UNIVERSIDAD DE LAS FUERZAS ARMADAS

“ESPE”

Integrantes: Elian Collaguazo

Nrc: 29100

Fecha: 21/12/2025

Tema: Proyecto

1. Resumen

Proyecto integral de e-commerce para tienda de ropa implementado con arquitectura microservicios Spring Boot + React, destacando por su incorporación de programación reactiva mediante Project Reactor. El sistema gestiona productos, categorías, clientes y pedidos con un backend robusto en Java 17 y Spring Boot 3.5.7, frontend moderno en React 19.2.0, y capacidades de despliegue Dockerizado. La innovación principal reside en la implementación de flujos reactivos con Subscriber personalizado y backpressure para procesamiento asíncrono de pedidos e inventario, demostrando patrones avanzados de programación reactiva en un contexto comercial real.

2. Introducción

El comercio electrónico moderno exige sistemas escalables, responsivos y resilientes capaces de manejar múltiples transacciones concurrentes. Este proyecto aborda el desafío mediante la implementación de una tienda de ropa completa que combina tecnologías establecidas con patrones reactivos emergentes. La arquitectura tradicional Spring Boot proporciona el fundamento para operaciones CRUD y gestión de estado, mientras que la capa reactiva añade capacidades de procesamiento no bloqueante esenciales para aplicaciones de alto rendimiento. El sistema no solo cumple con los requisitos funcionales de una plataforma e-commerce, sino que también sirve como caso

de estudio para la integración exitosa de programación reactiva en sistemas empresariales existentes.

3. Marco teórico

- **Programación Reactiva**

La programación reactiva se basa en el patrón Observer-Publisher, donde los flujos de datos (Streams) emiten eventos que los suscriptores consumen de forma asíncrona. Project Reactor implementa las especificaciones Reactive Streams mediante los tipos Flux (múltiples elementos) y Mono (0-1 elementos), permitiendo procesamiento no bloqueante y control de flujo mediante backpressure.

- **Arquitectura Spring Boot**

Spring Boot 3.5.7 facilita el desarrollo mediante autoconfiguración, inyección de dependencias y configuración convencional. La integración con Spring Data JPA abstrae el acceso a datos, mientras que Spring Security proporciona autenticación JWT y Spring Web MVC expone endpoints REST.

- **Patrón Subscriber Personalizado**

La implementación de Subscriber<T> requiere cuatro métodos fundamentales: onSubscribe() para inicializar la suscripción, onNext() para procesar cada elemento, onError() para manejar excepciones, y onComplete() para finalizar el flujo. El backpressure se implementa mediante subscription.request(n) controlando la cantidad de elementos procesados.

- **React y Arquitectura Frontend**

React 19.2.0 implementa el patrón Virtual DOM para renderizado eficiente, combinado con React Router para navegación SPA y Axios para comunicación HTTP asíncrona. Bootstrap 5 proporciona diseño responsivo y componentes UI consistentes.

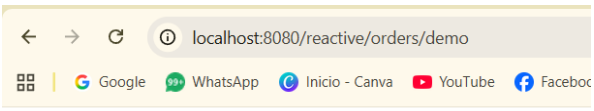
4. Pruebas - evidencia

A. Suscripción al Flujo Reactivo

Para demostrar la conexión entre el **Publisher** y el **Subscriber** mediante el método **subscribe**(). Este punto valida que el flujo reactivo se inicialice

correctamente y que el **Subscriber** esté listo para recibir eventos, con el objetivo de probar el ciclo de vida completo: **onSubscribe → onNext → onComplete/onError**

Para ello se ejecuta la siguiente dirección para obtener y evidencia la suscripción al flujo reactivo http://localhost:8080/reactive/orders/demo



Flujo reactivo iniciado. Revisa la consola para ver el procesamiento.

