



**FACULTAD
DE INGENIERIA**

Universidad de Buenos Aires

UNIVERSIDAD DE BUENOS AIRES

FACULTAD DE INGENIERÍA

**Trabajo Práctico Diseño:
Middleware y Coordinación de Procesos**

SISTEMAS DISTRIBUIDOS I (TA050)

2° CUATRIMESTRE 2025

Alumno	Padrón	E-mail
Federico Genaro	109447	fgenaro@fi.uba.ar
Santiago Sevitz	107520	ssevitz@fi.uba.ar
Máximo Utrera	109651	mutrera@fi.uba.ar

Fecha de Presentación: 16 de Septiembre de 2025

Índice

1. Introducción	2
2. Escenarios y necesidades del cliente	2
3. Análisis de los conjuntos de datos a procesar	2
4. Procesamiento y flujo de datos	3
4.1. Cadenas de procesado (pipelines)	3
4.1.1. Requisito (1)	3
4.1.2. Requisito (2)	3
4.1.3. Requisito (3) y (4)	4
4.2. Flujo de mensajes y procesos (punta a punta)	4
4.2.1. Resumen de comunicación del sistema	5
4.2.2. Profundización sobre la comunicación del sistema	6
4.3. Actividades de cada proceso del sistema	6
4.3.1. Punto de Entrada	6
4.3.2. Procesamiento	7
4.3.3. Post-procesamiento	8
4.3.4. Fin de transmisión	8
5. Arquitectura e Infraestructura	9
5.1. Arquitectura y medios de comunicación	9
5.1.1. Proceso inicial común	10
5.1.2. Arquitectura de procesos para cada requisito	10
5.2. Estructura de paquetes y módulos de cada actor	12
5.3. Infraestructura del sistema	14
6. Tareas a ejecutar	15
6.1. Definición de tareas	15
6.1.1. Infraestructura principal	15
6.1.2. Gateway	15
6.1.3. Workers	15
6.1.4. Middleware	15
6.1.5. Despliegue y escalado	15
6.1.6. Pruebas	15
6.2. División entre integrantes	15
7. Referencias	16

1 Introducción

Este trabajo propone el diseño de un sistema distribuido para analizar datos de ventas de una cadena de cafeterías en Malasia. Se busca obtener información clave sobre transacciones, clientes y productos, priorizando la escalabilidad y robustez del sistema mediante conceptos y buenas prácticas de sistemas distribuidos.

2 Escenarios y necesidades del cliente

Para comenzar se hace un análisis de los requisitos funcionales que el sistema debe cumplir para ser de utilidad al cliente. Para esto se escriben los posibles casos de uso del mismo, y se modelan en un simple diagrama auto-explicativo.

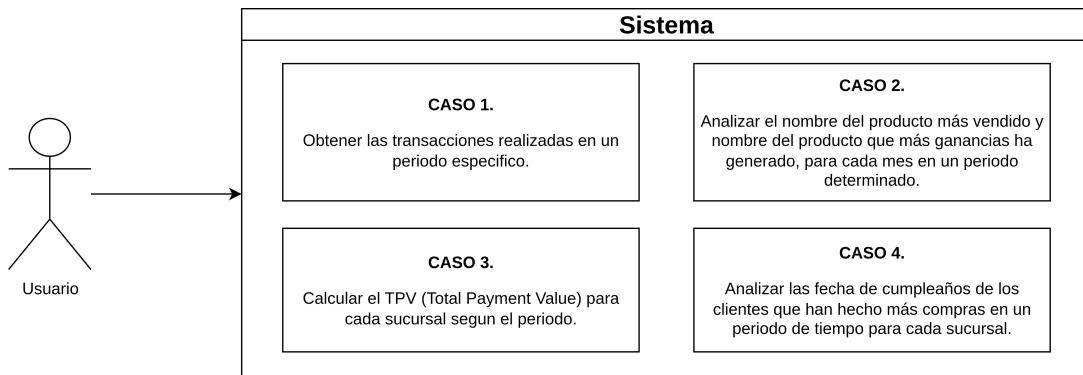


Figura 1: Actividades generales que el cliente debe ser capaz de realizar.

3 Análisis de los conjuntos de datos a procesar

Previo a detallar las capas de procesamiento per-se, se hace un análisis del conjunto de datos a procesar de manera de entender que transformaciones se necesitan aplicar.

El total de los conjuntos de datos que se tienen son:

1. Transactions (*múltiples archivos*)
2. Transaction Items (*múltiples archivos*)
3. Menu Items (*un archivo*)
4. Stores (*un archivo*)
5. Users (*múltiples archivos*)
6. Vouchers (*un archivo*)
7. Payment Methods (*un archivo*)

Dentro de los mismos, se consideran estrictamente necesarios los primeros dos conjuntos, ya que ofrecen la información principal que el cliente quiere analizar. En cuanto al resto de conjuntos, los siguientes tres (menu items, stores, users) se utilizarán para hacer uniones con la información principal obtenida, con el fin de que el resultado final sea un resumen fácil de entender al leerlo (sin números de identificación, etc).

Esto quiere decir que se deja sin uso a los últimos dos conjuntos de datos, ya que no son estrictamente necesarios para obtener el resultado esperado por el cliente.

4 Procesamiento y flujo de datos

En esta sección se tiene el objetivo de demostrar el funcionamiento del sistema a gran escala, explicando la cadena de procesos por la que pasara la información, así como también la comunicación entre el cliente y el sistema, y la comunicación interna.

4.1. Cadenas de procesado (pipelines)

A continuación se demuestra como la información pasa por varias etapas de procesamiento en forma de cadena, hasta llegar al cliente, representado como sumidero (*sink*).

4.1.1. Requisito (1)

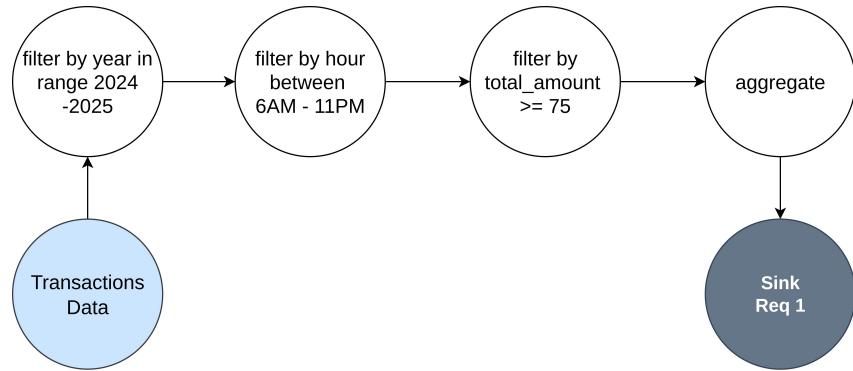


Figura 2: Procesado y confección del resultado para el primer requisito.

Como se puede observar en la figura, para este requisito se toma la información de las transacciones y se lleva a cabo una serie de filtrados con diferentes criterios (como por año, hora, cantidad total), para luego hacer un agregado de los datos y el envío del resultado al cliente.

