Índice de Documentación

Paradigmas y Lenguajes de Programación 2025

Trabajo Final

Autores: Küster Joaquín • Da Silva Marcos • Martinez Lázaro Ezequiel

Guía de Usuario

Introducción

- ¿Qué es este sistema?
- Características principales
- Arquitectura de alto nivel
- Stack tecnológico
- Flujo de datos completo
- Casos de uso

Arquitectura del Sistema

- Vista general
- Componentes principales
  - Mosquitto MQTT Broker
  - Go Publisher
  - Go Subscriber
  - WebSocket Server
  - Firebase Realtime Database
  - MPI Backend
  - Dashboard Frontend
- Patrones de diseño
- Flujo de ejecución completo

Instalación

- Requisitos previos
- Instalación de dependencias
  - Instalar Go
  - Instalar Node.js
  - Instalar Mosquitto
- Configuración del proyecto
  - Configurar Firebase
  - Ejecutar script de instalación
  - Inicializar base de datos
- Configuración avanzada
- Solución de problemas

Ejecución del Sistema

- Script de control
- Iniciar el sistema
  - Inicio completo
  - Inicio manual
- Verificar estado
- Acceder al dashboard
- Detener el sistema
- Monitoreo de logs
- Casos de uso comunes

Referencia de API

WebSocket API

- Servidor WebSocket
- Endpoints disponibles
  - /ws/resumenes - Resúmenes de consumo
  - /ws/avisos - Alertas y notificaciones
  - /ws/dispositivos - Control de dispositivos
  - /ws/params - Configuración del sistema
  - /ws/oficinas - Gestión de oficinas

- Ejemplo completo de cliente
- Manejo de errores

MQTT API

- Broker MQTT
- Estructura de tópicos
- Formato de mensajes
- Publisher (Go)
- Subscriber (Go)
- Flujo de mensajes
- Ejemplos de clientes
  - Cliente Python
  - Cliente JavaScript
- Monitoreo de tópicos
- Configuración de Mosquitto
- Seguridad

🔧 Información Técnica

Tecnologías Utilizadas

Backend

- **Go 1.19+** - Publisher y Subscriber MQTT
- **Node.js 14+** - WebSocket y HTTP servers
- **Mosquitto 2.0+** - MQTT Broker
- **Firebase** - Realtime Database
- **C + OpenMPI** - Procesamiento paralelo

Frontend

- **HTML5/CSS3/JavaScript** - Dashboard
- **Chart.js** - Visualizaciones
- **WebSocket API** - Comunicación en tiempo real

Documentación

- **VitePress** - Framework de documentación
- **Vue 3** - Componentes interactivos

- **Mermaid.js** - Diagramas
- **Markdown** - Contenido

Estructura de Datos

Firebase Schema

```
monitoreo_consumo/
├── configuracion/
├── tipos_avisos/
└── oficinas/
    ├── {id}/
    │   ├── avisos/
    │   ├── resumenes/
    │   └── estados_dispositivos/
```

Mensaje MQTT

json

```
{
  "oficina": "A",
  "timestamp": 1701648000,
  "presencia": true,
  "corriente_a": 12.5,
  "temperatura": 24.3
}
```

Diagramas del Sistema

Arquitectura General

mermaid

```
graph TB
    Sensores --> MQTT[Mosquitto]
    MQTT --> Publisher[Go Publisher]
    MQTT --> Subscriber[Go Subscriber]
    Subscriber --> Firebase
    Subscriber --> WebSocket
    WebSocket --> Dashboard
    Firebase --> MPI[MPI Analysis]
```

Flujo de Datos

1. **Generación**: Publisher simula datos cada 10s
2. **Publicación**: Datos → MQTT
3. **Procesamiento**: Subscriber detecta alertas
4. **Persistencia**: Datos → Firebase
5. **Distribución**: WebSocket → Dashboard
6. **Visualización**: Gráficos en tiempo real

🎓 Información Académica

Materia: Paradigmas y Lenguajes de Programación
Año: 2025
Tipo: Trabajo Final

Integrantes:

- Küster Joaquín
- Da Silva Marcos
- Martinez Lázaro Ezequiel

Institución: Universidad Nacional de Misiones (UNAM) **Facultad**: Facultad de Ciencias Exactas Quimicas y Naturales

Paradigmas Implementados

1. **Programación Imperativa** - Go (Publisher/Subscriber)
2. **Programación Orientada a Eventos** - Node.js (WebSocket)
3. **Programación Funcional** - JavaScript (Dashboard)
4. **Programación Paralela** - MPI (Análisis)
5. **Programación Reactiva** - WebSocket (Tiempo real)

Patrones de Diseño

- **Publisher-Subscriber** (MQTT)
- **Observer** (WebSocket)
- **Singleton** (Firebase client)
- **Factory** (Creación de sensores)
- **Strategy** (Detección de alertas)

Contacto y Soporte

Para preguntas sobre este proyecto:

- Consultar la documentación completa
- Revisar la sección de solución de problemas
- Verificar los logs del sistema

Notas de Versión

Versión 1.0.0 - Diciembre 2024

- Sistema completo de monitoreo en tiempo real
- Arquitectura MQTT + WebSocket
- Dashboard interactivo
- Procesamiento paralelo MPI
- Documentación completa con VitePress
- 13 tipos de alertas automáticas
- Gestión dinámica de oficinas
- Configuración en caliente

Sistema de Monitoreo de Consumo Eléctrico
Paradigmas y Lenguajes de Programación 2025
Küster Joaquín • Da Silva Marcos • Martinez Lázaro Ezequiel

API MQTT

El sistema utiliza **MQTT** (Message Queuing Telemetry Transport) para la comunicación entre sensores simulados y el procesador de eventos.

Broker MQTT

Implementación: Mosquitto
Puerto: 1883
Protocolo: MQTT v3.1.1
Autenticación: Anónima (desarrollo)

Estructura de Tópicos

Patrón de Tópicos

```
oficinas/{id}/sensores
```

Ejemplos:

- `oficinas/A/sensores`
- `oficinas/B/sensores`
- `oficinas/C/sensores`

Wildcard Subscriptions

El Subscriber usa wildcards para escuchar todas las oficinas:

```
oficinas/+/sensores
```

Donde `+` coincide con cualquier ID de oficina.

Formato de Mensajes

Mensaje de Sensor

```json
{
  "oficina": "A",
  "timestamp": 1701648000,
  "presencia": true,
  "corriente_a": 12.5,
  "temperatura": 24.3
}
```

Estructura de Datos

Campo	Tipo	Descripción	Rango
`oficina`	string	ID de la oficina	A-Z, 0-9
`timestamp`	number	Unix timestamp (segundos)	-
`presencia`	boolean	Detección de presencia	true/false
`corriente_a`	number	Corriente eléctrica (Amperes)	0.0 - 50.0
`temperatura`	number	Temperatura ambiente (°C)	15.0 - 35.0

Publisher (Go)

Configuración

```go
opciones := mqtt.NewClientOptions().
    AddBroker("tcp://localhost:1883").
    SetClientID("publicador-sensores")

clienteMQTT := mqtt.NewClient(opciones)
if token := clienteMQTT.Connect(); token.Wait() && token.Error() != nil {
    panic(token.Error())
}
```

Publicar Mensaje

```go
func SimularYPublicar(cliente mqtt.Client, oficina string) {
    ahora := time.Now()
    timestamp := ahora.Unix()

    presencia := DetectarPresencia(ahora)
    temperatura := CalcularSiguienteTemperatura(ultimaTemperatura[oficina])
    corriente := 0.0

    if presencia {
        corriente = CalcularCorriente(oficina, presencia, temperatura)
    }

    datos := DatosSensor{
        Oficina:     oficina,
        TiempoUnix:  timestamp,
        Presencia:   presencia,
        CorrienteA:  corriente,
        Temperatura: temperatura,
    }

    payload, _ := json.Marshal(datos)
    topico := fmt.Sprintf("oficinas/%s/sensores", oficina)

    token := cliente.Publish(topico, 0, false, payload)
    token.Wait()
}
```

QoS (Quality of Service)

El Publisher usa **QoS 0** (At most once):

- No confirmación de entrega
- Menor overhead
- Adecuado para datos en tiempo real donde la pérdida ocasional es aceptable

Frecuencia de Publicación

```go
temporizador := time.NewTicker(10 * time.Second)
defer temporizador.Stop()

for range temporizador.C {
    for _, oficina := range oficinas {
        SimularYPublicar(clienteMQTT, oficina)
    }
}
```

Intervalo: 10 segundos por oficina

Subscriber (Go)

Configuración

```go
opciones := mqtt.NewClientOptions().
    AddBroker("tcp://localhost:1883").
    SetClientID("subscriptor-edge")

clienteMQTT := mqtt.NewClient(opciones)
if token := clienteMQTT.Connect(); token.Wait() && token.Error() != nil {
    panic(token.Error())
}
```

