CARRERA DE ESPECIALIZACIÓN EN INTERNET DE LAS COSAS

MEMORIA DEL TRABAJO FINAL

Diseño e implementación de una central operativa para el control y monitoreo en el material rodante

Autor:
Fernando Julio Iglesias

Director:
Ing. Fernando Lichtschein (FI-UBA)

Jurados:
Nombre del jurado 1 (pertenencia)
Nombre del jurado 2 (pertenencia)
Nombre del jurado 3 (pertenencia)

Este trabajo fue realizado en la Ciudad Autónoma de Buenos Aires, entre marzo de 2022 y agosto de 2023.

Resumen

En la presente memoria se describe el desarrollo, para Trenes Argentinos, de una arquitectura modular basada en el paradigma de Internet de las Cosas (IoT) que permite visualizar y gestionar, por un supervisor u operario, las formaciones ferroviarias de forma remota desde una central operativa. De esta manera, se logra mejorar la seguridad, la eficiencia y aumentar la flexibilidad del sistema ferroviario.

El sistema se trata de una solución fullstack embebida en donde se expone cada uno de los componentes de hardware y software involucrados. Entre los conocimientos aplicados se destacan el diseño de una arquitectura cliente-servidor, el diseño de una API GraphQL, la interacción con un broker de mensajes y la seguridad en las comuicaciones.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción general

En este capítulo se presentará una breve introducción sobre la inserción de la Internet de las cosas (*IoT*) [1] en el ámbito ferroviario, abarcando las motivaciones que llevaron a la realización de este trabajo y el estado del arte en cuanto a esta temática. Además, se establecerán los alcances y objetivos del trabajo, los cuales se enfocan en diseñar e implementar un sistema IoT que permita monitorear y controlar los componentes de una formación ferroviaria, con el fin de mejorar la eficiencia y seguridad en el transporte ferroviario.

1.1. La Internet de las Cosas y las formaciones ferroviarias

La Internet de las Cosas (*IoT*, por sus siglas en inglés) es una tecnología que se ha convertido en una tendencia en la actualidad. Este concepto se estableció en el mercado en la década de 1990, aunque su verdadero auge comenzó en los últimos años gracias a la aparición de dispositivos interconectados y al aumento en el uso de internet. La IoT se enfoca en la interconexión de objetos cotidianos con la red, lo que permite recopilar datos en tiempo real, optimizar procesos y tomar decisiones basadas en la información recolectada. Algunos beneficios que la IoT ofrece son una mayor eficiencia, una reducción de costos, un mejor monitoreo y control, y una mejor calidad de vida para las personas.

En el sector ferroviario, la IoT ha demostrado ser una tecnología clave para mejorar la eficiencia y la seguridad de las formaciones ferroviarias. Las formaciones ferroviarias pueden ser equipadas con sensores y dispositivos interconectados que permiten recopilar información sobre la ubicación, velocidad, consumo de energía, estado mecánico y otros aspectos importantes. Esta información se puede enviar en tiempo real a la red, lo que permite una mejor gestión del tráfico ferroviario y una toma de decisiones más rápida y eficaz. Además, la IoT también puede ayudar a prevenir accidentes ferroviarios al detectar problemas mecánicos antes de que se conviertan en un riesgo para la seguridad.

[1] https://www.investopedia.com/terms/i/internet-things.asp

1.2. Motivación

El sistema ferroviario de la República Argentina cuenta con una gran cantidad de formaciones ferroviarias en las que se encuentran diferentes sistemas de seguridad a bordo. Estos equipos se encargan de supervisar el correcto funcionamiento de los subsistemas críticos. Ante una falla en uno de los subsistemas, una formación ferroviaria se detiene inmediatamente por la activación automática de las señales de corte de tracción (*CT*) y freno de emergencia (*FE*). En esta situación, el conductor debe llevar la formación a un lugar seguro para que los pasajeros puedan descender y, posteriormente, trasladarla a un taller para que pueda ser reparada.

En efecto, por parte de *CONICET-GICSAFe* [1] para la empresa Trenes Argentinos [2], se propone desarrollar una central operativa tal que se puede administrar y configurar de forma remota los dispositivos de supervisión de seguridad de cada formación ferroviaria, la visualización de los diferentes parámetros de interés involucrados por las personas asignadas dentro de una entidad y de este modo sea posible optimizar la toma de decisiones.

1.3. Estado del arte

A partir del análisis en las últimas tendencias referidas a la Central Operativa *SAL/T*, se listan aquellas herramientas, de gestión y control de dispositivos de seguridad en el sector ferroviario, que presentan características similares al producto propuesto,

1. *Indra*[2]: ofrece una solución integral de seguridad para el sector ferroviario, que incluye una plataforma de gestión centralizada y un sistema de supervisión remota de dispositivos de seguridad.
2. *Thales Group*[3]: desarrolla soluciones de seguridad para el sector ferroviario, que incluyen una aplicación de gestión y control centralizada para supervisar y configurar dispositivos de seguridad de forma remota.
3. *Alstom*[4]: ofrece una plataforma de gestión y control para el sector ferroviario, que permite la supervisión y configuración de dispositivos de seguridad de forma remota, así como el análisis en tiempo real de los datos recopilados por estos dispositivos.
4. Siemens Mobility[5]: ofrece una solución de gestión y control para el sector ferroviario, que incluye una aplicación web de central operativa para supervisar y configurar dispositivos de seguridad de forma remota, así como un sistema de análisis de datos en tiempo real.

[2] urlhttps://www.indracompany.com

[3] urlhttps://www.thalesgroup.com

[4] urlhttps://www.alstom.com/

[5] urlhttps://www.mobility.siemens.com/

1.4. Alcance y objetivos

El sistema resultante de este trabajo se encuentra destinado al desarrollo del software para una central operativa que permite administrar y configurar de forma remota dispositivos de supervisión de seguridad de formaciones ferroviarias denominados *SAL/T* (Sistema de Aislamiento Limitado/Total).

