PROYECTO TRANSIAP

3.1. Módulo de reporte y generación de ubicaciones geográficas:

3.1.1. Marco Contextual y Objetivos del Subsistema

Dentro de la arquitectura de integración del proyecto TransIAP, el subsistema de reporte de ubicaciones constituye el componente crítico para la ingesta de telemetría en tiempo real. La flota de transporte autorizada por el Sistema Nacional de Transportes para Navieras (SNTN) presenta un entorno tecnológico heterogéneo, donde coexisten dispositivos de diversas generaciones.

El objetivo primordial de este módulo es garantizar la interoperabilidad en la capa de entrada (Inbound Channel Adapters), normalizando la recepción de datos provenientes de tres tipos de fuentes:

1. Dispositivos Legados: Emisión mediante texto plano bajo un formato propietario (TransIAP-CSV).
2. Dispositivos Estándar: Representación geográfica basada en el estándar XML/KML.
3. Dispositivos de Última Generación: Implementación sobre el estándar GeoJSON.

Para validar el sistema, se han desarrollado tres aplicaciones independientes en lenguaje Java (denominadas "Generadores"), que actúan como Productores en la infraestructura de mensajería RabbitMQ. Estos componentes automatizan la publicación de coordenadas GPS y matrículas, asegurando que cada mensaje cumpla con las restricciones sintácticas de su formato respectivo.

3.1.2. Arquitectura de Integración Basada en Mensajería (RabbitMQ)

Con el fin de asegurar un desacoplamiento efectivo entre los dispositivos emisores y el sistema de procesamiento central (Soporte Logística Live), se ha diseñado una arquitectura basada en el patrón de intercambio Topic Exchange.

Especificaciones del Intercambio:
• **Identificador del Exchange:** transiap.gps
• **Patrón de Enrutamiento:** topic
• **Justificación Técnica:** La elección de un exchange de tipo topic proporciona la máxima flexibilidad. Los generadores etiquetan los mensajes con una Routing Key que identifica el formato de origen. Esto permite al subsistema consumidor realizar un filtrado selectivo *(por ejemplo, suscribiéndose a **gps.*** para una recepción total o a una clave específica para procesos de auditoría diferenciados)* sin necesidad de modificar la lógica de los emisores.

Tabla de Enrutamiento:
- Origen KML: gps.kml
- Origen GeoJSON: gps.json
- Origen CSV: gps.formato.csv

3.1.3. Especificaciones Técnicas y Normalización de Formatos

A continuación, se describen los protocolos de representación de datos implementados y las decisiones de diseño adoptadas en el desarrollo de las aplicaciones Java.

A. Protocolo basado en XML (Estándar KML)
Para la implementación del generador GeneradorGeoKML, se ha utilizado la API nativa Java DOM (Document Object Model). Se ha priorizado este enfoque sobre la concatenación de cadenas simple para asegurar la generación de documentos XML bien formados, cumpliendo estrictamente con la estructura de etiquetas <Placemark>, <Point> y la extensión de metadatos para la identificación del vehículo.

B. Protocolo basado en JSON (Estándar GeoJSON)
El generador GeneradorGeoJSON sigue la especificación moderna de objetos geográficos. Para su desarrollo se ha integrado la librería JSON-Java, lo que garantiza que los objetos sean serializados de forma robusta, manteniendo la integridad de los tipos de datos (numéricos para coordenadas y cadenas para identificadores).

C. Formato Propietario TransIAP-CSV
Para dispositivos de baja capacidad computacional, se mantiene el soporte de texto plano. Dado que este formato carece de metadatos estructurales, la validez de la información se sustenta en la integridad posicional de los campos (Matrícula, Latitud, Longitud).

3.1.4. Detalles de Implementación y Robustez

El desarrollo de las aplicaciones se ha regido por los siguientes criterios técnicos:

- **Conectividad:** Uso de la librería industrial amqp-client-4.0.2.
- **Tratamiento de Datos:** Implementación de rutinas de sanitización de entradas (limpieza de caracteres especiales en matrículas) y validación de tipos numéricos mediante gestión de excepciones.
- **Compatibilidad:** Los artefactos han sido compilados para asegurar la ejecución en entornos con Java 8 o superior, maximizando su portabilidad en infraestructuras heterogéneas.

3.2. Módulo de Procesamiento (Middleware "Soporte Logística")

Este componente es el núcleo central del sistema. Se trata de una aplicación Java de consola (exportada como Runnable JAR) que actúa como puente entre la recepción de datos brutos y su almacenamiento final. Su función principal es unificar los distintos formatos de entrada en uno solo estandarizado.

