.UBAfiuba
FACULTAD DE INGENIERÍA

# Modernización de contadores de tránsito con comunicación bidireccional

Ing. Diego Aníbal Vázquez

**Carrera de Especialización en Internet de las Cosas**

**Director:** Ing. Rogelio Diego González

**Jurados:**

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

*Ciudad de Buenos Aires, marzo de 2026*

.UBAfiuba
FACULTAD DE INGENIERÍA

Modernización de contadores de tránsito con comunicación bidireccional

Ing. Diego Aníbal Vázquez

Carrera de Especialización en Internet de las Cosas

Director: Ing. Rogelio Diego González

Jurados:

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

*Ciudad de Buenos Aires, Marzo de 2026*

# Índice general



v

# Índice general

**Resumen** I

**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 . . . 9
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 . . . 17
3.3.1. Arquitectura y tecnologías . . . 18
3.3.2. Funcionalidades principales . . . 19
3.3.3. Organización en controladores . . . 20
3.3.4. Mapa de endpoints . . . 20
3.3.5. Seguridad y extensibilidad . . . 21
3.4. Desarrollo del frontend . . . 21
3.4.1. Arquitectura y tecnologías . . . 22
3.4.2. Funcionalidades principales . . . 22
3.4.3. Integración con el backend . . . 23
3.5. Despliegue del sistema . . . 25
3.5.1. Entorno productivo y configuración . . . 25
3.5.2. Monitoreo post-implantación . . . 25
3.6. Integración con la infraestructura existente . . . 26

**Bibliografía** 27

# Índice de figuras



VII

# Índice de figuras

- Imposibilidad de actualización remota. Cualquier modificación de parámetros o ajustes de operación requiere intervención en el sitio. Esto incrementa tiempos de mantenimiento, costos logísticos y complica la aplicación rápida de mejoras.
- Falta de telemetría y diagnóstico preventivo. No se dispone de métricas sistemáticas sobre el estado operativo de los equipos (batería, temperatura, errores de hardware o comunicación).
- Riesgo de pérdida de datos ante conectividad intermitente. La ausencia de mecanismos de encolamiento persistente y de políticas claras de reenvío eleva la probabilidad de pérdida o duplicación de eventos cuando la red es inestable.

Estas deficiencias afectan la calidad del servicio de monitoreo, reducen la eficiencia operativa y constituyen los requisitos funcionales que orientan el diseño del prototipo.

### 1.1.3. Impacto esperado

La modernización permite reducir los desplazamientos de personal técnico y acortar los tiempos de resolución de incidentes con la consiguiente disminución de costos logísticos. La incorporación de telemetría facilita la planificación de intervenciones preventivas en lugar de responder exclusivamente a fallas, lo que optimiza la disponibilidad del servicio y la calidad de los datos recolectados. Además, la adopción de protocolos estandarizados y componentes de código abierto promueve la escalabilidad y la replicabilidad de la solución en distintos tramos de la red vial [4] ,[5] ,[6].

### 1.1.4. Diseño conceptual

La propuesta separa de forma explícita el transporte de mensajes (broker MQTT) de los servicios de aplicación (API REST, almacenamiento y frontend). Esta separación facilita la interoperabilidad con plataformas institucionales existentes y habilita opciones de despliegue flexibles: uso de brokers externos, instalación de brokers locales o modelos híbridos según políticas institucionales y condiciones de conectividad.

## 1.2. Objetivos

### 1.2.1. Objetivo general

Modernizar los contadores de tránsito en rutas nacionales mediante la renovación de su arquitectura de comunicaciones para habilitar comunicación bidireccional confiable, control remoto y telemetría de estado.

### 1.2.2. Objetivos específicos

A continuación, se detallan los objetivos específicos que orientan el desarrollo del trabajo:

- Garantizar la entrega de eventos aun con conectividad intermitente.

2 Capítulo 1. Introducción general

- Imposibilidad de actualización remota. Cualquier modificación de parámetros o ajustes de operación requiere intervención en el sitio. Esto incrementa tiempos de mantenimiento, costos logísticos y complica la aplicación rápida de mejoras.
- Falta de telemetría y diagnóstico preventivo. No se dispone de métricas sistemáticas sobre el estado operativo de los equipos (batería, temperatura, errores de hardware o comunicación).
- Riesgo de pérdida de datos ante conectividad intermitente. La ausencia de mecanismos de encolamiento persistente y de políticas claras de reenvío eleva la probabilidad de pérdida o duplicación de eventos cuando la red es inestable.

Estas deficiencias afectan la calidad del servicio de monitoreo, reducen la eficiencia operativa y constituyen los requisitos funcionales que orientan el diseño del prototipo.