Suscribirse a Tópicos

```go
topic := "oficinas/+/sensores"

clienteMQTT.Subscribe(topic, 0, func(_ mqtt.Client, msg mqtt.Message) {
    var datos DatosSensor
    if err := json.Unmarshal(msg.Payload(), &datos); err != nil {
        return
    }

    // Procesar datos
    estado := obtenerEstado(datos.Oficina)
    estado.Mutex.Lock()
    defer estado.Mutex.Unlock()

    // Acumular datos
    estado.Corrientes = append(estado.Corrientes, datos.CorrienteA)
    estado.Temperaturas = append(estado.Temperaturas, datos.Temperatura)

    if datos.Presencia {
        estado.TiempoPresente += 10
    }

    // Detectar alertas
    avisos := detectarAvisos(datos, estado)
    for _, av := range avisos {
        guardarAviso(ctx, datos.Oficina, av)
    }

```

```
    // Generar resumen cada 60 segundos
    if ahora - estado.UltimoResumen >= 60 {
        resumen := generarResumen(ahora, estado)
        guardarResumen(ctx, datos.Oficina, resumen)

        // Resetear acumuladores
        estado.Corrientes = nil
        estado.Temperaturas = nil
        estado.TiempoPresente = 0
    }
})
```

Flujo de Mensajes

mermaid

```
sequenceDiagram
    participant P as Publisher
    participant M as Mosquitto
    participant S as Subscriber
    participant F as Firebase

    loop Cada 10 segundos
        P->>P: Generar datos sensor
        P->>M: PUBLISH oficinas/A/sensores
        M->>S: DELIVER mensaje
        S->>S: Parsear JSON
        S->>S: Acumular datos
        S->>S: Detectar alertas

        alt Hay alertas
            S->>F: Guardar avisos
        end

        alt Pasaron 60 segundos
            S->>S: Generar resumen
            S->>F: Guardar resumen
            S->>S: Resetear acumuladores
        end
    end
```

Ejemplo de Cliente Python

```python
import paho.mqtt.client as mqtt
import json
import time

def on_connect(client, userdata, flags, rc):
    print(f"Conectado con código: {rc}")
    client.subscribe("oficinas/+/sensores")

def on_message(client, userdata, msg):
    datos = json.loads(msg.payload.decode())
    print(f"Oficina {datos['oficina']}: {datos['corriente_a']}A, {datos['temperatu

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect("localhost", 1883, 60)
client.loop_forever()
```

Ejemplo de Cliente JavaScript (Node.js)

```javascript
const mqtt = require('mqtt');

const client = mqtt.connect('mqtt://localhost:1883', {
    clientId: 'cliente-js-' + Math.random().toString(16).substr(2, 8)
});

client.on('connect', () => {
    console.log('Conectado a MQTT');
    client.subscribe('oficinas/+/sensores', (err) => {
        if (!err) {
            console.log('Suscrito a sensores');
        }
    });
});

client.on('message', (topic, message) => {
    const datos = JSON.parse(message.toString());
    console.log(`${topic}:`, datos);
});
```

Publicar Mensaje de Prueba

Usando mosquitto_pub

```bash
mosquitto_pub -h localhost -t "oficinas/TEST/sensores" -m '{
  "oficina": "TEST",
  "timestamp": 1701648000,
  "presencia": true,
  "corriente_a": 15.5,
  "temperatura": 23.8
}'
```

Usando MQTT.fx (GUI)

1. Conectar a `localhost:1883`
2. Ir a pestaña "Publish"
3. Topic: `oficinas/TEST/sensores`
4. Payload: JSON del mensaje
5. Click "Publish"

Monitoreo de Tópicos

Suscribirse a Todos los Mensajes

```bash
mosquitto_sub -h localhost -t "oficinas/#" -v
```

Salida:

```
oficinas/A/sensores {"oficina":"A","timestamp":1701648000,...}
oficinas/B/sensores {"oficina":"B","timestamp":1701648010,...}
oficinas/C/sensores {"oficina":"C","timestamp":1701648020,...}
```

Filtrar por Oficina Específica

```bash
mosquitto_sub -h localhost -t "oficinas/A/sensores" -v
```

Configuración de Mosquitto

Archivo de Configuración

`config/mosquitto.conf` :

```conf
# Puerto de escucha
listener 1883

# Permitir conexiones anónimas (solo desarrollo)
allow_anonymous true

# Logs
log_dest file logs/mosquitto.log
log_type all

# Persistencia
persistence true
persistence_location data/mosquitto/

# Límites
max_connections 100
max_queued_messages 1000
max_packet_size 104857600
```

Iniciar Mosquitto

```bash
mosquitto -c config/mosquitto.conf -v
```

Seguridad (Producción)

Para producción, configura autenticación:

Crear Archivo de Contraseñas

```bash
mosquitto_passwd -c config/passwd usuario1
# Ingresa contraseña cuando se solicite
```

Actualizar Configuración

```conf
listener 1883
allow_anonymous false

```

```
password_file config/passwd
```

Conectar con Autenticación

go

```go
opciones := mqtt.NewClientOptions().
    AddBroker("tcp://localhost:1883").
    SetClientID("publicador-sensores").
    SetUsername("usuario1").
    SetPassword("contraseña")
```

Troubleshooting

Verificar Conexión

bash

```bash
# Ping al broker
mosquitto_pub -h localhost -t "test" -m "ping"
mosquitto_sub -h localhost -t "test"
```

Ver Mensajes en Tiempo Real

bash

```bash
# Terminal 1: Suscribirse
mosquitto_sub -h localhost -t "oficinas/#" -v

# Terminal 2: Publicar
mosquitto_pub -h localhost -t "oficinas/A/sensores" -m '{"test":true}'
```

Logs del Broker

bash

```bash
tail -f logs/mosquitto.log
```

Consejo

Usa herramientas como **MQTT Explorer** o **MQTT.fx** para visualizar y depurar mensajes MQTT de forma gráfica.

API WebSocket

El sistema utiliza **WebSocket** para comunicación bidireccional en tiempo real entre el servidor y los clientes (dashboard).

Servidor WebSocket

URL Base: `ws://localhost:8081`

Puerto: 8081

Endpoints Disponibles

1. `/ws/resumenes`

Transmite resúmenes de consumo de todas las oficinas.

Conexión

javascript

```
const ws = new WebSocket('ws://localhost:8081/ws/resumenes');

ws.onopen = () => {
    console.log('Conectado a resumenes');
};

ws.onmessage = (event) => {
    const data = JSON.parse(event.data);
    console.log('Resumen recibido:', data);
};
```

Mensaje Recibido

json

```
{
  "tipo": "resumenes",
  "data": {
```

```
    "A": {
      "timestamp": 1701648000,
      "corriente_a": 12.5,
      "consumo_kvh": 2.75,
      "consumo_total_kvh": 45.6,
      "min_temp": 22.5,
      "max_temp": 25.8,
      "tiempo_presente": 300,
      "monto_estimado": 0.69,
      "monto_total": 11.4
    },
    "B": {
      "timestamp": 1701648000,
      "corriente_a": 8.2,
      "consumo_kvh": 1.80,
      "consumo_total_kvh": 33.6,
      "min_temp": 23.1,
      "max_temp": 26.2,
      "tiempo_presente": 240,
      "monto_estimado": 0.45,
      "monto_total": 8.4
    }
  }
}
```

Estructura de Datos

Campo	Tipo	Descripción
`timestamp`	number	Unix timestamp del resumen
`corriente_a`	number	Corriente promedio (Amperes)
`consumo_kvh`	number	Consumo del período actual (kWh)
`consumo_total_kvh`	number	Consumo total acumulado (kWh)
`min_temp`	number	Temperatura mínima (°C)
`max_temp`	number	Temperatura máxima (°C)
`tiempo_presente`	number	Tiempo con presencia (segundos)
`monto_estimado`	number	Costo del período actual
`monto_total`	number	Costo total acumulado

Frecuencia

- **Inicial**: Al conectar, envía datos actuales

- **Actualizaciones**: Cada 10 segundos

---

2. `/ws/avisos`

Transmite alertas y notificaciones del sistema.

