

| | | |
|---------------------|--|-----------|
| 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 | 37 |
| 4.3.1. | Enfoque general | 37 |
| 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 |

| | | |
|---------------------|--|-----------|
| 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 | 37 |
| 4.3.1. | Enfoque general | 37 |
| 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 | 44 |
| 4.5.1. | Metodología de la prueba | 44 |
| 4.5.2. | Resultados obtenidos | 49 |
| 4.6. | Comparación con otras soluciones | 50 |
| 5. | Conclusiones | 53 |
| 5.1. | Resultados y metas | 53 |
| 5.2. | Trabajo futuro | 54 |
| Bibliografía | | 55 |

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. | 32 |
| 4.4. Solicitud POST /api/medicion con todos los campos completos. | 34 |
| 4.5. Solicitud POST /api/medicion con campos faltantes. | 34 |
| 4.6. Solicitud POST /api/comando exitosa. | 35 |
| 4.7. Solicitud POST /api/comando con error por datos incompletos. | 35 |
| 4.8. Solicitud POST /api/respuesta exitosa. | 35 |
| 4.9. Solicitud POST /api/respuesta con error de validación. | 36 |
| 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. | 32 |
| 4.4. Solicitud POST /api/medicion con todos los campos completos. | 34 |
| 4.5. Solicitud POST /api/medicion con campos faltantes. | 34 |
| 4.6. Solicitud POST /api/comando exitosa. | 35 |
| 4.7. Solicitud POST /api/comando con error por datos incompletos. | 35 |
| 4.8. Solicitud POST /api/respuesta exitosa. | 35 |
| 4.9. Solicitud POST /api/respuesta con error de validación. | 36 |
| 4.10. Interacción entre el nodo de campo (ESP32-C3 + SIM800L) y el backend. | 38 |
| 4.11. Pantalla de inicio de sesión del frontend. | 40 |
| 4.12. Panel principal con visualización del listado de dispositivos. | 40 |
| 4.13. Panel de visualización del dispositivo. | 41 |
| 4.14. Panel de visualización del dispositivo: preparación para la ejecución de un comando y su ejecución. | 41 |
| 4.15. Panel de visualización del dispositivo: ejecución de un comando. | 42 |
| 4.16. Panel de visualización del dispositivo: comando ejecutado y la respuesta pendiente. | 42 |
| 4.17. Panel de visualización del dispositivo: comando ejecutado y su respuesta. | 42 |
| 4.18. Panel de visualización del historial de mediciones. | 43 |

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

| | |
|---|----|
| 4.19. Publicación de una medición por parte del módulo ESP32-C3 y recepción en el backend. | 45 |
| 4.20. Visualización de la medición recibida en la interfaz web del sistema. | 46 |
| 4.21. Generación de un comando desde la interfaz web. | 46 |
| 4.22. Visualización consola del módulo ESP32-C3 que registra la recepción del comando. | 47 |
| 4.23. Intercambio de mensajes entre el ESP32-C3 y el backend durante la recepción y procesamiento de una respuesta. | 48 |
| 4.24. Visualización de la respuesta del dispositivo en la interfaz web tras el procesamiento exitoso en el backend. | 49 |

Índice de tablas

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

Índice de tablas

| | |
|--|----|
| 3.1. Endpoints REST principales | 19 |
| 4.1. Resultados de pruebas de endpoints REST | 36 |
| 4.2. Comparación de la solución propuesta | 51 |

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.

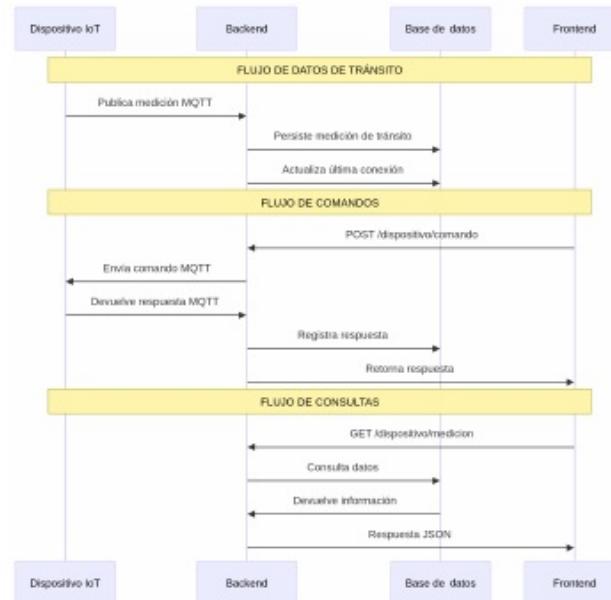


FIGURA 3.5. Diagrama de flujo de información del backend.

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. 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.

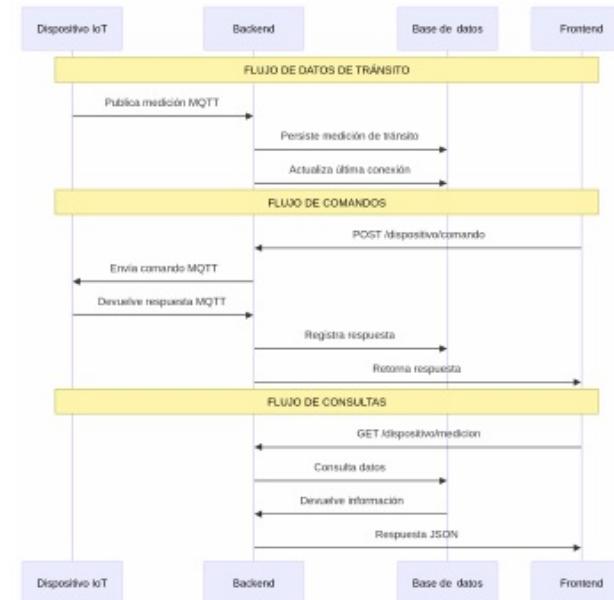


FIGURA 3.5. Diagrama de flujo de información del Backend.

