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

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

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

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

12 Capítulo 3. Diseño e implementación

- `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.

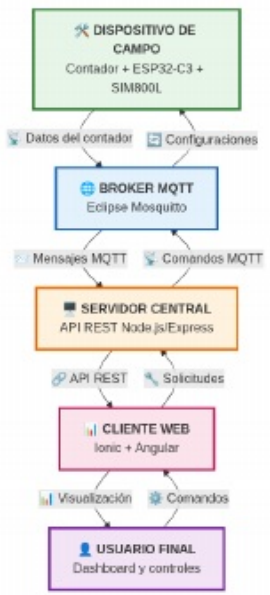

FIGURA 3.1. Diagrama de arquitectura del sistema y el flujo de datos.

3.1.2. Flujo de datos

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

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

3.1. Arquitectura del sistema 13

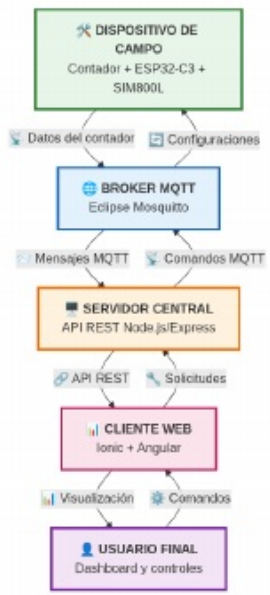

FIGURA 3.1. Diagrama de arquitectura del sistema y el flujo de datos.

3.1.2. Flujo de datos

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

- Detección: el contador detecta un paso, acumula y en determinado intervalo envía una trama por RS-232 al ESP32-C3.
- Preprocesado en nodo: el firmware valida la trama, añade sello temporal y metadatos, y encola el evento en memoria (FIFO).
- Transmisión: cuando la conexión GPRS está disponible, el nodo publica 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 /app/comando` al backend, que crea un `comm_id` único con dispostivold y publica en `dispositivo/id/comando`.
- Recepción nodo/Entrega al contador: el ESP32-C3 en `dispositivo/id/comando` recibe el comando, valida `cmd_id` y lo envía al contador por RS-232, se aplica un timeout configurable por comando.
- Ejecución y ack: el contador ejecuta la orden y responde por RS-232, el firmware publica el ack/resultado en `dispositivo/id/respuesta` con `cmd_id` y `status` (`ok`, `failed`, `timeout`, `value`).
- Actualización en backend y UI: el suscriptor MQTT del backend recibe el ack, actualiza la tabla `respuesta` (campo `valor`, `ack_ts`) y notifica a la UI para que el operador vea el resultado.

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

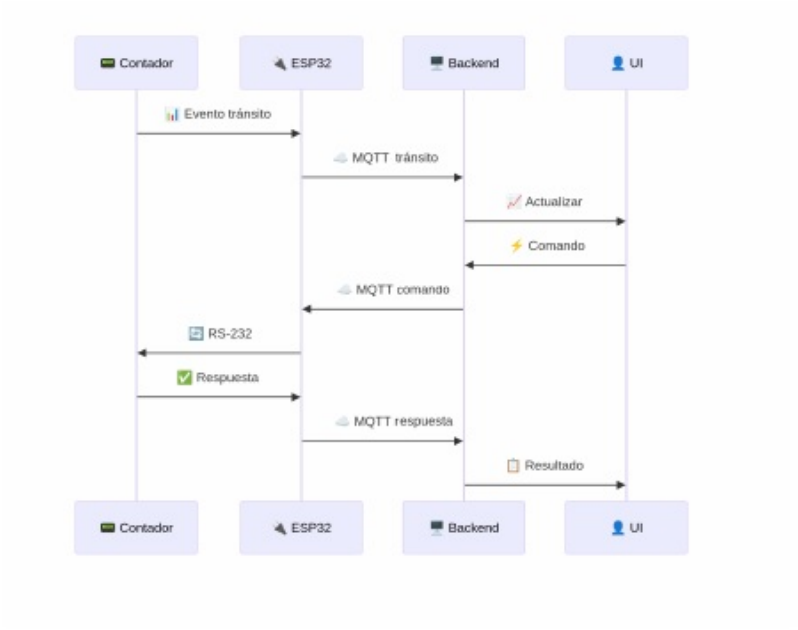

FIGURA 3.2. Diagrama de secuencia del flujo de datos.