4.1.2. Requisito (2)

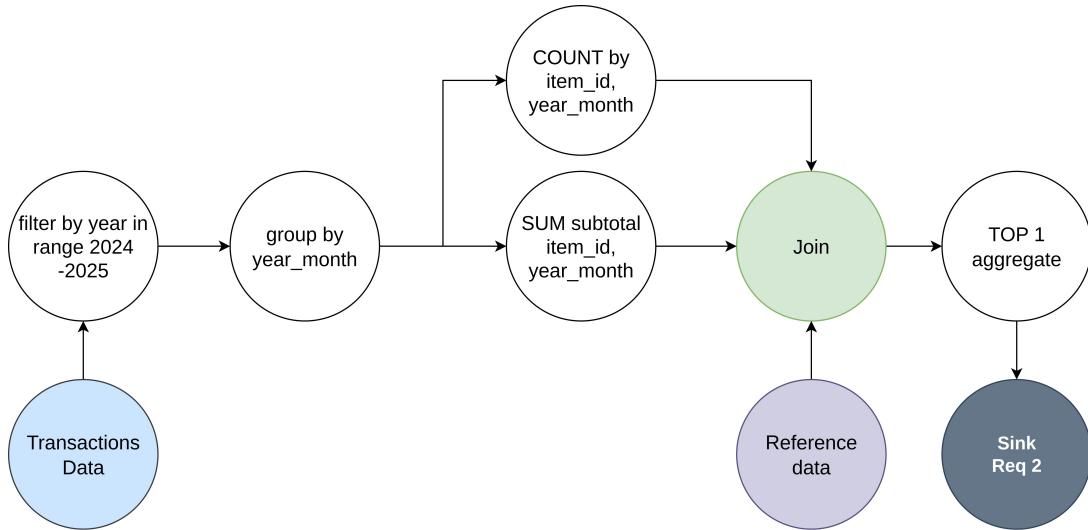


Figura 3: Procesado y confección del resultado para el segundo requisito.

Para el segundo requisito, el procesamiento ya no es tan trivial como un simple filtrado, sino que aquí la cadena se divide en dos caminos luego de filtrar y agrupar los datos por año y mes. Los dos caminos hacen operaciones de reducción diferentes pero ambos convergen en el *joiner*, que luego de desembocar en un *aggregator* y este en el sumidero.

4.1.3. Requisito (3) y (4)

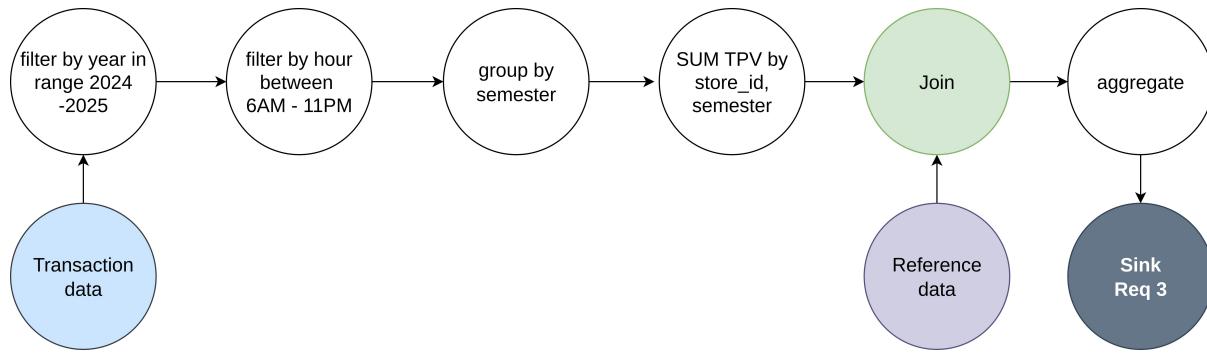


Figura 4: Procesado y confección del resultado para el tercer requisito.

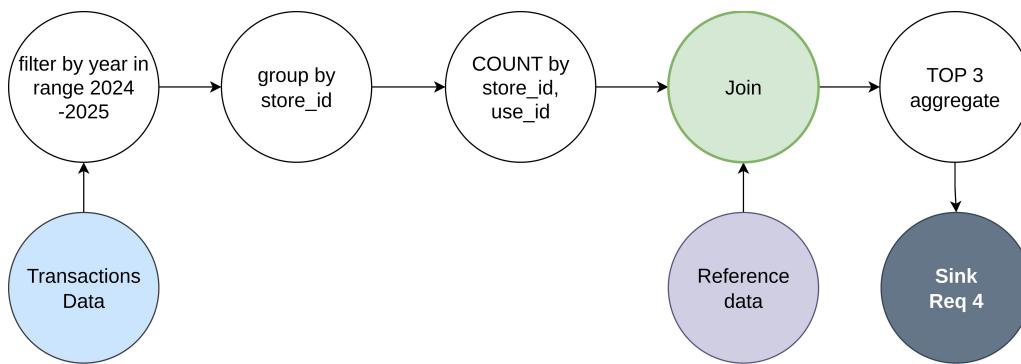


Figura 5: Procesado y confección del resultado para el cuarto requisito.

En el caso del tercer y cuarto requisito se lleva a cabo un proceso similar al del segundo, pero se simplifica un poco el flujo, ya que no hay una bifurcación para realizar la reducción (suma/conteo). Sin embargo ambos se diferencian principalmente en el comportamiento del paso final de agregación.

4.2. Flujo de mensajes y procesos (punta a punta)

En esta sección se presentan flujos de punta a punta que ilustran el funcionamiento general del sistema mediante diagramas.

A continuación, se detallan dos casos, uno más abstracto de las complejidades inherentes proponiendo una visión del procesamiento como un modelo de caja negra, y luego otro donde se profundiza en dicha caja negra para entender mejor la comunicación interna del sistema.

