

Modernización de contadores de tránsito con comunicación bidireccional

Ing. Diego Aníbal Vázquez

Carrera de Especialización en Internet de las Cosas

Director: Ing. Rogelio Diego González (FIUBA)

Jurados:

Esp. Ing. Martín Anibal Lacheski (FIUBA)
Esp. Ing. Emiliano Rodríguez Mora (FIUBA)
Esp. Ing. Martín Duarte (FIUBA)

Ciudad de Buenos Aires, diciembre de 2025

Resumen

Esta memoria describe el desarrollo e implementación de un sistema para el registro de eventos de tránsito y la transmisión segura de datos. El diseño asegura la disponibilidad de la información incluso ante interrupciones en la conexión a Internet. El sistema fortalece el monitoreo de las rutas nacionales mediante comunicación bidireccional. Permite realizar diagnósticos remotos, actualizar parámetros, incrementar la precisión y consistencia de los datos y optimizar el trabajo del personal técnico y del equipamiento de monitoreo. Para su realización se aplicaron conocimientos de Internet de las cosas, transmisión segura de datos y control de sistemas remotos.

Agradecimientos

Quiero expresar mi más profundo agradecimiento a mi familia, quienes han sido un pilar fundamental en cada etapa de mi vida.

Índice general

Resumen	I
1. Introducción general	1
1.1. Motivación	1
1.1.1. Contexto actual	1
1.1.2. Limitaciones del desarrollo previo	1
1.1.3. Impacto esperado	2
1.1.4. Diseño conceptual	2
1.2. Objetivos	2
1.2.1. Objetivo general	2
1.2.2. Objetivos específicos	2
1.3. Estado del arte y propuesta de valor	3
1.4. Alcance	3
2. Introducción específica	5
2.1. Protocolos y comunicación	5
2.2. Componentes de hardware utilizados	6
2.3. Tecnologías de software aplicadas	8
2.4. Software de control de versiones	9
3. Diseño e implementación	11
3.1. Arquitectura del sistema	11
3.1.1. Flujo de datos	12
3.2. Arquitectura del nodo	14
3.3. Desarrollo del backend	16
3.3.1. Arquitectura y tecnologías	16
3.3.2. Funcionalidades principales	17
3.3.3. Organización en controladores	18
3.3.4. Mapa de endpoints	18
3.3.5. Seguridad y extensibilidad	19
3.4. Desarrollo del frontend	20
3.4.1. Arquitectura y tecnologías	20
3.4.2. Funcionalidades principales	20
3.4.3. Integración con el backend	21
3.5. Desarrollo del firmware	21
3.5.1. Arquitectura general	22
3.5.2. Flujo de trabajo del firmware	22
3.6. Despliegue del sistema	24
3.6.1. Entorno productivo e integración continua	24
3.6.2. Monitoreo post-implantación	26
3.7. Integración con la infraestructura existente	27
4. Ensayos y resultados	29

4.1.	Banco de pruebas	29
4.1.1.	Diseño del entorno de pruebas	29
4.1.2.	Metodología experimental	30
4.1.3.	Resultados y observaciones	31
4.2.	Pruebas de la API REST	32
4.2.1.	Objetivos y alcance	32
4.2.2.	Metodología de prueba	32
4.2.3.	Resultados obtenidos	33
	Medición	33
	Comando	34
	Respuesta	35
4.3.	Pruebas de componentes	36
4.3.1.	Enfoque general	36
4.3.2.	Resultados por componente	37
4.4.	Pruebas del frontend	38
4.4.1.	Objetivos	38
4.4.2.	Metodología	39
4.4.3.	Resultados y observaciones	43
4.5.	Prueba final de integración	43
4.5.1.	Metodología de la prueba	44
4.5.2.	Resultados obtenidos	47
4.6.	Comparación con otras soluciones	48
5.	Conclusiones	51
5.1.	Conclusiones generales	51
5.2.	Trabajo futuro	52
	Bibliografía	53

Índice de figuras

1.1. Diagrama en bloques del sistema.	4
2.1. Contador de tránsito DTEC ¹	6
2.2. Módulo RS-232/TTL ²	7
2.3. Microcontrolador ESP32-C3 utilizado en los nodos de campo ³	7
2.4. Módulo SIM800L ⁴	8
3.1. Diagrama de arquitectura del sistema y el flujo de datos.	12
3.2. Diagrama de secuencia del flujo de datos.	13
3.3. Diagrama de conexión entre los módulos del sistema.	15
3.4. Fotografía contador de tránsito DTEC instalado en campo ⁵	16
3.5. Diagrama de flujo de información del backend.	17
3.6. Diagrama con la disposición de los controladores y flujo de dependencias.	18
3.7. Diagrama de estructura de los componentes por pantalla.	20
3.8. Diagrama de flujo de comunicación con el backend.	21
3.9. Flujo de trabajo del firmware del ESP32-C3.	23
3.10. Arquitectura de despliegue del sistema en entorno cloud on-premise.	25
4.1. Fotografía del banco de pruebas.	30
4.2. Registros de detecciones vehiculares almacenados en la base de datos MySQL.	31
4.3. Intercambio de mensajes entre el backend y el nodo de campo mediante MQTT.	31
4.4. Solicitud POST /api/medicion con todos los campos completos.	33
4.5. Solicitud POST /api/medicion con campos faltantes.	34
4.6. Solicitud POST /api/comando exitosa.	34
4.7. Solicitud POST /api/comando con error por datos incompletos.	34
4.8. Solicitud POST /api/resuesta exitosa.	35
4.9. Solicitud POST /api/resuesta con error de validación.	35
4.10. Interacción entre el nodo de campo (ESP32-C3 + SIM800L) y el backend.	38
4.11. Pantalla de inicio de sesión del frontend.	40
4.12. Panel principal con visualización del listado de dispositivos.	40
4.13. Panel de visualización del dispositivo.	41
4.14. Panel de visualización del dispositivo: preparación para la ejecución de un comando y su ejecución.	41
4.15. Panel de visualización del dispositivo: ejecución de un comando.	42
4.16. Panel de visualización del dispositivo: comando ejecutado y la respuesta pendiente.	42
4.17. Panel de visualización del dispositivo: comando ejecutado y su respuesta.	42
4.18. Panel de visualización del historial de mediciones.	43

4.19. Publicación de una medición por parte del módulo ESP32-C3 y recepción en el backend	44
4.20. Visualización de la medición recibida en la interfaz web del sistema.	45
4.21. Generación de un comando desde la interfaz web.	45
4.22. Visualización consola del módulo ESP32-C3 que registra la recepción del comando.	46
4.23. Flujo de respuesta desde el nodo.	46
4.24. Flujo de recepción de la respuesta en el backend.	47
4.25. Visualización de la respuesta del dispositivo en la interfaz web tras el procesamiento exitoso en el backend.	47

Índice de tablas

3.1. Endpoints REST principales	19
4.1. Resultados de pruebas de endpoints REST	36
4.2. Comparación de la solución propuesta	49

Capítulo 1

Introducción general

En este capítulo se presenta el marco de referencia y la justificación del trabajo. Se expone el contexto que motivó su desarrollo, los problemas detectados en la infraestructura actual, una revisión del estado del arte, la propuesta de valor y el alcance del prototipo planteado.

1.1. Motivación

Este trabajo propone la modernización de los contadores de tránsito instalados en rutas nacionales mediante la renovación de su arquitectura de comunicaciones.

1.1.1. Contexto actual

Actualmente, los equipos registran el paso de vehículos y transmiten eventos al servidor central a través de enlaces GPRS tercerizados. No obstante, no admiten la recepción de comandos ni la obtención de diagnósticos remotos. Esta limitación reduce la capacidad operativa, incrementa los costos de mantenimiento y demora la resolución de fallas, debido a que todo ajuste o reparación requiere una intervención presencial [1], [2].

1.1.2. Limitaciones del desarrollo previo

La propuesta se origina a partir del desarrollo de un contador de tránsito destinado a registrar y transmitir información en campo. Sin embargo, la conexión remota de este dispositivo presenta limitaciones en cuanto a su capacidad de transmisión de datos y carece de mecanismos de control remoto, lo que dificulta tanto la supervisión del funcionamiento como la actualización de sus parámetros. Frente a estas restricciones, surge la necesidad de modernizar el sistema mediante la incorporación de una comunicación bidireccional confiable [3], [2].

El análisis del sistema vigente permitió identificar las siguientes limitaciones que motivan el rediseño:

- Comunicación unidireccional. Los contadores envían datos al servidor, pero no existe un canal para enviar configuraciones, consultas y comandos desde el servidor hacia los equipos. Esta limitación impide realizar diagnósticos remotos y ejecutar acciones correctivas sin presencia física.
- Dependencia de proveedores GPRS tercerizados. La dependencia de servicios contratados genera costos recurrentes y limita el control sobre la calidad y disponibilidad de la conectividad.

- Imposibilidad de actualización remota. Cualquier modificación de parámetros o ajustes de operación requiere intervención en el sitio. Esto incrementa tiempos de mantenimiento, costos logísticos y complica la aplicación rápida de mejoras.
- Falta de telemetría y diagnóstico preventivo. No se dispone de métricas sistemáticas sobre el estado operativo de los equipos (batería, temperatura, errores de hardware o comunicación).
- Riesgo de pérdida de datos ante conectividad intermitente. La ausencia de mecanismos de encolamiento persistente y de políticas claras de reenvío eleva la probabilidad de pérdida o duplicación de eventos cuando la red es inestable.

Estas deficiencias afectan la calidad del servicio de monitoreo, reducen la eficiencia operativa y constituyen los requisitos funcionales que orientan el diseño del prototipo.

1.1.3. Impacto esperado

La modernización permite reducir los desplazamientos de personal técnico y acortar los tiempos de resolución de incidentes con la consiguiente disminución de costos logísticos. La incorporación de telemetría facilita la planificación de intervenciones preventivas en lugar de responder exclusivamente a fallas, lo que optimiza la disponibilidad del servicio y la calidad de los datos recolectados. Además, la adopción de protocolos estandarizados y componentes de código abierto promueve la escalabilidad y la replicabilidad de la solución en distintos tramos de la red vial [4],[5], [6].

1.1.4. Diseño conceptual

La propuesta separa de forma explícita el transporte de mensajes (broker MQTT) de los servicios de aplicación (API REST, almacenamiento y frontend). Esta separación facilita la interoperabilidad con plataformas institucionales existentes y habilita opciones de despliegue flexibles: uso de brokers externos, instalación de brokers locales o modelos híbridos según políticas institucionales y condiciones de conectividad.

1.2. Objetivos

1.2.1. Objetivo general

Modernizar los contadores de tránsito en rutas nacionales mediante la renovación de su arquitectura de comunicaciones para habilitar comunicación bidireccional confiable, control remoto y telemetría de estado.

1.2.2. Objetivos específicos

A continuación, se detallan los objetivos específicos que orientan el desarrollo del trabajo:

- Garantizar la entrega de eventos aun con conectividad intermitente.

- Implementar mecanismos de encolado y reintento que eviten duplicaciones y pérdidas de información.
- Habilitar la ejecución remota de comandos y la actualización de parámetros desde un servidor central, con confirmación explícita del estado del equipo.
- Disminuir la dependencia de enlaces tercerizados mediante una arquitectura configurable.
- Permitir la modificación remota de parámetros operativos y la carga de ajustes sin desplazamientos.
- Incorporar telemetría de estado (batería, temperatura y códigos de error) para mantenimiento preventivo.
- Validar la viabilidad técnica y la robustez operativa del sistema.

1.3. Estado del arte y propuesta de valor

En el mercado existen soluciones comerciales [7], [8], que ofrecen gestión remota y comunicación bidireccional para dispositivos de campo. Dichas soluciones suelen incluir plataformas propietarias con soporte técnico, servicios administrados y herramientas de análisis avanzadas. Estas alternativas, sin embargo, presentan costos elevados de adquisición y mantenimiento, así como dependencias tecnológicas que restringen la posibilidad de adaptación a condiciones locales específicas.

En el ámbito académico y técnico también se han documentado diversas experiencias orientadas a la gestión remota de infraestructuras distribuidas mediante protocolos de mensajería ligera como MQTT o tecnologías de comunicación celular [9], [10]. Estos trabajos resaltan la eficiencia de los modelos de publicación y suscripción, especialmente en entornos con restricciones de conectividad, pero en muchos casos se centran en aplicaciones industriales que difieren de las condiciones propias de las rutas nacionales [11].

Además, se han desarrollado plataformas de monitoreo basadas en API REST y bases de datos relacionales, que permiten la integración con interfaces web para la visualización y control [12]. Estas experiencias muestran la tendencia hacia arquitecturas abiertas y modulares, aunque su adopción práctica suele estar ligada a contextos con mayor disponibilidad de infraestructura y recursos.

El estado del arte muestra soluciones maduras para la gestión remota y la comunicación bidireccional, aunque con limitaciones asociadas a costos elevados, rigidez tecnológica o requerimientos de infraestructura. Estas restricciones evidencian la necesidad de alternativas específicas y adaptadas a entornos viales argentinos, donde la conectividad suele ser intermitente.

1.4. Alcance

El trabajo desarrolla un prototipo funcional destinado a validar las hipótesis técnicas y operativas. El alcance comprende los siguientes objetivos y límites:

- Rediseño de comunicaciones: implementación de un modelo bidireccional y seguro entre los dispositivos de conteo y el servidor central, basado en MQTT sobre GPRS y con autenticación por credenciales.
- Gestión de mensajes en el dispositivo: encolamiento en memoria RAM con política FIFO, control de reintentos ante fallos de conexión y lógica de descarte cuando la cola alcance su límite definido.
- Backend y persistencia: desarrollo de una API REST que se suscriba al broker MQTT, procese los eventos y almacene los registros en una base de datos relacional para consulta histórica.
- Interfaz web básica: panel de visualización en tiempo real de eventos de tránsito y módulo para envío de comandos y verificación de estado de los dispositivos.
- Funciones de telemetría: reporte de nivel de batería, temperatura y códigos de error para facilitar el diagnóstico remoto.

El alcance técnico incluye la adaptación de un contador existente para comunicarse bidireccionalmente por GPRS, que emplee MQTT como protocolo de mensajería ligera y confiable. Asimismo, comprende el desarrollo de un backend que reciba, procese y persista los eventos en una base de datos relacional. Además, una interfaz web básica que permita la visualización en tiempo real y el envío de comandos hacia los dispositivos. Se priorizan las funciones que validen la cadena completa: adquisición de eventos desde el sistema de detección, encolamiento local con política FIFO, reenvío seguro cuando haya conectividad, recepción y ejecución de comandos y verificación del estado del dispositivo tras cada acción.

La figura 1.1 muestra el diagrama en bloques del sistema. El dispositivo de conteo envía datos por GPRS a un broker MQTT, el servidor central los procesa y almacena en una base de datos, mientras que una API REST permite su consulta y la ejecución de comandos desde una interfaz web.

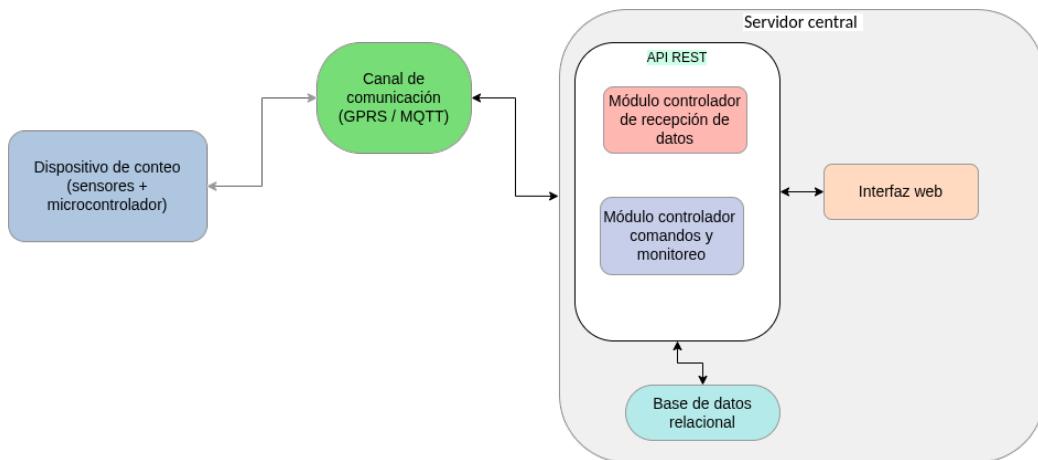


FIGURA 1.1. Diagrama en bloques del sistema.

Capítulo 2

Introducción específica

En este capítulo se detallan los aspectos técnicos específicos que constituyen la base del trabajo. En primer lugar, se describen los protocolos de comunicación que se emplean y la función de cada uno de ellos en la transmisión de datos. Luego, se presentan los componentes de hardware que utiliza el prototipo. A continuación, se analizan las tecnologías de software que integran la solución, la herramienta de control de versiones adoptada para el desarrollo colaborativo y la gestión del código fuente.

2.1. Protocolos y comunicación

El diseño de un sistema de conteo de tránsito con comunicación bidireccional demanda la incorporación de protocolos que aseguren confiabilidad y eficiencia en el intercambio de datos. En este trabajo se integran tres tecnologías principales:

- RS-232: es un estándar de comunicación serial utilizado tradicionalmente en sistemas embebidos. Permite la transmisión de datos punto a punto entre el contador de tránsito y el microcontrolador ESP32-C3 [13]. Su simplicidad lo hace adecuado para distancias cortas y ambientes donde la interferencia es controlada. Aunque se trata de un protocolo clásico, su adopción garantiza compatibilidad con dispositivos que aún dependen de interfaces seriales [14], [15].
- MQTT: este protocolo de mensajería ligera se emplea tanto para la transmisión de eventos de tránsito desde los dispositivos hacia el servidor central como para la recepción de comandos en sentido inverso. MQTT opera sobre TCP/IP [16] y emplea un modelo de publicación/suscripción a través de un broker, lo que facilita la escalabilidad y la integración de múltiples dispositivos. Además, permite implementar mecanismos de calidad de servicio (QoS) que reducen la probabilidad de pérdida de datos en condiciones de conectividad inestable, lo que resulta crucial para el escenario de rutas nacionales [17].
- API REST: constituye la interfaz de comunicación entre el backend y los clientes web. Su inclusión en el trabajo posibilita la consulta y el envío de información de manera estructurada, mediante operaciones estándar (GET, POST, PUT, DELETE). La API REST asegura la interoperabilidad con diferentes plataformas y brinda flexibilidad para desarrollar aplicaciones adicionales que utilicen los datos recolectados [18], [19].

