Proyecto Integrador 202610

Asignatura: Integración de Sistemas

Integrantes : Sammy Porras

1. Contexto del caso

Una empresa mediana (retail/servicios) creció y hoy opera con sistemas heterogéneos. Necesita integrar su flujo Order-to-Cash: toma de pedidos (canal digital), inventario, pagos, logística, notificaciones y analítica. La integración debe ser robusta ante fallos, segura y con trazabilidad.

El foco del proyecto no es “hacer un e-commerce”, sino diseñar e implementar una solución de integración empresarial usando patrones y prácticas vistas en el curso.

2. Introducción

IntegraHub es una plataforma de integracion empresarial que implementa el flujo Order-to-Cash para una empresa de retail. Permite crear pedidos, validar inventario, procesar pagos y notificar al cliente, todo de forma asincrona y resiliente.

2.1 Problema que resuelve

IntegraHub es una plataforma de integracion empresarial que implementa el flujo Order-to-Cash para una empresa de retail. Permite crear pedidos, validar inventario, procesar pagos y notificar al cliente, todo de forma asincrona y resiliente.

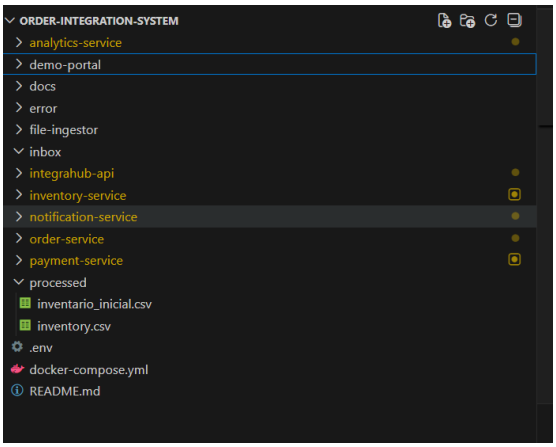
2.2 ARQUITECTURA

El sistema está diseñado bajo una arquitectura de microservicios, compuesta por siete servicios independientes que se comunican entre sí de manera asíncrona mediante un broker de mensajería. Este enfoque permite un alto nivel de desacoplamiento entre componentes, mejora la escalabilidad del sistema y facilita la tolerancia a fallos.

Servicio	Función
order-service	Expone una API REST para la creación y consulta de pedidos, siendo el punto de entrada principal del flujo de negocio.
inventory-service	Valida la disponibilidad de productos y gestiona la reserva de stock en función de los pedidos procesados.
payment-service	Procesa los pagos asociados a los pedidos y emite eventos de confirmación o rechazo.

Sammy Porras

Servicio	Función
notification-service	Gestiona la notificación y el registro de eventos del sistema, contribuyendo a la trazabilidad de las operaciones.
analytics-service	Consume eventos del sistema para generar métricas y estadísticas en tiempo real.
file-ingestor	Procesa archivos CSV para la carga y actualización masiva del inventario.
integrahub-api	Centraliza la autenticación mediante JWT y expone endpoints de verificación del estado del sistema (health checks).

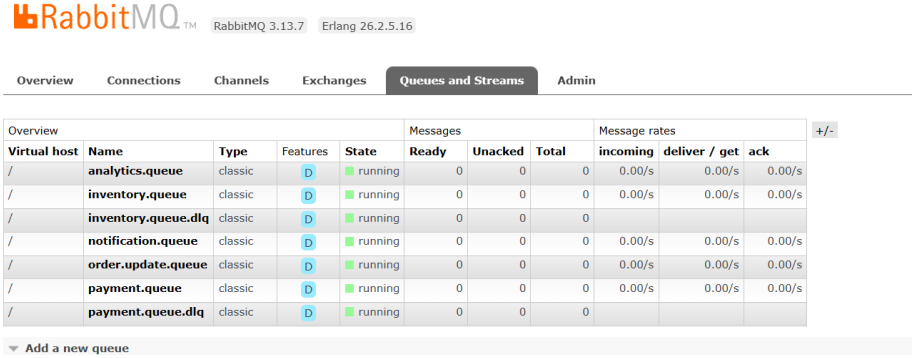


La comunicación entre los microservicios se realiza mediante RabbitMQ, utilizado como broker de mensajería. Se emplea un exchange de tipo topic, lo que permite un enrutamiento flexible de mensajes a través de routing keys. Este mecanismo posibilita que cada microservicio se suscriba únicamente a los eventos que le competen, eliminando dependencias directas entre productores y consumidores.

