Backend SmartEnviro ISPC

1. Descripción general

El backend del sistema SmartEnviro ISPC es el núcleo que recibe, procesa, almacena y expone los datos IoT provenientes de gateways, endpoints y sensores distribuidos en campo.
Está desarrollado en Node.js (ECMAScript moderno) y utiliza:

MariaDB para almacenamiento estructurado (información de dispositivos, sensores y gateways).

InfluxDB para datos históricos de sensores (lecturas en tiempo real y series temporales).

MQTT/HTTP como protocolos de comunicación desde los dispositivos IoT.

El backend expone endpoints RESTful y funciones de sincronización automáticas para mantener la base de datos actualizada con el estado de toda la red IoT.

2. Arquitectura de datos

El sistema organiza los datos en tres niveles:

Gateways → dispositivos que coordinan endpoints LoRa y comunican con la nube.

Endpoints → nodos intermedios que concentran varios sensores.

Sensores → dispositivos que miden variables ambientales (temperatura, humedad, presión, gases, etc.).

Cada entidad se registra en la tabla dispositivos, y sus datos se guardan en datos_sensores.

3. Funciones principales

 getAllDevices()

Obtiene todos los dispositivos registrados (gateways, endpoints y sensores).
Devuelve un listado ordenado alfabéticamente con nombre, tipo, ubicación, estado y fecha de última conexión.
☞ Ideal para el panel de la web app, donde se muestra la lista completa de equipos activos.

 getDeviceById(deviceId)

Recupera la información detallada de un dispositivo específico según su ID.
Incluye tipo, ubicación, metadatos (como batería o estado LoRa) y tiempos de creación y actualización.
☞ Utilizado cuando un usuario selecciona un dispositivo desde el dashboard para ver sus datos.

 getDeviceSensorData(deviceId, limit)

Obtiene los últimos valores medidos por un dispositivo sensor determinado, limitando la cantidad de registros (por defecto 100).
Los resultados incluyen tipo de sensor, valor, unidad, fecha y metadatos.
☞ Permite construir gráficos de evolución y análisis de tendencias en la interfaz web.

 getDeviceStats(deviceId)

Calcula estadísticas globales del dispositivo:

Total de datos almacenados

Primer y último registro

Tipos de sensores conectados

Promedio de valores medidos

☞ Esta función alimenta los indicadores estadísticos del dashboard (por ejemplo, “Sensores activos”, “Última lectura”).

 createOrUpdateDevice(device)

Permite crear o actualizar un dispositivo automáticamente según su id_dispositivo.
Si ya existe, actualiza sus atributos (nombre, estado, ubicación, gateway vinculado, metadatos); si no, lo crea.
☞ Esta función se usa en todas las sincronizaciones de red (gateway, endpoint y sensores).
Incluye control de duplicados y manejo de actualizaciones parciales con COALESCE.

 syncGatewayToDB(gatewayData)

Sincroniza la información del gateway principal:

Estado LoRa y WiFi

Uptime

Identificador y nombre
 ☞ Se ejecuta cada vez que el gateway envía su estado general vía MQTT al backend.

 syncEndpointToDB(endpointData)

Actualiza los datos de los endpoints conectados a un gateway:

ID del endpoint

Nivel de batería

Cantidad de sensores

Estado de conexión y LoRa
 ☞ Esta sincronización mantiene actualizado el mapa de endpoints activos en la red.

 syncSensorToDB(sensorData)

Actualiza o registra los sensores individuales conectados a cada endpoint:

ID del sensor

Temperatura y humedad actuales

Estado y posición dentro del entorno
 ☞ Permite que la base de datos refleje en tiempo real las condiciones de cada punto de medición.

 getHistoricalSensorDataFromInfluxDB(deviceId, limit, startTime)

Consulta datos históricos almacenados en InfluxDB (por ejemplo, las últimas 24 horas).
 Soporta filtros por dispositivo y ordena los resultados por tiempo.
 ☞ Esta información alimenta los gráficos de la interfaz (tendencias, reportes, histórico de 7 días, etc.).

4. Manejo de errores y logs

El sistema implementa manejo robusto de excepciones con try/catch y registro mediante console.error().
Los errores comunes como "Dispositivo no encontrado" o "Duplicado en DB" son tratados para mantener la estabilidad del backend.

Además, los mensajes de sincronización ([DB-SYNC], [INFLUXDB], [DB-UPDATE]) permiten trazar el flujo completo de los datos desde el gateway hasta el almacenamiento final.

5. Conexión a bases de datos

El backend importa dos objetos principales desde ../db/index.js:

pool: conexión a MariaDB para operaciones estructuradas.

influxDB: cliente InfluxDB para series temporales (registros de sensores).

Ambas conexiones son asíncronas y optimizadas para alto rendimiento.

6. Flujo de sincronización

```
Gateway (ESP32)
   ↓ MQTT/WebSocket
Backend Node.js
   ↓
MariaDB → Dispositivos y estados
InfluxDB → Datos históricos
Web App → Visualización
```

El backend recibe datos desde MQTT, los clasifica según su tipo (gateway, endpoint, sensor), y actualiza automáticamente las bases de datos.
Luego, la Web App consume estos datos mediante API REST para mostrarlos en tiempo real.

7. Conclusión

El backend SmartEnviro ISPC constituye la columna vertebral del ecosistema IoT.
Su diseño modular y asincrónico permite:

Escalabilidad (nuevos sensores y endpoints se sincronizan automáticamente)

Robustez (reintentos automáticos ante desconexión o errores de DB)

Integración fluida con la capa WebApp y los dispositivos embebidos.