



Ben-Gurion University  
of the Negev

**The Department of Electrical and  
Computer Engineering**

**Internet of Things**

**Project Report  
36114840**

**Smart Plant Monitoring and Notification  
System Using IoT and Cloud Services**

**Omri Aviram 312192669**

**Tal Adoni 319087300**

## Table of Contents

Abstract.....	3
Introduction.....	3
Related Work and Literature Review .....	3
System Architecture .....	4
Methodology.....	4
Results .....	7
Discussion.....	8
Challenges and Solutions.....	8
Cost Considerations.....	8
Conclusions .....	9
References .....	9
Appendix .....	9

## Abstract

This paper presents the design and implementation of a Smart IoT-based Plant Monitoring and Notification System developed as part of an academic Internet of Things course. The system addresses a real-life case in which plants failed due to lack of monitoring during a long absence of the owner. An ESP32 collects soil moisture, temperature, and humidity data, performs basic local preprocessing, and securely publishes measurements to AWS IoT Core using MQTT over TLS. Cloud services handle ingestion, storage, visualization, and alerting, while a hybrid cloud-local flow synchronizes data to QuestDB via a Python bridge for Grafana-based analytics. The system supports multiple plants and cross-account data sharing, and experiments demonstrate reliable monitoring, effective alerting, and low operational cost.

## Introduction

Home plant care often relies on manual watering and visual inspection, making it vulnerable to human absence and subjective judgment. During long periods away from home, such as international travel, plants may suffer from under-watering without any feedback to the owner. This project was motivated by such a real-life scenario, where the lack of continuous monitoring led to plant failure.

The goal of this work is to design a monitoring-oriented IoT system that provides continuous visibility of a plant's condition and notifies the user when watering is required. Unlike fully autonomous irrigation systems, this project deliberately separates monitoring and notification from actuation, allowing incremental system evolution and reduced risk.

To achieve this, the system focuses on reliable sensing, secure end-to-end data transmission, and practical visualization and alerting. An ESP32-based edge device samples soil moisture, temperature, and humidity, performs local signal preprocessing and state classification for offline indication, and publishes structured data to the cloud. AWS IoT Core is used for secure device communication, while cloud services support ingestion and storage of measurements and enable alert generation. In addition, a hybrid architecture synchronizes cloud data to a local time-series database (QuestDB) using a custom Python bridge, enabling efficient dashboards and analytics using Grafana on a dedicated local machine. The resulting design supports multiple plants and multiple users, and provides a low-cost, scalable foundation for home monitoring and future expansion.

## Related Work and Literature Review

IoT-based smart agriculture systems commonly compare manual monitoring with automated or semi-automated solutions. Many existing works rely on complex hardware platforms (e.g., Arduino Mega with Raspberry Pi [1]), multiple actuators, and controlled environments.

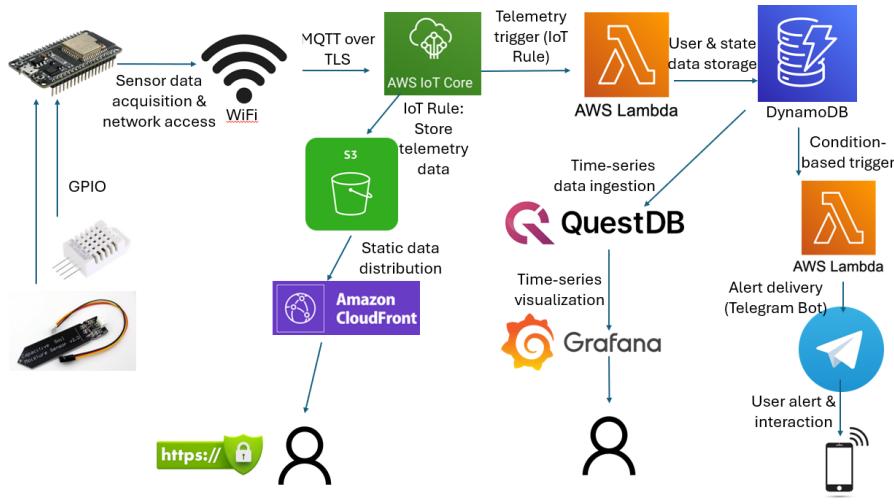
Kaur et al. (2023) presented a comparative analysis of automated and unautomated IoT-based hydroponic vertical farming systems, demonstrating that automated monitoring and control can significantly improve plant growth under controlled conditions. Their work highlights the importance of continuous sensing, cloud data collection, and visualization in improving agricultural outcomes [1].

