

Resumen

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

Resumen

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

Índice general

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

Índice general

| | |
|--|----|
| Resumen | 1 |
| 1. Introducción general | 1 |
| 1.1. Motivación | 1 |
| 1.1.1. Contexto actual | 1 |
| 1.1.2. Limitaciones del desarrollo previo | 1 |
| 1.1.3. Impacto esperado | 2 |
| 1.1.4. Diseño conceptual | 2 |
| 1.2. Objetivos | 2 |
| 1.2.1. Objetivo general | 2 |
| 1.2.2. Objetivos específicos | 2 |
| 1.3. Estado del arte y propuesta de valor | 3 |
| 1.4. Alcance | 3 |
| 2. Introducción específica | 5 |
| 2.1. Protocolos y comunicación | 5 |
| 2.2. Componentes de hardware utilizados | 6 |
| 2.3. Tecnologías de software aplicadas | 8 |
| 2.4. Software de control de versiones | 10 |
| 3. Diseño e implementación | 11 |
| 3.1. Arquitectura del sistema | 11 |
| 3.1.1. Descripción ampliada de bloques y responsabilidades | 11 |
| 3.1.2. Flujo de datos | 13 |
| 3.2. Arquitectura del nodo | 14 |
| 3.3. Desarrollo del backend | 16 |
| 3.3.1. Arquitectura y tecnologías | 17 |
| 3.3.2. Funcionalidades principales | 17 |
| 3.3.3. Organización en controladores | 18 |
| 3.3.4. Mapa de endpoints | 18 |
| 3.3.5. Seguridad y extensibilidad | 19 |
| 3.4. Desarrollo del frontend | 20 |
| 3.4.1. Arquitectura y tecnologías | 20 |
| 3.4.2. Funcionalidades principales | 20 |
| 3.4.3. Integración con el backend | 21 |
| 3.5. Despliegue del sistema | 23 |
| 3.5.1. Entorno productivo e integración continua | 23 |
| 3.5.2. Monitoreo post-implantación | 23 |
| 3.6. Integración con la infraestructura existente | 23 |
| 4. Ensayos y Resultados | 25 |
| 4.1. Banco de pruebas | 25 |
| 4.1.1. Diseño del entorno de pruebas | 25 |
| Bibliografía | 27 |

| | |
|---|-----------|
| 4.1.2. Metodología experimental | 26 |
| 4.1.3. Resultados y observaciones | 26 |
| 4.2. Pruebas de la API REST | 26 |
| 4.2.1. Objetivos y alcance | 27 |
| 4.2.2. Metodología de prueba | 27 |
| 4.2.3. Resultados obtenidos | 27 |
| 4.2.4. Conclusiones | 29 |
| 4.3. Pruebas de componentes | 29 |
| 4.3.1. Enfoque general | 29 |
| 4.3.2. Resultados por componente | 29 |
| 4.3.3. Conclusiones | 30 |
| 4.4. Pruebas del frontend | 30 |
| 4.4.1. Objetivos | 30 |
| 4.4.2. Metodología | 30 |
| 4.4.3. Resultados y observaciones | 31 |
| 4.5. Prueba final de integración | 31 |
| 4.5.1. Metodología de la prueba | 31 |
| 4.5.2. Resultados obtenidos | 32 |
| 4.5.3. Conclusiones de la integración | 33 |
| 4.6. Comparación con otras soluciones | 33 |
| Bibliografía | 35 |

Índice de figuras

| | |
|--|----|
| 1.1. Diagrama en bloques del sistema. | 4 |
| 2.1. Contador de tránsito DTEC ¹ . | 6 |
| 2.2. Módulo RS-232/TTL ² . | 7 |
| 2.3. Microcontrolador ESP32-C3 utilizado en los nodos de campo ³ . | 7 |
| 2.4. Módulo SIM800L ⁴ . | 8 |
| 3.1. Diagrama de arquitectura del sistema y el flujo de datos. | 12 |
| 3.2. Diagrama de secuencia del flujo de datos. | 14 |
| 3.3. Diagrama de conexión entre los módulos del sistema. | 15 |
| 3.4. Fotografía contador de tránsito DTEC instalado en campo ⁵ . | 16 |
| 3.5. Diagrama de flujo de información del Backend. | 17 |
| 3.6. Diagrama con la disposición de los controladores y flujo de dependencias. | 18 |
| 3.7. Diagrama de estructura de los componentes por pantalla. | 21 |
| 3.8. Diagrama de flujo de comunicación con el backend. | 22 |

