

# Índice general

<b>Resumen</b>	1
<b>1. Introducción general</b>	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
<b>2. Introducción específica</b>	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
<b>3. Diseño e implementación</b>	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
<b>Bibliografía</b>	27

# Índice general

<b>Resumen</b>	1
<b>1. Introducción general</b>	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
<b>2. Introducción específica</b>	5
2.1. Protocolos y comunicación . . . . .	5
2.2. Componentes de hardware utilizados . . . . .	6
2.3. Tecnologías de software aplicadas . . . . .	7
2.4. Software de control de versiones . . . . .	8
<b>3. Diseño e implementación</b>	11
3.1. Arquitectura del sistema . . . . .	11
3.1.1. Flujo de datos . . . . .	11
3.1.2. Descripción ampliada de bloques y responsabilidades . . . . .	12
3.2. Detalle de módulos de hardware . . . . .	15
3.2.1. ESP32-C3 (unidad de control) . . . . .	15
3.2.2. Módulo GPRS SIM800L . . . . .	15
3.2.3. Contador de tránsito y comunicación RS-232 . . . . .	16
3.2.4. Alimentación y montaje . . . . .	17
3.3. Desarrollo del backend . . . . .	19
3.3.1. Arquitectura y tecnologías . . . . .	19
3.3.2. Funcionalidades principales . . . . .	21
3.3.3. Organización en controladores . . . . .	21
3.3.4. Mapa de endpoints . . . . .	23
3.3.5. Seguridad y extensibilidad . . . . .	23
3.4. Desarrollo del frontend . . . . .	23
3.4.1. Arquitectura y tecnologías . . . . .	24
3.4.2. Funcionalidades principales . . . . .	24
3.4.3. Integración con el backend . . . . .	26
3.5. Despliegue del sistema . . . . .	28
3.5.1. Entorno productivo y configuración . . . . .	28
3.5.2. Monitoreo post-implantación . . . . .	28

3.6.	Nodos y Sensores	29
3.6.1.	Arquitectura del nodo	29
3.6.2.	Integración con la infraestructura existente	31

Bibliografía

33

## Índice de figuras

1.1. Diagrama en bloques del sistema.	4
2.1. Contador de tránsito DTEC <sup>1</sup> .	7
2.2. Módulo RS-232/TTL <sup>2</sup> .	7
2.3. Microcontrolador ESP32-C3 utilizado en los nodos de campo <sup>3</sup> .	8
2.4. Módulo SIM800L <sup>4</sup> .	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 <sup>5</sup> .	17
3.5. Diagrama de flujo de información del Backend.	19
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 figuras

1.1. Diagrama en bloques del sistema.	4
2.1. Contador de tránsito DTEC <sup>1</sup> .	7
3.1. Diagrama de secuencia del flujo de datos.	12
3.2. Diagrama de arquitectura del sistema y el flujo de datos.	14
3.3. Contador de tránsito DTEC <sup>2</sup> .	16
3.4. Diagrama de conexión entre los módulos del sistema.	17
3.5. Fotografía contador de tránsito DTEC instalado en campo <sup>3</sup> .	18
3.6. Diagrama de flujo de Información del Backend.	20
3.7. Diagrama con la disposición de los controladores y flujo de dependencias.	22
3.8. Diagrama de estructura de los componentes por pantalla.	25
3.9. Diagrama de flujo de Comunicación con el backend.	27
3.10. Contador de tránsito DTEC <sup>4</sup> .	30
3.11. Módulo RS-232/TTL <sup>5</sup> .	30
3.12. Microcontrolador ESP32-C3 utilizado en los nodos de campo <sup>6</sup> .	30
3.13. Módulo SIM800L <sup>7</sup> .	31

## Índice de tablas

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

## Índice de tablas

3.1. Endpoints REST principales . . . . .	23
---	----

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

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

### 1.1.3. Impacto esperado

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

### 1.1.4. Diseño conceptual

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

## 1.2. Objetivos

### 1.2.1. Objetivo general

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

### 1.2.2. Objetivos específicos

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

- Garantizar la entrega de eventos aun con conectividad intermitente.

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

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

### 1.1.3. Impacto esperado

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

### 1.1.4. Diseño conceptual

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

## 1.2. Objetivos

### 1.2.1. Objetivo general

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

### 1.2.2. Objetivos específicos

- Garantizar la entrega de eventos aun con conectividad intermitente.
- Implementar mecanismos de encolado y reintento que eviten duplicaciones y pérdidas de información.

**1.3. Estado del arte y propuesta de valor**

3

- Implementar mecanismos de encolado y reintentos que eviten duplicaciones y pérdidas de información.
- Habilitar la ejecución remota de comandos y la actualización de parámetros desde un servidor central, con confirmación explícita del estado del equipo.
- Disminuir la dependencia de enlaces tercerizados mediante una arquitectura configurable.
- Permitir la modificación remota de parámetros operativos y la carga de ajustes sin desplazamientos.
- Incorporar telemetría de estado (batería, temperatura y códigos de error) para mantenimiento preventivo.
- Validar la viabilidad técnica y la robustez operativa del sistema.

**1.3. Estado del arte y propuesta de valor**

En el mercado existen soluciones comerciales [7], [8], que ofrecen gestión remota y comunicación bidireccional para dispositivos de campo. Dichas soluciones suelen incluir plataformas propietarias con soporte técnico, servicios administrados y herramientas de análisis avanzadas. Estas alternativas, sin embargo, presentan costos elevados de adquisición y mantenimiento, así como dependencias tecnológicas que restringen la posibilidad de adaptación a condiciones locales específicas.

En el ámbito académico y técnico también se han documentado diversas experiencias orientadas a la gestión remota de infraestructuras distribuidas mediante protocolos de mensajería ligera como MQTT o tecnologías de comunicación celular [9], [10]. Estos trabajos resaltan la eficiencia de los modelos de publicación y suscripción, especialmente en entornos con restricciones de conectividad, pero en muchos casos se centran en aplicaciones industriales que difieren de las condiciones propias de las rutas nacionales [11].