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 ingesta de eventos de tránsito, validación de payloads y persistencia en la base de datos.
- ComandoController: administra la emisión y seguimiento de comandos remotos, generando un `cmd_id` único.
- RespuestaController: centraliza la recepción de estados y telemetría (batería, conectividad), en respuesta al comando que se envía, esto garantiza que la base de datos refleje la situación en tiempo real.
- UserController: implementa el ciclo de vida de usuarios y la autenticación mediante [JWT \[36\]](#), 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.

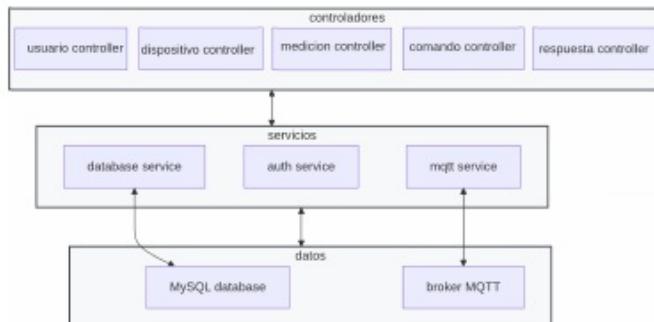


FIGURA 3.6. Diagrama con la disposición de los controladores y flujo de dependencias.

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 ingesta de eventos de tránsito, validación de payloads y persistencia en la base de datos.
- ComandoController: administra la emisión y seguimiento de comandos remotos, generando un `cmd_id` único.
- RespuestaController: centraliza la recepción de estados y telemetría (batería, conectividad), en respuesta al comando que se envía, esto garantiza que la base de datos refleje la situación en tiempo real.
- UserController: implementa el ciclo de vida de usuarios y la autenticación mediante [JWT \[36\]](#), 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.

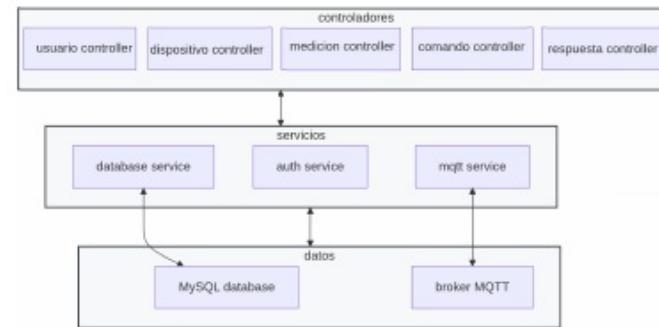


FIGURA 3.6. Diagrama con la disposición de los controladores y flujo de dependencias.

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.

3.4.3. 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.

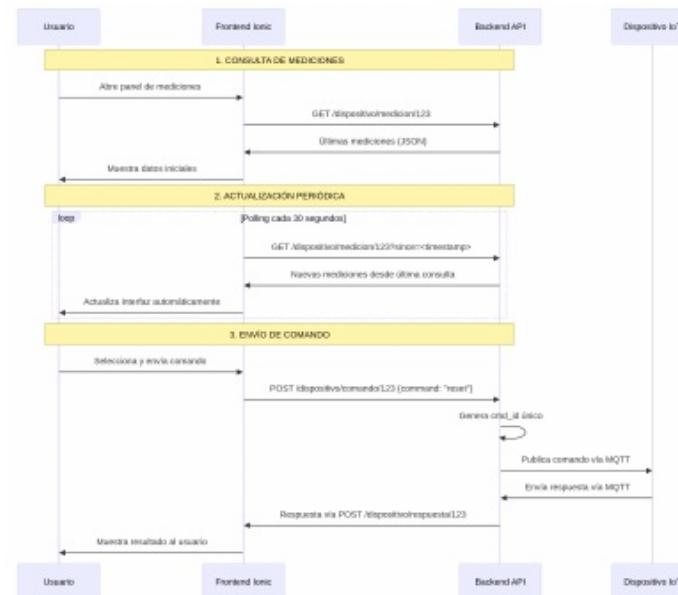


FIGURA 3.8. Diagrama de flujo de comunicación con el backend.

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.

3.4.3. 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.

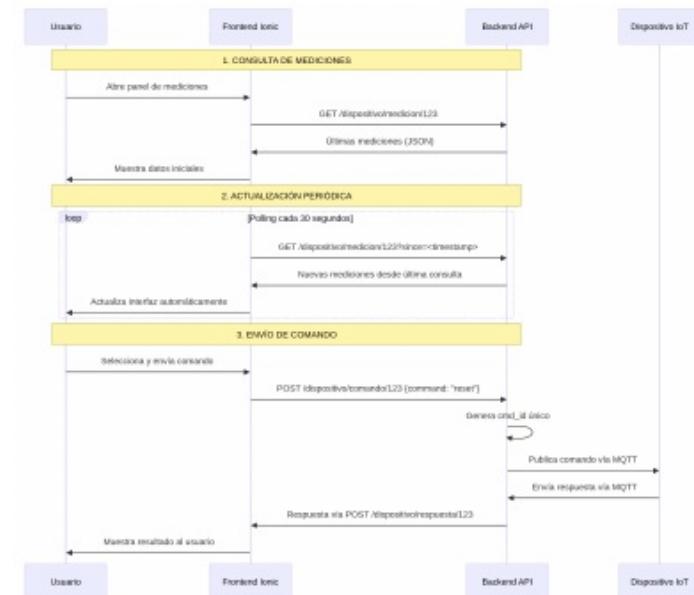


FIGURA 3.8. Diagrama de flujo de comunicación con el backend.