La combinación de estos protocolos permite cubrir distintos niveles de la arquitectura: comunicación local (RS-232), comunicación de dispositivos con el servidor (MQTT) y comunicación entre el servidor y las aplicaciones de usuario (API REST). De esta forma, se garantiza un flujo de datos seguro, confiable y bidireccional.

2.2. Componentes de hardware utilizados

El prototipo se implementa con un conjunto de componentes de hardware seleccionados por su disponibilidad, costo y adecuación al entorno de operación:

- Contador de tránsito: dispositivo de campo encargado de detectar y acumular el paso de vehículos y generar tramas de datos que contienen información de conteo, clasificación y marcas temporales. Estos equipos permiten registrar de manera continua el flujo vehicular en rutas nacionales y constituyen la base del sistema de monitoreo. Si bien en el mercado existen distintos contadores comerciales que ofrecen funcionalidades similares, muchos de ellos presentan altos costos de adquisición, dependencia de plataformas propietarias o limitaciones de integración. En este trabajo se optó por utilizar un contador desarrollado por la Dirección Nacional de Vialidad, que ha sido probado en distintas rutas nacionales del país y cuenta con una interfaz de salida RS-232 [15] con un adaptador MAX232 [20]. Esta elección responde a criterios de soberanía tecnológica, optimización de recursos ya disponibles y reducción de costos. Además, el contador no solo transmite información de eventos, sino que también admite la recepción de comandos externos a través del nodo, lo que habilita funcionalidades de diagnóstico, reconfiguración remota y confirmación de estado. De este forma, se garantiza la compatibilidad con la infraestructura existente y se potencia su integración dentro de una arquitectura moderna basada en protocolos abiertos.

En la figura 2.1 se observa el contador de tránsito. Este dispositivo constituye la base del sistema de detección de eventos.



FIGURA 2.1. Contador de tránsito DTEC ¹.

¹Imagen tomada de <http://transito.vialidad.gob.ar/>

- Módulo de adaptación RS-232/TTL (MAX232): este circuito se encarga de convertir los niveles de tensión de forma bidireccional, protege los dispositivos y garantiza una transmisión de datos confiable y libre de errores [20].

En la figura 2.2 se observa el módulo de adaptación RS-232/TTL (MAX232).



FIGURA 2.2. Módulo RS-232/TTL ².

- ESP32-C3: microcontrolador de bajo consumo que actúa como unidad de procesamiento y comunicación. Integra conectividad y capacidad de comunicación serial, lo que lo hace adecuado para la interacción con el contador y con el módulo GPRS [21].

En la figura 2.3 se muestra el microntrolador utilizado en campo.

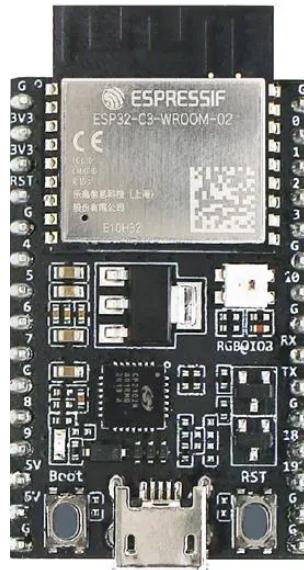


FIGURA 2.3. Microcontrolador ESP32-C3 utilizado en los nodos de campo ³.

- Módulo GPRS SIM800L: permite la conexión de los dispositivos a la red celular [22], que asegura la transmisión de datos al servidor central mediante MQTT. Se eligió por su bajo costo, disponibilidad en el mercado y facilidad de integración con el ESP32-C3.

²Imagen tomada de <https://www.alldatasheet.com/datasheet-pdf/pdf/73074/MAXIM/MAX232.html>

³Imagen tomada de <https://docs.espressif.com/projects/esp-idf/en/v5.0/esp32c3/hw-reference/esp32c3/user-guide-devkitm-1.html>

En la figura 2.4 se aprecia el módulo SIM800L que implementa la conectividad celular GPRS.



FIGURA 2.4. Módulo SIM800L⁴.

- Servidor central: responsable de recibir los datos transmitidos, almacenarlos en una base de datos y responder a las solicitudes de los usuarios.
- PC con navegador web: constituye el medio de interacción del usuario final con el sistema, a través de la interfaz desarrollada.
- Fuente de alimentación: fuente principal con batería interna recargable y recarga mediante panel solar, capaz de cubrir los picos de consumo del SIM800L. Incluye regulador de tensión y capacitores que estabilizan el voltaje y evitan reinicios.
- Carcasa y gabinete: protección para el contador y el módulo de comunicación (ESP32-C3 y SIM800L) en un gabinete cerrado resistente a humedad, polvo y vibraciones, con disipación térmica y reducción de interferencias electromagnéticas.
- Componentes adicionales: filtros (capacitores) para garantizar estabilidad y evitar reinicios inesperados.

La integración de estos componentes asegura que el sistema pueda operar en condiciones reales y que cumpla con los requisitos.

2.3. Tecnologías de software aplicadas

El desarrollo del prototipo se sustenta en la integración de tecnologías de software abiertas y estandarizadas, seleccionadas por su solidez, amplia adopción en la comunidad tecnológica y capacidad para interoperar de manera eficiente entre los distintos componentes del sistema:

- MQTT con Eclipse Mosquitto. La mensajería ligera y bidireccional se implementa mediante el protocolo MQTT, operando sobre el broker Eclipse

⁴Imagen tomada de <https://www.alldatasheet.com/datasheet-pdf/pdf/1741389/SIMCOM/SIM800L.html>

Mosquitto [23]. Este software es ampliamente utilizado en entornos de Internet de las Cosas (IoT).

- API REST con Node.js y Express. El backend se desarrolla en Node.js [24], un entorno de ejecución orientado a aplicaciones de red no bloqueantes, y se estructura con Express [25], un framework ligero que facilita la creación de servicios RESTful.
- Base de datos relacional MySQL. La persistencia de los datos se implementa en MySQL [26], un sistema gestor de bases de datos relacionales ampliamente adoptado. En ella se almacenan de manera estructurada tanto los eventos de tránsito como los estados reportados por los equipos.
- ORM con Sequelize. Para abstraer la interacción con la base de datos y mantener independencia frente a cambios en la capa de persistencia, se utiliza Sequelize [27], un ORM para Node.js que permite definir modelos y relaciones de manera declarativa.
- Sistema de logging con Winston. La trazabilidad de eventos y errores se gestiona mediante Winston [28], una biblioteca de logging que soporta múltiples transportes y permite configurar niveles de severidad, formatos y timestamps.
- Middleware de registro HTTP con Morgan. Para capturar y auditar el tráfico entrante al backend se emplea Morgan [29], un middleware especializado en logging de peticiones HTTP, que complementa funcionalidad de Winston.
- Interfaz web con Ionic y Angular. Para la interacción con el usuario final se diseñó una aplicación accesible desde cualquier navegador, construida con Ionic [30] y Angular [31]. La primera aporta componentes visuales responsivos que aseguran usabilidad tanto en entornos de escritorio como en dispositivos móviles.
- Orquestación con Docker Compose. Para simplificar el despliegue y la gestión de los servicios del backend, se utiliza Docker Compose [32]. Esto permite levantar de manera consistente los contenedores de la API REST, el broker MQTT y la base de datos MySQL, que asegura que todas las dependencias se inicien en el orden correcto y facilita la replicación del entorno en desarrollo, prueba y producción.
- Firmware para ESP32-C3 con ESP-IDF. El microcontrolador ejecuta un firmware desarrollado en C/C++, el cual utiliza el framework oficial de ESP-IDF [33], el framework oficial de Espressif. Este entorno proporciona bibliotecas optimizadas. El firmware controla la captura de datos desde el contador mediante RS-232, gestiona la comunicación con el módulo SIM800L mediante comandos AT y establece la conexión con el broker Mosquitto a través de MQTT.

2.4. Software de control de versiones

Para la gestión del código fuente se empleó GitHub [34], una plataforma que se basa en el sistema de control de versiones Git. Esta herramienta permitió almacenar el repositorio central de manera segura, registrar el historial de cambios

y garantizar la trazabilidad de cada modificación realizada durante la etapa de desarrollo del prototipo.

El repositorio incluye el firmware del ESP32-C3, el backend en Node.js con Express y la interfaz web desarrollada con Ionic y Angular. Esto facilita mantener el código organizado, con la posibilidad de crear ramas independientes para implementar nuevas funciones y, posteriormente, integrarlas al código principal mediante pull requests, lo que permite revisar y validar los cambios antes de su incorporación definitiva.

Asimismo, se emplean issues para documentar incidencias, planificar tareas y realizar el seguimiento de los avances. Esta práctica favorece la organización del trabajo y permite mantener un registro claro de los problemas detectados y las decisiones adoptadas.

Capítulo 3

Diseño e implementación

En este capítulo se describe la arquitectura global del prototipo, se detalla cada módulo de hardware y software que lo compone, y se documentan las decisiones de implementación y los criterios de diseño. Se explican los flujos de datos entre el dispositivo de campo, el broker MQTT, el backend (API REST), la interfaz web, se resumen las consideraciones para el despliegue y el monitoreo post-implantación.

3.1. Arquitectura del sistema

La arquitectura propuesta separa de forma explícita el dispositivo de campo (contador + ESP32-C3 + SIM800L), el transporte de mensajes (broker MQTT) y los servicios de aplicación (API REST, persistencia y frontend). Esta separación facilita la interoperabilidad y permite desplegar la solución de forma local, remota o híbrida según las políticas institucionales.

El sistema se organiza en cinco bloques con funciones definidas que aseguran un flujo de datos confiable, eficiente y seguro durante la captura, transmisión, procesamiento y visualización de eventos, que garantiza trazabilidad, persistencia y control remoto. A continuación, se describen los bloques y sus responsabilidades:

- Dispositivo de campo: integra el contador (RS-232), un ESP32-C3 y un módem SIM800L. El firmware, basado en ESP-IDF, lee y parsea tramas, valida y normaliza campos, agrega sello UTC, encola eventos y publica mensajes mediante MQTT. Gestiona reintentos, comandos y telemetría, y mantiene una persistencia mínima (últimas tramas y comandos pendientes) para recuperación tras reinicio.
- Transporte (broker MQTT): funciona como bus de mensajes desacoplado. Se sugiere usar Eclipse Mosquitto en la etapa inicial y considerar brokers gestionados para escalar. Implementa autenticación, control de tópicos y cifrado. Se emplean tópicos jerárquicos por dispositivo para facilitar filtrado y autorización:
 - dispositivo/{id}/medicion
 - dispositivo/{id}/comando
 - dispositivo/{id}/respuesta
- Servidor central: el servidor central reúne dos responsabilidades principales:

- Componente suscriptor MQTT que valida, transforma y enruta mensajes hacia la lógica de negocio y la persistencia.
- API REST ofrece servicios de consulta, gestión y comandos, manteniendo independencia del broker para permitir nuevos consumidores. Los datos se almacenan en MySQL con soporte para rangos temporales, alto rendimiento y auditoría de comandos.
- Cliente/Visualización: la interfaz web, desarrollada en Ionic + Angular, consume la API REST para consultas históricas y eventos en tiempo real. Ofrece visualización de eventos, consultas filtradas, envío de comandos y un panel de telemetría para mantenimiento.

Se separó el broker de la aplicación, se usó MQTT por su eficiencia y se creó una API REST en Node.js para gestión y autenticación.

La figura 3.1 muestra el diagrama de arquitectura del sistema y el flujo de datos.

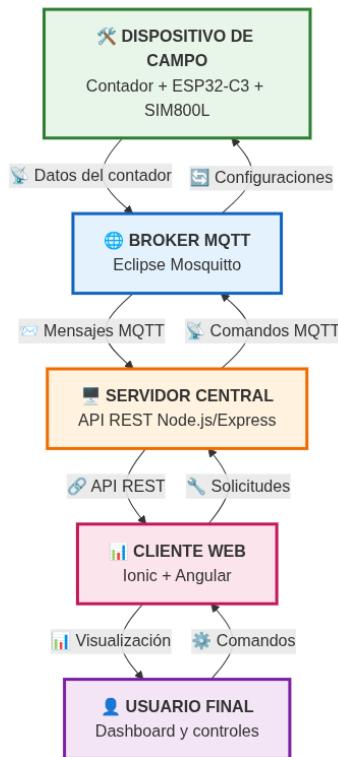


FIGURA 3.1. Diagrama de arquitectura del sistema y el flujo de datos.

3.1.1. Flujo de datos

El flujo de datos del sistema describe cómo se captura, procesa y comunica la información desde el contador hasta la interfaz de usuario, lo que permite la trazabilidad, persistencia y control de los eventos y comandos. Se detallan las etapas principales:

- Detección: el contador detecta un paso, acumula y en determinado intervalo envía una trama por RS-232 al ESP32-C3.

- Preprocesado en nodo: el firmware valida la trama, añade sello temporal y metadatos, y encola el evento en memoria (FIFO).
- Transmisión: cuando la conexión GPRS está disponible, el nodo publica las mediciones en el tópico MQTT dispositivo/id/medicion.
- Ingesta y persistencia: el broker Mosquitto entrega el mensaje al suscriptor backend, el servicio valida el payload y persiste el registro en la base de datos MySQL.
- Visualización/Control: la interfaz web consulta la API REST para datos históricos y recibe notificaciones en tiempo real.
- Emisión de comandos (desde UI): el operador genera un comando en la interfaz, la UI envía POST /app/comando al backend, que crea un comm_id único y publica en dispositivo/id/comando.
- Recepción nodo/Entrega al contador: el ESP32-C3 en dispositivo/id/comando recibe el comando, valida cmd_id y lo envía al contador por RS-232, se aplica un timeout configurable por comando.
- Ejecución y ack: el contador ejecuta la orden y responde por RS-232, el firmware publica el ack/resultado en dispositivo/id/respuesta con cmd_id y status (ok, failed, timeout, value).
- Actualización en backend y UI: el suscriptor MQTT del backend recibe el ack, actualiza la tabla respuesta (campo valor, ack_ts) y notifica a la UI para que el operador vea el resultado.

La figura 3.2 muestra el diagrama de secuencia del flujo de datos.

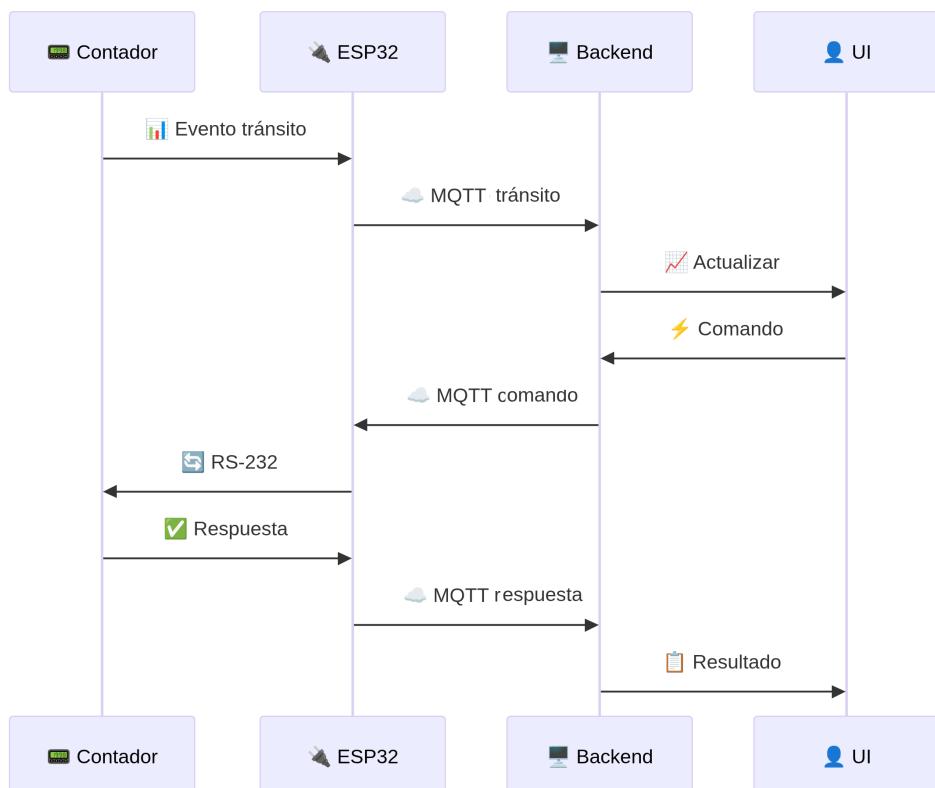


FIGURA 3.2. Diagrama de secuencia del flujo de datos.

3.2. Arquitectura del nodo

La arquitectura de cada nodo se diseñó con el objetivo de reutilizar los contadores de tránsito actualmente desplegados en las rutas nacionales. Cada nodo integra varios módulos que permiten la adquisición, procesamiento y transmisión de los datos de tránsito:

- Contador de tránsito modelo DTEC: dispositivo encargado de detectar el paso de vehículos y generar tramas de datos con la información del evento.
- Módulo de adaptación RS-232/TTL (MAX232): circuito de conversión de niveles eléctricos que asegura compatibilidad entre la interfaz serial del contador (RS-232) y el microcontrolador (niveles TTL).
- ESP32-C3: microcontrolador que ejecuta el firmware desarrollado sobre ESP-IDF. Sus funciones incluyen el preprocesamiento de eventos, el encolamiento FIFO, la suscripción a comandos remotos y la gestión integral de la comunicación con el servidor.
- Módulo de comunicación GPRS (SIM800L): interfaz de conectividad celular que publica eventos en el broker MQTT, recibe comandos desde el servidor y retransmite respuestas o estados del nodo.
- Alimentación y montaje: para garantizar el funcionamiento continuo del nodo en entornos de campo, se deben considerar dos aspectos fundamentales:
 - Fuente de alimentación: se implementó un sistema de energía basado en una batería interna recargable, específicamente dimensionada para cubrir los picos de consumo del módulo SIM800L durante las transmisiones de datos. La autonomía del sistema está garantizada mediante un panel solar que mantiene la carga de la batería de forma continua.