Además, se han desarrollado plataformas de monitoreo basadas en API REST y bases de datos relacionales, que permiten la integración con interfaces web para la visualización y control [12]. Estas experiencias muestran la tendencia hacia arquitecturas abiertas y modulares, aunque su adopción práctica suele estar ligada a contextos con mayor disponibilidad de infraestructura y recursos.

El estado del arte muestra soluciones maduras para la gestión remota y la comunicación bidireccional, aunque con limitaciones asociadas a costos elevados, rigidez tecnológica o requerimientos de infraestructura. Estas restricciones evidencian la necesidad de alternativas específicas y adaptadas a entornos viales argentinos, donde la conectividad suele ser intermitente.

**1.4. Alcance**

El trabajo desarrolla un prototipo funcional destinado a validar las hipótesis técnicas y operativas. El alcance comprende los siguientes objetivos y límites:

**1.3. Estado del arte y propuesta de valor**

3

- Habilitar la ejecución remota de comandos y la actualización de parámetros desde un servidor central, con confirmación explícita del estado del equipo.
- Disminuir la dependencia de enlaces tercerizados mediante una arquitectura configurable.
- Permitir la modificación remota de parámetros operativos y la carga de ajustes sin desplazamientos.
- Incorporar telemetría de estado (batería, temperatura y códigos de error) para mantenimiento preventivo.
- Validar la viabilidad técnica y la robustez operativa del sistema.

**1.3. Estado del arte y propuesta de valor**

En el mercado existen soluciones comerciales [7], [8], que ofrecen gestión remota y comunicación bidireccional para dispositivos de campo. Dichas soluciones suelen incluir plataformas propietarias con soporte técnico, servicios administrados y herramientas de análisis avanzadas. Estas alternativas, sin embargo, presentan costos elevados de adquisición y mantenimiento, así como dependencias tecnológicas que restringen la posibilidad de adaptación a condiciones locales específicas.

En el ámbito académico y técnico también se han documentado diversas experiencias orientadas a la gestión remota de infraestructuras distribuidas mediante protocolos de mensajería ligera como MQTT o tecnologías de comunicación celular [9], [10]. Estos trabajos resaltan la eficiencia de los modelos de publicación y suscripción, especialmente en entornos con restricciones de conectividad, pero en muchos casos se centran en aplicaciones industriales que difieren de las condiciones propias de las rutas nacionales [11].

Además, se han desarrollado plataformas de monitoreo basadas en API REST y bases de datos relacionales, que permiten la integración con interfaces web para la visualización y control [12]. Estas experiencias muestran la tendencia hacia arquitecturas abiertas y modulares, aunque su adopción práctica suele estar ligada a contextos con mayor disponibilidad de infraestructura y recursos.

En síntesis, el estado del arte muestra soluciones maduras para la gestión remota y la comunicación bidireccional, aunque con limitaciones asociadas a costos elevados, rigidez tecnológica o requerimientos de infraestructura. Estas restricciones evidencian la necesidad de alternativas específicas y adaptadas a entornos viales argentinos, donde la conectividad suele ser intermitente.

**1.4. Alcance**

El trabajo desarrolla un prototipo funcional destinado a validar las hipótesis técnicas y operativas. El alcance comprende los siguientes objetivos y límites:

- Rediseño de comunicaciones: implementación de un modelo bidireccional y seguro entre los dispositivos de conteo y el servidor central, basado en MQTT sobre GPRS y con autenticación por credenciales.

- Rediseño de comunicaciones: implementación de un modelo bidireccional y seguro entre los dispositivos de conteo y el servidor central, basado en MQTT sobre GPRS y con autenticación por credenciales.
- Gestión de mensajes en el dispositivo: encolamiento en memoria RAM con política FIFO, control de reintentos ante fallos de conexión y lógica de descarte cuando la cola alcance su límite definido.
- Backend y persistencia: desarrollo de una API REST que se suscriba al broker MQTT, procese los eventos y almacene los registros en una base de datos relacional para consulta histórica.
- Interfaz web básica: panel de visualización en tiempo real de eventos de tránsito y módulo para envío de comandos y verificación de estado de los dispositivos.
- Funciones de telemetría: reporte de nivel de batería, temperatura y códigos de error para facilitar el diagnóstico remoto.

El alcance técnico incluye la adaptación de un contador existente para comunicarse bidireccionalmente por GPRS, que emplee MQTT como protocolo de mensajería ligera y confiable. Asimismo, comprende el desarrollo de un backend que reciba, procese y persista los eventos en una base de datos relacional. Además, una interfaz web básica que permita la visualización en tiempo real y el envío de comandos hacia los dispositivos. Se priorizan las funciones que validen la cadena completa: adquisición de eventos desde el sistema de detección, encolamiento local con política FIFO, reenvío seguro cuando haya conectividad, recepción y ejecución de comandos y verificación del estado del dispositivo tras cada acción.

La figura 1.1 muestra el diagrama en bloques del sistema. El dispositivo de conteo envía datos por GPRS a un broker MQTT, el servidor central los procesa y almacena en una base de datos, mientras que una API REST permite su consulta y la ejecución de comandos desde una interfaz web.

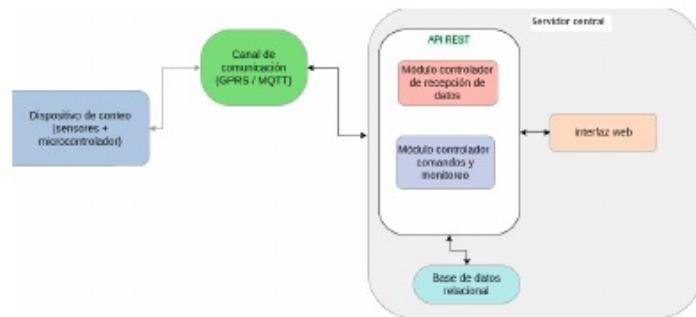


FIGURA 1.1. Diagrama en bloques del sistema.