### 1.1.3. Impacto esperado

La modernización permite reducir los desplazamientos de personal técnico y acortar los tiempos de resolución de incidentes con la consiguiente disminución de costos logísticos. La incorporación de telemetría facilita la planificación de intervenciones preventivas en lugar de responder exclusivamente a fallas, lo que optimiza la disponibilidad del servicio y la calidad de los datos recolectados. Además, la adopción de protocolos estandarizados y componentes de código abierto promueve la escalabilidad y la replicabilidad de la solución en distintos tramos de la red vial [4] ,[5] ,[6].

### 1.1.4. Diseño conceptual

La propuesta separa de forma explícita el transporte de mensajes (broker MQTT) de los servicios de aplicación (API REST, almacenamiento y frontend). Esta separación facilita la interoperabilidad con plataformas institucionales existentes y habilita opciones de despliegue flexibles: uso de brokers externos, instalación de brokers locales o modelos híbridos según políticas institucionales y condiciones de conectividad.

## 1.2. Objetivos

### 1.2.1. Objetivo general

Modernizar los contadores de tránsito en rutas nacionales mediante la renovación de su arquitectura de comunicaciones para habilitar comunicación bidireccional confiable, control remoto y telemetría de estado.

### 1.2.2. Objetivos específicos

A continuación se detallan los objetivos específicos que orientan el desarrollo del trabajo:

- Garantizar la entrega de eventos aun con conectividad intermitente.

La combinación de estos protocolos permite cubrir distintos niveles de la arquitectura: comunicación local (RS-232), comunicación de dispositivos con el servidor (MQTT) y comunicación entre el servidor y las aplicaciones de usuario (API REST). De esta forma, se garantiza un flujo de datos seguro, confiable y bidireccional.

## 2.2. Componentes de hardware utilizados

El prototipo se implementa con un conjunto de componentes de hardware seleccionados por su disponibilidad, costo y adecuación al entorno de operación:

- Contador de tránsito: dispositivo de campo encargado de detectar y acumular el paso de vehículos y generar tramas de datos que contienen información de conteo, clasificación y marcas temporales. Estos equipos permiten registrar de manera continua el flujo vehicular en rutas nacionales y constituyen la base del sistema de monitoreo. Si bien en el mercado existen distintos contadores comerciales que ofrecen funcionalidades similares, muchos de ellos presentan altos costos de adquisición, dependencia de plataformas propietarias o limitaciones de integración. En este trabajo se optó por utilizar un contador desarrollado por la Dirección Nacional de Vialidad, que ha sido probado en distintos rutas nacionales del país y cuenta con una interfaz de salida RS-232 [15] con un adaptador MAX232 [20]. Esta elección responde a criterios de soberanía tecnológica, optimización de recursos ya disponibles y reducción de costos. Además, el contador no solo transmite información de eventos, sino que también admite la recepción de comandos externos a través del nodo, lo que habilita funcionalidades de diagnóstico, reconfiguración remota y confirmación de estado. De este forma, se garantiza la compatibilidad con la infraestructura existente y se potencia su integración dentro de una arquitectura moderna basada en protocolos abiertos.

  En la figura 2.1 se observa el contador de tránsito. Este dispositivo constituye la base del sistema de detección de eventos.

FIGURA 2.1. Contador de tránsito DTEC [1].

6 Capítulo 2. Introducción específica

La combinación de estos protocolos permite cubrir distintos niveles de la arquitectura: comunicación local (RS-232), comunicación de dispositivos con el servidor (MQTT) y comunicación entre el servidor y las aplicaciones de usuario (API REST). De esta forma, se garantiza un flujo de datos seguro, confiable y bidireccional.

## 2.2. Componentes de hardware utilizados

El prototipo se implementa con un conjunto de componentes de hardware seleccionados por su disponibilidad, costo y adecuación al entorno de operación:

- Contador de tránsito: dispositivo de campo encargado de detectar y acumular el paso de vehículos y generar tramas de datos que contienen información de conteo, clasificación y marcas temporales. Estos equipos permiten registrar de manera continua el flujo vehicular en rutas nacionales y constituyen la base del sistema de monitoreo. Si bien en el mercado existen distintos contadores comerciales que ofrecen funcionalidades similares, muchos de ellos presentan altos costos de adquisición, dependencia de plataformas propietarias o limitaciones de integración. En este trabajo se optó por utilizar un contador desarrollado por la Dirección Nacional de Vialidad, cual ha sido probado en distintos rutas nacionales del país y cuenta con una interfaz de salida RS-232 [15] con un adaptador MAX232 [20]. Esta elección responde a criterios de soberanía tecnológica, optimización de recursos ya disponibles y reducción de costos. Además, el contador no solo transmite información de eventos, sino que también admite la recepción de comandos externos a través del nodo, lo que habilita funcionalidades de diagnóstico, reconfiguración remota y confirmación de estado. De este forma, se garantiza la compatibilidad con la infraestructura existente y se potencia su integración dentro de una arquitectura moderna basada en protocolos abiertos.