Conexión

javascript

```javascript
const ws = new WebSocket('ws://localhost:8081/ws/avisos');

ws.onmessage = (event) => {
    const data = JSON.parse(event.data);
    data.data.forEach(aviso => {
        mostrarAlerta(aviso);
    });
};
```

Mensaje Recibido

json

```json
{
  "tipo": "avisos",
  "data": [
    {
      "timestamp": 1701648000,
      "id_tipo": "1",
      "adicional": "Oficina A - Luces encendidas por detección de presencia"
    },
    {
      "timestamp": 1701648060,
      "id_tipo": "4",
      "adicional": "Oficina C - Aire acondicionado activado por temperatura elevad
    }
  ]
}
```

Tipos de Avisos

ID	Motivo	Detalle	Impacto
0	Luces apagadas	Estado desactivado	2
1	Luces encendidas	Detección de presencia	1
2	Luces apagadas	Ausencia detectada	2
3	Aire apagado	Estado desactivado	2

ID	Motivo	Detalle	Impacto
4	Aire encendido	Temperatura elevada con presencia	3
5	Aire apagado	Condiciones óptimas	2
6	Consumo anómalo	Corriente alta sin presencia	3
7	Corte de energía	Corriente en 0 por corte	3
8	Sensor no responde	Sin datos por >60s	3
9	Alerta de corriente	Consumo > umbral	3
10	Oficina agregada	Nueva oficina	1
11	Oficina eliminada	Oficina removida	1
12	Config modificada	Parámetros cambiados	1

---

3. `/ws/dispositivos`

Gestiona el estado de dispositivos (luces, aire acondicionado).

Conexión

javascript

```javascript
const ws = new WebSocket('ws://localhost:8081/ws/dispositivos');

// Recibir estado
ws.onmessage = (event) => {
    const data = JSON.parse(event.data);
    actualizarUI(data.data);
};
```

Mensaje Recibido

json

```json
{
  "tipo": "dispositivos",
  "data": {
    "A": { "aire": true, "luces": true },
    "B": { "aire": false, "luces": true },
    "C": { "aire": true, "luces": false }
  }
}
```

Enviar Actualización

```javascript
// Cambiar estado de un dispositivo
ws.send(JSON.stringify({
    tipo: 'actualizar_dispositivo',
    oficina: 'A',
    dispositivo: 'luces',  // 'luces' o 'aire'
    estado: false
}));
```

Flujo de Actualización

```mermaid
sequenceDiagram
    participant D as Dashboard
    participant WS as WebSocket Server
    participant P as Publisher
    participant S as Subscriber

    D->>WS: Actualizar dispositivo
    WS->>P: Broadcast cambio
    WS->>S: Broadcast cambio
    WS->>D: Confirmar actualización

    Note over P: Ajusta cálculo de corriente
    Note over S: Detecta alertas de cambio
```

---

4. `/ws/params`

Gestiona parámetros de configuración del sistema.

Conexión

```javascript
const ws = new WebSocket('ws://localhost:8081/ws/params');

// Recibir configuración
ws.onmessage = (event) => {
    const data = JSON.parse(event.data);
    cargarConfiguracion(data.data);
};
```

Mensaje Recibido

```json
{
  "tipo": "params",
  "data": {

```

```
    "hora_inicio": 8.0,
    "hora_fin": 20.0,
    "umbral_temperatura_ac": 25.0,
    "umbral_corriente": 21.5,
    "voltaje": 220.0,
    "costo_kwh": 0.25
  }
}
```

Enviar Actualización

```javascript
ws.send(JSON.stringify({
    tipo: 'actualizar_params',
    data: {
        hora_inicio: 9.0,
        hora_fin: 18.0,
        umbral_temperatura_ac: 24.0,
        umbral_corriente: 20.0,
        voltaje: 220.0,
        costo_kwh: 0.30
    }
}));
```

Parámetros

Campo	Tipo	Descripción	Rango
`hora_inicio`	number	Hora de inicio (formato 24h)	0.0 - 23.99
`hora_fin`	number	Hora de fin (formato 24h)	0.0 - 23.99
`umbral_temperatura_ac`	number	Temperatura para activar AC (°C)	15.0 - 35.0
`umbral_corriente`	number	Umbral de alerta (Amperes)	5.0 - 50.0
`voltaje`	number	Voltaje de red (V)	110 / 220
`costo_kwh`	number	Costo por kWh	0.01 - 10.0

---

5. `/ws/oficinas`

Gestiona la lista de oficinas monitoreadas.

Conexión

```javascript
const ws = new WebSocket('ws://localhost:8081/ws/oficinas');

ws.onmessage = (event) => {
    const data = JSON.parse(event.data);
    actualizarListaOficinas(data.data);
};
```

Mensaje Recibido

```json
{
  "tipo": "oficinas",
  "data": {
    "A": {
      "nombre": "A",
      "activa": true,
      "timestamp": 1701648000
    },
    "B": {
      "nombre": "B",
      "activa": true,
      "timestamp": 1701648000
    }
  }
}
```

Agregar Oficina

```javascript
ws.send(JSON.stringify({
    tipo: 'actualizar_oficinas',
    data: {
        "D": {
            nombre: "D",
            activa: true,
            timestamp: Date.now() / 1000
        }
    }
}));
```

Eliminar Oficina

```javascript
ws.send(JSON.stringify({
    tipo: 'eliminar_oficina',
    data: {
        oficina: 'D'

```

```
    }
}));
```

Ejemplo Completo de Cliente

javascript

```javascript
class MonitoreoClient {
    constructor() {
        this.connections = {};
    }

    conectar() {
        // Resumenes
        this.connections.resumenes = new WebSocket('ws://localhost:8081/ws/resumen
        this.connections.resumenes.onmessage = (e) => {
            const data = JSON.parse(e.data);
            this.onResumenes(data.data);
        };

        // Avisos
        this.connections.avisos = new WebSocket('ws://localhost:8081/ws/avisos');
        this.connections.avisos.onmessage = (e) => {
            const data = JSON.parse(e.data);
            this.onAvisos(data.data);
        };

        // Dispositivos
        this.connections.dispositivos = new WebSocket('ws://localhost:8081/ws/disp
        this.connections.dispositivos.onmessage = (e) => {
            const data = JSON.parse(e.data);
            this.onDispositivos(data.data);
        };

        // Params
        this.connections.params = new WebSocket('ws://localhost:8081/ws/params');
        this.connections.params.onmessage = (e) => {
            const data = JSON.parse(e.data);
            this.onParams(data.data);
        };

        // Oficinas
        this.connections.oficinas = new WebSocket('ws://localhost:8081/ws/oficinas
        this.connections.oficinas.onmessage = (e) => {
            const data = JSON.parse(e.data);
            this.onOficinas(data.data);
        };
```

```
    }

    actualizarDispositivo(oficina, dispositivo, estado) {
        this.connections.dispositivos.send(JSON.stringify({
            tipo: 'actualizar_dispositivo',
            oficina,
            dispositivo,
            estado
        }));
    }

    actualizarParams(params) {
        this.connections.params.send(JSON.stringify({
            tipo: 'actualizar_params',
            data: params
        }));
    }

    agregarOficina(id) {
        this.connections.oficinas.send(JSON.stringify({
            tipo: 'actualizar_oficinas',
            data: {
                [id]: {
                    nombre: id,
                    activa: true,
                    timestamp: Date.now() / 1000
                }
            }
        }));
    }

    eliminarOficina(id) {
        this.connections.oficinas.send(JSON.stringify({
            tipo: 'eliminar_oficina',
            data: { oficina: id }
        }));
    }

    // Callbacks (implementar según necesidad)
    onResumenes(data) { console.log('Resumenes:', data); }
    onAvisos(data) { console.log('Avisos:', data); }
    onDispositivos(data) { console.log('Dispositivos:', data); }
    onParams(data) { console.log('Params:', data); }
    onOficinas(data) { console.log('Oficinas:', data); }
}

// Uso
const client = new MonitoreoClient();
client.conectar();
```

```javascript

// Cambiar dispositivo
client.actualizarDispositivo('A', 'luces', false);

// Actualizar configuración
client.actualizarParams({
    hora_inicio: 9.0,
    hora_fin: 18.0,
    umbral_temperatura_ac: 24.0,
    umbral_corriente: 20.0,
    voltaje: 220.0,
    costo_kwh: 0.30
});
```

Manejo de Errores

javascript

```javascript
ws.onerror = (error) => {
    console.error('WebSocket error:', error);
};

ws.onclose = (event) => {
    console.log('WebSocket closed:', event.code, event.reason);
    // Intentar reconectar
    setTimeout(() => {
        conectar();
    }, 5000);
};
```

Códigos de Cierre

Código	Descripción
1000	Cierre normal
1001	Endpoint desapareciendo
1006	Conexión anormal
1011	Error interno del servidor

Consejo

Implementa reconexión automática en caso de pérdida de conexión para asegurar que el dashboard siempre muestre datos actualizados.

