75.74 Sistemas Distribuidos I
Primer cuatrimestre de 2024

TP1: Escalabilidad: Middleware y Coordinación de Procesos

Amazon Books Analyzer

Fecha de entrega parte 1: 16/04/2024
Fecha de entrega parte 2: 02/05/2024

Alumno	Padrón	email
Alvarez Windey Juan	95.242	jalvarezw@fi.uba.ar
Montenegro Lucas	102412	lmontenegro@fi.uba.ar

Índice

1. Introducción

Para el presente trabajo se realizará un sistema distribuido que analice las reseñas a libros en el sitio de Amazon para proponer campañas de marketing.

Las reseñas que se encuentran en el dataset poseen título del libro, texto del comentario y rating. Por cada título de libro, se conoce categoría, fecha de publicación y autores.

Requerimientos funcionales

El sistema deberá poder cumplir con las siguientes consultas:

- Título, autores y editoriales de los libros de categoría "Computers" entre 2000 y 2023 que contengan "distributed" en su título.
- Autores con títulos publicados en al menos 10 décadas distintas
- Títulos y autores de libros publicados en los 90' con al menos 500 reseñas.
- 10 libros con mejor rating promedio entre aquellos publicados en los 90' con al menos 500 reseñas.
- Títulos en categoría "Fiction" cuyo sentimiento de reseña promedio esté en el percentil 90 más alto.

Requerimientos no funcionales

- El sistema debe estar optimizado para entornos multicomputadoras
- Se debe soportar el incremento de los elementos de cómputo para escalar los volúmenes de información a procesar
- Se requiere del desarrollo de un Middleware para abstraer la comunicación basada en grupos.
- Se debe soportar una única ejecución del procesamiento y proveer graceful quit frente a señales SIGTERM.

