Propuesta Técnica y Funcional: Plataforma de Gestión Centralizada de Aires Acondicionados

1. RESUMEN EJECUTIVO

Plataforma web centralizada para monitoreo y control de múltiples unidades de aire acondicionado de diferentes marcas y modelos, con gestión multi-dispositivo, alertas inteligentes y reportería avanzada.

---

2. ARQUITECTURA TÉCNICA

2.1 Stack Tecnológico Recomendado

Backend:

- **Node.js + Express** (API REST)
- **TypeScript** (tipado fuerte, mantenibilidad)
- **PostgreSQL** (base de datos principal)
- **Redis** (caché y gestión de sesiones en tiempo real)
- **MQTT Broker** (Eclipse Mosquitto) - comunicación con dispositivos IoT
- **Bull/BullMQ** (gestión de colas para tareas asíncronas)

Frontend:

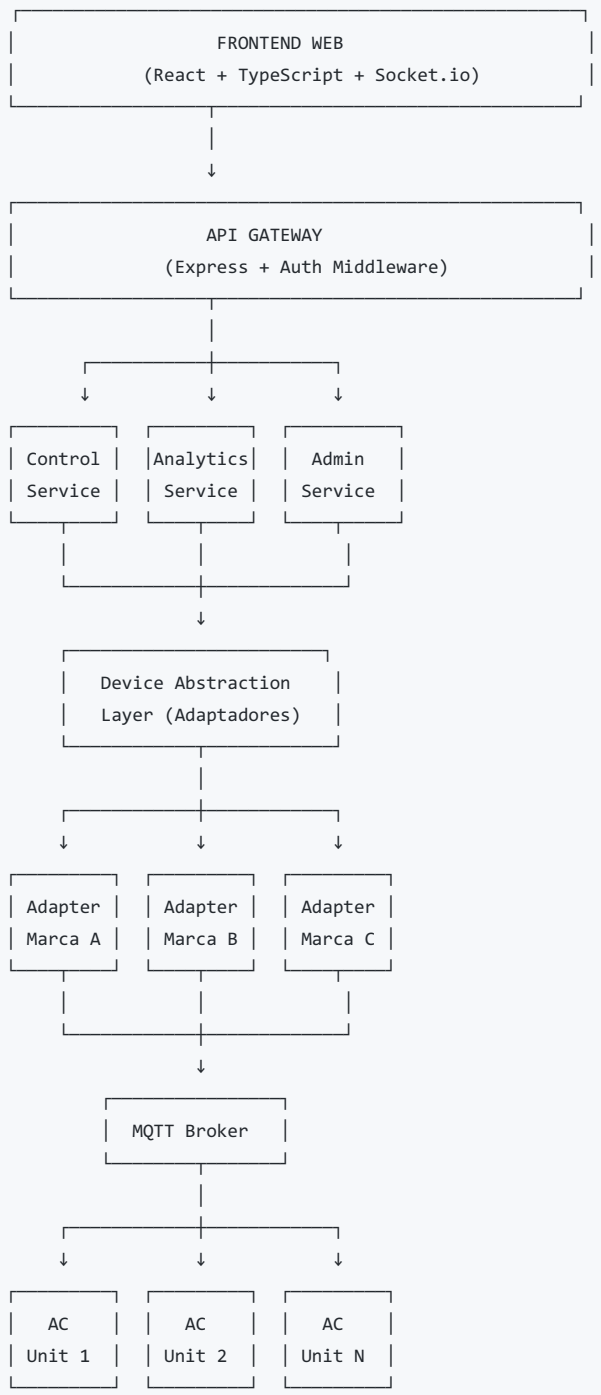
- **React 18+ con TypeScript**
- **Vite** (build rápido)
- **TailwindCSS** (UI responsivo rápido)
- **Zustand o Redux Toolkit** (estado global)
- **Socket.io Client** (actualizaciones en tiempo real)
- **React Query** (gestión de datos del servidor)

Infraestructura:

- **Docker + Docker Compose** (contenedorización)
- **Nginx** (reverse proxy)
- **PM2** (gestión de procesos Node.js)

---

3. ARQUITECTURA DEL SISTEMA



---

4. MÓDULOS FUNCIONALES

4.1 Gestión de Dispositivos

Funcionalidades:

- ✅ Registro de nuevas marcas y modelos
- ✅ Alta/baja de unidades AC
- ✅ Configuración de parámetros por marca/modelo
- ✅ Agrupación por zonas/áreas/edificios
- ✅ Estado en tiempo real (temperatura, modo, potencia)
- ✅ Comandos de control (encender, apagar, temperatura, modo, velocidad)

Interfaz:

```
Dashboard Principal
├── Mapa de ubicaciones
├── Lista de dispositivos con estados
├── Filtros (zona, estado, marca)
└── Panel de control rápido
```