3.2.1. Configuración Dinámica y Despliegue

Para garantizar la portabilidad entre entornos de desarrollo y producción, la aplicación no utiliza valores estáticos ("hardcoded") obligatorios. Se ha implementado un sistema de **argumentos por línea de comandos** con valores por defecto:

1. **Modo por Defecto:** Si se ejecuta sin argumentos, el sistema asume una configuración local estándar (RabbitMQ en 127.0.0.1 y la URL oficial del SNTN), facilitando las pruebas rápidas.
2. **Modo Configurable:** Permite la inyección de parámetros externos al iniciar el JAR (args[0] para la IP del broker y args[1] para la URL de la API), lo que permite desplegar el Middleware en servidores remotos sin modificar el código fuente.

3.2.2. Configuración de Mensajería (RabbitMQ)

El Middleware actúa simultáneamente como consumidor y productor:

- **Entrada (Consumidor):** Se conecta al Exchange *transiap.gps*. Utiliza una cola temporal con la binding key gps.# para recibir **todos** los mensajes, sin importar si vienen de los generadores CSV, JSON o KML.
- **Salida (Productor):** Redirige la información procesada hacia el Fanout Exchange *traslados.localizaciones*. Se ha seleccionado el tipo Fanout para permitir la difusión simultánea (broadcasting) a múltiples consumidores (Base de Datos, Dashboard, Auditoría) sin acoplamiento lógico.

3.2.3. Lógica de Normalización y Enriquecimiento

El flujo de ejecución por cada mensaje recibido consta de tres fases secuenciales:

1. **Lectura y Extracción de Datos:** El sistema detecta automáticamente si el mensaje llega en formato CSV, JSON o KML. Dependiendo del tipo, "abre" el mensaje y extrae los datos necesarios: la matrícula del vehículo y sus coordenadas (latitud y longitud).
2. **Conexión con la API (Enriquecimiento):** Una vez identificada la matrícula, el Middleware se conecta al servidor del SNTN para pedir dos datos adicionales:
   - La **clave de autorización** (auth) correspondiente a ese vehículo.
   - La **hora oficial** (timestamp) del servidor para tener un registro temporal exacto.
3. **Control de Errores:** El código está protegido para evitar caídas. Si la conexión con la API falla o hay problemas de red, el sistema captura el error y continúa funcionando, asegurando que el flujo de datos hacia la base de datos nunca se detenga.

3.3. Módulo de Registro en Base de Datos

Este componente consiste en una aplicación Java (Runnable JAR) que actúa como consumidor final en la arquitectura, escuchando el exchange traslados.localizaciones. Su responsabilidad principal es garantizar la persistencia de los datos, registrando cada ubicación validada en el histórico y manteniendo actualizado el estado en tiempo real de los transportes.

Para facilitar la portabilidad entre entornos (desarrollo, pre-producción, etc.), la aplicación no mantiene credenciales fijas en el código fuente. En su lugar, recibe los parámetros de conexión a través de argumentos de línea de comandos (CLI) durante su ejecución. Esto permite configurar dinámicamente:

- El **host** del servidor de mensajería (**RabbitMQ**).
- Los parámetros de conexión a la base de datos (**Host**, **Puerto**, **Usuario** y **Contraseña**).

El flujo de procesamiento implementado utilizando la librería STC-DAO es el siguiente:

- **Recepción y Parseo:** El módulo consume el mensaje JSON estandarizado y extrae la matrícula y coordenadas.
- **Vinculación:** Invoca TrasladoDAO.getTrasladoActivoPorVehiculo(matricula) para identificar el expediente de transporte activo asociado al vehículo.
- **Inserción Histórica:** Genera una nueva entrada en la tabla de seguimiento mediante LocalizacionGPSDAO.saveLocalizacionGPS().
- **Sincronización:** Actualiza el puntero de última ubicación en la tabla maestra de traslados usando trasladoDAO.updateUltimaLocalizaionTraslado().

3.4. Visualización de Mensajes: Componente Visualizador

En este capítulo se describe el desarrollo y la lógica operativa del componente **Visualizador**, diseñado para cumplir con el requisito de monitorización del sistema TransIAP-STC. Este componente actúa como un auditor de los datos procesados, permitiendo validar la integridad de la información enriquecida antes y después de su almacenamiento en la base de datos.

