Imperial College London

Consultancy Group Project
Summer 2024

AI Irrigation Robot

July 25, 2024

Authors	CID
Ajanthan Kanagasabapathy	02017931
Damani Mguni-Coker	01861634
Indira Thanigaikumar	02033258
Madhushree Manjunatha	02101544
Mihir Ashutosh Acharya	02026394
Noor Elsheikh	02025422

1 Introduction and Background

The brief of this project can be summarised as: *"Develop a robot capable of detecting optimal weather conditions for a specific plant and autonomously relocating the plant to those conditions. This involves accessing Weather data to determine local weather and optimal conditions, notifying the user via Text to Speech regarding the suitability of the local conditions, sensing the plants exposure to light and moisture."* This project offers a wide range of applications, such as minimising unnecessary water use by ensuring plants are watered only when necessary. It is designed to provide optimal care for plants, potentially enhancing their growth and health. Primarily aimed at domestic plant owners, the technology could be expanded for use in agricultural settings. Moreover, integrating real-time analytics, machine learning, and a user interface to display these capabilities could extend its benefits to agricultural crops. Such advancements could contribute to addressing global challenges like food shortages.

2 System Design

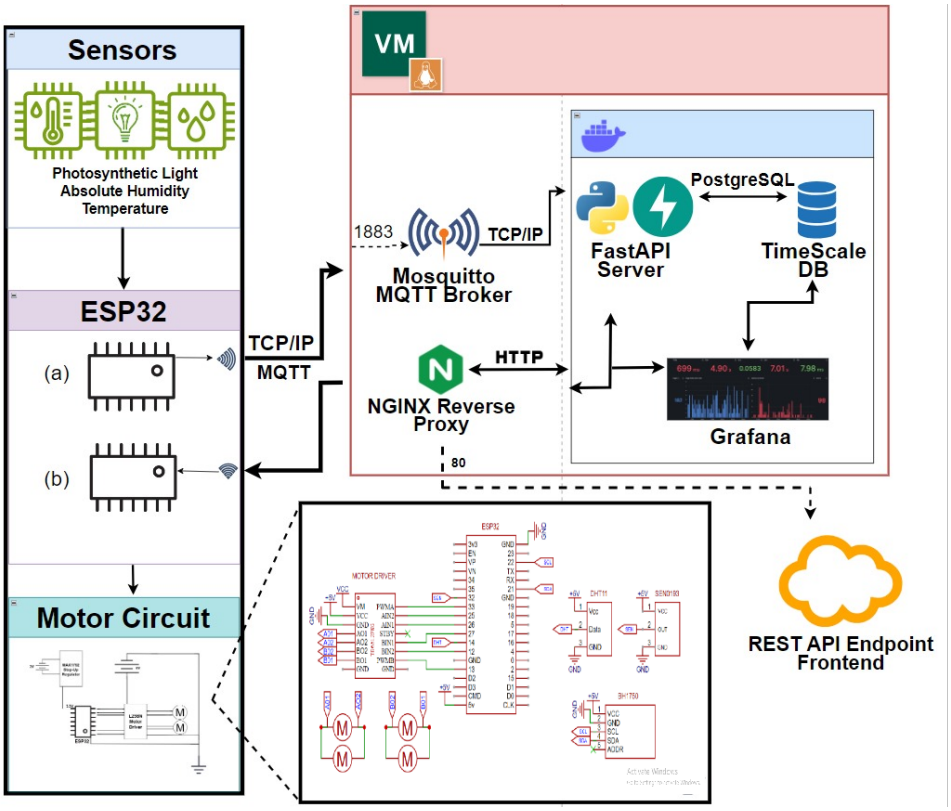


Figure 1: High-level system diagram

The robot integrates multiple sensors and microcontrollers with cloud services for optimal plant care. The DHT11, BH1750, and SEN0193 sensors, connected to the ESP32 microcontroller, continuously monitor environmental and soil conditions. The ESP32 processes this data locally and communicates via WiFi to a Mosquitto MQTT broker. The broker forwards data to a FastAPI server, which stores it in a PostgreSQL database. The TB6612FNG motor driver, controlled by the ESP32, adjusts the plant's position based on sensor readings.

Time-series data is visualised on the frontend using Grafana, while remote monitoring and control are facilitated through a REST API endpoint, ensuring efficient and autonomous plant care. The IBM WatsonAI LLM drives a chatbot on the frontend as the visualised graphing, frequently pulling updated sensor readings from the database for responding to user queries in an LLM fashion regarding the current status of the plant.

3 Website

A basic application was built using Javascript, HTML and CSS. The website showcased the real-time metrics collected from the sensor data for moisture, light levels and temperature to keep the user informed. Whilst the robot could autonomously move to correct for its environment (for example will move to areas with more light if it drops below a certain threshold), the user also has the option to manually move the robot if they wanted to. There is also an "Update" button where the user can hear about updates about on the plant. The IBM Watson chat assistant is also embedded into the website.

3.1 Text-to-Speech

The text-to-speech functionality in the application was enabled using IBM Watson's text-to-speech model, which was integrated through a sophisticated API interface. This advanced model proficiently dictates whether the plant's environmental conditions—such as temperature, humidity, moisture, and light—are below required levels, are optimal, or exceed established thresholds. The integration of this feature enhances the website's interactivity by providing immediate verbal feedback to users, thereby making the information more accessible and engaging. Additionally, the voice of the text-to-speech model was carefully customised to cater specifically to the needs of our application, ensuring a user-friendly experience that aligns with the overall design and purpose of the system.

