AirSync

Air. Sync. Done.

Abstract

This project implements a smart air quality monitoring system using an ESP32 microcontroller with DHT11 sensors to measure temperature and humidity. The system features a Flask-based web interface that displays real-time environmental data and enables Over-The-Air (OTA) firmware updates as well as remote access. Key components include a SQLite database for data logging, a responsive web dashboard with alert thresholds, and secure device authentication. The solution provides remote monitoring through a web interface with interactive data visualization, while the OTA update functionality allows for seamless firmware deployment without physical access to devices. The system demonstrates effective integration of IoT hardware with web technologies, implementing security features like password authentication and encrypted sessions. Designed for both home and industrial environments, AirSync offers reliable environmental monitoring with maintenance-free operation through its updating capability, representing a complete IoT solution from sensor to user interface.

Table of Contents

Introduction

Context and Motivation

With increasing concerns about environmental conditions and indoor air quality, this project implements an IoT-based monitoring system that collects, transmits, and visualizes temperature and humidity data in real-time. Air quality directly impacts health, productivity, and equipment performance, making continuous monitoring essential for residential, industrial, and commercial environments. Traditional monitoring systems often lack remote accessibility and real-time analytics, which this project addresses by integrating embedded sensors, wireless communication, and a cloud-based dashboard.

Objectives

The primary goals of this project are:

1. **Hardware Integration:**
    - Interface DHT11 sensor with ESP32 microcontroller for accurate temperature/humidity readings.
    - Ensure robust Wi-Fi connectivity for data transmission.

2. **Software Integration:**
    - Design a Flask-based backend to process sensor data.
    - Develop an interactive frontend dashboard (HTML/CSS/JS) for real-time visualization.

3. **Data Management:**
    - Store readings in SQLite database for historical analysis.
    - Display the latest 10 records dynamically on the web interface.

4. **Security and Maintenance:**
    - Implement a login system to prevent unauthorized access.
    - Enable OTA (Over-the-Air) firmware updates for seamless maintenance.

Scope of the Project

The system covers the following functionalities:

- Real-time sensor data acquisition (temperature, humidity).
- Wireless data transmission via HTTP/MQTT protocols.
- Web-based dashboard with authentication and responsive design.
- Local database storage for audit and analysis.
- OTA updates to enhance system longevity.

Significance of the Work

The project demonstrates:

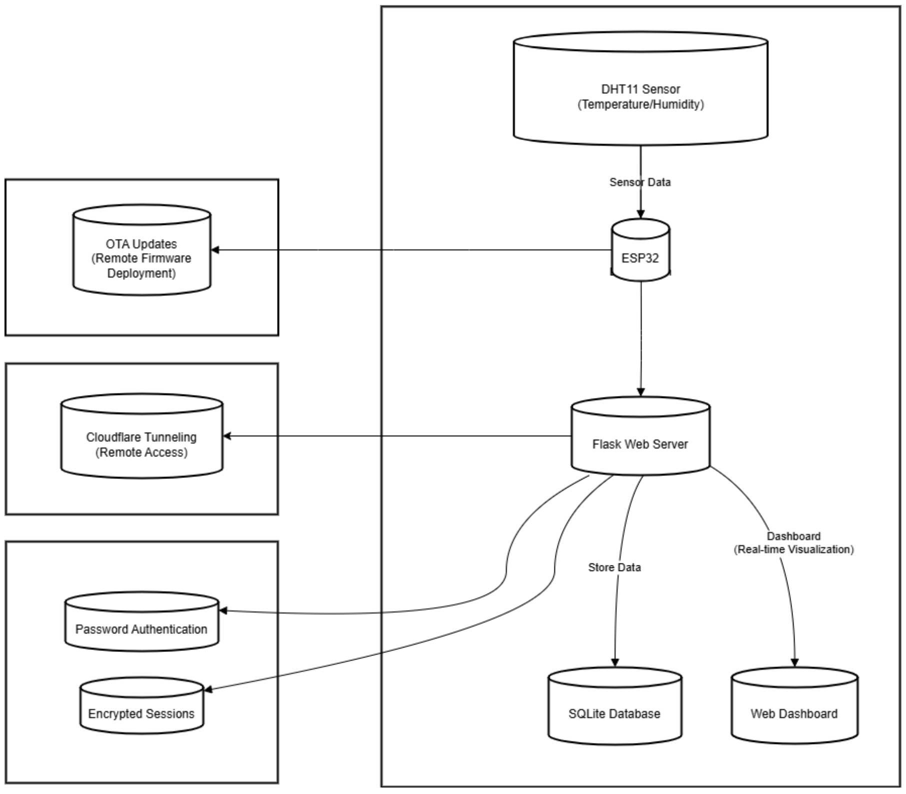
- Cost-Effectiveness: Uses affordable components (~$5 ESP32 + DHT11).
- Scalability: Modular design supports additional sensors (e.g., gas detectors).
- User-Centric Design: Intuitive dashboard for non-technical users.
- Research Potential: Provides a foundation for smart city or industrial IoT applications.

