FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Sistemas Distribuidos I
TP Diseño

[75.74] - Cátedra Roca
Segundo cuatrimestre de 2025

Nombre	Padrón	Correo Electrónico
Carolina Pico	105098	cpico@fi.uba.ar
Julián García Sánchez	104590	jgarcias@fi.uba.ar

Índice

1. Enunciado

Se solicitó un sistema distribuido que pueda realizar un análisis de datos de ventas en una cadena de negocios de cafés en Malasia. En base a información transaccional de ventas, clientes, tiendas y productos ofrecidos se debe obtener:

1. Transacciones (Id y monto) realizadas durante 2024 y 2025 entre las 06:00 AM y las 11:00 PM con monto total mayor o igual a 75.
2. Productos más vendidos (nombre y cantidad) y productos que más ganancias han generado (nombre y monto), para cada mes en 2024 y 2025.
3. TPV (Total Payment Value) por cada semestre en 2024 y 2025, para cada sucursal, para transacciones realizadas entre las 06:00 AM y las 11:00 PM.
4. Fecha de cumpleaños de los 3 clientes que han hecho más compras durante 2024 y 2025, para cada sucursal.

Donde además se deben cumplir los siguientes 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.

Finalmente para esta entrega presentaremos un diagrama 4+1 incluyendo diagramas de robustez, despliegue, actividades, paquetes, secuencia, DAG y una división final de las tareas entre los integrantes del grupo.

2. Vistas 4+1

2.1. Vista Lógica

La estructura del trabajo se organiza en tres niveles principales: el **cliente**, el **gateway** y los nodos de procesamiento que hacen uso de la clase **Middleware**.

En primer lugar, la clase Client representa el punto de inicio del sistema, ya que se encarga de procesar y enviar los archivos, guardar reportes en archivos locales y recibir los resultados obtenidos.

La clase Gateway funciona como intermediario entre el cliente, mediante un protocolo TCP. A su vez, envía los datos del cliente a los nodos de procesamiento mediante un middleware. Sus responsabilidades incluyen procesar mensajes los mensajes recibidos por el cliente, recolectar reportes desde el pipeline y coordinar el inicio o detención del consumo de datos. De esta manera, actúa como punto central de comunicación y control.

La clase Middleware encapsula operaciones relacionadas con el consumo de datos (start_consuming, stop_consuming) mediante las Queues y los Exchange de RabbitMQ. Esta clase es utilizada por el gateway y los nodos de procesamiento, para facilitar la comunicacion entre ellos, publicando y consumiento mensajes mediante el.

A partir de allí, se definen distintos tipos de nodos para implementar funciones específicas.