3.2 Chat-bot

In addition to text-to-speech capabilities, IBM Watson Assistant was integrated into our application to provide a more personalised and responsive user experience. This integration featured two distinct conversational streams: updates on the plant's current condition and real-time weather forecasts. The plant update stream delivers timely information about the plant's health, including temperature, moisture, humidity and light levels, enabling users to monitor and react to any changes effectively. The weather forecast stream provides up-to-date meteorological data, assisting users in planning for future environmental conditions that could affect their plants. The dialogues within these streams are structured through intuitive flowcharts, which are illustrated below.

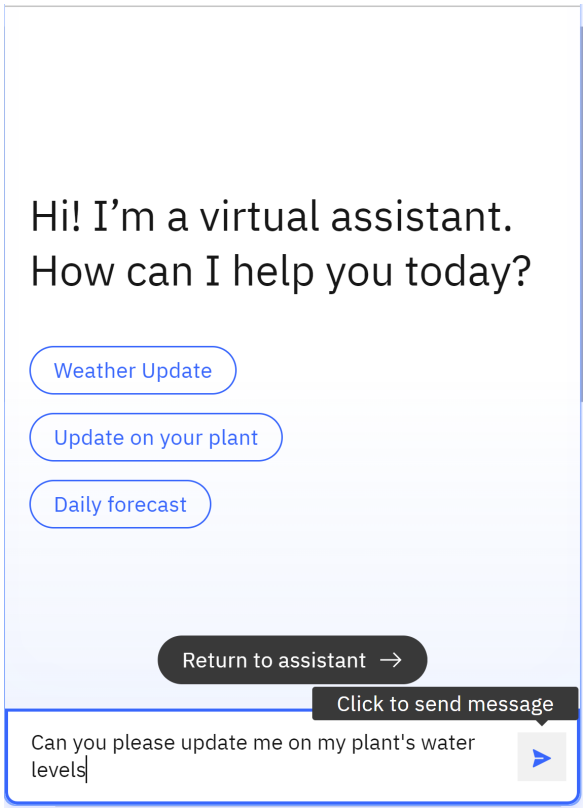


Figure 2: AI Irrigation Chat bot

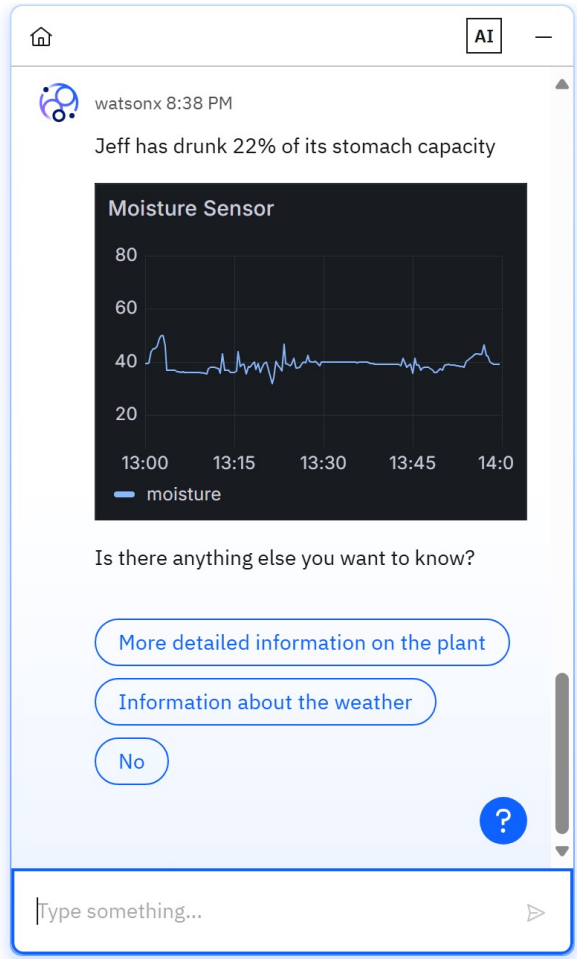
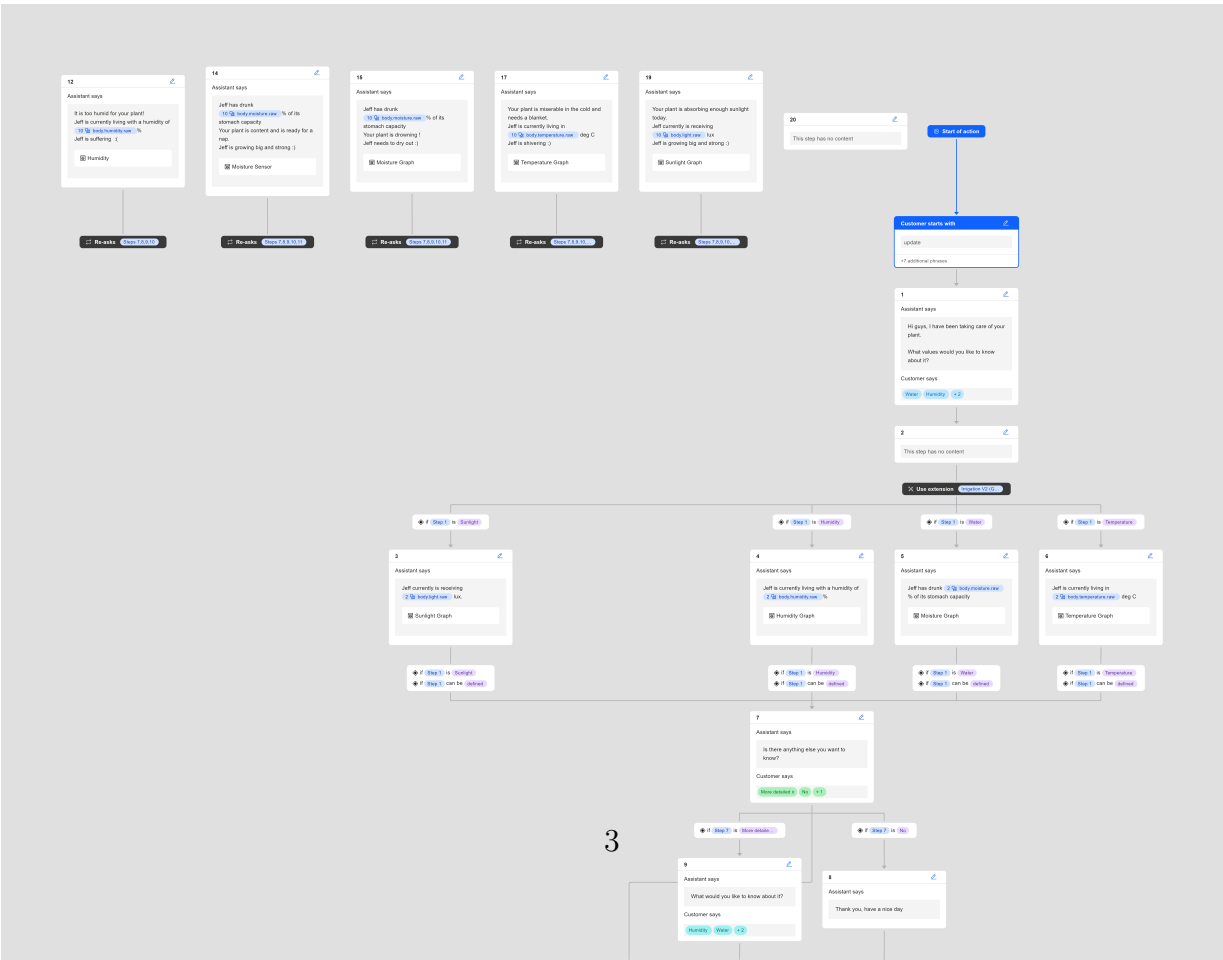


