Informe de Trabajo Práctico

Sistemas Distribuidos I

Integrantes:

Mateo Alvarez - 108666

Tomás Szejnfeld Sirkis - 107710

Martín González Prieto - 105738

Alcance **3**
Requerimientos Funcionales **3**
Requerimientos No Funcionales **3**
Escenarios **4**
Casos de Uso 4
Vista Lógica **5**
Clases 5
Vista de Procesos **6**
Actividades 6
Secuencias 8
Comunicación entre nodos procesado del mensaje END 10
Algoritmo de Elección de Líder 10
DAG 10
Vista de Desarrollo **12**
Paquetes 12
Componentes 14
Vista Física **15**
Arquitectura de Nodos 15
Middleware de Mensajería 15

Alcance

Requerimientos Funcionales

- Se solicita un sistema distribuido que analice la información de ventas en una cadena de negocios de Cafés en Malasia.
- Se cuenta con información transaccional por ventas (montos, items vendidos, etc), información de los clientes, de las tiendas y de los productos ofrecidos.
- Queries a hacer:
    - 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 cant) 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.

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.

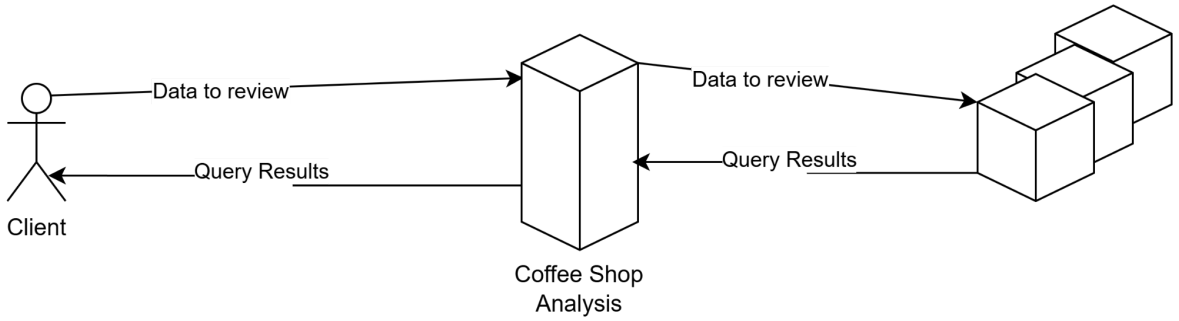
Escenarios

Casos de Uso

En el **Diagrama de Casos de Uso**, se identifican los siguientes elementos:

- **Actor Cliente**: Usuario que realiza las consultas al sistema, solicitando información para su análisis.
- **Servidor Coffee Shop Analyst**: Representa el sistema encargado de recibir y procesar los resultados de las consultas realizadas.
- Nodos de analisis: Estos serian los filtros a aplicar a los datos enviados
- **Casos de Uso (Pedido de Querys)**:
    - *R*epresenta los datos enviados y los distintos tipos de respuestas que el cliente puede recibir del sistema. Cada una de estas respuestas es enviada desde el sistema.

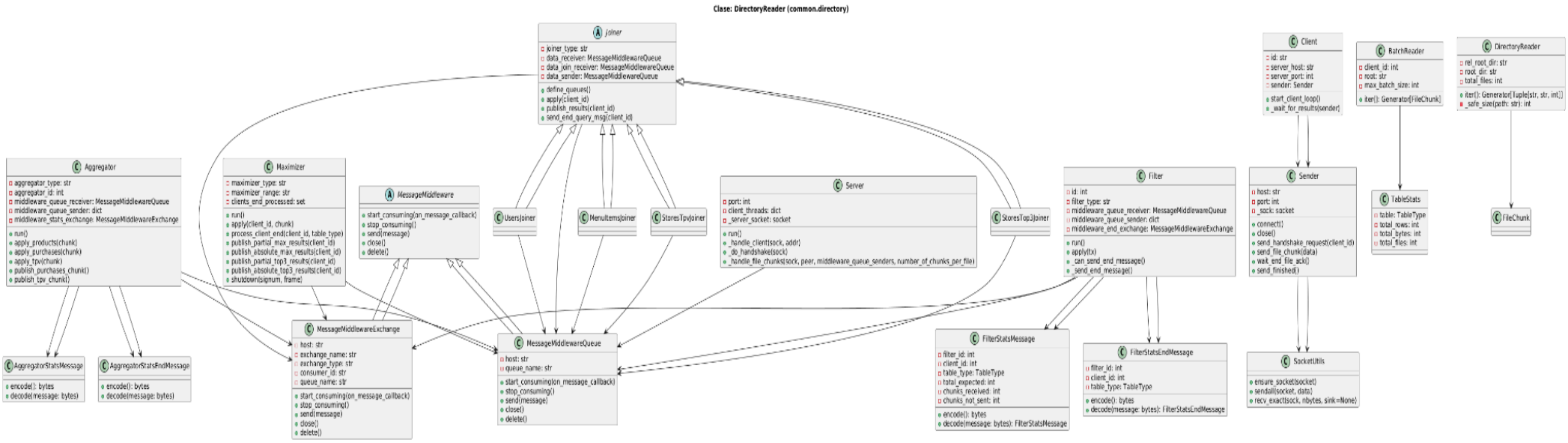
Los escenarios se centran en el flujo de información entre cliente y el sistema Coffee Shop Analysis, destacando cómo el sistema soporta múltiples tipos de consultas. Además, este modelo refuerza el rol del sistema como un **intermediario entre los datos y el análisis de negocio**, asegurando que la interacción sea clara, flexible y extensible a futuros tipos de consultas.