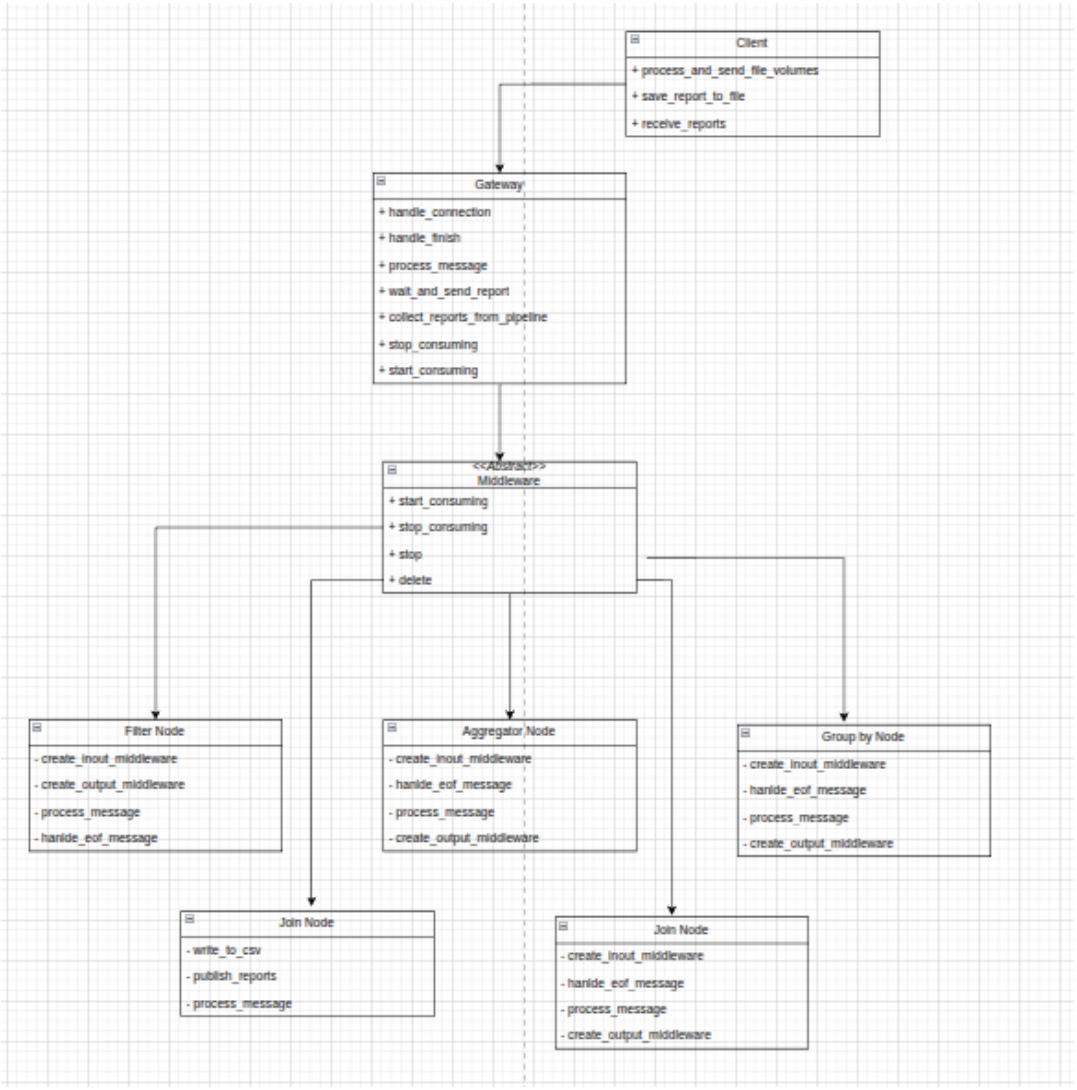

Figura 1: Diagrama de Clases.

Figura 2: Diagrama de Clases Nodo Filter.

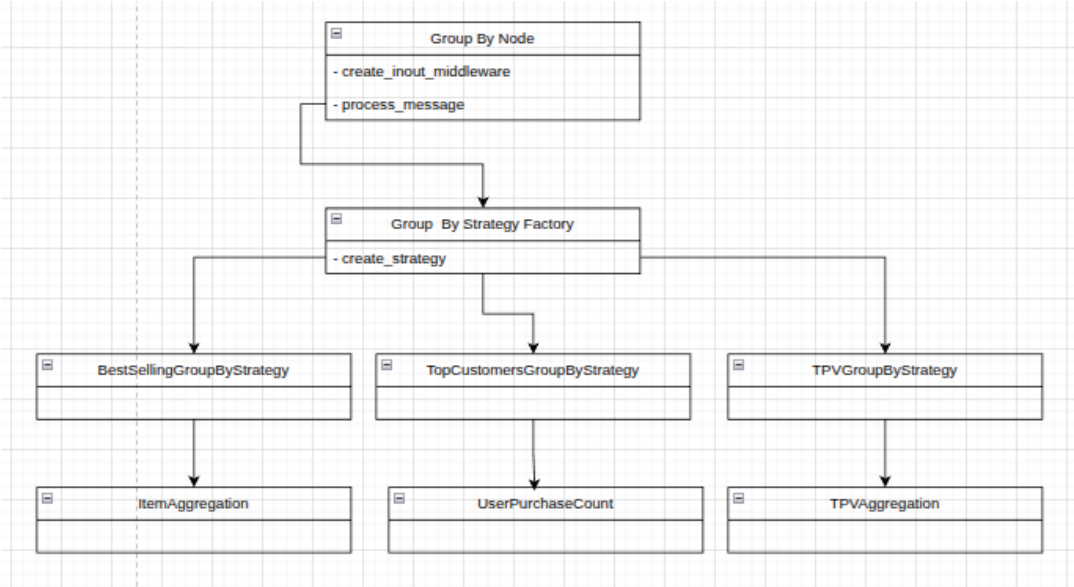

Figura 3: Diagrama de Clases Nodo Group By.

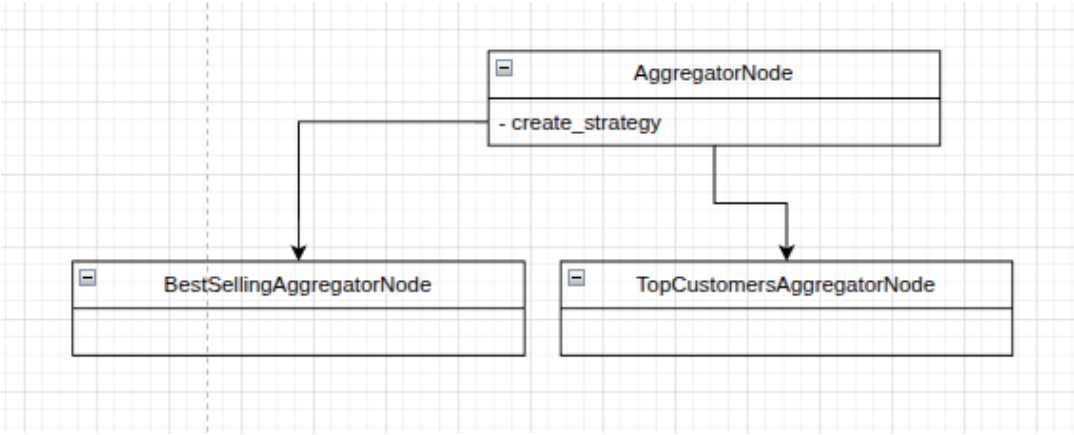

Figura 4: Diagrama de Clases Aggregator.