  En la figura 2.1 se observa el contador de tránsito. Este dispositivo constituye la base del sistema de detección de eventos.

- Módulo de adaptación RS-232/TTL (MAX232): este circuito se encarga de convertir los niveles de tensión de forma bidireccional, protege los dispositivos y garantiza una transmisión de datos confiable y libre de errores [20].

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


FIGURA 2.2. Módulo RS-232/TTL [2].

- ESP32-C3: microcontrolador de bajo consumo que actúa como unidad de procesamiento y comunicación. Integra conectividad y capacidad de comunicación serial, lo que lo hace adecuado para la interacción con el contador y con el módulo GPRS [21].

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


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

- Módulo GPRS SIM800L: permite la conexión de los dispositivos a la red celular [22], que asegura la transmisión de datos al servidor central mediante

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

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

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

2.2. Componentes de hardware utilizados 7

FIGURA 2.1. Contador de tránsito DTEC [1].

- Módulo de adaptación RS-232/TTL (MAX232): circuito que permite la correcta comunicación entre el contador y el microcontrolador que asegura compatibilidad entre la interfaz serial del contador (RS-232) y las señales TTL requeridas por el microcontrolador. Este circuito se encarga de convertir los niveles de tensión de manera bidireccional y protege los dispositivos y garantizando una transmisión de datos confiable y sin errores.

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


FIGURA 2.2. Módulo RS-232/TTL [2].

- ESP32-C3: microcontrolador de bajo consumo que actúa como unidad de procesamiento y comunicación. Integra conectividad y capacidad de comunicación serial, lo que lo hace adecuado para la interacción con el contador y con el módulo GPRS [21].

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

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

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

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

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


FIGURA 2.4. Módulo SIM800L [4].

- Servidor central: responsable de recibir los datos transmitidos, almacenarlos en una base de datos y responder a las solicitudes de los usuarios.
- PC con navegador web: constituye el medio de interacción del usuario final con el sistema, a través de la interfaz desarrollada.
- Fuente de alimentación: fuente principal con batería interna recargable y recarga mediante panel solar, capaz de cubrir los picos de consumo del SIM800L. Incluye regulador de tensión y capacitores que estabilizan el voltaje y evitan reinicios.
- Carcasa y gabinete: protección para el contador y el módulo de comunicación (ESP32-C3 y SIM800L) en un gabinete cerrado resistente a humedad, polvo y vibraciones, con disipación térmica y reducción de interferencias electromagnéticas.
- Componentes adicionales: filtros (capacitores) para garantizar estabilidad y evitar reinicios inesperados.

La integración de estos componentes asegura que el sistema pueda operar en condiciones reales y que cumpla con los requisitos.

## 2.3. Tecnologías de software aplicadas

El desarrollo del prototipo se sustenta en la integración de tecnologías de software abiertas y estandarizadas, seleccionadas por su solidez, amplia adopción en la comunidad tecnológica y capacidad para interoperar de manera eficiente entre los distintos componentes del sistema:

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



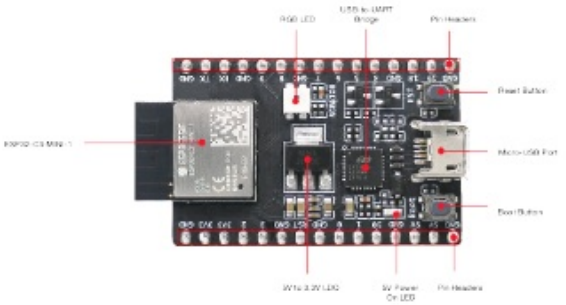

FIGURA 2.3. Microcontrolador ESP32-C3 utilizado en los nodos de campo [3].

- Módulo GPRS SIM800L: permite la conexión de los dispositivos a la red celular [22], que asegura la transmisión de datos al servidor central mediante MQTT. Se eligió por su bajo costo, disponibilidad en el mercado y facilidad de integración con el ESP32-C3.

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


FIGURA 2.4. Módulo SIM800L [4].

- Servidor central: responsable de recibir los datos transmitidos, almacenarlos en una base de datos y responder a las solicitudes de los usuarios.
- PC con navegador web: constituye el medio de interacción del usuario final con el sistema, a través de la interfaz desarrollada.

