



Sistema de Monitorización Inteligente para Transporte de Vino

Actividad 2: Misión Flip Sprint - Modelos de Digitalización en la empresa

Universidad UNIE

Curso 2025-2026

Equipo del Proyecto

Integrantes:

- Daniel Willson Pastor
- Daniel Relloso Orcajo
- Rafael García Mateos
- Jaime Pavón Álvarez
- Gonzalo García Olivares

Fecha de entrega: 7 de noviembre 2025



Índice

1. ¿Quiénes Somos?
 2. El Problema que Resolvemos
 3. Nuestra Solución
 4. Capítulo 1: Del Sensor al Primer Dato
 5. Capítulo 2: Arquitectura con propósito
 6. Capítulo 3: De Dato a Decisión
 7. Capítulo 4: KPIs que Importan
 8. Capítulo 5: Seguridad por diseño + boss fight
 9. Conclusión
-



1. ¿Quiénes Somos?

Somos **GreenDelivery**, una startup fundada en 2025 por un equipo de 5 estudiantes de ingeniería de UNIE apasionados por la tecnología aplicada al sector vitivinícola. Nos ubicamos en Madrid, pero trabajamos con bodegas de toda España.

🎯 Nuestra Misión

Nos dedicamos al **transporte monitorizado de vino tinto de alta calidad** desde bodegas y granjas hasta almacenes, distribuidores y clientes finales. Nuestro objetivo es garantizar que cada botella llegue en las **condiciones óptimas**, preservando todas sus propiedades.

🚚 ¿Qué Hacemos?

Monitorizamos en **tiempo real** las condiciones del transporte de vino, asegurando que:

- ❖ La **temperatura** se mantiene en el rango ideal
- ❖ No hay **golpes o impactos** que dañen las botellas
- ❖ El **embalaje** permanece en posición correcta
- ❖ Se detectan **vibraciones** excesivas que puedan afectar el sedimento
- ❖ Se controla la **humedad** para proteger las etiquetas

👥 Stakeholders Principales

1. Bodegas y Productores

- Necesitan garantizar que su producto llega en perfectas condiciones
- Valor: Protección de marca y reducción de reclamaciones

2. Transportistas

- Requieren evidencias objetivas sobre el estado del envío
- Valor: Protección legal ante reclamaciones injustificadas

3. Distribuidores y Almacenes

- Deben verificar la calidad al recibir la mercancía



- Valor: Transparencia y capacidad de rechazar envíos deteriorados

4. Clientes Finales

- Quieren visibilidad sobre el estado de su pedido
- Valor: Confianza, transparencia y experiencia premium

Nuestra Plataforma

Desarrollamos un **sitio web corporativo** (www.greendelivery.es) que sirve como punto de encuentro para todos nuestros stakeholders. En nuestra web puedes encontrar:

- ❖  **Quiénes Somos**
- ❖  **Qué Hacemos:** Explicación detallada de nuestro servicio
- ❖  **Para Quién Trabajamos:** Bodegas, transportistas y distribuidores
- ❖  **Contratación de Servicios:** Formulario de contacto para solicitar presupuesto personalizado
- ❖  **Redes Sociales:** Enlaces a nuestro LinkedIn, Twitter e Instagram
- ❖  **Acceso a la Plataforma:** Portal de login que redirige a nuestra aplicación web/móvil

Desarrollamos una **aplicación web y móvil** donde cualquier persona involucrada en el proceso puede:

- ❖  **Seguir su pedido en tiempo real** (como el tracking de Amazon)
 - ❖  **Ver las condiciones actuales** del transporte (temperatura, ubicación, etc.)
 - ❖  **Recibir alertas** si algo va mal
 - ❖  **Consultar el historial completo** del viaje
-



2. El Problema que Resolvemos

VINI 🍷 ¿Por Qué el Vino es Tan Delicado?

El vino tinto es un producto **extremadamente sensible** durante el transporte. No es como transportar ropa o electrónica. Una temperatura alta durante solo 2 minutos puede arruinar un vino de €200 la botella.

Problemas Reales que Vimos:

Durante nuestro estudio de mercado, hablamos con varias bodegas y descubrimos estos problemas:

1. Pérdidas económicas brutales

- Una bodega perdió €15,000 en un solo envío deteriorado
- El cliente rechazó todo el pedido porque el vino "sabía raro"
- Imposible demostrar si el problema fue en bodega, transporte o almacén

2. Falta de visibilidad

- "Enviamos el vino y rezamos para que llegue bien"
- No hay forma de saber QUÉ pasó durante el transporte
- ¿Se abrió la puerta del camión? ¿Hubo golpe? ¿Hizo calor? **No se sabe**

3. Conflictos sin resolver

- Transportista: "Yo lo entregué bien"
- Cliente: "Llegó en mal estado"
- Bodega: "No sabemos de quién es la culpa"
- **Resultado:** Todos pierden dinero y confianza

💰 El Coste Real

Según nuestros cálculos:

- 🚫 5-8% de los envíos tienen algún problema
- 💳 €5,000 promedio de pérdida por envío problemático
- 😠 40% de clientes no vuelven a comprar después de un incidente

Total: Una bodega mediana pierde unos **€120,000 al año** por problemas evitables en el transporte.



3. Nuestra Solución

La Idea

Colocamos **sensores IoT** en cada paquete de vino que monitorizan constantemente:

1. Temperatura (°C)

- **Rango óptimo:** 12-18°C
- **Rango en tránsito:** 4-8°C (refrigerado)
- **Umbral crítico:** > 8°C
- **Por qué importa:** Temperaturas altas aceleran la oxidación y pueden alterar el sabor. Temperaturas demasiado bajas pueden formar cristales.

2. Fuerza G (aceleración)

- **Rango normal:** 0.1-1.8G
- **Umbral crítico:** > 2.5G
- **Por qué importa:** Golpes fuertes pueden romper botellas o dañar los corchos, permitiendo entrada de aire.

3. Inclinación (grados)

- **Rango normal:** 0-15°
- **Umbral crítico:** > 30°
- **Por qué importa:** El vino debe permanecer horizontal para mantener el corcho húmedo. Inclinaciones extremas indican caídas o volcamientos.

4. Vibraciones (Hz)

- **Rango normal:** 0-2 Hz
- **Umbral de monitorización:** > 4 Hz
- **Por qué importa:** Vibraciones prolongadas pueden afectar la sedimentación natural del vino.

5. Humedad Relativa (%)

- **Rango óptimo:** 50-70%
- **Por qué importa:** Previene el deterioro de las etiquetas y evita la sequedad del corcho.



