Universidad Autónoma de Chiapas

Microservicios de Usuario - Pedido

Materia: Taller de Desarrollo 4

Alumno: Edgar Yahir Altunar Gómez

Carrera: Ingeniería en Desarrollo y Tecnologías de Software

Grado y Grupo: 6 M

Docente: Dr. Luis Gutiérrez Alfaro

Tuxtla Gutiérrez, Chiapas a 13 de febrero de 2026

Índice

1. Arquitectura del Sistema

El proyecto sigue estrictamente la **Arquitectura Hexagonal**. Esta decisión se tomó para proteger el núcleo de la aplicación (el dominio) de las dependencias externas. El código se organizó en tres capas concéntricas:

1.1. Estructura de Carpetas

Para cada microservicio, se respetó la siguiente jerarquía:

```
src/
 domain/          (Entidades y Modelos)
 application/     (Puertos y Servicios)
    ports/
    services/
 infrastructure/  (API y Adaptadores)
    api/
    adapters/
```

1.2. Explicación de los Componentes

1.2.1. Capa de Dominio (Domain)

Aquí residen los objetos fundamentales del negocio. No tienen ninguna dependencia externa.

- **Entidades:** Se definieron las clases `User` (con id, nombre, email) y `Order` (con id, items, total, estado).

1.2.2. Capa de Aplicación (Application)

Contiene las reglas de negocio y los casos de uso.

- **Puertos (Ports):** Son interfaces abstractas (clases abstractas en Python) que definen *qué* necesita el sistema para guardar datos, pero no *cómo* hacerlo. Ejemplo: `UserRepositoryPort`.
- **Servicios (Services):** Implementan la lógica de negocio. El `UserService` recibe una petición, valida reglas de negocio si es necesario, y llama al puerto correspondiente.

1.2.3. Capa de Infraestructura (Infrastructure)

Es la capa más externa y contiene los detalles técnicos.

- **Adaptadores (Adapters):** Implementan los puertos. Para esta práctica, creamos un `InMemoryReposito` que simula una base de datos usando listas de Python. Esto permite probar el sistema sin necesidad de instalar un servidor SQL real.
- **API:** Contiene los *Routers* de FastAPI. Es la puerta de entrada HTTP que recibe las peticiones del cliente (GET, POST, PUT, DELETE) y se las pasa a los servicios.

2. Microservicios Desarrollados

2.1. Microservicio de Usuarios (Puerto 8001)

Encargado de la gestión de la identidad.

- **Entidad:** Usuario (`idusuario`, `nombre`, `email`).
- **Funcionalidad:** Permite registrar nuevos usuarios, consultar su información, modificar su nombre/email y eliminarlos del sistema.

2.2. Microservicio de Pedidos (Puerto 8002)

Encargado de la gestión transaccional del restaurante.

- **Entidad:** Pedido (`id_pedido`, `id_usuario`, `items`, `total`, `estado`).
- **Funcionalidad:** Permite crear comandas vinculadas a un usuario, actualizar el estado del pedido (ej. de "Pendiente.ª .Entregado") y calcular totales.

3. Flujo de Trabajo

El flujo de datos para una operación típica, como «Crear un Usuario» , sigue estos pasos estrictos para mantener la arquitectura limpia:

1. **Petición HTTP:** El cliente envía un POST con el JSON del usuario al puerto 8001.
2. **Infrastructure (Router):** FastAPI recibe la petición y valida que el JSON cumpla con el modelo Pydantic.

3. **Application (Service):** El Router llama a `UserService.create_user()`. El servicio no sabe si se guardará en SQL o en memoria, solo sabe que debe guardarlo.

4. **Application (Port):** El servicio invoca al método `create()` definido en la interfaz `UserRepositoryPo`

5. **Infrastructure (Adapter):** La implementación concreta (`InMemoryRepository`) se ejecuta, guardando el objeto en la lista en memoria.

6. **Respuesta:** El dato viaja de vuelta hacia arriba hasta que el Router devuelve un código HTTP 201 (Created) al cliente.