By bridging the gap between physical sensors and cloud analytics, this system offers a template for future IoT environmental monitors.

System Architecture

Overall System Design

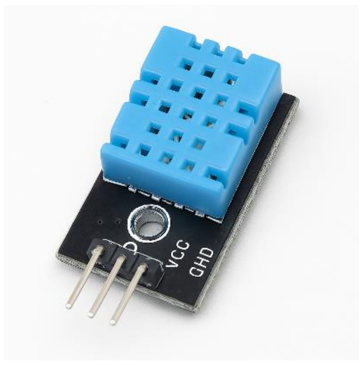
Here is an overall layer-based architecture .


DHT11 Sensor (Temperature/Humidity)
Sensor Data
ESP32
OTA Updates (Remote Firmware Deployment)
Cloudflare Tunneling (Remote Access)
Flask Web Server
Store Data
Dashboard (Real-time Visualization)
Password Authentication
Encrypted Sessions
SQLite Database
Web Dashboard

Hardware Components

1. **Hardware Integration:**
    - Wi-Fi 802.11 b/g/n + Bluetooth 4.2.
    - Dual-core 32-bit LX6 CPU (240 MHz).
    - 512 KB SRAM, 4MB Flash.
    - GPIO pins for sensor interfacing.

2. **DHT11 Sensor:**
    - Temperature Range: 0–50°C (±2°C accuracy).
    - Humidity Range: 20–90% RH (±5% accuracy).
    - Sampling Rate: 1Hz (1 reading per second).



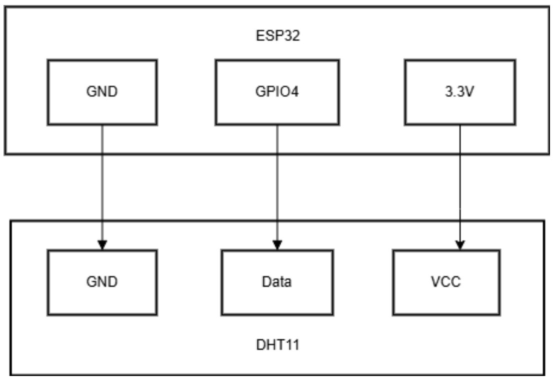
Software Components

1. **Frontend:** HTML, CSS, JavaScript
2. **Backend** Pyhton, Flask framework
3. **Database:** SQLite
4. **Firmware:** Arduino IDE (C++)