Figure 3: Brief overview on plant moisture levels



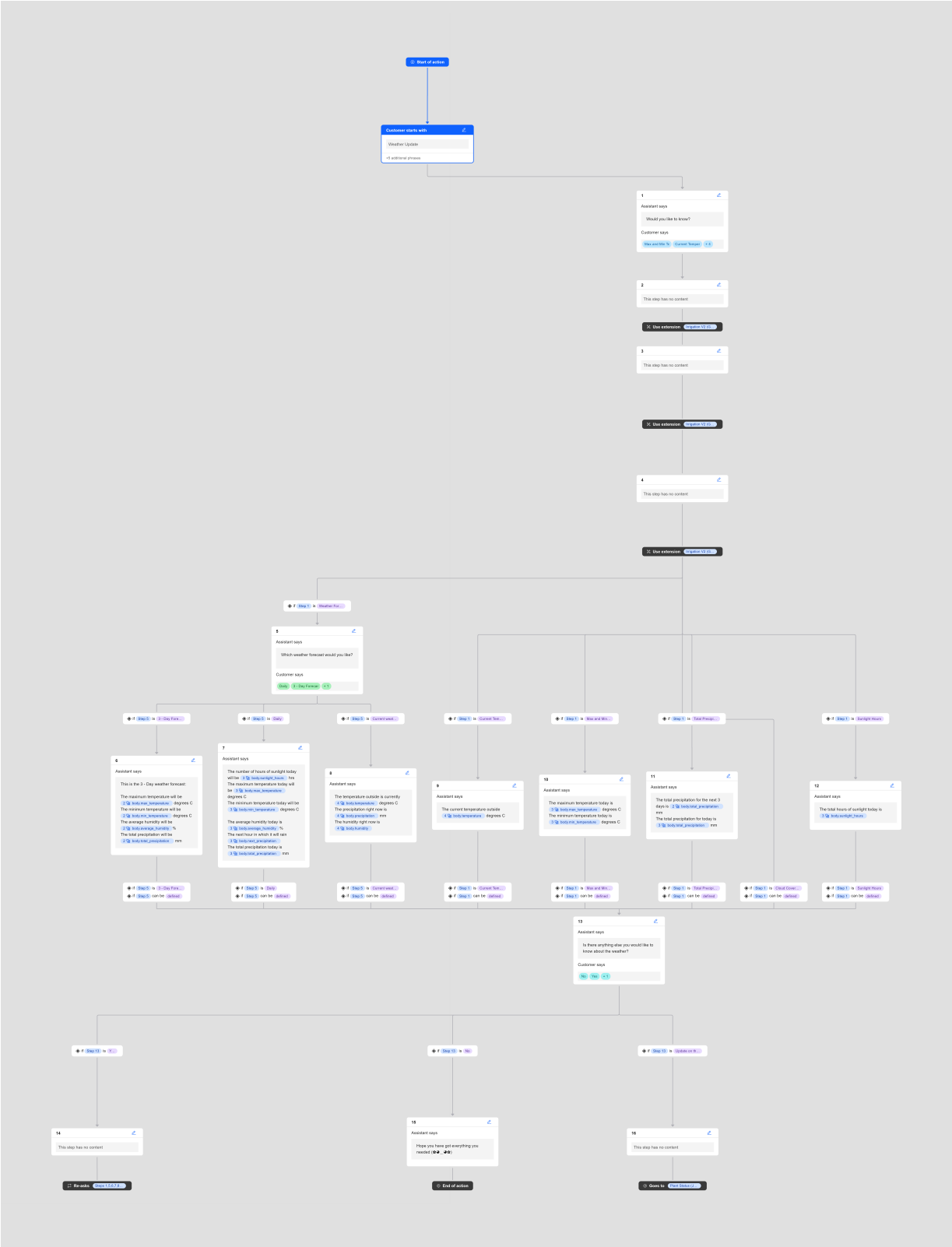


Figure 5: Overview of the weather update flowchart

4 Backend

4.1 Overview

The backend system for this project is designed to support a smart plant care system, integrating various components to collect, process, and analyse data from plant sensors, weather forecasts, and user interactions. The system is built with scalability, modularity, and efficiency in mind, utilising modern free open-source technologies.

