



Modernización de contadores de tránsito con comunicación bidireccional

Ing. Diego Aníbal Vázquez

Carrera de Especialización en Internet de las Cosas

Director: Ing. Rogelio Diego González

Jurados:

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

Ciudad de Buenos Aires, diciembre de 2025



Modernización de contadores de tránsito con comunicación bidireccional

Ing. Diego Aníbal Vázquez

Carrera de Especialización en Internet de las Cosas

Director: Ing. Rogelio Diego González

Jurados:

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

Ciudad de Buenos Aires, marzo de 2026

Índice general

Resumen	1
1. Introducción general	1
1.1. Motivación	1
1.1.1. Contexto actual	1
1.1.2. Limitaciones del desarrollo previo	1
1.1.3. Impacto esperado	2
1.1.4. Diseño conceptual	2
1.2. Objetivos	2
1.2.1. Objetivo general	2
1.2.2. Objetivos específicos	2
1.3. Estado del arte y propuesta de valor	3
1.4. Alcance	3
2. Introducción específica	5
2.1. Protocolos y comunicación	5
2.2. Componentes de hardware utilizados	6
2.3. Tecnologías de software aplicadas	8
2.4. Software de control de versiones	10
3. Diseño e implementación	11
3.1. Arquitectura del sistema	11
3.1.1. Flujo de datos	12
3.2. Arquitectura del nodo	14
3.3. Desarrollo del backend	16
3.3.1. Arquitectura y tecnologías	17
3.3.2. Funcionalidades principales	18
3.3.3. Organización en controladores	18
3.3.4. Mapa de endpoints	19
3.3.5. Seguridad y extensibilidad	20
3.4. Desarrollo del frontend	21
3.4.1. Arquitectura y tecnologías	21
3.4.2. Funcionalidades principales	21
3.4.3. Integración con el backend	22
3.5. Despliegue del sistema	23
3.5.1. Entorno productivo e integración continua	23
3.5.2. Monitoreo post-implantación	24
3.6. Integración con la infraestructura existente	24
4. Ensayos y resultados	25
4.1. Banco de pruebas	25
4.1.1. Diseño del entorno de pruebas	25
4.1.2. Metodología experimental	26

Índice general

Resumen	1
1. Introducción general	1
1.1. Motivación	1
1.1.1. Contexto actual	1
1.1.2. Limitaciones del desarrollo previo	1
1.1.3. Impacto esperado	2
1.1.4. Diseño conceptual	2
1.2. Objetivos	2
1.2.1. Objetivo general	2
1.2.2. Objetivos específicos	2
1.3. Estado del arte y propuesta de valor	3
1.4. Alcance	3
2. Introducción específica	5
2.1. Protocolos y comunicación	5
2.2. Componentes de hardware utilizados	6
2.3. Tecnologías de software aplicadas	8
2.4. Software de control de versiones	10
3. Diseño e implementación	11
3.1. Arquitectura del sistema	11
3.1.1. Descripción ampliada de bloques y responsabilidades	11
3.1.2. Flujo de datos	13
3.2. Arquitectura del nodo	14
3.3. Desarrollo del backend	16
3.3.1. Arquitectura y tecnologías	17
3.3.2. Funcionalidades principales	17
3.3.3. Organización en controladores	18
3.3.4. Mapa de endpoints	18
3.3.5. Seguridad y extensibilidad	19
3.4. Desarrollo del frontend	20
3.4.1. Arquitectura y tecnologías	20
3.4.2. Funcionalidades principales	20
3.4.3. Integración con el backend	21
3.5. Despliegue del sistema	23
3.5.1. Entorno productivo e integración continua	23
3.5.2. Monitoreo post-implantación	23
3.6. Integración con la infraestructura existente	23
4. Ensayos y Resultados	25
4.1. Banco de pruebas	25
4.1.1. Diseño del entorno de pruebas	25

4.1.3. Resultados y observaciones	26
4.2. Pruebas de la API REST	27
4.2.1. Objetivos y alcance	27
4.2.2. Metodología de prueba	27
4.2.3. Resultados obtenidos	28
4.3. Pruebas de componentes	29
4.3.1. Enfoque general	29
4.3.2. Resultados por componente	30
4.4. Pruebas del frontend	30
4.4.1. Objetivos	30
4.4.2. Metodología	31
4.4.3. Resultados y observaciones	35
4.5. Prueba final de integración	36
4.5.1. Metodología de la prueba	36
4.5.2. Resultados obtenidos	37
4.6. Comparación con otras soluciones	38
5. Conclusiones	41
5.1. Resultados y metas	41
5.2. Trabajo futuro	42
Bibliografía	43

4.1.2. Metodología experimental	26
4.1.3. Resultados y observaciones	26
4.2. Pruebas de la API REST	26
4.2.1. Objetivos y alcance	27
4.2.2. Metodología de prueba	27
4.2.3. Resultados obtenidos	27
4.2.4. Conclusiones	29
4.3. Pruebas de componentes	29
4.3.1. Enfoque general	29
4.3.2. Resultados por componente	29
4.3.3. Conclusiones	30
4.4. Pruebas del frontend	30
4.4.1. Objetivos	30
4.4.2. Metodología	30
4.4.3. Resultados y observaciones	31
4.5. Prueba final de integración	31
4.5.1. Metodología de la prueba	31
4.5.2. Resultados obtenidos	32
4.5.3. Conclusiones de la integración	33
4.6. Comparación con otras soluciones	33

Bibliografía	35
-------------------------------	-----------

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.	14
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.	19
3.7. Diagrama de estructura de los componentes por pantalla.	22
3.8. Diagrama de flujo de comunicación con el backend.	23
4.1. Fotografía banco de pruebas.	26
4.2. Pantalla de inicio de sesión del frontend.	32
4.3. Panel principal con visualización del listado de dispositivos.	32
4.4. Panel de visualización del dispositivo: datos generales.	33
4.5. Panel de visualización del dispositivo: preparación para la ejecución de un comando.	33
4.6. Panel de visualización del dispositivo: ejecución de un comando.	34
4.7. Panel de visualización del dispositivo: comando ejecutado y la respuesta pendiente.	34
4.8. Panel de visualización del dispositivo: comando ejecutado y su respuesta.	35
4.9. Panel de visualización del historial de mediciones.	35
4.10. Fotografía pruebas finales de integración.	37