3.4.1. Descripción General

El componente **Visualizador** es una aplicación Java independiente encargada de consumir mensajes desde el exchange central de salida, denominado traslados.localizaciones. Su objetivo fundamental es proporcionar una representación textual clara del formato unificado **TransIAP-loc/JSON**, asegurando que cada reporte incluya los campos obligatorios obtenidos mediante el middleware: coordenadas, matrícula del vehículo, clave de autenticación (auth) y marca temporal (timestamp) centralizada.

3.4.2. Configuración del Exchange y Mensajería

La infraestructura de mensajería del visualizador se basa en los siguientes pilares técnicos:

- **Exchange:** traslados.localizaciones.
- **Tipo: Fanout**. Este diseño es crítico para el sistema, ya que permite que múltiples consumidores (como el RegistroBD y el propio Visualizador) reciban la misma información de forma simultánea y desacoplada, facilitando la escalabilidad del sistema.
- **Gestión de Colas:** El componente declara una cola anónima, exclusiva y temporal (channel.queueDeclare().getQueue()). Esta configuración garantiza que el Visualizador reciba únicamente los mensajes publicados mientras la aplicación está en ejecución, sin generar persistencia innecesaria en el bróker RabbitMQ.

3.4.3. Flujo de Operación

El funcionamiento de Visualizador.java sigue una secuencia asíncrona robusta:

1. **Establecimiento de Conexión:** Se conecta al servidor RabbitMQ definiendo una ConnectionFactory para gestionar el ciclo de vida del canal de comunicación.
2. **Suscripción:** Declara el exchange y vincula la cola temporal, quedando en estado de escucha activa mediante el método basicConsume.
3. **Procesamiento JSON:** Al recibir una entrega, el cuerpo del mensaje se transforma a cadena UTF-8 y se parsea utilizando la librería org.json.
4. **Salida Estructurada:** Como se observa en la Figura 5.2, el mensaje se imprime en consola con una indentación de 4 espacios (json.toString(4)), permitiendo una inspección técnica rápida de los metadatos generados por el middleware.
5. **Persistencia del Proceso:** Se implementa un CountDownLatch para mantener el proceso activo para la recepción continua de datos hasta la interrupción manual.

3.5. Orquestación y Resultados

Este capítulo detalla la estrategia de despliegue automatizado diseñada para la gestión de la arquitectura distribuida y analiza los indicadores de éxito obtenidos durante las pruebas de integración de los componentes del ecosistema TransIAP.

3.5.1. Script de Orquestación: LanzarSistema.ps1

En cumplimiento con las **instrucciones de entrega** (Sección 9) relativas a la simplificación del despliegue y evaluación, se ha implementado el script LanzarSistema.ps1 en PowerShell. Esta herramienta centraliza la gestión del ciclo de vida de los servicios mediante las siguientes capacidades técnicas:

- **Despliegue de Servicios Core:** El script automatiza la instanciación de los servicios críticos (Monitor Visualizador, Registro en BD y Middleware de Soporte Logística) realizando una **parametrización dinámica** de los puntos de conexión al broker y las credenciales de la base de datos relacional (Host, Puerto, Usuario y Contraseña).
- **Sincronización de Procesos:** Para asegurar la estabilidad de la red y garantizar que los consumidores estén en estado de escucha activa, el script implementa una pausa de seguridad de 2 segundos antes de iniciar los generadores de telemetría.
- **Generación Automatizada de Telemetría:** Tras la pausa de sincronización, se procede al arranque de los tres generadores de protocolo (CSV, GeoJSON y KML), los cuales inyectan vectores de datos reales correspondientes a los vehículos autorizados.
- **Organización Avanzada de Ventanas**: El script utiliza la API de Windows (User32.dll) para organizar las seis ventanas en una cuadrícula predefinida. Como se evidencia en la **Figura 5.1**, esto permite una monitorización simultánea de toda la arquitectura lógica propuesta en el esquema general del proyecto.

3.5.2. Resultados tras la Ejecución

La validación del sistema bajo carga de trabajo demuestra la robustez de la integración asíncrona y el flujo de datos unidireccional entre productores y consumidores:

- **Convergencia de Protocolos:** Se verifica que los componentes generadores transmiten con éxito las tramas en sus formatos nativos hacia el exchange de entrada, cumpliendo estrictamente con las especificaciones de serialización y los contratos de datos requeridos .
- **Procesamiento y Enriquecimiento en el Middleware:** La instancia de SoporteLogistica (**Figura 5.4**) evidencia la capacidad del middleware para normalizar la entrada multiformato, realizar las peticiones síncronas a la API REST de la SNTN y publicar el mensaje enriquecido bajo el estándar **TransIAP-loc/JSON** .
- **Persistencia y Consistencia de Datos:**
    - El Servicio de RegistroBD (**Figura 5.3**) confirma la conexión persistente con el esquema de base de datos stc y la ejecución exitosa de las operaciones

de inserción y actualización mediante la capa DAO proporcionada, vinculando cada coordenada al traslado activo correspondiente.