El alcance del proyecto comprende los siguientes puntos,

- Servicio de monitoreo y control: visualizar en tiempo real los datos recibidos y enviar comandos de control a los dispositivos activos.
- Gestión de configuración de dispositivos: visualizar y modificar los parámetros configurables de los dispositivos.
- Base de datos: registrar la información recibida desde todos los dispositivos.
- Componentes de seguridad: brindar seguridad a las comunicaciones y a la información almacenada.
- Gestión de usuarios y perfiles: administrar roles y permisos de acceso.
- Realizar la migración de un microcontrolador de la familia Cortex tipo M [6] a uno de la misa familia que disponga de características destinadas a la seguridad funcional del sistema.

[6] https://developer.arm.com/Processors/Cortex-M4

Capítulo 2

Introducción específica

2.1. Protocolos de comununicación utilizados

Comenzar escribiendo que aqui se detallaran los protocolos utilizados segun el modelo TCP/IP y hablar sobre el diagrama

AGREGAR UNA IMAGEN CON LOS 3 ENTES INTERCONECTADOS BAJO EL MODELO OSI Nucleo F429ZI Board <—->Dedicated Server <—->Web Client

2.1.1. Protocolos de comunicación

De los modelos *stack IoT* y *lwIP* [3], basados en el stack *TCP/IP* [1], se emplean los siguientes protocolos de comunicación tal como se puede observar en la tabla 2.1.

TABLA 2.1. Protocolos de comunicación utilizados en cada capa del *stack TCP/IP*

Capa	Protocolo
Capa de Aplicación	HTTP [4], DHCP [5] y MQTT [6]
Capa de Transporte	UDP [7]
Capa de Red	IPv4 [8], ARP [9]
Capa de Enlace de Datos	Ethernet [10]

Cabe destacar que ambos modelos se encuentran diseñados para sistemas con recursos limitados y buscan garantizar una correcta comunicación entre los dispositivos conectados en la red.

2.2. Tecnologías *full-stack*

A partir de los modelos más relevantes que se encuentran establecidos para el desarrollo e implementación de aplicaciones de software se consideran el *stack web* y el *stack IoT*, tal como se puede ver en la figura 3.1. A pesar de las similitudes entre ambos, en la aplicación el *stack web* emplea el protocolo *HTTP* mientras que el *stack IoT* utiliza el protocolo *MQTT*; siendo este último el más adecuado para escenarios donde son limitados los recursos como el ancho de banda y el consumo de energía.

En especial, el protocolo *MQTT* dispone de mensajes más livianos y además es posible transmitir y/o recibir datos en formato binario sin la necesidad de una

[1] https://www.ibm.com/docs/en/aix/7.2?topic=protocol-tcpip-protocols

codificación previa. También, este protocolo permite asignar niveles de calidad de servicio (*QoS* [2]) a los mensajes transmitidos, resultando una característica primordial en aplicaciones donde la probabilidad de pérdidas de paquetes es considerable.

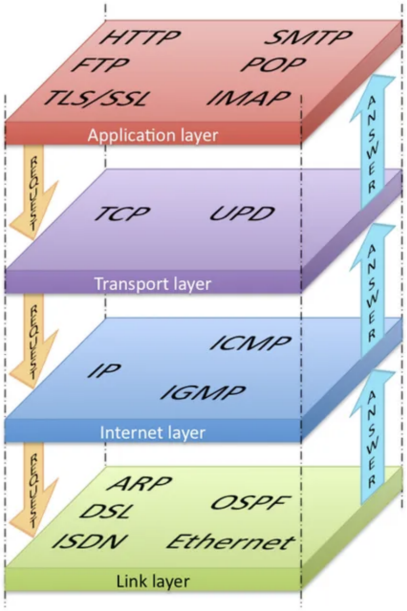

FIGURA 2.1. Arquitectura de la Central Operativa SAL/T.

2.2.1. Tecnologías del *front-end*

TypeScript

TypeScript [11] es una extensión de código abierto de JavaScript que agrega tipos estáticos opcionales y características de programación orientada a objetos avanzadas, lo que permite una mayor seguridad y mantenibilidad en el código.

React

React [12] es una biblioteca de JavaScript para construir UI reutilizables e interactivas, desarrollada por Facebook y ampliamente utilizada en el desarrollo web. Utiliza un enfoque basado en componentes y el DOM virtual para mejorar el rendimiento y es altamente integrable con otras bibliotecas y frameworks.

Parcel

Parcel [13] es una herramienta de construcción de código abierto con una estrategia de çero configuración"que optimiza y empaqueta módulos de JavaScript y otros archivos. En particular, mejora significativamente el flujo de trabajo del desarrollador al ser rápido, fácil de usar y altamente personalizable.

[2] https://www.ibm.com/docs/en/i/7.2?topic=services-quality-service

Apollo GraphQL

Apollo GraphQL [14] es una plataforma de código abierto para el desarrollo de aplicaciones GraphQL, con una amplia gama de herramientas y servicios para construir y mantener aplicaciones GraphQL de manera efectiva, ofreciendo una administración de caché y una amplia compatibilidad con diferentes frameworks y tecnologías.

Material UI

Material UI [15] es una biblioteca de componentes de interfaz de usuario de código abierto basada en Material Design de Google, que ofrece componentes preconstruidos altamente personalizados y una amplia compatibilidad con diferentes tecnologías y frameworks para construir aplicaciones web modernas y atractivas.

JWT Decode

JWT Decode [16] es una biblioteca JavaScript que decodifica tokens JWT, con características útiles como validación de tokens y verificación de firma. Se integra fácilmente en diferentes frameworks de JavaScript como React y Angular.

2.2.2. Tecnologías del *backend*

Kotlin

Kotlin [17] es un lenguaje de programación de tipado estático que puede correr sobre la máquina virtual de Java y ser compilado en JavaScript y LLVM. También ofrece interoperabilidad con Java, programación funcional, orientación a objetos y manejo de nulos seguro, lo que ha llevado a su popularidad debido a su facilidad de uso y legibilidad.

Ktor

Ktor [18] es un framework web de Kotlin que permite crear aplicaciones web y API REST de manera fácil y eficiente, con un enfoque en la programación funcional. Ktor es altamente modular y personalizable, lo que significa que los desarrolladores pueden seleccionar solo los componentes que necesitan para su aplicación, reduciendo así la complejidad y el tamaño del código.