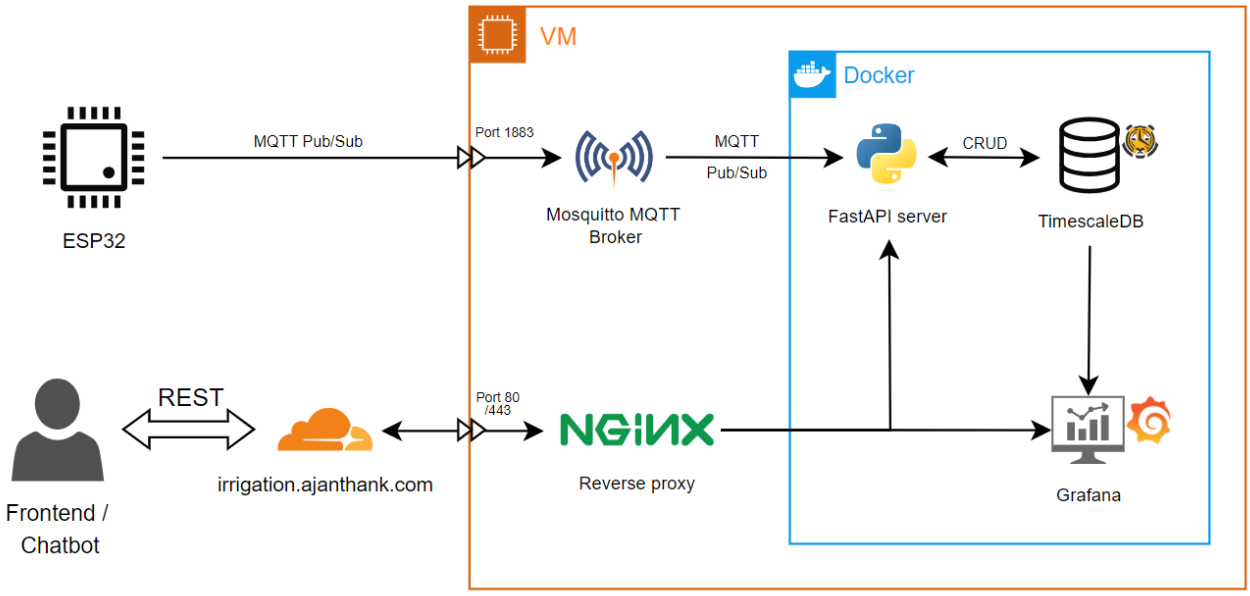


Figure 6: Backend system architecture

4.2 Communication

4.2.1 ESP32 and Server Communication

For communication between the ESP32 micro-controller and the server, the MQTT protocol was chosen after careful consideration of use-cases and alternatives. The primary communication was identified as:

1. Transmission of sensor data at regular intervals (approximately every 5 minutes) from the ESP32 to the server

2. Dispatching location commands from the server to the ESP32

While this communication is bidirectional, it is predominantly unidirectional, with the majority of data flowing from the ESP32 to the server.

Three main protocols were evaluated for this purpose: HTTP, WebSockets, and MQTT. MQTT was ultimately chosen due to its superior suitability for IoT applications. Its publish/subscribe model aligns well with the use case, allowing devices to publish sensor data to specific topics and subscribe to command topics, facilitating real-time communication without the overhead of constant polling (which would be required by HTTP).

MQTT's design for low-bandwidth and unreliable networks makes it well suited for IoT devices like the ESP32. The protocol's low overhead, with minimal packet size, helps with conserving battery life, especially when compared to Websockets, which requires maintaining a constant stateful connection. The scalability offered by MQTT's broker-based architecture was another significant factor in its selection. As the system expands to incorporate more devices, the MQTT broker can efficiently manage the increased communication load without requiring fundamental changes to the system architecture.

4.2.2 Frontend and Chatbot Communication

For the server's communication with the frontend and the Watson chat assistant, HTTP-based APIs were employed. This decision was primarily influenced by the constraints of the Watson chat assistant, which exclusively supports HTTP communication and necessitates an OpenAPI specification for integration.

4.3 Server

The server was implemented using FastAPI, an open-source Python backend framework. It was chosen for its high performance, comparable to other frameworks such as NodeJS and Go, and its native OpenAPI support (see https://irrigation.ajanthank.com/docs, which facilitates simple integration with the Watson chat assistant.

Core functionalities:

1. **Sensor Data Processing:** The server hosts an MQTT client that subscribes to sensor topics, and when data is published by ESP32 devices, it is processed and stored in the TimescaleDB database. This data includes light intensity, temperature, moisture and humidity readings.

2. **Weather Data Management:** A scheduled task fetches weather forecast data from the tomorrow.io API every three hours. This data is processed and stored in the database, providing cruicual information for plant care decisions.

3. **Plant Status Decision Engine:** Periodically, the server analyses the collected sensor data (from the last day), weather forecasts, and plant requirements to determine the optimal care actions. Commands to move to a location, as appropriate, are then sent to the ESP32 devices via MQTT.

4. **API Endpoints:** The server provides HTTP endpoints for the frontend and chatbot to retrieve plant status (e.g. if an intervention is required), weather data, and historical information.

