

Universidad de Las Américas
Facultad de Ingenierías y Ciencias Agropecuarias
Ingeniería De Software
Progreso 1

Nombres: Sammy Porras, Daniel Vizcarra

Fecha: 29/10/2025

TRANSFERENCIA DE ARCHIVOS – APACHE CAMEL

Objetivo de la actividad:

Implementar un flujo de integración con Apache Camel, utilizando Maven, que permita detectar archivos CSV en una carpeta de origen y trasladarlos automáticamente a una carpeta destino. Con esto se busca aplicar el patrón de transferencia de archivos como mecanismo básico de integración entre sistemas independientes.

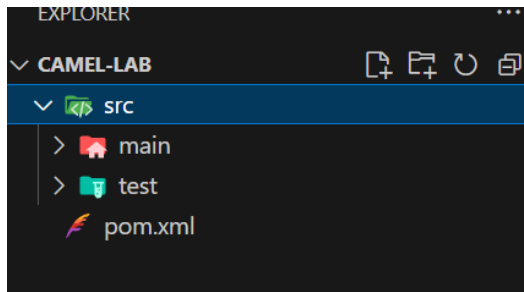
1. Paso a paso para desplegar la solución

- a. Antes de iniciar la implementación del flujo de integración con Apache Camel, se verificó que el entorno de trabajo cumpla con los requisitos técnicos mínimos necesarios para ejecutar el taller. Se descargó la versión **Apache Camel 4.14.1 LTS**, compatible con Java 17, desde el sitio oficial del proyecto. Esta versión fue seleccionada por su estabilidad y soporte extendido, garantizando compatibilidad con el entorno académico de desarrollo.

- b. Una vez instalado Apache Camel, Se generó un nuevo proyecto Maven utilizando el comando `mvn archetype:generate`, definiendo el identificador de grupo `edu.udla.isw` y el artefacto `camel-lab`.

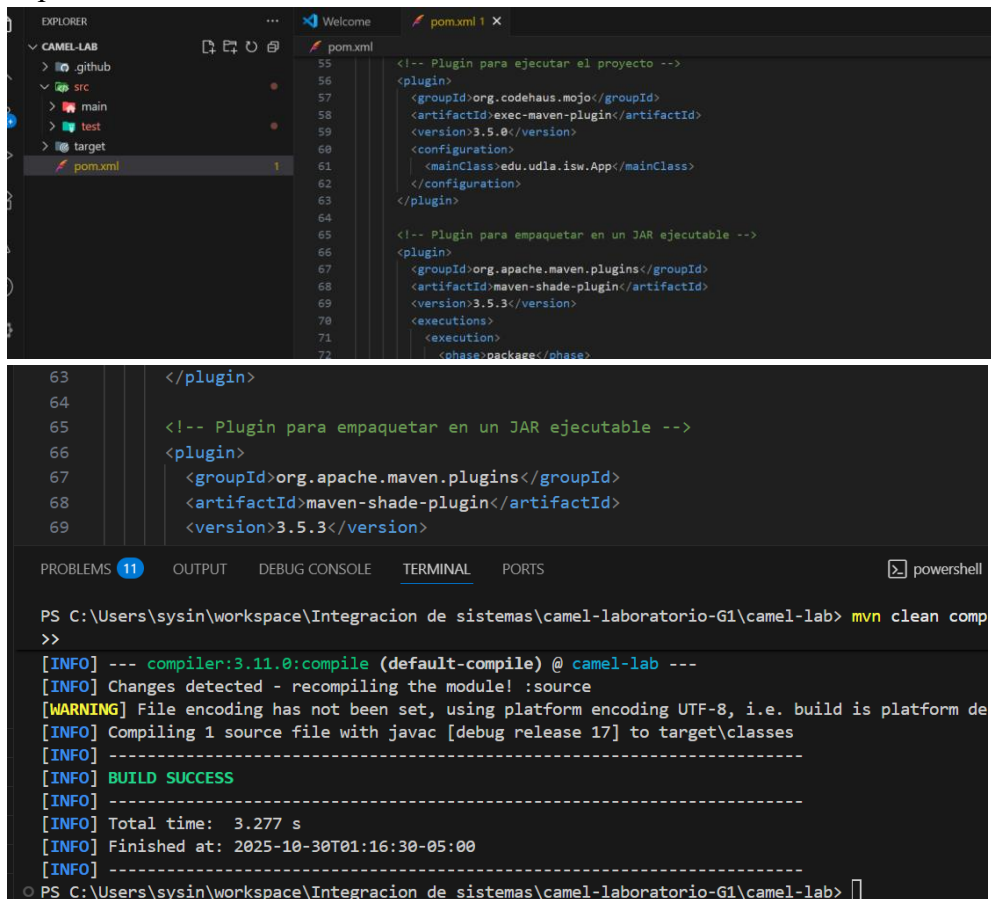
Este proceso creó automáticamente la estructura estándar del proyecto con las carpetas `src/main/java` y `src/test/java`, además del archivo `pom.xml`, que será utilizado posteriormente para configurar las dependencias de Apache Camel. Con esta configuración inicial se estableció la base para el desarrollo del flujo de integración que automatizará la transferencia de archivos.

```
/maven-archetype-quickstart-1.0.jar (4.3 kB at 14 kB/s)
[INFO] -----
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0
[INFO] -----
[INFO] Parameter: basedir, Value: C:\Users\sysin\workspace\Integracion de sistemas\camel-laboratorio-G1
[INFO] Parameter: package, Value: edu.udla.isw
[INFO] Parameter: groupId, Value: edu.udla.isw
[INFO] Parameter: artifactId, Value: camel-lab
[INFO] Parameter: packageName, Value: edu.udla.isw
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\Users\sysin\workspace\Integracion de sistemas\camel-laboratorio-G1\camel-lab
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 18.482 s
[INFO] Finished at: 2025-10-30T01:07:16-05:00
[INFO] -----
C:\Users\sysin\workspace\Integracion de sistemas\camel-laboratorio-G1>
```



