Universitatea Politehnica din Timișoara

PIIRRIGATE: A SMART IRRIGATION SYSTEM

Candidate: Virgil-Alexandru CRIȘAN

Scientific coordinator: Conf. dr. ing. Răzvan BOGDAN

Session: June 2025

REZUMAT

Această lucrare descrie proiectarea și realizarea sistemului “PiIrrigate”. Acest sistem are ca scop eficientizarea consumului de apă si optimizarea culturilor agricole sau a grădinilor, prin utilizarea tehnologiilor moderne IoT și a comunicațiilor radio de tip LoRa. Pe măsură ce efectele încălzirii globale se fac din ce în ce mai resimțite, automatizarea si monitorizarea irigațiilor devine o necesitate. Proiectul vizează dezvoltarea unui sistem de monitorizare si control destinat fermierilor care doresc monitorizarea unor suprafețe mari de teren, dar acest sistem poate fi folosit si pentru sere inteligente sau grădinărit normal.

Proiectul utilizează o arhitectură bazată pe microcontrolere Raspberry Pi și T-Beam LILYGO ESP32 LoRa. Aceste componente sunt folosite pentru colectarea si transmiterea datelor in timp real. Sistemul permite monitorizarea parametrilor esențiali (umiditate, temperatură, umiditatea solului și cantitatea de ploaie) prin senzori care sunt conectați la nodurile ESP32. Dupa colectare, datele urmează a fi transmise către un gateway care comunică apoi cu un API web dezvoltat in .NET. Apoi datele urmează a fi stocate într-o baza de date PostgreSQL și trimise folosind SignalR către o aplicație web pentru a fi vizualizate în timp real de către utilizatori.

Utilizatorii au la dispoziție o interfață web care le permite atât vizualizarea datelor în timp real cât și vizualizarea datelor istorice și controlul manual al sistemului. De asemenea, acest sistem implementează un mecanism de înregistrare dinamica a nodurilor în rețea, lucru care permite extinderea facilă a sistemului.

Prin integrarea componentelor hardware și software într-o soluție coerentă, PiIrrigate demonstrează fezabilitatea și eficiența unui sistem IoT dedicat agriculturii inteligente, cu un impact potențial in reducerea consumului de apă și creșterea randamentului agricol.

ABSTRACT

This paper describes the design and implementation of the “PiIrrigate” system. The purpose of this system is to optimize water consumption and improve the management of agricultural crops or gardens by using modern IoT technologies and LoRa radio communications. As the effects of global warming become increasingly evident, the automation and monitoring of irrigation systems is becoming a necessity. The project aims to develop a monitoring and control system intended for farmers who need to oversee large areas of land, but it can also be used for smart greenhouses or regular gardening.

The project uses an architecture based on Raspberry Pi microcontrollers and T-Beam LILYGO ESP32 LoRa modules. These components are used for the real-time collection and transmission of data. The system enables the monitoring of essential parameters (humidity, temperature, soil moisture, and rainfall) through sensors connected to ESP32 nodes. After collection, the data is transmitted to a gateway, which then communicates with a web API developed in .NET. The data is stored in a PostgreSQL database and sent in real time via SignalR to a web application, where it can be viewed by users. Users have access to a web interface that allows them to visualize both real-time and historical data, as well as to manually control the system. Additionally, the system implements a dynamic node registration mechanism, enabling easy expansion of the network. By integrating both hardware and software components into a coherent solution, PiIrrigate demonstrates the feasibility and efficiency of an IoT-based system dedicated to smart agriculture, with the potential to reduce water consumption and increase agricultural productivity.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SNIPPETS

1. INTRODUCTION

1.1 CONTEXT

Agriculture is a vital sector that plays a crucial role in sustaining human life and the economy. Agriculture automation and optimization has become a major concern in recent years. As the global population continues to grow, the demand for food is increasing and the developing need for food along with the effect of climate changes are forcing the agricultural industry to adapt and innovate[1].

In the last 35 years, the wold has seen a doubling of the agricultural production. This has been achieved throught the use of different fertilizers, pesticides and herbicides. This doubling was associated with a 6.87-fold increase in nitrogen fertilization, a 3.48-fold increase in phosphorus fertilization and 1.68-fold increase in the amount of irrigated cropland [2]. In addition, the water consumoption is expected to increase by 50% by 2050 [3].

This project aims to address the challenges of water scarcity and the need of efficient irrigation systems by presenting the plan, the implementation, the results and future work of a system that can be used in different scenarios and is meant to help reducing the water consumption and increasing the agricultural productivity. The PiIrrigate project intends achuive this by developing an innovative irrigation system that leverages the power of IoT and LoRa radio communication technologies. The main focus of this project is to create a system that can be easly used in different agricultural settings, starting from small gardens to large farms and even smart greenhouses. Beside this, I wanted to create a system that is easy to use and can be extended with ease.

The ESP32 boards with sensors are responsible for colecting the data. Then data is sent using LoRa to another ESP32 board that acts as a gateway connected to a Raspberry Pi, which is responsible for sending the data to a web API. The web API is developed in .NET and is responsible for storing the data in a PostgreSQL database. The data is then sent to a web application using SignalR, which allows real-time communication between the server and the client. The web application is responsible for displaying the live data and the historical data and also for providing a way to control the system manually and to add new nodes to the system.

This system takes advantage of the LoRa radio communication technology, which allows for long-range communication with low power consumption. Meaning that the system can be used in remote areas and it will work even if the internet connection is not available to all the nodes. The Raspberry Pi is the only component of this system that needs to be connected to the internet. Other components can be scatered on a area of 10km or more, depending on the environment and the node setup (mesh or star topology).

1.2 MOTIVATION

The reason why I choose to create such a system was fueled by my passion for technology and smart agriculture. Besides this, I like to observe the data path, from the moment it is collected by the sensors, to the moment it is displayed in a web application. I have always been intested in pieces of technology that can be used to solve real world problems and now I had the chance to create such a system. Initaially, I wanted to create a smaller system for my own graden, but then I reazied that this system can be used in a lerger scale and can be adapted to different scenarios.

This project is also motivated by the opportunity to apply core concepts of computer science, such as API design, data management, distributed systems and IoT to build a cost-effective ans scalable smart irrigaiton system. This project intends to provide not only a system for irrigation, but also a platform that can be used in different scenarios, such as smart greenhouses, meteorological stations, or even smart cities.

1.3 OBJECTIVES

The main objective of this project is to develop a system that leverages IoT, cloud services, and modern softwware development principles to optimize irrigation processes and reduce the water usage in agriculture. The focus of this project is to provide a theoretical approach of the system, but also to present the implementation details and the results of the system. At the and of this project the result should comprise the system architecture, including the hardware and software components, and the implementation details of the system. The implementation details will include the hardware and software components used, the communication protocols used, the data management and storage, and the user-interface design. To be more specific, the objectives of this project are:

- Develop a cost-effective and scalable smart irrigation system that can be used in different agricultural settings.
- Leverage IoT and LoRa radio communication technologies to create a system that can operate in remote areas with low power consumption.
- Provide a user-friendly web application for real-time monitoring and control of the irrigation system.
- Implement a modular architecture that allows for easy extension and customization of the system.
- Ensure data security and privacy by implementing best practices in API design and data management.

1.4 SCOPE AND LIMITATIONS

Scope of this project includes the design, implementation, and evaluation of a smart irrigaiton system. The system will be designed to collect data from enviromental sensors,

such as soil moisture, temperature, and humidity, process this data throught a web API, and provide a user friendly web application for real-time monitoring and control.

Key fetures of the system include:

- Data collection from environmental sensors using LILYGO Meshtastic AXP2101 T-Beam V1.2 ESP32 LoRa boards.
- Long-range communication using LoRa radio technology.
- Data processing and storage using a web API and PostgreSQL database.
- Real-time monitoring and control through a web application.
- Modular architecture for easy extension and customization.

Despite the comprehensive scope of this project, there are some limitations that need to be considered:

- Hardware limitations: The system is designed to work with specific components, any changes to the hardware may require significant changes in the project.
- This specific system will be tested in a controlled environment, so the result may not be representative of real-world scenarios.
- Reliability and calibration of the hardware components are assumed to be optimal, but they are not guaranteed.
- Weather prediction and other external factors are outside the scope of this project, the system will not take into account weather forecasts.
- Energy consumption and power management analysis are not included in the scope of this project.
- The UI design is focused on functionality and usability, but it may not be visually appealing or optimized for all devices.
- Machine learning and advanced data analytics are not included in the scope of this project.

1.5 METHODOLOGY OVERVIEW

The metodology adopted for this project is structured around the typical sofware development lifecycle, beggining with the requirements gathering and analysis, followed by the design, implementation, testing, and evaluation phases. This approach ensures that the system is developed in a systematic and organized manner, allowing for the creation of a robus and reliable smart irrigation system.

System requirements were gathederd based on on a research of modern irrigaiton practicies, with a great focus on the water consumoption usage and the need of real-time monitoring and control. These requirements helped identifying core functionalities of the

system, such as data collection from environmental sensors, processing, storage, and user interface design. Non-functional requirements, such as reliability and scalability were also taken into account during the requirements gathering phase.

The system architecture was designed using a layered approach. The hardware layer includes the sensors, the ESP32 boards and the Raspberry Pi. It is responsible for environmental data collection. The communication layer includes all protocols and technologies used for data transmission, such as LoRa, MQTT and HTTP. The data processing layer includes the IoT Hub, the webApi and the PostgreSQL database. Finally the user interface layer includes the web application that provides real-time monitoring and control of the system.

2. STATE OF THE ART

2.1 INTRODUCTION

The state of the art chapter provides an overview of the current state of smart irrigation systems and their applications in agriculture. This chapter will explore the existing technologies, methods, and solutions used in smart irrigation. It will also highlight the gaps and challenges in the current systems, and how the PiIrrigate project aims to address these issues.

2.2 EXISTING SMART IRRIGATION SOLUTIONS

2.2.1 TYPES OF SMART IRRIGATION SYSTEMS

There are several types of smart irrigation systems used in modern agriculture:

- **Weather-Based Controllers**
  These system take advantage of available weather data, such as temperature, humidity, and rainfall forecasts, to optimize irrigation schedules. This is not the most precise method as it does not use data from the soil, but it represent a good alternative for large areas.

- **Soil Moisture-Based Controllers**
  Soil moisture based controllers use sensors that are placed in the soil to measure the moisture level and adjust the irrigation accordingly. These are more precise than weather-based controllers, as they take into account the actual moisture level in the soil, but they require more maintenance and calibraiton [4].

- **Hybrid Systems**
  Many modern systems utilize a hybrid approach, combining data from both weather feeds and soil moisture sensors for more accurate and resilient irrigation decisions. Some research also explores "hybrid" in terms of integrating different energy sources (e.g., solar and wind) to power the systems or combining various irrigation methods (like drip and sprinkler) under one smart control[5].

  The PiIrrigate project place itself in the category of hybrid systems, using both soil moisture sensors and weather sensors to collect data.

  Some of the most popular hybrid smart irrigation systems include:

  - **Netafim's Precision Irrigation System**
    This system cobines data from soil moisture and flow sensors with sattelite weather data and predictive analytics to optimize the irrigation proccess.

    Key features include:

    * Real-time monitoring of soil moisture levels and weather forecasts.

* Automated irrigation scheduling based on weather forecasts.
* AI-based algorithms to optimize the irrigation timing and duration.

– **CropX Smart Farming System**
CropX is a cloud based platform that integrates soil moisture sensors, weather data and machine learning algorithms to optimize the irrigation process.
Key features include:
* Irrigation recomandations based on soil variability, crop type and weather.
* Farmers can apply recommandations or integrate with automated irrigaitons contrllers.
* Easy to scale and adapt to different farm sizes from small to large-scale farms.

– **Toro EVOLUTION® Series Controller with Smart ET Sensor**
Combines basic sprinkler system hardware with smart sensors and connectivity, offering both manual and intelligent irrigation options. The evaporation sensors are used to measure the amount of water lost throught evaporation and adjust the irrigation accordingly.
Key features include:
* Can be programmed manually or connected to a local weather station.
* Smart ET sensor measures evaporation rates and adjusts irrigation schedules.
* Compatible with smart devices for remote monitoring

2.3 COMPARATIVE ANALYSIS OF SMART IRRIGATION SYSTEMS

The table below provides a comparative analysis of some of the most popular smart irrigation systems available today and the PiIrrigate system.

Feature	**Netafim**	**CropX**	**Toro ET**	**PiIrrigate**
Irrigation Type	Drip	Any	Sprinkler	Custom
Automation	High (AI)	Med–High	Medium	Medium
Sensors	Soil, flow, weather	Soil, temp	ET sensor	Soil, temp, rain
Weather Data	Yes	Yes	Yes	Optional
Manual Control	App/cloud	App/web	Panel/app	Web UI
AI/Analytics	Yes	Yes	No	No
Scalability	Large farms	Small–large	Residential	Small farms/gardens
Cloud Sync	Yes	Yes	Optional	Yes
Use Case	Precision agri	Smart farming	Lawn care	Small to large scale
Cost	High	Med–High	Low–Mid	Low

Table 2.1: Comparison of Smart Irrigation Systems

2.4 IOT AND LORA IN AGRICULTURE

Agriculture is one of the biggest industries, critical to the global economy and food security. With the increasing food demant, the need for efficient and sustainable agricultural has grown in the last years. The IoT (Internet of Things) is is a key technology that can help to address these challenges by providing real-time data and insights into the agricultural processes. LoRa (Long Range) is a wireless communication technology that is well suited for agriculture applications. It has been inmvented in 2009 2010 by the company Cycleo, which was later acquired by Semtech in 2012 and until 2020 it was implemented in more than 100 million devices worldwide [6].