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.	14
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.	21
3.8. Diagrama de flujo de comunicación con el backend.	22

Índice de tablas

3.1. Endpoints REST principales	20
4.1. Resultados de pruebas de endpoints REST	29
4.2. Comparación de la solución propuesta	39

Índice de tablas

3.1. Endpoints REST principales	19
4.1. Resultados de pruebas de endpoints REST	28
4.2. Comparación de la solución propuesta	34

Capítulo 3

Diseño e implementación

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

3.1. Arquitectura del sistema

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

El sistema se organiza en cinco bloques con funciones definidas que aseguran un flujo de datos confiable, eficiente y seguro durante la captura, transmisión, procesamiento y visualización de eventos, garantizando trazabilidad, persistencia y control remoto.

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

Capítulo 3

Diseño e implementación

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

3.1. Arquitectura del sistema

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

3.1.1. Descripción ampliada de bloques y responsabilidades

El sistema se organiza en cinco bloques principales, cada uno con responsabilidades claramente definidas para garantizar un flujo de datos confiable, eficiente y seguro. Cada bloque cumple funciones específicas dentro del ciclo de captura, transmisión, procesamiento y visualización de los eventos, que asegura trazabilidad, persistencia y control de comandos remotos. Se describen los bloques y sus responsabilidades:

- Dispositivo de campo: el nodo de campo integra el contador existente (salida RS-232), un microcontrolador ESP32-C3 y un módem GPRS SIM800L. El firmware, desarrollado sobre [ESP-IDF](#), realiza las siguientes funciones: lectura continua de la trama serial, parsing tolerante a ruido, preprocesado (validación, normalización de campos y asignación de sello temporal UTC), encolamiento FIFO de eventos, gestión de reinicios y publicación MQTT cuando hay conectividad. Además, el nodo se suscribe a los tópicos de comandos y publica telemetría y acks. En el nodo se implementa persistencia mínima (registro de comandos pendientes y últimas N tramas) para recuperación tras reinicio.
- Transporte (broker MQTT): el broker actúa como bus de mensajes desacoplado. Se recomienda emplear Eclipse Mosquitto en la etapa inicial y evaluar brokers gestionados para despliegues a mayor escala. El broker gestiona autenticación por credenciales, control de tópicos y cifrado. Se emplean tópicos jerárquicos por dispositivo para facilitar filtrado y autorización:

- 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.

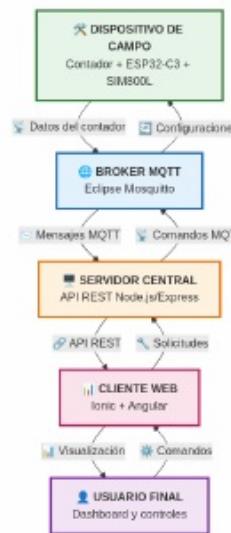


FIGURA 3.1. Diagrama de arquitectura del sistema y el flujo de datos.

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:

- dispositivo/{id}/medicion
 - dispositivo/{id}/comando
 - dispositivo/{id}/respuesta
- Servidor central: el servidor central reúne dos responsabilidades principales:

- Componente suscriptor MQTT que valida, transforma y enruta mensajes hacia la lógica de negocio y la persistencia.
- API REST ofrece servicios de consulta, gestión y comandos, manteniendo independencia del broker para permitir nuevos consumidores. Los datos se almacenan en MySQL con soporte para rangos temporales, alto rendimiento y auditoría de comandos.

- Cliente/Visualización: La interfaz web, desarrollada en Ionic + Angular, consume la API REST para consultas históricas y eventos en tiempo real. Ofrece visualización de eventos, consultas filtradas, envío de comandos y un panel de telemetría para mantenimiento.

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

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

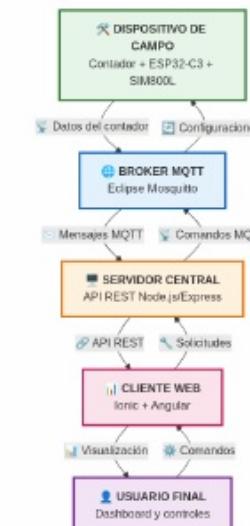


FIGURA 3.1. Diagrama de arquitectura del sistema y el flujo de datos.

3.1. 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.

3.1. Arquitectura del sistema

13

3.1.2. Flujo de datos

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

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

3.3. Desarrollo del backend

17

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 tópico dispositivo/*id*/medición, 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 backend, base de datos y 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.

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

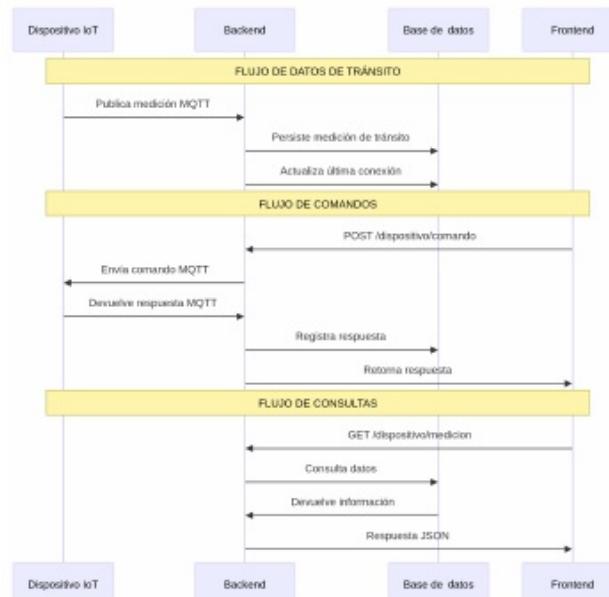


FIGURA 3.5. Diagrama de flujo de información del Backend.

