Aplicación Web IoT - Documentación Técnica

Descripción General

Esta es una aplicación web IoT full-stack desarrollada con backend Node.js/Express y frontend vanilla JavaScript, diseñada para el monitoreo de temperatura en tiempo real y visualización de datos a través de comunicación MQTT, con fin de tener un acceso frontend personalizado y adaptable a las necesidades presentadas por el proycto intertecnicatura

Índice

Arquitectura

Arquitectura del Backend

```
backend/
├── src/
│   ├── config/           # Gestión de configuración
│   │   ├── env.js        # Validación de variables de entorno
│   │   └── security.js   # Configuración de seguridad Helmet
│   ├── controllers/      # Manejadores de solicitudes
│   │   ├── auth.controllers.js      # Autenticación Google OAuth
│   │   ├── config.controllers.js    # Gestión de configuración del sistema
│   │   ├── data.controllers.js       # Operaciones de datos (placeholder)
│   │   └── temperature.controllers.js # Endpoints de datos de temperatura
│   ├── middlewares/     # Middlewares de Express
│   │   ├── auth.middlewares.js       # Autenticación JWT y acceso basado en roles
│   │   └── data.middlewares.js       # Middlewares de procesamiento de datos
│   ├── routes/          # Definiciones de rutas API
│   │   ├── auth.routes.js            # Rutas de autenticación
│   │   ├── config.routes.js          # Configuración pública
│   │   ├── config.system.routes.js  # Configuración del sistema (protegida)
│   │   ├── data.routes.js            # Rutas de datos (placeholder)
│   │   └── temperature.routes.js      # Rutas de datos de temperatura
│   ├── service/         # Servicios de lógica de negocio
│   │   ├── data.service.js           # Operaciones de base de datos
│   │   ├── jwt.service.js            # Gestión de tokens JWT
│   │   ├── mqtt.service.js           # Comunicación con broker MQTT
│   │   └── user.service.js           # Resolución de roles de usuario
│   ├── sw/              # Implementación WebSocket
│   │   ├── index.js                  # Servidor WebSocket con autenticación JWT
│   │   ├── handlers.js               # Manejadores de mensajes WebSocket
│   │   └── uWebSockets.js            # Implementación alternativa WebSocket
```

```
│   ├── db/              # Configuración de base de datos
│   │   └── index.js                # Pool de conexiones MariaDB
│   └── server.js        # Punto de entrada principal del servidor
└── package.json
```

Arquitectura del Frontend

```
frontend/public/
├── src/
│   ├── api.js                       # Cliente API con autenticación
│   ├── app.js                       # Inicialización principal de la aplicación
│   ├── components/                  # Componentes UI reutilizables
│   │   ├── chartWidget.js          # Componente de gráfico genérico
│   │   ├── footer.js               # Componente de pie de página
│   │   ├── navbar.js               # Barra de navegación con menús basados en
roles
│   │   └── temperatureChart.js     # Gráfico específico de temperatura
│   ├── pages/                       # Componentes de página
│   │   ├── configuracion.js        # Página de configuración general
│   │   ├── configuracionAvanzada.js # Configuración avanzada (solo admin)
│   │   ├── dashboard.js            # Dashboard principal
│   │   ├── login.js                # Login Google OAuth
│   │   ├── notFound.js             # Página de error 404
│   │   └── sobreNosotros.js        # Página sobre nosotros
│   ├── router/                      # Enrutamiento del lado del cliente
│   │   └── index.js                # Enrutamiento basado en hash con protección
de roles
│   ├── state/                       # Gestión de estado
│   │   └── store.js                # Store de estado reactivo simple
│   ├── utils/                       # Funciones utilitarias
│   │   ├── configService.js        # Servicio de gestión de configuración
│   │   ├── dom.js                  # Ayudantes de manipulación DOM
│   │   └── storage.js              # Wrapper de almacenamiento local/sesión
│   ├── ws.js                        # Cliente WebSocket
│   └── loader.js                   # Cargador de aplicación
├── style.css                       # Estilos globales
├── sw.js                           # Service worker
└── config.json                     # Configuración del frontend
```

Stack Tecnológico

Tecnologías del Backend

- **Node.js** - Entorno de ejecución
- **Express.js** - Framework web
- **JWT (jose)** - Autenticación basada en tokens
- **MQTT** - Protocolo de comunicación IoT
- **WebSocket (ws)** - Comunicación en tiempo real
- **MariaDB** - Base de datos relacional
- **Joi** - Validación de datos
- **Helmet** - Middleware de seguridad
- **CORS** - Intercambio de recursos de origen cruzado