Índice de 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. | 13 |
| 3.2. Diagrama de secuencia del flujo de datos. | 14 |
| 3.3. Diagrama de conexión entre los módulos del sistema. | 16 |
| 3.4. Fotografía contador de tránsito DTEC instalado en campo ⁵ . | 17 |
| 3.5. Diagrama de flujo de información del Backend. | 18 |
| 3.6. Diagrama con la disposición de los controladores y flujo de dependencias. | 20 |
| 3.7. Diagrama de estructura de los componentes por pantalla. | 23 |
| 3.8. Diagrama de flujo de comunicación con el backend. | 24 |

Índice de tablas

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

Índice de tablas

| | |
|---|----|
| 3.1. Endpoints REST principales | 21 |
|---|----|

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

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

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

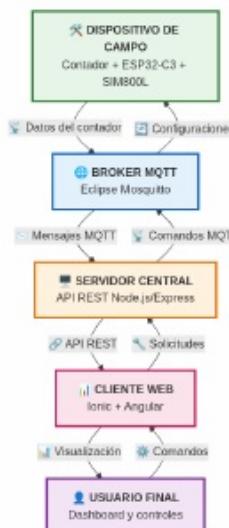


FIGURA 3.1. Diagrama de arquitectura del sistema y el flujo de datos.

- dispositivo/{id}/medicion
 - dispositivo/{id}/comando
 - dispositivo/{id}/respuesta
- Servidor central: el servidor central reúne dos responsabilidades principales:
- Componente suscriptor MQTT que valida, transforma y enruta mensajes hacia la lógica de negocio y la persistencia.
 - API REST que expone servicios de consulta, gestión y emisión de comandos. Esta separación permite que consumidores adicionales (por ejemplo, módulos analíticos) se suscriban al broker sin impactar la disponibilidad de la API. La persistencia se implementa en MySQL con un esquema relacional que soporta consultas por rango temporal, índices para rendimiento y auditoría de comandos.
- Cliente/Visualización: la interfaz web consume la API REST para consultas históricas y para recibir eventos en tiempo real. Se eligió Ionic + Angular por su compatibilidad con entornos de escritorio y móviles y por facilitar un despliegue unificado. Las funciones principales del cliente son: visualización de eventos en tiempo real, consulta histórica filtrada, envío de comandos remotos con seguimiento de estado y panel de telemetría para mantenimiento preventivo.

Se tomaron decisiones de diseño clave para el sistema. En primer lugar, se separó el broker de la aplicación, lo que permite cambiar de broker o desplegar uno local sin afectar la lógica de negocio. Para la mensajería se optó por MQTT, debido a que es un protocolo ligero que minimiza el overhead en redes GPRS y facilita la comunicación mediante el modelo pub/sub. Además, se implementó una API REST utilizando Node.js y Express para exponer endpoints tanto transaccionales como de gestión, centralizando la autenticación y el control de accesos.

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

3.1. Arquitectura del sistema

13

3.1.2. Flujo de datos

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

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

3.1. Arquitectura del sistema

13



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

3.1.2. Flujo de datos

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

- Detección: el contador detecta un paso, acumula y en determinado intervalo envía una trama por RS-232 al ESP32-C3.
- Preprocesado en nodo: el firmware valida la trama, añade sello temporal y metadatos, y encola el evento en memoria (FIFO).
- Transmisión: cuando la conexión GPRS está disponible, el nodo publica las mediciones en el tópico MQTT dispositivo/id/medicion.
- Ingesta y persistencia: el broker Mosquitto entrega el mensaje al suscriptor backend, el servicio valida el payload y persiste el registro en la base de datos MySQL.
- Visualización/Control: la interfaz web consulta la API REST para datos históricos y recibe notificaciones en tiempo real.