- c. Se configuró el archivo pom.xml para incluir las dependencias esenciales de **Apache Camel** (camel-core, camel-main, camel-file) y el sistema de registro slf4j-simple.

Además, se añadieron los plugins de compilación, ejecución y empaquetado (maven-compiler-plugin, exec-maven-plugin y maven-shade-plugin). Con esta configuración, Maven puede compilar el proyecto, ejecutar el flujo Camel directamente y generar un archivo .jar ejecutable con todas las dependencias incluidas.



- d. Una vez configurada la clase App.java con la ruta de integración en Apache Camel, se procedió a compilar el proyecto mediante el comando `mvn clean compile`.

Maven descargó las dependencias necesarias, compiló el código fuente y generó los archivos correspondientes en la carpeta `target`. Al finalizar el proceso, se obtuvo el mensaje **BUILD SUCCESS**, confirmando que la configuración del entorno y la implementación de la ruta fueron exitosas.

```
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 7.869 s  
[INFO] Finished at: 2025-04-06T11:18:58-05:00  
[INFO] -----
```

se preparó la estructura de carpetas donde se ejecutará el flujo de integración. Estas carpetas simulan el intercambio de información entre los sistemas de ventas e inventario, representando un proceso real de transferencia de archivos.

Para ello, dentro de la carpeta de trabajo principal (C:\Cursos\IntegracionSistemas\camel-lab\) se crearon las siguientes subcarpetas:

