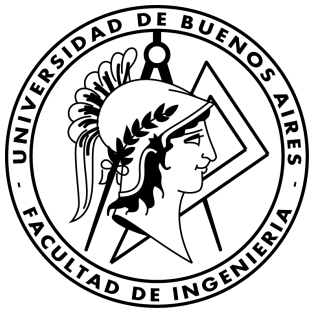

Trabajo Práctico: Escalabilidad

[TA050] Sistemas Distribuidos I
Segundo cuatrimestre de 2025

Alumno	**Nombre y apellido**	**Padrón**	**Mail**
	Baratta, Facundo	104486	fbaratta@fi.uba.ar
	KIM, Daniel	107188	dakim@fi.uba.ar
	Koo, Hangyeol	108401	hkoo@fi.uba.ar

Fecha de entrega: 02/10/2025

Índice

1. Diagramas **2**
1.1. DAG (Directed Acyclic Graph) . . . 2
1.2. Diagrama de Secuencia . . . 3
1.3. Diagrama de Actividades . . . 4
1.4. Diagrama de Robustez . . . 4
1.5. Diagrama de Paquetes . . . 5
1.6. Diagrama de Despliegue . . . 5
1.7. Cómo manejamos los EOF's . . . 6
1.7.1. Cliente → Gateway . . . 6
1.7.2. Gateway → Workers . . . 6
1.7.3. Workers → Gateway . . . 7
1.7.4. Gateway → Cliente . . . 7
1.8. Testing del broker . . . 7
1.8.1. Problema Identificado en los Tests . . . 7
1.9. Anexo . . . 8

