**UNIVERSIDAD PRIVADA**

**ANTENOR ORREGO**

INGENIERÍA DE COMPUTACIÓN Y SISTEMAS

**FACULTAD DE INGENIERÍA**



**ARQUITECTURA DE SOFTWARE**

TEMA

**Implementar el uso de Message brokers**

**DOCENTE**

MENDOZA CORPUS, CARLOS ALFREDO;

**INTEGRANTES**

* ALFARO ASUNCIÓN JUAN CARLOS
* JACOBO ROMERO WENDY MITCHEL
* MIÑANO SICCHA BRAYAN DANIEL

**TRUJILLO \_ 2024**

**Message Brokers**

**INTRODUCCIÓN**

En la arquitectura de microservicios, la comunicación eficiente entre los servicios es crucial para el rendimiento y la escalabilidad del sistema. Un "intermediario de mensajes" (message broker) desempeña un papel fundamental al facilitar esta comunicación. RabbitMQ es uno de los brokers de mensajes más populares y ampliamente utilizados, conocido por su confiabilidad y flexibilidad.

**¿QUÉ ES RABBITMQ?**

RabbitMQ es un intermediario de mensajes de código abierto que implementa el protocolo Advanced Message Queuing Protocol (AMQP). Actúa como un middleware que maneja el envío y recepción de mensajes entre los productores (servicios que envían mensajes) y los consumidores (servicios que reciben mensajes). Esto permite que los microservicios se comuniquen de manera asincrónica y desacoplada, mejorando la modularidad y la resiliencia del sistema.

**VENTAJAS DE USAR RABBITMQ EN MICROSERVICIOS**

* **Desacoplamiento:** Permite que los servicios se comuniquen sin conocer la existencia del otro, reduciendo las dependencias directas.
* **Escalabilidad:** Facilita la escalabilidad horizontal de los servicios al distribuir la carga de trabajo mediante colas.
* **Fiabilidad:** Garantiza la entrega de mensajes a través de mecanismos de confirmación y persistencia.
* **Flexibilidad:** Soporta múltiples patrones de mensajería como punto a punto, publicación/suscripción y enrutamiento basado en temas.
* **Resiliencia:** Mejora la tolerancia a fallos mediante la reintención y el almacenamiento persistente de mensajes en caso de errores.

**CASO DE USO: Gestión de Productos y Comentarios**

Para ilustrar el uso de RabbitMQ en una arquitectura de microservicios, consideremos un negocio que gestiona productos y comentarios. Este sistema consta de dos microservicios principales: ms\_product y ms\_review. La problemática es actualizar los detalles de los productos en tiempo real con el número de comentarios y el promedio de evaluación cada vez que se crea un nuevo comentario.

**En este escenario:**

* **ms\_review** es responsable de manejar los comentarios y producir mensajes cuando se crea un nuevo comentario.
* **ms\_product** es responsable de consumir estos mensajes y actualizar la información del producto en consecuencia.

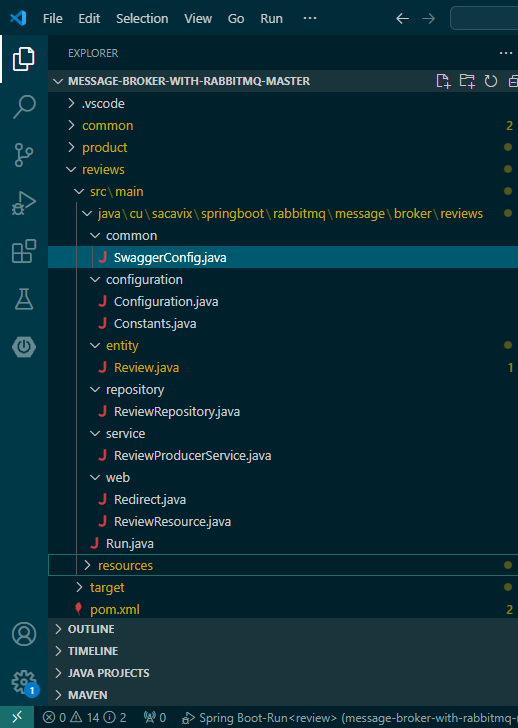
**Flujo de Trabajo con RabbitMQ**

* **Creación de Comentarios:** Cuando un usuario crea un comentario en ms\_review, se guarda en la base de datos y se envía un mensaje a RabbitMQ con los detalles del comentario (identificador del producto, número de comentarios y promedio de evaluación).
* **Procesamiento del Mensaje:** RabbitMQ encola el mensaje en la cola correspondiente (REVIEW\_CREATED\_QUEUE).
* **Actualización del Producto:** ms\_product consume el mensaje desde la cola, procesa la información y actualiza los detalles del producto en su base de datos.