input

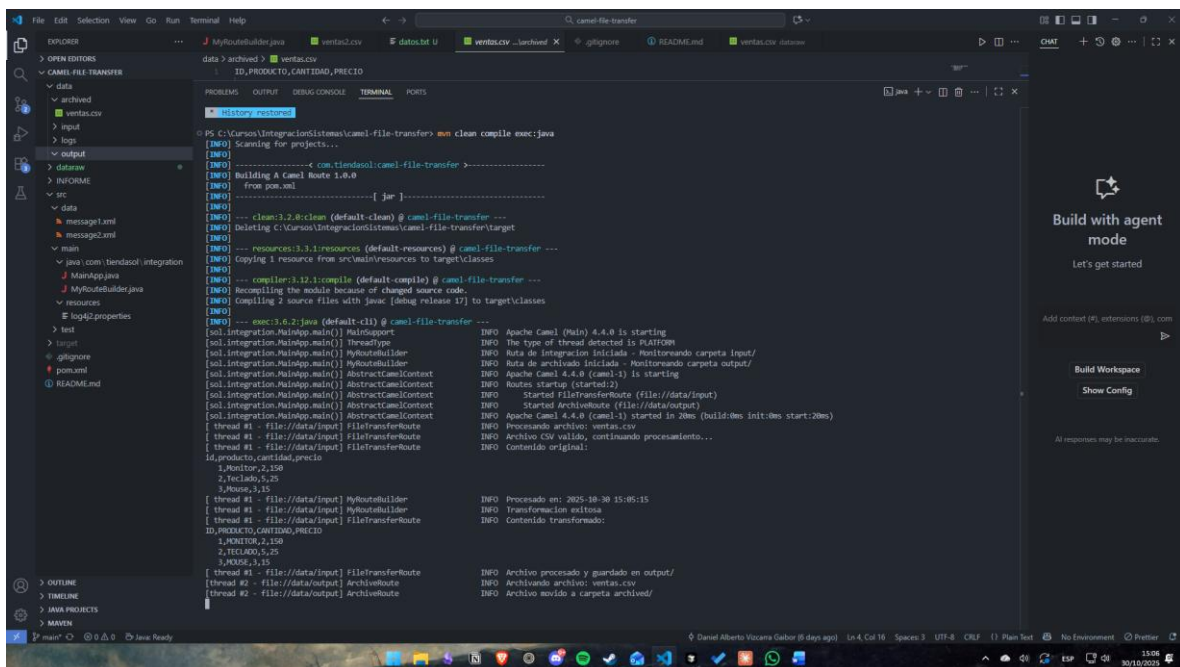
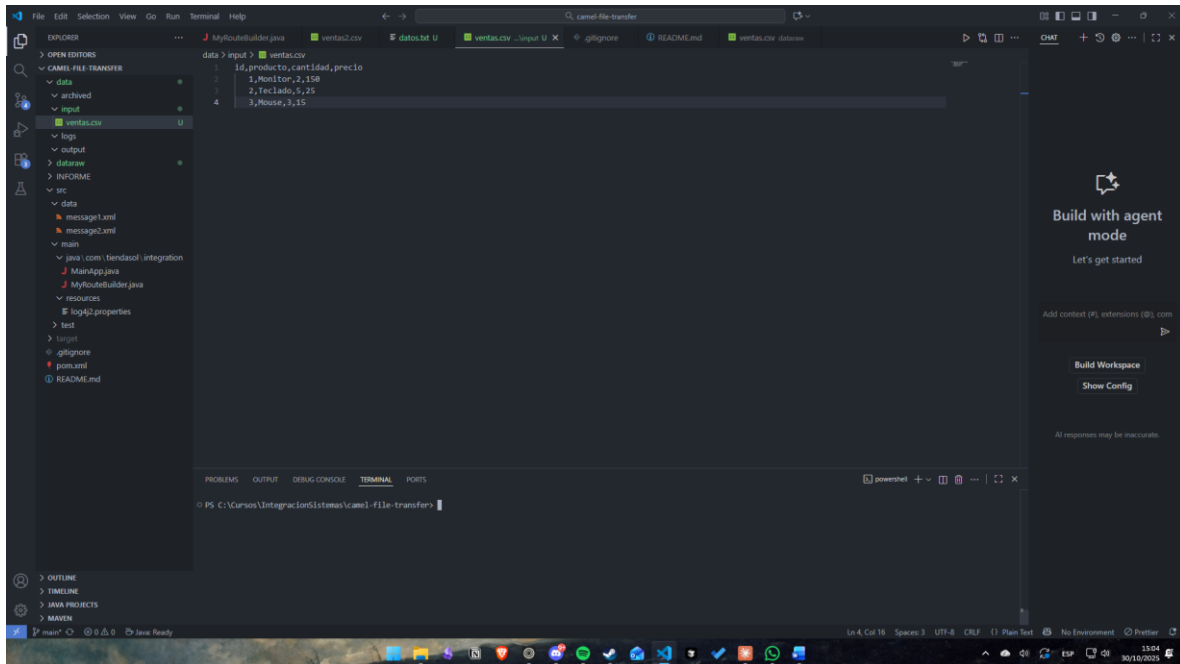
output