Implementation Details

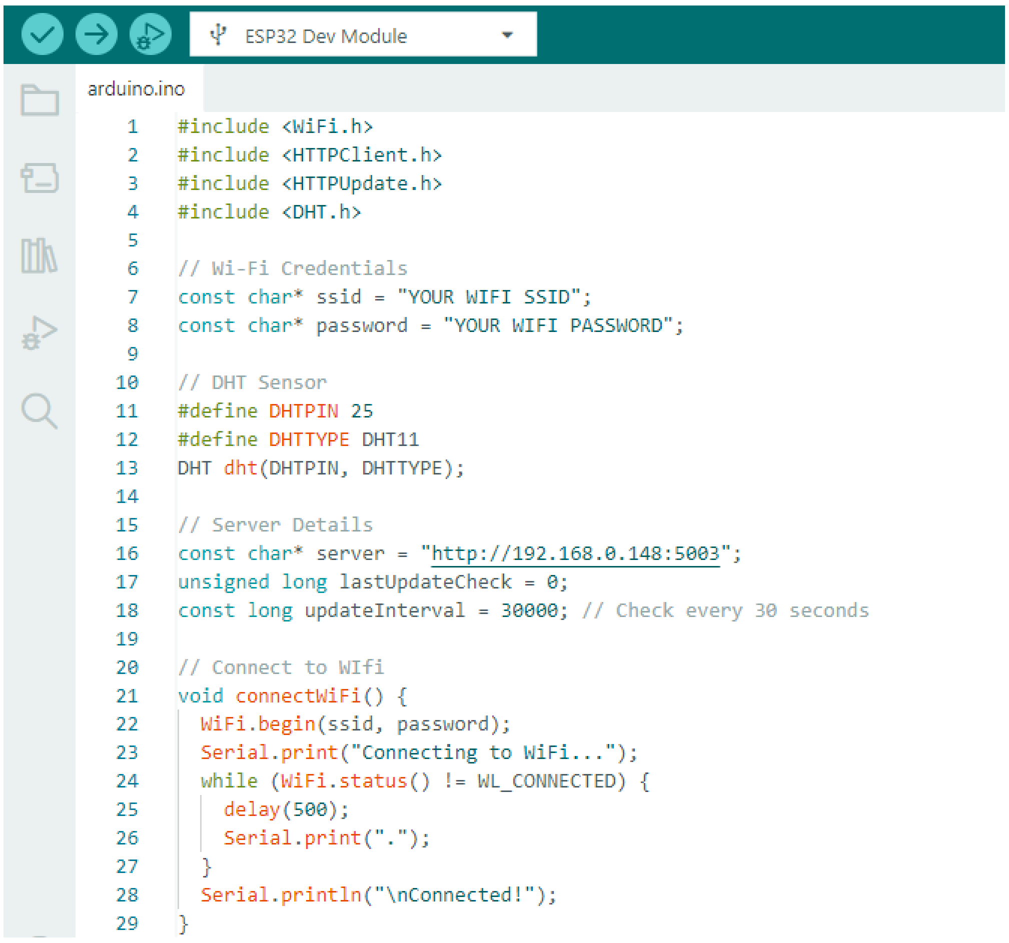
Circuit Diagram

ESP32 (3.3V logic) connected to DHT11.

