

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

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.	Resultados y metas ??	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/respuesta exitosa.	35
4.9. Solicitud POST /api/respuesta 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

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/respuesta exitosa.	35
4.9. Solicitud POST /api/respuesta 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

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.

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.

- 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.

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 DTEC¹.

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

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 DTEC¹.

2.2. Componentes de hardware utilizados

7

- 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>

2.2. Componentes de hardware utilizados

7

- 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 <http://transito.vialidad.gob.ar/>

³Imagen 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>

- 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.

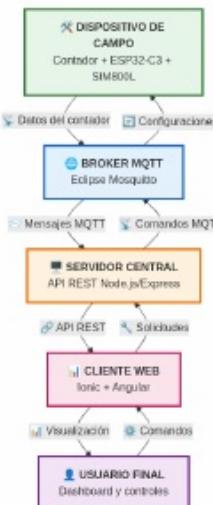


FIGURA 3.1. Diagrama de arquitectura del sistema y el flujo de datos.

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.

- 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.

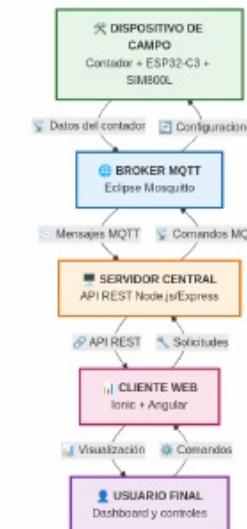


FIGURA 3.1. Diagrama de arquitectura del sistema y el flujo de datos.

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:

3.1. Arquitectura del sistema

13

- 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.

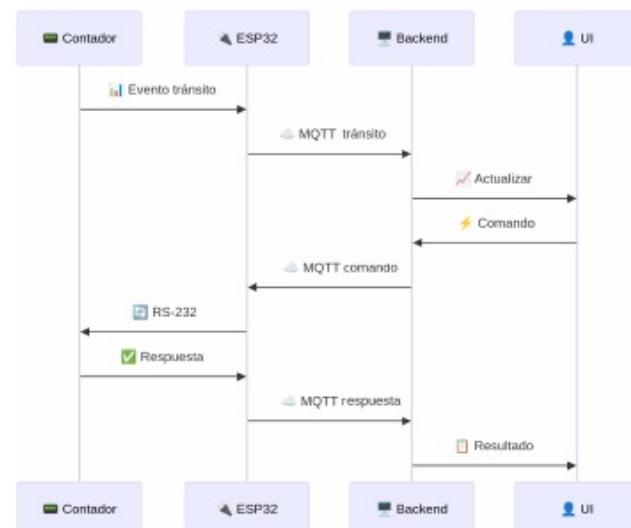


FIGURA 3.2. Diagrama de secuencia del flujo de datos.

3.1. Arquitectura del sistema

13

- 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.

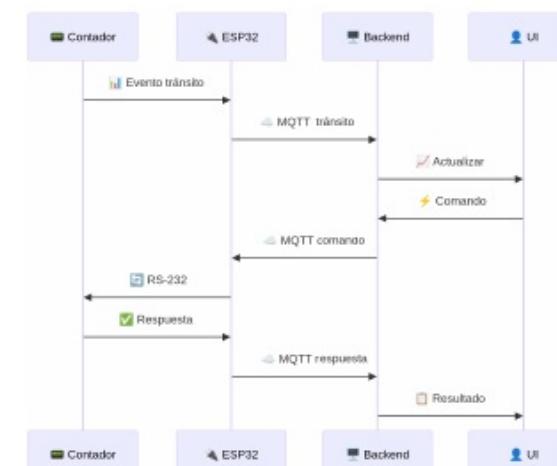


FIGURA 3.2. Diagrama de secuencia del flujo de datos.

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.

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.

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.