1. Diagramas

1.1. DAG (Directed Acyclic Graph)

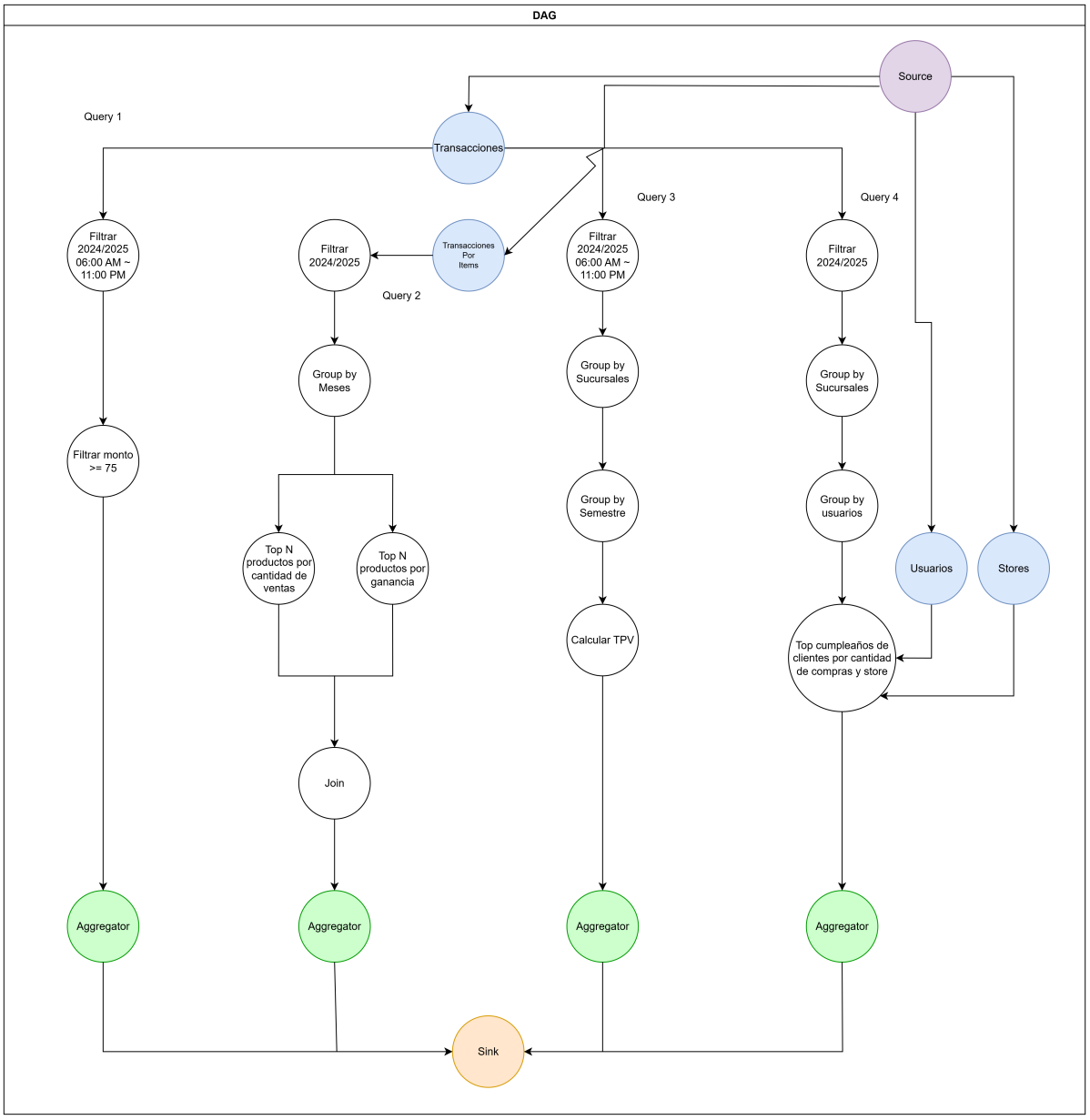

Figura 1: DAG

En el presente diagrama de grafo acíclico es posible ver el flujo general de la resolución de las queries a modo abstracto, con referencia a las fuentes de datos para cada una de ellas:

- Query 1: Transacciones realizadas durante 2024 y 2025 entre las 06:00 AM y las 11:00 PM con monto total mayor o igual a 75.
  - Se filtran las transacciones por año y hora.
  - Se filtran las transacciones por monto.
- Query 2: Productos más vendidos y productos que más ganancias han generado para cada mes en 2024 y 2025.

  - Se filtran las transacciones de ítems por año.
  - Se calculan los top productos agrupados por meses.

- Query 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.
  - Se filtran las transacciones por año y hora.
  - Se calcula TPV por semestre/sucursales

- Query 4: Fecha de cumpleaños de los 3 clientes que han hecho más compras durante 2024 y 2025, para cada sucursal.
  - Se filtran las transacciones por año.
  - Se calculan los top clientes por sucursal.
  - Se agregan los nombres de tiendas y los cumpleaños.

