.UBAfiuba
FACULTAD DE INGENIERÍA

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

Jurados:

Jurado 1 (pertenencia)
Jurado 2 (pertenencia)
Jurado 3 (pertenencia)

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

Índice de figuras

Índice de tablas

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 [**asiain2021lora**] , [**micko2023iot**].

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 [**peruzzi2022lorawan**], [**micko2023iot**].

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 [**miovision**] ,[**sensys**] ,[**metrocount**].

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 [**exemys**], [**digiRemoteManager**], 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 [**monitoringVehicles2023**], [**iotSmartTraffic2021**]. 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 [**iopMQTTSystem2023**].

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 [**openRemoteSolution**]. 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.

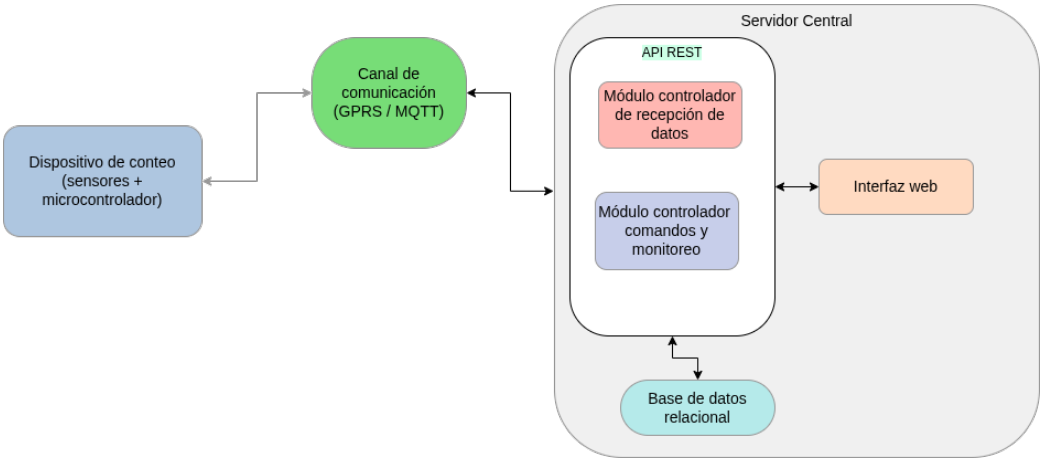

FIGURA 1.1. Diagrama en bloques del sistema.

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 [**esp32c3**]. 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 [**analogRS232**], [**tiRS232**].
- 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 [**comerTCPIP**] 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 [**mqttSpec**].
- 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 [**ibmRest**], [**microsoftApiDesign**] .

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 distintos rutas nacionales del país y cuenta con una interfaz de salida RS-232 [**tiRS232**] con un adaptador MAX232 [**max232**]. 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].

- 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 [**max232**].

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

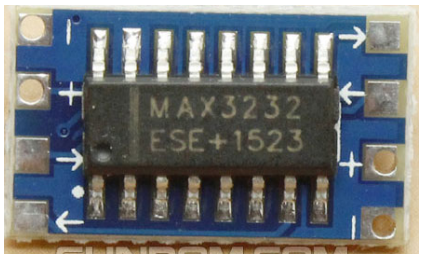

FIGURA 2.2. Módulo RS-232/TTL [2].

- 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 [**esp32c3IDF**].

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

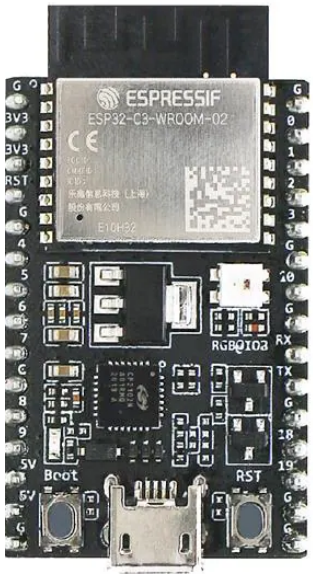

FIGURA 2.3. Microcontrolador ESP32-C3 utilizado en los nodos de campo [3].

- Módulo GPRS SIM800L: permite la conexión de los dispositivos a la red celular [**sim800l_datasheet**], que asegura la transmisión de datos al servidor

[1]Imagen tomada de http://transito.vialidad.gob.ar/

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

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

central mediante MQTT. Se eligió por su bajo costo, disponibilidad en el mercado y facilidad de integración con el ESP32-C3.