In contrast, the system presented in this paper focuses on lightweight, low-cost monitoring, using an ESP32 microcontroller and cloud-native services, without full environmental control or actuation. This design choice emphasizes reliability, scalability, and accessibility for home users rather than industrial automation.

## System Architecture

The system follows a distributed hybrid architecture, combining cloud infrastructure with local data analytics.

### End-to-end flow:



*Fig 1: System Architecture*

The architecture is divided into the following layers:

- **Edge Layer:** ESP32 with sensors and local LED indication
- **Cloud Ingestion Layer:** AWS IoT Core and IoT Rules
- **Cloud Storage & Sharing Layer:** Amazon S3 and DynamoDB
- **Middleware Layer:** AWS Lambda + Python Bridge
- **Local Analytics Layer:** QuestDB and Grafana on a dedicated PC
- **Notification Layer:** Telegram Bot API

This hybrid approach balances cloud reliability with local analytics performance and cost efficiency.

## Methodology

### 1. Hardware Layer

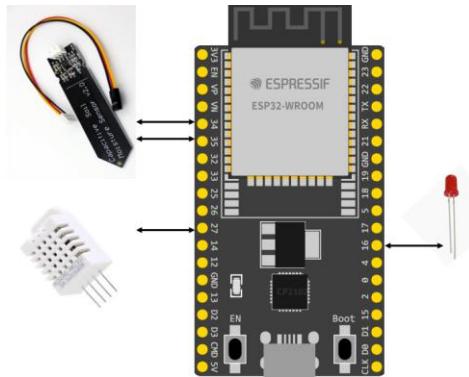
The hardware platform is based on an ESP32 DevKit with the following components:

- Two capacitive soil moisture sensors (analog)
- DHT22 temperature and humidity sensor
- LED for local status indication

- Breadboard and jumper wires

#### **Connections:**

- Soil moisture sensors → GPIO 34, GPIO 35 (ADC)
- DHT22 → GPIO 27
- LED → GPIO 16



*Fig 2: Hardware Connectivity*

The LED operates independently of cloud connectivity and provides immediate local feedback to the user.

## **2. Edge Software Logic**

The ESP32 firmware performs data acquisition and preprocessing only.

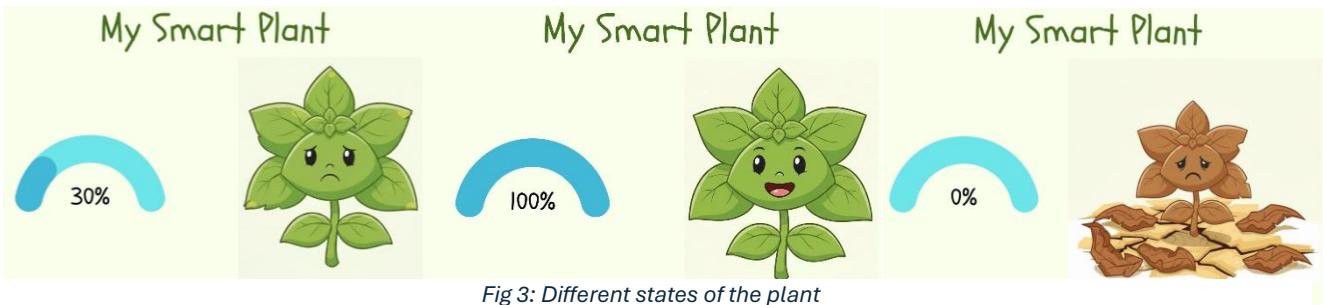
Key functions:

- High-sample averaging (~500 samples) for noise reduction
- Calibration using RAW\_AT\_0 and RAW\_AT\_100
- Sampling interval: 30 seconds
- Conversion of ADC values to soil moisture percentage (0–100%)

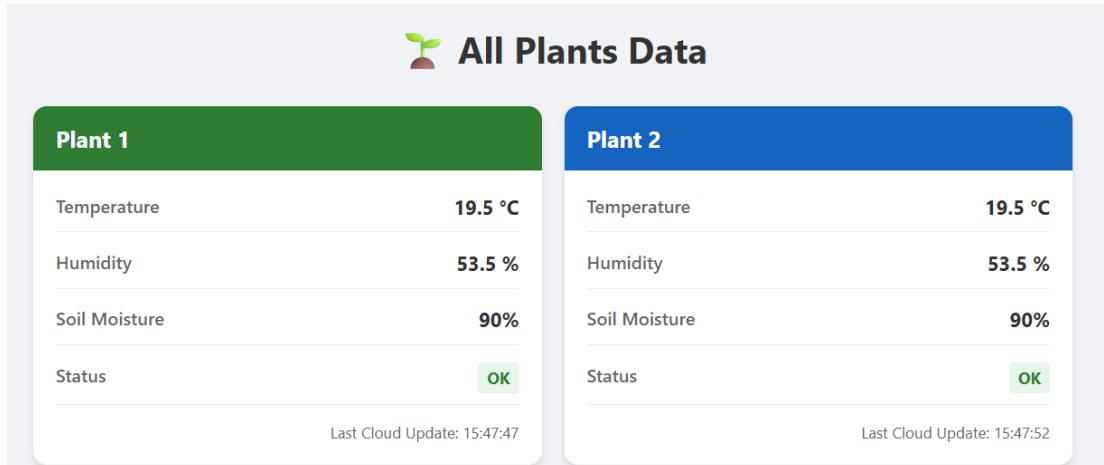
#### **Derived local states:**

- OK ( $\geq 30\%$ )
- NEEDS WATER (10–30%)
- DRY ( $< 10\%$ )

- DISCONNECTED (ADC < 700)



*Fig 3: Different states of the plant*



*Fig 4: All Plants Data Website*

The LED reflects these states:

- OFF → OK
- ON → NEEDS WATER
- BLINK → DRY

This behavior ensures usability even during WiFi or cloud outages.

### 3. Cloud Infrastructure (AWS)

The cloud backend uses multiple AWS services:

- **AWS IoT Core:** Secure MQTT ingestion using X.509 certificates
- **Amazon S3:** Storage of timestamped JSON telemetry and latest.json
- **AWS CloudFront:** Secure HTTPS access to visualization websites
- **AWS Lambda:** Cross-account data forwarding
- **Amazon DynamoDB:** Cloud storage of telemetry data and alert state management to prevent duplicate notifications and support downstream analytics.
- **Telegram Bot API:** User notifications

Two AWS accounts were used:

- **Account A:** MQTT-based device ingestion and secure web interface hosting.
- **Account B:** Data storage, data visualization, and user notification services.

#### 4. QR Code Access

To facilitate convenient access to the cloud-hosted dashboards, static QR codes were generated linking directly to each plant's interface. These QR codes provide a quick, mobile-friendly way to open the website dashboards without manually typing URLs. The QR code images are included in the project repository's Presentation Files and documented in **Appendix H**.

#### 5. Middleware and Local Analytics (Second Computer)

To enable advanced data analysis, a local analytics pipeline was implemented on a second PC via Docker containers.

Key components:

- **Python Bridge Script**
  - Reads telemetry from AWS (via DynamoDB).
  - Implements deduplication using unique PlantID + Timestamp.
  - Transmits processed data to QuestDB via Port 9000 (Ingestion Line Protocol).
  - Runs only when the PC is powered on.
- **QuestDB**
  - Time-series database running in Docker
  - Ingestion via Port 9000.
  - Exposes Port 8812 (PostgreSQL Wire Protocol) for external SQL queries.
- **Grafana**
  - Connected to QuestDB via Port 8812 using the standard PostgreSQL driver.
  - Provides real-time dashboards and historical trend analysis.