Para estabilizar la entrega de energía se incorporó un módulo regulador de tensión que garantiza el nivel adecuado de voltaje para el módem. El circuito se complementó con capacitores dimensionados para absorber los picos de tensión del SIM800L, que evita caídas de voltaje que puedan reiniciar el sistema.

- Carcasa y gabinete: tanto el contador como el módulo de comunicación remota (ESP32-C3 y SIM800L) cuentan con su propia carcasa de protección y, adicionalmente, ambos se alojan en un gabinete metálico estandarizado ya existente en la infraestructura vial, diseñado para resistir humedad, polvo y vibraciones. Esta solución aprovecha la protección ambiental, disipación térmica y blindaje electromagnético del gabinete original, que garantiza la confiabilidad del sistema en condiciones de intemperie.

La figura 3.3 se presenta el diagrama de conexión entre los módulos del sistema, que detalla las líneas de comunicación serie RS232, la interfaz UART entre el microcontrolador y el módem GSM, así como la distribución de la alimentación eléctrica y los circuitos de estabilización de potencia.

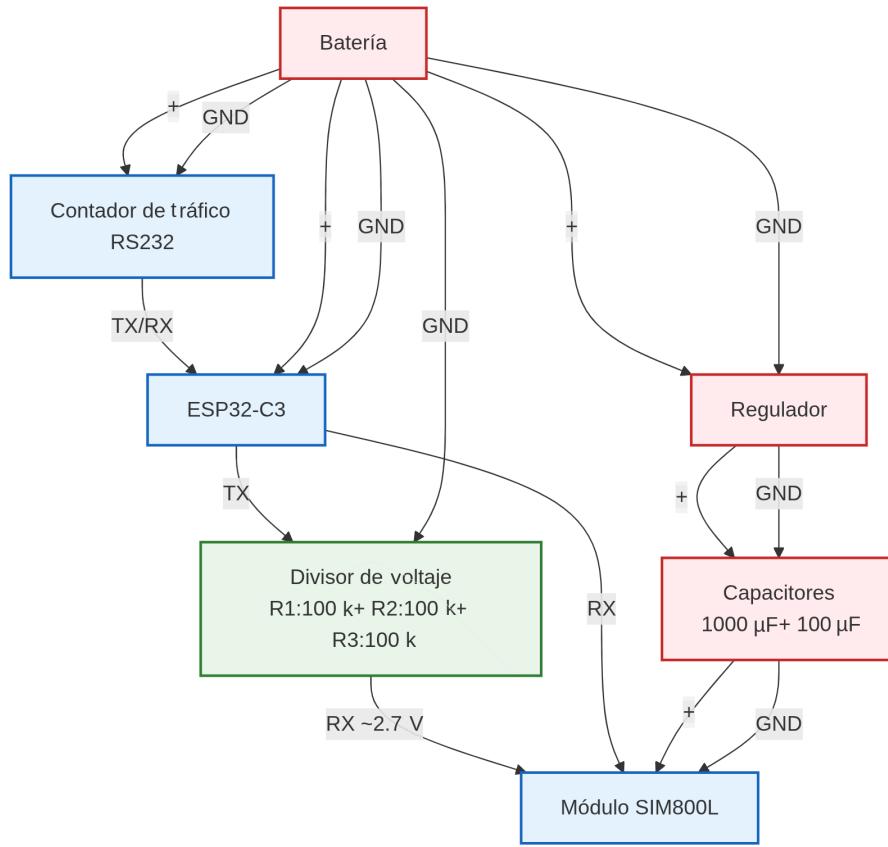


FIGURA 3.3. Diagrama de conexión entre los módulos del sistema.

En la figura 3.4 se observa el contador de tránsito instalado en campo, junto con su gabinete de protección y la batería interna que asegura autonomía energética. El conjunto está montado al costado de la ruta, en condiciones reales de operación, lo que permite apreciar el encapsulado diseñado para resistir las exigencias ambientales del entorno vial. La instalación forma parte de la infraestructura existente y se integra sin modificar la estructura original. El gabinete metálico proporciona aislamiento frente a humedad, polvo y vibraciones propios del entorno carretero. Además, ofrece blindaje electromagnético que reduce interferencias y garantiza el funcionamiento estable del sistema. La batería interna mantiene el suministro de energía durante períodos sin radiación solar o interrupciones en la carga, lo que asegura continuidad operativa y confiabilidad en mediciones prolongadas en intemperie.



FIGURA 3.4. Fotografía contador de tránsito DTEC instalado en campo¹.

3.3. Desarrollo del backend

El backend es el núcleo lógico del sistema, encargado de integrar dispositivos, base de datos e interfaz. Su diseño prioriza la modularidad, escalabilidad y seguridad, con tecnologías comunes en entornos IoT.

3.3.1. Arquitectura y tecnologías

El servicio se implementó en Node.js con Express, organizando la aplicación en controladores, rutas y middlewares, y usando Sequelize [27], un ORM [35] que facilita los modelos y asegura independencia de la persistencia. La comunicación con los dispositivos se realiza mediante tópicos MQTT en Eclipse Mosquitto. Las mediciones se publican en el tópico dispositivo/id/medicion, mientras que el backend se encarga de validarlas y almacenarlas en MySQL. Por su parte, los comandos y respuestas se gestionan mediante los tópicos dispositivo/id/ comando y dispositivo/id/respuesta, respectivamente. El despliegue del sistema se realiza con Docker Compose [32], que permite ejecutar el backend, base de datos y el broker [17] en contenedores independientes. Además, el registro y la trazabilidad se gestionan con Winston [28] y Morgan [29], lo que garantiza un monitoreo completo del sistema.

¹Imagen tomada de <http://transito.vialidad.gob.ar/>

En la figura 3.5 se observa el diagrama de flujo de información del backend.

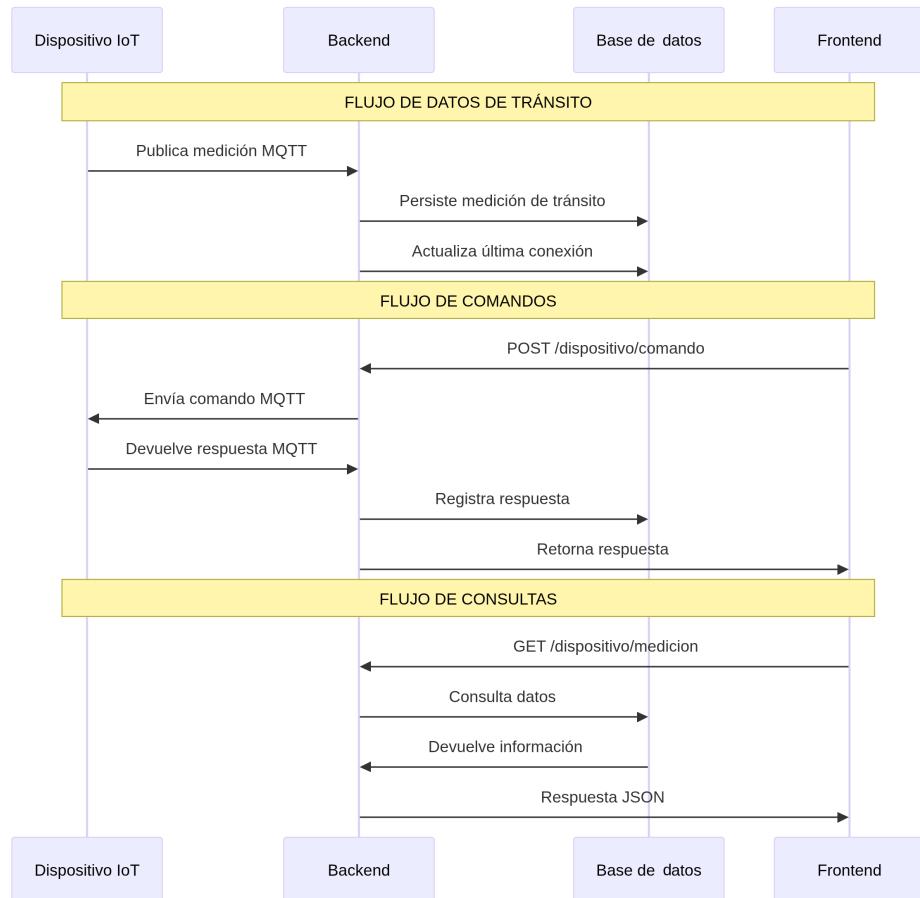


FIGURA 3.5. Diagrama de flujo de información del backend.

3.3.2. Funcionalidades principales

El sistema cuenta con varias funcionalidades esenciales que permiten gestionar de manera eficiente los dispositivos, los eventos generados por ellos, los comandos enviados y la seguridad de acceso. Estas funcionalidades se detallan a continuación:

- Gestión de dispositivos: alta, baja, modificación y consulta.
- Gestión de eventos: almacenamiento de detecciones y consultas filtradas por dispositivo o rango temporal.
- Gestión de comandos: emisión de órdenes a un dispositivo, persistencia de la orden con identificador único cmd_id y actualización según respuesta.
- Estado de dispositivos: consulta de parámetros como nivel de batería, temperatura o conectividad.
- Autenticación y autorización: control de acceso mediante tokens JWT.

3.3.3. Organización en controladores

La lógica de negocio del backend se organiza en controladores, cada uno asociado a un recurso del sistema, lo que favorece la separación de responsabilidades, el mantenimiento y la escalabilidad. Los principales controladores son:

- **DispositivoController**: gestiona las operaciones CRUD sobre los dispositivos de campo, además de registrar los eventos recibidos vía MQTT y asociarlos a un dispositivo específico.
- **MedicionController**: encapsula la lógica de ingestión de eventos de tránsito, validación de payloads y persistencia en la base de datos.
- **ComandoController**: administra la emisión y seguimiento de comandos remotos, generando un `cmd_id` único.
- **RespuestaController**: centraliza la recepción de estados y telemetría (batería, conectividad), en respuesta al comando que se envía, esto garantiza que la base de datos refleje la situación en tiempo real.
- **UserController**: implementa el ciclo de vida de usuarios y la autenticación mediante JWT [36], así como la validación de permisos en cada endpoint.

En la figura 3.6 se observa el diagrama con la disposición de los controladores y flujo de dependencias.

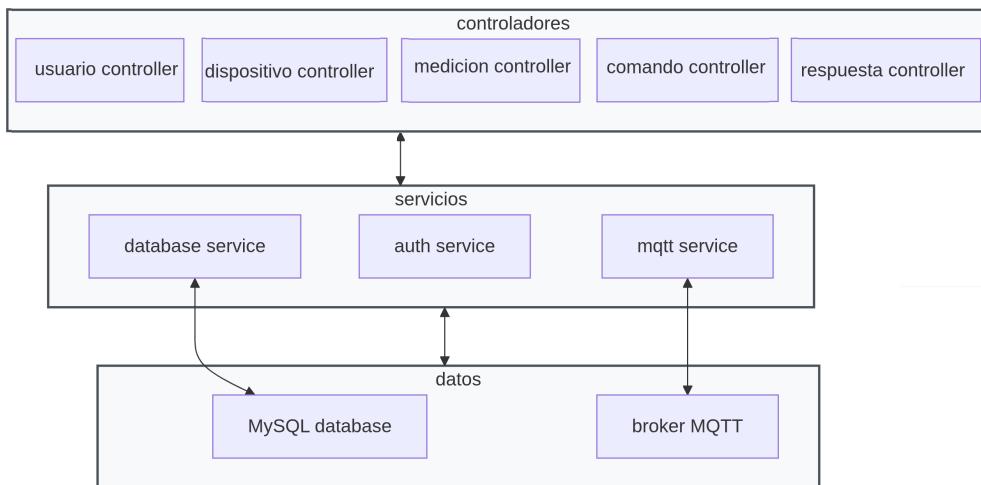


FIGURA 3.6. Diagrama con la disposición de los controladores y flujo de dependencias.

3.3.4. Mapa de endpoints

El backend expone una serie de endpoints REST que conforman la interfaz principal de comunicación con los servicios de aplicación y los dispositivos de campo. En la tabla 3.1 se presentan los endpoints generales.

TABLA 3.1. Endpoints REST principales expuestos por el backend, junto con el controlador que implementa su lógica.

Endpoint	Controlador	Descripción
GET /dispositivo	DispositivoController	Lista todos los dispositivos registrados.
GET /dispositivo/{id}	DispositivoController	Devuelve información de un dispositivo específico.
POST /dispositivo	DispositivoController	Alta de un nuevo dispositivo.
PATCH /dispositivo/{id}	DispositivoController	Actualización de atributos de un dispositivo.
DELETE /dispositivo/{id}	DispositivoController	Eliminación de un dispositivo.
POST /medicion	MedicionController	Crea mediciones de un dispositivo.
GET /medicion/dispositivo/{id}	MedicionController	Consultar mediciones por dispositivo.
GET /medicion/range	MedicionController	Consultar mediciones por rango temporal.
POST /comando	ComandoController	Crear un comando remoto y publicarlo en MQTT.
GET /comando/{id}	ComandoController	Consultar un comando.
GET /respuesta/{id}	RespuestaController	Consultar respuesta de un comando.
POST /usuario/login	UserController	Autenticación de usuario, devuelve token JWT.
POST /usuario	UserController	Alta de usuario.
GET /usuario	UserController	Listar usuarios registrados.
DELETE /usuario/{id}	UserController	Eliminar usuario.

3.3.5. Seguridad y extensibilidad

Además de la autenticación mediante JWT, todos los endpoints aplican validaciones y sanitización de parámetros de entrada y salida. El sistema de logging, implementado con Winston y Morgan, garantiza trazabilidad de las operaciones tanto en la capa HTTP como en la mensajería MQTT. La arquitectura modular basada en controladores permite extender el backend con nuevos recursos o funcionalidades sin afectar la lógica ya implementada.

3.4. Desarrollo del frontend

El frontend, desarrollado como *Single Page Application* en Ionic con Angular y TypeScript, ofrece una interfaz moderna y responsive que permite autenticación, supervisión en tiempo real, consulta de históricos y envío de comandos a los nodos.

3.4.1. Arquitectura y tecnologías

La aplicación se compone de módulos reutilizables de Ionic, optimizados para escritorio y móviles. La comunicación con el backend se realiza mediante API REST y, para actualizaciones en tiempo real, a través de WebSocket.

3.4.2. Funcionalidades principales

El frontend integra las siguientes funciones clave:

- Login de usuario: ingreso con credenciales, validación contra la API y obtención de un token JWT.
- Listado de dispositivos: muestra todos los contadores registrados, junto con información de ubicación y estado básico.
- Detalle de dispositivo: despliega datos específicos de un contador y últimas tramas recibidas.
- Panel de mediciones: permite visualizar los eventos de tránsito procesados, con actualización dinámica cuando el dispositivo transmite nuevas tramas.
- Historial de eventos: consulta de registros almacenados en la base de datos, filtrados por dispositivo y rango temporal.
- Envío de comandos: permite emitir órdenes remotas al nodo (reset, cambio u obtención de parámetros, etc.). El sistema verifica el acuse de recibo y muestra el resultado (ok, failed, timeout o value).

En la figura 3.7 se observa la estructura de los componentes por pantalla.

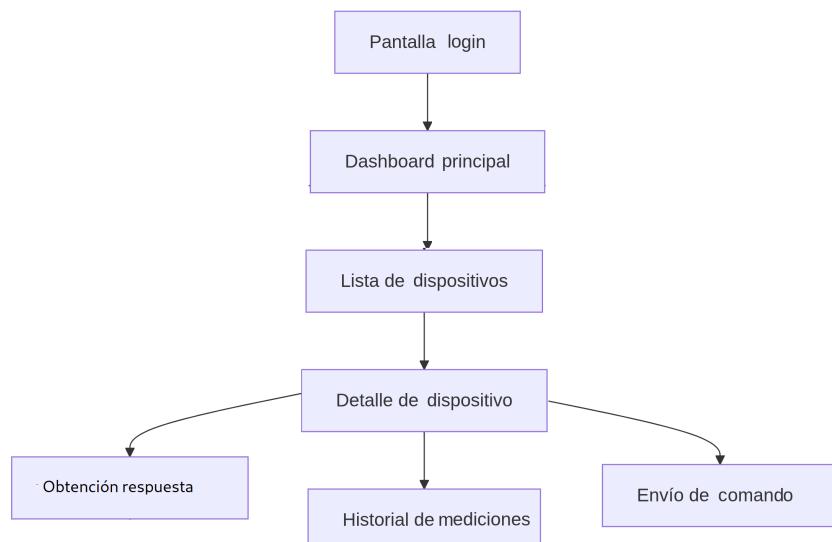


FIGURA 3.7. Diagrama de estructura de los componentes por pantalla.

3.4.3. Integración con el backend

El frontend utiliza los endpoints REST del backend (ver Sección 3.1), enviando en cada petición el token JWT obtenido en el login para asegurar el acceso autorizado. Las respuestas JSON se interpretan en tiempo real, y mantiene la interfaz sincronizada con el estado de los dispositivos. En la figura 3.8 se observa el diagrama de flujo de comunicación con el backend.