Arquitectura del Sistema

Vista General

El sistema está diseñado con una **arquitectura distribuida basada en microservicios**, donde cada componente tiene una responsabilidad específica y se comunica mediante protocolos estándar (MQTT, WebSocket, HTTP).

mermaid

```
graph TB
    subgraph "Capa de Datos"
        SENS1[Sensores Oficina A]
        SENS2[Sensores Oficina B]
        SENS3[Sensores Oficina C]
    end

    subgraph "Capa de Mensajería MQTT"
        BROKER[Mosquitto Broker<br/>Puerto 1883]
        PUB[Go Publisher<br/>Simulador de Sensores]
        SUB[Go Subscriber<br/>Procesador de Eventos]
    end

    subgraph "Capa de Persistencia"
        FB[(Firebase<br/>Realtime Database)]
    end

    subgraph "Capa de Distribución"
        WS[WebSocket Server<br/>Puerto 8081]
        HTTP[HTTP Server<br/>Puerto 8080]
    end

    subgraph "Capa de Procesamiento"
        MPI[MPI Backend<br/>Análisis Paralelo]
    end

    subgraph "Capa de Presentación"
        DASH[Dashboard HTML/JS]
        CHARTS[Chart.js Visualizations]
    end

```

```
    SENS1 -.Simula.-> PUB
    SENS2 -.Simula.-> PUB
    SENS3 -.Simula.-> PUB

    PUB -->|Publish| BROKER
    BROKER -->|Subscribe| SUB

    SUB -->|Guarda Datos| FB
    SUB -->|Envia Eventos| WS

    FB -->|Lee Datos| MPI
    MPI -->|Resultados| FB

    WS <-->|WebSocket| DASH
    HTTP -->|Sirve| DASH
    DASH -->|Renderiza| CHARTS

    style BROKER fill:#ff6b6b,color:#fff
    style FB fill:#4ecdc4,color:#fff
    style WS fill:#f9ca24,color:#333
    style DASH fill:#95e1d3,color:#333
```

Componentes Principales

1. Mosquitto MQTT Broker

Propósito: Broker de mensajes que implementa el protocolo MQTT.

Características:

- Puerto: `1883`
- Permite conexiones anónimas (configurado para desarrollo)
- Persistencia de mensajes habilitada
- Logs en `logs/mosquitto.log`

Configuración (`config/mosquitto.conf`):

conf

```
listener 1883
allow_anonymous true
persistence true
persistence_location data/mosquitto/
max_connections 100
```

Tópicos MQTT:

```
oficinas/{id}/sensores
```

2. Go Publisher (Simulador de Sensores)

Ubicación: `mqtt/publisher/main.go`

Propósito: Simula sensores de múltiples oficinas publicando datos cada 10 segundos.

Funcionalidades:

Generación de Datos Realistas

```go
type DatosSensor struct {
    Oficina     string  `json:"oficina"`
    TiempoUnix  int64   `json:"timestamp"`
    Presencia   bool    `json:"presencia"`
    CorrienteA  float64 `json:"corriente_a"`
    Temperatura float64 `json:"temperatura"`
}
```

Lógica de Presencia

```go
func DetectarPresencia(t time.Time) bool {
    dia := t.Weekday()
    hora := float64(t.Hour()) + float64(t.Minute())/100.0
    return dia >= time.Monday && dia <= time.Friday &&
        hora >= params.HoraInicio && hora < params.HoraFin
}
```

Cálculo de Corriente

La corriente se calcula basándose en:

- **Consumo base**: 0.5-3.0 A (equipos siempre encendidos)
- **Luces**: +3.0 A (si hay presencia y están encendidas)
- **Aire acondicionado**: +10.0 A (si temperatura > umbral y está encendido)
- **Consumo adicional**: +1.0-7.0 A (equipos varios)

```go
func CalcularCorriente(oficina string, presencia bool, temperatura float64) float6
    base := 0.5 + rand.Float64()*(3.0-0.5)

    if presencia {
        if estado["luces"] {
            base += 3.0  // Luces
```

```
        }
        if temperatura >= params.UmbralTemperaturaAC && estado["aire"] {
            base += 10.0  // Aire acondicionado
        }
        base += 1.0 + rand.Float64()*(7.0-1.0)  // Equipos
    }

    return base
}
```

Sincronización con WebSocket

El Publisher escucha cambios en:

- `/ws/dispositivos` - Estado de luces y aire
- `/ws/oficinas` - Lista de oficinas activas
- `/ws/params` - Parámetros de configuración

3. Go Subscriber (Procesador de Eventos)

Ubicación: `mqtt/subscriber/main.go`

Propósito: Procesa mensajes MQTT, detecta alertas y genera resúmenes.

Funcionalidades:

Detección de Alertas

El sistema detecta 13 tipos de alertas:

ID	Tipo	Descripción
0	Luces apagadas	Estado desactivado
1	Luces encendidas	Presencia detectada
2	Luces apagadas	Ausencia prolongada
3	Aire apagado	Estado desactivado
4	Aire encendido	Temperatura elevada
5	Aire apagado	Condiciones óptimas
6	Consumo anómalo	Corriente alta sin presencia
7	Corte de energía	Corriente en 0
8	Sensor no responde	Sin datos por >60s

ID	Tipo	Descripción
9	Alerta de corriente	Consumo > umbral
10	Oficina agregada	Nueva oficina
11	Oficina eliminada	Oficina removida
12	Config modificada	Parámetros cambiados

Generación de Resúmenes

Cada 60 segundos, se genera un resumen por oficina:

```go
type Resumen struct {
    Timestamp       int64   `json:"timestamp"`
    CorrienteA      float64 `json:"corriente_a"`       // Promedio
    ConsumoKvh      float64 `json:"consumo_kvh"`       // Período actual
    ConsumoTotalKvh float64 `json:"consumo_total_kvh"` // Acumulado
    MinTemp         float64 `json:"min_temp"`
    MaxTemp         float64 `json:"max_temp"`
    TiempoPresente  int     `json:"tiempo_presente"`   // Segundos
    MontoEstimado   float64 `json:"monto_estimado"`    // Costo período
    MontoTotal      float64 `json:"monto_total"`       // Costo total
}
```

Cálculo de Consumo:

```go
// Potencia promedio (Watts)
promedioW := promedioAmp * voltaje

// Energía consumida (kWh)
duracionHoras := float64(duracionSegundos) / 3600.0
promedioKwh := promedioW * duracionHoras / 1000.0

// Costo
monto := promedioKwh * costoKwh
```

4. WebSocket Server

Ubicación: `socket.js`

Propósito: Distribuir datos en tiempo real a clientes conectados.

Endpoints:

```javascript
ws://localhost:8081/ws/resumenes      // Resúmenes de consumo
ws://localhost:8081/ws/avisos         // Alertas del sistema
ws://localhost:8081/ws/dispositivos   // Estado de dispositivos
ws://localhost:8081/ws/params         // Configuración
ws://localhost:8081/ws/oficinas       // Gestión de oficinas
```

Flujo de Datos:

```mermaid
sequenceDiagram
    participant C as Cliente (Dashboard)
    participant WS as WebSocket Server
    participant SUB as Go Subscriber
    participant FB as Firebase

    C->>WS: Conectar a /ws/resumenes
    WS->>C: Enviar datos iniciales

    loop Cada 10 segundos
        SUB->>FB: Guardar resumen
        SUB->>WS: Notificar nuevo resumen
        WS->>C: Broadcast resumen
    end

    C->>WS: Actualizar dispositivo
    WS->>SUB: Propagar cambio
    WS->>C: Confirmar actualización
```

5. Firebase Realtime Database

Estructura de Datos:

```
monitoreo_consumo/
├── configuracion/
│   ├── hora_inicio: 8.0
│   ├── hora_fin: 20.0
│   ├── umbral_temperatura_ac: 25.0
│   ├── umbral_corriente: 21.5
│   ├── voltaje: 220.0
│   └── costo_kwh: 0.25
├── tipos_avisos/
│   ├── 0: { motivo, detalle, impacto }
│   ├── 1: { motivo, detalle, impacto }
│   └── ...
├── oficinas/
│   ├── A/
│   │   ├── nombre: "Oficina A"
```

```
16  |   |   ├── sector: "Informatica"
17  |   |   ├── avisos/
18  |   |   |   └── {push_id}: { timestamp, id_tipo, adicional }
19  |   |   ├── resumenes/
20  |   |   |   └── {push_id}: { timestamp, corriente_a, consumo_kvh, ... }
21  |   |   └── estados_dispositivos/
22  |   |       ├── aire: true
23  |   |       └── luces: true
24  |   ├── B/
25  |   └── C/
```

6. MPI Backend (Procesamiento Paralelo)

Ubicación: `backend/mpi/mpi_analysis.c`

Propósito: Análisis avanzado de eficiencia energética usando procesamiento paralelo.