- **Morgan** - Registrador de solicitudes HTTP

Tecnologías del Frontend

- **Vanilla JavaScript (ES6+)** - Sin dependencias de framework
- **Canvas API** - Renderizado de gráficos
- **WebSocket API** - Datos en tiempo real
- **Google Identity Services** - Autenticación OAuth
- **Service Worker** - Capacidades offline
- **CSS3** - Estilos con CSS Grid y Flexbox

Características Principales

Autenticación y Autorización

Integración con Google OAuth

- Utiliza Google Identity Services para autenticación
- Valida tokens ID contra JWKS de Google
- Emite tokens JWT personalizados con roles embebidos

Control de Acceso Basado en Roles (RBAC)

```
// Jerarquía de roles
- admin: Acceso completo al sistema
- action: Control de dispositivos y operaciones
- readonly: Acceso solo de lectura
```

Gestión de Tokens JWT

- Tokens de acceso con expiración de 15 minutos
- Soporte para tokens de renovación (expiración de 7 días)
- Middleware de validación automática de tokens

Integración MQTT

Gestión de Conexión

- Reconexión automática con retroceso exponencial
- Configuración de broker configurable (host, puerto, credenciales)
- Gestión de suscripciones a tópicos
- Nivel QoS 1 para entrega confiable de mensajes

Procesamiento de Datos

- Ingesta de datos de temperatura en tiempo real
- Validación y análisis automático de datos
- Buffer circular para almacenamiento de datos (máximo 100 puntos)

- Transmisión WebSocket a clientes conectados

Formatos de Datos Soportados

(sujeto a cambios una vez definamos el formato que envia el dispositivo)

```
// Formato JSON
{
  "temperature": 23.5,
  "humidity": 65.2,
  "sensor_id": "temp_001"
}

// Formato numérico simple
"23.5"
```

Comunicación WebSocket

Transmisión de Datos en Tiempo Real

- Conexiones WebSocket autenticadas con JWT
- Sistema de suscripción basado en tópicos
- Heartbeat/ping-pong automático para salud de conexión
- Transmisión de datos MQTT a clientes suscritos

Protocolo de Mensajes

```
// Cliente a Servidor
{
  "type": "sub",
  "topic": "temperature"
}

// Servidor a Cliente
{
  "type": "temperature_update",
  "data": { /* datos de temperatura */ },
  "timestamp": "2024-01-01T00:00:00.000Z"
}
```

Esquema de Base de Datos

Tablas Principales

- **usuarios** - Cuentas de usuario vinculadas a Google
- **roles** - Roles del sistema (admin, operador, visualizador)
- **permisos** - Permisos granulares
- **dispositivos** - Registro de dispositivos IoT
- **datos_sensores** - Almacenamiento de datos de sensores
- **proyectos** - Gestión de proyectos
- **configuraciones_sistema** - Configuraciones del sistema

Relaciones Clave

- Muchos a muchos: usuarios ↔ roles ↔ permisos
- Uno a muchos: dispositivos → datos_sensores
- Muchos a muchos: usuarios ↔ proyectos

Endpoints de API

Autenticación

```
POST /api/auth/google
Body: { "credential": "google_id_token" }
Response: { "user", "role", "accessToken", "tokenType", "expiresIn" }
```

Configuración

```
GET  /api/config                    # Configuración pública
GET  /api/config/general            # Configuración general (autenticado)
GET  /api/config/advanced           # Configuración avanzada (solo admin)
PUT  /api/config/advanced           # Actualizar configuración avanzada (solo
admin)
GET  /api/config/mqtt/status        # Estado de conexión MQTT
POST /api/config/mqtt/restart       # Reiniciar MQTT (solo admin)
POST /api/config/cache/clear        # Limpiar cache de datos (solo admin)
```

Datos de Temperatura

```
GET /api/temperature                # Datos históricos de temperatura
GET /api/temperature/stats          # Estadísticas de temperatura
GET /api/temperature/latest         # Última lectura de temperatura
GET /api/mqtt/status                # Información de conexión MQTT
```

WebSocket

```
WS /ws?token=<jwt_token>
```

Gestión de Configuración

Variables de Entorno

Variables Requeridas

```
JWT_SECRET=<cadena_hex_64_caracteres>     # Secreto de firma JWT
GOOGLE_CLIENT_ID=<google_client_id>       # ID de cliente Google OAuth
```

Variables Opcionales

```
NODE_ENV=development                       # Modo de entorno
PORT=3000                                  # Puerto del servidor
CORS_ORIGIN=*                              # Orígenes CORS permitidos
MQTT_BROKER_HOST=localhost                 # Host del broker MQTT
MQTT_BROKER_PORT=1883                      # Puerto del broker MQTT
MQTT_BROKER_USERNAME=                      # Usuario MQTT (opcional)
MQTT_BROKER_PASSWORD=                      # Contraseña MQTT (opcional)
MQTT_TOPICS=vittoriodurigutti/prueba,vittoriodurigutti/temperature
ADMIN_WHITELIST=admin@ejemplo.com          # Lista de emails de admin
ACTION_WHITELIST=operador@ejemplo.com      # Lista de emails de operador
```