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

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

- MQTT con Eclipse Mosquitto. La mensajería ligera y bidireccional se implementa mediante el protocolo MQTT, operando sobre el broker Eclipse Mosquitto [23]. Este software es ampliamente utilizado en entornos de Internet de las Cosas (IoT).
- API REST con Node.js y Express. El backend se desarrolla en Node.js [24], un entorno de ejecución orientado a aplicaciones de red no bloqueantes, y se estructura con Express [25], un framework ligero que facilita la creación de servicios RESTful.
- Base de datos relacional MySQL. La persistencia de los datos se implementa en MySQL [26], un sistema gestor de bases de datos relacionales ampliamente adoptado. En ella se almacenan de manera estructurada tanto los eventos de tránsito como los estados reportados por los equipos.
- ORM con Sequelize. Para abstraer la interacción con la base de datos y mantener independencia frente a cambios en la capa de persistencia, se utiliza Sequelize [27], un ORM para Node.js que permite definir modelos y relaciones de manera declarativa.
- Sistema de logging con Winston. La trazabilidad de eventos y errores se gestiona mediante Winston [28], una biblioteca de logging que soporta múltiples transportes y permite configurar niveles de severidad, formatos y timestamps.
- Middleware de registro HTTP con Morgan. Para capturar y auditar el tráfico entrante al backend se emplea Morgan [29], un middleware especializado en logging de peticiones HTTP, que complementa funcionalidad de Winston.
- Interfaz web con Ionic y Angular. Para la interacción con el usuario final se diseñó una aplicación accesible desde cualquier navegador, construida con Ionic [30] y Angular [31]. La primera aporta componentes visuales responsivos que aseguran usabilidad tanto en entornos de escritorio como en dispositivos móviles.
- Orquestación con Docker Compose. Para simplificar el despliegue y la gestión de los servicios del backend, se utiliza Docker Compose [32]. Esto permite levantar de manera consistente los contenedores de la API REST, el broker MQTT y la base de datos MySQL, que asegura que todas las dependencias se inicien en el orden correcto y facilita la replicación del entorno en desarrollo, prueba y producción.
- Firmware para ESP32-C3 con ESP-IDF. El microcontrolador ejecuta un firmware desarrollado en C/C++, el cual utiliza el framework oficial de ESP-IDF [33], el framework oficial de Espressif. Este entorno proporciona bibliotecas optimizadas. El firmware controla la captura de datos desde el contador mediante RS-232, gestiona la comunicación con el módulo SIM800L mediante comandos AT y establece la conexión con el broker Mosquitto a través de MQTT.

2.3. Tecnologías de software aplicadas 9

- Fuente de alimentación: fuente principal con batería interna recargable y recarga mediante panel solar, capaz de cubrir los picos de consumo del SIM800L. Incluye regulador de tensión y capacitores que estabilizan el voltaje y evitan reinicios.
- Carcasa y gabinete: protección para el contador y el módulo de comunicación (ESP32-C3 y SIM800L) en un gabinete cerrado resistente a humedad, polvo y vibraciones, con disipación térmica y reducción de interferencias electromagnéticas.
- Componentes adicionales: filtros (capacitores) para garantizar estabilidad y evitar reinicios inesperados.

La integración de estos componentes asegura que el sistema pueda operar en condiciones reales y que cumpla con los requisitos.

## 2.3. Tecnologías de software aplicadas

El desarrollo del prototipo se sustenta en la integración de tecnologías de software abiertas y estandarizadas, seleccionadas por su solidez, amplia adopción en la comunidad tecnológica y capacidad para interoperar de manera eficiente entre los distintos componentes del sistema:

- MQTT con Eclipse Mosquitto. La mensajería ligera y bidireccional se implementa mediante el protocolo MQTT, operando sobre el broker Eclipse Mosquitto [23]. Este software es ampliamente utilizado en entornos de Internet de las Cosas (IoT).
- API REST con Node.js y Express. El backend se desarrolla en Node.js [24], un entorno de ejecución orientado a aplicaciones de red no bloqueantes, y se estructura con Express [25], un framework ligero que facilita la creación de servicios RESTful.
- Base de datos relacional MySQL. La persistencia de los datos se implementa en MySQL [26], un sistema gestor de bases de datos relacionales ampliamente adoptado. En ella se almacenan de manera estructurada tanto los eventos de tránsito como los estados reportados por los equipos.
- ORM con Sequelize. Para abstraer la interacción con la base de datos y mantener independencia frente a cambios en la capa de persistencia, se utiliza Sequelize [27], un ORM para Node.js que permite definir modelos y relaciones de manera declarativa.
- Sistema de logging con Winston. La trazabilidad de eventos y errores se gestiona mediante Winston [28], una biblioteca de logging que soporta múltiples transportes y permite configurar niveles de severidad, formatos y timestamps.
- Middleware de registro HTTP con Morgan. Para capturar y auditar el tráfico entrante al backend se emplea Morgan [29], un middleware especializado en logging de peticiones HTTP, que complementa funcionalidad de Winston.
- Interfaz web con Ionic y Angular. Para la interacción con el usuario final se diseñó una aplicación accesible desde cualquier navegador, construida