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 C utilizando el framework ESP-IDF, con FreeRTOS como sistema operativo de tiempo real.

3.6. Despliegue del sistema

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

3.6.1. Entorno productivo e integración continua

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

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

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

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

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

3.6. Despliegue del sistema

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

3.6.1. Entorno productivo e integración continua

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

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

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

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

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

3.6. Despliegue del sistema

25

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

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

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

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

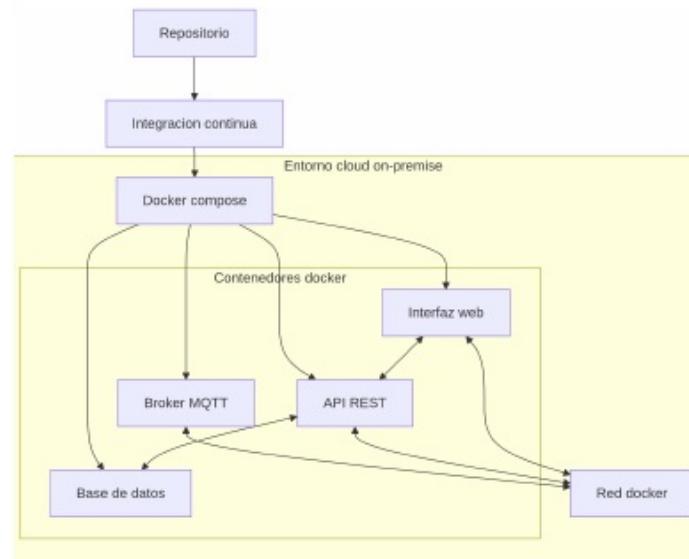


FIGURA 3.10. Arquitectura de despliegue del sistema en entorno *cloud on-premise*.

3.6. Despliegue del sistema

25

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

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

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

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

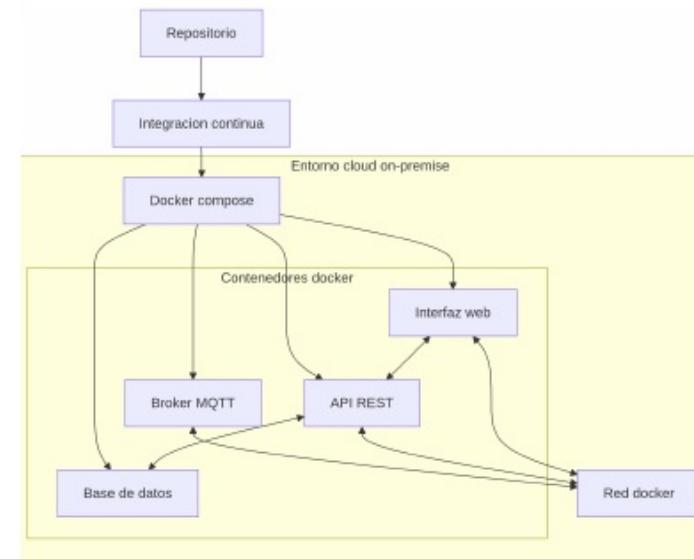


FIGURA 3.10. Arquitectura de despliegue del sistema en entorno *cloud on-premise*

3.6.2. Monitoreo post-implantación

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

- Logs centralizados: el backend y el broker MQTT generan registros detallados de los eventos que procesa cada módulo. Estos registros se almacenan en archivos y también aparecen en la consola de ejecución, lo que permite un análisis inmediato ante incidentes. La solución incluye la integración con **Grafana** [38], que permite correlacionar los registros del sistema con métricas temporales de los servidores y los contenedores. De esta forma, el operador puede identificar patrones, detectar comportamientos anómalos y reconstruir la secuencia exacta de un problema sin necesidad de recurrir a inspecciones manuales dispersas.
- Alertas y métricas: la plataforma Grafana reúne indicadores provenientes del sistema operativo, de los contenedores y del broker MQTT. Entre los valores más relevantes se encuentran la utilización de CPU, el consumo de memoria, la ocupación del disco y la disponibilidad de los servicios. El panel también muestra información vinculada al tráfico MQTT, como la cantidad de mensajes publicados, la frecuencia de publicaciones, los tiempos de respuesta y las posibles pérdidas durante la transmisión. Estas métricas permiten evaluar la carga del sistema y anticipar situaciones de saturación o degradación antes de que afecten a los usuarios finales.
- Supervisión de dispositivos: la API REST del sistema expone endpoints específicos que informan el estado de los dispositivos instalados en campo. El frontend transforma estos datos en un panel claro y accesible que actúa como tablero de salud. Gracias a esta visualización, el personal técnico identifica dispositivos inactivos, comportamientos inusuales o desvíos respecto del funcionamiento esperado, lo 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

3.6.2. 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. Cada uno de estos mecanismos contribuye 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**, 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

4.1. Banco de pruebas

31

4.1.3. Resultados y observaciones

Los resultados experimentales demostraron que el sistema fue capaz de:

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

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

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

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

En la figura 4.3 se presenta un ejemplo del intercambio de mensajes entre el servidor y el nodo de campo.

En este caso, el backend envía un comando remoto al dispositivo mediante una publicación en el tópico MQTT dispositivo/id/comando. El nodo recibe ese comando, lo procesa localmente y responde a través del dispositivo/id/response.

El mensaje de respuesta incluye el campo de cmdId y el valor asociado al comando ejecutado, lo que permite verificar el funcionamiento bidireccional del sistema.