## Results

The system successfully demonstrated:

- Real-time plant monitoring via secure web dashboards
- Reliable Telegram alerts for critical moisture levels and recovery
- Offline local indication via LED
- Multi-plant and multi-user support
- Cross-account secure data sharing
- Functional local analytics using QuestDB and Grafana

## Discussion

The hybrid cloud-local architecture proved effective for balancing cost, scalability, and analytical capability. While the system does not perform autonomous irrigation, it provides timely and reliable information required for informed user intervention.

Sensor noise and wiring limitations highlighted the challenges of analog measurements in low-cost systems. Attempts to detect sensor disconnection via thresholding were partially successful but unreliable under noisy conditions, emphasizing the need for robust physical design and signal validation.

## Challenges and Solutions

Challenge	Solution
Limited GPIO / ADC resources	Use multiple ESP32 devices operating in parallel
Unstable wiring & sensor disconnection	Physical design improvements: disconnection state added but limited by noise
Noisy analog measurements	High-sample averaging and per-sensor calibration
WiFi / Cloud connectivity issues	Retry mechanisms and offline LED indication
Duplicate Telegram alerts due to multi-plant state comparison	Each plant sends data at a different time offset (5 s), preventing race conditions and incorrect cross-plant state comparison



Fig 5 : Disconected State

## Cost Considerations

The hardware components were selected to minimize cost while maintaining reliable sensing and connectivity. The total hardware cost includes an ESP32 DevKit V4 (16.48 ILS), a DHT22 temperature and humidity sensor (5.12 ILS), jumper wires (7.31 ILS), and five capacitive soil moisture sensors (16.48 ILS). Overall, the low hardware cost makes the system suitable for home and small-scale use.

Estimated monthly AWS cloud costs based on current usage are low and distributed across two AWS accounts. The primary AWS account incurs a monthly cost of approximately \$0.88, mainly attributed to AWS IoT Core and supporting services. The secondary AWS account incurs an additional monthly cost of approximately \$0.64, primarily related to AWS IoT services and auxiliary cloud components. These costs cover secure data ingestion, device communication, alert handling, and cross-account data forwarding.

The hybrid cloud-local architecture further reduces long-term expenses by offloading historical data analytics to a local computer using QuestDB and Grafana, resulting in a cost-efficient monitoring and notification system. Based on a conversion rate of 1 ILS = 0.32 USD, reflecting the exchange rate as of 01/02/2026, the total hardware cost of 45.39 ILS corresponds to \$14.52 USD [3]. When combined with the monthly AWS cloud cost of approximately \$1.52 USD across both AWS accounts, the total cost of the project for one month is approximately \$16.04 USD, while the ongoing monthly operational cost after deployment is approximately \$1.52 USD.

## Conclusions

This project demonstrates a practical, secure, and scalable IoT plant monitoring system that combines edge sensing, cloud infrastructure, and local analytics. By focusing on monitoring and notification rather than full automation, the system remains robust, affordable, and extensible.

Future work may include controlled actuation, predictive analytics, and machine learning-based watering recommendations.

## References

- [1] Kaur, G., Upadhyaya, P., & Chawla, P. (2023). *Comparative analysis of IoT-based controlled environment and uncontrolled environment plant growth monitoring system for hydroponic indoor vertical farm*. Environmental Research, 222, Article 115313. <https://doi.org/10.1016/j.envres.2023.115313>
- [2] <https://he.aliexpress.com>
- [3] <https://www.google.com/finance/>

## Appendix

### Appendix A: ESP32 Firmware

This appendix describes the ESP32 firmware responsible for sensor data acquisition, basic preprocessing, and secure data transmission.

The firmware samples soil moisture, temperature, and humidity sensors, applies averaging and calibration, classifies plant conditions into discrete states, and publishes telemetry to AWS IoT Core using MQTT over TLS.

