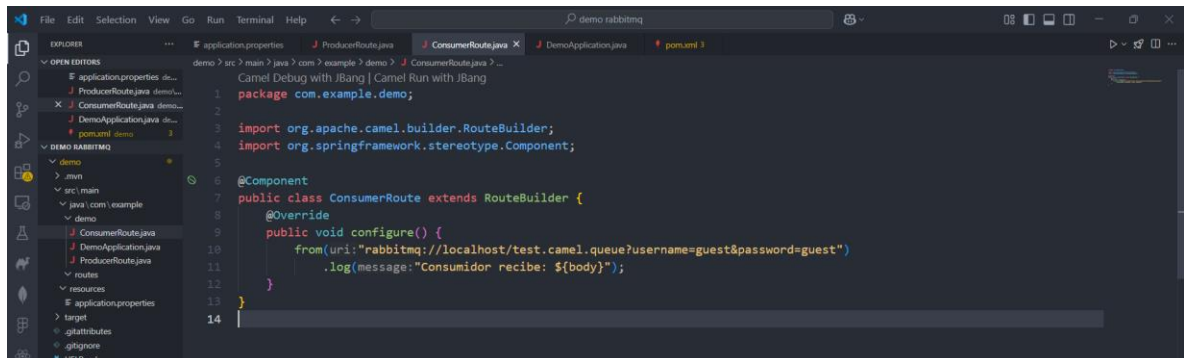
<input type="checkbox"/>	Name	Container ID	Image	Port(s)	CPU (%)	Last start... <div>↓</div>	Actions
<input type="checkbox"/>	<div><div></div><div>rabbitmq</div></div>	ccd72d2d3c1a	<a href="#">rabbitmq:3-management</a>	15672:15672 <a href="#">Show all ports (2)</a>	0.49%	2 hours ago	<div><div></div><div></div><div></div></div>
<input type="checkbox"/>	<div><div></div><div>testMinikube</div></div>	5d0293425ddf	<a href="#">k8s-minikube/kicad</a>	0.22	0%	2 days ago	<div><div></div><div></div><div></div></div>