4.1. Banco de pruebas

31

4.1.3. Resultados y observaciones

Los resultados experimentales demostraron que el sistema fue capaz de:

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

Se presentan a continuación ejemplos representativos de los resultados obtenidos durante la prueba. En la figura 4.2 se muestra la tabla Medicion en phpMyAdmin [37], con los registros almacenados en la base de datos MySQL. Cada uno es una detección vehicular procesada por el sistema, con su fecha, valor, carril y clasificación. Esto evidencia que el backend recibió y registró correctamente las mediciones enviadas por el nodo, incluso tras interrupciones de red.

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

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

En la figura 4.3 se presenta un ejemplo del intercambio de mensajes entre el servidor y el nodo de campo.

En este caso, el backend envía un comando remoto al dispositivo mediante una publicación en el tópico MQTT dispositivo/id/comando. El nodo recibe ese comando, lo procesa localmente y responde a través del dispositivo/id/response.

El mensaje de respuesta incluye el campo de cmdId y el valor asociado al comando ejecutado, lo que permite verificar el funcionamiento bidireccional del sistema.

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

| SELECT * FROM `Respuesta` | | | | | |
|---|---------------------|-------|-------|---------------|---------------------|
| <input type="checkbox"/> Perfilando Editar en línea [Editar] Explicar SQL Crear código PHP [Actualizar] | | | | | |
| respId | fecha | cmdId | valor | dispositivoId | createdAt |
| 7 | 2025-11-06 12:03:50 | 16 | OK | 1 | 2025-11-06 12:06:00 |

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

4.2. Pruebas de la API REST

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

4.2.1. Objetivos y alcance

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

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

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

4.2.2. Metodología de prueba

El proceso de validación se realizó con la herramienta Postman [40]. Se elaboraron colecciones de solicitudes y scripts de prueba en la pestaña Tests, que verificaron

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

| SELECT * FROM `Respuesta` | | | | | |
|---|---------------------|-------|-------|---------------|---------------------|
| <input type="checkbox"/> Perfilando Editar en línea [Editar] Explicar SQL Crear código PHP [Actualizar] | | | | | |
| respId | fecha | cmdId | valor | dispositivoId | createdAt |
| 7 | 2025-11-06 12:03:50 | 16 | OK | 1 | 2025-11-06 12:06:00 |

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

4.2. Pruebas de la API REST

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

4.2.1. Objetivos y alcance

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

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

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

4.2.2. Metodología de prueba

El proceso de validación se realizó con la herramienta Postman [38]. Se elaboraron colecciones de solicitudes y scripts de prueba en la pestaña Tests, que verificaron

los códigos de estado HTTP, la estructura de las respuestas y el contenido de los mensajes.

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

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

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

4.2.3. Resultados obtenidos

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

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

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

Medición

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

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

los códigos de estado HTTP, la estructura de las respuestas y el contenido de los mensajes.

El Collection Runner [39] 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 [40].

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

Medición

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

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

```
t diego@banghe-validated:~/proyectos/Nejimir.c Log módulo ESP32C3
I (34806) GPRS: 0x3fc95f80 2f 31 2f 6d 65 64 69 63 69 6f 6e 7b 22 64 69 73 ||/1/m
edicion('dis')
I (34806) GPRS: 0x3fc95f90 76 6f 73 69 74 69 76 6f 49 64 22 3e 31 2c 22 76 |posi
tivoId":1,"v|
I (34816) GPRS: 0x3fc95fa0 61 6c 6f 72 22 3a 33 33 2c 22 03 61 72 72 69 6c |aler
t":33,"carril|
I (34826) GPRS: 0x3fc95fb0 22 3a 32 2c 22 63 6c 61 73 69 66 69 65 61 68 69 ||":2,
"clasificaci1|
I (34836) GPRS: 0x3fc95fc0 6f 6e 49 64 22 3a 31 2c 22 66 65 63 68 61 22 3a |onId
":1,"fecha":|
I (34846) GPRS: 0x3fc95fdb 22 32 30 32 35 2d 31 30 2d 31 35 34 31 37 3a 38 ||"202
5-10-15T17:0|
T (34856) GPRS: 0x3fc95fe0 38 3a 30 30 2d 30 33 3a 30 30 22 7d ||@:00
-03:00"|
I (36536) GPRS: CIPSEND OK:
SEND OK

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



```
2 diego@banghe-validated:~/proyectos/flow-track/app_8 Log backend
node-backend | [2025-11-06 12:01:37] : Trato de obtener ultima medicion para este d
ispositivoId: 1
node-backend | [2025-11-06 12:01:37] : Trato de obtener ultimo comando para este di
spositivoId: 1
node-backend | [2025-11-06 12:01:52] : Token en chequeo: eyJhbGciOiJIUzI1NiIsInR5C
CIElkpWCJ9eyJcI2VybmlbQ1nsimFtZS16ImFkbmluIiwiFc3dvcnM0IiIyMTizMny0TdhN
TE3NDM4OTRNMGU8YTwmWzjMyIsImRlc2NyXAx10m51b6g0LcJpYXQ10jE3Mj10NDEyNDJ9.cg_TUY9h14cz
FHLDd8YfimAPPww%ca7_s6gvodgYRUnH| Recopilación en el
node-backend | [2025-11-06 12:01:52] : Trato de obtener ultí
ste d
ispositivoId: 1
node-backend | [2025-11-06 12:01:52] : Trato de obtener ultimo comando para este di
spositivoId: 1
node-backend | [2025-11-06 12:01:54] : MQTT mensaje recibido: /dispositivo/1/medi
on
node-backend | [2025-11-06 12:01:54] : MQTT Ingreso Medicion : dispositivoId: 1, fe
cha: 2025-10-15T17:00:00-03:00, valor: 33, carril: 2, clasificacionId: 1|
```

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

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

4.4. Pruebas del frontend

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

4.4.1. Objetivos

Los objetivos específicos de esta etapa fueron los siguientes:

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

