.UBAfiuba
FACULTAD DE INGENIERÍA

Sistemas Distribuidos 1
(TA050) Roca

# Diseño
# Coffee Shop Analysis

Fecha: 28 de septiembre de 2025

Ascencio Felipe Santino - 110675

Gamberale Luciano Martín - 105892

Zielonka Axel - 110310

# Índice

# 1. Introducción

El presente documento describe el diseño de un sistema distribuido para el análisis de datos de una cadena de cafeterías en Malasia. El objetivo principal es procesar grandes volúmenes de información transaccional, de clientes, de sucursales y de productos, para obtener métricas clave que apoyen la toma de decisiones de negocio.

De acuerdo con los requisitos existentes, el sistema debe permitir responder las siguientes consultas:

1. Listado de transacciones (ID y monto) realizadas entre los años 2024 y 2025, en el rango horario de 06:00 a 23:00, con un monto total mayor o igual a 75.
2. Identificación de los productos más vendidos (nombre y cantidad) y los que más ganancias han generado (nombre y monto) para cada mes de 2024 y 2025.
3. Cálculo del *Total Payment Value* (TPV) por cada semestre en 2024 y 2025, discriminado por sucursal, considerando sólo transacciones entre 06:00 y 23:00.
4. Obtención de la fecha de cumpleaños de los tres clientes con mayor cantidad de compras en 2024 y 2025, para cada sucursal.

Además de los requerimientos funcionales, el sistema deberá cumplir con los siguientes requisitos no funcionales:

- Optimización para entornos multicomputadoras, asegurando la escalabilidad ante el crecimiento de datos.
- Inclusión de un *middleware* que abstraiga la comunicación entre nodos mediante grupos.
- Ejecución única del procesamiento con capacidad de *graceful quit* frente a señales de terminación (SIGTERM).

El diseño se realizará bajo un enfoque de arquitectura distribuida, buscando flexibilidad, robustez y capacidad de escalar en entornos reales de procesamiento de datos.

# 2. Definición Técnica del Sistema Distribuido

El sistema distribuido propuesto permite al usuario interactuar mediante una consola, desde la cual los usuarios envían las instrucciones y datasets necesarios para el procesamiento, y a cambio reciben como salida de la misma el resultado de las consultas realizadas. La infraestructura se implementará sobre contenedores Docker, los cuales emulan múltiples nodos computacionales. Cada nodo se especializa en la ejecución de una operación determinada, tales como filtrado, mapeo, reducción, ordenamiento o combinación de datos.

El sistema recibe como entrada un conjunto de archivos con la información a procesar:

- `menu_items.csv`
- `payment_methods.csv`
- `stores.csv`
- Múltiples archivos `transaction_items.csv`
- Múltiples archivos `transactions.csv`
- Múltiples archivos `users.csv`
- `vouchers.csv`

Estos archivos son distribuidos entre los distintos nodos según lo que requiera cada operación. La comunicación entre el nodo coordinador y los nodos de procesamiento se gestiona mediante un *middleware*, el cual se describe en detalle en una sección posterior.

## 2.1. Operaciones Distribuidas

Las operaciones fundamentales soportadas por el sistema son las siguientes:

1. **Cleaner**: Se encarga de la limpieza de los datos de entrada, eliminando campos que resultan innecesarios para las consultas.
2. **Filter**: Permite seleccionar subconjuntos de datos en función de condiciones específicas. Una misma instancia puede aplicar diferentes filtros según el criterio definido en la consulta (por ejemplo, filtrar por fechas, montos o rangos horarios).
3. **Map**: Transforma los datos de entrada generando nuevas columnas o reformateando la información existente. Ejemplo: Agregar un campo con el mes y año a partir de una fecha, o calcular un contador auxiliar.
4. **Reduce**: Agrupa datos por una clave determinada y aplica una función de agregación (como sumas, conteos o acumulaciones). Existen múltiples variantes, dependiendo de qué métrica se busque consolidar.
5. **Join**: Combina registros de diferentes datasets en función de un campo común, permitiendo enriquecer la información (ejemplo: Unir transacciones con usuarios para obtener la fecha de nacimiento del cliente).
6. **SortBy**: Ordena los registros de acuerdo con uno o más criterios, ya sea de manera ascendente o descendente. Es fundamental para identificar los elementos más significativos en cada consulta (por ejemplo, productos más vendidos).
7. **OutputBuilder**: Es el último paso en la cadena de operaciones, se encarga de formatear la respuesta que va a ser enviada al usuario.

Cada nodo de operación está diseñado para ser genérico y flexible: Puede ejecutar cualquier variante de las funciones mencionadas según el rol específico del nodo, siendo el sistema coordinador quien le envía los datos y la configuración necesaria para que realice la tarea solicitada de la mejor forma posible.

# 3. Implementación del Usuario

La interacción de los usuarios con el sistema distribuido se realizará mediante un esquema secuensial. Para cada usuario que se conecte, se respetera una secuencia específica de ejecución.

- **Etapa de envío de información:** En esta etapa el usuario envía toda la información en base a la que el sistema debe operar. Esto incluye todos los datasets a procesar para resolver las 4 consultas.
- **Etapa de espera:** El cliente queda a la espera de que el sistema distribuido procese la información y genere las respuestas a las consultas.
- **Etapa de recepción de resultados:** Una vez que el sistema ha procesado la información, el usuario recibe las respuestas a las consultas solicitadas.

De esta manera se logra una interacción segura y escalable.

La arquitectura está diseñada para soportar múltiples usuarios en simultáneo, escalando el número de conexiones establecidas proporcionalmente a la cantidad de clientes activos.

# 4. Mecanismo de Optimización del Uso de Recursos

Cada computadora que participa como nodo de procesamiento en el sistema distribuido estará dedicada a ejecutar una operación determinada (por ejemplo, filtrado o reducción).

Si bien en una primera versión se teorizó la posibilidad de generar varios procesos por 'CPU' para buscar algún tipo de optimización del rendimiento de cada computadora, luego de la primera entrega con el corrector se definió que no resultaría necesario para la implementación de la resolución.

Por lo tanto, se descarta esta mejora para esta versión del sistema, pero se deja como recomendación por si se realiza una actualización a futuro en búsqueda de optimizar tiempos de procesamiento de consultas.

De ser así, se debe recordar que al implementar la paralelización, se debe hacer mediante **multiprocesos** y no mediante **multithreading**. La justificación de esta elección radica en que Python presenta limitaciones para tareas de cómputo intensivo debido al *Global Interpreter Lock* (GIL).

La librería de `multiprocessing` permite crear procesos independientes que aprovechan mejor las capacidades de CPUs con múltiples núcleos.

# 5. Uso de Archivos Intermedios

En el diseño del sistema se contempla la generación de archivos intermedios durante el procesamiento de las consultas. Esta decisión se fundamenta en un aspectos clave:

**Respaldo de información:** Los archivos intermedios actúan como puntos de control que facilitan futuras implementaciones de tolerancia a fallas. En caso de interrupciones, será posible retomar el procesamiento desde un estado parcial previamente almacenado.

El uso de archivos intermedios, si bien introduce un costo adicional de I/O, aumenta la robustez del sistema y habilita mejoras posteriores en términos de recuperación y confiabilidad.

Para esta entrega del sistema todavía no están implementados de manera funcional, pero si se deja esta consideración para luego documentar su funcionamiento cuando corresponda.

# 6. Descripción de las Consultas

El sistema debe ser capaz de responder a las siguientes consultas, de acuerdo con la lógica planteada en el enunciado. A continuación, se detalla el paso a paso de cada una de ellas, indicando las tablas necesarias, las columnas relevantes y la secuencia de operaciones distribuidas.

## 6.1. Consulta 1: Transacciones filtradas

**Objetivo:** Obtener las transacciones (ID y monto) realizadas durante 2024 y 2025, entre las 06:00 y las 23:00 horas, con un monto total mayor o igual a 75.

- **Tablas requeridas:** `transactions`.
- **Columnas utilizadas:** `transaction_id`, `created_at`, `final_amount`.
- **Pasos:**
  1. Filtrar transacciones entre 2024 y 2025, en el rango horario indicado.
  2. Filtrar transacciones con monto mayor o igual a 75.
  3. Generar el reporte y devolverlo al usuario.

## 6.2. Consulta 2: Productos más vendidos y más rentables

**Objetivo:** Identificar, para cada mes de 2024 y 2025, los productos más vendidos (nombre y cantidad) y los productos que más ganancias generaron (nombre y monto).

- **Tablas requeridas:** `transaction_items`, `menu_items`.
- **Columnas utilizadas:**
  - De `transaction_items`: `item_id`, `created_at`, `quantity`, `subtotal`.
  - De `menu_items`: `item_id`, `item_name`.
- **Pasos:**
  1. Filtrar los `transaction_items` de 2024 y 2025.
  2. Generar una nueva columna `year_month` a partir de la fecha.
  3. Para productos más vendidos:
     *a*) Calcular contador de cantidad (`item_counter`).
     *b*) Reducir por clave `year_month, item_id` sumando las cantidades.
     *c*) Ordenar por cantidad descendente.
  4. Para productos más rentables:
     *a*) Calcular monto acumulado (`total_amount_counter`).
     *b*) Reducir por clave `year_month, item_id` sumando subtotales.
     *c*) Ordenar por monto descendente.
  5. Unir con `menu_items` para obtener nombres de los productos.
  6. Generar el reporte y devolverlo al usuario.

### 6.3. Consulta 3: TPV por semestre y sucursal

**Objetivo:** Calcular el *Total Payment Value* (TPV) por cada semestre de 2024 y 2025, para cada sucursal, considerando únicamente transacciones realizadas entre 06:00 y 23:00.

- **Tablas requeridas:** `transactions` y `stores`.
- **Columnas utilizadas:** `store_id`, `created_at`, `final_amount`.
- **Pasos:**
  1. Filtrar transacciones entre 2024 y 2025, dentro del rango horario.
  2. Generar nueva columna `year_semester` a partir de la fecha.
  3. Reducir por clave `year_semester, store_id` sumando los montos finales.
  4. Unir con `stores` para obtener nombres de las sucursales.
  5. Generar el reporte y devolverlo al usuario.

### 6.4. Consulta 4: Clientes más frecuentes y cumpleaños

**Objetivo:** Obtener la fecha de cumpleaños de los tres clientes con mayor número de compras durante 2024 y 2025, discriminado por sucursal.

- **Tablas requeridas:** `transactions`, `users` y `stores`.
- **Columnas utilizadas:**
  - De `transactions`: `user_id`, `store_id`, `created_at`.
  - De `users`: `user_id`, `birthdate`.
- **Pasos:**
  1. Filtrar transacciones de 2024 y 2025.
  2. Generar clave combinada `store_user` con `store_id` y `user_id`.
  3. Calcular contador de compras (`buys_counter`).
  4. Reducir por clave `store_user` sumando los contadores.
  5. Ordenar en forma descendente por número de compras.
  6. Seleccionar los tres usuarios con más compras por sucursal.
  7. Unir con `users` para obtener las fechas de nacimiento.
  8. Unir con `stores` para obtener nombres de las sucursales.
  9. Generar el reporte y devolverlo al usuario.

# 7. Vista de Escenarios

## 7.1. Casos de uso

Para modelar la interacción principal entre los actores y el sistema, se presenta un diagrama de casos de uso. Este diagrama identifica al actor principal/client, denominado Çafetería", y las cuatro funcionalidades clave que el sistema debe proveer, correspondiendo directamente a las consultas de negocio definidas en los requerimientos.