3.3. Desarrollo del backend

17

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 dispositivo/*id*/medición y el backend las valida y guarda en MySQL, mientras que los comandos y respuestas se gestionan en dispositivo/*id*/comando y dispositivo/*id*/respuesta. El despliegue usa Docker Compose [32] para ejecutar backend, base de datos y broker [17] en contenedores, y el registro se maneja con Winston [28] y Morgan [29] para trazabilidad completa.

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

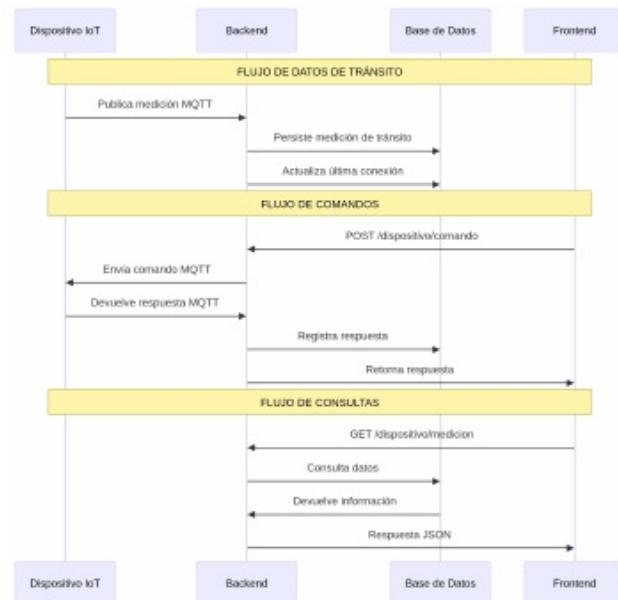


FIGURA 3.5. Diagrama de flujo de información del Backend.

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.

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.

- 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.

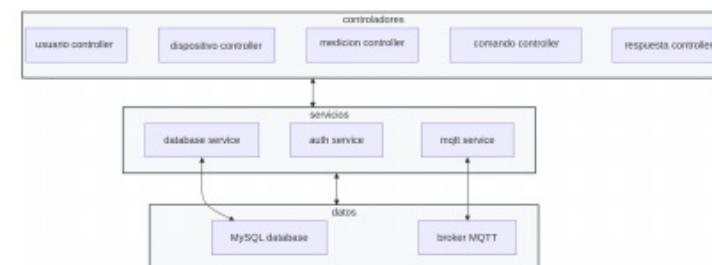


FIGURA 3.6. Diagrama con la disposición de los controladores y flujo de dependencias.

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.

3.3. Desarrollo del backend

19

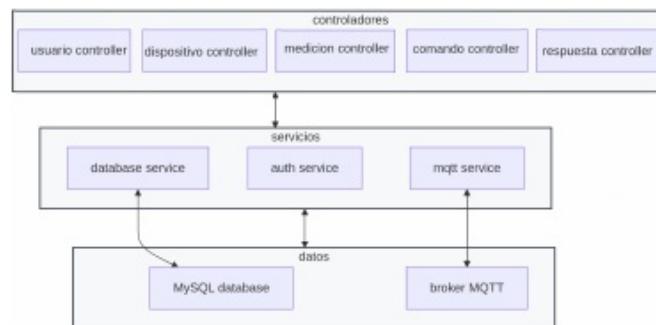


FIGURA 3.6. Diagrama con la disposición de los controladores y flujo de dependencias.

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

A continuación, se presenta la tabla general de los endpoints y controladores más importantes para el flujo:

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

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.

basada en controladores permite extender el backend con nuevos recursos o funcionalidades sin afectar la lógica ya implementada.

3.4. Desarrollo del frontend

El frontend del sistema se diseñó como una *Single Page Application* se desarrolla en Ionic con Angular y TypeScript. El objetivo es proporcionar una interfaz moderna e intuitiva, accesible desde navegador, que permita al operador autenticarse, supervisar en tiempo real los eventos captados por los contadores de tránsito, consultar históricos almacenados en la base de datos y emitir comandos remotos hacia los nodos de campo.

3.4.1. Arquitectura y tecnologías

El cliente web se estructura en componentes reutilizables de Ionic, lo que facilita la navegación y asegura un diseño responsive tanto en entornos de escritorio como móviles. La comunicación con el backend se realiza mediante peticiones HTTP a la API REST, y en casos donde se requiere actualización en tiempo real se emplea un canal de notificación basado en WebSocket.

3.4.2. Funcionalidades principales

El frontend integra las siguientes funciones clave:

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

3.4. Desarrollo del frontend

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

3.4.1. Arquitectura y tecnologías

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

3.4.2. Funcionalidades principales

El frontend integra las siguientes funciones clave:

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

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

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

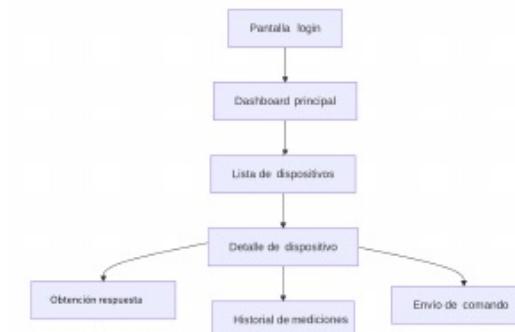


FIGURA 3.7. Diagrama de estructura de los componentes por pantalla.

3.4.3. Integración con el backend

Todas las operaciones del frontend se apoyan en los endpoints REST definidos en el backend (ver Sección 3.1). Cada petición incluye en sus cabeceras el token JWT obtenido en el login, lo que garantiza que solo usuarios autorizados puedan acceder a datos sensibles o emitir comandos. El backend devuelve respuestas en formato JSON, que son interpretadas y representadas en la interfaz en tiempo real, lo que asegura consistencia entre la vista del operador y el estado real de los dispositivos.

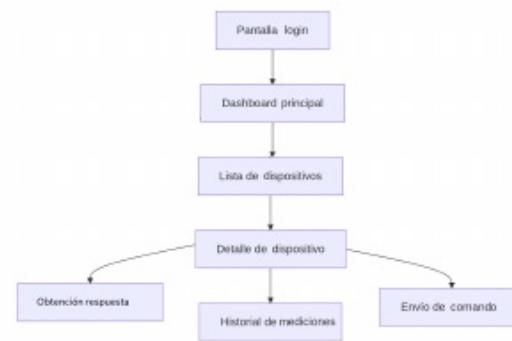


FIGURA 3.7. Diagrama de estructura de los componentes por pantalla.