4.2.1. Resumen de comunicación del sistema

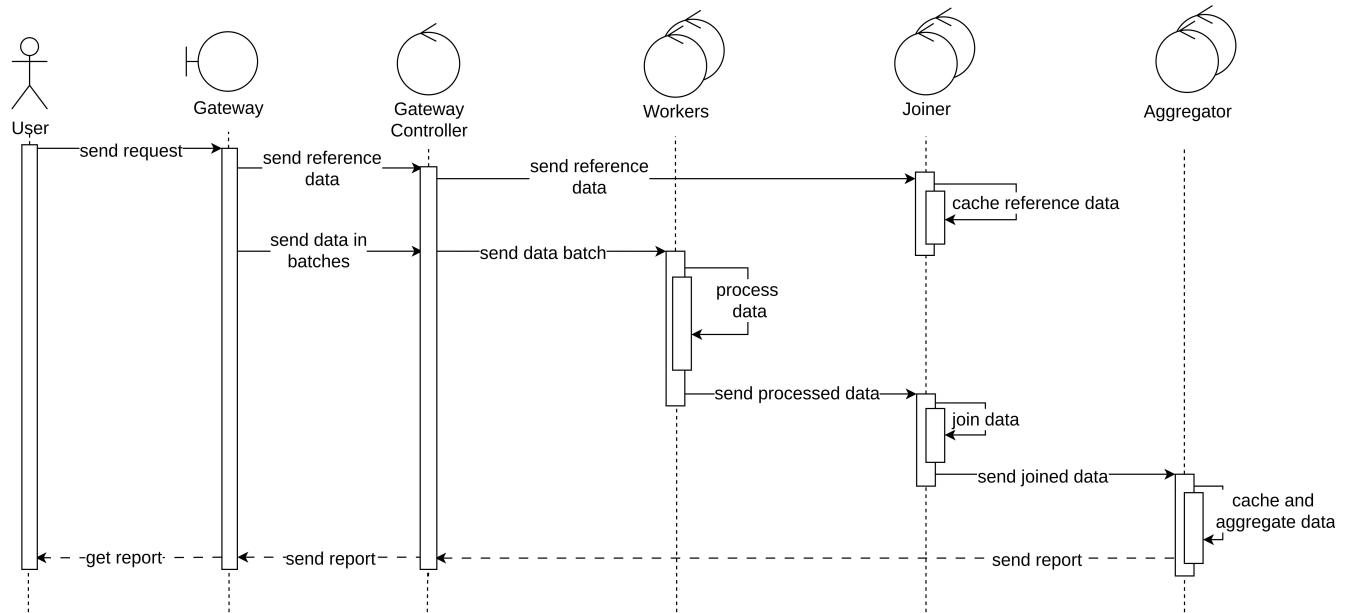


Figura 6: Funcionamiento general de sistema, con un resumen a gran escala del funcionamiento de los *workers*

El usuario interactúa únicamente a través del *gateway*, el cual transmite al *controller* la *reference data*, que posteriormente será utilizada por el *joiner* para vincular *datasets*. Una vez enviada la información de referencia, el *controller* envía en *batches* los datos de transacciones a procesar.

En la figura se resumen las tareas ejecutadas por los *workers*: tras finalizar el procesamiento, los datos son integrados en el *joiner* y posteriormente almacenados en el *aggregator*. Una vez transmitidos todos los *batches*, el *aggregator* genera y envía el reporte al *controller*, luego al *gateway*, y finalmente el usuario recibe el reporte consolidado.

4.2.2. Profundización sobre la comunicación del sistema

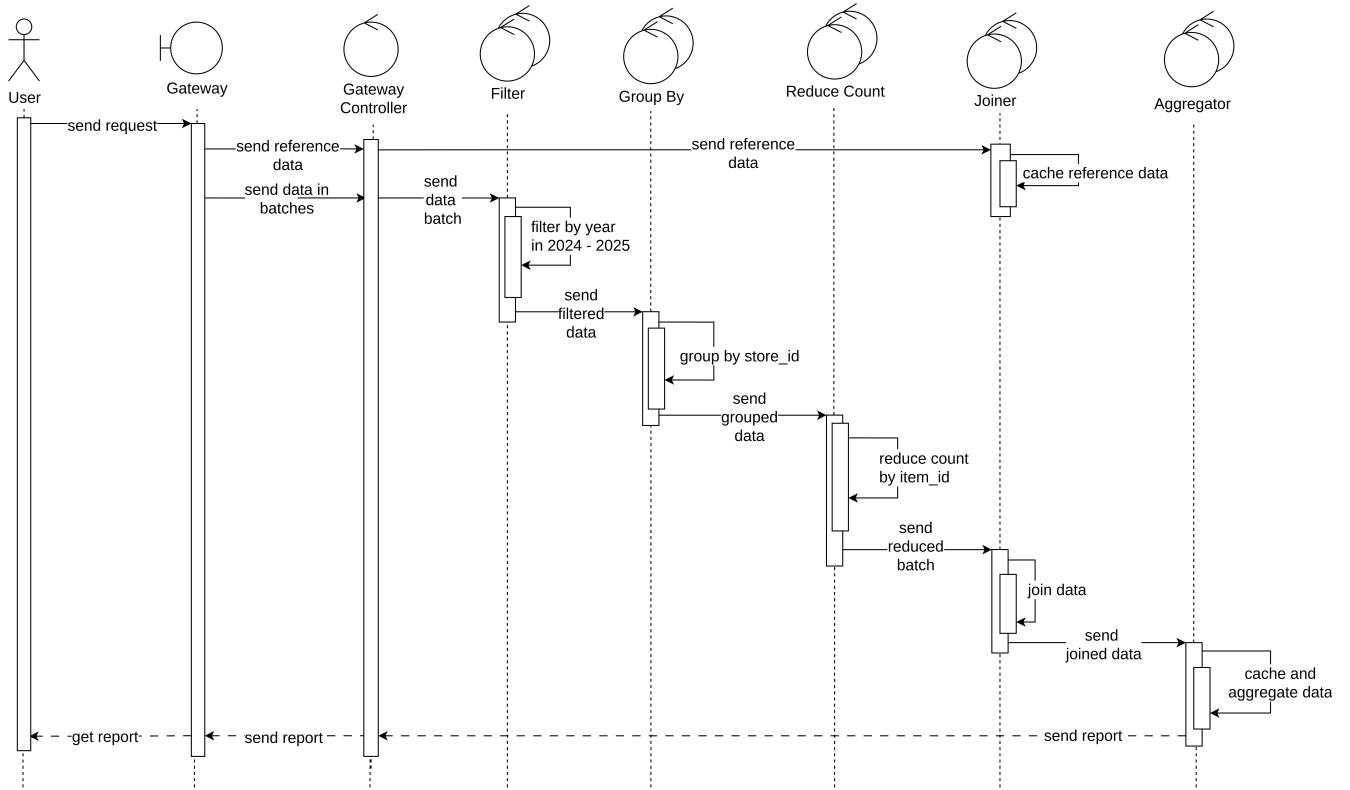


Figura 7: Requisito (3), Calcular el TPV (Total Payment Value) para cada sucursal entre 2024 y 2025

Como se observa en la Figura de arriba, los *workers* procesan cada *batch* aplicando su tarea específica (*filter*, *groupby*, *reduce*) y posteriormente lo envían al siguiente *worker*, donde la información se refina mediante la operación correspondiente. Este flujo se repite hasta llegar al *joiner*, que vincula los datos procesados con el *dataset* de referencia, para finalmente ser enviados al *aggregator*, quien se encarga de crear el reporte.

4.3. Actividades de cada proceso del sistema

En esta sección se describen las actividades realizadas por cada actor del sistema, tomando como caso de referencia el requisito 2.

4.3.1. Punto de Entrada

En principio, se observa la interacción entre los nodos *Gateway* y *Gateway Controller*, que funcionan como punto de entrada al sistema.

El *Gateway* constituye la interfaz utilizada por el usuario para enviar comandos y recibir reportes.

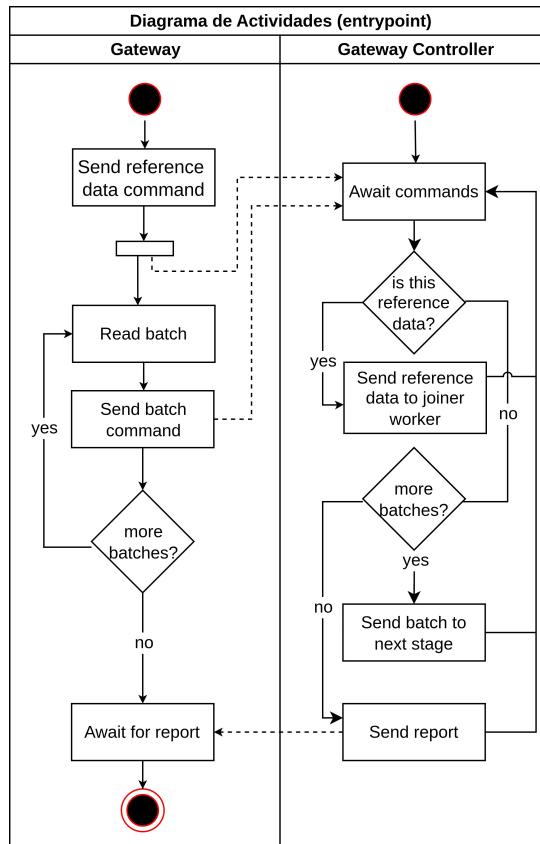


Figura 8: Actividades del punto de entrada al sistema.