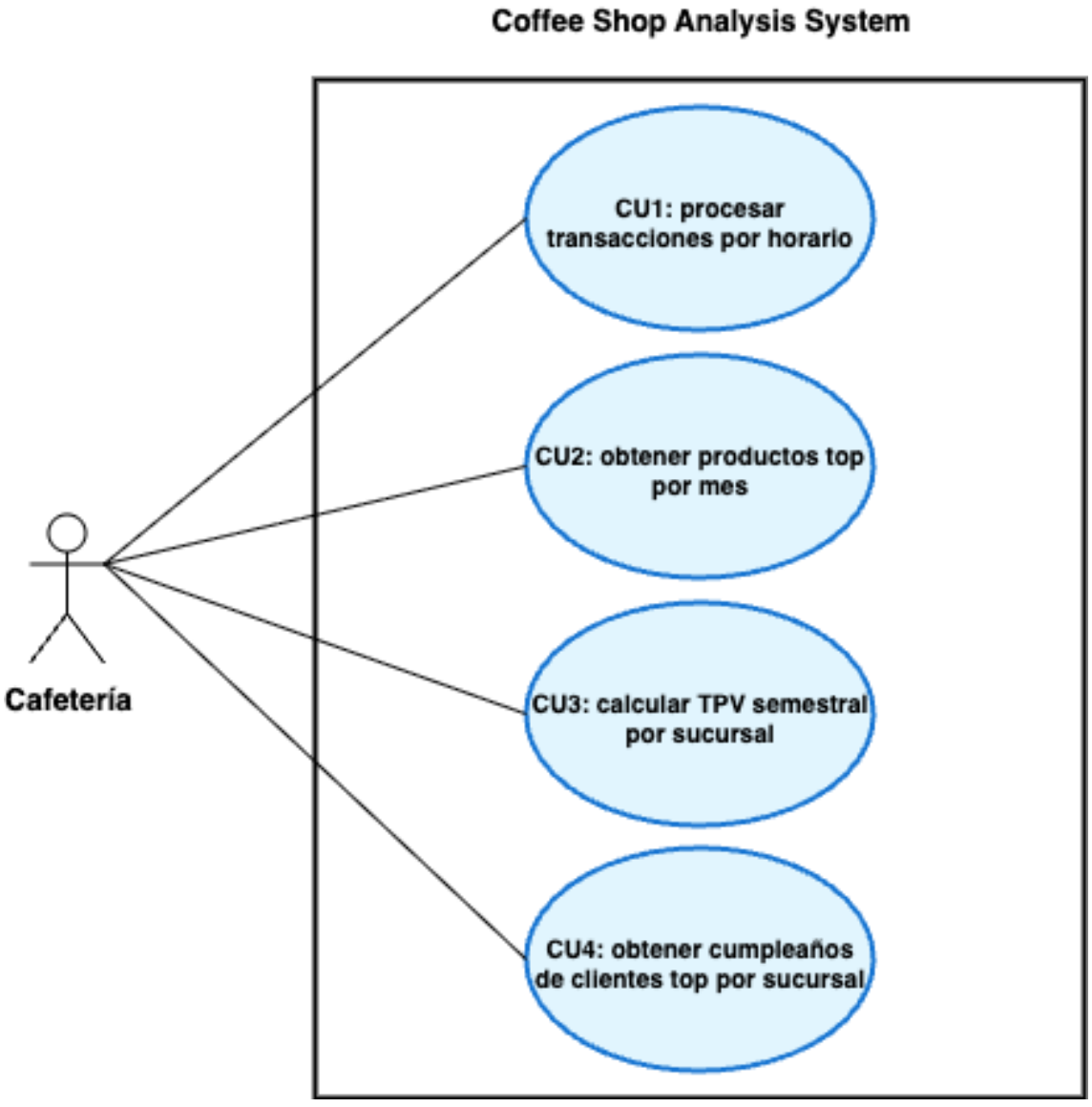


Figura 1: Casos de Uso

# 8. Vista de Procesos

## 8.1. Diagramas de Actividades

Los diagramas de actividad describen el flujo de trabajo lógico para cada una de las consultas funcionales. A continuación, se presenta un diagrama para cada consulta, detallando la secuencia de acciones y las decisiones desde el inicio de la solicitud hasta la presentación de los resultados finales.