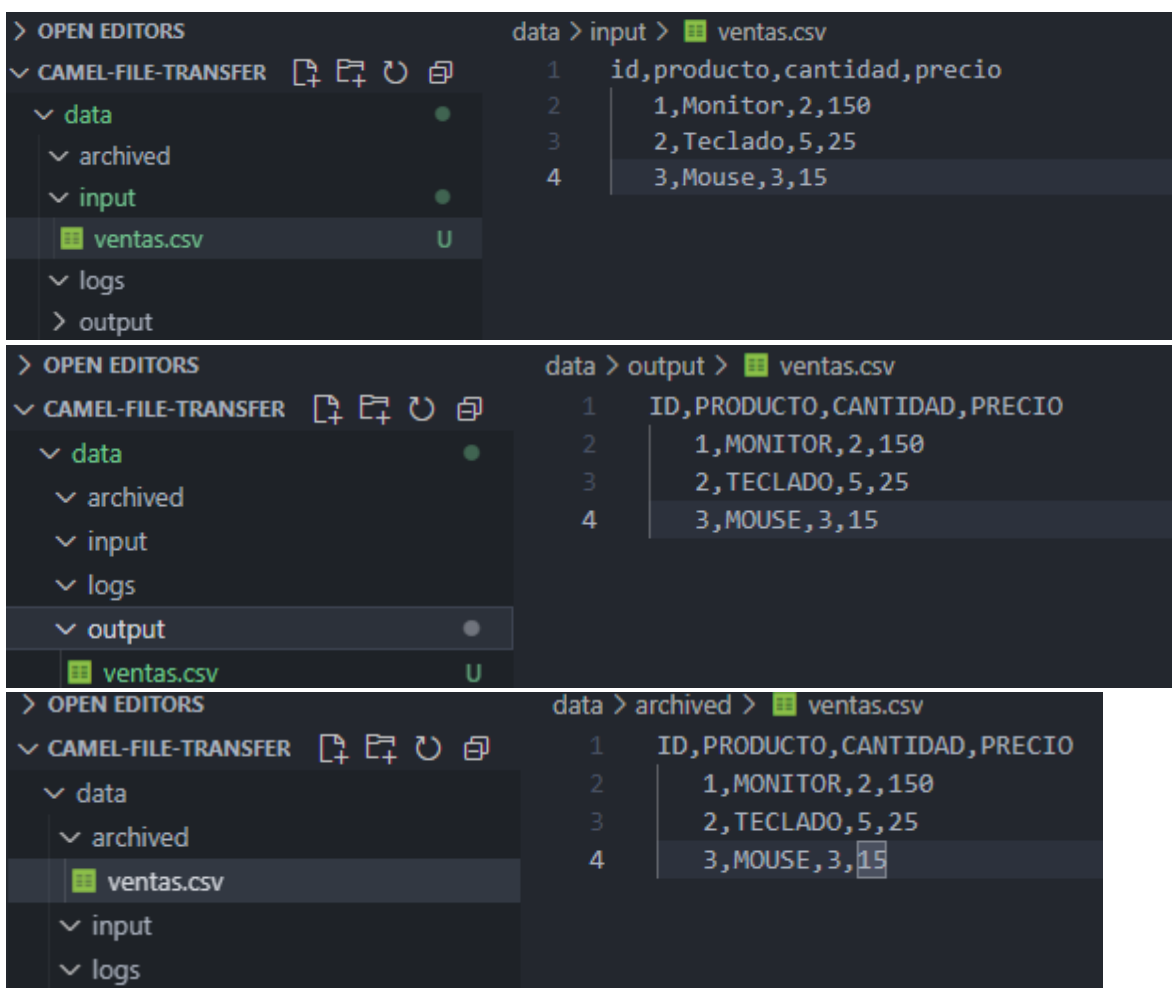
logs

- input: almacena los archivos de entrada (CSV) generados por el sistema de ventas.
- output: recibe los archivos procesados por Camel.
- logs: guarda los registros o mensajes generados durante la ejecución del flujo.