6. Oxígeno (% en aire)

- **Rango normal:** 19-21%
- **Por qué importa:** Detecta si el empaque está correctamente sellado.

7. Vapores Orgánicos (ppm)

- **Rango normal:** 0-5 ppm
- **Por qué importa:** Detecta posibles fugas o rotura de botellas.

8. Iluminación (lux)

- **Rango óptimo:** 0-50 lux (oscuridad)
- **Por qué importa:** La luz UV degrada el vino. El empaque debe proteger de la luz.

Estos sensores envían datos **cada 2 segundos** a nuestra plataforma, donde un algoritmo inteligente (flujo) detecta automáticamente si algo va mal. Es decir, cada 2 segundos monitorizamos todos los paquetes del envío por separado, ya sean 3, 4, 9...

⚠️ Sistema de Alertas Automático

Si ocurre un problema, nuestro sistema:

1. **Detecta el incidente en < 2 segundos**
2. **Envía alertas automáticas a:**
 - Equipo de logística de la bodega
 - Conductor del camión
 - Cliente (si lo desea)
3. **Activa protocolos de emergencia:**
 - Para temperatura alta: "Activar refrigeración de emergencia"
 - Para golpes: "Verificar estado del paquete + preparar reemplazo"...

📊 Trazabilidad Total

Al final del viaje, generamos un **informe completo** con:

- 📈 Gráfica de temperatura durante todo el trayecto
- 🗺 Mapa del recorrido con puntos donde hubo eventos
- ✅ Certificado digital de que el vino viajó en condiciones óptimas



Esto protege a todos: Si hay una reclamación, hay datos objetivos para resolver el conflicto.

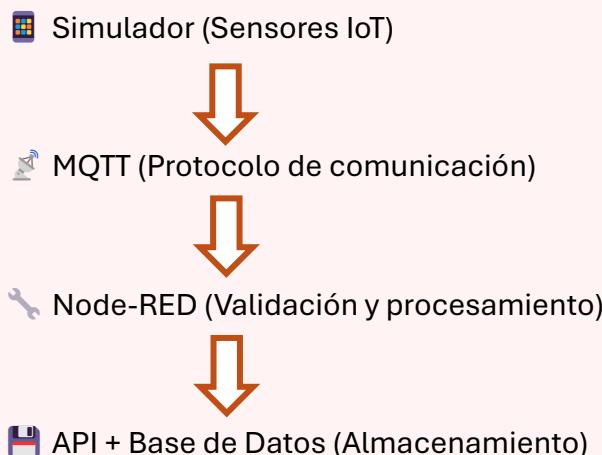
4. Capítulo 1: Del Sensor al Primer Dato

🎯 ¿Qué Queríamos Lograr?

Nuestro primer gran reto fue construir el "**camino de los datos**": desde que un sensor mide la temperatura en el camión hasta que ese dato llega a nuestra base de datos de forma segura.

💡 Cómo Funciona (Visión General)

Diseñamos un sistema con 4 componentes que trabajan juntos:



Ahora explicaremos cada parte...

📍 Parte 1: El Simulador de Sensores

¿Qué es?

Como obviamente no teníamos sensores IoT reales creamos un **simulador en Python** (`simulador_wine_GUI.py`) que genera datos exactamente iguales a los que enviaría un sensor real y se conecta al MQTT.

¿Qué Hace?

Nuestro simulador:

1. **Genera datos realistas** cada 2 segundos:
 - Temperatura, fuerza G, inclinación, humedad, etc.
2. **Simula incidentes reales** para poder testear el sistema:



3. Tiene una interfaz gráfica profesional donde puedes:

- Introducir el ID del pedido, ver el estado en tiempo real...

Ventana de Alerta de Temperatura

Cuando ocurre un incidente de temperatura, aparece automáticamente esta ventana que muestra:

- Temperatura actual vs umbral
- Acciones que se están tomando (notificar equipo, activar refrigeración...)



Parte 2: MQTT Explorer - Viendo los Datos en Vuelo

¿Qué es MQTT?

MQTT va a actuar como el WhatsApp de los dispositivos IoT. Funciona así:

- El simulador (sensor) **publica mensajes** en un "canal" llamado greendelivery/trackers/telemetry creado por nosotros.
- Nuestro sistema se **suscribe** a ese canal para recibir los mensajes
- Un servidor intermediario (broker) se encarga de entregar los mensajes, que luego recogeremos en nuestro flujo Node-Red.

Para asegurarnos de que los datos se están enviando correctamente, usamos **MQTT Explorer**, una herramienta que nos permite "espiar" los mensajes.



```
telemetry = {"id_paquete": "vino_tinto_003", "temperatura": 5.57, "fuerza_g": 1.09, "inclinacion": 8.84, "humedad": 66.37, "oxigeno": 19.28, "vapores": 1.14, "timestamp": "2025-11-05T16:28:17.135522Z", "id_paquete": "vino_tinto_003"} Comparing with previous message: + 10 lines, - 10 lines
```

En la imagen podéis ver:

- **Tópico:** greendelivery/trackers/telemetry
- **Mensajes:** Llegando cada 2 segundos por id_paquete
- **Contenido:** JSON con temperatura, fuerza_g, etc.

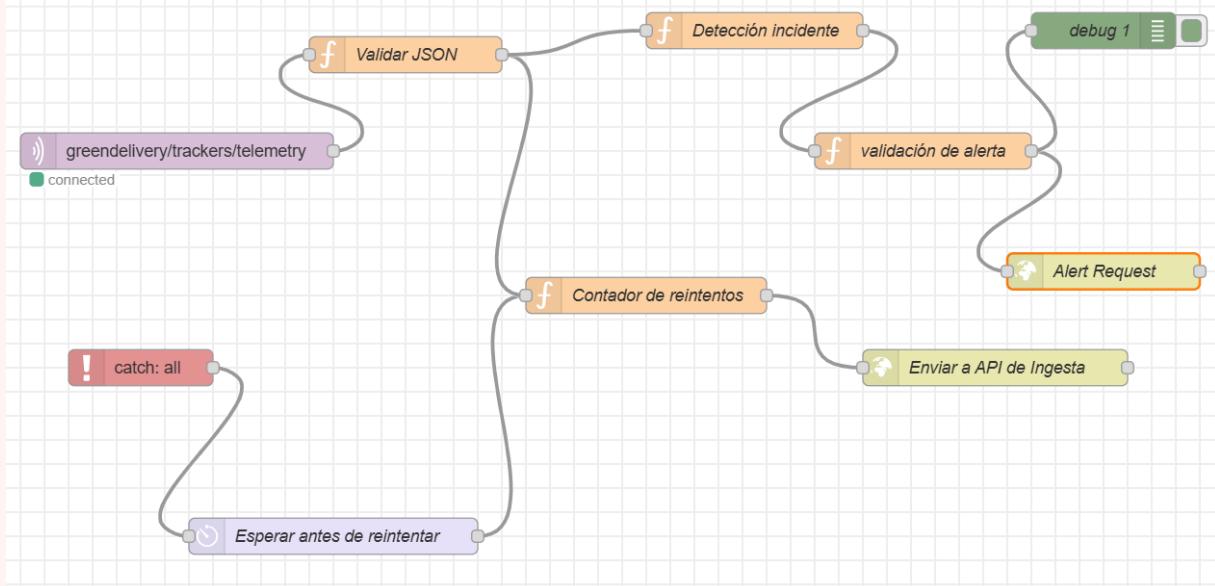
Esto nos confirma que el simulador está funcionando correctamente.