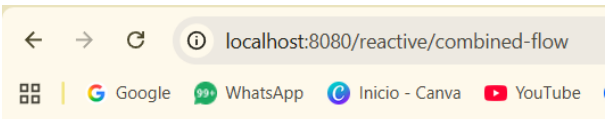
```
INICIANDO FLUJO REACTIVO DE PEDIDOS
Tiempo de inicio: 2025-12-21T14:12:46.992954400
onSubscribe: suscripción iniciada
Backpressure -> solicitando 2 pedidos
============================================================

Filtro: evaluando pedido de $2.5
Filtro: evaluando pedido de $5.0
Calculando impuestos: $5.0 -> $5.6000000000000005
onNext: pedido procesado = 5.6000000000000005
Filtro: evaluando pedido de $12.0
Calculando impuestos: $12.0 -> $13.440000000000001
onNext: pedido procesado = 13.440000000000001
Lote procesado -> solicitando 2 más
Filtro: evaluando pedido de $1.0
Filtro: evaluando pedido de $25.0
Pedido inválido detectado: $25.0
Manejo de error: Pedido inválido: $25.0
Aplicando valores de respaldo...
onNext: pedido procesado = 6.0
onNext: pedido procesado = 7.0
Lote procesado -> solicitando 2 más
onNext: pedido procesado = 8.0
onComplete: flujo de pedidos finalizado
```

B. Verificación de Eventos

Probar flujo completo con logs

Validar la ejecución end-to-end del flujo reactivo capturando todos los eventos en los logs. Demuestra que cada etapa **(filter, map, transform)** procesa correctamente los datos y que el Subscriber recibe los elementos esperados.



Flujo combinado iniciado. Revisa la consola.

```
FLUJO REACTIVO COMBINADO
2025-12-21T14:17:49.352-05:00 DEBUG 18308 --- [tienda_ropa] [nio-8080-exec-2] org.hibernate.SQL
    select
        p1_0.id,
        p1_0.categoria_id,
        p1_0.color,
        p1_0.descripcion,
        p1_0.imagen_url,
        p1_0.nombre,
        p1_0.precio,
        p1_0.stock,
        p1_0.talla
    from
        productos p1_0
Hibernate:
    select
        p1_0.id,
        p1_0.categoria_id,
        p1_0.color,
        p1_0.descripcion,
        p1_0.imagen_url,
        p1_0.nombre,
        p1_0.precio,
        p1_0.stock,
        p1_0.talla
    from
        productos p1_0
```

```
Procesando producto: Camiseta B├ísica Blanca
Producto: Camiseta B├ísica Blanca
Procesando producto: Camiseta B├ísica Negra
Pedido: $10.0
Producto: Camiseta B├ísica Negra
[1] Producto procesado: Venta: Camiseta B├ísica Blanca por $10.0
Procesando producto: Jeans Cl├ísicos Azul
Producto: Jeans Cl├ísicos Azul
Procesando producto: Jeans Cl├ísicos Negro
Producto: Jeans Cl├ísicos Negro
Pedido: $15.5
[2] Producto procesado: Venta: Camiseta B├ísica Negra por $15.5
Lote completado (2 productos)
Solicitando siguiente lote de 2 productos
Procesando producto: Vestido Floral Rosa
Producto: Vestido Floral Rosa
Pedido: $22.0
[3] Producto procesado: Venta: Jeans Cl├ísicos Azul por $22.0
Pedido: $8.75
[4] Producto procesado: Venta: Jeans Cl├ísicos Negro por $8.75
Lote completado (2 productos)
Solicitando siguiente lote de 2 productos
Pedido: $30.0
[5] Producto procesado: Venta: Vestido Floral Rosa por $30.0
[2025-12-21T14:17:53.115790600] ProductSubscriber completado
Total de productos procesados: 5
```

Procesamiento reactivo pedido real (debe existir en BD)

Para probar la integración del flujo reactivo con datos reales de la base de datos. Demostrando aplicabilidad práctica en producción.

http://localhost:8080/reactive/real-order/1

```
        p1_0.id=?
================================================
INICIANDO FLUJO REACTIVO REAL DEL PEDIDO #1
Cliente ID: 2
Total del pedido: $111.97
2025-12-21T14:36:48.364-05:00 DEBUG 18308 --- [tienda_ropa] [nio-8080-exec-3] org.hibernate.SQL
    select
        dp1_0.id,
        dp1_0.cantidad,
        dp1_0.pedido_id,
        dp1_0.precio_unitario,
        dp1_0.producto_id,
        dp1_0.subtotal
    from
        detalle_pedidos dp1_0
    where
        dp1_0.pedido_id=?
Hibernate:
    select
        dp1_0.id,
        dp1_0.cantidad,
        dp1_0.pedido_id,
        dp1_0.precio_unitario,
        dp1_0.producto_id,
        dp1_0.subtotal
    from
        detalle_pedidos dp1_0
    where
```