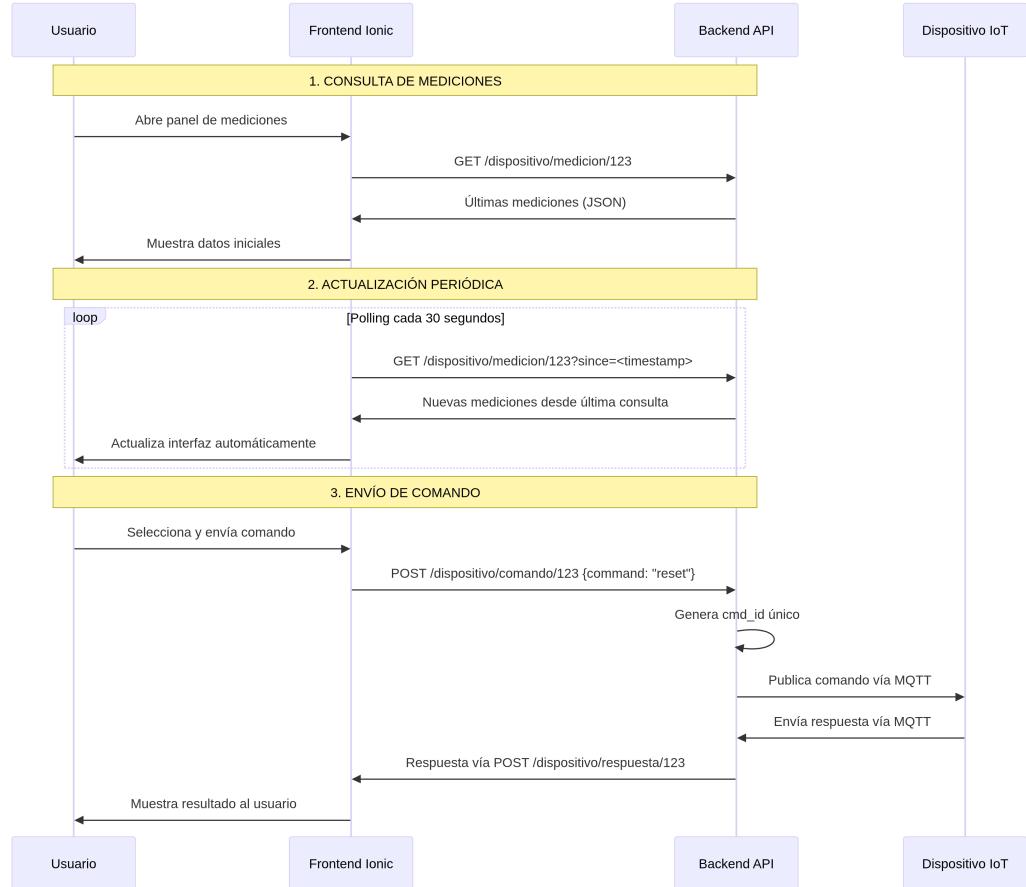


FIGURA 3.8. Diagrama de flujo de comunicación con el backend.

La integración asegura que el frontend pueda operar de manera consistente con el estado real del sistema, sin contradicciones ni demoras que afecten la experiencia del usuario. El resultado se manifiesta en una interfaz clara, estable y alineada con los procesos que el backend ejecuta en segundo plano. Esta arquitectura brinda una plataforma sólida para la incorporación de nuevas funcionalidades, ya que cada módulo se apoya en una comunicación estructurada, verificable y estandarizada.

3.5. Desarrollo del firmware

El firmware del nodo constituye uno de los componentes centrales del sistema, ya que actúa como intermediario entre el contador de tránsito y los servicios remotos. El desarrollo se realizó en lenguaje C utilizando el framework ESP-IDF, con FreeRTOS [37] como sistema operativo de tiempo real.

3.5.1. Arquitectura general

El firmware se estructuró en módulos funcionales independientes, lo que permitió separar responsabilidades y simplificar el mantenimiento. Los módulos principales son:

- Gestión de UART: administra las dos interfaces seriales del dispositivo: una para el módem SIM800L y otra para el contador de tránsito DTEC. Incluye inicialización dinámica, control de errores y asignación de buffers independientes.
- Procesamiento de tramas RS-232: encapsula el parseo de los mensajes emitidos y recibidos por el contador. Valida la estructura JSON, extrae los campos relevantes y construye los objetos internos que se incorporan a la cola de datos. Además, aplica el mismo procedimiento a los comandos que ingresan por la interfaz serial.
- Cola FIFO de eventos: implementa un buffer circular basado en FreeRTOS para asegurar que todas las detecciones permanezcan disponibles aun durante períodos sin conectividad GPRS. La cola evita la pérdida de información.
- Manejo de estados y reconexiones: define una máquina de estados que administra el ciclo GPRS, TCP y MQTT. El firmware detecta automáticamente cortes de red, reinicia el proceso cuando es necesario y reanuda operaciones sin intervención externa.
- Biblioteca SIM800L: se desarrolló una biblioteca específica para encapsular el control del módem. La biblioteca abstrae el envío de comandos AT, verifica respuestas (OK, ERROR, CONNECTOK), administra configuraciones de APN y construye las tramas necesarias para publicar, suscribirse y mantener viva la sesión MQTT.
- Módulo MQTT embebido: implementa la construcción manual de paquetes MQTT, incluyendo mensajes CONNECT, PUBLISH, SUBSCRIBE, PINGREQ y el procesamiento de respuestas como CONNACK, SUBACK y PINGRESP. Este módulo opera sobre TCP crudo, debido a que el SIM800L no ofrece una pila MQTT integrada.
- Módulo de seguridad: incorpora cifrado AES-128 en modo ECB con codificación Base64, aplicable a tramas de tránsitos, comandos y respuestas.

3.5.2. Flujo de trabajo del firmware

El comportamiento del firmware adopta un flujo compuesto por etapas consecutivas:

1. Inicialización del hardware: UART, colas, semáforos y estructuras internas.
2. Verificación del módulo SIM800L y registro en la red celular.
3. Establecimiento de la conexión GPRS y obtención de dirección IP.
4. Apertura de un socket TCP hacia el broker MQTT.
5. Envío del mensaje CONNECT y espera de CONNACK.
6. Suscripción al tópico de comandos.

7. Entrada al ciclo operativo:

- Lectura de tramas RS-232 del contador.
- Almacenamiento en la cola FIFO.
- Extracción y publicación en el tópico correspondiente.
- Procesamiento de comandos recibidos.
- Reenvío de respuestas.
- Envío periódico de CPINGREQ para conservar la sesión.

8. Reinicio automático del ciclo en caso de pérdida de enlace o ausencia de PI NGRESP .

En la figura 3.9 se presenta el diagrama correspondiente al flujo de trabajo del firmware.

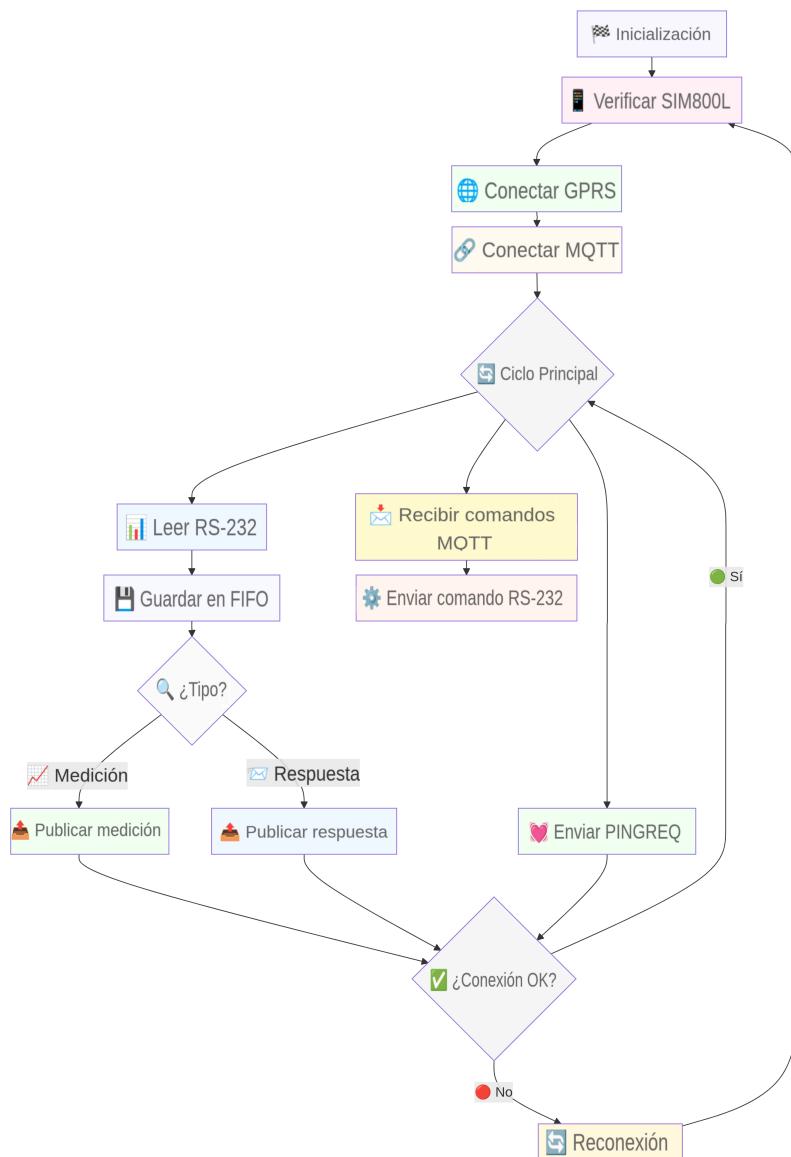


FIGURA 3.9. Flujo de trabajo del firmware del ESP32-C3.

3.6. Despliegue del sistema

El despliegue del sistema comprende la puesta en marcha coordinada de los distintos servicios que componen la arquitectura: el broker MQTT, la API REST, la base de datos relacional y la interfaz web. El objetivo es trasladar el prototipo desde un entorno de desarrollo hacia un entorno productivo, que asegura escalabilidad, confiabilidad y capacidad de monitoreo post-implantación.

3.6.1. Entorno productivo e integración continua

El entorno productivo se implementa bajo un esquema de *cloud on-premise*, es decir, una nube privada alojada en los servidores locales de Vialidad Nacional. Este enfoque ofrece un equilibrio adecuado entre las capacidades de virtualización típicas de las soluciones en la nube y el control directo sobre la infraestructura física. La institución mantiene así autonomía total sobre los recursos, evita la dependencia de proveedores externos y garantiza que los datos operen dentro de un entorno seguro y regulado por políticas internas. Esta estrategia resulta especialmente adecuada para sistemas que procesan información sensible o que requieren disponibilidad permanente sin la incertidumbre asociada a la conectividad externa.

El entorno *on-premise* admite la creación de máquinas virtuales dedicadas a cada componente del sistema. De este modo, la arquitectura evita interferencias entre módulos y conserva la capacidad de asignar recursos según la demanda real de operación. Cada servicio puede recibir un ajuste individual de CPU, memoria o almacenamiento, lo que facilita una administración eficiente y permite reaccionar ante incrementos en el volumen de datos o en la cantidad de dispositivos que participan del sistema. La infraestructura también admite políticas internas de redundancia que refuerzan la tolerancia a fallos y aseguran continuidad de servicio ante incidentes en el hardware principal.

Para la orquestación se utiliza Docker Compose, que ofrece un mecanismo uniforme para instanciar todos los servicios del sistema. Cada módulo (el broker MQTT, la API REST, la base de datos y la aplicación web) opera dentro de su propio contenedor, lo que evita conflictos en las dependencias y define entornos controlados y reproducibles. El sistema mantiene comunicaciones internas mediante redes virtuales de Docker, lo que asegura aislamiento externo y, al mismo tiempo, una conexión eficiente entre servicios relacionados. Este enfoque facilita la escalabilidad horizontal, ya que permite replicar contenedores en caso de aumento de carga o necesidad de tareas distribuidas.

La estandarización que introduce Docker Compose también simplifica la trazabilidad del sistema. Las imágenes de cada servicio representan versiones concretas y verificables, lo que permite retroceder ante fallas en una actualización, comparar configuraciones previas o validar que los entornos de prueba y producción mantengan configuraciones equivalentes. Esta consistencia favorece la depuración de errores y reduce el tiempo de resolución durante incidentes operativos.

La integración continua (CI) cumple un rol esencial dentro del proceso de despliegue. El repositorio del sistema activa una nueva ejecución de la canalización de CI ante cada actualización del código. La canalización crea nuevas imágenes Docker, compila los componentes necesarios, ejecuta pruebas unitarias y valida el comportamiento del sistema antes de autorizar su instalación en el entorno.

on-premise. Este procedimiento asegura que cada versión ingrese al servidor productivo solo después de superar controles de calidad definidos y uniformes.

El proceso también incluye validaciones automáticas que detectan errores de configuración, dependencias faltantes o inconsistencias entre módulos. Si la canalización identifica un problema, detiene el despliegue y envía una notificación al equipo técnico. Esta dinámica evita la propagación de fallas al entorno productivo y elimina la posibilidad de errores manuales durante la instalación. La CI también documenta cada ejecución, lo que permite un seguimiento detallado del historial de actualizaciones, la duración de cada proceso y los cambios incorporados en cada versión.

En conjunto, la combinación del entorno *cloud on-premise*, la orquestación mediante Docker Compose y la implementación de integración continua construye una plataforma productiva robusta, reproducible y adaptable a las necesidades del sistema. La arquitectura permanece preparada para incorporar nuevos servicios, escalar ante aumentos de demanda o responder a contingencias sin comprometer la integridad de los datos ni la disponibilidad del sistema.

En la figura 3.10 se presenta la arquitectura de despliegue del sistema en el entorno productivo de Vialidad Nacional. El diagrama ilustra la organización de los componentes principales, su contenerización mediante Docker y el flujo de integración continua que garantiza la calidad del software antes de su puesta en producción.

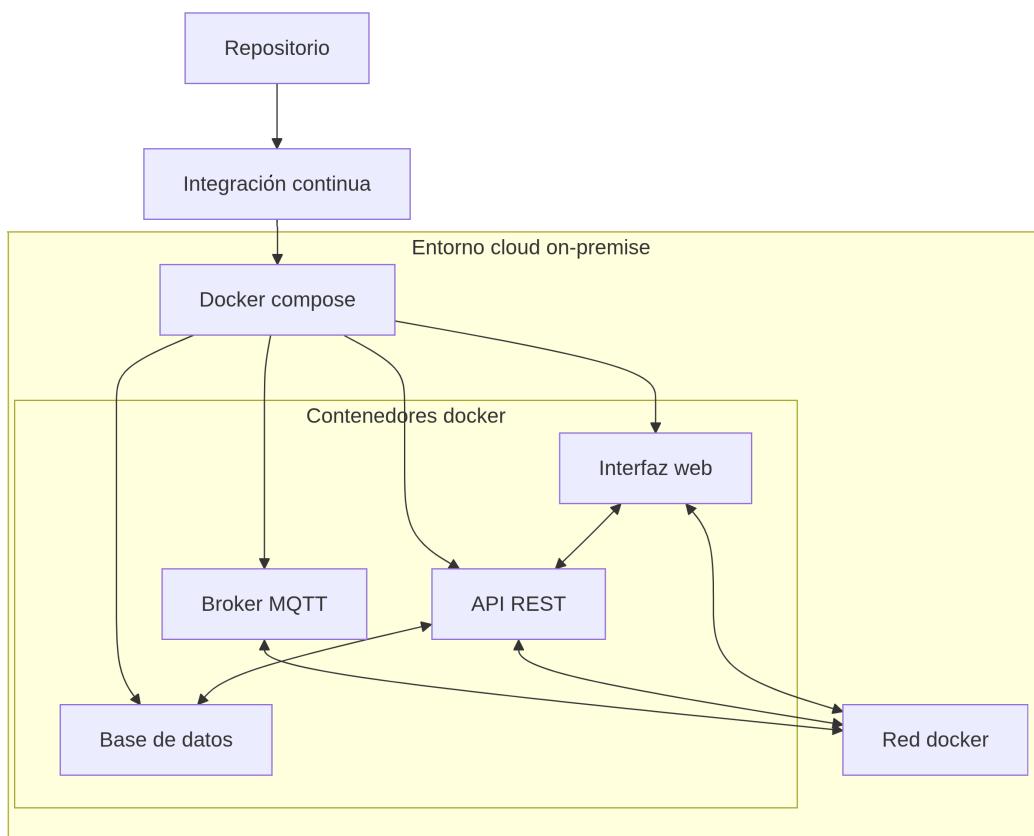


FIGURA 3.10. Arquitectura de despliegue del sistema en entorno *cloud on-premise*.

3.6.2. Monitoreo post-implantación

Una vez que el sistema se encuentra en funcionamiento dentro del entorno real, la etapa de monitoreo adquiere un rol esencial. La operación en campo introduce variaciones, condiciones ambientales cambiantes y situaciones que no siempre aparecen durante las pruebas en laboratorio. Por este motivo, el trabajo incorpora un conjunto de herramientas y prácticas que permiten detectar fallas, medir el desempeño general y asegurar la continuidad del servicio. A continuación, se describen los mecanismos que contribuyen a mantener la estabilidad de la solución y a obtener información precisa para futuras mejoras o ajustes:

- Logs centralizados: el backend y el broker MQTT generan registros detallados de los eventos que procesa cada módulo. Estos registros se almacenan en archivos y también aparecen en la consola de ejecución, lo que permite un análisis inmediato ante incidentes. La solución incluye la integración con Grafana [38], que permite correlacionar los registros del sistema con métricas temporales de los servidores y los contenedores. De esta forma, el operador puede identificar patrones, detectar comportamientos anómalos y reconstruir la secuencia exacta de un problema sin necesidad de recurrir a inspecciones manuales dispersas.
- Alertas y métricas: la plataforma Grafana reúne indicadores provenientes del sistema operativo, de los contenedores y del broker MQTT. Entre los valores más relevantes se encuentran la utilización de CPU, el consumo de memoria, la ocupación del disco y la disponibilidad de los servicios. El panel también muestra información vinculada al tráfico MQTT, como la cantidad de mensajes publicados, la frecuencia de publicaciones, los tiempos de respuesta y las posibles pérdidas durante la transmisión. Estas métricas permiten evaluar la carga del sistema y anticipar situaciones de saturación o degradación antes de que afecten a los usuarios finales.
- Supervisión de dispositivos: la API REST del sistema expone endpoints específicos que informan el estado de los dispositivos instalados en campo. El frontend transforma estos datos en un panel claro y accesible que actúa como tablero de salud. Gracias a esta visualización, el personal técnico identifica dispositivos inactivos, comportamientos inusuales o desvíos respecto del funcionamiento esperado, lo que agiliza las tareas de mantenimiento.
- Respaldo y recuperación: la base de datos incorpora un mecanismo automático de generación de copias de seguridad. Estas copias se producen con una periodicidad definida y permiten restaurar la información en caso de fallas de hardware, corrupción de datos o errores durante una actualización. El sistema admite restauraciones parciales, lo que evita la necesidad de reemplazar toda la base ante inconsistencias en un conjunto acotado de tablas o registros. Esta capacidad resulta crucial, ya que los eventos registrados poseen valor operativo y legal, y su pérdida comprometería el análisis histórico del sistema.

