PM130 (medidor SATEC) ←RS-485→ Arduino Opta (gateway Modbus Master)

Arduino Opta (servidor Modbus TCP) ←Ethernet→ Servidor Python (cliente Modbus TCP)

Servidor Python → API / almacenamiento local (opcional DB)

Servidor Python → Frontend web (React/Angular, o dashboard básico)

🔹 1. Capa de dispositivos

PM130: comunica por RS-485 Modbus RTU.

Arduino Opta: actúa como maestro RTU en RS-485 para leer el PM130 y expone esos valores en Modbus TCP (puerto 502) en su interfaz Ethernet.

Así, el Python no toca RS-485 directamente, solo TCP/IP.

🔹 2. Capa de backend (servidor)

Servidor Python:

Cliente Modbus TCP con pymodbus.

Consulta al Opta los registros del PM130 (voltaje, corriente, energía, etc.).

Expone los datos como API REST con FastAPI o los guarda en una base local (ej. SQLite o PostgreSQL).

Si necesitas control (ej. setpoint para variac), el servidor escribe directamente a registros holding en el Opta vía Modbus TCP.

🔹 3. Capa de frontend

Como solo hay un backend, puedes:

Crear una app web ligera en React o Angular, que consuma la API REST del servidor Python.

O usar un dashboard directo (ej. FastAPI + Jinja2 para una interfaz básica).

Si quieres histórico → puedes graficar desde DB (ej. con Chart.js en el frontend).

🔹 4. Ventajas de este esquema

Simplicidad: solo un cliente Modbus TCP → no hay colisiones ni necesidad de MQTT/broker.

Determinismo: tú controlas el polling y las escrituras.

Facilidad de prueba: puedes usar QModMaster o mbpoll desde PC para verificar el Opta.

Escalabilidad básica: si luego necesitas más nodos, puedes mantener el mismo esquema y centralizar todo en el backend.

✅ En resumen:

Arduino Opta se encarga de traducir Modbus RTU (RS-485) del PM130 a Modbus TCP (Ethernet).

Servidor Python actúa como único cliente TCP que lee/escribe registros.

Frontend web consume la API que expone el backend en Python.

┌───────────────────┐ RS-485 (Modbus RTU) ┌─────────────────────┐

│ │◄────────────────────►│ │

│ PM130 PLUS │ │ Arduino Opta │

│ (Medidor SATEC) │ │ (Gateway/PLC) │

│ │ │ - Maestro RTU │

└───────────────────┘ │ - Servidor TCP502 │

└─────────▲──────────┘

│ Ethernet (TCP/IP)

│

▼

┌─────────────────────┐

│ Servidor Python │

│ - Cliente Modbus │

│ - FastAPI / DB │

│ - Lógica control │

└─────────▲──────────┘

│ HTTP/REST API

│

▼

┌───────────────────────────┐

│ Frontend Web (React/JS) │

│ - Dashboard │

│ - Gráficas (Chart.js) │

└───────────────────────────┘

**📌 Descripción de cada bloque:**

* **PM130 PLUS**
  + Dispositivo de medición eléctrica.
  + Habla **Modbus RTU por RS-485**.
* **Arduino Opta**
  + Hace de **maestro RTU** para leer/escribir al PM130.
  + Expone esos registros en **Modbus TCP (puerto 502)** por Ethernet.
* **Servidor Python**
  + Usa **pymodbus** como **cliente TCP** para conectarse al Opta.
  + Extrae datos, guarda históricos en DB (SQLite/Postgres), y expone una **API REST** con FastAPI.
  + También puede escribir setpoints (ej. control variac).
* **Frontend Web**
  + App React/Angular o incluso dashboard simple en FastAPI.
  + Consume los endpoints del backend (ej. /api/v1/pm130/voltaje).
  + Muestra gráficos y estado en tiempo real (con Chart.js o Recharts).

👉 Con este esquema, solo tienes **un cliente Modbus** (el servidor Python), lo cual evita colisiones y simplifica todo.  
👉 El frontend nunca habla directo con el Opta o el PM130: **siempre pasa por el backend** (mejor seguridad y control).

**🔹 Arquitectura Final**

**Nivel de campo:**

* **PM130 PLUS (SATEC)** → conectado por **RS-485 Modbus RTU** al **Arduino OPTA**.
* El **Arduino OPTA** actúa como **gateway Modbus TCP**:
  + Es **cliente Modbus RTU** (maestro) frente al PM130.
  + Es **servidor Modbus TCP** frente al backend Python.

**Nivel backend:**

* **Servidor Python**:
  + Se conecta al Arduino OPTA vía **Modbus TCP**.
  + Hace **polling** para leer variables eléctricas del PM130 (energía, voltajes, corrientes, etc.).
  + Puede escribir valores en el **variac** si lo conectas también por Modbus.
  + Exponer los datos mediante una **API REST** (Flask/FastAPI).
  + Opcional: publicar los datos a **InfluxDB** para históricos.

**Nivel frontend / monitoreo:**

* Opción 1 (**rápida y sin mucho código**): **Grafana** conectado a InfluxDB → dashboard de energía en minutos.
* Opción 2 (**más flexible y personalizado**): **Frontend en Angular/React** → consume la API Python y muestra gráficas interactivas.

**🔹 Justificación de Modbus TCP directo**

✅ **Simple**: solo necesitas que el Arduino exponga Modbus TCP, sin protocolos extra.  
✅ **Escalable**: si luego quieres agregar MQTT o InfluxDB, el backend Python puede publicarlo sin tocar el Arduino.  
✅ **Estandarizado**: Modbus TCP es soportado por casi todas las librerías industriales y SCADA.  
✅ **Seguridad y control**: backend centraliza la lógica, autenticación y reglas de negocio.