API RESTful para Sistemas Embebidos

Documentación Técnica

Integrantes del Equipo:

Juan Julián Paniagua Rico - a2213332303
Isaac Sayeg Posadas Perez - a2213332197
Jorge García Azua - matricula aqui

Universidad Autonoma de Tamaulipas
8° Semestre - 2025

Índice

1. Introducción

Este proyecto implementa una API RESTful desarrollada en Node.js que permite la gestión de registros y decisiones para dispositivos embebidos.

La solución integra una base de datos SQL Server para gestionar las tablas y procedimientos almacenados necesarios para sensores o actuadores. Este enfoque asegura la consolidación, consulta, y optimización de decisiones mediante algoritmos básicos embebidos.

2. Estructura de la Base de Datos

La base de datos se organiza en varias tablas y procedimientos almacenados. A continuación se detalla cada uno:

2.1. Tablas

La base de datos contiene las siguientes tablas principales utilizadas por la API:

- **devices_info:** Maneja la información básica de cada dispositivo.
  - **id_device:** Identificador único autoincremental.
  - **id_type:** Indica el tipo (sensor o actuador).
  - **id_signal_type:** Relacionado al tipo de señal (digital o analógica).
  - **name:** Nombre descriptivo del dispositivo.
  - **vendor:** Fabricante del dispositivo.
- **devices_records:** Almacena los datos históricos de cada dispositivo.
  - **id_record:** Identificador único para cada lectura.
  - **id_device:** Relación con el dispositivo al que pertenece el registro.
  - **current_value:** Valor registrado.
  - **date_record:** Fecha y hora del registro.
- **toma_decisiones:** Contiene los datos para decisiones calculadas.
  - **id_decision:** Identificador único de decisión.
  - **velocidad, distancia, decision:** Datos utilizados en el algoritmo.
  - **date_record:** Fecha y hora de la decisión.

2.2. Procedimientos Almacenados

Los principales procedimientos almacenados implementados son:

1. **SP_Insert_DevicesRecords:** Utilizado por la ruta POST para registrar el valor actual de un dispositivo.

```
    CREATE PROCEDURE SP_Insert_DevicesRecords
        @id_device as numeric(18,0),
        @current_value as numeric(18,0)
    AS
    BEGIN
        INSERT INTO [devices_records]
            ([id_device], [date_record], [current_value])
        VALUES (@id_device, GETDATE(), @current_value)
    END
```

2. **SP_SelectALL_records:** Recupera todos los registros existentes en la tabla devices_records.

```
    CREATE PROCEDURE SP_SelectALL_records
    AS
    BEGIN
        SELECT DI.id_device, DI.name "NAME",
               DR.current_value "CURRENT_VALUE",
               DR.date_record "DATE_RECORD"
        FROM devices_records DR
        INNER JOIN devices_info DI
            ON DR.id_device = DI.id_device
    END
```

3. **SP_SelecLastDecision:** Recupera la última decisión registrada basada en las marcas de tiempo.

```
    CREATE PROCEDURE SP_SelecLastDecision
    AS
    BEGIN
        SELECT TOP 1 *
        FROM toma_decisiones
        ORDER BY date_record DESC
    END
```

3. Explicación del Código Backend

3.1. Punto de Entrada del Servidor

El archivo **index.js** inicializa el servidor Express, configurando el API para escuchar en el puerto 3000. Además, gestiona la ruta raíz y da soporte a JSON.

```
const express = require('express');
const v1 = require('./v1/routes/Routes');

const app = express();
const PORT = process.env.PORT || 3000;

app.use(express.json());  // Habilita el soporte para JSON

```

```
// Configuraci n de rutas
app.use("/api/v1", v1);

app.get("/", (req, res) => {
    res.send("<h1>API RESTful en NodeJS para Servicios
       Embebidos</h1>")
});

// El servidor escucha en el puerto 3000
app.listen(PORT, () => {
    console.log(`Servidor escuchando en el puerto: ${PORT}`);
});
```

3.2. Gestión de Rutas

En **Routes.js**, las principales rutas a controlar son declaraciones GET y POST:

- **GET /api/v1/**
  Devuelve registros al llamar al procedimiento almacenado SP_SelectALL_records:

```
    router.get("/", controller.getAll_records);
    // getAll_records llama a los servicios y retorna los
        datos.
```

- **POST /api/v1/registros**
  Inserta un registro nuevo utilizando el procedimiento SP_Insert_DevicesRecords:

```
    router.post("/registros", controller.insertRecord);
    // El cuerpo debe incluir Id_device y Current_value.
```

3.3. Servicios

El archivo **devicesService.js** implementa las funciones que conectan directamente con los procedimientos almacenados definidos:

```
const insertRecord = async function(JsonObj) {
    const id = JsonObj.Id_device;
    const current_value = JsonObj.Current_value;
    return await SP_Insert_DevicesRecords(id, current_value);
};
```

4. Evidencia de Ejecución del Código

A continuación, se presentan capturas de pantalla de la ejecución del código y las pruebas respectivas con la API:

4.1. Servidor Express Configurado

4.2. Prueba de Inserción en la Base de Datos

—

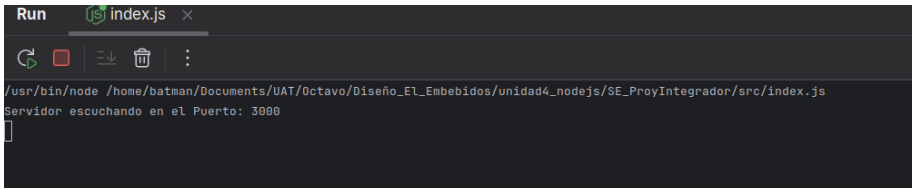

Figura 1: Servidor Express ejecutándose correctamente en el puerto 3000.

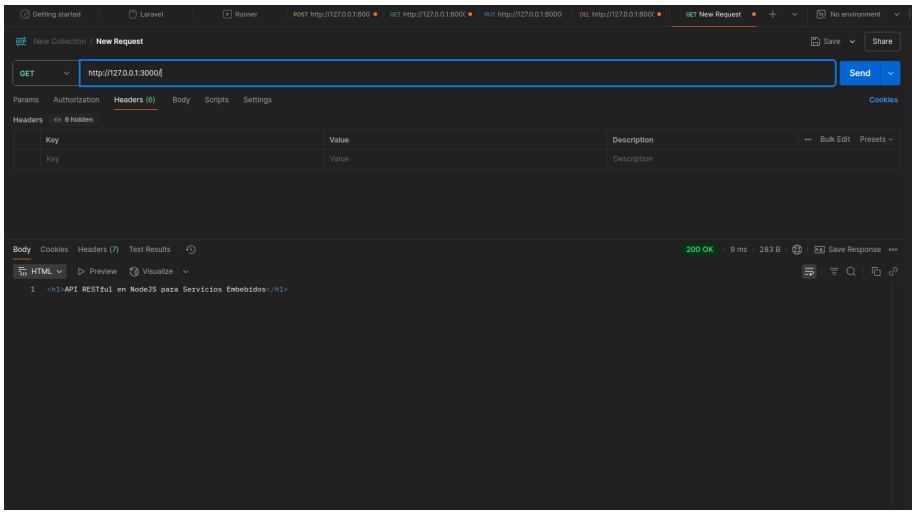

Figura 2: Prueba de registro nuevo utilizando el endpoint POST /api/v1/registros.