Este modelo de comunicación sigue un enfoque event-driven, en el cual los servicios reaccionan a eventos del dominio en lugar de realizar llamadas síncronas directas, lo que contribuye a la resiliencia y escalabilidad del sistema.

Persistencia y mensajería

Cada microservicio gestiona su propia persistencia utilizando PostgreSQL como sistema gestor de bases de datos relacional, garantizando la integridad y consistencia de la información transaccional.
RabbitMQ actúa como intermediario de mensajería asíncrona, asegurando la entrega de eventos y permitiendo la implementación de mecanismos de manejo de errores, tales como colas de reintento y Dead Letter Queues (DLQ), lo que fortalece la robustez del sistema ante fallos operativos.



Sammy Porras

2.3 DEMO FLOW

Objetivo de la demostración
Demostrar el funcionamiento correcto del sistema mediante la ejecución de un flujo exitoso de negocio, evidenciando la interacción entre los microservicios y la comunicación asíncrona a través de RabbitMQ.
Acceso al portal de demostración

Se accede al portal web del sistema mediante la siguiente URL:

http://localhost:3000

Este portal corresponde al entorno de demostración del sistema. A continuación, se procederá a crear un pedido desde el frontend.

Visualización del inventario

En la sección de inventario se muestra el stock actual de los productos registrados en el sistema. Por ejemplo, se puede observar que el producto seleccionado cuenta con X unidades disponibles. Esta información es provista por el inventory-service, el cual consulta su base de datos en PostgreSQL.

Creación de un pedido

Se realiza la creación de un nuevo pedido ingresando los siguientes datos:
Nombre del cliente: Cliente Demo
Producto: Producto con stock disponible
Cantidad: 2 unidades
Una vez ingresados los datos, se presiona el botón “Crear Pedido” para enviar la solicitud al sistema.
Ejecución del flujo del sistema
Al crear el pedido, el sistema ejecuta automáticamente el siguiente flujo de procesamiento:

1. El order-service registra el pedido en la base de datos con el estado inicial CREATED y publica el evento order.created en RabbitMQ.
2. El inventory-service consume el evento order.created, valida la disponibilidad del stock, descuenta la cantidad solicitada y publica el evento order.validated.
3. El payment-service consume el evento order.validated, procesa el pago correspondiente y publica el evento order.confirmed.
4. El notification-service recibe los eventos generados y registra las notificaciones del proceso para fines de trazabilidad.
5. El analytics-service consume todos los eventos del flujo y registra métricas en tiempo real para análisis y monitoreo del sistema.

Este proceso se ejecuta de manera asíncrona, sin dependencias directas entre los microservicios, utilizando RabbitMQ como intermediario.

Resultado de la operación

Una vez finalizado el flujo, el estado del pedido se actualiza automáticamente a PAID. Adicionalmente, el inventario refleja la disminución del stock correspondiente a la cantidad solicitada. Esto evidencia la correcta ejecución del flujo completo del sistema.

Valor técnico evidenciado

La demostración permite validar la correcta comunicación entre microservicios, el uso de mensajería asíncrona basada en eventos, la consistencia de los datos, la trazabilidad del proceso y el cumplimiento del flujo exitoso definido en la arquitectura del sistema.

2.4 DEMO – Flujo de Rechazo

Objetivo de la demostración

Demostrar el comportamiento del sistema ante un escenario de rechazo por regla de negocio válida, específicamente cuando no existe stock suficiente para procesar un pedido.

