1. Diseño de Arquitectura

Teniendo en cuenta que se aconseja usar una arquitectura de Microservicios se proponen la siguiente solución.

Tecnologías principales:

TECNOLOGÍA	*USO*
.NET 8	Desarrollo de todos los microservicios (Minimal APIs)
OCELOT	API Gateway con enrutamiento, rate limiting y resiliencia
KEYCLOAK	Identity Provider (IdP) para autenticación y autorización (OIDC/OAuth2)
REDIS	Caché distribuido y rate limiting
POLLY	Circuit Breaker, Retry, Timeout y Fallback
SERILOG + SEQ	Logging estructurado y observabilidad
DOCKER COMPOSE	Orquestación y despliegue local

Como se hace referencia en la tabla de tecnologías a usar **Keycloak** será el gestor de autorización y autenticación, por ende, lo excluyo del microservicio de usuarios.

Adicional propongo hacer la implementación de Resiliencia y Observabilidad con:

- Circuit Breaker, Retry, Timeout, fallback con polly en Ocelot
- Rate Limiting con Redis cache
- Logs estructurados con Serilog con Seq.

Componentes principales:

COMPONENTE	*RESPONSABILIDAD*
APIGATEWAY	Punto único de entrada, enrutamiento, seguridad y resiliencia
USERSERVICE	Gestión del **perfil de negocio** del usuario
ORDERSERVICE	Ciclo de vida completo del pedido
PAYMENTSERVICE	Procesamiento de pagos con métodos variables

Protocolos de comunicación:

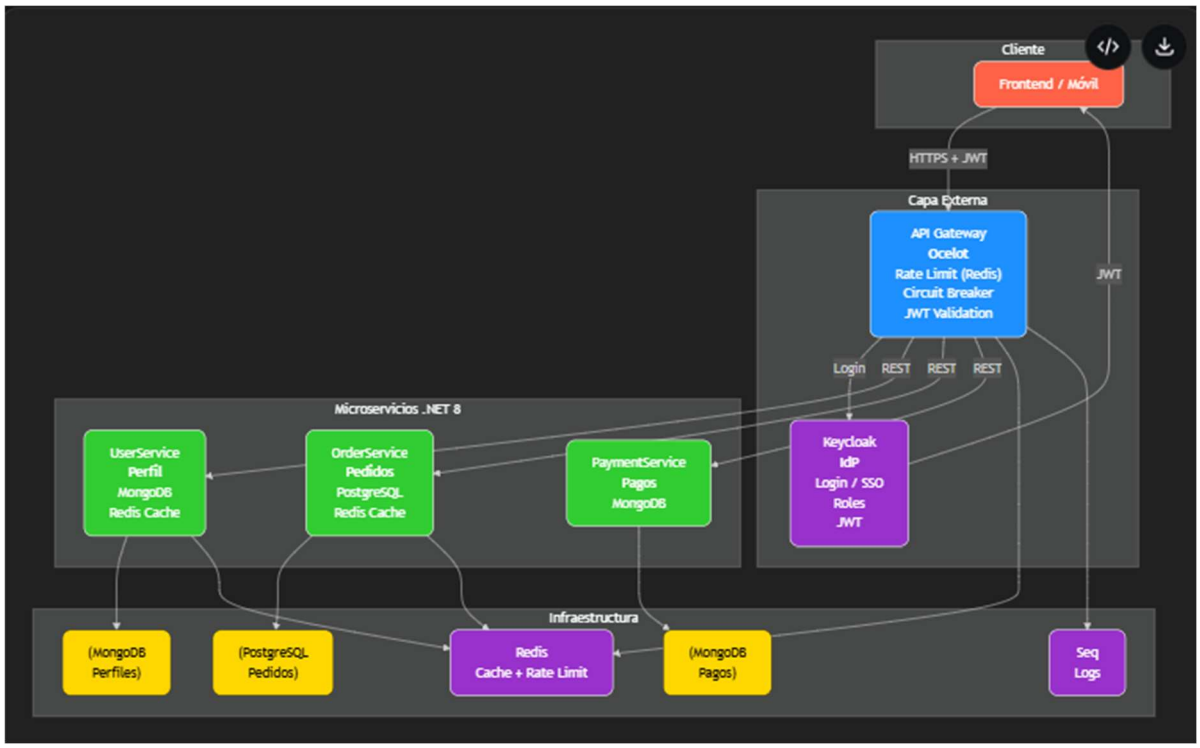
TIPO	*PROTOCOLO*	*USO*
EXTERNA	**HTTPS + REST**	Cliente → API Gateway
INTERNA	**HTTP**s + REST	Gateway → Microservicios
EVENTUAL	**Kafka** o R**abbitmq (**Posible implementación**)**	Eventos asíncronos (Procesamiento de pagos → estado de la solicitud)

MICROSERVICIO	*MOTOR*	*¿POR QUE USAR?*
USERSERVICE	**MongoDB**	Esquema flexible, alta escritura, datos anidados
ORDERSERVICE	**PostgreSQL**	Relaciones complejas, transacciones ACID
PAYMENTSERVICE	**MongoDB**	Esquema flexible, alta escritura, Métodos de pago variables

Dominios:

DOMINIO	REPONSABILIDAD
USUARIOS	Gestión del perfil del usuario: nombre, dirección, teléfono, preferencias, historial de navegación.
PEDIDOS	Ciclo de vida del pedido: creación, modificación, estado, productos (items), envío.
PAGOS	Procesamiento de pagos, reembolsos, estado del pago, métodos de pago

Diagrama de arquitectura:



2. **Gestión de cambios entre servicios**

La solución se eligió pensando en garantizar que se pase de un monolito frágil a un sistema robusto asegurando una fácil escalabilidad y mantenimiento adicional ofreciendo:

Desacoplamiento: se proponen tres dominios independientes (Usuarios, Pedidos, Pagos) con base de datos por servicio y Keycloak como gestor de identidad externo;

Resiliencia: Ocelot con Polly aplican Circuit Breaker, Retry y Fallback en 3 líneas; un pico de tráfico o caída de alguno de los microservicios no se verán afectados los demás.

Velocidad: Redis cachea perfiles y pedidos en un tiempo muy bajo adicional protege con rate-limit, en el caso MongoDB da flexibilidad para añadir campos sin migraciones; PostgreSQL asegura transacciones ACID.

Control de logs : Serilog con Seq te permite filtra por logs y ver el trace completo.

3. ***Protocolos de comunicación:***

El uso de rest es recomendado para este tipo de solución ya que permite simplicidad en la implementación, adicional **Ocelot** es diseñado para usar rest así poder aprovechar el enrutamiento, **QoS**, **rate-limit**, **autenticación JWT**, **logging** etc. Y eso sin hacer ninguna configuración rara o compleja.

Podría agregar que también la elijo por que la implementación de Polly + Ocelot aplican Circuit Breaker, Retry y Timeout son muy sencillas y blindaran la solución entregada.

Kafka o RabbitMq podrían usarse en casos de uso específicos por lo cual también sería incluido en la implementación.

Conclusión:

Con la solución planteada se asegura unos microservicios desacoplados, tolerante a fallos, observable, con mejoras claras en disponibilidad, escalabilidad y mantenibilidad (menos costos de mantenimiento), además que se asegura el uso de tecnologías que nos simplificaran mucho el desarrollo y la implementación.