Todos estos mecanismos establecen un entorno de operación confiable y permiten una supervisión continua del sistema después de su implantación. El monitoreo no solo detecta fallas, sino que también proporciona información objetiva para evaluar el rendimiento, planificar optimizaciones y respaldar decisiones técnicas relacionadas con la evolución. Esta etapa garantiza la continuidad del servicio y

crea una base sólida para escalar la solución cuando el número de dispositivos o el volumen de datos aumente en el futuro.

3.7. Integración con la infraestructura existente

Una de las principales ventajas de la arquitectura propuesta consiste en su capacidad para incorporarse al equipamiento vial actual sin introducir modificaciones internas en el contador de tránsito. El nodo de campo recibe los pulsos y tramas del sensor mediante la interfaz RS-232, lo que permite preservar la integridad del hardware legado y evitar alteraciones en los procedimientos de calibración establecidos por el fabricante. Esta decisión técnica también reduce riesgos durante la instalación, ya que el personal solo conecta el módulo externo sin intervenir en el dispositivo de medición original.

El ESP32-C3 no se limita a retransmitir información hacia el servidor central. El módulo ejecuta un conjunto de funciones locales que incrementan la confiabilidad del sistema y disminuyen la carga sobre la infraestructura remota. El procesamiento local incluye la filtración de tramas inválidas o incompletas, la agrupación de eventos en ventanas temporales definidas y la verificación de secuencia para detectar pérdidas o duplicaciones. Estas tareas mejoran la calidad de los datos registrados y permiten mantener coherencia en la información, incluso cuando el enlace de comunicaciones experimenta fallas temporales.

El nodo también mantiene políticas de reintento que aseguran la entrega de los datos. Cuando el módulo detecta ausencia de conectividad, almacena los eventos en memoria local y activa un mecanismo de verificación continua del enlace. Una vez restablecido el acceso a la red, el dispositivo remite los registros pendientes en el orden correcto. Este comportamiento que cada detección ingrese al sistema central con trazabilidad completa.

La comunicación con el servidor se estructura mediante el protocolo MQTT, lo cual facilita la integración con aplicaciones externas y con servicios que Vialidad Nacional ya utiliza en su infraestructura tecnológica. El broker admite suscripciones múltiples, publicación con calidad de servicio y retención de mensajes, lo que facilita la incorporación del sistema en plataformas de monitoreo existentes, tableros operativos o repositorios de análisis histórico sin requerir modificaciones de fondo. La interoperabilidad resulta especialmente valiosa para proyectos que evolucionan con el tiempo o que requieren interacción con sistemas de terceros.

En este marco, los nodos de campo cumplen un doble rol. Por un lado, operan como captadores de información proveniente de los sensores de tránsito, con capacidad para registrar variaciones de intensidad vehicular, secuencias de pulsos y comportamientos anómalos del dispositivo. Por otro lado, funcionan como puntos de control remoto capaces de recibir instrucciones desde la plataforma central. El backend permite el envío de comandos específicos, tales como la solicitud de un reinicio, la consulta de parámetros internos o la activación de funciones diagnósticas. De esta manera, el operador obtiene un control directo sobre cada unidad en ruta sin necesidad de desplazarse físicamente al sitio.

Esta dualidad fortalece la flexibilidad de la plataforma y la convierte en una herramienta adaptable a diversas políticas de gestión vial. El sistema admite cambios en las reglas de operación, ampliación de funcionalidad o incorporación de

nuevas métricas sin reemplazar el hardware instalado. Además, el diseño modular facilita la adopción de tecnologías futuras (sensores adicionales, nuevos protocolos de comunicación o módulos de análisis predictivo) sin comprometer la estabilidad del sistema base.

La integración con la infraestructura existente demuestra que el proyecto respeta las limitaciones del equipamiento ya desplegado, ofrece mejoras operativas sin requerir intervenciones invasivas y habilita una expansión progresiva hacia sistemas de monitoreo más completos. La compatibilidad con los dispositivos actuales y la capacidad de adaptación a futuras necesidades posicionan al sistema como una solución sostenible y alineada con los requerimientos reales del sector vial.

Capítulo 4

Ensayos y resultados

En este capítulo se presentan en detalle los ensayos realizados sobre el sistema desarrollado, con el propósito de validar su funcionamiento en condiciones representativas de uso real. Los ensayos se organizaron en diferentes niveles: banco de pruebas en laboratorio, validación de la API REST, pruebas unitarias e integración de componentes, pruebas del frontend, prueba final de integración end-to-end y una comparación con soluciones comerciales y académicas.

4.1. Banco de pruebas

El banco de pruebas se diseñó con el propósito de reproducir las condiciones reales de operación del sistema de detección de tránsito. De este modo, se garantizó la validez de los resultados dentro de un entorno controlado. El montaje permitió evaluar la robustez del firmware, la estabilidad de las comunicaciones y la capacidad del backend para procesar eventos en distintos escenarios de conectividad. El objetivo principal consistió en analizar el comportamiento integral del sistema ante situaciones representativas de campo, que incluyeron la pérdida temporal del enlace GPRS, el almacenamiento local de eventos y la recuperación automática una vez restablecida la conexión.

4.1.1. Diseño del entorno de pruebas

El banco se compuso de los siguientes elementos principales:

- Contador de tránsito DTEC: configurado para generar tramas de detección simuladas con distintos intervalos de paso vehicular.
- Nodo de campo (ESP32-C3 + SIM800L): encargado de recibir las tramas RS-232, almacenarlas temporalmente y transmitirlas mediante MQTT al servidor central.
- Servidor de backend: implementado en Node.js/Express, con base de datos MySQL y broker Eclipse Mosquitto, desplegado mediante Docker Compose.
- Interfaz web de monitoreo: utilizada para visualizar en tiempo real los eventos recibidos y el estado de los dispositivos.

La figura 4.1 se muestra el banco de pruebas utilizado.

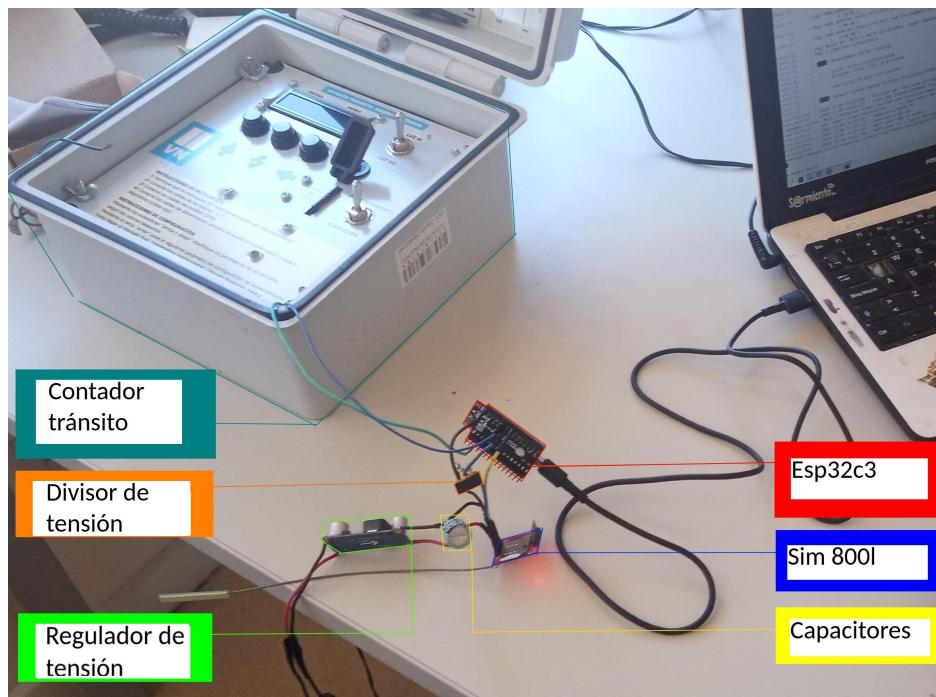


FIGURA 4.1. Fotografía del banco de pruebas.

El montaje permitió reproducir tres escenarios de prueba diferenciados:

1. Conectividad estable: transmisión continua sin pérdidas de enlace.
2. Conectividad intermitente: cortes GPRS aleatorios con verificación de la persistencia de los datos en la cola interna del nodo.
3. Modo desconectado prolongado: interrupción total de red durante intervalos extensos, lo que permitió evaluar la capacidad del firmware para conservar eventos en memoria y transmitirlos al restablecer la conexión.

4.1.2. Metodología experimental

Las pruebas se realizaron mediante la generación de tramas seriales controladas que representaban detecciones vehiculares. Se empleó un módulo de simulación que envió secuencias de tramas RS-232 al ESP32-C3. Durante cada ensayo se registraron los tiempos de procesamiento y la cantidad de eventos almacenados en la cola FIFO.

Para simular la pérdida de conectividad, se interrumpió manualmente el enlace GPRS del módulo SIM800L. Se verificó que los mensajes no enviados quedaran en cola local y que, una vez restablecida la conexión, los eventos se publicaran correctamente en los tópicos MQTT correspondientes:

- dispositivo/{id}/medicion
- dispositivo/{id}/respuesta

El backend registró la llegada de los eventos en la base de datos MySQL y comprobó su integridad, marcas de tiempo y ausencia de duplicaciones.

4.1.3. Resultados y observaciones

Los resultados experimentales demostraron que el sistema fue capaz de:

- Mantener la integridad de los datos en escenarios de conectividad inestable.
- Asegurar la entrega de eventos por medio de la cola FIFO implementada en el firmware.
- Ejecutar comandos remotos y recibir respuestas de forma confiable.
- Reanudar la transmisión después de cortes de red sin pérdida de información.

Se presentan a continuación ejemplos representativos de los resultados obtenidos durante la prueba. En la figura 4.2 se muestra la tabla `Medicion` en phpMyAdmin [39], con los registros almacenados en la base de datos MySQL. Cada uno es una detección vehicular procesada por el sistema, con su fecha, valor, carril y clasificación. Esto evidencia que el backend recibió y registró correctamente las mediciones enviadas por el nodo, incluso tras interrupciones de red.

medicionId	fecha	valor	carril	clasificacionId	dispositivoId	1	createdAt
328896	2025-10-15 17:00:00	33	2	1		1	2025-11-06 12:01:54
328897	2025-10-15 17:05:00	14	1	2		1	2025-11-06 12:02:11
328898	2025-10-15 17:10:00	38	2	2		1	2025-11-06 12:03:00
328899	2025-10-15 17:15:00	31	1	1		1	2025-11-06 12:03:51
328900	2025-10-15 17:15:00	31	1	1		1	2025-11-06 12:03:57
328901	2025-10-15 17:20:00	24	1	2		1	2025-11-06 12:05:48
328902	2025-10-15 17:25:00	17	2	2		1	2025-11-06 12:05:58
328903	2025-10-15 17:30:00	39	2	1		1	2025-11-06 12:06:21
329008	2025-10-15 17:45:00	25	2	2		1	2025-11-07 15:06:11

FIGURA 4.2. Registros de detecciones vehiculares almacenados en la base de datos MySQL.

En la figura 4.3 se muestra un intercambio entre el servidor y el nodo. El backend envía un comando al dispositivo en `dispositivo/id/comando` y el nodo responde en `dispositivo/id/respuesta` con el cmdId y el valor asociado, validando la comunicación bidireccional.

SELECT * FROM `Comando`
<input type="checkbox"/> Perfilando [Editar en línea] [Editar] [Explicar SQL] [Crear código PHP] [Actualizar]
cmdId fecha tipoComandoId valor dispositivoId createdAt
16 2025-11-06 12:03:50 1 resetear 1 2025-11-06 12:03:50
SELECT * FROM `Respuesta`
<input type="checkbox"/> Perfilando [Editar en línea] [Editar] [Explicar SQL] [Crear código PHP] [Actualizar]
respId fecha cmdId valor dispositivoId createdAt
7 2025-11-06 12:03:50 16 OK 1 2025-11-06 12:06:09

FIGURA 4.3. Intercambio de mensajes entre el backend y el nodo de campo mediante MQTT.

4.2. Pruebas de la API REST

Esta sección presenta las pruebas realizadas sobre la API REST implementada en el backend del sistema. El propósito fue validar la interacción entre los componentes principales (backend, base de datos y broker MQTT) y comprobar la integridad, la seguridad y el rendimiento de las operaciones ofrecidas por los endpoints.

4.2.1. Objetivos y alcance

Los objetivos específicos que guiaron la planificación de las pruebas fueron los siguientes:

- Verificar la implementación correcta de los endpoints asociados a dispositivos, mediciones, comandos, respuestas y usuarios.
- Confirmar la persistencia y consistencia de los datos en la base de datos MySQL.
- Validar el esquema de autenticación y autorización mediante tokens JWT.
- Evaluar la integración con el broker MQTT para la publicación y recepción de mensajes.
- Medir el tiempo de respuesta y la estabilidad del servicio en diferentes condiciones de red y carga.
- Comprobar el manejo de errores y la coherencia de las respuestas ante solicitudes inválidas.

El alcance incluyó operaciones sincrónicas (consultas, altas, modificaciones y eliminaciones) y asíncronas (envío y recepción de comandos MQTT) con el fin de cubrir todos los flujos funcionales.

4.2.2. Metodología de prueba

El proceso de validación se realizó con la herramienta Postman [40]. Se elaboraron colecciones de solicitudes y scripts de prueba en la pestaña Tests, que verificaron los códigos de estado HTTP, la estructura de las respuestas y el contenido de los mensajes.

El Collection Runner [41] permitió ejecutar los casos de prueba en distintos entornos: desarrollo local, red simulada GPRS e integración con el broker MQTT. Los resultados se exportaron en formato JSON y se analizaron mediante la extensión Newman [42].

El middleware Morgan registró las solicitudes HTTP, mientras que el sistema de logging Winston almacenó eventos críticos del backend, como errores de conexión, tiempos de procesamiento y publicaciones MQTT. La trazabilidad obtenida permitió optimizar parámetros como la concurrencia de conexiones MySQL y la retención de mensajes MQTT.

Para evaluar la tolerancia a fallos, se interrumpieron deliberadamente las conexiones del broker MQTT y del enlace GPRS. Los mensajes en cola se reenviaron al restablecer la red sin generar duplicaciones ni pérdidas.

4.2.3. Resultados obtenidos

Las pruebas confirmaron la estabilidad y solidez de la API REST. Todas las operaciones CRUD se ejecutaron correctamente y devolvieron respuestas en formato JSON con los códigos HTTP correspondientes, lo que permite destacar lo siguiente:

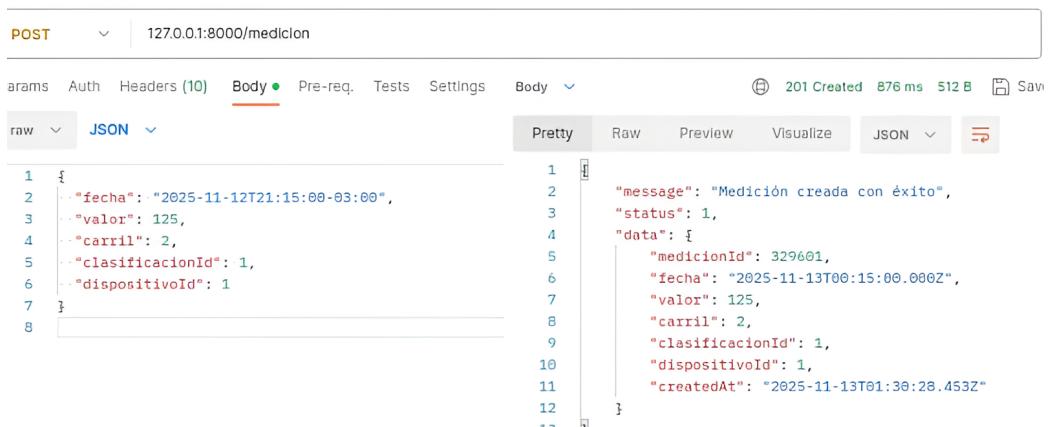
- Autenticación: las solicitudes sin token o con credenciales inválidas fueron rechazadas con los códigos 401 y 403.
- Integración MQTT: los comandos se publicaron en los tópicos dispositivo/{id}/comando, y las respuestas se recibieron en dispositivo/{id}/respuesta, que actualizó los estados en la base de datos.
- Persistencia: no se registraron pérdidas ni duplicaciones de datos en la base de datos MySQL.
- Rendimiento: el tiempo de respuesta promedio fue de 210 ms en entorno local y de 550 ms bajo simulación GPRS, con un máximo de 1,2 s en carga alta.
- Manejo de errores: los mensajes de error fueron claros y usaron códigos estandarizados (400, 404, 423, 500).

A continuación, se presentan las pruebas realizadas en el backend, destinadas a verificar el correcto funcionamiento de los servicios implementados.

