



hiberus

www.hiberus.com

PROYECTO KAFKA

E-commerce Mate

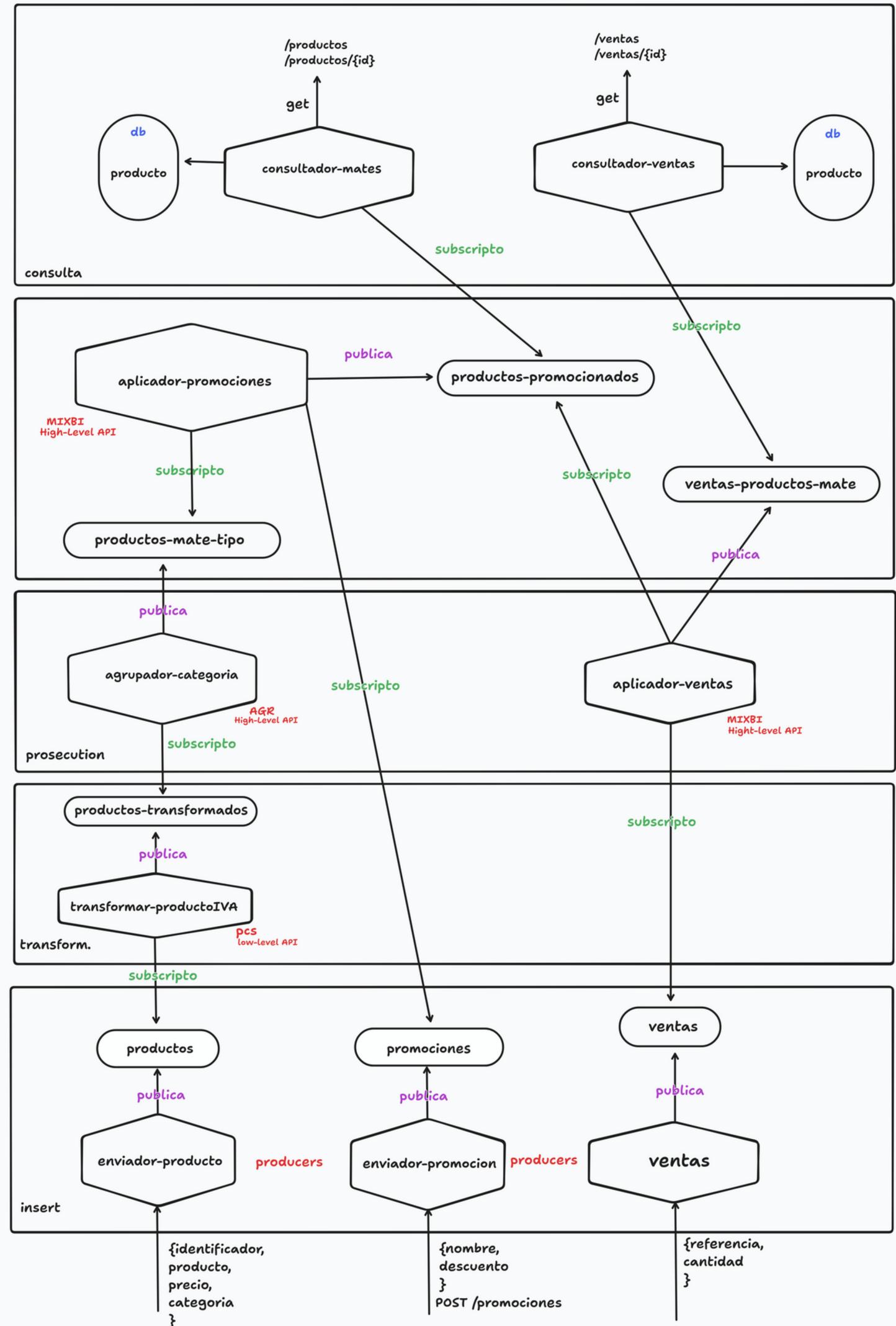
• QUE PROBLEMA FUNCIONAL RESUELVE?