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

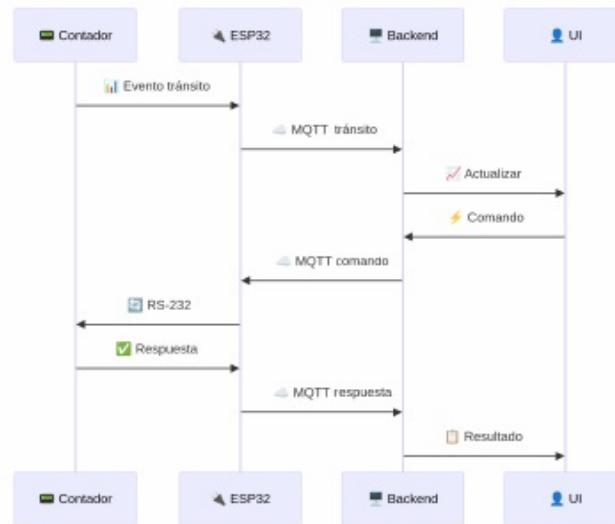


FIGURA 3.2. Diagrama de secuencia del flujo de datos.

3.2. Arquitectura del nodo

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

- Contador de tránsito modelo DTEC: dispositivo encargado de detectar el paso de vehículos y generar tramas de datos con la información del evento.
- Módulo de adaptación RS-232/TTL (MAX232): circuito de conversión de niveles eléctricos que asegura compatibilidad entre la interfaz serial del contador (RS-232) y el microcontrolador (niveles TTL).
- ESP32-C3: microcontrolador que ejecuta el firmware desarrollado sobre ESP-IDF. Sus funciones incluyen el preprocessamiento de eventos, el encolamiento FIFO, la suscripción a comandos remotos y la gestión integral de la comunicación con el servidor.
- Módulo de comunicación GPRS (SIM800L): interfaz de conectividad celular que publica eventos en el broker MQTT, recibe comandos desde el servidor y retransmite respuestas o estados del nodo.

- Emisión de comandos (desde UI): el operador genera un comando en la interfaz, la UI envía POST /app/comando al backend, que crea un `comm_id` único con dispositivo y publica en `dispositivo/id/comando`.
- Recepción nodo/Entrega al contador: el ESP32-C3 en `dispositivo/id/comando` recibe el comando, valida `cmd_id` y lo envía al contador por RS-232, se aplica un timeout configurable por comando.
- Ejecución y ack: el contador ejecuta la orden y responde por RS-232, el firmware publica el ack/resultado en `dispositivo/id/respuesta` con `cmd_id` y `status` (ok, failed, timeout, value).
- Actualización en backend y UI: el suscriptor MQTT del backend recibe el ack, actualiza la tabla respuesta (campo `valor`, `ack_ts`) y notifica a la UI para que el operador vea el resultado.

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

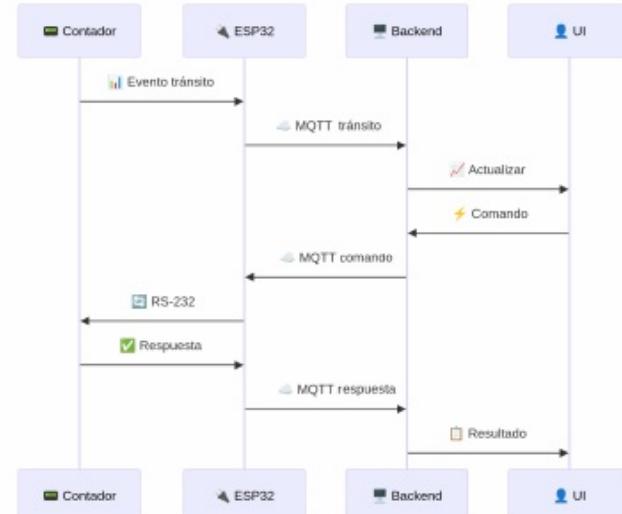


FIGURA 3.2. Diagrama de secuencia del flujo de datos.

3.2. Arquitectura del nodo

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

3.2. Arquitectura del nodo

15

- Alimentación y montaje: para garantizar el funcionamiento continuo del nodo en entornos de campo, se deben considerar dos aspectos fundamentales:

- Fuente de alimentación: se implementó un sistema de energía basado en una batería interna recargable, específicamente dimensionada para cubrir los picos de consumo del módulo SIM800L durante las transmisiones de datos. La autonomía del sistema está garantizada mediante un panel solar que mantiene la carga de la batería de forma continua.