- El Monitor Visualizador (**Figura 5.2**) corrobora la integridad del payload final, validando que el flujo de integración respeta los requisitos de seguridad (AppKey) y sincronización temporal (Timestamp) establecidos por la organización.

Figura 5. 1

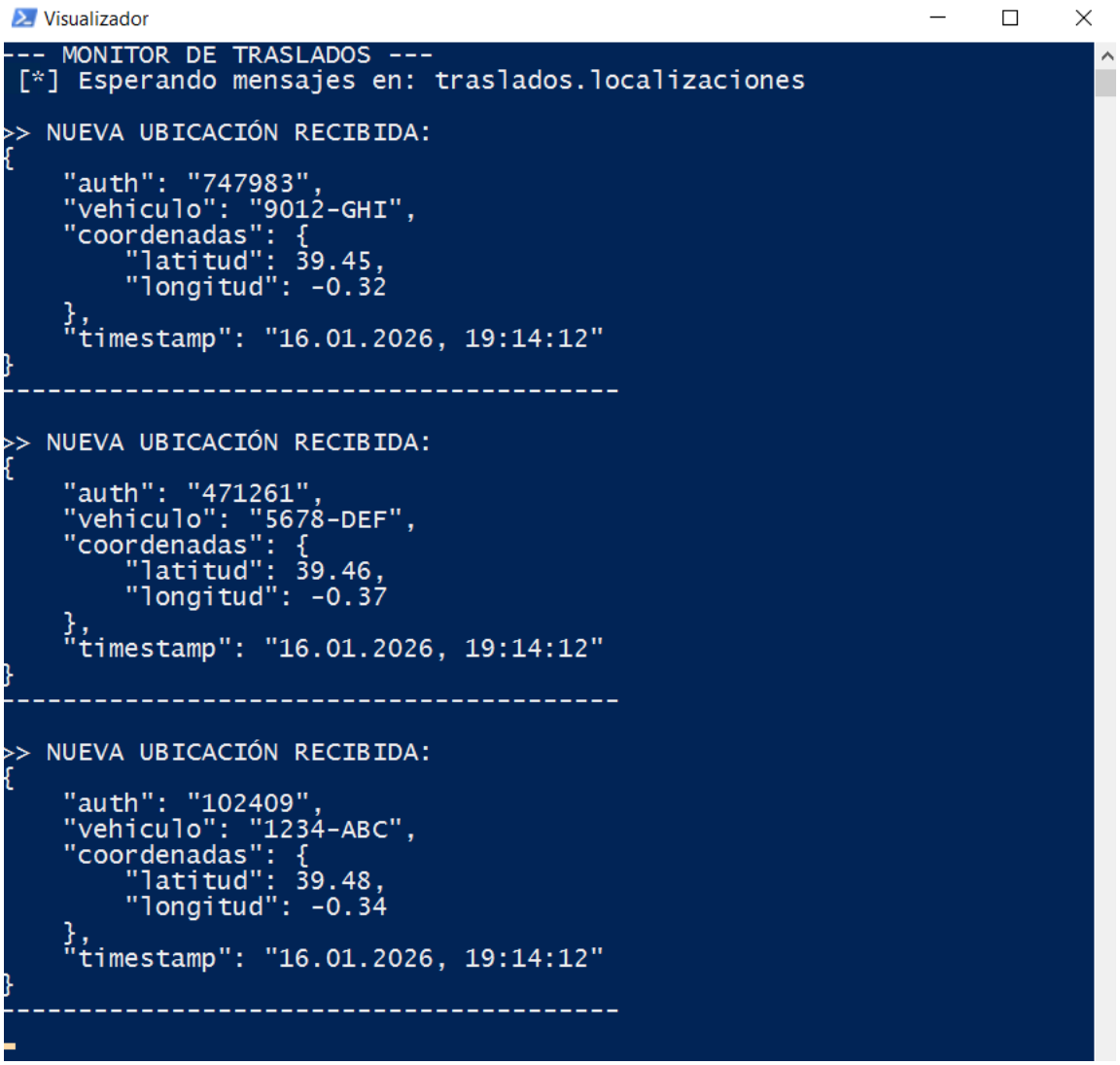

Figura 5. 2

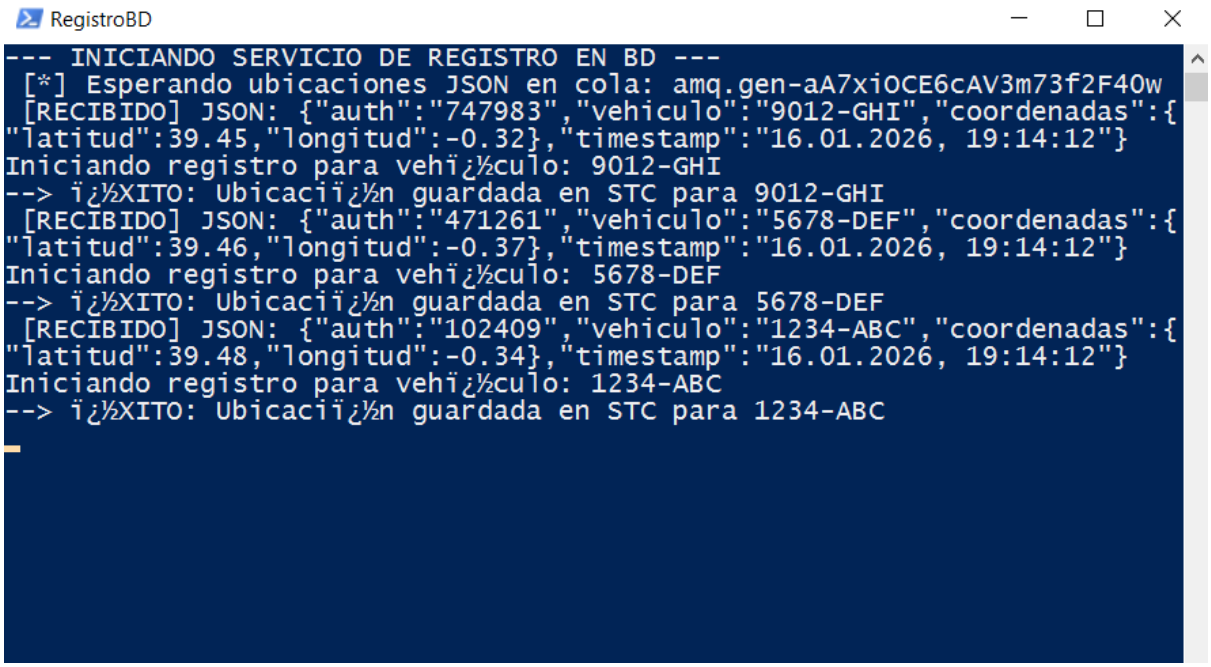

Figura 5. 3

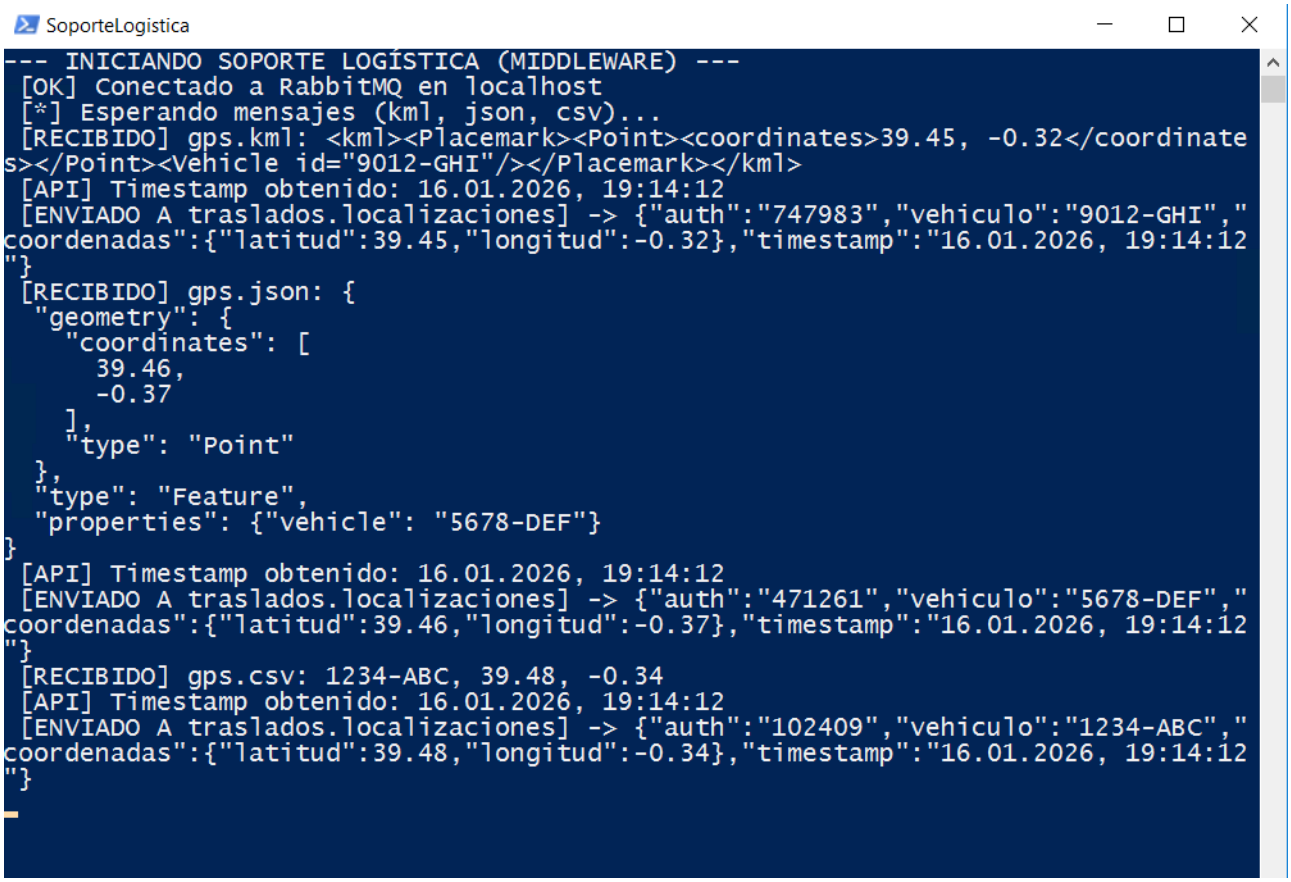

Figura 5. 4

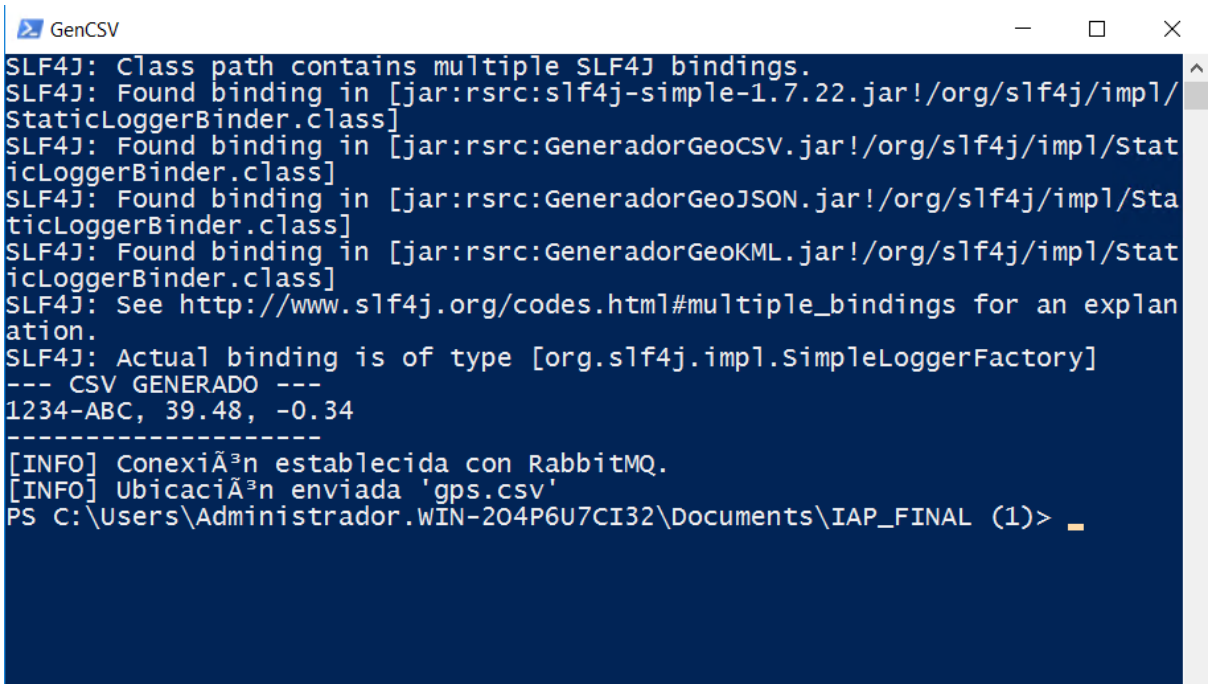

Figura 5. 5

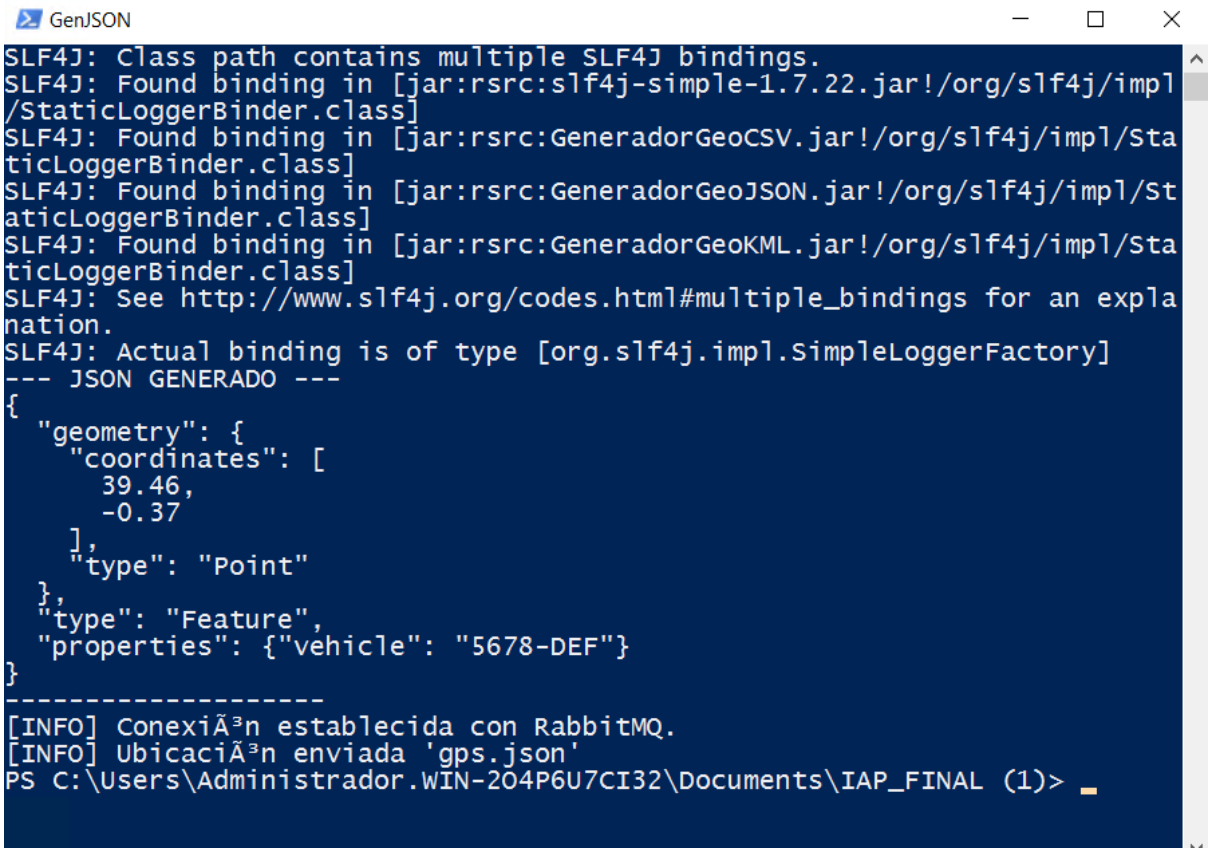

Figura 5. 6

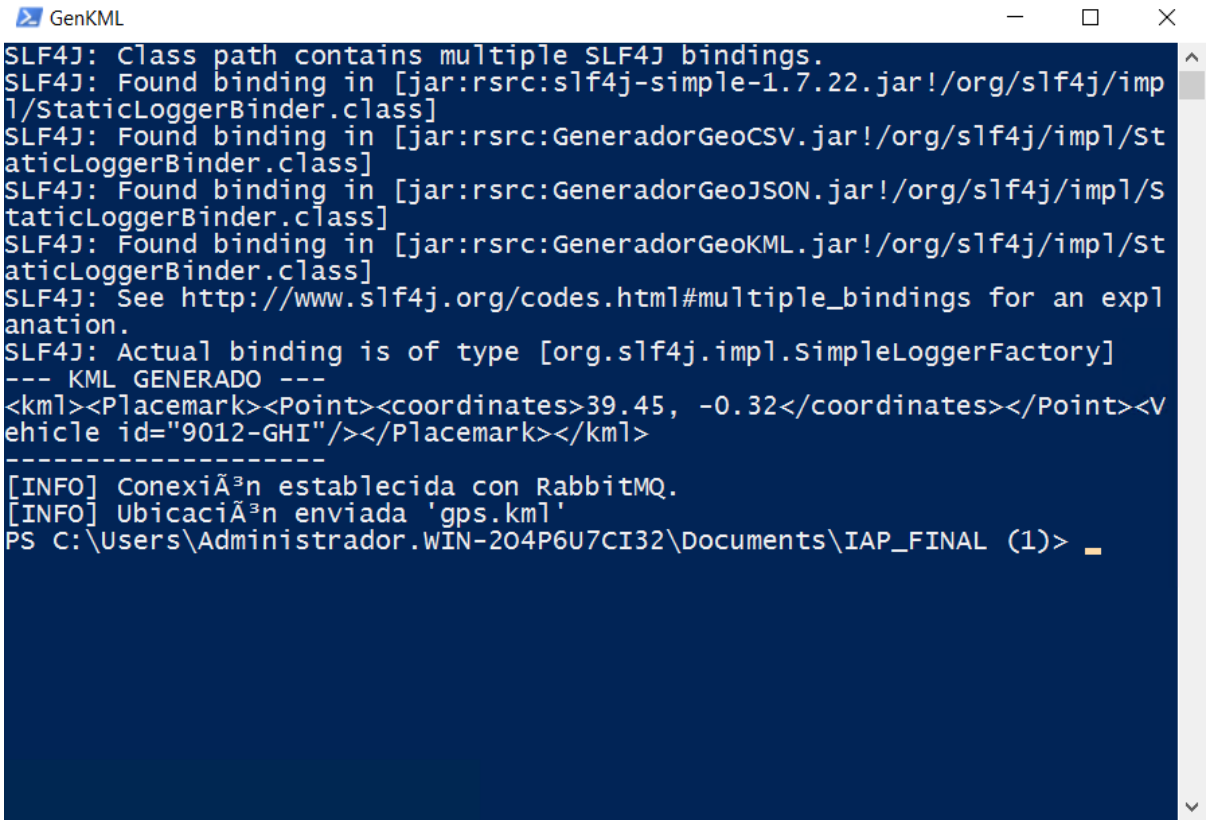

Figura 5. 7