**Listing A1:** ESP32 firmware implementation.

Source code available at:

[https://github.com/TalAdoni/IoT-Project\\_My\\_Smart\\_Plant/tree/main/ESP32%20code](https://github.com/TalAdoni/IoT-Project_My_Smart_Plant/tree/main/ESP32%20code)

## **Appendix B: Python Data Bridge (DynamoDB to QuestDB)**

This appendix presents the Python-based data bridge used to synchronize sensor measurements from Amazon DynamoDB into a local QuestDB time-series database.

The bridge supports full historical data loading and incremental updates using paginated DynamoDB scans. A lightweight deduplication mechanism based on plant ID and timestamp is applied to prevent duplicate entries before ingestion into QuestDB.

**Listing B1:** Python data bridge implementation.

Source code available at:

[https://github.com/TalAdoni/IoT-Project\\_My\\_Smart\\_Plant/blob/main/Cloud%20services%20codes/bridge.py](https://github.com/TalAdoni/IoT-Project_My_Smart_Plant/blob/main/Cloud%20services%20codes/bridge.py)

## **Appendix C: AWS Lambda for Telegram Alerts (State-Based Notification Logic)**

This appendix describes the AWS Lambda function responsible for generating user notifications based on changes in plant soil moisture conditions.

The function processes DynamoDB stream events, determines the current plant state, and sends Telegram alerts only upon detected state transitions. This mechanism prevents redundant notifications while maintaining timely user awareness.

**Listing C1:** Telegram alert Lambda implementation.

Source code available at:

[https://github.com/TalAdoni/IoT-Project\\_My\\_Smart\\_Plant/blob/main/Cloud%20services%20codes/Telegram\\_Plant\\_Alert.py](https://github.com/TalAdoni/IoT-Project_My_Smart_Plant/blob/main/Cloud%20services%20codes/Telegram_Plant_Alert.py)

## **Appendix D-Cross-Account AWS Lambda (Data Forwarding Between Accounts)**

This appendix presents the AWS Lambda function used for forwarding selected telemetry data between two AWS accounts.

The multi-account architecture separates device ingestion (Account A) from user-facing services such as visualization and notifications (Account B). Cross-account data transfer is implemented using dedicated IAM roles following the principle of least privilege.

**Listing D1:** Cross-account Lambda implementation.

Source code available at:

[https://github.com/TalAdoni/IoT-Project\\_My\\_Smart\\_Plant/blob/main/Cloud%20services%20codes/lambda\\_function.py](https://github.com/TalAdoni/IoT-Project_My_Smart_Plant/blob/main/Cloud%20services%20codes/lambda_function.py)

## **Appendix E: Website Dashboards Access**

The plant dashboards developed in this project are hosted on AWS CloudFront.

Live access is available via the following URLs:

- **Plant 1 Dashboard:**

<https://d5n0dw2hlqafq.cloudfront.net/plant1/index.html>

- **Plant 2 Dashboard:**  
<https://d5n0dw2hlgafq.cloudfront.net/plant2/index.html>
- **Combined Dashboard:**  
[https://d5n0dw2hlgafq.cloudfront.net/dashboard/all\\_plants\\_data.html](https://d5n0dw2hlgafq.cloudfront.net/dashboard/all_plants_data.html)

\***Note:** Live dashboards were hosted via AWS CloudFront during experimentation. URLs are retained for documentation but may not presently be active due to decommissioning.

#### **Appendix F: Plant Dashboard QR Codes**

To facilitate quick access to individual plant dashboards, two static QR codes were generated that link directly to the CloudFront-hosted dashboard URLs. These codes were created using free online QR generation tools and incur no ongoing cost.

QR code images are included in the “Presentation Files” directory of the repository (see **Appendix H**) and can be accessed here:

[https://github.com/TalAdoni/IoT-Project\\_My\\_Smart\\_Plant/tree/main/Presentation%20Files](https://github.com/TalAdoni/IoT-Project_My_Smart_Plant/tree/main/Presentation%20Files)

#### **Appendix G: Website Front-End Code (Site Code and Pictures)**

This appendix describes the front-end source files used for the plant dashboards.