A través del *Gateway*, primero se envía un *dataset* de referencia que será utilizado posteriormente para vincular los datos procesados. Luego, el *Gateway* transmite los *batches* de datos hasta agotar la información disponible, quedando a la espera de la generación del reporte. Durante este proceso, el *Controller* recibe cada comando y determina si corresponde a *reference data*, *batch data* o a la finalización del envío de *batches* por parte del usuario (este último proceso se detalla más adelante).

4.3.2. Procesamiento

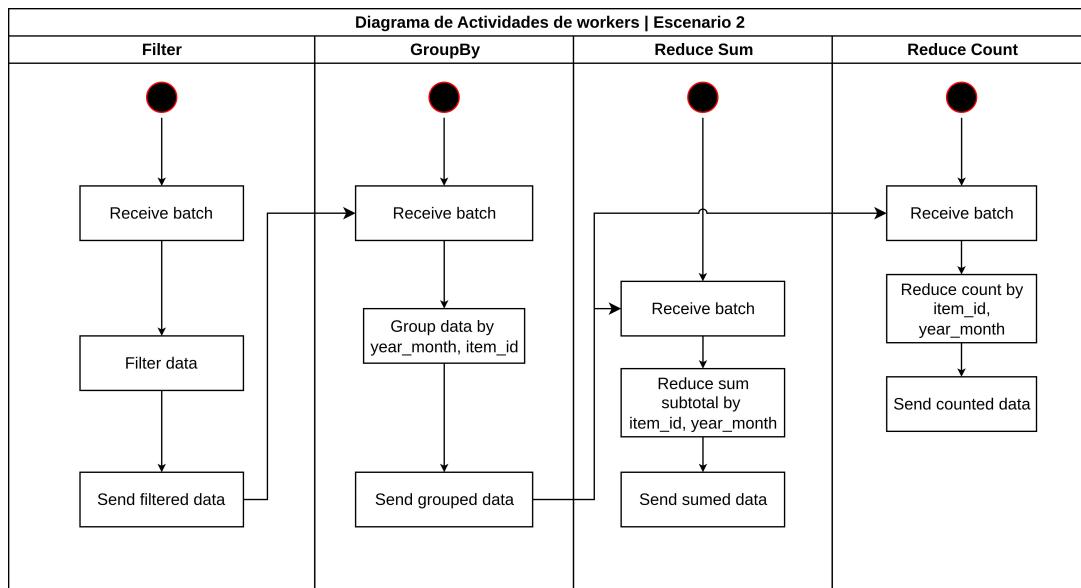


Figura 9: Actividades principales de procesamiento de información.

Cuando el *Gateway* recibe un *batch* de información para ser procesado, este es enviado inmediatamente al *worker* correspondiente.

Como se observa en la figura anterior, cada *worker* ejecuta tareas similares que consisten en recibir los datos, procesarlos y remitir el resultado al siguiente *worker*, encargado de continuar el refinamiento de la información.

Cada uno de los workers tienen estos tareas que hacer:

- **Filter Worker** selecciona datos en función de criterios definidos (por ejemplo, rango temporal).
- **GroupBy Worker**: agrupa los registros según diferentes atributos, como el ID de un ítem o el mes del año.
- **Reducer Workers**: realizan operaciones de reducción, ya sea sumatoria o conteo.

4.3.3. Post-procesamiento

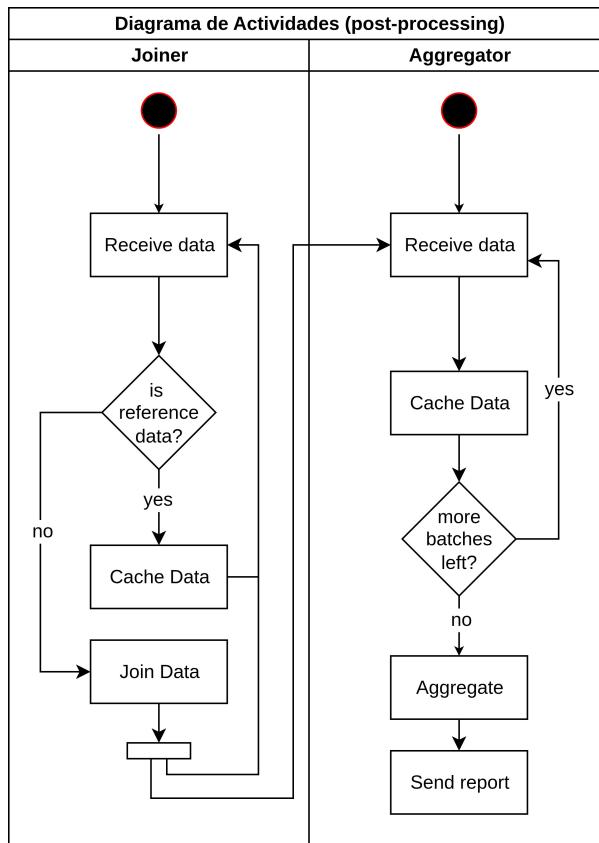


Figura 10: Actividades principales de post-procesamiento de información.

Este diagrama detalla el funcionamiento interno de los *workers* responsables del post-procesamiento, en particular el *Joiner* y el *Aggregator*. El *Joiner* recibe la *reference data* directamente del *Controller* y la almacena en su caché. Además, recibe los datos reducidos para vincularlos con el *dataset* de referencia, y posteriormente los envía al *Aggregator*.

El *Aggregator* tiene la responsabilidad de recibir y almacenar los datos hasta completar la recepción de todos los *batches* procesados, con el fin de generar el reporte final. Dicho reporte es enviado al *Controller* y, posteriormente, al *Gateway*.

4.3.4. Fin de transmisión

Por completitud, en la siguiente figura se ilustra el comportamiento del sistema ante la finalización del *input* del cliente. El diagrama inicia en el *Gateway Controller* para simplificar la representación, y cuando este recibe el reporte, lo devuelve de inmediato al cliente.