### 2. Correr RabbitMQ en navegador



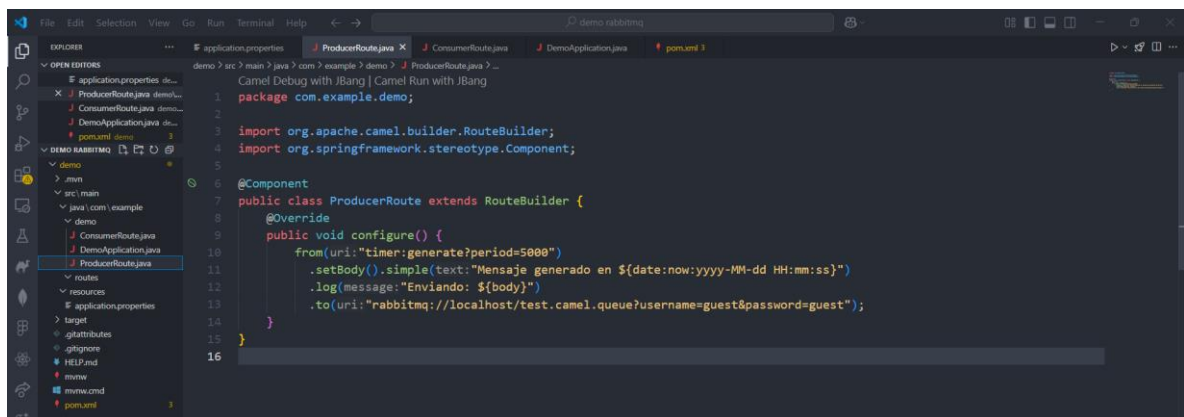
### 3. Proyecto con Apache Camel

#### ConsumerRoute.java



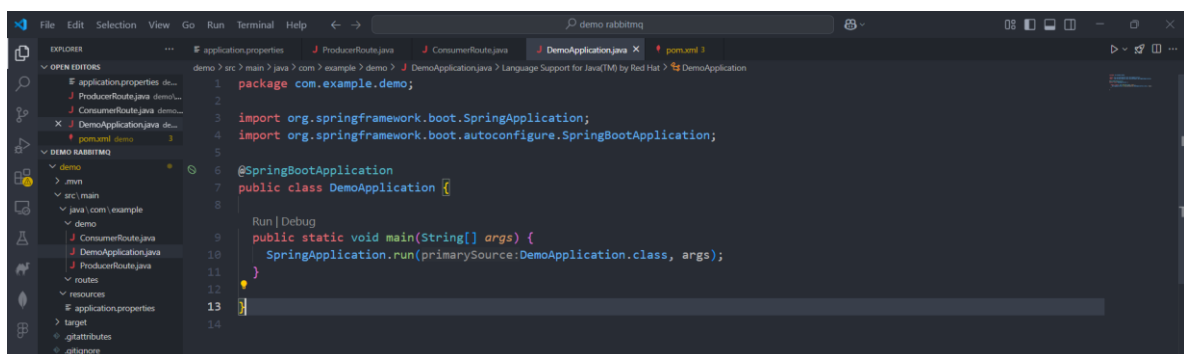
```
1 package com.example.demo;
2
3 import org.apache.camel.builder.RouteBuilder;
4 import org.springframework.stereotype.Component;
5
6 @Component
7 public class ConsumerRoute extends RouteBuilder {
8     @Override
9     public void configure() {
10         from(uri:"rabbitmq://localhost/test.camel.queue?username=guest&password=guest")
11             .log(message:"Consumidor recibe: ${body}");
12     }
13 }
14
```