Kafka

Kafka [19] es una plataforma de streaming de datos que permite a las aplicaciones enviar y recibir datos en tiempo real. Basada en el modelo de publicación/-suscripción, se caracteriza por su alta velocidad y rendimiento, haciéndola ideal para aplicaciones que necesitan procesar grandes cantidades de datos en tiempo real.

Hasura

Hasura [20] es una plataforma sin servidor que permite a los desarrolladores crear y escalar aplicaciones con GraphQL rápidamente. Ofrece seguridad de extremo

a extremo y se integra fácilmente con diferentes bases de datos, generando automáticamente una API de GraphQL.

PostgreSQL

PostgreSQL [21] es un sistema de gestión de bases de datos de código abierto y gratuito, conocido por su fiabilidad, escalabilidad y seguridad. Ofrece soporte para transacciones ACID, integración con lenguajes de programación populares y una amplia variedad de herramientas y extensiones.

Mosquitto Broker

Mosquitto [22] es un broker MQTT de código abierto utilizado en IoT para transmitir mensajes entre dispositivos. Es popular por su escalabilidad, facilidad de uso y características de seguridad.

2.2.3. Tecnologías del *firmware*

C lang

C [23] es un lenguaje de programación estructurado y de propósito general, ampliamente utilizado en sistemas operativos, aplicaciones de bajo nivel y dispositivos embebidos. Su sintaxis simple y directa permite escribir código claro y legible, y ofrece gran control sobre la memoria y acceso directo al hardware.

FreeRTOS

FreeRTOS [24] es un RTOS de código abierto y gratuito que controla sistemas embebidos y microcontroladores, utilizado en control industrial, automoción y electrónica de consumo, con capacidad de rendimiento confiable y predecible. Es altamente portátil y ofrece gestión de tareas, semáforos, colas y temporizadores para sistemas complejos y eficientes en recursos.

Paho MQTT *client*

Paho [25] es una biblioteca cliente MQTT de código abierto que se utiliza para conectar aplicaciones a un broker MQTT. Paho es compatible con una amplia variedad de plataformas y lenguajes de programación, y ofrece una API sencilla para publicar y suscribir mensajes. Es ampliamente utilizado en aplicaciones de IoT y M2M para la transmisión eficiente de datos.

2.3. Herramientas utilizadas

Docker

Docker [26] es una plataforma de software que permite a los desarrolladores crear, desplegar y ejecutar aplicaciones en contenedores. Estos contenedores permiten que las aplicaciones se ejecuten de manera aislada del sistema operativo y otras aplicaciones, lo que facilita su portabilidad y escalabilidad.

Docker Compose

Docker Compose [27] es una herramienta que permite definir y ejecutar aplicaciones de múltiples contenedores Docker. Permite a los desarrolladores especificar los servicios y la configuración de cada contenedor en un archivo YAML para simplificar la creación, ejecución y gestión de aplicaciones complejas.

Redpanda

Redpanda [28] es una plataforma de streaming de datos distribuida y de alto rendimiento que combina las funcionalidades de Kafka y Redis. Es una solución escalable y confiable para el procesamiento de datos en tiempo real en entornos empresariales, y es compatible con una amplia variedad de casos de uso, desde el análisis de datos hasta la inteligencia artificial y el aprendizaje automático.

Git

Git [29] es un sistema de control de versiones distribuido que se utiliza para rastrear los cambios en el código fuente de un proyecto de software. Permite a los desarrolladores trabajar en colaboración en el mismo código fuente y hacer un seguimiento de las diferentes versiones y ramas del proyecto.

MQTT.fx

MQTT.fx [30] es una herramienta de escritorio de código abierto que se utiliza para probar y depurar conexiones MQTT. Ofrece una interfaz gráfica de usuario fácil de usar para interactuar con brokers MQTT y suscribirse a temas y mensajes. MQTT.fx es compatible con una variedad de características de seguridad, como TLS/SSL y autenticación, lo que lo hace adecuado para su uso en entornos de producción.

CLion

CLion [31] es un entorno de desarrollo integrado (IDE) para programar en C y C++, que ofrece herramientas para la edición de código, depuración, refactorización y gestión de proyectos. Es conocido por su alta capacidad de análisis estático de código, lo que permite a los desarrolladores encontrar errores de forma eficiente.

Wireshark

Wireshark [32] es una herramienta de análisis de redes de código abierto y gratuita, utilizada para capturar y analizar paquetes de datos en tiempo real. Permite examinar el tráfico de red para identificar problemas de rendimiento, seguridad y configuración, y es compatible con una amplia variedad de protocolos de red.

Herramientas del navegador de internet

Las herramientas del navegador de Internet son características incorporadas en los navegadores web que permiten a los usuarios analizar y modificar elementos de una página web, como su estructura HTML, CSS y JavaScript. Algunas herramientas comunes incluyen la consola de desarrollador, el inspector de elementos,

el depurador de JavaScript y el analizador de red, que ayudan a los desarrolladores a depurar problemas y mejorar la calidad de sus sitios web.

JTAG

JTAG (Joint Test Action Group) [33] es un estándar para la depuración y programación de dispositivos electrónicos. Permite acceder a los pines de prueba de un dispositivo para realizar pruebas de hardware y depuración a nivel de circuito. JTAG se ha convertido en un estándar común para la depuración y programación de dispositivos, y muchas herramientas de desarrollo de software y hardware lo admiten.

ST-Link

ST-Link [34] es una herramienta de programación y depuración de microcontroladores fabricada por STMicroelectronics. Se utiliza para programar y depurar microcontroladores STM32 y otros dispositivos compatibles con JTAG o SWD. La herramienta se conecta al ordenador mediante USB y se puede usar con una variedad de entornos de desarrollo integrados (IDE) y herramientas de depuración.

STM32CubeMX

STM32CubeMX [35] es una herramienta de configuración gráfica para dispositivos STM32 que permite a los desarrolladores generar automáticamente código inicial para su proyecto. Ofrece una interfaz de usuario intuitiva que simplifica la configuración de periféricos y pines, y admite una variedad de opciones de generación de código.