Medición

La entidad Medicion constituye el núcleo del sistema, ya que representa los datos enviados por el firmware al backend. A continuación, se muestran las pruebas realizadas sobre el endpoint POST /api/medicion, donde se validó la correcta recepción y verificación de los campos obligatorios.

En la figura 4.4 se observa el envío correcto de los datos requeridos: fecha, valor, carril, clasificacionId y dispositivoId. El backend almacena la medición y retorna una confirmación en formato JSON.



```

POST      | 127.0.0.1:8000/medicion
          | Headers (10) Body (1) Pre-req. Tests Settings Body (1)
          | raw JSON
          | {
          |   "fecha": "2025-11-12T21:15:00-03:00",
          |   "valor": 125,
          |   "carril": 2,
          |   "clasificacionId": 1,
          |   "dispositivoId": 1
          | }
          |
          | 201 Created 876 ms 512 B Save
          | Pretty Raw Preview Visualize JSON
          | {
          |   "message": "Medición creada con éxito",
          |   "status": 1,
          |   "data": {
          |     "medicionId": 329601,
          |     "fecha": "2025-11-13T00:15:00.000Z",
          |     "valor": 125,
          |     "carril": 2,
          |     "clasificacionId": 1,
          |     "dispositivoId": 1,
          |     "createdAt": "2025-11-13T01:30:28.453Z"
          |   }
          | }
  
```

FIGURA 4.4. Solicitud POST /api/medicion con todos los campos completos.

En la figura 4.5 se muestra la respuesta ante una solicitud incompleta, donde falta al menos uno de los campos obligatorios. El sistema devuelve un mensaje en formato JSON indicando la causa del error.

```

POST 127.0.0.1:8000/medicion
Params Auth Headers (10) Body Pre-req. Tests Settings
Body
Pretty Raw Preview Visualize JSON
1
2 "valor": 125,
3 "carril": 2,
4 "claseficacionId": 1,
5 "dispositivoId": 1
6
7
1
2 "message": "fecha, valor, carril, claseficacionId y dispositivoId son obligatorios",
3
4 "status": 0

```

FIGURA 4.5. Solicitud POST /api/medicion con campos faltantes.

Comando

La entidad Comando permite enviar instrucciones remotas a los nodos de campo a través del broker MQTT. Las pruebas se realizaron sobre el endpoint POST /api/comando, que verifica tanto la creación exitosa como la validación de datos.

La figura 4.6 muestra una solicitud válida con los campos fecha, valor y dispositivoId, que genera un nuevo comando y lo envía al dispositivo indicado.

```

POST 127.0.0.1:8000/comando
Params Auth Headers (10) Body Pre-req. Tests Settings
Body
Pretty Raw Preview Visualize JSON
1
2 {
3   "fecha": "2025-11-12T21:15:00-03:00",
4   "tipoComandId": 1,
5   "valor": "reset",
6   "dispositivoId": 1
7 }
1
2 {
3   "message": "Comando creado con éxito y publicado.",
4   "status": 1,
5   "data": {
6     "cmdId": 29,
7     "fecha": "2025-11-13T00:15:00.000Z",
8     "tipoComandId": 1,
9     "valor": "reset",
10    "dispositivoId": 1,
11    "createdAt": "2025-11-13T02:39:50.665Z"
12  }

```

FIGURA 4.6. Solicitud POST /api/comando exitosa.

En la figura 4.7 se observa una solicitud con parámetros faltantes.

```

Auth Headers (10) Body Pre-req. Tests Settings
Body
Pretty Raw Preview Visualize JSON
1
2 {
3   "fecha": "2025-11-12T21:15:00-03:00",
4   "cmdId": 29,
5 }
1
2 "message": "Faltan datos obligatorios",
3
4 "status": 0

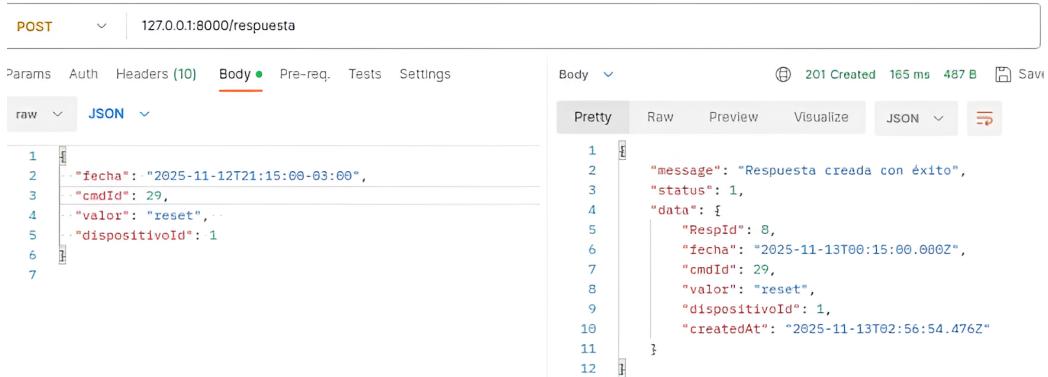
```

FIGURA 4.7. Solicitud POST /api/comando con error por datos incompletos.

Respuesta

La entidad Respuesta almacena los mensajes enviados por los nodos de campo en respuesta a los comandos recibidos. Se evaluó el endpoint POST /api/respuesta, y se comprobó la asociación correcta con el comando original y el dispositivo.

La figura 4.8 muestra un ejemplo de respuesta registrada exitosamente, con los campos fecha, cmdId, valor y dispositivoId.



The screenshot shows a POST request to the endpoint `127.0.0.1:8000/respuesta`. The request body is JSON-formatted:

```

1  {
2    "fecha": "2025-11-12T21:15:00-03:00",
3    "cmdId": 29,
4    "valor": "reset",
5    "dispositivoId": 1
6  }

```

The response status is 201 Created, with a response time of 165 ms and a size of 487 B. The response body is also JSON:

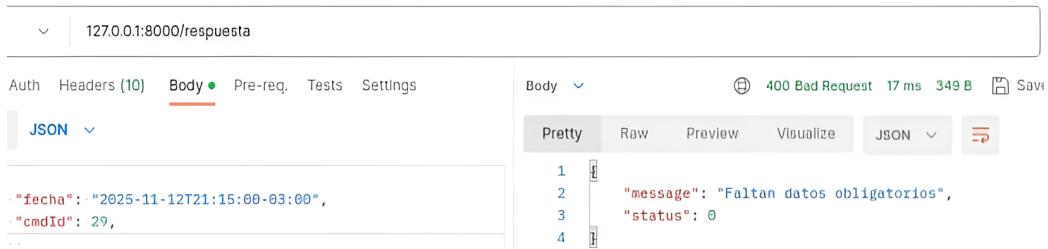
```

1  {
2    "message": "Respuesta creada con éxito",
3    "status": 1,
4    "data": {
5      "RespId": 8,
6      "fecha": "2025-11-13T00:15:00.000Z",
7      "cmdId": 29,
8      "valor": "reset",
9      "dispositivoId": 1,
10     "createdAt": "2025-11-13T02:56:54.476Z"
11   }
12 }

```

FIGURA 4.8. Solicitud POST /api/respuesta exitosa.

En la figura 4.9 se observa la respuesta generada cuando alguno de los campos requeridos no fue proporcionado.



The screenshot shows a POST request to the endpoint `127.0.0.1:8000/respuesta`. The request body is JSON-formatted:

```

1  {
2    "fecha": "2025-11-12T21:15:00-03:00",
3    "cmdId": 29,
4    ...
}

```

The response status is 400 Bad Request, with a response time of 17 ms and a size of 349 B. The response body is JSON:

```

1  {
2    "message": "Faltan datos obligatorios",
3    "status": 0
4  }

```

FIGURA 4.9. Solicitud POST /api/respuesta con error de validación.

A continuación, se presenta la tabla 4.1 con los resultados de las pruebas de los endpoints REST, donde se registraron las respuestas del sistema, los códigos HTTP obtenidos y el tiempo medio de procesamiento para cada operación:

TABLA 4.1. Resultados de las pruebas realizadas sobre los principales endpoints de la API REST mediante Postman.

Endpoint	Tipo	Resultado	Código HTTP	Tiempo medio (ms)
GET /dispositivo	GET	Consulta correcta de todos los dispositivos.	200	215
GET /dispositivo/{id}	GET	Recuperación exitosa de un dispositivo específico.	200	225
POST /dispositivo	POST	Alta de nuevo dispositivo	201	245
GET /medicion/dispositivo/{id}	GET	Consulta de mediciones por dispositivo.	200	230
POST /comando	POST	Publicación de comando en MQTT.	201	310
GET /comando/{id}	GET	Consulta de estado de comando.	200	520
POST /respuesta	POST	Registro de respuesta.	201	245
GET /respuesta/{id_com}	GET	Recuperación de respuesta asociada.	200	225
POST /usuario/login	POST	Autenticación válida (JWT).	200	180
GET /usuario	GET	Acceso restringido (JWT).	403	190

Los ensayos confirmaron que la API REST cumple los criterios de fiabilidad, seguridad y desempeño definidos en el diseño. El uso de colecciones automatizadas permitió repetir las pruebas en distintos entornos y documentar los resultados con precisión.

4.3. Pruebas de componentes

Las pruebas de componentes tuvieron como propósito verificar la integración entre los módulos del sistema (firmware, backend, broker MQTT, base de datos y frontend) y asegurar el correcto comportamiento de manera individual y conjunta.

A diferencia del banco de pruebas y de la validación de la API REST, esta etapa se centró en la integridad del flujo de datos completo, el manejo de errores y la coherencia operativa ante fallas o sobrecarga.

4.3.1. Enfoque general

El sistema se evaluó bajo un esquema progresivo:

1. Pruebas unitarias: destinadas a validar la funcionalidad de cada componente de software.
2. Pruebas de integración: diseñadas para verificar la comunicación entre módulos y la consistencia de los datos.
3. Pruebas de tolerancia a fallos: enfocadas en la recuperación automática ante desconexiones, errores de red o reinicios.

El entorno completo se desplegó en contenedores Docker independientes, lo que permitió reproducir escenarios de prueba con precisión y medir el impacto de fallas.

4.3.2. Resultados por componente

Cada módulo del sistema fue evaluado de forma independiente para verificar su funcionamiento, la integridad de los datos y la robustez ante fallos de conexión. Este proceso permitió analizar el comportamiento de cada componente en situaciones reales de operación, asegurando una arquitectura estable y confiable en todas las etapas del flujo de información. Se resumen los principales resultados obtenidos en las pruebas de cada componente:

- Firmware (ESP32-C3): se validó el análisis correcto de tramas RS-232, el almacenamiento temporal en colas FIFO y la publicación confiable de mensajes MQTT hacia el broker. Estas pruebas confirmaron que el dispositivo interpreta adecuadamente los datos recibidos y mantiene la consistencia de los paquetes aun en escenarios de mayor carga de transmisión.
- Broker MQTT: se ejecutaron desconexiones simuladas para medir la tolerancia a fallos. El sistema mantuvo la sesión activa y transmitió los mensajes pendientes una vez restablecida la conexión, lo que aseguró continuidad del servicio sin pérdida de información.
- Backend: se comprobó la correcta gestión de solicitudes REST, el procesamiento adecuado de cada operación y la sincronización efectiva con el broker MQTT para el envío y recepción de comandos y respuestas.
- Frontend: se verificó la comunicación bidireccional con la API REST, la presentación inmediata de la información obtenida y la visualización en el panel del estado actualizado de dispositivos y mediciones.
- Manejo de errores: los registros obtenidos mediante Winston y Morgan reflejaron reconexiones exitosas ante cortes de red y operaciones sin pérdida de datos, lo que confirmó un sistema con mecanismos sólidos de recuperación frente a situaciones adversas.

En la figura 4.10 se puede observar el intercambio de mensajes entre el nodo de campo y el servidor. En la parte superior de la imagen se muestra la consola del ESP32-C3, donde el firmware registra el envío de una medición a través del módulo SIM800L utilizando el protocolo MQTT. En la parte inferior, se visualiza la consola del backend desarrollada en Node.js, que recibe y procesa el mensaje, verificando los datos de la medición antes de almacenarlos en la base de datos.

The screenshot displays two terminal windows. The top window, titled 'Log módulo ESP32C3', shows the ESP32-C3 sending an MQTT publish message to the topic '/dispositivo/1/medicion'. The message payload is a JSON object containing 'dispositivoId': 1, 'valor': 33, 'carril': 2, 'clasificacionId': 1, and 'fecha': '2025-10-15T17:00:00-03:00'. A red box highlights the 'Publicación en el tópico de medición' (Publication in the measurement topic) part of the log. The bottom window, titled 'Log backend', shows the Node.js application receiving this message and logging it. A red box highlights the 'Recepción en el tópico de medición' (Receipt in the measurement topic) part of the log.

```

1: diego@bangho-vialidad:~/proyectos/_viejo/cr_c Log módulo ESP32C3
I (34806) GPRS: 0x3fc95f80 2f 31 2f 6d 65 64 69 63 69 6f 6e 7b 22 64 69 73 | /1/m
edicion("dis"
I (34806) GPRS: 0x3fc95f90 70 6f 73 69 74 69 76 6f 49 64 22 3a 31 2c 22 76 | pos
tivoId":1,"v|
I (34816) GPRS: 0x3fc95fa0 61 6c 6f 72 22 3a 33 33 2c 22 63 61 72 72 69 6c | alor
":33,"carril|
I (34826) GPRS: 0x3fc95fb0 22 3a 32 2c 22 63 6c 61 73 69 66 69 63 61 63 69 | ":"2,
"clasificaci|
I (34836) GPRS: 0x3fc95fc0 6f 6e 49 64 22 3a 31 2c 22 66 65 63 68 61 22 3a | onId
":1,"fecha":|
I (34846) GPRS: 0x3fc95fd0 22 32 30 32 35 2d 31 30 2d 31 35 54 31 37 3a 30 | "202
5-10-15T17:0|
I (34856) GPRS: 0x3fc95fe0 30 3a 30 30 2d 30 33 3a 30 30 22 7d | 0:00
-03:00"}|
I (36536) GPRS: CIPSEND OK:
SEND OK
Publicación en el
tópico de medición

I (36536) GPRS: PUBLISH enviado: /dispositivo/1/medicion -> {"dispositivoId":1,"valo
r":33,"carril":2,"clasificacionId":1,"fecha":"2025-10-15T17:00:00-03:00"}
```



```

2: diego@bangho-vialidad:~/proyectos/flow_track/app_ft Log backend
node-backend | [2025-11-06 12:01:37] : Trato de obtener ultima medicion para este d
ispositivoId: 1
node-backend | [2025-11-06 12:01:37] : Trato de obtener ultimo comando para este di
spositivoId: 1
node-backend | [2025-11-06 12:01:52] : Token en chequeo: eyJhbGciOiJIUzI1NiIsInR5cC
I6IKpxVCJ9.eyJlc2VyRm9ibmQioNsibmFtZSI6ImFkbWluIiwicGFzc3dvcmQiOiIyMTIzMmYyOTdhNTdhN
WE3NDM40TRhMGU0YTgwMWZjMyIsImRlc2NyAXAi0m51bGx9LCJpYXQi0jE3NjI0NDEyNDJ9.cG_TUY9hl4Cz
FHL8d0YfiwAPww9caf_s6gVo6yRYuNU
node-backend | [2025-11-06 12:01:52] : Trato de obtener ult
Recepción en el
tópico de medición
ste d
ispositivoId: 1
node-backend | [2025-11-06 12:01:52] : Trato de obtener ultimo comando para este di
spositivoId: 1
node-backend | [2025-11-06 12:01:54] : MQTT mensaje recibido: /dispositivo/1/medici
on
node-backend | [2025-11-06 12:01:54] : MQTT Ingreso Medicion : dispositivoId: 1, fe
cha: 2025-10-15T17:00:00-03:00, valor: 33, carril: 2, clasificacionId: 1

```

FIGURA 4.10. Interacción entre el nodo de campo (ESP32-C3 + SIM800L) y el backend.

Las pruebas confirmaron la cohesión del sistema y su capacidad de recuperación ante fallas. El uso de contenedores Docker facilitó la integración y la detección de incompatibilidades. Los resultados validaron la solidez de la arquitectura distribuida y su adecuación a entornos con conectividad limitada.

4.4. Pruebas del frontend

Las pruebas del frontend tuvieron como finalidad evaluar el correcto funcionamiento de la interfaz web desarrollada, garantizar su compatibilidad, usabilidad, rendimiento y capacidad de interacción con el backend del sistema.

4.4.1. Objetivos

Los objetivos específicos de esta etapa fueron los siguientes:

- Verificar la compatibilidad del frontend con los navegadores más utilizados (Chrome, Firefox) y con dispositivos móviles Android.

- Evaluar el rendimiento general de la aplicación: tiempos de carga, latencia en consultas a la API y velocidad de actualización de los gráficos.
- Analizar la usabilidad de la interfaz mediante pruebas con usuarios: oportunidades de mejora en la disposición de elementos visuales y en los flujos de interacción.
- Validar la correcta ejecución de comandos y confirmaciones visuales en tiempo real a través de la comunicación con el broker MQTT y la API REST.
- Comprobar que la API maneja correctamente los errores de conexión y de autenticación, devolviendo mensajes claros y proporcionando retroalimentación inmediata al cliente.