- Gestión de mensajes en el dispositivo: encolamiento en memoria RAM con política FIFO, control de reintentos ante fallos de conexión y lógica de descarte cuando la cola alcance su límite definido.
- Backend y persistencia: desarrollo de una API REST que se suscriba al broker MQTT, procese los eventos y almacene los registros en una base de datos relacional para consulta histórica.
- Interfaz web básica: panel de visualización en tiempo real de eventos de tránsito y módulo para envío de comandos y verificación de estado de los dispositivos.
- Funciones de telemetría: reporte de nivel de batería, temperatura y códigos de error para facilitar el diagnóstico remoto.

El alcance técnico incluye la adaptación de un contador existente para comunicarse bidireccionalmente por GPRS, que emplee MQTT como protocolo de mensajería ligera y confiable. Asimismo, comprende el desarrollo de un backend que reciba, procese y persista los eventos en una base de datos relacional. Además, una interfaz web básica que permita la visualización en tiempo real y el envío de comandos hacia los dispositivos. Se priorizan las funciones que validen la cadena completa: adquisición de eventos desde el sistema de detección, encolamiento local con política FIFO, reenvío seguro cuando haya conectividad, recepción y ejecución de comandos y verificación del estado del dispositivo tras cada acción.

La figura 1.1 muestra el diagrama en bloques del sistema. El dispositivo de conteo envía datos por GPRS a un broker MQTT, el servidor central los procesa y almacena en una base de datos, mientras que una API REST permite su consulta y la ejecución de comandos desde una interfaz web.

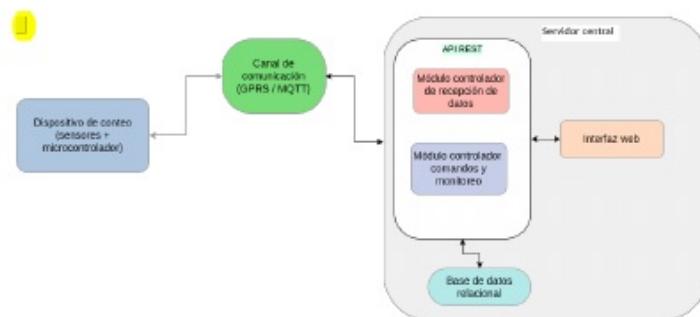


FIGURA 1.1. Diagrama en bloques del sistema.

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.

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 proyecto se optó por utilizar un contador desarrollado por la Dirección Nacional de Vialidad, el cual ha sido probado en distintos corredores viales 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 3.3 se observa el contador de tránsito. Este dispositivo constituye la base del sistema de detección de eventos.

- 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].
- Módulo GPRS SIM800L: permite la conexión de los dispositivos a la red celular [22], asegurando 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.
- 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.

**2.2. Componentes de hardware utilizados**

7

FIGURA 2.1. Contador de tránsito DTEC <sup>1</sup>.

- 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 <sup>2</sup>.

- 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.

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

<sup>2</sup>Imagen tomada de <https://www.alldatasheet.com/datasheet-pdf/pdf/73074/MAXIM/MAX232.html>

**2.3. Tecnologías de software aplicadas**

7

- 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.

FIGURA 2.1. Contador de tránsito DTEC <sup>1</sup>.

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 tecnologías de software abiertas y estandarizadas, seleccionadas por su solidez, disponibilidad y capacidad de integración.

- 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.

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

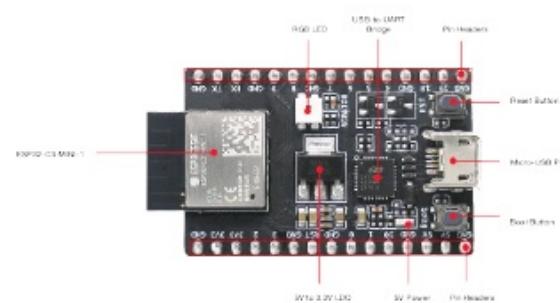


FIGURA 2.3. Microcontrolador ESP32-C3 utilizado en los nodos de campo<sup>3</sup>.

- 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<sup>4</sup>.

- 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.

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

<sup>4</sup>Imagen tomada de <https://www.alldatasheet.com/datasheet-pdf/pdf/1741389/SIMCOM/SIM800L.html>

- Base de datos relacional MySQL. La persistencia de los datos se implementa en MySQL [26], un sistema gestor de bases de datos relativamente 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 librería 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, complementando la 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, asegurando que todas las dependencias se inicien en el orden correcto y facilitando 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

### 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

9

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.

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.

## Capítulo 3

### Diseño e implementación

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

#### 3.1. Arquitectura del sistema

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

##### 3.1.1. Descripción ampliada de bloques y responsabilidades

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

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

## Capítulo 3

### Diseño e implementación

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

#### 3.1. Arquitectura del sistema

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

##### 3.1.1. 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`.

- devices/{id}/events
  - devices/{id}/comm
  - devices/{id}/status
- 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.

- 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 3.1 muestra el diagrama de secuencia del flujo de datos.

