CENTRO UNIVERSITARIO UAEM ZUMPANGO

Sistema de compras en línea con notificación por eventos

Docente:

Rosa Erendira Reyes Luna

Alumnos:

Morales Jasso Jonathan

Reyes Casasola Marcos Adrian

Villagrana Tovar Monsserat

DOCUMENTACIÓN TÉCNICA

DESCRIPCIÓN GENERAL DEL SISTEMA:

Este sistema simula el funcionamiento de una tienda en línea moderna, donde los procesos como registrar una compra, enviar notificaciones por correo y actualizar el inventario suceden de forma automática, gracias a una arquitectura conocida como arquitectura basada en eventos. Pero también incluye una interfaz de acceso para dos tipos de usuarios: ADMIN y CLIENTE. A través de esta interfaz, los usuarios pueden interactuar con el sistema de manera:

- **El CLIENTE** puede iniciar sesión con su correo electrónico y realizar pedidos desde un formulario.
- **El ADMIN** puede acceder a la consola de administración H2 para ver todos los pedidos registrados en la base de datos.

Gracias al uso de eventos, el sistema asegura que cada acción desencadena automáticamente procesos de manera que permite que cuando un cliente realice una compra, el sistema pueda reaccionar de manera automática realizando las siguientes acciones:

1. Registre y guarde la orden en la base de datos.
2. Envíe una notificación automática al comprador.
3. Se publica un evento para que otros componentes reaccionen sin acoplamiento directo.
4. Se activan:
    - Un listener de notificación, que simula el envío de correo electrónico.
    - Un listener de inventario, que simula la actualización del stock.
5. Actualice el inventario de productos junto con el evento de listener de inventario.

Todo esto ocurre sin que el servicio que gestiona las compras tenga que controlar directamente las otras tareas y sin que el código esté directamente acoplado entre sí. En su lugar, cada parte del sistema escucha los eventos y responde cuando es necesario.

ACCIONES DEL CLIENTE:

El cliente accede con su correo electrónico.
Rellena un formulario indicando:

- Producto deseado.
- Cantidad.

Al dar clic en el botón "Realizar pedido" automáticamente:

- Se guarda el pedido.
- Se dispara un evento.
- Se envía un correo (mensaje simulado).
- Se actualiza el inventario (mensaje simulado).

ACCIONES DEL ADMIN:

Accede con credenciales de administrador.
Es redirigido a la consola H2, donde puede consultar:

- La tabla de órdenes.
- Todas las órdenes realizadas por los clientes.

Puede verificar que nuevos pedidos aparezcan automáticamente después de cada envío.

FUNCIONAMIENTO DEL SISTEMA: Cuando usuario entra a la tienda en línea lo que ocurre cuando realiza una compra es:

Paso 1: Usuario realiza una compra

Llena un formulario de compra con:

- ♦ Producto que desea.
- ♦ Cantidad.

Paso 2: Se registra la orden

- Esa información es recibida por el sistema mediante un componente llamado OrderController, que actúa como puerta de entrada.
- Luego, se guarda en la base de datos gracias al OrderService usando Spring Data JPA.

Paso 3: Se emite un evento

- Después de guardar la orden, el sistema emite un evento: OrderCreatedEvent.
- Este evento representa el hecho de que "una orden ha sido creada".