1.2. Diagrama de Secuencia

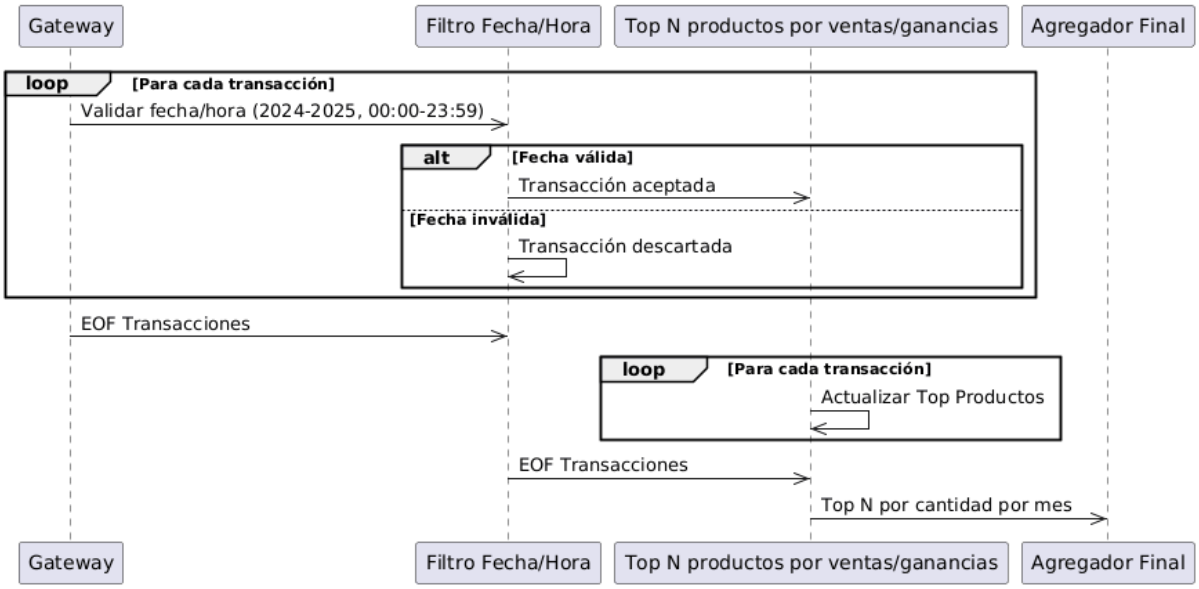

Figura 2: Diagrama de secuencia de Q2

En el presente diagrama es posible observar el flujo de la query 2 (Top Productos por venta/ganancia por mes en 2024/2025) y las acciones de los workers:

- 1. Gateway envía las transacciones para el Filtro de Fecha/Hora
- 2. Las transacciones son validadas y enviadas al siguiente paso en caso que corresponda
- 3. Se van calculando los top productos según cada criterio
- 4. El gateway envía EOF al Filtro, que se propaga hacia Top Productos, donde se da por finalizado la query y se envían los resultados al agregador final.

1.3. Diagrama de Actividades

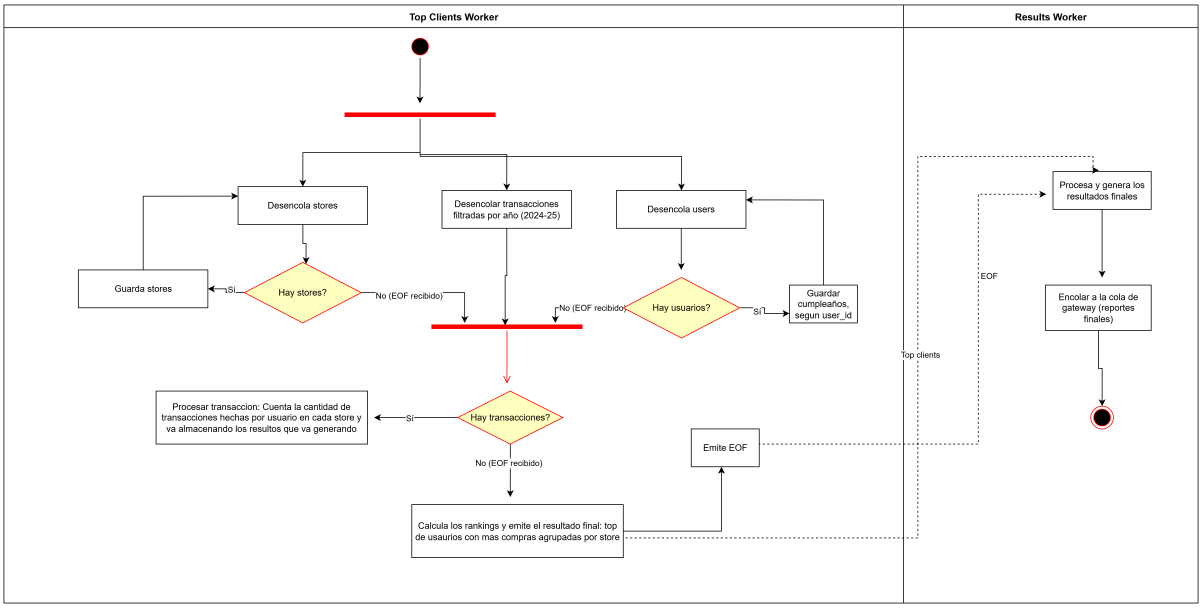

Figura 3: Diagrama de actividades

Este diagrama corresponde a las fechas de cumpleaños de los tres clientes que han hecho más compras durante 2024 y 2025, para cada sucursal. Se puede ver que el worker se encarga de guardar en memoria los stores y usuarios para después contar la cantidad de transacciones hechas por usuario en cada store y va almacenando los resultados que va generando, para luego enviarle los resultados finales al results worker.