4.4.2. Metodología

Las pruebas se realizaron sobre la versión estable del frontend, desarrollado con los frameworks Ionic y Angular, y desplegado en un entorno controlado junto al backend y el broker MQTT. Se observó la interacción del usuario, a fin de verificar el comportamiento integral del sistema en distintos escenarios:

- Compatibilidad y visualización: se validó la correcta visualización de los componentes en distintas resoluciones y navegadores, utilizando Chrome DevTools [43] y el modo responsivo de Ionic. El diseño adaptativo permitió mantener la legibilidad de los gráficos y menús tanto en pantallas de escritorio como en dispositivos móviles. Las pruebas demostraron una compatibilidad completa con los navegadores modernos, presentando solo ligeras diferencias en el renderizado de íconos SVG en Firefox.
- Rendimiento: el análisis de desempeño se realizó con Lighthouse [44] y el monitor de red de los navegadores. El tiempo promedio de carga inicial fue de 2,3 segundos en Chrome y 2,7 segundos en Firefox. En dispositivos móviles Android, el tiempo de carga fue de 3,8 segundos, debido a la menor capacidad de procesamiento. La latencia promedio de las consultas REST se mantuvo por debajo de los 250 milisegundos en red local, que aumentó a 600 milisegundos bajo simulación GPRS.
- Usabilidad: se realizaron pruebas con un grupo reducido de usuarios familiarizados con sistemas de monitoreo vial, quienes interactuaron con la interfaz durante sesiones controladas. Los resultados indicaron una alta comprensión del flujo de navegación y una percepción positiva de la organización visual. Las observaciones fueron incorporadas en una versión posterior mediante ajustes de color, iconografía y jerarquía visual.
- Comunicación y validación funcional: se comprobó que los comandos emitidos desde la interfaz se transmitieran correctamente al backend y se visualizaran sus estados en tiempo real. Del mismo modo, los eventos de tránsito publicados por los dispositivos se reflejaron de forma inmediata en la aplicación, sin inconsistencias ni retrasos notables. Las alertas visuales ante pérdida de conexión, errores de autenticación o respuestas HTTP inválidas funcionaron según lo esperado, mostrando mensajes informativos y acciones de recuperación.

A continuación, se presentan las distintas pantallas que conforman la aplicación:

La figura 4.11 muestra la pantalla de inicio de sesión, donde el usuario debe ingresar sus credenciales para acceder al sistema.

La imagen muestra una interfaz de usuario para iniciar sesión. El título "Login" está en la parte superior izquierda. Abajo de él, hay dos campos de texto: el primero contiene la palabra "admin" y el segundo contiene ".....". A continuación, un botón grande de color verde con el texto "INGRESAR" en blanco.

FIGURA 4.11. Pantalla de inicio de sesión del frontend.

Una vez autenticado, el usuario accede al panel principal mostrado en la figura 4.12, donde se presenta el listado de dispositivos registrados junto con la ubicación y el tipo de contador.

La imagen muestra el "Listado de Dispositivos" con cuatro entradas:

Nombre	Ubicación	Tipo	Opción
contador 1	Padre Buodo	DTEC	VER >
contador 2	Tres Arroyos	PADR	VER >
contador 3	Gorostiaga	DTEC	VER >
contador 4	Petion	DTEC_GRD	VER >

FIGURA 4.12. Panel principal con visualización del listado de dispositivos.

Al seleccionar un dispositivo del listado, el sistema despliega un panel detallado con la información específica de dicho equipo. La figura 4.13 ilustra la primera vista del panel de dispositivo, donde se presenta la información general de identificación del equipo (nombre, ubicación y tipo), el último comando enviado y su correspondiente respuesta, la última medición registrada y un botón que permite acceder a la visualización de las mediciones históricas. Esta interfaz facilita el análisis del estado del dispositivo y el acceso rápido a los datos relevantes sin necesidad de recorrer menús adicionales.

Dispositivo 1

Nombre: contador 1
Ubicación: Padre Buodo
Tipo contador: DTEC

Último Comando

Tipo de comando: Val bateria
Valor: hay algun valor?
Fecha: 2025-11-04 17:55:59
Respuesta asociada:
Valor: OK
Fecha: 2025-11-04 17:55:59

Crear Comando

Tipo de Comando
Seleccionar tipo de comando ▾

Valor del Comando

Última Medición

Fecha: 2025-10-15 17:15:00
Carril: 2
Clasificación: pesado
Valor: 40

Detalles dispositivo (highlighted with a red box)

Último comando y respuesta asociada (highlighted with a blue box)

Creción nuevo comando (highlighted with an orange box)

Última medición registrada y botón de mediciones históricas (highlighted with a blue box)

CREAR COMANDO

VER MEDICIONES

FIGURA 4.13. Panel de visualización del dispositivo.

En la figura 4.14 se muestra la creación de un comando asociado a un tipo de comando específico del contador. Al seleccionarse el tipo, el sistema habilita la opción para generar el nuevo comando.

Respuesta asociada:
Valor: OK
Fecha: 2025-11-04 17:55:59

Crear Comando

Tipo de Comando
Val bateria

Valor del Comando

CREAR COMANDO

DETALLE

A modal dialog box is displayed, showing two radio button options: "Reset" and "Val bateria". The "Val bateria" option is selected. Below the buttons are "CANCEL" and "OK" buttons, with "Val bateria" highlighted under the "OK" button.

FIGURA 4.14. Panel de visualización del dispositivo: preparación para la ejecución de un comando y su ejecución.

En la figura 4.15 se muestra la habilitación y ejecución del comando. Además, es posible ingresar un valor asociado junto con la selección del tipo de comando.

Crear Comando



Este formulario para crear un comando tiene los siguientes campos:

- Tipo de Comando:** Reset
- Valor del Comando:** resetear
- Botón:** CREAR COMANDO (en azul)

FIGURA 4.15. Panel de visualización del dispositivo: ejecución de un comando.

La figura 4.16 muestra el comando ejecutado con sus detalles y la respuesta en estado pendiente.

Último Comando

Tipo de comando: Reset
Valor: resetear
Fecha: 2025-11-05 16:40:21
No hay respuesta asociada al último comando.

FIGURA 4.16. Panel de visualización del dispositivo: comando ejecutado y la respuesta pendiente.

La figura 4.17 se muestra la respuesta al comando, con los detalles que devuelve esta.

Último Comando

Tipo de comando: Reset
Valor: resetear
Fecha: 2025-11-05 16:40:21
Respuesta asociada:

Valor: OK
Fecha: 2025-11-05 16:40:21

FIGURA 4.17. Panel de visualización del dispositivo: comando ejecutado y su respuesta.

Luego, al presionar el botón **Mediciones**, se mostrará el historial correspondiente, como se observa en la figura 4.18. En esta vista se indica, para cada registro, el carril, la clasificación y el valor del volumen de los vehículos.



FIGURA 4.18. Panel de visualización del historial de mediciones.

4.4.3. Resultados y observaciones

Los resultados globales de las pruebas de frontend se resumen en los siguientes puntos:

- Compatibilidad completa con navegadores Chrome y Firefox, y compatibilidad en móviles Android.
- Tiempos de carga promedio inferiores a 3 s en escritorio y 4 s en dispositivos móviles.
- Comunicación estable con la API REST y el broker MQTT, incluso ante re-conexiones de red.
- Interfaz intuitiva y valorada positivamente por los usuarios de prueba, con mejoras implementadas en la versión final.

Las pruebas permitieron validar la madurez funcional de la interfaz web y su adecuación a las necesidades operativas de los usuarios finales.

4.5. Prueba final de integración

La prueba final de integración tuvo como objetivo validar el flujo completo del sistema bajo condiciones de operación equivalentes a las de un entorno real. Esta etapa permitió comprobar la interoperabilidad entre todos los componentes involucrados: el nodo de campo, el firmware embebido, el módulo de comunicación GPRS, el broker MQTT, la API REST, la base de datos y la interfaz web.

El ensayo buscó garantizar que el sistema, en su conjunto, cumpliera con los requisitos de confiabilidad, sincronización y trazabilidad definidos en las fases de diseño y desarrollo.

4.5.1. Metodología de la prueba

Para la validación end-to-end se habilitó un entorno de prueba con todos los subsistemas en operación simultánea. El flujo general fue el siguiente:

1. El contador DTEC emitió una trama RS-232 recibida por el nodo ESP32-C3.
2. El firmware procesó la trama, generó el evento y lo publicó por MQTT en dispositivo/{id}/medicion.
3. El backend validó el mensaje, lo almacenó en la base de datos y notificó a la interfaz web mediante su API REST.
4. El frontend consultó la API REST y actualizó la visualización con la nueva medición.
5. Desde la interfaz se envió un comando de prueba que el backend publicó en dispositivo/{id}/comando.
6. El firmware recibió el comando, ejecutó la acción, respondió en dispositivo/{id}/respuesta y se registró en el backend.

En la figura 4.19 se muestra el proceso completo de medición: en la parte superior aparece la publicación del mensaje MQTT en dispositivo/1/medicion y, en la inferior, el registro del backend con su recepción, validación y almacenamiento.

The figure consists of two terminal windows. The top window is titled 'Log ESP32C3' and shows the following output:

```

1: diego@bangho-vialidad:~/proyectos/_vieja/cr_c Log ESP32C3
I (100776) GPRS: 0x3fc95f80 2f 31 2f 6d 65 64 69 63 69 6f 6e 7b 22 64 69 73 | /1/
medicion{"dis|
I (100776) GPRS: 0x3fc95f90 70 6f 73 69 74 69 76 6f 49 64 22 3a 31 2c 22 76 | pos
positivoId":1,"v|
I (100786) GPRS: 0x3fc95fa0 61 6c 6f 72 22 3a 33 38 2c 22 63 61 72 72 69 6c | alo
r":38,"carril|
I (100796) GPRS: 0x3fc95fb0 22 3a 32 2c 22 63 6c 61 73 69 66 69 63 61 63 69 | ":"2
,"clasificaci|
I (100806) GPRS: 0x3fc95fc0 6f 6e 49 64 22 3a 32 2c 22 66 65 63 68 61 22 3a | onl
d":2,"fecha":|
I (100816) GPRS: 0x3fc95fd0 22 32 30 32 35 2d 31 30 2d 31 35 54 31 37 3a 31 | "20
25-10-15T17:1|
I (100826) GPRS: 0x3fc95fe0 30 3a 30 30 2d 30 33 3a 30 30 22 7d | 0:0
0-03:00"}|
I (102476) GPRS: CIPSEND OK:
SEND OK

```

A red box highlights the line 'Publicación medición'.

The bottom window is titled 'LLog backend' and shows the following output:

```

2: diego@bangho-vialidad:~/proyectos/flow_track/app_ft LLog backend
node-backend | [2025-11-06 12:02:52] : Trato de obtener mediciones para este dispositivoId: 1
node-backend | [2025-11-06 12:02:57] : Buscando mediciones del dispositivoId: 1 (limit 100, offset 0, desde -, hasta -)
node-backend | [2025-11-06 12:03:00] : MQTT mensaje recibido: /dispositivo/1/medicion
node-backend | [2025-11-06 12:03:00] : MQTT Ingreso Medicion : dispositivoId: 1, fecha: 2025-10-15T17:10:00-03:00, valor: 38, carril: 2, clasificacionId: 2
node-backend | [2025-11-06 12:03:00] : MQTT medición guardada para dispositivo 1
node-backend | [2025-11-06 12:03:07] : Token en chequeo: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlc2VyRm91bmQiOnsibmFtZSI6ImFkbWluIiwicGFzc3dvcmQiOiIyMTIzMmY0TdhNTdhNWE3NDM4OTRhMGU0YTgwMWZjMyIsImRlc2NyaXAiO51bGx9LCJpYXQ10jE3NjI0NDEyNDJ9.cG_TUY9hl4CzEHL8dQYfiwAPww9caf_s6gVn6vRYuNUI

```

A red box highlights the line 'Recepción medición'.

FIGURA 4.19. Publicación de una medición por parte del módulo ESP32-C3 y recepción en el backend.

En la figura 4.20 se visualiza la interfaz web luego de recibir la medición publicada.



FIGURA 4.20. Visualización de la medición recibida en la interfaz web del sistema.

Después se validó la comunicación descendente del sistema, en la cual la interacción se inicia desde la interfaz web. El objetivo fue comprobar que los comandos generados por el usuario fueran correctamente transmitidos al backend, publicados en el tópico MQTT correspondiente y finalmente recibidos y ejecutados por el nodo de campo ESP32-C3.

En la figura 4.21 se muestra la interfaz del sistema desde la cual se ha emitido un comando hacia el dispositivo. El usuario seleccionó el tipo de comando a enviar y su valor (si es necesario) y el backend recibió esta solicitud mediante la API REST y la publica en el tópico dispositivo/1/comando, que queda a la espera de la respuesta del dispositivo.

The figure shows a screenshot of a web-based command generation interface. It has two main sections: 'Último Comando' and 'Crear Comando'.

- Último Comando:**
 - Tipo de comando: Reset
 - Valor: resetear
 - Fecha: 2025-11-06 12:03:50
 - No hay respuesta asociada al último comando.
- Crear Comando:**

Tipo de Comando	Seleccionar tipo de comando ▾
Valor del Comando	
CREAR COMANDO	

FIGURA 4.21. Generación de un comando desde la interfaz web.

En la figura 4.22 se muestra parte de la traza del proceso de ejecución del comando, donde la consola del módulo ESP32-C3 registra la recepción del comando.

```

1: diego@bangho-vialidad:~/proyectos/_vieja/cr_c - Log ESP32C3
I (246946) GPRS: 0x3fc96150 20 02 00 00
.
.
I (246946) GPRS: CONNACK recibido, MQTT conectado
I (246946) GPRS: MQTT conectado
I (247346) GPRS: Comando enviado: AT+CIPSEND=29

I (247476) GPRS: 0x3fc95f90 82 1b 00 01 00 16 2f 64 69 73 70 6f 73 69 74 69 | ...
.../dispositivo|
I (247476) GPRS: 0x3fc95fa0 76 6f 2f 31 2f 63 6f 6d 61 6e 64 6f 00 | vo/
1/comando.| 
I (248506) GPRS: CIPSEND OK:
SEND OK
❖
I (248506) GPRS: SUBSCRIBE enviado: /dispositivo/1/comando
I (252406) APP_ESP32C3: PUBLISH recibido. Retained: Si
I (252406) APP_ESP32C3: Tópico: /dispositivo/1/comando
I (252406) APP_ESP32C3: Payload: {"cmdId":16,"tipoComandId":1,"valor":"resetear","fecha":"2025-11-06T15:03:50.825Z"} 
```

FIGURA 4.22. Visualización consola del módulo ESP32-C3 que registra la recepción del comando.

En la figura 4.23 se visualiza la ejecución del flujo de respuesta desde el nodo. En la consola se observa el log del ESP32-C3, que envía la respuesta con el resultado del proceso interno del contador. El firmware publica esta información en el tópico dispositivo/1/respuesta, con los campos fecha, cmdId, valor y dispositivoId.

```

oId":1|
I (290936) GPRS: CIPSEND OK:
SEND OK

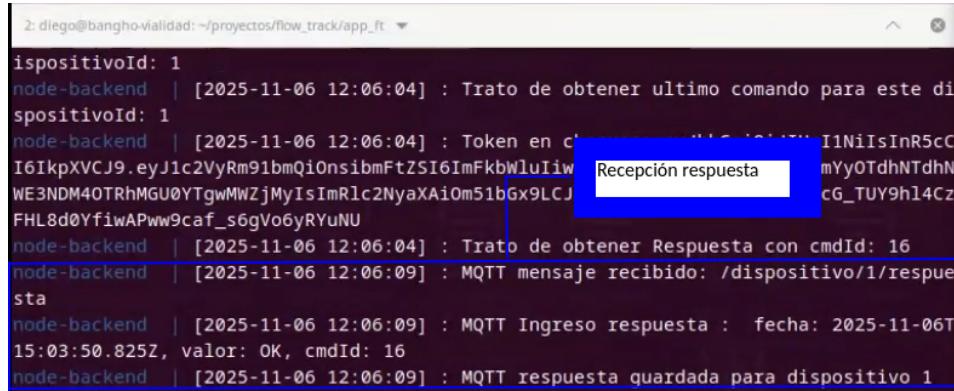
I (290936) GPRS: PUBLISH enviado: /dispositivo/1/respuesta -> {"fecha": "2025-11-06T15:03:50.825Z", "cmdId": 16, "valor": "OK", "dispositivoId": 1}
I (290936) GPRS: Comando enviado: AT+CIPSEND=26

I (291076) GPRS: 0x3fc95f70 31 18 00 16 2f 64 69 73 70 6f 73 69 74 69 76 6f | ...
./dispositivo|
I (291076) GPRS: 0x3fc95f80 2f 31 2f 63 6f 6d 61 6e 64 6f 00 | /1/
comando|
I (291936) GPRS: CIPSEND OK:
SEND OK

I (291936) GPRS: PUBLISH enviado: /dispositivo/1/comando ->
I (291996) APP_ESP32C3: PUBLISH recibido. Retained: No
I (291996) APP_ESP32C3: Tópico: /dispositivo/1/comando
I (291996) APP_ESP32C3: Payload: 
```

FIGURA 4.23. Flujo de respuesta desde el nodo.

En la figura 4.24 se muestra el registro correspondiente en el backend. El sistema recibe el mensaje desde el broker MQTT, valida su estructura y almacena los datos en la base de datos MySQL.



```

2: diego@bangho-vialidad: ~/proyectos/flow_track/app_ft ~
ispositivoId: 1
node-backend | [2025-11-06 12:06:04] : Trato de obtener ultimo comando para este dispositivoId: 1
node-backend | [2025-11-06 12:06:04] : Token en c... I1NiIsInR5cC
I6IkpxVCJ9.eyJc2VybRm91bmQiOnsibmFtZSI6ImFkbWluIiw... mYyOTdhNTdhN
WE3NDM40TRhMGU0YTgwMWZjMyIsImRlc2NyAioM51bGx9LCJ... cG_TUY9h14Cz
FHL8d0YfiwAPww9caf_s6gVo6yRYuNU
node-backend | [2025-11-06 12:06:04] : Trato de obtener Respuesta con cmdId: 16
node-backend | [2025-11-06 12:06:09] : MQTT mensaje recibido: /dispositivo/1/respuesta
node-backend | [2025-11-06 12:06:09] : MQTT Ingreso respuesta : fecha: 2025-11-06T
15:03:50.825Z, valor: OK, cmdId: 16
node-backend | [2025-11-06 12:06:09] : MQTT respuesta guardada para dispositivo 1

```

FIGURA 4.24. Flujo de recepción de la respuesta en el backend.