3.2. Arquitectura del nodo

15

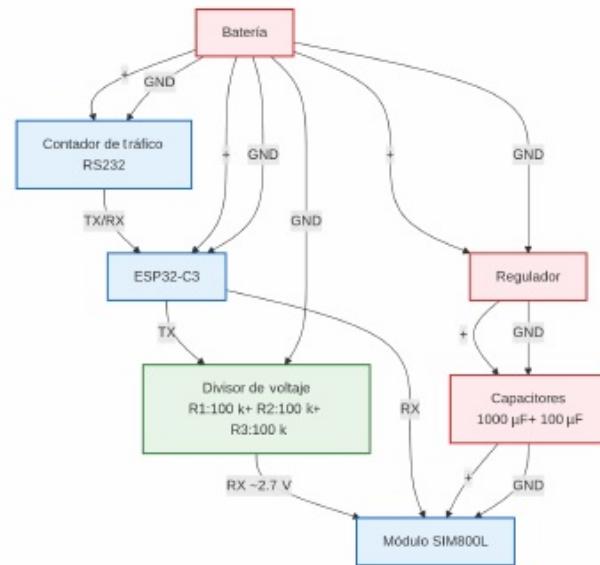


FIGURA 3.3. Diagrama de conexión entre los módulos del sistema.

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.

3.2. Arquitectura del nodo

15

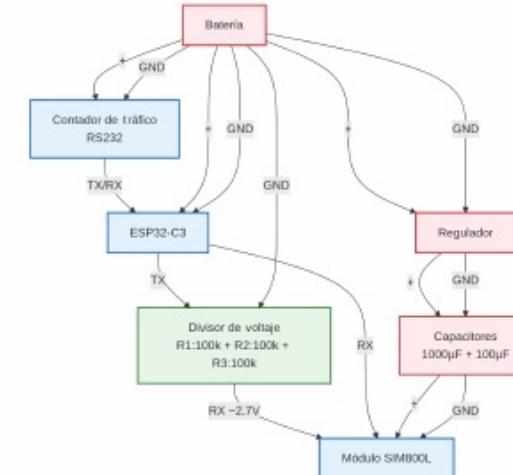


FIGURA 3.3. Diagrama de conexión entre los módulos del sistema.

- **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. 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/>



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/>

3.3. Desarrollo del backend

17

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

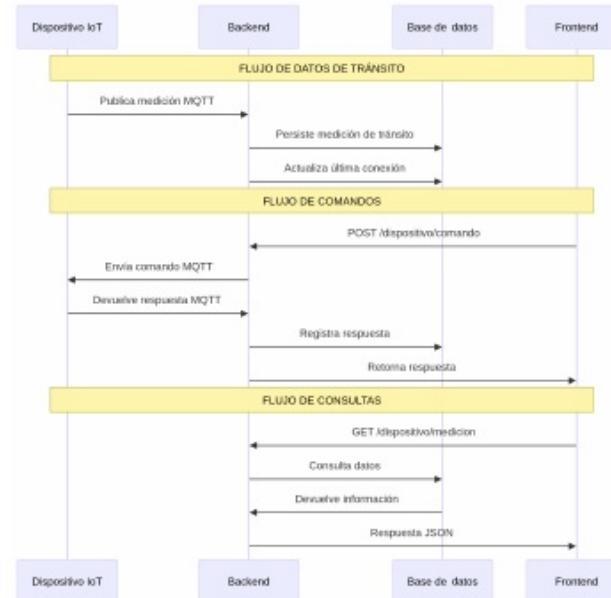


FIGURA 3.5. Diagrama de flujo de información del backend.

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. Desarrollo del backend

17

la trazabilidad se gestionan con Winston [28] y Morgan [29], lo que garantiza un monitoreo completo del sistema. En la figura 3.5 se observa el diagrama de flujo de información del backend.

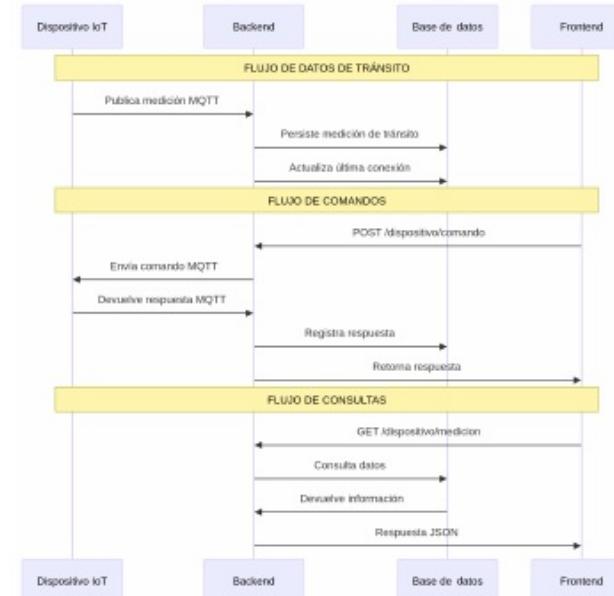


FIGURA 3.5. Diagrama de flujo de información del backend.