E-commerce Mate es una aplicación diseñada para facilitar la gestión integral de productos y ventas para comerciantes que venden mates u otros productos similares.

Resuelve los siguientes problemas:

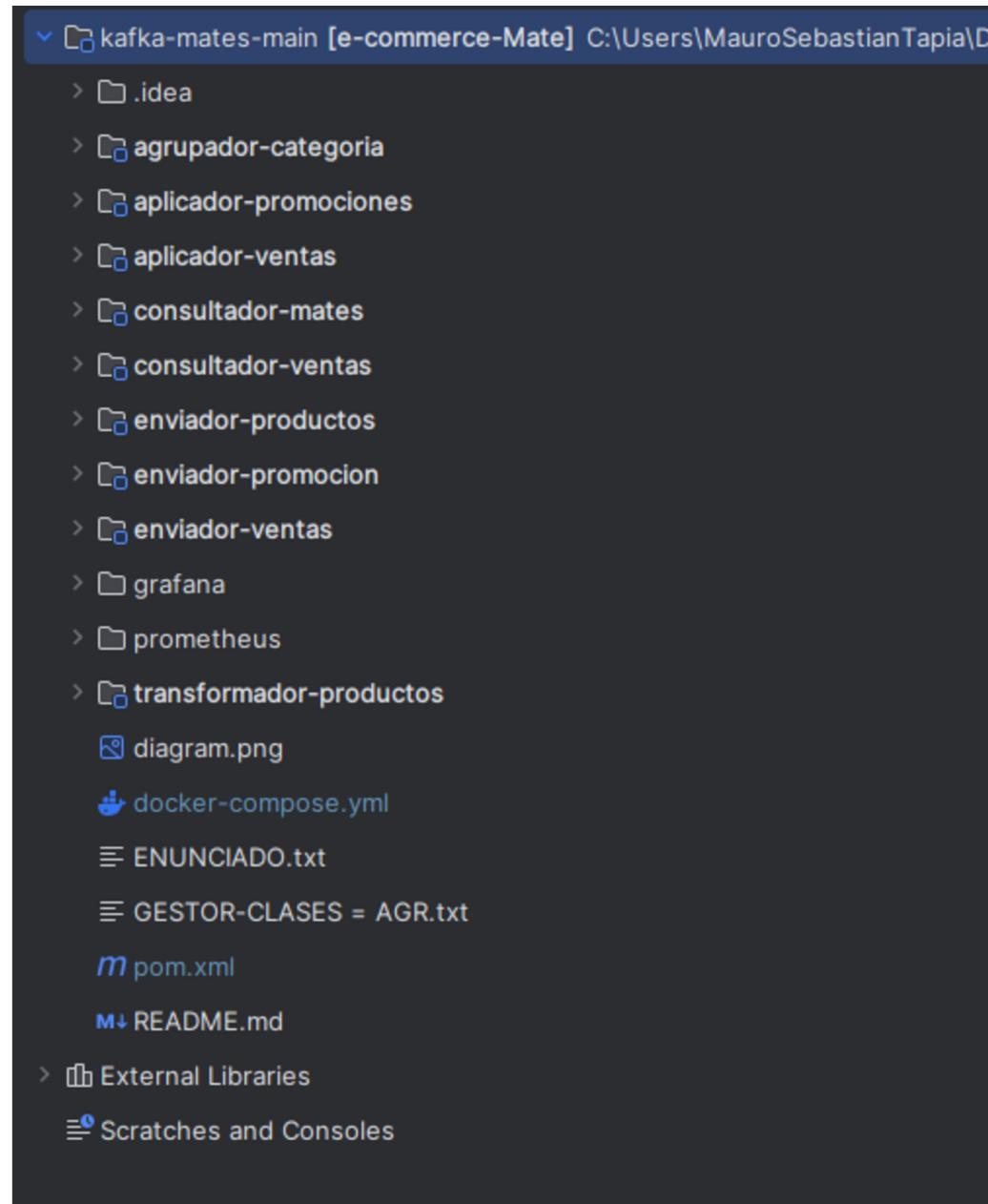
- **Cálculo automático del IVA:** Elimina la necesidad de calcular manualmente el IVA en los productos, reduciendo errores y ahorrando tiempo.
- **Gestión de promociones inteligentes:** Permite aplicar promociones automáticamente a productos que cumplen con las condiciones establecidas, optimizando la experiencia del comerciante.
- **Consulta rápida de inventarios:** Ofrece la posibilidad de consultar de forma sencilla qué productos (mates) están disponibles.

