VIETNAM-KOREA UNIVERSITY OF INFORMATION AND COMMUNICATIONS TECHNOLOGY

**FACULTY OF COMPUTER ENGINEERING AND ELECTRONICS**



**GRADUATION PROJECT**

**AIoT-Based Weather Forecasting System Using Machine Learning**

Student's Full Name : **Thai Van Hoa**

Class : **21IR**

Major : **Information Technology**

Specialization : **IoT & Robotics**

Instructor : **Ph.D. Vuong Cong Dat**

Da Nang, December 2025

VIETNAM-KOREA UNIVERSITY OF INFORMATION AND COMMUNICATIONS TECHNOLOGY

**FACULTY OF COMPUTER ENGINEERING AND ELECTRONICS**



**GRADUATION PROJECT**

**AIoT-Based Weather Forecasting System Using Machine Learning**

Student's Full Name : **Thai Van Hoa**

Class : **21IR**

Major : **Information Technology**

Specialization : **IoT & Robotics**

Instructor : **Ph.D. Vuong Cong Dat**

Da Nang, December 2025

# ACKNOWLEDGEMENTS

First of all, I would like to express my sincere gratitude to the Faculty of Computer Engineering and Electronics at the VietNam - Korean University Of Information And Communication Technology for providing favorable conditions and supporting me throughout my four years of study and research. It is thanks to the care and facilitation of the university that I have had the best learning environment to develop my knowledge and skills.

I would like to sincerely thank Ph.D. Vuong Cong Dat for his dedicated guidance and valuable feedback, which have greatly helped me complete this thesis. He has devoted significant time and effort to assist me in overcoming the challenges and difficulties during the research process, as well as imparting invaluable knowledge and practical experience.

At the same time, I would also like to extend my heartfelt thanks to my family and friends—those who have always been by my side, sharing, encouraging, and supporting me throughout the process of studying and completing this thesis. The support and encouragement from my family and friends have given me the motivation and confidence to successfully accomplish my tasks.

Inevitably, there are shortcomings and mistakes in this thesis. I sincerely hope to receive feedback from esteemed lecturers so that I can improve in the future. I also look forward to continuing to receive guidance and advice from you in my future research endeavors.

Thank you very much !

Student

Thai Van Hoa

# INSTRUCTOR’S COMMENT

**(Of the instructor)**

**………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………**

Da Nang, December 2024

Instructor

Ph.D. Vuong Cong Dat

# TABLE OF CONTENT

[ACKNOWLEDGEMENTS i](#_Toc216910215)

[INSTRUCTOR’S COMMENT ii](#_Toc216910216)

[TABLE OF CONTENT iii](#_Toc216910217)

[LIST OF FIGURE vii](#_Toc216910218)

[LIST OF TABLES vii](#_Toc216910219)

[PREAMBLE 1](#_Toc216910220)

[1. Reason for Choosing the Topic 1](#_Toc216910221)

[2. Research purpose 2](#_Toc216910222)

[CHAPTER I : OVERVIEW 3](#_Toc216910223)

[1.1 Background and Motivation 3](#_Toc216910224)

[1.1.1 Current context and practical needs 3](#_Toc216910225)

[1.1.2 Motivation for combining IoT and Machine Learning 3](#_Toc216910226)

[1.2 Problem Statement 4](#_Toc216910227)

[1.2.1 Limitations of existing solutions 4](#_Toc216910228)

[1.2.2 Key challenges addressed by this project 4](#_Toc216910229)

[1.3 Objectives 5](#_Toc216910230)

[1.3.1 System and data objectives 5](#_Toc216910231)

[1.3.2 orecasting and application objectives 5](#_Toc216910232)

[1.4 Methodology 5](#_Toc216910233)

[1.4.1 Proposed methodology 5](#_Toc216910234)

[CHAPTER II : THEORETICAL BASIS 6](#_Toc216910235)

[2.1 Internet of Things (IoT) 6](#_Toc216910236)

[2.1.1 Introduction to IoT 6](#_Toc216910237)

[2.1.2 IoT Architecture (4 layers) 6](#_Toc216910238)

[2.1.3 IoT in environmental monitoring 7](#_Toc216910239)

[2.2 Communication Protocols 8](#_Toc216910240)

[2.2.1 MQTT Protocol 8](#_Toc216910241)

[2.2.2 TCP/IP Protocol 9](#_Toc216910242)

[2.3 LoRa Technology 10](#_Toc216910243)

[2.3.1 LoRa Introduction 10](#_Toc216910244)

[2.3.2 Main Specifications 10](#_Toc216910245)

[2.3.3 Module E32-433T20D 11](#_Toc216910246)

[2.3.4 Comparison of LoRa with Wi-Fi/4G 13](#_Toc216910247)

[2.4 Machine Learning for Weather Forecasting 14](#_Toc216910248)

[2.4.1 ML Overview for Forecasting 14](#_Toc216910249)

[2.4.2 Time Series Forecasting 15](#_Toc216910250)

[2.4.3 Prophet Model 15](#_Toc216910251)

[2.4.4: LightGBM Model 17](#_Toc216910252)

[2.5 Web Technologies and Data Processing 18](#_Toc216910253)

[2.5.1 Node-RED 18](#_Toc216910254)

[2.5.2 MySQL Database 18](#_Toc216910255)

[2.5.3 FastAPI Framework 18](#_Toc216910256)

[2.6 The hardware 20](#_Toc216910257)

[CHAPTER III : SYSTEM ANALYSIS AND DESIGN 21](#_Toc216910258)

[3.1 System Overview 21](#_Toc216910259)

[3.2 Hardware connections in the system 22](#_Toc216910260)

[3.2.1 System power supply 22](#_Toc216910261)

[3.2.2 First Esp32 - Esp32 30Pin 23](#_Toc216910262)

[3.2.3 Second Esp32 - Esp32 38Pin 23](#_Toc216910263)

[3.2.4 Esp32-CAM 24](#_Toc216910264)

[3.3 Algorithm design 24](#_Toc216910265)

[3.3.1 Node Esp32 30Pin Processing Algorithm 24](#_Toc216910266)

[3.3.2 Node Esp32 38Pin Processing Algorithm 25](#_Toc216910267)

[3.3.3 Gateway - Node Processing Algorithm 26](#_Toc216910268)

[3.4 Data Stream 27](#_Toc216910269)

[3.4.1 Firebase 27](#_Toc216910270)

[3.4.2 Node-RED Flow 28](#_Toc216910271)

[CHAPTER IV : RESULT 30](#_Toc216910272)

[4.1 Proposed results 30](#_Toc216910273)

[4.2 Mobile application 31](#_Toc216910274)

[4.2.1 Monitor Screen 31](#_Toc216910275)

[4.2.2 Control Screen 32](#_Toc216910276)

[4.2.3 Images Screen 33](#_Toc216910277)

[4.3 Face recognition model 34](#_Toc216910278)

[4.4 Plant health monitoring 36](#_Toc216910279)

[4.4.1 Why are leaves green? 36](#_Toc216910280)

[4.4.2 NDVI index 37](#_Toc216910281)

[4.4.3 Monitoring plant health with NDVI index 38](#_Toc216910282)

[4.4.4 Plant health monitoring system 38](#_Toc216910283)

[4.5 System model 38](#_Toc216910284)

[4.5.1 Garden base model 38](#_Toc216910285)

[4.5.2 Upper part of the garden model 40](#_Toc216910286)

[CONCLUSION AND DEVELOPMENT DIRECTION 42](#_Toc216910287)

[1. Conclude 42](#_Toc216910288)

[2. Limitations of the topic 43](#_Toc216910289)

[3. Development direction 43](#_Toc216910290)

[REFERENCES 45](#_Toc216910291)

# 

# LIST OF FIGURE

**Figure 1.1**: MimosaTEK Soil Moisture Sensor11

**Figure 1.2**: MimosaTEK Weather monitoring station12

# LIST OF TABLES

**Table 2.1**: Data transmission mode in SPI21

# 

# PREAMBLE

## 1. Reason for Choosing the Topic

In recent years, weather and environmental conditions have become increasingly unpredictable due to climate change and rapid urbanization. Extreme heat, sudden rainfall, large humidity fluctuations, and deteriorating air quality have directly affected public health, transportation safety, and daily activities. For areas such as schools, public parks, and densely populated neighborhoods, continuous monitoring of environmental parameters and early weather forecasting is essential to reduce risks and support proactive decision-making.

However, in many locations, people often rely only on generalized online weather information, which may not accurately represent the microclimate at a specific site (e.g., inside a school campus or a public park). In addition, traditional monitoring methods if available are often manual, discontinuous, and lack long-term data storage, making it difficult to analyze trends, detect anomalies early, or provide timely warnings.

With the advancement of the Internet of Things (IoT), low-cost sensor nodes such as ESP32 enable continuous measurement of key parameters including temperature, humidity, pressure, CO₂, dust/PM2.5, and air-quality indicators. Nevertheless, monitoring alone is not sufficient; users increasingly need predictive capabilities to anticipate upcoming conditions (e.g., rainfall likelihood, temperature/humidity trends, or air-quality risks) for planning and prevention.

Meanwhile, Machine Learning has demonstrated strong performance in time-series forecasting by learning trends and seasonal patterns from historical data. When IoT sensor data is continuously collected, stored in a database (e.g., MySQL), and integrated with Weather API data, it becomes feasible to build a complete AIoT pipeline that supports both real-time monitoring and short-term forecasting with measurable accuracy.

For these reasons, I chose the graduation project **“AIoT-Based Weather Forecasting System Using Machine Learning”** to develop an integrated system for data acquisition, storage and management, model training for forecasting, and practical visualization for end users.

## 2. Research purpose

The purpose of this project is to develop an AIoT-based environmental monitoring and weather forecasting system that improves upon conventional approaches by combining real-time sensing, gateway processing, database management, and machine learning forecasting.

Specifically, the project aims to:

* **Build a real-time monitoring system** using ESP32 and environmental sensors to measure parameters such as temperature, humidity, pressure, CO₂, and dust/PM2.5 for a specific site.
* **Integrate Weather API data** (e.g., wind speed, wind direction, rainfall, UV index) to enrich the dataset and support improved forecasting performance.
* **Design a complete data pipeline** where sensor and API data are transmitted to a Raspberry Pi gateway, processed via Node-RED, and stored in a **MySQL database** for long-term analysis.
* **Develop and deploy machine learning time-series forecasting models**, trained on the collected data to produce short-term forecasts (e.g., 24 hours and 7 days), supporting early awareness of rain occurrence and environmental changes.
* **Provide a user-friendly dashboard/web interface** to display real-time readings, historical charts, forecasting results, and to support data management operations (filtering, exporting, deleting old records) for efficient system maintenance.

# CHAPTER I : OVERVIEW

## 1.1 Background and Motivation

### 1.1.1 Current context and practical needs

In recent years, climate change and rapid urbanization have made weather and environmental conditions more volatile and less predictable. Extreme heat, sudden heavy rainfall, abrupt humidity changes, and increasing fine-dust pollution negatively impact public health as well as daily activities. In sites such as **schools**, **public parks**, and densely populated areas, real-time monitoring of environmental conditions is increasingly necessary to support timely warnings and practical decisions (e.g., outdoor activity scheduling, limiting exposure to fine dust, or preparing for rain).

In practice, users often rely on general forecasting apps or websites. However, such sources usually provide region-level information and may not accurately represent the **local microclimate** at a specific site. Even within the same city, temperature, humidity, and air quality can vary depending on traffic density, vegetation, and terrain. Therefore, an on-site monitoring system can provide more relevant and accurate information.

### 1.1.2 Motivation for combining IoT and Machine Learning

The advancement of the **Internet of Things (IoT)** enables the deployment of low-cost sensor nodes such as **ESP32** for continuous, automated, and scalable data acquisition. Key parameters for environmental assessment and forecasting include **temperature**, **humidity**, **pressure**, **CO₂**, **dust/PM2.5**, and related indicators. With a gateway such as **Raspberry Pi** and data-flow processing via **Node-RED**, the collected data can be normalized and stored centrally in **MySQL** for long-term querying and analytics.

Nevertheless, real-time monitoring only reflects the “current state,” while practical users often require **forecasting** to take proactive actions: anticipating temperature/humidity changes, rainfall likelihood, or air quality trends. This is where Machine Learning becomes crucial. With sufficient historical data, ML models can learn trends, seasonal patterns, and relationships among variables to produce short-term forecasts such as **24-hour** and **7-day** predictions.

Moreover, fusing on-site sensor data with **Weather API** information (wind speed, wind direction, rainfall, UV index, etc.) enriches the dataset and can improve forecasting stability and accuracy. Therefore, this project adopts an **AIoT (AI + IoT)** approach to build a practical system that supports both monitoring and forecasting.

## 1.2 Problem Statement

### 1.2.1 Limitations of existing solutions

Common limitations in current weather/environment monitoring include:

* **Lack of on-site measurements:** users mostly rely on generalized forecasts rather than local conditions.
* **Manual and discontinuous monitoring:** sparse data makes long-term trend analysis difficult.
* **Poor data management:** data is often fragmented without centralized storage for efficient queries and reporting.
* **Limited forecasting capability:** many systems only display real-time sensor readings without predictive models.

### 1.2.2 Key challenges addressed by this project

Based on the above limitations, this project addresses key challenges:

* Building a robust architecture for **stable data acquisition** from multiple sources (sensors + API).
* Designing a **MySQL** database schema optimized for historical queries and ML training.
* Implementing preprocessing steps (unit normalization, missing-value handling, basic noise/outlier removal).
* Selecting and deploying suitable Machine Learning models for time-series forecasting.
* Presenting forecasts clearly in the user interface for practical interpretation and use.

## 1.3 Objectives

### 1.3.1 System and data objectives

* Build an ESP32-based sensing system to measure environmental parameters: temperature, humidity, pressure, CO₂, and dust/PM2.5.
* Collect additional Weather API data: wind speed, wind direction, rainfall, and UV index.
* Deploy Raspberry Pi + Node-RED as a gateway to process data streams and store data in MySQL, building structured historical datasets for analytics and training.

### 1.3.2 orecasting and application objectives

* Develop ML time-series forecasting models for key indicators (24-hour and 7-day horizons).
* Implement rainfall occurrence prediction based on historical and related variables.
* Design a web interface for real-time monitoring, historical charts, forecasts, and warning status.
* Provide data administration functions (time filtering, export, deleting old records) to support long-term stability.

## 1.4 Methodology

### 1.4.1 Proposed methodology

The methodology follows:

1. Acquire data from ESP32 sensors and Weather API.
2. Process streams in Node-RED: normalization and packaging.
3. Store in MySQL to build historical time-series datasets.
4. Train forecasting models and evaluate using metrics such as MAE and RMSE.
5. Save forecast outputs into result tables and visualize via web dashboards.