Paso 4: Otros servicios escuchan el evento

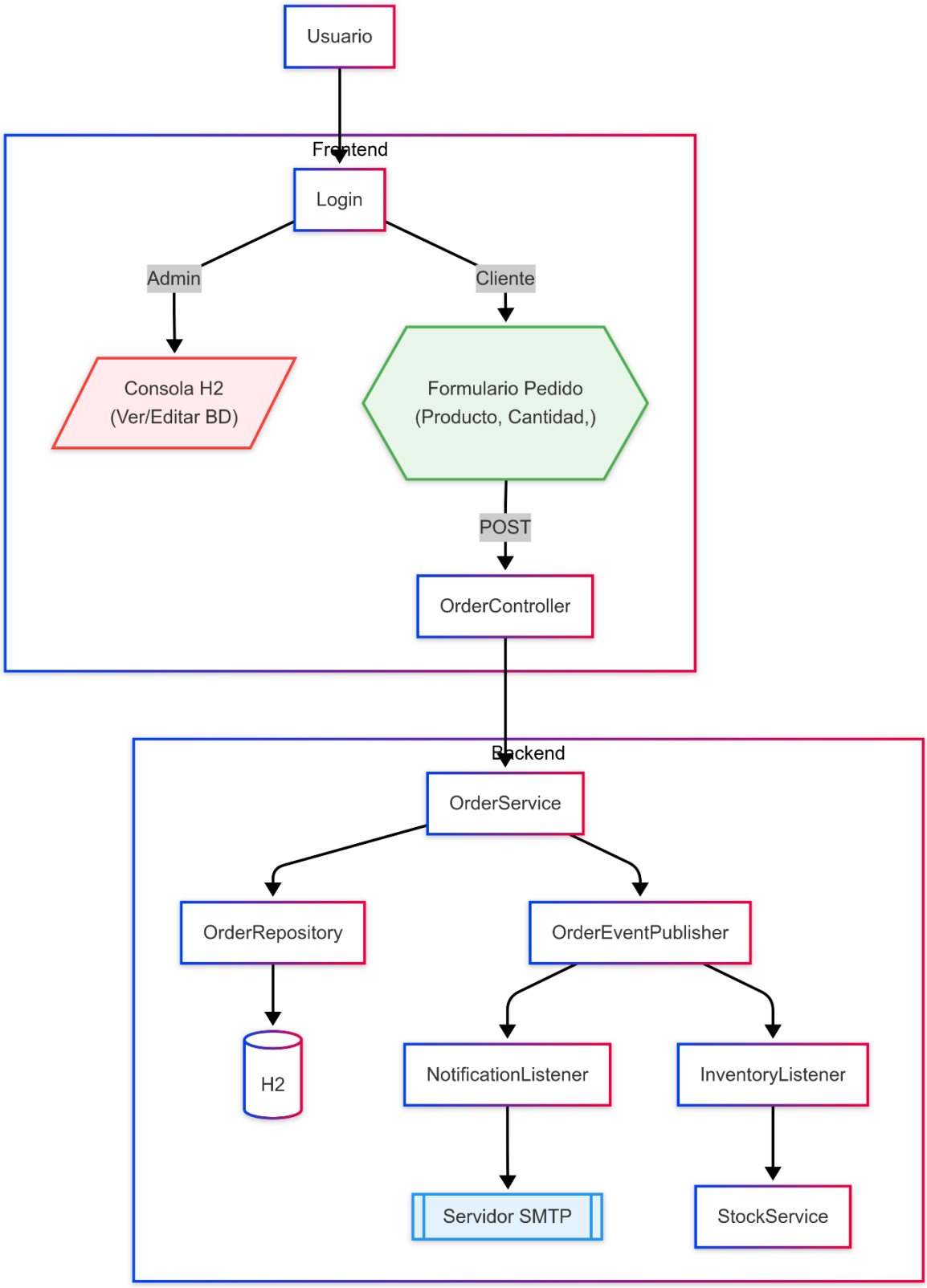
Hay dos componentes en el sistema que escuchan estos eventos:

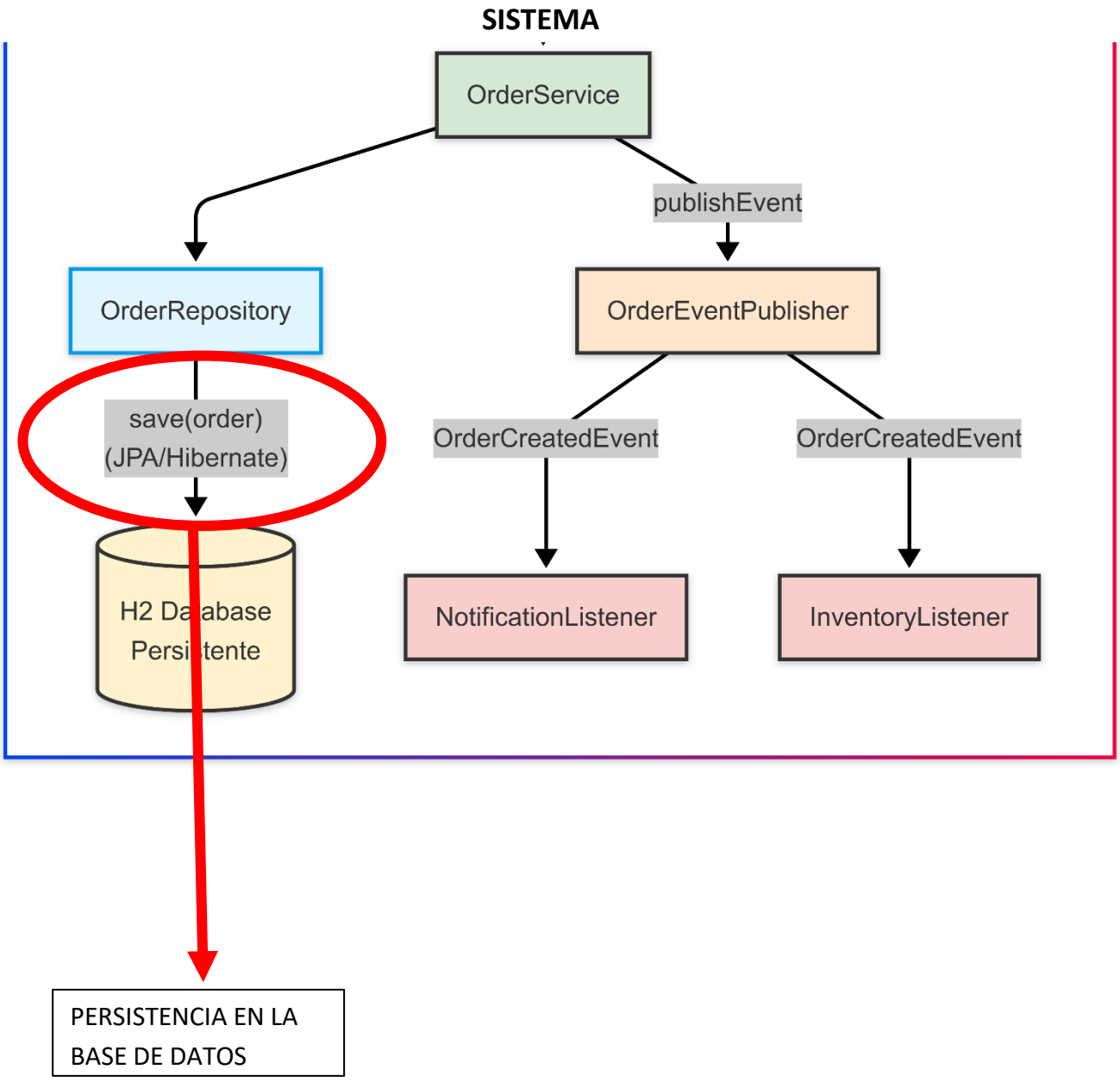
- **NotificationListener:** detecta el evento y envía una notificación por correo electrónico al cliente (simulado con un mensaje en consola).
- **InventoryListener:** detecta el evento y actualiza el inventario del producto comprado (también simulado con un mensaje en consola).

TECNOLOGÍAS QUE SE UTILIZAN:

Java 17+: Lenguaje de programación.
Spring Boot 3+: Framework que facilita la creación de aplicaciones web.
Spring Events: Permite que diferentes partes del sistema se comuniquen a través de eventos.
Spring Data JPA: Para guardar y leer datos en la base de datos de forma sencilla.
Base de datos H2: Una base de datos en memoria que simula el almacenamiento de datos reales.
Maven: Herramienta que gestiona las dependencias del proyecto.
Visual Studio Code : Editores para programar.