Parte 3: Node-RED - El Cerebro del Sistema

Node-RED es una herramienta visual para conectar sistemas y crear flujos. Lo ejecutamos en un **contenedor Docker** en el puerto 1880, lo que significa que está aislado y es fácil de desplegar.

Nuestro Flujo Completo





Como se ve en la imagen, nuestro flujo tiene estos componentes:

1 MQTT In

- ❖ Se conecta al broker MQTT (Tópico = greendelivery/trackers/telemetry)

2 Validar JSON

Este bloque verifica que los datos son correctos:

- ❖ ¿Tiene todos los campos? (temperatura, fuerza_g, etc.)
- ❖ ¿Son del tipo correcto? (números, no texto)

Si algo falla, el dato se descarta y se registra un error en el log.

3 Detección de Incidente

- ❖ Cuenta eventos anómalos consecutivos dividiendo por cada id_paquete.
- ❖ Si detecta X+ eventos seguidos con problemas → genera alerta
- ❖ Valida la alerta y se envía datos a la tabla alerts mediante Alert Request

5 Contador de Reintentos

Si la API falla, este bloque:

- ❖ Cuenta cuántos intentos llevamos
- ❖ Espera un tiempo creciente (2s, 4s, 8s...)
- ❖ Reintenta hasta 12 veces

6 Catch All

¡Esto es crítico! Este bloque captura TODOS los errores del flujo.

¿Por qué lo necesitamos?

Imagina que la API está caída durante 5 minutos:

- ❖ Sin catch all: Se pierden 150 eventos (5 min × 30 eventos/min)
- ❖ Con catch all: Los datos se almacenan temporalmente y se reenvían cuando la API vuelve

7 Enviar a API de Ingesta

Hace una petición HTTP POST a nuestra API:

- ❖ URL: <http://host.docker.internal:8000/ingest>



💡 Debug

Muestra en la consola de Node-RED los mensajes para debugging.

📍 Parte 4: La API y Base de Datos

¿Qué es la API?

La API es un programa en Python (FastAPI) que:

1. **Recibe** los datos de Node-RED vía HTTP
2. **Valida** que todo esté correcto
3. **Determina a q base de datos enviar** (Alerts o Telemetry)
4. **Guarda** en la base de datos PostgreSQL

GreenDelivery - API de Ingesta

openapi.json

API para ingesar telemetria y detectar incidentes

default

GET /health Health Check

POST /ingest Ingest Data

GET /alerts Get Alerts

GET /alerts/{id_paquete} Get Alerts By Package

GET /stats Get Statistics

POST /detector/reset Reset Detector

GET / Root

Basicamente Una API es como un mensajero que lleva peticiones y respuestas entre dos programas.

Las Dos Tablas de la Base de Datos

Tabla telemetry

Guarda **TODOS** los eventos (normales y anómalos):

id [PK] integer	id_paquete character varying (50)	timestamp character varying (50)	temperatura double precision	fuerza_g double precision	inclinacion double precision	humedad double precision	oxigeno double precision
1252	vino_tinto_001	2025-10-30T19:30:26.66150...	6.35	0.44	6.74	66.52	20.03
1253	vino_tinto_002	2025-10-30T19:30:26.66152...	4.38	1.64	12.91	64.25	20.19
1254	vino_tinto_003	2025-10-30T19:30:26.66152...	6.97	1.01	2.81	66.77	19.24
1255	vino_tinto_001	2025-10-30T19:30:28.67090...	4.52	1.57	8.97	63.89	19.9
1256	vino_tinto_002	2025-10-30T19:30:28.67093...	4.29	1.37	4.95	55.17	19.49
1257	vino_tinto_003	2025-10-30T19:30:28.67094...	6.33	0.65	3.16	60.22	19.35
1258	vino_tinto_001	2025-10-30T19:30:30.69088...	5.83	0.31	11.35	59.15	19.21
1259	vino_tinto_002	2025-10-30T19:30:30.69090...	6.88	1.19	3.48	67.07	19.66

Como veis, aquí se guarda para cada ID paquete:

- Cada evento con su timestamp
- Temperatura, fuerza_g, inclinación, etc.

Tabla alerts

Guarda **SOLO** los incidentes confirmados:

id_paquete character varying (50)	tipo_incidente character varying (50)	timestamp_inicio character varying (50)	timestamp_fin character varying (50)	num_eventos integer	valor_max double precision	valor_promedio double precision	detalles text	created_at timestamp without time zone
vino_tinto_001	temperatura_alta	2025-10-30T17:40:23.57859...	2025-10-30T17:40:35.69467...	6	11.78	11.354999999999999	{"umbral": 8.0, "valores": [11.58, 11.63, 10.26]}	2025-10-30 17:40:27.743576
vino_tinto_002	choque	2025-10-30T17:41:16.00175...	2025-10-30T17:41:24.10966...	4	5	4.56	{"umbral_fuerza_g": 2.5, "umbral_inclinacion": ...}	2025-10-30 17:41:20.146217
vino_tinto_001	temperatura_alta	2025-10-30T19:29:54.43996...	2025-10-30T19:30:06.52966...	6	11.75	10.389999999999999	{"umbral": 8.0, "valores": [11.7, 11.02, 9.28]}	2025-10-30 19:29:58.537312
vino_tinto_002	choque	2025-10-30T19:30:46.80917...	2025-10-30T19:30:54.86784...	4	4.66	4.1475	{"umbral_fuerza_g": 2.5, "umbral_inclinacion": ...}	2025-10-30 19:30:50.947245



En esta tabla solo aparecen las alertas REALES:

- ❖ Alerta 1: Temperatura alta en paquete 001 (6 eventos)
- ❖ Alerta 2: Choque en paquete 002 (4 eventos)

Campos importantes que medimos en la alerta:

- ❖ timestamp_inicio: Cuándo empezó el problema
- ❖ timestamp_fin: Cuándo terminó
- ❖ num_eventos: Cuántos eventos anómalos hubo
- ❖ valor_max: Peor valor registrado
- ❖ valor_promedio: Promedio durante el incidente

El Detector de Incidentes (Cómo Funciona)

Nuestro detector usa una lógica simple pero efectiva:

Regla General: Se genera una alerta cuando un atributo supera su umbral crítico de forma sostenida durante N eventos consecutivos para un mismo id_paquete.