3.4.3. Integración con el backend

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

En la figura 3.8 se observa el diagrama de flujo de comunicación con el backend.

En la figura 3.8 se observa el diagrama de flujo de comunicación con el backend.

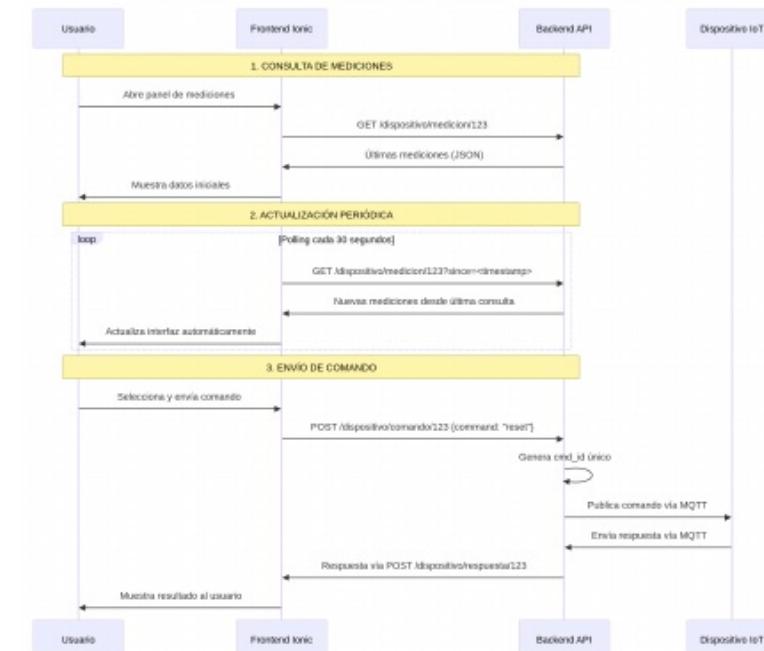


FIGURA 3.8. Diagrama de flujo de comunicación con el backend.

3.5. Despliegue del sistema

23

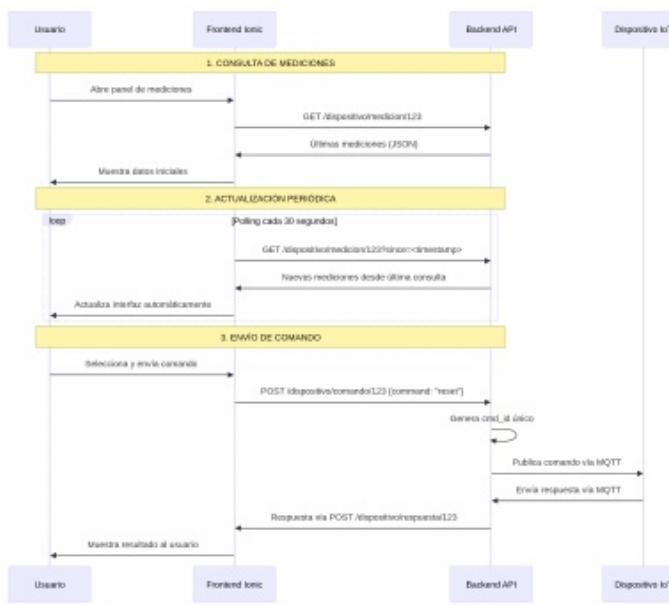


FIGURA 3.8. Diagrama de flujo de comunicación con el backend.

3.5. Despliegue del sistema

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

3.5.1. Entorno productivo e integración continua

El entorno productivo se implementa bajo un esquema de *cloud on-premise*, es decir, una nube privada alojada en los servidores locales de Vialidad Nacional. Este enfoque combina las ventajas de la virtualización y la gestión centralizada propias del entorno *cloud*, con el control, la seguridad y la independencia de un despliegue local.

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

3.5. Despliegue del sistema

23

3.5. Despliegue del sistema

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

3.5.1. Entorno productivo e integración continua

El entorno productivo se implementa bajo un esquema de *cloud on-premise*, es decir, una nube privada alojada en los servidores locales de Vialidad Nacional. Este enfoque combina las ventajas de la virtualización y la gestión centralizada propias del entorno *cloud*, con el control, la seguridad y la independencia de un despliegue local.

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

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

3.5.2. Monitoreo post-implantación

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

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

3.6. Integración con la infraestructura existente

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

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

3.5.2. Monitoreo post-implantación

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

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

3.6. Integración con la infraestructura existente

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

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

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

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

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

Capítulo 4

Ensayos y resultados

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

4.1. Banco de pruebas

El banco de pruebas se diseñó con el propósito de reproducir las condiciones reales de operación del sistema de detección de tránsito. De este modo, se garantizó la validez de los resultados dentro de un entorno controlado. El montaje permitió evaluar la robustez del firmware, la estabilidad de las comunicaciones y la capacidad del backend para procesar eventos en distintos escenarios de conectividad.

El objetivo principal consistió en analizar el comportamiento integral del sistema ante situaciones representativas de campo, que incluyeron la pérdida temporal del enlace GPRS, el almacenamiento local de eventos y la recuperación automática una vez restablecida la conexión.

4.1.1. Diseño del entorno de pruebas

El banco se compuso de los siguientes elementos principales:

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

La figura 4.1 se muestra el banco de pruebas utilizado.

Capítulo 4

Ensayos y Resultados

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

4.1. Banco de pruebas

El banco de pruebas se diseñó con el propósito de reproducir las condiciones reales de operación del sistema de detección de tránsito. De este modo, se garantizó la validez de los resultados dentro de un entorno controlado. El montaje permitió evaluar la robustez del firmware, la estabilidad de las comunicaciones y la capacidad del backend para procesar eventos en distintos escenarios de conectividad.

El objetivo principal consistió en analizar el comportamiento integral del sistema ante situaciones representativas de campo, que incluyeron la pérdida temporal del enlace GPRS, el almacenamiento local de eventos y la recuperación automática una vez restablecida la conexión.