DIAGRAMA

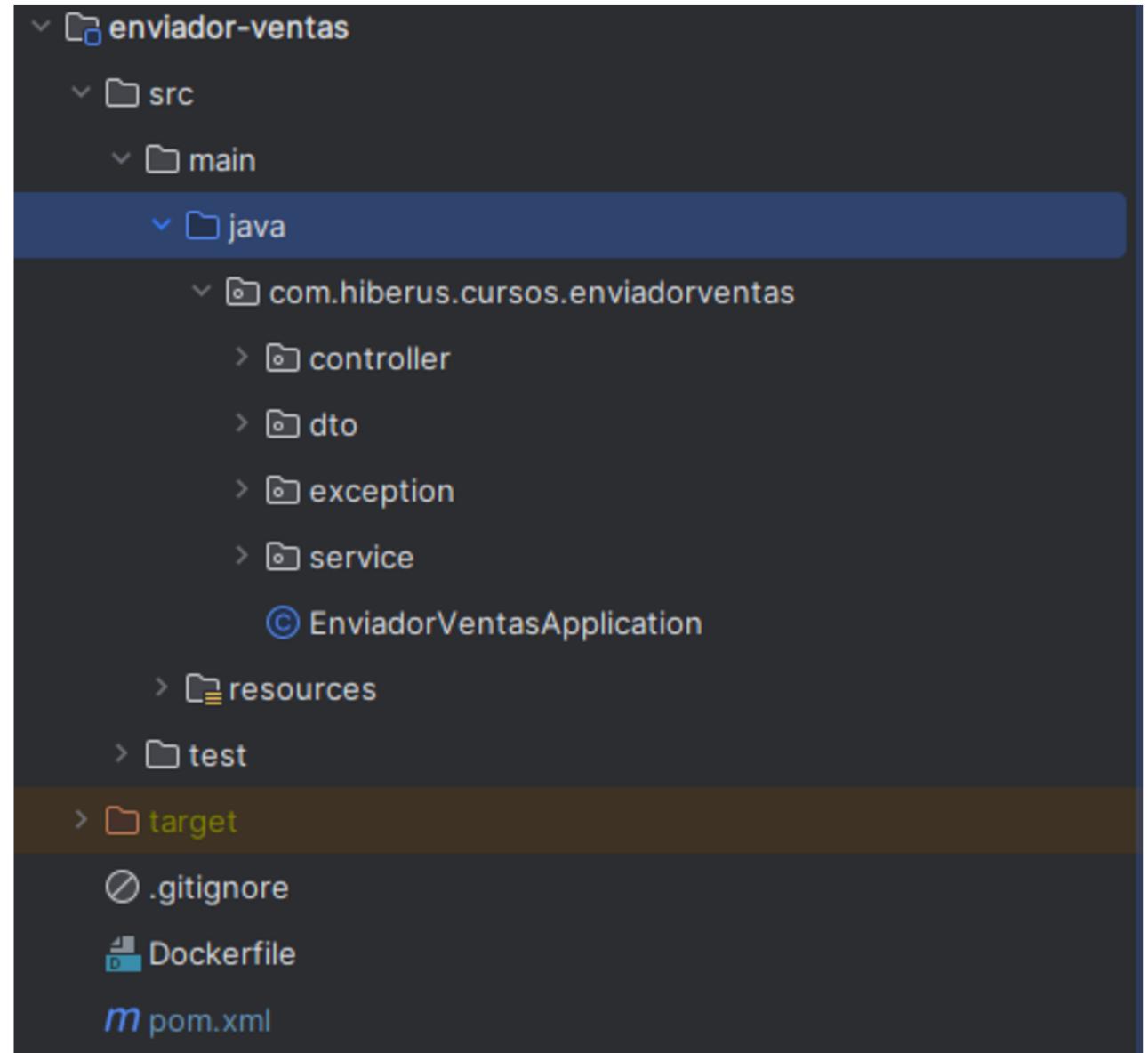


DEMOSTRACION PROYECTO

Microservicios independientes



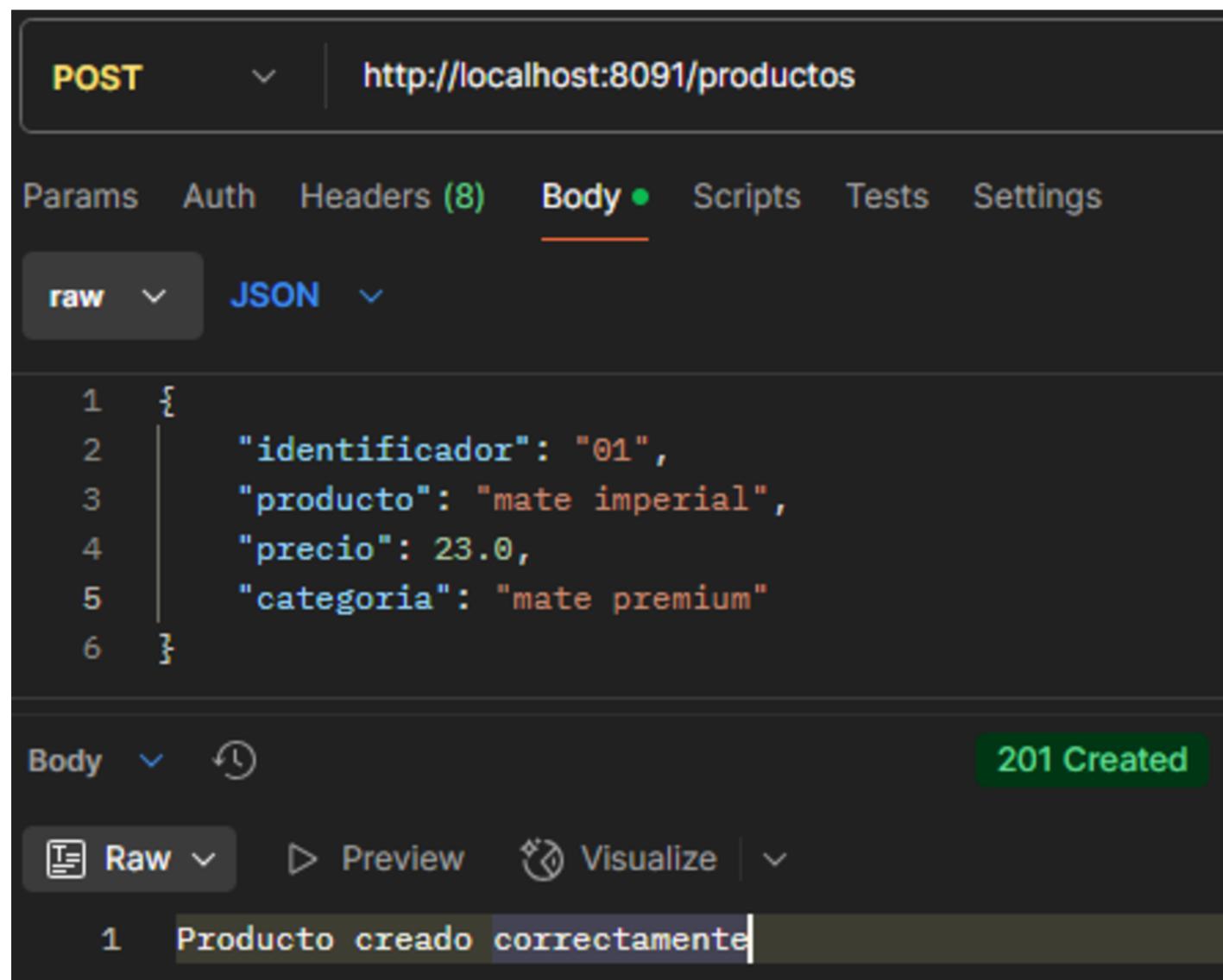
Patrón de diseño de capas



Enviador productos

Para enviar un producto necesitamos completar sus atributos

Si todos los campos son completados sin dejar ninguno vacío o con valores negativos en precio, se creará correctamente



POST <http://localhost:8091/productos>

Params Auth Headers (8) **Body** Scripts Tests Settings

raw JSON

```
1 {  
2   "identificador": "01",  
3   "producto": "mate imperial",  
4   "precio": 23.0,  
5   "categoria": "mate premium"  
6 }
```

Body Raw Preview Visualize 201 Created

1 Producto creado correctamente

Es el encargado de calcular el precio de los productos incluyendo el IVA (Impuesto al Valor Agregado) y enviar el resultado al tópico correspondiente en Kafka.

Lee del topic productos y escribe en el topic productos-transformados

input

```
{"identificador": "10",