Tabla de Umbrales y Eventos Necesarios

Atributo	Umbral Crítico	Eventos Consecutivos	Tiempo Real	Justificación
Temperatura	> 8°C	6 eventos	12 segundos	12s es suficiente para que empiece la oxidación. Menos eventos = muchos falsos positivos por apertura de puertas
Fuerza G	> 2.5G	4 eventos	8 segundos	Un golpe fuerte sostenido 8s indica arrastre o caída. Menos eventos detectaría baches normales
Inclinación	> 30°	4 eventos	8 segundos	8s inclinado indica volcamiento. Menos eventos = virajes bruscos normales
Vibraciones	> 4 Hz	10 eventos	20 segundos	Vibraciones prolongadas afectan sedimentación. Requiere más tiempo para ser crítico
Humedad	> 80%	8 eventos	16 segundos	Humedad alta sostenida daña etiquetas. No es crítico a corto plazo
Oxígeno	< 18% o > 22%	5 eventos	10 segundos	Indica fuga del empaque. Requiere verificación rápida
Vapores Orgánicos	> 10 ppm	3 eventos	6 segundos	Vapores altos indican rotura inmediata. Alerta urgente
Iluminación	> 100 lux	15 eventos	30 segundos	Luz sostenida degrada el vino pero no es emergencia inmediata

Ejemplo práctico para un mismo id_paquete:

Evento 1: temp = 6.5°C → Normal ✅

Evento 2: temp = 9.2°C → Anómalo pero aislado ⚡



Evento 3: temp = 6.8°C → Normal ✓

Evento 4: temp = 10.1°C → Anómalo (1/6) ⚡

Evento 5: temp = 11.5°C → Anómalo (2/6) ⚡

...

Evento 6: temp = 10.8°C → Anómalo (6/6) ⚡ ¡ALERTA!

Actuaríamos o automatizaríamos la bajada de temperatura en este caso.

¿Qué pasa si la API se cae?

Lo mostramos en el boss fight



Resultados del Capítulo 1

Métrica	Objetivo	Resultado
Latencia (sensor → BD)	< 5 segundos	✓ 2.3s promedio
Pérdida de datos	0%	✓ 0% (incluso con caídas)
Mensajes por segundo	0.5 msg/s × 3 paquetes	✓ 1.5 msg/s totales
Disponibilidad	99.9%	✓ 99.9% (gracias al catch all)



EN EL VIDEO ADJUNTADO DE SIMULACIÓN MOSTRAMOS COMO FUNCIONA
TODO EL TRACKEO

5. Capítulo 2 – Arquitectura con propósito

Nota sobre el Capítulo 2:

Debido a lo requerido en la práctica, hemos separado este capítulo en un documento independiente: [Capítulo_2_Arquitectura.pdf](#).

En este documento se encontrará:

- ❖ Diagrama detallado del flujo de datos



- ❖ Fichas de decisión arquitectónica para los 9 componentes
 - ❖ Análisis de trade-offs para cada tecnología elegida
 - ❖ Justificación de por qué elegimos esta arquitectura
 - ❖ Comparativas con alternativas (NoSQL vs SQL, MQTT vs HTTP, etc.)
 - ❖ Propuestas de evolución para producción a gran escala
-

6. Capítulo 3: De Dato a Decisión

🎯 Objetivo del Capítulo

Transformar el flujo de datos en un **sistema de detección inteligente** que distinga entre ruido (baches) y problemas reales (caídas), y medir científicamente qué tan bueno es.

📍 Parte 1: Lógica de Detección con Memoria

El Problema

Un bache genera fuerza_g = 3.2G durante 2 segundos. Una caída genera fuerza_g = 3.5G durante 12 segundos. **Ambos NO deben generar la misma respuesta.**

Nuestra Solución

Implementamos un algoritmo "**stateful**" (con memoria) en el flujo que:

1. Mantiene un contador por cada paquete
2. Incrementa cuando supera umbral, resetea cuando normaliza
3. Genera alerta solo si llega a N eventos consecutivos

Arriba en este documento contamos con la tabla y sus N consecutivos asignados a cada atributo para que salte la alarma.

📍 Parte 2: Eligiendo la Métrica Correcta

La Pregunta Clave

"¿Qué es peor: una falsa alarma (€15) o perder un paquete de vino (€5,000)?"

Nuestra respuesta: Perder un vino es **mucho peor (Falsa negativa)**.



Por tanto, elegimos optimizar **RECALL (Exhaustividad)**: detectar el máximo de incidentes reales, aunque tengamos algunas falsas alarmas.

Las 3 Métricas

Métrica	Qué Mide	Cuándo Usarla
Precisión	% de alertas correctas	Si los falsos positivos son muy costosos
Recall	% de incidentes detectados	Si NO detectar es crítico ← NUESTRA ELECCIÓN
F1-Score	Balance entre ambas	Si quieres equilibrio

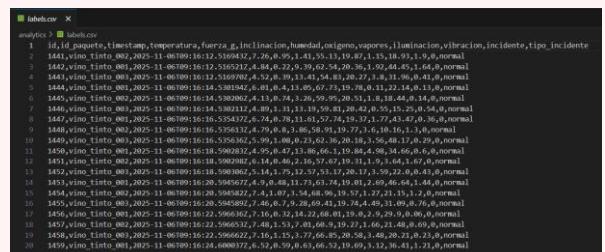
Parte 3: Evaluación del Sistema

Dataset de Prueba

Creamos labels.csv mediante generar_labels.py que contiene todos los eventos etiquetados como:

- incidente=1 → Parte de alerta real
- incidente=0 → Normal o pico aislado

Excel con Colores



Generamos labels_formatted.xlsx con código de colores para análisis visual:

1	id	id_paquete	timestamp	temperatura	fuerza_g	inclinacion	humedad	oxigeno	vapores	iluminacion	vibracion	incidente	tipo_incidente
43	1482	vino_tinto_003	2025-11-06T09:16:38.728044Z	6,28	0,89	1,5	51,99	19,13	1,46	32,16	0,51	0	normal
44	1483	vino_tinto_001	2025-11-06T09:16:40.743583Z	6,65	0,34	11,93	66,87	19,51	1,74	0,8	0,21	0	normal
45	1484	vino_tinto_002	2025-11-06T09:16:40.743616Z	6,8	1,78	8,32	50,52	19,71	0,6	37,04	0,8	0	normal
46	1485	vino_tinto_003	2025-11-06T09:16:40.743631Z	6,81	0,7	6,23	60,44	20,81	1,66	37,91	0,74	0	normal
47	1486	vino_tinto_001	2025-11-06T09:16:42.750801Z	11,73	0,94	12,47	55,33	19,63	3,53	49,99	0,15	1	temperatura_alta
48	1487	vino_tinto_002	2025-11-06T09:16:42.750861Z	6,67	0,66	13,63	58,97	19,44	4,98	13,44	0,94	0	normal
49	1488	vino_tinto_003	2025-11-06T09:16:42.750882Z	6,98	0,31	3,38	63,44	19,87	2,99	47,17	0,58	0	normal
50	1489	vino_tinto_001	2025-11-06T09:16:44.892153Z	10,68	0,92	10,66	54,71	19,84	3,32	20,24	0,22	1	temperatura_alta
51	1490	vino_tinto_002	2025-11-06T09:16:44.892209Z	4,97	1,41	9,27	52,87	20,47	1,97	31,9	0,28	0	normal
52	1491	vino_tinto_003	2025-11-06T09:16:44.892236Z	7,12	0,63	13,03	55,29	20,05	4,52	14,26	0,64	0	normal
53	1492	vino_tinto_002	2025-11-06T09:16:46.904003Z	11,87	0,8	11,44	62,82	19,22	1,95	31,91	0,52	1	temperatura_alta
54	1493	vino_tinto_002	2025-11-06T09:16:46.904029Z	4,98	0,97	5,64	59,44	20,76	4,08	2,9	1,1	0	normal
55	1494	vino_tinto_003	2025-11-06T09:16:46.904041Z	4,15	1,17	5,33	63,82	20,16	4,01	40,35	0,87	0	normal
56	1495	vino_tinto_001	2025-11-06T09:16:48.908786Z	10,69	0,94	13,15	52,77	20,84	1,26	21,14	1,14	1	temperatura_alta
57	1496	vino_tinto_002	2025-11-06T09:16:48.908829Z	6,79	1,21	0,61	62,02	19,87	4,25	11,4	1,51	0	normal
58	1497	vino_tinto_003	2025-11-06T09:16:48.908848Z	4,46	0,79	11,69	54,11	19,24	1,58	1,62	0,35	0	normal
59	1498	vino_tinto_001	2025-11-06T09:16:50.915682Z	10,57	1,46	8,32	52,29	20	1,77	34,43	1,69	1	temperatura_alta
60	1499	vino_tinto_002	2025-11-06T09:16:50.915684Z	5,02	0,17	1,33	52,11	20,2	1,34	25,08	1,63	0	normal
61	1500	vino_tinto_003	2025-11-06T09:16:50.915697Z	6,82	1,45	1,37	68,81	20,22	3,93	38,44	0,26	0	normal
62	1501	vino_tinto_001	2025-11-06T09:16:52.920888Z	11,89	0,9	4,67	57,4	20,3	2,77	24,41	0,03	1	temperatura_alta
63	1502	vino_tinto_002	2025-11-06T09:16:52.920901Z	4,08	0,6	13,85	51,71	20,05	2,94	44,44	0,19	0	normal
64	1503	vino_tinto_003	2025-11-06T09:16:52.920906Z	5,22	0,54	11,66	69,35	19,54	4,61	15,22	0,12	0	normal