```
t diego@banghe-validated:~/proyectos/Nejimir.c Log módulo ESP32C3
I (34806) GPRS: 0x3fc95f80 2f 31 2f 6d 65 64 69 63 69 6f 6e 7b 22 64 69 73 ||/1/m
edicion('dis')
I (34806) GPRS: 0x3fc95f90 76 6f 73 69 74 69 76 6f 49 64 22 3e 31 2c 22 76 |posi
tivoId":1,"v|
I (34816) GPRS: 0x3fc95fa0 61 6c 6f 72 22 3a 33 33 2c 22 03 61 72 72 69 6c |aler
t":33,"carril|
I (34826) GPRS: 0x3fc95fb0 22 3a 32 2c 22 63 6c 61 73 69 66 69 65 61 68 69 ||":2,
"clasificaci1|
I (34836) GPRS: 0x3fc95fc0 6f 6e 49 64 22 3a 31 2c 22 66 65 63 68 61 22 3a |onId
":1,"fecha":|
I (34846) GPRS: 0x3fc95fdb 22 32 30 32 35 2d 31 30 2d 31 35 34 31 37 3a 38 ||"202
5-10-15T17:0|
T (34856) GPRS: 0x3fc95fe0 38 3a 30 30 2d 30 33 3a 30 30 22 7d ||@:00
-03:00"|
I (36536) GPRS: CIPSEND OK:
SEND OK

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



```
2 diego@banghe-validated:~/proyectos/flow-track/app_8 Log backend
node-backend | [2025-11-06 12:01:37] : Trato de obtener ultima medicion para este d
ispositivoId: 1
node-backend | [2025-11-06 12:01:37] : Trato de obtener ultimo comando para este di
spositivoId: 1
node-backend | [2025-11-06 12:01:52] : Token en chequeo: eyJhbGciOiJIUzI1NiIsInR5C
CIElkpWCJ9eyJcI2VybmlbQ1nsimFtZS16ImFkbmluIiwiFc3dvcnM0IiIyMTizMny0TdhN
TE3NDM4OTRNMGU8YTwmWzjMyIsImRlc2NyXAx10m51b6g0LcJpYXQ10jE3Mj10NDEyNDJ9.cg_TUY9h14cz
FHLDd8YfimAPPww%ca7_s6gvodgYRUnH| Recopilación en el
node-backend | [2025-11-06 12:01:52] : Trato de obtener ultí
ste d
ispositivoId: 1
node-backend | [2025-11-06 12:01:52] : Trato de obtener ultimo comando para este di
spositivoId: 1
node-backend | [2025-11-06 12:01:54] : MQTT mensaje recibido: /dispositivo/1/medi
on
node-backend | [2025-11-06 12:01:54] : MQTT Ingreso Medicion : dispositivoId: 1, fe
cha: 2025-10-15T17:00:00-03:00, valor: 33, carril: 2, clasificacionId: 1|
```

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

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

4.4. Pruebas del frontend

Las pruebas del frontend tuvieron como finalidad evaluar el correcto funcionamiento de la interfaz web desarrollada, que garantiza 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.

4.4. Pruebas del frontend

39

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

4.4.2. Metodología

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

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

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

4.4. Pruebas del frontend

39

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

4.4.2. Metodología

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

- Compatibilidad y visualización: se validó la correcta visualización de los componentes en distintas resoluciones y navegadores, utilizando Chrome DevTools [41] 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 [42] 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:

4.4. Pruebas del frontend

41

Dispositivo 1

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

Último Comando

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

Última Medición

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

Crear Comando

Tipo de Comando: Creación nuevo comando
Valor del Comando:

Última medición registrada y botón de mediciones históricas

VER MEDICIONES

FIGURA 4.13. Panel de visualización del dispositivo.

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

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

Crear Comando

Tipo de Comando: Val batería

Última medición registrada y botón de mediciones históricas

VER MEDICIONES

Crear Comando

Tipo de Comando: Val batería

Última medición registrada y botón de mediciones históricas

VER MEDICIONES

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

4.4. Pruebas del frontend

41

Dispositivo 1

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

Último Comando

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

Última Medición

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

Crear Comando

Tipo de Comando: Creación nuevo comando
Valor del Comando:

Última medición registrada y botón de mediciones históricas

VER MEDICIONES

FIGURA 4.13. Panel de visualización del dispositivo.

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

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.



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.



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

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

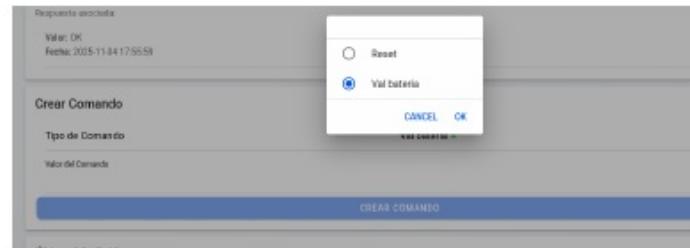


FIGURA 4.14. Panel de visualización del dispositivo: preparación para la ejecución de un comando y su ejecución.

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



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.



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.



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

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.

4.4. Pruebas del frontend

43

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



FIGURA 4.18. Panel de visualización del historial de mediciones.