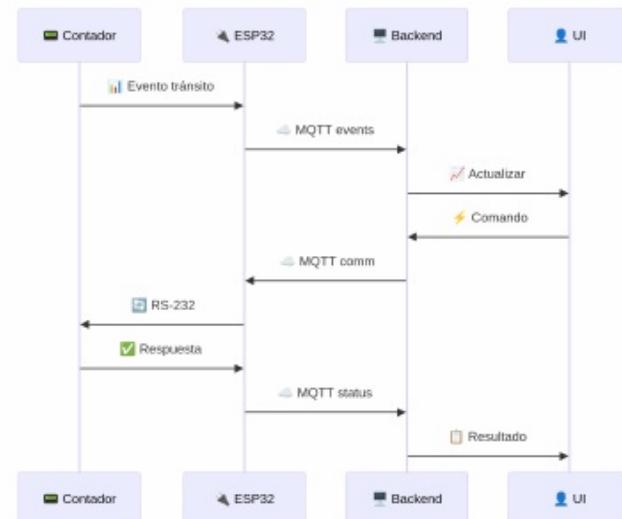


FIGURA 3.1. Diagrama de secuencia del flujo de datos.

### 3.1.2. Descripción ampliada de bloques y responsabilidades

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

## 3.1. Arquitectura del sistema

13

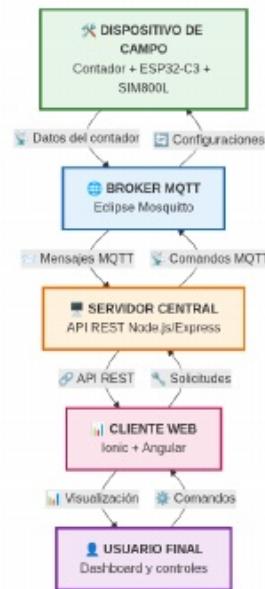


FIGURA 3.1. Diagrama de arquitectura del sistema y el flujo de datos.

## 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.

## 3.1. Arquitectura del sistema

13

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

- 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.

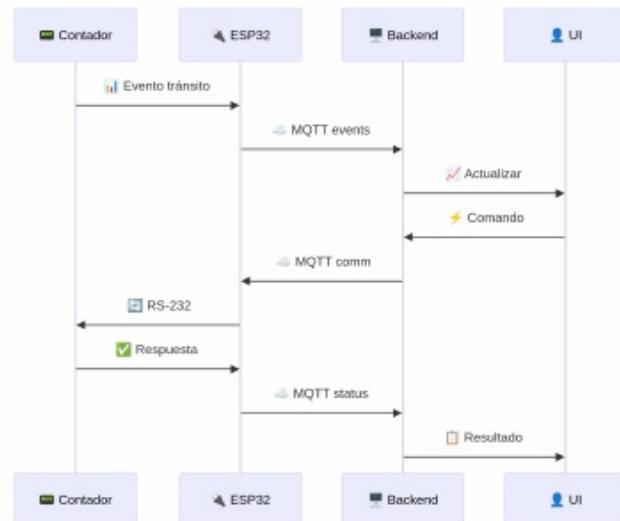


FIGURA 3.2. Diagrama de secuencia del flujo de datos.

### 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 nacionales. Cada nodo integra varios módulos que permiten la adquisición, procesamiento y transmisión de los datos de tránsito:

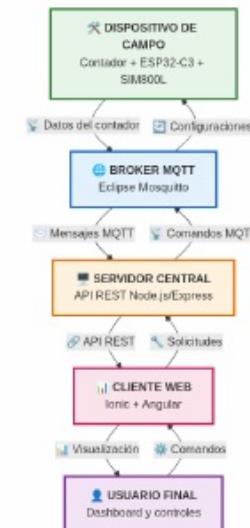


FIGURA 3.2. Diagrama de arquitectura del sistema y el flujo de datos.

### 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.

### 3.2. Detalle de módulos de hardware

15

#### 3.2.1. ESP32-C3 (unidad de control)

El ESP32-C3 constituye el núcleo de procesamiento del nodo de campo. Se trata de un microcontrolador de bajo consumo con conectividad, elegido principalmente por su capacidad de cómputo, su soporte de entornos de desarrollo abiertos y la disponibilidad de bibliotecas optimizadas para protocolos de comunicación.

- Rol en la arquitectura: coordina la recepción de eventos desde el contador a través de RS-232, gestiona el encolamiento FIFO, controla la comunicación con el módem SIM800L mediante comandos AT y actúa como cliente MQTT.
- Entorno de desarrollo: el firmware se desarrolló sobre ESP-IDF, el framework oficial de Espressif, que permite gestionar tareas concurrentes mediante FreeRTOS y facilita la integración de bibliotecas de red y drivers UART.
- Ventajas técnicas: bajo costo, consumo reducido, capacidad de ejecutar varias tareas en paralelo y soporte nativo para criptografía y seguridad en comunicaciones.
- Consideraciones de diseño: se provee una correcta disipación térmica y estabilidad de la alimentación, especialmente durante la transmisión de datos.

#### 3.2.2. Módulo GPRS SIM800L

El módulo SIM800L implementa la conectividad celular GPRS, que permite la transmisión bidireccional de datos entre dispositivos remotos y servidor centralizado.

- Funciones principales: establecimiento de sesiones TCP/IP sobre GPRS, controlado por el ESP32-C3 mediante comandos AT. Soporta publicación y suscripción MQTT a través de sockets TCP persistentes.
- Integración con ESP32-C3: la comunicación entre ambos se realiza mediante UART secundaria. El firmware implementa comandos de inicialización, registro en red, apertura de contexto y gestión de reconexiones.
- Aspectos críticos: el SIM800L presenta picos de consumo que pueden superar los 2 A durante la transmisión, se dispone de una fuente con suficiente margen y filtros adecuados para evitar reinicios inesperados.
- Limitaciones: ancho de banda reducido (máximo teórico de 85,6 kbps en GPRS), lo que refuerza la decisión de emplear MQTT por su bajo overhead.