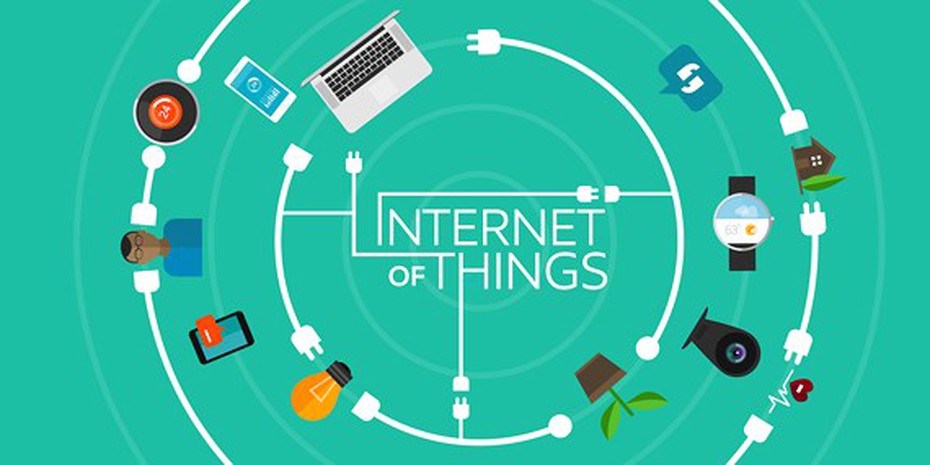
# CHAPTER II : THEORETICAL BASIS

## 2.1 Internet of Things (IoT)

### 2.1.1 Introduction to IoT

The Internet of Things (IoT) represents a paradigm shift in how physical devices interact with digital systems. IoT encompasses a network of physical objects embedded with sensors, software, and connectivity capabilities that enable them to collect, exchange, and act upon data without human intervention. According to recent estimates, the number of connected IoT devices worldwide has surpassed 15 billion and continues to grow exponentially.

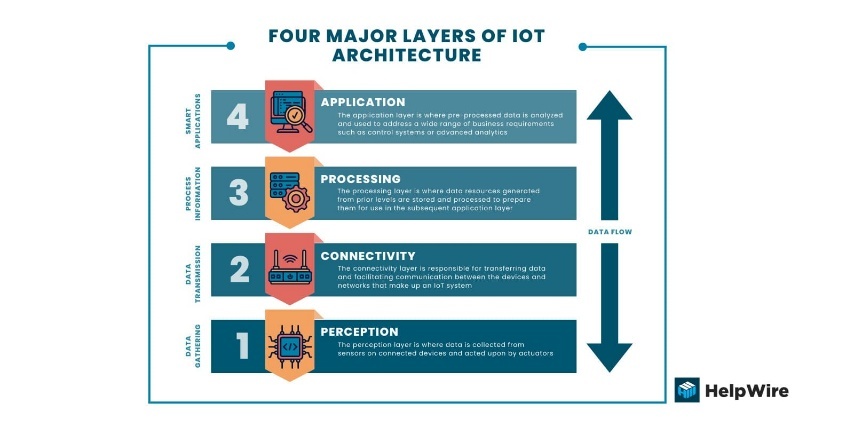
In the context of environmental monitoring, IoT technology provides unprecedented opportunities for real-time data collection, analysis, and decision-making. The fundamental concept involves deploying sensor nodes across geographical areas to continuously monitor environmental parameters such as temperature, humidity, atmospheric pressure, and air quality.



**Figure 2.1**: Internet of Things

### 2.1.2 IoT Architecture (4 layers)

A comprehensive IoT system is typically structured in four hierarchical layers, each serving distinct functions:



**1. Perception Layer (Sensing Layer)** This foundational layer consists of physical sensors and actuators that interact directly with the environment. In weather monitoring systems, this includes temperature sensors, humidity sensors, barometric pressure sensors, and other environmental monitoring devices. The perception layer is responsible for data acquisition and preliminary signal processing.

**2. Network Layer (Transmission Layer)** The network layer handles data transmission from sensing devices to processing systems. This layer employs various communication protocols and technologies such as LoRa, Wi-Fi, Zigbee, or cellular networks. The choice of technology depends on factors including transmission range, power consumption, data rate requirements, and deployment environment.

**3. Processing Layer (Middleware Layer)** This layer manages data storage, processing, and analysis. It typically includes cloud servers, databases, and computational resources that handle large volumes of sensor data. The processing layer performs functions such as data aggregation, filtering, analytics, and machine learning model execution.

**4. Application Layer** The topmost layer provides user interfaces and application-specific services. This includes web dashboards, mobile applications, alert systems, and APIs that enable end-users to access processed information and insights derived from sensor data.

### 2.1.3 IoT in environmental monitoring

IoT technology has revolutionized environmental monitoring by enabling continuous, automated, and large-scale data collection. Traditional weather stations were limited by high costs, manual data collection requirements, and sparse geographical coverage. IoT-based monitoring systems overcome these limitations through:

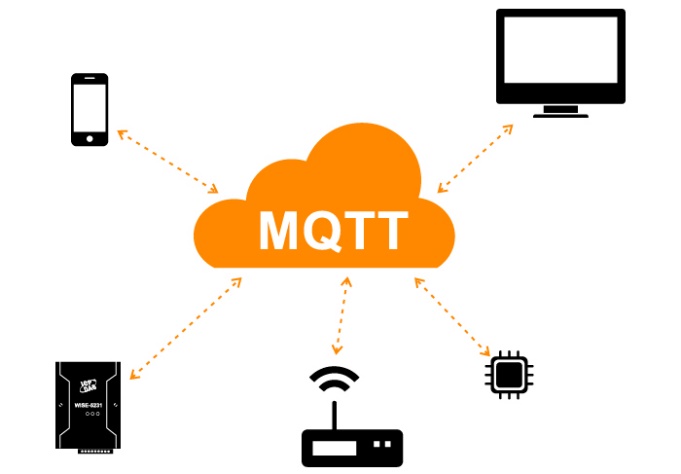
* **Distributed Sensing**: Deployment of multiple low-cost sensor nodes across wide areas provides granular spatial resolution
* **Real-time Monitoring**: Continuous data streaming enables immediate detection of environmental changes
* **Cost Efficiency**: Modern sensor technology and wireless communication reduce infrastructure costs significantly
* **Scalability**: Systems can easily expand by adding more sensor nodes without major infrastructure changes
* **Data Integration**: IoT platforms facilitate integration of multiple data sources for comprehensive environmental analysis

Applications in environmental monitoring include weather forecasting, air quality monitoring, agricultural management, disaster early warning systems, and climate research.

## 2.2 Communication Protocols

### 2.2.1 MQTT Protocol

MQTT (Message Queuing Telemetry Transport) is a message transmission protocol based on the Publish/Subscribe communication model, suitable for remote data transport. It is a very lightweight protocol therefore used to communicate devices (M2M Machine to Machine), WSN (Wireless Sensor Networks) and is most popular in IoT projects. This protocol is designed to exchange data between the server and the client. Also with its compact, simple size, low energy usage, optimized and easy-to-implement data plans make it even more ideal.



**Figure 2.3**: MQTT Protocol

**Figure 2.4:** SPI connection 1 Master – multiple Slave mechanism

### 2.2.2 TCP/IP Protocol

Transmission Control Protocol/Internet Protocol (TCP/IP) forms the foundation of Internet communication and provides the underlying transport mechanism for MQTT and other application-layer protocols.

**TCP/IP Protocol Suite Layers:**

1. **Application Layer**: Hosts protocols like HTTP, MQTT, FTP that applications use directly
2. **Transport Layer**: TCP provides reliable, connection-oriented data delivery with error checking and flow control
3. **Network Layer**: IP handles routing and addressing, enabling data packets to reach their destination across networks
4. **Link Layer**: Manages physical network connections and hardware addressing

**MQTT over TCP/IP**

MQTT operates at the application layer and relies on TCP for reliable data transmission. The typical communication flow involves:

1. Client establishes TCP connection with MQTT broker (default port 1883 or 8883 for secure connections)
2. Client sends MQTT CONNECT packet to authenticate and initialize the session
3. Broker responds with CONNACK packet confirming connection
4. Client publishes or subscribes to topics using MQTT packets transmitted over the TCP connection
5. Broker distributes messages according to subscriptions

TCP's reliability mechanisms (acknowledgments, retransmission, ordering) complement MQTT's QoS levels, ensuring dependable data delivery in IoT systems. For resource-constrained devices or networks with high packet loss, alternatives like MQTT-SN (over UDP) may be considered.

**[Suggested Image: TCP/IP layer stack with MQTT positioned at application layer, showing encapsulation]**

## 2.3 LoRa Technology

### 2.3.1 LoRa Introduction

Long Range (LoRa) is a wireless modulation technique designed for long-distance, low-power communication in IoT applications. Developed by Semtech Corporation, LoRa uses Chirp Spread Spectrum (CSS) modulation to achieve exceptional range and penetration characteristics while maintaining minimal power consumption.

LoRa operates in unlicensed ISM (Industrial, Scientific, and Medical) frequency bands, which vary by region: 868 MHz in Europe, 915 MHz in North America, and 433 MHz in Asia. This project utilizes the 433 MHz band, which offers excellent range characteristics and building penetration in the Vietnamese deployment environment.

**LoRaWAN Quick Overview**

LoRaWAN (Long Range Wide Area Network) is a network protocol built on top of LoRa physical layer technology. While this project uses point-to-point LoRa communication, LoRaWAN provides additional features for large-scale deployments:

* Network architecture with end-devices, gateways, network servers, and application servers
* Device classes (A, B, C) for different power/latency trade-offs
* Built-in security with AES-128 encryption
* Adaptive data rate (ADR) for optimization

**[Suggested Image: LoRa vs LoRaWAN comparison diagram, or LoRaWAN network architecture]**

### 2.3.2 Main Specifications

**Long Range Capability** LoRa technology achieves transmission ranges of 2-5 km in urban environments and up to 15-20 km in rural areas with line-of-sight. This extended range results from CSS modulation, which spreads the signal across a wide frequency band, making it resilient to interference and enabling the receiver to decode signals below the noise floor.

**Data Rate and Trade-offs** LoRa data rates typically range from 0.3 kbps to 50 kbps depending on configuration. The relatively low data rate is acceptable for IoT applications transmitting small data packets periodically (e.g., sensor readings every few minutes).

**Spreading Factor (SF)** Spreading Factor determines how much the signal is spread across the frequency band, ranging from SF7 (fastest, shortest range) to SF12 (slowest, longest range). Higher spreading factors improve sensitivity and range but reduce data rate and increase transmission time. For weather monitoring, SF9 or SF10 typically provides optimal balance.

**Bandwidth (BW)** LoRa supports bandwidths of 125 kHz, 250 kHz, and 500 kHz. Narrower bandwidths improve receiver sensitivity and range but reduce data rate. The 125 kHz bandwidth is most common for long-range applications.

**Link Budget** LoRa modems can achieve link budgets exceeding 155 dB, enabling reception of extremely weak signals. This is the key enabler of long-range communication with low transmission power.

**[Suggested Image: LoRa spreading factor comparison chart showing range vs data rate trade-off]**

### 2.3.3 Module E32-433T20D

The E32-433T20D is a LoRa transceiver module manufactured by Ebyte, featuring the Semtech SX1278 LoRa chip. This module is particularly suitable for weather monitoring applications due to its robust specifications and low power consumption.



**Key Specifications:**

* Frequency: 410-441 MHz (433 MHz ISM band)
* Transmission Power: 20 dBm (100 mW) maximum
* Receiver Sensitivity: -136 dBm at SF12/125kHz
* Communication Distance: Up to 3 km in urban areas, 8 km in open areas
* Interface: UART (TTL level)
* Supply Voltage: 2.3V - 5.2V
* Sleep Current: < 2 µA

**Operating Modes:** The module supports multiple operating modes to optimize power consumption:

1. **Normal Mode**: Full functionality for transmitting and receiving
2. **Wake-Up Mode**: Periodically checks for incoming messages while conserving power
3. **Power-Saving Mode**: Ultra-low power consumption with configurable wake-up periods
4. **Sleep Mode**: Minimal power consumption with wake-up on external trigger

**Battery Operation Strategy** For battery-powered sensor nodes, the module operates primarily in power-saving mode, waking periodically to collect sensor data and transmit to the gateway. With optimized duty cycling (e.g., 1-minute transmission intervals with sleep between), a node can operate for 1-2 years on battery power.

**Role in the System:**

* **Sensor Node**: E32 module transmits sensor data to the gateway
* **Gateway**: E32 module receives data from multiple sensor nodes and forwards to the server

### 2.3.4 Comparison of LoRa with Wi-Fi/4G

Different wireless technologies suit different IoT application requirements. Here's a comparative analysis:

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **LoRa** | **Wi-Fi** | **4G/LTE** |
| **Range** | 2-15 km | 50-100 m | 2-10 km |
| **Data Rate** | 0.3-50 kbps | 150+ Mbps | 1-100+ Mbps |
| **Power Consumption** | Very Low | Medium-High | High |
| **Battery Life** | Years | Days-Weeks | Days |
| **Infrastructure Cost** | Low | Medium | High (subscription) |
| **Deployment** | Private network | Requires router | Network coverage |
| **License** | Unlicensed ISM | Unlicensed | Licensed spectrum |
| **Best For** | Long-range sensors | High-bandwidth local | Mobile, high-speed |

**Why LoRa for Weather Monitoring:**

* Extended range enables sparse gateway deployment covering large areas
* Ultra-low power consumption allows long-term battery operation in remote locations
* Low infrastructure cost for private network deployment
* Sufficient data rate for periodic transmission of small sensor data packets
* No recurring subscription fees unlike cellular networks
* Excellent penetration through buildings and vegetation

## 2.4 Machine Learning for Weather Forecasting

### 2.4.1 ML Overview for Forecasting

Machine Learning (ML) has emerged as a powerful approach for weather forecasting, complementing traditional numerical weather prediction models. ML algorithms excel at identifying complex patterns in historical data and generating predictions based on learned relationships.

**Advantages of ML in Weather Forecasting:**

* **Pattern Recognition**: ML models can identify non-linear relationships between meteorological variables
* **Scalability**: Once trained, models generate predictions rapidly without intensive computation
* **Adaptability**: Models can be continuously updated with new data to improve accuracy
* **Local Optimization**: Models can be trained on local data to capture region-specific patterns
* **Multiple Variables**: ML handles multivariate time series effectively

**ML Workflow for Weather Prediction:**

1. Data Collection: Gather historical weather observations
2. Data Preprocessing: Clean, normalize, and engineer features
3. Model Training: Train algorithms on historical patterns
4. Validation: Evaluate model performance on test data
5. Deployment: Generate real-time forecasts from current conditions
6. Monitoring: Continuously assess and improve model accuracy

### 2.4.2 Time Series Forecasting

Time series forecasting involves predicting future values based on previously observed data points ordered chronologically. Weather data is inherently a time series with distinct characteristics that influence modeling approaches.

**Time Series Characteristics:**

**1. Trend** Long-term increase or decrease in the data. For example, gradual temperature changes across seasons or climate trends over years.

**2. Seasonality** Regular patterns that repeat at fixed intervals. Daily temperature cycles (warm days, cool nights) and seasonal variations (summer heat, winter cold) are classic examples in weather data.

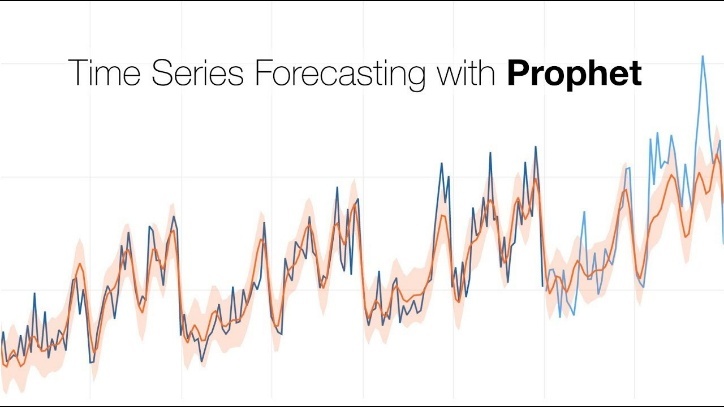
**3. Cyclicity** Patterns that repeat at irregular intervals, such as multi-year climate oscillations (El Niño/La Niña effects).