4.3.1 Decision-Making flow

The decision-making process involves several steps, as illustrated in Fig. 7:

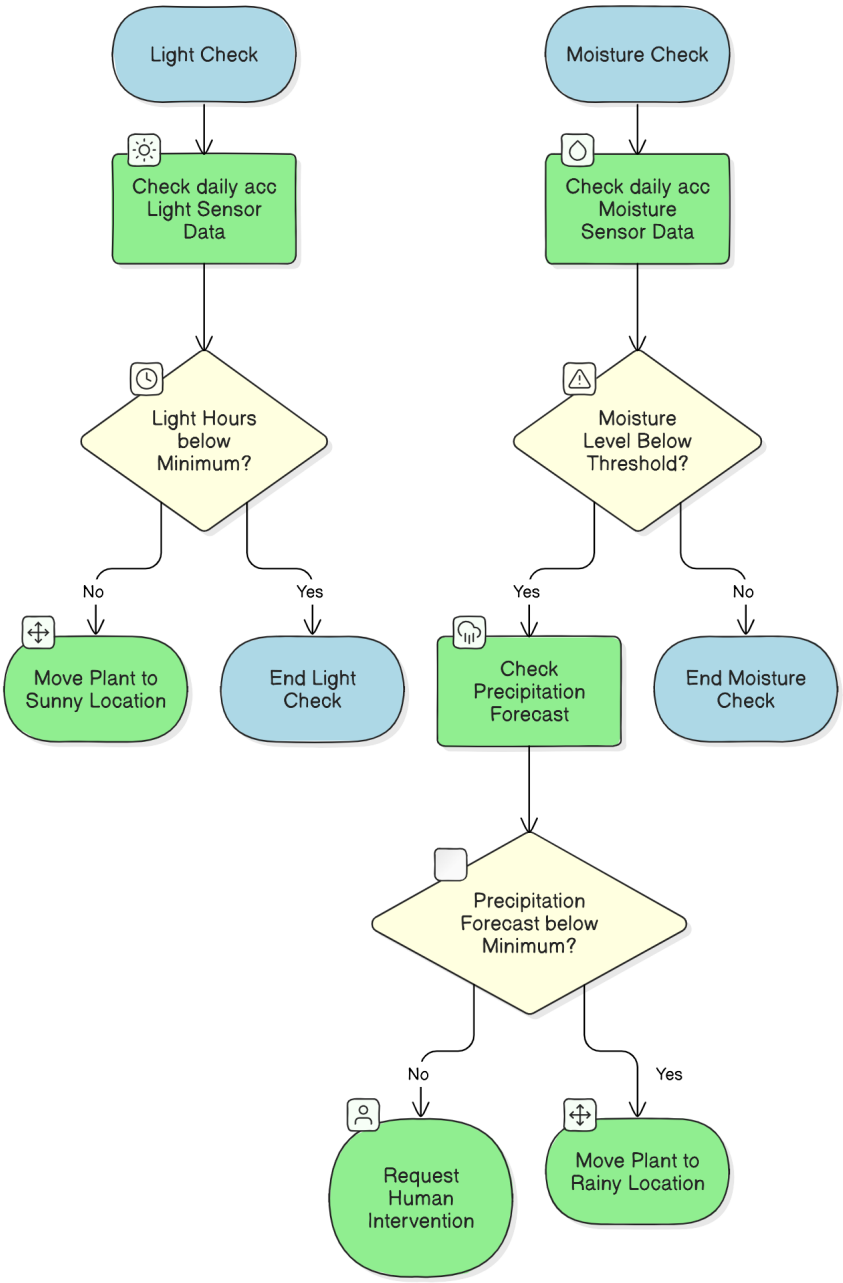


Figure 7: Example decision making logic

This flow chart outlines the logic used by the server to determine when and how to intervene in plant care. It takes into account daily accumulative sensor readings, weather forecasts, and plant-specific requirements to make an informed decision on the plant status, and location it needs to move to.

4.4 Database

The selection of an appropriate database system was driven by the predominance of time-series data in the project, primarily consisting of sensor readings collected at regular intervals. After an evaluation of specialised time-series databases, TimescaleDB was chosen due to its ease of use and versatility, being built on PostgresSQL, whilst still being performant with time series data. Being built on PostgresSQL, it also supports regular relational tables, allowing use of the

same database to store other tables like ‘devices‘ and ‘plants‘ in 8, unlike InfluxDB, which is another commonly used database for IoT applications.