#### ProducerRoute.java



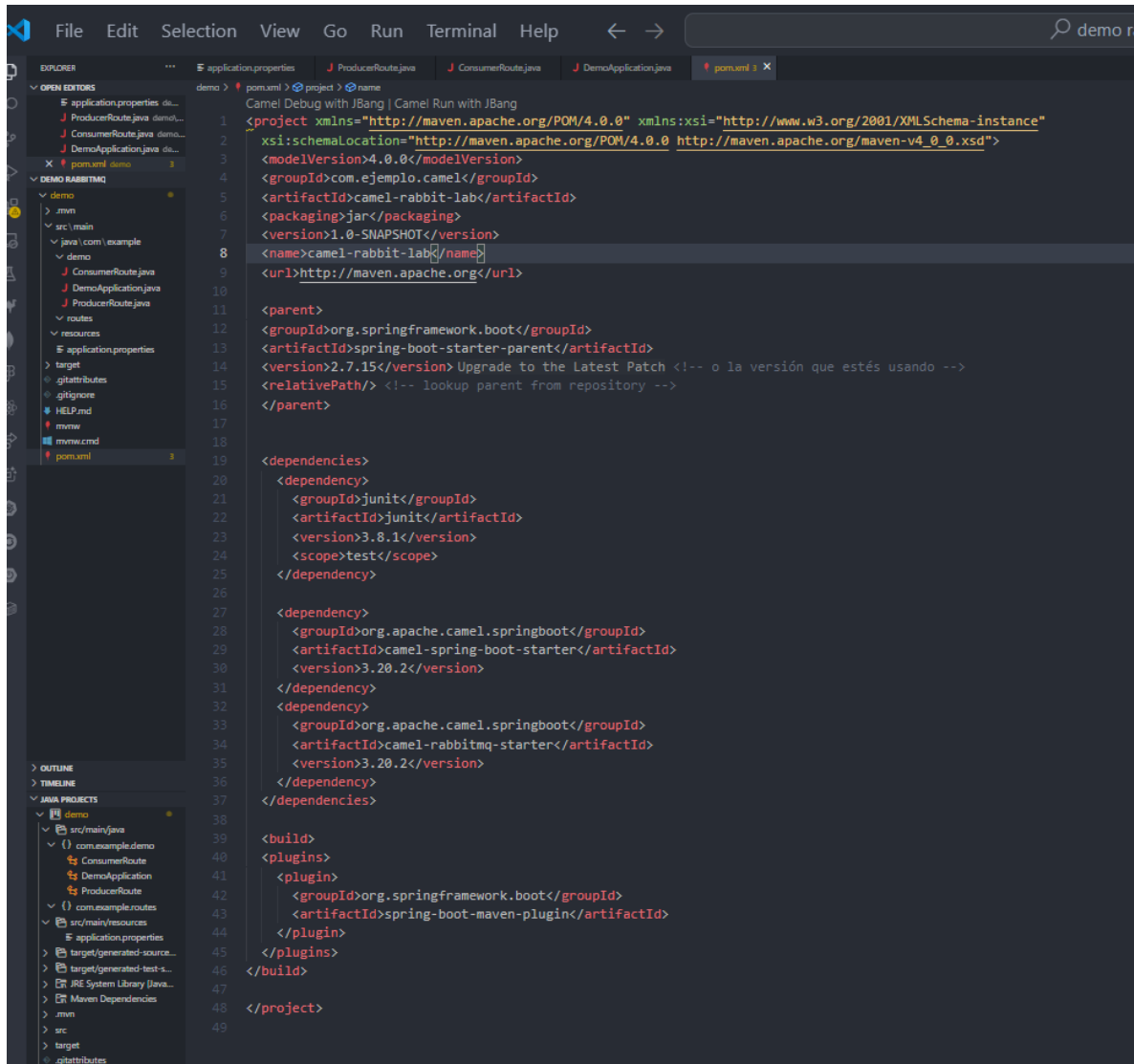
```
1 package com.example.demo;
2
3 import org.apache.camel.builder.RouteBuilder;
4 import org.springframework.stereotype.Component;
5
6 @Component
7 public class ProducerRoute extends RouteBuilder {
8     @Override
9     public void configure() {
10         from(uri:"timer:generate?period=5000")
11             .setBody().simple(text:"Mensaje generado en ${date:now:yyyy-MM-dd HH:mm:ss}")
12             .log(message:"Enviando: ${body}")
13             .to(uri:"rabbitmq://localhost/test.camel.queue?username=guest&password=guest");
14     }
15 }
16
```