2. Pruebas y resultados

Logs completos de ejecución en terminal: Para empezar con la prueba colocamos un archivo csv, en este caso `ventas.csv` en la carpeta `input` y ejecutamos el proyecto.





The image displays three sequential screenshots of an IDE interface, illustrating the processing of a CSV file named 'ventas.csv'.

Top Screenshot: The file explorer on the left shows the directory structure: 'data' > 'input' > 'ventas.csv'. The editor window displays the original CSV data:

```
1 id,producto,cantidad,precio
2 1,Monitor,2,150
3 2,Teclado,5,25
4 3,Mouse,3,15
```

Middle Screenshot: The file explorer shows the file has been moved to 'data' > 'output' > 'ventas.csv'. The editor window shows the transformed data with uppercase headers and values:

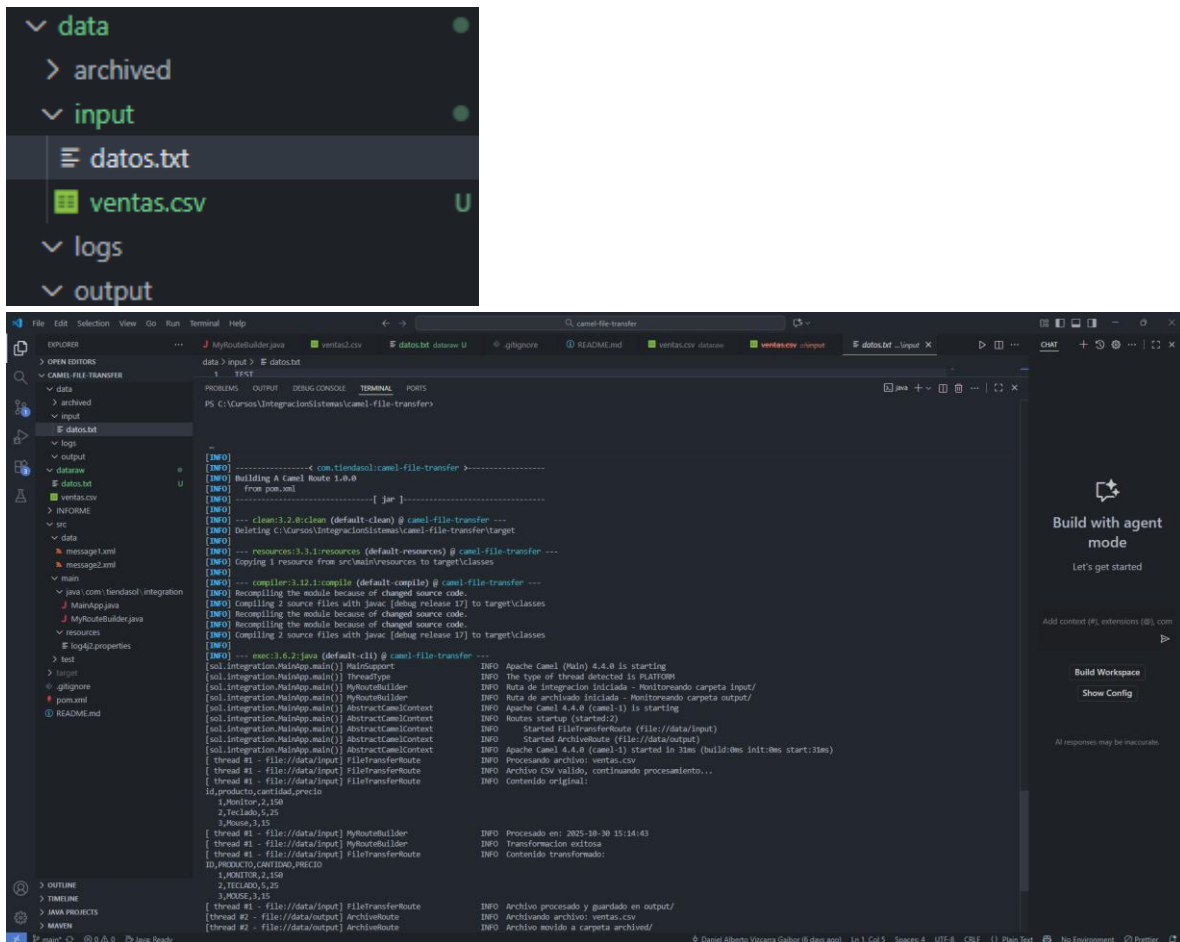
```
1 ID,PRODUCTO,CANTIDAD,PRECIO
2 1,MONITOR,2,150
3 2,TECLADO,5,25
4 3,MOUSE,3,15
```

