IoT Sensor API System

Abstract

This project demonstrates a seamless integration of embedded systems and web-based technology to remotely access and control real-time sensor data. Using Django REST Framework, a custom API was built to interact with sensors connected to the ESP32 microcontroller programmed in C++. This enabled reliable, two-way communication for monitoring and managing hardware over any network connection, without relying on third-party platforms.

Introduction

The goal was to develop a lightweight, cost-effective IoT system capable of sending sensor data to a proprietary API and receiving commands from it. Unlike generic IoT platforms, this initiative focuses on customisation, speed, and local control. Every component of the stack, from submitting data to secure API endpoints to controlling hardware actions such as relay switching, was designed from the ground up.

Layers	Tool/Language	Purpose
Backend	Python + Django REST	API creation, authentication, database
Firmware	Embedded C++ (ESP32)	Reading sensors, HTTP POST/GET requests
Hardware	ESP32, Sensors, Relays	Data collection and actuator control
Database	SQLite (default)	Storing sensor data, logs
Optional	Postman	API testing and documentation

Software Workflow

1. Django REST API:
    - Custom endpoints to receive sensor values via POST.
    - GET endpoints to return live data or commands.
    - Authentication middleware for safe communication.
    - Admin panel to monitor requests and logs.
2. ESP32 Embedded Code (C++):
    - Connects to WiFi and API.
    - Reads analog/digital sensor data.
    - Sends readings using HTTP POST to the API.
    - Receives control flags via GET requests to toggle outputs like relays.

Hardware Components

- ESP32 Development Board
- Sensors: (e.g., temperature, gas, sound, IR)
- Relay Module for switching AC/DC devices
- Power Supply and wiring setup

The ESP32 was chosen for its WiFi capability, GPIO flexibility, and low power consumption.

Usefulness & Application

- Home Automation: Remote control of appliances using relays.

- Environment Monitoring: Log and access temperature, gas, or motion data from anywhere.
- Industrial IoT: Lightweight gateway for machine sensors.
- Security: Sensors can trigger alerts remotely via API integrations.

The flexibility to build your own API enables full control over the data pipeline—no vendor lock-in.

Conclusion

This project successfully demonstrates a working model of a self-hosted, API-driven sensor system powered by embedded C++ and Python. It combines low-level hardware control with modern web infrastructure, creating an efficient and scalable IoT solution. The project is modular, easy to expand, and ideal for developers or organizations seeking full autonomy over their IoT stack.

Future iterations may include token-based security, mobile app access, MQTT integration, or cloud deployment using Docker or AWS.