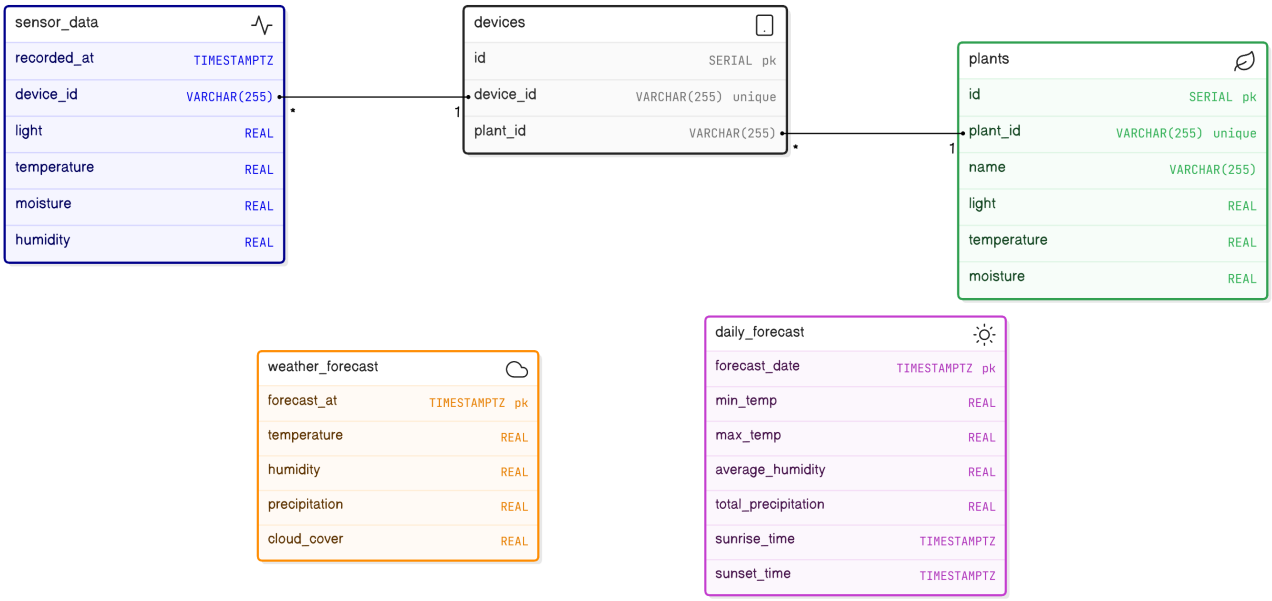


Figure 8: Database schema

The database schema is designed with scalability and extensibility in mind. The primary tables in the schema include:

1. **sensor_data**: A hypertable (TimescaleDB’s version of a partitioned table) where all sensor readings from an ESP32 are stored
2. **devices**: A table that stores information about each plant care device, including its unique identifier and associated plant.
3. **plants**: A table containing data about different plant types and their optimal growing conditions.
4. **weather_forecast**: Table used to store hourly weather forecast data.
5. **daily_forecast**: Table used to store daily forecast data, which is found by processing hourly data - reduce expensive larger queries on hourly table

This schema design allows for the definition of optimal conditions for any number of plant types, with each plant potentially associated with multiple devices. This flexibility supports scenarios where multiple robots might be caring for the same type of plant.

To optimise system performance and manage costs associated with external API calls, weather forecast data is stored in the database. This approach allows for a reduction in the frequency of API calls to external weather services, which becomes increasingly important as the number of devices in the system grows.

Data retention policies have been implemented to manage the growth of time-series data. For sensor data, a retention period of 30 days has been set, after which the data is automatically

aggregated into hourly averages. Forecast data, in contrast, is deleted after 7 days, as this is not relevant data to store long term for this project. This policy balances the need for historical data analysis with database performance and storage considerations.

4.5 Deployment

A containerised approach has been adopted for the deployment of the entire backend system, see Fig. 6. This encapsulation ensures environment consistency, minimises conflicts between components, and simplifies dependency management. The isolation provided by containers also enhances security by limiting the potential impact of vulnerabilities in one component on the rest of the system.

The containerised backend is deployed on a cloud virtual machine, leveraging the scalability and reliability offered by cloud infrastructure. Only essential ports (80/443 for HTTP/HTTPS and 1883 for MQTT) are exposed on the host machine, minimising the attack surface. An Nginx reverse proxy handles incoming traffic, adding an additional layer of security and allowing for flexible request routing.

5 Sensors

For each sensor, the minimum and maximum readings were taken and scaled from 0-100%. For any given plant, the optimal conditions are stored into the server and appropriate bounds are created from 0-25%, 26-75% and 76-100% respectively returning the status -1, 0 and 1. These status updates then triggers responses in the robot's movement, text to speech and alternative communications with the chat bot. Each sensor was tested individually in a controlled environment before the values had been scaled and tested using the plants. All sensors worked integrated together using the ESP 32, and the final plant that was chosen for the demo allowed for easy access to remove water from the pot (after non-stop repeated watering).

5.1 Temperature and Humidity Sensor

The DHT11 sensor is integrated for measuring ambient temperature and relative humidity. This sensor utilises a thermistor and a capacitive humidity sensor to provide digital outputs. It operates within a temperature range of 0 to 50°C with an accuracy of ±2°C, and a humidity range of 20 to 90% RH with an accuracy of ±5% RH. The data from the DHT11 is critical for environmental monitoring and control within the robotic system. If the readings from the temperature and the humidity sensor were not returning the status 0, such that it was in the optimal condition then the text to speech function alerts the user with the appropriate steps and will automatically move into conditions to better suit itself.

5.2 Light Intensity Sensor

The BH1750 light intensity sensor is employed to measure the illuminance in lux. It uses an I2C interface for communication, providing a wide range of measurement from 0 to 65,535 lux. The sensor has a spectral response close to that of the human eye, ensuring accurate light measurement. The BH1750's data helps in optimising the plant's exposure to light by positioning them to receive optimal sunlight for photosynthesis.

5.3 Capacitive Moisture Sensor

A capacitive soil moisture sensor is used to determine the volumetric water content of the soil. Unlike resistive sensors, the capacitive sensor avoids corrosion issues, ensuring longevity and reliability. It operates by measuring the dielectric permittivity of the surrounding medium, which varies with soil moisture. The sensor outputs analog signals that are read by the ESP32 microcontroller, providing real-time soil moisture data to guide irrigation decisions.

Finally, similar to the use of the other sensors, if the readings from the capacitive moisture sensor are not in the optimal range the corresponding text to speech phrases are triggered to alert the user of the problem. The capacitive moisture sensor is also paired with the weather component, such that if the precipitation in the next three days is not enough to keep the plant in it's optimal conditions then the text to speech is also triggered and corresponding messages can be seen on the chat bot.

6 Motors

There was a total of four motors used powered with four AA batteries totalling 6V.

6.1 Movement

The movement of the robot was planned out for it to move to four different corners of an environment containing all the different conditions for the plant; rain, shade, sunlight and the resting position i.e. where the robot would be charged.

The server sends the updated state that the robot should move to through the ESP and the robot does this by keeping track of the difference between the target and current location (coordinate). To allow for movement between any two pairs of coordinates, the robot tracks its orientation, whether it is facing North, East, South or West and turns accordingly to reach its target location. The current location is updated after the completion of each movement.