### 8.1.1. Diagrama de Actividades de la primera consulta

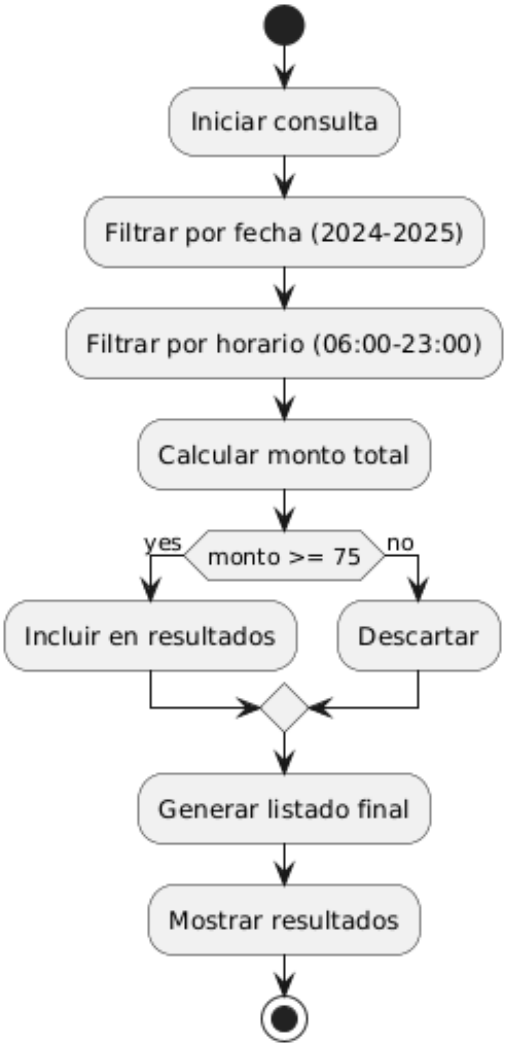


Figura 2: Actividad de la primera consulta

### 8.1.2. Diagrama de Actividades de la segunda consulta

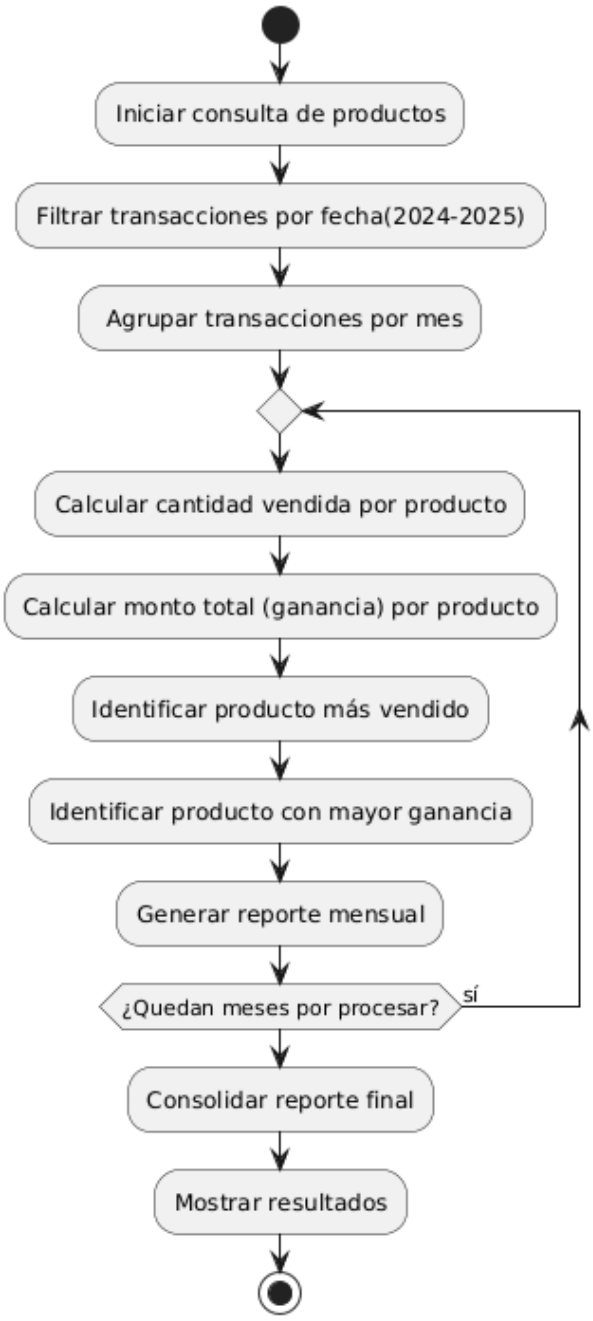


Figura 3: Actividad de la segunda consulta

### 8.1.3. Diagrama de Actividades de la tercera consulta

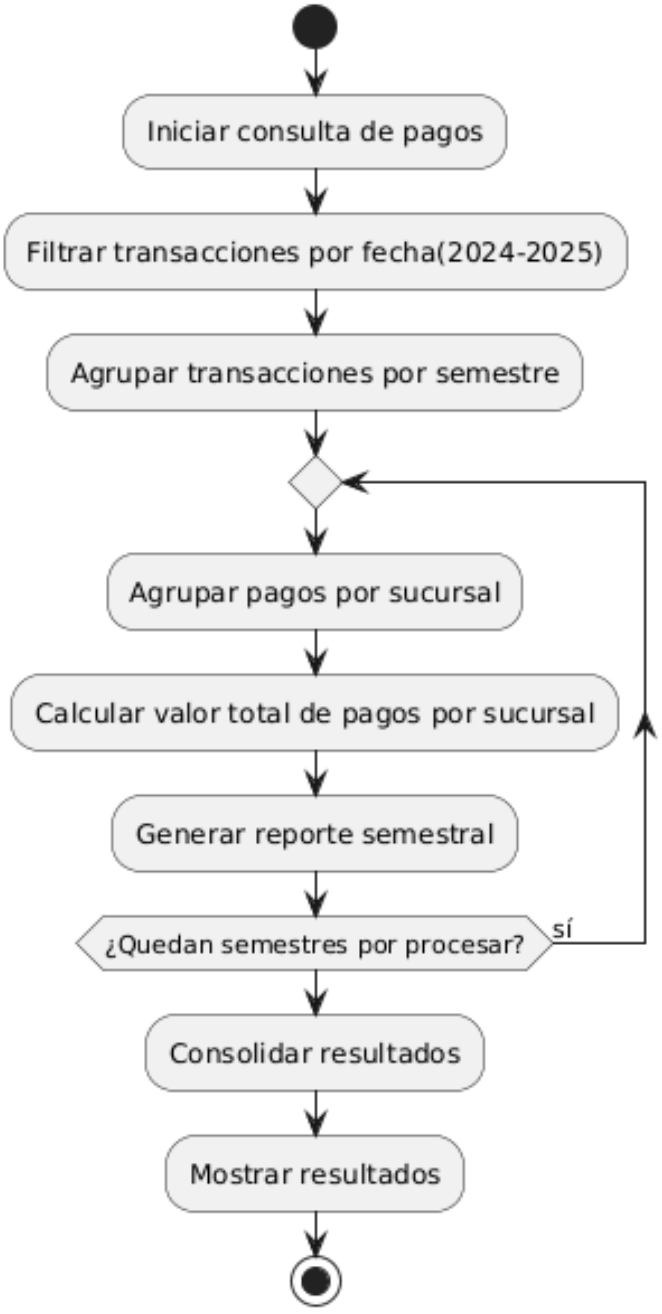


Figura 4: Actividad de la tercera consulta

### 8.1.4. Diagrama de Actividades de la cuarta consulta

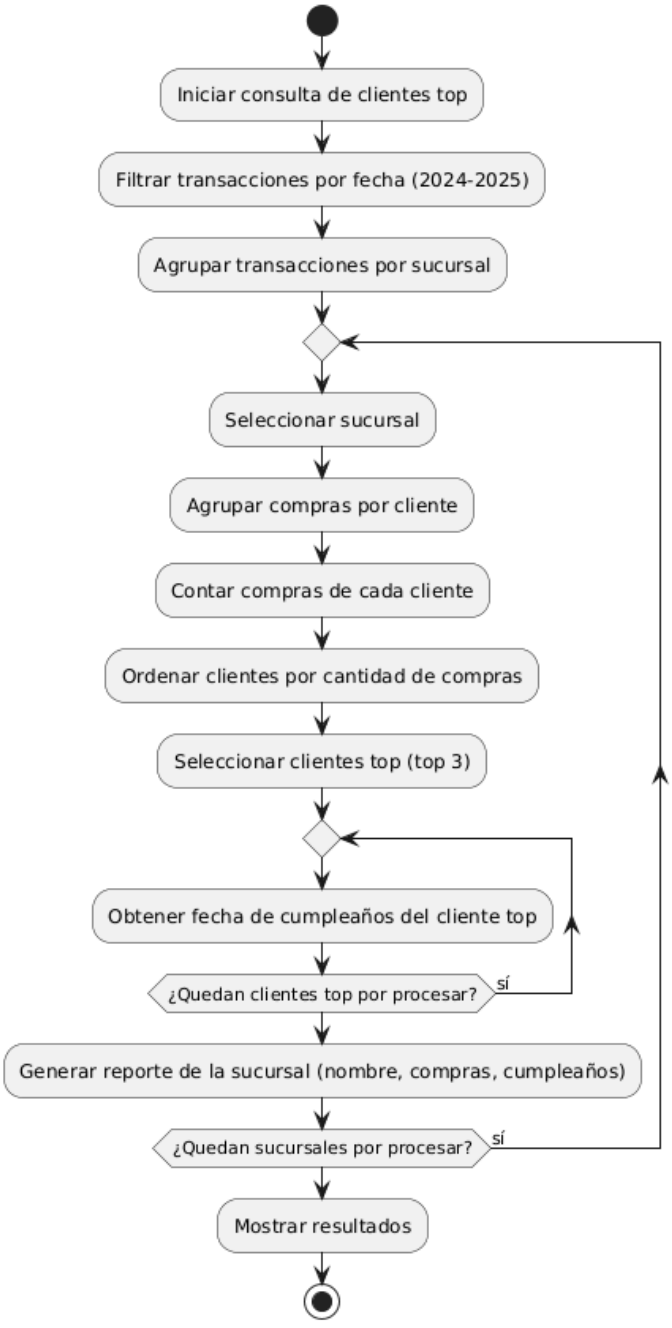


Figura 5: Actividad de la cuarta consulta

## 8.2. Diagramas de Secuencia

Para ilustrar la interacción temporal y el intercambio de mensajes entre los distintos componentes del sistema, se utilizan los diagramas de secuencia. Cada diagrama modela la ejecución de una consulta, mostrando cómo el Client Process envía lotes de datos al Server, el cual coordina las operaciones distribuidas entre los nodos especializados como Filter, GroupBy y ReduceBy hasta obtener y devolver el reporte final.

### 8.2.1. Diagrama de Secuencia de la primera consulta

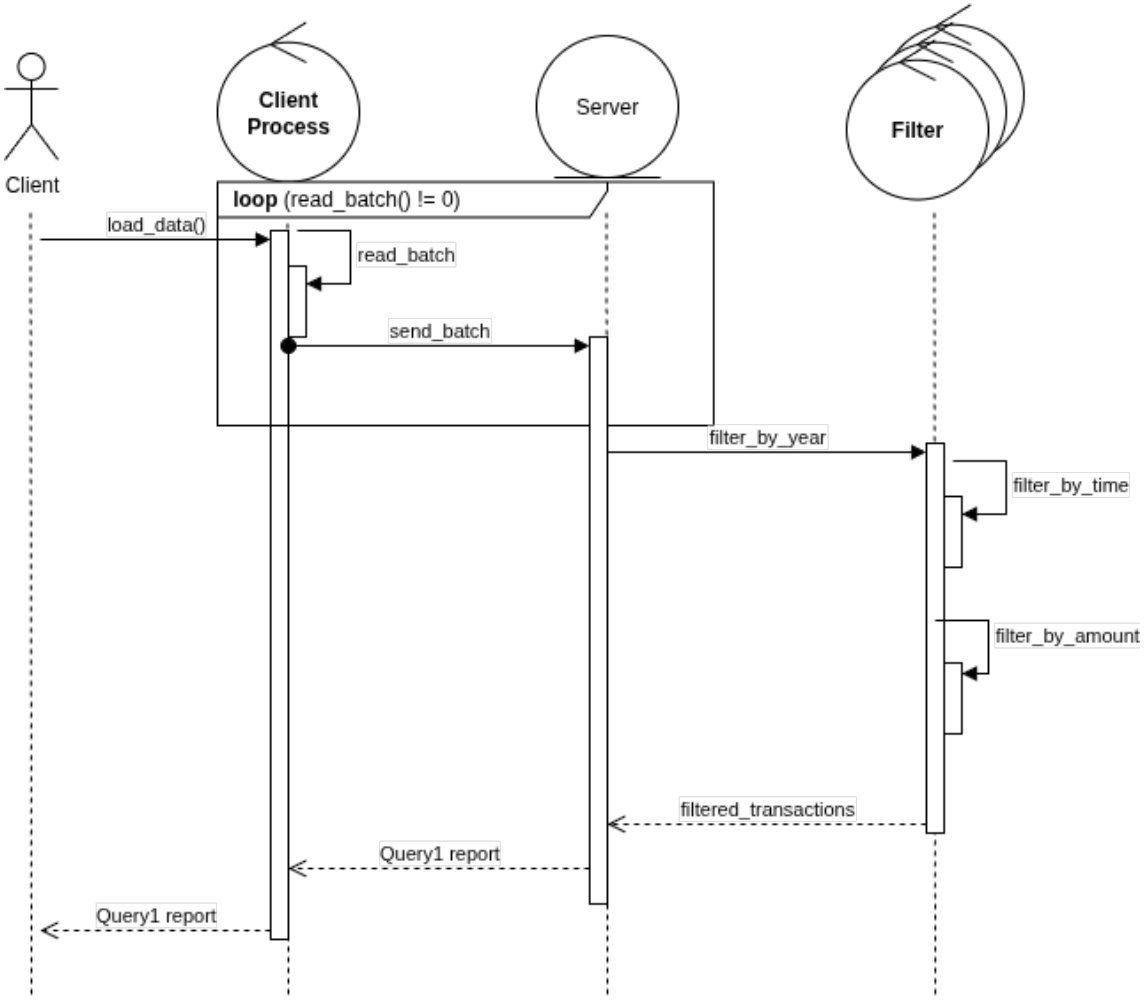


Figura 6: Secuencia de la primera consulta

### 8.2.2. Diagrama de Secuencia de la segunda consulta

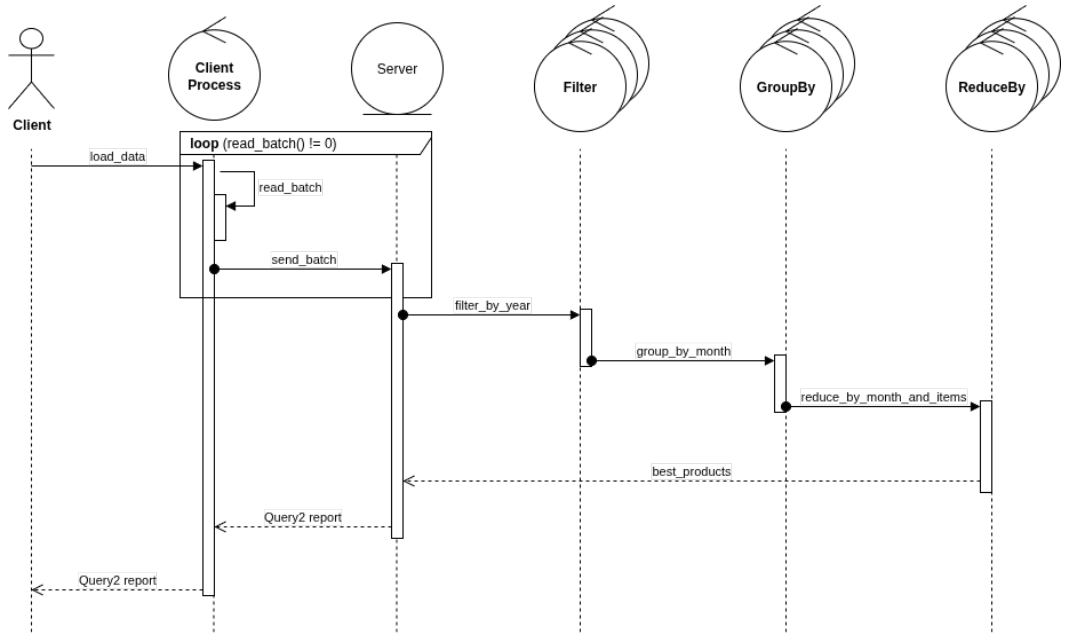


Figura 7: Secuencia de la segunda consulta

### 8.2.3. Diagrama de Secuencia de la tercera consulta

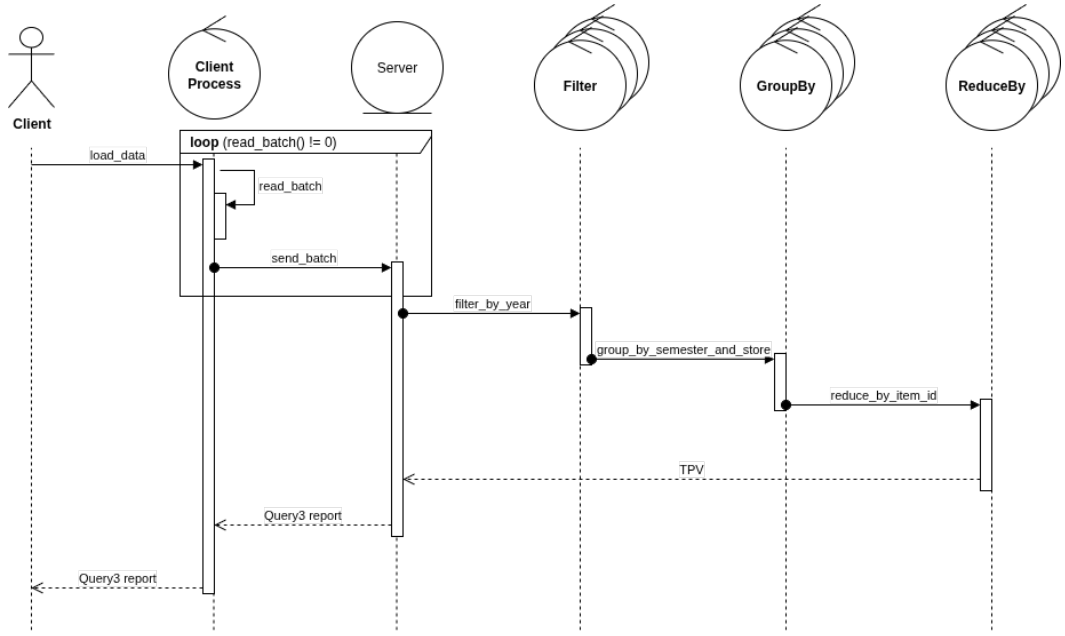


Figura 8: Secuencia de la tercera consulta

### 8.2.4. Diagrama de Secuencia de la cuarta consulta

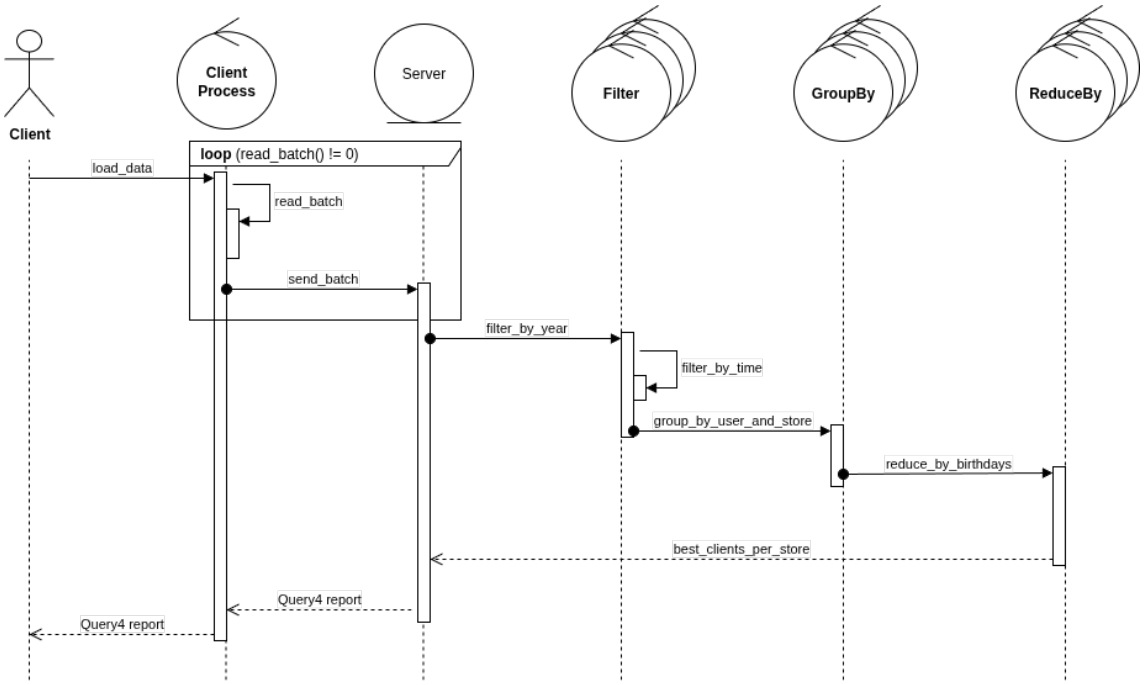


Figura 9: Secuencia de la cuarta consulta

# 9. Vista Física

## 9.1. Diagrama de Robustez

El diagrama de robustez forma parte de la vista lógica y sirve como puente entre los casos de uso y el diseño detallado. Este diagrama permite identificar y organizar los principales elementos del sistema, diferenciando entre actores externos, límites de la aplicación y las entidades o controladores que gestionan la lógica de negocio.