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

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

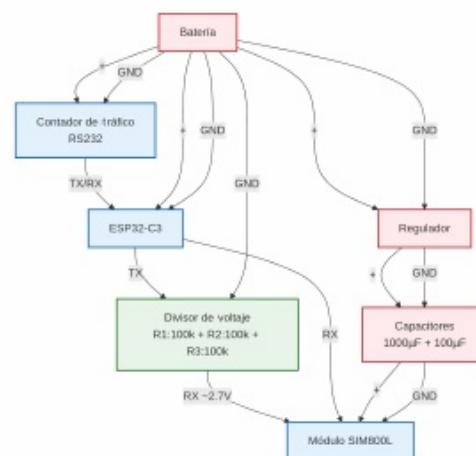


FIGURA 3.3. Diagrama de conexión entre los módulos del sistema.

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

3.2. Arquitectura del nodo

15

- Contador de tránsito modelo DTEC: dispositivo encargado de detectar el paso de vehículos y generar tramas de datos con la información del evento.

- Módulo de adaptación RS-232/TTL (MAX232): circuito de conversión de niveles eléctricos que asegura compatibilidad entre la interfaz serial del contador (RS-232) y el microcontrolador (niveles TTL).

- ESP32-C3: microcontrolador que ejecuta el firmware desarrollado sobre ESP-IDF. Sus funciones incluyen el preprocesamiento de eventos, el encolamiento FIFO, la suscripción a comandos remotos y la gestión integral de la comunicación con el servidor.

- Módulo de comunicación GPRS (SIM800L): interfaz de conectividad celular que publica eventos en el broker MQTT, recibe comandos desde el servidor y retransmite respuestas o estados del nodo.

- Alimentación y montaje: para garantizar el funcionamiento continuo del nodo en entornos de campo, se deben considerar dos aspectos fundamentales:

- Fuente de alimentación: se implementó un sistema de energía basado en una batería interna recargable, específicamente dimensionada para cubrir los picos de consumo del módulo SIM800L durante las transmisiones de datos. La autonomía del sistema está garantizada mediante un panel solar que mantiene la carga de la batería de forma continua.

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

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

gabinete original, que garantiza la confiabilidad del sistema en condiciones de intemperie.

En la figura 3.4 se observa el contador de tránsito instalado en campo, junto con su gabinete de protección y la batería interna que asegura autonomía energética. El conjunto se encuentra montado al costado de la ruta, en condiciones reales de operación, lo que permite apreciar el encapsulado diseñado para resistir las exigencias ambientales del entorno vial.



FIGURA 3.4. Fotografía contador de tránsito DTEC instalado en campo.¹

3.3. Desarrollo del backend

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

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

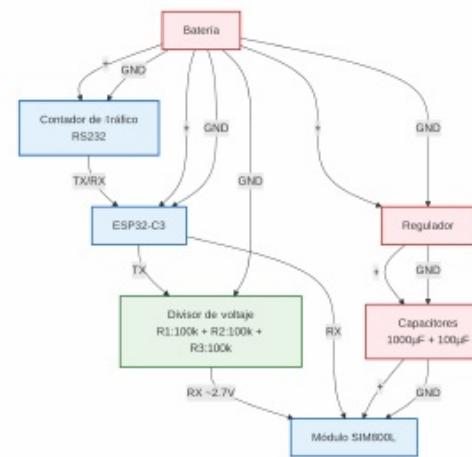


FIGURA 3.3. Diagrama de conexión entre los módulos del sistema.

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

En la figura 3.4 se observa el contador de tránsito instalado en campo, junto con su gabinete de protección y la batería interna que asegura autonomía energética. El conjunto se encuentra montado al costado de la ruta, en condiciones reales de operación, lo que permite apreciar el encapsulado diseñado para resistir las exigencias ambientales del entorno vial.

3.3. Desarrollo del backend

17

3.3.1. Arquitectura y tecnologías

El servicio se implementó en Node.js con Express, organizando la aplicación en controladores, rutas y middlewares, y usando Sequelize [27], un ORM [35] que facilita los modelos y asegura independencia de la persistencia.

La comunicación con los dispositivos se realiza mediante tópicos MQTT en Eclipse Mosquitto. Las mediciones se publican en dispositivo/id/medicion y el backend las valida y guarda en MySQL, mientras que los comandos y respuestas se gestionan en dispositivo/id/comando y dispositivo/id/respuesta. El despliegue usa Docker Compose [32] para ejecutar backend, base de datos y broker [17] en contenedores, y el registro se maneja con Winston [28] y Morgan [29] para trazabilidad completa.