## 2.4. Software de control de versiones

Para la gestión del código fuente se empleó GitHub [34], una plataforma que se basa en el sistema de control de versiones Git. Esta herramienta permitió almacenar el repositorio central de manera segura, registrar el historial de cambios y garantizar la trazabilidad de cada modificación realizada durante la etapa de desarrollo del prototipo.

El repositorio incluye el firmware del ESP32-C3, el backend en Node.js con Express y la interfaz web desarrollada con Ionic y Angular. Esto facilita mantener el código organizado, con la posibilidad de crear ramas independientes para implementar nuevas funciones y, posteriormente, integrarlas al código principal mediante pull requests, lo que permite revisar y validar los cambios antes de su incorporación definitiva.

Asimismo, se emplean issues para documentar incidencias, planificar tareas y realizar el seguimiento de los avances. Esta práctica favorece la organización del trabajo y permite mantener un registro claro de los problemas detectados y las decisiones adoptadas.

10 Capítulo 2. Introducción específica

con Ionic [30] y Angular [31]. La primera aporta componentes visuales responsivos que aseguran usabilidad tanto en entornos de escritorio como en dispositivos móviles.

- Orquestación con Docker Compose. Para simplificar el despliegue y la gestión de los servicios del backend, se utiliza Docker Compose [32]. Esto permite levantar de manera consistente los contenedores de la API REST, el broker MQTT y la base de datos MySQL, que asegura que todas las dependencias se inicien en el orden correcto y facilita la replicación del entorno en desarrollo, prueba y producción.
- Firmware para ESP32-C3 con ESP-IDF. El microcontrolador ejecuta un firmware desarrollado en C/C++, el cual utiliza el framework oficial de ESP-IDF [33], el framework oficial de Espressif. Este entorno proporciona bibliotecas optimizadas. El firmware controla la captura de datos desde el contador mediante RS-232, gestiona la comunicación con el módulo SIM800L mediante comandos AT y establece la conexión con el broker Mosquitto a través de MQTT.

## 2.4. Software de control de versiones

Para la gestión del código fuente se empleó GitHub [34], una plataforma que se basa en el sistema de control de versiones Git. Esta herramienta permitió almacenar el repositorio central de manera segura, registrar el historial de cambios y garantizar la trazabilidad de cada modificación realizada durante la etapa de desarrollo del prototipo.

El repositorio incluye el firmware del ESP32-C3, el backend en Node.js con Express y la interfaz web desarrollada con Ionic y Angular. Esto facilita mantener el código organizado, con la posibilidad de crear ramas independientes para implementar nuevas funciones y, posteriormente, integrarlas al código principal mediante pull requests, lo que permite revisar y validar los cambios antes de su incorporación definitiva.

Asimismo, se emplean issues para documentar incidencias, planificar tareas y realizar el seguimiento de los avances. Esta práctica favorece la organización del trabajo y permite mantener un registro claro de los problemas detectados y las decisiones adoptadas.

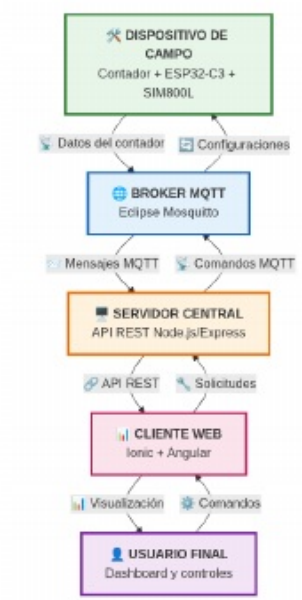


FIGURA 3.1. Diagrama de arquitectura del sistema y el flujo de datos.

### 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 el evento en el tópico MQTT `devices/id/events`.
- 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.

3.1. Arquitectura del sistema 13

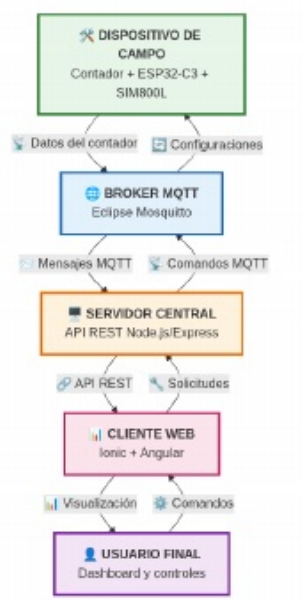


FIGURA 3.1. Diagrama de arquitectura del sistema y el flujo de datos.