DIAGRAMA DE ARQUITECTURA:


Usuario
Frontend
Login
Admin
Cliente
Consola H2
(Ver/Editar BD)
Formulario Pedido
(Producto, Cantidad,)
POST
OrderController
Backend
OrderService
OrderRepository
OrderEventPublisher
H2
NotificationListener
InventoryListener
Servidor SMTP
StockService


SISTEMA
OrderService
publishEvent
OrderRepository
OrderEventPublisher
save(order)
(JPA/Hibernate)
OrderCreatedEvent
OrderCreatedEvent
H2 Database
Persistente
NotificationListener
InventoryListener
PERSISTENCIA EN LA
BASE DE DATOS

FLUJO DE EVENTOS Y FUNCIONAMIENTO DEL SISTEMA

El sistema utiliza Spring Events para manejar eventos de forma interna, simulando un entorno de microservicios pero dentro de un solo proyecto.

1. Cliente realiza pedido (OrderController)

El usuario CLIENTE accede a una interfaz en primer instancia de login donde ingresa su correo y contraseña, posteriormente lo manda a la sección donde puede llenar el formulario de pedido. Al hacer clic en "Realizar pedido", se envía una solicitud POST al endpoint /ordenes.

2. Creación de la orden (OrderService)

El controlador llama al servicio OrderService, que guarda la orden en la base de datos utilizando OrderRepository. Una vez almacenada, el servicio publica un evento OrderCreatedEvent a través de OrderEventPublisher.

3. Publicación del evento (OrderEventPublisher)

Este componente crea y emite el evento OrderCreatedEvent utilizando el ApplicationEventPublisher de Spring. El evento contiene la orden como información clave.

4. Listeners de Eventos

Dos clases anotadas con @Component y @EventListener escuchan el evento emitido:

- **NotificationListener**:
    - Escucha el evento OrderCreatedEvent.
    - Extrae el correo del pedido y simula el envío de un correo al cliente mediante un mensaje por consola.
- **InventoryListener**:
    - Escucha el mismo evento.
    - Extrae el producto del pedido y simula la actualización del inventario con un mensaje por consola.

5. Persistencia

La orden se guarda en la base de datos H2 en memoria, lo que permite consultar las órdenes mediante la consola H2 (accesible solo para el ADMIN). Esto garantiza que la información del pedido esté registrada y consultable mientras la aplicación esté en ejecución.

6. Separación de responsabilidades

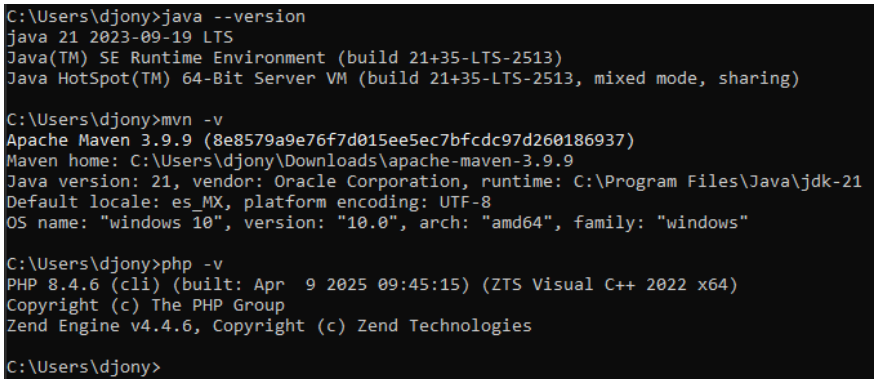
Cada módulo cumple un rol específico sin conocer la lógica interna de los otros:

- OrderController recibe la solicitud.
- OrderService guarda la orden y lanza el evento.
- OrderEventPublisher publica el evento.
- NotificationListener y InventoryListener manejan las acciones derivadas del evento de manera aislada.