Funcionalidades:

Análisis de Eficiencia

```c
void analyze_energy_data(OfficeData *offices, int office_count,
                         AnalysisResult *result, int rank, int size) {
    // Dividir datos entre procesos
    int chunk_size = office_count / size;

    // Cada proceso calcula métricas locales
    for (int i = start_index; i < end_index; i++) {
        local_total_consumption += offices[i].consumption;
        double efficiency = calculate_efficiency(offices[i]);
        // ...
    }

    // Reducir resultados con MPI_Reduce
    MPI_Reduce(&local_total, &result->total, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_W
}
```

Clustering de Consumo

K-means paralelo para clasificar oficinas en 3 grupos:

- **Bajo consumo**: < 0.5 kWh
- **Consumo medio**: 0.5-2.0 kWh
- **Alto consumo**: > 2.0 kWh

7. Dashboard Frontend

Ubicación: `resources/template.html`

Tecnologías:

- HTML5 + CSS3
- JavaScript vanilla
- Chart.js para visualizaciones
- WebSocket API

Características:

- Tema claro/oscuro
- Gráficos en tiempo real
- Control de dispositivos
- Gestión de oficinas
- Configuración de parámetros

Patrones de Diseño

Publisher-Subscriber (MQTT)

Desacopla productores de datos (Publisher) de consumidores (Subscriber) mediante un broker central.

Ventajas:

- Escalabilidad
- Bajo acoplamiento
- Resiliencia

WebSocket para Tiempo Real

Conexión bidireccional persistente para actualizaciones instantáneas.

Ventajas:

- Baja latencia
- Eficiencia (sin polling)
- Comunicación bidireccional

Procesamiento Paralelo (MPI)

Divide análisis pesados entre múltiples nodos.

Ventajas:

- Alto rendimiento
- Escalabilidad horizontal
- Aprovechamiento de múltiples cores

Flujo de Ejecución Completo

mermaid

```mermaid
sequenceDiagram
    participant U as Usuario
    participant D as Dashboard
    participant WS as WebSocket
    participant SUB as Subscriber
    participant MQTT as Mosquitto
    participant PUB as Publisher
    participant FB as Firebase

    U->>D: Abrir dashboard
    D->>WS: Conectar WebSocket
    WS->>D: Enviar datos iniciales

    loop Cada 10 segundos
        PUB->>MQTT: Publicar datos sensor
        MQTT->>SUB: Entregar mensaje
        SUB->>SUB: Detectar alertas
        SUB->>FB: Guardar datos
        SUB->>WS: Notificar cambios
        WS->>D: Actualizar dashboard
        D->>D: Renderizar gráficos
    end

    U->>D: Cambiar configuración
    D->>WS: Enviar nuevo config
    WS->>FB: Guardar config
    WS->>PUB: Notificar cambio
    WS->>SUB: Notificar cambio
```

Próximos Pasos

- Instalación - Configura el sistema
- Ejecución - Aprende a usar el script de control
- API WebSocket - Referencia técnica de endpoints

Instalación del Sistema

Esta guía te llevará paso a paso por la instalación y configuración del Sistema de Monitoreo de Consumo Eléctrico.

Requisitos Previos

Software Requerido

Software	Versión Mínima	Propósito
Go	1.19+	Publisher y Subscriber MQTT
Node.js	14.0+	WebSocket y HTTP servers
Mosquitto	2.0+	MQTT Broker
Git	2.0+	Control de versiones
Firebase Account	-	Base de datos en la nube

Verificar Instalaciones

bash

```bash
# Verificar Go
go version
# Salida esperada: go version go1.24.2 windows/amd64

# Verificar Node.js
node --version
# Salida esperada: v18.x.x o superior

# Verificar npm
npm --version
# Salida esperada: 9.x.x o superior

# Verificar Mosquitto
mosquitto -h
# Debe mostrar ayuda de Mosquitto
```

Instalación de Dependencias

1. Instalar Go

Windows:

```bash
# Descargar desde https://go.dev/dl/
# Ejecutar el instalador
# Verificar instalación
go version
```

Linux/macOS:

```bash
# Ubuntu/Debian
sudo apt update
sudo apt install golang-go

# macOS (Homebrew)
brew install go
```

2. Instalar Node.js

Windows:

```bash
# Descargar desde https://nodejs.org/
# Ejecutar el instalador (incluye npm)
```

Linux:

```bash
# Ubuntu/Debian
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt-get install -y nodejs
```

macOS:

```bash
brew install node
```

3. Instalar Mosquitto

Windows:

```bash
# Descargar desde https://mosquitto.org/download/
# Ejecutar instalador
# Agregar a PATH: C:\Program Files\mosquitto
```

Linux:

```bash
# Ubuntu/Debian
sudo apt update
sudo apt install mosquitto mosquitto-clients

# Habilitar servicio
sudo systemctl enable mosquitto
sudo systemctl start mosquitto
```

macOS:

```bash
brew install mosquitto

# Iniciar servicio
brew services start mosquitto
```

Configuración del Proyecto

1. Clonar el Repositorio

```bash
git clone https://github.com/tu-usuario/monitoreo-consumo.git
cd monitoreo-consumo
```

2. Configurar Firebase

Crear Proyecto Firebase

1. Ve a [Firebase Console]
2. Crea un nuevo proyecto
3. Habilita **Realtime Database**
4. Configura reglas de seguridad (para desarrollo):

```json
{
  "rules": {

```

```
    ".read": true,
    ".write": true
  }
}
```

Obtener Credenciales

1. En Firebase Console → Project Settings → Service Accounts
2. Click "Generate new private key"
3. Descarga el archivo JSON
4. Guárdalo como `credentials/firebase-credentials.json`

bash
```
# Crear directorio de credenciales
mkdir -p credentials

# Copiar archivo descargado
cp ~/Downloads/firebase-credentials.json credentials/
```

Actualizar URL de Firebase

Edita `socket.js` y `mqtt/subscriber/main.go` con tu URL:

javascript
```
// socket.js
admin.initializeApp({
    credential: admin.credential.cert(serviceAccount),
    databaseURL: "https://TU-PROYECTO-default-rtdb.firebaseio.com/"
});
```

go
```
// mqtt/subscriber/main.go
clienteFirebase, err = app.DatabaseWithURL(ctx,
    "https://TU-PROYECTO-default-rtdb.firebaseio.com/")
```

3. Ejecutar Script de Instalación

El sistema incluye un script automatizado que configura todo:

bash
```
# Dar permisos de ejecución (Linux/macOS)
chmod +x monitoreo.sh

# Ejecutar instalación
./monitoreo.sh instalar
```

El script realizará:

- Verificación de dependencias (Go, Node.js, Mosquitto)
- Creación de estructura de carpetas
- Instalación de dependencias Go (`go mod tidy`)
- Instalación de dependencias Node.js (`npm install`)
- Generación de configuración de Mosquitto

Salida esperada:

```
<i class="fas fa-rocket"></i> Configurando el sistema de Monitoreo de Consumo...
<i class="fas fa-check-circle"></i> Go encontrado: go version go1.24.2 windows/amd
<i class="fas fa-check-circle"></i> Node.js encontrado: v18.17.0
<i class="fas fa-check-circle"></i> Mosquitto encontrado
📁 Creando estructura de carpetas...
<i class="fas fa-check-circle"></i> Credenciales Firebase encontradas
📦 Instalando dependencias Go...
<i class="fas fa-check-circle"></i> Dependencias Go instaladas
📦 Instalando dependencias Node.js...
<i class="fas fa-check-circle"></i> Dependencias Node.js instaladas
<i class="fas fa-file-alt"></i> Creando configuración Mosquitto...
<i class="fas fa-check-circle"></i> Configuración Mosquitto creada
<i class="fas fa-check-circle"></i> Configuración completada!

<i class="fas fa-bullseye"></i> Ahora puedes usar:
   ./monitoreo.sh comenzar  - Para iniciar el sistema
   ./monitoreo.sh estado    - Para ver el estado
   ./monitoreo.sh parar     - Para detener todo
```

4. Inicializar Base de Datos

Ejecuta el script de semilla para crear la estructura inicial en Firebase:

```bash
node semilla_firebase.js
```

Esto creará:

- Parámetros de configuración por defecto
- Tipos de avisos (13 tipos)
- Oficinas iniciales (A, B, C)

Estructura de Carpetas

Después de la instalación, tendrás:

```
monitoreo-consumo/
├── backend/
│   └── mpi/
│       ├── mpi_analysis.c
│       └── test_data.json
├── config/
│   ├── firebase-config.js
│   └── mosquitto.conf
├── credentials/
│   └── firebase-credentials.json  ← Tu archivo
├── data/
│   └── mosquitto/                 ← Persistencia MQTT
├── logs/
│   └── mosquitto.log              ← Logs del broker
├── mqtt/
│   ├── publisher/
│   │   └── main.go
│   └── subscriber/
│       └── main.go
├── resources/
│   ├── assets/
│   └── template.html
├── dashboard.js
├── socket.js
├── semilla_firebase.js
├── monitoreo.sh
├── go.mod
├── go.sum
├── package.json
└── package-lock.json
```