3.6. Conclusiones

La implementación del sistema **TransIAP-STC** representa una solución integral y robusta al desafío de la interoperabilidad en entornos logísticos heterogéneos. Tras el diseño, desarrollo y validación de los diversos módulos que componen la arquitectura, se pueden extraer las siguientes determinaciones técnicas:

- **Adaptabilidad y Normalización de Datos**: La implementación de tres generadores independientes para los formatos **KML, GeoJSON y TransIAP-CSV** asegura la compatibilidad con una flota tecnológica diversa, permitiendo la ingesta de telemetría desde dispositivos de distintas generaciones sin pérdida de información.
- **Eficiencia en el Flujo de Mensajería**: El uso estratégico de RabbitMQ como middleware destaca por garantizar un flujo organizado mediante el uso de exchanges específicos. La configuración del exchange de tipo *topic* (transiap.gps) permite un enrutamiento selectivo de los formatos de origen, mientras que el exchange *fanout* (traslados.localizaciones) asegura la distribución simultánea de los datos procesados a los componentes de visualización y registro.
- **Valor Añadido mediante el Enriquecimiento**: El módulo Soporte Logística 'Live' actúa como el núcleo de inteligencia del sistema. Su integración con la API REST del SNTN aporta un valor significativo al proporcionar una firma de autenticidad (auth) y una marca temporal (timestamp) centralizada, lo que garantiza la trazabilidad y seguridad de cada traslado registrado en la base de datos STC.
- **Orquestación y Supervisión**: La solución destaca por su escalabilidad y flexibilidad. El componente Visualizador permite validar la integración en tiempo real, brindando una herramienta de supervisión intuitiva. Asimismo, la automatización lograda mediante el script LanzarSistema.ps1 eleva la calidad profesional del software, permitiendo un despliegue dinámico y eficiente mediante argumentos de línea de comandos.

En conclusión, el proyecto no solo cumple con los requisitos técnicos exigidos, sino que establece una infraestructura de integración moderna y confiable. El éxito en la sincronización de ubicaciones en tiempo real refuerza la importancia de utilizar arquitecturas basadas en mensajería en el diseño de soluciones de ingeniería informática, proporcionando un modelo efectivo y modular para la futura expansión de la red logística de TransIAP.