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

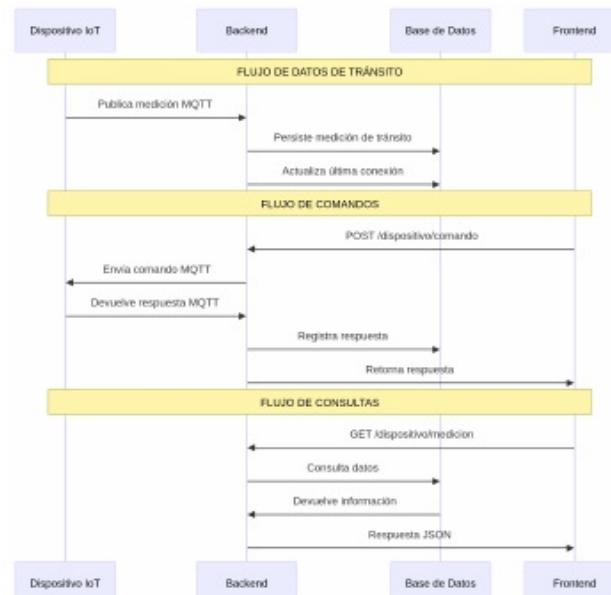


FIGURA 3.5. Diagrama de flujo de información del Backend.

3.3.2. Funcionalidades principales

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

- Gestión de dispositivos: alta, baja, modificación y consulta.

3.3. Desarrollo del backend

17



FIGURA 3.4. Fotografía contador de tránsito DTEC instalado en campo.¹

3.3. Desarrollo del backend

El backend constituye el núcleo lógico del sistema, encargado de articular la comunicación entre los dispositivos de campo, la base de datos y la interfaz de usuario. Su diseño se basó en principios de modularidad, escalabilidad y seguridad, empleando un conjunto de tecnologías ampliamente utilizadas en entornos de Internet de las Cosas (IoT).

3.3.1. Arquitectura y tecnologías

El servicio fue implementado en Node.js con el framework Express, lo que permitió organizar la aplicación en controladores, rutas y middlewares. Para la interacción con la base de datos relacional se adoptó Sequelize [27], un ORM [35] que facilita la definición de modelos y garantiza independencia frente a cambios en la capa de persistencia.

La comunicación con los dispositivos se realiza mediante suscripción a tópicos MQTT en el broker Eclipse Mosquitto.

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

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

3.3.3. Organización en controladores

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

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

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

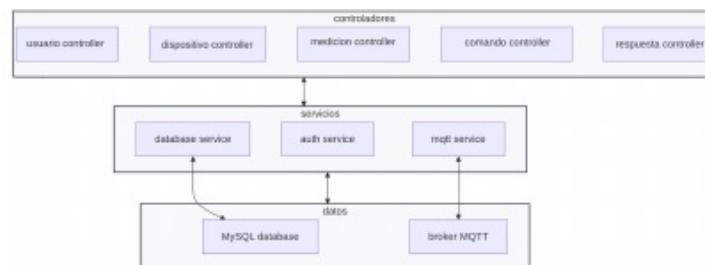


FIGURA 3.6. Diagrama con la disposición de los controladores y flujo de dependencias.

3.3.4. Mapa de endpoints

El backend expone una serie de endpoints REST que conforman la interfaz principal de comunicación con los servicios de aplicación y los dispositivos de campo.

Cada vez que un dispositivo publica una medición en dispositivo/{id}/medición, el backend lo recibe, valida el payload y lo persiste en MySQL.

En el backend, los comandos se publican en dispositivo/{id}/comando y las respuestas se actualizan según los envíos en dispositivo/{id}/respuesta incluyendo en el body el cmd_id.

Para el despliegue y la integración continua se empleó Docker Compose [32], lo que permite levantar simultáneamente el backend, la base de datos y el broker MQTT [17] en contenedores aislados.

El sistema de registro se implementó con Winston [28] y Morgan [29], integrados para disponer de trazabilidad tanto de las solicitudes HTTP como de la interacción con MQTT.