**4. Autocorrelation** Correlation of the time series with lagged versions of itself. Tomorrow's temperature is highly correlated with today's temperature.

**5. Stationarity** Statistical properties (mean, variance) remain constant over time. Many ML models assume or require stationary data, though weather data often exhibits non-stationarity due to trends and seasonality.

### 2.4.3 Prophet Model

Prophet is an open-source forecasting library developed by Facebook's Core Data Science team, designed for forecasting time series data with strong seasonal patterns and multiple seasons of historical data.



**Principle of Operation**

Prophet decomposes time series into three main components using an additive model:

**y(t) = g(t) + s(t) + h(t) + εₜ**

Where:

* **g(t)**: Trend function modeling non-periodic changes (piecewise linear or logistic growth)
* **s(t)**: Seasonal component representing periodic changes (Fourier series)
* **h(t)**: Holiday effects and irregular events
* **εₜ**: Error term representing idiosyncratic changes

**Key Features:**

**Automatic Seasonality Detection** Prophet automatically detects daily, weekly, and yearly seasonal patterns using Fourier series. For weather data, this captures diurnal temperature cycles and annual seasonal variations.

**Trend Changepoints** The model automatically identifies points where the time series trend changes, allowing for flexible trend modeling without manual intervention.

**Robust to Missing Data** Prophet handles missing values and outliers gracefully, making it suitable for real-world sensor data that may have gaps.

**Interpretable Parameters** Model components (trend, seasonality, holidays) are interpretable, enabling understanding of what drives predictions.

**Pros and Cons**

**Advantages:**

* Intuitive parameters requiring minimal tuning
* Excellent handling of seasonal patterns and trends
* Robust to missing data and outliers
* Fast training and prediction
* Uncertainty intervals included in forecasts
* Works well with relatively small datasets
* Interpretable results aid in understanding predictions

**Disadvantages:**

* Limited ability to capture complex non-linear relationships
* Assumes additive structure may not fit all data patterns
* Less effective for short-term (hourly) forecasts with high volatility
* Cannot automatically incorporate external variables (requires manual feature engineering)
* May struggle with data having irregular patterns or multiple overlapping cycles

### 2.4.4: LightGBM Model

LightGBM (Light Gradient Boosting Machine) is a gradient boosting framework developed by Microsoft that uses tree-based learning algorithms. It is highly efficient and has become popular for both classification and regression tasks, including time series forecasting.

**Gradient Boosting Principle**

Gradient Boosting builds an ensemble of decision trees sequentially, where each new tree corrects errors made by the existing ensemble.

**Mathematical Formulation:**

**F(x) = F₀(x) + Σ γₘ hₘ(x)**

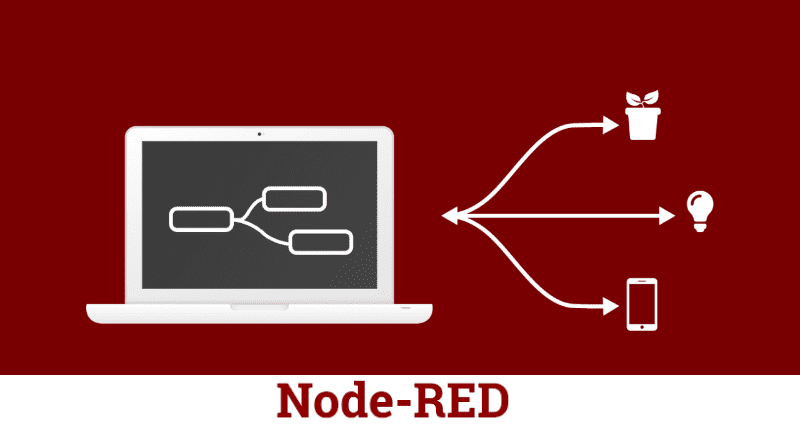
Where:

* F(x) is the final prediction
* F₀(x) is the initial prediction (baseline)
* hₘ(x) is the m-th decision tree
* γₘ is the learning rate (shrinkage parameter)

## 2.5 Web Technologies and Data Processing

### 2.5.1 Node-RED

Node-RED is a flow-based development tool originally developed by IBM for wiring together hardware devices, APIs, and online services. It provides a browser-based visual editor that makes it easy to create data processing workflows using a wide range of pre-built nodes.



**Advantages:**

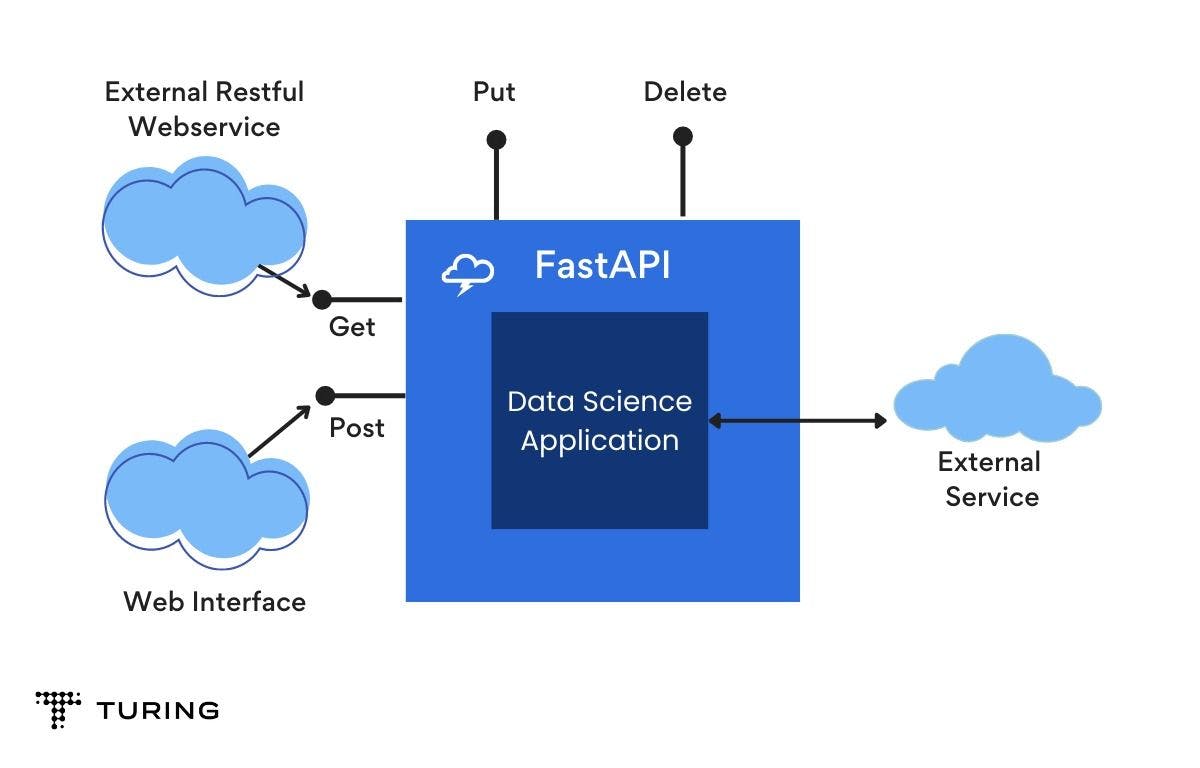
* Visual programming reduces development time
* Extensive library of pre-built nodes for common tasks
* Real-time data processing with low latency
* Easy integration of multiple protocols and services
* Built-in debugging and monitoring tools

### 2.5.2 MySQL Database

Node

### 2.5.3 FastAPI Framework

FastAPI is a modern, fast (high-performance) web framework for building APIs with Python 3.7+ based on standard Python type hints. It serves as the backend API layer connecting the database to web/mobile clients and integrating machine learning models.



**Key Features:**

**1. High Performance** FastAPI is one of the fastest Python frameworks, comparable to NodeJS and Go, built on Starlette for web parts and Pydantic for data validation.

**2. Automatic API Documentation** FastAPI automatically generates interactive API documentation (Swagger UI and ReDoc) based on OpenAPI standards, making API testing and integration straightforward.

**3. Type Safety** Python type hints enable automatic request validation, serialization, and documentation, reducing bugs and development time.

**4. Async Support** Native support for asynchronous request handling enables high concurrency for I/O-bound operations like database queries.

**API Endpoints in the Project:**

**GET /api/sensors** Returns list of all registered sensor nodes with their metadata and status.

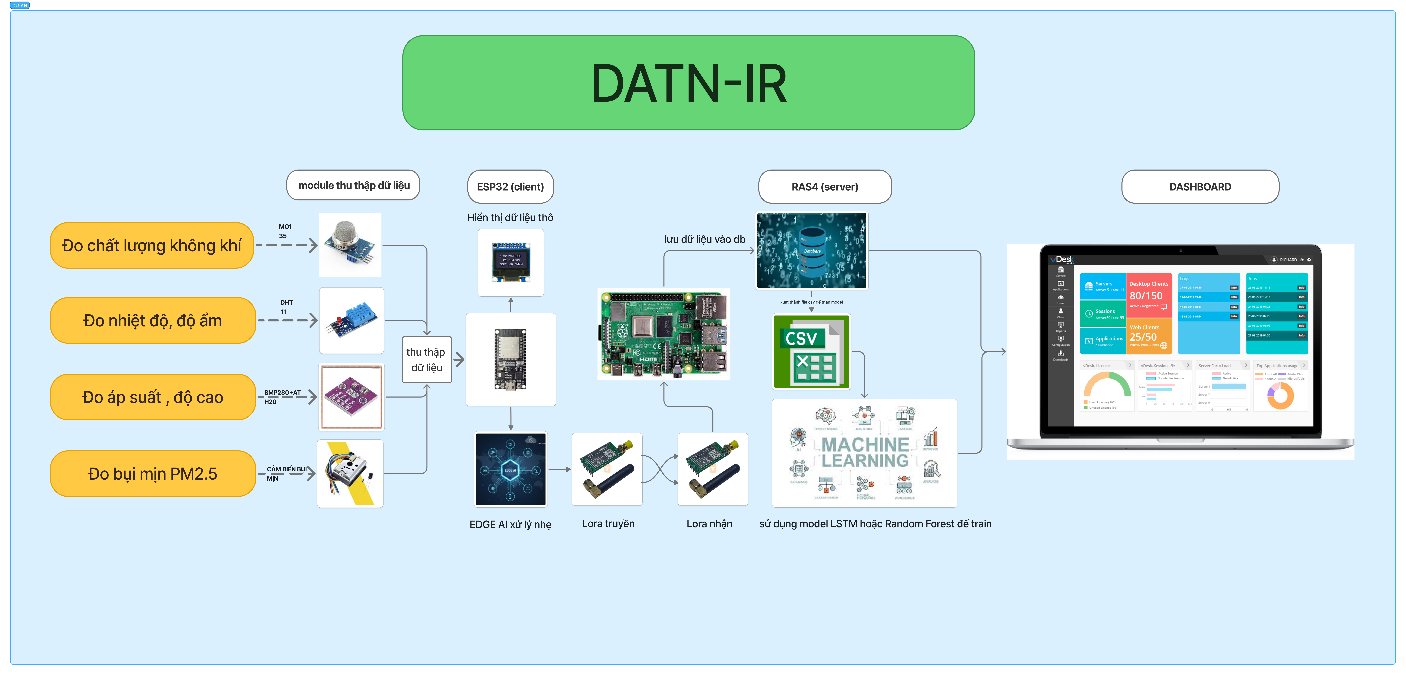
**GET /api/sensors/{sensor\_id}/data** Retrieves historical sensor readings for a specific sensor, with optional time range filtering.

**GET /api/sensors/{sensor\_id}/forecast** Returns weather forecasts for a specific sensor location, including predictions from both Prophet and LightGBM models.

**POST /api/forecast/generate** Triggers forecast generation for specified sensors, running ML models on latest data.

**GET /api/statistics** Provides aggregated statistics (min, max, average) for specified time periods.

## 2.6 The hardware



Hệ thống gồm các nút cảm biến sàn bên trái với 4 loại cảm biến chính: MQ-135 đo chất lượng không khí (khí gas, CO2 và khí độc hại), DHT22 đo nhiệt độ và độ ẩm, BMP280 đo áp suất khí quyển và độ cao, cùng GP2Y1010 đo bụi mịn PM2.5. Dữ liệu được thu thập và truyền bởi module ESP32 (vi điều khiển chính, bảng đen nhỏ) kết hợp với module LoRa E32-433T20D (tần số 433MHz, truyền nhận tầm xa) để gửi về gateway. Lớp xử lý trung tâm sử dụng Raspberry Pi 4 làm server, nhận dữ liệu qua LoRa, chạy MQTT Broker, Node-RED, MySQL và FastAPI để xử lý và lưu trữ.

# CHAPTER III : SYSTEM ANALYSIS AND DESIGN

## 3.1 System Overview

**Figure 3.1**: System model overview

Overview of the system model includes:

* The Raspberry Pi 4B, 2 ESP32 modules, and the ESP32-CAM are all connected to the same Wi-Fi network.
* The two ESP32 modules will be connected in series with the sensor modules and relays.
* The Raspberry Pi acts as an MQTT broker and also runs the Node-RED tool.
* The ESP32 devices will act as MQTT nodes, publishing and subscribing to data on the appropriate topics in the MQTT protocol.
* Node-RED running on the Raspberry Pi connects to Firebase to store the data.
* Create a control and monitoring application using React Native, which will get and set data on Firebase.

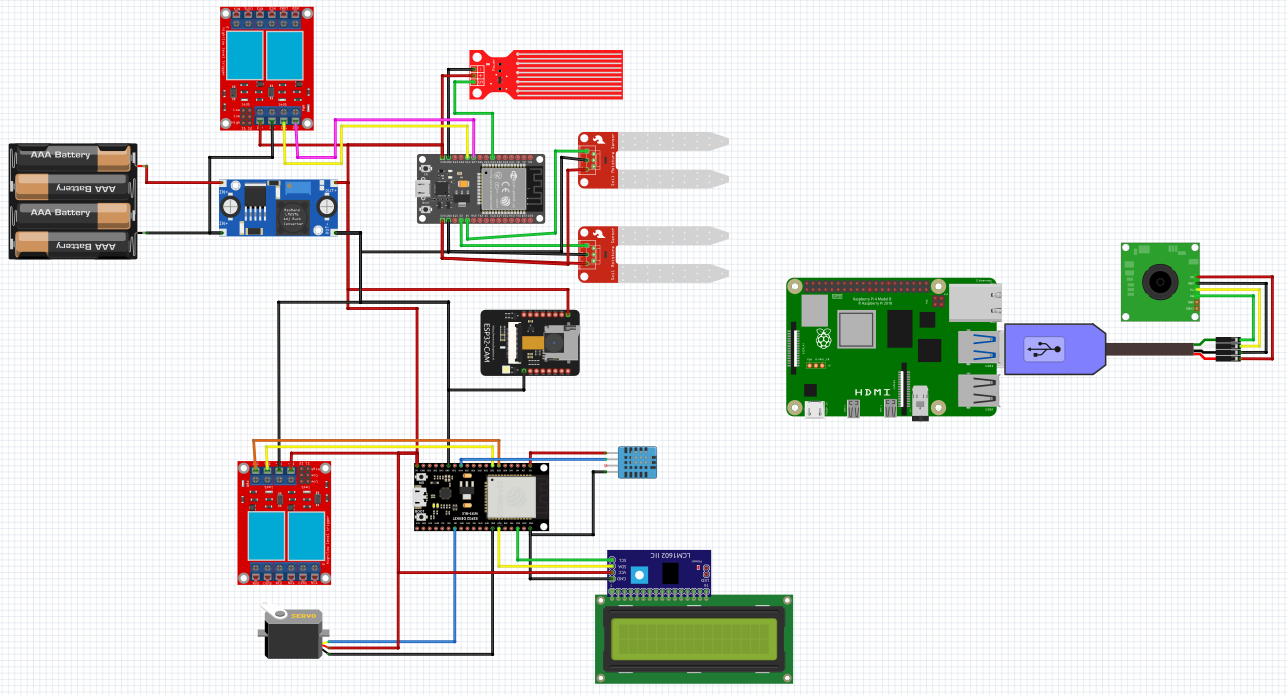
The Raspberry Pi will be the central hub or gateway of the system. It will receive data from devices connected to the ESP32 modules and send signals to control devices such as relays and servos.