El diagrama presentado representa un DAG que describe el flujo de procesamiento de datos a partir de los archivos de origen. Antes de distribuir los datos a los nodos correspondientes, el Gateway se encarga de filtrar y enviar solo las columnas necesarias para realizar las consultas. A partir de esta fuente, se aplican distintos filtros iniciales. Los filtros de año se dividen entre los que procesan archivos de transactions y los que procesan archivos de transaction_items, pero ambos filtran sobre el campo transaction_id para obtener las de transacciones de 2024 y 2025, considerando la fecha de creación (created_at). Luego, los datos filtrados se someten a diferentes operaciones:

- Filtrado por hora de creacion (created_at), verificando que esté entre 06:00-23:00. Luego se extrae la columna final_amount y filtra transacciones 75.
- Agrupación por fecha de creación (created_at) y producto (item_id) segmentada por mes y año (2024 y 2025). Se extraen las columnas item_id, created_at, quantity y subtotal para crear agregaciones que acumulan sellings_qty (suma de cantidades) y profit_sum (suma de

subtotales) por producto y mes. Se calcula el Top 1 local por mes mediante ordenamiento por métricas y item_id.

Luego se consolidan todos los candidatos Top 1 recibidos de ambos años (2024 y 2025). Una vez recibidos todos los EOF, se calcula el Top 1 global y se envian los resultados finales (best_selling y most_profit).

- Agrupación por fecha de creación (created_at) y tienda (store_id) segmentada por semestre y año (2024 y 2025). El proceso extrae las columnas store_id, created_at y final_amount de cada transacción filtrada por hora (06:00 AM - 11:00 PM), creando agregaciones que acumulan total_payment_value (suma de final_amount) agrupadas por (year_half, store_id). A su vez se genera year_half en formato "YYYY-Hsemestre".
- Agrupación por tienda (store_id) y usuario (user_id). Esta agrupación implementa una estructura jerárquica de tres niveles: por cliente, por tienda y por usuario, donde se contabilizan las compras realizadas por cada usuario en cada tienda específica. El resultado genera métricas de purchases_qty que permiten identificar los clientes más frecuentes por establecimiento. Los datos agrupados son posteriormente procesados por tienda. Aqui se consolida el conteo de las compras recibidas, se calcula el Top 3 de usuarios por tienda mediante ordenamiento por purchases_qty y user_id.

Los resultados parciales son enviados al nodo join, que consolida los distintos cálculos con los archivos correspondientes, recibido desde el csv source. Finalmente, cada resultado de una query es enviada al nodo Generate Report.