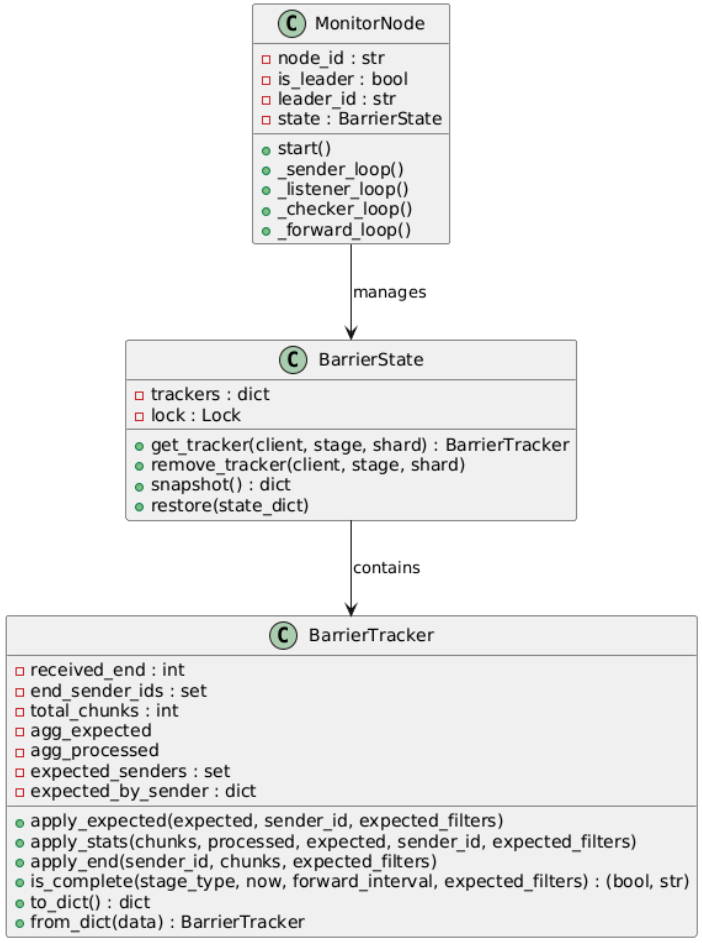
Vista Lógica

En esta vista se describen los principales módulos del sistema y sus responsabilidades dentro del procesamiento de consultas y manejo de archivos:

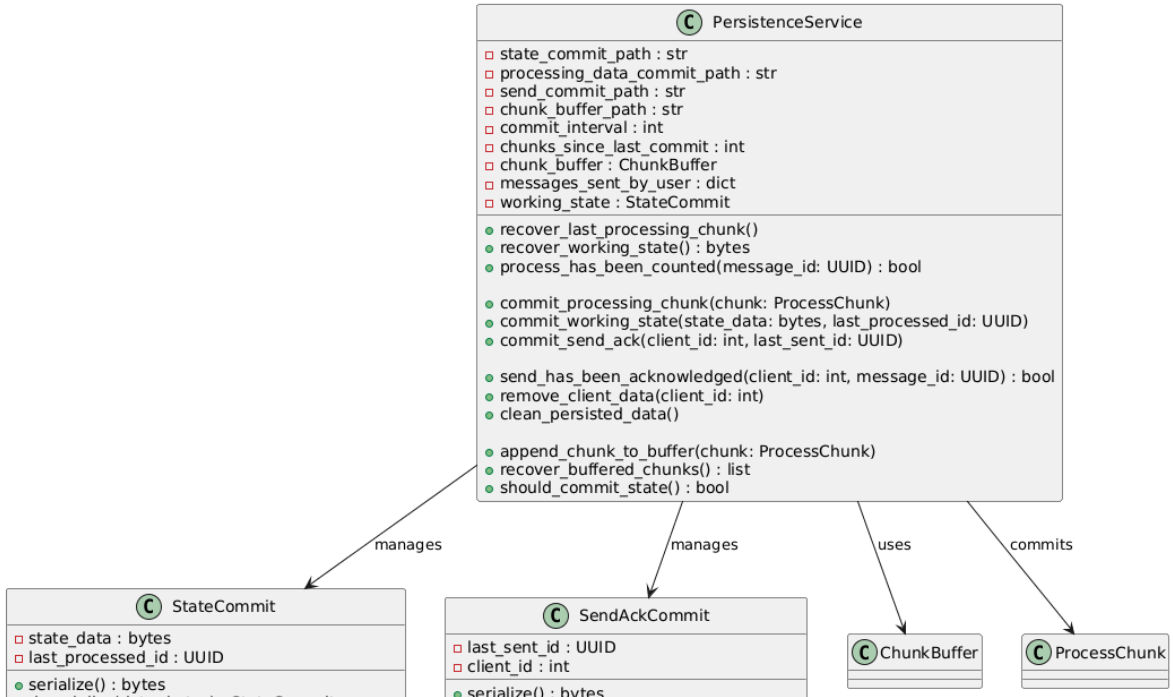
Clases



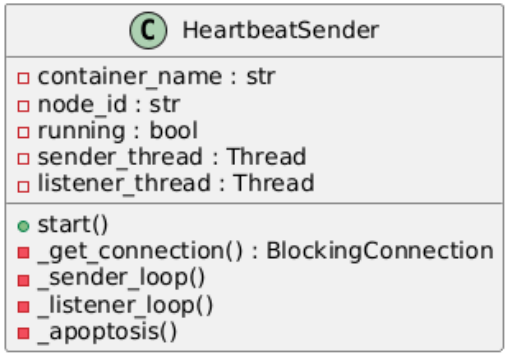
Monitor Node Classes



Persistence Service - Unified Classes

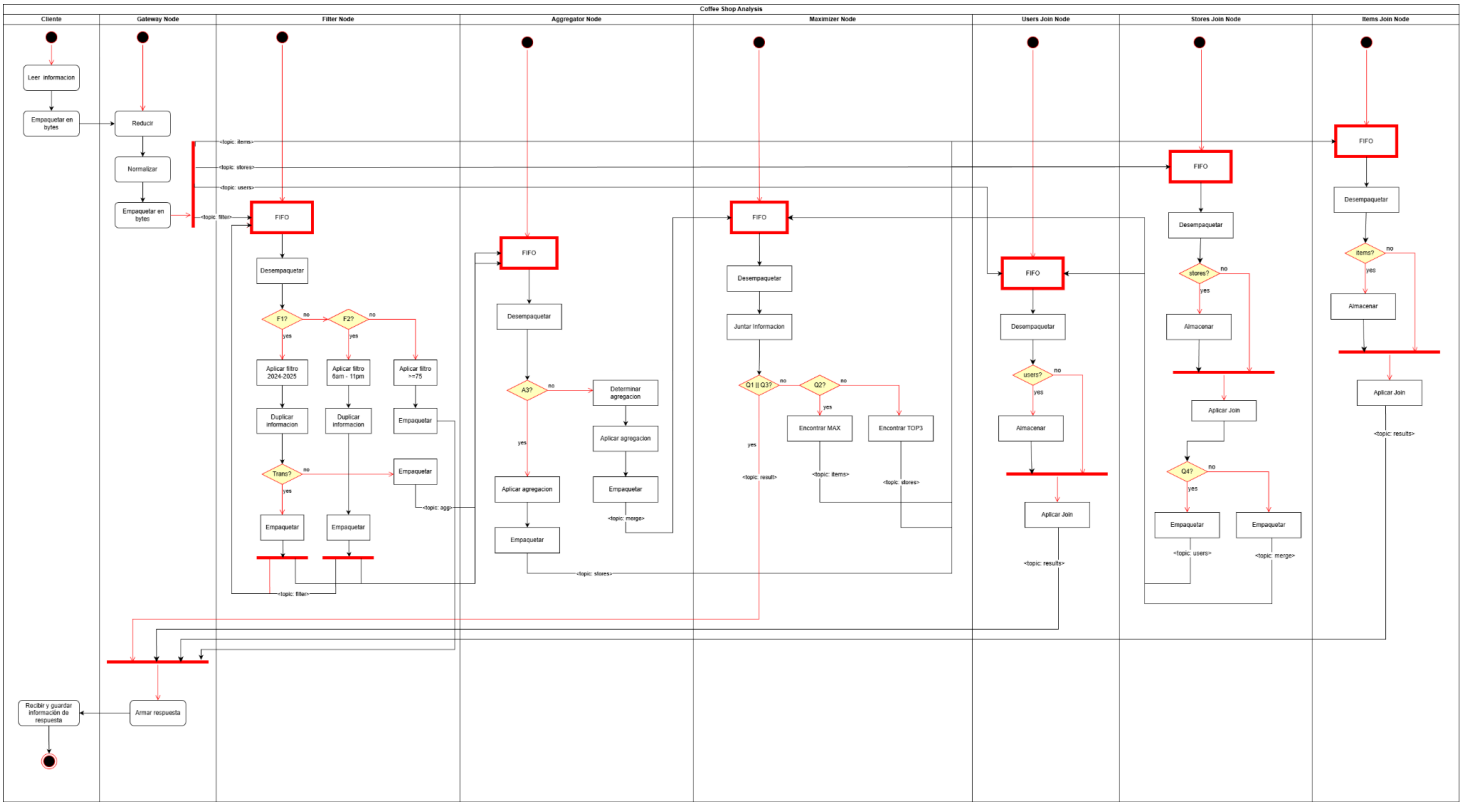


HeartbeatSender



Vista de Procesos

Actividades



El diagrama representa el **flujo completo de procesamiento distribuido** del sistema, organizado por nodos funcionales especializados. Cada carril describe las actividades que ocurren desde la recepción inicial de datos hasta la generación final de resultados.

El proceso inicia en el **Cliente**, donde la información es leída y empaquetada para ser enviada al **Server Node**. En este nodo se realiza una reducción y normalización de los datos, que luego se reempaquetan y distribuyen a los distintos tópicos del sistema.

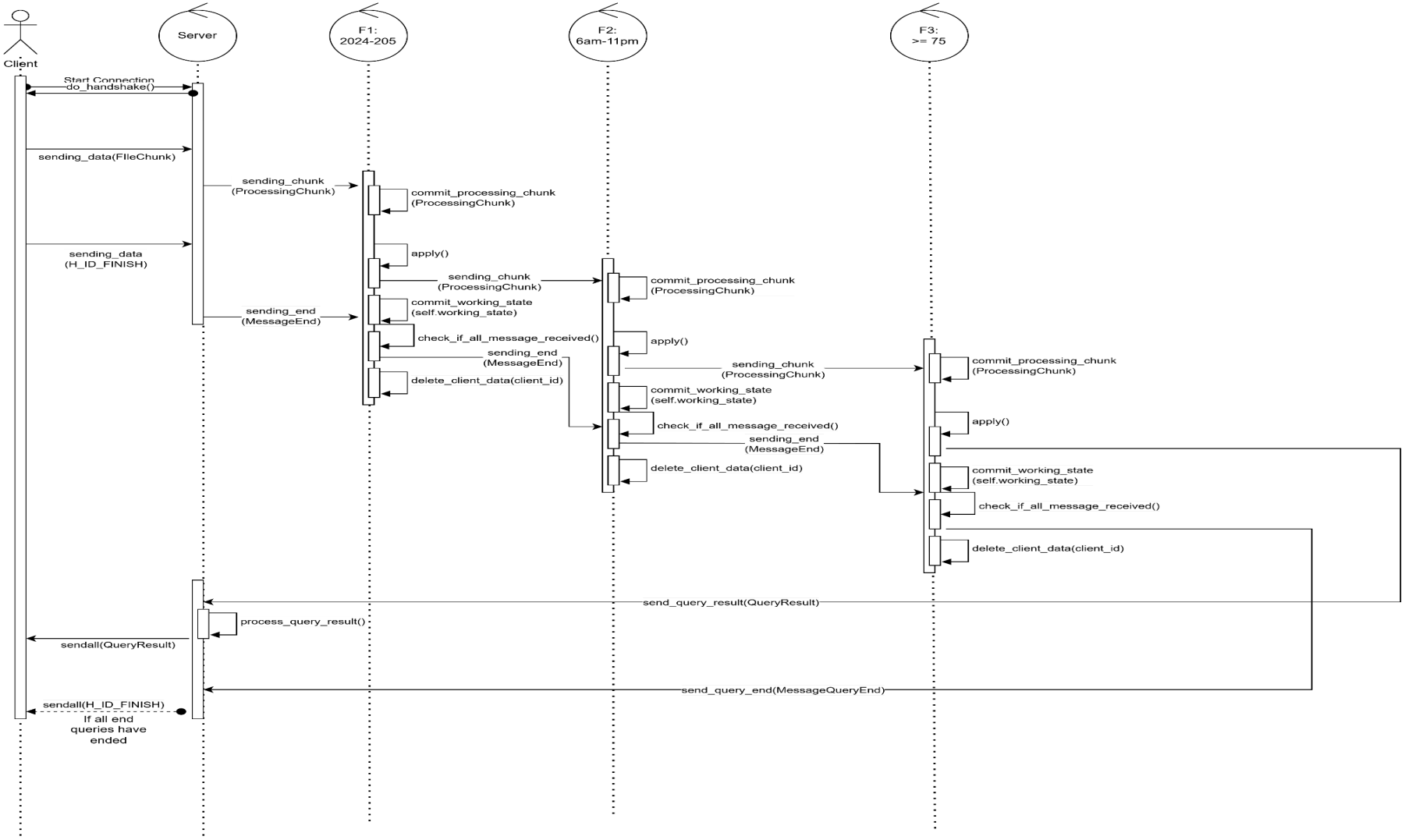
Después tenemos los **Worker Nodes** (Filter, Joiner, Aggregator y Maximizer) que son los encargados de procesar los datos para lograr los resultados correctos.
Ejecutan las operaciones específicas dependiendo de cada query:

- Query 1: filtrar por fechas, horarios y montos, luego agregar montos.
- Query 2: calcular productos más vendidos y más rentables por mes.
- Query 3: calcular TPV semestral por sucursal en franjas horarias válidas.
- Query 4: contar compras por cliente, obtener top 3 y cruzar con datos de cumpleaños.

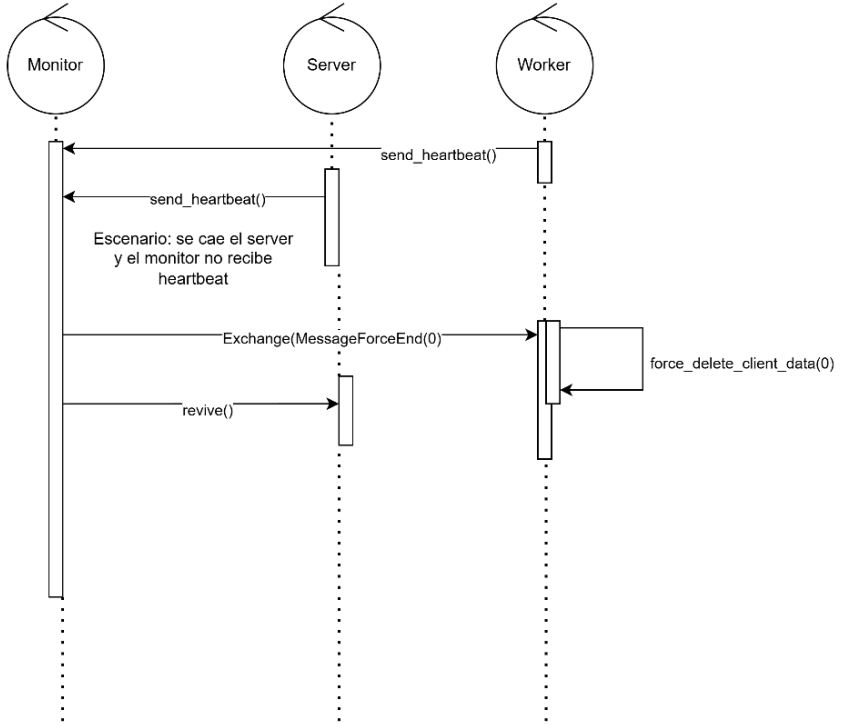
Después de procesar manda los resultados parciales al servidor.
El servidor reúne todos los resultados parciales, los consolida y los envía al cliente, quien los recibe y los presenta en pantalla, finalizando así el proceso.

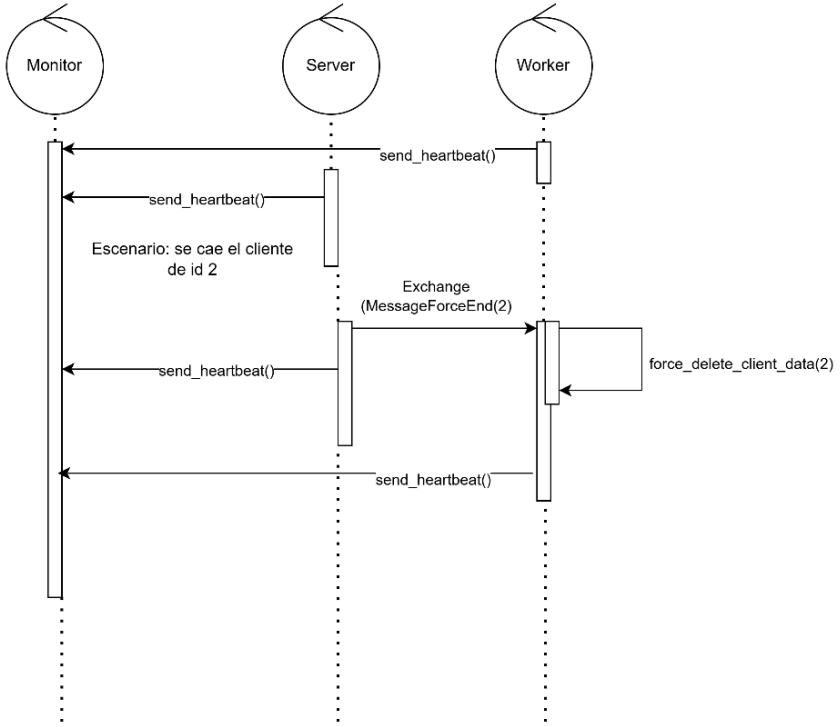
Secuencias

Query 1 – Transacciones filtradas por año, hora y monto



Escenarios de caída de servidor y cliente





Comunicación entre nodos del mensaje END

El manejo de **End** dentro del pipeline se basa en un mecanismo de conteo y sincronización que asegura que todos los chunks hayan sido procesados antes de permitir que el flujo continúe. El servidor inicialmente distribuye los chunks en distintos shards según su ID y, además de enviarlos, informa a cada shard cuántos chunks deberá procesar.