STM32CubeProgrammer

STM32CubeProgrammer [36] es una herramienta de programación y actualización de firmware para dispositivos STM32 que permite programar y depurar dispositivos STM32, así como actualizar su firmware en campo. Es compatible con una variedad de interfaces de programación, como JTAG, SWD y UART, y es fácil de usar gracias a su interfaz gráfica de usuario.

Capítulo 3

Diseño e implementación

3.1. Descripción del sistema

El *SAL/T*, según sus siglas, Sistema de Aislamiento Limitado o Total [37], es un sistema que le permite al conductor la activación y desactivación del modo aislado limitado. En este modo, el equipo permite la circulación de la formación al desactivar las señales de corte de tracción y freno de emergencia generadas por los otros subsistemas. Para que esta operación se realice de forma segura, se debe monitorear la velocidad de la formación tal que sea posible evitar que supere cierto valor máximo. Se considera un sistema crítico debido a que, en caso de fallar, puede ocasionar lesiones o muertes de personas, dañar el medio ambiente y/o generar grandes pérdidas materiales.

En trabajos anteriores se han desarrollado las primeras cinco fases de catorce que componen el ciclo de vida según la norma *UNE-EN 50126*, centradas en el relevamiento de la necesidad y la obtención de los requerimientos técnicos del sistema.

En esta oportunidad, tal como se puede observar en el margen inferior derecho de la figura 1, se plantea el desarrollo de una central operativa. Se trata de una plataforma web que cuenta con una unidad lógica de compartición y empaquetado de software posibilitando la administación, la configuración y el monitoreo en tiempo real de la información recibida y transmitida por parte de cada dispositivo *SAL/T*.

En caso de ser necesario, a partir de la activación de una señal crítica provista por el artefacto *Hasler* [1] que se encuentra instalado en cada formación ferroviaria, un operario puede accionar alguno de los comandos que se listan a continuación:

- **Modo aislado total**: se habilita la tracción y se libera el freno de emergencia, independientemente de cualquier condición.
- **Modo coche en deriva**: se corta la tracción y se libera el freno de emergencia, independientemente de cualquier condición.
- **Modo parada total**: se corta la tracción y se aplica el freno de emergencia, independientemente de cualquier condición.
- **Modo intermitente**: se habilita la tracción y se aplica el freno de emergencia en ciclos de tiempo configurables.
- Anulación de comandos remotos vigentes.

[1] Es un sistema electromecánico que registra los eventos de velocidad del material rodante, la señal de anuncio y la señal de frenado automático.

Para el seguimiento de las variables, publicadas por cada dispositivo *SAL/T*, que permiten la visualización en la plataforma, se realiza la suscripción a un tópico del *broker MQTT* y se registra la información indexada en la base de datos de un servidor. En consecuencia, resulta de extrema importancia que la comunicación a través de *MQTT* opere sobre el protocolo de encriptación *SSL/TSL*, de modo que se exponga un considerable nivel de seguridad; de igual manera, es necesaria la utilización de certificados para el acceso a la lectura y la escritura en la base de datos.

Así mismo, la página web ofrece la configuración de los parámetros de cada dispositivo conectado a la formación ferroviaria, la administración de los roles y la autenticación de los usuarios.

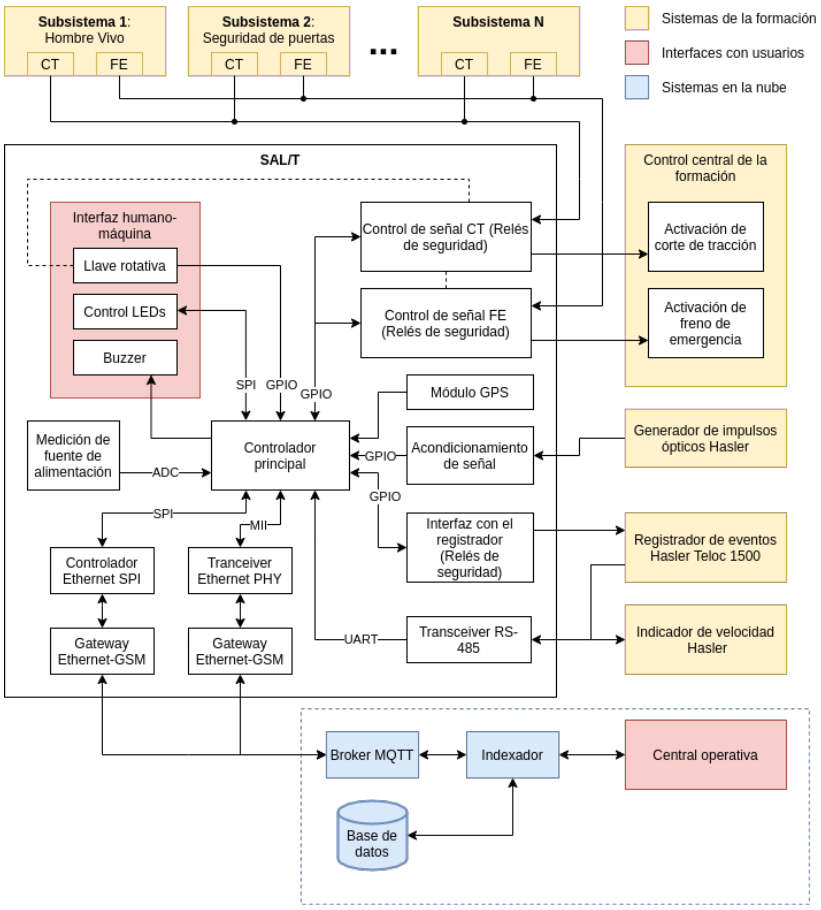

FIGURA 3.1. Diagrama en bloques del sistema SALT.

3.2. Arquitectura del sistema

En este apartado se ahonda en la estratificación del *stack IoT*, como se puede ver en la figura 3.2, que se compone de varias capas que cumplen funciones específicas en el ámbito de los sistemas *IoT*. Se analiza, de este modo, cómo se relacionan las diferentes tecnologías que se utilizan en cada capa, y cómo se aprovechan estas interacciones para obtener soluciones integral y escalable.