The Raspberry Pi will also capture frames from the ESP32-CAM to perform face recognition using a pre-trained machine learning model. Additionally, it will analyze the NDVI index from the frames obtained by the USB camera connected to it to assess plant health.

The control and monitoring application created using the React Native framework will interact with Firebase Realtime Database.

Firebase will communicate with Node-RED. Therefore, the React Native application can easily interact with MQTT through the Firebase - Node-RED data flow.

## 3.2 Hardware connections in the system



**Figure 3.2**: Hardware connection diagram in the system

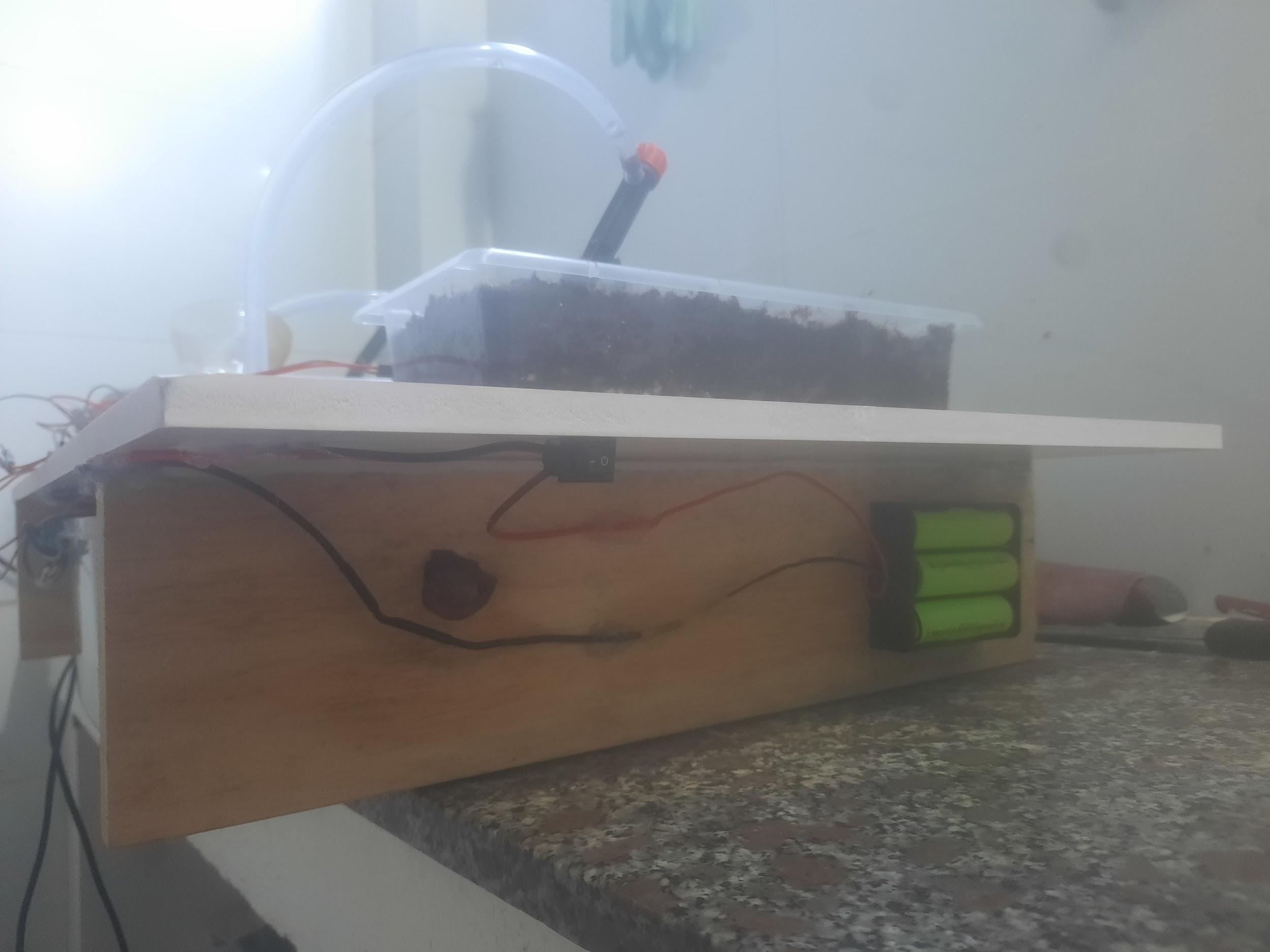
The system will include a small setup inside the garden, consisting of ESP32 nodes and devices connected to them.

The ESP32-CAM will be placed outside the garden gate, capturing frames for the Raspberry Pi to perform face recognition and open the gate using a servo.

The Raspberry Pi will be placed outside the garden, but the USB camera connected to it will be positioned appropriately for monitoring the plants.

### 3.2.1 System power supply

The system uses ~12V power with 3 18650 batteries.

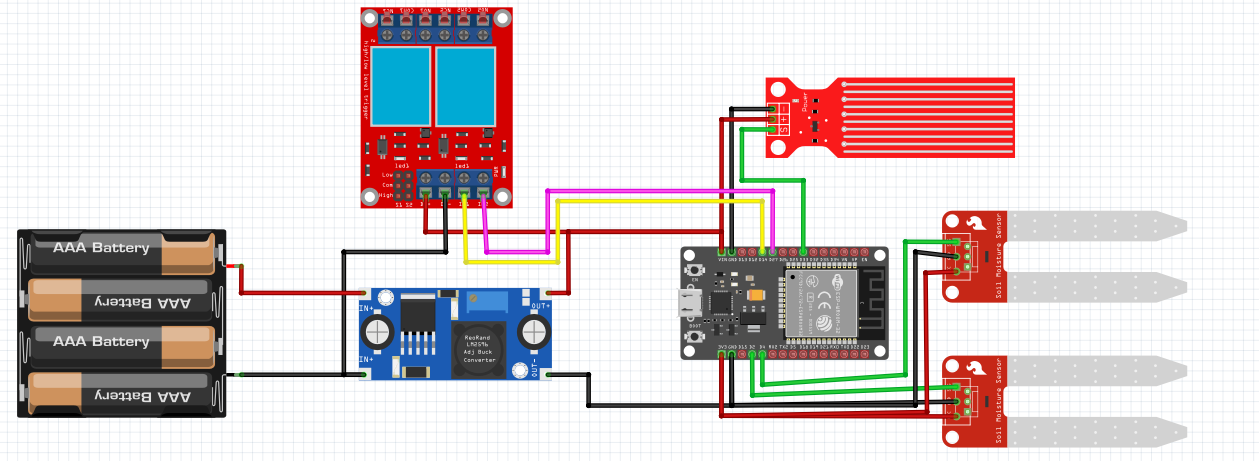


**Figure 3.3**: 12V power supply

The system uses an additional LM2596 step-down module to reduce the voltage to 5V for use with modules that operate within a 5V range.

The two ESP32 and ESP32-CAM modules will use the 5V power output from the LM2596 step-down module.

### 3.2.2 First Esp32 - Esp32 30Pin

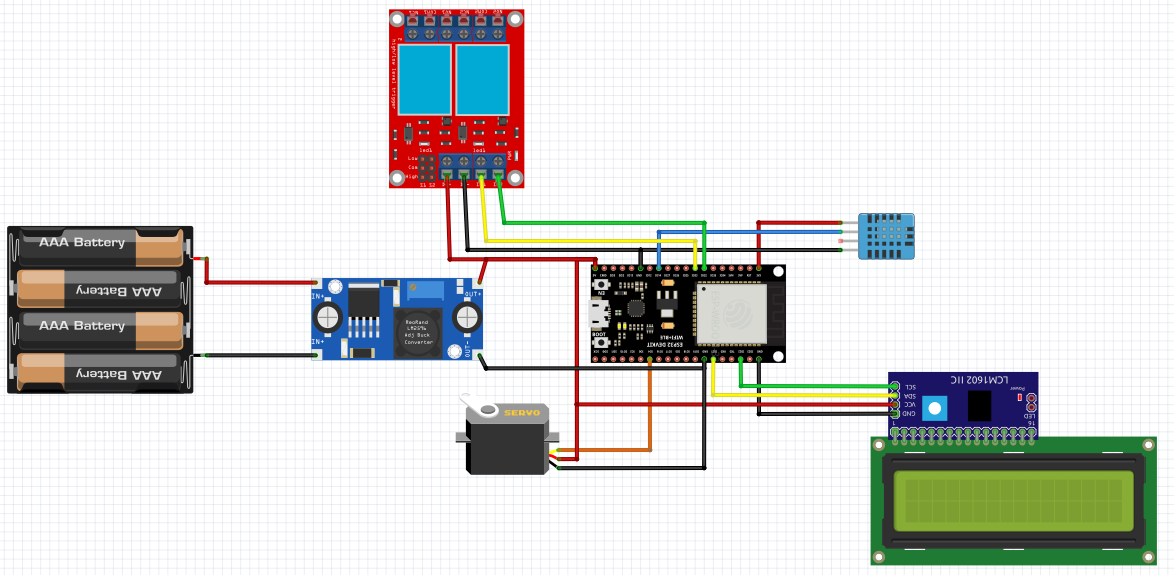


**Figure 3.4**: Esp32 30Pin connection diagram

This Esp32 30Pin will be connected to:

* A water level sensor, located under the water tank.
* Two soil moisture sensors, placed in two plant containers.
* One 2 channel relay module to control 2 submersible water pumps.

### 3.2.3 Second Esp32 - Esp32 38Pin

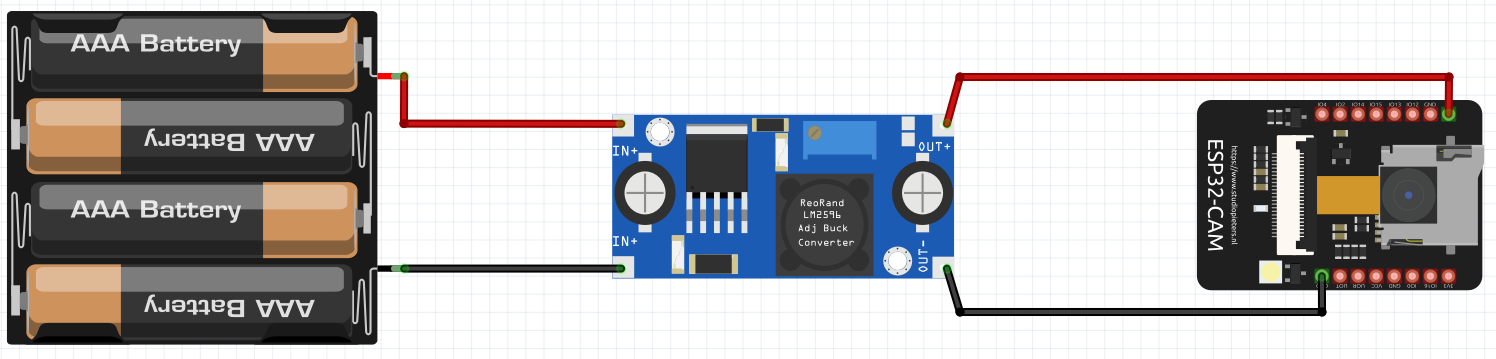


**Figure 3.5**: Esp32 38Pin connection diagram

This Esp32 38Pin will be connected to:

* A DHT11 sensor measures the temperature and humidity in the environment.
* A 16x2 LCD display connected to I2C.
* A SG90 Servo motor module to control the door.
* A 2-channel relay module to control the fan and LED.

### 3.2.4 Esp32-CAM

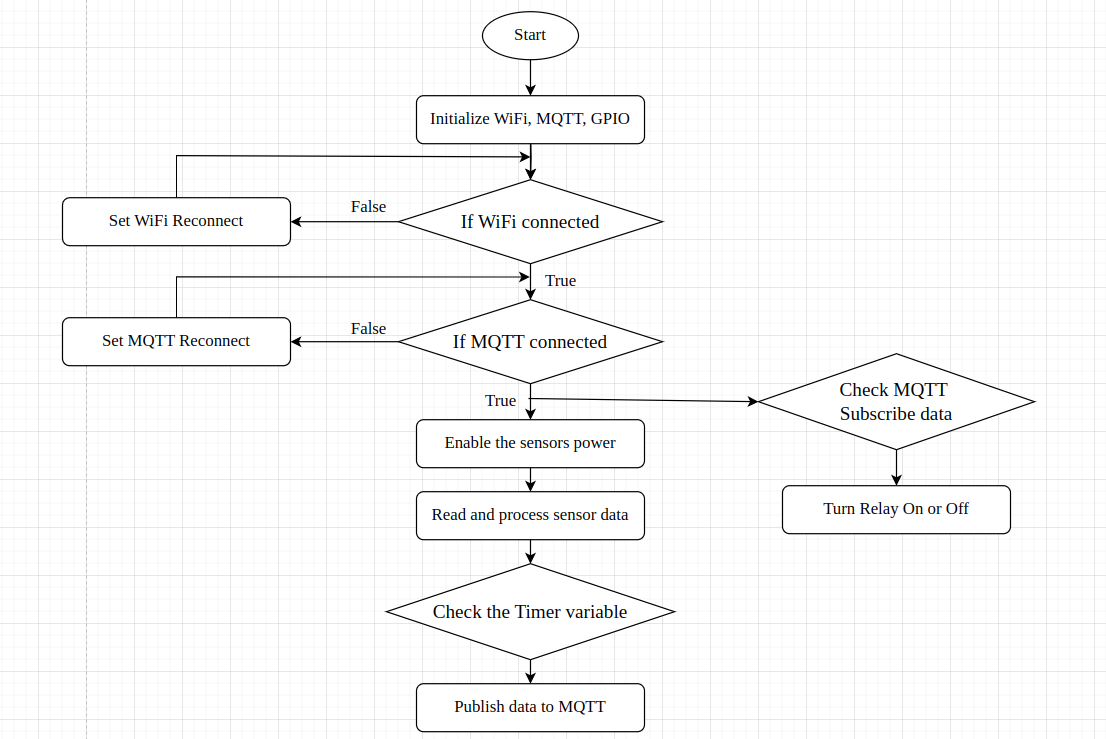


**Figure 3.6**: Esp32-CAM connection diagram

For Esp32-CAM it's quite simple, it just needs to be connected to a 5V power supply and GND.