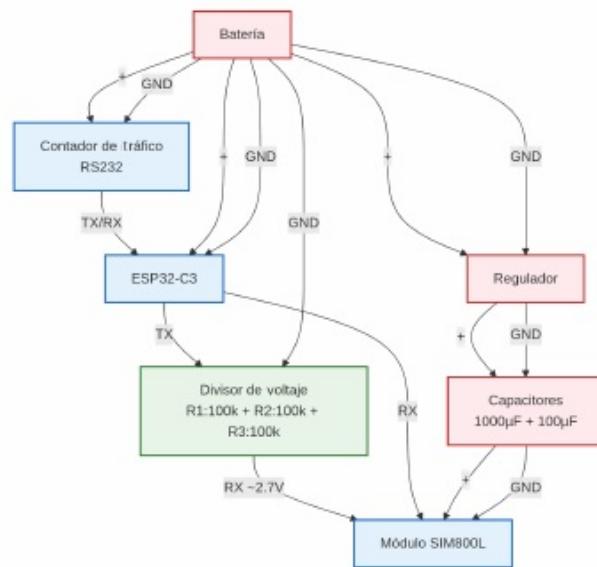


FIGURA 3.3. Diagrama de conexión entre los módulos del sistema.

- 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.

### 3.2.3. Contador de tránsito y comunicación RS-232

El sistema de conteo existente genera tramas con información de los eventos de paso de vehículos acumulados en un intervalo de tiempo (por defecto 1 hora). La comunicación con el ESP32-C3 se establece mediante interfaz RS-232, estándar ampliamente utilizado para transmisión serial de datos.

- Formato de trama: el equipo incluye identificador de Dispositivo, timestamp local, valores de clasificación tránsito por carril.
- Adaptación eléctrica: se utiliza un conversor de niveles (MAX232) para adaptar las señales RS-232 al rango TTL del microcontrolador.
- Ventaja: la reutilización de la interfaz serial del contador evita modificar el sistema de detección existente, reduciendo costos de integración.

En la figura 3.3 se observa el contador de tránsito. Este dispositivo constituye la base del sistema de detección de eventos y se mantiene sin modificaciones en su lógica interna.

FIGURA 3.3. Contador de tránsito DTEC<sup>1</sup>.

<sup>1</sup>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.<sup>1</sup>

### 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).

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

### 3.2. Detalle de módulos de hardware

17

#### 3.2.4. 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 3.4 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.

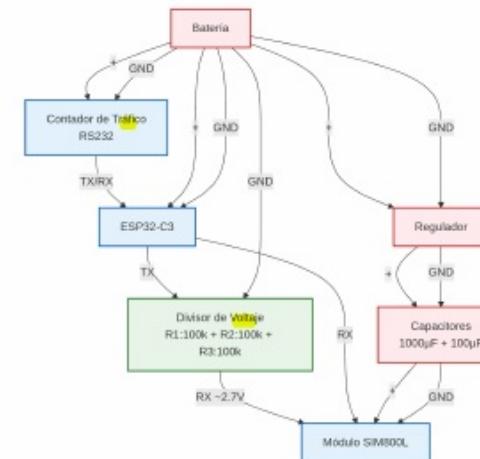


FIGURA 3.4. Diagrama de conexión entre los módulos del sistema.

### 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.

- **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 3.5 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 3.5. Fotografía contador de tránsito DTEC instalado en campo<sup>2</sup>.

<sup>2</sup>Imagen tomada de <http://transito.vialidad.gob.ar/>

### 3.3. Desarrollo del backend

19

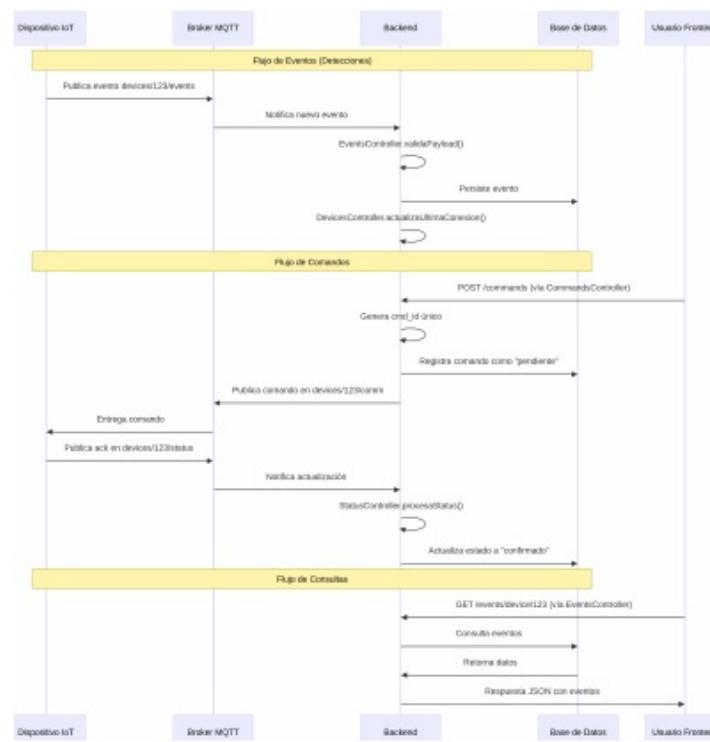


FIGURA 3.5. Diagrama de flujo de información del Backend.

#### 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.

### 3.3. Desarrollo del backend

19

#### 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.

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).

En la figura 3.6 se observa el diagrama de flujo de Información del Backend.

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.

### 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 ingestión 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.

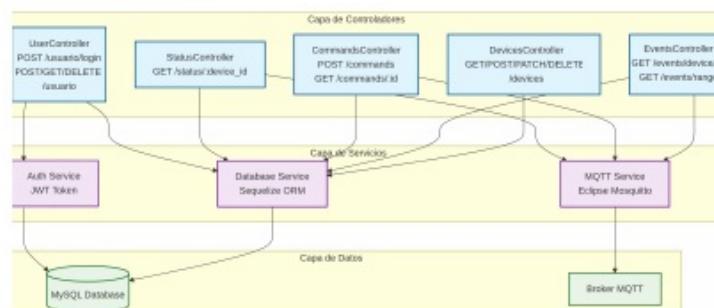


FIGURA 3.6. Diagrama con la disposición de los controladores y flujo de dependencias.