Creación de un pedido sin stock suficiente

Se procede a crear un nuevo pedido ingresando una cantidad mayor al stock disponible del producto seleccionado. Esta acción simula un escenario real en el que el inventario no puede satisfacer la

solicitud del cliente.

- Al enviar el pedido, el sistema procesa la solicitud y valida la disponibilidad de stock a través del inventory-service.
- Ejecución del flujo de rechazo
  Durante el procesamiento del pedido ocurre el siguiente flujo:

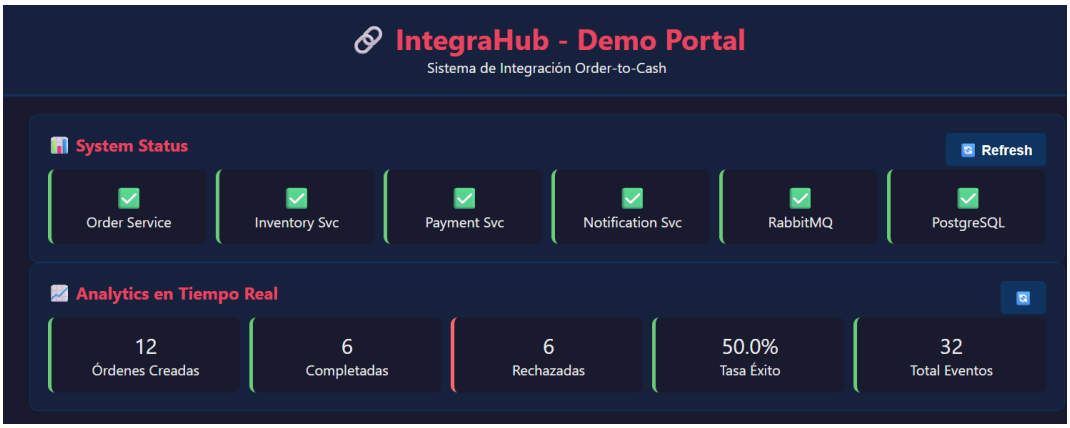
    1. El order-service registra el pedido con estado inicial CREATED y publica el evento order.created en RabbitMQ.
    2. El inventory-service consume el evento order.created y detecta que no existe stock suficiente para cubrir la cantidad solicitada.
    3. Al tratarse de una regla de negocio válida, el inventory-service publica el evento order.rejected indicando la causa del rechazo.
    4. El order-service actualiza el estado del pedido a REJECTED en la base de datos.
    5. El notification-service registra el evento de rechazo para fines de trazabilidad y auditoría.
    6. El analytics-service registra el evento como un rechazo de negocio dentro de las métricas del sistema.

Diferenciación entre rechazo de negocio y error técnico

Es importante destacar que este tipo de rechazo no corresponde a un error técnico del sistema, sino a una validación propia de la lógica de negocio. Por esta razón, el mensaje no es enviado a una Dead Letter Queue (DLQ), ya que el flujo se ejecuta correctamente desde el punto de vista técnico.
Resultado de la operación

El pedido permanece en estado REJECTED y el inventario no se ve afectado, confirmando que el sistema maneja correctamente los escenarios de rechazo por falta de stock sin generar errores innecesarios ni afectar la estabilidad del sistema.

2.5 Demo de Analítica



Objetivo de la demostración

Demostrar la capacidad del sistema para integrarse con sistemas legacy mediante el procesamiento de archivos CSV, sin necesidad de comunicación directa por APIs.
Contexto de la integración

El sistema soporta integración con sistemas externos o heredados a través de archivos CSV. Este mecanismo permite la carga y actualización de información de inventario proveniente de fuentes que no cuentan con servicios web o mensajería moderna.