One approach to integrate IoT and LoRa in agriculture it's to use a layered architecture. This architecture consists of four layers: data aquisition, gateways, network and application. where the sensors and actuators are connected to a LoRa gateway, which is then connected to a cloud platform. This architecture is proposed in this paper [7] and it is a similar approach to the one used in the PiIrrigate project. The sensors collect data from the environment, such as soil moisture, temperature, and humidity, and send it to the LoRa gateway using LoRa radio communication. The gateway then sends the data to a cloud platform, where it can be processed and analyzed. The cloud platform can also send commands to the actuators, such as irrigation valves, to control the irrigation process based on the data collected by the sensors.

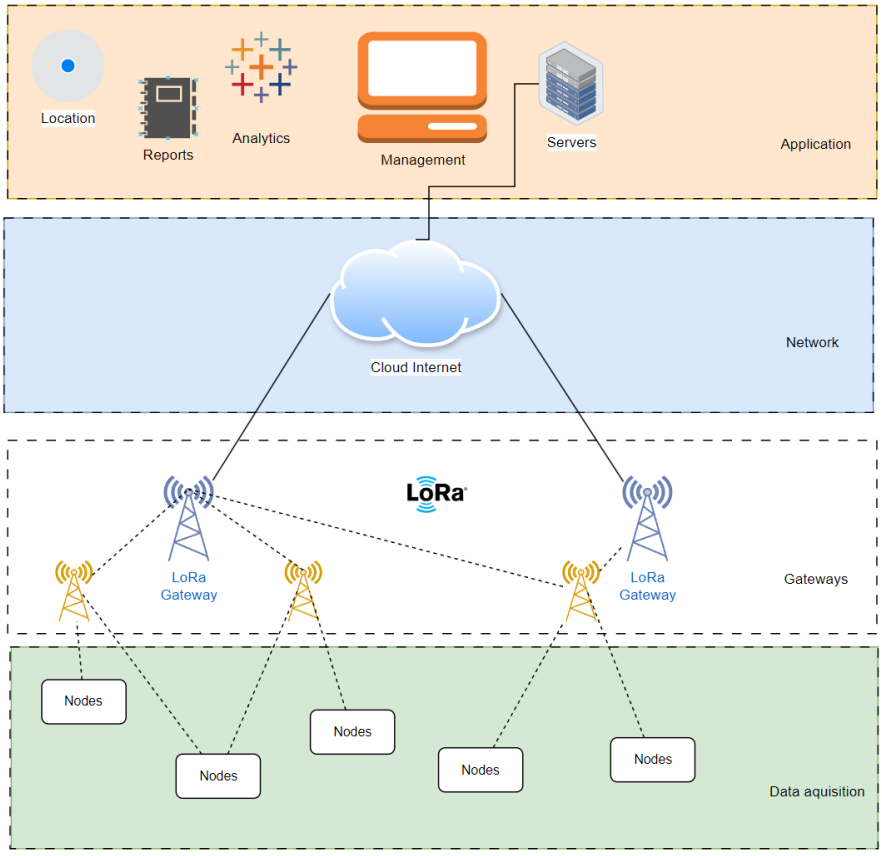

Figure 2.1: LoRa-based IoT Architecture for Agriculture

2.5 LORA VS OTHER WIRELESS TECHNOLOGIES

At this moment, theere are several wireless technologies available for IoT applications. After a comparative analysis of the most popular wireless technologies used in IoT, LoRa came out as one of the most suitable technologies for smart irrigation systems because of its long range, low power consumption and scalability. The table below compares LoRa with other wireless technologies commonly used in IoT applications, such as Wi-Fi, Bluetooth (BLE), Zigbee, and NB-IoT. In this section We will analyze the capabilities of these technologies and their suitability for smart irrigation systems. The comparison is based on several features, including range, data rate, power consumption, topology, license band, scalability, cloud integration, infrastructure cost and latency.

Wifi, Bluetooth and Zigbee have a decent data rate, and very good latency, but they have a very short range and high power consumption. This makes them more suitable for local applications, such as home automation or local sensor networks. NB-IoT(Narrowband IoT) represents a poteantial solution for smart irrigation systems, but it has a higher infrastructure cost and requires a cellular network, which may not be available in all areas.

Feature	**LoRa**	**Wi-Fi**	**Bluetooth (BLE)**	**Zigbee**	**NB-IoT**
Range	2–15 km	∼100 m	10–100 m	10–100 m	10–35 km
Data Rate	0.3–50 kbps	Up to 600 Mbps	Up to 2 Mbps	20–250 kbps	Up to 250 kbps
Power Consumption	Very Low	High	Very Low	Low	Low
Topology	Star (LoRaWAN)	Star	Star	Mesh	Star (cellular)
License Band	Unlicensed (ISM)	Unlicensed (2.4 GHz)	Unlicensed (2.4 GHz)	Unlicensed (2.4 GHz)	Licensed
Scalability	High	Low	Low–Medium	Medium	High
Cloud Integration	Easy via LoRaWAN	Native IP stack	Requires gateway	Requires gateway	Native via operator
Infrastructure Cost	Low	Medium	Low	Medium	High
Latency	High (seconds)	Low (ms)	Very Low (ms)	Low (ms)	Moderate
Ideal Use Case	Smart City Infrastructure	Local, high-speed data transfer	Short-range sensors/wearables	Indoor sensor networks	National-scale deployment

Table 2.2: Comparison of Wireless Technologies for Smart Irrigation

2.6 IDENTIFIED GAPS AND CHALLENGES

Despite the advancements in smart irrigation systems, several gaps and challenges remain:

- **High Costs**
  Many existing systems are expensive, making them inaccessible for small farmers or home gardeners.
- **Complexity of Use**
  Some systems require specialized knowledge to set up and maintain, which can be a barrier for adoption.
- **Limited Customization**
  Many commercial solution are designed for speicific crops and environments and that limits their applicability to diverse agricultural settings.
- **Data Integration Issues**
  Integration of data from different sources (e.g. wather, soil) can be more complex and it can lead to inefficiencies in the irrigaiton management.

In addition to the gaps mentioned, there are a few other issues that must be resolved, like the need for systems that can function in challenging environmental conditions, the need for dependable internet connectivity in remote locations, and data security and privacy issues.

The effect that smart irrigation systems have on the environment is another factor that must be taken into account. Even though these systems are made to use water as efficiently as possible, the manufacturing and disposal of electronic components can harm the environment. Therefore, developing sustainable and ecologically friendly systems is another challenge. This means that the systems should be built to last a long time and be simple to use, and that the parts used in them should be composed of recyclable materials.

2.7 SUMMARY

In summary, the state of the art in smart irrigation systems shows significant advancements in technology and methods, but also highlights several gaps and challenges that need to be addressed. The PiIrrigate project aims to fill these gaps by providing a cost-effective, easy-to-use, and customizable solution that leverages the power of IoT and LoRa

3. SYSTEM ARCHITECTURE

3.1 OVERVIEW

The Internet of Things (IoT) is a new technology that allows devices to connect remotely to achieve smart farming [8]. The IoT has a wide range of applications in agriculture, and it has began to influence many other industries as well, such as healthcare, transportation, and manufacturing. This was done to improve the efficiency and productivity of these industries, as well as to reduce costs and improve the quality of products and services[9].

The PiIrrigate smart irrigation system is build using a combination of hardware and software technologies. It leverages both low-power edge devices and cloud-based infrastructure to provide real-time monitoring and data collection, as well as remote control capabilities. The core components and their roles in the system are as follows:

- **ESP32 (LILYGO Meshtastic AXP2101 T-Beam V1.2 ESP32 LoRa)**
  The LILYGO Meshtastic AXP2101 T-Beam V1.2 ESP32 LoRa is a development board based on the ESP32 microcontroller, it is equipped with LoRa radio communication capabilities, Wifi, Bluetooth, GPS, and a battery management system. It is used to collect data from sensors and send it to the gateway using LoRa radio communication.

- **Raspberry Pi**
  Raspberry Pi is a small, affordable computer that can be used for a wide range of applications. The Raspberry Pi is the core of the PiIrrigate irrigation module, it is responsible for receiving data from the ESP32 nodes and sending it to Azure IoT Hub.

- **Azure IoT Hub**
  Azure IoT Hub is a cloud-based service that enables secure and reliable communication between IoT devices and the cloud. It manages the bidirectional communication between the Raspberry Pi and the web API.

- **Web API**
  The web API is developed in .NET and is responsible for receiving data from the Raspberry Pi, storing it in a PostgreSQL database, and providing a way to access the data. SignalR is used to provide real-time communication between the server and the client. It also provides a way to control the system manually and to add new nodes to the system.

- **PostgreSQL Database**
  PostgreSQL is a powerful, open-source relational database management system. It is used to store the data collected from the sensors and the schedules sent to the system. It also stores the user data and the configuration of the system. The database is hosted in Neon.

- **Web Application**
  The web applicaiton is developed using Angular and is responsible for displaying live, historical data and provide the user interface for controlling the system.

In the following sections, we will explore each of these components in more detail, starting with the hardware components and then moving on to the software components.

3.2 HARDWARE COMPONENTS

3.2.1 SENSORS

The PiIrrigate system uses a variety of sensors to collect data from the environment. The sensors used in the PiIrrigate system are:

- **Soil Moisture Sensor**
  The soil moisture sensor is used to measure the moisture level in the soil. It is used to determine when to irrigate the plants.
- **Temperature and Humidity Sensor**
  The temperature and humidity sensor is used to measure the temperature and humidity of the environment. It is used to determine the optimal conditions for plant growth and to adjust the irrigation schedule accordingly.
- **Rain Sensor**
  The rain sensor is used to detect rain and prevent irrigation during rainy weather. It helps to conserve water and prevent over-irrigation.
- **Water Flow Sensor**
  The water flow sensor is used to measure the flow rate of water in the irrigation system. It is used to monitor the water consumption.
- **Water Temperature Sensor**
  The water temperature sensor is used to measure the temperature of the water in the irrigation system. It is used to ensure that the water temperature is within the optimal range for plant growth.

3.2.2 ESP32 (LILYGO T-BEAM)

The T-Beam ESP32 LoRa Wireless Module is a compact development board thaht combines an ESP32 microcontroller, LoRa transceiver (SX1278), GPS module, and a battery management system into a single unit. This board is ideal for long-range, low-power IoT applications such as mesh networks, asset tracking, smart agriculture and environmental monitoring. Besides this, it has a built-in OLED display. The communication range of the LoRa transceiver can reach up to 10 km in open areas.

Figure 3.1: ESP32 (LilyGo T-Beam) module used in PiIrrigate

3.2.3 RASPBERRY PI 4 MODEL B

The Raspberry Pi 4 Model B is a small, affordable computer that can be used for a wide range of applications. It is equipped with a quad-core ARM Cortex-A72 processor, up to 8GB of RAM, and supports dual-band Wi-Fi and Bluetooth. The Raspberry Pi 4 Model B together with an ESP32LoRa is used in the PiIrrigate project as the gateway that receives data from the ESP32 nodes and sends it to IoT Hub.

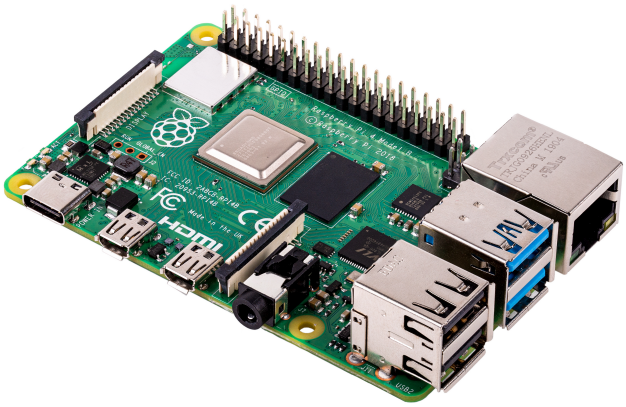

Figure 3.2: Raspberry Pi 4 Model B used in PiIrrigate

3.3 HARDWARE ARCHITECTURE

The hardware architecture of the PiIrrigate system consists of multiple ESP32 nodes that collect data from the sensors and send it to a gateway ESP32 connected to a Raspberry Pi. The Raspberry Pi acts as a gateway that receives data from the ESP32 nodes and sends it to Azure IoT Hub using MQTT protocol. The ESP32 nodes are connected to various sensors that collect data from the environment.

The ESP32 nodes are connected to the sensors and actuators as follows:

- Soil moisture (YL-69) sensor is connected to GPIO39.
- Temperature and humidity sensor (DHT11) is connected to GPIO13.
- Rain sensor (YL-83) is connected to GPIO36.
- Water flow sensor YF-S201 is connected to GPIO35.
- Water temperature sensor (DS18B20) is connected to GPIO32.
- The relay module that controls the water pump is connected to GPIO22.

In the figure below, you can see a simplified digram of the ESP32 pinout and connections.