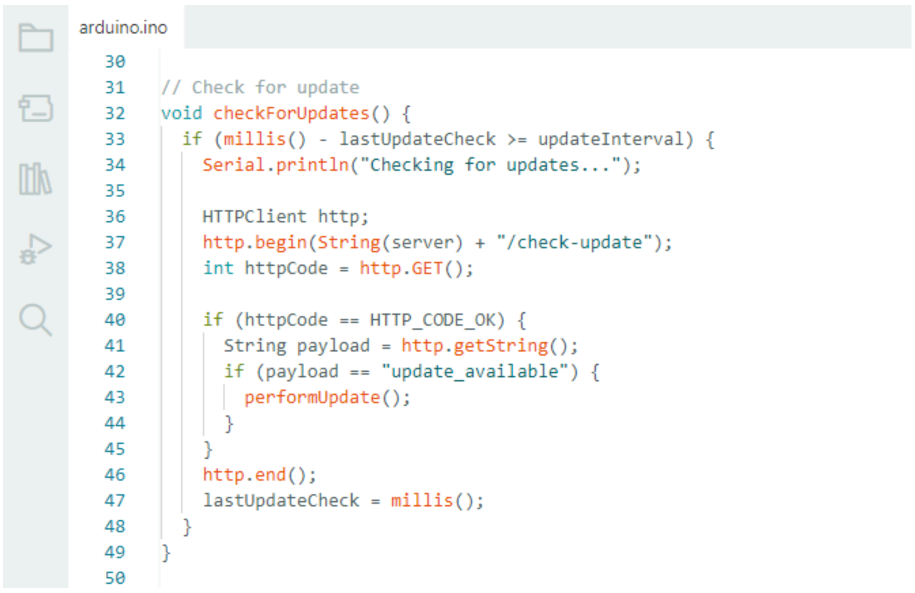


Firmware Development

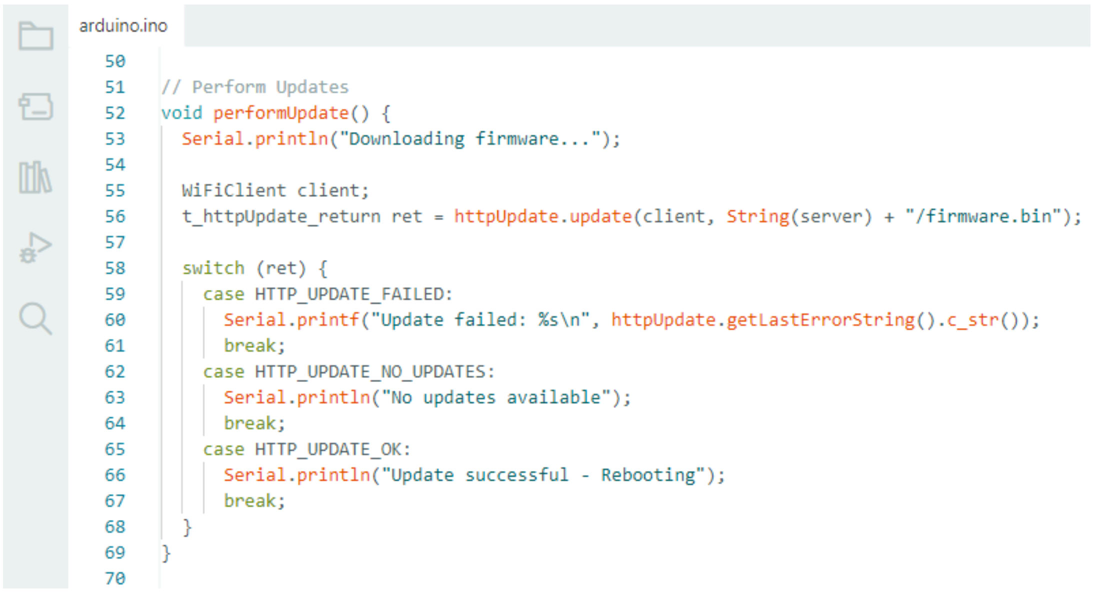
Code Structure:



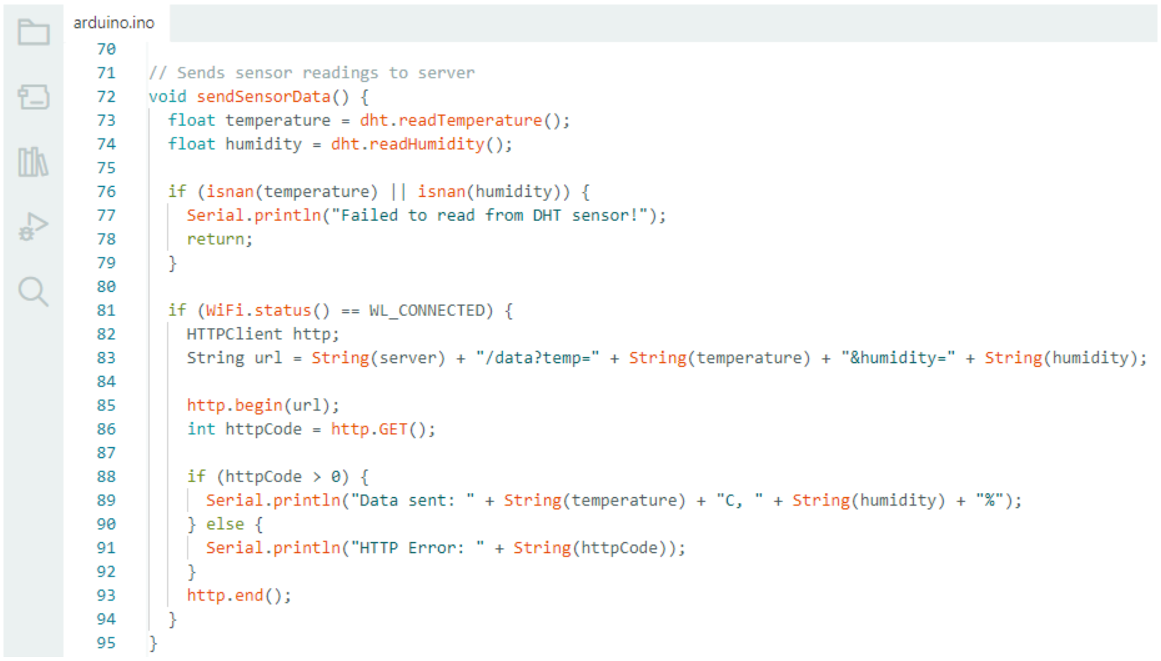
```
#include <WiFi.h>
#include <HTTPClient.h>
#include <HTTPUpdate.h>
#include <DHT.h>

// Wi-Fi Credentials
const char* ssid = "YOUR WIFI SSID";
const char* password = "YOUR WIFI PASSWORD";

// DHT Sensor
#define DHTPIN 25
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

// Server Details
const char* server = "http://192.168.0.148:5003";
unsigned long lastUpdateCheck = 0;
const long updateInterval = 30000; // Check every 30 seconds

// Connect to WIfi
void connectWiFi() {
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConnected!");
}
```



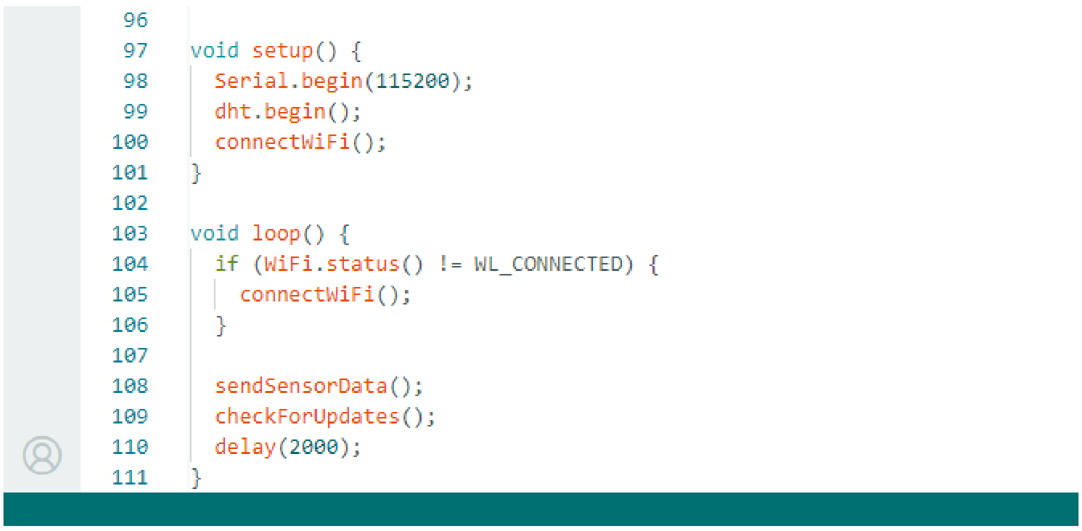
```
// Check for update
void checkForUpdates() {
  if (millis() - lastUpdateCheck >= updateInterval) {
    Serial.println("Checking for updates...");

    HTTPClient http;
    http.begin(String(server) + "/check-update");
    int httpCode = http.GET();

    if (httpCode == HTTP_CODE_OK) {
      String payload = http.getString();
      if (payload == "update_available") {
        performUpdate();
      }
    }
    http.end();
    lastUpdateCheck = millis();
  }
}
```



```
// Perform Updates
void performUpdate() {
  Serial.println("Downloading firmware...");

  WiFiClient client;
  t_httpUpdate_return ret = httpUpdate.update(client, String(server) + "/firmware.bin");

  switch (ret) {
    case HTTP_UPDATE_FAILED:
      Serial.printf("Update failed: %s\n", httpUpdate.getLastErrorString().c_str());
      break;
    case HTTP_UPDATE_NO_UPDATES:
      Serial.println("No updates available");
      break;
    case HTTP_UPDATE_OK:
      Serial.println("Update successful - Rebooting");
      break;
  }
}
```



```
// Sends sensor readings to server
void sendSensorData() {
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    String url = String(server) + "/data?temp=" + String(temperature) + "&humidity=" + String(humidity);

    http.begin(url);
    int httpCode = http.GET();

    if (httpCode > 0) {
      Serial.println("Data sent: " + String(temperature) + "C, " + String(humidity) + "%");
    } else {
      Serial.println("HTTP Error: " + String(httpCode));
    }
    http.end();
  }
}
```



```
void setup() {
  Serial.begin(115200);
  dht.begin();
  connectWiFi();
}

void loop() {
  if (WiFi.status() != WL_CONNECTED) {
    connectWiFi();
  }

  sendSensorData();
  checkForUpdates();
  delay(2000);
}
```