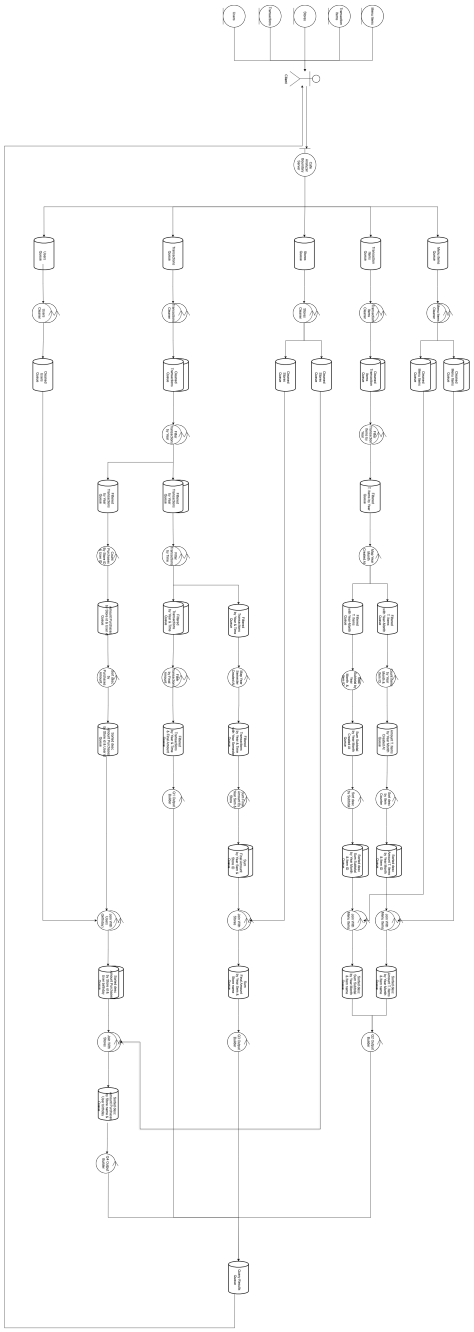

Figura 10: Diagrama de Robustez - Completo

## 9.2. Diagrama de Robustez - Primera Consulta

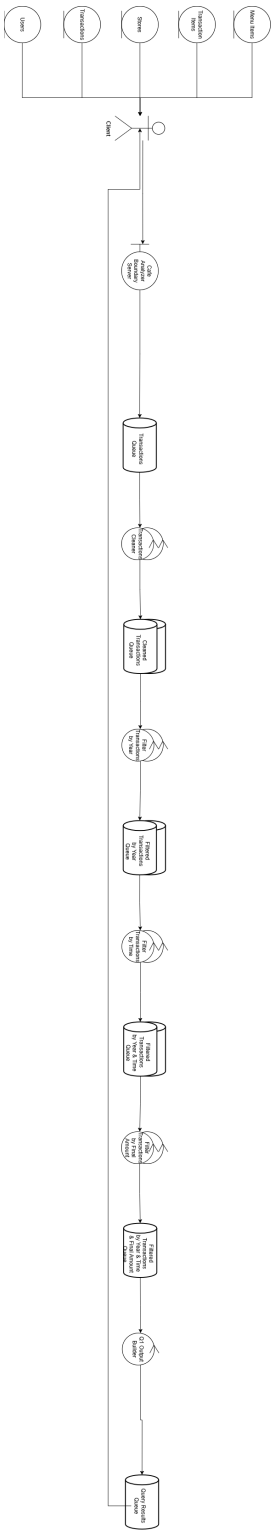


Figura 11: Diagrama de Robustez - Primera Consulta

## 9.3. Diagrama de Robustez - Segunda Consulta

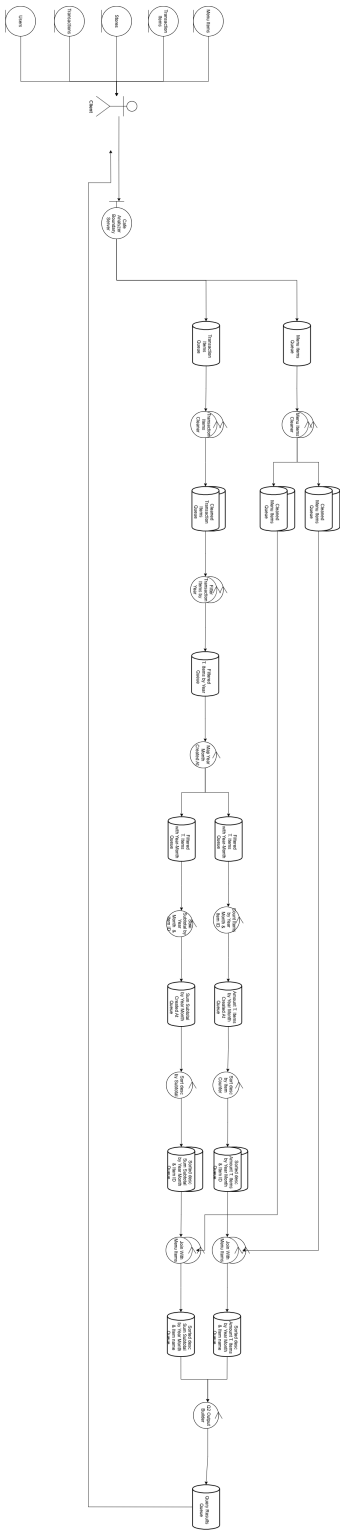

Figura 12: Diagrama de Robustez - Segunda Consulta

## 9.4. Diagrama de Robustez - Tercera Consulta

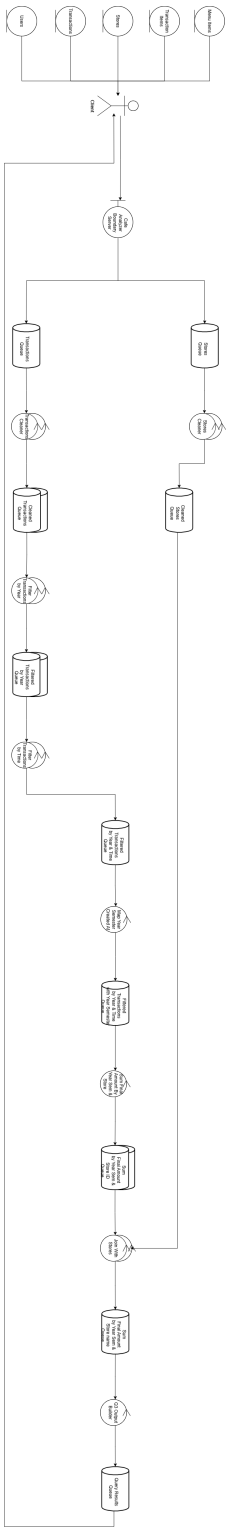

Figura 13: Diagrama de Robustez - Tercera Consulta

## 9.5. Diagrama de Robustez - Cuarta Consulta

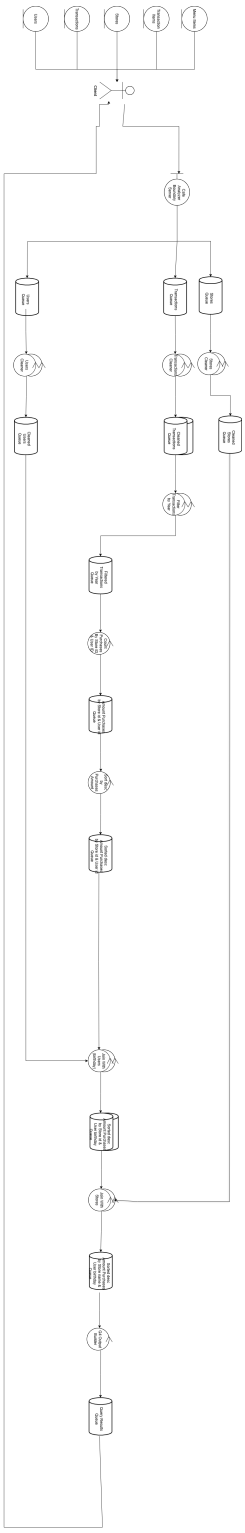

Figura 14: Diagrama de Robustez - Cuarta Consulta

## 9.6. Escalabilidad de las Operaciones

Dentro de las operaciones del sistema distribuido, existen tres que admiten escalabilidad en la cantidad de nodos de procesamiento:

- **Cleaner**
- **Filter**
- **Join**

Como decisión de diseño, se definió que en los casos donde el sistema cuente con una disposición de $N$ nodos de alguno de los tres tipos mencionados, comunicándose con $M$ nodos de la misma categoría, cada uno de los $M$ nodos receptores tendrá su propia cola. Dichas colas recibirán los mensajes enviados por los $N$ nodos anteriores. Cuando un nodo de la segunda capa reciba en su cola un mensaje indicador de finalización, podrá tener la certeza de que no recibirá más información desde esos nodos, y podrá trasladar dicho aviso hacia etapas posteriores en caso de ser necesario.

Para garantizar este comportamiento, al iniciar la ejecución del sistema todos los nodos conocerán la topología de sus vecinos, lo que permitirá coordinar de forma correcta la comunicación y el flujo de datos durante el procesamiento.

A continuación, se presenta un esquema ilustrativo de la topología entre $N$ nodos *Filter* y $M$ nodos *Join*, junto con sus respectivas $M$ colas:

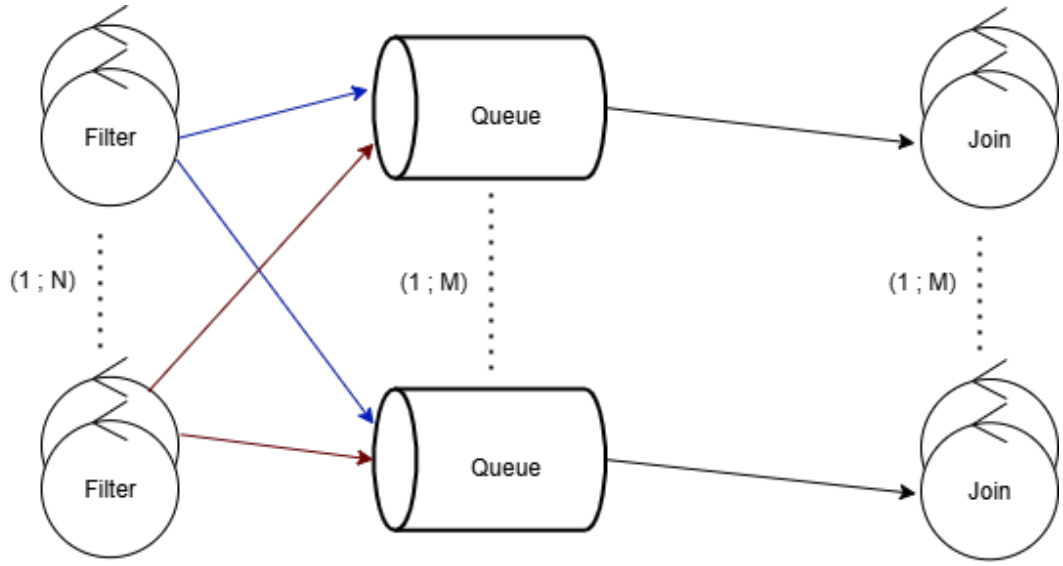


Figura 15: Esquema representativo de la comunicación de 'N' nodos con 'M' colas de otros 'M' nodos

Asimismo, se adoptó la decisión de diseño de no escalar con múltiples computadoras los *Join* que se realizan con las tablas de *Menú* y de *Stores*, ya que ambas son lo suficientemente pequeñas como para cargarse íntegramente en memoria. De esta forma, se mantiene la tabla más pequeña residente en memoria y se la utiliza para realizar *joins* eficientes a medida que llegan las transacciones, maximizando así la performance.