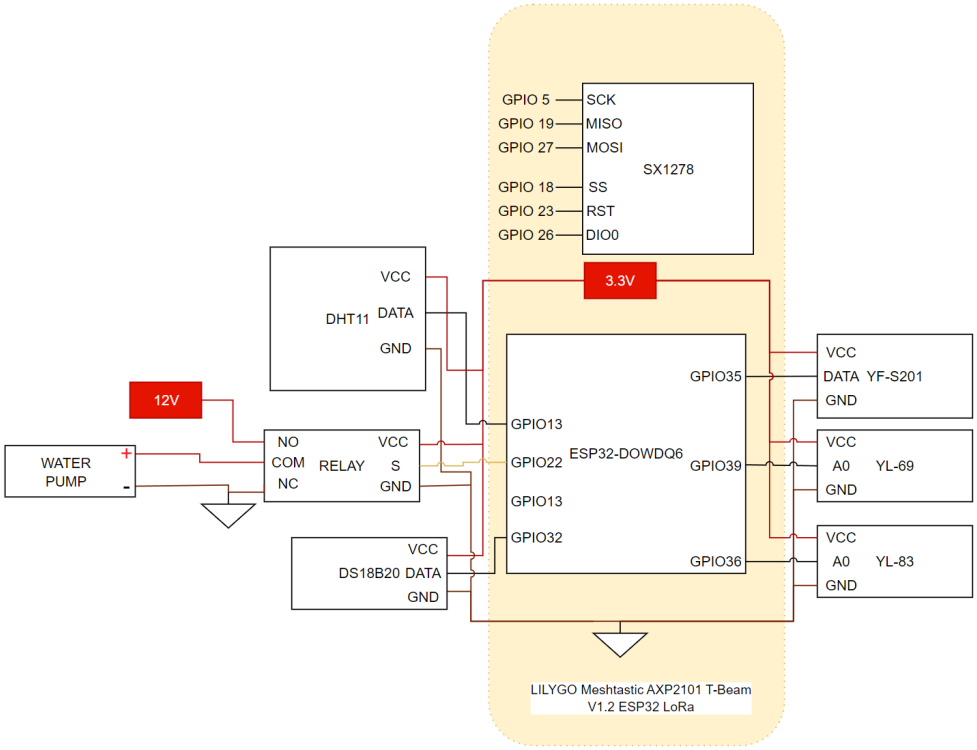

Figure 3.3: ESP32 pinout and connections

The communication between the gateway ESP32 and the Raspberry Pi is done using UART (Universal Asynchronous Receiver-Transmitter) protocol. The gateway ESP32 receives data from the nodes using LoRa radio communication and sends it to the Raspberry Pi

using UART. In the figure below it is described the connection between the ESP32 and the Raspberry Pi.

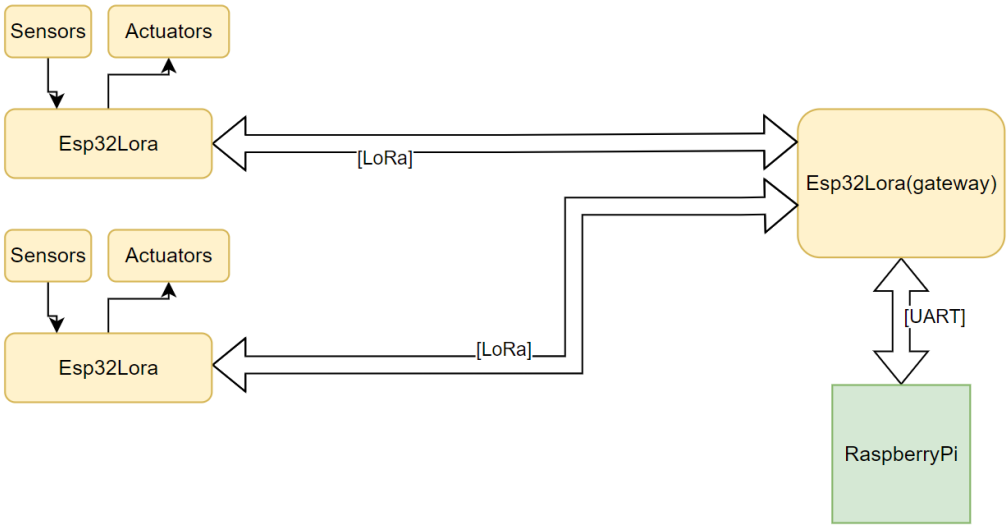

Figure 3.4: ESP32 Gateway connection to Raspberry Pi

3.4 SOFTWARE COMPONENTS

PiIrrigate system contains multiple software components that work together to provide a complete solution. The main sofware components of the PiIrrigate system are: ESP32 firmware, Raspberry Pi script, Web API together with the PostgreSQL database, and Web Application.

3.5 SOFTWARE ARCHITECTURE

The software architecture of the PiIrrigate system is a headless architecure. Headless architecture is a software development approache that separates the frontend and backend of an application, allowing for faster development and better customization [10].

Excluding the necessary code for the firmware running on the ESP32 nodes and the python script running on the RaspberryPi, the code can be divided into 2 main components: Frontend and Backend. Both backend and frontend architectures are layer based, with each layer having a specific role in the system. The backend is divided into 5 layers: Api, Service, Data, Models and Integration.

The Api layer is responsible for exposing endpoints, handling HTTP requests, performing input validation, and returning responses to the client. It represents the entry point of the backend application[11]. The service layer is the one that contains the bussiness logic of the application and is responsible for processing the data received from the Data layer and returning it to the Api layer. Data layer's only role is to interact with the database. It is responsible for data access and persistence and it is the only layer that interacts with the database as any other layer should not be aware of the database and should interact with the database only through the Data layer. Models layer is responsible for defining the data

structures used in the application. The integration layer is responsible for integrating with external services, such as Azure IoT Hub and SignalR.

The frontend is also divided into 5 layers: Presenation, State, Services, Models and Guards. The Presentation layer is responsible for rendering the user interface and handling user interactions. The State layers is responsible for storing the application state and managin the data flow between the components [12]. Service layers is responsible for interacting with the backend API and provifing ready to use data to the Presentation layer. Models layer, similar to the backend Models layer is responsible for defining the data structures used in the application[13]. Routing layer manages the navigation between different components[14] and the Guards layer makes sure that the user has the necessary permissions to access a specific route or component[15]. In the following figure, you can see the sofware architecure of the PiIrrigate system.

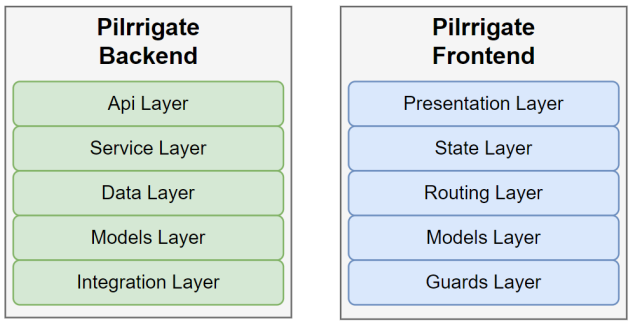

Figure 3.5: PiIrrigate Software Architecture

3.6 DATA FLOW

The data flow in the PiIrrigate system is as follows: Data is collected from the sensors connected to the ESP32 nodes. Then the data is sent to the gateway via LoRa radio communication. The gateway transmits the data to the Raspberry Pi using UART protocol and the Raspberry Pi send the data to Azure Iot Hub using MQTT protocol. The Web API receives the data from the Azure IoT Hub and stores in the PostgreSQL database and send the data to the web application using SignalR. Then the web application displays the data to the end user in real-time. The data flow is illustrated in the figure below:

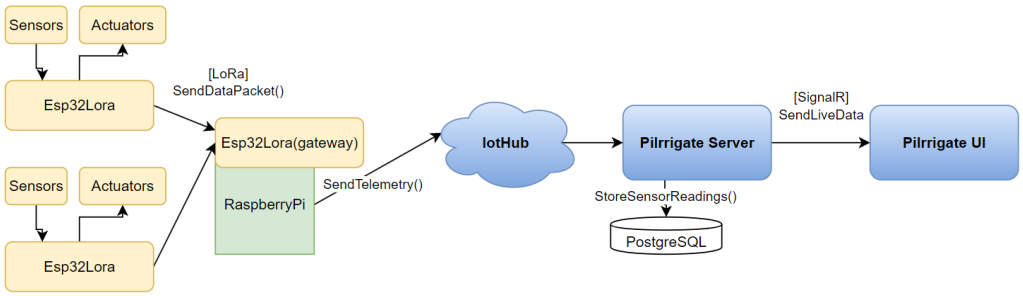

Figure 3.6: PiIrrigate System overview

4. IMPLEMENTATION

4.1 OVERVIEW

This chapter describes the implementation of the PiIrriate system. If the previous chapters described the architecture and the main components of the system, this chapter focuses on how these components are implemented in practice. The implementation is divided into several sections, each focusing on a specific aspect of the system.

4.2 ESP32 FIRMWARE

For the ESP32 development I had to choose between two main frameworks: ESP-IDF and Arduino. I chosed the Arduino framework because it is more user-friendly. As IDE I used Platform IO, which is a VSCode extention that allows to develop on many different platforms, including the ESP32[16].

4.2.1 DATA ACQUISITION

The data acquisition is done using the ESP32 nodes that collect data from the sensors. For each sensor type, I created a specific library that can be used to read the data from the sensor. Temperature and humidity data is collected using a DHT11 sensor. The communication with the sensor is done using 1-wire digital interface which involves 3 main steps[17]:

1. The microcontroller initiates communication by sending the start signal. The start signal is an 18 ms LOW signal followed by a 20–$40\,\mu$s HIGH signal.

2. The sensor responds a fixed LOW and HIGH handshake pattern, indicating that it is ready to send data. Usually the acknowledgment is a $80\,\mu$s LOW signal followed by a $80\,\mu$s HIGH signal.

3. After the handshake, the sensor sends a 40-bit data stream, which includes the humidity and temperature data. The bits are sent in a specific order: first the humidity data (16 bits), then the temperature data (16 bits), and finally a checksum (8 bits). Each bit is sent as a $50\,\mu$s LOW signal followed by a HIGH signal that lasts for either 26–$28\,\mu$s (for a 0 bit) or $70\,\mu$s (for a 1 bit). In code, for each bit, the microcontroller waits for the LOW signal to start, then waits for $30\,\mu$s then ig the signal is HIGH, the bit is a 1, otherwise it is a 0. The checksum is used to verify the integrity of the data received from the sensor.

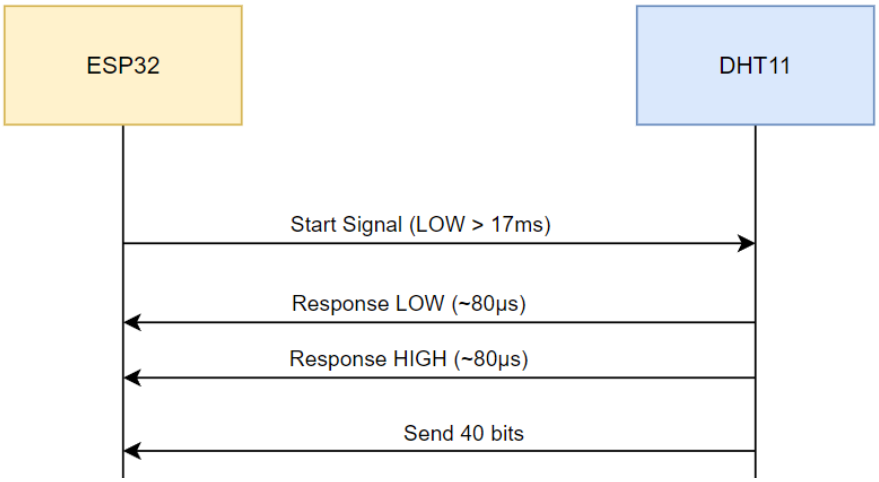

Figure 4.1: Steps in data aquisition from DHT11 sensor

For the soil moisure data acquisition, I used a resistive soil moisture sensor. The principle of operation is based on measuring the resistance of the soil. The sensor consists of two probes that are inserted into the soil. When the soil is dry, the resistance between the probes is high, and when the soil is wet, the resistance is low[18]. Then an ADC is used to measure the voltage across the probes, which is transofmerd into digital value. In this case, the ADC is a 12-bit ADC, which means that the digital value can range from 0 to 4095.

The raindrop data is collected using a resistive raindrop sensor. The principle of operation is similar to the soil moisture sensor, but in this case, the sensor consists of a plate with two conductive traces that are separated by a small gap. When the rain falls on the plate, it closes the gap and allows the current to flow between the traces. The is connected to an ADC which measures the voltage across the traces.