Configuración Avanzada

Personalizar Parámetros

Edita `semilla_firebase.js` antes de ejecutarlo:

javascript

```javascript
const paramsPorDefecto = {
    hora_inicio: 8.0,          // Hora de inicio (8:00 AM)
    hora_fin: 20.0,            // Hora de fin (8:00 PM)
    umbral_temperatura_ac: 25.0, // °C para activar AC
    umbral_corriente: 21.5,    // Amperes máximos
    voltaje: 220.0,            // Voltaje de red

```

```
    costo_kwh: 0.25,           // Costo por kWh
};
```

Configurar Mosquitto

El archivo `config/mosquitto.conf` se genera automáticamente, pero puedes personalizarlo:

conf

```
listener 1883
allow_anonymous true

# Logs
log_dest file logs/mosquitto.log
log_type all

# Persistencia
persistence true
persistence_location data/mosquitto/

# Límites
max_connections 100
max_queued_messages 1000
max_packet_size 104857600
```

Compilar Backend MPI (Opcional)

Si deseas usar el procesamiento paralelo:

bash

```
# Instalar OpenMPI
sudo apt install openmpi-bin libopenmpi-dev  # Linux
brew install open-mpi                         # macOS

# Compilar
cd backend/mpi
mpicc -o mpi_analysis mpi_analysis.c -ljansson -lm

# Ejecutar con 4 procesos
mpirun -np 4 ./mpi_analysis test_data.json
```

Verificación de Instalación

Test de Componentes

```bash
# 1. Verificar que Mosquitto inicia
mosquitto -c config/mosquitto.conf -v
# Debe mostrar: "mosquitto version X.X.X running"

# 2. Verificar dependencias Node.js
cd Monitoreo-Consumo
npm list
# No debe mostrar errores

# 3. Verificar dependencias Go
cd mqtt/publisher
go build
cd ../subscriber
go build
# Ambos deben compilar sin errores
```

Test de Conectividad Firebase

Crea un archivo `test-firebase.js` :

```javascript
const admin = require('firebase-admin');
const serviceAccount = require('./credentials/firebase-credentials.json');

admin.initializeApp({
    credential: admin.credential.cert(serviceAccount),
    databaseURL: "https://TU-PROYECTO-default-rtdb.firebaseio.com/"
});

const db = admin.database();
db.ref('test').set({ timestamp: Date.now() })
    .then(() => {
        console.log('<i class="fas fa-check-circle"></i> Conexión Firebase exitosa
        process.exit(0);
    })
    .catch(err => {
        console.error('<i class="fas fa-times-circle"></i> Error:', err);
        process.exit(1);
    });
```

```bash
node test-firebase.js
```

Solución de Problemas

Error: "Mosquitto not found"

Solución:

```bash
# Verificar instalación
which mosquitto  # Linux/macOS
where mosquitto  # Windows

# Si no está en PATH, agregar manualmente
export PATH=$PATH:/usr/local/sbin  # Linux/macOS
```

Error: "Firebase credentials not found"

Solución:

```bash
# Verificar que el archivo existe
ls -la credentials/firebase-credentials.json

# Verificar permisos
chmod 600 credentials/firebase-credentials.json
```

Error: "Port 1883 already in use"

Solución:

```bash
# Ver qué proceso usa el puerto
lsof -i :1883  # Linux/macOS
netstat -ano | findstr :1883  # Windows

# Detener Mosquitto existente
sudo systemctl stop mosquitto  # Linux
brew services stop mosquitto   # macOS
taskkill /F /IM mosquitto.exe  # Windows
```

Error: "Cannot find module 'firebase-admin'"

Solución:

```bash
# Reinstalar dependencias
rm -rf node_modules package-lock.json
npm install
```

Próximos Pasos

Una vez completada la instalación:

1. Ejecutar el Sistema - Aprende a iniciar todos los componentes
2. Usar el Dashboard - Guía de uso del panel de control
3. API Reference - Documentación técnica de endpoints

Consejo

Guarda una copia de seguridad de tu archivo `firebase-credentials.json` en un lugar seguro. No lo subas a Git (ya está en .gitignore).

Previous page
Arquitectura

Next page
Ejecución

Introducción al Sistema de Monitoreo de Consumo Eléctrico

¿Qué es este sistema?

El **Sistema de Monitoreo de Consumo Eléctrico** es una plataforma distribuida en tiempo real diseñada para monitorear, analizar y optimizar el consumo energético en múltiples oficinas o espacios. Combina tecnologías modernas de mensajería, procesamiento paralelo y visualización de datos para proporcionar información accionable sobre el uso de energía.

Características Principales

Monitoreo en Tiempo Real

El sistema recopila datos cada **10 segundos** de cada oficina, incluyendo:

- **Corriente eléctrica** (Amperes)
- **Temperatura ambiente** (°C)
- **Detección de presencia** (boolean)
- **Estado de dispositivos** (luces, aire acondicionado)

Análisis Inteligente

- **Resúmenes cada 60 segundos**: Consumo promedio, temperaturas min/max, tiempo de presencia
- **Detección automática de alertas**: Consumo anómalo, cortes de energía, sensores sin respuesta
- **Cálculo de costos**: Estimación de costos basada en consumo y tarifa configurable
- **Procesamiento paralelo MPI**: Análisis de eficiencia energética y clustering de consumo

Gestión Dinámica

- **Agregar/eliminar oficinas** en tiempo real

- **Configuración de parámetros** (horarios, umbrales, costos)
- **Control de dispositivos** (encender/apagar luces y aire acondicionado)

Arquitectura de Alto Nivel

mermaid

```mermaid
graph LR
    A[Sensores] -->|MQTT| B[Mosquitto Broker]
    B --> C[Go Publisher]
    B --> D[Go Subscriber]
    D --> E[Firebase]
    D --> F[WebSocket Server]
    F --> G[Dashboard]
    E --> H[MPI Analysis]

    style B fill:#ff6b6b,color:#fff
    style E fill:#4ecdc4,color:#fff
    style G fill:#95e1d3,color:#333
```

Stack Tecnológico

Backend

Componente	Tecnología	Propósito
Message Broker	Mosquitto (MQTT)	Comunicación pub/sub entre componentes
Publisher	Go + Paho MQTT	Simulación de datos de sensores
Subscriber	Go + Paho MQTT	Procesamiento de mensajes y lógica de negocio
WebSocket Server	Node.js + ws	Distribución de datos en tiempo real
HTTP Server	Node.js + http	Servir dashboard frontend
Database	Firebase Realtime DB	Persistencia de datos
Parallel Processing	C + OpenMPI	Análisis avanzado de eficiencia

Frontend

Tecnología	Uso
HTML5/CSS3/JavaScript	Estructura y lógica del dashboard
Chart.js	Gráficos de consumo en tiempo real
WebSocket API	Conexión bidireccional con servidor

Flujo de Datos Completo

1. Generación de Datos (Publisher)

go

```go
// Cada 10 segundos
datos := DatosSensor{
    Oficina:     "A",
    Timestamp:   time.Now().Unix(),
    Presencia:   true,
    CorrienteA:  12.5,
    Temperatura: 24.3,
}
```

2. Publicación MQTT

```
Topic: oficinas/A/sensores
Payload: {"oficina":"A","timestamp":1701648000,"presencia":true,"corriente_a":12.5
```

3. Procesamiento (Subscriber)

El Subscriber:

- Recibe el mensaje MQTT
- Detecta alertas (consumo elevado, ausencia de corriente, etc.)
- Acumula datos para resúmenes
- Guarda en Firebase

4. Distribución (WebSocket)

javascript

```javascript
// Servidor envía a todos los clientes conectados
ws.send(JSON.stringify({
    tipo: 'resumenes',
    data: {

```

```
        A: { corriente_a: 12.5, consumo_kvh: 2.75, ... },
        B: { corriente_a: 8.2, consumo_kvh: 1.80, ... }
    }
}));
```

5. Visualización (Dashboard)

El dashboard recibe los datos y actualiza gráficos en tiempo real.

Casos de Uso

1. Monitoreo de Oficinas

Escenario: Una empresa quiere monitorear el consumo de 3 oficinas.

Solución: El sistema recopila datos de cada oficina y muestra:

- Consumo actual y total
- Costos estimados
- Temperaturas
- Tiempo de ocupación

2. Detección de Anomalías

Escenario: Se detecta consumo elevado sin presencia.

Solución: El sistema genera una alerta automática:

json

```
{
  "timestamp": 1701648000,
  "id_tipo": "6",
  "adicional": "Consumo: 15.2 A sin presencia detectada"
}
```

3. Optimización Energética

Escenario: Analizar patrones de consumo para reducir costos.