"producto": "Bombilla"
"precio": 50.00,
"categoria": "Accesorios"}
```

output

```
{"identificador": "10",
"producto": "Bombilla",
"precioConImpuesto": 61.0,
"categoria": "Accesorios"}
```

```
@Slf4j
@Service
public class TransformProductIVAService {

    2 usages
    @Value("${environment.productos-impuesto-topic}")
    private String productosImpuestoTopic;
    1 usage
    @Autowired
    private KafkaTemplate<ProductoKey, ProductoConImpuestoValue> kafkaTemplate;

    1 usage
    private static final double IVA_RATE = 0.21;
    1 usage  Mauro Sebastian Tapia
    private double redondear(double valor) {
        BigDecimal bd = new BigDecimal(valor).setScale( newScale: 0, RoundingMode.HALF_UP);
        log.info("Valor original: {}, Valor redondeado: {}", valor, bd.doubleValue());
        return bd.doubleValue();
    }

    1 usage  Mauro Sebastian Tapia
    public void transformar(ProductoKey key, ProductoValue value) {
        double precioBase = value.getPrecio();
        double precioConIVA = redondear( valor: precioBase * (1 + IVA_RATE));

        ProductoConImpuestoValue productoConImpuesto = ProductoConImpuestoValue.newBuilder()
            .setIdentificador(value.getIdentificador())
            .setProducto(value.getProducto())
            .setPrecioConImpuesto(precioConIVA)
            .setCategoria(value.getCategoría())
            .build();

        log.info("Enviando producto con impuesto al topic '{}': Key -> {}, Producto -> {}, Precio con IVA -> {}",
            productosImpuestoTopic, key, productoConImpuesto.getProducto(),
            productoConImpuesto.getPrecioConImpuesto());

        kafkaTemplate.send(productosImpuestoTopic, key, productoConImpuesto);
    }
}
```

El Aggregator es un componente que permite agrupar productos por categoría. Combina productos nuevos con productos previamente agregados en una misma categoría, manteniendo una lista actualizada de productos por categoría.