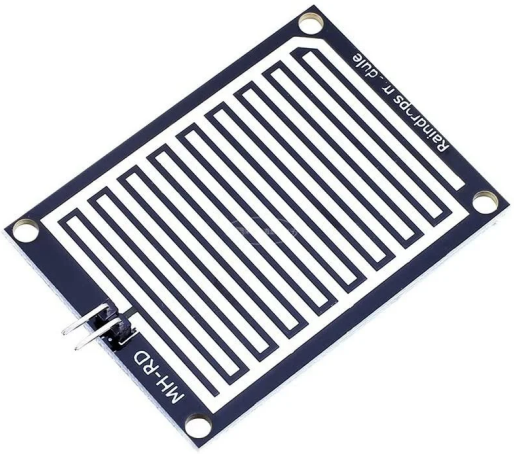

Figure 4.2: Raindrop sensor based on resistive principle

Since the water consumption is a very important aspect in agriculture, I wanted to add a water flow sensor to the system. For the purpose of this project I used an YF-S201 water flow sensor, which is a low-cost sensor that can be used to measure the flow rate of water in a pipe. The sensor consists of a plastic body with a turbine inside that rotates when water flows through it. The rotation of the turbine generates a pulse signal that can be used to measure

the flow rate of water. The sensor has a maximum flow rate of 30 liters per minute and a minimum flow rate of 1 liter per minute. The sensor has three wires: red (VCC), black (GND), and yellow (signal). At each rotation of the tubine, the sesnsor generates a pulse signal on the signal wire. This sensor will be used to check if the irrigation system is working properly or not. The ESP32 counts the number of pulses in a given time interval (e.g., 1 second) to calculate the flow rate. The flow rate can be calculated using the following formula:

$$\text{Flow Rate} = \frac{\text{Number of Pulses} \times 60}{\text{Time Interval (seconds)}} \tag{4.1}$$

Figure 4.3: Water flow sensor used in PiIrrigate

In my implementation the water source is a water tank that will be connected to the irrigation system. The tank will be equipped with a water temperature sensor. The purpose of monitoring the water temperature is to ensure that the water is not too cold or too hot for the plants. The water temperature sensor is a DS18B20 sensor, which is a digital temperature sensor that can be used to measure the temperature of liquids. The DS18B20 sensor uses the 1-wire digital interface to communicate with the ESP32. The sensor can measure temperatures from -55°C to +125°C with an accuracy of ±0.5°C.

Figure 4.4: DS18B20 water temperature sensor used in PiIrrigate

To be able to locate each node in the field, a GPS module is used. More precisely, I used a NEO-6M GPS module which is integrated in the LILYGO Meshtastic AXP2101 T-Beam V1.2 ESP32 LoRa development board.

4.2.2 DATA TRANSMISSION

Once the data is collected from the sensors, it needs to be transmited to the gateway. The ESP32 nodes use LoRa communication to send the data to the gateway. The sensor reading has a period of 10 seconds, which is the default value and it can be changed from the Web Application. The data collected from the sensors is serialized into a sensor reading structure, which is then is included in a LoRa packet structure that will be binary serialized and sent over LoRa radio communication.

Listing 4.1: LoRa packet structure

```
struct LoRaPacket {
    int packetCount;      // Packet count for tracking
    SensorData sensorData; // Sensor data to be sent
    int stationMac[6]; // MAC address of the device
};
```

Listing 4.2: Sensor reading structure

```
struct SensorData {
    time_t timestamp;
    float temperature;
    float humidity;
    float soilMoisture;
```

```
    float rainLevel;
    float waterTemp;
    float totalWaterFlow;
    float longitude;
    float latitude;
};
```

All the values regarding the environmental data represent the raw voltages read from the sensors. I chose to sent the raw values, because they can be used to calculate any related values in the web api. This allows a better flexibility in the future.

4.3 RASPBERRY PI AND ESP32 GATEWAY

The gateway ESP23 acts like a proxy between the nodes and the Raspberry Pi. It receives the LoRa packets from the nodes, and then it sends the data to the Raspberry Pi using Serial communnication by UART.

UART is an integrated cicuit that plays the most important role in serial communication. It containts a parallel-to serial converter and a serial-to-parallel converter[19] [20]. The parrallel-to-serial converter ia used for data sent from Raspberry Pi to the ESP32, and the serial-to-parallel converter is used for data sent from the ESP32 to the Raspberry Pi. The UART frame format is as follows:

- Start bit: 1 bit, always 0
- Data bits: 5 to 9 bits, usually 8 bits
- Parity bit: 1 bit, optional, used for error detection
- Stop bit: 1 or 2 bits, always 1
- Idle bit: 1 bit, always 1

4.4 WEB API

4.4.1 WEB API

The web API is developed in ṄET ans it is responsible for user management, data storage, and communication with the IoT Hub. It uses Entity Framework Core to interact with the PostgreSQL database and SignalR to provide real-time communication. It uses the controller pattern to handle the HTTP requests. I created a controller for each resource in the system, such as users, zones, data, devices, schedules and cloud to device messages.

Zone Controller

Since the first step in the activation process is to verify the code enterede by the user, I will start describing the zone controller. Using this controller, the user can create a zone, activate a zone, get the IoT Device connection string, retrieve all zonees, get a zone by id, and delete a zone. For the database interaction, I created a zone repository that implements the IZoneRepository interface.

Listing 4.3: Zone Repository interface

```
public interface IZoneRepository
{
    public Task<bool> CreateZone(Zone zone);
    public Task<bool> UpdateZone(Guid Id, string Name, Guid userId);
    public Task<bool> DeleteZone(Guid Id);
    public Task<Zone> GetZoneById(Guid Id);
    public Task<Zone> GetZoneByName(string name);
    public Task<IEnumerable<Zone>> GetAllByUserId(Guid Id);
}
```

The controller is also using the IoTDeviceManager service to manage the IoT devices in Azure IoT Hub, which implement the IioTDeviceManager interface and is repsonsible for creating, deleting, and checking the existence of IoT devices.

Listing 4.4: IoT Device manager interface

```
public interface IiotDeviceManager
{
    public Task<bool> CreateIotDevice(string zoneId);
    public Task<string> GetDeviceConnectionString(string zoneId);
    public Task<bool> DeviceExists(string zoneId);
    public Task<bool> DeleteIotDevice(string zoneId);
}
```

4.4.2 ZONE ACTIVATION

The Raspbery Pi represents a "zone" in the system and it connects to a single Azure IoT Hub Device. Each zone needs to connect to the Azure IoT Hub Device using a connection string. To obtain that connection string, a a reuest is sent to the web API to create a new zone. The web api will create the zone in the database and it will create a new IoT Device in the Azure IoT Hub. The connection string will be returened to the "zone". Aftr that, the RaspberryPi won't send any reading to the cloud, the sending of the data will be done only when the zone is activated. The activation process is as follows:

1. The user enters the code displayed on the Raspberry Pi OLED display in the web application.
2. The web application sends the code to the web API.
3. The web API verifies the code and activates the irrigation zone.
4. The web API sends a message to the Iot Device to activate the irrigation zone.
5. The Raspberry Pi receives the message from Azure IoT Hub and activates the irrigation zone.
6. The Raspberry Pi sends a message to the ESP32 nodes to start collecting data from the sensors.

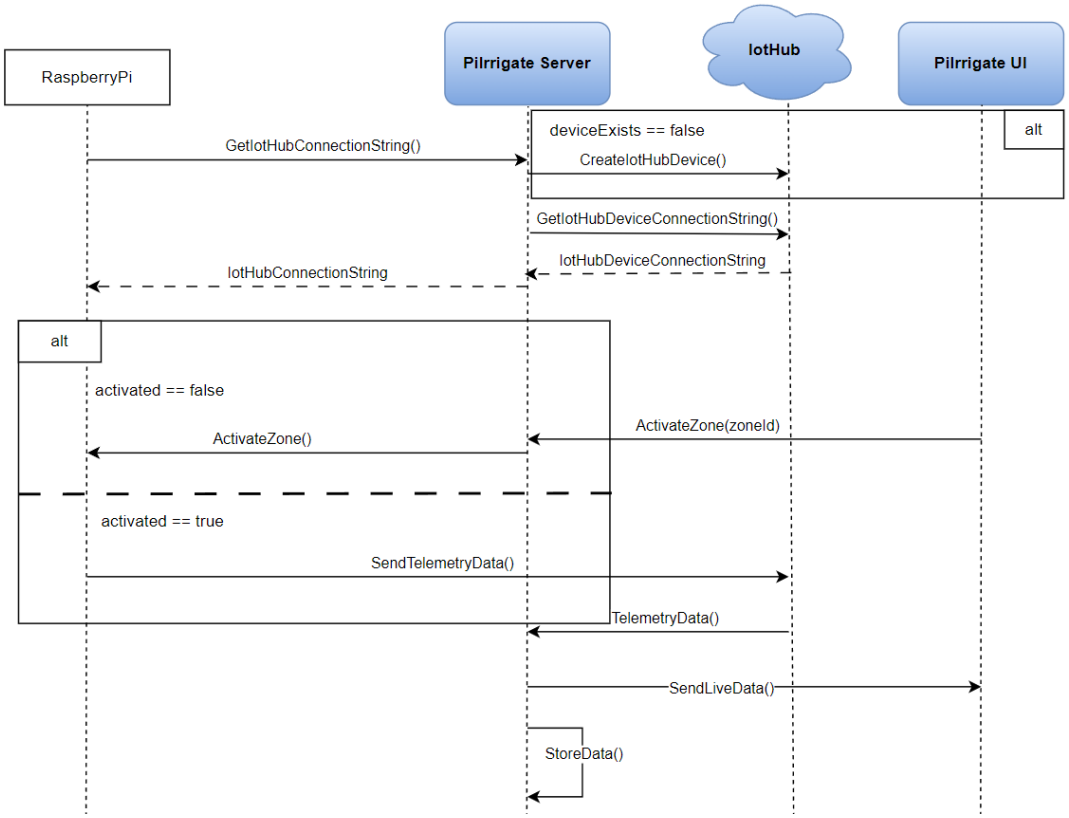

Figure 4.5: Zone activation process

Data Controller

The data controller is used to retrieve stored data from the Web API. This data is stored in the PostgreSQL database and it will be used to display historical data and analytics in the web application. It provides endpoints for:

- Get all data for a zone, including the data for all devices in that zone.
- Get data for a certain period of time.
- Get all data foor a device.
- Get data for a device for a certain period of time.

IData repository is used to interact with the database and it implments the IDataRepository interface.

```
1 public interface IDataRepository
2 {
3     public Task StoreData(SensorReading sensorReading);
4     public Task<IEnumerable<SensorReading>> GetTimedZoneData(DateTime
          from, DateTime to, Guid zoneId);
5     public Task<IEnumerable<SensorReading>> GetTimedDeviceData(DateTime
          from, DateTime to, Guid zoneId, string deviceId);
6     public Task<IEnumerable<SensorReading>> GetAllZoneData(Guid zoneId);
7     public Task<IEnumerable<SensorReading>> GetAllDeviceData(Guid zoneId
          , string DeviceId);
8 }
```

Device Controller

The device controller is used to manage the devices in the system. It provides endpoints for:

- Get all devices for a zone.
- Get all devices for a user.
- Get a device by id.
- Get a device by MAC address.
- Register a new device.

C2D Controller

The C2D (Cloud to Device) controller is used to manage the cloud to device messages in the system. It provides only one endpoint to send a message to a device. The endpoint accepts a C2DMesageRequest object, which contains the zoneId, and an object of type C2DMethodCall, which contains the the deviceId, the method name, and a list of parameters.

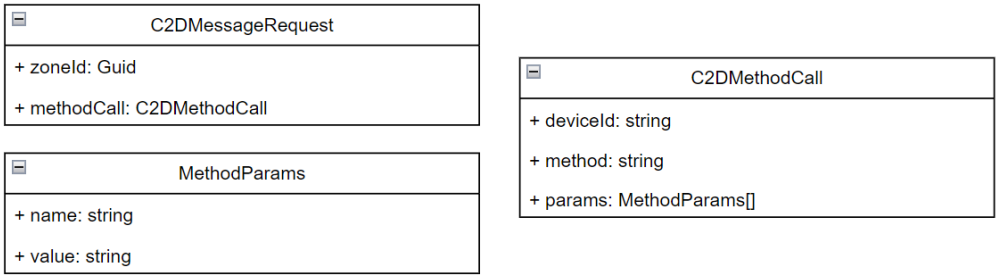

Figure 4.6: C2D Message Request object

Schedule Controller

The schedule controller is used to manage the irrigation schedules in the system. It provides endpoints for:

- Get all schedules.
- Get a schedule by id.
- Create a new schedule.
- Update an existing schedule.
- Delete a schedule.

The Schedule object is defined as follows:

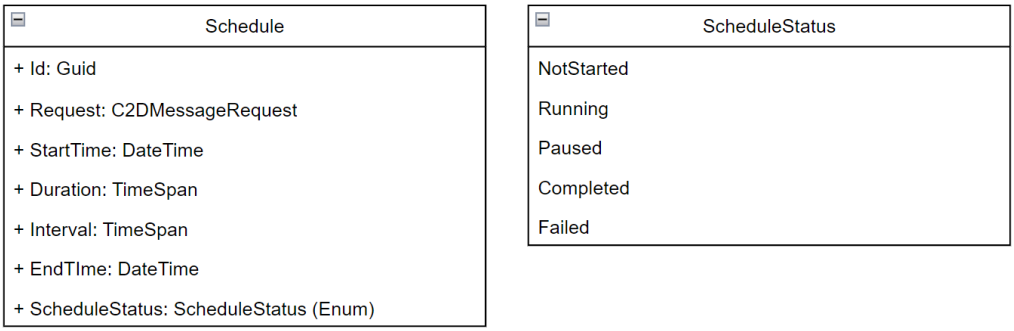

Figure 4.7: Schedule object

1. The Id is a unique identifier for the schedule.
2. The Request is an object of type C2DMesageRequest that contains the zoneId, deviceId, method name, and parameters.
3. The StartTime is the time when the schedule should start.
4. The EndTime is the time when the schedule should end.
5. The Interval is the time interval between two consecutive executions of the schedule.
6. The Duration is the duration of the schedule execution.
7. The ScheduleStatus is the status of the schedule, which can be NotStarted, Running, Paused, Completed, Failed.