Bottom Screenshot: The file explorer shows the file has been moved to 'data' > 'archived' > 'ventas.csv'. The editor window shows the final archived data:

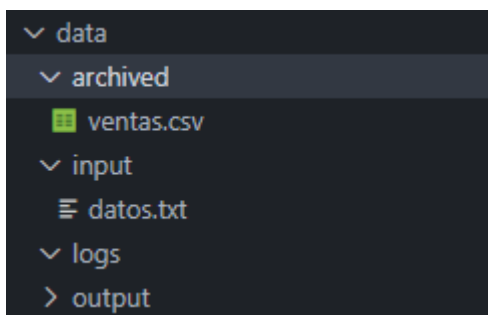
```
1 ID,PRODUCTO,CANTIDAD,PRECIO
2 1,MONITOR,2,150
3 2,TECLADO,5,25
4 3,MOUSE,3,15
```

Se observan los logs de ejecución donde el archivo ha sido transformado correctamente y se lo ha clasificado en las carpetas correspondientes. Como no han existido errores luego de pasar por la carpeta output se mueve el archivo a la carpeta archived. También podemos observar en los logs que se ha agregado hora y fecha en la cual se han procesado los archivos.

Prueba del filtro .csv: Se ha agregado la funcionalidad para que solo procese archivos con extensión .csv, para la siguiente prueba agregaremos dos archivos un csv y un archivo txt, con el fin de demostrar que el sistema solo tomará en cuenta al archivo csv e ignorará al txt, esto también se verá reflejado en los logs de ejecución.



Como fue planteado previamente y lo demuestran los logs de ejecución, solo se procesó el archivo con extensión .csv, el mismo que paso por las carpetas output ara luego moverse a archived mientras que el archivo .txt permanece en input. Con esto hemos cubierto todo el flujo del sistema.



3. Link a repositorio GIT

<https://github.com/daniel-vizcarra/camel-file-transfer-grupo1.git>

4. Preguntas

¿Qué representa el patrón “File Transfer”?

El patrón File Transfer es un método clásico de integración de sistemas que permite el intercambio de información entre aplicaciones mediante el envío y recepción de archivos. En lugar de una comunicación directa o en tiempo real, los sistemas comparten archivos en un formato común (por ejemplo, CSV, XML o JSON) a través de una carpeta o ubicación compartida.

Este patrón se caracteriza por su simplicidad y bajo acoplamiento, ya que cada sistema puede operar de forma independiente: uno genera los archivos y otro los procesa posteriormente. Es ideal cuando los sistemas involucrados son antiguos o no cuentan con APIs modernas, permitiendo una integración funcional sin necesidad de modificar su arquitectura interna. En el taller, este patrón se implementó a través de Apache Camel, automatizando el proceso de lectura, copia y registro de archivos de manera continua, eliminando la necesidad de intervención manual.

¿Qué limitaciones tiene este enfoque?

En primer lugar, no opera en tiempo real, ya que existe un desfase entre la creación y el procesamiento del archivo, lo que genera latencia en entornos que requieren inmediatez. Su manejo de errores es limitado, dificultando la detección automática de archivos incompletos o con formato incorrecto. Además, su escalabilidad es reducida, pues con grandes volúmenes de información puede volverse lento o consumir mucho espacio de almacenamiento. Tampoco ofrece confirmación automática, por lo que el sistema origen no sabe si el archivo fue recibido y procesado correctamente. A esto se suma una seguridad restringida, ya que los archivos pueden quedar expuestos si no se aplican medidas como cifrado o control de accesos. Finalmente, al operar en modo batch, no permite sincronización transaccional ni actualizaciones incrementales de datos. Aun así, estas limitaciones son aceptables en entornos donde la simplicidad y confiabilidad son más importantes que la inmediatez.