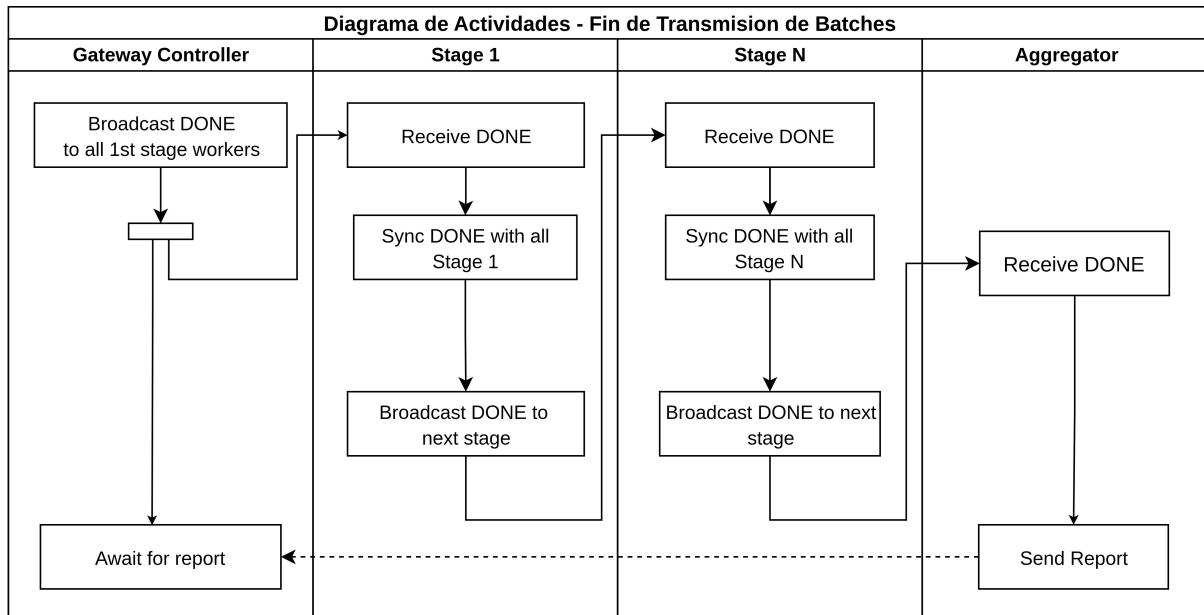


Figura 11: Actividades principales de post-procesamiento de información.

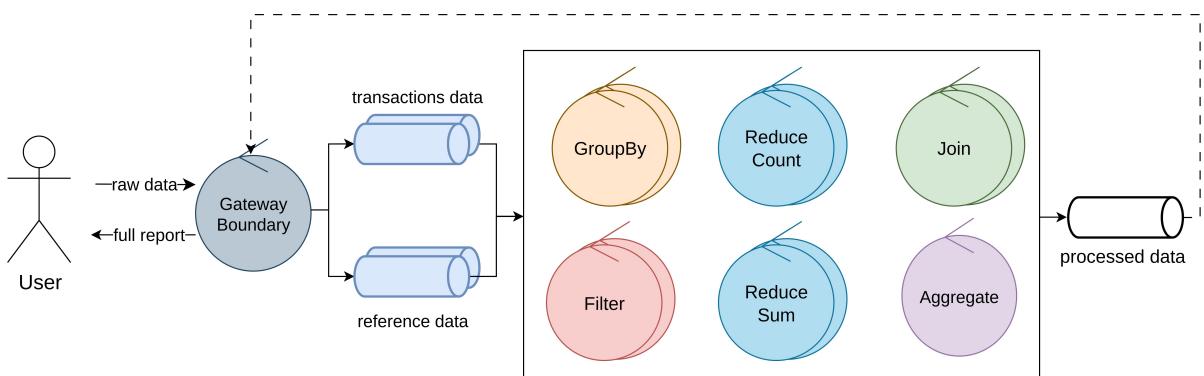
5 Arquitectura e Infraestructura

Para conformar la cadena de procesos mencionada en la sección anterior, se usara la estructura 'Worker per filter', que consta de tener una unidad de procesamiento aislada para cada etapa intermedia (filter, groupby, reduce count/sum, join y aggregate). Se opto por esta división ya que en el contexto de un sistema altamente distribuido, lleva a un mejor aprovechamiento de recursos, paralelismo y menos carga por *worker*, así como también mayor escalabilidad horizontal, a comparación de la división 'Worker per item'. Sin embargo, esta estructura también conlleva una mayor complejidad y necesidad de coordinación y comunicación entre nodos.

5.1. Arquitectura y medios de comunicación

En esta sección se presenta un diagrama de robustez, donde se ilustran las principales interacciones entre actores, fronteras y colas del sistema.

Se observa que toda comunicación entre nodos (exceptuando la comunicación cliente-gateway) se realiza a través de colas de mensajes. Cada worker toma información de una cola de entrada, la procesa y la envía a una cola de salida, de la cual leerá el próximo nodo en la linea de procesamiento. A su vez, las *replicas* de los nodos se grafican como círculos duplicados y apilados.

Figura 12: Visión general de la interacción entre *workers* y la transmisión de información mediante colas de mensajes.

En las siguientes figuras se amplían partes de interés del anterior diagrama de robustez, de modo de demostrar un seguimiento paso a paso de las interacciones.

5.1.1. Proceso inicial común

Proceso inicial común al que se someten todos los requerimientos antes de iniciar sus respectivos flujos

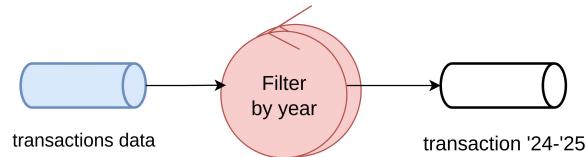


Figura 13: Filtrado básico por año (entre 2024 y 2025).

5.1.2. Arquitectura de procesos para cada requisito

A continuación se profundiza la arquitectura utilizada para el procesamiento de cada requisito, tomando como punto de partida el proceso inicial común mencionado.

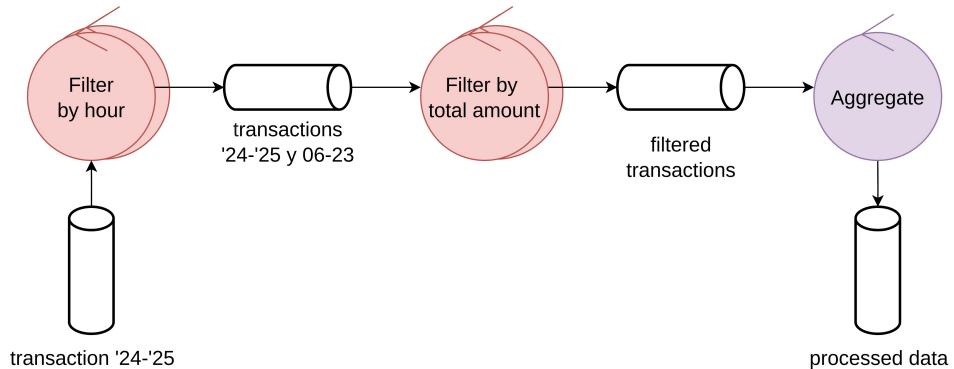


Figura 14: Camino relacionado al requisito 1.

En la figura previa se demuestra el flujo correspondiente al requisito 1. El proceso continua luego de los filtros previos mencionados en el proceso inicial, con un filtrado por hora y por monto total de las transacciones, para luego consolidar los resultados en una etapa de agregación, obteniendo finalmente los datos procesados. A su vez la comunicación entre *workers* se lleva a cabo siempre por medio de una cola de mensajes.

