Módulo de Inicialización y Configuración del Servidor

Propósito General

Este módulo constituye el punto de entrada principal de la aplicación, configurando y lanzando un servidor completo que combina Express.js para APIs REST y WebSocket para comunicación en tiempo real. El servidor integra múltiples componentes del sistema en una arquitectura cohesiva que soporta tanto solicitudes HTTP tradicionales como conexiones WebSocket persistentes.

Arquitectura del Servidor

El servidor utiliza una arquitectura híbrida donde Express maneja las rutas HTTP/REST y un servidor WebSocket se monta sobre el mismo puerto HTTP. Esta aproximación unificada simplifica el despliegue y permite compartir la misma configuración de red y seguridad para ambos protocolos.

Configuración de Middlewares Base

La aplicación Express se configura con un conjunto completo de middlewares esenciales para seguridad, logging, y procesamiento de requests:

Helmet.js: Aplicado mediante el módulo de seguridad personalizado para proteger contra vulnerabilidades web comunes

Morgan: Configurado para logging detallado en desarrollo ("dev") y formato combinado en producción

Express.json: Con límite de 1MB para prevenir ataques de DoS por payloads grandes

Cookie Parser: Para manejo de cookies en autenticación

CORS: Configurado dinámicamente desde variables de entorno, soportando múltiples orígenes y credenciales

Sistema de Rutas API

El servidor organiza las rutas en módulos especializados que siguen una estructura lógica:

Autenticación: /api/auth - Endpoints de login con Google y desarrollo

Configuración Pública: /api/config - Configuración accesible para el frontend

Configuración del Sistema: /api/config - Endpoints administrativos con control de acceso

Datos de Temperatura: /api/temperature - Datos en tiempo real desde MQTT

Gestión de Datos: /api/data - Operaciones CRUD para dispositivos y sensores

Esta organización modular facilita el mantenimiento y permite una clara separación de responsabilidades entre diferentes funcionalidades del sistema.

Endpoint de Salud

El endpoint /health proporciona un mecanismo simple de verificación de estado, esencial para orquestadores de contenedores y sistemas de monitoreo. Retorna un JSON { ok: true } cuando el servidor está operativo.

Integración WebSocket

El servidor HTTP creado con createServer se pasa al módulo de WebSocket (initWebSocket) que maneja las conexiones en tiempo real en la ruta /ws. Esta integración permite que el mismo puerto maneje tanto tráfico HTTP como WebSocket, con el servidor WebSocket escuchando eventos de "upgrade" para promover conexiones HTTP a WebSocket.

Inicialización de Servicios

Durante el arranque, el servidor inicializa servicios críticos:

Conexión MQTT: Llama a mqttService.connect() para establecer conexión con el broker MQTT

WebSocket Server: Inicializa el servidor WebSocket para comunicación en tiempo real

Express Routes: Monta todas las rutas API definidas

Configuración desde Variables de Entorno

El servidor utiliza extensivamente las variables de entorno definidas en el módulo env.js:

PORT: Puerto de escucha del servidor

NODE_ENV: Ambiente de ejecución (desarrollo/producción)

CORS_ORIGIN: Orígenes permitidos para CORS

MQTT_BROKER_HOST/PORT: Configuración del broker MQTT

MQTT_TOPICS: Topics MQTT a los que suscribirse

Proceso de Arranque

Al iniciar, el servidor realiza las siguientes operaciones en secuencia:

Configura Express con middlewares de seguridad y parsing

Registra todas las rutas API organizadas por funcionalidad

Crea el servidor HTTP base

Inicializa el servidor WebSocket

Establece conexión con el broker MQTT

Pone el servidor en escucha en el puerto configurado

Logging de Inicialización

Durante el arranque, el servidor proporciona información detallada en consola:

Puerto de escucha HTTP y ruta WebSocket

Configuración del broker MQTT (host y puerto)

Lista de topics MQTT suscritos

Esta información es valiosa para verificar que la configuración se cargó correctamente y para diagnóstico de problemas de conectividad.

Consideraciones de Seguridad

La configuración de seguridad se aplica de manera consistente a todas las rutas:

Helmet protege contra vulnerabilidades comunes

CORS restringe orígenes según configuración

Las rutas sensibles implementan autenticación JWT individualmente

WebSocket requiere autenticación durante el handshake

Escalabilidad y Producción

Para entornos de producción, el servidor está preparado para:

Logging apropiado según el ambiente (morgan combined)

Manejo de múltiples orígenes CORS

Limitación de tamaño de payloads

Conexiones persistentes WebSocket con keepalive

Manejo de Errores

Aunque no se muestra explícitamente en este módulo, el sistema depende de los manejadores de error de cada módulo individual (rutas, servicios, WebSocket). En producción, podría añadirse un middleware global de manejo de errores.

Este servidor principal actúa como el orquestador central que integra todos los componentes del sistema - APIs REST, WebSocket en tiempo real, y conexión MQTT - en una aplicación cohesiva y lista para producción que soporta los requisitos complejos de una plataforma de monitoreo IoT moderna.