Cada **filter** mantiene un registro de cuántos chunks ha enviado a cada shard, y esa información fluye hacia abajo en el pipeline para que los siguientes nodos sepan exactamente cuánta data deben esperar. Paralelamente, los **filters** comunican al monitor la cantidad total de chunks que enviaron. Con esta información, el **monitor** calcula el número total de chunks que el sistema debería procesar.

A medida que los **aggregators** reciben y procesan los chunks, le notifican al **monitor** cuántos han completado. El monitor compara continuamente el total procesado con el total esperado. En el momento exacto en que ambas cifras coinciden, el monitor envía un mensaje de "*barrier forward*" a los **aggregators**. Este mensaje actúa como una señal de cierre o end, indicando que se han procesado todos los chunks previstos y que ya pueden liberar la data acumulada hacia los siguientes componentes del sistema.

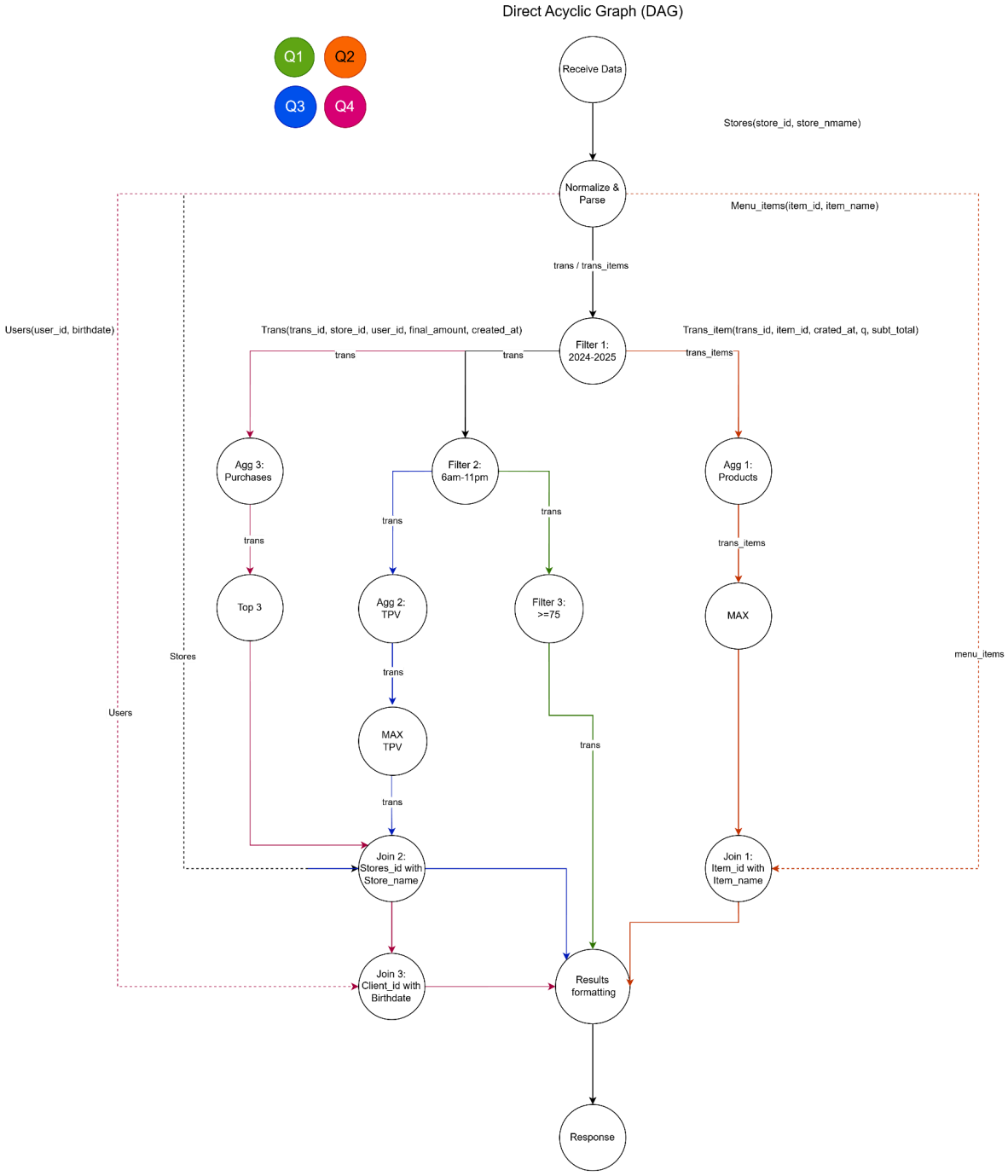
Finalmente, el maximizer espera recibir el end de todos los aggregators para poder completar su propio procesamiento, y el **joiner**, que es un nodo simple, continúa funcionando como antes: recibe toda la data del **maximizer** y detecta el fin del flujo una vez que recibe su señal de end.

Algoritmo de Elección de Líder

Para los monitores, se decidió implementar un algoritmo de elección de líder. El líder es el que hace los trabajos de revivir nodos caídos. Pero si se cae un monitor, tenemos que garantizar que se vuelva a levantar. Para eso, distribuimos el nodo y hacemos que si otro de los monitores detecta la caída del líder, hace un llamado a elección mientras el anterior líder se levanta.