## 3.3 Algorithm design

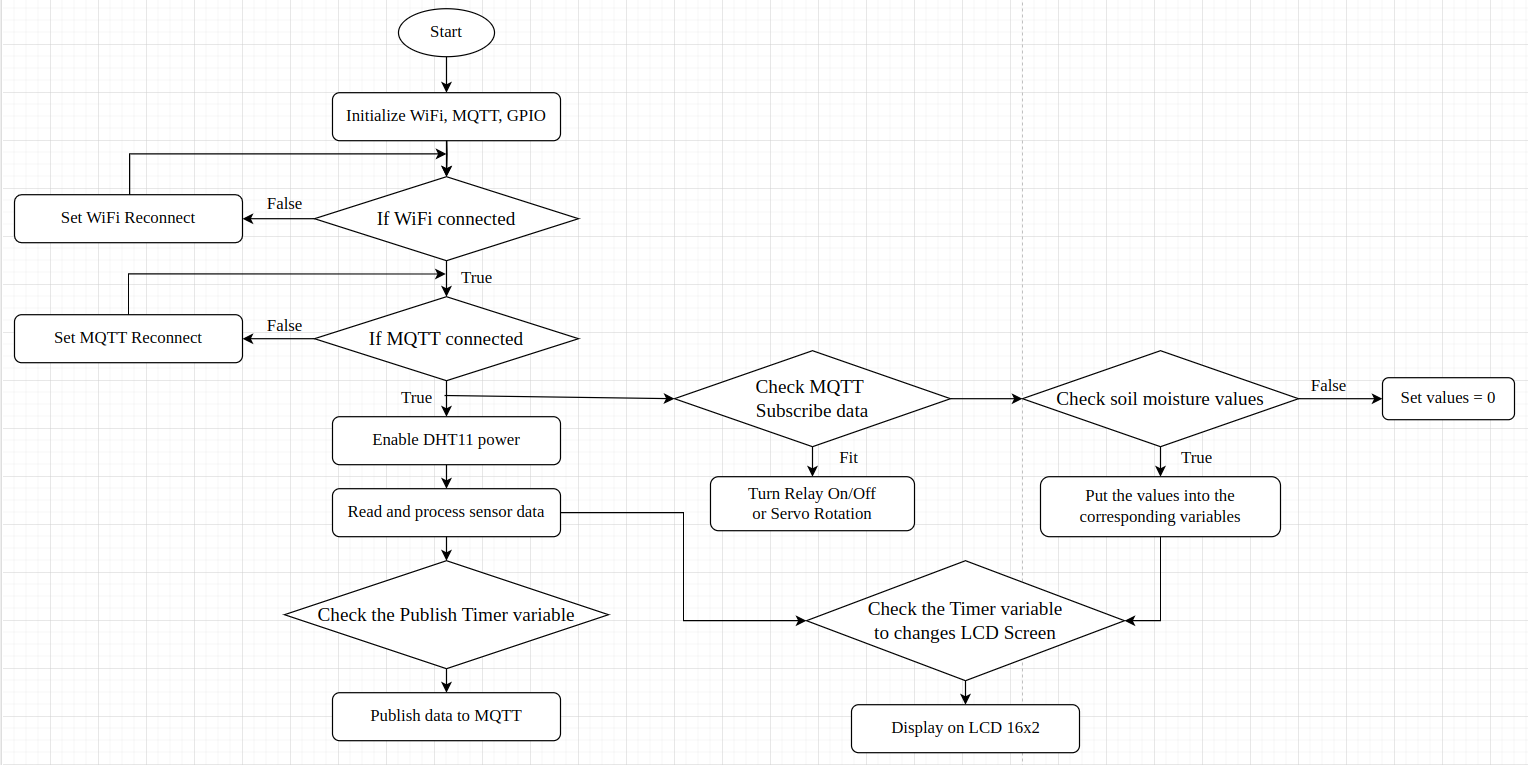
### 3.3.1 Node Esp32 30Pin Processing Algorithm



**Figure 3.7**: Processing algorithm of Node Esp32 30Pin

* At startup, the ESP32 will initialize Wi-Fi, MQTT configurations, and the GPIOs connected to the devices.
* Immediately after, it will check the Wi-Fi network connection. If the check fails, it will attempt to reconnect to the Wi-Fi.
* Once the Wi-Fi connection is successful, it will proceed to check the MQTT connection. If the check fails, it will attempt to reconnect to MQTT.
* Once the MQTT connection is successful, it will power on the sensor modules and check the MQTT topics that have been subscribed to.
* If the MQTT Topic and Message are appropriate, it will turn the corresponding Relays On/Off. It will also check the Timer variable to publish the processed sensor data.

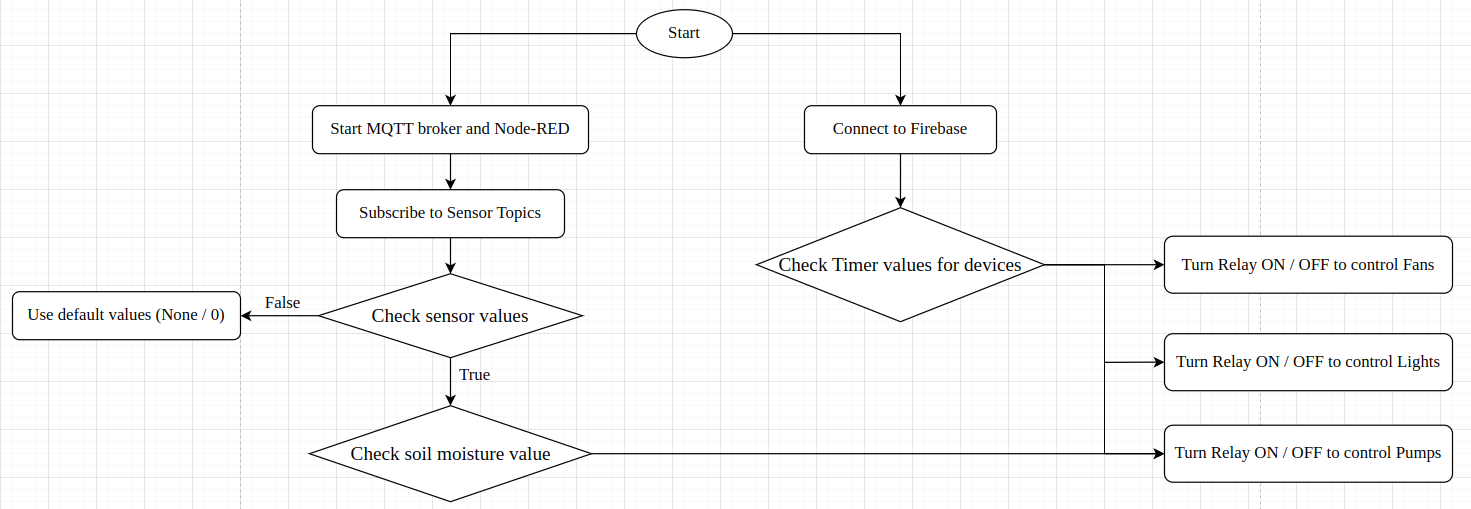
### 3.3.2 Node Esp32 38Pin Processing Algorithm



**Figure 3.8**: Processing algorithm of Node Esp32 38Pin

* At the start, the ESP32 will initialize Wi-Fi, MQTT configurations, and GPIOs connected to the devices.
* Immediately after, it will check the Wi-Fi connection. If the check fails, it will attempt to reconnect to Wi-Fi.
* Once the Wi-Fi connection is successful, it will proceed to check the MQTT connection. If the check fails, it will attempt to reconnect to MQTT.
* Once the MQTT connection is successful, it will activate the power of the sensor modules and check the MQTT Topics it has subscribed to.
* If the MQTT Topic and Message match, it will turn the corresponding Relays on/off or rotate the Servo motor. It will also check the Timer variable to publish the processed sensor data.
* In addition, after receiving data from the DHT11 sensor and the soil moisture sensor via MQTT, it will check another Timer variable to sequentially display the values on the 16x2 LCD screen.

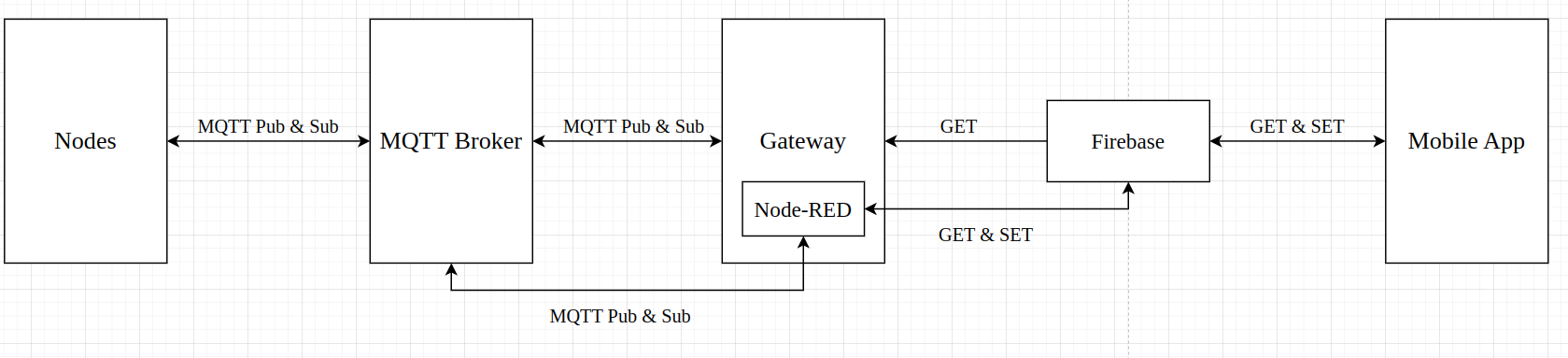
### 3.3.3 Gateway - Node Processing Algorithm



**Figure 3.9**: Gateway - Node Processing Algorithm

* When the Gateway (Raspberry Pi) starts, it will initialize default values for the sensor readings and device statuses, and it will also start the MQTT Broker, Node-RED, as well as connect to Firebase.
* From there, it can publish and subscribe to MQTT topics from the Esp32 nodes to receive sensor data and control devices. It will also receive timer values from Firebase to schedule device activation.
* It receives soil moisture values from the MQTT topics it subscribes to, then publishes them to control the water pump connected to the relay accordingly.
  + If soil moisture is < 35%, it’ll publish "ON" to the pump control topic.
  + If soil moisture is > 60%, it’ll publish "OFF" to the pump control topic.
* It will also receive Timer values from the Firebase database to publish MQTT messages that activate devices according to the schedule set by the user on the React Native monitoring app.

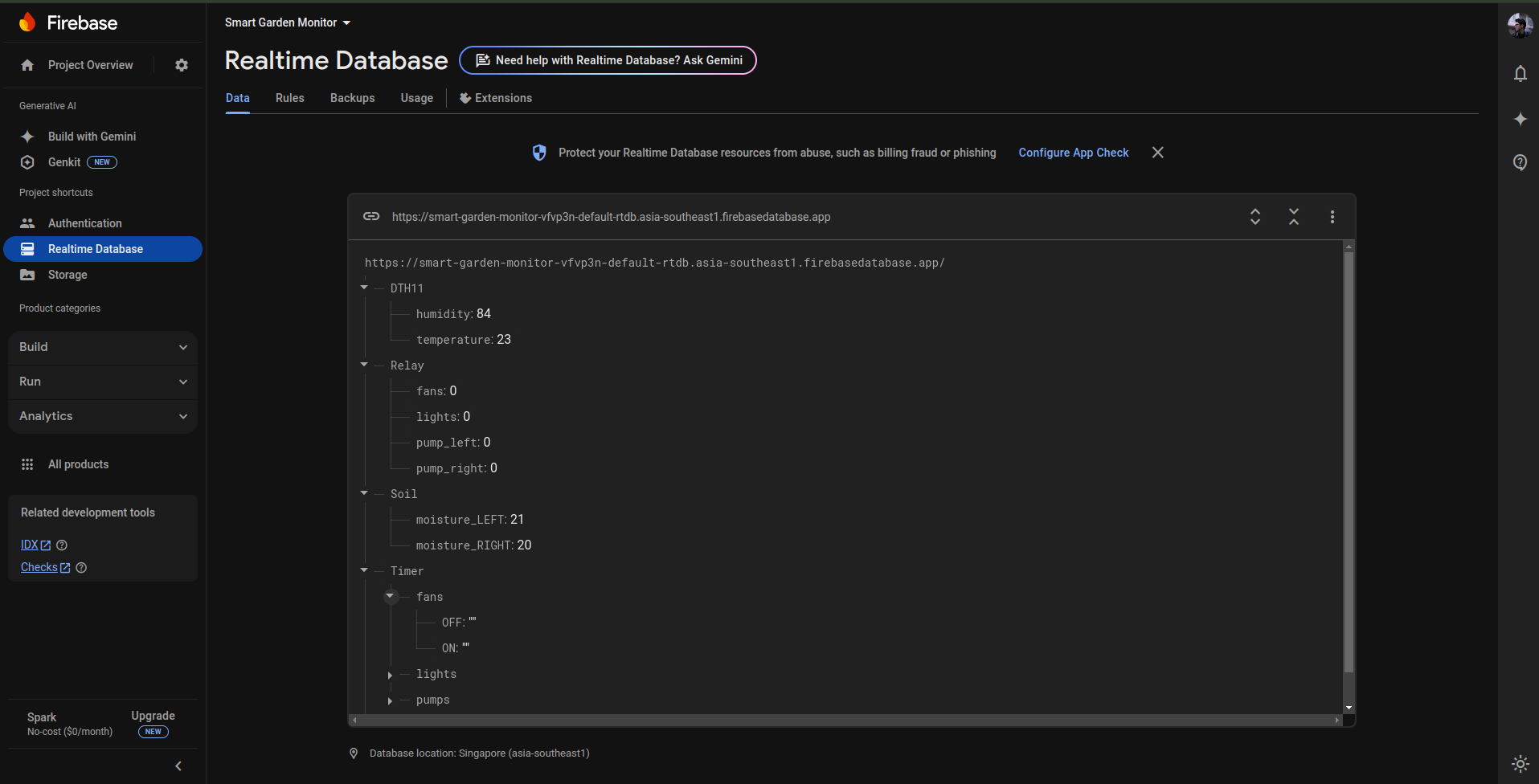
## 3.4 Data Stream



**Figure 3.10**: Data flow in the system

* The devices and sensor modules are connected to the Nodes.
* The Nodes communicate with the Gateway through the MQTT Broker.
* On the Gateway, Node-RED runs and communicates with the MQTT Broker and Firebase to transfer data from MQTT to the database.
* Once the data is stored in the Firebase database, it will be retrieved to display on the monitoring application and used to control the devices.