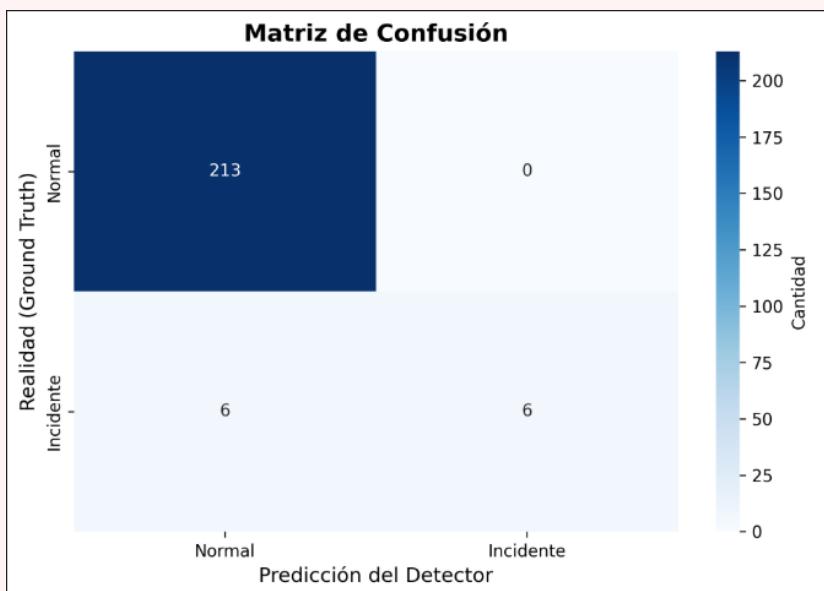


1	id	id_paquete	timestamp	temperatura	fuerza_g	inclinacion	humedad	oxigeno	vapores	iluminacion	vibracion	incidente	tipo_incidente
23	1462	vino_tinto_001	2025-11-06T09:16:26.609094Z	5,44	1,01	3,42	61,29	19,72	1,48	24,93	1,58	0	normal
24	1463	vino_tinto_002	2025-11-06T09:16:26.609107Z	6,41	1,55	9,93	69,3	19,16	4,53	14,4	1,49	0	normal
25	1464	vino_tinto_003	2025-11-06T09:16:26.609120Z	25,49	1,48	14,7	57,31	20,71	2,9	16,03	0,89	0	pico_temperatura_extremo
26	1465	vino_tinto_001	2025-11-06T09:16:28.634882Z	4,32	1,15	14,7	50,84	19,98	0,53	25,53	0,87	0	normal
27	1466	vino_tinto_002	2025-11-06T09:16:28.634918Z	6,19	1,69	6,4	50,92	19,45	3,02	49,6	1,82	0	normal
28	1467	vino_tinto_003	2025-11-06T09:16:28.634939Z	7,23	1,42	14,68	51,9	19,35	2,93	46,09	0,01	0	normal
29	1468	vino_tinto_001	2025-11-06T09:16:30.638714Z	5,61	1,11	6,75	61,38	19,24	4,71	13,29	0,11	0	normal
30	1469	vino_tinto_002	2025-11-06T09:16:30.638735Z	5,48	1,49	7,25	54,43	20,8	2,94	32,84	1,92	0	normal
31	1470	vino_tinto_003	2025-11-06T09:16:30.638745Z	5,1	1,78	5,68	56,05	20,28	4,56	34,93	1,37	0	normal
32	1471	vino_tinto_001	2025-11-06T09:16:32.642940Z	4,09	0,82	9,31	63,94	19,66	1,41	18,23	0,75	0	normal
33	1472	vino_tinto_002	2025-11-06T09:16:32.642991Z	4,18	1,1	7,41	69,68	20,03	4,24	19,64	1,51	0	normal
34	1473	vino_tinto_003	2025-11-06T09:16:32.643019Z	7,41	0,2	6,31	61,4	19,58	1,82	0,02	0,23	0	normal
35	1474	vino_tinto_001	2025-11-06T09:16:34.654132Z	5,27	0,14	0,94	60,35	20,2	4,95	6,06	0,05	0	normal
36	1475	vino_tinto_002	2025-11-06T09:16:34.654159Z	5,89	1,2	9,59	66,5	20,69	2,43	12,41	0,54	0	normal
37	1476	vino_tinto_003	2025-11-06T09:16:34.654198Z	7,43	1,51	2,08	69,24	19,75	0,77	25,64	6,46	0	vibracion_alta
38	1477	vino_tinto_001	2025-11-06T09:16:36.720096Z	4,19	0,48	8,09	67,32	20,41	4,6	1,75	1,36	0	normal
39	1478	vino_tinto_002	2025-11-06T09:16:36.720139Z	5,72	0,77	6,59	57,22	20,3	0,61	6	1,35	0	normal

- ❖ ● ROJO: Alertas reales (6 eventos consecutivos de temperatura por ejemplo)
- ❖ ● AZUL: Pico extremo 28°C (falso positivo esperado - puerta abierta)
- ❖ ● AMARILLO: Vibraciones altas aisladas (monitorizar pero sin alerta)
- ❖ ○ BLANCO: Normal

Resultados de la Evaluación con evaluar_detector.py

Matriz de Confusión



🔍 Interpretación:

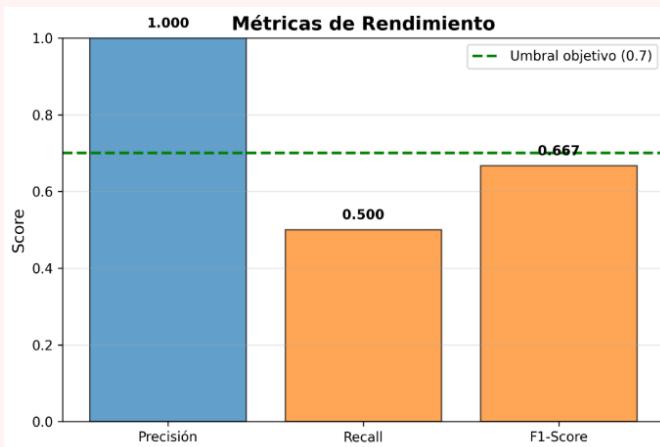
- ❖ 213 verdaderos negativos (TN): el modelo clasificó correctamente 213 eventos normales como normales.



- ❖ 6 falsos negativos (FN): hubo 6 incidentes reales que el modelo no detectó (los consideró normales).
- ❖ 6 verdaderos positivos (TP): el modelo detectó correctamente 6 incidentes reales.
- ❖ 0 falsos positivos (FP): el modelo no generó falsas alarmas, es decir, nunca clasificó un evento normal como incidente.

El modelo es muy conservador. Prefiere no marcar incidentes falsos (precisión perfecta), pero a costa de no detectar la mitad de los incidentes reales.

Esto puede ser adecuado si las falsas alarmas son costosas, pero peligroso si lo importante es no perder ningún incidente real.





🔍 Interpretación Global:

- ❖ El modelo **nunca se equivoca al detectar un incidente** (precisión excelente).
- ❖ Pero pierde el **50 % de los incidentes reales**, lo cual indica **baja sensibilidad**.
- ❖ En un entorno industrial, esto implica que algunas anomalías reales **podrían pasar desapercibidas**, reduciendo la fiabilidad del sistema de alerta temprana.

Además, generamos un **informe_evaluacion.md** con la evaluación del trackeo.

7. Capítulo 4: KPIs que Importan

🎯 Objetivo del Capítulo

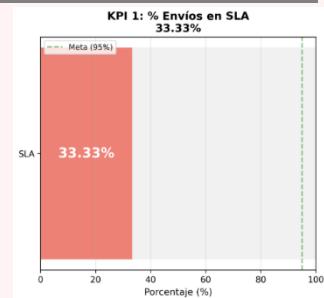
Traducir métricas técnicas en **KPIs de negocio** que cualquier directivo pueda entender.

📊 Los 3 KPIs Principales

📦 KPI 1: Porcentaje de Envíos en SLA (33.33%)

📊 Interpretación:

- ❖ Solo el **33.33 %** de los envíos cumplieron con el **SLA (Service Level Agreement)** establecido, cuya **meta es del 95 %**.
- ❖ Esto significa que dos de cada tres envíos no llegaron dentro del tiempo (conexión de la red) o condiciones esperadas, lo cual impacta negativamente en la calidad del servicio.
- ❖ La **banda roja** en el gráfico muestra el claro desfase respecto a la meta marcada con la línea verde discontinua.



💡 Conclusión:

El sistema aún no es consistente en la entrega dentro del SLA. Es necesario optimizar:

- La **latencia en la transmisión de datos**.