### 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, asegurando 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 el evento en el tópico MQTT `devices/id/events`.
- 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 /api/devices/id/comm` al backend, que crea un `cmd_id` único y publica en `devices/id/comm`.
- Recepción nodo/Entrega al contador: el ESP32-C3 en `devices/id/comm` 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 `devices/id/status` 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 `comm` (campo `status`, `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.

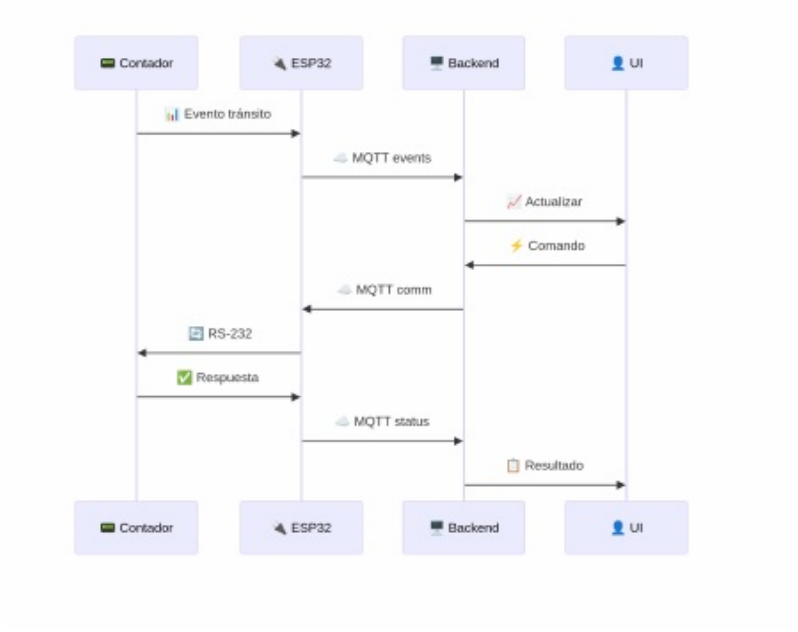


FIGURA 3.2. Diagrama de secuencia del flujo de datos.

## 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:

14 Capítulo 3. Diseño e implementación

- Emisión de comandos (desde UI): el operador genera un comando en la interfaz, la UI envía `POST /api/devices/id/comm` al backend, que crea un `cmd_id` único y publica en `devices/id/comm`.
- Recepción nodo/Entrega al contador: el ESP32-C3 en `devices/id/comm` 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 `devices/id/status` 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 `comm` (campo `status`, `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.

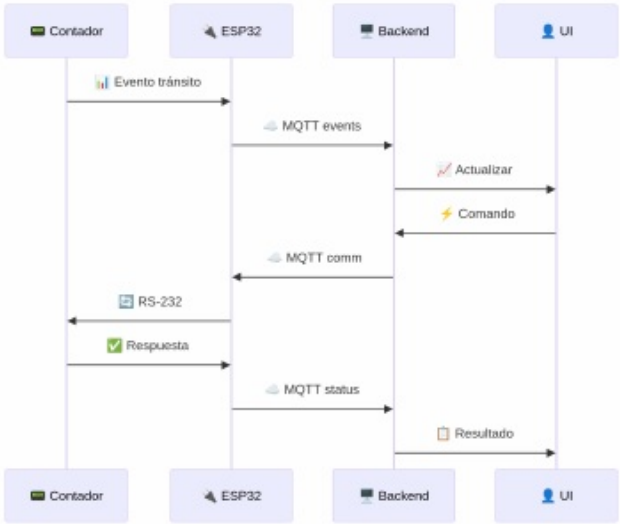


FIGURA 3.2. Diagrama de secuencia del flujo de datos.

