**Universidad de Ingenieria y Tecnologia**
**Departamento de Ciencias de la Computación**

**Arquitectura de Software**

**Caso de Estudio #14**
**2025 - I**

**Excalidraw del proyecto: Excalidraw**

| | |
|---|---|
| **TEAM** | **Grupo D** |
| **INTEGRANTES** | **Neo Zapata, Juan Sara, Luis Gutierrez** |
| **Puntos** | **20 ptos = 20 ptos Diseño, Fitness Functions y Architecture Test** |

**Keywords:**
ALL

### Caso de Estudio

De su proyecto elija uno de los módulos con mayor complejidad y haga un bosquejo de su diagrama de diseño con énfasis en **UNA** de las siguientes alternativas:

- Availability
- Consistency
- Security

**Entregables**:

## Descripción del proyecto seleccionado.

Sistema que conecta a personas interesados en comprar productos de consumo masivo (supermercados, abarrotes, productos de limpieza, etc.) en grandes cantidades o en grupo, con el fin de obtener precios más económicos. La plataforma facilita la creación de "carritos de compra" comunitarios, donde varias personas pueden unirse para realizar un pedido único a un proveedor, aprovechando descuentos por volumen. Posteriormente, el sistema gestiona la distribución interna de los productos entre los participantes del grupo.

- **Reducción de Costos y Ahorro para el Solicitante:** Los usuarios se benefician de precios más bajos al comprar al por mayor o en grupo.
- **Fomento de la Economía Colaborativa Local:** Incentiva la cooperación entre vecinos y fortalece los lazos comunitarios.

- **Eficiencia en Logística:** Optimiza los envíos al consolidar múltiples pedidos en uno solo, reduciendo el tráfico y el impacto ambiental.

En resumen, un vecino crea o se une a un "carrito de compra grupal" para un tipo de producto o proveedor específico. La plataforma gestiona la acumulación de pedidos, el procesamiento del pago grupal, y la notificación para la distribución de los productos una vez recibidos.

**Requerimientos no funcionales clave**

- **Escalabilidad Geográfica:** La plataforma debe crecer distrito por distrito, ciudad por ciudad, con incremento rápido de usuarios y volumen de pedidos.
- **Disponibilidad y Tolerancia a Fallos:** La aplicación debe estar operativa casi todo el tiempo para permitir la creación y unión a carritos, así como el seguimiento de pedidos.
- **Seguridad y Confianza:** Verificación de identidad de usuarios y proveedores, protección de datos de pago y personales, detección de fraude en la creación de grupos o distribución.
- **Mantenibilidad:** Los equipos independientes deben poder evolucionar módulos sin romper todo el sistema, dado el potencial de crecimiento y nuevas funcionalidades.
- **Rendimiento:** Tiempos de respuesta razonables, especialmente en la búsqueda de grupos, la adición de productos a carritos y la consolidación de pedidos.
- **Observabilidad:** Trazabilidad de eventos de compra, logs estructurados de transacciones y métricas de rendimiento.
- **Costo Óptimo:** Especialmente al arrancar en una ciudad; permitir crecimiento controlado de costos a medida que aumenta la base de usuarios y el volumen de transacciones.

## Elección de la alternativa a enfatizar - Módulo de compras grupales - Availability

La alternativa seleccionada es **‘Availability’** porque el critical path del proyecto se centra en que el usuario sea capaz de seleccionar sus productos, hacer su carrito de compras sin problema y realizar sus pedidos.

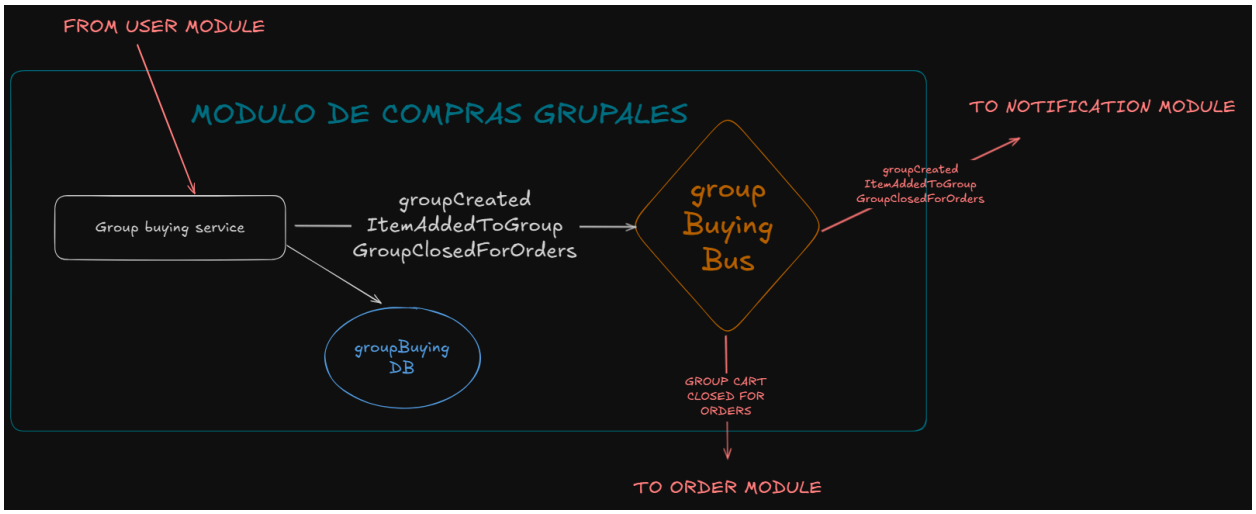
Identificamos que el flujo más crítico para el éxito de nuestra plataforma es el siguiente:

- Un usuario crea un carrito grupal
- Otros usuarios se unen y agregan productos
- Se alcanza el mínimo requerido -> se cierra el carrito
- Se genera un pedido consolidado
- Se inicia y completa el pago grupal
- Se gestiona la logística y distribución
- Se notifica a los usuarios para recojo o entrega

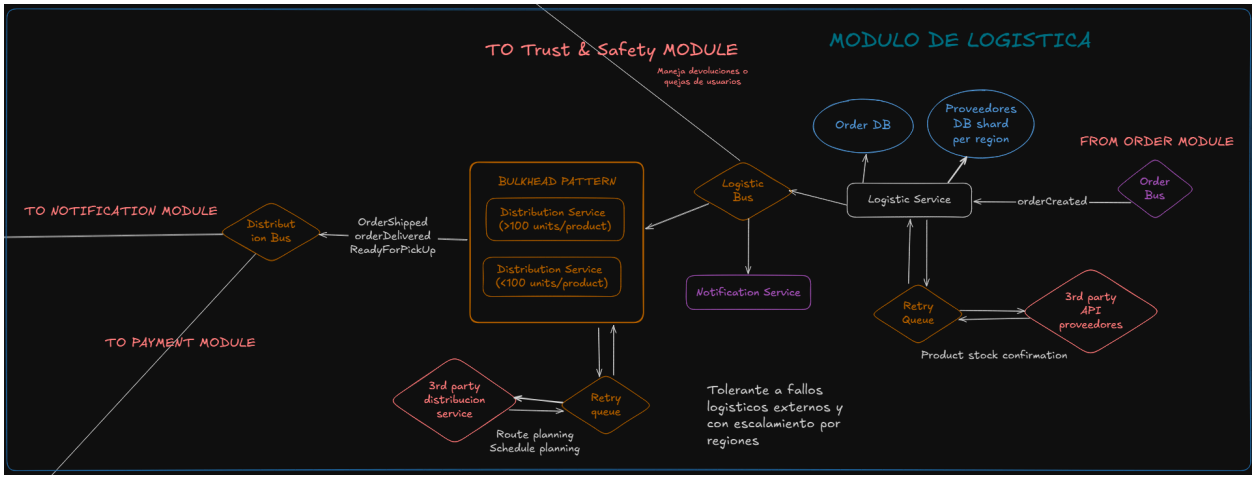
Este flujo involucra: Group Buying Service -> Order Service -> Payment Service -> Logistics Service -> Notification Service

Sin embargo el módulo más crítico de todos es el **módulo de compras grupales**

### Modulo original




### Módulo optimizado para Availability




El módulo de logística comienza en el servicio de logística después de completar la orden. Logistic service accede a la orden DB y se comunica con la API de proveedores para comprobar si los productos solicitados están disponibles y hay stock disponible. Caso contrario se comunica con el usuario a través del notification service y puede remover el producto del carrito o cancelar la compra y se realiza la devolución (toda esta gestión es manejada por el Trust & Safety module). Una vez se notifica al usuario y se procede, el distribution service usa el bulkhead pattern para darle prioridad a carritos de compra con un volumen alto (por ejemplo, +100 productos), luego se comunica con el servicio de distribución de productos para coordinar los puntos de recojo, el planea de rutas y los horarios de entrega. Finalmente, se envía toda esta información al usuario a través del notification service y se hace el pago efectivo (payment service).

## Fitness Functions ( Usar github actions o similares )

Para asegurar que la arquitectura evolucione correctamente, proponemos funciones automatizadas que midan:

- Performance -> latencia media por endpoint < 250ms
- Reliability -> Ratio de 2xx > 98% en pedidos y pagos
- Security -> Login invalido > 3 -> alerta de intento sospechoso
- Modularity -> Ningún módulo debe tener dependencias circulares (not scripted)
- Load -> Throughput mínimo de 100 req/s sostenido con bajo error rate.

El sistema no mide la carga (**load**) por “número de usuarios”, sino por:

- cantidad de carritos grupales activos simultáneamente
- Volumen de productos agregados por minuto
- Número de eventos asíncronos (ej. pagos iniciados, pedidos cerrados, notificaciones enviadas)

## Architecture Tests (Hacer POC simple + Architecture Tests)