7. Instrucciones para probar proyecto localmente

1. Requisitos previos

- Java JDK 17+ (verificar con java -version)
- Maven (verificar con mvn -v)
- PHP 8.0+ (solo para el frontend, verificar con php -v)
- Git (opcional, para clonar el repositorio)



2. Clonar el repositorio (opcional)

   git clone https://github.com/JonyMan1929/IngSoftwareII.git

3. Configuración del Backend (Spring Boot)

   1. Abrir el proyecto en VS Code/IDE o en la Terminal:

      Navegar a la carpeta backend.

2. Ejecutar la aplicación:

   mvn spring-boot:run

```
C:\Users\djony\OneDrive\Escritorio\U\6to Semestre\Ingenieria de Software II\ComprasLinea>mvn spring-boot:run
```

Se espera ver:

```
2025-04-09T20:49:33.137-06:00  INFO 14896 --- [  restartedMain] c.t.TiendaEventosApplication             : Started TiendaEventosApplication in 7.145 seconds (pro
nning for 8.464)
2025-04-09T20:50:20.463-06:00  INFO 14896 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServl
2025-04-09T20:50:20.464-06:00  INFO 14896 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2025-04-09T20:50:20.468-06:00  INFO 14896 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 2 ms
```

4. Configuración del Frontend (PHP)

1. **Abrir otra terminal** y navegar a la carpeta frontend:

   cd frontend

2. **Iniciar el servidor PHP**:

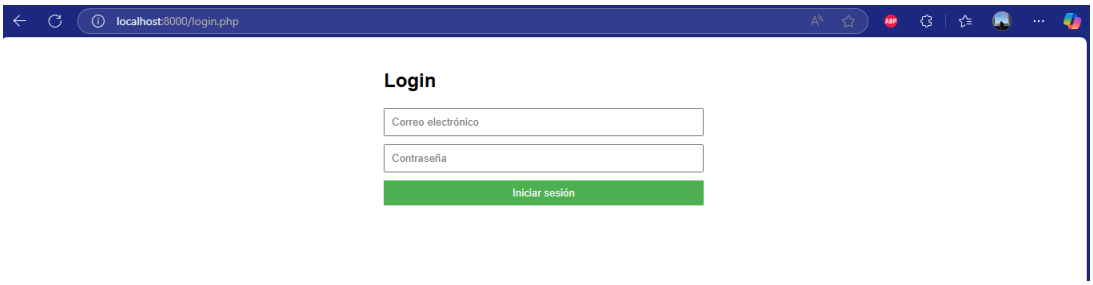
   php -S localhost:8000

```
C:\Users\djony\OneDrive\Escritorio\U\6to Semestre\Ingenieria de Software II\ComprasLinea\frontend>php -S localhost:8000
[Wed Apr  9 20:49:30 2025] PHP 8.4.6 Development Server (http://localhost:8000) started
```

5. Probar la aplicación

1. **Acceder al login**:
   Abrir en el navegador:
   http://localhost:8000/login.php
2. **Credenciales de prueba**:
   - **Admin**:
     Email: admin@tienda.com
     Contraseña: admin123
   - **Cliente**:
     Email: cliente@tienda.com
     Contraseña: cliente123