4.1.1. Diseño del entorno de pruebas

El banco se compuso de los siguientes elementos principales:

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

El montaje permitió reproducir tres escenarios de prueba diferenciados:

1. Conectividad estable: transmisión continua sin pérdidas de enlace.



FIGURA 4.1. Fotografía banco de pruebas.

El montaje permitió reproducir tres escenarios de prueba diferenciados:

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

4.1.2. Metodología experimental

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

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

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

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

4.1.3. Resultados y observaciones

Los resultados experimentales demostraron que el sistema fue capaz de:

- Mantener la integridad de los datos en escenarios de conectividad inestable.

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

4.1.2. Metodología experimental

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

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

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

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

4.1.3. Resultados y observaciones

Los resultados experimentales demostraron que el sistema fue capaz de:

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

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

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

4.2. Pruebas de la API REST

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

- Mantener la integridad de los datos en escenarios de conectividad inestable.

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

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

4.2. Pruebas de la API REST

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

4.2.1. Objetivos y alcance

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

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

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

4.2.2. Metodología de prueba

El proceso de validación se realizó con la herramienta Postman [37]. Se elaboraron colecciones de solicitudes y scripts de prueba en la pestaña Tests, que verificaron los códigos de estado HTTP, la estructura de las respuestas y el contenido de los mensajes.

El Collection Runner [38] permitió ejecutar los casos de prueba en distintos entornos: desarrollo local, red simulada GPRS e integración con el broker MQTT. Los resultados se exportaron en formato JSON y se analizaron mediante la extensión Newman [39].

integridad, la seguridad y el rendimiento de las operaciones ofrecidas por los endpoints.

4.2.1. Objetivos y alcance

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

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

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

4.2.2. Metodología de prueba

El proceso de validación se realizó con la herramienta Postman [37]. Se elaboraron colecciones de solicitudes y scripts de prueba en la pestaña Tests, que verificaron los códigos de estado HTTP, la estructura de las respuestas y el contenido de los mensajes.

El Collection Runner [38] permitió ejecutar los casos de prueba en distintos entornos: desarrollo local, red simulada GPRS e integración con el broker MQTT. Los resultados se exportaron en formato JSON y se analizaron mediante la extensión Newman [39].

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

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

4.2.3. Resultados obtenidos

Las pruebas confirmaron la estabilidad y solidez de la API REST. Todas las operaciones CRUD se ejecutaron correctamente y devolvieron respuestas en formato JSON con los códigos HTTP apropiados.

resultados se exportaron en formato JSON y se analizaron mediante la extensión Newman [39].

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

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

4.2.3. Resultados obtenidos

Las pruebas confirmaron la estabilidad y solidez de la API REST. Todas las operaciones CRUD se ejecutaron correctamente y devolvieron respuestas en formato JSON con los códigos HTTP apropiados.

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

A continuación, se presenta la tabla 4.1 con los resultados de las pruebas de los endpoints REST:

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

A continuación, se presenta la tabla de los resultados de pruebas de endpoints REST:

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

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

4.3. Pruebas de componentes

29

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:

4.3. Pruebas de componentes

29

4.2.4. Conclusiones

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

4.3. Pruebas de componentes

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

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

4.3.1. Enfoque general

El sistema se evaluó bajo un esquema progresivo:

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

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

4.3.2. Resultados por componente

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

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

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

4.3.2. Resultados por componente

Cada módulo del sistema fue evaluado de forma independiente para verificar su funcionamiento, la integridad de los datos y la robustez ante fallos de conexión. Se resumen los principales resultados obtenidos en las pruebas de cada componente:

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

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

4.4. Pruebas del frontend

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

4.4.1. Objetivos

Los objetivos específicos de esta etapa fueron los siguientes:

- Verificar la compatibilidad del frontend con los navegadores más utilizados (Chrome, Firefox) y con dispositivos móviles Android.
- Evaluar el rendimiento general de la aplicación: tiempos de carga, latencia en consultas a la API y velocidad de actualización de los gráficos.

4.3.3. Conclusiones

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

4.4. Pruebas del frontend

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

4.4.1. Objetivos

Los objetivos específicos de esta etapa fueron los siguientes:

- Verificar la compatibilidad del frontend con los navegadores más utilizados (Chrome, Firefox) y con dispositivos móviles Android.
- Evaluar el rendimiento general de la aplicación: tiempos de carga, latencia en consultas a la API y velocidad de actualización de los gráficos.
- Analizar la usabilidad de la interfaz mediante pruebas con usuarios: oportunidades de mejora en la disposición de elementos visuales y en los flujos de interacción.
- Validar la correcta ejecución de comandos y confirmaciones visuales en tiempo real a través de la comunicación con el broker MQTT y la API REST.
- Comprobar el manejo de errores de conexión y autenticación, así se generen mensajes claros y retroalimentación inmediata al usuario.

4.4.2. Metodología

Las pruebas se realizaron sobre la versión estable del frontend, desarrollado con los frameworks Ionic y Angular, y desplegado en un entorno controlado junto al backend y el broker MQTT. Se llevaron a cabo ensayos funcionales, de compatibilidad y de rendimiento, que combinó herramientas automáticas y observación directa de la interacción del usuario.

- Compatibilidad y visualización: se validó la correcta visualización de los componentes en distintas resoluciones y navegadores, utilizando *Chrome DevTools* [40] y el modo responsive de Ionic. El diseño adaptativo permitió mantener la legibilidad de los gráficos y menús tanto en pantallas de escritorio como en dispositivos móviles. Las pruebas demostraron una compatibilidad completa con los navegadores modernos, presentando solo ligeras diferencias en el renderizado de iconos SVG en Firefox.
- Rendimiento: el análisis de desempeño se realizó con *Lighthouse* [41] y el monitor de red de los navegadores. El tiempo promedio de carga inicial fue de 2,3 segundos en Chrome y 2,7 segundos en Firefox. En dispositivos móviles Android, el tiempo de carga fue de 3,8 segundos, debido a la menor capacidad de procesamiento. La latencia promedio de las consultas REST se

4.4. Pruebas del frontend

31

- Analizar la usabilidad de la interfaz mediante pruebas con usuarios: oportunidades de mejora en la disposición de elementos visuales y en los flujos de interacción.
- Validar la correcta ejecución de comandos y confirmaciones visuales en tiempo real a través de la comunicación con el broker MQTT y la API REST.
- Comprobar el manejo de errores de conexión y autenticación, así se generen mensajes claros y retroalimentación inmediata al usuario.