Tampoco se escalaran con múltiples computadoras las operaciones 'Map', 'Reduce', 'Sort' y 'Output Builder'. Ya que son 'Stateful' y su correcto funcionamiento e implementación se ve muy beneficiado de ejecutarse en una sola instancia de computación, sin perder un rendimiento perceptible.

En contraste, para el caso de los *Join* con la tabla de *Usuarios* sí se habilita la escalabilidad a múltiples nodos, dado que el volumen de datos puede ser significativamente mayor. En este escenario, el listado de usuarios se dividirá entre los $M$ nodos mediante una función de *hash*. Los nodos que depositen la información de transacciones en las colas aplicarán la misma función de *hash* sobre los identificadores de usuario, determinando de manera directa a qué cola (y por ende a qué nodo) deben dirigir cada registro para asegurar que el *join* se realice correctamente.

## 9.7. Diagrama de Despliegue

Dentro de la vista física realizamos el diagrama de despliegue, el cual ilustra la topología del sistema distribuido en términos de sus nodos computacionales. El diagrama muestra los distintos grupos de nodos especializados (Data Cleaners, Filters, Maps, Joins, etc.) y cómo se interconectan a través de un componente central: el MOM Broker. Esta arquitectura facilita la comunicación asincrónica y el desacoplamiento entre los procesos.

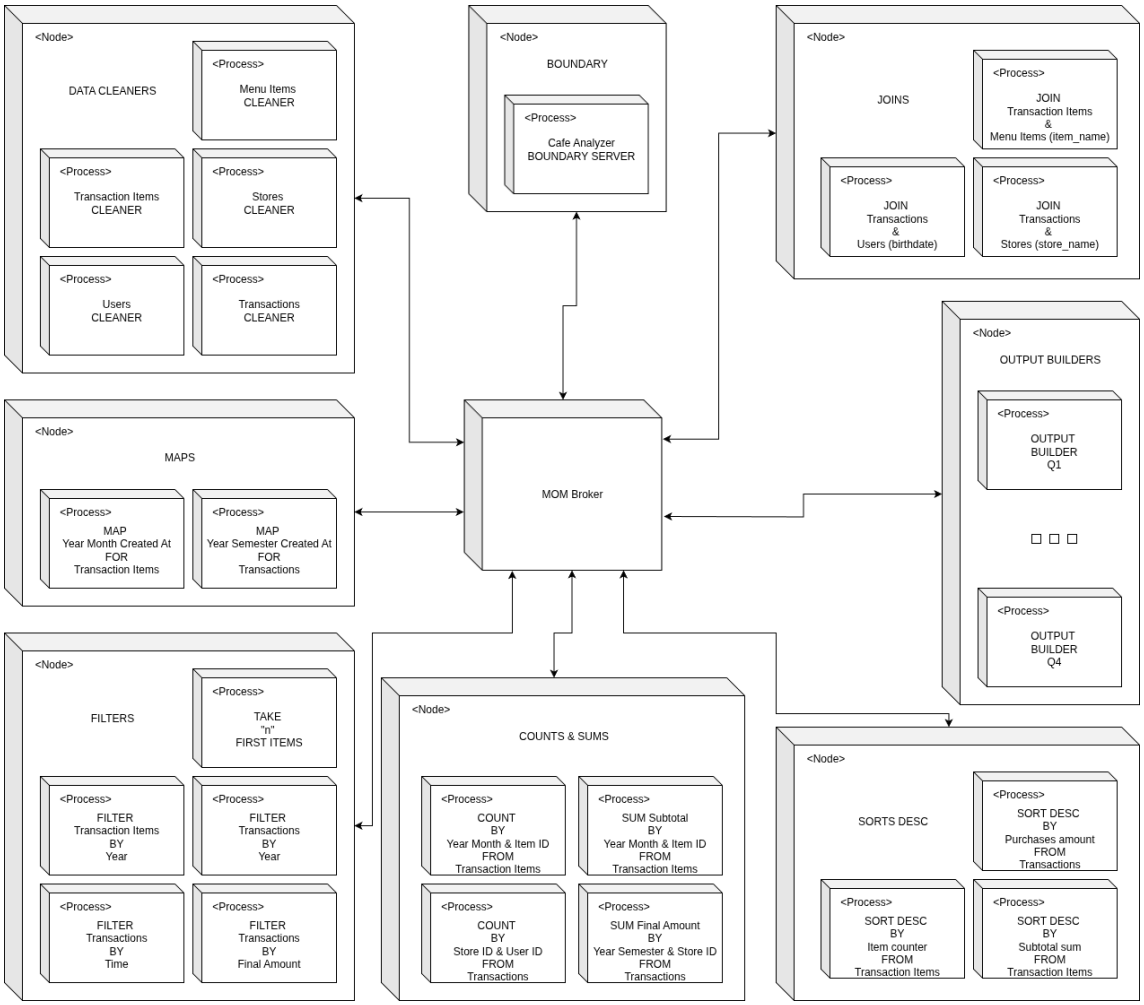


Figura 16: Diagrama de Despliegue

# 10. Vista de Desarrollo

## 10.1. Diagrama de Paquetes

La vista de desarrollo organiza el sistema en un conjunto de paquetes lógicos para promover una alta cohesión y un bajo acoplamiento. El diagrama muestra los componentes clave: el paquete shared contiene la lógica de comunicación común, como el MQConnectionHandler; el paquete controllers agrupa las distintas operaciones distribuidas (filtros, mapeos, etc.); y los paquetes client y server definen los puntos de entrada de la aplicación.

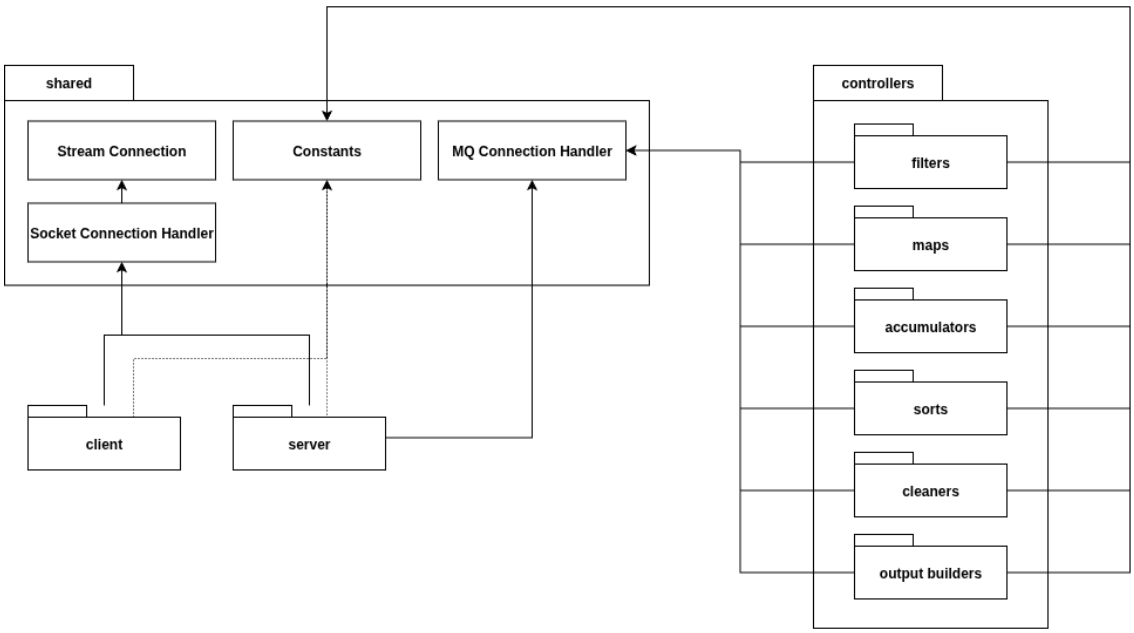


Figura 17: Vista de Desarrollo - Diagrama de Paquetes

# 11. Vista Lógica

## 11.1. Diagrama de Clases

La vista lógica detalla la estructura estática del sistema a través de diagramas de clases. El diseño se basa en la herencia y la abstracción para maximizar la reutilización de código. Como se observa en los siguientes diagramas, se define una clase base abstracta Controller de la cual heredan las distintas operaciones especializadas (Filter, Count, Sum, etc.), permitiendo que el sistema maneje diferentes tipos de tareas de manera uniforme y extensible.