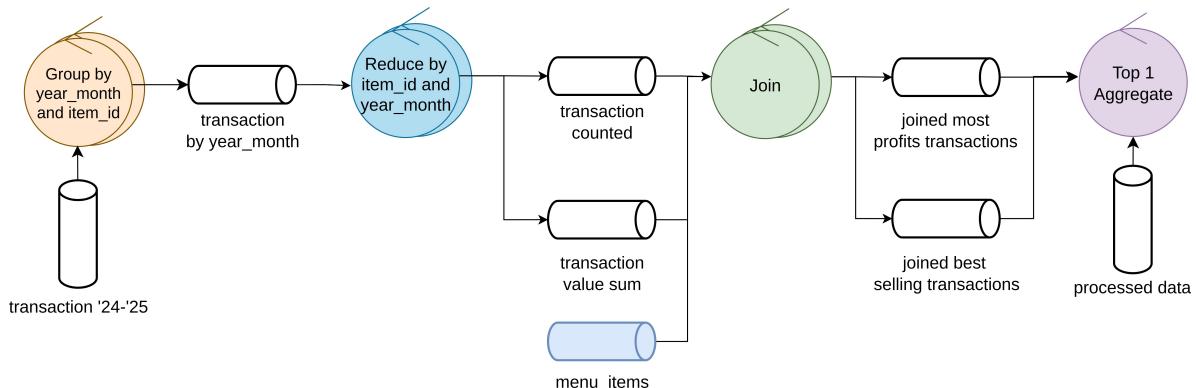


Figura 15: Camino relacionado al requisito 2.

La figura ilustra el flujo correspondiente al requisito 2. En este caso, las transacciones se agrupan por mes y posteriormente se envían a dos *workers reducer*: uno encargado de contar la cantidad de ítems y otro de calcular la suma del *final_value* de cada ítem. Esta bifurcación permite generar dos reportes distintos para el usuario. A continuación, cada conjunto de datos pasa por la etapa de *join*, donde se vinculan con el *dataset* de referencia, y finalmente ingresan al *aggregator*. Este último no solo almacena los *batches* procesados, sino que además ordena los resultados para seleccionar el *Top 1* ítem, ya sea el más vendido (mayor *count*) o el que generó mayores ganancias (mayor *final_value*).

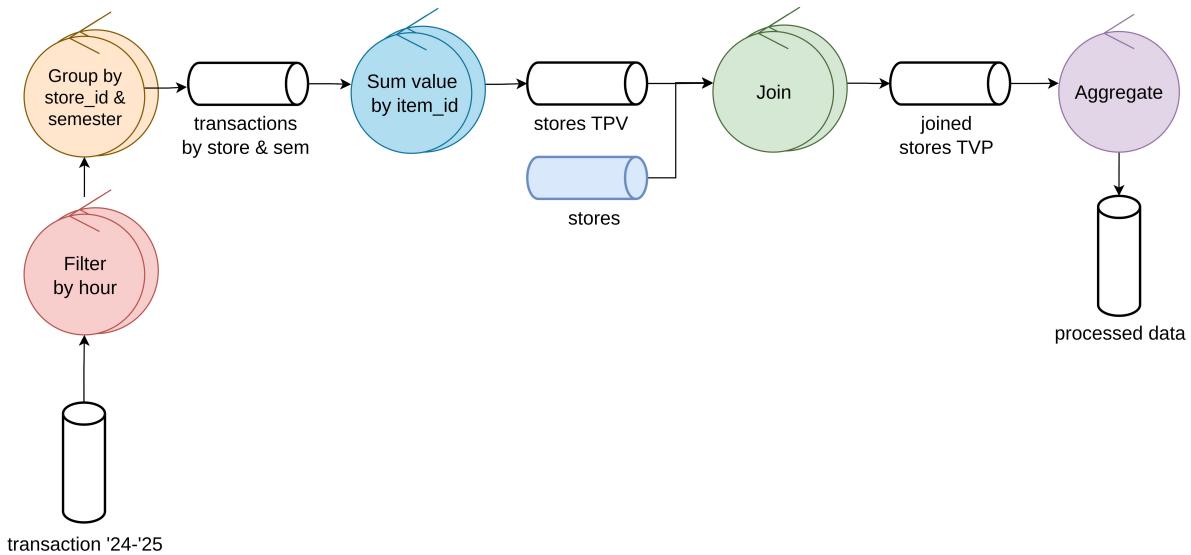


Figura 16: Camino relacionado al requisito 3.

La figura muestra el flujo asociado al requisito 3. En este escenario, las transacciones se agrupan por *store_id* y semestre, para luego calcular la suma de valores por *item_id*. Posteriormente, los resultados se vinculan en la etapa de *join* con el *dataset* de tiendas (*stores*), y finalmente se consolidan en el *aggregator*, obteniendo el reporte procesado.

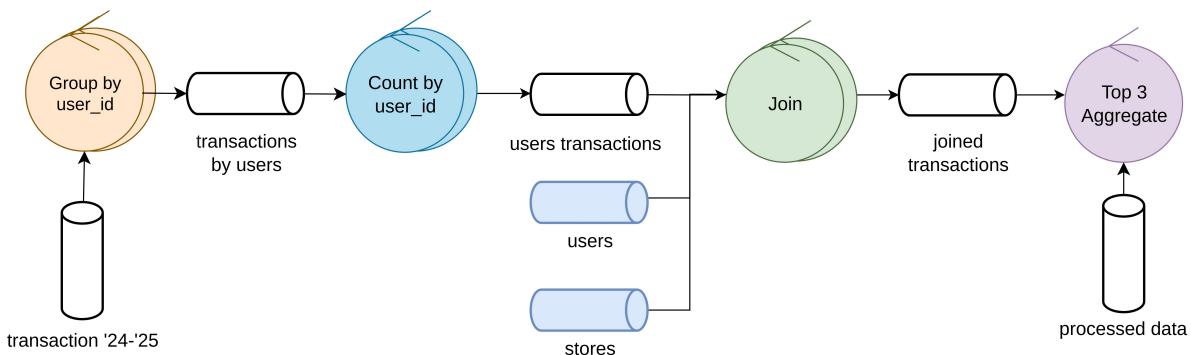


Figura 17: Camino relacionado al requisito 4.

La figura ilustra el flujo correspondiente al requisito 4. Las transacciones se agrupan por *user_id* y luego se cuentan en un *worker reducer*. Estos datos se combinan mediante la etapa de *join* con los conjuntos de usuarios (*users*) y tiendas (*stores*). Finalmente, el *aggregator* consolida la información y selecciona el *Top 3* de usuarios en función a la cantidad de transacciones hechas por usuario.

Observación: Cabe mencionar que el cierre ordenado del sistema en su completitud sera manejado mediante el uso de una cola definida especialmente para este propósito (*shutdown-queue*).

5.2. Estructura de paquetes y módulos de cada actor

En esta sección se observan diagramas explicativos de la estructura tentativa de módulos y paquetes de código a popular durante la implementación del sistema.

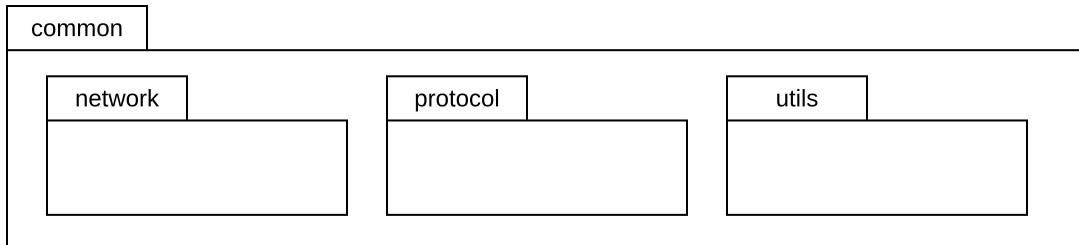


Figura 18: Modulo común a todos los paquetes de código.