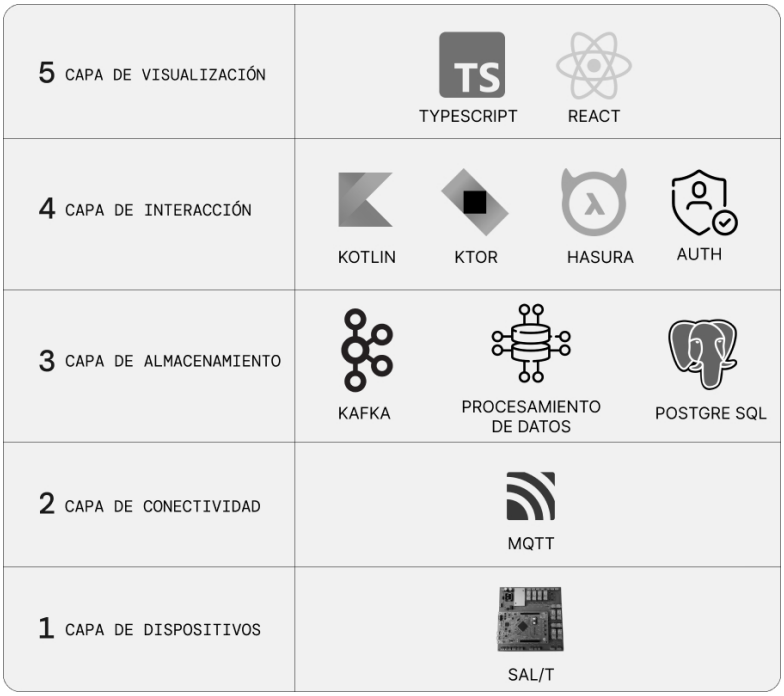

FIGURA 3.2. Arquitectura de la Central Operativa SAL/T.

3.2.1. Capa de dispositivos

Sobre la base de un prototipo del Sistema de Aislamiento Limitado o Total, *SAL/T*, se desarrolló mediante el kit de desarrollo *Nucleo F429* [38], un cliente *MQTT* que permite la publicación de la velocidad de la formación y de los parámetros provistos por los subsistemas de falla, entre otras variables; y llegado el caso, el dispositivo pueda realizar la lectura de diferentes parámetros configurables.

3.2.2. Capa de conectividad

Para la comunicación entre los dispositivos *SAL/T* y la capa de almacenamiento se emplea un *Broker MQTT* que oficia de orquestador entre el cliente, el dispositivo *SAL/T* que publica los mensajes y *Apache Kafka*, un sistema distribuido que efectúa la lectura de la información y su posterior procesamiento.

Dado que los mensajes intercambiados entre las partes contienen información sensible, se ha optado por agregar la capa de seguridad *TLS*. En este sentido, resulta indispensable utilizar el certificado *X509* para prevenir ataques del tipo *adversary-in-the-middle* y los certificados emitidos por autoridades reconocidas.

3.2.3. Capa de almacenamiento

La capa de almacenamiento se encuentra constituida por el sistema distribuido *Kafka* que se encarga de almacenar los datos de la aplicación. Entre ellos se destacan los mensajes *MQTT* enviados por los dispositivos *SAL/T*, como también la información referida a las formaciones y a los *SAL/T* que las ocupan. Además, se tienen servicios de procesamiento de datos que se encargan de adaptar la información de los tópicos en datos que posteriormente serán consumidos por los servicios que integran la capa superior.

Por otro lado, se tiene una base de datos relacional *PostgreSQL* para el almacenamiento de los datos provenientes del servicio de autenticación y el motor que facilita la interacción con la plataforma web. Los conectores se encargan de insertar los datos que se quieran disponer a la capa de visualización.

En la figura **devicesTrainEntity** se observa el modelo de datos que representa a los dispositivos *SAL/T*, las formaciones y las entidades. El primero cuenta con un identificador del dispositivo en el sistema, el estado de operación y una referencia al tren en el que se desplegó el artefacto. El modelo de la formación está enriquecido con la entidad a la que pertenece y el número de serie del tren. Por último, las entidades están compuestas por nombre y descripción. En la figura **devicesSubsystems** se aprecian los subsistemas de fallas relacionados con el dispositivo *SAL/T*. Su modelo está integrado con el nombre del subsistema, el identificador del dispositivo al que está asociado y el estado actual del mismo.

3.2.4. Capa de interacción

La capa de interacción se encuentra conformada por tres unidades. Por un lado, se ha desarrollado en el lenguaje *Kotlin* en conjunto con el *framework Ktor*, un microservicio de autenticación donde es posible la gestión de los usuarios con sus respectivos roles y también el *backend*. Entre sus tareas principales se encuentran las relacionadas con el protocolo *MQTT* para la actualización de los certificados que brindan seguridad a las comunicaciones, mediante el uso de la capa de seguridad *TLS*, y la indexación de los eventos que se publiquen en tiempo real; como la configuración general que permite el funcionamiento integral de los servicios y módulos dispuestos en el sistema.

Además, se dispone del motor *Hasura*, el cual se encuentra conectado a una base de datos relacional (*PostgreSQL*) basado en el lenguaje de consulta estructurado *SQL*, que permite exponer una *API GraphQL* a aquellos clientes que deseen obtener una visualización del modelo de datos propuesto en que se conjuga la información de cada formación ferroviaria con su correspondiente dispositivo *SAL/T* y diseño destinado al *frontend*. Más aún, el enlace permite realizar modificaciones y hasta remociones de las columnas propuestas en las tablas de cada esquema dentro de la base de datos.

3.2.5. Capa de visualización

En esta capa se ha utilizado el lenguaje de programación *TypeScript* junto con la biblioteca gráfica *React* para brindar una web reactiva en la que se elaboraron los formularios, las tablas y los paneles a partir de la explotación de la información

almacenada en la base de datos. Cabe destacar que la plataforma web se encuentra condicionada por el rol y la entidad a la que pertenece cada usuario, de modo que, cada formación ferroviaria tendrá un operador asignado, el cual podrá modificar los parámetros correspondientes con el dispositivo asociado.

