Documentación del Proyecto API

Figura 1: *
Logo de la Universidad

Integrantes del Equipo:

Juan Julián Paniagua Rico - a2213332303
Isaac Sayeg Posadas Perez - a2213332197
Jorge Alberto García Azzua - a2221335006

Universidad Autónoma de Tamaulipas
Diseño de Sistemas Embebidos
16 de mayo de 2025

Índice

1. Introducción

Este documento proporciona una descripción detallada de la API REST desarrollada para el curso de Diseño de Sistemas Embebidos. El proyecto consiste en una API que interactúa con una base de datos PostgreSQL para almacenar y recuperar datos relacionados con el monitoreo de dispositivos y la toma de decisiones basada en parámetros de velocidad y distancia.

2. Tecnologías Utilizadas

2.1. Stack Tecnológico Principal

- **Backend:** Node.js v20.10.5 con Express v4.18.2
- **Lenguaje:** TypeScript v5.3.3
- **Base de Datos:** PostgreSQL v8.11.3
- **ORM:** Prisma v6.7.0
- **Motor de Plantillas:** EJS v3.1.10
- **Contenedorización:** Docker

2.2. Dependencias Principales

- **dotenv:** v16.3.1 - Gestión de variables de entorno
- **@prisma/client:** v6.7.0 - Cliente de Prisma para interacción con la base de datos
- **ts-node:** v10.9.2 - Ejecución de TypeScript
- **nodemon:** v3.0.2 - Reinicio automático del servidor durante desarrollo

3. Arquitectura del Proyecto

El proyecto sigue una arquitectura MVC (Modelo-Vista-Controlador) modificada con la siguiente estructura:

3.1. Estructura de Directorios

- **src/** - Directorio principal del código fuente
  - **config/** - Archivos de configuración y constantes
  - **controller/** - Lógica de controladores
  - **models/** - Modelos de datos y consultas
  - **routes/** - Definición de rutas API
  - **services/** - Lógica de negocio
  - **views/** - Vistas EJS
  - **generated/** - Código generado por Prisma
- **prisma/** - Definición del esquema de base de datos
- **dist/** - Código JavaScript compilado

4. Base de Datos

4.1. Esquema de Base de Datos

El sistema utiliza dos tablas principales:

4.1.1. devices_info

Almacena información sobre los dispositivos:

- **id_device:** Clave primaria autoincremental
- **id_type:** Identificador del tipo de dispositivo
- **id_signal_type:** Identificador del tipo de señal
- **nombre:** Nombre del dispositivo
- **vendor:** Fabricante del dispositivo

4.1.2. devices_records

Registra las lecturas de los dispositivos:

- **id_record:** Clave primaria autoincremental
- **id_device:** Clave foránea referenciando devices_info
- **current_value:** Valor registrado
- **date_record:** Marca de tiempo del registro

5. API Endpoints

5.1. Endpoints Disponibles

5.1.1. PUT /insert_test

- **Descripción:** Inserta un registro de decisión
- **Parámetros:**
  - velocidad (número)
  - distancia (número)
  - decision (número)
- **Respuesta:** Confirmación de inserción

6. Configuración del Entorno de Desarrollo

6.1. Requisitos Previos

- Node.js (v20.10.5 o superior)
- PostgreSQL
- Docker (opcional)
- npm (incluido con Node.js)

6.2. Instalación y Configuración

1. Clonar el repositorio
2. Ejecutar `npm install` para instalar dependencias
3. Crear archivo `.env` con las variables de entorno necesarias:

```
DATABASE_URL="postgresql://usuario:contrase a@localhost:5432/nombre_db"
DB_USER=usuario
DB_HOST=localhost
DB_NAME=nombre_db
DB_PASSWORD=contrase a
DB_PORT=5432

```

4. Ejecutar migraciones de Prisma: `npx prisma migrate dev`
5. Iniciar el servidor: `npm run dev`

6.3. Scripts Disponibles

- `npm start` - Inicia el servidor en producción
- `npm run dev` - Inicia el servidor en modo desarrollo
- `npm run build` - Compila el código TypeScript
- `npm run watch` - Compila en modo observador

7. Despliegue

7.1. Despliegue con Docker

1. Construir la imagen: `docker build -t api-proyecto .`
2. Ejecutar el contenedor: `docker run -p 3000:3000 api-proyecto`

7.2. Despliegue Manual

1. Ejecutar `npm run build`
2. Configurar variables de entorno
3. Ejecutar `npm start`

8. Documentación Detallada del Código

8.1. Módulos Principales

8.1.1. Conexión a Base de Datos (conexion_db.ts)

Este módulo maneja la conexión a la base de datos PostgreSQL:

- **Pool de Conexiones:** Implementa un pool de conexiones usando pg-pool
- **Variables de Entorno:** Utiliza dotenv para gestionar la configuración segura
- **Funciones Principales:**
  - `get_connection()`: Establece y verifica la conexión a la base de datos
  - `pool`: Instancia del pool de conexiones con configuración desde variables de entorno

8.1.2. Punto de Entrada (index.ts)

Archivo principal que inicializa la aplicación:

- **Configuración Express:** Inicialización del servidor web
- **Gestión de Rutas:** Implementación del endpoint PUT /insert_test
- **Manejo de Errores:** Implementación de try-catch para gestión de errores
- **Cierre Seguro:** Manejo del evento SIGINT para cierre limpio de conexiones

8.1.3. Consultas Almacenadas (sp_queries.ts)

Módulo para gestionar procedimientos almacenados:

- **sp_insert_decision:**
  - **Propósito:** Inserta registros de decisiones en la base de datos
  - **Parámetros:**
    - velocidad_param_numeric: número
    - distancia_param_numeric: número
    - decision_param_numeric: número
  - **Manejo de Errores:** Implementa try-catch-finally con cierre de conexión

8.2. Módulos en Desarrollo

8.2.1. Controladores (sp_controller.ts)

Módulo para la lógica de controladores:

- Actualmente importa servicios para futura implementación
- Pendiente implementación de lógica de negocio

8.2.2. Servicios (services.ts)

Módulo para la lógica de negocio:

- Archivo creado y preparado para implementación
- Destinado a contener la lógica de negocio principal

8.2.3. Rutas (routes.ts)

Módulo para la definición de rutas:

- Preparado para implementación de rutas adicionales
- Permitirá una mejor organización de endpoints

8.2.4. Operaciones CRUD (cruds.ts)

Módulo para operaciones básicas de base de datos:

- Planificado para implementar operaciones CRUD usando Prisma ORM
- Pendiente desarrollo de funciones básicas (Create, Read, Update, Delete)

9. Flujo de Datos

1. El cliente realiza una petición HTTP al endpoint /insert_test
2. El servidor valida y procesa los datos recibidos
3. Se establece conexión con la base de datos a través del pool
4. Se ejecuta el procedimiento almacenado con los parámetros
5. Se devuelve la respuesta al cliente

10. Mejores Prácticas Implementadas

- **Seguridad:** Uso de variables de entorno para datos sensibles
- **Gestión de Conexiones:** Implementación de pool de conexiones
- **Tipado Fuerte:** Uso de TypeScript para prevenir errores
- **Manejo de Errores:** Implementación de try-catch en operaciones críticas
- **Modularidad:** Separación clara de responsabilidades en módulos

11. Conclusión

Este proyecto implementa una API REST robusta para el monitoreo de dispositivos y toma de decisiones. Utiliza tecnologías modernas y sigue las mejores prácticas de desarrollo. La arquitectura modular permite una fácil expansión y mantenimiento del sistema.