4.2 Sistema de Adaptadores (Patrón Strategy)

Estructura:

```
interface ACAdapter {
  brand: string;
  model: string;
  connect(): Promise<void>;
  getStatus(): Promise<DeviceStatus>;
  setTemperature(temp: number): Promise<void>;
  setPowerState(on: boolean): Promise<void>;
  setMode(mode: ACMode): Promise<void>;
  setFanSpeed(speed: FanSpeed): Promise<void>;
}

// Ejemplo implementación
class DaikinAdapter implements ACAdapter {
  // Implementación específica para Daikin
}

class LGAdapter implements ACAdapter {
  // Implementación específica para LG
}
```

4.3 Monitoreo y Alertas

Funcionalidades:

- Dashboard con métricas en tiempo real
- Sistema de alertas configurables:
  - Temperatura fuera de rango
  - Fallas de comunicación
  - Consumo anómalo
  - Mantenimiento preventivo
- Histórico de datos (temperatura, consumo)
- KPIs: disponibilidad, eficiencia, consumo

4.4 Gestión de Usuarios y Permisos

Roles:

- **Super Admin**: Acceso total
- **Administrador**: Gestión de dispositivos y usuarios
- **Operador**: Control de dispositivos
- **Visualizador**: Solo lectura

Permisos por zona/área

4.5 Automatización y Horarios

- Programación de encendido/apagado
- Perfiles de temperatura por horario
- Modos automáticos (ahorro energético, confort)
- Calendario de mantenimiento

4.6 Reportes y Analítica

- Consumo energético
- Tendencias de temperatura
- Tiempo de operación por unidad
- Estimación de costos
- Exportación (PDF, Excel, CSV)

---

5. MODELO DE DATOS

5.1 Entidades Principales

```
 -- Marcas y Modelos
CREATE TABLE brands (
  id UUID PRIMARY KEY,
  name VARCHAR(100) UNIQUE NOT NULL,
  created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE models (
  id UUID PRIMARY KEY,
  brand_id UUID REFERENCES brands(id),
  name VARCHAR(100) NOT NULL,
  protocol_type VARCHAR(50), -- MQTT, HTTP, Modbus, etc.
  connection_config JSONB, -- Configuración específica
  capabilities JSONB, -- {hasHumidity, hasTimer, etc}
  created_at TIMESTAMP DEFAULT NOW()
);

-- Ubicaciones
CREATE TABLE locations (
  id UUID PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  parent_id UUID REFERENCES locations(id),
  type VARCHAR(50), -- building, floor, room
  created_at TIMESTAMP DEFAULT NOW()
);

-- Dispositivos
CREATE TABLE devices (
  id UUID PRIMARY KEY,
  model_id UUID REFERENCES models(id),
  location_id UUID REFERENCES locations(id),
  serial_number VARCHAR(100) UNIQUE,
  name VARCHAR(100) NOT NULL,
  ip_address INET,
  mqtt_topic VARCHAR(200),
  status VARCHAR(50) DEFAULT 'offline',
  last_seen TIMESTAMP,
  config JSONB,
  created_at TIMESTAMP DEFAULT NOW()
);

-- Estado en tiempo real
CREATE TABLE device_status (
  id UUID PRIMARY KEY,
  device_id UUID REFERENCES devices(id),
  temperature DECIMAL(4,1),
  target_temperature DECIMAL(4,1),
  humidity DECIMAL(4,1),
  mode VARCHAR(50), -- cool, heat, fan, dry, auto
  fan_speed VARCHAR(50),
  power_state BOOLEAN,
  error_code VARCHAR(50),
  timestamp TIMESTAMP DEFAULT NOW()
);

-- Histórico de datos (particionado por tiempo)
CREATE TABLE device_telemetry (
  device_id UUID NOT NULL,
  temperature DECIMAL(4,1),
  humidity DECIMAL(4,1),
```

```
  power_consumption DECIMAL(10,2),
  timestamp TIMESTAMP NOT NULL
) PARTITION BY RANGE (timestamp);

-- Comandos
CREATE TABLE device_commands (
  id UUID PRIMARY KEY,
  device_id UUID REFERENCES devices(id),
  user_id UUID REFERENCES users(id),
  command_type VARCHAR(50),
  parameters JSONB,
  status VARCHAR(50), -- pending, executing, completed, failed
  executed_at TIMESTAMP,
  created_at TIMESTAMP DEFAULT NOW()
);

-- Alertas
CREATE TABLE alerts (
  id UUID PRIMARY KEY,
  device_id UUID REFERENCES devices(id),
  alert_type VARCHAR(100),
  severity VARCHAR(50), -- info, warning, critical
  message TEXT,
  acknowledged BOOLEAN DEFAULT FALSE,
  acknowledged_by UUID REFERENCES users(id),
  created_at TIMESTAMP DEFAULT NOW()
);

-- Horarios
CREATE TABLE schedules (
  id UUID PRIMARY KEY,
  device_id UUID REFERENCES devices(id),
  name VARCHAR(100),
  enabled BOOLEAN DEFAULT TRUE,
  schedule_config JSONB, -- cron expression, días, acciones
  created_at TIMESTAMP DEFAULT NOW()
);

-- Usuarios
CREATE TABLE users (
  id UUID PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  full_name VARCHAR(200),
  role VARCHAR(50),
  permissions JSONB,
  created_at TIMESTAMP DEFAULT NOW()
);
```