Libraries used:

- **WiFi.h:** Handles Wi-Fi connectivity (connection, status checks) and connects ESP32 to the specified SSID/password in connectWiFi().
- **HTTPClient.h:** Manages HTTP requests (GET/POST) and sends sensor data to the server (sendSensorData()) and checks for updates (checkForUpdates()).
- **HTTPUpdate.h:** Enables Over-The-Air (OTA) firmware updates and downloads and installs new firmware via performUpdate().
- **DHT.h:** Interfaces with the DHT11 temperature/humidity sensor and reads sensor data in sendSensorData().

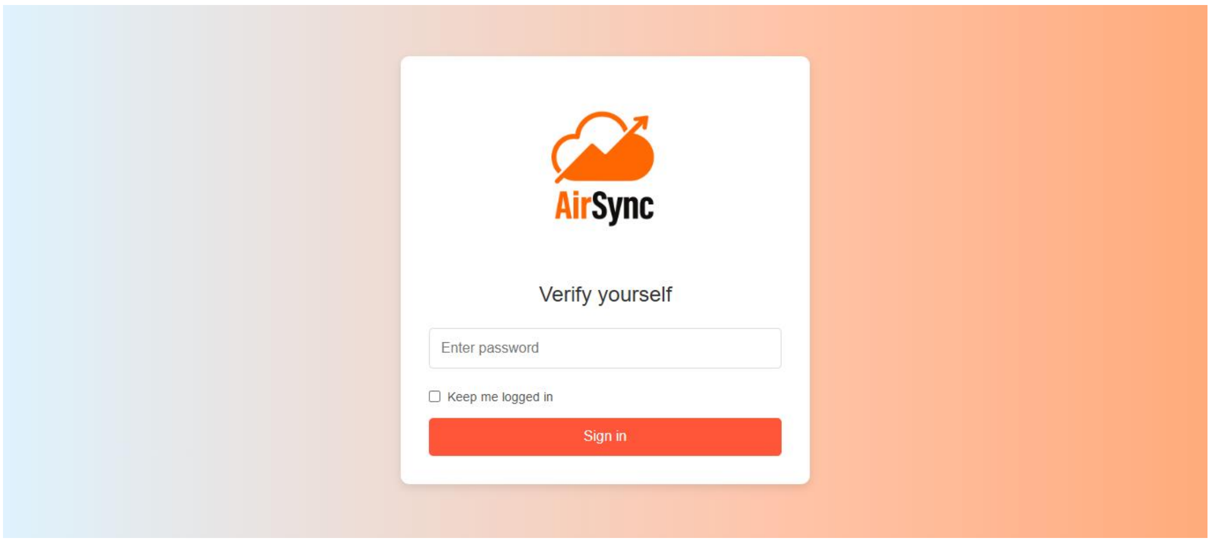
Functions Explained:

- **connectWiFi():** Establishes Wi-Fi connections and blocks until Wi-Fi is connected.
- **checkForUpdates():** Polls server for firmware updates. Triggers performUpdate() if the server responds with “update_available”.
- **performUpdate():** Executes OTA firmware update.
- **sendSensorData():** Transmits sensor readings to server
- **setup():** Starts serial communication by setting baud rate, initializes DHT11 sensor and calls connectWiFi()..

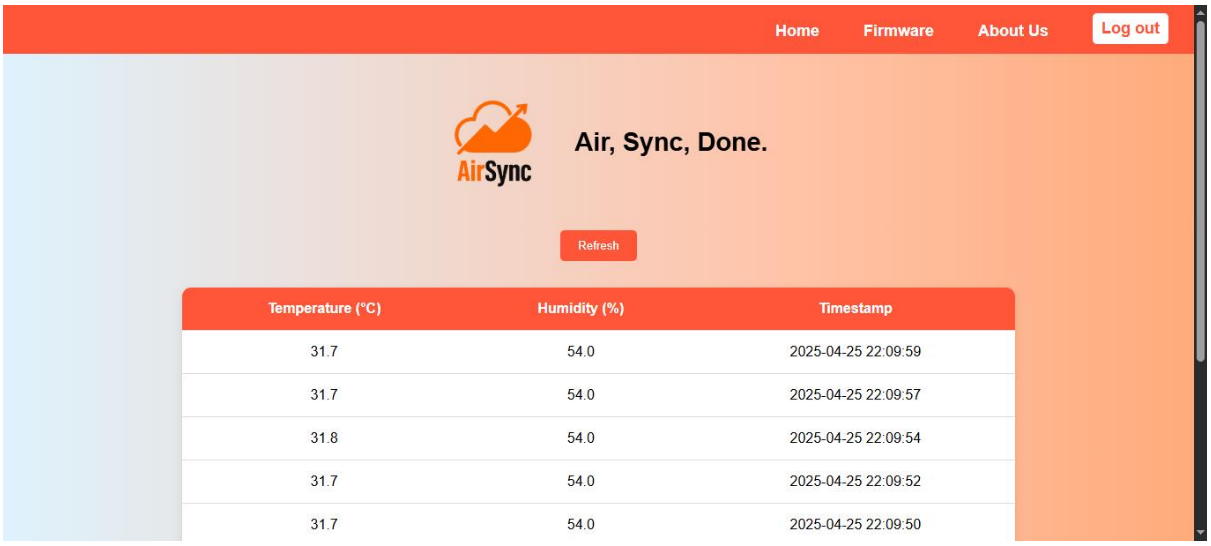
Web Application Development

The front-end is designed to provide an intuitive, responsive, and real-time user interface for monitoring sensor data and managing device firmware.

1. **Log-in page:** In order to prevent unauthorized access, a login page has been setup.

