Application and Development and Emerging Technology

GreenTech: IoT-Based Smart Hydroponic System with Mobile App Integration

Ello, Jhusthine

Muripaga, Raihanie

Omandam, Richie Mae D.

Plaza, Benjie T.

Vergio, Hannah Grace D.

Department of Information Technology

College of Information Technology and Computing

University of Science and Technology of Southern Philippines

Cagayan de Oro City

May 2025

1. System Overview

The GreenTech Mobile Application that integrates backend and IoT components monitors the real-time sensor data. This system is designed for convenience that allows users to view their device connected with hydroponics through a mobile interface.

a. Frontend – Flutter:

The mobile application is developed using Flutter as frontend for cross-platform interface for responsive UI. This retrieves and displays data on the dashboard in real-time, giving the users an up-to-date overview of environmental conditions for their hydroponics. After the user logs into their account, the dashboard will display sensor data such as:

- **Temperature**
- **Humidity**

b. Backend – Fastapi:

The data from sensors is sent to the fastapi backend via HTTP requests. The fastapi then parses and stores the data on the dashboard, allowing the frontend to access it.

c. IoT Integration – ESP32 & Arduino IDE:

The ESP32 microcontroller, programmed via Arduino IDE, reads environmental data using DHT11 for both temperature and humidity. These readings are sent to the FastApi backend in real time using HTTP requests.

Hardware Components

1. **Temperature and Humidity Sensor (e.g., DHT11/DHT22)**
   - This sensor collects environmental data—specifically temperature and humidity.
2. **ESP32 Microcontroller**
   - Reads data from the sensor via GPIO pins.
   - Connects to Wi-Fi to send data to the backend over the internet.
   - Typically uses HTTP or MQTT protocols to communicate.
3. **DHT11**
   - Temperature and humidity sensor.
4. **Breadboard**
   - Used for prototyping and connecting components (sensor + ESP32) without soldering.

System Architecture

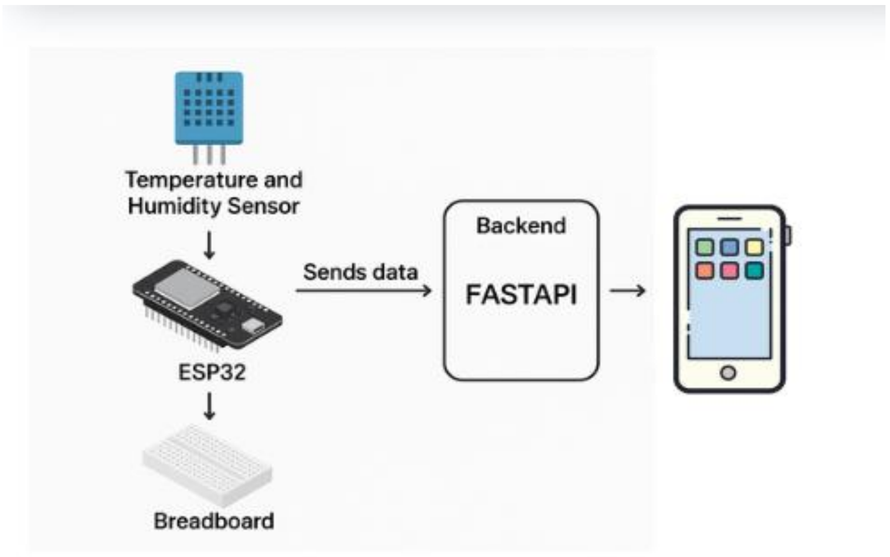

Figure 01

This system architecture shows how temperature data is collected and sent to a dashboard using a simple IoT setup. The process starts with a temperature sensor connected to an ESP32 microcontroller through a breadboard. The sensor reads the current temperature, and the ESP32 collects this data.

Once the ESP32 receives the temperature reading, it connects to Wi-Fi and sends the data to a backend server using HTTP. The backend is built with FastAPI. The FastAPI server receives the data, processes it, and makes it available for display.

This setup allows the user to monitor temperature readings remotely. From sensing the environment to sending data over the internet and storing or displaying it, the entire process is automated and efficient. The design is simple, scalable, and ideal for IoT applications.

Data Flow Summary

1. Sensor → ESP32 (reads data)
2. ESP32 → FastAPI (sends data via Wi-Fi)
3. FastAPI → Flutter (provides API to show data)
4. Flutter → User (visual display of data)

2. Development Challenges

Developing a real-time environmental monitoring mobile application using Flutter and integrate it with fastapi, Arduino-based IoT devices is the challenge itself. From hardware components to backend integration and frontend visualization.

- **Hardware Wiring Errors**

One of the most challenge encountered during the development of IoT was faulty wiring that leads to unstable connections and errors on the ESP32 board. Since the sensors require precise read and stable signal lines, a slight miswirings resulted in unreadable device through the Arduino IDE that leads to unavailability of data.

- **Real-Time Data Fetching from ESP32 to FastApi**:

Another challenge was setting up a reliable connection between the microcontroller and the FastAPI backend. Because the microcontroller has limited features, it was important to manage power usage and keep a steady Wi-Fi connection in order to send real-time data without interruption.

- **UI/UX Styling**

Although the user interface and experience were already visualized during the design phase, implementing them using Flutter presented additional complexities. While the interface needed to be visually engaging, it was equally important to maintain performance and ensure the real-time display of updated sensor data.

3. Future Improvements

To improve the system's performance, scalability, and user's experience and convenience, the following are the proposed enhancements;

- **Sensor Diagnostics**

Add a diagnostic page to monitor the health or status of each sensor, and a push notification if one of the devices are not working.

- **Offline Mode**

The system's limitations include the reliability of power consumption and internet connectivity, which means the user must have internet connections at all times for both the devices and mobile application to ensure a real-time data fetching. Enabling an offline caching in the app with auto-sync when availability of internet is present may enhance the system.