The user can create a new schedule using the UI of the web application, which will send a request to the web API to create a new schedule. The web API will validate the request and create a new schedule in the database. A bragraound service will be used to check and execute the schedules. The background service will run every minute and check if there are any schedules that should be executed or stoped. To be more robust, the duration of the schedule, as well as the start and end time, will be sent to the Raspberry Pi, wich will also check if the schedule should be executed or stopped. This way if the Web API or the network connectivity is down, the Raspberry Pi will still be able to execute the schedules.

A background service is used to check the schedules. A baground services is a service that implements the IHostedService interface and runs tasks in the background. It is used to perform long-running tasks and it implements the StartAsync and StopAsync methods [21].

User Controller

As the user needs to be authenticated to access the web API, I created a user controller that handles user registration and login.

For accessing the user data, UserRepository is used, which implements the IUserRepository interface. The authorization is done using JWT tokens, which are generated when the user logs in. To handle all the logic related to JWT tokens, I created a JWT service that implements the IJwtService interface. Besides IUserRespository, and IJwtService, the UserService is also using IPasswordHasher to hash the user passwords. This provides abstraction for hashing and verifying passwords, allowing for easy integration with different hashing algorithms. For the moment, the implementation uses SHA256 hashing algorithm, but it can be easily changed to any other algorithm. In the future, I plan to replace this with Microsoft.AspNetCore.Identity, which provides a more secure and flexible way to handle user authentication and authorization and it is widely used in the ṄET community.

Listing 4.5: User Repository interface

```
public interface IUserRepository
{
    Task<User> GetByIdAsync(Guid id);
    Task<User> GetByEmailAsync(string email);
    Task<IEnumerable<User>> GetAllAsync();
    Task<bool> CreateAsync(User user);
    Task<bool> UpdateAsync(User user);
    Task<bool> DeleteAsync(Guid id);
}
```

Listing 4.6: JWT Service interface

```
public interface IJwtService
{
    string GenerateJwtToken(User user);
    bool ValidateToken(string token);
}
```

Listing 4.7: Password Hasher interface

```
public interface IPasswordHasher
{
    string HashPassword(User user, string password);
    bool VerifyPassword(User user,string hashedPassword, string
        providedPassword);
}
```

Registration Flow

The registration flow is as follows:

1. The user fills the registration form in the web application.
2. The web application sends the request to the web API. More sdpecifically, it sends a POST request to the register endpoint of the user controller. The controller receives the request as an object of type RegisterRequest.

3. The request is being validated by the web API.
4. If the request is valid, the web API creates a new user in the database
5. The web API generates a JWT token and it sends back an object of type AuthResult.
6. The web application receives the AuthResult object and stores the JWT token in the local storage.

The registration flow is shown in the figure below.

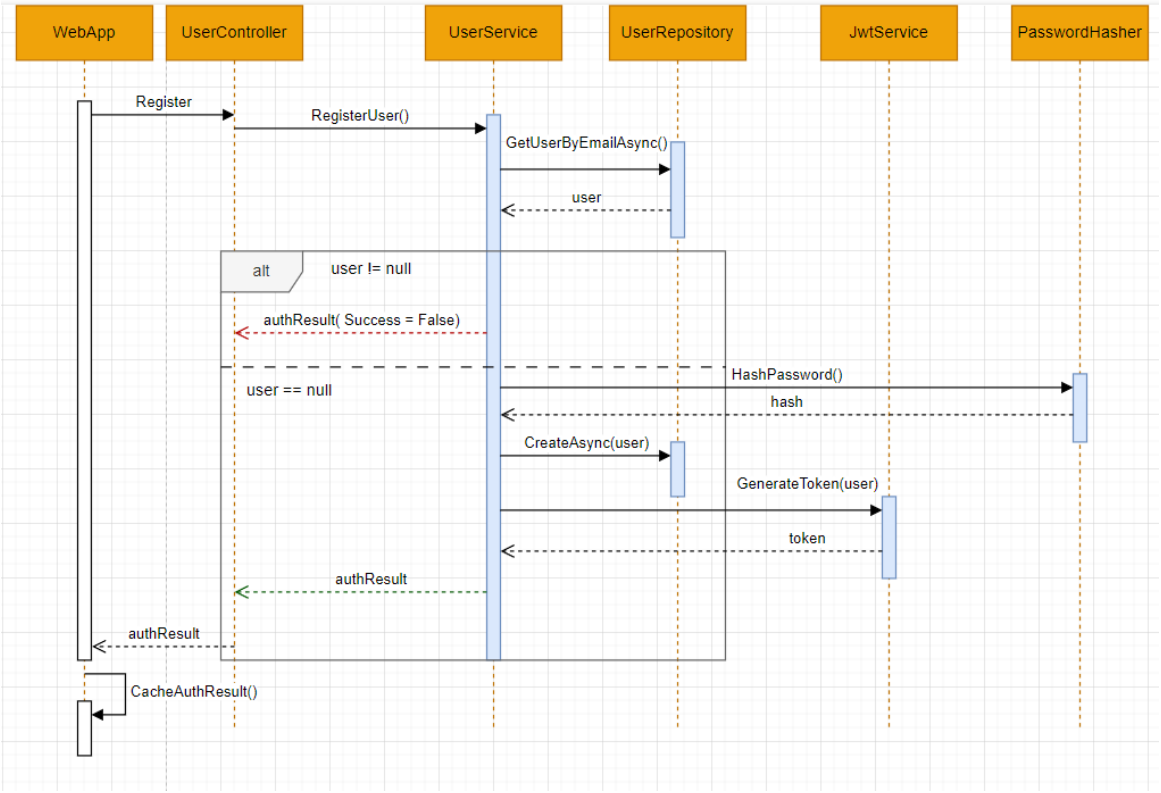

Figure 4.8: Registration flow

The AuthResult object and the RegisterRequest object are defined as follows:

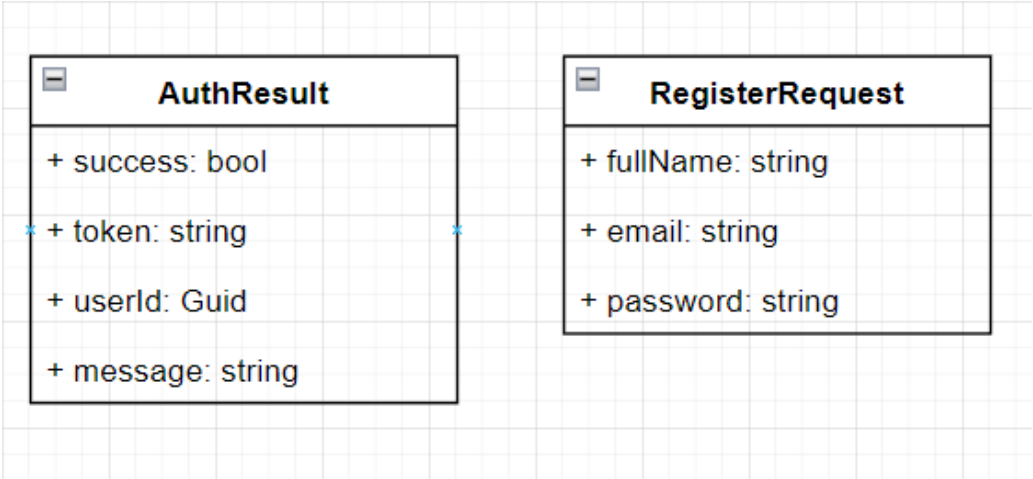

Figure 4.9: AuthResult and RegisterRequest objects

Login Flow

The login flow is similar to the registration flow, but it does not create a new user. The login flow is as follows:

1. The user fills the login form in the web application.
2. The web application sends a POST request to the login endpoint of the user controller. The controller receives the request as an object of type LoginRequest.
3. The request is being validated by the web API.
4. If the request is valid, the web API checks if the user exists in the database.
5. If the user exists, the web API verifies the password and generates a JWT token.
6. The web API sends back an object of type AuthResult.
7. The web application receives the AuthResult object and stores the JWT token in the local storage.

The login flow is shown in the figure below.

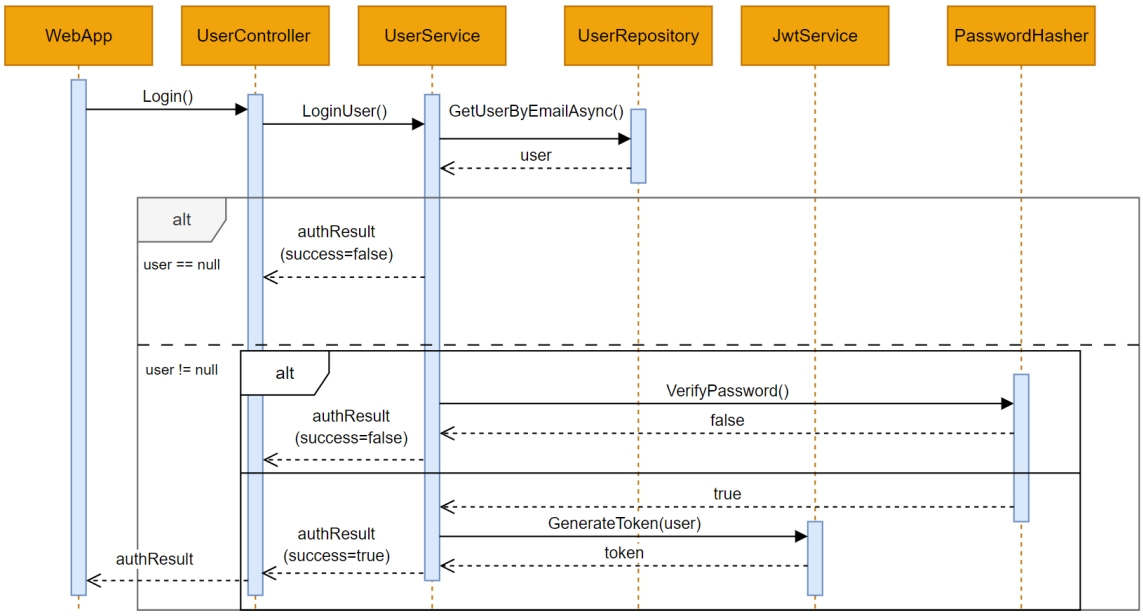

Figure 4.10: Login flow

4.5 WEB APPLICATION

The frontend interface of the smart irrigation system was implemented using Angular. The web application serves as the main point of interaction between the user and the backend system. It allow the users to monitor the environmental conditions in real-time and manually controll irrigation events.

The communication between the frontend and web API is done via 2 methods: HTTP requests - for fetching sensor data, trigger irrigaiton actions and display system status and

SignalR - for sending live data from the API to fronend. In the HTTP requests, a JWT token is included, ensuring secure access to system's features.

The user interface is divided into several key components: Dasboard, Devicem Management, Configuration, Schedules, Data Analytics and Live Charts. The user can navigate between these components using a side drawer menu that is visible in 4.14. Each value is displayed in a separate gauge-style widget to improve the readability.In this section the user can also controll the irrigation system. Now I will describe each component in detail, starting with the landing page.

4.5.1 LANDING PAGE

The landing page is the first page that the users sees when they access the web application. It presints the user with a brief introduction into the system and it provides the the user with the main features of the system. The page is responsive, meaning that it can be accessed from any device, including mobile phones and tablets. The page contains a scrollable carousel with the main features of the system and a call to action button that redirects the user to the registration page. In the following figure you can see the landing page on a desktop and on a mobile phone.

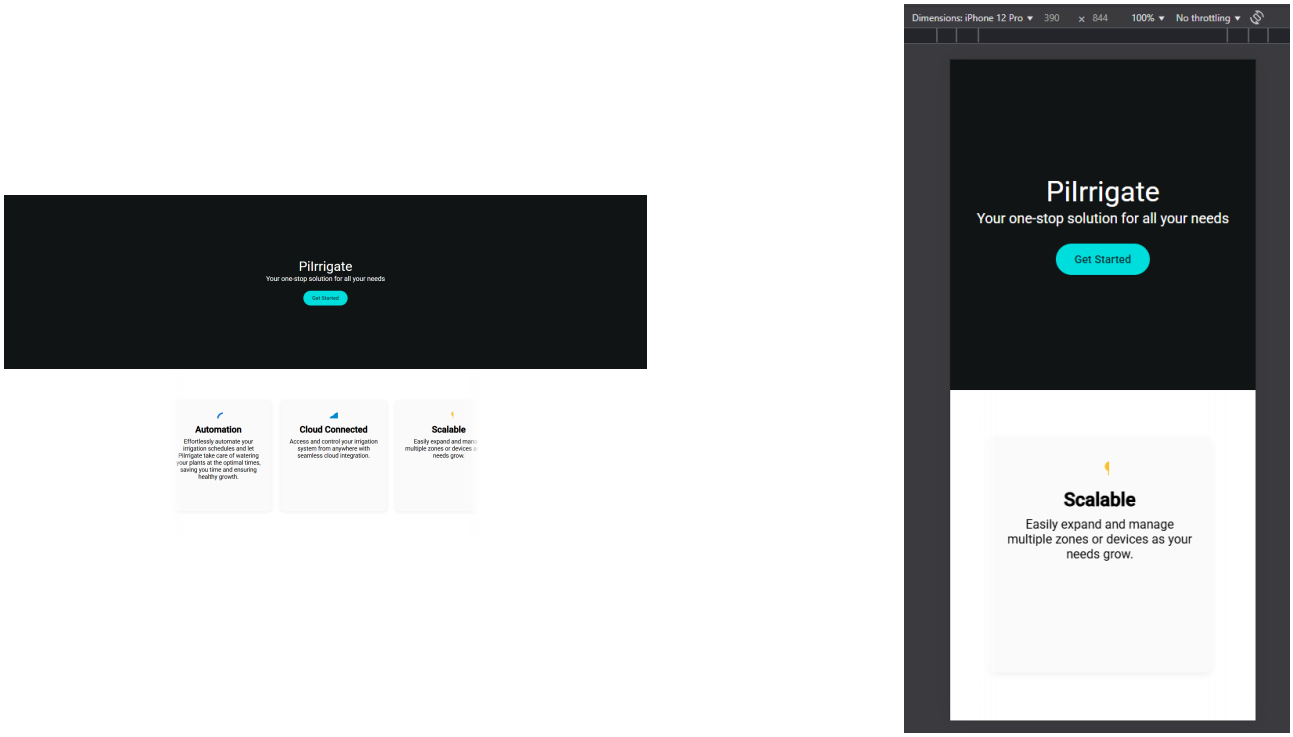

Figure 4.11: Landing page of the web application

4.5.2 REGISTER AND LOGIN

The registration and the Login pages are designed to be as simple as possible. Registration page contains a form with the following fields: Full Name, Email, Password, , Confirm Password and a checkbox for accepting the terms and conditions. The login page

contains a form with only 2 fields: Email and Password.