1.4. Diagrama de Robustez

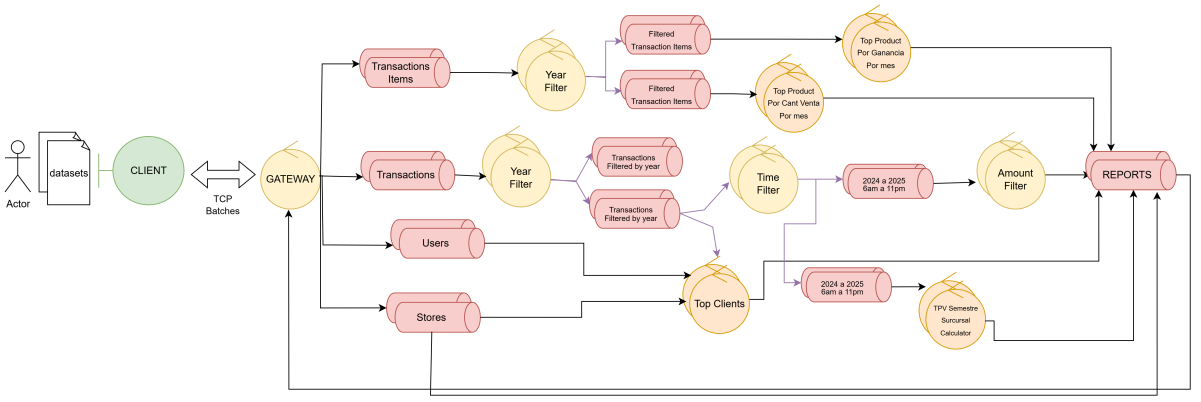

Figura 4: Diagrama de Robustez

En este diagrama se explica el sistema interno, mostrando las colas, los controladores y como interactúan entre ellos. En primer lugar, el gateway recibe el dataset y, adelantándonos al final de todo, recibe el reporte de todos los queries.

Nuestra idea es un sistema de forma "caudal de agua" donde algunos workers (en particular el filtro por año 2024-2025) está repetido, pero esto es porque toma datos de una cola distinta y encola el resultado del procesamiento en otras colas. Las flechas violetas indican que el resultado es duplicado en dos colas que

reciben información idéntica pero son utilizada por controladores que se encargan de queries distintas, desencolando ítems a su necesidad.

Finalmente, hay algunos controladores que tienen un estado interno, ya que encuentran el top productos/clientes/ventas por sucursal, mes, y/o semestre. Se encarga de hacer el computo de la agregación de una agrupación por las columnas necesarias para resolver cada query en particular.

1.5. Diagrama de Paquetes

El diagrama de paquetes se encuentra dividido en distintos elementos lógicos en función de las responsabilidades de cada nodo. Cada módulo tiene su propio protocolo y usan la interfaz de rabbitmq.

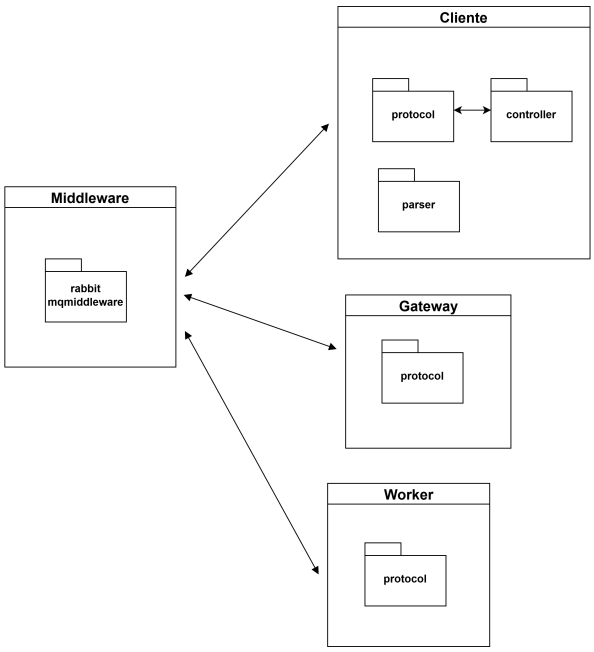

Figura 5: Diagrama de paquetes

1.6. Diagrama de Despliegue

En el diagrama se muestra la distribución de procesos en distintas computadoras. Todos los nodos se comunican a través del middleware RabbitMQ, excepto en el caso de la interacción entre el cliente y el gateway.