As a fail-safe additional manual control was added so that the robot could be moved from the website which is done by the server directly controlling how the robot moves rather than using the coordinates.

6.2 Intelligent Plant Movement - Future Work

To determine the optimal location for plant movement based on sensor readings (light, temperature, humidity, and soil moisture), we outlined a multi-objective optimisation approach. First, we normalised the sensor readings to a common scale for comparability:

$$x'_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

We then defined objective functions for each sensor, evaluating the suitability of a location:

$$f_{\text{light}}(x) = 1 - |x'_{\text{light}} - t_{\text{light}}|$$
$$f_{\text{temp}}(x) = 1 - |x'_{\text{temp}} - t_{\text{temp}}|$$
$$f_{\text{humidity}}(x) = 1 - |x'_{\text{humidity}} - t_{\text{humidity}}|$$
$$f_{\text{moisture}}(x) = 1 - |x'_{\text{moisture}} - t_{\text{moisture}}|$$

These functions were combined into a single composite objective function using a weighted sum, reflecting the importance of each factor:

$$F(x) = w_{\text{light}} f_{\text{light}}(x) + w_{\text{temp}} f_{\text{temp}}(x) + w_{\text{humidity}} f_{\text{humidity}}(x) + w_{\text{moisture}} f_{\text{moisture}}(x)$$

Finally, we maximised this composite function to find the optimal location:

$$x^* = \arg\max F(x)$$

This approach ensures precise environmental conditions for enhanced plant health and growth. The weighting of each environmental factor can be parameterised based on each specific plant species and circumstance, or a more dynamic approach could be followed in which the weights are continuously adjusted. Features that could be taken in to account in a dynamic weighting system could include the the real-time historical performance of the irrigation robot, the detection of some sensor variables being more "stubborn" to change than others, or the changing needs of the specific plant species at different stages of its life-cycle. While this was not integrated within the final design, future scaling of the project could benefit from utilising this methodology.

Furthermore, if the lux measured from the sensor was not in the optimal range for the plant, the corresponding status would be received. An appropriate text to speech comment is played out to the user, and the plant will either move towards or away from the light until it reaches satisfactory conditions.

7 Circuit and Assembly Diagrams

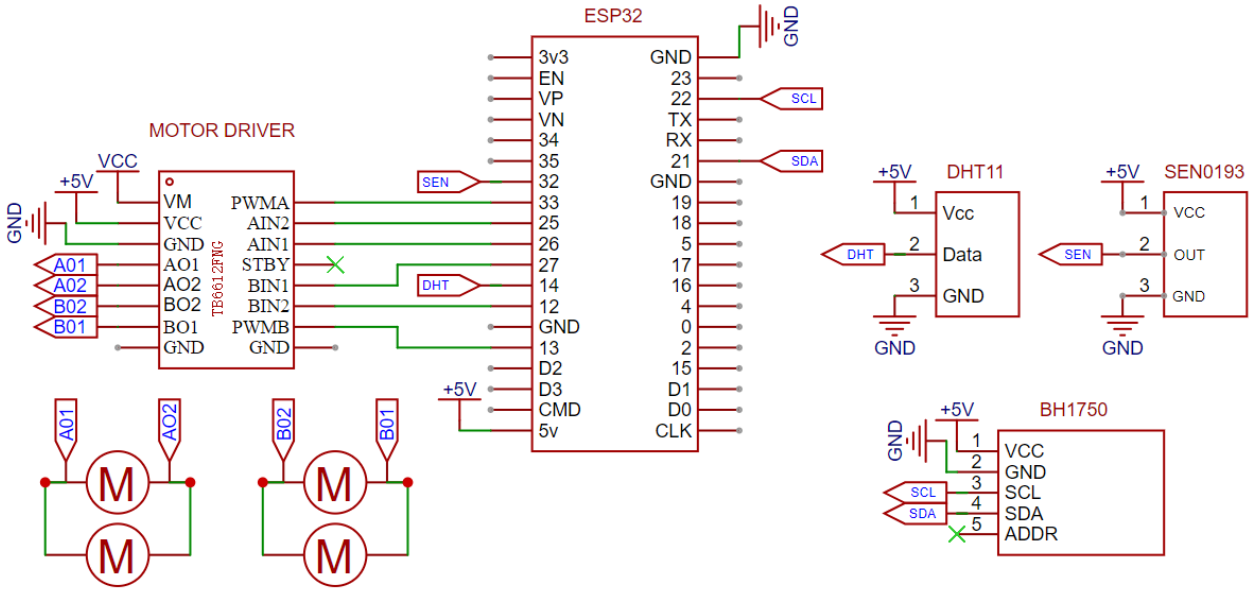


Figure 9: High-level Schematic Diagram

7.1 Component Outline

The primary components used in the robot's circuit include:

- DHT11 Temperature and Humidity Sensor
- BH1750 Light Intensity Sensor
- SEN0193 Capacitive Moisture Sensor
- ESP32-WROOM Micro-controller
- TB6612FNG Motor Driver
- 3-6V Motors (x4)

7.2 Sensor Connections

The light sensor connected via the I2C interface with default pin settings Interfaced via I2C (GPIO 21 for SDA and GPIO 22 for SCL). The DHT11 connected via GPIO 14 and the moisture sensor by GPIO 32. The robot uses an ESP32 micro-controller, which is responsible for different aspects of the robot's operations. The ESP32 handles sensor data collection and server-communication, while also managing motor control and movement.

7.3 Direction Control

AIN1/AIN2 (Motor A) and **BIN1/BIN2 (Motor B)** determine motor direction. For example:

- Forward: AIN1 = LOW, AIN2 = HIGH; BIN1 = HIGH, BIN2 = LOW.
- Backward: AIN1 = HIGH, AIN2 = LOW; BIN1 = LOW, BIN2 = HIGH.