Figure 4.12: Register and Login pages

Now I will describe the registration and login flows. Afther ther user presses the call to action button from the landing page, he is redirected to the registraiton page. If the users fills the registration form and presses the register button it will be automatically logged in and redirected to the Dashboard. From the register page he can navigate to the login page where he can login. If the login fails, the user will can reset his password and try again. Bellow is presented the user flow diagram.

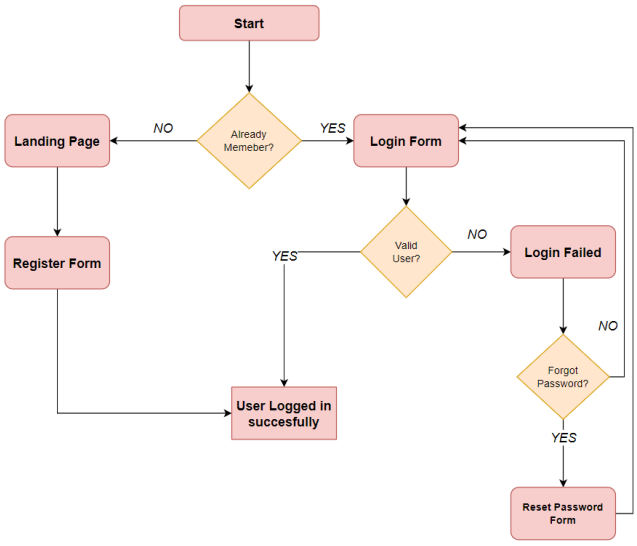

Figure 4.13: User flow

4.5.3 DASHBOARD

The Dashboard components contains several widgets some widgets are used to display live data and others are used to controll the irrigation system. The live data widgets are: Live Soil Moisture, Live Temperature, Live Humidity, Live Water Flow Rate and Live Water Temperature. In the control widget, the user can start or stop the irrigation process.

Figure 4.14: Dashboard page

4.5.4 DEVICE MANAGEMENT

In this section the user canm visualize all the devices that are registered in the system. The devices are display in tree format, where each zone is a parent node and the devices are the child nodes. Two buttons are available in this section: one for registering a new device and another for registering a new zone. Also, the user can delete a device or a zone by clicking the button from the right side of the tree. See 4.15 for the Device Management page.

Figure 4.15: Device Management page

4.5.5 CONFIGURATION

In the configuration section the user can configure the system settings. The user can change the following settings: soil moisture threshold, water flow rate threshold, water temperature threshold, raindrop sensor threshold, the period of the sensor readings, and to enable or disable notifications or automation. A reset button will be added in the future. See 4.16 for the configuration page.

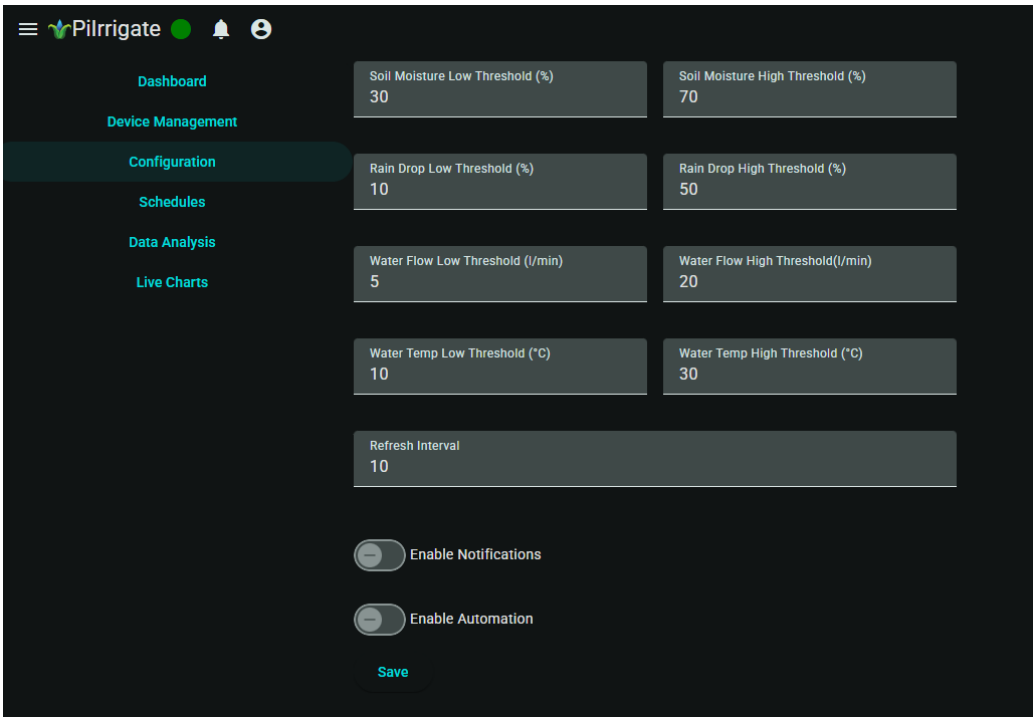

Figure 4.16: Configuration page

4.5.6 SCHEDULES

In this section the user can view, create, update and delete irrigation schedules. When the "Create Schedule" button is pressed, a dialog will open wehre the user can fill the schedule details. THe dialog contains a form with the following fields: Zone, Device, Start Time, End Time, Interval and Duration. For the Start and End Time fileds, a date range picker is used, which allows the user to select a date range for the schedule. The interval is the time interval between two consecutive executions of the schedule in hours. The duration is the duration of the schedule execution in minutes. See 4.17 for the Schedules page.

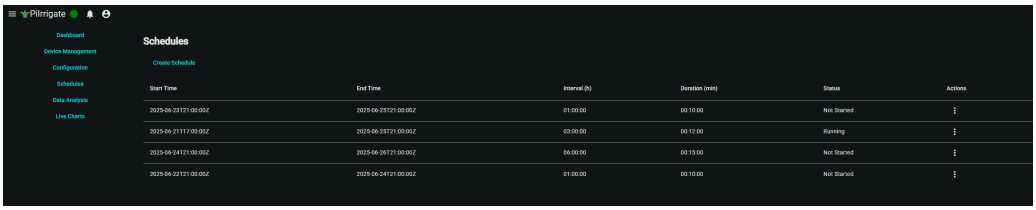

Figure 4.17: Schedules page

4.5.7 DATA ANALYTICS

Data Analytics section allows the user to view the historical data collected from the sensors under the form of charts. One example of chart is shown in 4.18. The user can also chose what features to display in the chart and it can also choose for what zone to view the charts.

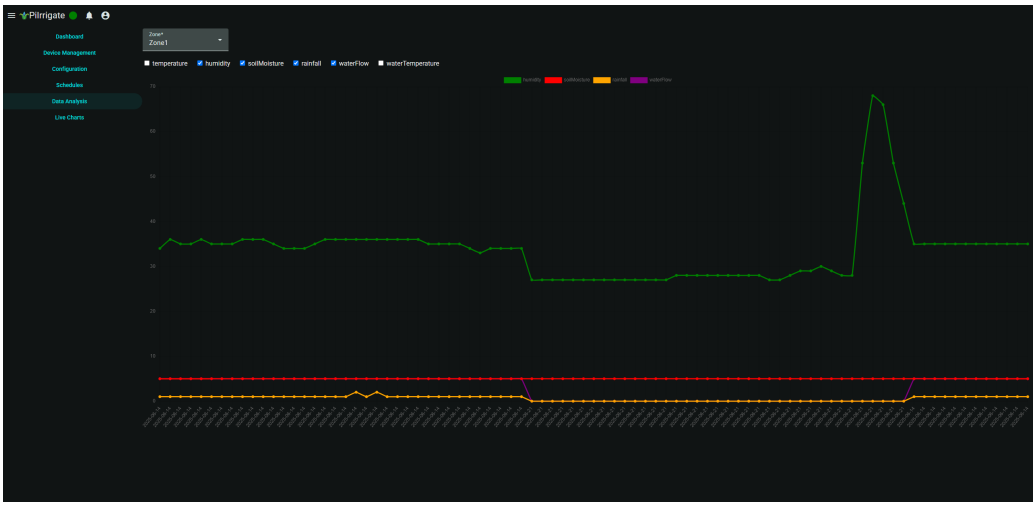

Figure 4.18: Data Analytics page

4.5.8 LIVE CHARTS

In this section, the user can view in real time the data collected from the sensors, similar to the dashboard page. But in this case, the data is displayed in a chart format. The user can choose what features to display in the char and it can also choose for what zone to view the charts. See 4.19 for the Live Charts page.

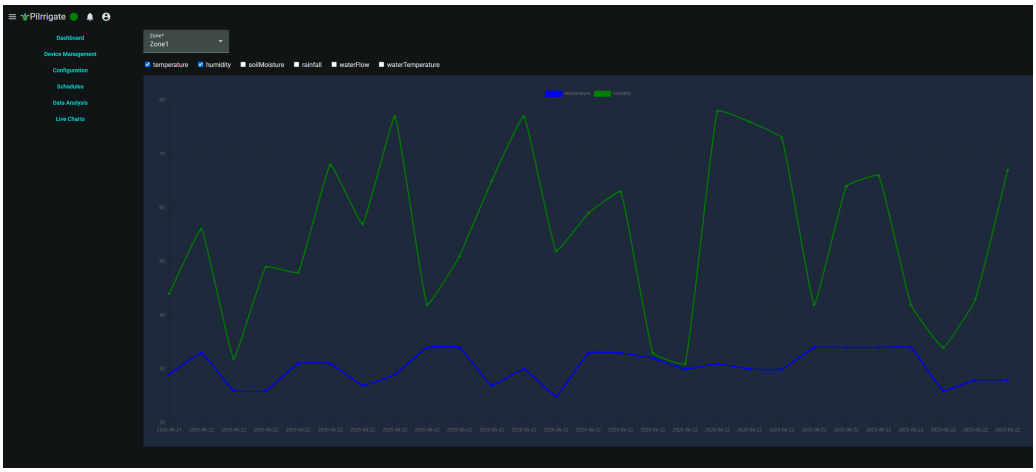

Figure 4.19: Live Charts page

4.6 AUTOMATION

Automation is a key feature of the PiIrrigate system, to achieve this, I wrote some rules that allow the system to make decisions based on the environmental data collected from the sensors. The rule-based system is implemented in the Web API and it coonsists of the following rules:

1. If the soil moisture is below a certain threshold and the raindop sensor detects no rain, then start the irrigation.

2. If the soil moisture is above a certain threshold and the water flow rate is above a certain

level, the irrigation will be stoped.

3. If the soil moisture reading is not available, then the irrigation will be stopped.
4. If the soil moisture is below the threshold and the water flow rate is zero, then an alert will b sent to the user.
5. If the irrigation process is running and the raindrop sensor detects rain, then stop the irrigation and send an alert to the user.
6. If the water temperature is below or above a certain threshold, stop the irrigation, then send an alert to the user.
7. If the water flow rate remains unchanged for a certain period of time, then send an alert to the user.

This set of rules is executed every time a new sensor reading is received. The rules are implemented in the IrrigaitonManager service which implements the IIrrigationManager interface.

4.7 USED TECHNOLOGIES

4.7.1 OVERVIEW

The PiIrrigate system is build using a variety of technologies, each serving a specific purpose. For programming the ESP32, I used Arduino framework, within the Platform IO IDE. To handle the LoRa communication protocol I used L̈oRa.hl̈ibrary from developed by Sandeep Mistry. The Raspberry Pi is programemd in Python and it uses Azure IoT SDK for Python to communicate with the Azure IoT Hub. The web API it was developed in Ṅ ET 8. The web application is developed in Anguler 19. The live communication between the web API and the web application is done using SignalR. The used database is PostgreSQL. The deployment of the web API and the web application is done using Azure App Services.