En la figura **dashboard** se puede apreciar el panel de control de la central operativa al que tendrán acceso los operadores de cada entidad. En el mismo se observan distintas tarjetas con el estado de cada subsistema de fallas y del tren, como así también un velocímetro que informa la velocidad de la formación.

3.3. Arquitectura de datos

La estructura de la base de datos *SQL* se realiza a partir de un análisis detallado del sistema en el que se identifican las principales entidades: la formación ferroviaria, los subsistemas de falla y las áreas. Toda la información de estas entidades se nuclea en la entidad tren, la cual representa una formación ferroviaria con todos sus subsistemas y áreas asociadas. En efectp, el modelado de estas entidades y sus atributos correspondientes, se definen las relaciones entre ellas y se establecen las claves primarias y foráneas.

Para visualizar de manera más clara la estructura de la base de datos y sus relaciones, se utiliza el diagrama UML (Lenguaje de Modelado Unificado). En la siguiente figura se muestra el diagrama UML de la base de datos, donde se pueden observar las diferentes entidades, sus atributos y relaciones. A partir de este diagrama se procede a crear las tablas en la base de datos SQL y a definir sus columnas y restricciones, garantizando así la integridad de los datos y el correcto funcionamiento del sistema.

CONTINUAR CON EL DIAGRAMA UML CONECTANDO LAS ENTIDADES
VER Figures/salt_uml_drawio https://app.diagrams.net/

3.3.1. Entidades del modelo de datos

Schema auth_service

- **credentials**: Esta tabla almacena las credenciales de los usuarios para acceder al sistema. Tiene una clave foránea a la tabla **auth_service users** y es referenciada por la tabla **auth_service roles**.
- **roles**: Esta tabla almacena los roles de los usuarios. Tiene una clave foránea a la tabla **auth_service users** y es referenciada por la tabla **enums roles**.
- **users**: Esta tabla almacena la información básica de los usuarios del sistema. Es referenciada por la tabla **auth_service credentials** y la tabla **public users**.

Schema enums

- **device status**: Esta tabla almacena los estados posibles para los dispositivos. Es referenciada por la tabla **public devices**.
- **events**: Esta tabla almacena los tipos de eventos posibles. Es referenciada por la tabla **public internal events**.
- **fail system**: Esta tabla almacena los sistemas de falla posibles. Es referenciada por la tabla **public remote events** y la tabla **public subsystems**.
- **roles**: Esta tabla almacena los roles posibles para los usuarios del sistema. Es referenciada por la tabla **auth_service roles**.

Schema public

- **devices**: Esta tabla almacena información sobre los dispositivos del sistema. Tiene una clave foránea a la tabla **public train** y es referenciada por la tabla **public remote events** y la tabla **public internal events**.

- **entities**: Esta tabla almacena información sobre las entidades del sistema. Es referenciada por la tabla **public train**.
- **internal events**: Esta tabla almacena información sobre los eventos internos del sistema. Tiene claves foráneas a las tablas **public devices**, **enums events** y **public users**.
- **remote events**: Esta tabla almacena información sobre los eventos remotos del sistema. Tiene una clave foránea a la tabla **public devices** y es referenciada por la tabla **enums fail system**.
- **subsystems**: Esta tabla almacena información sobre los subsistemas del sistema. Tiene una clave foránea a la tabla **public devices** y es referenciada por la tabla **enums fail system**.
- **train**: Esta tabla almacena información sobre los trenes del sistema. Tiene una clave foránea a la tabla **public entities** y es referenciada por la tabla **public devices**.
- **user data**: Esta tabla almacena información adicional de los usuarios del sistema. Es referenciada por la tabla **public users**.
- **users**: tabla que contiene información de los usuarios. Tiene una relación de clave externa con la tabla entities.

3.4. Seguridad del sistema

En la actualidad, la seguridad en los protocolos de comunicación es una preocupación constante en el mundo digital. Con el creciente número de amenazas cibernéticas y la sofisticación de los ataques, es esencial que los protocolos de comunicación sean lo más seguros posible. Es fundamental que los protocolos de comunicación se desarrollen y se implementen con medidas de seguridad adecuadas para garantizar la privacidad y la integridad de los datos transmitidos.

3.4.1. JWT

JWT (JSON Web Token) es un estándar de token de seguridad que se utiliza para autenticar y autorizar a los usuarios en aplicaciones web y móviles. Dentro del protocolo HTTP, JWT se utiliza como un mecanismo de autenticación para permitir el acceso seguro a recursos protegidos.

Para utilizar JWT en HTTP, primero se debe generar un token JWT en el servidor después de que el usuario se autentica correctamente. Este token se envía de vuelta al cliente como respuesta a la solicitud de autenticación.

Luego, para acceder a recursos protegidos, el cliente debe incluir el token JWT en la solicitud HTTP en la cabecera de autorización. La cabecera de autorización tiene el prefijo "Bearer"seguido del token JWT. De esta manera, el servidor puede verificar la validez del token y si el usuario tiene los permisos necesarios para acceder a la información solicitada.

Una vez que el servidor valida el token JWT y verifica que el usuario tiene los permisos necesarios, puede enviar la respuesta solicitada al cliente. Si el token no es válido o ha expirado, el servidor puede denegar el acceso al recurso solicitado.

En resumen, JWT se utiliza dentro del protocolo HTTP como un mecanismo de autenticación seguro para permitir el acceso a recursos protegidos. Al utilizar JWT, se puede evitar la necesidad de mantener una sesión persistente en el servidor y se puede compartir información de usuario entre diferentes servicios sin la necesidad de almacenarla en el servidor.

3.4.2. MQTT v5.0

En la última versión del protocolo de mensajería *MQTT*, la versión 5.0, se incluyen varias mejoras en términos de seguridad en comparación con versiones anteriores, lo que lo hace más seguro y confiable para su uso en entornos *IoT* y *M2M*.

Una de las mejoras más importantes en *MQTT v5.0* es la inclusión de un mecanismo de autenticación mejorado. Ahora, el cliente y el servidor pueden autenticarse mutuamente utilizando certificados y credenciales de usuario. Esto ayuda a prevenir ataques de suplantación de identidad y a garantizar que solo los usuarios autorizados puedan acceder al servidor *MQTT*.