4.4.2. Metodología

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

- Compatibilidad y visualización: se validó la correcta visualización de los componentes en distintas resoluciones y navegadores, utilizando Chrome DevTools [40] y el modo responsive de Ionic. El diseño adaptativo permitió mantener la legibilidad de los gráficos y menús tanto en pantallas de escritorio como en dispositivos móviles. Las pruebas demostraron una compatibilidad completa con los navegadores modernos, presentando solo ligeras diferencias en el renderizado de íconos SVG en Firefox.
- Rendimiento: el análisis de desempeño se realizó con Lighthouse [41] y el monitor de red de los navegadores. El tiempo promedio de carga inicial fue de 2,3 segundos en Chrome y 2,7 segundos en Firefox. En dispositivos móviles Android, el tiempo de carga fue de 3,8 segundos, debido a la menor capacidad de procesamiento. La latencia promedio de las consultas REST se mantuvo por debajo de los 250 milisegundos en red local, que aumentó a 600 milisegundos bajo simulación GPRS.
- Usabilidad: se realizaron pruebas con un grupo reducido de usuarios familiarizados con sistemas de monitoreo vial, quienes interactuaron con la interfaz durante sesiones controladas. Los resultados indicaron una alta comprensión del flujo de navegación y una percepción positiva de la organización visual. Las observaciones fueron incorporadas en una versión posterior mediante ajustes de color, iconografía y jerarquía visual.
- Comunicación y validación funcional: se comprobó que los comandos emitidos desde la interfaz se transmitieran correctamente al backend y se visualizaran sus estados en tiempo real. Del mismo modo, los eventos de tránsito publicados por los dispositivos se reflejaron de forma inmediata en la aplicación, sin inconsistencias ni retrasos notables. Las alertas visuales ante pérdida de conexión, errores de autenticación o respuestas HTTP inválidas funcionaron según lo esperado, mostrando mensajes informativos y acciones de recuperación.

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

La figura 4.2 muestra la pantalla de inicio de sesión, donde el usuario debe ingresar sus credenciales para acceder al sistema.

4.5. Prueba final de integración

31

mantuvo por debajo de los 250 milisegundos en red local, que aumentó a 600 milisegundos bajo simulación GPRS.

- Usabilidad: se realizaron pruebas con un grupo reducido de usuarios familiarizados con sistemas de monitoreo vial, quienes interactuaron con la interfaz durante sesiones controladas. Los resultados indicaron una alta comprensión del flujo de navegación y una percepción positiva de la organización visual. Las observaciones fueron incorporadas en una versión posterior mediante ajustes de color, iconografía y jerarquía visual.
- Comunicación y validación funcional: se comprobó que los comandos emitidos desde la interfaz se transmitieran correctamente al backend y se visualizaran sus estados en tiempo real. Del mismo modo, los eventos de tránsito publicados por los dispositivos se reflejaron de forma inmediata en la aplicación, sin inconsistencias ni retrasos notables. Las alertas visuales ante pérdida de conexión, errores de autenticación o respuestas HTTP inválidas funcionaron según lo esperado, mostrando mensajes informativos y acciones de recuperación.

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.

4.5.1. Metodología de la prueba

Para la validación end-to-end se configuró un entorno de ensayo en el que participaron todos los subsistemas desplegados simultáneamente:



FIGURA 4.2. Pantalla de inicio de sesión del frontend.

Una vez autenticado, el usuario accede al panel principal mostrado en la figura 4.3, donde se presenta el listado de dispositivos registrados junto con su estado de conexión y otras variables relevantes.

Listado de Dispositivos	
contador 1	<button>VTR</button> >
Ubicación: Pabellón Buleo Tipo: DTSC	
contador 2	<button>VTR</button> >
Ubicación: Tres Arroyos Tipo: PADR	
contador 3	<button>VTR</button> >
Ubicación: Gobernador Tipo: DTSC	
contador 4	<button>VTR</button> >
Ubicación: Peltón Tipo: DTSC_ORD	

FIGURA 4.3. Panel principal con visualización del listado de dispositivos.

Al seleccionar un dispositivo del listado, el sistema despliega un panel detallado con la información específica de dicho equipo. La figura 4.4 ilustra la primera vista del panel, donde se muestran los datos generales de identificación, el último comando enviado, su respuesta y la última medición.

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

El flujo de integración se estructuró de la siguiente manera:

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

4.5.2. Resultados obtenidos

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

En particular, se observaron los siguientes aspectos destacados:

- Interoperabilidad completa: todos los componentes del sistema hardware, middleware y software interactuaron sin incompatibilidades ni pérdidas de información.
- Sincronización en tiempo real: la actualización de la interfaz web frente a la recepción de un nuevo evento.
- Confiabilidad del flujo de comandos: las órdenes enviadas desde el frontend fueron recibidas y ejecutadas correctamente por el nodo, con confirmaciones visibles en pantalla.
- Tiempos de ida y vuelta (round-trip): entre 3 y 5 segundos en condiciones normales de red GPRS, y hasta 12 segundos bajo conectividad inestable, sin pérdida de comandos ni duplicación de respuestas.

4.4. Pruebas del frontend

3

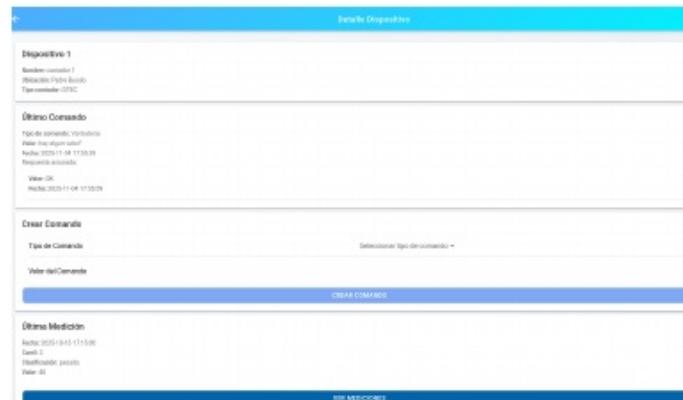


FIGURA 4.4. Panel de visualización del dispositivo: datos generales.