**Lee del topic productos-transformados
y escribe en productos-mate-tipo**

```

@Component
public class Aggregator implements org.apache.kafka.streams.KStream<AgrupadorCategoriaKey,
    ProductoPromocionadoValue, AgrupadorCategoriaValue> {
    ▲ Mauro Sebastian Tapia
    @Override
    public AgrupadorCategoriaValue apply(AgrupadorCategoriaKey agrupadorCategoriaKey,
        ProductoPromocionadoValue productoPromocionadoValue,
        AgrupadorCategoriaValue agrupadorCategoriaValue) {

        List<Producto> productosExistentes = agrupadorCategoriaValue != null && agrupadorCategoriaValue.getProductos() != null
            ? agrupadorCategoriaValue.getProductos()
            : new ArrayList<>();

        List<Producto> productosActualizados = productosExistentes.stream()
            .filter(c -> !productoPromocionadoValue.getProducto().equals(c.getProducto()))
            .collect(Collectors.toList());

        productosActualizados.add(createProducto(productoPromocionadoValue));

        return AgrupadorCategoriaValue.newBuilder()
            .setProductos(productosActualizados)
            .build();
    }

    1 usage ▲ Mauro Sebastian Tapia
    private Producto createProducto(ProductoPromocionadoValue productoPromocionadoValue){
        return Producto.newBuilder()
            .setIdentificador(productoPromocionadoValue.getIdentificador())
            .setProducto(productoPromocionadoValue.getProducto())
            .setPrecioConImpuesto(productoPromocionadoValue.getPrecioConImpuesto())
            .setPrecioPromocionado(productoPromocionadoValue.getPrecioPromocionado())
            .setPromocionTimestamp(productoPromocionadoValue.getPromocionTimestamp())
            .build();
    }
}

```

MIXBI

Es el encargado de procesar flujos de datos en Kafka Streams para aplicar promociones a productos. Combina la información de productos agrupados por categoría y promociones disponibles para generar un flujo de productos promocionados.

**Lee de dos topics
y escribe en uno**

```
topic-in-0: productos-mate-tipo
topic-in-1: promociones
topic-out-0: productos-promocionados
```

1-Input:

Recibe dos flujos de datos (KStream):

Productos agrupados:

 Información de productos con el precio ya calculado con IVA y agrupados por categoría

Promociones:

 Detalles de descuentos asociados a categorías de productos.

2-Transformacion:

Los datos se reorganizan utilizando la categoría como clave común:

 Productos por Categoría: Se crea un KTable para los productos.

 Promociones por Categoría: Se crea un KTable para las promociones.

Se realiza un outerJoin para combinar los productos y las promociones:

Si no hay promoción para un producto, el precio final es igual al precio con IVA.

Si hay una promoción, se calcula el precio promocionado aplicando el descuento.

3-Cálculo del Precio Promocionado:

Si la promoción está disponible: Aplica la promoción

Si no hay promoción:

 El precio promocionado es igual al precio con IVA.

4-Output:

Publica un flujo resultante (KStream) de productos promocionados:

 Incluye identificador, nombre, categoría, precio con IVA, precio promocionado y marca un timestamp de la promoción.

Consultas mate

Traemos todos los mates

GET http://localhost:8096/productosMate

Params Auth Headers (6) Body Scripts Tests Settings

Query Params

Key	Value

Body { } JSON ▾ 200 OK

```

2 {
3   "categoria": "mate premium",
4   "identificador": "01",
5   "producto": "mate imperial",
6   "precioConImpuesto": 28.0,
7   "precioPromocionado": 28.0
8 },
9 {
10  "categoria": "mate basico",
11  "identificador": "02",
12  "producto": "mate imperial",
13  "precioConImpuesto": 18.0,
14  "precioPromocionado": 18.0
15 }
16 ]