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

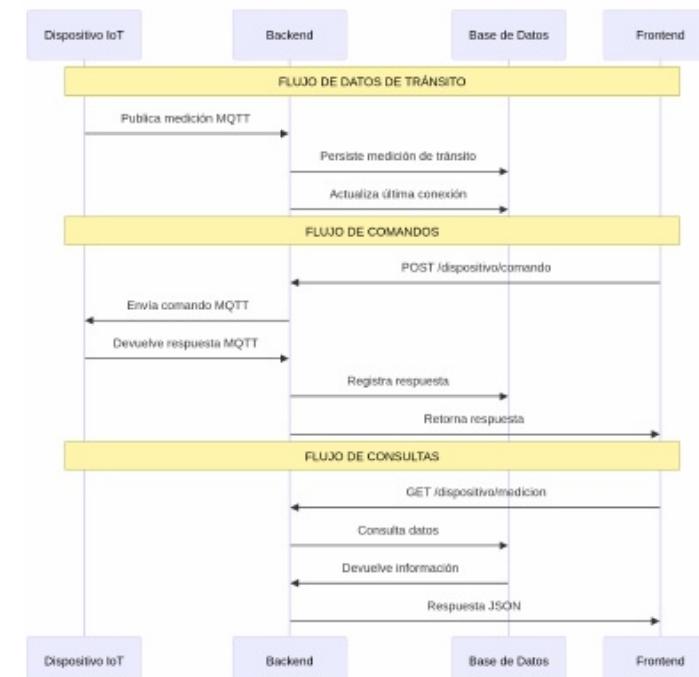


FIGURA 3.5. Diagrama de flujo de información del Backend.

3.3. Desarrollo del backend

19

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

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

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

3.3.5. Seguridad y extensibilidad

Además de la autenticación mediante JWT, todos los endpoints aplican validaciones y sanitización de parámetros de entrada y salida. El sistema de logging, implementado con Winston y Morgan, garantiza trazabilidad de las operaciones tanto en la capa HTTP como en la mensajería MQTT. La arquitectura modular

3.3. Desarrollo del backend

19

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 asegura una clara separación de responsabilidades, facilita el mantenimiento y permite la escalabilidad de la aplicación. Entre los controladores principales se encuentran los siguientes:

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

En la figura 3.7 se observa el diagrama con la disposición de los controladores y flujo de dependencias.

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

3.4. Desarrollo del frontend

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

3.4.1. Arquitectura y tecnologías

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

3.4.2. Funcionalidades principales

El frontend integra las siguientes funciones clave:

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

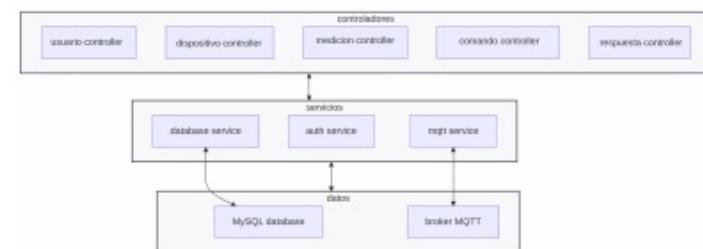


FIGURA 3.6. Diagrama con la disposición de los controladores y flujo de dependencias.

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

3.4. Desarrollo del frontend

21

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

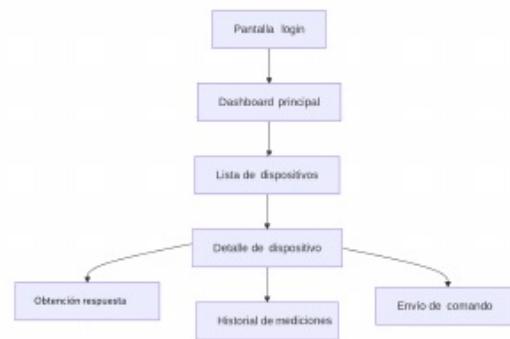


FIGURA 3.7. Diagrama de estructura de los componentes por pantalla.