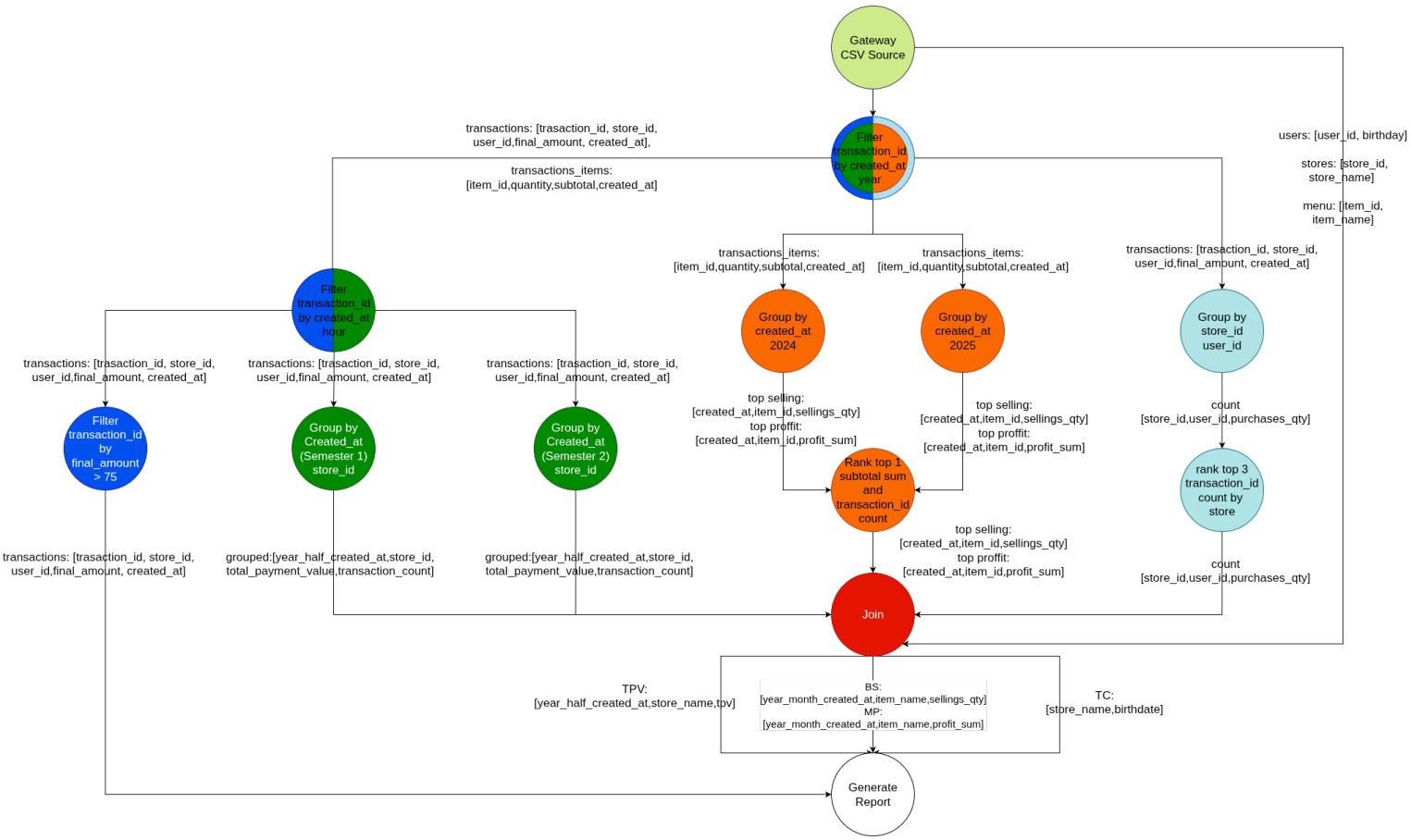

Figura 5: DAG.

2.2. Vista Desarrollo

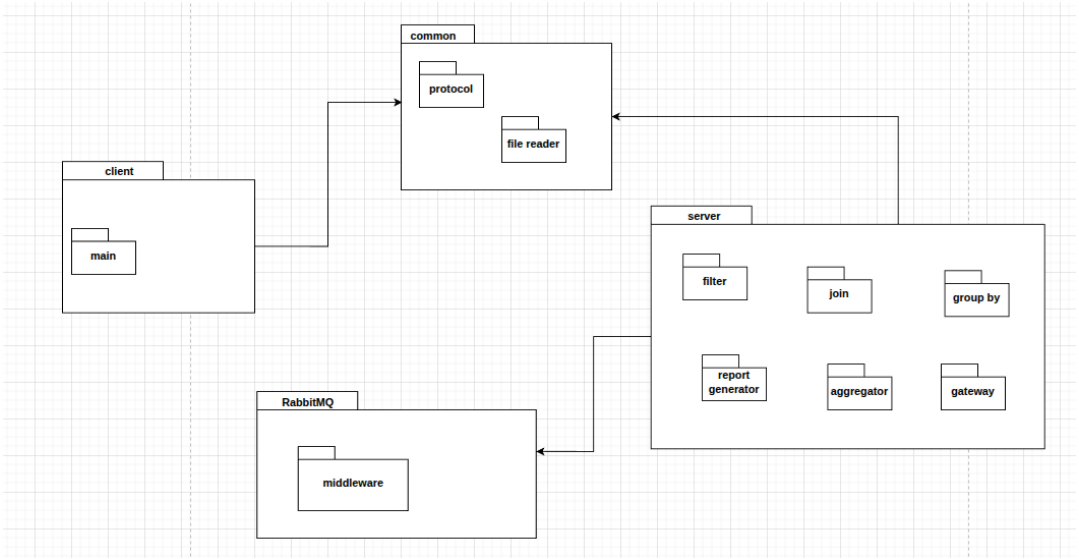

Figura 6: Diagrama de paquetes.