Otra mejora importante en *MQTT v5.0* es el soporte para *TLS 1.3*, la última versión del protocolo de seguridad de transporte. *TLS 1.3* ofrece un cifrado más fuerte y un intercambio de claves más seguro que las versiones anteriores, lo que ayuda a proteger los datos transmitidos contra ataques de escucha y manipulación.

MQTT v5.0 también introduce la capacidad de establecer límites de tamaño máximo de paquete, lo que ayuda a prevenir ataques de inundación de paquetes que pueden afectar la disponibilidad del servidor *MQTT*.

Además, la versión 5.0 de *MQTT* también incluye la capacidad de definir propiedades adicionales en los mensajes *MQTT*, lo que permite a los clientes y servidores intercambiar información adicional sobre los mensajes, como la hora en que se creó el mensaje o su nivel de prioridad.

3.5. Desarrollo del frontend

En esta sección, se aborda la estructura de la aplicación web, incluyendo la organización de sus componentes y la relación entre ellos. Además, se analizan los diferentes roles que pueden tener los usuarios de una aplicación y cómo afectan a la funcionalidad de la misma. Por último, se describe la implementación del cliente web y los clientes que se ven involucrados en cada etapa.

3.5.1. Estructura de la aplicación web

La estructura de la aplicación se inspira en la herramienta *Create React App* [2], que postula una estructura de archivos predefinida para crear aplicaciones web con React. En efecto, se plasman un conjunto de convenciones y buenas prácticas que permiten organizar el código de una forma clara y fácilmente escalable.

La organización de archivos, utilizando Typescript y *inline JSS*, consta de los siguiente elementos:

- La carpeta *"public"*, que contiene los archivos *HTML* y otros recursos que se deben servir estáticamente al navegador.
- La carpeta *"src"*, que es el directorio principal de la aplicación y contiene los archivos de código fuente. Dentro de esta carpeta se encuentra:
  - Los archivos de configuración, como ïndex.tsx

    y *.App.tsx"*, que se encargan de inicializar la aplicación y definir su estructura básica, utilizando la sintaxis de *Typescript*.
  - Los componentes de la aplicación, organizados en subdirectorios según su funcionalidad. Cada componente suele estar formado por un archivo *".tsx"* en la que define su lógica y un archivo de extensión *".ts"* que define el procesamiento de la información previo a su renderización.
  - Otros recursos, como imágenes, fuentes o archivos JSON que se utilizan en la aplicación.
- La carpeta "node_modules

  , que contiene todas las dependencias de la aplicación.
- Los archivos *"package.json"* y *"package-lock.json"*, que definen la configuración y las dependencias del proyecto.
- Otros archivos de configuración, como ".gitignore

  o ".env

  , que se utilizan para personalizar la configuración de la aplicación.

3.5.2. Roles

La aplicación web cuenta con una estructura de roles que determinan los permisos y acciones que los usuarios pueden realizar en la plataforma. En total, se establecen cuatro roles, que se listan a continuación:

[2] https://create-react-app.dev/

- **Visitante:** Este rol tiene los permisos más limitados en la plataforma y se le permite tan solo registrarse o ingresar al sistema.
- **Operador:** Las principales tareas del operador se centran en visualizar y/o controlar una o más formaciones ferroviarias, a través de un panel de control, asociadas a una entidad.
- **Supervisor:** Es aquel que asigna el *SAL/T* a una formación ferroviaria y a un operador. Además, tiene el control absoluto de todas las formaciones correspondientes a una entidad.
- **Súper usuario:** Este es el rol con mayores permisos en la plataforma, y quien se responsabiliza de observar las tres entidades de la plataforma y a la asignación de supervisores a cada entidad, entre otras acciones disponibles en la plataforma.

Cada uno de estos roles se diseña para brindar distintos niveles de acceso y control en la plataforma, según las necesidades y responsabilidades de los usuarios en cuestión. Con esta estructura de roles, se busca garantizar la seguridad y privacidad de los datos y acciones de los usuarios.

3.5.3. Modelo

ESCRIBIR ACERCA DEL MODELADO DE DATOS EN LA PLATAFORMA WEB

3.5.4. Implementación de la plataforma

La plataforma consiste en un conjunto de páginas que se desarrollan con la biblioteca *React Router DOM* [3]. La misma facilita la navegación y el enrutamiento de la aplicación web de una sola página.

En consecuencia, se presentan las principales páginas de la página web con sus rutas asoaciadas y componentes asociados:

dashboard, sign in, configuration, entities, trains, subsystems of each train, logs, etc

Login: **hablar sobre la interacción con jwt** *Referecia - bastian pag 48* **add figure**

Tabla de trenes+salt: **add figure**

Tabla de usuarios: **add figure**

Dashboard: **add figure**

Modales: *crear: usuario, tren, dispositivo ; configuracion: sub sistemas de seguridad, etc*

[3] https://reactrouter.com/en/main

3.6. Desarrollo del backend

A partir del uso de archivo docker-compose, que se ejecuta en un entorno Docker, para proporcionar el conjunto completo de servicios para enviar, recibir y almacenar los datos a través de diferentes protocolos. Cada servicio se ejecuta en su propio contenedor y se conecta a los otros servicios según el siguiente formato:

AGREGAR IMAGEN DEL ENTORNO DOCKER CON SUS CONTENEDORES Y CONEXIONES https://www.gotoiot.com/pages/articles/docker_intro/index.html https://www.gotoiot.com/pages/articles/docker_intro/images/image2.png