### 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:

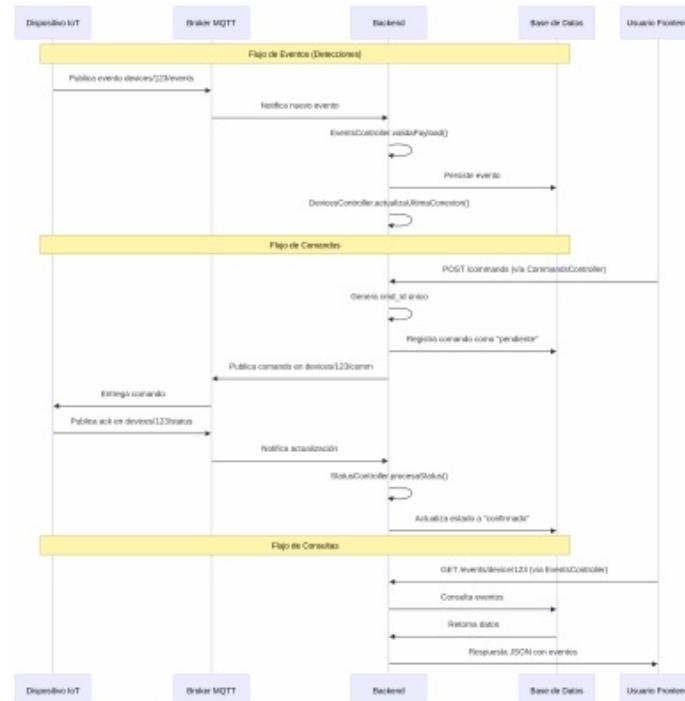


FIGURA 3.6. Diagrama de flujo de Información del Backend

**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 /devices	DevicesController	Lista todos los dispositivos registrados
GET /devices/{id}	DevicesController	Devuelve información de un dispositivo específico
POST /devices	DevicesController	Alta de un nuevo dispositivo
PATCH /devices/{id}	DevicesController	Actualización de atributos de un dispositivo
DELETE /devices/{id}	DevicesController	Eliminación de un dispositivo
GET /events/device/{id}	EventsController	Consultar eventos por dispositivo
GET /events/range	EventsController	Consultar eventos por rango temporal
POST /commands	CommandsController	Crear un comando remoto y publicarlo en MQTT
GET /commands/{id}	CommandsController	Consultar estado de un comando (pendiente, ok, failed, timeout, value)
GET /status/{id}	StatusController	Consultar estado operativo y telemetría del dispositivo
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

**3.3. Desarrollo del backend**

21

**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. Esto asegura separación de responsabilidades, facilita el mantenimiento y permite la escalabilidad de la aplicación.

- 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 ingestión 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 3.8 se observa el diagrama con la disposición de los controladores y flujo de dependencias.

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.

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.
- Histórial 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.

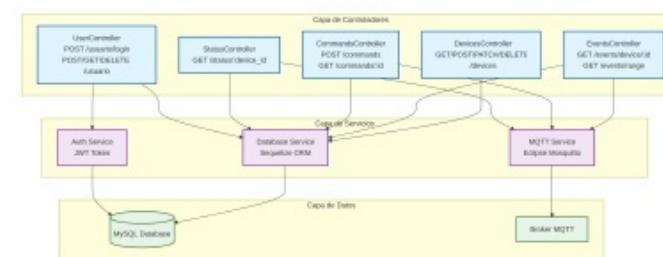


FIGURA 3.7. Diagrama con la disposición de los controladores y flujo de dependencias.

### 3.4. Desarrollo del frontend

23

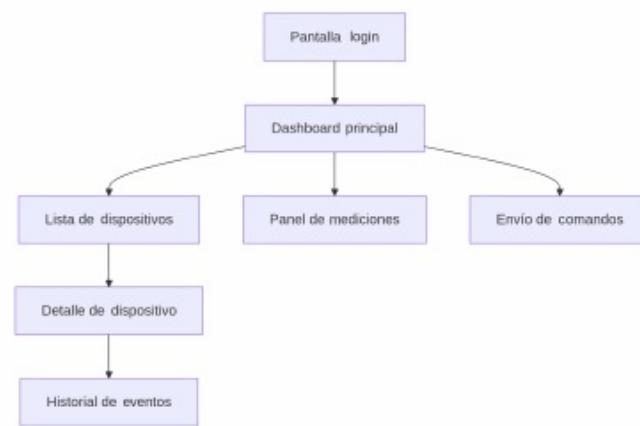


FIGURA 3.7. Diagrama de estructura de los componentes por pantalla.

#### 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.

### 3.4. Desarrollo del frontend

23

#### 3.3.4. Mapa de endpoints

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

Endpoint	Controlador	Descripción
GET /devices	DevicesController	Lista todos los dispositivos registrados
GET /devices/{id}	DevicesController	Devuelve información de un dispositivo específico
POST /devices	DevicesController	Alta de un nuevo dispositivo
PATCH /devices/{id}	DevicesController	Actualización de atributos de un dispositivo
DELETE /devices/{id}	DevicesController	Eliminación de un dispositivo
GET /events/device/{id}	EventsController	Consultar eventos por dispositivo
GET /events/range	EventsController	Consultar eventos por rango temporal
POST /commands	CommandsController	Crear un comando remoto y publicarlo en MQTT
GET /commands/{id}	CommandsController	Consultar estado de un comando (pendiente, ok, failed, timeout, value)
GET /status/{id}	StatusController	Consultar estado operativo y telemetría del dispositivo
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