Solución: El backend MPI procesa datos históricos y genera:

- Clustering de oficinas por consumo
- Métricas de eficiencia
- Recomendaciones de optimización

Ventajas del Sistema

- **Escalable**: Agregar oficinas dinámicamente sin reiniciar
- **Tiempo Real**: Visualización instantánea de cambios
- **Distribuido**: Componentes independientes y resilientes
- **Inteligente**: Detección automática de anomalías
- **Flexible**: Configuración de parámetros en caliente
- **Eficiente**: Procesamiento paralelo para análisis pesados

---

Próximos Pasos

Ahora que entiendes qué es el sistema, explora:

- Arquitectura Detallada - Profundiza en cada componente
- Instalación - Configura el sistema paso a paso
- Ejecución - Aprende a usar el script de control

¿Necesitas ayuda?

Consulta la Referencia de API para detalles técnicos de cada endpoint.

Ejecución del Sistema

Esta guía explica cómo iniciar, detener y monitorear el Sistema de Monitoreo de Consumo Eléctrico.

Script de Control (`monitoreo.sh`)

El sistema incluye un script Bash que facilita la gestión de todos los componentes.

Comandos Disponibles

bash

```bash
./monitoreo.sh instalar   # Primera configuración
./monitoreo.sh comenzar   # Iniciar sistema completo
./monitoreo.sh estado     # Ver estado de servicios
./monitoreo.sh parar      # Detener todos los servicios
./monitoreo.sh ayuda      # Mostrar ayuda
```

Iniciar el Sistema

Inicio Completo

bash

```bash
./monitoreo.sh comenzar
```

Proceso de inicio:

mermaid

```mermaid
graph LR
    A[Ejecutar Script] --> B[Iniciar Mosquitto]
    B --> C[Iniciar WebSocket Server]
    C --> D[Iniciar Dashboard HTTP]
    D --> E[Iniciar Subscriber]
    E --> F[Iniciar Publisher]
    F --> G[Sistema Listo]

```

```
    style G fill:#95e1d3
```

Salida esperada:

```
<i class="fas fa-rocket"></i> Iniciando Sistema de Monitoreo de Consumo...
<i class="fas fa-broadcast-tower"></i> Iniciando Mosquitto MQTT Broker...
<i class="fas fa-check-circle"></i> Mosquitto iniciado (PID: 12345)
<i class="fas fa-plug"></i> Iniciando WebSocket Server...
<i class="fas fa-check-circle"></i> Servidor WebSocket escuchando en puerto 8081
<i class="fas fa-globe"></i> Iniciando Dashboard...
<i class="fas fa-check-circle"></i> Servidor HTTP escuchando en http://localhost:8
<i class="fas fa-download"></i> Iniciando Subscriber...
<i class="fas fa-check-circle"></i> Subscriber conectado a MQTT
<i class="fas fa-chart-bar"></i> Iniciando Publisher...
<i class="fas fa-check-circle"></i> Publisher conectado a MQTT
<i class="fas fa-check-circle"></i> Todos los componentes iniciados correctamente!

<i class="fas fa-globe"></i> Abre tu navegador en: http://localhost:8080

<i class="fas fa-chart-line"></i> El sistema está funcionando. Esperando datos...

<i class="fas fa-lightbulb"></i> Presiona Ctrl+C para detener todos los servicios
```

Inicio Manual (Componente por Componente)

Si prefieres control granular:

1. Iniciar Mosquitto

bash

```bash
mosquitto -c config/mosquitto.conf -v
```

Verás:

```
1701648000: mosquitto version 2.0.18 running
1701648000: Opening ipv4 listen socket on port 1883.
```

2. Iniciar WebSocket Server

bash

```bash
node socket.js
```

Verás:

```
<i class="fas fa-plug"></i> Iniciando servidor WebSocket...
<i class="fas fa-check-circle"></i> Servidor WebSocket escuchando en puerto 8081
<i class="fas fa-check-circle"></i> Servidores WebSocket listos:
   <i class="fas fa-chart-bar"></i> ws://localhost:8081/ws/resumenes
   <i class="fas fa-bell"></i> ws://localhost:8081/ws/avisos
   <i class="fas fa-lightbulb"></i> ws://localhost:8081/ws/dispositivos
<i class="fas fa-sync-alt"></i> Iniciando simulación de datos en tiempo real...
```

3. Iniciar Dashboard HTTP

bash

```
node dashboard.js
```

Verás:

```
<i class="fas fa-check-circle"></i> Servidor HTTP escuchando en http://localhost:8(
```

4. Iniciar Subscriber

bash

```
cd mqtt/subscriber
go run main.go
```

Verás:

```
<i class="fas fa-plug"></i> Cliente conectado a PARAMS
<i class="fas fa-clipboard"></i> Lista de oficinas actualizada: [A B C]
<i class="fas fa-check-circle"></i> Nueva oficina inicializada: A
<i class="fas fa-check-circle"></i> Nueva oficina inicializada: B
<i class="fas fa-check-circle"></i> Nueva oficina inicializada: C
```

5. Iniciar Publisher

bash

```
cd mqtt/publisher
go run main.go
```

Verás:

```
<i class="fas fa-plug"></i> Probando conexión WebSocket...
<i class="fas fa-check-circle"></i> WebSocket conectado correctamente
<i class="fas fa-check-circle"></i> LISTA DE OFICINAS ACTUALIZADA: [A B C]
[PUBLICADO] oficinas/A/sensores -> {"oficina":"A","timestamp":1701648000,...}
```

Verificar Estado

```bash
./monitoreo.sh estado
```

Salida esperada:

```
<i class="fas fa-chart-bar"></i> Estado del Sistema de Monitoreo de Consumo

<i class="fas fa-check-circle"></i> Mosquitto: EJECUTÁNDOSE
<i class="fas fa-check-circle"></i> Servicios Node.js: EJECUTÁNDOSE (2 procesos)
<i class="fas fa-check-circle"></i> Servicios Go: EJECUTÁNDOSE (2 procesos)
<i class="fas fa-check-circle"></i> Sistema principal: EJECUTÁNDOSE (PID: 12340)

<i class="fas fa-lightbulb"></i> Usa './monitoreo.sh comenzar' para iniciar el sis
<i class="fas fa-lightbulb"></i> Usa './monitoreo.sh parar' para detener todo
```

Acceder al Dashboard

Una vez iniciado el sistema:

1. Abre tu navegador
2. Navega a: http://localhost:8080
3. Verás el dashboard con datos en tiempo real

Características del Dashboard

Panel de Resúmenes

Muestra para cada oficina:

- **Corriente actual** (Amperes)
- **Consumo** (kWh)
- **Consumo total acumulado**
- **Temperatura mínima/máxima**
- **Tiempo de presencia**
- **Costo estimado**
- **Costo total**

Panel de Avisos

Lista de alertas en tiempo real:

- Timestamp
- Tipo de aviso
- Información adicional

Panel de Dispositivos

Control de:

- **Luces** (ON/OFF)
- **Aire acondicionado** (ON/OFF)

Panel de Configuración

Ajustar parámetros:

- Hora de inicio/fin
- Umbral de temperatura para AC
- Umbral de corriente
- Voltaje
- Costo por kWh

Gestión de Oficinas

- Agregar nuevas oficinas
- Eliminar oficinas existentes

Detener el Sistema

Detención Completa

bash

```
./monitoreo.sh parar
```

O presiona **Ctrl+C** si iniciaste con `./monitoreo.sh comenzar`

Salida esperada:

```
<i class="fas fa-stop-circle"></i> Deteniendo Sistema de Monitoreo (PID: 12340)...
<i class="fas fa-check-circle"></i> Mosquitto detenido (PID: 12345)
<i class="fas fa-check-circle"></i> Todos los servicios detenidos
```

Detención Manual

Si iniciaste componentes manualmente:

```bash
# Detener cada proceso con Ctrl+C
# O buscar y matar procesos

# Linux/macOS
pkill -f mosquitto
pkill -f "node socket.js"
pkill -f "node dashboard.js"
pkill -f "go run main.go"

# Windows
taskkill /F /IM mosquitto.exe
taskkill /F /IM node.exe
taskkill /F /IM go.exe
```

Monitoreo de Logs

Logs de Mosquitto

```bash
tail -f logs/mosquitto.log
```

Verás:

```
1701648000: New connection from 127.0.0.1:54321 on port 1883.
1701648000: New client connected from 127.0.0.1:54321 as publicador-sensores
1701648000: Received PUBLISH from publicador-sensores (d0, q0, r0, m0, 'oficinas/A,
```

Logs del Publisher

El Publisher muestra:

```
<i class="fas fa-plug"></i> Calculando corriente - Presencia: true, Temp: 24.5, Um
<i class="fas fa-lightbulb"></i> Oficina A - Luces: true, Aire: false
<i class="fas fa-lightbulb"></i> Luces encendidas (+3.0A)
<i class="fas fa-plug"></i> Corriente final: 8.2A
[PUBLICADO] oficinas/A/sensores -> {"oficina":"A",...}
```