3.2. Arquitectura del nodo

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

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

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

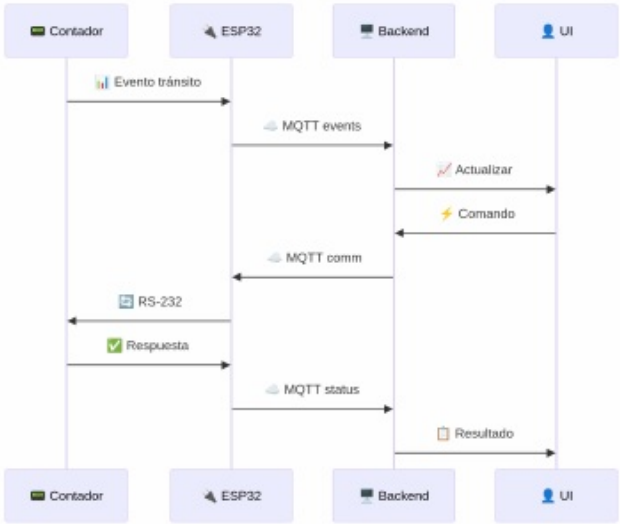

FIGURA 3.2. Diagrama de secuencia del flujo de datos.

3.2. Arquitectura del nodo

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

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

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

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

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

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

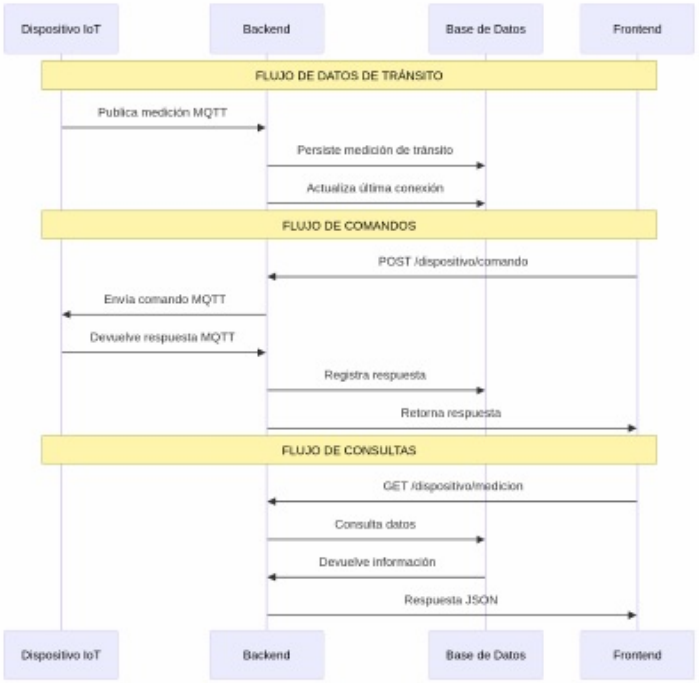

FIGURA 3.5. Diagrama de flujo de información del Backend.