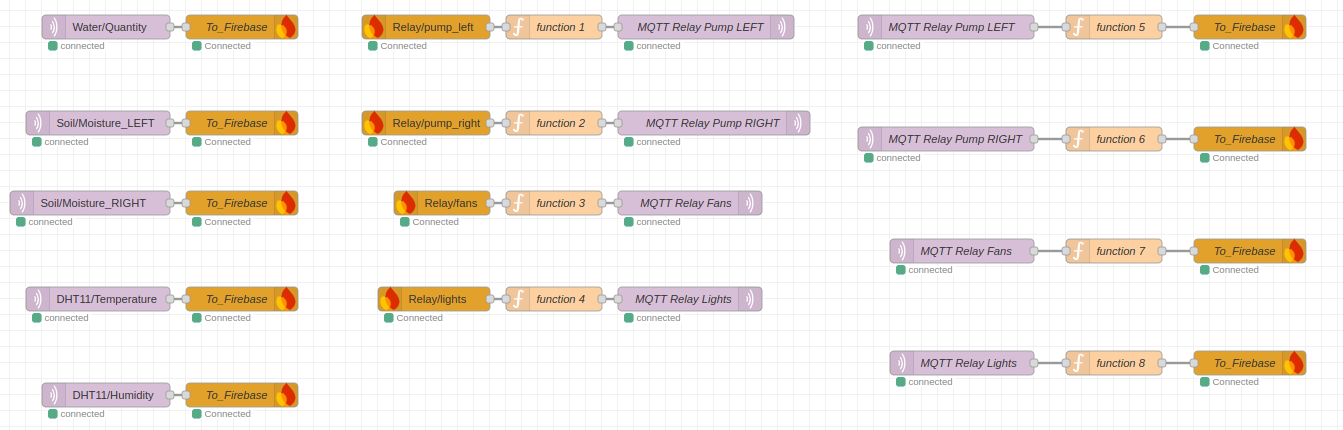
### 3.4.1 Firebase



**Figure 3.11**: Realtime database of Firebase

* The data stored in Firebase Realtime Database is in the form of a JSON tree. This is a hierarchical structure where nodes represent data values, and child nodes are organized like a tree.
* The system data is divided into the following nodes:
  + DHT11: Contains child nodes for temperature and humidity data, with the data type being int.
  + Relay: Contains child nodes representing the status of relays, named according to the connected devices, with data type 0 / 1.
  + Soil: Contains child nodes representing the soil moisture data for the left and right plant pots, with data type as int.
  + Timer: Contains child nodes named after the devices connected to the relays. Within these child nodes, there are two further child nodes, "ON" and "OFF", which store the times to turn the devices on and off. The data type is String.
  + Water Quantity: Does not contain any child nodes. Its data is the value obtained from the water level sensor, and the data type is int.

### 3.4.2 Node-RED Flow

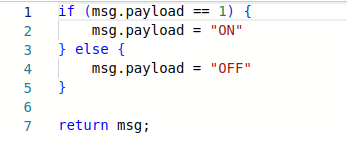


**Figure 3.12**: Data Flow in Node-RED

The Node-RED flow simply takes data from the MQTT topics containing values obtained from the modules, processes it, and uploads it to Firebase for data storage.

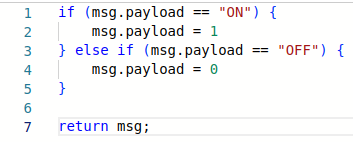
It also processes data from Firebase to publish it back to the MQTT topics that control the devices.

Since to activate the relays, we need to publish "ON" or "OFF" to the corresponding MQTT topics, but the relay status in Firebase is stored as 0/1, we need a function to convert these values.



**Figure 3.13**: Function to convert values ​​from Firebase to MQTT

In order to display the correct current status of the relays, we will also need a function to convert the values from MQTT to Firebase.



**Figure 3.14**: Function to convert values ​​from MQTT to Firebase

As a result, the device status values displayed on the Mobile application will correspond to the actual status of the relays controlling those devices in real-time.

# 

# CHAPTER IV : RESULT

## 4.1 Proposed results

During the implementation process, I achieved the following results:

* Collecting values from the temperature-humidity sensor and soil moisture sensor, these are two sensors that are suitable for use in agricultural environments.
* Perform data transmission and reception between 3 Nodes and the Gateway:
  + The Gateway receives sensor values from the 2 ESP32 Nodes via the MQTT protocol. Additionally, it can control the devices connected to these 2 Nodes by using MQTT Publish.
  + The Gateway also receives the frame captured by the ESP32-CAM through its own fixed IP address.
* The data flow between MQTT and Firebase is linked through the Node-RED tool running on the Gateway.
* Complete the development of a mobile application to monitor sensor values and control devices, as well as set schedules for turning devices on/off. Additionally, the app should allow users to view images taken by the camera to monitor plant health, captured twice a day and from the previous day's afternoon.
* Complete the development of a face recognition model using Python libraries, which will be applied to frames captured by the ESP32-CAM to recognize the garden owner and open the garden door.
* Analyze the NDVI index from the frames captured by the USB camera, then take photos and highlight the weak vegetation areas.
* The NDVI analysis images are scheduled to be captured at two specific times of the day: once at 8 AM and once at 5 PM.
* The captured images will be organized into specific folders and then uploaded to the HTTP Server, allowing the Mobile app to GET and display them.

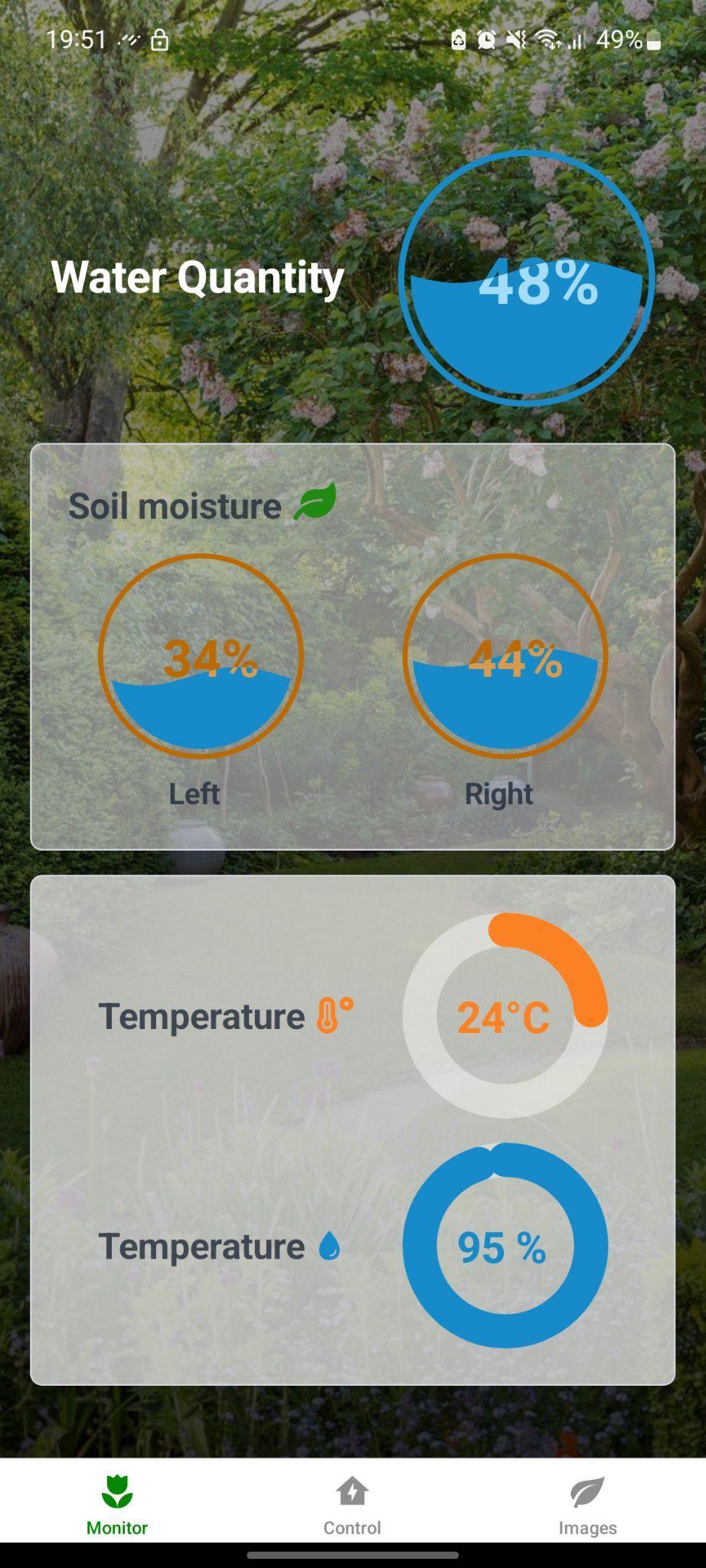
## 4.2 Mobile application

The Mobile application, built using the React Native framework and written in TypeScript, is connected to Firebase Realtime Database to display sensor data and control devices, as well as set timers for the devices.

The Mobile application will include 3 main screens, allowing users to monitor the values of the system's sensors and control the devices.

### 4.2.1 Monitor Screen

The Monitor screen will display the sensor values.



**Figure 4.1:** Monitor screen on the application

At the top, it displays the percentage of the remaining water in the water tank, measured by the water level sensor.

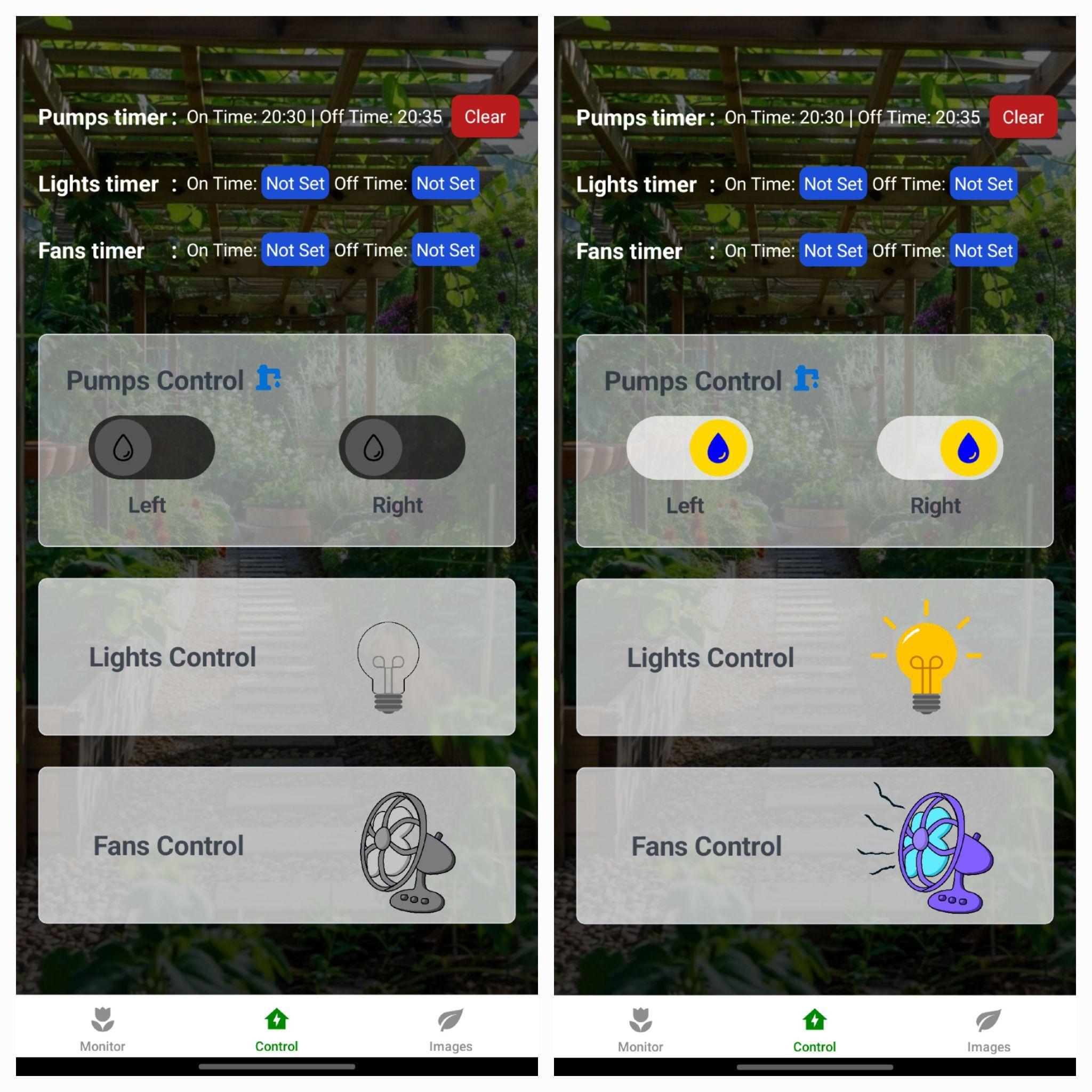
Below that is a card displaying the soil moisture readings from two soil moisture sensors located in the left and right plant containers.

Finally, there is a card displaying the temperature and humidity readings from the DHT11 sensor.

The sensors collect the values, which are then processed and transmitted to the Firebase database, where they can be displayed on the app.

### 4.2.2 Control Screen

The Monitor screen will allow users to control devices and set timers to turn them on/off.

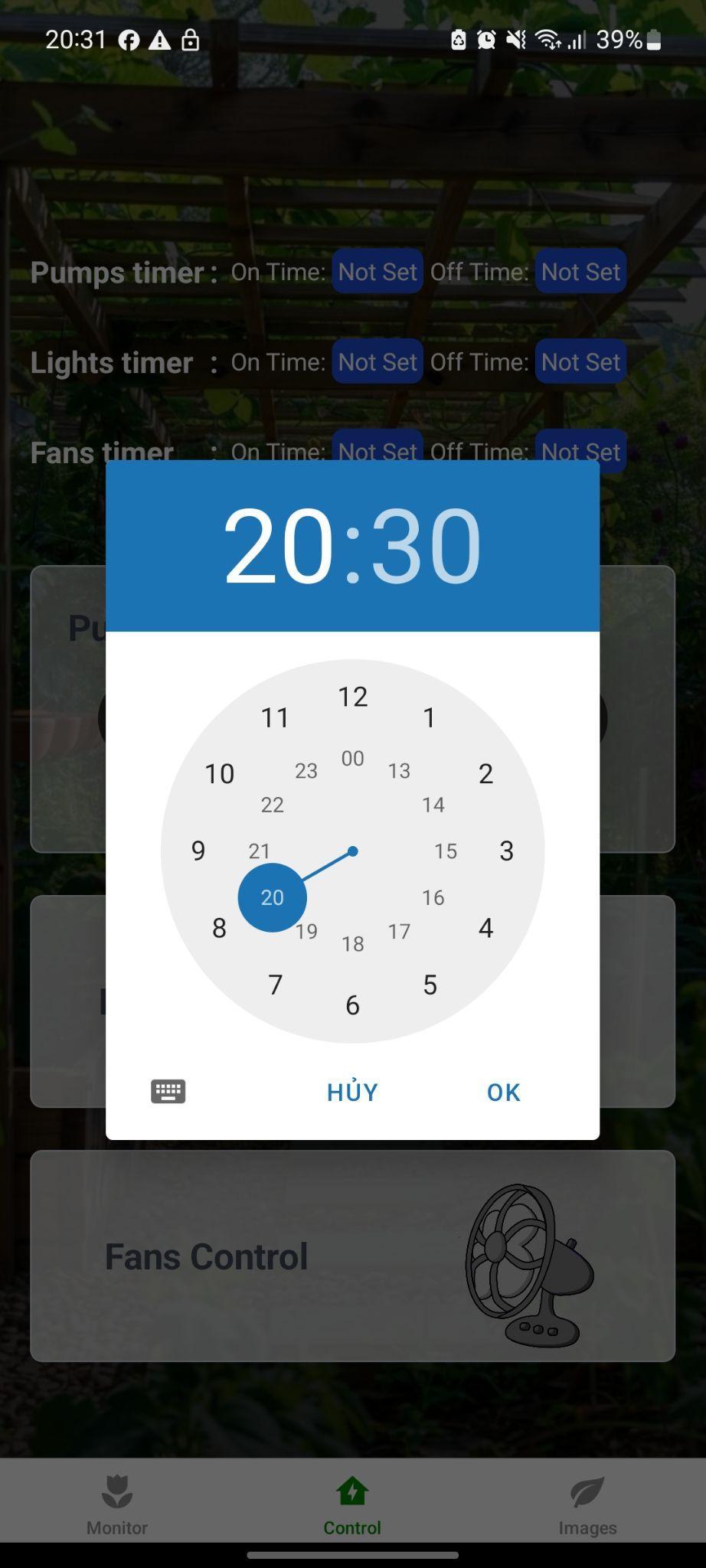


**Figure 4.2:** Control Screen when Turning Off and On Devices

It is simply used to control devices connected to the Relay and display their current status correctly.