DAG

Los **DAGs (Direct Acyclic Graphs)** representan la ejecución de consultas como un flujo de pasos secuenciales y ramificados. Cada consulta se modela como una cadena de transformaciones de datos: primero se recibe y normaliza la información, luego se aplican filtros, agregaciones o uniones, y finalmente se formatea la respuesta.


Direct Acyclic Graph (DAG)
Q1
Q2
Q3
Q4
Receive Data
Stores(store_id, store_nmame)
Normalize & Parse
Menu_items(item_id, item_name)
trans / trans_items
Users(user_id, birthdate)
Trans(trans_id, store_id, user_id, final_amount, created_at)
Filter 1: 2024-2025
Trans_item(trans_id, item_id, crated_at, q, subt_total)
trans
trans
trans_items
Agg 3: Purchases
Filter 2: 6am-11pm
Agg 1: Products
trans
trans
trans
trans_items
Top 3
Agg 2: TPV
Filter 3: >=75
MAX
Stores
menu_items
trans
Users
MAX TPV
trans
trans
Join 2: Stores_id with Store_name
Join 1: Item_id with Item_name
Join 3: Client_id with Birthdate
Results formatting
Response

Vista de Desarrollo

Paquetes

Diagrama de Paquetes

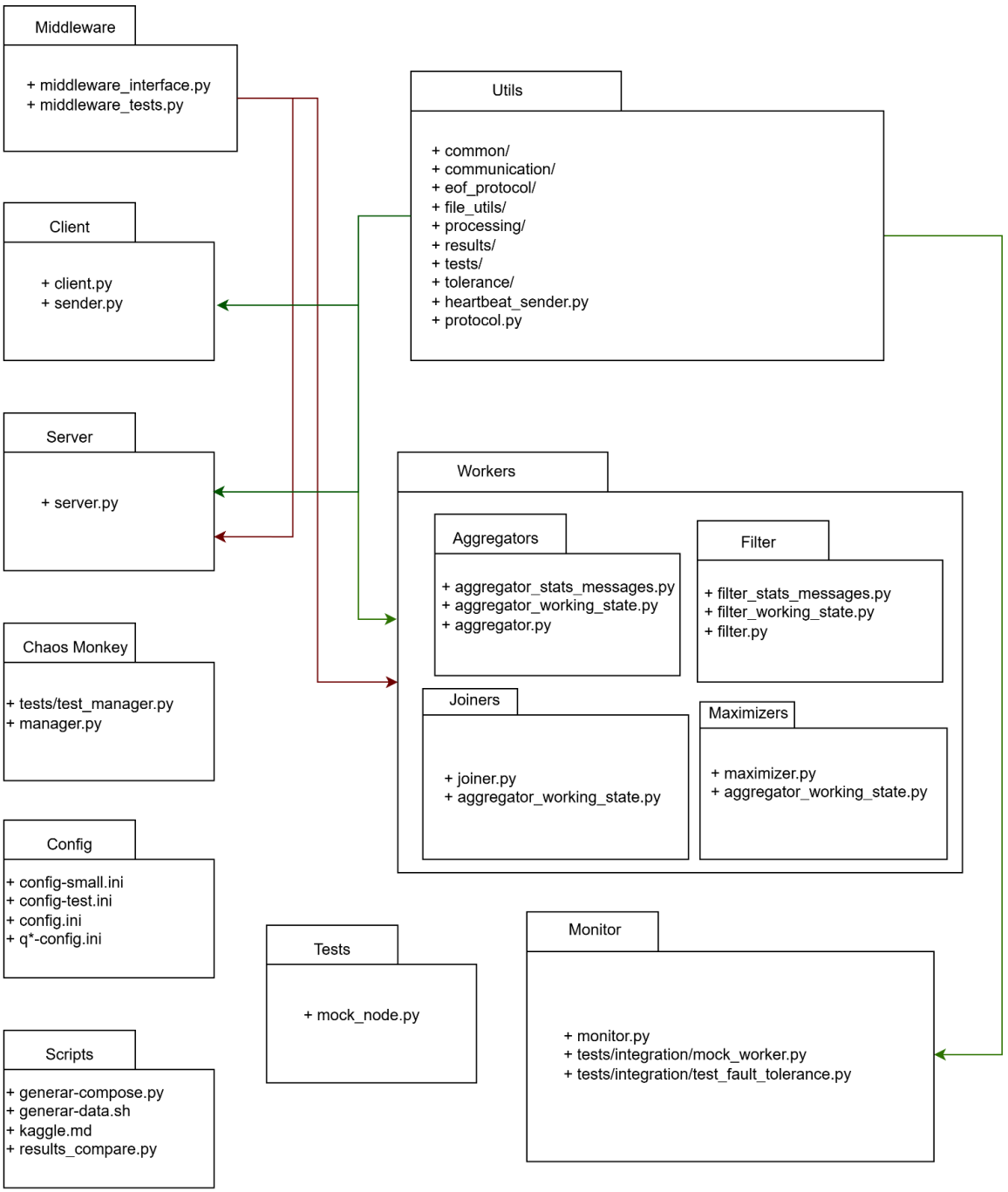
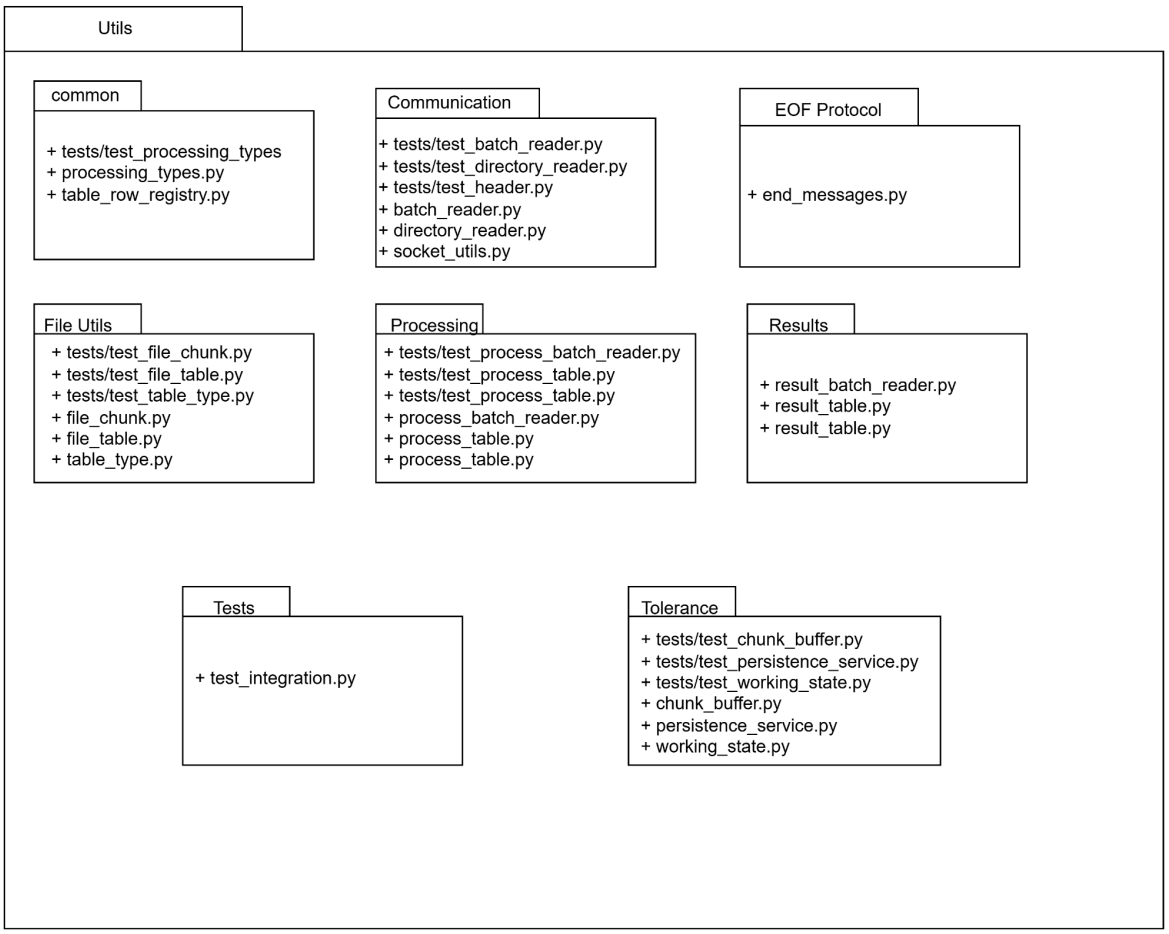


