Módulo de Comunicación MQTT en Tiempo Real

Propósito General

Este módulo implementa un servicio completo de cliente MQTT para conectar la aplicación con un broker Mosquitto, facilitando la recepción, procesamiento y distribución de datos en tiempo real desde dispositivos IoT y sensores. La clase proporciona una abstracción robusta sobre el protocolo MQTT, manejando automáticamente la conexión, reconexión, procesamiento de mensajes y broadcasting a clientes WebSocket.

Arquitectura del Servicio

El servicio está diseñado como una clase singleton que gestiona un único cliente MQTT para toda la aplicación. Esta aproximación asegura que solo exista una conexión al broker MQTT, optimizando el uso de recursos y manteniendo la consistencia en el estado de conexión.

Estado y Propiedades del Servicio

El servicio mantiene un estado comprehensivo que incluye el estado de conexión actual, un buffer circular de datos de temperatura, un registro de suscriptores WebSocket, y contadores de intentos de reconexión. El buffer de datos está limitado a los últimos 100 puntos por defecto, previniendo el consumo excesivo de memoria en ejecuciones prolongadas.

Establecimiento de Conexión

El método connect() inicializa la conexión con el broker MQTT utilizando la configuración proveniente de variables de entorno. Construye la URL del broker y las opciones de cliente dinámicamente, soportando tanto conexiones MQTT standard como MQTT sobre SSL/TLS para el puerto 8883.

La conexión implementa un sistema completo de manejo de eventos que incluye: confirmación de conexión exitosa, detección de errores, manejo de desconexiones, lógica de reconexión automática, y monitoreo del estado offline. El sistema de reconexión incluye un límite configurable de intentos para prevenir bucles infinitos en caso de problemas persistentes del broker.

Configuración de Cliente MQTT

La configuración del cliente incluye opciones optimizadas para entornos de producción: clientId único generado dinámicamente, sesiones limpias (clean), periodo de reconexión de 5 segundos, timeout de conexión de 30 segundos, y keepalive de 60

segundos. El nivel de calidad de servicio (QoS) se establece en 1 para garantizar la entrega al menos una vez de los mensajes.

El sistema soporta autenticación con username y password cuando están configurados en las variables de entorno. Para conexiones seguras en el puerto 8883, se configura el manejo de certificados SSL, con una opción de desarrollo que permite certificados self-signed.

Suscripción a Topics

El servicio se suscribe automáticamente a todos los topics especificados en la variable de entorno MQTT_TOPICS al establecer la conexión. Cada suscripción utiliza QoS 1 y maneja errores individualmente por topic. El sistema registra tanto las suscripciones exitosas como los fallos para facilitar el diagnóstico.

Procesamiento de Mensajes

El método handleMessage() procesa todos los mensajes entrantes de los topics suscritos. Implementa un sistema robusto de parsing que intenta primero interpretar el mensaje como JSON, luego como texto plano, y finalmente como valor numérico simple. Esta flexibilidad permite integrar con diversos tipos de dispositivos y formatos de datos.

Para los datos identificados como mediciones de temperatura, el servicio extrae y normaliza la información en un formato estandarizado que incluye timestamp, valor de temperatura, humedad (si está disponible), identificador del sensor, y los datos originales. Todos los puntos de temperatura se almacenan en el buffer circular y se distribuyen inmediatamente a los suscriptores WebSocket.

Sistema de Broadcasting en Tiempo Real

El servicio mantiene un registro de clientes WebSocket suscritos para broadcasting en tiempo real. Cuando llega un nuevo dato de temperatura, se serializa y envía inmediatamente a todos los suscriptores conectados. El sistema maneja automáticamente la limpieza de suscriptores desconectados y previene errores por envíos a conexiones cerradas.

API de Consulta de Datos

El servicio proporciona varios métodos para acceder a los datos almacenados: getTemperatureData() para obtener un conjunto histórico limitado, getLatestTemperature() para la lectura más reciente, y getStats() para cálculos estadísticos como promedio, mínimo y máximo. Estos métodos son utilizados por los controladores REST para servir datos a los clientes HTTP.

Monitoreo y Diagnóstico

El método getConnectionInfo() proporciona un resumen completo del estado del servicio, incluyendo estado de conexión, URL del broker, topics suscritos, número de suscriptores WebSocket, cantidad de datos almacenados, e intentos de reconexión. Esta información es valiosa para dashboards administrativos y diagnóstico de problemas.

Gestión de Ciclo de Vida

El servicio incluye métodos para una desconexión controlada (disconnect()) y para publicación de mensajes (publish()), aunque la funcionalidad principal es la recepción de datos. La publicación puede utilizarse para testing o para enviar comandos a dispositivos.

Manejo de Errores y Robustez

Cada operación incluye manejo de errores comprehensivo con logging descriptivo. El servicio es tolerante a mensajes malformados, desconexiones de red, y problemas del broker, recuperándose automáticamente cuando es posible.

Integración con el Ecosistema

El servicio se integra con el sistema de variables de entorno para toda su configuración, con el sistema de WebSocket para broadcasting en tiempo real, y con los controladores REST para proporcionar acceso histórico a los datos. Su diseño modular permite fácil extensión para soportar nuevos tipos de datos o formatos de mensaje.

Esta implementación proporciona una base sólida y production-ready para aplicaciones IoT que requieren recepción y distribución eficiente de datos de sensores en tiempo real, balanceando rendimiento, robustez y facilidad de uso.