### 11.1.1. Diagrama de Clases - DataCleaner

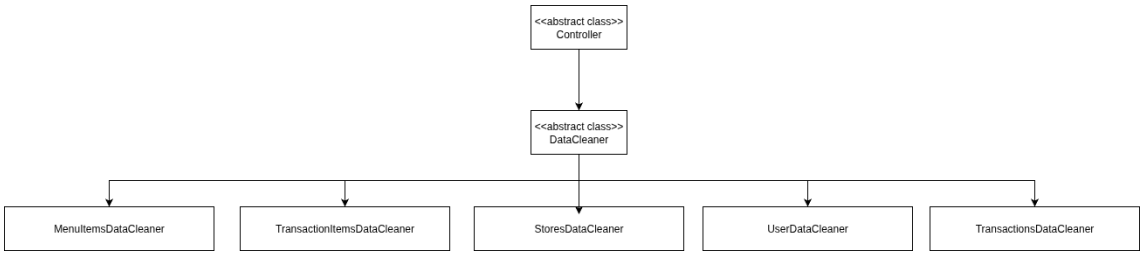


Figura 18: Clase DataCleaner

### 11.1.2. Diagrama de Clases - Filter

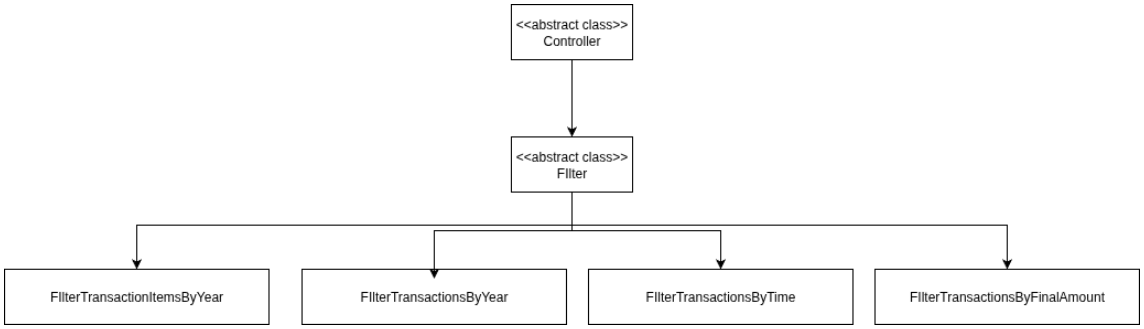


Figura 19: Clase Filter

### 11.1.3. Diagrama de Clases - Count y Sum

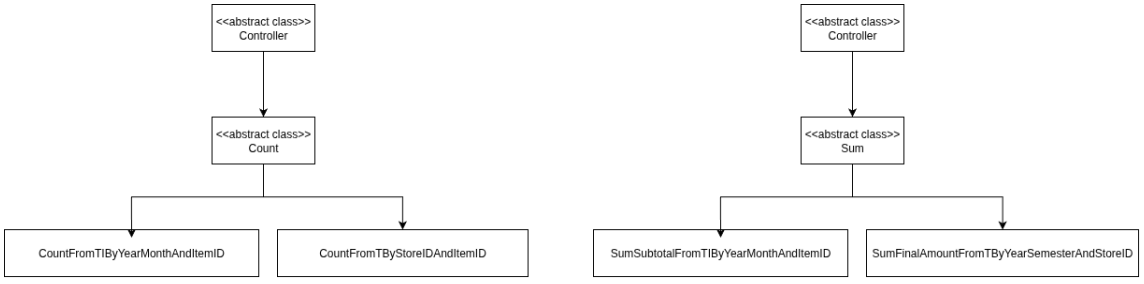


Figura 20: Clases Count y Sum

### 11.1.4. Diagrama de Clases - SortDesc

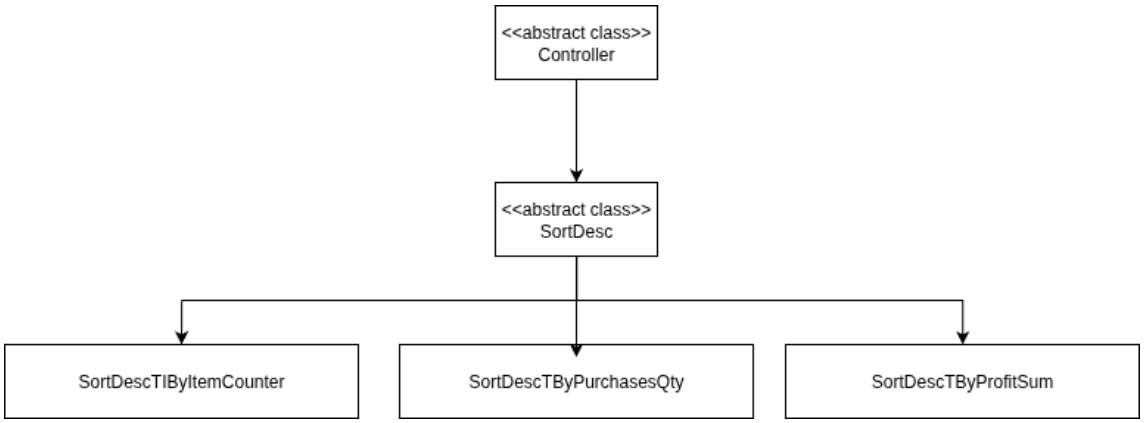


Figura 21: Clase SortDesc

### 11.1.5. Diagrama de Clases - Join

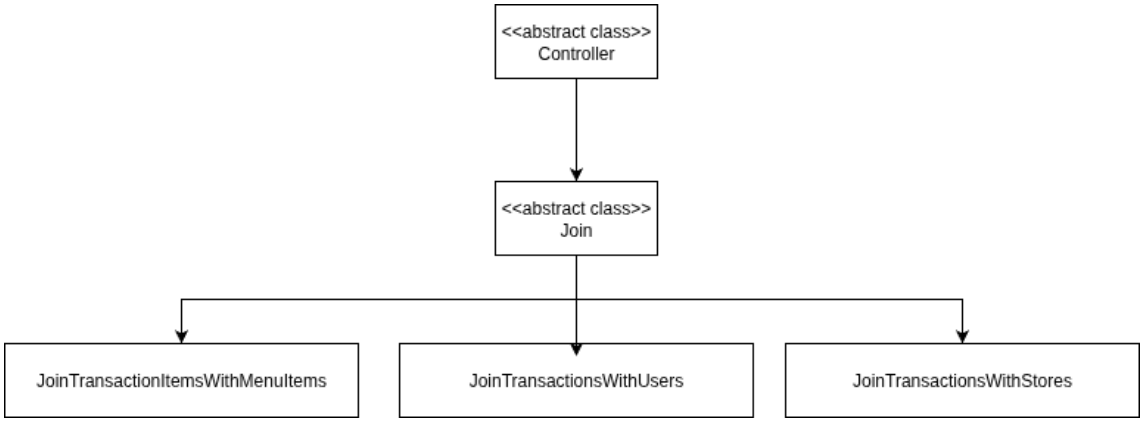


Figura 22: Clase Join

### 11.1.6. Diagrama de Clases - OutputBuilder

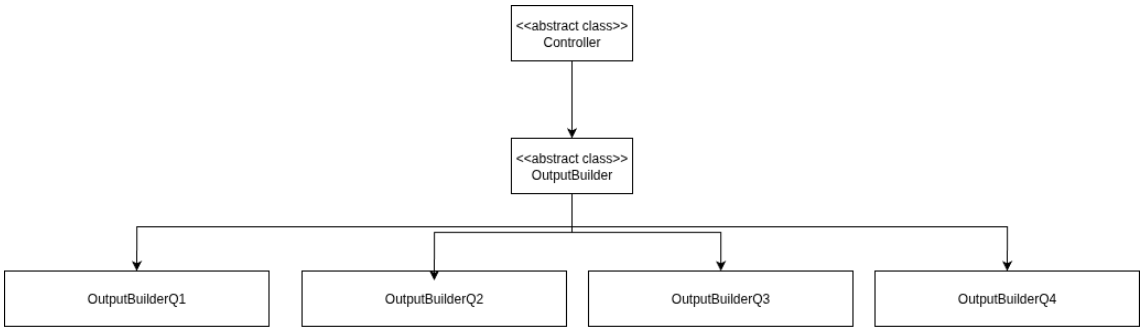


Figura 23: Clase OutputBuilder

# 12. Diagrama de Grafo Acíclico Dirigido (DAG) - Completo

El flujo de procesamiento completo para resolver las consultas se modela como un Grafo Acíclico Dirigido (DAG). Este diagrama visualiza las dependencias entre las distintas operaciones distribuidas. Cada nodo en el grafo representa una tarea (ej. Clean, Filter by Year, Count Items), y las aristas indican el flujo de datos desde una etapa hacia la siguiente, mostrando cómo se combinan las distintas fuentes de datos para producir los resultados finales (Q1, Q2, Q3, Q4).