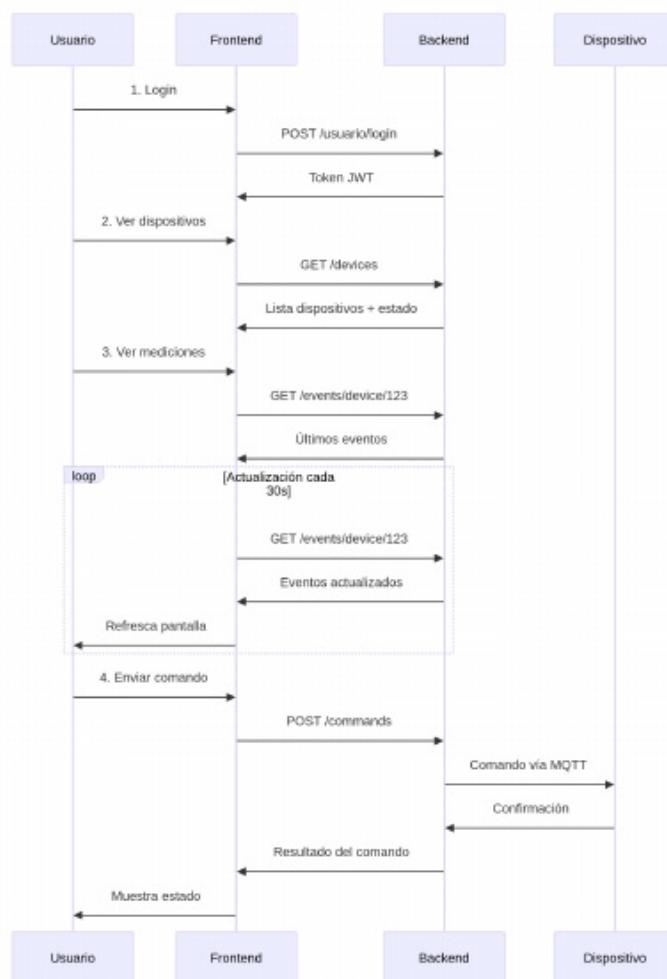


FIGURA 3.8. Diagrama de flujo de comunicación con el backend.

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.

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 3.8 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 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).

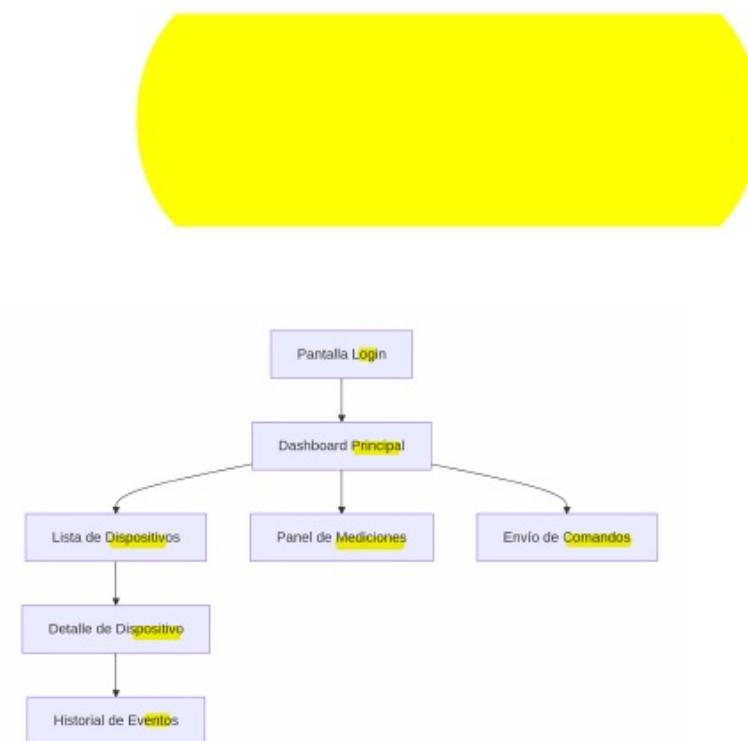


FIGURA 3.8. Diagrama de estructura de los componentes por pantalla.

- 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.

#### 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 3.9 se observa el diagrama de flujo de Comunicación con el backend.

## Bibliografía

- [1] Juan Asiain et al. «LoRa-Based Traffic Flow Detection for Smart-Road». En: *Sensors* 21.2 (2021), pág. 338. DOI: 10.3390/s21020338. URL: <https://www.mdpi.com/1424-8220/21/2/338>.
- [2] Jan Micko et al. «Review of IoT Sensor Systems Used for Monitoring the Road Infrastructure». En: *Sensors* 23.9 (2023), pág. 4469. DOI: 10.3390/s23094469. URL: <https://www.mdpi.com/1424-8220/23/9/4469>.
- [3] Gianluca Peruzzi et al. «Combining LoRaWAN and NB-IoT for Edge-to-Cloud Low-Power Connectivity». En: *Applied Sciences* 12.3 (2022), pág. 1497. DOI: 10.3390/app12031497. URL: <https://www.mdpi.com/2076-3417/12/3/1497>.
- [4] Miovision. *TrafficLink / Managed Connectivity*. <https://miovision.com/trafficlink/>. Soluciones comerciales. 2023.
- [5] Sensys Networks. Documentación técnica y productos. <https://sensysnetworks.com/>. 2023.
- [6] MetroCount. Contadores y guías técnicas. <https://www.metrocount.com/>. 2023.
- [7] Exemys Managed Connectivity. Accedido: 16-Sep-2025. 2025. URL: <https://www.exemys.com/site/index.shtml>.
- [8] Digi International. *Digi Remote Manager: IoT Device Monitoring and Management Solution*. <https://www.digi.com/products/iot-software-services/digi-remote-manager>. Accedido: 11 de septiembre de 2025. 2025.
- [9] A. A. Sukmandhani, M. Zarlis y Nurudin. «Monitoring Applications for Vehicle based on Internet of Things (IoT) using the MQTT Protocol». En: *BINUS Conference Proceedings*. Accedido: 11 de septiembre de 2025. 2023. URL: <https://research.binus.ac.id/publication/C1B25545-66P9-49C3-B873-D6C537EA23B3/monitoring-applications-for-vehicle-based-on-internet-of-things-iot-using-the-mqtt-protocol/>.
- [10] S. Bharath y C. Khusi. «IoT Based Smart Traffic System Using MQTT Protocol: Node-Red Framework». En: *2nd Global Conference for Advancement in Technology (GCAT)*. Accedido: 11 de septiembre de 2025. 2021. DOI: 10.1109/GCAT52182.2021.9587636.
- [11] Zuoling Niu. «Research and Implementation of Internet of Things Communication System Based on MQTT Protocol». En: *Journal of Physics: Conference Series* 012019 (2023). Accedido: 11 de septiembre de 2025.
- [12] OpenRemote. *OpenRemote: 100 % Open Source IoT Device Management Platform*. <https://openremote.io/>. Accedido: 11 de septiembre de 2025. 2025.
- [13] Espressif Systems. *ESP32-C3 Series Datasheet*. Accedido: 23-Sep-2025. Espressif Systems. 2021. URL: [https://www.espressif.com/sites/default/files/documentation/esp32-c3\\_datasheet\\_en.pdf](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.