4.7.2 PROGRAMMING LANGUGES

C++

C++ is a general-purpose programming languuage that is widely used in the embeded systems domain. It is well known for its performance and efficiency. It supports multiple programming paradigms from procedural to object-oriented programming. The programing language was created in 1985 by Bjarne Stroustrup as an extension of the C programming language. On the top of C , C++ adds features shuch as classes, inheritancem polymorphism, templates and exception handling.

Arduino C++

The firmware was developed using Arduino Framework. In this framework, the programing language used is a simplified version of C++. The framework provides a set o libraries and

APIs that are meant to abtract the low-level details of the hardware operations. Some of the mose used functions are digitalRead, digitalWrite, analogRead, delay, etc.

Python

Python is a high-level, interpreted programming language. The language was created By Guido van Rossum and it was first released in 1991. It is well known for it's reability, simplicity and versatillity. Python supports multiple programming paradigms, like functiona, procedural and object-oriented programming. It can be used for a wide range of applications, from web development to data science and machine learning. In the PiIrrigate system, Python was used to develop the Raspberry Pi gateway.

C-Sharp

C# is a modern object oriented programming language developed by Microsoft. It was first released in 2000 as part of the ṄET framework. It was designed to be simple robust and versatile. C# combined the performance of compiled languages with the ease of development offers typical of managed languages. It supports modern programming features such as generics and it is a strongly typed language[22][23]. In the PiIrrigate system, C# was used to develop the web API.

TypeScript

TypeScript is a statically typed language. It is a superset of JavaScript and was designed to enchance the scalability and maintability of large codesbes. It introduced intefaces, decorators, enums and generics while maintaining compatibility with existing JavaScript code. Typescrip became the standard for developing large applications, especially when using frameworks like Angular. Typescrips was used to develop the Angular web application in the PiIrrigateSystem. It was chosen because it provides a better development exceperience.

4.7.3 FRAMEWORKS AND SDKS

Ardunio Framework

Arduino framework is an open-source electronics development platform that enables developers to create project on a wide rande of microcontrollers. It provides a wide range of libraries (ex: LoRa.h, Wifi.h, HTTPClient.h, etc). The integration is very smooth with both Arduino IDEa and Platform ID. I used the Arduino framwerk to develop the ESP32 firmware because it's ideal for rapid prototyping and it is widely use in this kind of projects[24].

Azure IoT SDK for Python

Azure IoT SDK for Python consist of a set of libraries that allows developers to interact with Azure IoT Hub. I chose the client library to connect the Raspberry Pi the Azure IoT Hub. More specifically I used the telemetry transimission feature to send the sendor readings to the cloud. The SDK offers suport for multiple protocols, including MQTT, AMQP and HTTP[25][26].

dotNET 8

ṄET 8 is the latest version of the ṄET framework, a cross-platform, open-source framework for building applications. It is supported by Microsoft and it is widely used to build web application, web APIs, desktop applicaitons and even mobile applications. It offers support for multiple programming languages, including C#, F# and Visual Basic. ṄET 8 was used to implement a RESTful backend API for the PiIrrigate system, which prvides the neccesarry endpoints for user management, data storage, and communication with the IoT Hub. Another important aspect of ṄET 8 is that it has natice support for asynchronous programming, dependency injection[27][28].

Entity Framework Core

EF Core is an open-source ORMİt is developed by Microsoft and it is part of the ṄET framwork. It enables developers to work with databases using class objects. Most of the raw queries that are usualy used in SQL databases are replaced with LINQ queries, wich are more readable and easier to maintain. EF core supports multiple database, including SQL Server, PostgreSQL, MySQL and SQLite. It was used in this project to interact with the PostgreSQL database[29][30].

Angular 19

Angular is an open-source frontedn web application framework maintained by Google. It can be used to build SPAs. The framework is written in TypeScript and it provides a set of tools and libraries for building modern web applicaitons. Angular is based on components, which are reusable. With the last version, Angular shifts to a more modular architecture, more and more components are being developed as standalone components. Angular 19 was used to develop the web application for the PiIrrigate system[31][32].

4.7.4 COMPONENT LIBRARIES

Angular Material

Angular Material is a UI compoent library thath implements the Material Design specification. It was created by the Angular team at Google and it provides a wide range of reusable compoenents, including buttons, crdsm tables, dialogs, and form controls. By using Angular Material in the UI development, the apps obtained are consistent and visually appealing, while also being responsive and accessible. Another aspect of using component libraries is thah it reduces the amount of custom CSS that needs to be written, as most of the components come with prebuild style and themes[33][34].

4.7.5 LIBRARIES

4.7.6 LORA.H

Lora.h is a libray developed by Sandeep Mistry that provides an easy way to use LoRa communication protocol. It is intednded for Semtech Semtech SX1276/77/78/79 based boards/shields[35]. In this project I used the following functions from the libray:

1. begin() - to initialize the LoRa module.
2. beginPacket() - to start a new LoRa packet.
3. endPacket() - to send the LoRa packet.
4. parsePacket() - to check if a new LoRa packet is available.
5. readbytes() - to read the data from the LoRa packet.
6. readStringUntil() - to read a string from the LoRa packet.

SignalR

SignalR is a libray built by Microsoft as part of the ṄET framework. It provides real-time web functionality to applications, allowing bidirectional communication between the server and the client. It is used to send real-time updates to the web application form the web API without the need for the client to poll the server[36].

4.7.7 CLOUD AND HOSTING

Azure App Services

Azure App Service is a fully managed PaaS that allows developers to deploy web appicaitons, RESTful APIs without having to manage the underlying infrastructure. It supports multiple programming languages, including ṄET, Python, Node.js, and Java. It can be easly integrated with CI/CD pipelines and it provides build-in support for scaling, monitoring and security[37]. In the PiIrrigate system the deploy was done using the Publish feature of Visual Studio. With a few clicks, the web API and the web applicaiton were deployed to Azure AppServices.

Neon Database

Neon is a fully managed serverless PostgreSQL database platform. It is designated for moder cloud-native applications. It is designed to be easy to use, scalable and cost-effective. Neon provides a serverless architecture and it separates the storage and the compute layers, allowing for automatic scaling and high availability.

4.7.8 DEVELOPMENT TOOLS

Platform IO

Platform IO is an open-source IDE that comes as a VSCode extension. It supports a wide range of platforms, including the ESP32. They have an unique philosophy in the embeded market, which is to provide developers withs a modern integrated development environment that allows them to access a wide range of libraries and frameworks. Besides that, Platform IO works cross-platform and offers uni testing, automatic code analysis and debbuging capabilities[38][39][40].

Visual Studio Code

Visual Studion Code is a free open-source code editor. It is developed by Microsoft and it is available on all major platforms. The editor is higly customizable and it supports a wider range of programming languages. In this project, I used VSCode as the main IDE for developing the ESP32 firmware, the web appication and also the code for the Raspberry Pi was written using VSCode. One of the nice fetures of VSCode is that supports SSH connections, which alllowed me to edit and run the code on the Raspberry Pi directly from the integrated terminal [41].

4.7.9 VISUAL STUDIO

Visual Studio is an IDE developed by Microsoft. It is available only on Windows and it is used to develop applications with the ṄET framework. The IDE provides a wide range of tools for debugging, profiling, testing and deploying applications. In this project, I used Visual Studio to develop the web API[42].

4.7.10 GITHUB

Github is a web-based platform for version control and collaboration. It is built on the top of Git and it allows developers to host and manage code repositories, track changes and collaborate with other developers. It supports continuous integration and continous deployment pipelines, which automates the build, test and deployment processes[42]. In this project, I used Github to host the code repositories for the entire system.

4.7.11 DRAW.IO

Draw.io is a free online diagramming tool that allows users to create flowcharts, UML diagrams and other types of diagrams. It was used to create all the presented diagrams.

5. TESTING AND RSULTS

5.1 INTRODUCTION

This chapter presents the testing and evaluation processes to validate the functionality, reliability and perforamnce of hte developed system. The goal is to ensure that each compoentn of the ssytem works as intended. The system's components are tested individually and in integration to make sure thy meet the specified requirements.

For this project I tested the software using unit tests and integration tests. The unit tests are designed to test individual components of the system, while the integration tests are designed to test the system as a whole. The hardware components are tested manually to ensure they work as expected. Results from these tests are documented to provide a clear overview of the system's performance and reliability.

5.2 TESTING

5.2.1 UNIT TESTING

Unit testing is a software testing techniques where individual components are tested in isolation to ensure they function correctly. I will present the unit tests that were created for the JwtService, which is a key component of the system, responsible of handling the JWT tokens. Usually one unit test is created for each metho of the class. In this section I will present only the unit test for the GenerateJwtToken() method. A unit test is split into three parts called Arrange, Act and Assert or Given, When and Then. In the Arrange part, the test sets up the necessary conditions for the test, such as mocking ans setting-up deppendencies or creating test data. In the Act part, the test calls the tested method with the prepared data. In the Assert part, the test checks if the result of the call is as expected.

Listing 5.1: Unit Test for GenerateJwtToken() Method in JwtService

```
[TestMethod ]
public void GenerateJwtToken_ShouldReturnValidToken()
{
    // Arrange
    var user = new User
    {
        Id = Guid.NewGuid(),
        Email = "test@example.com",
        FullName = "Test User",
        Role = UserRole.Admin
    };

    // Act
    var token = _jwtService.GenerateJwtToken(user);
```

```
15
16      // Assert
17      Xunit.Assert.False(string.IsNullOrWhiteSpace(token));
18
19      // Parse token to validate structure
20      var handler = new JwtSecurityTokenHandler();
21      var readable = handler.CanReadToken(token);
22      Xunit.Assert.True(readable);
23
24      var jwtToken = handler.ReadJwtToken(token);
25      Xunit.Assert.Equal("test@example.com", jwtToken.Payload["email"]);
26      Xunit.Assert.Equal("Test User", jwtToken.Payload["name"]);
27      Xunit.Assert.Equal("Admin", jwtToken.Payload[ClaimTypes.Role]);
28  }
```

5.2.2 INTEGRATION TESTING

Integration testing is a critical phase in the software development testing. In this phase, individual components are combined an tested together to ensure they work as a cohesive syste. This type of testing can be approached in various ways, such as top-down and bottom-up. In this project, I used the top-down approach fot integration testing. Manual integration testing was chosen for this project, due to the the presence of the hardware components, wich are not easily testable with automated tests. Additionally, the system is relatively small and It is at an early prototyping stage, where inttorducins automated tests would be an overhead. The perfomed manual tests are presented in the following table.

ID	Test Name	Description	Expected Outcome	Result
T1	Sensor Data Flow	Trigger a real sensor reading on the ESP32 node. Ensure it is received by the gateway and forwarded to the backend API.	Data appears in the backend database with correct timestamp and values.	Pass
T2	Frontend Data Display	After a new reading is received, open the Angular app dashboard.	UI shows latest sensor data.	Pass
T3	Automatic Irrigation Trigger	Set a low moisture threshold and insert dry soil. System should automatically trigger irrigation.	Relay activates.	Pass
T4	Manual Irrigation Control	From the frontend, press "Stop irrigation" button.	The relay should close.	Pass
T5	System Recovery	Restart the ESP32 node and Raspberry Pi.	System resumes operation.	Fail

Table 5.1: Manual Integration Test Plan

As you can see, the system, in general, works as expected, excepting test T5, whuch failed. The system did not recover after a restart. This is because the running process on the Raspberry Pi does not automatically start after a reboot. To fix this, I need to add the main process to the crontab of the Raspberry Pi, so it starts automatically after a reboot.

5.2.3 PERFORMANCE TESTING

Performance testing is a type of testing that evaluates the system's performance under various conditions.

End-to-End Data Transmission Time Test

End-to-end data transmission time testing measures the time it takes for a sensor reading to be sent from the ESP32 to the server. For this test, I added a new parameter to the sensor readings, which is the reading time, collected from the ESP32. When the reading is acknoleged by the server, the server calculates the time it took for the reading to be sent and stored. Then for each readigm the latency is calculated as the difference between the reading time and the time it was received by the server. The average latency is calculated over a number of readings, which is 100 in this case. I repeated this test in 3 different scenarios. Firts scenario is when the Raspbery Pi is connecthed to the local network via Ethernet cable, the second scenarion is when the Raspberry Pi is connected to the local network via Wi-Fi. The third scenario is when the Raspbbery Pi is connected to the internet the Ethernet cable, and the and the ESP32 node is placed in a different location than the ESP32 gateway. Below are presented the test scenarios:

S.	Description	Network Setup	Expected Impact on Latency
1	Raspberry Pi connected via Ethernet	ESP32 Gateway is connected to the Raspberry Pi via UART; and the Pi is wired to the router	Low and stable latency due to reliable Ethernet connection
2	Raspberry Pi connected via Wi-Fi	ESP32 Gateway is connected to the Raspberry Pi via UART; Pi connects to the router over Wi-Fi	Slightly increased latency due to potential Wi-Fi interference or variability in wireless throughput
3	LoRa Node communicates with remote Gateway, which is UART-connected to Raspberry Pi	ESP32 Node is in a different location and sends data over LoRa to the Gateway; the Gateway forwards data via UART to the Raspberry Pi	Higher latency caused by LoRa transmission time and distance; UART adds negligible delay, backend response depends on network setup

Table 5.2: Sensor-to-Backend Latency Test Scenarios with UART-connected ESP32 Gateway

The asumption is that the latency will be lower when the Raspberry Pi is connected to the local network via Ethernet cable. The first 2 scenarios are expected to have similar results, and the third scenario is expected to have a higher latency due to the LoRa transmission time and distance between the ESP32 Node and the Gateway. This would mean that the bootleneck is the LoRa transmission time, and not the UART transmission time, Ethernet or Wi-Fi connection.