The dashboards were implemented using HTML and fetch telemetry data (JSON) from AWS S3 to display plant metrics such as soil moisture, temperature, and humidity.

The directory contains:

- index.html (Plant 1 dashboard + Plant 2 dashboard)
- all\_plants\_data.html (combined dashboard)

The HTML files implement client-side logic to:

- Retrieve and parse the latest soil moisture, temperature, and humidity values
- Render the current plant status and images based on moisture levels
- Provide an interactive view of plant telemetry

Source code available at:

[https://github.com/TalAdoni/IoT-Project\\_My\\_Smart\\_Plant/tree/main/Site%20Code%20and%20Pictures](https://github.com/TalAdoni/IoT-Project_My_Smart_Plant/tree/main/Site%20Code%20and%20Pictures)

#### **Appendix H: Presentation Files and Supporting Media**

This appendix describes the supporting documentation and media included for the project.

The directory contains:

- Project presentation slides
- QR code images used for quick dashboard access
- System diagrams and visual assets
- Any demonstration images or videos

These files support understanding of the system architecture, data flow, and user interaction.

QR code images can be used to access the dashboards directly when the cloud infrastructure is active.

Source files available at:

[https://github.com/TalAdoni/IoT-Project\\_My\\_Smart\\_Plant/tree/main/Presentation%20Files](https://github.com/TalAdoni/IoT-Project_My_Smart_Plant/tree/main/Presentation%20Files)

### **Appendix I: System Demonstration Video**

This appendix provides a link to a short demonstration video showing the system in operation, including the plant soil condition notification service and the local LED status indication. The video captures:

- Real-time plant condition monitoring
- ESP32 LED behavior for different moisture states
- Telegram notification delivery when critical moisture levels are detected

YouTube video link:

<https://youtube.com/shorts/fw47-m6z4dE?si=jTMvf2UMQxGpYRT>

### **Appendix J: Grafana Time-Series Visualization**

This appendix describes the Grafana dashboards used to visualize plant telemetry data.

Data Source :Telemetry data collected by the ESP32 devices and stored in Amazon DynamoDB is synchronized to a local QuestDB time-series database via the Python Bridge script. Grafana is connected to QuestDB using the PostgreSQL driver.

Two main data representations are involved:

- GrafanaDashboard.json
- grafana\_query.sql

Example Panels (from QuestDB data):

Plant 1 Soil Moisture Trend: Shows gradual changes in soil moisture for Plant 1.

Link : [https://github.com/TalAdoni/IoT-Project\\_My\\_Smart\\_Plant/blob/main/GRAFANA/soil%20moisture%20plant1.png](https://github.com/TalAdoni/IoT-Project_My_Smart_Plant/blob/main/GRAFANA/soil%20moisture%20plant1.png)

- Plant 2 Soil Moisture Trend: Shows soil moisture behavior for Plant 2, including response to environmental changes.

Link : [https://github.com/TalAdoni/IoT-Project\\_My\\_Smart\\_Plant/blob/main/GRAFANA/soil%20moisture%20plant2.png](https://github.com/TalAdoni/IoT-Project_My_Smart_Plant/blob/main/GRAFANA/soil%20moisture%20plant2.png)

- Combined Dashboard View: Aggregates multiple metrics across both plants, facilitating trend comparison and correlation between sensors.

Link : [https://github.com/TalAdoni/IoT-Project\\_My\\_Smart\\_Plant/blob/main/GRAFANA/garfana%20all%20graph.png](https://github.com/TalAdoni/IoT-Project_My_Smart_Plant/blob/main/GRAFANA/garfana%20all%20graph.png)

The dashboards provide a visual summary of long-term trends and real-time data behavior, enhancing interpretability beyond static web dashboards.

Relevant configuration and data ingestion scripts (QuestDB setup and Python Bridge) are available in the project repository under “Cloud services codes”.

Source code available at:

[https://github.com/TalAdoni/IoT-Project\\_My\\_Smart\\_Plant/tree/main/GRAFANA](https://github.com/TalAdoni/IoT-Project_My_Smart_Plant/tree/main/GRAFANA)