4.4.3. Resultados y observaciones

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

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

4.5.1. Metodología de la prueba

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

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

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

```

Log ESP32C
I (180770) GPRS: 0x3fc95f00 2f 31 2f 0d 65 04 09 63 09 0f 0e 7b 22 04 09 73 |/|
medicion|/|
I (180770) GPRS: 0x3fc95f00 70 0f 73 69 74 09 76 8f 49 64 22 3e 31 2c 22 76 |pos|
|tivoId|:1,"v|
I (180786) GPRS: 0x3fc05fa0 61 6c 6f 72 22 3a 33 38 2c 22 63 61 72 72 69 6c |alo|
|r":38,"carril|
I (180790) GPRS: 0x3fc95f00 22 38 32 2c 22 03 0c 61 73 09 00 09 03 01 03 09 |"/|
|clasificacion|
I (180806) GPRS: 0x3fc95fc0 6f 6e 49 64 22 3a 32 2c 22 66 65 63 68 61 22 3a |/|
|d":2,"fecha":|
I (180810) GPRS: 0x3fc95fd0 22 32 30 32 35 2d 31 30 2d 31 35 34 31 37 3a 31 |"/|
25-10-15T17:10:00-03:00|
I (180820) GPRS: 0x3fc95fe0 30 3d 38 30 2d 30 33 3a 30 30 22 76 |/|
0-03:00"}|/|
I (182476) GPRS: CIPSEND OK: SEND OH
Publicación medición
I (182476) GPRS: PUBLISH enviado: /dispositivo/1/medicion -> {"dispositivoId":1,"val|
or":38,"carril":2,"clasificacionId":2,"fecha":"2025-10-15T17:10:00-03:00"}|/|
I (182476) GPRS: CIPSEND OK: SEND OH

Log backend
node-backend | [2025-11-06 12:02:52] : Trato de obtener medición para este dispositivoId: 1
node-backend | [2025-11-06 12:02:57] : Buscando medición con dispositivoId: 1 (lmit 100, offset 0, desde -, hasta -)
node-backend | [2025-11-06 12:03:00] : MQTT mensaje recibido: /dispositivo/1/medicion
node-backend | [2025-11-06 12:03:00] : MQTT Ingreso Medicion : dispositivoId: 1, fe|
cha: 2025-10-15T17:10:00-03:00, valor: 38, carril: 2, clasificacionId: 2
node-backend | [2025-11-06 12:03:00] : MQTT medición guardada para dispositivo 1
node-backend | [2025-11-06 12:03:07] : Token en chequeo: eyJhbGciOiJIUzI1NiIsInR5C|
I6IkpxVCJ9.eJJC2VYRm91bmQ1OnsibmF2ZS16ImRkTWluIiWicGFzc3dvcmQIOTIyMTIzMjMydTnhTdbN|
ME3NDM4OTRhMGUyYTgjMkZjMyI5ImRlc2NyxA10m51bGx9LCJpYXQiOjE3NjI4NDEyNDQ9.c6_TUYsh14cz|
uJ14DyYFwA9sQd-A-9nq3o4vVml

```

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

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.

Se presentan los pasos realizados durante la prueba integral del sistema, cuyo objetivo es verificar la correcta interacción entre el dispositivo, el backend y la interfaz web en un escenario completo de funcionamiento.

En la figura 4.19 se muestra la captura del log del módulo ESP32-C3 junto con el log del backend, durante la publicación y recepción de una medición. En la parte superior se observa el envío del mensaje MQTT en el tópico dispositivo/1/medicion, mientras que en la parte inferior el backend registra la recepción del mensaje, su validación y almacenamiento en la base de datos.

4.5. Prueba final de integración

45

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



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

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

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

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

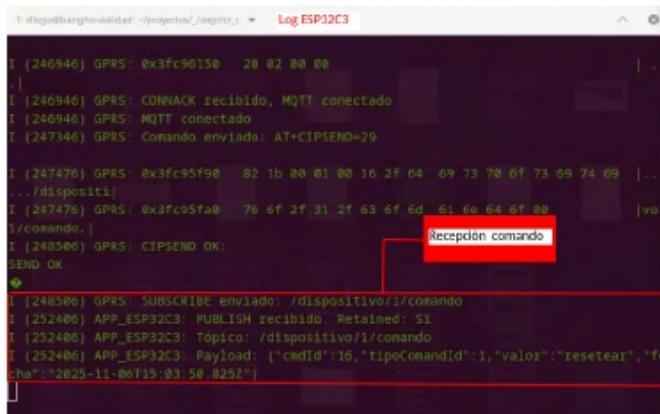
En la figura 4.22 se muestra parte de la traza del proceso de ejecución del comando, donde la consola del módulo ESP32-C3 registra la recepción del comando.

4.5. Prueba final de integración

45

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

En la figura 4.20 se visualiza la interfaz web luego de recibir la medición publicada. La aplicación consulta la API REST del backend y actualiza en tiempo real la tabla de mediciones, que muestra el nuevo registro con su fecha, valor, carril, clasificación y dispositivo asociado.

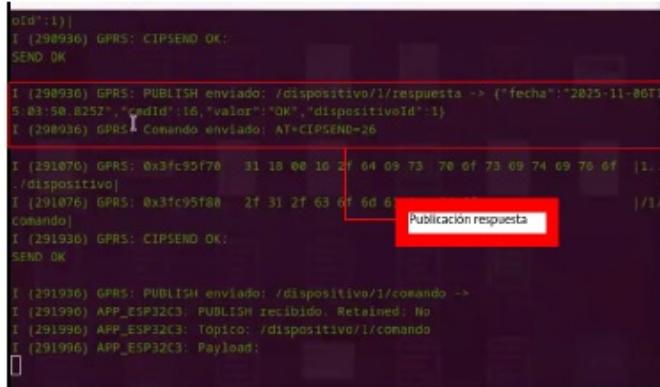


```
I (246946) GPRS: 0x3fc9015e 28 02 80 00
I (246946) GPRS: CONNACK recibido, MQTT conectado
I (246946) GPRS: MQTT conectado
I (247346) GPRS: Comando enviado: AT+CIPSENDO=29
I (247476) GPRS: 0x3fc95f9e 82 1b 80 01 08 16 2f 64 69 73 78 6f 73 69 74 69 | ...
.../dispositivo/
I (247476) GPRS: 0x3fc95fa0 76 6f 2f 31 2f 63 6f 6d 61 6e 64 6f 00
I/comando/
I (248500) GPRS: CIPSENDO OK:
SEND OK
+
I (248506) GPRS: SUBSCRIBE enviado: /dispositivo/1/comando
I (252406) APP_ESP32C3: PUBLISH recibido. Retained: Si
I (252406) APP_ESP32C3: Topico: /dispositivo/1/comando
I (252406) APP_ESP32C3: Payload: {"cmdId":16,"tipoComandoId":1,"valor":"resetear","fecha":"2025-11-06T15:03:50.825Z"}]
```

A red box highlights the text "Recepción comando".