```

Redondeo de precio:

```

private double redondear(double valor) {
    BigDecimal bd = new BigDecimal(valor).setScale( newScale: 0, RoundingMode.HALF_UP);
    log.info("Valor original: {}, Valor redondeado: {}", valor, bd.doubleValue());
    return bd.doubleValue();
}

1 usage  ↗ Mauro Sebastian Tapia *
public void transformar(ProductoKey key, ProductoValue value) {
    double precioBase = value.getPrecio();
    double precioConIVA = redondear( valor: precioBase * (1 + IVA_RATE));

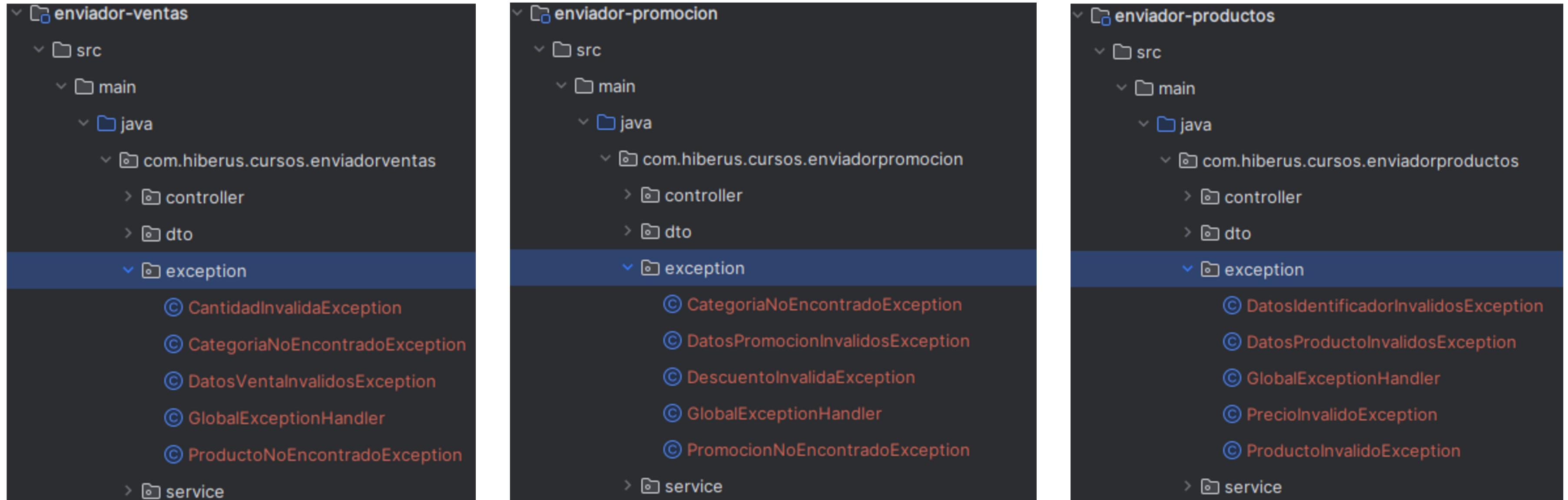
    ProductoConImpuestoValue productoConImpuesto = ProductoConImpuestoValue.newBuilder()
        .setIdentificador(value.getIdentificador())
        .setProducto(value.getProducto())
        .setPrecioConImpuesto(precioConIVA)
        .setCategoria(value.getCategoría())
        .build();

    log.info("Enviando producto con impuesto al topic '{}': Key -> {}, Producto -> {}, Precio con IVA -> {}",
            productosImpuestoTopic, key, productoConImpuesto.getProducto(), productoConImpuesto.getPrecioConImpuesto());
    kafkaTemplate.send(productosImpuestoTopic, key, productoConImpuesto);
}

```

Excepciones

Todos los productores tienen excepciones



DEMOSTRACION

POST http://localhost:8091/productos

Params Auth Headers (8) Body Scripts Tests Settings

raw JSON

```
1 {  
2   "identificador": "02",  
3   "producto": "",  
4   "precio": 15.0,  
5   "categoria": "mate premium"  
6 }
```