## 12.1. Diagrama de Grafo Acíclico Dirigido (DAG) - Completo

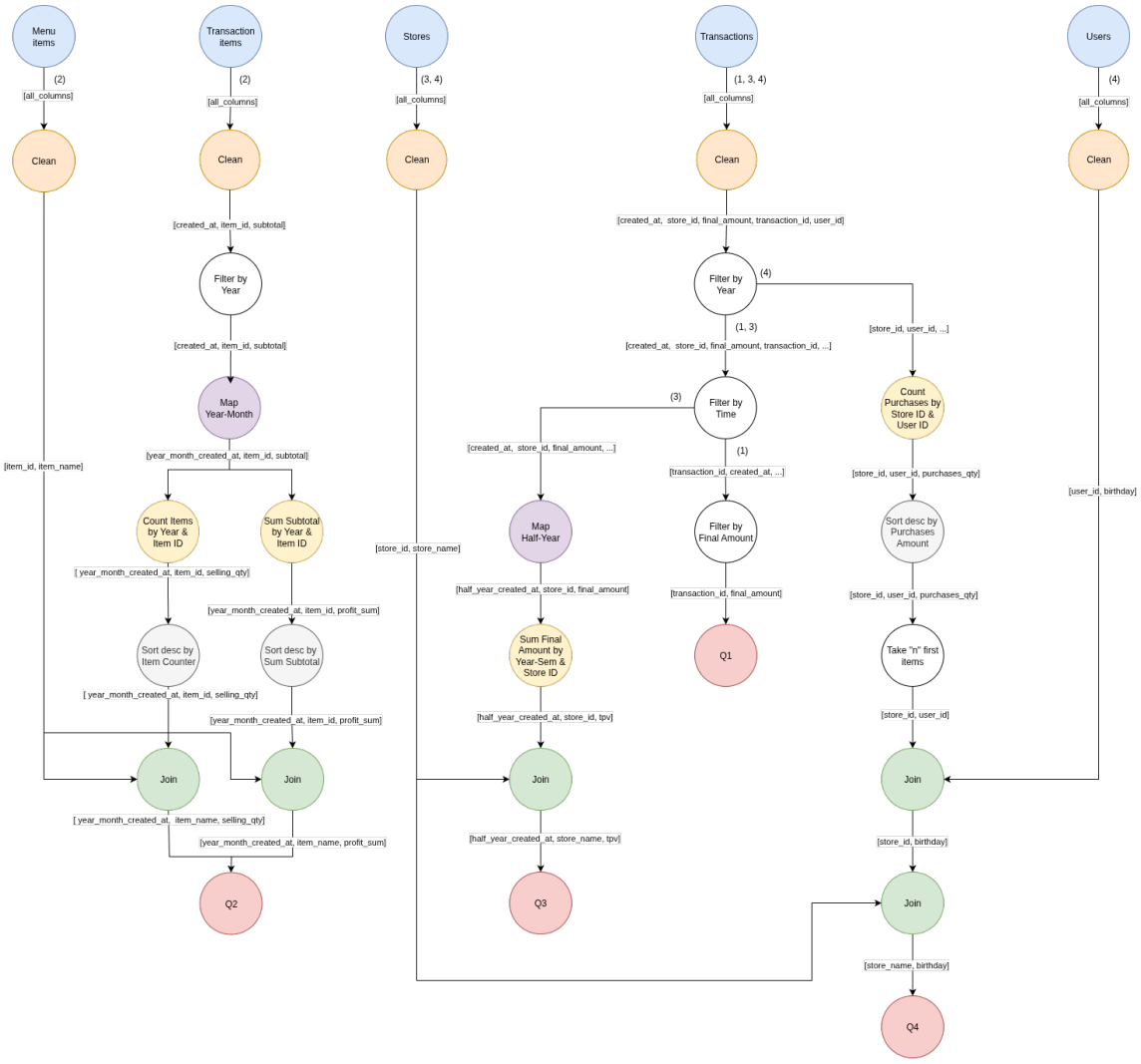


Figura 24: Diagrama de Grafo Acíclico Dirigido (DAG) - Completo

## 12.2. Diagrama de Grafo Acíclico Dirigido (DAG) - Primera consulta

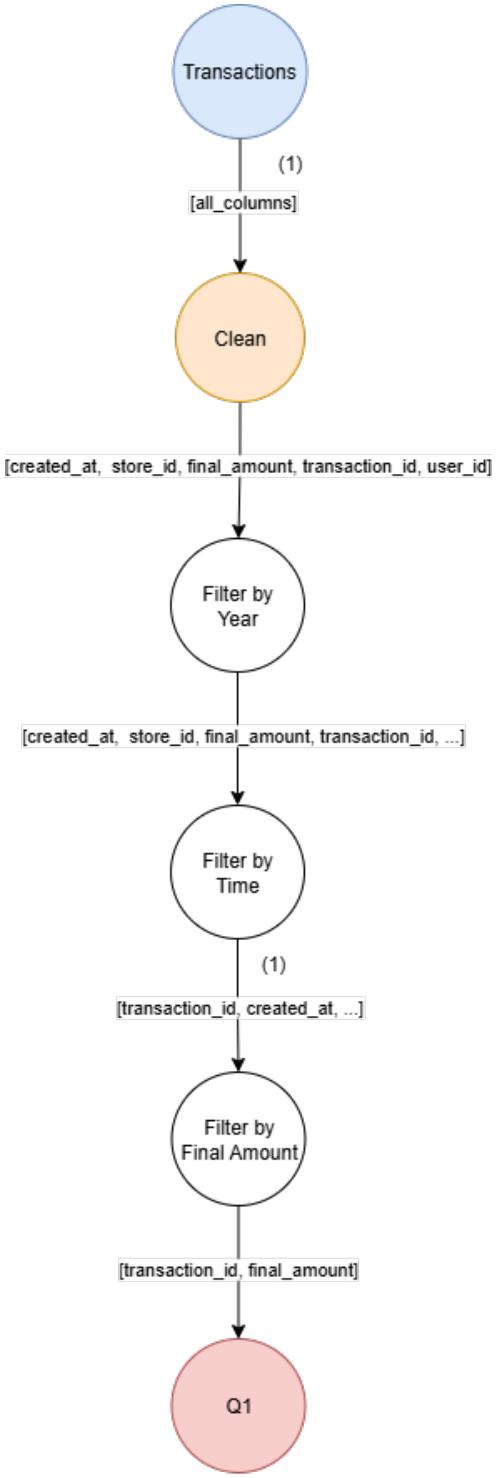


Figura 25: Diagrama de Grafo Acíclico Dirigido (DAG) - Primera consulta

## 12.3. Diagrama de Grafo Acíclico Dirigido (DAG) - Segunda consulta

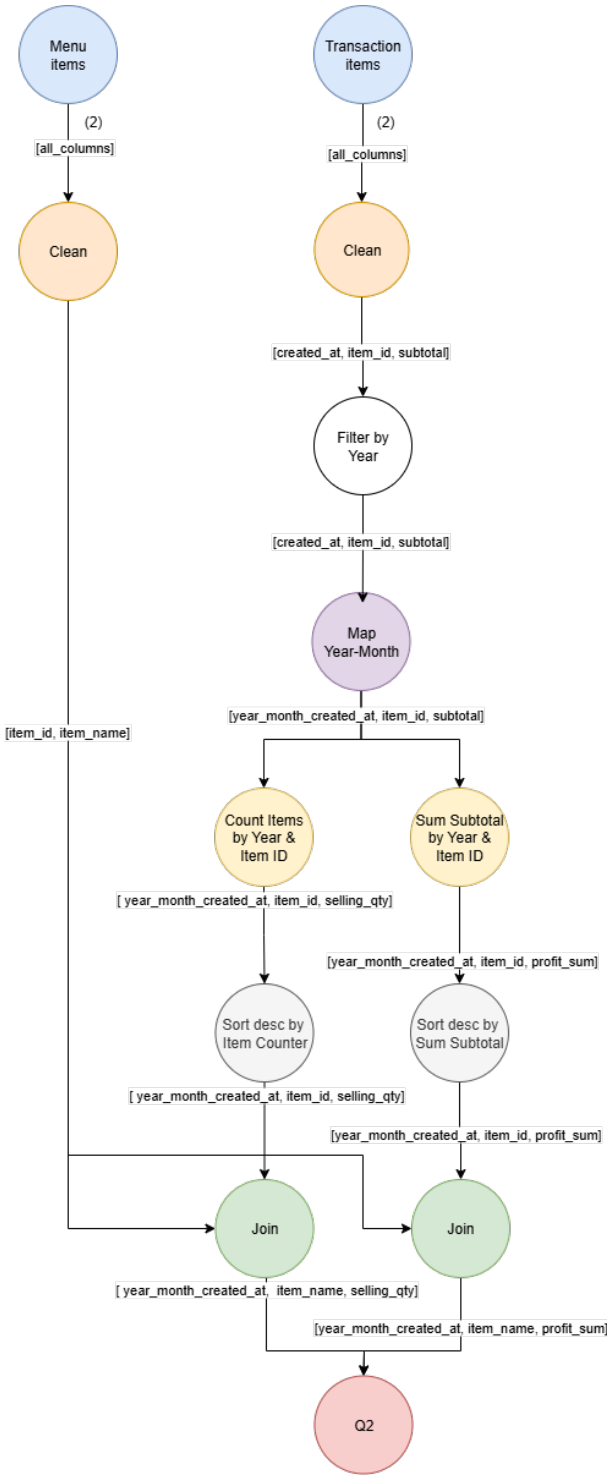


Figura 26: Diagrama de Grafo Acíclico Dirigido (DAG) - Segunda consulta

## 12.4. Diagrama de Grafo Acíclico Dirigido (DAG) - Tercera consulta

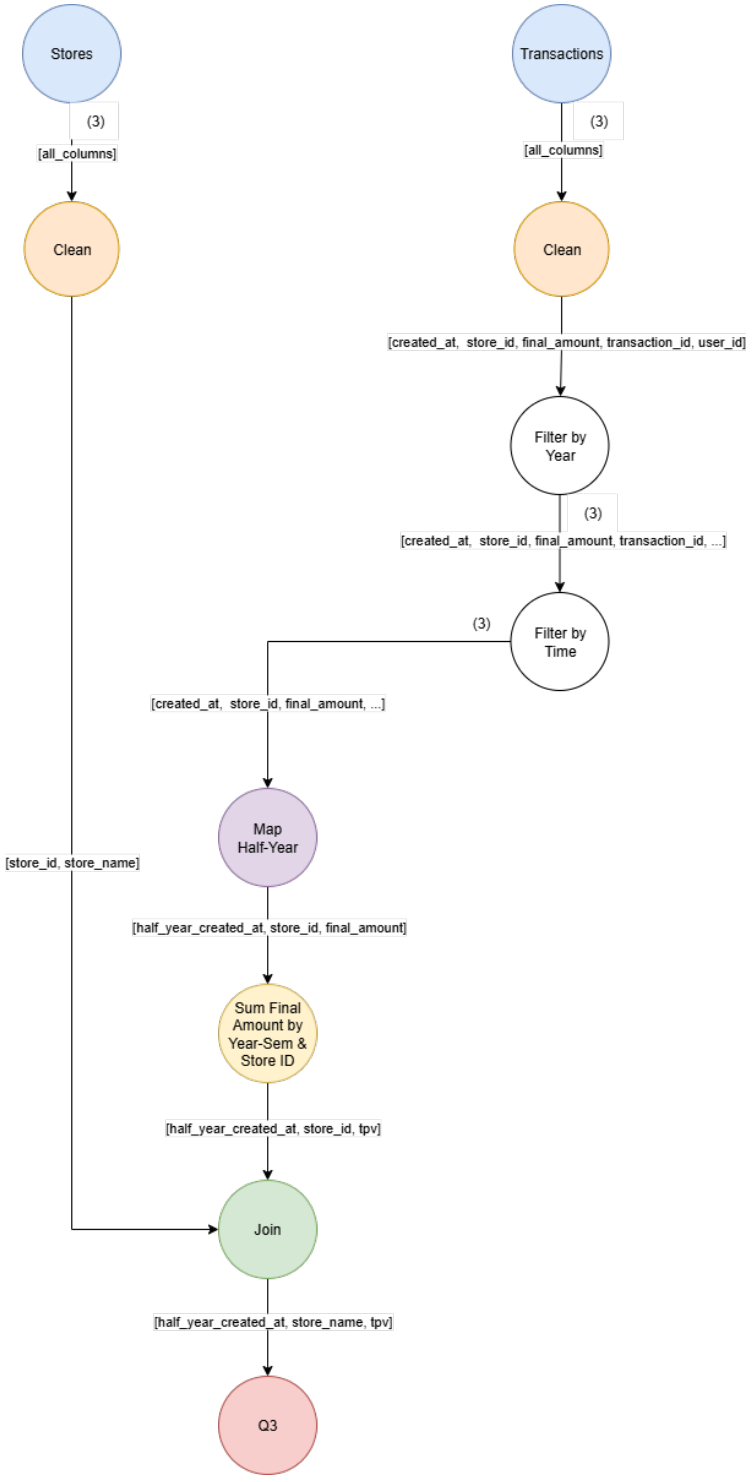


Figura 27: Diagrama de Grafo Acíclico Dirigido (DAG) - Tercera consulta

## 12.5. Diagrama de Grafo Acíclico Dirigido (DAG) - Cuarta consulta

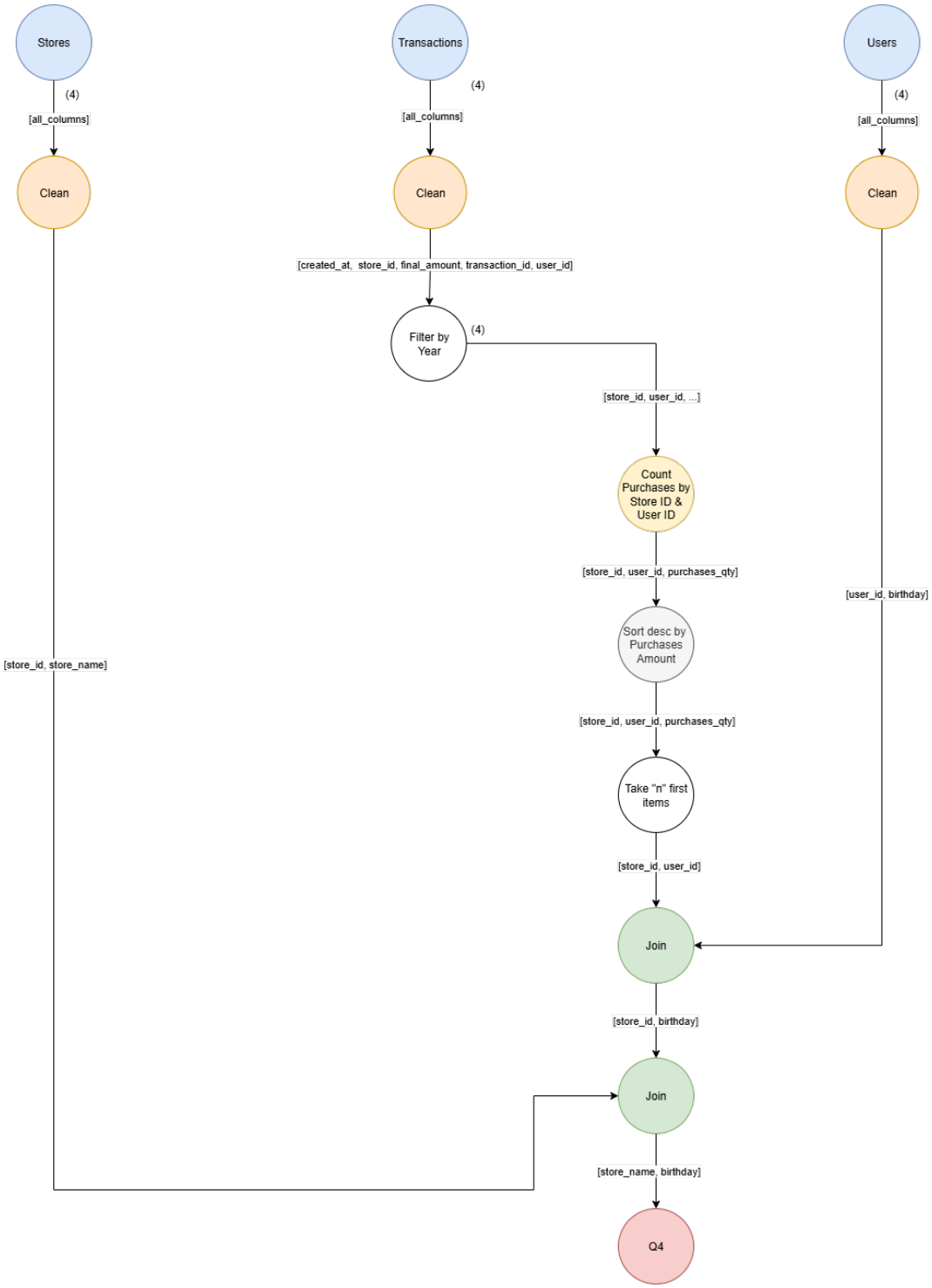


Figura 28: Diagrama de Grafo Acíclico Dirigido (DAG) - Cuarta consulta

# 13. Middleware

El middleware interno constituye el pilar fundamental de la comunicación en el sistema *Çoffee Shop Analysis"*, siendo el responsable de orquestar la interacción entre el servidor central y los múltiples nodos de procesamiento. Su arquitectura se basa en un modelo de Middleware Orientado a Mensajes (MOM), implementado mediante la herramienta **RabbitMQ**, que actúa como bróker central, tal como se visualiza en el Diagrama de Despliegue.

La principal función de este middleware es abstraer la complejidad de la comunicación en red, garantizando que los nodos puedan interactuar de forma desacoplada y asincrónica. En lugar de conexiones punto a punto, los componentes se comunican a través de colas y tópicos, definidos estratégicamente para cada escenario. Este diseño aporta escalabilidad, flexibilidad y la posibilidad de añadir o remover nodos de procesamiento dinámicamente sin alterar la lógica central.