3.4.3. Integración con el backend

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

3.4. Desarrollo del frontend

21

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

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

3.3.5. Seguridad y extensibilidad

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

3.4. Desarrollo del frontend

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

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

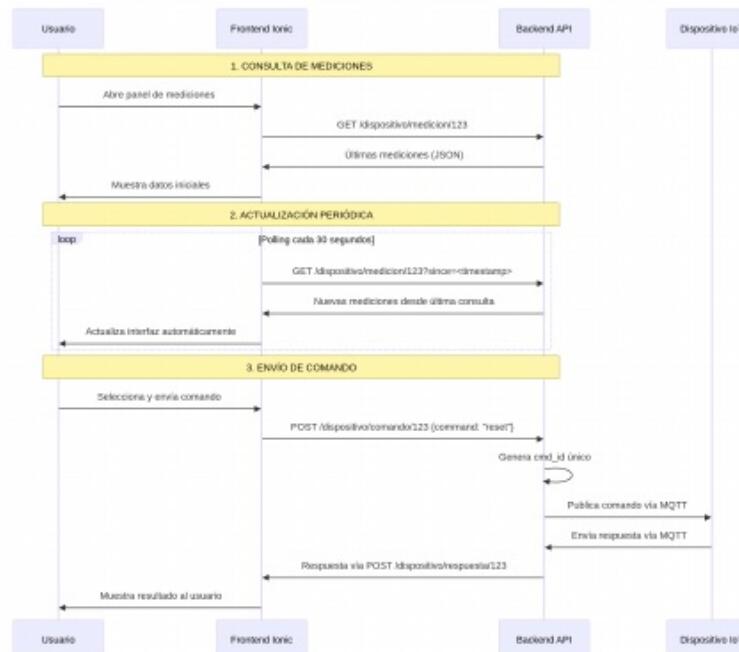


FIGURA 3.8. Diagrama de flujo de comunicación con el backend.

3.4.1. Arquitectura y tecnologías

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

Las tecnologías principales aportan ventajas concretas:

- Ionic: conjunto de componentes UI listos para usar que permiten construir pantallas consistentes y adaptables.
- Angular: organización de la aplicación en módulos y servicios, favoreciendo la escalabilidad.
- TypeScript: tipado estático para mejorar la robustez del código y prevenir errores en tiempo de compilación.
- JWT: integración con el backend para autenticar todas las operaciones posteriores al login.

3.4.2. Funcionalidades principales

El frontend integra las siguientes funciones clave:

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

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

3.5. Despliegue del sistema

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

3.5.1. Entorno productivo e integración continua

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

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

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

3.5.2. Monitoreo post-implantación

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

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

3.6. Integración con la infraestructura existente

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

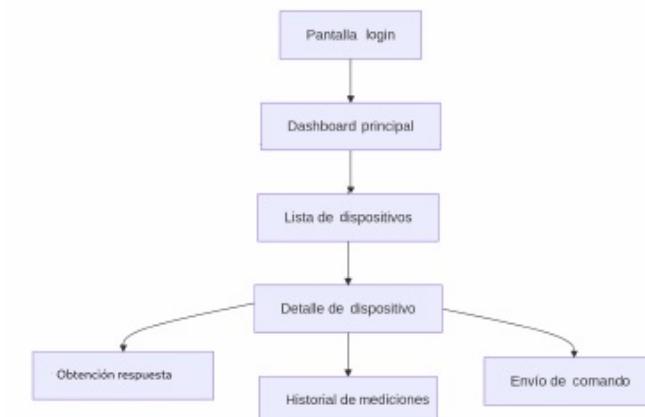


FIGURA 3.7. Diagrama de estructura de los componentes por pantalla.

