# Índice de Documentación Técnica

1. Introducción

2. Tecnologías utilizadas

3. Arquitectura de la Aplicación

4. Descripción de los Modelos de Datos

4.1 Modelo Rail

4.2 Modelo Baliza

4.3 Modelo TrackerLogs

4.4 Modelo Associations

5. Servicios

5.1 ServiceBase

5.2 BalizasService

5.3 TrackerService

6. Proceso de Recepción de Datos MQTT

7. Flujo de Datos

8. Consideraciones de Despliegue

9. Pruebas y Ejecución del Sistema

9.1 Iniciar el Servidor Node.js

9.2 Iniciar el Servicio MQTT - baliza.js

9.3 Iniciar el Listener MQTT - mqttlistener.js

10.Conclución

Documentación Técnica de la Aplicación

# 1. Introducción

Esta aplicación es una solución para la gestión de balizas y seguimiento de trackers en un sistema ferroviario. Utiliza MQTT para la transmisión de datos en tiempo real, gestionando la información de las balizas y la ubicación de los trackers a través de un conjunto de servicios. La aplicación permite registrar, almacenar y procesar los datos relacionados con las balizas y los trackers, así como realizar actualizaciones de posición en tiempo real.

# 2. Tecnologías utilizadas

- \*\*Node.js\*\*: Plataforma de JavaScript del lado del servidor utilizada para construir la aplicación.  
- \*\*Sequelize\*\*: ORM (Object-Relational Mapping) utilizado para interactuar con la base de datos MySQL.  
- \*\*MQTT\*\*: Protocolo de mensajería liviano utilizado para la transmisión de datos en tiempo real entre el servidor y los dispositivos.  
- \*\*MySQL\*\*: Base de datos relacional donde se almacenan los registros de las balizas y los trackers.  
- \*\*Express\*\*: Framework para Node.js utilizado para crear el servidor web y manejar las rutas de la aplicación.

# 3. Arquitectura de la Aplicación

La aplicación sigue una arquitectura basada en microservicios, con una división clara entre los modelos de datos, los servicios que gestionan la lógica de negocio y los controladores que exponen las funcionalidades a través de una API. El flujo de datos sigue el siguiente patrón:  
1. Los dispositivos (trackers) envían datos de ubicación a través del protocolo MQTT.  
2. Los servicios procesan esos datos y los almacenan en una base de datos MySQL.  
3. Los controladores manejan la lógica de la API y permiten la consulta de datos a través de endpoints RESTful.  
Además, se utilizan asociaciones entre modelos en Sequelize para representar las relaciones entre los elementos del sistema, como la relación entre las balizas y los rieles.

# 4. Descripción de los Modelos de Datos

## 4.1 Modelo Rail

Atributos:  
- `id`: Identificador único del riel.  
Relaciones:  
- Tiene muchas \*\*balizas\*\*.  
- Tiene muchos \*\*trackerLogs\*\*.

## 4.2 Modelo Baliza

Atributos:  
- `id`: Identificador único de la baliza.  
- `nombre`: Nombre de la baliza.  
- `mayor`: El riel al que está asociada.  
- `minor`: Identificador de la baliza dentro del riel.  
- `intensidad`: Intensidad de la señal de la baliza.  
- `trackerId`: ID del tracker asociado.  
Relaciones:  
- Pertenece a un \*\*rail\*\*.

## 4.3 Modelo TrackerLogs

Atributos:  
- `id`: Identificador único del log de tracker.  
- `nombre`: Nombre de la baliza.  
- `mayor`: El riel al que está asociado el tracker.  
- `minor`: Identificador de la posición del tracker dentro del riel.  
- `trackerId`: ID del tracker.  
Relaciones:  
- Pertenece a un \*\*rail\*\*.

## 4.4 Modelo Associations

Define las asociaciones entre los modelos de Sequelize.

# 5. Servicios

## 5.1 BaseService

El `BaseService` es un servicio genérico que proporciona funciones CRUD (Crear, Leer, Actualizar, Eliminar) para interactuar con los modelos de datos.  
Métodos principales:  
- `create (Model, data) `: Crea un nuevo registro en la base de datos.  
- `findAll (Model, include) `: Obtiene todos los registros de un modelo.  
- `findById (Model, id, include) `: Obtiene un registro por su ID.  
- `update (Model, id, data) `: Actualiza un registro existente.  
- `remove (Model, id) `: Elimina un registro.