ESP32 to Motor Driver (TB6612FNG)

- **AIN1 (GPIO 26), AIN2 (GPIO 25)** for Motor A direction control.
- **BIN1 (GPIO 27), BIN2 (GPIO 12)** for Motor B direction control.
- **PWMA (GPIO 33), PWMB (GPIO 13)** for motor speed control using PWM.

7.4 Speed Control

PWMA and PWMB: These pins receive PWM signals from the ESP32 to control the speed of the motors. The PWM duty cycle determines the motor speed, with higher duty cycles resulting in faster speeds. A 50% duty cycle (analogWrite(PWMA, 128)) results in moderate speed.

8 Code

The GitHub contains all the code which has a README explaining it.

9 Robot Design

9.1 Chassis

Originally a tracked chassis was to be used to carry the electronics and the plant, however when testing there was an issue of the chassis not moving in a straight line and hence the tank-like wheels and the two motors were replaced with regular wheels and four motors.

9.2 CAD

The following images are of the CAD designs for the exterior of the robot to carry the plant and hold all the electronics.

The lid of the container was designed to have a lip to ensure it sits securely on top of the container and an indent that was measured to fit the plant pot on top of the robot.

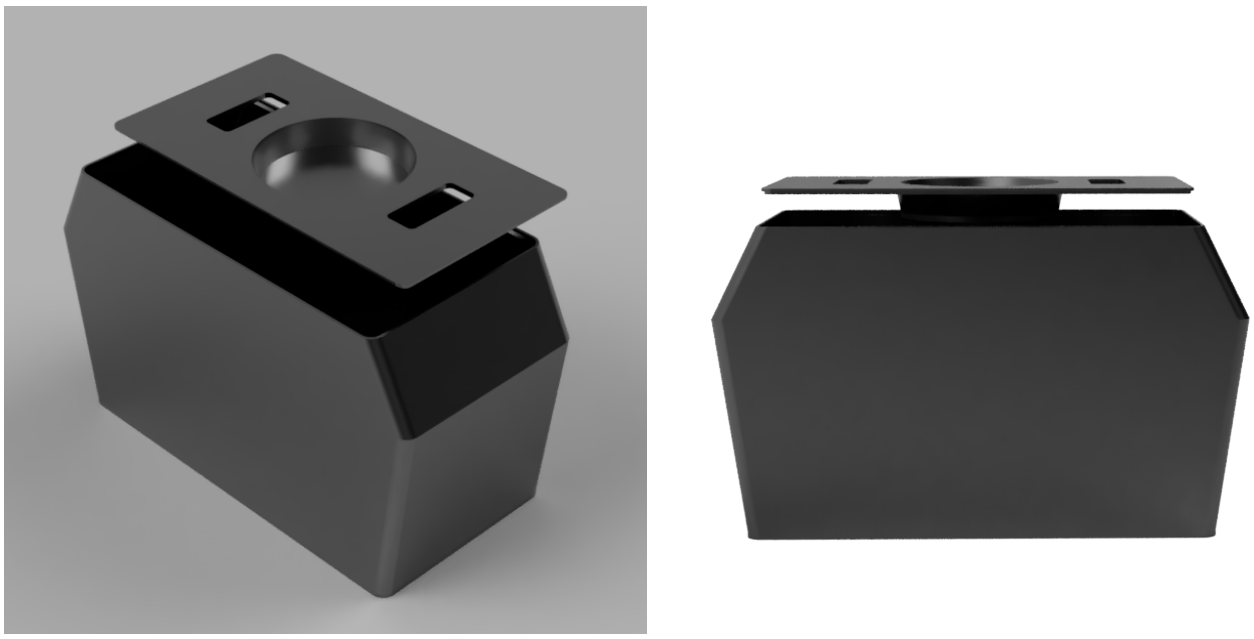


Figure 10: CAD

10 Record of meetings held

For the duration of the project the team met at least weekly starting from the 29th April until the 27th June. During this time we had a combination of meetings with our Imperial supervisor, our IBM coordinator and the team itself. Designated channels of communication had been set up between the team and the supervisor/ IBM Coordinator so that everyone was always informed of the updates to the project.

- 26th April - Internal Team Meeting to discuss initial ideas
- 29th April - IBM Coordinator Meeting to discuss project briefing and client expectations
- 3rd May - Internal Team Meeting to discuss division of tasks and smaller team objectives
- 7th May - Imperial Supervisor Team meeting to introduce the project, structured breakdown of tasks and timeline of events

- 10th May - Internal Team Meeting
- 14th May - Imperial Supervisor Team meeting to discuss the progress and initial struggles with the smaller tasks
- 17th May - Internal Team Meeting
- 21th May - Imperial Supervisor Team meeting to explain the progression of the robot
- 24th May - Internal Team Meeting
- 28th May - Imperial Supervisor Team meeting to clarify which points from the breifing had not yet been completed and the teams upcoming steps
- 24th May - IBM Coordinator Team Meeting to explain our progress and outline our aims and objectives and any further extensions to the project that is achievable in the time frame
- 29th May - IBM Coordinator Team Meeting to explain our progress and outline our aims and objectives and any further extensions to the project that is achievable in the time frame
- 4rd June - Internal Team Meeting
- 6th June - Imperial Supervisor Team meeting to clarify which points from the briefing had not yet been completed and the teams upcoming steps
- 11th June - Internal Team Meeting
- 18th June - Internal Team Meeting
- 25th June - Internal Team Meeting

The dates above show when full team meetings were called, however many meetings were called with smaller teams of 2 or 3 students to discuss individual breakdown of tasks.