2.3. Vista Procesos

El flujo de propagación de EOF de la query 1 para los nodos de filtrado cambió respecto de la primer entrega. Antes un nodo recibía un EOF, lo volvía a meter en la cola de entrada y dejaba su escucha de la cola, asi no volvía a consumir ese EOF y daba la oportunidad a todos de consumirlo. Siendo el EOF puesto en la cola junto con el contador de nodos que lo recibieron, el último nodo propagaba el dicho dato. El problema encontrado para este TP fue que dichos nodos ya no podían apagar su esucha de la cola porque deben seguir procesando datos de otros clientes. Con este escenario tampoco podían volver a poner el EOF en la cola de entrada, porque un mismo nodo podria consumir indeterminadamente ese dato y no sería eficiente. Se decidió crear un helper para cada nodo llamado cordinador. Cada nodo tiene dos colas, una que esucha los datos principales, y otra que propaga el EOF y recibe ACKs de los otros nodos. Cuando un nodo recibe el EOF del hilo de datos principales, utiliza el coordinator para tomar el liderazgo del EOF de ese cliente y hace un fanout al resto de los nodos conocidos anunciando el fin de archivo. Cuando un nodo recibe el eof fanout, tiene la posibilidad de procesar un mensaje más (definido por el prefetch de las colas de rabbit) o tiene un timeout, al finalizar cualquiera de los dos eventos enviará un ack al nodo lider. Cuando el nodo líder recibe la cantidad de acks esperados, propaga el ack al resto de los nodos del pipeline.

Para el resto de los nodos involucrados en las otras queries, se utilizaron exchanges con routing keys, haciendo un fanout del EOF a todos los nodos. Por lo tanto estos nodos no presentaron dificultades en la coordinación de finalización de archivo.

La figura 7 demuestra el flujo completo de comunicación entre nodos filter para el EOF.

La figura 8 refleja el uso interno del coordinator en cada nodo filter, mediante el cual permite la comunicación de eof fanout y acks con otros nodos

Otro de los desafios encontrados con el procesamiento multicliente, fue el envio de los archivos users, por tener tanta cantidad de mensajes las colas de rabbit no alcanzaban a completar el procesamiento de los datos. Se decidió hacer un sharding de los datos desde el gateway al join por user id. Luego, cuando llegan los datos de los batches filtrados y procesados, cada nodo evalua si tiene ese user id en sus archivos guardados para hacer el join. En caso de no tenerlo, le solicita el dato a sus nodos pares. La figura 9 refleja la coordinación entre nodos join para obtener los datos de los users de cada cliente enviado por el gateway.

