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. La aplicación se construye sobre el módulo http de Node.js, creando un servidor que puede manejar tanto requests HTTP estándar como upgrades a WebSocket.

Configuración de Middlewares Base

La aplicación Express se configura con un conjunto completo de middlewares esenciales para seguridad, logging, y procesamiento de requests. El middleware de seguridad se aplica primero mediante la función security que configura Helmet.js para proteger contra vulnerabilidades web comunes. El logging se maneja con Morgan, configurado para proporcionar output detallado en ambiente de desarrollo y formato combinado más conciso en producción.

El middleware express.json con límite de 1MB permite el parsing de cuerpos de solicitud JSON mientras previene ataques de denegación de servicio por payloads excesivamente grandes. El cookie parser habilita el manejo de cookies, útil para futuras implementaciones de autenticación basada en sesiones. La configuración CORS es dinámica, soportando tanto el wildcard * como una lista de orígenes específicos separados por comas, con credenciales habilitadas para manejar autenticación.

Sistema de Rutas API

El servidor organiza las rutas en módulos especializados que siguen una estructura lógica y mantenible. Las rutas de autenticación se montan bajo /api/auth y manejan los procesos de login con Google OAuth y autenticación de desarrollo. Las rutas de configuración pública en /api proporcionan configuraciones esenciales para el frontend, mientras que las rutas de configuración del sistema en /api/config ofrecen endpoints administrativos con control de acceso basado en roles.

Los datos de temperatura en tiempo real provenientes del sistema MQTT se exponen através de rutas en /api que permiten acceso a datos históricos, estadísticas y el estado actual. Finalmente, las rutas de gestión de datos en /api proporcionan operaciones CRUD para dispositivos y sensores, implementando autenticación en todas las operaciones.

Endpoint de Salud

El endpoint /health implementa un mecanismo simple pero efectivo de verificación de estado del servidor. Este endpoint es crítico para orquestadores de contenedores como Kubernetes y sistemas de monitoreo que necesitan verificar la salud del servicio. Retorna un objeto JSON simple { ok: true } cuando el servidor está operativo correctamente.

Integración WebSocket

El servidor HTTP creado con createServer se pasa al módulo de WebSocket a través de la función initWebSocket. Esta función configura el servidor WebSocket para escuchar en la ruta /ws y manejar eventos de upgrade HTTP a WebSocket. La integración permite que el mismo puerto y misma instancia de servidor maneje tanto tráfico HTTP tradicional como conexiones WebSocket persistentes, compartiendo la misma configuración de seguridad y red.

Inicialización de Servicios

Durante el proceso de arranque, el servidor inicializa servicios críticos del sistema. La conexión MQTT se establece llamando a mqttService.connect(), que inicia la conexión con el broker Mosquitto y comienza a recibir datos de sensores. El servidor WebSocket se inicializa para permitir comunicación bidireccional en tiempo real con los clientes. Todas las rutas Express se montan y configuran según la estructura definida.

Configuración desde Variables de Entorno

El servidor utiliza extensivamente las variables de entorno definidas en el módulo de configuración. El puerto de escucha se obtiene de ENV.PORT, mientras que ENV.NODE_ENV determina el ambiente de ejecución y afecta el comportamiento de middlewares como Morgan. La configuración CORS se deriva de ENV.CORS_ORIGIN, permitiendo flexibilidad en diferentes ambientes de despliegue.

La conexión MQTT se configura completamente desde variables de entorno, incluyendo host del broker, puerto, y la lista de topics a los que suscribirse. Esta aproximación basada en configuración externalizada facilita el despliegue en diferentes ambientes sin modificar el código.

Proceso de Arranque

El proceso de inicialización sigue una secuencia cuidadosamente orquestada. Primero se configura la aplicación Express con todos los middlewares necesarios. Luego se registran todas las rutas API organizadas por funcionalidad y dominio. El servidor HTTP se crea basado en la aplicación Express configurada. El servidor WebSocket se inicializa y configura para manejar conexiones en tiempo real. Se establece la conexión con el broker MQTT para comenzar a recibir datos de sensores. Finalmente, el servidor comienza a escuchar en el puerto configurado.

Logging de Inicialización

Durante el arranque, el servidor proporciona información detallada en consola que es invaluable para verificación y troubleshooting. Se muestra el puerto de escucha HTTP junto con la ruta WebSocket disponible. La configuración del broker MQTT se muestra con host y puerto, y se lista todos los topics a los que el sistema está suscrito. Esta transparencia en la inicialización ayuda a identificar rápidamente problemas de configuración.

Consideraciones de Seguridad

La seguridad se aplica de manera consistente across toda la aplicación. Helmet.js protege contra vulnerabilidades web comunes mediante headers de seguridad apropiados. La configuración CORS restringe los orígenes permitidos según la configuración del ambiente. Las rutas sensibles implementan autenticación JWT individualmente según sus requisitos. El servidor WebSocket requiere autenticación durante el handshake inicial, previniendo conexiones no autorizadas.

Escalabilidad y Consideraciones de Producción

Para ambientes de producción, el servidor está configurado con características que soportan escalabilidad y robustez. El logging se ajusta automáticamente según el ambiente, proporcionando información detallada en desarrollo y logs concisos en producción. El manejo de múltiples orígenes CORS permite integración con frontends distribuidos. La limitación de tamaño de payloads previene ataques de recursos. Las conexiones WebSocket incluyen mecanismos de keepalive para detectar y limpiar conexiones zombi.

Manejo de Errores y Resiliencia

Aunque el manejo explícito de errores globales no se muestra en este módulo, el sistema está diseñado para manejar errores de manera graceful a través de los manejadores específicos en cada módulo. La conexión MQTT incluye lógica de reconexión automática, y el servidor WebSocket implementa monitoreo de salud de conexiones. En una implementación de producción completa, se podría añadir un middleware global de manejo de errores para capturar y loguear excepciones no manejadas.

Este servidor principal actúa como el orquestador central que integra armoniosamente todos los componentes del sistema - APIs REST para operaciones tradicionales, WebSocket para comunicación en tiempo real, y MQTT para ingesta de datos de sensores - creando una plataforma cohesiva y lista para producción que satisface los requisitos complejos de una aplicación moderna de monitoreo IoT.