Ambos servicios estén desacoplados, ya que ms\_review y ms\_product no se comunican directamente entre sí. En su lugar, utilizan RabbitMQ como intermediario, lo que facilita la escalabilidad y el mantenimiento del sistema.

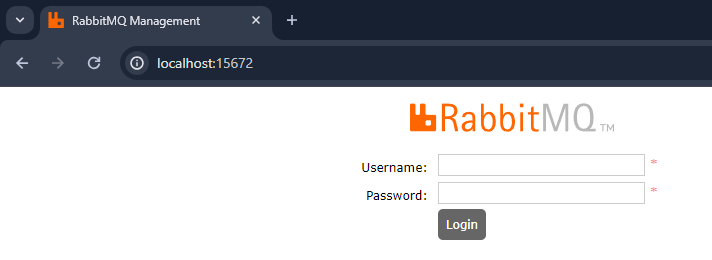
**ESTRUCTURA DEL PROYETO**

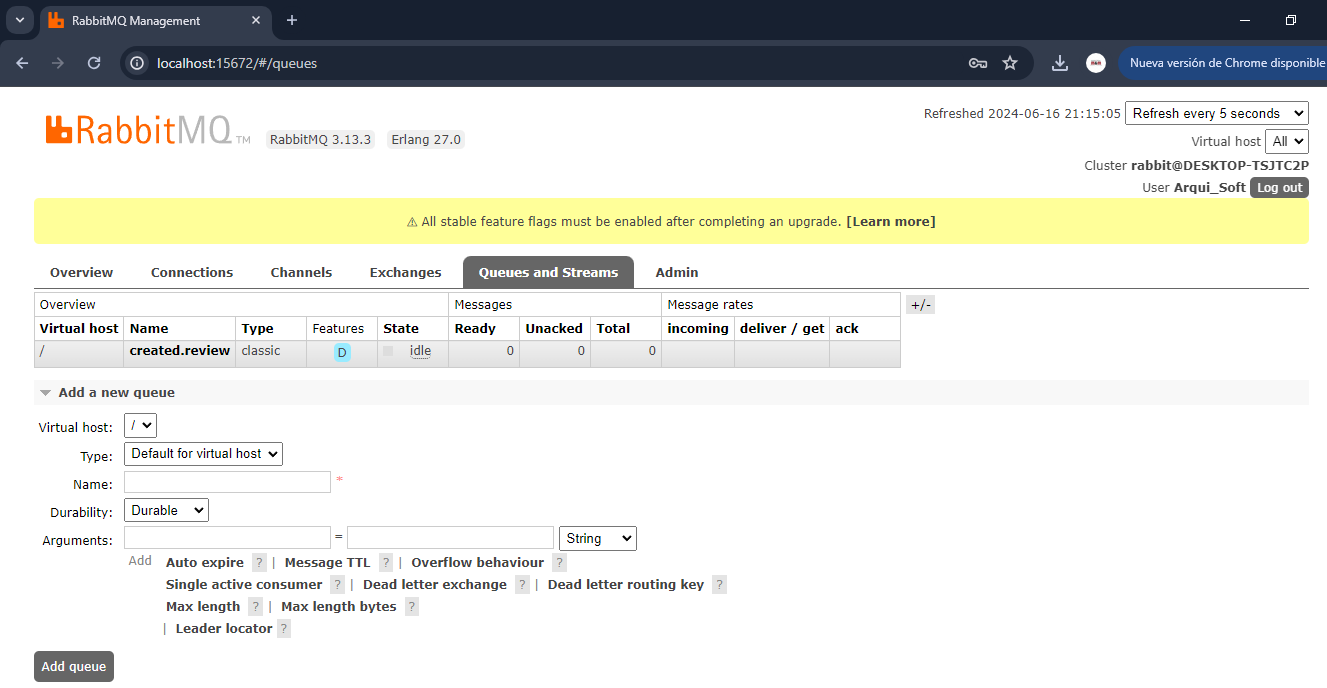
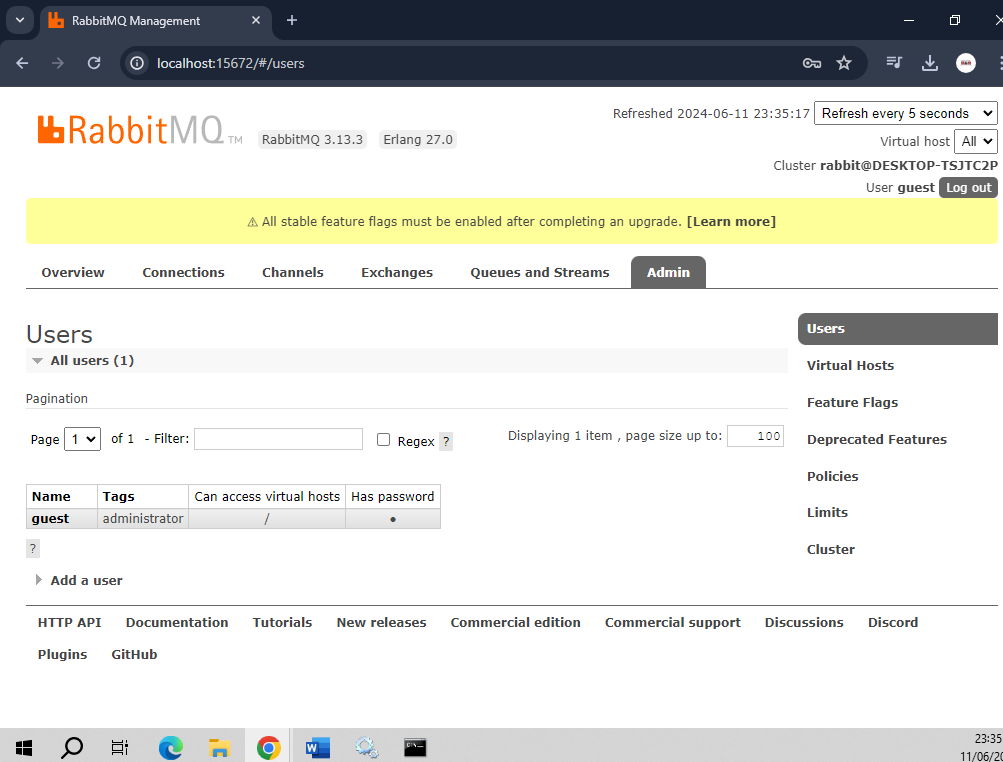
El proyecto se implementó en Visual Studio Code teniendo la siguiente estructura.