Aquí se muestra el login para iniciar sesión



Interfaz para administradores

Interfaz para Clientes

6. Hacer un pedido (Flujo cliente)

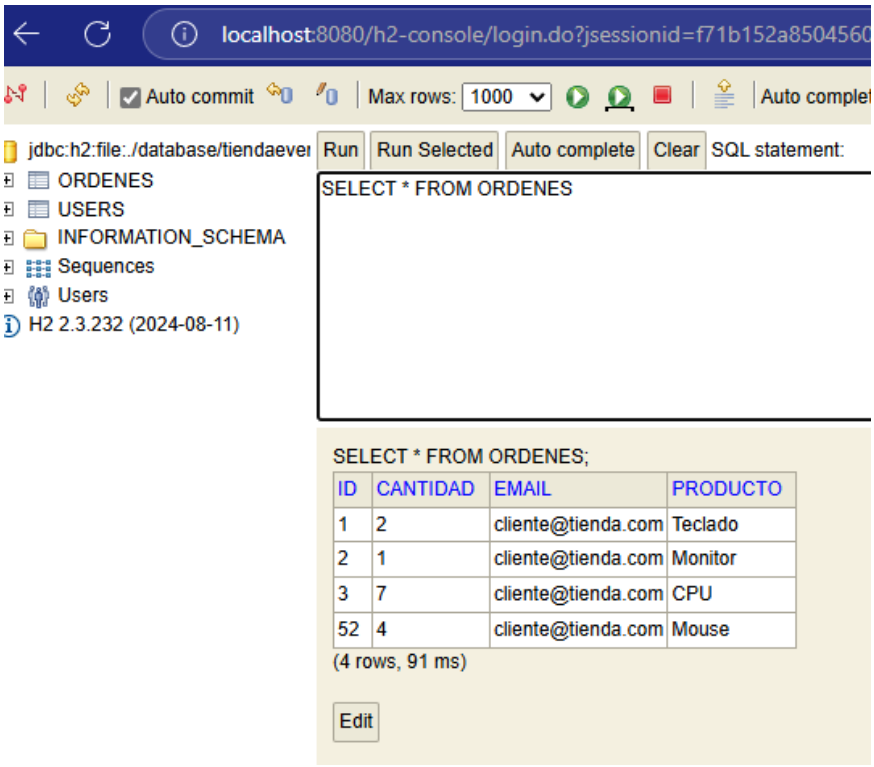
1. Iniciar sesión como cliente.
2. Llenar el formulario:
    - Producto: Seleccionar de la lista.
    - Cantidad: Ingresar un número.
3. Hacer clic en **"Realizar pedido"**.
4. Verificar el mensaje de confirmación.


```
2025-04-09T22:53:34.491-06:00  INFO 4308 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed i
ialization in 3 ms
[Inventario] Orden #52
   Producto: Mouse
   Cantidad: 4
   -- Stock actualizado --
Enviando correo a: cliente@tienda.com
   Producto: Mouse
   Cantidad: 4
```

7. Ver pedidos (Flujo admin)

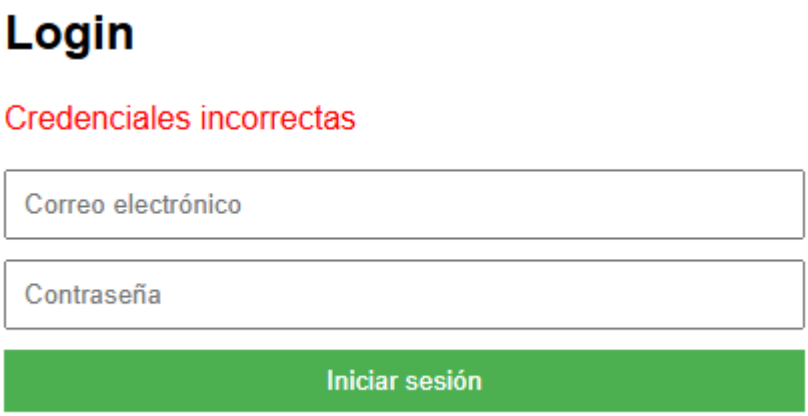
1. Iniciar sesión como admin.
2. **Acceder a la consola H2**:
   Hacer clic en el enlace *"Acceder a H2 Console"*.
3. En la consola H2:
   - JDBC URL: jdbc:h2:file:./database/tiendaeventosdb
   - User: sa
   - Password: (vacío)
4. Ejecutar consulta:

   SELECT * FROM orders;



8. Posibles errores y soluciones

- **Error 401 en login**: Verificar credenciales en data.sql.
- **PHP no inicia**: Verificar instalación con php -v.