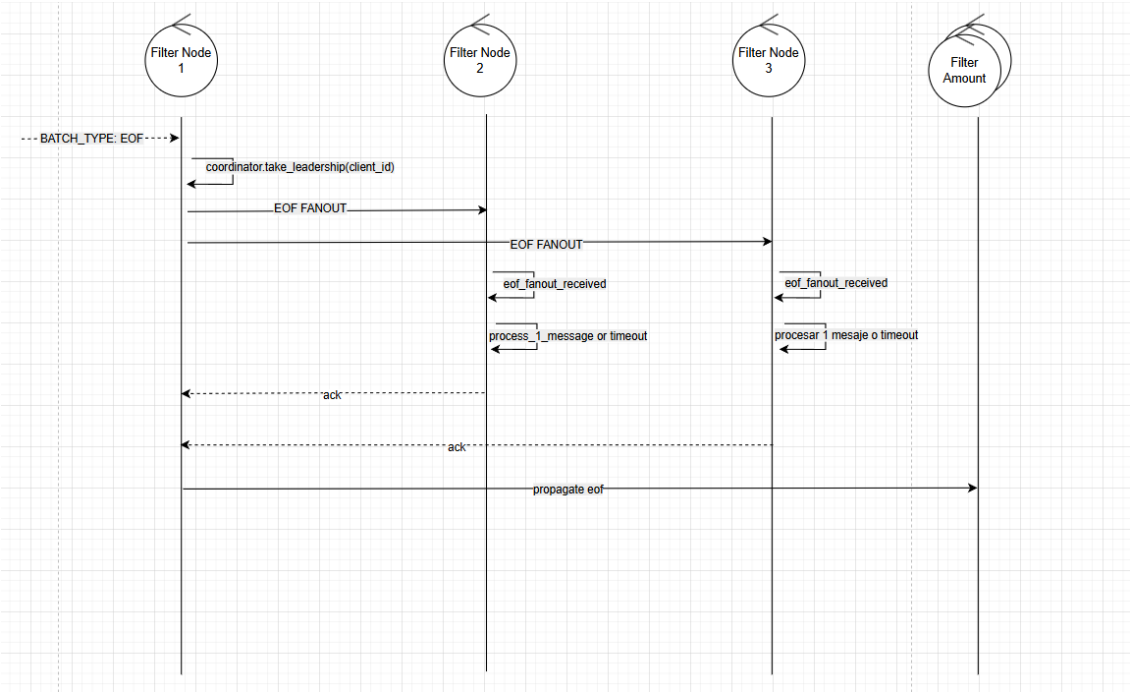

Figura 7: Flujo de comunicación entre nodos para EOF y ACK

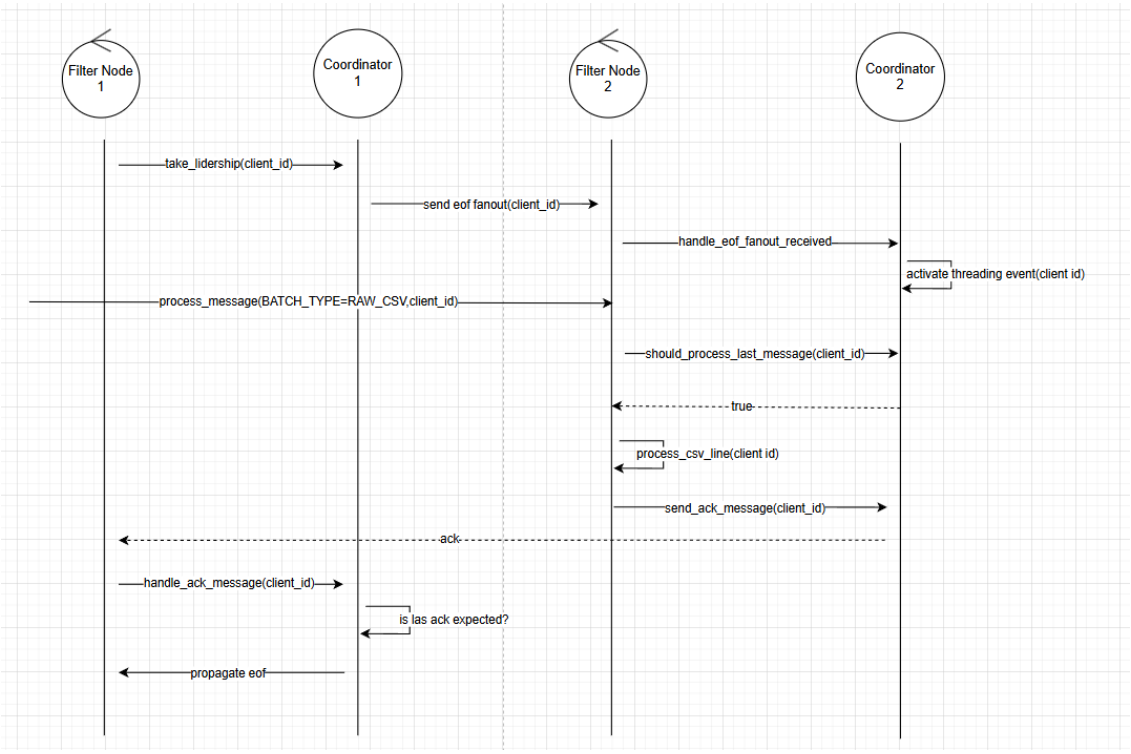

Figura 8: Flujo de secuencia del uso de coordinador para EOF

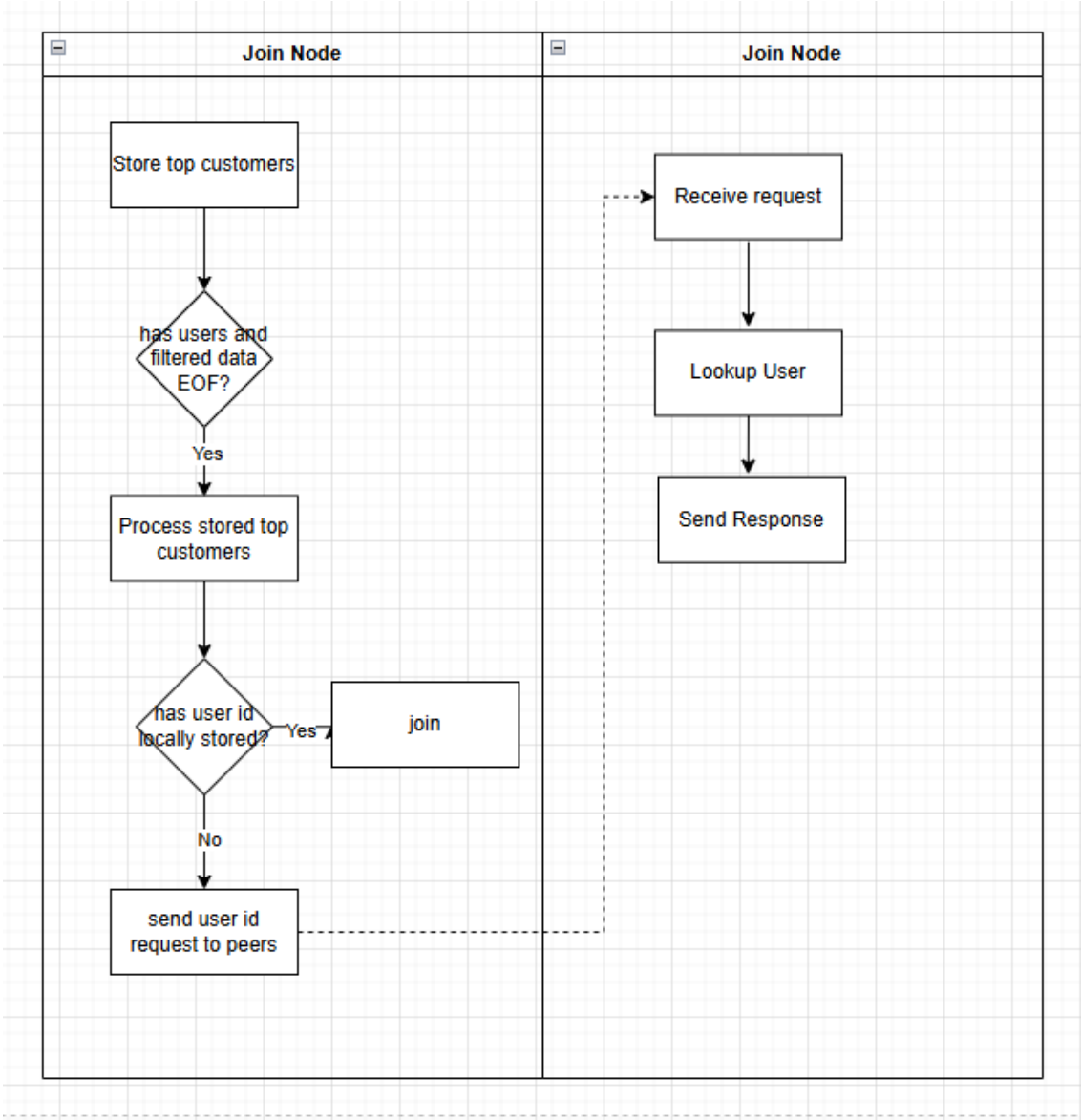

Figura 9: Flujo de secuencia de coordinación de datos Users entre nodos join