For the timer for each device, it will display "Not Set" for any timers that have not been configured.

The user will not be able to set only the On Time or Off Time; they will be required to set both On Time and Off Time for each device.

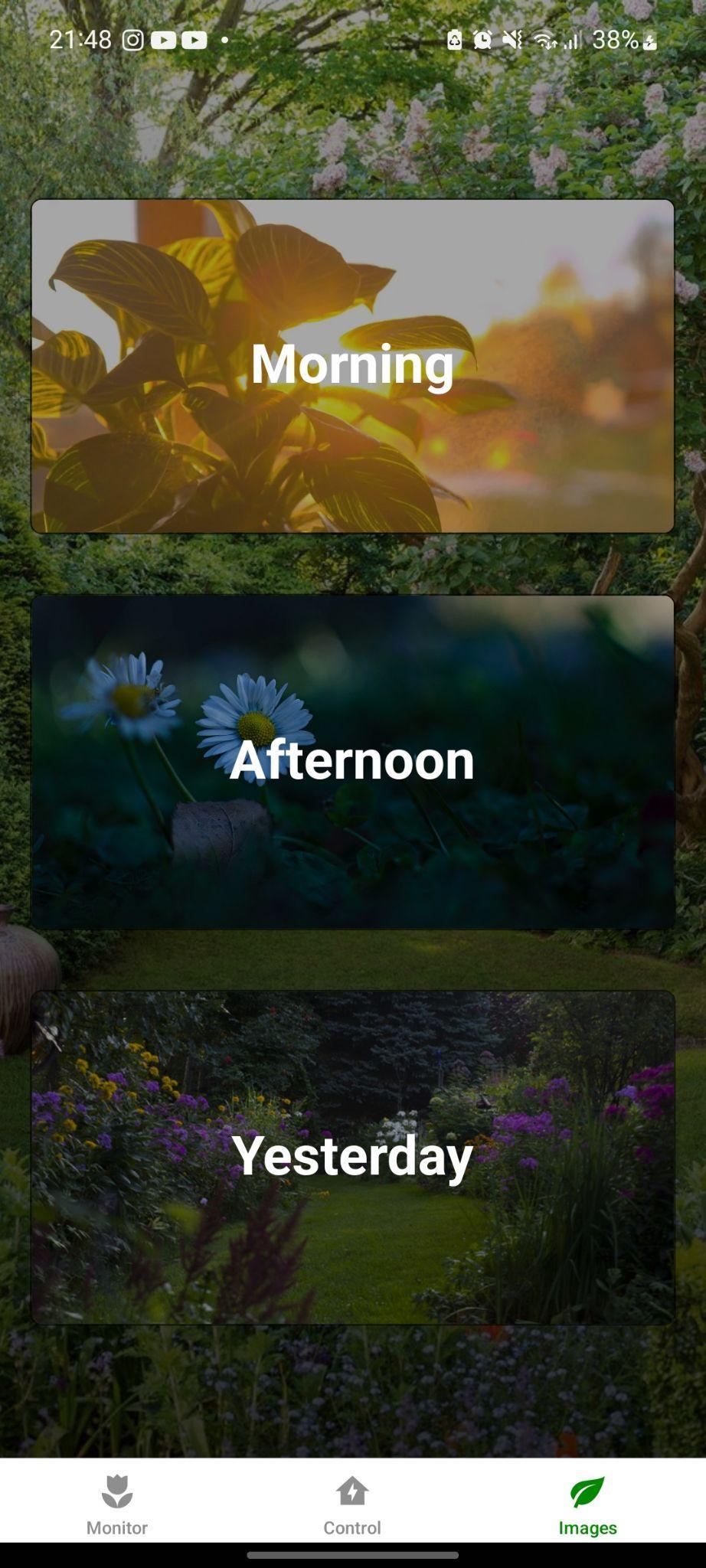


**Figure 4.3:** Device timer screen

When setting the timer, if the Off Time is earlier than the On Time or if the Off Time is more than 2 hours later than the On Time, a warning will be displayed: "You have set the usage time for the device too long," and the user will be asked to reset the Off Time immediately.

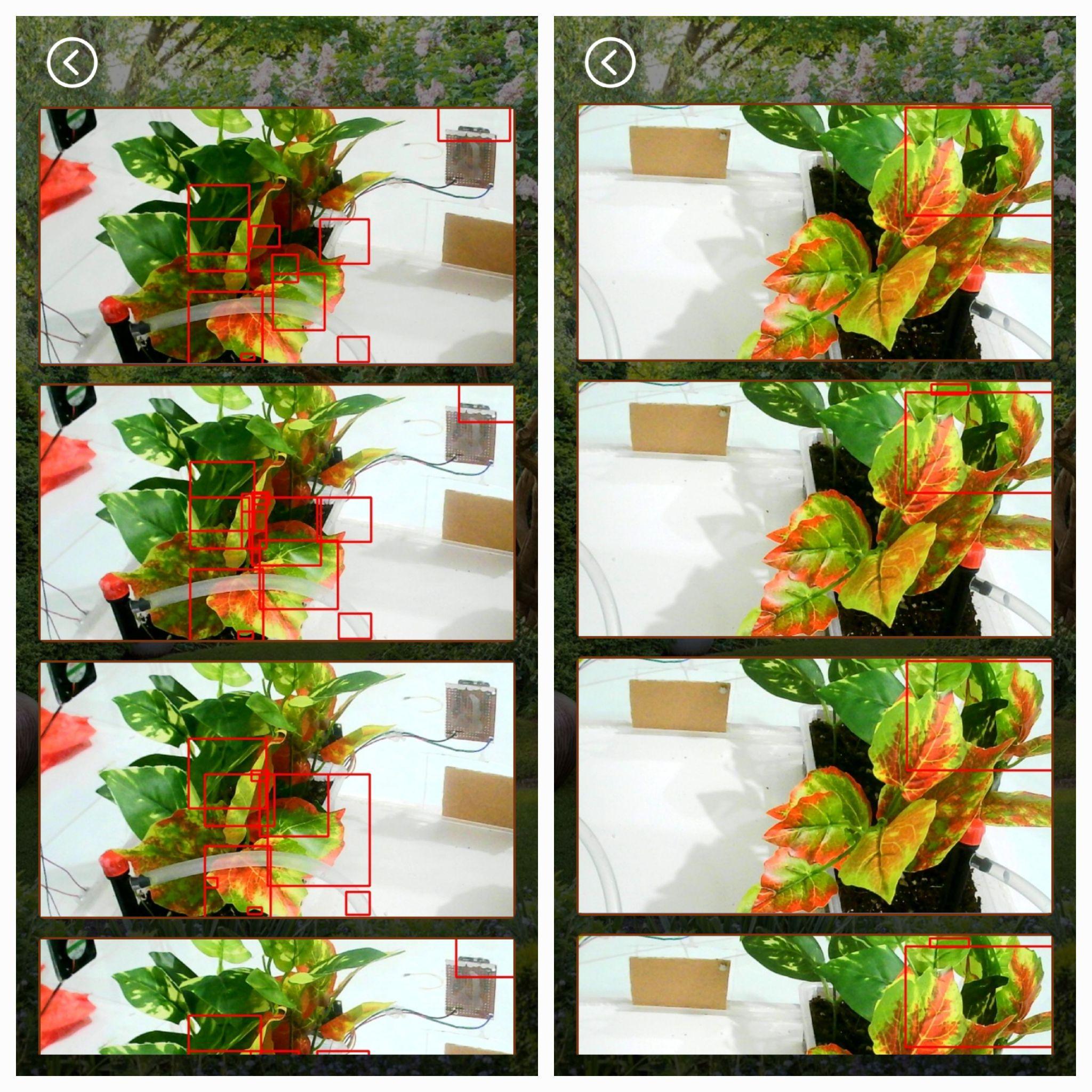
### 4.2.3 Images Screen

The Images screen will display 3 Card Buttons: Morning, Afternoon, and Yesterday. When one of the buttons is pressed, the user will be redirected to a new screen to display the corresponding images.



**Figure 4.4:** Images screen on the application

In the new screen, after pressing one of the 3 buttons, there will be a list containing 15 images captured by the system at 8 AM, 5 PM, and 15 images from the previous afternoon.



**Figure 4.5:** The screen displays a list of photos taken in both planters.

## 4.3 Face recognition model

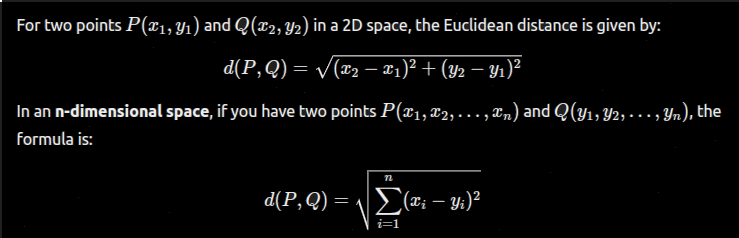
The recognition model is built on a PC (not on the Raspberry Pi). After obtaining the model file, it will be uploaded to the Raspberry Pi to perform face recognition through the frames captured by the ESP32-CAM.

First, it will capture 200 photos of the user's face and save them in a folder named after the user. If you want to use it with two or more users, you just need to run it multiple times and modify the folders corresponding to each specific user.

From the folder containing the 200 captured photos mentioned above, we will build a face recognition model using Python's face\_recognition library.

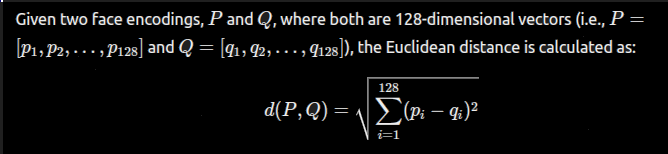
* The face\_recognition library is a powerful Python tool for performing face recognition and processing tasks, such as face detection, face recognition, and face matching.
* One of the key features of the face\_recognition library is the face\_encodings function, which is used to extract face encodings. These encodings are feature vectors that represent a face as numerical values.
* The face\_encodings function uses a deep learning model trained to convert a face into a 128-dimensional vector. This vector is an abstract representation of the facial features, which helps in face recognition by comparing the similarity between vectors.
  + When two faces are similar, the Euclidean distance between their vectors will be small.
  + When two faces are different, the distance will be greater than a certain threshold.

**Note**: Euclidean distance is a way to measure the distance between two points in space. It is one of the mathematical distance metrics, especially when working with vectors.



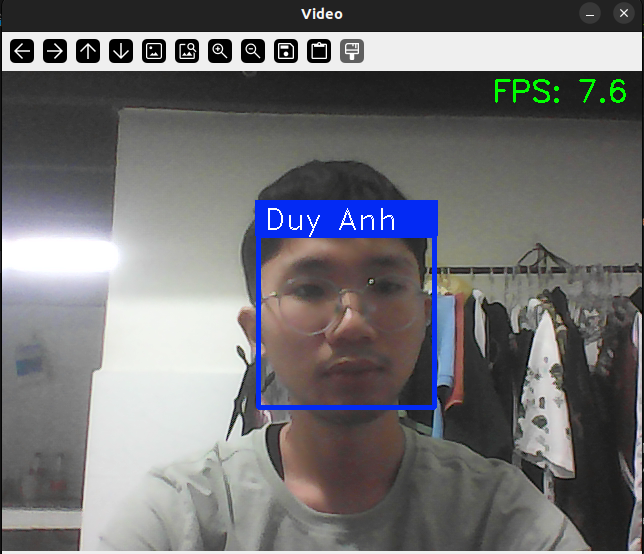
**Figure 4.6:** Formula for calculating Euclidean distance

In the facial recognition task, each face is encoded into a 128-dimensional vector by the face\_encodings function.



**Figure 4.7:** Calculating Euclidean distance in face recognition

* After building the facial recognition model, it will export a .pickle file to be used for facial recognition in the system.
* The .pickle file will contain a dictionary with two lists: face encodings and corresponding user names.



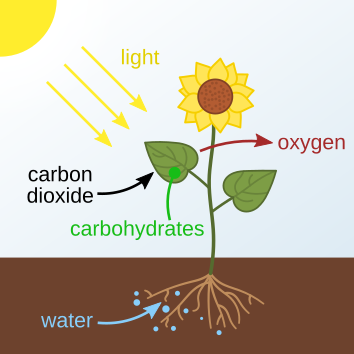
**Figure 4.8:** Experiment with face recognition model

* We will use this file on the Raspberry Pi for face recognition and employ the Paho-MQTT library in Python to publish to an MQTT topic to open the garden model door.

## 4.4 Plant health monitoring

### 4.4.1 Why are leaves green?

The reason most plant leaves are green is because they contain a chemical called chlorophyll. This chemical helps them use sunlight to convert carbon dioxide and water into useful chemical substances called carbohydrates, in a process known as photosynthesis.



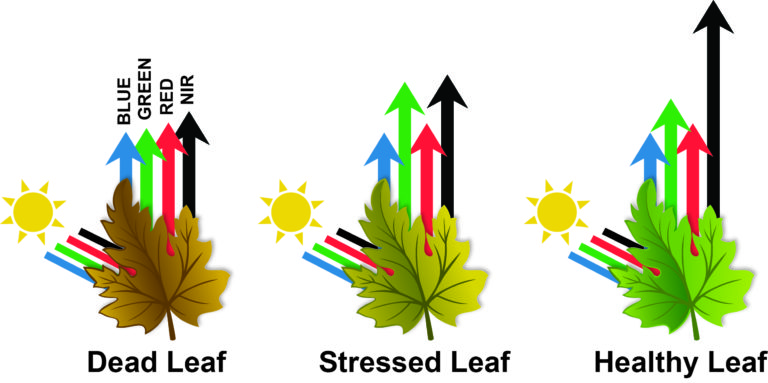
**Figure 4.9:** Photosynthesis process of plants

Sunlight comes in various forms. Like a rainbow, we can see many different colors of light. Among them, ultraviolet (UV) light is the type that can cause sunburn, and infrared (IR) light is why we feel warmth when we hold our hand near a fire.

However, chlorophyll can only use a portion of sunlight for photosynthesis. It cannot absorb blue light, so this light gets reflected, which is why plants appear green. Plants also don't like infrared radiation much because it heats them up. They have evolved to reflect as much infrared light as possible.

From this, we know that if a plant is healthy, it will reflect a lot of near-infrared (NIR) light. If the plant is dying, it will absorb a lot of NIR light.

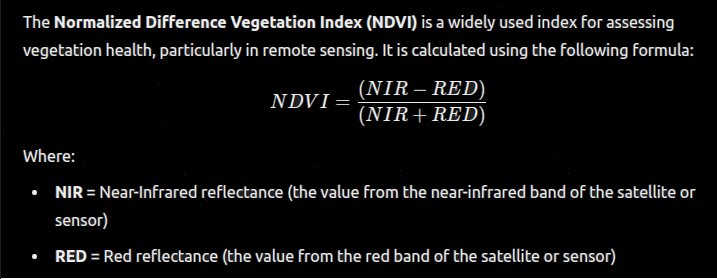
By observing the image of the leaves reflecting light below, we can see that more infrared light is reflected from healthy leaves compared to weak or dead leaves.



**Figure 4.10:** Light rays reflected from leaves

### 4.4.2 NDVI index

NDVI stands for Normalized Difference Vegetation Index. It is an index calculated from the spectral bands of light in images captured by a camera. NDVI helps measure plant health and density by comparing the light absorbed and reflected by the plants.