Archivo CSV de ejemplo
Se presenta un archivo CSV con la siguiente estructura: sku,name,quantity,price
MOUSE-001,Mouse Logitech MX,50,49.99

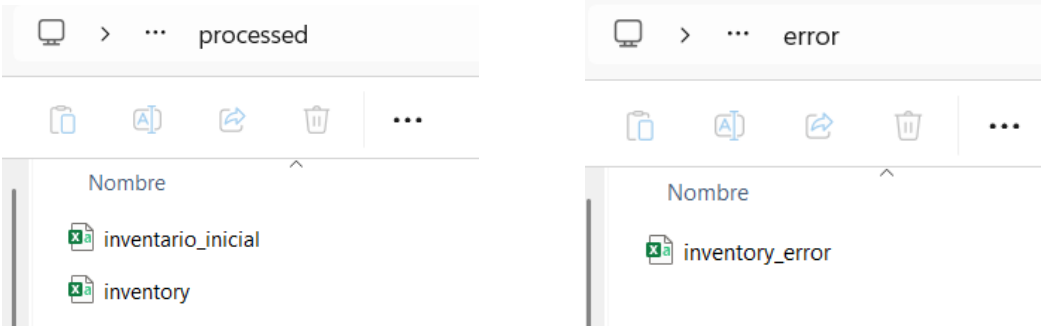
Este archivo representa un lote de actualización de inventario proveniente de un sistema externo.
Proceso de ingestión del archivo
Al copiar el archivo CSV en la carpeta designada como inbox, el sistema ejecuta automáticamente el siguiente flujo:

1. El microservicio file-ingestor detecta la presencia de un nuevo archivo en la carpeta de entrada.
2. El archivo es leído y procesado fila por fila, validando el formato y los datos contenidos.
3. Por cada registro, el file-ingestor actualiza el inventario correspondiente, sumando las cantidades indicadas a las existencias actuales del producto.
4. Las actualizaciones se persisten en la base de datos PostgreSQL del inventory-service.

Este proceso se ejecuta de forma automática, sin intervención manual ni dependencias directas con otros sistemas.

Resultado de la integración

Una vez procesado el archivo, el inventario del sistema refleja el incremento de stock correspondiente a los datos contenidos en el CSV. Esto confirma que el sistema soporta exitosamente la integración por archivos y es capaz de coexistir con sistemas legacy dentro del ecosistema tecnológico.



2.6 Patrones de integración

El sistema implementa seis patrones de integración empresarial, los cuales se detallan a continuación junto con su aplicación dentro de la arquitectura:

Patrón: Publish–Subscribe (Pub/Sub)
Este patrón se implementa mediante el uso de un exchange de tipo topic en RabbitMQ, permitiendo que múltiples microservicios consuman los mismos eventos sin acoplamiento directo. Cada servicio se suscribe únicamente a los eventos relevantes para su función, facilitando la extensibilidad del sistema.

Patrón: Point-to-Point
Cada cola de RabbitMQ es consumida por un único microservicio, garantizando que cada mensaje sea procesado exactamente por un consumidor. Este patrón asegura un procesamiento controlado y evita ejecuciones duplicadas no deseadas.

Patrón: Message Router
El enrutamiento de mensajes se realiza mediante routing keys definidas en el exchange de tipo topic. Estas claves determinan de forma explícita qué colas recibirán cada evento, permitiendo una distribución inteligente de los mensajes según el tipo de evento generado.

Patrón: Message Translator
Los microservicios transforman objetos de dominio o DTOs internos en eventos estandarizados antes de publicarlos en RabbitMQ. Este patrón desacopla los modelos internos de los servicios del formato de los mensajes intercambiados, facilitando la evolución independiente de cada componente.

Patrón: Dead Letter Queue (DLQ)
El sistema cuenta con colas de tipo Dead Letter Queue, en las cuales se almacenan los mensajes que no pueden ser procesados debido a errores técnicos, como fallos de formato, excepciones no controladas o errores de infraestructura. Esto permite el análisis posterior y la recuperación controlada de mensajes fallidos.