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`.



```
old':1)
I (290936) GPRS: CIPSENDO 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+CIPSENDO=26

I (291070) GPRS: 0x3fc95f70 31 18 00 10 2f 64 09 73 78 6f 73 09 74 69 76 6f | ...
.../dispositivo/
I (291076) GPRS: 0x3fc95f80 2f 31 2f 63 6f 60 68 | /1/
comando/
I (291936) GPRS: CIPSENDO OK:
SEND OK

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

A red box highlights the text "Publicación respuesta".

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.

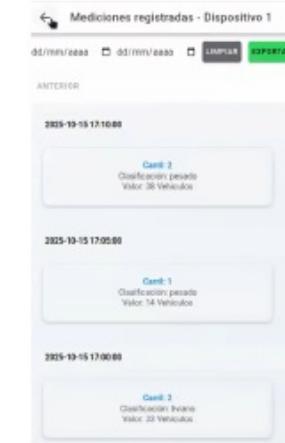


FIGURA 4.20. Visualización de la medición recibida en la interfaz web del sistema.

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

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



The interface has two tabs:

- Último Comando:** Shows a command history entry: "Tipo de comando: Reset", "Valor: resetear", "Fecha: 2025-11-06 12:03:50", and "No hay respuesta asociada al último comando".
- Crear Comando:** A form with fields "Tipo de Comando" (selected to "Reset") and "Valor del Comando" (set to "resetear"). A blue button at the bottom right says "CREAR COMANDO".

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

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

4.5. Prueba final de integración

47

```

2 Diego@diego-OptiPlex-5090: ~ [projecto] $ ./nodejs/mqttapp.js
ispositivoId: 1
node-backend | [2025-11-06 12:06:04] : Trato de obtener ultimo comando para este dispositivo
ispositivoId: 1
node-backend | [2025-11-06 12:06:04] : Token en c" 161kpXVCj9...eyJlc2VyRm9lbnm0nsibmF1ZS16IemFkbWluIix... Recepión respuesta
node-backend | [2025-11-06 12:06:04] : Trato de obtener Respuesta con cmdId: 16
node-backend | [2025-11-06 12:06:09] : MQTT mensaje recibido: /dispositivo/1/respuesta
node-backend | [2025-11-06 12:06:09] : MQTT Ingreso respuesta : fecha: 2025-11-06T15:03:50Z, valor: OK, cmdId: 16
node-backend | [2025-11-06 12:06:09] : MQTT respuesta guardada para dispositivo 1

```

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

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

| | |
|-----------------------|---------------------|
| Dispositivo 1 | |
| Nombre: | contador 1 |
| Ubicación: | Padre Budo |
| Tipo contador: | DTEC |
| | |
| Último Comando | |
| Tipo de comando: | Reset |
| Valor: | resetear |
| Fecha: | 2025-11-06 12:03:50 |
| Respuesta asociada: | |
| Valor: | OK |
| Fecha: | 2025-11-06 12:03:50 |

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

4.5.2. Resultados obtenidos

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

En particular, se observaron los siguientes aspectos destacados:

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

4.5. Prueba final de integración

47

```

1 Diego@diego-OptiPlex-5090: ~ [projecto] $ ./log_ESP32C3
(246946) GPRS: 0x3fc96150 28 02 00 00
...
(246946) GPRS: CONNACK recibido, MQTT conectado
(246946) GPRS: MQTT conectado
(247346) GPRS: Comando enviado: AT+CIPSENDO=29
...
(247476) GPRS: 0x3fc95f98 82 1b 00 01 00 16 2f 64 69 73 70 6f 73 89 74 69
.../dispositivo/
(247476) GPRS: 0x3fc95fa0 76 6f 2f 31 2f 63 6f 6d 61 6e 64 6f 80
.../comando_
(248580) GPRS: CIPSENDO OK
END OK
...
(248586) GPRS: SUBSCRIBE enviado: /dispositivo/1/comando
(252486) APP_ESP32C3: PUBLISH recibido. Retained: 51
(252486) APP_ESP32C3: Tópico: /dispositivo/1/comando
(252486) APP_ESP32C3: Payload: {"cmdId":16,"tipoComando":1,"valor":"resetear","fecha":"2025-11-06T15:03:50Z"}

```

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 entre el nodo y el backend. En la consola superior se observa el log del ESP32-C3, que publica la respuesta con el resultado del procesamiento interno del contador. Esta respuesta es publicada por el firmware en el tópico `dispositivo/1/respuesta`, incluyendo los campos `fecha`, `cmdId`, `valor` y `dispositivoId`. En la consola inferior se aprecia el registro del backend, el cual recibe el mensaje desde el broker MQTT, valida su estructura y almacena la información en la base de datos MySQL.

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

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

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

4.6. Comparación con otras soluciones

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

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

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

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

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

```

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":10,"valorOK":true,"dispositivoId":1}
I (290936) GPRS: Comando enviado: AT+CIPSEND=20