**RabbitMQ**

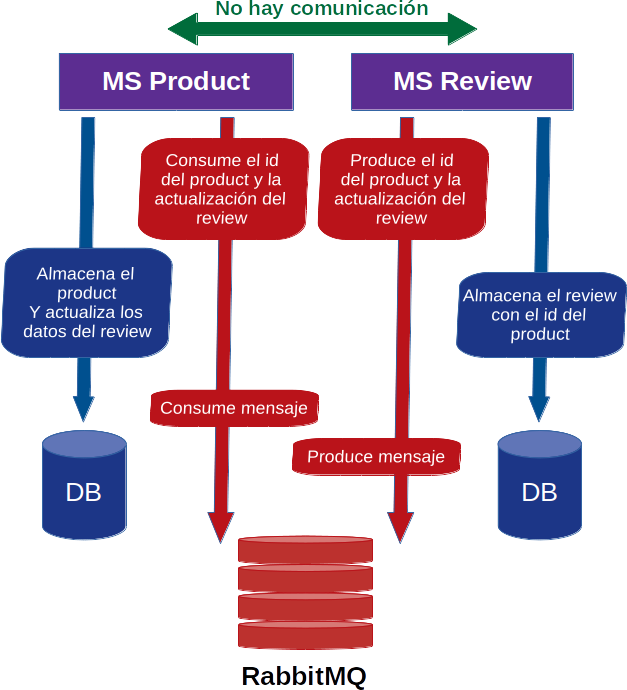
A continuación, se presentan los pasos esenciales para implementar esta solución utilizando RabbitMQ.





**El negocio**

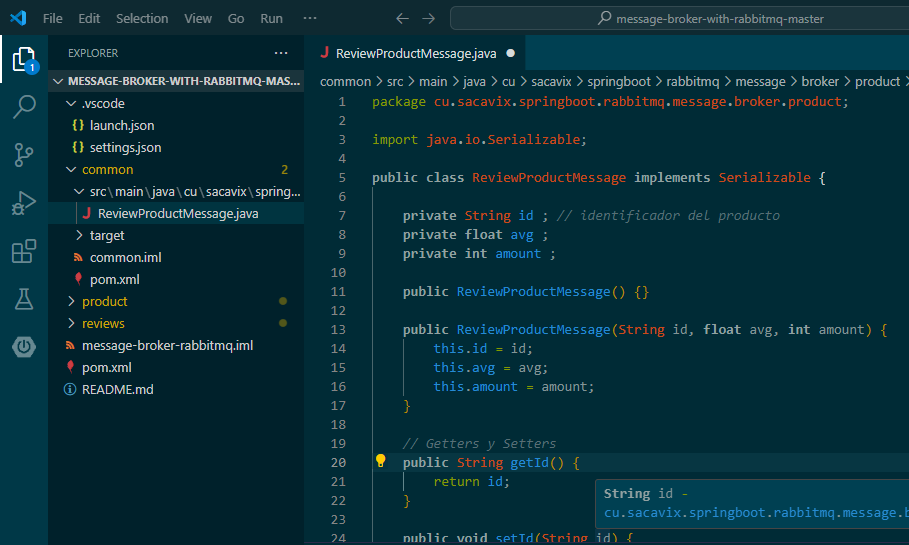
Para el ejemplo partimos de un negocio donde se gestionan productos y comentarios (los comentarios se le asignan a los productos con una evaluación), como arquitectura de microservicios tendríamos entonces dos de ellos, ***ms\_product*** y ***ms\_review***. La problemática es que siempre queremos representar en productos (al menos), la cantidad de comentarios y el promedio de la valoración del producto (*1 a 5*). Es aquí donde juega un papel fundamentar nuestro “*intermediario de mensajes*“. Analicemos la siguiente gráfica:



Veamos las principales clases que hacen posible el “intermediario de mensajes“

##### **Paquete común “common”**

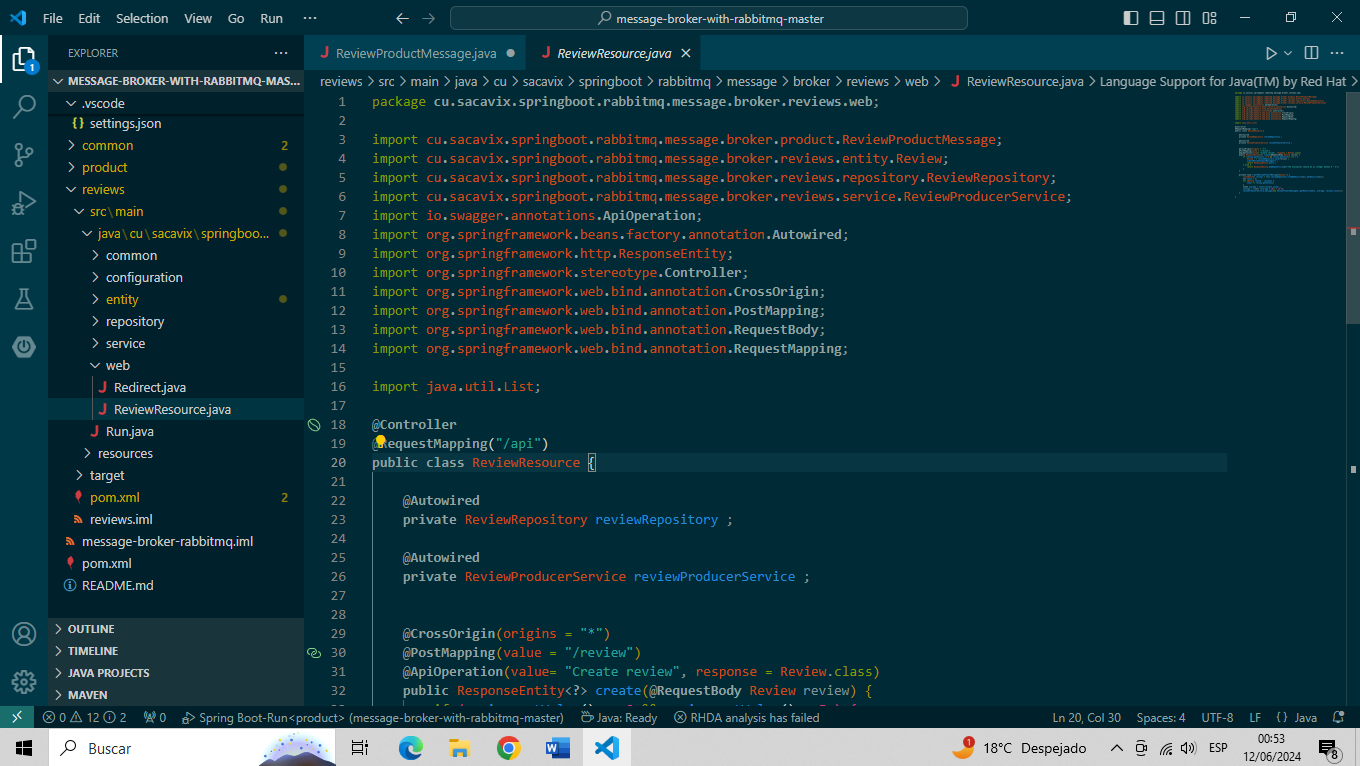
Este paquete tiene la clase que manejará los datos del mensaje y que ambos microservicios utilizarán:



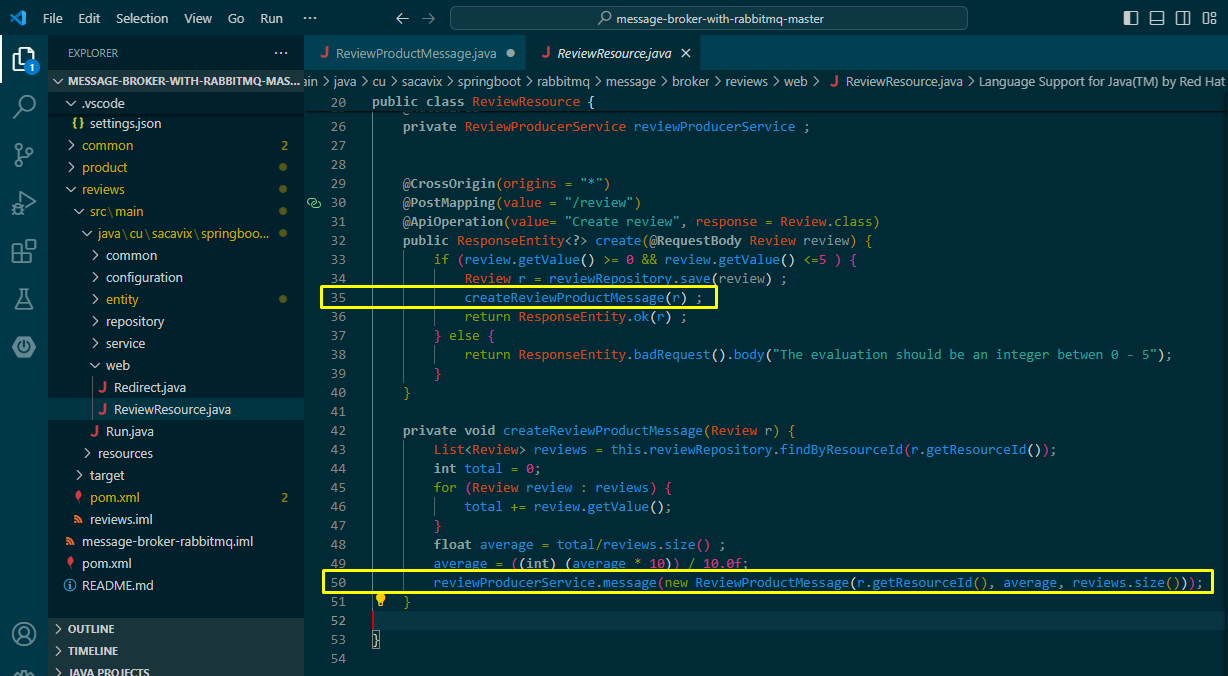
##### **Microservicio Review**

Este microservicio es el encargado de producir el mensaje ReviewProductMessage. En el recurso que brinda el endpoint para crear un review es donde se manda el mensaje:

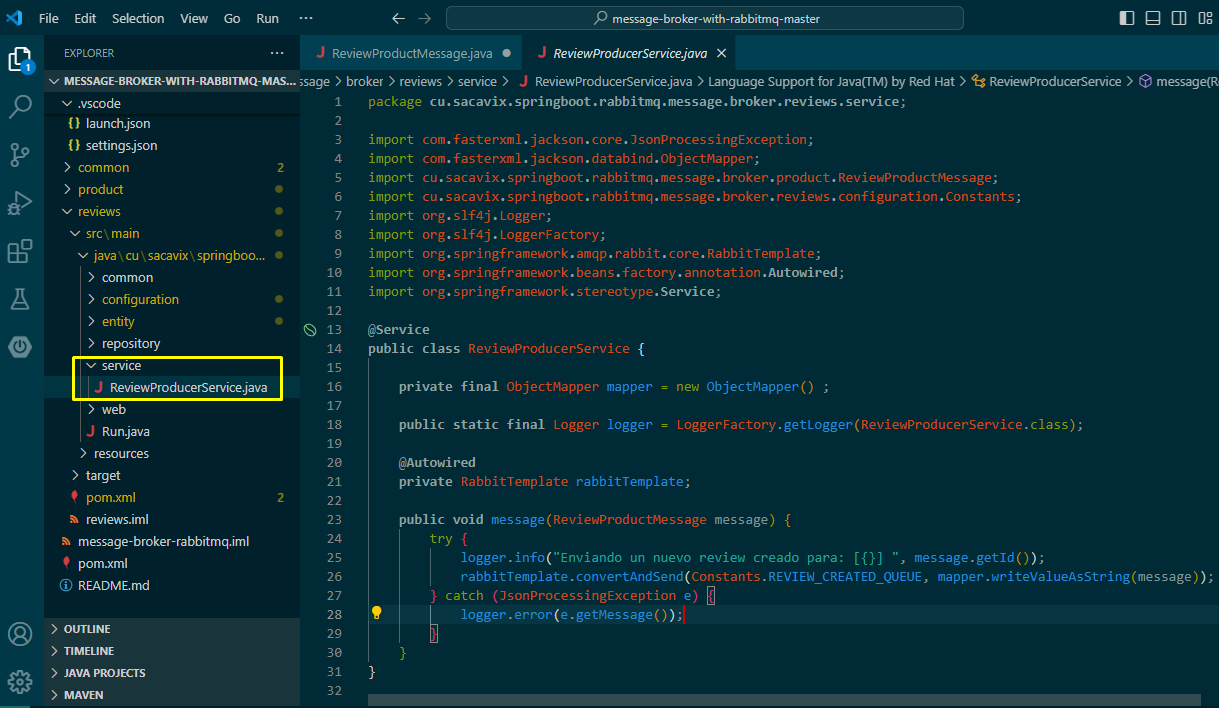
Clase **ReviewResource**



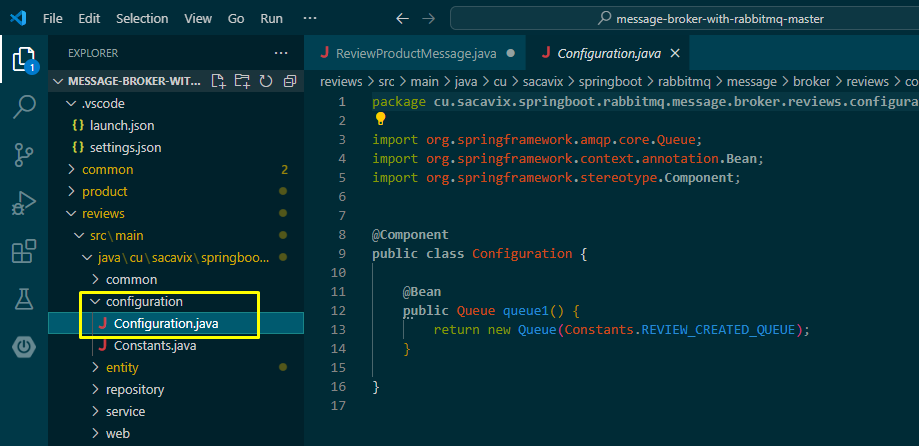
En la **línea 35** se realiza la llamada de la funcionalidad createReviewProductMessage y en las **línea 50** se llama el servicio encargado de enviar el mensaje.



Clase **ReviewProducerService**



Estas líneas de código son suficientes para enviar el mensaje para la cola REVIEW\_CREATED\_QUEUE, convirtiendo el mensaje ReviewProductMessage al formato **JSON**, la configuración de la cola se inicializa en la clase ***Configuration***, donde REVIEW\_CREATED\_QUEUE es una constante literal con el valor “**created.review**“



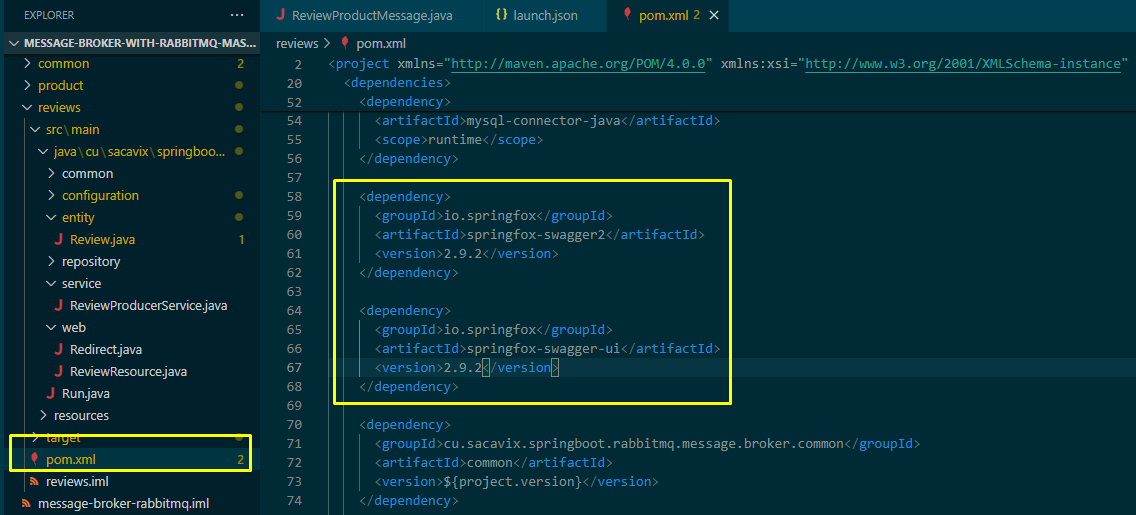
Si iniciamos este servicio sin tener el cosumidor (***ms\_product***) funcionando y creamos review, entonces los mensajes se encolarán hasta que sean consumidos, veamos esto en las siguientes tres imágenes:

**Creación de review s través del Swagger en ms\_review**

Paso 1: Configuración de Swagger en ms\_review

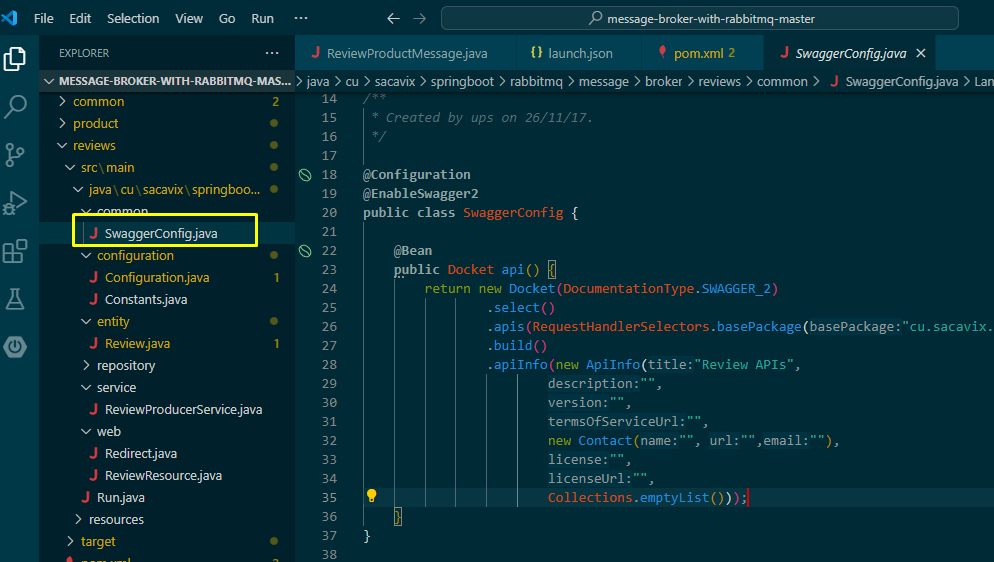
**Dependencias de Maven**

Incluir las dependencias necesarias para Swagger en tu pom.xml.

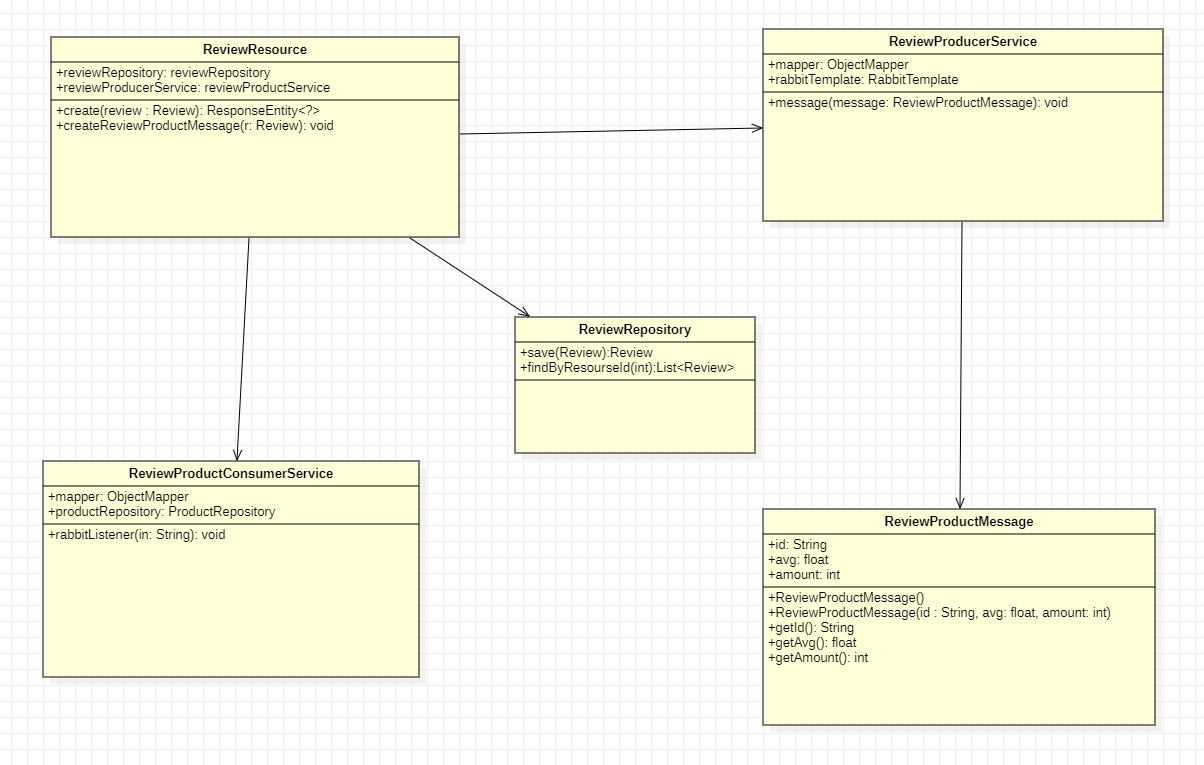


**Configuración de Swagger**

Crea una clase de configuración de Swagger en el paquete



**DIAGRAMA DE CLASES:**

****

**GITBUB**

https://github.com/WendyJacobo/ARQUITECTURA\_SOFT

**BIBLIOGRAFÍA**

https://sacavix.com/2020/11/message-broker-con-rabbitmq-para-microservicios/