- mosquitto: es un servidor MQTT de código abierto que se utiliza para enviar y recibir mensajes. Se utiliza la imagen .ecllipse-mosquittoz se mapea el puerto 1883 en el host al puerto 1883 en el contenedor. También se montan dos volúmenes, uno para la configuración del servidor y otro para los datos.
- hasura: es un motor de GraphQL que proporciona una API para interactuar con una base de datos. Se utiliza la imagen "hasura/graphql-engine:v2.1.0z se mapea el puerto 8080 en el host al puerto 8080 en el contenedor. Se configuran varias variables de entorno, incluyendo la URL de la base de datos y el secreto de administrador. También depende del servicio "postgres".
- postgres: es una base de datos relacional que se utiliza para almacenar datos. Se utiliza la imagen "postgres:13z se mapea el puerto 5432 en el host al puerto 5432 en el contenedor. Se configuran varias variables de entorno, incluyendo el nombre de la base de datos, el nombre de usuario y la contraseña. También se monta un volumen para almacenar los datos de la base de datos.
- zookeeper: es un servicio que se utiliza para coordinar los nodos de Kafka. Se utiliza la imagen çonfluentinc/cp-zookeeper:6.2.4z se configura el puerto de cliente en 2181.
- kafka: es un servicio de mensajería de código abierto que se utiliza para enviar y recibir mensajes. Se utiliza la imagen çonfluentinc/cp-kafka:6.2.4z se mapea el puerto 9092 en el host al puerto 9092 en el contenedor. Se configuran varias variables de entorno, incluyendo la conexión de Zookeeper y el número de réplicas de las particiones. También depende del servicio "zookeeper".
- kafka-ui: es una interfaz de usuario para administrar y monitorear clústeres de Kafka. Se utiliza la imagen "provectuslabs/kafka-uiz se mapea el puerto 8085 en el host al puerto 8080 en el contenedor. Se configuran varias variables de entorno, incluyendo la conexión al clúster de Kafka, el esquema de registro y el servidor de KSQLDB.

Mosquitto Broker

TALK ABOUT THE MOSQUITTO CONFIG FILE AND THE CLI INSTALLATION, USAGE, ETC.

Hasura

TALK ABOUT HASURA GUI, CLI AND USAGE. ADD AN EXAMPLE QUERY

Kafka

TALK ABOUT KAFKA AND REDPANDA. ADD AN EXAMPLE QUERY

3.7. Desarrollo del firmware

Explicar como se combinan todas las tecnologias y herramientas en el firmware. basarlo en el stack lwip

Referecia - pagina 43 *https://lse-posgrados-files.fi.uba.ar/tesis/LSE-FIUBA-Trabajo-Final-CEIoT-Pedro-Rosito-2021.pdf*

- based on: https://blog.naver.com/eziya76/221938551688
- nucleo F429ZI kit development
- free rtos
- tools: cube mx, cube programmer, mosquitto cli

3.8. Desarrollo de la API GraphQL

Referencias

- https://github.com/nandroidj/app-fullstack-base-2021-1c
  Detalles de implementación ->Ver los endpoints disponibles

Bibliografía

[1] CONICET-GICSAFe. Online. 2023. URL: https://sites.google.com/view/conicet-gicsafe/inicio.
[2] Trenes Argentinos. Online. 2023. URL: https://www.argentina.gob.ar/transporte/trenes.
[3] lwIP. Online. 2023. URL: https://savannah.nongnu.org/projects/lwip/.
[4] HTTP. Online. 2023. URL: https://www.ietf.org/rfc/rfc2616.txt.
[5] DHCP. Online. 2023. URL: https://www.rfc-editor.org/rfc/rfc2131.
[6] MQTT. Online. 2023. URL: https://mqtt.org/.
[7] UDP. Online. 2023. URL: https://www.rfc-es.org/rfc/rfc0768-es.txt.
[8] IPv4. Online. 2023. URL: https://www.rfc-editor.org/rfc/rfc791.
[9] ARP. Online. 2023. URL: https://tools.ietf.org/html/rfc826.
[10] Ethernet. Online. 2023. URL: https://www.rfc-editor.org/rfc/rfc894.
[11] TypeScript. Online. 2023. URL: https://www.typescriptlang.org/.
[12] React. Online. 2023. URL: https://reactjs.org/.
[13] Parcel. Online. 2023. URL: https://parceljs.org/.
[14] Apollo GraphQL. Online. 2023. URL: https://www.apollographql.com/.
[15] Material UI. Online. 2023. URL: https://mui.com/.
[16] JWT Decode. Online. 2023. URL: https://jwt.io/.
[17] Kotlin. Online. 2023. URL: https://kotlinlang.org/.
[18] Ktor. Online. 2023. URL: https://ktor.io/.
[19] Kafka. Online. 2023. URL: https://kafka.apache.org/.
[20] Hasura. Online. 2023. URL: https://hasura.io/.
[21] PostgreSQL. Online. 2023. URL: https://www.postgresql.org/.
[22] Eclipse Mosquitto. Online. 2023. URL: https://mosquitto.org/.
[23] Lenguaje C. Online. 2023. URL: https://www.iso.org/standard/74528.html.
[24] FreeRTOS. Online. 2023. URL: https://www.freertos.org/.
[25] Paho MQTT Client. Online. 2023. URL: https://www.eclipse.org/paho/.
[26] Docker. Online. 2023. URL: https://www.docker.com/.
[27] Docker Compose. Online. 2023. URL: https://docs.docker.com/compose/.
[28] Redpanda. Online. 2023. URL: https://vectorized.io/.
[29] Git. Online. 2023. URL: https://git-scm.com/.
[30] MQTT.fx. Online. 2023. URL: https://softblade.de/en/mqtt-fx/.
[31] CLion. Online. 2023. URL: https://www.jetbrains.com/es-es/clion/.
[32] Wireshark. Online. 2023. URL: https://www.wireshark.org/.
[33] JTAG. Online. 2023. URL: https://www.jtag.com/.
[34] ST-Link. Online. 2023. URL: https://www.st.com/en/development-tools/st-link-v2.html.
[35] STM32CubeMX. Online. 2023. URL: https://www.st.com/en/development-tools/stm32cubemx.html.
[36] STM32CubeProgrammer. Online. 2023. URL: https://www.st.com/en/development-tools/stm32cubeprog.html.

[37] Ariel Lutenberg Ivan Mariano Di Vito Pablo Gomez. «Sistema de supervisión de la seguridad del material ferroviario utilizando patrones de diseño». En: *Journal* (2019).
[38] NUCLEO-F429ZI. Online. 2023. URL: https://www.st.com/en/evaluation-tools/nucleo-f429zi.html.