Recomendaciones para escalar la solución a Kafka o RabbitMQ

1. Introducción a Message Brokers

Para escalar tu sistema de pedidos actual, se recomienda implementar un message broker como Kafka o RabbitMQ. Esto permite:

- Procesamiento asíncrono de pedidos
- Mayor escalabilidad
- Tolerancia a fallos
- Desacoplamiento de componentes

2. Opción 1: Apache Kafka
Mejor para: Sistemas de alto volumen y procesamiento en tiempo real.

Ventajas:

- Maneja miles de mensajes por segundo
- Permite múltiples consumidores para un mismo mensaje
- Retiene mensajes por periodos configurables

3. **Opción 2: RabbitMQ**
   Mejor para: Sistemas que requieren garantía de entrega y son más sencillos.

Ventajas:

- Protocolo AMQP estándar
- Manejo de colas tradicionales
- Configuración más simple que Kafka

6. Próximos Pasos

1. Configurar entorno de prueba con Docker
2. Implementar producer en OrderController
3. Crear consumers para los servicios
4. Realizar pruebas de carga

Reflexión final

- ¿Qué ventaja tiene emitir eventos frente a llamar servicios directamente?

- La principal ventaja es que los servicios quedan desacoplados. El servicio que crea la orden no tiene que preocuparse por cómo se actualiza el inventario o cómo se envía un correo. Solo emite un evento, y los demás servicios reaccionan si quieren. Esto hace que el sistema sea más flexible y fácil de modificar en el futuro.

- ¿Qué pasaría si agregamos más listeners?

- Si agregamos más listeners, simplemente se van a activar cuando ocurra el evento, sin tocar el código del publicador. Por ejemplo, podríamos agregar un listener que genere una factura o que envíe un mensaje por WhatsApp, y no necesitamos modificar nada del servicio de órdenes. Es muy útil para hacer crecer el sistema sin romper lo que ya funciona.

- ¿Cómo podríamos escalar esto usando Kafka o RabbitMQ?

- En lugar de usar eventos locales como hicimos acá, en un sistema más grande podríamos usar Kafka o RabbitMQ para enviar los eventos a través de una cola o un bus de mensajes. Esto permitiría que los microservicios estén en diferentes servidores o incluso lenguajes, y que los eventos se manejen de forma más robusta, con mayor tolerancia a fallos y mejor rendimiento.

CONCLUSIÓNES

Para cerrar, este proyecto nos permitió ver en la práctica cómo funciona una arquitectura basada en eventos, y cómo ayuda a organizar mejor un sistema distribuido.

En lugar de tener un código que esté todo conectado entre sí, aprendimos a separar responsabilidades usando eventos. Esto significa que cada módulo solo se preocupa por lo suyo, y reacciona cuando es necesario, sin depender directamente de otros servicios.

Por ejemplo, el servicio de órdenes simplemente publica un evento cuando se crea una compra, y luego los demás componentes, como el inventario o las notificaciones, responden automáticamente. Este enfoque no solo hace el sistema más limpio y fácil de mantener, sino que también permite agregar nuevas funcionalidades sin romper lo que ya funciona.

También nos dimos cuenta de que este tipo de arquitectura es muy útil si en algún momento queremos escalar el sistema. Por ahora usamos eventos locales dentro de una sola aplicación, pero más adelante

podríamos aplicar lo mismo usando herramientas como Kafka o RabbitMQ, lo cual es ideal para microservicios distribuidos o sistemas en la nube.

En lo personal, fue una experiencia bastante enriquecedora. No solo nos ayudó a entender mejor cómo se organiza un sistema real, sino que también nos dio la oportunidad de trabajar en equipo, dividir responsabilidades y ver cómo cada parte encaja dentro del todo.

Sin duda, este tipo de enfoques modernos son muy valiosos para nuestro futuro como desarrolladores de software.