a_tienda-online > src > main > resources > application-xampp.properties

```
        detalle_pedidos dp1_0
    where
        dp1_0.pedido_id=?
Total detalles: 2
================================================
onSubscribe: suscripción iniciada
Backpressure -> solicitando 2 pedidos
Procesando detalle - Producto ID: 2, Cantidad: 2, Subtotal: $39.98
Filtro: evaluando pedido $39.98 - VALIDO
Aplicando IVA: $39.98 -> $44.78
onNext: pedido procesado = 44.78
Procesando detalle - Producto ID: 3, Cantidad: 1, Subtotal: $59.99
Filtro: evaluando pedido $59.99 - VALIDO
Aplicando IVA: $59.99 -> $67.19
onNext: pedido procesado = 67.19
Lote procesado -> solicitando 2 más
================================================
FLUJO REACTIVO COMPLETADO PARA PEDIDO #1
================================================
onComplete: flujo de pedidos finalizado
Procesamiento del pedido #1 finalizado
```

Probar todos los pedidos

Evaluar el rendimiento y comportamiento del flujo reactivo con múltiples elementos simultáneamente. Permite observar backpressure en acción y cómo el sistema maneja volúmenes mayores de datos y para ello se ingresa la siguiente dirección.

http://localhost:8080/reactive/all-orders

```
PROCESAMIENTO REACTIVO DE TODOS LOS PEDIDOS
2025-12-21T14:45:01.434-05:00 DEBUG 18308 --- [tienda_ropa] [nio-8080-exec-8] org.hibernate.SQL
    select
        p1_0.id,
        p1_0.cliente_id,
        p1_0.direccion_envio,
        p1_0.estado,
        p1_0.fecha_pedido,
        p1_0.observaciones,
        p1_0.total
    from
        pedidos p1_0
Hibernate:
    select
        p1_0.id,
        p1_0.cliente_id,
        p1_0.direccion_envio,
        p1_0.estado,
        p1_0.fecha_pedido,
        p1_0.observaciones,
```

```
onNext: pedido procesado = 67.19
Lote procesado -> solicitando 2 más
================================================
FLUJO REACTIVO COMPLETADO PARA PEDIDO #1
================================================
onComplete: flujo de pedidos finalizado
Procesamiento del pedido #1 finalizado
```

C. Prueba de Asincronía

Flujo con delay para ver asincronía

Se demestra explícitamente el procesamiento no bloqueante mediante la introducción de retrasos artificiales (**delayElements**()). Este permite observar timestamps diferentes entre eventos, validando que el sistema no se bloquea mientras espera y que puede procesar otras tareas concurrentemente. Es crucial para evidenciar que la programación reactiva realmente opera de forma asíncrona y no síncrona simulada.

http://localhost:8080/reactive/inventory/process

```
PROCESAMIENTO REACTIVO DE INVENTARIO
[2025-12-21T14:52:02.864112900] ProductSubscriber suscrito
Backpressure: solicitando 2 productos iniciales
============================================================

Inventario validado: Camisa Formal - EN STOCK (92 unidades)
Inventario procesado: Camisa Formal - EN STOCK (92 unidades)
[1] Producto procesado: Camisa Formal - EN STOCK (92 unidades)
Inventario validado: Pantalón Jeans - EN STOCK (77 unidades)
Inventario procesado: Pantalón Jeans - EN STOCK (77 unidades)
[2] Producto procesado: Pantalón Jeans - EN STOCK (77 unidades)
Lote completado (2 productos)
Solicitando siguiente lote de 2 productos
Inventario validado: Vestido Elegante - EN STOCK (39 unidades)
Inventario procesado: Vestido Elegante - EN STOCK (39 unidades)
[3] Producto procesado: Vestido Elegante - EN STOCK (39 unidades)
Inventario validado: Chaqueta de Cuero - EN STOCK (64 unidades)
Inventario procesado: Chaqueta de Cuero - EN STOCK (64 unidades)
[4] Producto procesado: Chaqueta de Cuero - EN STOCK (64 unidades)
Lote completado (2 productos)
Solicitando siguiente lote de 2 productos
Inventario validado: Zapatillas Deportivas - EN STOCK (74 unidades)
Inventario procesado: Zapatillas Deportivas - EN STOCK (74 unidades)
[5] Producto procesado: Zapatillas Deportivas - EN STOCK (74 unidades)
Inventario validado: Blusa Casual - EN STOCK (14 unidades)

[7] Producto procesado: Falda Plisada - EN STOCK (80 unidades)
Inventario validado: Sombrero - EN STOCK (85 unidades)
Inventario procesado: Sombrero - EN STOCK (85 unidades)
[8] Producto procesado: Sombrero - EN STOCK (85 unidades)
Lote completado (2 productos)
Solicitando siguiente lote de 2 productos
Inventario validado: Cinturón de Cuero - EN STOCK (34 unidades)
Inventario procesado: Cinturón de Cuero - EN STOCK (34 unidades)
[9] Producto procesado: Cinturón de Cuero - EN STOCK (34 unidades)
Inventario validado: Bufanda - EN STOCK (72 unidades)
Inventario procesado: Bufanda - EN STOCK (72 unidades)
[10] Producto procesado: Bufanda - EN STOCK (72 unidades)
Lote completado (2 productos)
Solicitando siguiente lote de 2 productos
Inventario completamente procesado
[2025-12-21T14:52:08.059246600] ProductSubscriber completado
Total de productos procesados: 10
```