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

FIGURA 2.4. Módulo SIM800L [4].

- 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:

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

- MQTT con Eclipse Mosquitto. La mensajería ligera y bidireccional se implementa mediante el protocolo MQTT, operando sobre el broker Eclipse Mosquitto [**mosquitto**]. 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 [**nodejs**], un entorno de ejecución orientado a aplicaciones de red no bloqueantes, y se estructura con Express [**expressjs**], un framework ligero que facilita la creación de servicios RESTful.
- Base de datos relacional MySQL. La persistencia de los datos se implementa en MySQL [**mysql**], 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 [**sequelize**], 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 [**winston**], 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 [**morgan**], 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 [**ionic**] y Angular [**angular**]. 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 [**docker_compose**]. 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 [**espidf**], 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 [**github**], 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, garantizando trazabilidad, persistencia y control remoto.

- 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 por 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.

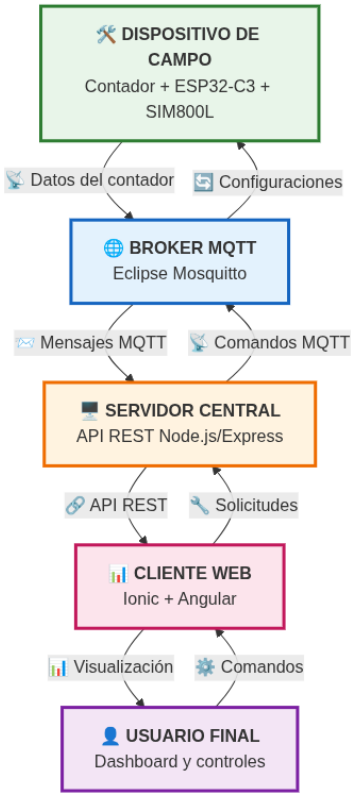

FIGURA 3.1. Diagrama de arquitectura del sistema y el flujo de datos.

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.

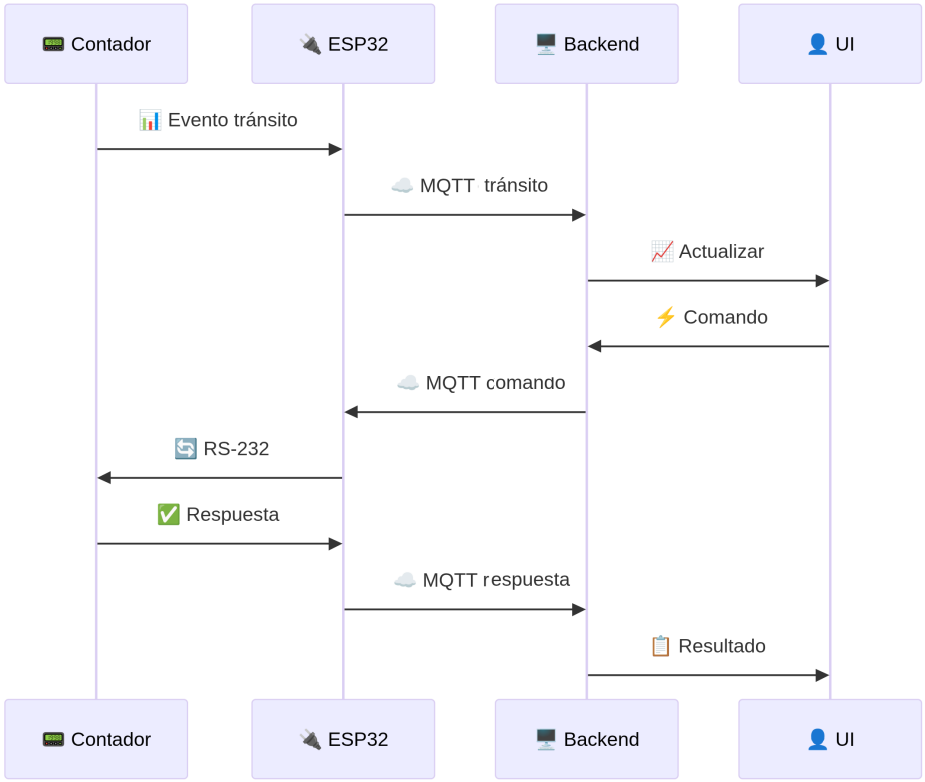

FIGURA 3.2. Diagrama de secuencia del flujo de datos.