#### DemoApplication.java (main)



```
1 package com.example.demo;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class DemoApplication {
8     Run | Debug
9     public static void main(String[] args) {
10         SpringApplication.run(primarySource:DemoApplication.class, args);
11     }
12 }
13
```

pom.xml



The screenshot shows an IDE with a dark theme. The Explorer on the left shows a project structure with a 'demo' folder containing 'pom.xml'. The main editor displays the content of 'pom.xml' with line numbers 1 through 49. The code is as follows:

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <groupId>com.ejemplo.camel</groupId>
6   <artifactId>camel-rabbit-lab</artifactId>
7   <packaging>jar</packaging>
8   <version>1.0-SNAPSHOT</version>
9   <name>camel-rabbit-lab</name>
10  <url>http://maven.apache.org</url>
11
12  <parent>
13    <groupId>org.springframework.boot</groupId>
14    <artifactId>spring-boot-starter-parent</artifactId>
15    <version>2.7.15</version> Upgrade to the Latest Patch <!-- o la versión que estás usando -->
16    <relativePath/> <!-- lookup parent from repository -->
17  </parent>
18
19  <dependencies>
20    <dependency>
21      <groupId>junit</groupId>
22      <artifactId>junit</artifactId>
23      <version>3.8.1</version>
24      <scope>test</scope>
25    </dependency>
26
27    <dependency>
28      <groupId>org.apache.camel.springboot</groupId>
29      <artifactId>camel-spring-boot-starter</artifactId>
30      <version>3.20.2</version>
31    </dependency>
32
33    <dependency>
34      <groupId>org.apache.camel.springboot</groupId>
35      <artifactId>camel-rabbitmq-starter</artifactId>
36      <version>3.20.2</version>
37    </dependency>
38  </dependencies>
39
40  <build>
41    <plugins>
42      <plugin>
43        <groupId>org.springframework.boot</groupId>
44        <artifactId>spring-boot-maven-plugin</artifactId>
45      </plugin>
46    </plugins>
47  </build>
48 </project>
49
```

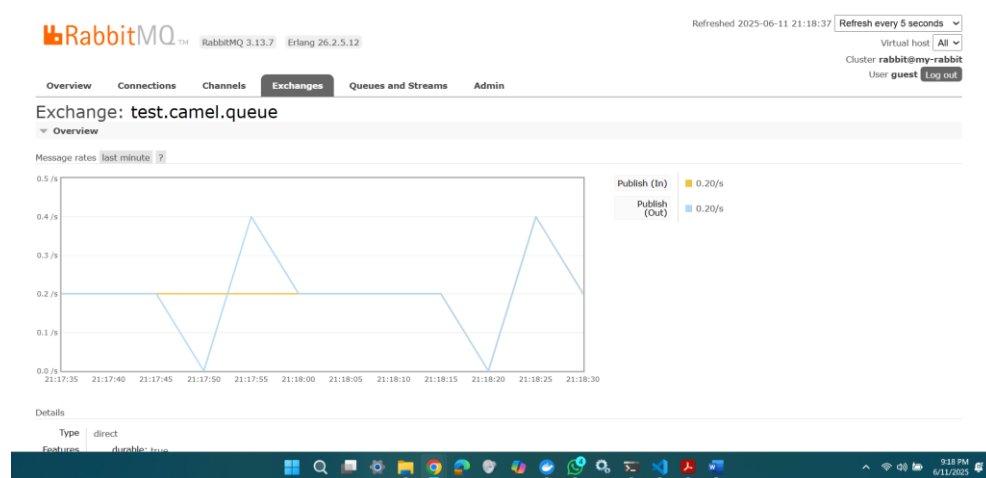
## Entregables del taller

### 1. Link a repositorio:

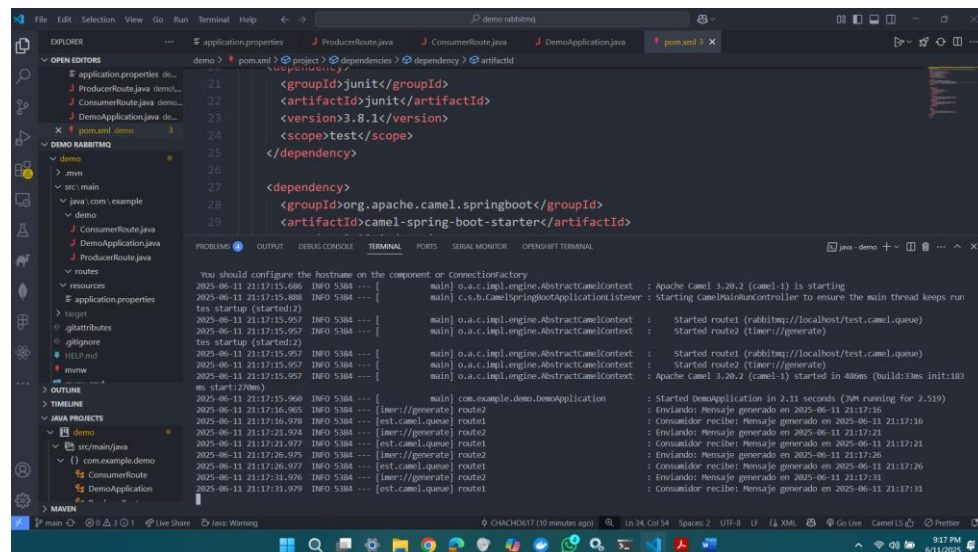
<https://github.com/CHACHO617/ApacheRabbitMQ/tree/main>

### 2. Capturas de funcionamiento

#### a. Consola Rabbit MQ



#### b. Consola de logs Camel



```
demo > pom.xml > project > dependencies > dependency > artifact
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>

<dependency>
<groupId>org.apache.camel.springboot</groupId>
<artifactId>camel-spring-boot-starter</artifactId>
</dependency>
```

PROBLEMS | OUTPUT | DEBUG CONSOLE | **TERMINAL** | PORTS | SERIAL MONITOR | OPENSHIFT TERMINAL

You should configure the hostname on the component or connection factory