Configuración de Base de Datos

```
DB_HOST=localhost
DB_USER=tst_da_user
DB_PASS=tst_da_password_2024
DB_NAME=TST-DA
```

Configuración de Desarrollo

Prerrequisitos

- Node.js 18+
- MariaDB 10.6+
- Broker MQTT (Mosquitto)
- Proyecto Google Cloud Console con credenciales OAuth

Instalación

1. **Clonar repositorio**

```
git clone https://github.com/ISPC-PI-II-2024/DdA-IoT-Web-App.git
cd C-Prototipo
```

2. **Configuración del backend**

```
cd backend
npm install
cp .env.example .env
# Editar .env con tu configuración
```

3. **Configuración de base de datos**

```
# Iniciar contenedor MariaDB
docker-compose up -d mariadb

# La base de datos se inicializará automáticamente con init/01-init.sql
```

4. **Configuración del broker MQTT**

```
# Iniciar contenedor Mosquitto
docker-compose up -d mosquitto
```

5. **Generar secreto JWT**

```
node -e "console.log(require('crypto').randomBytes(64).toString('hex'))"
```

6. **Iniciar servidor de desarrollo**

```
npm start
```

Desarrollo del Frontend

El frontend se sirve de forma estática y no requiere proceso de construcción. Simplemente sirve el directorio `frontend/public` con cualquier servidor HTTP.

```
# Usando Python
cd frontend/public
python -m http.server 8080

# Usando Node.js
npx serve frontend/public -p 8080
```

Consideraciones de Seguridad

Seguridad de Autenticación

- Los tokens JWT usan algoritmo HS256 con secreto de 64 caracteres
- Los tokens tienen expiración corta (15 minutos)
- Los tokens ID de Google se validan contra JWKS de Google
- Las conexiones WebSocket requieren tokens JWT válidos

Seguridad de Datos

- Middleware Helmet para headers de seguridad
- Configuración CORS para solicitudes de origen cruzado
- Validación de entrada con esquemas Joi
- Prevención de inyección SQL con consultas parametrizadas

Seguridad de Red

- MQTT soporta autenticación usuario/contraseña
- Las conexiones WebSocket usan autenticación JWT
- HTTPS recomendado para producción (configurar en proxy inverso)

Optimizaciones de Rendimiento

Optimizaciones del Backend

- Pool de conexiones para conexiones de base de datos
- Buffer circular para datos de temperatura (previene pérdidas de memoria)
- Gestión de conexiones WebSocket con limpieza automática
- Nivel QoS 1 MQTT para entrega confiable de mensajes

Optimizaciones del Frontend

- Renderizado de gráficos basado en Canvas para rendimiento
- Service worker para capacidades offline
- Manipulación DOM eficiente con utilidades personalizadas
- Carga diferida de componentes de página

Monitoreo y Registro

Registros de Aplicación

- Registro de solicitudes HTTP Morgan
- Registro de estado de conexión MQTT
- Seguimiento de conexiones WebSocket
- Registro de errores con stack traces

Verificaciones de Salud

```
GET /health
Response: { "ok": true }
```

Métricas Disponibles

- Estado de conexión MQTT
- Cantidad de suscriptores WebSocket
- Cantidad de datos de temperatura
- Estado del pool de conexiones de base de datos

Solución de Problemas

Problemas Comunes

1. **Falló la Conexión MQTT**
    - Verificar configuración de host/puerto del broker
    - Verificar conectividad de red
    - Verificar credenciales de autenticación
2. **Token JWT Inválido**
    - Verificar que JWT_SECRET esté configurado correctamente
    - Verificar expiración del token
    - Asegurar que el Google Client ID sea correcto
3. **Error de Conexión a Base de Datos**
    - Verificar credenciales de base de datos
    - Verificar estado del servicio MariaDB
    - Asegurar que la base de datos existe
4. **Falló la Conexión WebSocket**
    - Verificar validez del token JWT
    - Verificar configuración de URL WebSocket
    - Verificar configuración del firewall

Modo Debug

Configurar `NODE_ENV=development` para registro detallado y mensajes de error.

Contribución

Estilo de Código

- Usar características ES6+
- Seguir patrones async/await
- Implementar manejo apropiado de errores

- Agregar comentarios JSDoc para funciones
- Usar nombres de variables significativos

Pruebas

- Pruebas unitarias para funciones de servicio
- Pruebas de integración para endpoints API
- Pruebas de comunicación WebSocket
- Pruebas de procesamiento de datos MQTT

Licencia