Diagrama de Paquetes - Utils



En la parte superior se encuentra el paquete **Utils**, que contiene todos los submódulos de uso general en todo el proyecto como por ejemplo todo lo que tiene que ver con la tolerancia o la comunicación TCP.

El paquete **Middleware** incluye la lógica intermedia de comunicación, con los archivos middleware_interface.py y middleware_tests.py.

Por otro lado, el paquete **Client** representa la capa que envía los datos para hacer la consulta, mientras que el paquete Server actúa como receptor o procesador principal.

También, tenemos el paquete **Monitor** que representa la capa que monitorea que los diferentes nodos están activos y funcionando.

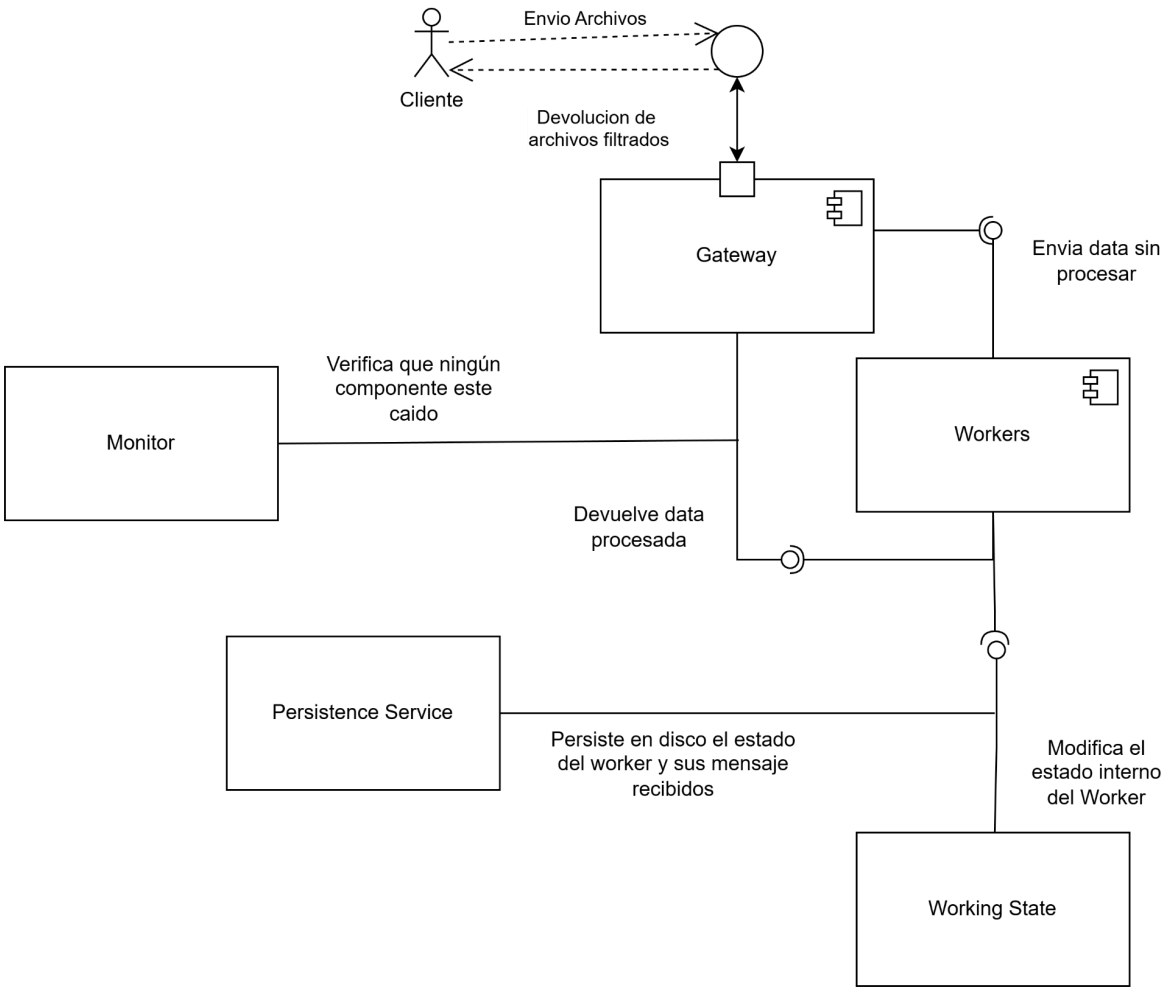
Además, tenemos paquetes más pequeños para testing, y para levantar más rápidamente el proyecto.