```
2025-06-11 21:17:15.686 INFO 5384 --- [main] o.a.c.i.e.a.AbstractCamelContext : Apache Camel 3.20.2 (camel-1) is starting
2025-06-11 21:17:15.888 INFO 5384 --- [main] c.s.b.CamelSpringBootApplicationListener : Starting CamelMainRunController to ensure the main thread keeps run
2025-06-11 21:17:15.957 INFO 5384 --- [main] o.a.c.i.e.a.AbstractCamelContext : Started route1 (rabbitmq://localhost/test.camel.queue)
2025-06-11 21:17:15.957 INFO 5384 --- [main] o.a.c.i.e.a.AbstractCamelContext : Started route2 (timer://generate)
2025-06-11 21:17:15.957 INFO 5384 --- [main] o.a.c.i.e.a.AbstractCamelContext : Started route1 (rabbitmq://localhost/test.camel.queue)
2025-06-11 21:17:15.957 INFO 5384 --- [main] o.a.c.i.e.a.AbstractCamelContext : Started route2 (timer://generate)
2025-06-11 21:17:15.957 INFO 5384 --- [main] o.a.c.i.e.a.AbstractCamelContext : Apache Camel 3.20.2 (camel-1) started in 480ms (build:3ms init:183ms start:276ms)
2025-06-11 21:17:15.960 INFO 5384 --- [main] com.example.demo.DemoApplication : Started demoApplication in 2.31 seconds (Web running for 2.519)
2025-06-11 21:17:16.905 INFO 5384 --- [timer://generate] route2 : Enviando: Mensaje generado en 2025-06-11 21:17:16
2025-06-11 21:17:16.978 INFO 5384 --- [timer://generate] route2 : Considero recibir: Mensaje generado en 2025-06-11 21:17:16
2025-06-11 21:17:21.974 INFO 5384 --- [timer://generate] route2 : Enviando: Mensaje generado en 2025-06-11 21:17:21
2025-06-11 21:17:21.977 INFO 5384 --- [est-camel.queue] route1 : Consumidor recibe: Mensaje generado en 2025-06-11 21:17:21
2025-06-11 21:17:26.975 INFO 5384 --- [timer://generate] route2 : Enviando: Mensaje generado en 2025-06-11 21:17:26
2025-06-11 21:17:26.977 INFO 5384 --- [est-camel.queue] route1 : Considero recibir: Mensaje generado en 2025-06-11 21:17:26
2025-06-11 21:17:31.976 INFO 5384 --- [timer://generate] route2 : Enviando: Mensaje generado en 2025-06-11 21:17:31
2025-06-11 21:17:31.979 INFO 5384 --- [est-camel.queue] route1 : Consumidor recibe: Mensaje generado en 2025-06-11 21:17:31
```

### 3. Documento explicando

#### Qué patrón de integración se aplicó.

En esta práctica se aplicó el patrón de integración Mensajería Asíncrona. Este patrón permite que los sistemas intercambien información sin necesidad de estar conectados al mismo tiempo. En lugar de que el productor y el consumidor se comuniquen directamente, se utiliza un canal para el envío y gestión de mensajes, en este caso, una cola en RabbitMQ, que actúa como intermediario entre las partes de Productor y Consumer como intermediario. Este enfoque es ideal para escenarios donde se requiere tolerancia a fallos, escalabilidad y desacoplamiento entre componentes, permitiendo que cada uno procese los mensajes a su propio ritmo sin perder información y sin tener que estar a la espera de otro servicio que responda para poder seguir con su funcionamiento normal.

#### Cómo se logró el desacoplamiento productorconsumidor.

El desacoplamiento entre el productor y el consumidor se logró mediante la utilización de Apache Camel como middleware de integración y RabbitMQ como broker de mensajes. En concreto:

- El productor fue implementado como una ruta Camel que genera un mensaje cada 5 segundos usando un timer, lo transforma con una fecha actual y lo envía a una cola en RabbitMQ (test.camel.queue).
- El consumidor fue implementado como otra ruta Camel independiente, que se suscribe a esa misma cola y procesa cualquier mensaje que llegue.

Ambos componentes no están directamente conectados entre sí. Si el consumidor no está disponible temporalmente, RabbitMQ mantiene los mensajes en la cola hasta que pueda ser procesado. De igual manera, el productor no necesita conocer cuántos consumidores existen ni cuándo procesan los datos. Este desacoplamiento permite cambiar, escalar o reiniciar el productor o consumidor sin afectar al otro componente, facilitando así la mantenibilidad del sistema.

### **Ventajas que observaron durante la práctica.**

Durante la implementación y ejecución del sistema se pudieron observar varias ventajas clave que ofrece la mensajería asincrónica:

- ✓ Tolerancia a fallos: Si el consumidor se detiene, los mensajes no se pierden; quedan almacenados en la cola de RabbitMQ.
- ✓ Escalabilidad: Se pueden añadir múltiples consumidores para procesar los mensajes en paralelo sin modificar el productor.
- ✓ Mantenibilidad: Cada componente (productor o consumidor) se puede desarrollar, probar o desplegar de forma independiente.
- ✓ Visibilidad: RabbitMQ proporciona una interfaz de administración donde se pueden monitorear colas, flujos de mensajes y el estado del sistema.
- ✓ Simplicidad de implementación: Gracias a Apache Camel, se pudieron definir las rutas de integración con una sintaxis clara y concisa, facilitando la conexión entre temporizadores, transformaciones y colas de mensajería.