Body Raw Preview Visualize

1 El nombre del producto no puede ser nulo o vacío

400 Bad Request

POST http://localhost:8090/promocion

Params Auth Headers (8) Body Scripts Tests Settings

raw JSON

```
1 {  
2   "nombre": "black friday 2",  
3   "descuento": -1,  
4   "categoria": "mate"  
5 }
```

Body Raw Preview Visualize

1 La cantidad debe ser igual o mayor a cero

400 Bad Request

POST http://localhost:8097/ventas

Params Auth Headers (8) Body Scripts Tests Settings

raw JSON

```
1 {  
2   "categoria": "",  
3   "identificador": "06",  
4   "cantidad": 1,  
5   "producto": "mate 06"  
6 }
```

Body Raw Preview Visualize

1 La categoría no puede ser nula o vacía

400 Bad Request

KAFKA

- Los topics estan configurado para que se creen automaticamente
- Configuración de los productores y consumidores para que utilicen 2 brokers
- Por qué usar múltiples brokers?: Cuando un broker se sobrecarga o falla, el sistema sigue funcionando correctamente gracias a la replicación de particiones entre los brokers.

The screenshot shows two parts of the Redpanda UI. The left part is the 'Overview' page, which displays cluster statistics: Cluster Status (Running, 2.81 MiB), Cluster Version (v2.7), Brokers Online (2), Topics (20), and Replicas (81). The right part is the 'Broker Details' section, showing two brokers: Broker ID 1 is Running and 1.44 MiB in size; Broker ID 2 is Running and 1.36 MiB in size.

The screenshot shows the 'Topics' page of the Redpanda UI. It displays three key metrics: Total topics (17), Total partitions (17), and Total replicas (17).

Total topics	Total partitions	Total replicas
17	17	17

PYTHON

Desarrollé un script que consulta productos y ventas desde la base de datos, procesa la información y la exporta a un archivo XLSX. Además, implementé una interfaz en React que muestra los datos en tablas interactivas, permite recargar la información en tiempo real y ofrece la opción de descargar el reporte mediante un botón 'Generar Reporte XLSX'

Categoría	Producto	Precio con Impuesto	Precio Promocionado
mate	mate torpedo	24	24
mate	mate imperial	24	24
accesorios	matera	24	21.6
accesorios	termo stanley	24	21.6

Categoría	Producto	Precio	Cantidad
mate	mate torpedo	24	2
accesorios	matera	24	2

[Recargar Productos](#)
 [Recargar Ventas](#)

[GENERAR REPORTE XLSX](#)

	A	B	C	D	E	F
1	identificador	categoria	producto	precioConImpu	precioPromocion	promocionTim
2	01	mate	mate torpedo	24	24	1,738,269,208,02
3	03	mate	mate imperial	24	24	1,738,269,208,02
4	02	accesorios	matera	24	21.60	1,738,331,957,34
5	04	accesorios	termo stanley	24	21.60	1,738,331,957,34

[Productos](#)
 [Ventas](#)
 [Estadísticas](#)

Prometheus y Grafana

The screenshot shows the Prometheus web interface with the title "Prometheus". The top navigation bar includes "Query", "Alerts", and a "Status > Target health" dropdown. Below the navigation are filters for "Select scrape pool", "Filter by target health", and "Filter by endpoint or labels". The main list displays the health status of ten targets:

- aplicador-promociones: 1 / 1 up (green)
- aplicador-ventas: 1 / 1 up (green)
- consultador-mates: 1 / 1 up (green)
- consultador-ventas: 1 / 1 up (green)
- enviador-productos: 1 / 1 up (green)
- enviador-promocion: 1 / 1 up (green)
- enviador-ventas: 1 / 1 up (green)
- node_exporter: 1 / 1 up (green)
- prometheus: 1 / 1 up (green)
- transformador-productos: 1 / 1 up (green)