Flujo continuo (Flux)

Demostrar el procesamiento no bloqueante característico de la programación reactiva. Un Flux continuo permite observar cómo el sistema maneja múltiples eventos concurrentemente sin bloquear el hilo.

http://localhost:8080/reactive/orders/flux


```
[
  {
    "status": "PROCESSING",
    "orderId": 1,
    "customer": "Cliente-1",
    "timestamp": "2025-12-21T14:55:14.296830900",
    "amount": 38
  },
  {
    "status": "PROCESSING",
    "orderId": 2,
    "customer": "Cliente-2",
    "timestamp": "2025-12-21T14:55:14.822143300",
    "amount": 15.72
  },
  {
    "status": "PROCESSING",
    "orderId": 3,
    "customer": "Cliente-3",
    "timestamp": "2025-12-21T14:55:15.336984",
    "amount": 36.05
  },
  {
    "status": "PROCESSING",
    "orderId": 4,
    "customer": "Cliente-4",
    "timestamp": "2025-12-21T14:55:15.838702700",
    "amount": 29.04
  },
  {
    "status": "PROCESSING",
    "orderId": 5,
    "customer": "Cliente-5",
    "timestamp": "2025-12-21T14:55:16.351420600",
    "amount": 8.78
```


Verificación en consola

```
Pedido generado: {status=PROCESSING, orderId=1, customer=Cliente-1, timestamp=2025-12-21T14:55:14.296830900, amount=38.0}
Pedido generado: {status=PROCESSING, orderId=2, customer=Cliente-2, timestamp=2025-12-21T14:55:14.822143300, amount=15.72}
Pedido generado: {status=PROCESSING, orderId=3, customer=Cliente-3, timestamp=2025-12-21T14:55:15.336984, amount=36.05}
Pedido generado: {status=PROCESSING, orderId=4, customer=Cliente-4, timestamp=2025-12-21T14:55:15.838702700, amount=29.04}
Pedido generado: {status=PROCESSING, orderId=5, customer=Cliente-5, timestamp=2025-12-21T14:55:16.351420600, amount=8.78}
Pedido generado: {status=PROCESSING, orderId=6, customer=Cliente-6, timestamp=2025-12-21T14:55:16.862536100, amount=9.81}
Pedido generado: {status=PROCESSING, orderId=7, customer=Cliente-7, timestamp=2025-12-21T14:55:17.374100100, amount=11.89}
Pedido generado: {status=PROCESSING, orderId=8, customer=Cliente-8, timestamp=2025-12-21T14:55:17.886068600, amount=7.7}
Pedido generado: {status=PROCESSING, orderId=9, customer=Cliente-9, timestamp=2025-12-21T14:55:18.389886100, amount=20.41}
Pedido generado: {status=PROCESSING, orderId=10, customer=Cliente-10, timestamp=2025-12-21T14:55:18.900131500, amount=9.01}
```

5. Conclusiones

- La programación reactiva se integra exitosamente en Spring Boot sin afectar operaciones tradicionales, complementando el sistema existente.
- El control de flujo mediante request(n) previene sobrecarga y permite procesamiento controlado de grandes volúmenes de datos.
- Aunque añade complejidad conceptual, los beneficios en rendimiento y escalabilidad justifican su implementación en sistemas críticos.

6. Recomendaciones

- Extender flujos reactivos a pagos, notificaciones push y actualizaciones de inventario en tiempo real para maximizar beneficios.
- Implementar métricas reactivas con Prometheus/Micrometer para visibilidad del comportamiento en producción.
- Desarrollar pruebas con StepVerifier para garantizar corrección de flujos asíncronos y facilitar evolución del sistema.