3.4.3. Integración con el backend

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

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

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

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

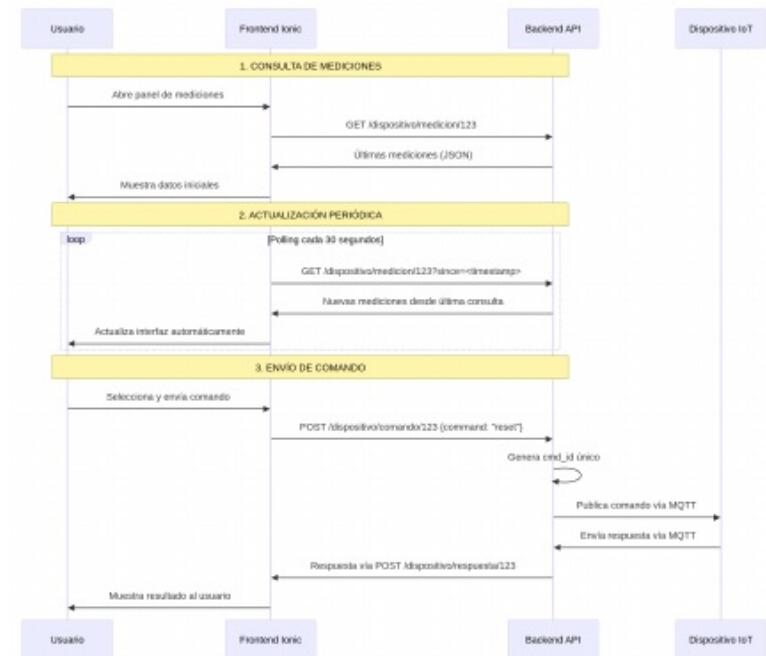


FIGURA 3.8. Diagrama de flujo de comunicación con el backend.

Capítulo 4

Ensayos y Resultados

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

4.1. Banco de pruebas

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

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

4.1.1. Diseño del entorno de pruebas

El banco se compuso de los siguientes elementos principales:

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

El montaje permitió reproducir tres escenarios de prueba diferenciados:

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

3.5. Despliegue del sistema

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

3.5.1. Entorno productivo y configuración

Para simplificar la orquestación se emplea Docker Compose, lo que permite instanciar todos los servicios en contenedores aislados pero comunicados entre sí. Cada componente cumple un rol específico:

- Broker MQTT (Eclipse Mosquitto): configurado como servicio de mensajería central. Se habilitan credenciales de acceso, control de tópicos por dispositivo y soporte de cifrado TLS en despliegues productivos. Su rol es recibir eventos desde los nodos de campo y distribuirlos a los suscriptores autorizados (API REST u otros consumidores).
- API REST (Node.js/Express): implementada como contenedor independiente, integra lógica de negocio y suscripción al broker MQTT. De esta forma, cada evento recibido es validado, transformado y almacenado en la base de datos. La API expone endpoints para:
 - Gestión de dispositivos y usuarios.
 - Consulta de eventos por rango temporal o por dispositivo.
 - Emisión y seguimiento de comandos remotos.
 - Acceso al estado operativo y telemetría de cada nodo.

Todos los endpoints están protegidos mediante autenticación con tokens JWT.

- Base de datos relacional (MySQL): instancia dedicada a persistencia de información. Su esquema incluye tablas de dispositivos, eventos, comandos y usuarios. Se definen índices para optimizar consultas históricas y se configuran respaldos automáticos diarios.
- Interfaz web (Ionic/Angular): desplegada como servicio accesible en navegador. Consuma la API REST para mostrar eventos en tiempo real, ejecutar comandos y consultar históricos.

3.5.2. Monitoreo post-implantación

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

- Logs centralizados: tanto el backend como el broker MQTT registran eventos en archivos y consola. Se integra con Grafana para correlacionar métricas.
- Alertas y métricas: mediante Grafana es posible recolectar indicadores de CPU, memoria y estado de contenedores. También se pueden graficar métricas de tráfico MQTT (mensajes publicados, latencias, pérdidas).

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

4.1.2. Metodología experimental

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

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

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

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

4.1.3. Resultados y observaciones

Los resultados experimentales demostraron que el sistema fue capaz de:

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

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

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

4.2. Pruebas de la API REST

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

- Supervisión de dispositivos: la API REST expone endpoints que informan conectividad y parámetros básicos (nivel de batería, último evento recibido). Estos datos se representan en la interfaz web como panel de salud del sistema.
- Respaldo y recuperación: la base de datos implementa backups automáticos y permite restauraciones parciales. Esto garantiza que el historial de eventos no se pierda ante fallas de hardware o corrupción de datos.

3.6. Integración con la infraestructura existente

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

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

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

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

4.2.1. Objetivos y alcance

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

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

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

4.2.2. Metodología de prueba

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

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

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

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

4.2.3. Resultados obtenidos

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

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.

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

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

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

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

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

4.2.4. Conclusiones

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

4.3. Pruebas de componentes

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

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

4.3.1. Enfoque general

El sistema se evaluó bajo un esquema progresivo:

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

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

4.3.2. Resultados por componente

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

- [36] FHWA. *Traffic Monitoring Guide*. Inf. téc. Federal Highway Administration, 2022. URL: https://www.fhwa.dot.gov/policyinformation/tmguide/2022_TMG_Final_Report.pdf.
- [37] 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>.