---

6. PLAN DE IMPLEMENTACIÓN (4 SEMANAS)

Semana 1: Fundamentos

- ✅ Setup de infraestructura (Docker, DB, MQTT)
- ✅ Autenticación y gestión de usuarios
- ✅ CRUD básico de marcas, modelos y dispositivos
- ✅ Dashboard principal con lista de dispositivos

Semana 2: Integración y Control

- ✅ Implementación del Device Abstraction Layer
- ✅ Adaptador genérico MQTT
- ✅ Conexión en tiempo real con Socket.io
- ✅ Panel de control de dispositivos
- ✅ Comandos básicos (on/off, temperatura)

Semana 3: Monitoreo y Alertas

- □ Sistema de telemetría y almacenamiento histórico
- □ Dashboard de monitoreo en tiempo real
- □ Sistema de alertas
- □ Notificaciones (email, en app)
- □ Gráficos de históricos

Semana 4: Funcionalidades Avanzadas

- □ Sistema de horarios y automatización
- □ Reportes y exportación
- □ Optimizaciones de rendimiento
- □ Testing y corrección de bugs
- □ Documentación y capacitación

---

7. INTEGRACIÓN CON DISPOSITIVOS

7.1 Protocolos Soportados (Prioridad)

1. **MQTT** (más común en IoT)
2. **HTTP/REST API** (marcas con APIs web)
3. **Modbus TCP** (sistemas industriales)
4. **BACnet** (edificios inteligentes)

7.2 Adaptador Genérico Configurable

```
// Configuración por modelo
{
  "brand": "Generic",
  "model": "MQTT-AC-01",
  "protocol": "mqtt",
  "connection": {
    "broker": "mqtt://localhost:1883",
    "topicPrefix": "ac/{deviceId}",
    "topics": {
      "status": "ac/{deviceId}/status",
      "command": "ac/{deviceId}/command",
      "telemetry": "ac/{deviceId}/telemetry"
    }
  },
  "mappings": {
    "temperature": "payload.temp",
    "humidity": "payload.hum",
    "mode": "payload.mode"
  },
  "commands": {
    "setPower": {
      "topic": "ac/{deviceId}/command",
      "payload": {"action": "power", "value": "{value}"}
    }
  }
}
```

---

8. SEGURIDAD

- □ Autenticación JWT con refresh tokens
- □ Encriptación de contraseñas (bcrypt)
- □ Rate limiting en API
- □ Comunicación MQTT con TLS
- □ Logs de auditoría
- □ Validación de inputs (Joi/Zod)
- □ CORS configurado

---

9. ESCALABILIDAD

Para futuro crecimiento:

- Arquitectura de microservicios modular
- Cache Redis para consultas frecuentes
- Particionamiento de tablas por fecha
- Índices optimizados en PostgreSQL
- Load balancing con Nginx
- WebSockets para actualizaciones en tiempo real

---

10. ENTREGABLES

1. ✅ **Código fuente completo** (GitHub/GitLab)
2. ✅ **Documentación técnica** (API, arquitectura)
3. ✅ **Manual de usuario**
4. ✅ **Scripts de despliegue** (Docker Compose)
5. ✅ **Base de datos con datos de prueba**
6. ✅ **Guía de integración de nuevas marcas**

---

11. RIESGOS Y MITIGACIONES

Riesgo	Probabilidad	Impacto	Mitigación
Incompatibilidad con marcas específicas	Alta	Alto	Capa de abstracción flexible, adaptadores configurables
Falta de documentación de APIs de fabricantes	Media	Alto	Ingeniería inversa, sniffing de red, adaptador genérico
Latencia en comunicación con dispositivos	Media	Medio	Cache, MQTT QoS, timeouts configurables
Sobrecarga con muchos dispositivos	Baja	Alto	Arquitectura escalable, optimización de queries

---

12. PRÓXIMOS PASOS

1. **Validar** requerimientos específicos del cliente
2. **Identificar** al menos 1-2 marcas/modelos prioritarios
3. **Obtener** acceso a dispositivos de prueba o documentación
4. **Configurar** ambiente de desarrollo
5. **Iniciar** sprint 1