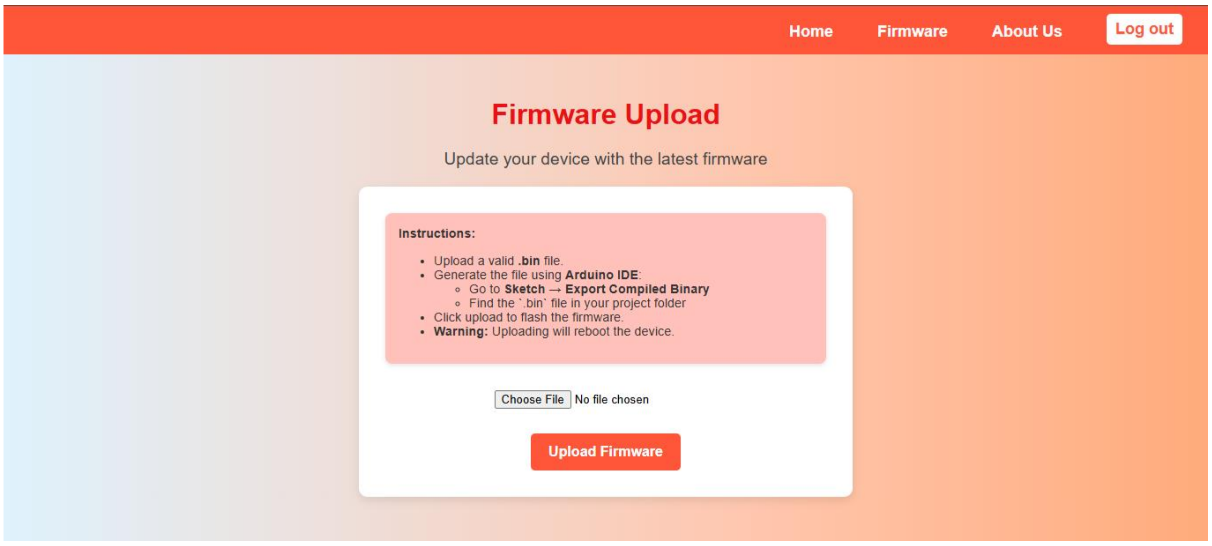


2. **Dashboard/Homepage:** The dashboard displays real-time temperature and humidity data via interactive charts and a scrollable table.



Temperature (°C)	Humidity (%)	Timestamp
31.7	54.0	2025-04-25 22:09:59
31.7	54.0	2025-04-25 22:09:57
31.8	54.0	2025-04-25 22:09:54
31.7	54.0	2025-04-25 22:09:52
31.7	54.0	2025-04-25 22:09:50

3. **Firmware upload page:** Provides a secure interface for uploading new firmware binaries (.bin files) to the ESP32, with status alerts for update progress.



Firmware Upload

Update your device with the latest firmware

Instructions:

- Upload a valid **.bin** file.
- Generate the file using **Arduino IDE**:
  - Go to **Sketch → Export Compiled Binary**
  - Find the `.bin` file in your project folder
- Click upload to flash the firmware.
- **Warning:** Uploading will reboot the device.

No file chosen

Backend API Endpoints

The backend implements a RESTful API using Python Flask to handle data processing, authentication, and firmware management.

Endpoint	Method	Description
/data	GET	Receives sensor data from ESP32
/login	POST	Authenticates users for dashboard access
/check-update	GET	Checks for available firmware updates
/firmware.bin	GET	Serves the latest firmware binary for OTA updates
/upload-firmware	POST	Accepts new firmware binaries from admin users

Database Schema Design

The system utilizes SQLite for lightweight and efficient data storage with the following schema:

```
CREATE TABLE IF NOT EXISTS readings (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        timestamp TEXT NOT NULL,
        temperature REAL NOT NULL,
        humidity REAL NOT NULL )
```

Conclusion

Limitations

- **Sensor Accuracy:** The DHT11 sensor has limited precision (±2°C for temperature, ±5% for humidity), making it unsuitable for high-precision applications.
- **Basic Authentication:** Password-based login lacks multi-factor authentication.
- **HTTP Overhead:** Frequent polling (GET requests) increases server load.
- **No Alerting System:** Users must manually check the dashboard for anomalies.
- **Limited Visualization:** Charts lack historical trend analysis (e.g., daily/weekly averages).

Potential Enhancements

- **Expanded Sensor Capabilities:** Integrate MQ-135 (CO_2/VOC detection) and PMS5003 (PM2.5/PM10) for comprehensive air quality analysis.
- **Mobile App:** Develop a mobile app with push notifications for threshold breaches and OTA update triggers.
- **Advanced Analytics:** Automated PDF reports (weekly/monthly).

References/Code

- Tunneling: https://youtu.be/BnWfbv7Fy-k?si=QH6MdCo7imOHwIGl
- GitHub: https://github.com/mehdiali7/Air-Quality-Monitoring-System-AirSync-.git