Logs del Subscriber

El Subscriber muestra:

```
[AVISO] Oficina:A Tipo:1 Más:
[RESUMEN] Oficina:A {Timestamp:1701648000 CorrienteA:8.2 ConsumoKvh:1.8 ...}
Resumen guardado en: monitoreo_consumo/oficinas/A/resumenes/-NqXYZ123
```

Flujo de Datos en Ejecución

mermaid

```
sequenceDiagram
    participant P as Publisher
    participant M as Mosquitto
    participant S as Subscriber
    participant F as Firebase
    participant W as WebSocket
    participant D as Dashboard

    Note over P,D: Sistema iniciado

    loop Cada 10 segundos
        P->>M: Publicar datos sensor
        M->>S: Entregar mensaje
        S->>S: Procesar datos
        S->>S: Detectar alertas
        S->>F: Guardar en Firebase
        S->>W: Notificar cambios
        W->>D: Actualizar dashboard
        D->>D: Renderizar gráficos
    end

    loop Cada 60 segundos
        S->>S: Generar resumen
        S->>F: Guardar resumen
        S->>W: Enviar resumen
        W->>D: Actualizar resumen
    end
```

Casos de Uso Comunes

Agregar una Nueva Oficina

1. En el dashboard, ve a la sección "Gestión de Oficinas"

2. Ingresa el ID de la nueva oficina (ej: "D")
3. Click en "Agregar Oficina"
4. El Publisher comenzará a generar datos para la nueva oficina automáticamente

Cambiar Configuración

1. En el dashboard, ve a "Configuración de Parámetros"
2. Modifica los valores deseados
3. Click en "Guardar Configuración"
4. Los cambios se aplican inmediatamente a Publisher y Subscriber

Controlar Dispositivos

1. En el panel de cada oficina, usa los toggles:
   - 💡 Luces
   - ❄️ Aire Acondicionado
2. El cambio se refleja inmediatamente en el consumo

Solución de Problemas

Dashboard no carga

Verificar:

```bash
# ¿Está corriendo el servidor HTTP?
curl http://localhost:8080
# Debe devolver HTML

# ¿Está corriendo WebSocket?
curl http://localhost:8081
# Debe devolver "Upgrade Required"
```

Solución:

```bash
# Reiniciar servidores
./monitoreo.sh parar
./monitoreo.sh comenzar
```

No se ven datos en el dashboard

Verificar:

```bash
# ¿Está corriendo el Publisher?
./monitoreo.sh estado
# Debe mostrar "Servicios Go: EJECUTÁNDOSE (2 procesos)"
```

Solución:

```bash
# Reiniciar Publisher
cd mqtt/publisher
go run main.go
```

Mosquitto no inicia

Verificar:

```bash
# ¿Hay otro Mosquitto corriendo?
ps aux | grep mosquitto  # Linux/macOS
tasklist | findstr mosquitto  # Windows
```

Solución:

```bash
# Detener instancias existentes
sudo systemctl stop mosquitto  # Linux
brew services stop mosquitto   # macOS
taskkill /F /IM mosquitto.exe  # Windows

# Reiniciar con el script
./monitoreo.sh comenzar
```

Error de conexión a Firebase

Verificar:

```bash
# ¿Existen las credenciales?
ls -la credentials/firebase-credentials.json

# ¿Es válido el JSON?
cat credentials/firebase-credentials.json | jq .
```

Solución:

- Regenerar credenciales desde Firebase Console
- Verificar URL de base de datos en código

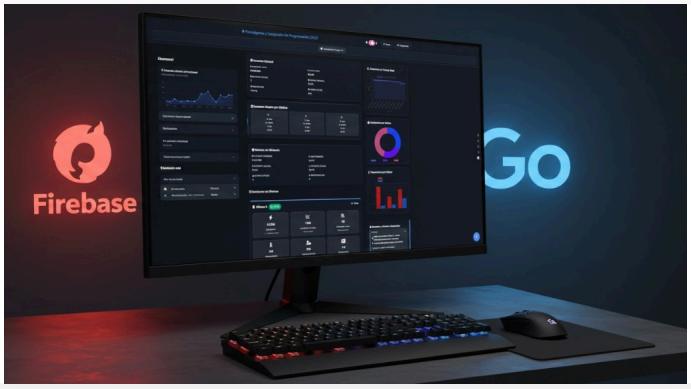
Próximos Pasos

- Uso del Dashboard - Guía detallada del panel de control
- API WebSocket - Referencia técnica de endpoints
- API MQTT - Estructura de tópicos y mensajes

Importante

Siempre usa `./monitoreo.sh parar` para detener el sistema correctamente. Esto asegura que todos los procesos se cierren limpiamente y los datos se guarden.

Previous page
Instalación



Monitoreo de Consumo Eléctrico

Sistema Distribuido en Tiempo Real

Monitoreo inteligente de consumo energético con arquitectura MQTT, WebSocket y procesamiento paralelo

Paradigmas y Lenguajes de Programación 2025

Küster Joaquín • Da Silva Marcos • Martinez Lázaro Ezequiel

Comenzar Ver en GitHub

Monitoreo en Tiempo Real

Visualización instantánea de consumo eléctrico,

Análisis Avanzado

Procesamiento paralelo con MPI para análisis de

Arquitectura Distribuida

Sistema basado en MQTT con Publisher/Subscriber en Go y backend Node.js

temperatura y presencia mediante WebSockets

eficiencia energética y clustering de consumo

Firebase Integration

Persistencia de datos en tiempo real con Firebase Realtime Database

Visualizaciones Interactivas

Gráficos dinámicos con Chart.js, ECharts y D3.js

Detección de Alertas

Sistema inteligente de detección de anomalías y generación de avisos automáticos

---

Arquitectura del Sistema

mermaid

```mermaid
graph TB
    subgraph "Capa de Sensores"
        S1[Sensor Oficina A]
        S2[Sensor Oficina B]
        S3[Sensor Oficina C]
    end

    subgraph "Capa de Mensajería"
        MQTT[Mosquitto MQTT Broker]
        PUB[Go Publisher]
        SUB[Go Subscriber]
    end

    subgraph "Capa de Procesamiento"
        MPI[MPI Parallel Processing]
        WS[WebSocket Server]
        FB[Firebase Realtime DB]
    end

    subgraph "Capa de Presentación"
        DASH[Dashboard Frontend]
        CHARTS[Visualizaciones]
    end

    S1 --> PUB
    S2 --> PUB
    S3 --> PUB
    PUB --> MQTT
    MQTT --> SUB
    SUB --> FB
```

```
    SUB --> WS
    FB --> MPI
    WS --> DASH
    DASH --> CHARTS

    style MQTT fill:#ff6b6b
    style FB fill:#4ecdc4
    style DASH fill:#95e1d3
```

Características Principales

Comunicación en Tiempo Real

El sistema utiliza **WebSockets** para transmitir datos en tiempo real desde el backend hacia el dashboard:

- `/ws/resumenes` - Resúmenes de consumo por oficina
- `/ws/avisos` - Alertas y notificaciones del sistema
- `/ws/dispositivos` - Estado de dispositivos (luces, aire acondicionado)
- `/ws/params` - Parámetros de configuración
- `/ws/oficinas` - Gestión de oficinas

Arquitectura MQTT

Utiliza el patrón **Publisher/Subscriber** con Mosquitto:

```
oficinas/{id}/sensores
```

Cada mensaje contiene:

- Timestamp
- Presencia (boolean)
- Corriente (Amperes)
- Temperatura (°C)

Inicio Rápido

bash

```
# 1. Instalar dependencias
./monitoreo.sh instalar

# 2. Iniciar el sistema completo
```

```
./monitoreo.sh comenzar

# 3. Acceder al dashboard
# http://localhost:8080
```

Componentes del Sistema

Componente	Tecnología	Puerto	Descripción
MQTT Broker	Mosquitto	1883	Broker de mensajes
Publisher	Go	-	Simulación de sensores
Subscriber	Go	-	Procesamiento de datos
WebSocket Server	Node.js	8081	Distribución en tiempo real
Dashboard	Node.js	8080	Interfaz web
Firebase	Cloud	-	Base de datos

Flujo de Datos

1. **Generación**: El Publisher simula datos de sensores cada 10 segundos
2. **Publicación**: Los datos se publican en tópicos MQTT
3. **Suscripción**: El Subscriber recibe y procesa los mensajes
4. **Análisis**: Se detectan alertas y se generan resúmenes cada 60 segundos
5. **Persistencia**: Los datos se guardan en Firebase
6. **Distribución**: WebSocket transmite datos al dashboard
7. **Visualización**: El frontend muestra gráficos en tiempo real

Próximos Pasos

> **Recomendación**
>
> Comienza con la Guía de Introducción para entender la arquitectura completa del sistema.

¿Listo para comenzar?

Explora la documentación completa y aprende a utilizar todas las características del sistema.

Ir a la Guía →

Küster Joaquín • Da Silva Marcos • Martinez Lázaro Ezequiel