**Figure 4.11:** Formula for calculating NDVI index

### 4.4.3 Monitoring plant health with NDVI index

Contrast refers to the difference in brightness or color within an image, and to accurately calculate the NDVI index, we need to enhance the contrast of the image.

Then, we will use the NDVI formula above to calculate it. Due to budget constraints, the system will not use an NIR camera; instead, we will use the GREEN channel in place of it for the calculation.

The NDVI value, after being calculated, will range from -1 to 1.

* NDVI < 0: Typically non-vegetative materials, such as rocks or water.
* NDVI from 0 to 0.2: Surfaces like rocks, sand, or very sparse vegetation.
* NDVI from 0.2 to 0.5: Moderately healthy vegetation, or plants in conditions of water or energy deficiency.
* NDVI > 0.5: Healthy, thriving vegetation with high photosynthetic activity.

### 4.4.4 Plant health monitoring system

The system will take 15 pictures at 8 AM and 5 PM every day.

The first camera will identify the plant area by comparing frames and selecting the one with more green hues, as it will be identified as the vegetation area.

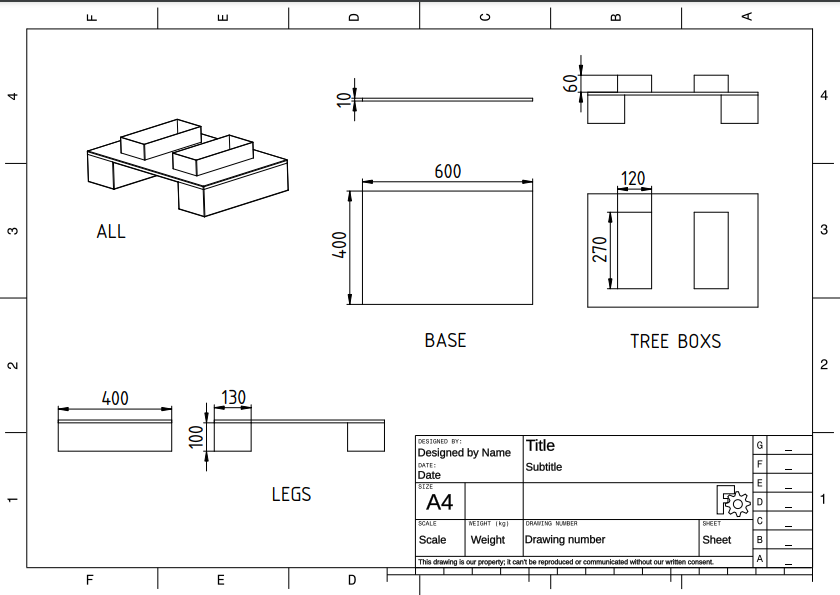
After identifying the vegetation area, the system will calculate the NDVI value and highlight the frames in the vegetation area with low NDVI values, designating them as weak vegetation areas.

The photos taken will be stored in a folder and an HTTP server will be used to allow the application to access them.

## 4.5 System model

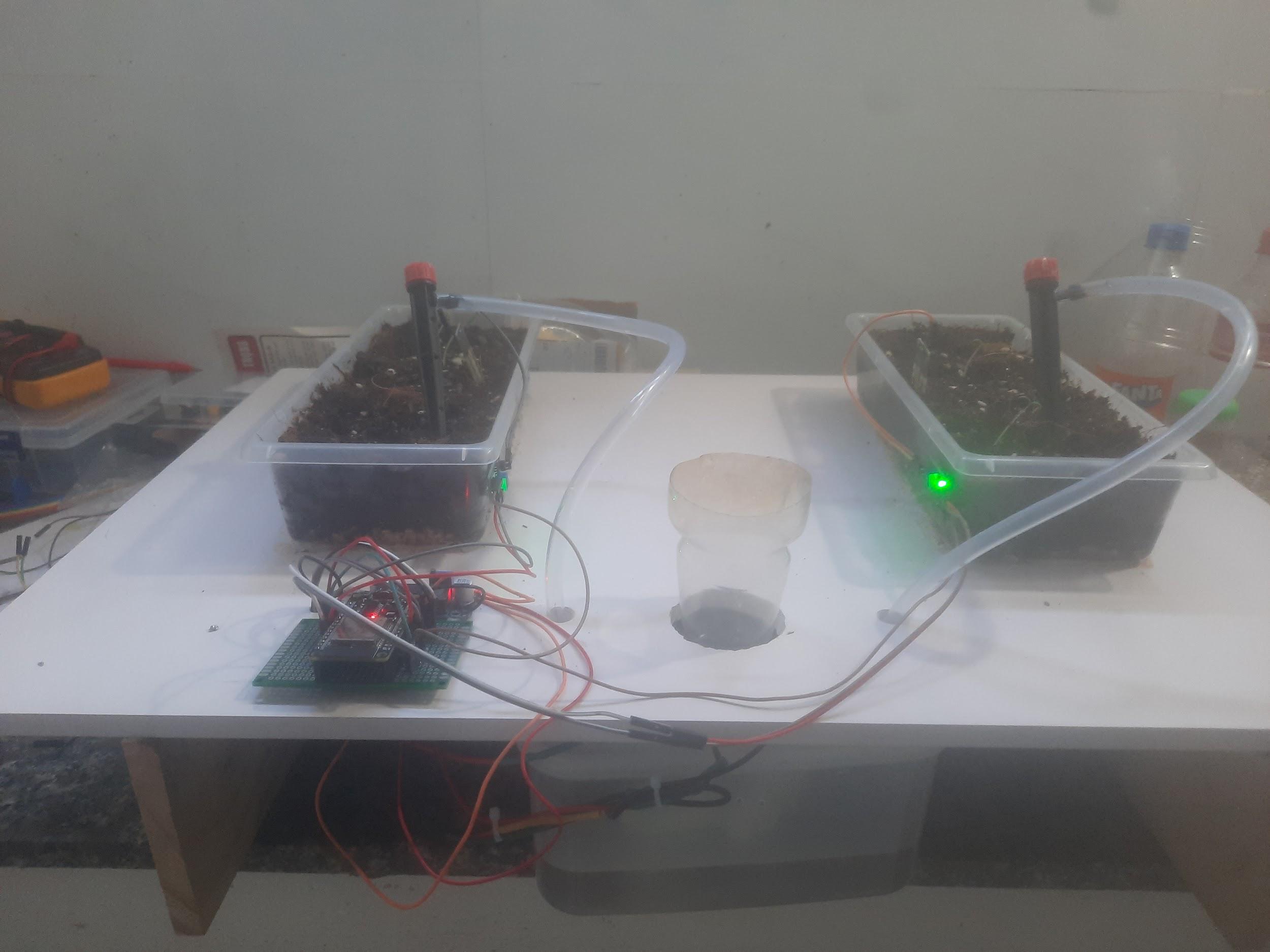
### 4.5.1 Garden base model

The initial model was designed using FreeCAD, with the base of the garden being 40x60cm. The garden stand legs are made of wood with dimensions of 40x10cm, and the two plant containers are made from plastic boxes measuring 27x12x6cm.



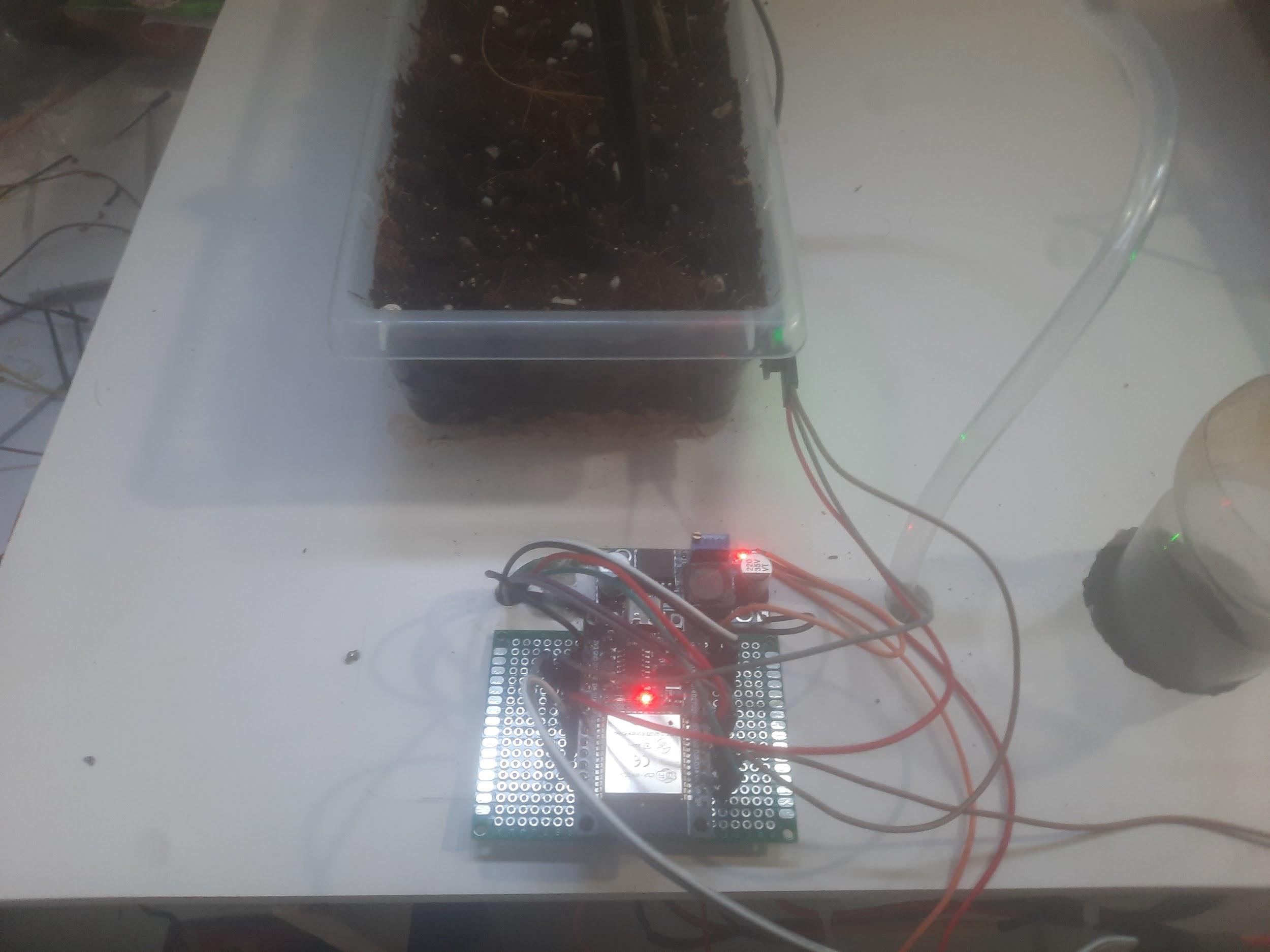
**Figure 4.12:** Initial design of the garden base.

After completing the base, the system includes an ESP32 30-pin Node controlling 2 relays connected to 2 submersible water pumps, 1 water level sensor, and 2 soil moisture sensors placed in the 2 plant containers.



**Figure 4.13:** Garden dike after completion.

The devices are connected to the ESP32 through a perforated PCB, with a 5V power supply provided from the output of the LM2596 voltage regulator module.



**Figure 4.14:** Esp32 30Pin connection on hole PCB

### 4.5.2 Upper part of the garden model

The upper part of the garden (including the roof) will be made entirely of transparent acrylic plastic (Mica) with a height of 30 cm.



**Figure 4.15:** Model when completed.

On the upper part of the garden model, two 12V fan modules, an I2C-connected LCD screen, and an ESP32-CAM module are installed outside the door. The door is connected to a servo motor. Additionally, a DHT11 sensor module is used to collect temperature and humidity data from the environment inside the model.



**Figure 4.16:** Model when all devices are turned on

The roof of the model is equipped with 12V LED strip lights.

Both the fan and the 12V LED strips are connected to relays.

All the devices in the upper part of the model are connected to the Node Esp32 38Pin, with a 5V power supply provided from the output of the LM2596 voltage regulator module.

# 

# CONCLUSION AND DEVELOPMENT DIRECTION

## 1. Conclude

After 3 months of research and project implementation, I have successfully realized some of the set requirementsL

* The theoretical requirements:
  + Overview of the topic: Conducted research and acquired basic knowledge about agriculture and the importance of caring for clean vegetables in relation to health.
  + Theoretical research and necessary equipment: Studied relevant theories, planned the implementation of the project, and selected the required hardware for execution.
  + Analysis and design of the system overview model: Based on existing research, I analyzed the limitations and developed an overview model for each module, designing solutions and data structures.
* Deployment requirements:
  + Implement a system consisting of 1 Gateway and 3 Nodes, where 2 Esp32 Nodes are responsible for collecting data and sending it to the Gateway, and 1 Esp32-CAM Node is tasked with capturing frames and creating a stream with its fixed IP. The Gateway's role is to receive data and send control signals to the Esp32 Nodes. It also receives frames from the Esp32-CAM for face recognition evaluation.
  + The system operates with two separate power sources: one DC ~12V source for the Nodes, which includes a step-down module to power devices with limited voltage, and an AC power source for the Gateway.
  + Create a data stream to send data from the Nodes (including sensor data and device status data) to be stored in Firebase Database.
  + Design and implement on a perforated PCB board for the Nodes in the system, test and ensure that the components are properly connected through the PCB board.
  + Design and implement garden models.
  + Design a mobile application to display the measured parameters, enabling monitoring and management of the garden system. Integrate features to turn devices on/off, as well as set timers for them. This allows users to easily control the system.

## 2. Limitations of the topic

Although the project has been completed, I have noticed some limitations in the system as follows:

* The system is currently implemented as a model, and when deployed in practice, it will require further improvements such as extending the operation time of the Nodes and optimizing them with Sleep Mode to reduce power consumption. Additionally, further research is needed to determine the actual water requirements for each type of plant.
* The system is currently implemented as a model, and when deployed in practice, it will require further improvements such as extending the operation time of the Nodes and optimizing them with Sleep Mode to reduce power consumption. Additionally, further research is needed to determine the actual water requirements for each type of plant.
* The NDVI analysis part has not yet achieved high stability due to the lack of an NIR Camera, which is necessary for more accurate NDVI calculations. Currently, the system only uses an RGB Camera and substitutes the Green channel for the NIR index.

## 3. Development direction

I would like to propose some methods that can be applied to improve the system and make it more complete:

* Apply additional wireless technologies with a wide range to integrate into large-scale agricultural systems. Technologies that can be applied include LoRaWAN, Sigfox, and others.
* Integrate a pH sensor in the water to monitor nutrients and toxins. To achieve high crop yields, it is essential to maintain the water and soil pH within the optimal range suitable for each type of plant.
* Use a battery charging circuit and optionally integrate a solar panel to utilize clean energy, ensuring the system operates for an extended period.
* Add additional features such as sending emails or text messages to users for alerts. Instead of only accessing the system through the mobile app, also include a website and desktop application for the system.

# REFERENCES

[1]. "Digital Design and Computer Architecture" của David Money Harris và Sarah L. Harris

[2]. "The 8051 Microcontroller and Embedded Systems: Using Assembly and C" của Muhammad Ali Mazidi, Janice Gillispie Mazidi, và Rolin D. McKinlay

[3]. "Digital Electronics: A Practical Approach with VHDL" của William Kleitz

[4]. Electronics.stackexchange.com

[5]. Forum.arduino.cc

[6]. Electronics-lab.com