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, evitando caídas de voltaje que puedan reiniciar el sistema.

    La figura 3.3 se presenta el diagrama de conexión entre los módulos del sistema, detallando 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.

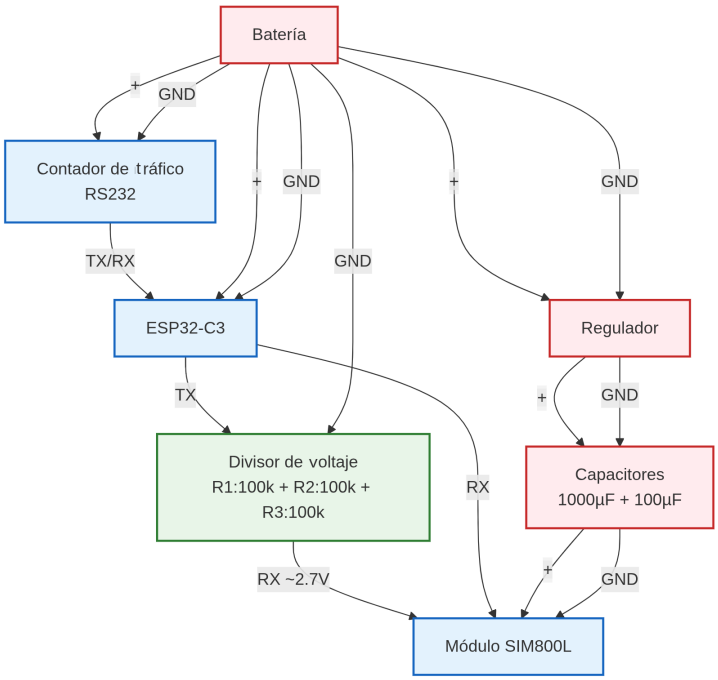

FIGURA 3.3. Diagrama de conexión entre los módulos del sistema.

  - 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.

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 se encuentra 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.