## 5.2 BalizasService

El `BalizasService` gestiona las operaciones específicas para las balizas. Utiliza el `BaseService` para realizar las operaciones CRUD, pero se especializa en la manipulación de datos de balizas.  
Métodos principales:  
- `create(data)`: Crea una nueva baliza.  
- `findAll () `: Obtiene todas las balizas, con la relación de rieles.  
- `findById(id)`: Obtiene una baliza por su ID.  
- `update (id, data) `: Actualiza una baliza existente.  
- `remove(id)`: Elimina una baliza.

## 5.3 TrackerService

El `TrackerService` gestiona las operaciones específicas para los logs de trackers.  
Métodos principales:  
- `create(data)`: Crea un nuevo log de tracker.  
- `findAll () `: Obtiene todos los logs de trackers.  
- `findById(id)`: Obtiene un log de tracker por su ID.  
- `update (id, data) `: Actualiza un log de tracker.  
- `remove(id)`: Elimina un log de tracker.  
- `actualizarPosicionDesdeMQTT(posicion)`: Actualiza la posición de un tracker desde los datos recibidos por MQTT.

# 6. Proceso de Recepción de Datos MQTT

El cliente MQTT se conecta a un broker y se suscribe al tópico `baliza/gps/#`. Cada vez que un mensaje es recibido, este es procesado, convirtiéndolo de un buffer a un objeto JSON. Si el mensaje contiene un `trackerId`, se guarda la baliza en el buffer correspondiente. Si el número de balizas para un tracker excede el límite, la baliza más antigua se elimina. Cada 20 segundos, el sistema verifica los buffers y procesa las balizas, enviando la información al controlador para ser gestionada.

# 7. Flujo de Datos

1. Los dispositivos (trackers) envían datos de ubicación a través del protocolo MQTT.  
2. Los datos son recibidos en el backend, procesados y almacenados temporalmente en buffers.  
3. Cada 20 segundos, los datos almacenados en los buffers son procesados y enviados a la base de datos.  
4. La base de datos almacena los logs de las balizas y los trackers.

# 8. Consideraciones de Despliegue

Para desplegar la aplicación, se recomienda configurar un servidor Node.js con soporte para MQTT y MySQL. Asegúrese de tener un archivo (.env) con las credenciales de acceso al servidor MySQL y las configuraciones del broker MQTT.

**9. Pruebas y Ejecución del sistema**

Para probar el sistema y ejecutar correctamente los servicios de la aplicación, se deben seguir los siguientes pasos en tres terminales diferentes:

## 9.1 Iniciar el Servidor Node.js

En la primera terminal, navega a la ruta \backend

Inicia el servidor backend con el siguiente comando:

npm start

Este comando inicia el servidor de la aplicación, permitiendo la interacción con la base de datos y las APIs para la gestión de los trackers, balizas y logs.

## 9.2 Iniciar el Servicio MQTT - baliza.js

En la segunda terminal, navega a la ruta backend/mqtt y ejecuta el servicio MQTT que simula la publicación de datos de las balizas:

node baliza

Este servicio simula las publicaciones de datos de las balizas a través del protocolo MQTT, generando mensajes con la información de las balizas cercanas a cada tracker.

## 9.3 Iniciar el Listener MQTT - mqttlistener.js

En la tercera terminal, navega a la ruta backend/utils y ejecuta el listener que recibe los mensajes de MQTT y los procesa:

node mqttlistener

Este servicio escucha los mensajes de MQTT que se publican desde el servicio baliza.js, procesando la información de las balizas y actualizando la base de datos con las nuevas posiciones de los trackers.

# 10. Conclusión

La aplicación proporciona una solución robusta para el seguimiento de trackers y balizas en un entorno ferroviario. Se han implementado soluciones eficientes para la transmisión en tiempo real y el procesamiento de datos, asegurando un sistema confiable para el seguimiento de la posición de los trackers.