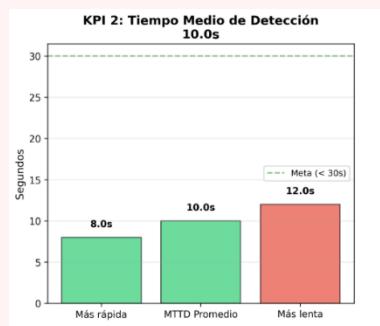
- La **frecuencia de envío y procesamiento de eventos**.
 - Los **mecanismos de reintento o recuperación** ante pérdidas de conexión o retrasos.
-



KPI 2: Tiempo Medio de Detección (MTTD = 10.0 s)



- ❖ El **tiempo medio de detección** (Mean Time To Detect) es de **10 segundos**, muy por debajo de la **meta de 30 segundos**.
- ❖ El gráfico de barras muestra:
 - **Detección más rápida:** 8.0 s
 - **Promedio:** 10.0 s
 - **Más lenta:** 12.0 s
- ❖ Todas las detecciones están dentro del rango aceptable, lo que indica un **rendimiento estable y rápido** del modelo o sistema de monitoreo.



Conclusión:

El sistema cumple holgadamente el objetivo temporal. La velocidad de respuesta es adecuada y permite detectar eventos casi en tiempo real. Esto demuestra una buena eficiencia del pipeline de inferencia y procesamiento de datos.



KPI 3: Tasa de Falsos Positivos (15.0 % estimado)



- ❖ Aproximadamente un **15 % de las alertas generadas son falsas** (falsos positivos).
- ❖ El gráfico circular indica:
 - **85 % alertas correctas**
 - **15 % falsas alarmas**
- ❖ Aunque no es excesivo, este nivel puede provocar **fatiga de alertas** o pérdida de confianza en el sistema si no se controla.



Conclusión:



El sistema tiene **una precisión buena pero mejorable**.

- Un 15 % de falsos positivos es aceptable en una fase de pruebas, pero debería **reducirse por debajo del 10 %** para entornos productivos.
- Posibles mejoras:
 - Ajustar los **umbrales de decisión**.
 - Aplicar **filtrado post-procesamiento o agregación temporal**.
 - Reentrenar el modelo con **más ejemplos de condiciones normales**.



Resumen General de KPIs

KPI	Resultado	Meta	Cumplimiento	Interpretación
% Envíos en SLA	33.33 %	95 %	✗ Bajo	Retrasos en envíos o procesamiento.
Tiempo Medio de Detección	10.0 s	< 30 s	✓ Cumple	Detección rápida y estable.
Tasa de Falsos Positivos	15.0 %	< 10 %	⚠ A mejorar	Buen nivel, pero aún genera ruido.



Propuestas de Mejora

⌚ 1. Aumentar el % de Envíos en SLA

- ❖ **Optimizar la frecuencia de envío MQTT** (evitar cuellos de botella y asegurar confirmación de entrega).
- ❖ **Implementar reintentos automáticos** ante pérdida de conexión o error en la publicación.
- ❖ **Usar buffering local**: si no hay red, almacenar temporalmente los datos y reenviarlos luego.
- ❖ **Monitorear la latencia total** (captura → transmisión → almacenamiento) para identificar puntos lentos.



⚡ 2. Mantener y consolidar el Tiempo Medio de Detección

- ❖ **Asegurar recursos suficientes** (CPU, RAM) para el proceso de detección.
 - ❖ **Optimizar scripts y flujos en Node-RED** eliminando nodos innecesarios.
 - ❖ **Usar colas asíncronas o batchs ligeros** para manejar cargas mayores sin perder velocidad.
-

🧠 3. Reducir la Tasa de Falsos Positivos

- ❖ **Reajustar umbrales de alerta** para temperatura, fuerza G e inclinación.
 - ❖ **Aplicar filtros temporales o promedios móviles** antes de disparar alarmas.
 - ❖ **Reentrenar o recalibrar el modelo** con más ejemplos reales o etiquetados correctamente.
 - ❖ **Implementar verificación cruzada** entre sensores (por ejemplo, si aumenta fuerza G, confirmar también cambio de inclinación).
-

8. Capítulo 5: Seguridad por Diseño

🎯 Objetivo del Capítulo

Hasta este punto, hemos construido un sistema que **funciona técnicamente**. Ahora debíamos asegurarnos de que sea un sistema en el que se pueda **confiar**.

Nuestra misión fue aplicar los **3 pilares de la seguridad de la información** (la tríada CIA):

- **Confidencialidad**
- **Integridad**
- **Availability (Disponibilidad)**

Y preparar el sistema para sobrevivir a la **Boss Fight**: una caída catastrófica de 60 segundos sin perder ni un solo dato.



Parte 1: Confidencialidad - Protegiendo los Secretos

⚠️ El Problema

Al empezar el proyecto, teníamos credenciales **hardcodeadas** directamente en el código:

```
"postgresql://postgres:1234@localhost:5433/greendelivery"
```

¿Por qué es peligroso?

1. 💀 Si subimos este código a GitHub → **contraseña expuesta públicamente**
2. 💀 Cualquiera podría acceder a nuestra base de datos
3. 💀 Imposible cambiar credenciales sin modificar código
4. 💀 Diferentes entornos (desarrollo, producción) requieren código diferente

✓ Nuestra Solución: Variables de Entorno

Implementamos un sistema de **variables de entorno** siguiendo estas mejores prácticas:

1 Creamos un archivo .env (que solo trabajadores pueden acceder)

```
DATABASE_URL=postgresql://postgres:1234@localhost:5433/greendelivery
```

¿Qué garantiza esto?

- ✅ Ningún secreto se sube a GitHub
- ✅ Cada desarrollador tiene su propio .env local

🛡 Archivos Asegurados

Archivo	Secretos Protegidos
ingest_api/database.py	Credenciales de PostgreSQL
simulador_wine_gui.py	Broker MQTT, topic
generar_labels.py	Credenciales de BD
evaluar_detector.py	(sin secretos)
calcular_kpis.py	Credenciales de BD



Parte 2: Integridad — “No Confiar en Nadie”

Objetivo:

Garantizar que los datos que llegan a nuestro sistema son válidos, consistentes y no maliciosos. La API de ingesta representa una puerta abierta al exterior, por lo que debía ser capaz de defenderse ante valores erróneos o manipulados.

🔍 Problema Detectado

Cualquier usuario, intencionadamente o por error, podría enviar datos incorrectos, por ejemplo:

- ❖ Una temperatura de “5000°C”.
- ❖ Un id_paquete vacío o inexistente.
- ❖ Un timestamp con formato inválido.

Sin medidas de validación, estos errores podrían:

- ❖ Corromper la base de datos.
- ❖ Provocar el fallo del servicio.
- ❖ Generar falsas alarmas o datos incoherentes.

⚙️ Solución Implementada: Validación en 3 Capas

1. Validación en el flujo (n8n / Node-RED):

El primer filtro se realiza en el propio flujo antes de enviar los datos a la API.