## 3.2. Arquitectura del nodo

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

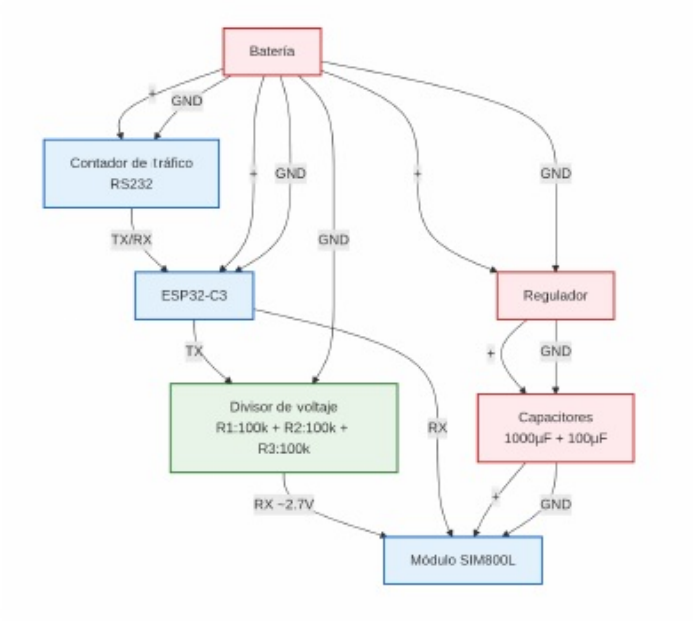


FIGURA 3.3. Diagrama de conexión entre los módulos del sistema.

- 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.

16 Capítulo 3. Diseño e implementación

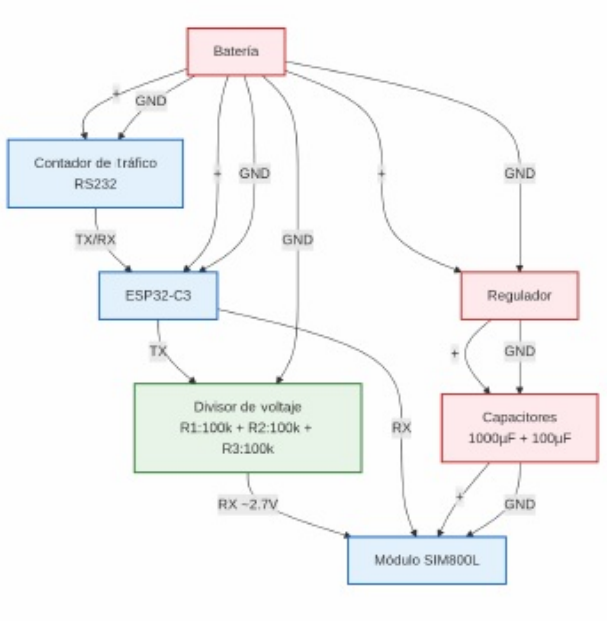


FIGURA 3.3. Diagrama de conexión entre los módulos del sistema.

- 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.


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.

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

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).

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

Cada vez que un dispositivo publica un evento en `devices/{id}/events`, el backend lo recibe, valida el payload y lo persiste en MySQL.

En el backend, los comandos se publican en (`devices/id/comm`) y el estado se actualiza según las confirmaciones (`devices/id/status`).

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.

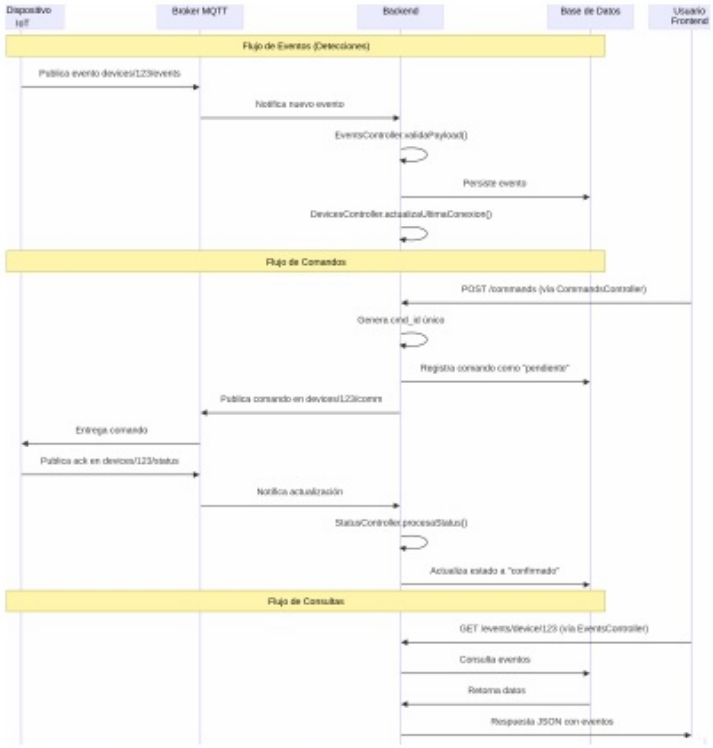


FIGURA 3.5. Diagrama de flujo de información del Backend.

18 Capítulo 3. Diseño e implementación

### 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.

Cada vez que un dispositivo publica un evento en `devices/{id}/events`, el backend lo recibe, valida el payload y lo persiste en MySQL.