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 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.

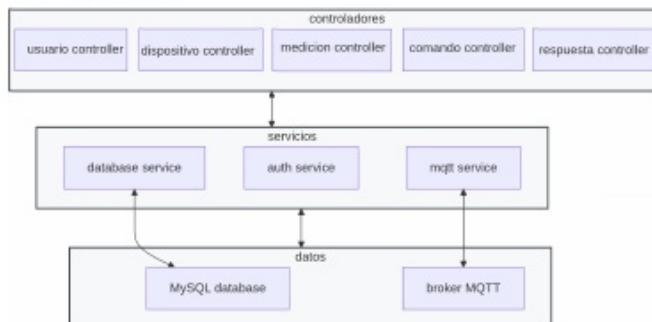


FIGURA 3.6. Diagrama con la disposición de los controladores y flujo de dependencias.

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.

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.

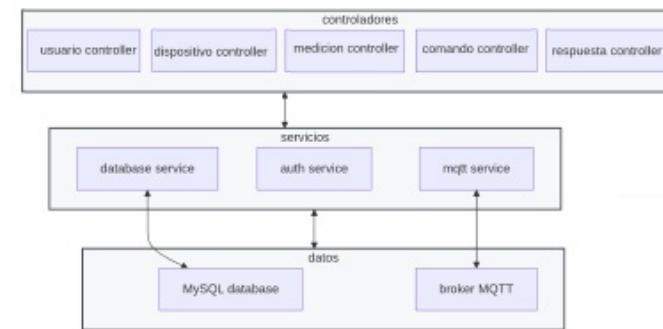


FIGURA 3.6. Diagrama con la disposición de los controladores y flujo de dependencias.

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.

3.3. Desarrollo del backend

19

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.

3.3. Desarrollo del backend

19

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.

3.5. Desarrollo del firmware

21

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.

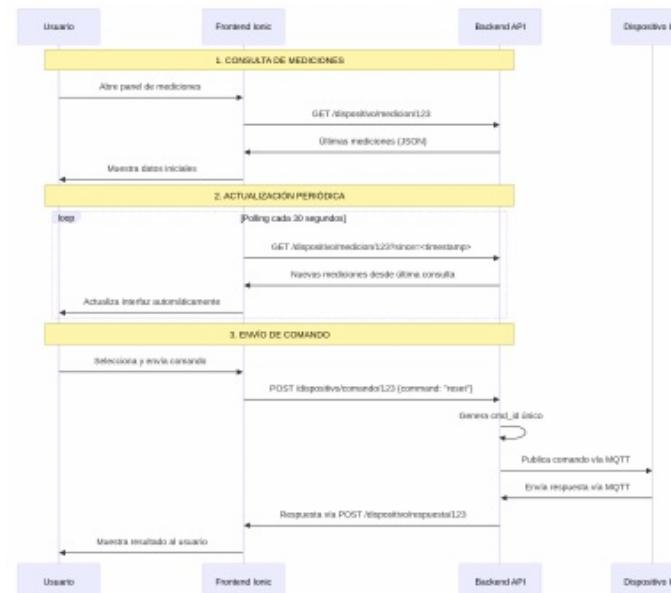


FIGURA 3.8. Diagrama de flujo de comunicación con el backend.

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. Desarrollo del firmware

21

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.