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

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

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

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

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

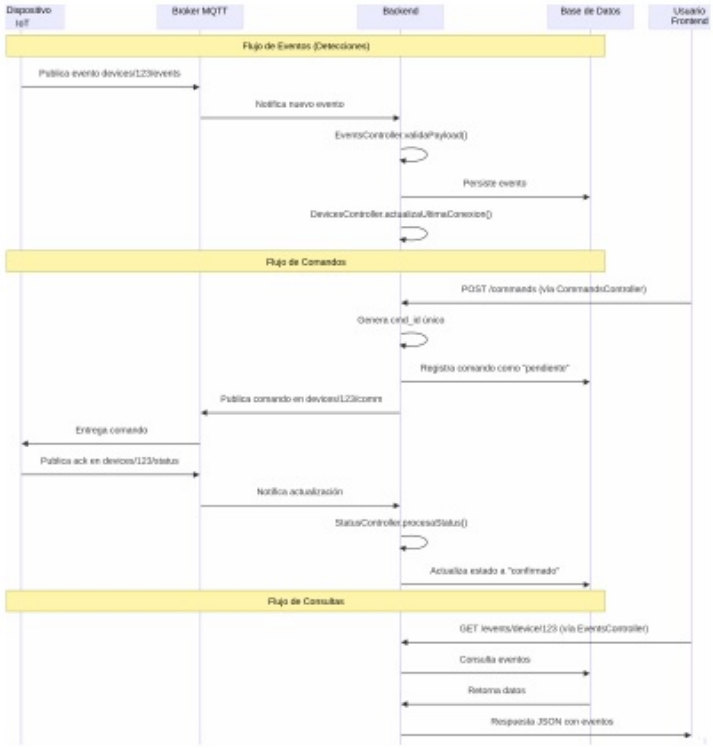

FIGURA 3.5. Diagrama de flujo de información del Backend.

3.3.2. Funcionalidades principales

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

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

3.3.3. Organización en controladores

La lógica de negocio del backend se organiza en controladores, cada uno asociado a un recurso del sistema, lo que asegura una clara separación de responsabilidades, facilita el mantenimiento y permite la escalabilidad de la aplicación. Entre los controladores principales se encuentran los siguientes:

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

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

3.3. Desarrollo del backend 19

3.3.2. Funcionalidades principales

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

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

3.3.3. Organización en controladores

La lógica de negocio del backend se organiza en controladores, cada uno asociado a un recurso del sistema, lo que asegura una clara separación de responsabilidades, facilita el mantenimiento y permite la escalabilidad de la aplicación. Entre los controladores principales se encuentran los siguientes:

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

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

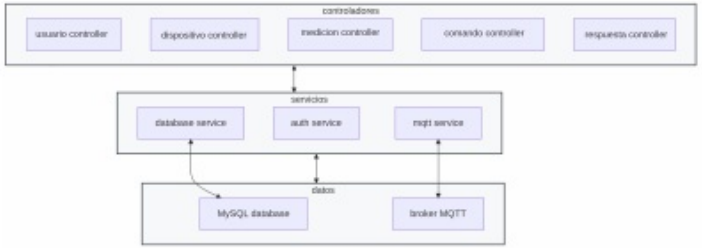

FIGURA 3.6. Diagrama con la disposición de los controladores y flujo de dependencias.

3.3.4. Mapa de endpoints

El backend expone una serie de endpoints REST que conforman la interfaz principal de comunicación con los servicios de aplicación y los dispositivos de campo. A continuación, se presenta el mapa general de los endpoints y controladores mas importantes para el flujo:

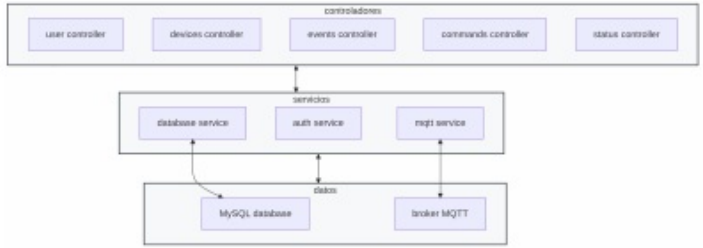

FIGURA 3.6. Diagrama con la disposición de los controladores y flujo de dependencias.

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:

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

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

3.3.5. Seguridad y extensibilidad

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

3.4. Desarrollo del frontend

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

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

3.4.1. Arquitectura y tecnologías

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

Las tecnologías principales aportan ventajas concretas:

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

3.4.2. Funcionalidades principales

El frontend integra las siguientes funciones clave:

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

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

22 Capítulo 3. Diseño e implementación

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 responsivo tanto en entornos de escritorio como móviles. La comunicación con el backend se realiza mediante peticiones HTTP a la API REST, y en casos donde se requiere actualización en tiempo real se emplea un canal de notificación basado en WebSocket.

Las tecnologías principales aportan ventajas concretas:

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

3.4.2. Funcionalidades principales

El frontend integra las siguientes funciones clave:

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

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