The results of the end-to-end data transmission time test are presented in Table 5.3.

Scenario	Min Latency (ms)	Max Latency (ms)	Avg Latency (ms)
1	110	145	127
2	118	152	134
3	340	460	392

Table 5.3: Measured Sensor-to-Backend Latency Across Scenarios

The first 2 scenarios show similar result, with an average latency of around 130 ms, confirming that the Ethernet and Wi-Fi connections have similar performance for this applicaton. The third scenario shows a significantly higher latency of around 390 ms. The results confirm the assumption that the bootlenecl is the LoRa transmission time. The obtained performance is acceptable for the application, as the latency is within the range of a few hundred milliseconds.

5.2.4 HARDWARE TESTING

Sensor Testing

The sensors used in the system were tested to ensure they provide accurate readings. The tests involved measuring the sensor readings in different conditions and comparing them to known values. The soil moisture sensor's VCC and GND pins were connected to the pins of a stable power source of 3.3V and the voltmeter's positive and negative probes were connected to the sensor's output pin and GND pinm respectively. Then the sensor was placed in different soil conditions (dry, wed) and the voltage readings were recorded. The result are presented in Table The rain sensor was tested in a similar manner, by plancing it in different conditions. The DHT111 sensor could not be tested with a voltmeter, as it provides digital readings. Fot this sensor I used the ESP32 to read the sensor data and display it on the serial monitor. I compared the reading with the values from a thermometer and hydrometer. The result were not quite perfect, but they were within the acceptable range. The results of the manual sensor testing are presented in the following table:

Sensor	**Condition Tested**	**Measured Voltage (V)**	**Interpretation**
Soil Moisture Sensor	Inserted in wet soil	2.8 – 3.3 V	High voltage → soil is wet (low resistance)
Soil Moisture Sensor	Inserted in dry soil	0.8 – 1.2 V	Low voltage → soil is dry (high resistance)
Rain Sensor	Plate completely dry	0.5 – 1.0 V	Low voltage → no rain detected
Rain Sensor	Plate with water droplets	2.9 – 3.3 V	High voltage → rain detected

Table 5.4: Manual Sensor Testing Using a Voltmeter

6. CONCLUSIONS AND FUTURE WORK

This chapter summarizes key findings and conclusions drwan from this project, and it will also outline potential future work and improvements that can be made.

In this paper, we have explored the use of IoT technologies in an agricultural setting. I presented the advantages of using IoT for monitoring and managing agricultural processes. A prottype as also presented, which demontrated the proof of concept for use of IoT in agriculture. We also explored the use of LoRa technologies for long-range communication.

6.1 FUTURE WORK AND IMPROVEMENTS

At the completion of this project, I found that there are several areas for future work and improvements. I will outline them in the following order: I will start with the sensors, then I will move on to the firmware, backend, and finally the frontend. The first area of improvement represents the sensors used. If I were to redo this project, I would use more qualitative sensors, because the sensors used in this project were not very reliable. For example the water temperature sensor was not working properly. I would also have used more agricultural specialazed sensors, to get multiple readings from the soil and air, but that can be done in the future.

The second area of improvement is the firmware. The firmware is not very well optimized, the code is not the cleanest and it can be improved. I would also like to use a single sensor per node, and to have separate nodes for actuators. This would also help reduce the complexity of the firmaware and of the script that runs on the gateway. THe hardware setup canalso be improved, by using a lora adapter for the Raspberry Pi, instead of using another microsontroller. This would have also reduced the complexity of the system. On the hardware side, a better gateway register system can be implemented. To avoid duplication of messages a RSSI based system can be implemented, which would allow the node to connect to the gateway with the best signal strength. See **lora_gateway_rssi** for more details.

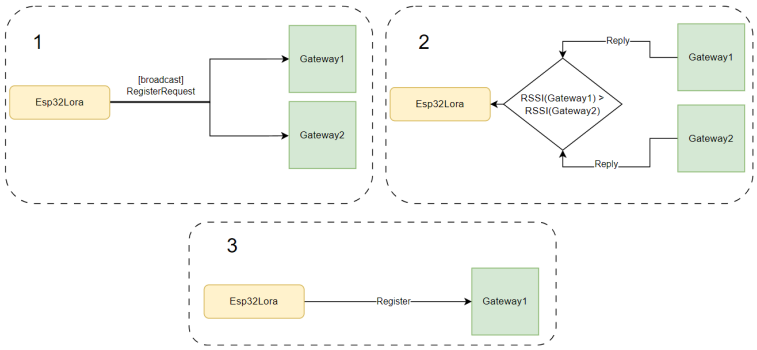

Figure 6.1: RSSI based gateway registration mechanism.

The third area of improvement is the backend. In the current implementation, the

backend in only one service that handles all. In the future, I would split the backend into multiple sercives, each handling a specific set of tasks, such as data storage, user management, irrigation management notifications, etc. This would allow for better scalability and mainainability of the system. It would also allow for better security and better robustness. Beside this improvement, I would also like to to use a separate database for the sensor readings, more specifically, a time-series database.

The last area of improvement is the frontend. In my opinion, the UI design looks good, but the code is not the cleanest, I would like to create a even more modular design, with better separation of responsabilities. A feature that I would like to add is the ability to see each node's position on a map, and to be able to see the readings from each node on the map. This will allow the users to better visualize the data and better understand the reading from each node. Another feature would be using of machine learning. It would be intersting to collect the data from the system, and together with weather forecasts, to feed a machine learning model, which would be able to predict the best time to irrigate the crops.

7. BIBLIOGRAPHY

[1] K. Obaideen et al., "An overview of smart irrigation systems using iot," *Energy Nexus*, vol. 7, p. 100 124, 2022, ISSN: 2772-4271. DOI: https://doi.org/10.1016/j.nexus.2022.100124. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2772427122000791.

[2] D. Tilman, "Global environmental impacts of agricultural expansion: The need for sustainable and efficient practices," *Proceedings of the National Academy of Sciences*, vol. 96, no. 11, pp. 5995–6000, 1999. DOI: 10.1073/pnas.96.11.5995. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.96.11.5995. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.96.11.5995.

[3] I. Froiz-Míguez et al., "Design, implementation, and empirical validation of an iot smart irrigation system for fog computing applications based on lora and lorawan sensor nodes," *Sensors*, vol. 20, no. 23, 2020, ISSN: 1424-8220. DOI: 10.3390/s20236865. [Online]. Available: https://www.mdpi.com/1424-8220/20/23/6865.

[4] J. Q. M. Malarie Gotcher Saleh Taghvaeian, "Smart irrigation technology: Controllers and sensors," [Online]. Available: https://extension.okstate.edu/fact-sheets/smart-irrigation-technology-controllers-and-sensors.html.

[5] U. S. E. P. Agency, "Soil moisture-based irrigation controllers," [Online]. Available: https://www.epa.gov/watersense/soil-moisture-based-irrigation-controllers.

[6] L. Slats, *A brief history of lora®: Three inventors share their personal story at the things conference*, 2020. [Online]. Available: https://blog.semtech.com/a-brief-history-of-lora-three-inventors-share-their-personal-story-at-the-things-conference.

[7] M. A. Ahmed et al., "Lora based iot platform for remote monitoring of large-scale agriculture farms in chile," *Sensors*, vol. 22, no. 8, 2022, ISSN: 1424-8220. DOI: 10.3390/s22082824. [Online]. Available: https://www.mdpi.com/1424-8220/22/8/2824.

[8] M. Dhanaraju, P. Chenniappan, K. Ramalingam, S. Pazhanivelan, and R. Kaliaperumal, "Smart farming: Internet of things (iot)-based sustainable agriculture," *Agriculture*, vol. 12, no. 10, 2022, ISSN: 2077-0472. DOI: 10.3390/agriculture12101745. [Online]. Available: https://www.mdpi.com/2077-0472/12/10/1745.

[9] X. Shi et al., "State-of-the-art internet of things in protected agriculture," *Sensors*, vol. 19, no. 8, 2019, ISSN: 1424-8220. DOI: 10.3390/s19081833. [Online]. Available: https://www.mdpi.com/1424-8220/19/8/1833.

[10] L. Stelitano, "Headless architecture: What is it, and why is it the future?," 2023. [Online]. Available: https://alokai.com/blog/headless-architecture.

[11] K. Borimechkov, *From request to database: Understanding the three-layer architecture in api development*, Mediuim, 2023. [Online]. Available: https://konstantinmb.medium.com/from-request-to-database-understanding-the-three-layer-architecture-in-api-development-1c44c973c7af.

[12] N. Team, *Ngrx store: State management*, Comprehensive guide to state management using NgRx in Angular applications, 2023. [Online]. Available: https://ngrx.io/guide/store.

[13] T. Team, *Interfaces*, TypeScript documentation on interfaces for modeling data structures, 2023. [Online]. Available: https://www.typescriptlang.org/docs/handbook/interfaces.html.

[14] A. Team, *Introduction to services and dependency injection*, Overview of how services encapsulate business logic and communicate with APIs in Angular, 2023. [Online]. Available: https://angular.io/guide/architecture-services.

[15] A. Team, *Milestone 5: Route guards*, Explains route guards for protecting navigation based on auth or roles, 2023. [Online]. Available: https://angular.io/guide/router#milestone-5-route-guards.

[16] PlatformIO Labs, *Platformio documentation*, Accessed: 2025-06-14, 2024. [Online]. Available: https://docs.platformio.org/en/latest/what-is-platformio.html.

[17] E. Garage, "What is the 1-wire protocol?," [Online]. Available: https://www.engineersgarage.com/what-is-the-1-wire-protocol/.

[18] S. Adla, N. K. Rai, S. H. Karumanchi, S. Tripathi, M. Disse, and S. Pande, "Laboratory calibration and performance evaluation of low-cost capacitive and very low-cost resistive soil moisture sensors," *Sensors*, vol. 20, no. 2, 2020, ISSN: 1424-8220. DOI: 10.3390/s20020363. [Online]. Available: https://www.mdpi.com/1424-8220/20/2/363.

[19] Rohde-Schwarz, "Understanding uart," [Online]. Available: https://www.rohde-schwarz.com/us/products/test-and-measurement/essentials-test-equipment/digital-oscilloscopes/understanding-uart_254524.html.

[20] N. R. Laddha and A. Thakare, "A review on serial communication by uart," *International journal of advanced research in computer science and software engineering*, vol. 3, no. 1, 2013.

[21] Microsoft, "Ihostedservice interface," [Online]. Available: https://learn.microsoft.com/en-us/dotnet/api/microsoft.extensions.hosting.ihostedservice?view=net-9.0-pp.

[22] Microsoft, *The .net platform*, Accessed: 2025-06-20, 2024. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/.

[23] Microsoft Developer Network, *C# guide*, Accessed: 2025-06-20, 2024. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/csharp.

[24] Arduino, *Arduino core for esp32*, ESP32 support via Arduino framework. Accessed: 2025-06-20, 2024. [Online]. Available: https://github.com/espressif/arduino-esp32.

[25] Microsoft, *Azure iot hub documentation*, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/iot-hub.

[26] Microsoft Azure Team, *Azure iot sdk for python*, 2024. [Online]. Available: https://github.com/Azure/azure-iot-sdk-python.

[27] Microsoft Developer Network, *.net 8 documentation*, 2024. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-8.

[28] Microsoft, *The .net platform*, Accessed: 2025-06-20, 2024. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/.

[29] Microsoft, "Entity framework core," [Online]. Available: https://learn.microsoft.com/en-us/ef/core/.

[30] Microsoft, *Entity framework core documentation*, Accessed: 2025-06-20, 2024. [Online]. Available: https://learn.microsoft.com/en-us/ef/core.

[31] Angular Team, *Angular v19 now available*, 2024. [Online]. Available: https://blog.angular.io/angular-v19-now-available.

[32] Angular Team at Google, *Angular documentation*, 2024. [Online]. Available: https://angular.io/docs.

[33] Google, *Material design guidelines*, Accessed: 2025-06-20, 2024. [Online]. Available: https://m3.material.io.

[34] Angular Team at Google, *Angular material*, Accessed: 2025-06-20, 2024. [Online]. Available: https://material.angular.io.

[35] Arduino, *Lora library*, Accessed: 2025-06-20, Arduino Project, 2024. [Online]. Available: https://docs.arduino.cc/libraries/lora/.

[36] Microsoft, *Asp.net core signalr*, Accessed: 2025-06-20, 2024. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction.

[37] Microsoft Azure, *Azure app service documentation*, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/app-service/.

[38] Python Software Foundation, *The python programming language*, Accessed: 2025-06-20, 2024. [Online]. Available: https://www.python.org.

[39] P. S. Foundation, "Python," [Online]. Available: https://www.python.org/.

[40] P. S. Foundation, *Python enhancement proposals (peps)*, Accessed: 2025-06-20, 2024. [Online]. Available: https://peps.python.org.

[41] Microsoft, *Visual studio documentation*, Accessed: 2025-06-20, 2024. [Online]. Available: https://learn.microsoft.com/en-us/visualstudio/.

[42] GitHub, Inc., *Github docs*, Accessed: 2025-06-20, 2024. [Online]. Available: https://docs.github.com.