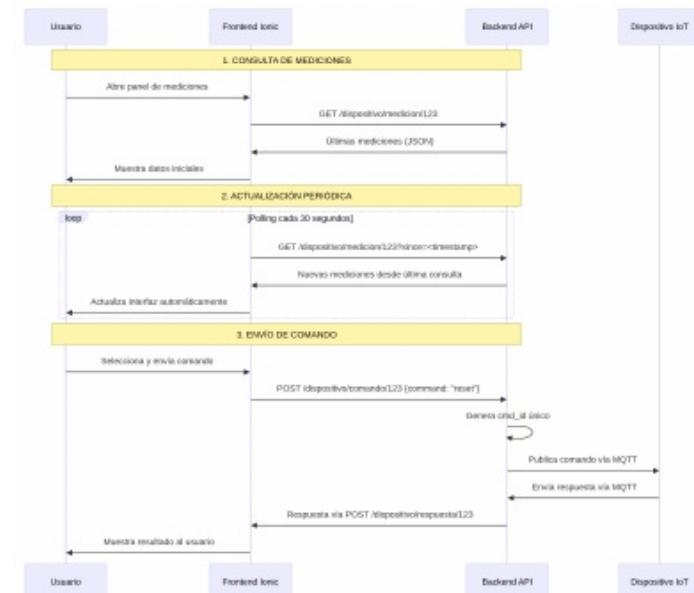


FIGURA 3.8. Diagrama de flujo de comunicación con el backend.

En conjunto, esta 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.

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 transits, 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.

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

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 cual 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.

En conjunto, 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

3.7. Integración con la infraestructura existente

27

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

3.7. Integración con la infraestructura existente

27

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.

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.

En conjunto, 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.

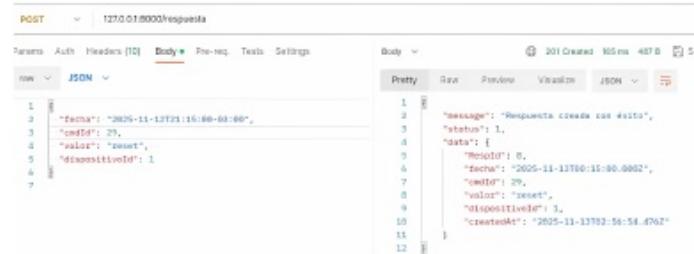
4.2. Pruebas de la API REST

35

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.

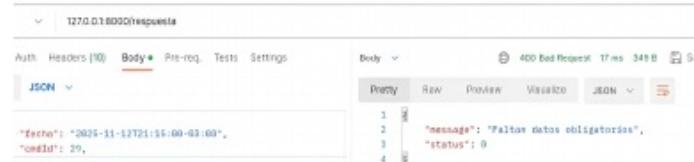
La figura 4.8 muestra un ejemplo de respuesta registrada exitosamente, con los campos `fecha`, `cmdId`, `valor` y `dispositivoId`.



```
POST 127.0.0.1:8000/respuesta
{
    "fecha": "2025-11-12T21:15:00-03:00",
    "cmdId": 29,
    "valor": "reset",
    "dispositivoId": 1
}
```

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.



```
POST 127.0.0.1:8000/respuesta
{
    "fecha": "2025-11-12T21:15:00-03:00",
    "cmdId": 29
}
```

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:

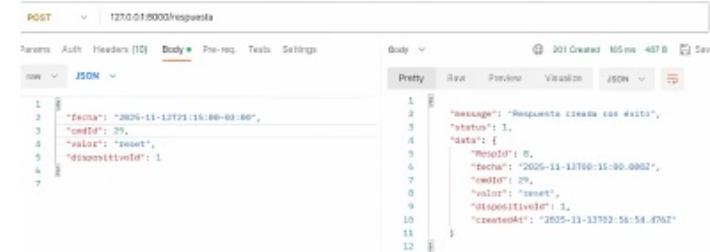
4.2. Pruebas de la API REST

35

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.

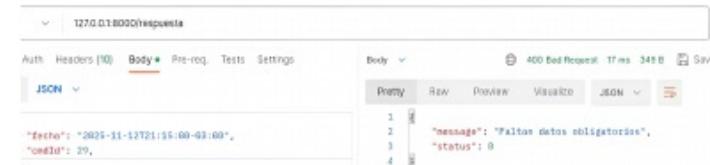
La figura 4.8 muestra un ejemplo de respuesta registrada exitosamente, con los campos `fecha`, `cmdId`, `valor` y `dispositivoId`.



```
POST 127.0.0.1:8000/respuesta
{
    "fecha": "2025-11-12T21:15:00-03:00",
    "cmdId": 29,
    "valor": "reset",
    "dispositivoId": 1
}
```

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.



```
POST 127.0.0.1:8000/respuesta
{
    "fecha": "2025-11-12T21:15:00-03:00",
    "cmdId": 29
}
```

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:

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:

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.



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.

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



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 la misma.

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.