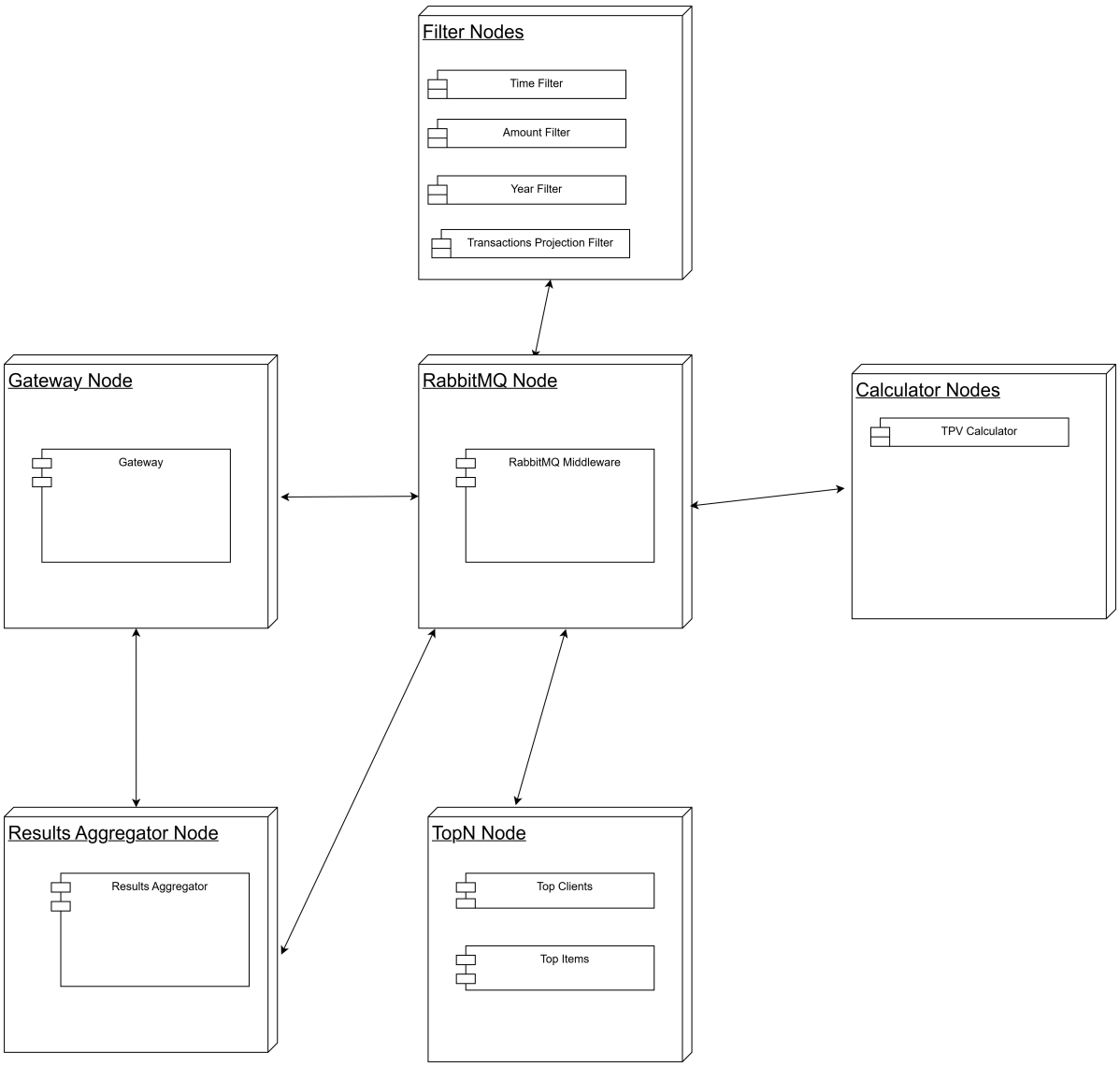

Figura 6: Diagrama de despliegue

1.7. Cómo manejamos los EOF's

1.7.1. Cliente → Gateway

El cliente, una vez que termina de enviar todos los archivos CSV correspondientes a un dataset específico (`users`, `stores`, `menu_items`, `transactions`, `transaction_items`), envía un mensaje **EOF** al gateway indicando el tipo de datos que ha completado. Este EOF incluye un código de respuesta que el gateway debe confirmar antes de proceder.