Este modulo es el que contiene comportamiento relacionado a la red, comunicaciones y protocolos (e.g. protobufs).

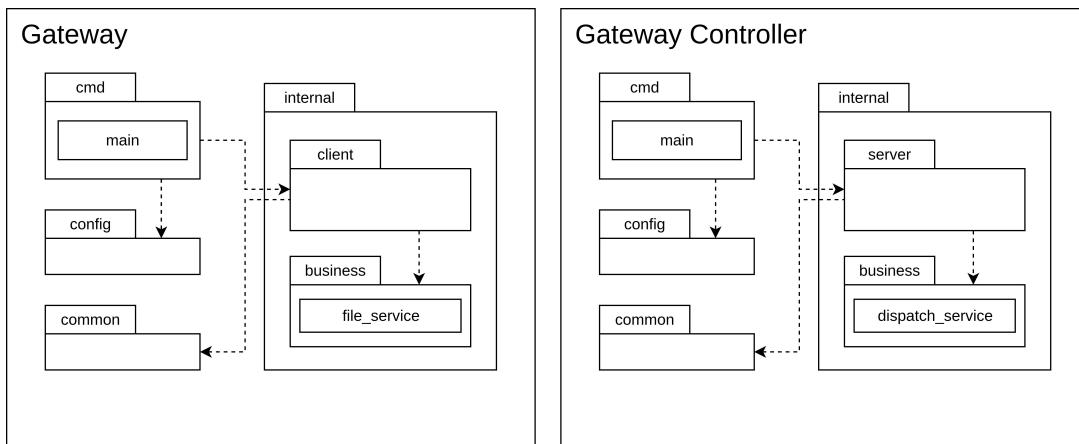
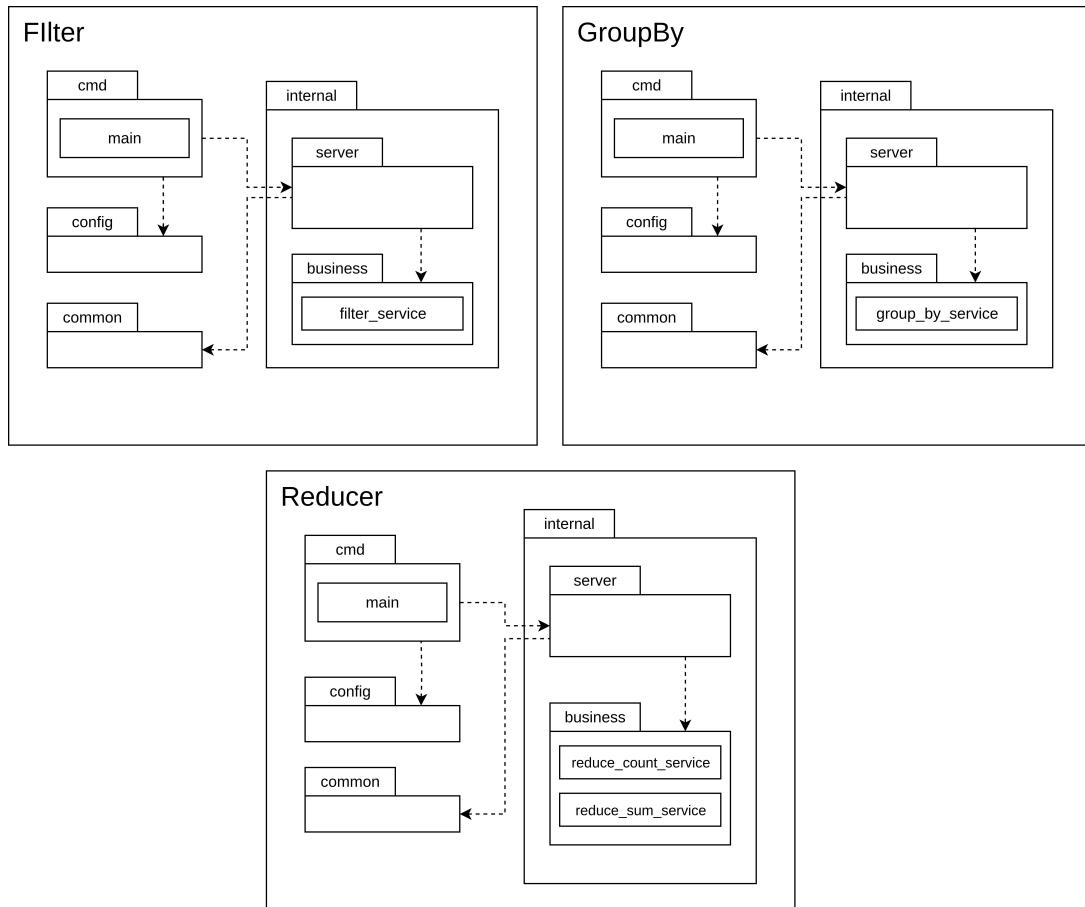
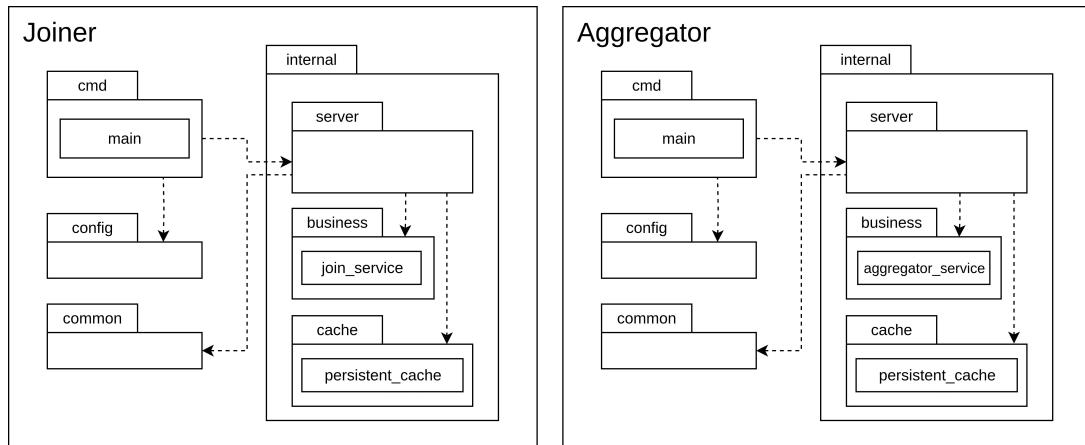


Figura 19: Paquetes de gateway y gateway controller (punto de entrada al sistema).

El paquete de *Gateway* contiene lógica de envío de información al servidor y recepción de datos completamente procesados, para lo que se comunica a través de la red con el servicio *Gateway Controller* cuyo paquete contiene comportamiento de recepción de clientes y despachado de tanto los datos crudos enviados por el cliente como los datos procesados saliendo del sistema.

Figura 20: Paquetes de *workers* de filtrado/reducción de datos.