La figura 4.5 se muestra la preparación para la ejecución de un comando.

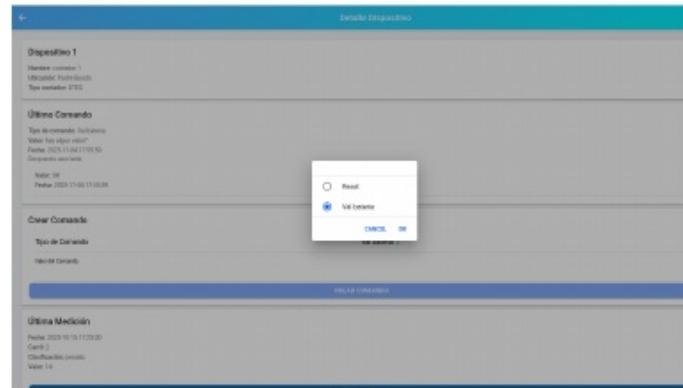


FIGURA 4.5. Panel de visualización del dispositivo: preparación para la ejecución de un comando.

En la figura 4.6 se habilita para la ejecución del comando

4.6. Comparación con otras soluciones

33

- Trazabilidad completa: cada evento quedó registrado en la base de datos con su respectivo timestamp e id de dispositivo.

4.5.3. Conclusiones de la integración

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

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

4.6. Comparación con otras soluciones

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

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

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

La solución desarrollada combina bajo costo, independencia tecnológica y arquitectura modular basada en estándares abiertos (MQTT, REST, JSON), permitiendo rápida adaptación a entornos con conectividad variable, como rutas argentinas, sin depender de infraestructura propietaria ni servicios móviles externos. A continuación, se presenta la tabla de comparación de la solución propuesta frente a alternativas comerciales y académicas:



FIGURA 4.6. Panel de visualización del dispositivo: ejecución de un comando.

La figura 4.7 muestra el comando ejecutado y la respuesta pendiente.

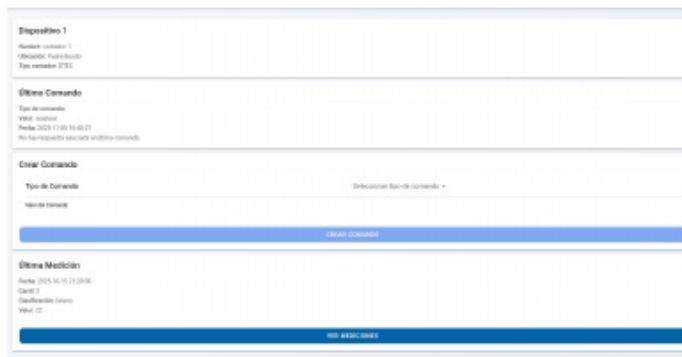


FIGURA 4.7. Panel de visualización del dispositivo: comando ejecutado y la respuesta pendiente.

La figura 4.8 se muestra la respuesta al comando.

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

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

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

4.4. Pruebas del frontend

35



FIGURA 4.8. Panel de visualización del dispositivo: comando ejecutado y su respuesta.

Después, en la figura 4.9 se presenta el historial de mediciones, con paginación y exportación a csv.

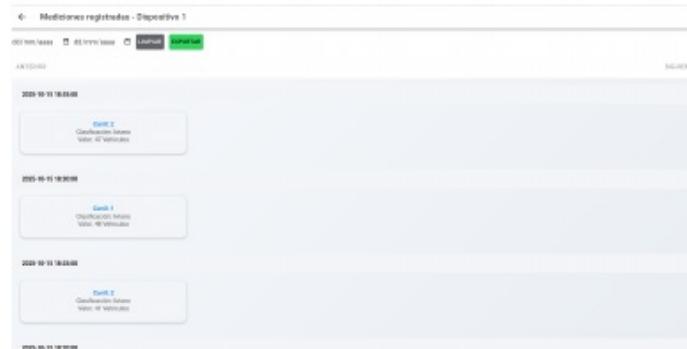


FIGURA 4.9. Panel de visualización del historial de mediciones.

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.