Finalmente, en la figura 4.25 se muestra la visualización del resultado en la interfaz web. Una vez almacenada la respuesta, la aplicación cliente consulta el endpoint correspondiente de la API REST y actualiza la vista con la información más reciente del dispositivo. De esta manera, el usuario puede verificar en tiempo real que el comando fue recibido y ejecutado correctamente, completando el ciclo de comunicación bidireccional del sistema.

Dispositivo 1

Nombre: contador 1
 Ubicación: Padre Buodo
 Tipo contador: DTEC

Último Comando

Tipo de comando: Reset
 Valor: resetear
 Fecha: 2025-11-06 12:03:50
 Respuesta asociada:
 Valor: OK
 Fecha: 2025-11-06 12:03:50

FIGURA 4.25. Visualización de la respuesta del dispositivo en la interfaz web tras el procesamiento exitoso en el backend.

4.5.2. Resultados obtenidos

Los resultados obtenidos en la prueba integral confirmaron el correcto funcionamiento de todo el sistema bajo condiciones reales de comunicación y sincronización de datos.

En particular, se observaron los siguientes aspectos destacados:

- Interoperabilidad completa: todos los componentes del sistema hardware, middleware y software interactuaron sin incompatibilidades ni pérdidas de información.
- Sincronización en tiempo real: la actualización de la interfaz web frente a la recepción de un nuevo evento.

- Confiabilidad del flujo de comandos: las órdenes enviadas desde el front-end fueron recibidas y ejecutadas correctamente por el nodo, con confirmaciones visibles en pantalla.
- Tiempos de ida y vuelta (round-trip): entre 3 y 5 segundos en condiciones normales de red GPRS, y hasta 12 segundos bajo conectividad inestable, sin pérdida de comandos ni duplicación de respuestas.
- Trazabilidad completa: cada evento quedó registrado en la base de datos con su respectivo timestamp e id de dispositivo.

La prueba end-to-end permitió validar la solidez de la arquitectura propuesta y su adecuación a escenarios reales de operación. El flujo completo, desde la generación de un evento hasta su visualización en la web y el control remoto del nodo, se ejecutó de forma estable, con tiempos de respuesta consistentes y trazabilidad total.

Estos resultados confirman que la solución es técnicamente viable para su despliegue en entornos de campo, que ofrece una integración transparente entre hardware embebido, servicios de red y aplicaciones web. Además, la arquitectura modular basada en estándares abiertos garantiza la posibilidad de futuras expansiones y adaptaciones a nuevas tipologías de dispositivos o protocolos de comunicación.

4.6. Comparación con otras soluciones

Con el fin de evaluar el desempeño y pertinencia del sistema desarrollado frente a alternativas existentes, se realizó un análisis comparativo entre la solución propuesta y sistemas comerciales y académicos de gestión de dispositivos de campo y monitoreo vehicular.

Se consideraron seis criterios principales: costo de implementación, flexibilidad tecnológica, escalabilidad, comportamiento ante conectividad intermitente, adecuación a entornos locales y disponibilidad de soporte técnico. Esto permitió situar la solución frente a plataformas consolidadas y trabajos académicos similares.

Las soluciones comerciales ofrecen plataformas robustas con soporte integral, pero presentan altos costos y baja flexibilidad para integrar hardware heterogéneo o protocolos abiertos. Las propuestas académicas son más experimentales y abiertas tecnológicamente, aunque con limitaciones de madurez, soporte y adaptación a producción.

La solución desarrollada combina bajo costo, independencia tecnológica y arquitectura modular basada en estándares abiertos (MQTT, REST, JSON), que permite rápida adaptación a entornos con conectividad variable, como rutas argentinas, sin depender de infraestructura propietaria ni servicios móviles externos.

En la tabla 4.2 se observa la comparación de la solución propuesta frente a alternativas comerciales y académicas:

TABLA 4.2. Comparación de la solución propuesta frente a alternativas comerciales y académicas.

Criterio	Propuesta	Comerciales	Académicas
Costo	Bajo (hardware económico + software abierto)	Alto (licencias y servicios)	Medio
Flexibilidad	Alta (protocolos estándar, modular)	Baja (propietaria)	Media
Escalabilidad	Alta (MQTT + API REST)	Alta	Media
Conectividad intermitente (colas FIFO, reintentos)	Totalmente soportada Parcial	Poco explotada	
Adecuación a rutas	Adaptada a entornos rurales y semiurbanos	Genérica	Variable
Soporte y documentación	Media (open source, docs técnicas)	Alta (soporte empresarial)	Baja

Finalmente, la solución desarrollada logra un equilibrio entre robustez, bajo costo y adaptabilidad, que la posiciona como alternativa viable para trabajos de monitoreo vial en entornos con infraestructura limitada. Su orientación a estándares abiertos y su independencia de plataformas propietarias aseguran sostenibilidad y facilidad de futuras ampliaciones.

Capítulo 5

Conclusiones

En este capítulo se presentan las conclusiones generales del trabajo, junto con una reflexión sobre los resultados alcanzados y las posibles líneas de desarrollo futuro.

5.1. Conclusiones generales

El sistema desarrollado permitió cumplir los objetivos propuestos, logra una solución integral para la detección y transmisión de eventos de tránsito en entornos con conectividad limitada. La arquitectura implementada demostró ser eficiente, modular y adaptable a distintos escenarios de despliegue, que mantiene la integridad de los datos y la estabilidad del flujo de comunicación entre nodos de campo, servidor y cliente web.

A continuación, se sintetizan los principales resultados y aportes del trabajo:

- Se logró diseñar e implementar un sistema distribuido basado en protocolos abiertos (MQTT, REST y JSON), capaz de operar de forma confiable ante conectividad intermitente.
- El firmware embebido en el ESP32-C3 validó su capacidad para procesar tramas RS-232, almacenar eventos temporalmente y asegurar su retransmisión una vez restablecida la conexión.
- Se desarrolló una biblioteca específica para el manejo del módulo SIM800L, que permite ejecutar comandos AT y establecer la comunicación con el broker MQTT y que garantiza la publicación y suscripción de mensajes bajo condiciones variables de red.
- El backend, desarrollado en Node.js con Express y MySQL, garantizó la persistencia de los datos, la trazabilidad de los eventos y la autenticación segura mediante tokens JWT.
- El broker MQTT (Eclipse Mosquitto) permitió la comunicación asincrónica y la publicación confiable de mensajes entre dispositivos y servidor, con soporte para retención de mensajes.
- La interfaz web, desarrollada en Ionic y Angular, proporcionó una visualización clara y funcional de los eventos de tránsito, junto con la posibilidad de emitir comandos y supervisar el estado de los nodos en tiempo real.
- Las pruebas realizadas en laboratorio demostraron la estabilidad del sistema y su capacidad de recuperación ante fallas o desconexiones.

- La arquitectura modular permitió aislar componentes, facilitar el mantenimiento y posibilitar futuras extensiones sin comprometer la estabilidad del sistema base.

El cronograma original se mantuvo en líneas generales, con ajustes menores asociados a la disponibilidad de hardware y a la calibración de los tiempos de respuesta del módulo GPRS. Los riesgos identificados durante la planificación, vinculados principalmente a la inestabilidad del enlace móvil y a la gestión de colas locales, fueron mitigados mediante estrategias de reconexión automática y verificación de integridad de datos, que demostraron ser efectivas en las pruebas finales.

5.2. Trabajo futuro

El trabajo realizado es un punto de partida para la evolución del sistema hacia una plataforma más completa, eficiente y adaptable a entornos operativos reales. Entre las líneas de continuidad propuestas se destacan las siguientes:

- Incorporar un módulo de comunicación que integre nativamente el protocolo MQTT, en reemplazo del SIM800L. Esta modificación permitiría reducir la complejidad del firmware, aumentar la confiabilidad del enlace y mejorar los tiempos de transmisión.
- Implementar mecanismos de actualización remota OTA para el firmware de los nodos de campo, con el fin de simplificar el mantenimiento y asegurar la uniformidad de versiones en despliegues múltiples.
- Incorporar un módulo de análisis histórico y visualización avanzada que permita detectar patrones de tránsito, generar reportes automáticos y apoyar la toma de decisiones operativas.
- Integrar servicios de geolocalización y mapas interactivos para representar la ubicación de los dispositivos y eventos en tiempo real.
- Evaluar el desempeño del sistema en entornos de campo prolongados, con el propósito de obtener métricas de confiabilidad, consumo energético y latencia bajo condiciones reales de operación.
- Desarrollar herramientas de monitoreo remoto y alertas automáticas, que notifiquen anomalías en los dispositivos o interrupciones de comunicación sin intervención manual.
- Incorporar técnicas de inteligencia artificial para la detección de tránsito no registrado, lo que permitiría incrementar la precisión del sistema en escenarios con oclusión parcial o interferencias.
- Migrar la comunicación móvil a redes 4G o 5G, con el fin de mejorar la disponibilidad, reducir la latencia y soportar mayor volumen de datos en despliegues masivos.

Bibliografía

- [1] Juan Asiaín et al. «LoRa-Based Traffic Flow Detection for Smart-Road». En: *Sensors* 21.2 (2021), pág. 338. DOI: [10.3390/s21020338](https://doi.org/10.3390/s21020338). URL: <https://www.mdpi.com/1424-8220/21/2/338>.
- [2] Jan Micko et al. «Review of IoT Sensor Systems Used for Monitoring the Road Infrastructure». En: *Sensors* 23.9 (2023), pág. 4469. DOI: [10.3390/s23094469](https://doi.org/10.3390/s23094469). URL: <https://www.mdpi.com/1424-8220/23/9/4469>.
- [3] Gianluca Peruzzi et al. «Combining LoRaWAN and NB-IoT for Edge-to-Cloud Low-Power Connectivity». En: *Applied Sciences* 12.3 (2022), pág. 1497. DOI: [10.3390/app12031497](https://doi.org/10.3390/app12031497). URL: <https://www.mdpi.com/2076-3417/12/3/1497>.
- [4] Miovision. *TrafficLink / Managed Connectivity*. <https://miovision.com/trafficlink/>. Soluciones comerciales. 2023.
- [5] Sensys Networks. *Documentación técnica y productos*. <https://sensysnetworks.com/>. 2023.
- [6] MetroCount. *Contadores y guías técnicas*. <https://www.metrocount.com/>. 2023.
- [7] Exemys Managed Connectivity. Accedido: 16-Sep-2025. 2025. URL: <https://www.exemys.com/site/index.shtml>.
- [8] Digi International. *Digi Remote Manager: IoT Device Monitoring and Management Solution*. <https://www.digi.com/products/iot-software-services/digi-remote-manager>. Accedido: 11 de septiembre de 2025. 2025.
- [9] A. A. Sukmandhani, M. Zarlis y Nurudin. «Monitoring Applications for Vehicle based on Internet of Things (IoT) using the MQTT Protocol». En: *BINUS Conference Proceedings*. Accedido: 11 de septiembre de 2025. 2023. URL: <https://research.binus.ac.id/publication/C1B25545-66F9-49C3-B873-D6C537EA23B3/monitoring-applications-for-vehicle-based-on-internet-of-things-iot-using-the-mqtt-protocol/>.
- [10] S. Bharath y C. Khusi. «IoT Based Smart Traffic System Using MQTT Protocol: Node-Red Framework». En: *2nd Global Conference for Advancement in Technology (GCAT)*. Accedido: 11 de septiembre de 2025. 2021. DOI: [10.1109/GCAT52182.2021.9587636](https://doi.org/10.1109/GCAT52182.2021.9587636).
- [11] Zuoling Niu. «Research and Implementation of Internet of Things Communication System Based on MQTT Protocol». En: *Journal of Physics: Conference Series* 012019 (2023). Accedido: 11 de septiembre de 2025.
- [12] OpenRemote. *OpenRemote: 100 % Open Source IoT Device Management Platform*. <https://openremote.io/>. Accedido: 11 de septiembre de 2025. 2025.
- [13] Espressif Systems. *ESP32-C3 Series Datasheet*. Accedido: 23-Sep-2025. Espressif Systems. 2021. URL: https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf.
- [14] Analog Devices. *Fundamentals of RS-232 Serial Communications*. <https://www.analog.com/en/resources/technical-articles/fundamentals-of-rs232-serial-communications.html>. Accedido: 11-Sep-2025. 2020.

- [15] Texas Instruments. *RS-232 Glossary and Selection Guide*. <https://www.ti.com/lit/SLLA607>. Accedido: 11-Sep-2025. 2016.
- [16] Douglas E. Comer. *Internetworking with TCP/IP: Principles, Protocols, and Architecture*. 6th. Pearson, 2014. ISBN: 9780136085300.
- [17] OASIS y MQTT.org. *MQTT Version 5.0 Specification*. <https://mqtt.org/mqtt-specification/>. Accedido: 11-Sep-2025. 2019.
- [18] IBM. *What Is a REST API (RESTful API)?* <https://www.ibm.com/think/topics/rest-apis>. Accedido: 11-Sep-2025. 2021.
- [19] Microsoft Azure Architecture Center. *Web API Design Best Practices*. <https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design>. Accedido: 11-Sep-2025. 2022.
- [20] Texas Instruments. *MAX232: Dual EIA-232 Driver/Receiver*. Datasheet. 2016. URL: <https://www.ti.com/lit/ds/symlink/max232.pdf>.
- [21] Espressif Systems. *ESP-IDF Programming Guide for ESP32-C3*. <https://docs.espressif.com/projects/esp-idf/en/stable/esp32c3/index.html>. Consultado el 11 de septiembre de 2025. 2024.
- [22] *SIM800L GSM/GPRS Module Datasheet*. Disponible en: https://simcom.ee/documents/SIM800L/SIM800L_Hardware_Design_V1.01.pdf. SIMCom Wireless Solutions. 2019.
- [23] Eclipse Foundation. *Eclipse Mosquitto: An Open Source MQTT Broker*. <https://mosquitto.org/>. Consultado el 11 de septiembre de 2025. 2025.
- [24] OpenJS Foundation. *Node.js JavaScript Runtime*. <https://nodejs.org/>. Consultado el 11 de septiembre de 2025. 2025.
- [25] Express.js Foundation. *Express: Fast, unopinionated, minimalist web framework for Node.js*. <https://expressjs.com/>. Consultado el 11 de septiembre de 2025. 2025.
- [26] Oracle Corporation. *MySQL: The World's Most Popular Open Source Database*. <https://www.mysql.com/>. Consultado el 11 de septiembre de 2025. 2025.
- [27] *Sequelize*. <https://sequelize.org/>. Accedido: 2025-09-25. 2025.
- [28] *Winston - A logger for just about everything*. <https://github.com/winstonjs/winston>. Accedido: 2025-09-25. 2025.
- [29] *Morgan - HTTP request logger middleware for Node.js*. <https://github.com/expressjs/morgan>. Accedido: 2025-09-25. 2025.
- [30] Ionic Team. *Ionic Framework - Cross-platform mobile apps with web technologies*. Último acceso: 19 de septiembre de 2025. 2025. URL: <https://ionicframework.com/>.
- [31] Angular Team. *Angular - One framework. Mobile desktop*. Último acceso: 19 de septiembre de 2025. 2025. URL: <https://angular.io>.
- [32] Docker Documentation. *Docker Compose: Define and run multi-container applications*. <https://docs.docker.com/compose/intro/>. Accedido: 02-10-2025. 2025.
- [33] Espressif Systems. *ESP-IDF Programming Guide*. Último acceso: 19 de septiembre de 2025. 2025. URL: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/>.
- [34] GitHub, Inc. *GitHub: Where the world builds software*. Último acceso: 19 de septiembre de 2025. 2025. URL: <https://github.com/>.
- [35] Martin Fowler. *Patterns of Enterprise Application Architecture*. Capítulo 11: Object-Relational Behavioral Patterns. Addison-Wesley Professional, 2002. ISBN: 978-0321127426.

- [36] M. Jones, J. Bradley y N. Sakimura. *JSON Web Token (JWT)*. RFC 7519. Internet Engineering Task Force (IETF). 2015. URL: <https://datatracker.ietf.org/doc/html/rfc7519>.
- [37] FreeRTOS.org / Real Time Engineers Ltd. *FreeRTOS Reference Manual*. Manual de referencia del kernel FreeRTOS (API y configuración). 2018. URL: <https://www.freertos.org/FreeRTOS-Reference-Manual.pdf> (visitado 18-11-2025).
- [38] Grafana Labs. *Grafana Documentation*. Documentación oficial de Grafana. 2025. URL: <https://grafana.com/docs/> (visitado 18-11-2025).
- [39] phpMyAdmin. *phpMyAdmin: Free software tool for MySQL database administration*. <https://www.phpmyadmin.net/>. Herramienta web para la administración de bases de datos MySQL. 2025.
- [40] Postman, Inc. *Postman API Platform*. Herramienta para pruebas y automatización de APIs REST. 2025. URL: <https://www.postman.com/>.
- [41] Postman, Inc. *Postman Collection Runner and Newman CLI*. Herramienta de Postman para ejecutar pruebas automatizadas de colecciones de API. 2025. URL: <https://learning.postman.com/docs/collections/running-collections/intro-to-collection-runs/>.
- [42] Inc. Postman. *Newman: Command-line Collection Runner for Postman*. <https://www.npmjs.com/package/newman>. Último acceso: 31 de octubre de 2025. 2025.
- [43] Google Developers. *Chrome DevTools*. Accedido: octubre 2025. 2024. URL: <https://developer.chrome.com/docs/devtools/>.
- [44] Google Developers. *Google Lighthouse*. Accedido: octubre 2025. 2024. URL: <https://developer.chrome.com/docs/lighthouse/>.
- [45] FHWA. *Traffic Monitoring Guide*. Inf. téc. Federal Highway Administration, 2022. URL: https://www.fhwa.dot.gov/policyinformation/tmguide/2022_TMG_Final_Report.pdf.
- [46] FHWA. *Traffic Detector Handbook, 3rd Edition*. Inf. téc. Federal Highway Administration, 2006. URL: <https://www.fhwa.dot.gov/publications/research/operations/its/06108/06108.pdf>.