## 📘 Documentación Técnica - Sistema de Simulación y Localización de Trackers y Agujas vía MQTT

### Índice - Documentación Técnica

1. Descripción General del Proyecto
2. Tecnologías Utilizadas
3. Estructura del Proyecto
4. Simulación MQTT  
   4.1 Objetivo  
   4.2 MQTT Topics  
   4.3 transbordador-simulation  
   4.4 aguja-simulation
5. Modelos y Servicios  
   5.1 TrackerTransbordador  
   5.2 BalizasTransbordador  
   5.3 TrackerActivos  
   5.4 Agujas
6. Base de Datos
7. Cómo Probar el Servicio MQTT
8. Próximos pasos

### Descripción General del Proyecto

Simulación de posicionamiento de transbordadores, activos y lógica de aguja mediante mensajes MQTT.  
Este sistema permite recrear escenarios reales de operación ferroviaria en un entorno controlado, utilizando trackers y balizas Bluetooth para simular la localización de vehículos y activos. Además, incorpora lógica de desvío basada en el estado de agujas, permitiendo evaluar rutas posibles y validar algoritmos de decisión automática. El objetivo principal es facilitar el desarrollo, prueba y depuración de algoritmos de localización y control de tráfico ferroviario, todo esto en tiempo real mediante el protocolo MQTT.

### Tecnologías Utilizadas

- Node.js: Lógica de simulación y backend.

- Mosquitto (MQTT broker): Comunicación de mensajes en tiempo real.

- MySQL: Base de datos de persistencia.

- Sequelize: ORM para Node.js.

### 3. Estructura del Proyecto

📁 mqtt-simulation/

├── 📁 transbordador-simulation/

│ ├── publisher.js

│ ├── listener.js

│ ├── utils.js

├── 📁 aguja-simulation/

│ ├── cases/

│ ├── publisher.js

│ ├── listener.js

│ ├── utils.js

📁 models/

├── TrackerTransbordador.js

├── BalizasTransbordador.js

├── TrackerActivos.js

├── Aguja.js

📁 services/

├── TrackerTransbordador.Service.js

├── BalizasTransbordador.Service.js

├── TrackerActivos.Service.js

├── Aguja.Service.js

📁 sqlScripts/

├── Creacion\_tablas\_tracker\_agujas\_activos\_balizas.sql

├── Insertar\_tablas\_tracker\_agujas\_activos\_balizas.sql

├── Creacion\_tablas\_logs.sql

### 4.Simulación MQTT

4.1 Objetivo

Simular el comportamiento de los transbordadores, balizas y activos mediante envío de datos vía MQTT.  
- Determinar:  
 • En qué vía está el transbordador según intensidad de balizas.

• Si un transbordador está acoplado a otro transbordador.

• Si un activo está en un transbordador.

• Determinar la ubicación de un tracker mediante posición de agujas y señal de baliza.

### 4.2 MQTT Topics

|  |  |
| --- | --- |
| Topic | Descripción |
| 'position/tracker' | Envía datos de balizas de transbordadores |
| position/asset | Envía datos de balizas de activos |
| simulation/tracker | Cambios en el estado de las agujas |

### 4.3 transbordador-simulation

- publisher.js: simula envío de balizas de transbordadores.

- listener.js: escucha las señales simuladas para validación.

- /utils: funciones comunes como generación de payloads o intensidades aleatorias.

### 4.4 aguja-simulation

- cases/: casos de prueba que modifican el estado de las agujas.

- publisher.js: publica cambios de estado de aguja.

- listener.js: escucha las señales que afectan el camino de un tracker.

- /utils: lógica de decisión de vía según intensidades y configuración de aguja.

### 5. Modelos y Servicios

### 5.1 TrackerTransbordador

Modela los transbordadores con sus trackers asociados.

### 5.2 BalizasTransbordador

Modela las balizas Bluetooth fijas en los transbordadores.

### 5.3 TrackerActivos

Modela los activos (cargas u objetos) que se montan en los transbordadores.

### 5.4 Agujas

Modela las agujas con su localización, vía origen y sus vías destino.

Cada servicio (\*.service.js) contiene CRUD completo y funciones auxiliares

### 6. Base de Datos

* Script: Creacion\_tablas\_tracker\_agujas\_activos\_balizas.sql
* Insertar\_tablas\_tracker\_agujas\_activos\_balizas.sql
* Creacion\_tablas\_logs.sql (**Pendiente de implementar la funcionalidad**)

Tablas incluidas:  
 trackerstranbordador  
 balizastransbordador  
 trackeractivos

balizas  
 agujas

### 7. Como Probar el Servicio MQTT

Para realizar pruebas del sistema de simulación MQTT y verificar el funcionamiento de los transbordadores y las agujas, se deben seguir los siguientes pasos:

### ✅ Prueba de Transbordadores

1. Navegar a la ruta del proyecto:  
  
 backend\mqtt-simulation\transbordador-simulation\

2. Ejecutar el simulador de transbordadores con el siguiente comando:  
  
 node runAll.js

Este script inicia la simulación completa de los transbordadores, incluyendo la publicación de datos MQTT y el manejo de eventos.

### ✅ Prueba de Agujas

Tienes dos opciones para probar la simulación de las agujas:

* Opción 1: Ejecutar el simulador completo

1. Navegar a la carpeta:  
  
 backend\mqtt-simulation\aguja-simulation\

2. Ejecutar el simulador de agujas:  
  
 node runAguja.js

* Opción 2: Ejecutar casos individuales

1. Ir a la subcarpeta con los casos de prueba:  
  
 backend\mqtt-simulation\aguja-simulation\cases\

2. Ejecutar cualquier archivo individual según el caso que se desee probar:  
  
 node nombreDelArchivo.js  
 Por ejemplo:  
 node simulateCase1.js

### 8.Próximos pasos

A continuación, se detallan las acciones necesarias para completar la integración del sistema de registro de logs en la simulación y localización vía MQTT:

### Creación de Modelos Sequelize

Ya se cuenta con los scripts SQL para las tablas trackerPositionLogs, couplingLogs y assetMountLogs. El siguiente paso es crear los modelos Sequelize correspondientes. Para mantener la estructura actual del proyecto, estos modelos deben agregarse en la carpeta /models/, y seguir las convenciones usadas en los modelos existentes (TrackerTransbordador, BalizasTransbordador, etc.).

Cada modelo debe incluir su respectiva estructura (campos, tipos de datos, claves foráneas si aplica (se registran en associations.js) y las configuraciones necesarias según las convenciones de Sequelize.

### Servicios Asociados

Una vez definidos los modelos, será necesario crear sus servicios asociados en /services/. Cada servicio debe permitir, como mínimo, insertar nuevos registros, consultar logs por filtros como trackerId, tipo de evento y cualquier utilidad específica requerida por la lógica del sistema.

Servicios a implementar:

* TrackerPositionLogs.Service.js
* CouplingLogs.Service.js
* AssetMountLogs.Service.js

### Funciones de Inserción desde los Listeners

Será fundamental definir funciones reutilizables que puedan ser invocadas desde los listeners MQTT (tracker, asset, activeTrackerListener o logger) para insertar información en los logs. Validar y transformar los datos si es necesario. Llamar al servicio correspondiente para registrar la entrada en la base de datos.

Estas funciones actuarán como un puente entre el mensaje MQTT recibido y el acceso a base de datos a través de los servicios.