4.5. Prueba final de integración

43

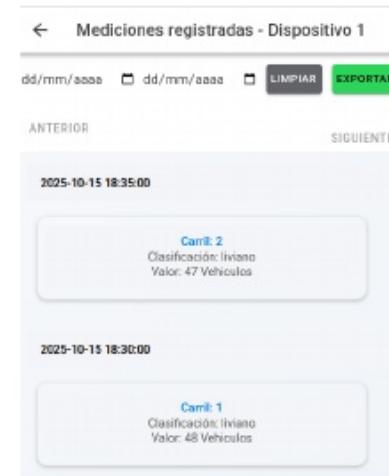


FIGURA 4.18. Panel de visualización del historial de mediciones.

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. Prueba final de integración

43



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:

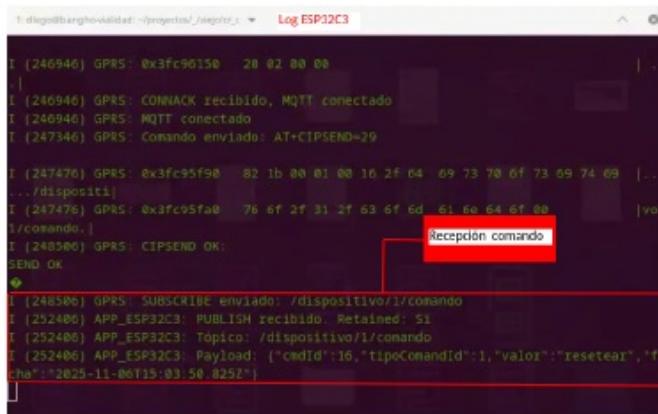
- Compatibilidad completa con navegadores Chrome y Firefox, y compatibilidad en móviles Android.
- Tiempos de carga promedio inferiores a 3 s en escritorio y 4 s en dispositivos móviles.
- Comunicación estable con la API REST y el broker MQTT, incluso ante re conexiones de red.
- Interfaz intuitiva y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

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.

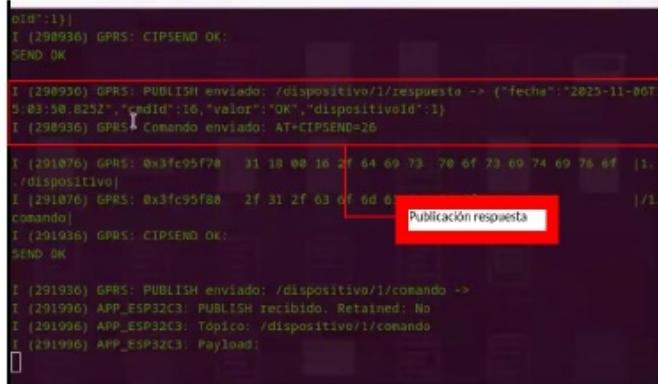


The screenshot shows the serial monitor interface with the title 'Log ESP32C3'. The log output includes several MQTT messages. A red box highlights the line 'I (247476) GPRS: 0x3fc95f90 82 1b 80 01 08 16 2f 64 69 73 78 0f 73 69 74 69 | .../dispositivo|' with the annotation 'Recepción comando' (Command reception). Another red box highlights the line 'I (248506) GPRS: SUBSCRIBE enviado: /dispositivo/1/comando'.

```
I (246946) GPRS: 0x3fc90150 28 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 80 01 08 16 2f 64 69 73 78 0f 73 69 74 69 | ...
.../dispositivo|
I (247476) GPRS: 0x3fc95fa0 76 6f 2f 31 2f 63 6f 6d 61 6e 64 6f 00
I/comando|
I (248500) 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.



The screenshot shows the serial monitor interface with the title 'Log ESP32C3'. The log output includes MQTT messages. A red box highlights the line 'I (291876) GPRS: 0x3fc95f70 31 18 00 16 2f 64 69 73 78 0f 73 69 74 69 76 0f | .../dispositivo|' with the annotation 'Publicación respuesta' (Response publication). Another red box highlights the line 'I (291936) GPRS: CIPSEND OK:'.