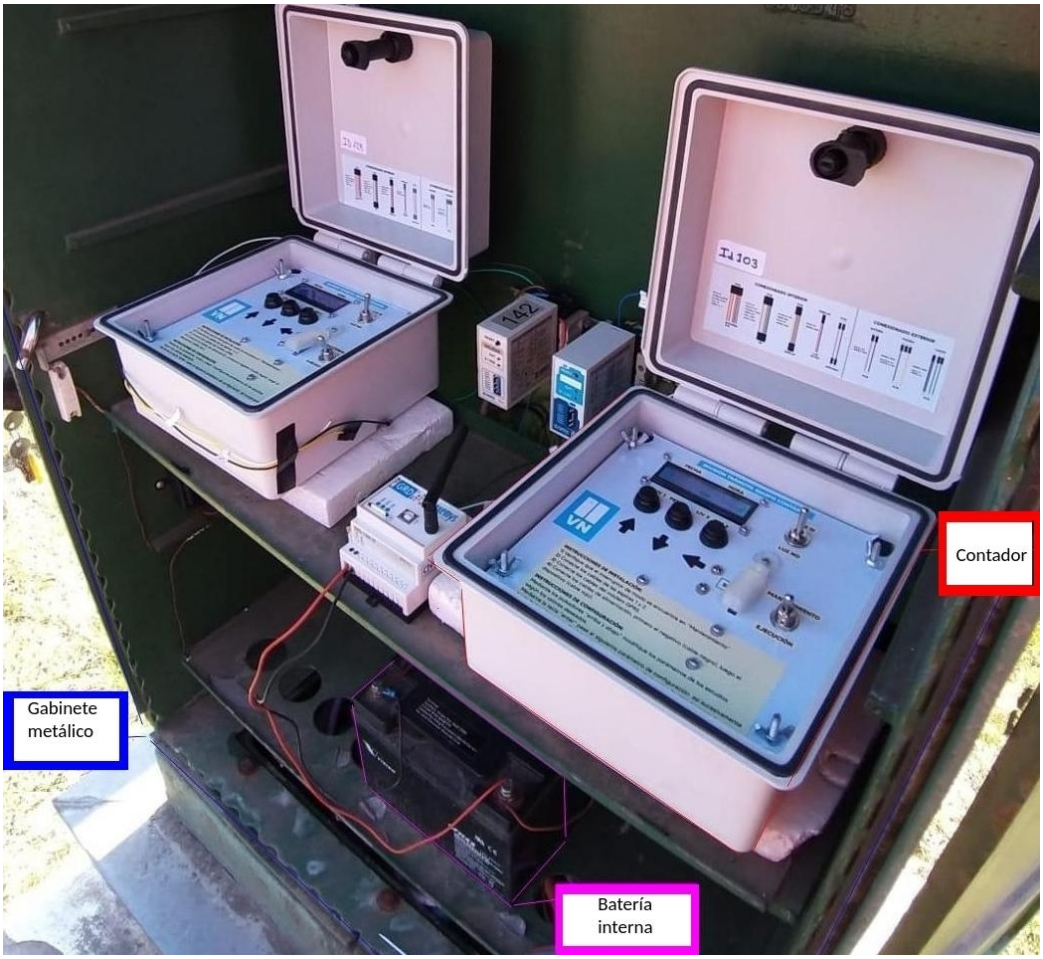

FIGURA 3.4. Fotografia contador de tránsito DTEC instalado en campo [1].

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.

[1] Imagen tomada de http://transito.vialidad.gob.ar/

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 [**sequelize**], un ORM [**fowler2002patterns**] 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 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 [**docker_compose**], que permite ejecutar backend, base de datos y broker [**mqttSpec**] en contenedores independientes. Además, el registro y la trazabilidad se gestionan con Winston [**winston**] y Morgan [**morgan**], lo que garantiza un monitoreo completo del sistema

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

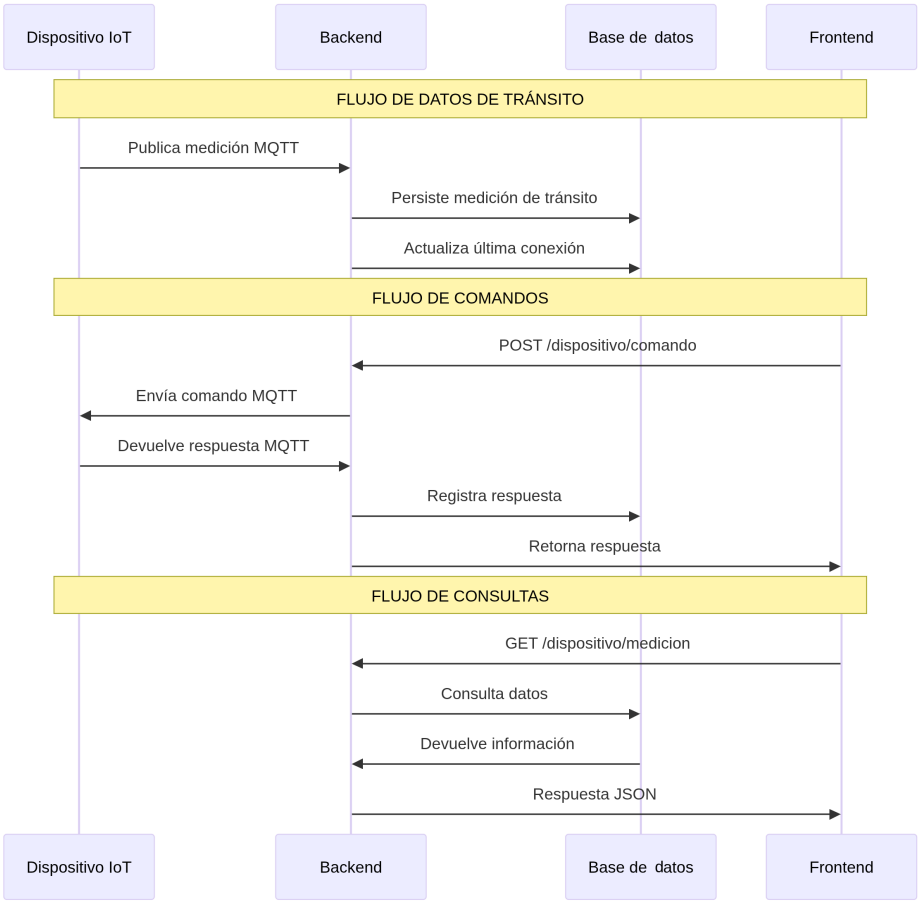

FIGURA 3.5. Diagrama de flujo de información del Backend.

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 ingesta 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[**jwtRFC7519**], 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.