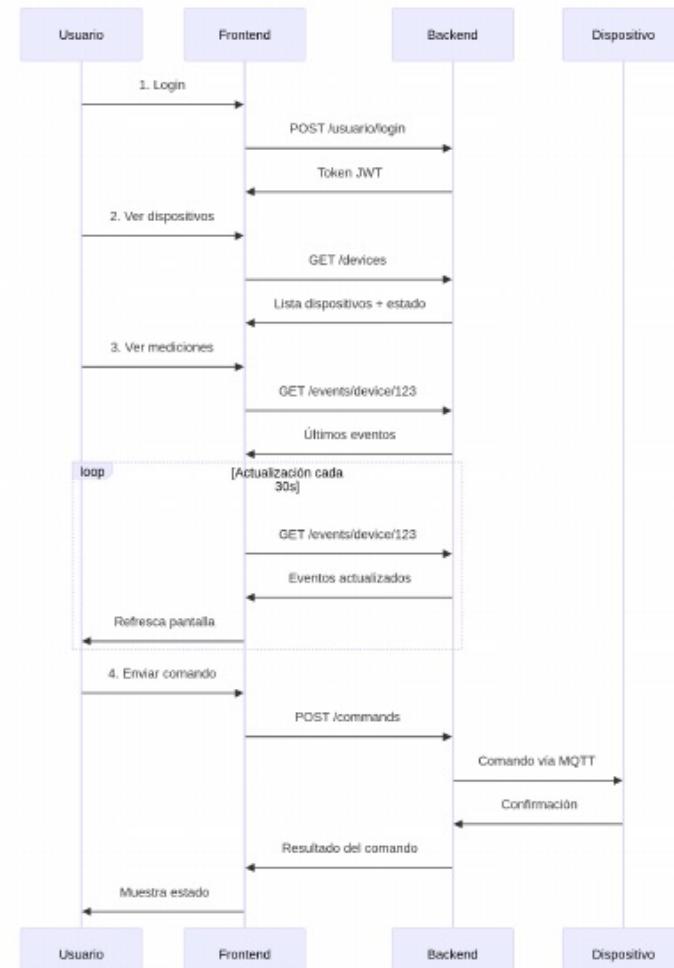


FIGURA 3.9. Diagrama de flujo de Comunicación con el backend.

- [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](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.

### 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. Consumir 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).

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

- 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. Nodos y Sensores

En este capítulo se detalla el hardware de los nodos de campo que constituyen la base del sistema de detección de tránsito. Cada nodo combina un microcontrolador ESP32-C3 con interfaces de comunicación estándar, lo que permite procesar eventos localmente y establecer un vínculo bidireccional con el servidor central. De esta manera, se logra una solución flexible, escalable y compatible con la infraestructura vial existente.

#### 3.6.1. Arquitectura del nodo

La arquitectura de cada nodo se diseñó con el objetivo de reutilizar los contadores de tránsito actualmente desplegados en rutas nacionales, minimizando modificaciones en el equipamiento existente. Para ello, se emplea una interfaz RS-232 que conecta directamente al contador con el microcontrolador. Dado que el ESP32-C3 opera con niveles de señal TTL, se incorporó un conversor MAX232, encargado de realizar la adaptación de niveles eléctricos y asegurar la robustez de la comunicación.

- Contador de tránsito modelo DTEC: dispositivo comercial encargado de detectar el paso de vehículos y generar tramas de datos con la información del evento.  
En la Figura 3.10 se observa el contador de tránsito. Este dispositivo constituye la base del sistema de detección.
- 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). En la Figura 3.11 se observa el Módulo de adaptación RS-232/TTL (MAX232).
- ESP32-C3: microcontrolador que ejecuta el firmware desarrollado sobre ESP-IDF. Sus funciones incluyen el preprocesamiento de eventos, el encolamiento FIPO, la suscripción a comandos remotos y la gestión integral de la comunicación con el servidor. En la Figura 3.12 se muestra el Microcontrolador utilizado en campo.

<sup>2</sup>Imagen tomada de <http://transito.vialidad.gob.ar/>

<sup>3</sup>Imagen tomada de <https://www.alldatasheet.com/datasheet-pdf/pdf/73074/MAXIM/MAX232.html>

<sup>4</sup>Imagen tomada de <https://docs.espressif.com/projects/esp-idf/en/v5.0/esp32c3-hw-reference/esp32c3/user-guide-devkitm-1.html>