2. Modelo de vistas 4 + 1

2.1. Escenarios

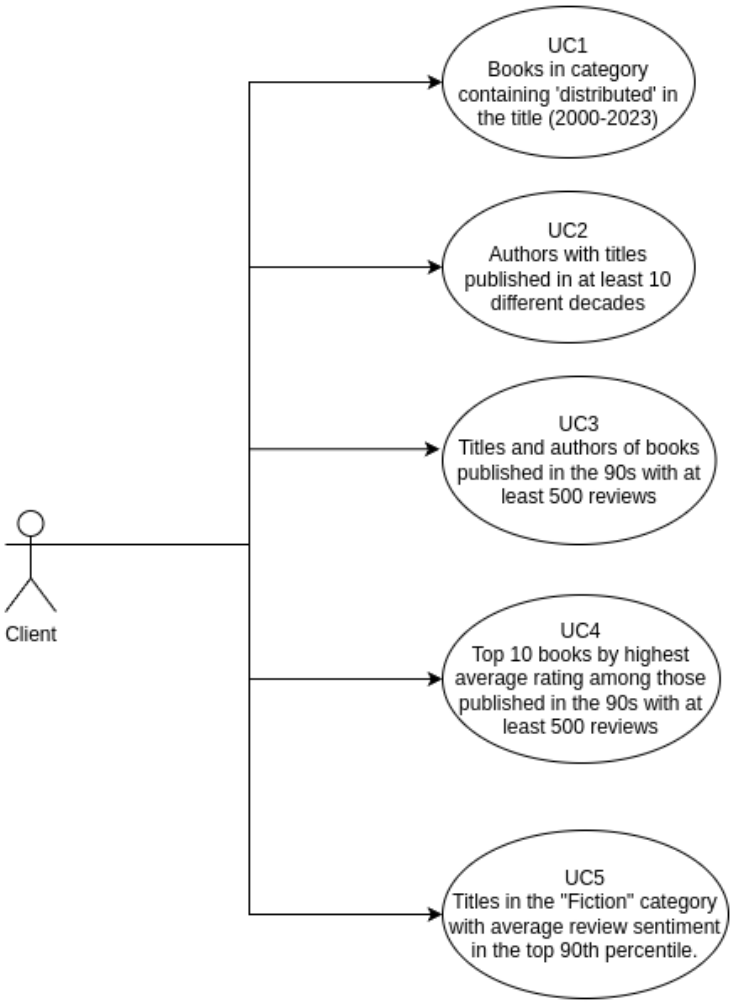

Figura 1: Diagrama de casos de uso

2.2. Vista lógica

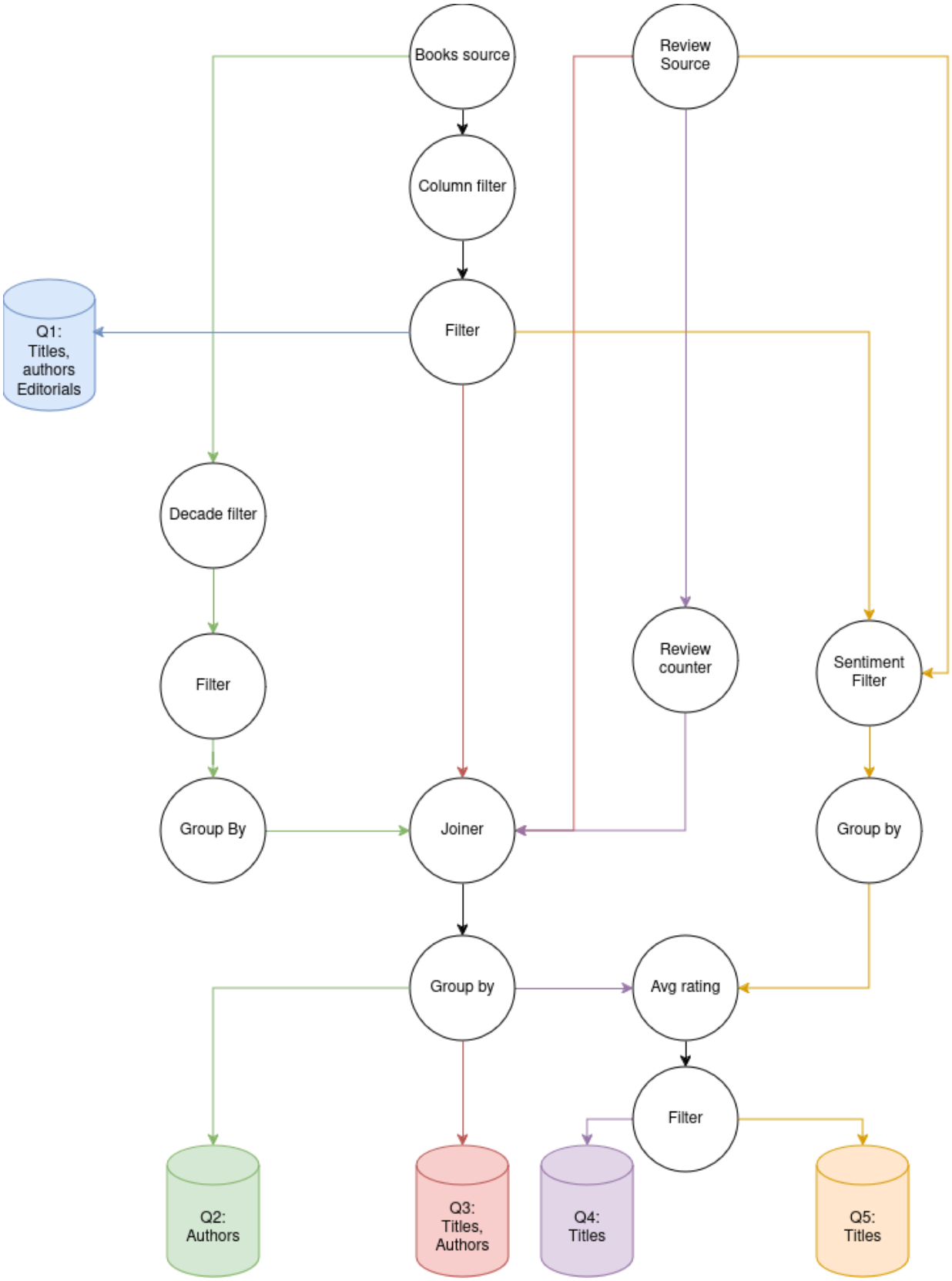
Books source
Review Source
Column filter
Filter
Q1: Titles, authors Editorials
Decade filter
Filter
Review counter
Sentiment Filter
Group By
Joiner
Group by
Group by
Avg rating
Filter
Q2: Authors
Q3: Titles, Authors
Q4: Titles
Q5: Titles

Figura 2: DAG

2.3. Vista física

2.3.1. Diagrama de despliegue

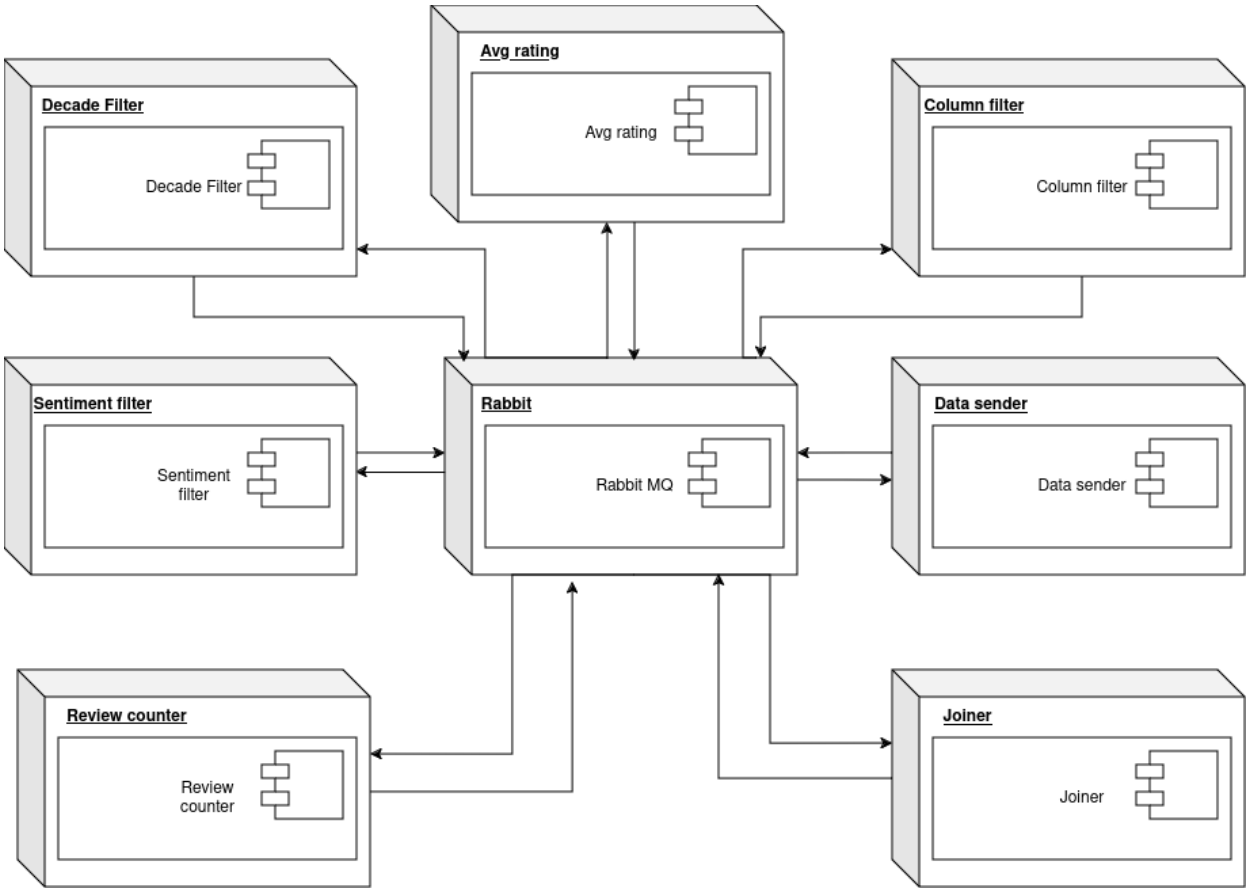

Figura 3: Deploy diagram

2.3.2. Diagrama de robustez

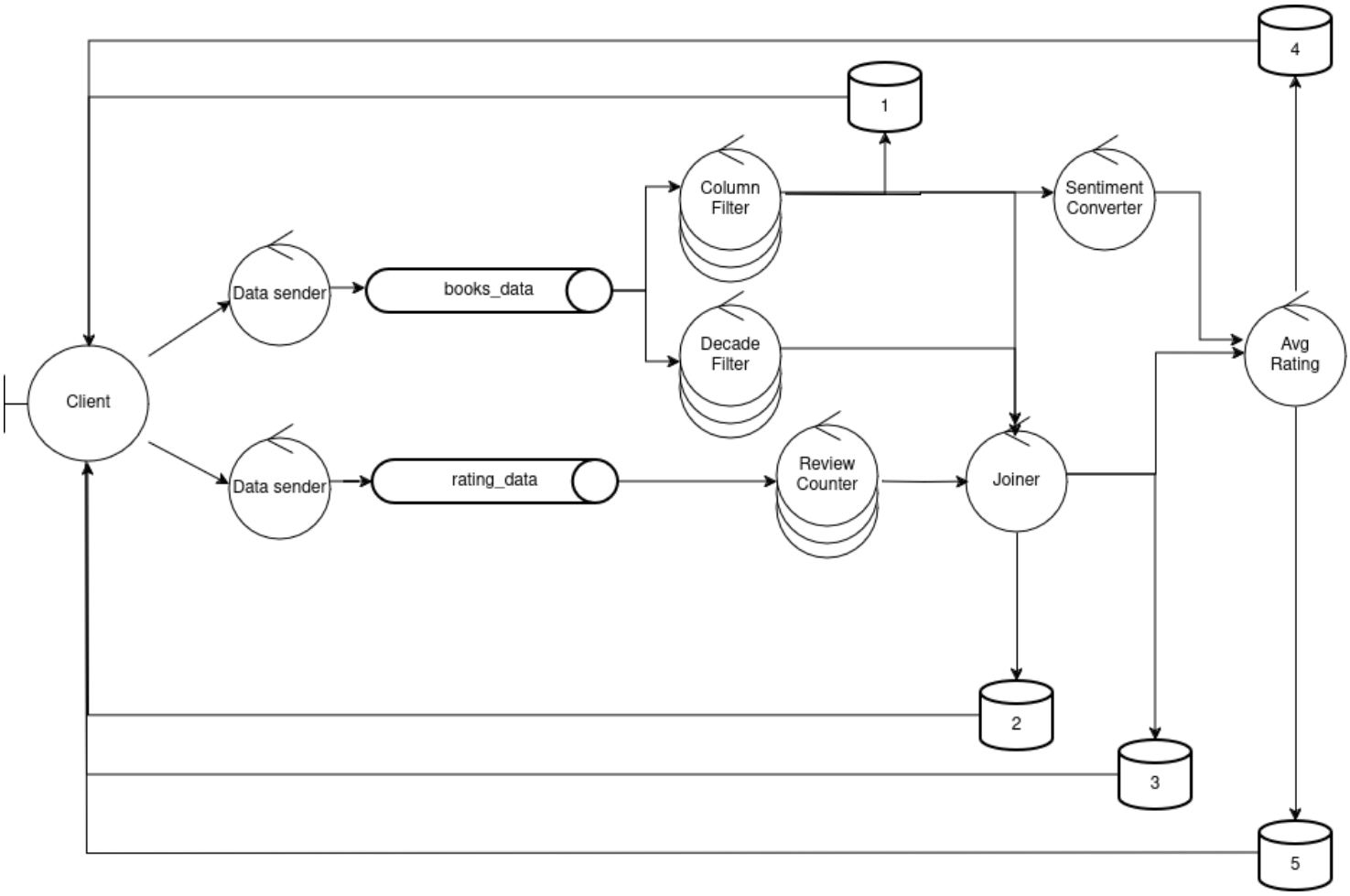

Figura 4: Robustness diagram

2.4. Vista de procesos

2.4.1. Diagrama de actividades

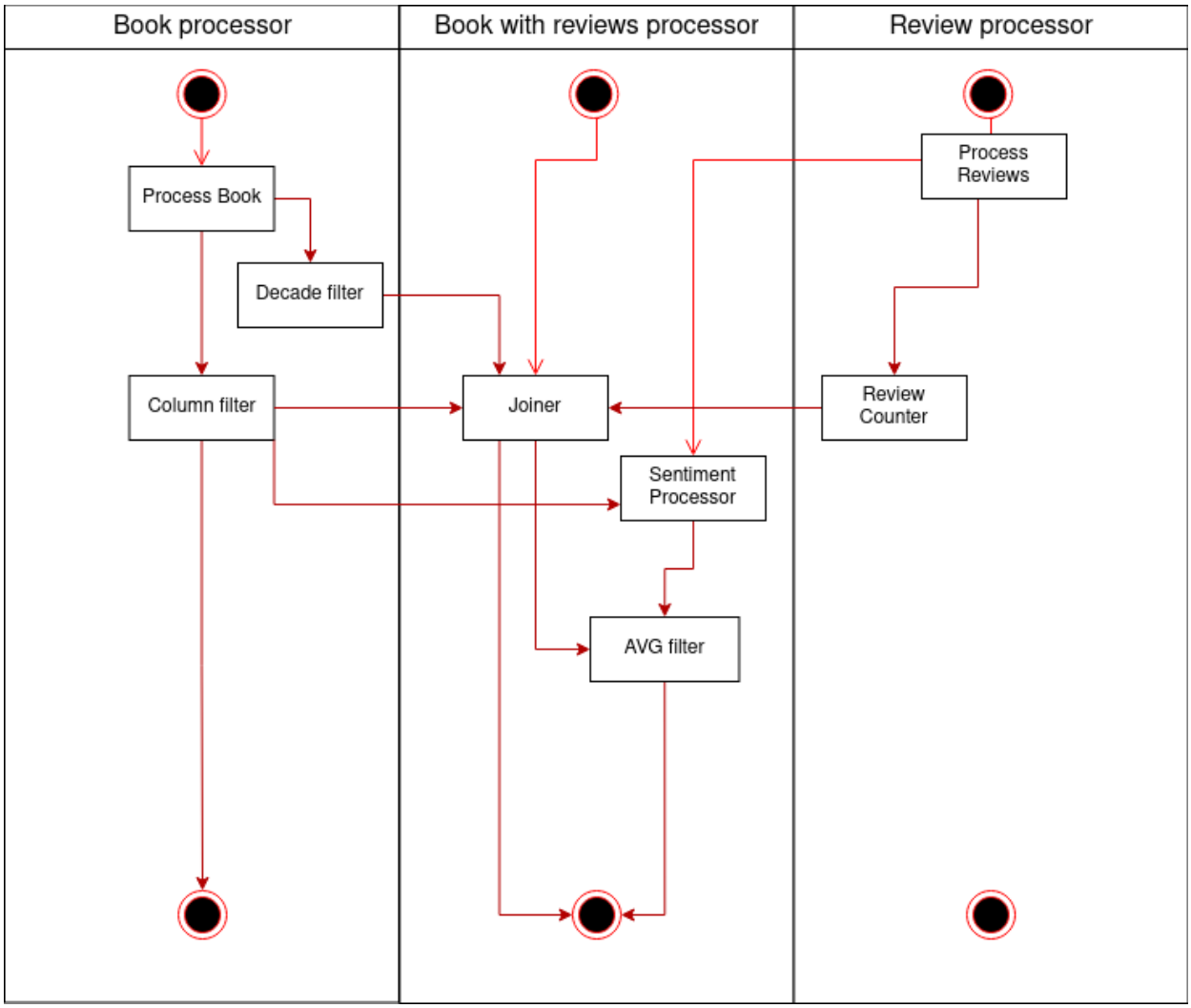

Figura 5: Activity diagram

2.5. Vista de Desarrollo

2.5.1. Diagrama de paquetes

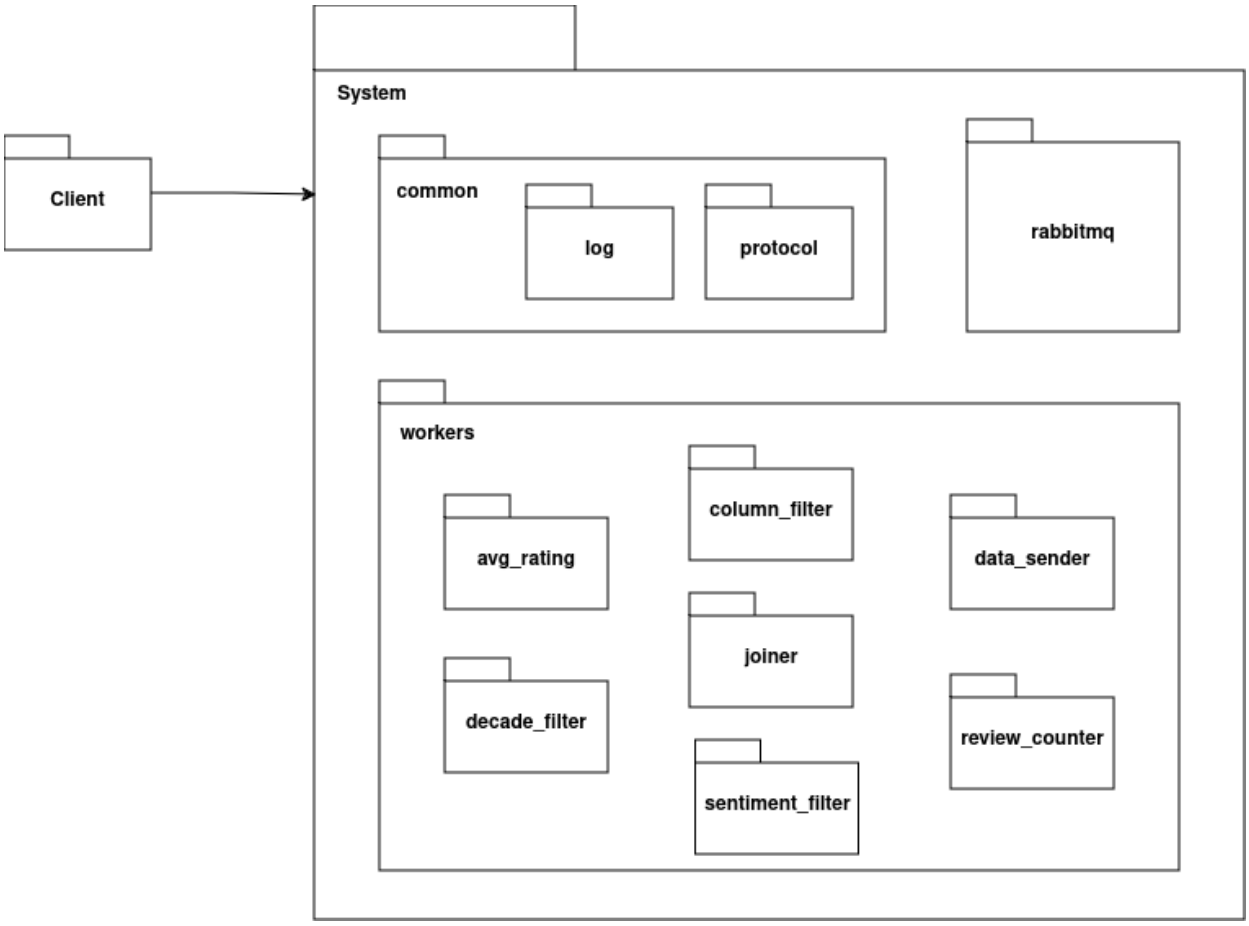

Figura 6: Package diagram

3. Detalles de implementación

3.1. Configuración del Entorno de Desarrollo

- Se instaló y se configuró RabbitMQ que fue necesario para la comunicación entre los distintos nodos del sistema.
- Se dockerizaron las distintas entidades del sistema, desde los nodos, rabbitmq y el cliente. Esto facilito en gran medida el esquema de pruebas y el trabajo de manera concurrente
- Para facilitar un poco más las pruebas se hicieron distintos archivos de docker-compose y un script .sh para poder levantar los componentes necesarios para resolver cada consulta

3.2. Desarrollo del Cliente

Se hizo un cliente simple en python que, usando unas variables de entorno, le comunica a los nodos encargados de iniciar el flujo que tipo de consulta es la requerida.

3.3. Conectividad con rabbit mq

Se utilizó la biblioteca pika en python para implementar tanto la comunicación del cliente al enviar las consultas como del sistema para los distintos filtros de procesamiento y los envios de datos.

Para el uso de la biblioteca se implementó en una clase llamada **QueueManager**. Dicha clase se encarga de la conexión a RabbitMQ, la administración de los distintos tipos de colas para la comunicación y del envío y recepción de mensajes mediante dichas colas.

Principalmente se usaron dos tipos de colas a la hora del uso de RabbitMQ: Productor/Consumidor y (en menor medida) Pub sub.

3.4. Desarrollo de los nodos del sistema

Se desarrollaron en python Los distintos nodos que se encargan de enviar los datos y su procesamiento. Todos los nodos para su comunicación utilizan la librería Pika que esta manejada por el **QueueManager**. A continuación una breve descripción de los nodos.

- **data_sender**: Es el nodo al que el cliente le envía el tipo de consulta y se encarga de, luego de hacer un filtrado base, mandar los datos a los distintos nodos del sistema. Los datos de los libros se los manda a **column_filter** en la Query1, Query3, Query4 y Query5, mientras que para la Query2 se los manda a **decade_filter**. Por otro lado, los datos de las reviews
- **column_filter**: Es el principal filtro para los datos de los libros. En la Query1, una vez filtrado envía los datos de resultado al cliente. En la Query3 y Query4 se los envía al joiner. Y por último en la Query5 se los envía al **sentiment_filter**
- **decade_filter**: Para la Query2 se encarga de hacer un filtrado de datos, y de agrupar libros con decadas. Luego de este procesamiento se envían los resultados al **joiner**
- **joiner**: Para la Query2 luego de procesar (filtrado) lo enviado por **decade_filter** envía los resultados al cliente. Para la Query3 luego de recibir lo del **review_counter** y el **column_filter** envía los resultados al cliente, mientras que para la Query4 envía los resultados al **avg_rating**

- **sentiment_filter**: Se usa para la Query5, recibe datos tanto del **column_filter** como de los datos de reviews por parte del **data_sender**. Se encarga de filtrar los datos de las reviews y hacer un procesamiento review usando la libreria TextBlob, que la ponderará entre -1 y 1 valor del sentimiento, y luego enviará estos datos a **avg_rating**
- **avg_rating**: Se usa para sacar promedio, tanto de los sentimientos de la Query5 como del valor de las reviews en la Query4. Luego de esto envía el resultado al cliente
- **review_counter**: Recibe los datos de las reviews, se encarga de agruparlas por libro y que cumplan con un mínimo de cantidad para la Query4