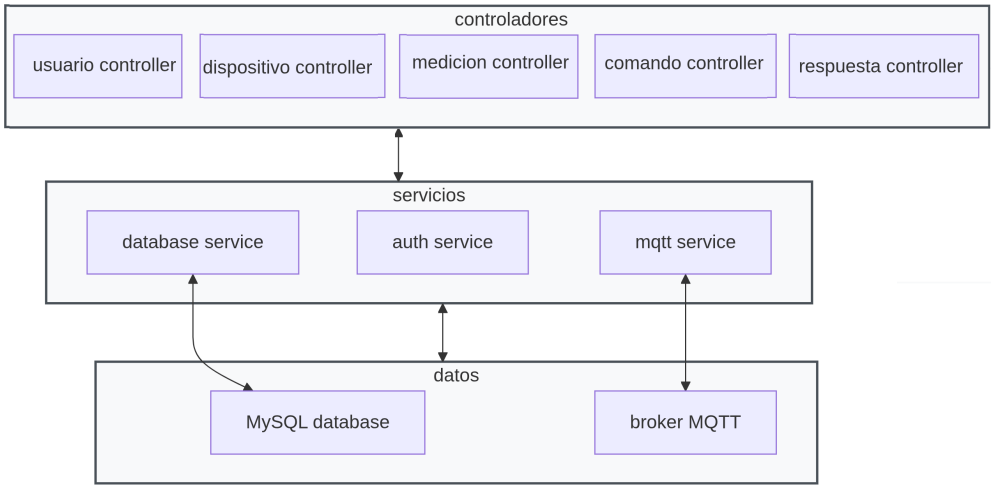

FIGURA 3.6. Diagrama con la disposición de los controladores y flujo de dependencias.

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 responsiva 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.

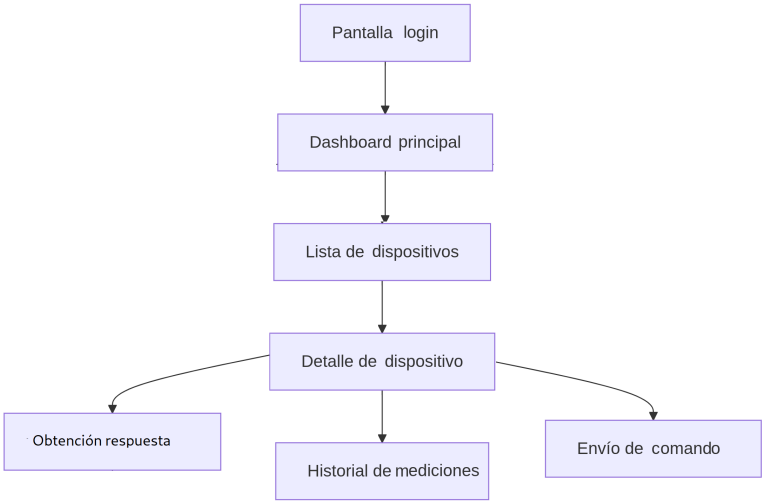

FIGURA 3.7. Diagrama de estructura de los componentes por pantalla.

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, manteniendo 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.