1.7.2. Gateway → Workers

El gateway, al recibir un EOF del cliente, lo procesa y lo propaga selectivamente a los workers que requieren esa información específica.

- Cuando recibe un EOF de `transactions`, lo propaga a la cadena de procesamiento de transacciones.
- Cuando recibe un EOF de `stores`, lo envía a todos los workers que necesitan información de tiendas.

Una vez que el gateway termina de transmitir todos los datos correspondientes a un dataset, envía un **EOF final** a los workers que estaban procesando esa información.

1.7.3. Workers → Gateway

Los workers, cuando terminan de procesar todos los datos recibidos y detectan el EOF, realizan las siguientes acciones:

1. Completan el procesamiento de cualquier lote pendiente.
2. Generan y envían resúmenes o resultados finales si corresponde.
3. Envían un EOF de confirmación al gateway indicando que han terminado su procesamiento.
4. Detienen su consumo de mensajes de la cola de entrada.
5. Liberan los recursos asociados.

1.7.4. Gateway → Cliente

Finalmente, el gateway recopila todos los EOF de los workers y, una vez que confirma que todo el procesamiento ha terminado, envía un **EOF final** al cliente junto con los resultados procesados, cerrando así el ciclo completo de procesamiento.

Problema detectado. Si existen 2 (o más) réplicas de un worker consumiendo de la misma cola, cada una recibirá el mismo EOF y enviará 2 (o más) EOF's hacia adelante. Esto puede causar problemas de sincronización.

Posibles soluciones.

- Definir un **worker líder** dentro de cada grupo de réplicas, de forma que sólo ese envíe el EOF.
- Asignar a cada réplica su propia cola de salida, y que el gateway consolide los EOF.

1.8. Testing del broker

Los tests del broker se encuentran en el directorio `/tests`. Las instrucciones para correr los tests son:

1. Tener RabbitMQ levantado con Docker.
2. Ejecutar `source venv/bin/activate` para levantar el entorno virtual.
3. Instalar dependencias con `pip install -r requirements.txt`.
4. Correr `pytest tests/test_middleware.py`.

1.8.1. Problema Identificado en los Tests

Durante el desarrollo de los tests para la interfaz del broker, nos encontramos con un problema técnico específico: los tests fallan sin el uso de `time.sleep`, a pesar de que el sistema funciona correctamente en el ambiente de desarrollo dockerizado.

Análisis del problema. En desarrollo dockerizado:

- Los workers se ejecutan continuamente en contenedores (`start_consuming()`).
- Los mensajes se procesan de forma asíncrona y persistente en RabbitMQ.
- Los workers mantienen conexiones estables y están siempre disponibles.

En testing:

- Los consumers se inician en hilos separados de forma síncrona.
- Existe un delay natural entre la configuración del consumer y su disponibilidad.
- Los tests necesitan garantizar que el consumer esté completamente listo antes de enviar mensajes.

Rol de `time.sleep` en los tests. El uso de `time.sleep` asegura que:

- Los consumers estén configurados completamente.
- Las conexiones con RabbitMQ estén establecidas.
- Los bindings de colas/exchanges estén activos.
- No se pierdan mensajes por problemas de timing.

Creemos que el uso de `time.sleep` en los tests está técnicamente justificado porque:

- Es específico del ambiente de testing y no afecta el ambiente dockerizado.
- Garantiza la sincronización correcta entre producers y consumers.
- Simula el comportamiento real de workers siempre activos en contenedores.
- Es una práctica común en el testing de sistemas de mensajería asíncronos.

1.9. Anexo

A continuación se detallan algunos problemas detectados en el sistema implementado al día de hoy, que trataremos en próximas iteraciones:

- En la implementación del worker que calcula el **top de usuarios con más compras**, actualmente este joinea los datos de usuarios y tiendas con las transacciones. Dicho worker es el que hace el cálculo final del top.
- Al inicio habíamos propuesto tener varios workers de **Top 3**, que enviarían su output a un worker que consolidara el top global. Esta idea no pudo ser implementada por falta de tiempo.
- El cliente actualmente imprime el output final de forma incorrecta. Concluimos que debemos automatizar las pruebas contra resultados de Kaggle para evitar comparaciones "a ojo".
- En la demo detectamos que para el output de la tercera query hubo discrepancias en el cálculo del **profit final** para algunos registros.