```
old":1}]
I (290936) GPRS: CIPSEND OK:
SEND OK

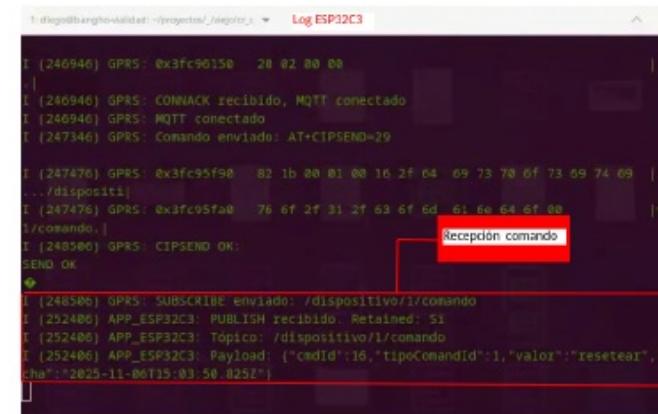
I (290936) GPRS: PUBLISH enviado: /dispositivo/1/respuesta -> {"fecha":"2025-11-06T15:03:50.825Z","cmdId":16,"valor":"OK","dispositivoId":1}
I (290936) GPRS: Comando enviado: AT+CIPSEND=26

I (291876) GPRS: 0x3fc95f70 31 18 00 16 2f 64 69 73 78 0f 73 69 74 69 76 0f | ...
.../dispositivo|
I (291876) GPRS: 0x3fc95f80 2f 31 2f 63 6f 6d 61 6e 64 6f 00
I/comando|
I (291936) GPRS: CIPSEND OK:
SEND OK

I (291936) GPRS: PUBLISH enviado: /dispositivo/1/comando ->
I (291990) APP_ESP32C3: PUBLISH recibido. Retained: No
I (291990) APP_ESP32C3: Tópico: /dispositivo/1/comando
I (291990) 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.

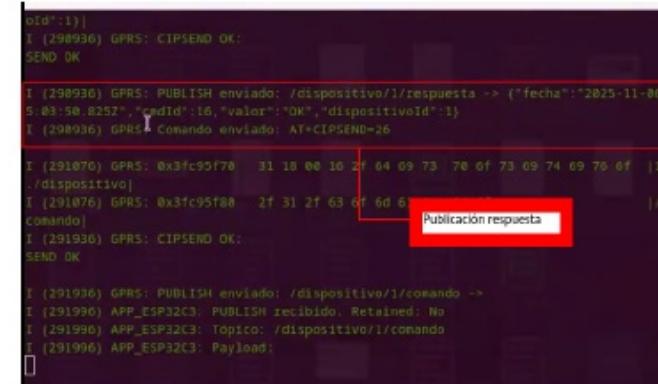


The screenshot shows the serial monitor interface with the title 'Log ESP32C3'. The log output includes MQTT messages. A red box highlights the line 'I (291876) GPRS: 0x3fc95f70 31 18 00 16 2f 64 69 73 78 0f 73 69 74 69 76 0f | .../dispositivo|' with the annotation 'Publicación respuesta' (Response publication). Another red box highlights the line 'I (291936) GPRS: CIPSEND OK:'.

```
I (246946) GPRS: 0x3fc90150 28 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 80 01 08 16 2f 64 69 73 78 0f 73 69 74 69 | ...
.../dispositivo|
I (247476) GPRS: 0x3fc95fa0 76 6f 2f 31 2f 63 6f 6d 61 6e 64 6f 00
I/comando|
I (248500) 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, el cual 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.



The screenshot shows the serial monitor interface with the title 'Log ESP32C3'. The log output includes MQTT messages. A red box highlights the line 'I (291876) GPRS: 0x3fc95f70 31 18 00 16 2f 64 69 73 78 0f 73 69 74 69 76 0f | .../dispositivo|' with the annotation 'Publicación respuesta' (Response publication). Another red box highlights the line 'I (291936) GPRS: CIPSEND OK:'.

```
old":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 (291876) GPRS: 0x3fc95f70 31 18 00 16 2f 64 69 73 78 0f 73 69 74 69 76 0f | ...
.../dispositivo|
I (291876) GPRS: 0x3fc95f80 2f 31 2f 63 6f 6d 61 6e 64 6f 00
I/comando|
I (291936) GPRS: CIPSEND OK:
SEND OK

I (291936) GPRS: PUBLISH enviado: /dispositivo/1/comando ->
I (291990) APP_ESP32C3: PUBLISH recibido. Retained: No
I (291990) APP_ESP32C3: Tópico: /dispositivo/1/comando
I (291990) 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.

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.

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. Resultados y metas

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.