Patrón: Idempotent Consumer
Para evitar el procesamiento duplicado de mensajes, cada consumidor implementa un mecanismo de idempotencia mediante una tabla denominada processed_messages. Antes de procesar un mensaje, el sistema verifica si este ya fue tratado previamente, garantizando consistencia incluso ante reintentos o duplicaciones de mensajes.

2.7 Resiliencia

Simulación de fallo de un microservicio
Para simular un escenario de fallo, se procede a detener manualmente uno de los microservicios críticos del sistema mediante el siguiente comando:

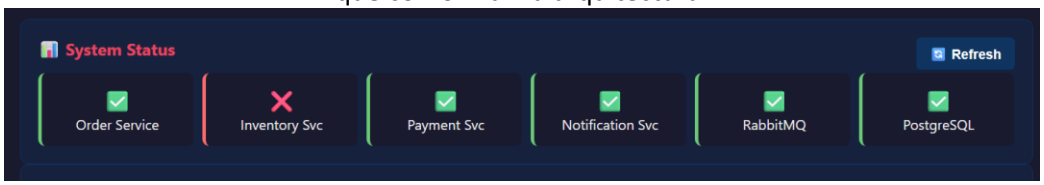
```
docker compose stop inventory-service
```

Esta acción representa una interrupción inesperada del servicio de inventario, como podría ocurrir en un entorno real por fallos de infraestructura o despliegue.

Detección del fallo
Una vez detenido el servicio, se accede al portal web del sistema y se visualiza la sección de estado del sistema (System Status). En esta vista se evidencia que el health check detecta correctamente que el inventory-service se encuentra en estado DOWN.

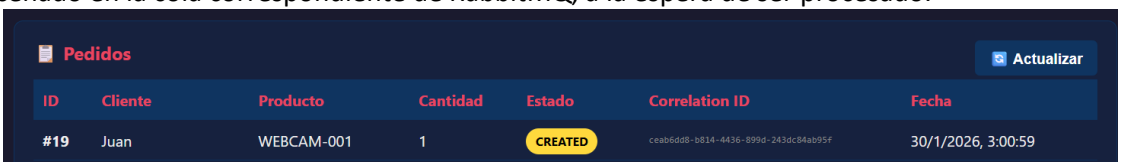
Este mecanismo de monitoreo permite identificar de forma temprana la indisponibilidad de los servicios

que conforman la arquitectura.



Creación de un pedido durante el fallo
Con el inventory-service detenido, se procede a crear un nuevo pedido desde el portal de demostración. El pedido es registrado correctamente por el order-service y permanece en estado CREATED.

Dado que no existe un consumidor activo para procesar el evento order.created, el mensaje queda almacenado en la cola correspondiente de RabbitMQ, a la espera de ser procesado.

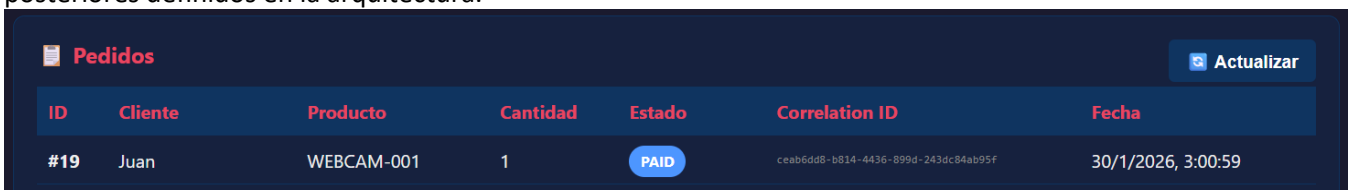


Recuperación del servicio
Posteriormente, se restablece el servicio detenido mediante el siguiente comando:

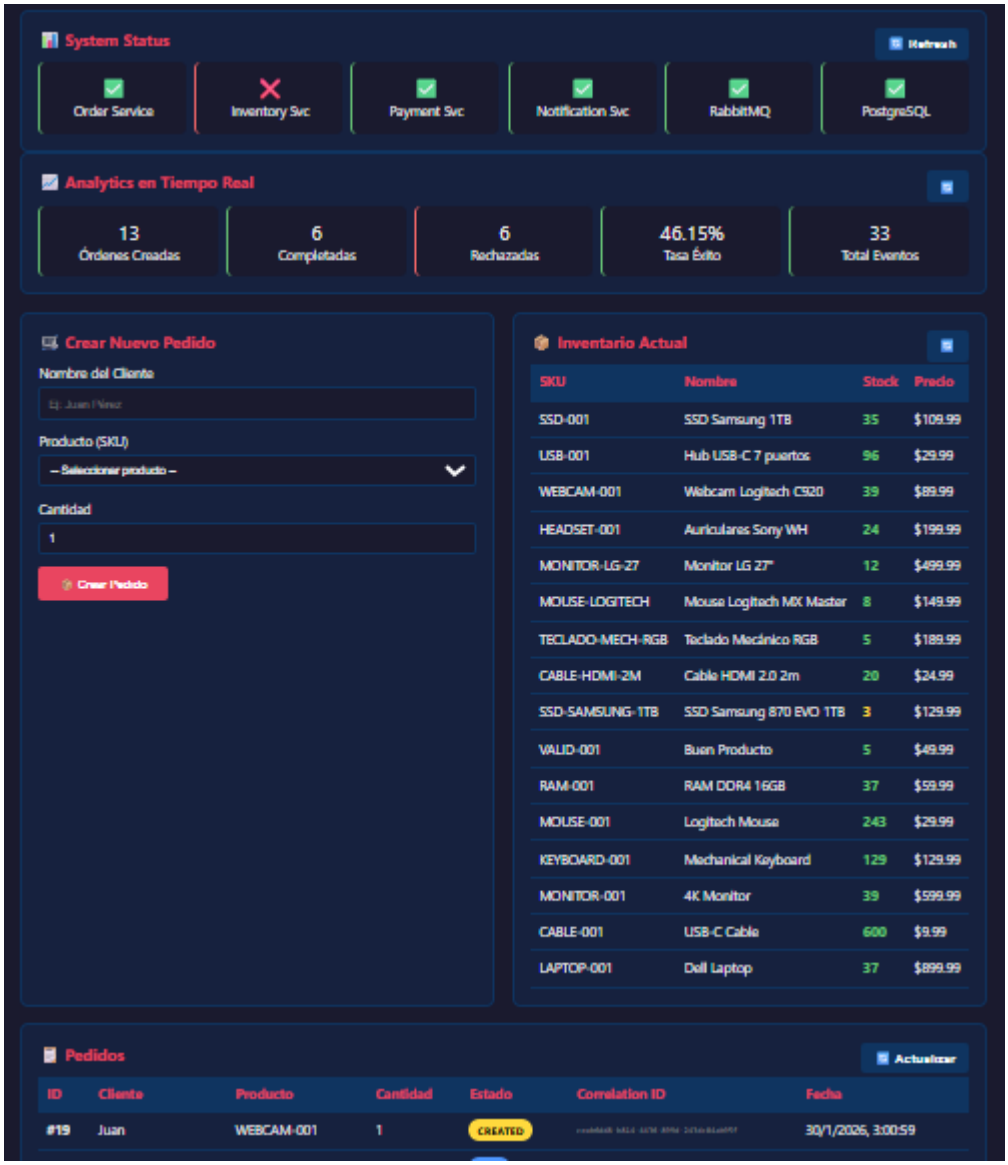
docker start inventory-service

Una vez levantado el servicio, este retoma automáticamente el consumo de mensajes pendientes en la cola.

Procesamiento automático del mensaje
El mensaje que se encontraba en espera es procesado de manera automática sin intervención manual. El flujo del sistema continúa con normalidad, actualizando el estado del pedido y ejecutando los pasos posteriores definidos en la arquitectura.