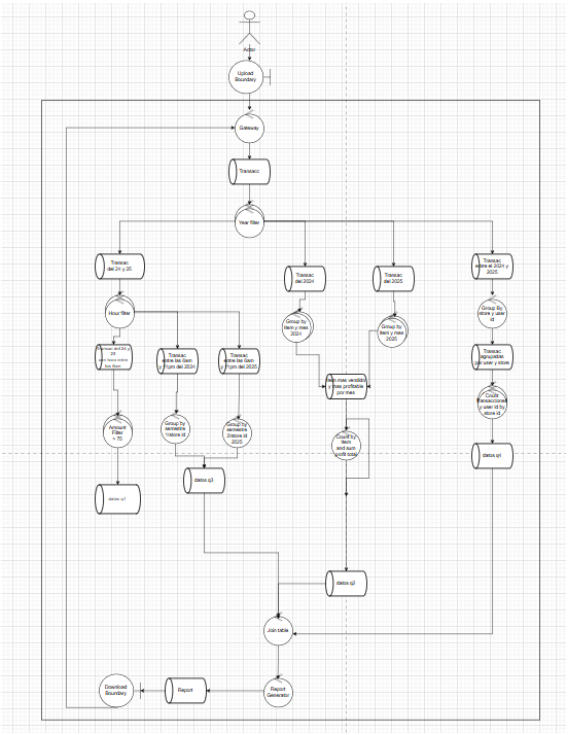

Figura 10: Diagrama de Robustez

2.4. Vista Física

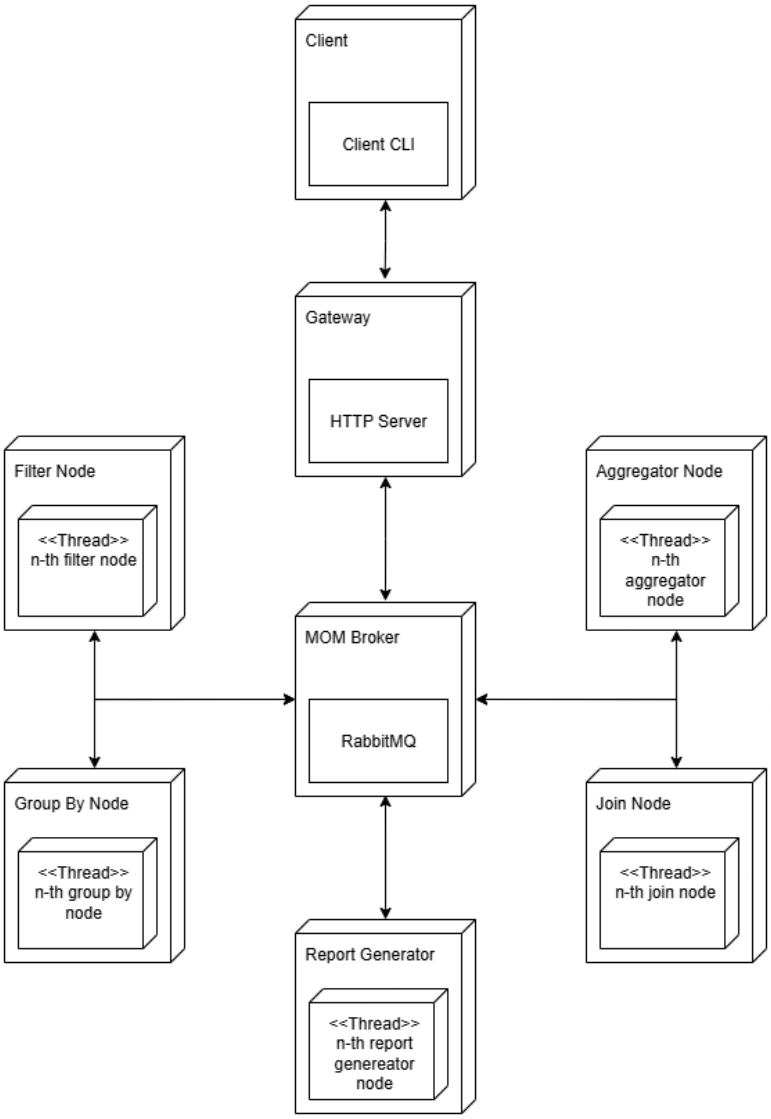

Figura 11: Diagrama de Despliegue.

2.5. Casos de Uso

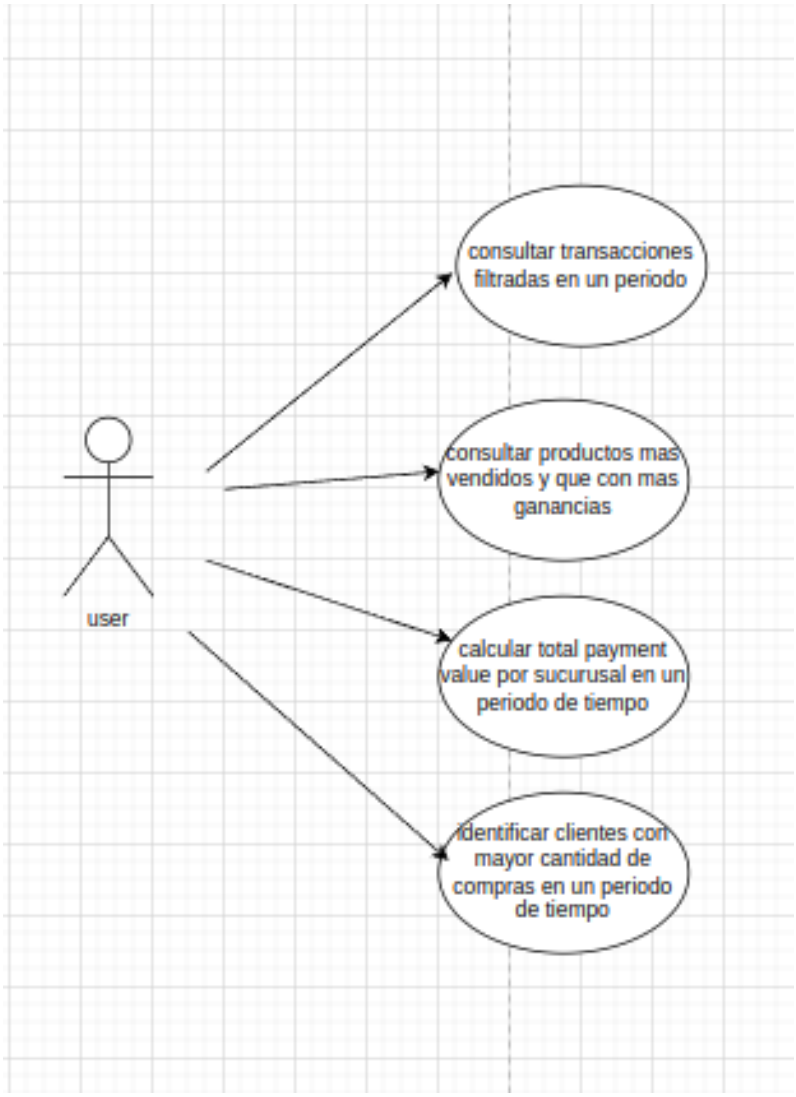

Figura 12: Diagrama de Casos de Uso.

3. División de Tareas

Nombre	Tareas
Cliente	Julián García Sánchez, Carolina Pico
Middlware	Julián García Sánchez, Carolina Pico
Filter Node	Julián García Sánchez, Carolina Pico
Group By Node	Julián García Sánchez, Carolina Pico
Join Node	Julián García Sánchez, Carolina Pico
Aggregator Node	Julián García Sánchez, Carolina Pico
Report Generator	Julián García Sánchez, Carolina Pico