35

Bibliografía

- [1] Juan Asiaín et al. «LoRa-Based Traffic Flow Detection for Smart-Road». En: *Sensors* 21.2 (2021), pág. 338. DOI: [10.3390/s21020338](https://doi.org/10.3390/s21020338). URL: <https://www.mdpi.com/1424-8220/21/2/338>.
- [2] Jan Micko et al. «Review of IoT Sensor Systems Used for Monitoring the Road Infrastructure». En: *Sensors* 23.9 (2023), pág. 4469. DOI: [10.3390/s23094469](https://doi.org/10.3390/s23094469). URL: <https://www.mdpi.com/1424-8220/23/9/4469>.
- [3] Gianluca Peruzzi et al. «Combining LoRaWAN and NB-IoT for Edge-to-Cloud Low-Power Connectivity». En: *Applied Sciences* 12.3 (2022), pág. 1497. DOI: [10.3390/app12031497](https://doi.org/10.3390/app12031497). URL: <https://www.mdpi.com/2076-3417/12/3/1497>.
- [4] Miovision. *TrafficLink / Managed Connectivity*. <https://www.miovision.com/trafficlink/>. Soluciones comerciales. 2023.
- [5] Sensys Networks. *Documentación técnica y productos*. <https://www.sensysnetworks.com/>. 2023.
- [6] MetroCount. *Contadores y guías técnicas*. <https://www.metrocount.com/>. 2023.
- [7] Exemys. *Managed Connectivity*. Accedido: 16-Sep-2025. 2025, URL: <https://www.exemys.com/site/index.shtml>.
- [8] Digi International. *Digi Remote Manager: IoT Device Monitoring and Management Solution*. <https://www.digi.com/products/iot-software-services/digi-remote-manager>. Accedido: 11 de septiembre de 2025. 2025.
- [9] A. A. Sukmandhani, M. Zarlis y Nurudin. «Monitoring Applications for Vehicle based on Internet of Things (IoT) using the MQTT Protocol». En: *BINUS Conference Proceedings*. Accedido: 11 de septiembre de 2025. 2023. URL: <https://research.binus.ac.id/publication/C1B25545-66P9-49C3-B873-D6C537EA23B3/monitoring-applications-for-vehicle-based-on-internet-of-things-iot-using-the-mqtt-protocol/>.
- [10] S. Bharath y C. Khusi. «IoT Based Smart Traffic System Using MQTT Protocol: Node-Red Framework». En: *2nd Global Conference for Advancement in Technology (GCAT)*. Accedido: 11 de septiembre de 2025. 2021. DOI: [10.1109/GCAT5128.2021.9587636](https://doi.org/10.1109/GCAT5128.2021.9587636).
- [11] Zuoling Niu. «Research and Implementation of Internet of Things Communication System Based on MQTT Protocol». En: *Journal of Physics: Conference Series* 012019 (2023). Accedido: 11 de septiembre de 2025.
- [12] OpenRemote. *OpenRemote: 100 % Open Source IoT Device Management Platform*. <https://openremote.io/>. Accedido: 11 de septiembre de 2025. 2025.
- [13] Espressif Systems. *ESP32-C3 Series Datasheet*. Accedido: 23-Sep-2025. Espressif Systems. 2021. URL: https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf.
- [14] Analog Devices. *Fundamentals of RS-232 Serial Communications*. <https://www.analog.com/en/resources/technical-articles/fundamentals-of-rs232-serial-communications.html>. Accedido: 11-Sep-2025. 2020.

- 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.

4.5.1. Metodología de la prueba

Para la validación end-to-end se configuró un entorno de ensayo en el que participaron todos los subsistemas desplegados simultáneamente. El flujo de integración se estructuró de la siguiente manera:

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

La figura 4.10 se muestra la ejecución de las prueba de integración.

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

4.5. Prueba final de integración

37

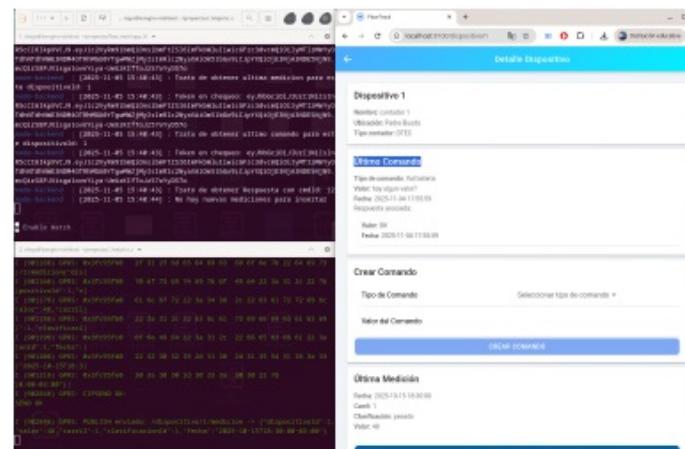


FIGURA 4.10. Fotografía pruebas finales de integración.

En el video 4.1 se muestra la prueba completa del funcionamiento del sistema, que verifica la integración de los distintos componentes, la comunicación entre módulos y la operación general.

Video 4.1: Prueba integral del sistema en funcionamiento.
<https://youtu.be/T4vcsNNGdc8?si=QBCrA6GHNZckLFO9>

4.5.2. Resultados obtenidos

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

En particular, se observaron los siguientes aspectos destacados:

- Interoperabilidad completa: todos los componentes del sistema hardware, middleware y software interactuaron sin incompatibilidades ni pérdidas de información.
- Sincronización en tiempo real: la actualización de la interfaz web frente a la recepción de un nuevo evento.
- Confiabilidad del flujo de comandos: las órdenes enviadas desde el frontend fueron recibidas y ejecutadas correctamente por el nodo, con confirmaciones visibles en pantalla.
- Tiempos de ida y vuelta (round-trip): entre 3 y 5 segundos en condiciones normales de red GPRS, y hasta 12 segundos bajo conectividad inestable, sin pérdida de comandos ni duplicación de respuestas.
- Trazabilidad completa: cada evento quedó registrado en la base de datos con su respectivo timestamp e id de dispositivo.

Bibliografía

37

- [36] M. Jones, J. Bradley y N. Sakimura. *JSON Web Token (JWT)*. RFC 7519. Internet Engineering Task Force (IETF). 2015. URL: <https://datatracker.ietf.org/doc/html/rfc7519>.
- [37] Postman, Inc. *Postman API Platform*. Herramienta para pruebas y automatización de APIs REST. 2025. URL: <https://www.postman.com/>.
- [38] Postman, Inc. *Postman Collection Runner and Newman CLI*. Herramienta de Postman para ejecutar pruebas automatizadas de colecciones de API. 2025. URL: <https://learning.postman.com/docs/collections/running-collections/intro-to-collection-runs/>.
- [39] Inc. Postman. *Newman: Command-line Collection Runner for Postman*. <https://www.npmjs.com/package/newman>. Último acceso: 31 de octubre de 2025. 2025.
- [40] Google Developers. *Chrome DevTools*. Accedido: octubre 2025. 2024. URL: <https://developer.chrome.com/docs/devtools/>.
- [41] Google Developers. *Google Lighthouse*. Accedido: octubre 2025. 2024. URL: <https://developer.chrome.com/docs/lighthouse/>.
- [42] FHWA. *Traffic Monitoring Guide*. Inf. téc. Federal Highway Administration, 2022. URL: https://www.fhwa.dot.gov/policyinformation/tmguide/2022_TMG_Final_Report.pdf.
- [43] FHWA. *Traffic Detector Handbook, 3rd Edition*. Inf. téc. Federal Highway Administration, 2006. URL: <https://www.fhwa.dot.gov/publications/research/operations/its/06108/06108.pdf>.