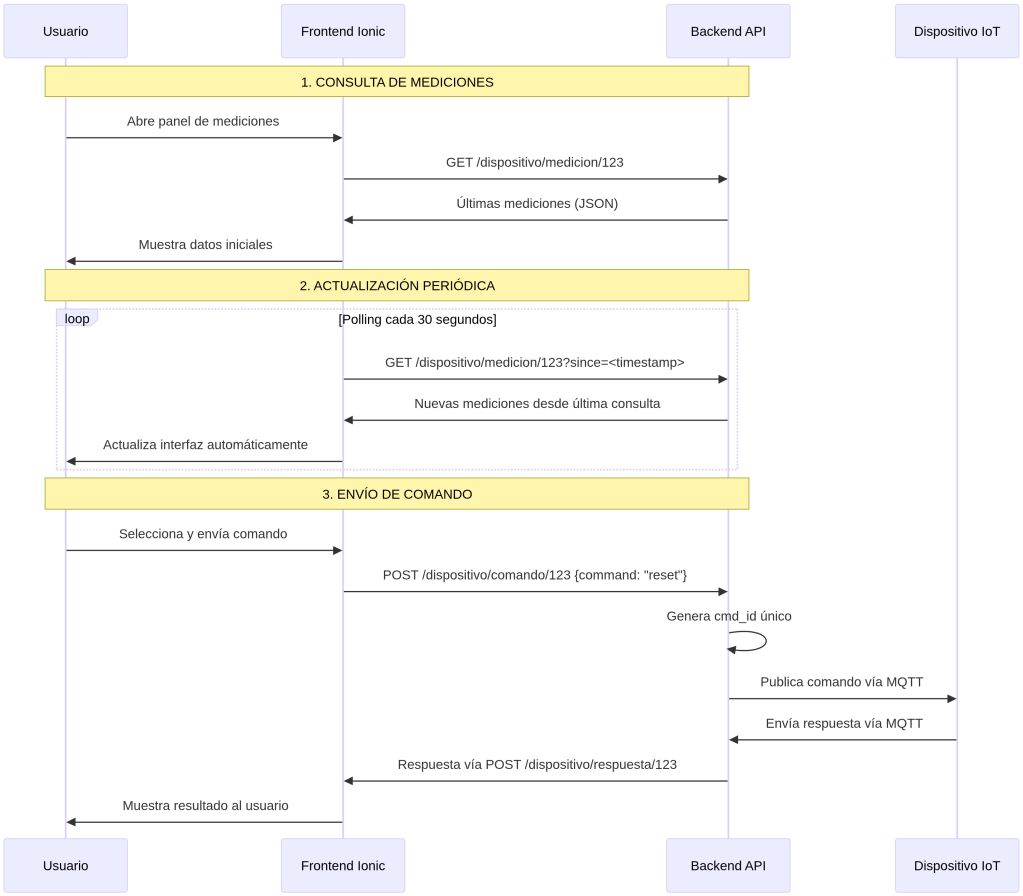

FIGURA 3.8. Diagrama de flujo de comunicación con el backend.

3.5. 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.

3.5.1. 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 combina las ventajas de la virtualización y la gestión centralizada propias del entorno *cloud*, con el control, la seguridad y la independencia de un despliegue local.

Para la orquestación se emplea Docker Compose, que permite instanciar todos los servicios (broker MQTT, API REST, base de datos y aplicación web) en contenedores aislados pero comunicados entre sí. De esta forma, el sistema puede escalar, actualizarse y mantenerse de manera unificada, preservando la trazabilidad y la integridad de los datos.

La integración continua (CI) automatiza la construcción, prueba y despliegue del sistema. Cada actualización del repositorio genera nuevas imágenes Docker, ejecuta validaciones automáticas y despliega los servicios en el entorno *on-premise*, que garantiza coherencia entre versiones y reduciendo errores manuales.

3.5.2. Monitoreo post-implantación

Una vez desplegado el sistema, resulta fundamental contar con mecanismos de monitoreo que permitan evaluar su correcto funcionamiento en campo:

- Logs centralizados: tanto el backend como el broker MQTT registran eventos en archivos y consola. Se integra con Grafana para correlacionar métricas.
- Alertas y métricas: mediante Grafana es posible recolectar indicadores de CPU, memoria y estado de contenedores. También se pueden graficar métricas de tráfico MQTT (mensajes publicados, latencias, pérdidas).
- Supervisión de dispositivos: la API REST expone endpoints que informan conectividad y parámetros básicos (nivel de batería, último evento recibido). Estos datos se representan en la interfaz web como panel de salud del sistema.
- Respaldo y recuperación: la base de datos implementa backups automáticos y permite restauraciones parciales. Esto garantiza que el historial de eventos no se pierda ante fallas de hardware o corrupción de datos.

3.6. Integración con la infraestructura existente

Una de las principales ventajas de este diseño es que no requiere modificaciones internas en el contador de tránsito. El nodo recibe los pulsos de detección mediante la interfaz RS-232, que preserva la integridad del equipo original.

El ESP32-C3 no se limita a reenviar datos, sino que añade valor al sistema al realizar un preprocesado local: filtra tramas, agrupa eventos en función de ventanas de tiempo y asegura la transmisión con políticas de reintento. Asimismo, la conexión con el servidor central mediante MQTT garantiza interoperabilidad con aplicaciones externas y facilita la escalabilidad del sistema.

En este contexto, los nodos de campo cumplen un doble rol: por un lado, son captadores de datos provenientes de los sensores de tránsito y por otro, actúan como puntos de control remoto, capaces de ejecutar comandos enviados desde la plataforma central. Esta dualidad refuerza la flexibilidad del sistema y lo hace adaptable a distintas políticas de gestión 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.