La lógica de conexión se implementa en el componente `MQConnectionHandler`, ubicado en el paquete `shared`, asegurando un manejo uniforme de las comunicaciones. Sobre esta capa de mensajería se diseñó un protocolo de aplicación específico que define la estructura de los mensajes para el envío de datasets, la transmisión de resultados y el manejo de errores.

Para interactuar con el Middleware se implementaron las interfaces provistas por la cátedra, y se desarrollaron todos los tests unitarios correspondientes con el fin de validar su correcto funcionamiento. De esta manera, se cumple con el requisito planteado en el enunciado de abstraer la comunicación entre los nodos del sistema distribuido.

## Esquemas de comunicación

Durante el desarrollo se identificaron distintos patrones de comunicación, y para cada uno se diseñó una solución particular apoyada en las herramientas de RabbitMQ. A continuación, se describen los principales casos implementados:

### Comunicación mediante colas dedicadas

En los nodos escalables, se definió una cola por cada instancia de nodo. Esto asegura la ausencia de *race conditions* y permite direccionar la información de forma controlada, garantizando el funcionamiento correcto y balanceado del sistema.

Para decidir a qué cola enviar cada batch de datos se utilizaron dos estrategias:

1. **Round Robin:** cada emisor mantiene un contador que se incrementa en cada envío, distribuyendo los lotes de datos de forma equitativa entre las colas disponibles. Una vez alcanzado el último nodo, el contador retorna al primero. Esta técnica fue utilizada en:
   - Servidor → Cleaners.
   - Cleaners → Filters.
   - Filters → Filters.
2. **Hashing por clave (Sharding):** se empleó en los casos en que los datos debían ser dirigidos a un nodo específico según una clave. Por ejemplo, en la comunicación *Cleaners de Usuarios* → *Joins de Usuarios*, la asignación de transacciones a los nodos de Join se resolvió aplicando una función hash sobre el ID de usuario. Concretamente, se utilizó el resto de la división entera del ID por la cantidad de nodos shardeados, lo que asegura que todas las transacciones de un mismo usuario lleguen al mismo nodo de Join.

### Comunicación mediante exchanges

En situaciones donde la misma información debía ser replicada en múltiples colas, se utilizó un **exchange** de RabbitMQ. Este patrón resultó especialmente útil en la **Query 2**, donde se debía dividir la información para dos subconsultas distintas: los ítems más vendidos y los que generaron mayor facturación.

En este caso, el exchange distribuye la información procesada por el nodo *Map Year Month* hacia las colas de:

- Count Items.
- Sum Items.

### Comunicación de nodos escalables hacia un nodo no escalable

En escenarios donde múltiples nodos escalados debían enviar resultados a un nodo único no escalable (generalmente de tipo *Reduce*), se diseñó una cola de recolección centralizada. Dicho nodo recopilaba toda la información y generaba el reporte final una vez recibidos los resultados de todos los emisores.

Este esquema fue utilizado en:

- Filters → Map Year Month / Map Year Semester.
- Filters → Count Purchases.
- Filters → Output Builder.
- Join Users → Join Stores.

### Comunicación 1 a 1

Finalmente, en los casos donde la comunicación era estrictamente secuencial entre nodos no escalables, se implementaron colas directas de 1 a 1. Los principales casos fueron:

- Count Items → Sort Items.
- Count Purchases → Sort Purchases.
- Sum Items → Sort Items.
- Sort Items → Join Items With Menu.
- Join Items With Menu → Output Builder.
- Join Stores → Output Builder.
- Sum Final Amount → Join Stores.

## Conclusión

La combinación de estas estrategias de comunicación permitió cumplir con los requisitos de abstracción y desacoplamiento planteados, al mismo tiempo que se aprovecharon al máximo las capacidades de RabbitMQ. Gracias a esta arquitectura, el sistema puede escalar horizontalmente de manera sencilla y mantener la coherencia en el flujo de datos, asegurando un comportamiento correcto incluso en escenarios de alta concurrencia.

# 14. Mensajes y Protocolos

## 14.1. Protocolo

Para la implementación del sistema distribuido se desarrolló un protocolo de comunicación específico, adaptado a las necesidades del presente trabajo.

Este protocolo define las reglas de interacción entre el nodo coordinador, los nodos de procesamiento y los clientes del sistema.

La comunicación se basa en el intercambio de mensajes estructurados, que incluyen información sobre las operaciones a realizar, los datos a procesar y los resultados obtenidos.

## 14.2. Mensajes

Como parte del protocolo, se estableció un conjunto de mensajes bien definidos, orientados a cubrir las operaciones básicas del sistema: envío de datasets, distribución de tareas, transmisión de resultados y manejo de errores.

Cada mensaje tiene su propia estructura, que incluye campos para el tipo de mensaje, identificadores únicos, datos relevantes y metadatos necesarios para la correcta interpretación del mensaje.

Toda la implementación del protocolo y los mensajes se encuentra definido en el archivo `src/shared/communication_protocol.py`

# 15. Mecanismos de Control

## 15.1. Mecanismos de Control de Sincronización

En el diseño del sistema distribuido cuenta con la incorporación de mecanismos de control de sincronización, con el objetivo de evitar problemas asociados a la concurrencia, tales como *race conditions* o *deadlocks*.

Estos mecanismos garantizan un acceso ordenado a los recursos compartidos y la correcta coordinación entre los procesos en ejecución.

## 15.2. Mecanismos de Control de Señales

El sistema incorpora un manejo explícito de señales a fin de asegurar una finalización correcta y segura de los procesos.

En particular, se controla la señal `SIGTERM`, permitiendo un *graceful quit* que libera recursos, cierra archivos abiertos y mantiene la consistencia del sistema.

## 15.3. Mecanismos de Control de Fallas

Se contemplará el desarrollo de mecanismos de control de fallas para aumentar la robustez y confiabilidad del sistema. Estos mecanismos estarán orientados a gestionar situaciones adversas como caídas de procesos, pérdida de nodos de cómputo o interrupciones inesperadas durante la ejecución.

El objetivo es garantizar, en la medida de lo posible, la continuidad del procesamiento y la recuperación parcial de información. En futuras versiones del documento se describirán las estrategias de detección, mitigación y recuperación ante fallas.

# 16. Mediciones de rendimiento del sistema implementado

Para evaluar el rendimiento del sistema distribuido implementado, se realizaron diversas mediciones bajo diferentes configuraciones y cargas de trabajo. A continuación, se detallan los resultados obtenidos y su análisis.

## 16.1. Configuración del entorno de pruebas

Las pruebas se llevaron a cabo en un entorno controlado, utilizando una red local con múltiples nodos de cómputo.

Todos los nodos escalables, fueron configurados para levantar 2 nodos de cómputo cada uno.

Repsecto a los nodos no escalables, solo se levantó un nodo de cómputo por cada uno (Es decir, para esta entrega no se implementó redundancia en los nodos no escalables, pero no se descarta para futuras versiones del desarrollo).

## 16.2. Tiempos medidos

APARTADO EN PROCESO...

# 17. Cronograma teórico del desarrollo

## 17.1. Diseño

Previo al inicio del desarrollo efectivo del diseño, el equipo realizó una instancia de planificación para organizar la distribución de responsabilidades. El objetivo fue asegurar que cada integrante asumiera un conjunto de tareas equilibrado, alineado tanto con sus fortalezas como con las necesidades del proyecto.

La planificación se estructuró en función de las vistas arquitectónicas requeridas, los diagramas de modelado, la documentación conceptual y las definiciones de base.

## 17.2. Escalabilidad

El desarrollo de esta entrega cuenta con un tiempo estimado de 2.5 semanas. Durante la primera semana se realizará un análisis detallado de los criterios de escalabilidad, los requisitos técnicos y el alcance de la entrega, así como la planificación y distribución de tareas entre los integrantes del equipo.

En las semanas posteriores se definirán y ejecutarán las labores específicas de cada integrante, las cuales se documentarán en versiones futuras de este informe. La presente sección cumple únicamente la función de establecer la planificación inicial.

## 17.3. Multi-Client

El desarrollo de esta entrega cuenta con un tiempo estimado de 2 semanas. La primera semana estará destinada al análisis de los criterios, requisitos y alcance de la funcionalidad de múltiples clientes, junto con la planificación y asignación de tareas a cada integrante del equipo.

En la semana restante se abordará la ejecución de las labores planificadas, que se detallarán en próximas versiones de este documento. En esta instancia se deja asentada únicamente la planificación teórica.

## 17.4. Tolerancia

El desarrollo de esta entrega cuenta con un tiempo estimado de 3.5 semanas. La primera semana se dedicará al análisis de criterios, requisitos de tolerancia a fallos y alcance de la entrega, además de la planificación y distribución de responsabilidades entre los integrantes.

Las semanas siguientes estarán orientadas a la implementación progresiva de los mecanismos planificados. Las labores específicas por integrante se incluirán en futuras actualizaciones de este documento.

## 17.5. Paper

El desarrollo de esta entrega cuenta con un tiempo estimado de 1.5 semanas. Durante la primera semana se llevará a cabo el análisis de criterios, requisitos de formato y alcance de la entrega, así como la planificación y reparto de tareas.

El tiempo restante será utilizado para la redacción y consolidación del documento final, cuyos detalles se incorporarán en versiones posteriores de este informe. La presente sección constituye únicamente la documentación inicial de la planificación.

# 18. Cronograma real del desarrollo

## 18.1. Diseño

El desarrollo del diseño se llevó a cabo en un período de dos semanas, conformando un equipo de tres integrantes.

Durante la **primera semana**, el grupo se enfocó en interpretar en profundidad los requisitos planteados en el enunciado, analizar la lógica de las consultas y definir el diseño conceptual del sistema. Asimismo, en esta etapa se realizó la repartición de tareas entre los integrantes, de manera de optimizar el tiempo de ejecución restante.

En la **segunda semana**, cada integrante se abocó a las actividades asignadas, según el siguiente detalle:

### Luciano

- Vista Lógica.
- Vista de Desarrollo.
- Vista Física.
- Vista de Procesos.
- Diseño inicial/conceptual del Middleware.
- Diagrama de Paquetes.

### Axel

- Escenarios de uso (Casos de Uso).
- Diagrama de Secuencia.
- Diagrama de Actividades.
- Diagrama de Robustez.
- DAG (Directed Acyclic Graph).

### Felipe

- Redacción de las definiciones iniciales.
- Creación de los cronogramas.
- Descripción detallada de la ejecución de las consultas.
- Explicaciones introductorias de los mecanismos de control y protocolos.

## 18.2. Middleware y Coordinación de Procesos

El desarrollo de la primera versión del sistema distribuido se llevó a cabo en un período de dos semanas y media, conformando un equipo de tres integrantes.

Durante la **primera semana**, el grupo se enfocó en interpretar en profundidad los requisitos planteados en el enunciado, analizar posibles implementaciones para las consultas solicitadas, definir el lenguaje de programación a utilizar, pactar protocolos e identificar bibliotecas necesarias para el desarrollo.

Asimismo, en esta etapa se realizó la repartición de tareas entre los integrantes, de manera de optimizar el tiempo de ejecución restante.

En la **segunda semana**, cada integrante se abocó a las actividades asignadas, según el siguiente detalle:

### Luciano

- Implementación del Middleware.
- Creación de los tests unitarios para validad funcionamiento del Middleware.
- Establecimiento de conexiones entre Cliente y Servidor.
- Redacción de protocolos de comunicación.
- Integración del Middleware con los controladores.

### Axel

- Implementación de los controladores:
  - Join.
  - Sorts.
  - Maps.
  - Filters.
- Integración de los controladores con el Middleware.
- Construcción del ecosistema de nodos de cómputo distribuido mediante a Dockerfiles.
- Medición del rendimiento del sistema.

### Felipe

- Implementación de los controladores:
  - Cleaners.
  - Output Builders.
  - Reduces.
- Corrección del diseño inicial según las necesidades del desarrollo y las correcciones indicadas por el docente a cargo.
- Redacción de la nueva documentación del sistema.
- Implementación de los mecanismos de cierre 'graceful' de los nodos de cómputo.