I (291070) GPRS: 0x3fc95f70 31 18 00 10 2f 64 09 73 70 0f 73 09 74 89 70 0f |1.
./dispositivo
I (291076) GPRS: 0x3fc95f80 1f 31 2f 63 6f 6d 8 |1/
comando
I (291936) GPRS: CIPSEND OK:
SEND OK

I (291936) GPRS: PUBLISH enviado: /dispositivo/1/conando ->
I (291990) APP_ESP32C3: PUBLISH recibido. Retained: No
I (291990) APP_ESP32C3: Topico: /dispositivo/1/comando
I (291996) APP_ESP32C3: Payload:
[]

$ http://192.168.1.10:8080/validad -d '{"track": "esp_32c3"}'
dispositivoId: 1
node-backend | [2025-11-06 12:06:04] : Trato de obtener ultimo comando para este dispositivo
dispositivoId: 1
node-backend | [2025-11-06 12:06:04] : Token en cache: I61kpXVCJ9_eYlCl2VYRm91bmQ1Dns1bmFtZS161mfKbmbluIxw
node-backend | [2025-11-06 12:06:04] : Reacción respuesta: I1NIIsInR5Cc
node-backend | [2025-11-06 12:06:04] : Token en cache: NE3NDM40TRhMGU8YTgWMZjMyIsImRlc2NyaXA10m51bGx9LCJ
node-backend | [2025-11-06 12:06:04] : Trato de obtener Respuesta con cmdId: 16
node-backend | [2025-11-06 12:06:04] : MQTT mensaje recibido: /dispositivo/1/response
node-backend | [2025-11-06 12:06:09] : MQTT Ingreso respuesta : fecha: 2025-11-06T15:03:50.825Z, valorOK: true, cmdId: 16
node-backend | [2025-11-06 12:06:09] : MQTT respuesta guardada para dispositivo 1

```

FIGURA 4.23. Intercambio de mensajes entre el ESP32-C3 y el backend durante la recepción y procesamiento de una respuesta.

Finalmente, en la Figura 4.24 se muestra la visualización del resultado en la interfaz web. Una vez almacenada la respuesta, la aplicación cliente consulta el endpoint correspondiente de la API REST y actualiza la vista con la información más reciente del dispositivo. De esta manera, el usuario puede verificar en tiempo real que el comando fue recibido y ejecutado correctamente, completando el ciclo de comunicación bidireccional del sistema.

4.6. Comparación con otras soluciones

49

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

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

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

4.5. Prueba final de integración

49

Dispositivo 1

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

Último Comando

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

Valor: OK
Fecha: 2025-11-06 12:03:50

FIGURA 4.24. Visualización de la respuesta del dispositivo en la interfaz web tras el procesamiento exitoso en el backend.

4.5.2. Resultados obtenidos

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

En particular, se observaron los siguientes aspectos destacados:

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

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

Estos resultados confirman que la solución es técnicamente viable para su despliegue en entornos de campo, que ofrece una integración transparente entre

hardware embebido, servicios de red y aplicaciones web. Además, la arquitectura modular basada en estándares abiertos garantiza la posibilidad de futuras expansiones y adaptaciones a nuevas tipologías de dispositivos o protocolos de comunicación.

4.6. Comparación con otras soluciones

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

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

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

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

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

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.

4.6. Comparación con otras soluciones

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

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

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

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

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

5.2. Trabajo futuro

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

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

Bibliografía

- [1] Juan Asiain et al. «LoRa-Based Traffic Flow Detection for Smart-Road». En: *Sensors* 21.2 (2021), pág. 338. DOI: 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. 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. URL: <https://www.mdpi.com/2076-3417/12/3/1497>.
- [4] Miovision. *TrafficLink / Managed Connectivity*. <https://miovision.com/trafficlink/>. Soluciones comerciales. 2023.
- [5] Sensys Networks. Documentación técnica y productos. <https://sensysnetworks.com/>. 2023.
- [6] MetroCount. Contadores y guías técnicas. <https://www.metrocount.com/>. 2023.
- [7] Exemys Managed Connectivity. Accedido: 16-Sep-2025. 2025. URL: <https://www.exemys.com/site/index.shtml>.
- [8] Digi International. *Digi Remote Manager: IoT Device Monitoring and Management Solution*. <https://www.digi.com/products/iot-software-services/digi-remote-manager>. Accedido: 11 de septiembre de 2025. 2025.
- [9] A. A. Sukmandhani, M. Zarlis y Nurudin. «Monitoring Applications for Vehicle based on Internet of Things (IoT) using the MQTT Protocol». En: *BINUS Conference Proceedings*. Accedido: 11 de septiembre de 2025. 2023. URL: <https://research.binus.ac.id/publication/C1B25545-66P9-49C3-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/GCAT52182.2021.9587636.
- [11] Zuoling Niu. «Research and Implementation of Internet of Things Communication System Based on MQTT Protocol». En: *Journal of Physics: Conference Series* 012019 (2023). Accedido: 11 de septiembre de 2025.
- [12] OpenRemote. *OpenRemote: 100 % Open Source IoT Device Management Platform*. <https://openremote.io/>. Accedido: 11 de septiembre de 2025. 2025.
- [13] Espressif Systems. *ESP32-C3 Series Datasheet*. Accedido: 23-Sep-2025. Espressif Systems. 2021. URL: https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf.
- [14] Analog Devices. *Fundamentals of RS-232 Serial Communications*. <https://www.analog.com/en/resources/technical-articles/fundamentals-of-rs232-serial-communications.html>. Accedido: 11-Sep-2025. 2020.

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.

- [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.

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

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

5.2. Trabajo futuro

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

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

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

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/GCAT5182.2021.9587636](https://doi.org/10.1109/GCAT5182.2021.9587636).
- [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.