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.

Los tiempos promedio de publicación por evento se mantuvieron entre 300 y 600 ms en escenarios con conexión estable, con demoras proporcionales durante los períodos de reconexión.

En conclusión, el banco de pruebas permitió validar la arquitectura propuesta. Los resultados confirmaron un comportamiento confiable frente a condiciones reales de operación y demostraron la efectividad de los mecanismos de encolado y retransmisión.

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 asincrónicas (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* [**postman**]. 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* [**postman_runner**] 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* [**newman_postman**].

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 apropiados.

- 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 presenta la tabla de los resultados de pruebas de endpoints REST:

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

4.2.4. Conclusiones

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

- Firmware (ESP32-C3): se validó el análisis de tramas RS-232, el almacenamiento temporal en colas FIFO y la publicación confiable de mensajes MQTT.
- Broker MQTT: se realizaron desconexiones simuladas. El sistema mantuvo la sesión y retransmitió los mensajes pendientes.
- Backend: se comprobó la correcta gestión de solicitudes REST y la sincronización con el broker MQTT.
- Frontend: se verificó la comunicación bidireccional con la API REST y la actualización en tiempo real de las mediciones.
- Manejo de errores: los registros de Winston y Morgan mostraron reconexiones exitosas sin pérdida de información.

4.3.3. Conclusiones

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, garantizando 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 el manejo de errores de conexión y autenticación, asi se generen mensajes claros y retroalimentación inmediata al usuario.

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 llevaron a cabo ensayos funcionales, de compatibilidad y de rendimiento, que combinó herramientas automáticas y observación directa de la interacción del usuario.

- Compatibilidad y visualización: se validó la correcta visualización de los componentes en distintas resoluciones y navegadores, utilizando *Chrome DevTools* [**chromedevtools**] 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* [**lighthouse**] 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 aumento 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.

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 reconexiones de red.
- Interfaz intuitiva y valorada positivamente por los usuarios de prueba, con mejoras implementadas en la versión final.

En conjunto, 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 configuró un entorno de ensayo en el que participaron todos los subsistemas desplegados simultáneamente:

- Nodo de campo: compuesto por el microcontrolador ESP32-C3, el módulo GPRS SIM800L y la interfaz RS-232 hacia el contador de tránsito DTEC, encargado de generar detecciones simuladas.
- Broker MQTT: instancia del servidor Eclipse Mosquitto, actuando como intermediario para la mensajería entre los nodos de campo y el backend.
- Backend: implementado en Node.js/Express, con base de datos MySQL y ORM Sequelize, encargado de procesar los eventos, registrar los datos y gestionar los comandos remotos.
- Interfaz web (frontend): desarrollada con Ionic y Angular, utilizada para visualizar en tiempo real las detecciones, el estado de los dispositivos y emitir comandos de control.

El flujo de integración se estructuró de la siguiente manera:

1. El contador de tránsito DTEC generó una trama RS-232 que fue recibida por el nodo ESP32-C3.
2. El firmware procesó la trama, generó un evento y lo publicó mediante MQTT en el tópico `dispositivo/{id}/medicion`.
3. El backend, suscrito a dicho tópico, validó el mensaje, lo registró en la base de datos y notificó a la interfaz web a través de su API REST.
4. El frontend consultó la API y actualizó la vista en tiempo real con la nueva medición.
5. Desde la interfaz, se envió un comando de prueba al nodo, el cual fue publicado por el backend en el tópico `dispositivo/{id}/comando`.
6. El firmware recibió el comando, ejecutó la acción asociada y respondió mediante un mensaje en el tópico `dispositivo/{id}/respuesta`, que fue procesado nuevamente por el backend y mostrado al usuario.

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 frontend 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.

4.5.3. Conclusiones de la integración

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, ofreciendo 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 proyectos 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), permitiendo 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	Totalmente soportada		
(colas FIFO, reintentos)	Parcial	Poco explorada	
Adecuación a rutas	Adaptada a entornos rurales y semiurbanos	Genérica	Variable
Soporte y documentación	Media (open source, docs técnicas)	Alta (soporte empresarial)	Baja

En síntesis, la solución desarrollada logra un equilibrio entre robustez, bajo costo y adaptabilidad, posicionándola como alternativa viable para proyectos 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.