Los paquetes de 'Filter', 'GroupBy' y 'Reducer', contienen la lógica de negocio principal del sistema, con la que se realizara el procesado completo de datos de manera distribuida.

Figura 21: Paquetes de *workers* de post-procesamiento y unificación de la información.

Los dos últimos paquetes diagramados, constan del comportamiento relacionado al refinado de datos (unir con los conjuntos de datos correspondientes, e.g. Transactions + Menu Items), y unificación de la información distribuida para poder ser enviada al cliente. Estos cuentan con un **cache** en disco de los datos procesados para evitar perdida de información durante re-inicios del *worker* por problemas inesperados.

5.3. Infraestructura del sistema

A continuación se detalla la infraestructura ideada para el funcionamiento distribuido del sistema, haciendo foco en la confiabilidad ante fallos y escalabilidad horizontal (agregando más nodos/computadores).

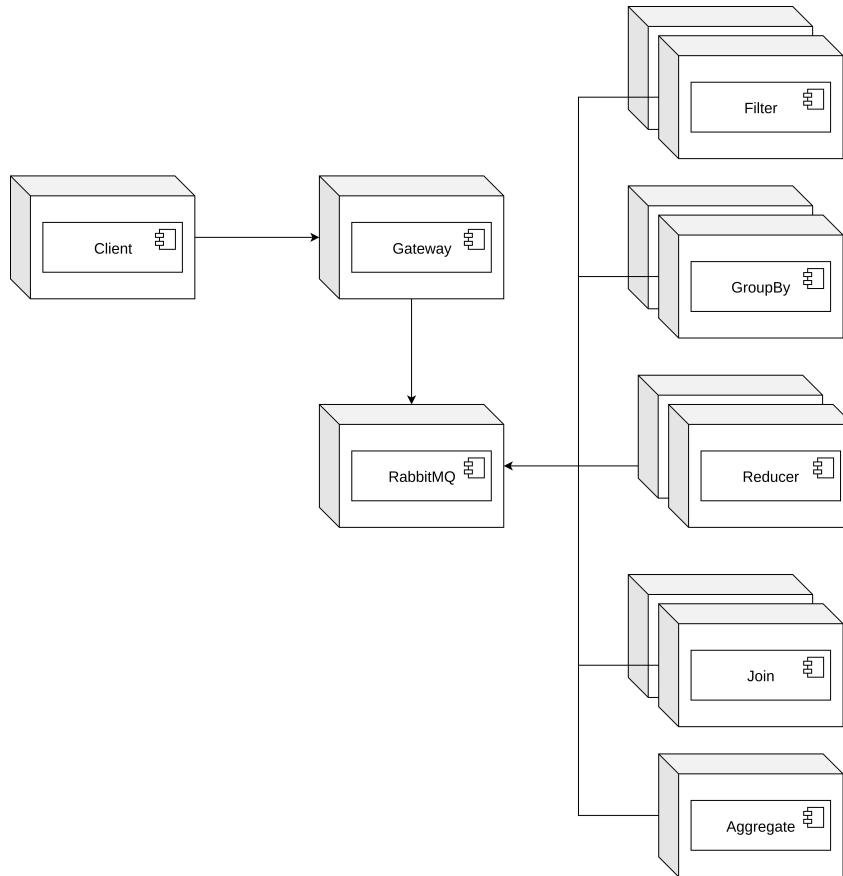


Figura 22: Arquitectura necesaria para desplegar el sistema.

Características:

- Se demuestra al cliente como un nodo aparte, que interactúa directamente con un *Gateway* para subir sus conjuntos de datos a ser procesados. Este último como se mencionó anteriormente, cumple el rol de punto de entrada al sistema.
- Entre nodos del sistema se comunican a través de un Middleware orientado a mensajes (MOM), utilizando colas.
- Reafirmando lo dicho en la sección de 'Vista de Desarrollo', los *worker nodes*, están separados por filtro, es decir hay un solo tipo de *worker* por computador. Alternativamente se puede modelar el sistema con procesos tomando el rol de nodos, en caso de no contar con un entorno multi-computadoras.
- Se toma como **único punto de falla** el servicio de colas de mensajería, proporcionado por RabbitMQ, se evalúa la posibilidad de tener réplicas para mayor disponibilidad.
- Los nodos de *Gateway* y *Aggregate* se identifican como componentes críticos del sistema, por lo que se propone en un futuro incorporar réplicas que aumenten la tolerancia a fallos.

6 Tareas a ejecutar

6.1. Definición de tareas

Definición y separación de tareas a realizar para el desarrollo del sistema por categoría.

6.1.1. Infraestructura principal

- Definir estructura del proyecto y paquetes de Go.
- Implementación de mensajes y serializado (protobuf, mensajes, manejo de errores).
- Uso de RabbitMQ e implementación de un wrapper.
- Creación de *worker* simple como base para los demás.

6.1.2. Gateway

- Implementación de interfaz de usuario con manejo *uploads* y *responses*.
- Re-envío de *batches* a la cola de los datasets correspondientes.
- Recolectado de datos procesados y envío al cliente.

6.1.3. Workers

- **Filter Worker:** Lógica de filtrado con lectura, y escritura de las colas.
- **GroupBy Worker:** Lógica de agrupado de batches.
- **Reducer Worker:** Lógica de reducción de batches, con los tipos *sum* y *count*.
- **Join Worker:** Lógica de unión de tablas, consumiendo datos de la cola correspondiente al dataset necesario.
- **Aggregator Worker:** Lógica de ordenamiento (Top N), acumulado de datos, confección y envío de resultados.

6.1.4. Middleware

- Configurar imagen de Docker de RabbitMQ.
- Definición de colas necesarias del sistema.

6.1.5. Despliegue y escalado

- Dockerizar cada paquete necesario para desplegar el sistema.
- Configuración de despliegue (numero de workers, etc) y Docker-compose.
- Plan de replicas para gateway y aggregators.

6.1.6. Pruebas

- Pruebas de protocolo y Networking.
- Pruebas punta a punta cliente-servidor.
- Validación con lo esperado del notebook de Kaggle.

6.2. División entre integrantes

División tentativa de tareas generales entre integrantes (**Sujeto a modificaciones**).

Sección	Máximo Utrera	Santiago Sevitz	Federico Genaro
Infraestructura	Estructura Go, mensajes (protobuf), wrapper RabbitMQ	Worker base	—
Gateway	API cliente, uploads/responses, batches → colas, resultados → cliente	—	—
Workers	—	Filter, GroupBy, Reducer	Join, Aggregator
Middleware	Docker de RabbitMQ y definición de colas	—	—
Despliegue & Escalado	—	Dockerización de workers	Configuración y Docker-compose, réplicas gateway/aggregator
Pruebas	Protocolo y networking	—	punta a punta, validación con Kaggle

Cuadro 1: División de tareas entre integrantes

7 Referencias

1. Gerald Ooi. (2025). G Coffee Shop Transaction 202307 to 202506 [Dataset]. Kaggle. <https://www.kaggle.com/datasets/geraldooizx/g-coffee-shop-transaction-202307-to-202506>.
2. Gabriel Robles. (2025). FIUBA - Distribuidos 1 - Coffee Shop Analysis [Notebook]. Kaggle. <https://www.kaggle.com/code/gabrielrobles/fiuba-distribuidos-1-coffee-shop-analysis>.