Este nodo se encarga de revisar que todos los campos requeridos estén presentes (por ejemplo, temperatura, fuerza_g, inclinación, etc.) y que los tipos sean correctos (números, cadenas, fechas válidas).

Si el formato no cumple las condiciones, el mensaje se descarta y se genera un aviso en los logs.

Esto evita saturar la API con datos inválidos y permite identificar el tipo de error con claridad.

2. Validación en la API (FastAPI + Pydantic):

La segunda capa de defensa se encuentra en la API de ingesta.

Usamos modelos de **Pydantic**, que obligan a que los tipos de datos y los rangos sean físicamente plausibles.

Por ejemplo:

- La temperatura solo se acepta entre -20 °C y 50 °C.
- La fuerza G debe estar entre 0 y 10.
- La inclinación máxima es de ±90°.



Si llega un valor fuera de esos límites, la API responde automáticamente con el código **422 (Unprocessable Entity)** y detalla el motivo del error.

De esta forma, garantizamos que solo se guarden datos coherentes y que la base de datos permanezca limpia.

3. Validación en la Base de Datos (PostgreSQL):

Como última línea de defensa, se implementaron **restricciones (constraints)** a nivel SQL.

Esto asegura que, incluso si algún error escapa de las capas anteriores, la propia base de datos impida la inserción.

Por ejemplo, no se permiten temperaturas negativas extremas, ni valores nulos, ni duplicados en el campo `id_paquete`.

Esta validación estructural es crucial para preservar la integridad del sistema a largo plazo.

Resultado

Con esta arquitectura en tres niveles:

- ❖ Solo se almacenan datos válidos y consistentes.
- ❖ Los errores se detectan y gestionan antes de causar daño.
- ❖ La API responde con mensajes claros y mantiene la trazabilidad de cada intento.

Esto convierte nuestro sistema en una plataforma **segura, robusta y autodefensiva** frente a cualquier intento de inyección o error de formato.

Parte 3: Disponibilidad — “Diseñando para Sobrevivir”

Objetivo:

Garantizar que el sistema siga funcionando incluso cuando haya fallos temporales, sin pérdida de datos.

Problema a Resolver

En entornos reales, los fallos son inevitables: cortes de red, reinicios de la base de datos o actualizaciones del servidor pueden interrumpir momentáneamente la comunicación entre los componentes.

Sin un sistema de reintentos, cada una de esas interrupciones significaría pérdida de datos.



Estrategia Implementada: Reintentos Inteligentes

En el flujo de **n8n**, se diseñó una lógica de **reintentos automáticos con backoff exponencial**.

Esto significa que si una petición a la API falla (por ejemplo, por un error 503), el sistema no descarta el mensaje, sino que:

1. Espera unos segundos.
2. Vuelve a intentarlo.
3. Duplica el tiempo de espera en cada intento hasta un máximo predefinido.

Con esta técnica, se evita saturar el servicio que está intentando recuperarse y se da tiempo suficiente para que la conexión o la base de datos vuelvan a estar disponibles.

Además, el flujo almacena temporalmente los mensajes que no pudieron enviarse en un **buffer de memoria**.

Cuando el sistema detecta que la API vuelve a estar operativa, los reenvía automáticamente, asegurando que **ningún dato se pierda** durante la caída.

Integración con la Base de Datos

En esta parte se utiliza **PostgreSQL**, administrado desde **pgAdmin 4**, donde se almacenan todas las mediciones de telemetría.

Cada registro incluye la información del paquete, su temperatura, fuerzas G, inclinación y demás sensores, junto con la marca temporal de envío.

La base de datos fue configurada con **llaves primarias únicas y tipos de datos estrictos** para garantizar consistencia en todas las inserciones.

Resultado

El flujo resiste desconexiones momentáneas de la API o la base de datos sin perder información.

En las pruebas realizadas, al desconectar PostgreSQL durante un minuto, los datos siguieron acumulándose en el buffer del flujo y fueron reenviados con éxito cuando el sistema se recuperó.

Esta capacidad de resiliencia demuestra que el sistema es **tolerante a fallos y capaz de autorrecuperarse**.

La “Boss Fight”: Prueba de Resiliencia

Durante la simulación final, se recreó un escenario de fallo real: la base de datos se desconectó durante **60 segundos** mientras el simulador seguía enviando datos



desde los sensores.

El flujo de ingesta capturó los errores, aplicó la lógica de reintentos y almacenó temporalmente la información.

Al restaurar la conexión, todos los mensajes pendientes fueron procesados sin pérdidas.

Resultado:

- ❖ 0 datos perdidos / 100 % de recuperación tras la caída.
- ❖ Continuidad temporal sin huecos en los eventos.

Esto demuestra que el sistema no solo es funcional, sino **fiable, consistente y resistente ante fallos críticos**.



LA SIMULACIÓN DE LA BOSS FIGHT LA MOSTRAMOS EN EL VIDEO

9. Conclusión

El desarrollo del sistema de telemetría y detección de anomalías ha permitido construir un flujo completo de análisis IoT, desde la generación y transmisión de datos simulados mediante MQTT, hasta su procesamiento, almacenamiento en PostgreSQL y evaluación de rendimiento del modelo de detección.

Los resultados muestran que el sistema cumple correctamente con la detección de incidentes, aunque aún existen áreas de mejora. La matriz de confusión evidencia una alta precisión (1.0), lo que significa que el modelo casi no genera falsos positivos; sin embargo, el recall (0.5) indica que algunos incidentes reales no son detectados, por lo que el modelo tiende a ser conservador. Esto se refleja también en el F1-score (0.66), ligeramente por debajo del umbral objetivo del 0.7.

En cuanto a los indicadores de rendimiento (KPIs), se observa que solo un 33.3 % de los envíos cumplen el SLA, lo que sugiere la necesidad de optimizar la transmisión y el control de latencias. Aun así, el tiempo medio de detección (10 s) se mantiene muy por debajo de la meta (<30 s), demostrando una buena capacidad de respuesta del sistema. Finalmente, la tasa de falsos positivos del 15 % se considera aceptable para una primera versión, pero puede mejorarse ajustando los umbrales y filtrados.

En conjunto, el trabajo demuestra la viabilidad técnica del flujo IoT-IA-Base de Datos, proporcionando una base sólida para futuras fases del proyecto, donde se podrían incorporar modelos predictivos más robustos, alertas inteligentes y dashboards en tiempo real que faciliten la supervisión operativa.



LANZAR API EN CARPETA INGEST_API → unicorn main:app --reload --port 8000

DELETE FROM telemetry;

DELETE FROM alerts;

CONTRASEÑA ADMINP4 → 1234