¿En qué escenarios reales sería útil?

Es ideal para la integración de sistemas antiguos, como un sistema contable que exporta reportes diarios en CSV para ser procesados por otro sistema. También se aplica en procesamiento nocturnos, como la consolidación de ventas o actualización de inventarios fuera del horario laboral, y en intercambios B2B, donde empresas comparten archivos de pedidos, facturación o inventario mediante FTP o SFTP. Además, resulta efectivo en migraciones o respaldos de datos, permitiendo la transferencia de información entre plataformas, y en procesos ETL, donde se extraen, transforman y cargan datos hacia un almacén analítico. Este patrón también es útil cuando no existe conectividad directa entre sistemas y el intercambio de archivos es la única alternativa viable. En el contexto del taller, su implementación permitió simular la integración entre un sistema de ventas y uno de inventario, automatizando la transferencia de información de forma simple, eficiente y robusta. Es ideal para la integración de sistemas antiguos, como un sistema contable que exporta reportes diarios en CSV para ser procesados por otro sistema. También se aplica en procesamiento nocturnos, como la consolidación de ventas o actualización de inventarios fuera del horario laboral, y en intercambios B2B, donde empresas comparten archivos de pedidos, facturación o inventario mediante FTP o SFTP. Este patrón también es útil cuando no existe conectividad directa entre sistemas y el intercambio de archivos es la única alternativa viable. En el contexto del taller, su implementación permitió simular la integración entre un sistema de ventas y uno de inventario, automatizando la transferencia de información de forma simple, eficiente y robusta.

5. Conclusiones y Recomendaciones

Conclusiones:

- Se logró implementar de forma satisfactoria un flujo de integración utilizando Apache Camel, aplicando el patrón File Transfer, lo que permitió automatizar el proceso de lectura, transformación y movimiento de archivos entre diferentes carpetas.
- El uso de Maven facilitó la gestión de dependencias y la construcción del proyecto, garantizando una compilación limpia y reproducible en cualquier entorno.
- La herramienta Apache Camel demostró su potencia al simplificar tareas complejas de integración mediante una sintaxis declarativa, mostrando cómo los Enterprise Integration Patterns (EIP) pueden aplicarse de forma práctica.
- El ejercicio permitió comprender cómo los flujos de integración basados en archivos pueden conectar sistemas heterogéneos sin requerir comunicación directa o en tiempo real.

- La implementación evidenció la importancia de definir una estructura clara de carpetas (input, output, archived, logs) y mantener un registro detallado del procesamiento mediante logs.
- En general, se cumplió el objetivo del taller al demostrar cómo la automatización mejora la eficiencia, reduce errores humanos y permite un control más confiable del intercambio de datos entre sistemas.

Recomendaciones:

- Implementar validaciones previas para asegurar que los archivos tengan el formato y extensión correcta antes de ser procesados.
- Agregar un sistema de logs persistentes y detallados, que registre fecha, hora, nombre del archivo y resultados del procesamiento, para mejorar la trazabilidad y auditoría.
- Incorporar mecanismos de seguridad como cifrado de archivos y permisos de acceso, especialmente en entornos empresariales o con información sensible.
- Añadir notificaciones automáticas (por correo o consola) al completar un procesamiento o al detectar errores, facilitando el monitoreo continuo.
- Incluir tolerancia a fallos y reintentos automáticos, permitiendo que el sistema se recupere ante errores de lectura o escritura sin intervención manual.
- Optimizar la frecuencia de monitoreo y considerar la integración con sistemas de monitoreo externos como Prometheus o Grafana para analizar el rendimiento.
- Extender el flujo implementando transformaciones adicionales (como limpieza de datos o validación de campos), haciéndolo aplicable a casos de uso más complejos.
- En proyectos futuros, evaluar el uso de otros patrones de integración más avanzados (como Message Queue o API Gateway) según las necesidades de sincronización y tiempo real.