Resultado de la demostración
Esta prueba evidencia que el sistema es resiliente ante fallos temporales de servicios, ya que los mensajes no se pierden y el procesamiento se reanuda automáticamente una vez que el servicio afectado vuelve a estar disponible. Esto confirma el correcto uso de mensajería asíncrona y colas persistentes como mecanismo de tolerancia a fallos.



Los mensajes que superan el número máximo de reintentos son enviados a colas Dead Letter Queue (por ejemplo, `inventory.queue.dlq`), las cuales pueden ser visualizadas desde la interfaz de administración de RabbitMQ.

2.8 Seguridad (obligatorio)

Implementación de autenticación JWT
El sistema implementa un esquema de autenticación basado en JWT, centralizado en el servicio integrahub-api. Este mecanismo permite validar la identidad de los usuarios y autorizar el acceso a los recursos protegidos del sistema.

Para la demostración, se accede a la documentación interactiva del API a través de la siguiente URL:

http://localhost:8081/docs

Sammy Porras

Generación del token de acceso

Desde la documentación del API, se ejecuta el endpoint POST /token utilizando credenciales válidas. Para efectos de la demostración se emplean las siguientes credenciales:

Usuario: admin
Contraseña: password123

Como resultado de esta operación, el sistema genera un token JWT válido.

Características del token
El token generado cuenta con una expiración configurada de 30 minutos. Durante este periodo, el token permite el acceso a los endpoints protegidos del sistema sin necesidad de reenviar credenciales, fortaleciendo la seguridad y reduciendo la exposición de información sensible.

Acceso a rutas protegidas
A continuación, se intenta acceder a un endpoint protegido, por ejemplo /protected, sin enviar el token de autenticación. El sistema rechaza la solicitud, evidenciando que el acceso está restringido.

Posteriormente, se realiza la misma solicitud incluyendo el token JWT en la cabecera de autorización. En este caso, el acceso es concedido correctamente, confirmando que el mecanismo de autenticación y autorización funciona de acuerdo con lo esperado.

3. Conclusión

En conclusión, IntegraHub se presenta como un sistema de integración moderno basado en microservicios y mensajería asíncrona, el cual cumple con los principios fundamentales de las arquitecturas distribuidas actuales.

- El sistema es asíncrono, ya que las operaciones iniciadas por el usuario no dependen de la finalización inmediata de los procesos internos, los cuales se ejecutan mediante eventos.
- La interacción del usuario es no bloqueante, mientras que el procesamiento del negocio se realiza de forma desacoplada a través de mensajería.
- Es resiliente, debido a que tolera la caída temporal de servicios sin pérdida de información, apoyándose en colas persistentes y reprocesamiento automático.
- Es observable, puesto que registra eventos, métricas y trazabilidad del flujo completo de negocio.
- Es seguro, al implementar autenticación basada en JSON Web Tokens (JWT) para el acceso a recursos protegidos.
- Es extensible, ya que la incorporación de nuevos microservicios requiere únicamente la suscripción a los eventos existentes, sin modificar los servicios actuales.

Este conjunto de características demuestra que IntegraHub cumple con buenas prácticas de integración empresarial y arquitectura orientada a eventos.

URLs utilizadas durante la demostración

Recurso: Demo Portal

Sammy Porras

URL: http://localhost:3000

Recurso: Swagger Orders
URL: http://localhost:8082/docs

Recurso: Swagger IntegraHub
URL: http://localhost:8081/docs

Recurso: Swagger Analytics
URL: http://localhost:8083/docs

Recurso: RabbitMQ Management UI
URL: http://localhost:15672

Credenciales: guest / guest

Comandos útiles utilizados en la demostración

Inicio del entorno completo:
docker compose up -d

Detener un microservicio específico:
docker compose stop inventory-service

Levantar un microservicio específico:
docker compose start inventory-service

Detener todo el entorno:
docker compose down

Sammy Porras