Finalmente, el paquete Workers agrupa diferentes tipos de tareas: Aggregators (para consolidar estadísticas), Filter (para filtrar datos innecesarios), Joiners (para unir tablas) y Maximizers (para obtener valores máximos como Top3 o Top1).

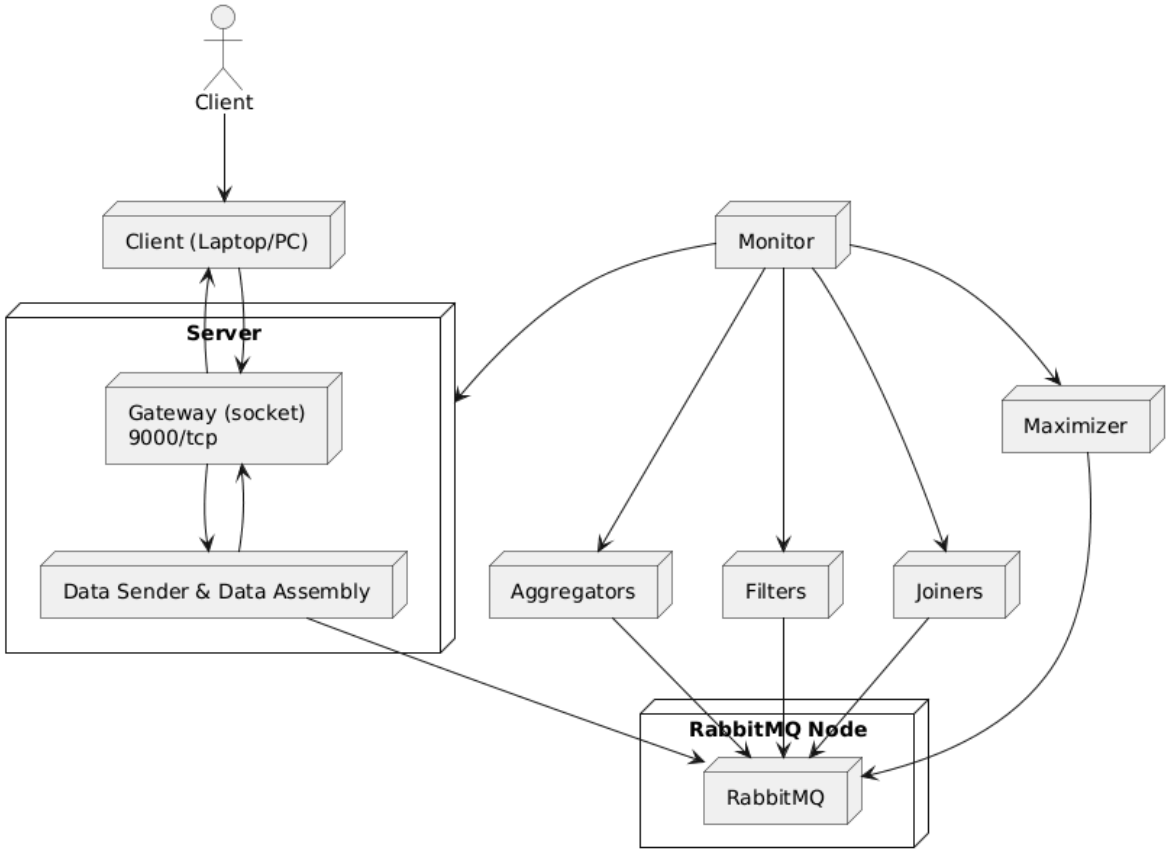
Las flechas entre los paquetes indican dependencias: cómo cada módulo utiliza funciones o clases de otros.

Componentes

Diagrama de Componentes



Vista Física



Arquitectura de Nodos

En primer lugar, el cliente se comunica con el servidor. Este recibe los datos y se encarga de formatearlos y enviarlos hacia el sistema de mensajería.
Los datos procesados se publican luego en un nodo RabbitMQ, que actúa como intermediario y distribuye los mensajes hacia distintos módulos especializados.

Un componente central denominado Monitor verifica y coordina el funcionamiento de los módulos de procesamiento —Aggregators, Filters, Joiners y Maximizers —, que consumen los mensajes desde RabbitMQ y generan nuevas salidas. Esta estructura modular y orientada a eventos permite una alta escalabilidad y separación de responsabilidades, facilitando la extensión o modificación de cada etapa del flujo sin afectar el resto del sistema.

Middleware de Mensajería

El sistema emplea **RabbitMQ** como bus de comunicación asíncrona. Este middleware desacopla la interacción entre los clusters, permitiendo escalabilidad y tolerancia a fallos. Los mensajes enviados incluyen instrucciones de procesamiento y resultados intermedios, lo que asegura una ejecución distribuida eficiente.

Diagrama de Robustez