En el backend, los comandos se publican en (`devices/id/comm`) y el estado se actualiza según las confirmaciones (`devices/id/status`).

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.

### 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:

- DevicesController: gestiona las operaciones CRUD sobre los dispositivos de campo, además de registrar los eventos recibidos vía MQTT y asociarlos a un dispositivo específico.
- EventsController: encapsula la lógica de ingesta de eventos de tránsito, validación de payloads y persistencia en la base de datos.
- CommandsController: administra la emisión y seguimiento de comandos remotos, generando un `cmd_id` único y actualizando el estado conforme se reciben los `ack`.
- StatusController: 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.

3.3. Desarrollo del backend 19

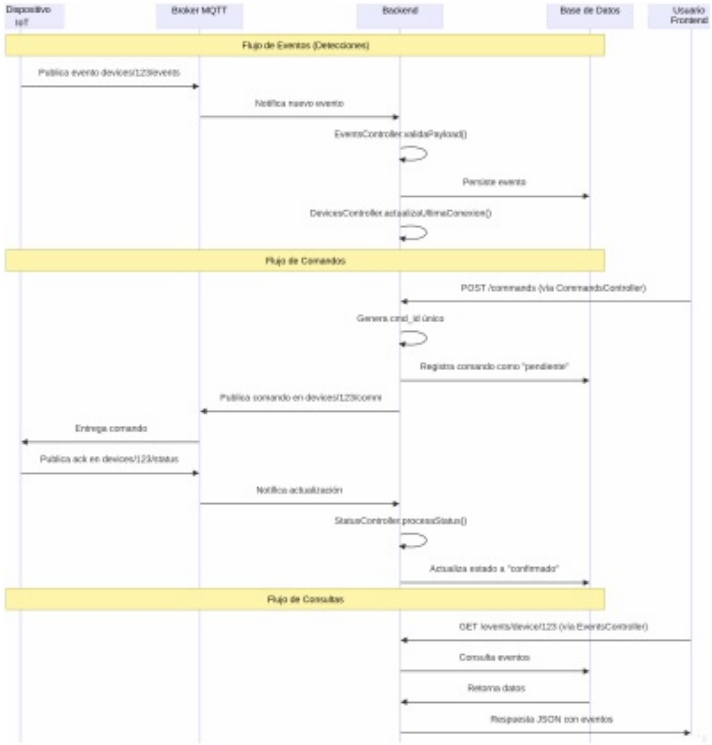


FIGURA 3.5. Diagrama de flujo de información del Backend.

### 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.

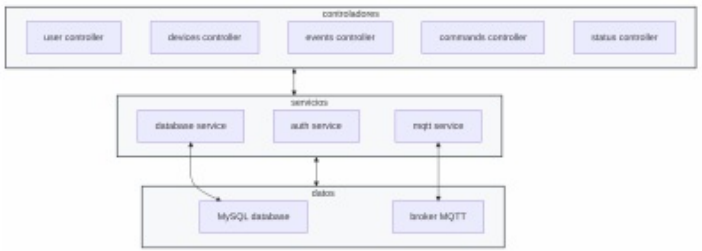


FIGURA 3.6. Diagrama con la disposición de los controladores y flujo de dependencias.

### 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 sus respectivos controladores:



### 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:

- DevicesController: gestiona las operaciones CRUD sobre los dispositivos de campo, además de registrar los eventos recibidos vía MQTT y asociarlos a un dispositivo específico.
- EventsController: encapsula la lógica de ingesta de eventos de tránsito, validación de payloads y persistencia en la base de datos.
- CommandsController: administra la emisión y seguimiento de comandos remotos, generando un `cmd_id` único y actualizando el estado conforme se reciben los `ack`.
- StatusController: 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.

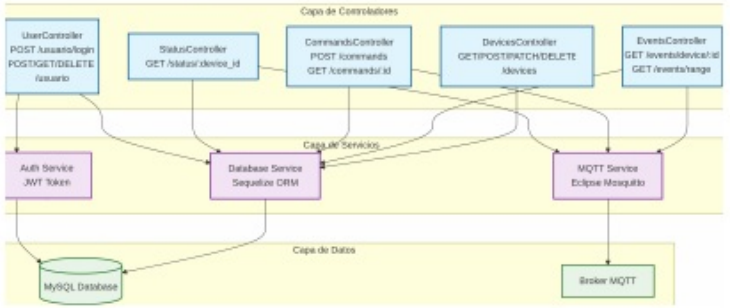


FIGURA 3.6. Diagrama con la disposición de los controladores y flujo de dependencias.

### 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 sus respectivos controladores: