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

1 Introduction **6**
1.1 Context . . . 6
1.2 Motivation . . . 7
1.3 Objectives . . . 7
1.4 Scope and Limitations . . . 7
1.5 Methodology overview . . . 8

2 State of the Art **10**
2.1 Introduction . . . 10
2.2 Existing Smart Irrigation Solutions . . . 10
2.2.1 Types of Smart Irrigation Systems . . . 10
2.3 Comparative Analysis of Smart Irrigation Systems . . . 11
2.4 IoT and LoRa in Agriculture . . . 12
2.5 LoRa vs Other Wireless Technologies . . . 13
2.6 Identified Gaps and Challenges . . . 14
2.7 Summary . . . 14

3 Used Technologies **15**
3.1 Development Process Tools . . . 15
3.1.1 Version Control: GitHub . . . 15
3.1.2 Software Development: Visual Studio Code and Visual Studio . . . 15
3.1.3 System Architecture: Draw.io . . . 15
3.1.4 Iot Device Management: Azure IoT Hub . . . 15
3.2 Communication Technologies . . . 16
3.2.1 LoRa Radio Communication . . . 16
3.2.2 MQTT Protocol . . . 16
3.2.3 HTTP Protocol . . . 16
3.2.4 SignalR and WebSockets . . . 16
3.3 Programming Languages and Frameworks . . . 16
3.3.1 C# and .NET . . . 16
3.3.2 Entity Framework Core . . . 17
3.3.3 Python . . . 17
3.3.4 Arduino C/C++ . . . 17
3.3.5 Angular and TypeScript . . . 17

4 PiIrrigate System Overview **18**
4.1 Overview . . . 18
4.2 Hardware Components . . . 19
4.2.1 Sensors . . . 19

4.2.2 ESP32 (LilyGo T-Beam) 19
4.2.3 Raspberry Pi 4 Model B 20
4.3 Hardware Architecture 21
4.4 Data Flow 22
4.4.1 Data aquisition 23
4.4.2 LoRa Radio Communication 26
4.4.3 Raspberry Pi and ESP32 Gateway 27
4.5 Software Components 28
4.5.1 Azure IoT Hub 28
4.5.2 Zone Activation 28
4.5.3 Web API 29

5 Conclusions **37**
5.1 Bibliography 37
5.2 Authenticity Declaration 37

6 Bibliography **38**

LIST OF FIGURES

2.1 LoRa-based IoT Architecture for Agriculture 12

4.1 ESP32 (LilyGo T-Beam) module used in PiIrrigate 20
4.2 Raspberry Pi 4 Model B used in PiIrrigate 21
4.3 ESP32 pinout and connections 22
4.4 ESP32 Gateway connection to Raspberry Pi 22
4.5 PiIrrigate System overview 23
4.6 Steps in data aquisition from DHT11 sensor 24
4.7 Soil moisture sensor based on resistive principle 24
4.8 Water flow sensor used in PiIrrigate 25
4.9 DS18B20 water temperature sensor used in PiIrrigate 26
4.10 Irrigaiton zone 28
4.11 Zone activation process 29
4.12 C2D Message Request object 31
4.13 Schedule object 32
4.14 Registration flow 34
4.15 AuthResult and RegisterRequest objects 35
4.16 Login flow 36

LIST OF TABLES

2.1 Comparison of Smart Irrigation Systems 11
2.2 Comparison of Wireless Technologies for Smart Irrigation 13

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 systems use weather data to adjust irrigation schedules based on evapotranspiration rates, ensuring that plants receive the right amount of water.
- **Soil Moisture-Based Controllers**
  These systems rely on data from soil moisture sensors placed in the root zone of the plants. Irrigation cycles are triggered when the soil moisture drops below a predetermined threshold, ensuring plants receive water only when necessary. This method is very precise for specific zones[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.

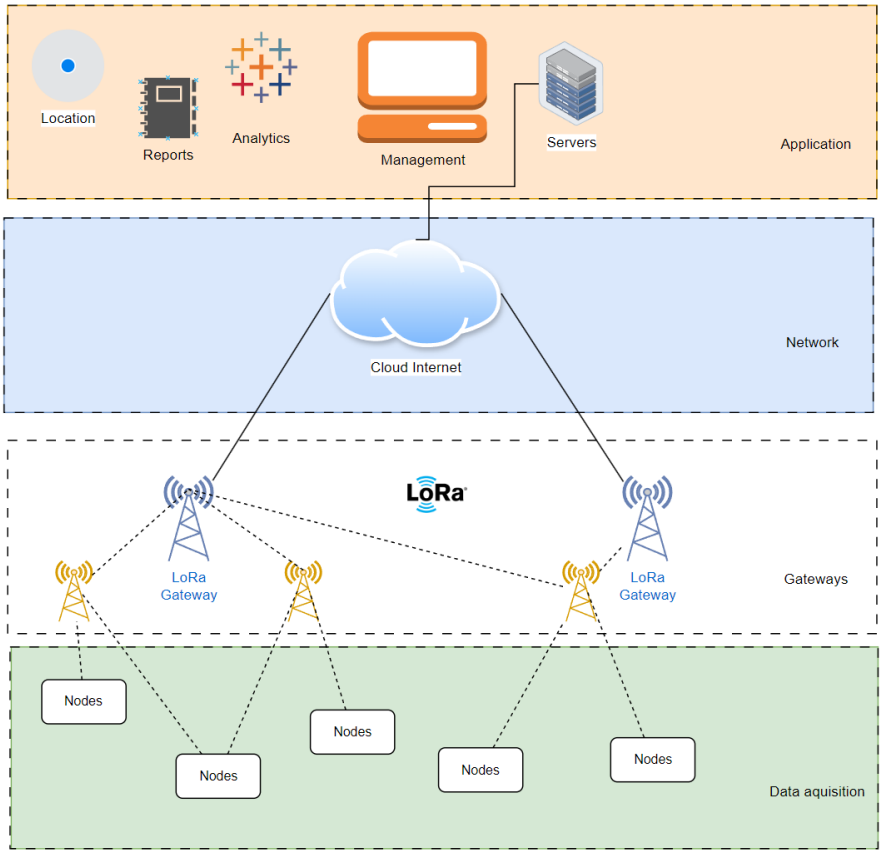

Figure 2.1: LoRa-based IoT Architecture for Agriculture

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 systems are designed for specific crops or environments, limiting their applicability in diverse agricultural settings.
- **Data Integration Issues**
  Integrating data from multiple sources (e.g., weather, soil sensors) can be challenging, leading to inefficiencies in irrigation management.

Beside the presented gaps, there are also some other challenges that need to be addressed, such as: data security and privacy concerns, the need for reliable internet connectivity in remote areas, and the need for systems that can operate in harsh environmental conditions.

Another aspect that needs to be considered is the environmental impact of smart irrigation systems. While these systems are designed to optimize water usage, the production and disposal of electronic components can have a negative impact on the environment. So another challenge is to create systems that are environmentally friendly and sustainable. This means that the components used in these systems shlould be madea from recyclable materials and the systems should be designed to have a long lifespan and to be easily repairable.

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. USED TECHNOLOGIES

3.1 DEVELOPMENT PROCESS TOOLS

3.1.1 VERSION CONTROL: GITHUB

GitHub is a web-based platform that uses Git for version control and collaboration on software projects. It enables developers to collaborate in real-time. Thhey can track changes, manage issues, and review code. It was released in 2008 and in 2018 it was acquired by Microsoft[8]. Github is widely used in the software development industry and in has become standard practice to use GitHub for version control.

3.1.2 SOFTWARE DEVELOPMENT: VISUAL STUDIO CODE AND VISUAL STUDIO

Visual Studio Code (VS Code) is a free lightweight code editor developed by Microsoft. It supports a wide range of programming langueges and has a rich collection of extensiont that can be used to enchance its functionality. It was very convinient to use a single code editor for multiple programming languages and frameworks. To be more specific, I used Visual Studio Code for ESP32 programming, along with Platform IO, which is an open-source ecosystem for IoT development. VSCode was also used for the Angular web application development. The Raspberry Pi was programmed using Python, I used Visual Studio Code and for remote access I used SSH.

Visual Studio is a more comprehensive integrated development environment (IDE) that provides advance features for software development, such as debugging, profiling, and testing. Visual Studio is also developed by Microsoft and it is widely used in the software development industry. Since the PiIrrigate's web API was developed in .NET, I used Visual Studio for the API development.

3.1.3 SYSTEM ARCHITECTURE: DRAW.IO

Draw.io is a free online diagramming tool that allows users to create flowcharts, UML diagrams, network diagrams, and more. Other alternatives include Lucidchart, Microsoft Visio, and Gliffy. But I found that Draw.io is easy to use and provides a wide range of templates and shapes that can be used to create diagrams.

3.1.4 IOT DEVICE MANAGEMENT: AZURE IOT HUB

Azure IoT Hub is a cloud-based service that enables secure and reliable communication between IoT devices and the cloud. It provides a way to connect, monitor, and manage IoT devices at scale. Azure IoT Hub is used in the PiIrrigate project to manage the communication between the Raspberry Pi and the web API.

3.2 COMMUNICATION TECHNOLOGIES

3.2.1 LORA RADIO COMMUNICATION

LoRa is a wireless modulation technique derived from Chirp Spread Spectrum (CSS) technology. LoRa modulated transmission is robust against disturbances and can be received across great distances. It has become popular, as one of the most used standards for device interconnection, mainly because of its low power consumption and long-range capabilities[9]. This technology was used in the PiIrrigate project to enable the communication between the ESP32 nodes and the ESP32 gateway connected to the Raspberry Pi.

3.2.2 MQTT PROTOCOL

MQTT (Message Queuing Telemetry Transport) is a an OASIS standard lightweight messaging protocol for the Internet of Things (IoT). It is designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth. In the PiIrrigate project, MQTT is used to send data from the Raspberry Pi to IoTHub and from the web API to the web application.

3.2.3 HTTP PROTOCOL

HTTP is an application layer protocol for transmitting hypermedia documents, such as HTML. It was designed for communication between web browsers and servers, but it can also be used for machine-to-machine communication. In the PiIrrigate project, HTTP is used to send data from the web API to the web application[10].

3.2.4 SIGNALR AND WEBSOCKETS

SignalR is an open-source library for ASP.NET that simplifies the process of adding real-time web functionality to applications. It allows server-side code to push content to connected clients instantly as it becomes available. SignalR uses WebSockets as its primary transport protocol, but it can also fall back to other techniques like Server-Sent Events or Long Polling if WebSockets are not available. In the PiIrrigate project, SignalR is used to provide real-time communication between the web API and the web application.

3.3 PROGRAMMING LANGUAGES AND FRAMEWORKS

3.3.1 C# AND .NET

C# is a modern, object-oriented programming language developed by Microsoft. It is widely used for developing Windows applications, web applications, and cloud services. .NET is a free, open-source developer platform that provides a wide range of tools and libraries for building applications. In the PiIrrigate project, C# and .NET are used to develop the web API that manages the communication between the Raspberry Pi and the web application, as well as to handle data storage in the PostgreSQL database.

3.3.2 ENTITY FRAMEWORK CORE

Entity Framework Core (EF Core) is an open-source, lightweight, extensible, and cross-platform version of the Entity Framework, which is an object-relational mapper (ORM) for .NET. EF Core allows developers to work with databases using .NET objects, eliminating the need for most of the data access code that developers usually need to write. In the PiIrrigate project, EF Core is used to interact with the PostgreSQL database,

3.3.3 PYTHON

Python is a high-level, interpreted programming language known for its simplicity and readability. It is widely used in various domains, including web development, data analysis, machine learning, and IoT. In the PiIrrigate project, Python is used to develop the code that runs on the Raspberry Pi, which is responsible for receiving data from the ESP32 nodes and sending it to the web API.

3.3.4 ARDUINO C/C++

Arduino C/C++ is a simplified version of C/C++ that is used to program Arduino boards and other microcontrollers. It provides a set of libraries and functions that make it easy to interact with hardware components. In the PiIrrigate project, Arduino C/C++ is used to program the ESP32 nodes that collect data from the sensors and send it to the gateway using LoRa radio communication.

3.3.5 ANGULAR AND TYPESCRIPT

Angular is a platform and framework for building single-page client applications using HTML and TypeScript. It is developed and maintained by Google and is widely used for building modern web applications. In the PiIrrigate project, Angular is used to develop the web application that provides a user interface for monitoring and controlling the irrigation system.

TypeScript is a superset of JavaScript that adds static typing and other features to the language. It is designed for large-scale applications and provides better tooling and error checking compared to JavaScript. JavaScript is a high-level, interpreted programming language that is widely used for building web applications. It is primarly used for client-side scripting, but it can also be used for server-side development using Node.js.

4. PIIRRIGATE SYSTEM OVERVIEW

4.1 OVERVIEW

The Internet of Things (IoT) is a new technology that allows devices to connect remotely to achieve smart farming [11]. 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[12].

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.

4.2 HARDWARE COMPONENTS

4.2.1 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. The sensor is connected to the ESP32 board and sends the data to the gateway using LoRa radio communication.
- **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.

4.2.2 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 4.1: ESP32 (LilyGo T-Beam) module used in PiIrrigate

4.2.3 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.

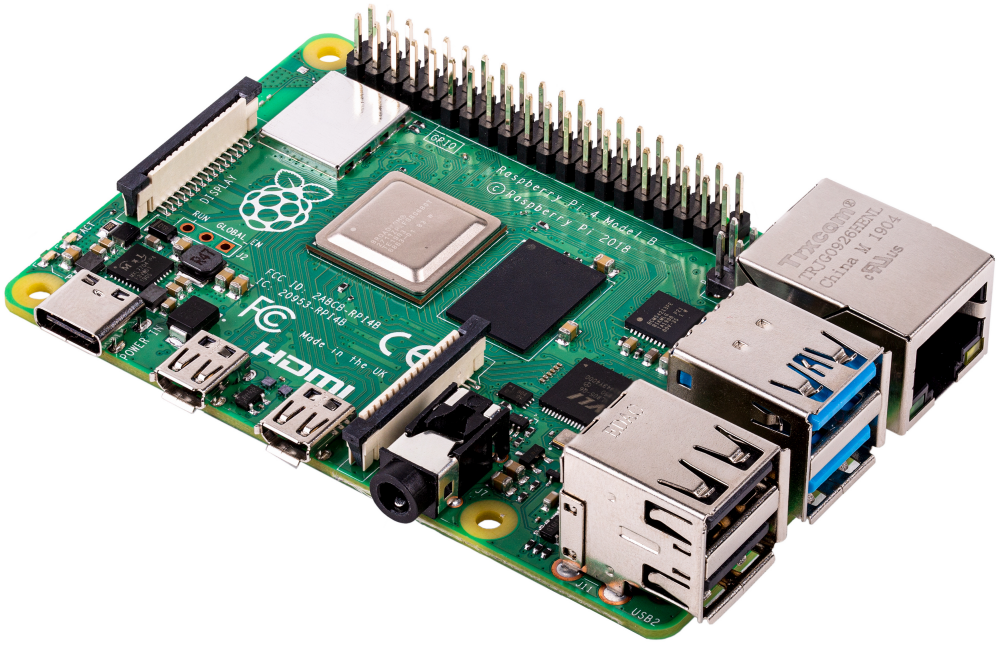

Figure 4.2: Raspberry Pi 4 Model B used in PiIrrigate

4.3 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 GPIO32.
- Temperature and humidity sensor (DHT11) is connected to GPIO27.
- Rain sensor (YL-83) is connected to GPIO32.
- Water flow sensor YF-S201 is connected to GPIO35.
- Water temperature sensor (DS18B20) is connected to GPIO26.
- The relay module that controls the electronic valve is connected to GPIO22.

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

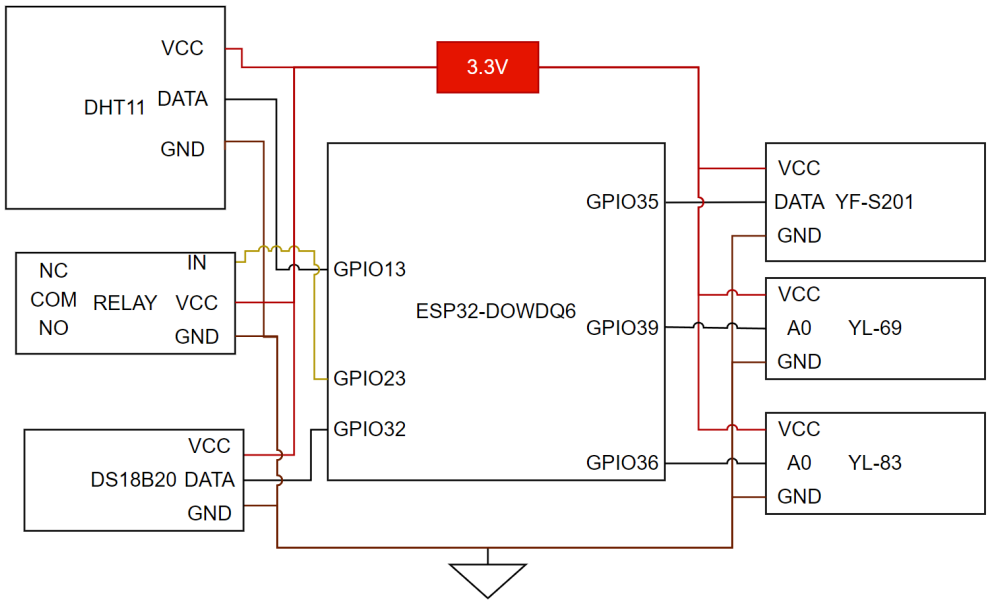

Figure 4.3: ESP32 pinout and connections

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.

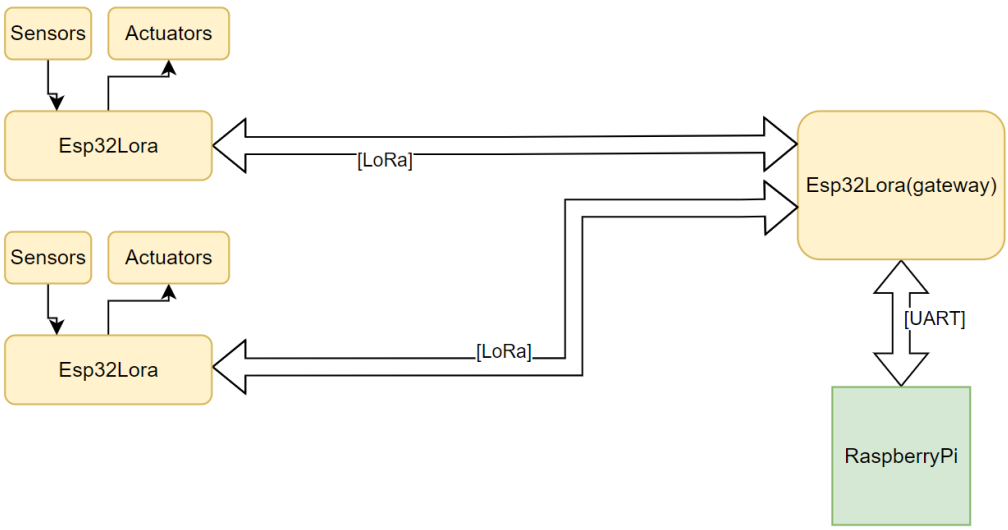

Figure 4.4: ESP32 Gateway connection to Raspberry Pi

4.4 DATA FLOW

The data flow in the PiIrrigate system is as follows:

1. The ESP32 nodes collect data from the sensors and send it to the gateway using LoRa radio communication.
2. The Raspberry Pi receives the data from the ESP32 nodes and sends it to Azure IoT Hub using MQTT protocol.
3. The web API receives the data from Azure IoT Hub and stores it in the PostgreSQL database using Entity Framework Core.
4. The web application retrieves the data from the web API and displays it to the user in real-time using SignalR.

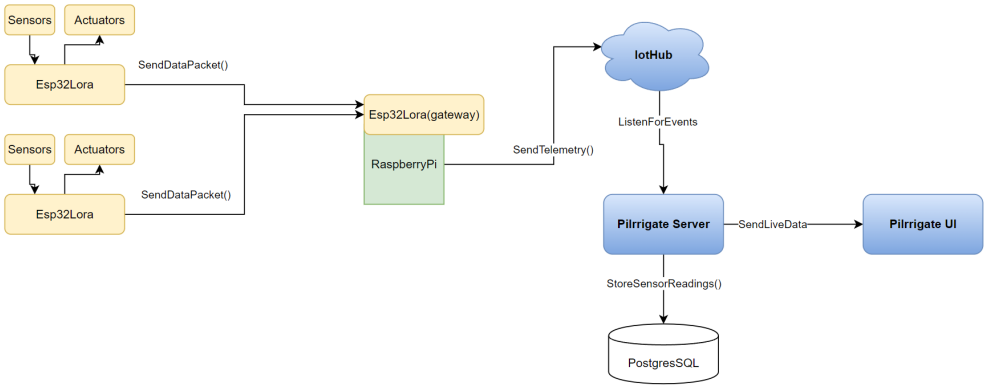

Figure 4.5: PiIrrigate System overview

4.4.1 DATA AQUISITION

The data acquisition is done using the ESP32 nodes that collect data from the sensors. For each sensor type, I created a specific library thath 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. The communication is done in 3 steps[13]:

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.

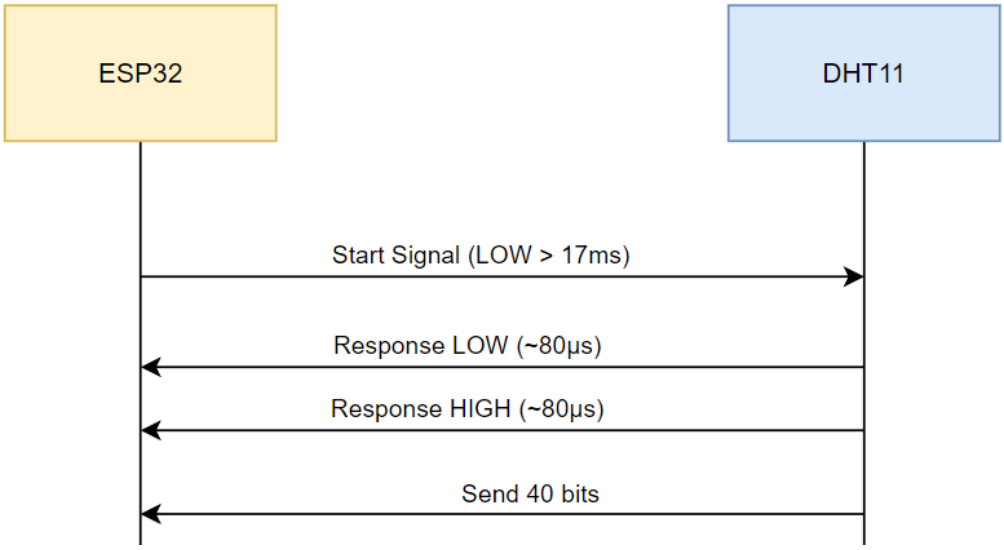

Figure 4.6: Steps in data aquisition from DHT11 sensor

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[14]. 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.

For the rain sensor, I used a resistive rians sensor. The principle of operation is similar to the soil moisture sensor, but it is designed to detect the presence of water on the sensor surface. When the sensor is dry, the resistance between the probes is high, and when it is wet, the resistance is low.

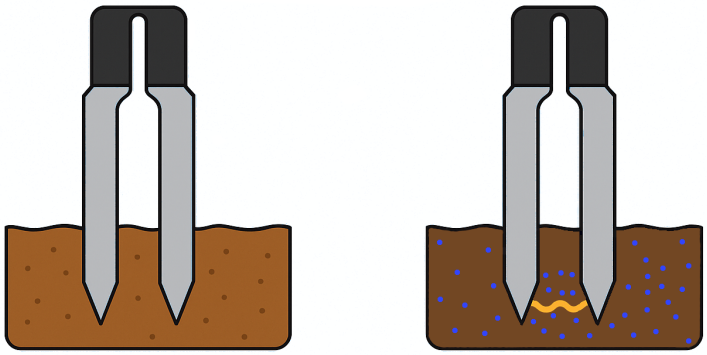

Figure 4.7: Soil moisture sensor based on resistive principle

Since the water consumoption 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. 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 4.8: Water flow sensor used in PiIrrigate

In my implementation, the water will be in a container, and during the irrigation process, the water will be deliverd to the plants using the gravity force. Because of this, I used water temperature sensor to measure the temperature of the water in the container. 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 4.9: DS18B20 water temperature sensor used in PiIrrigate

Beside the sensors, I also used the GPS module of the T-Beam ESP32 board to ollect the GPS coordinates of the node. This can be useful for tracking the location of the node and for mapping the irrigation system. The LILYGO Meshtastic AXP2101 T-Beam V1.2 ESP32 LoRa development board has a NEO-6M GPS module, which is a high-performance GPS module that can provide accurate location data.

The GPIO25 pin of the ESP32 board is used to control the relay module that controls the electronic valve that opens and closes the water flow to the plants. The relay module is a simple electronic switch that can be used to control high-voltage devices, such as water pumps or valves.

4.4.2 LORA RADIO COMMUNICATION

LoRa (Long Range) is a wireless communication technology that is designed for long-range, low-power applications. In this implementation, LoRa is used to send data from the nodes to the gateway. The sensor reading has a period of 10 seconds, which is the default value, but this 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 {
    float temperature;
    float humidity;
    float soilMoisture;
    float rainLevel;
    float waterTemp;
    float totalWaterFlow;
    float longitude;
    float latitude;
};
```

All the values, except the totalWaterFlow, and the GPS coordinates, represent the raw voltage values read from the sensors. I chose to sent the raw values, because they can be used to calculate the actual values in the web application. Allowing a better flexibility in the future.

4.4.3 RASPBERRY PI AND ESP32 GATEWAY

The gateway ESP23 act 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[15] [16]. 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

After receiveing the data, the Raspberry Pi will deserialize the LoRa packet structure and will send the data to Azure IoT Hub using MQTT protocol. For handling the communication between the Raspberry Pi and the Azure IoT Hub, I used the Azure IoT SDK for Python. I chose to use the Azure IoT SDK for Python, because it is easy to use and it provides a simple way to connect to Azure IoT Hub. Besides this, it also provides a way to manage the devices and to send and receive messages.

4.5 SOFTWARE COMPONENTS

4.5.1 AZURE IOT HUB

Azure IoT Hub is a cloud based platform that enables secure and reliable communication between IoT devices and cloud. It is very user friendly, providing sdk for multiple programming languages, including Python, C#, Java, and JavaScript. In Azure IoT Hub, devices are represented as ÏoT devices, all data from a single Raspberry Pi is sent to a single IoT device. In this paper, In this project, I will refere to the IoT device as an Ïrrigation zone, because it represents a single irrigation zone that is controlled by a single Raspberry Pi. And the term d̈evicewill refer to the ESP32 nodes.

When the Raspberry Pi is started, it will automatically connect to the Azure IoT Hub, and it will create a new IoT device if it does not exist. That irrigation zone will not be active until the user will activate it from the web application. The user is able to activate the irrigation zone using the code shown on RaspberryPi OLED display. The code is a unique identifier for the irrigation zone and it is used only once.

As I said before, a single irrigation zone is represented as a single IoT device in Azure IoT Hub and the connected nodes.

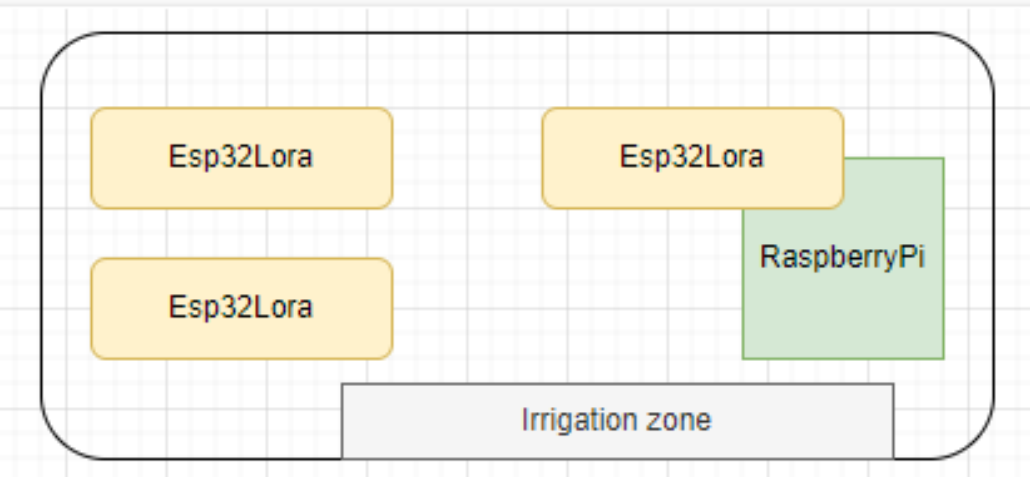

Figure 4.10: Irrigaiton zone

4.5.2 ZONE ACTIVATION

The activation process is done in the following steps:

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.

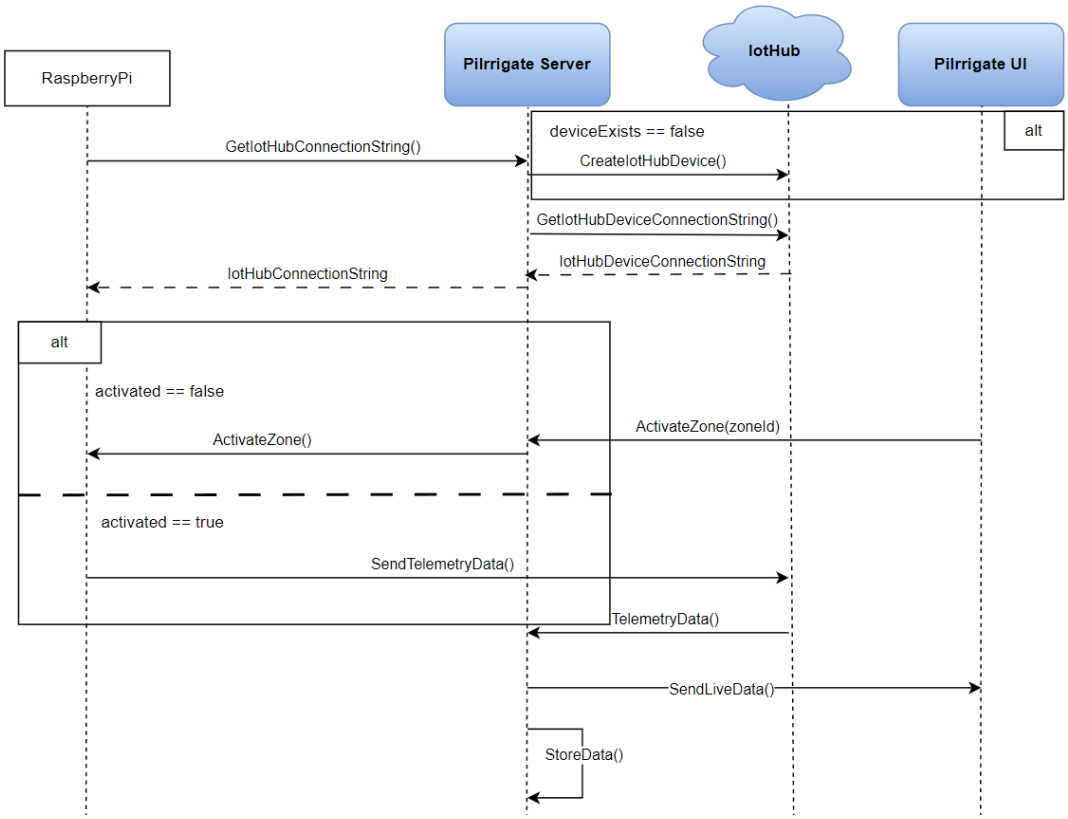

Figure 4.11: Zone activation process

4.5.3 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);
}
```

Data Controller

The data controller is used to retrieve data from the PostgreSQL database. 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.

The data controller will use the IDataService for all the logic related to data retrieval. For the DataService, all the database interaction is done within that service.

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.

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.

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.

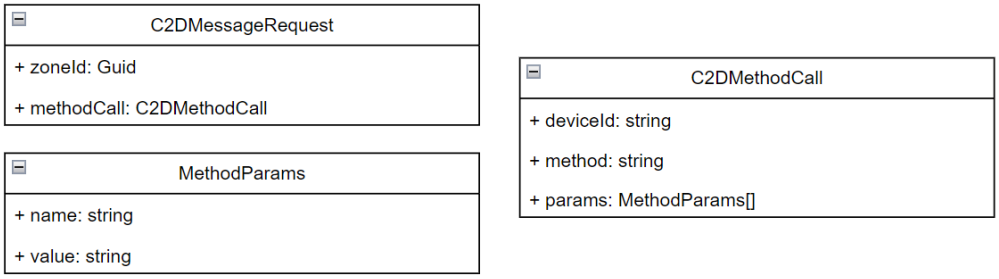

Figure 4.12: C2D Message Request object

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:

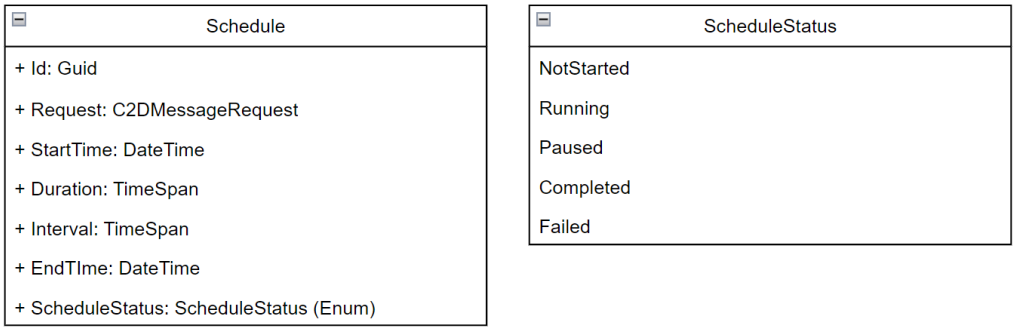

Figure 4.13: Schedule object

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.

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.

Listing 4.5: User Service interface

```
1 public interface IUserService
2 {
3     Task<UserDto> GetUserByIdAsync(Guid userId);
```

```
    Task<IEnumerable<UserDto>> GetAllUsersAsync();
    Task<bool> UpdateUserProfileAsync(Guid userId, UpdateProfileRequest
        request);
    Task<bool> ChangeUserRoleAsync(Guid userId, string newRole);
    Task<AuthResult> RegisterUser(RegisterRequest register);
    Task<AuthResult> LoginUser(LoginRequest request);
}
```

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 4.6: 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 4.7: JWT Service interface

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

Listing 4.8: 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.

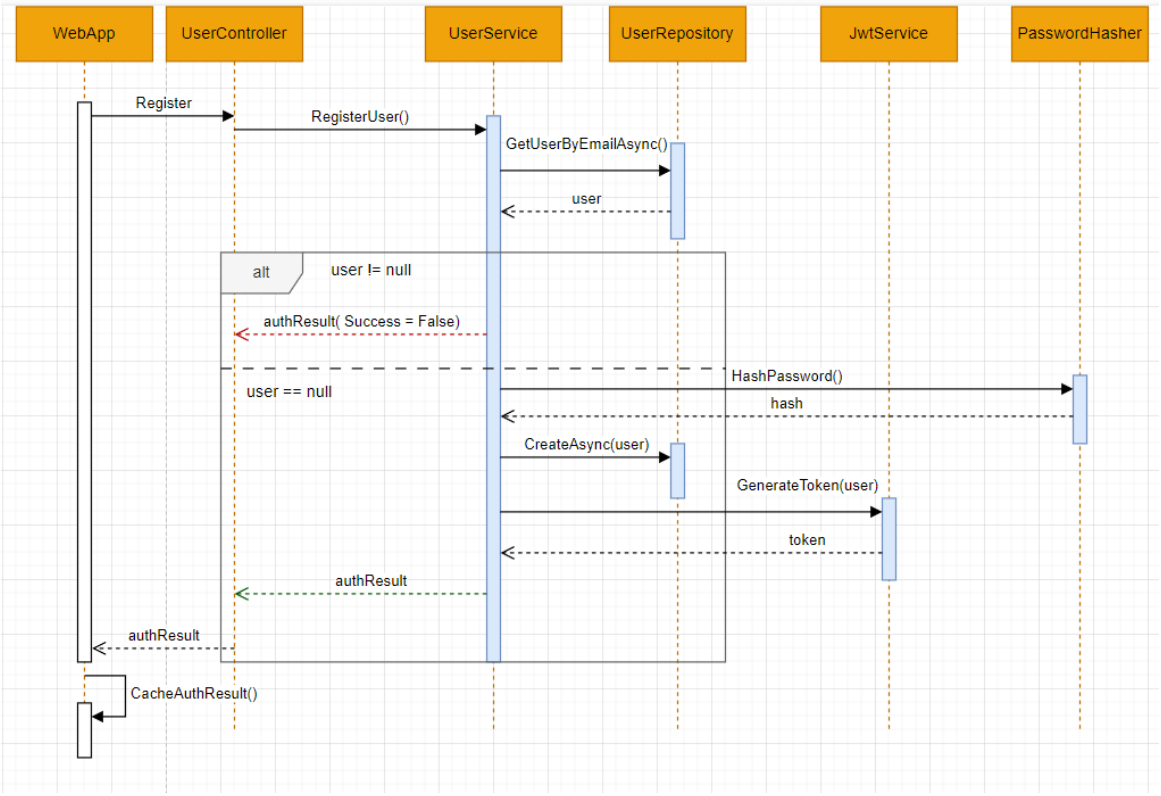

Figure 4.14: Registration flow

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

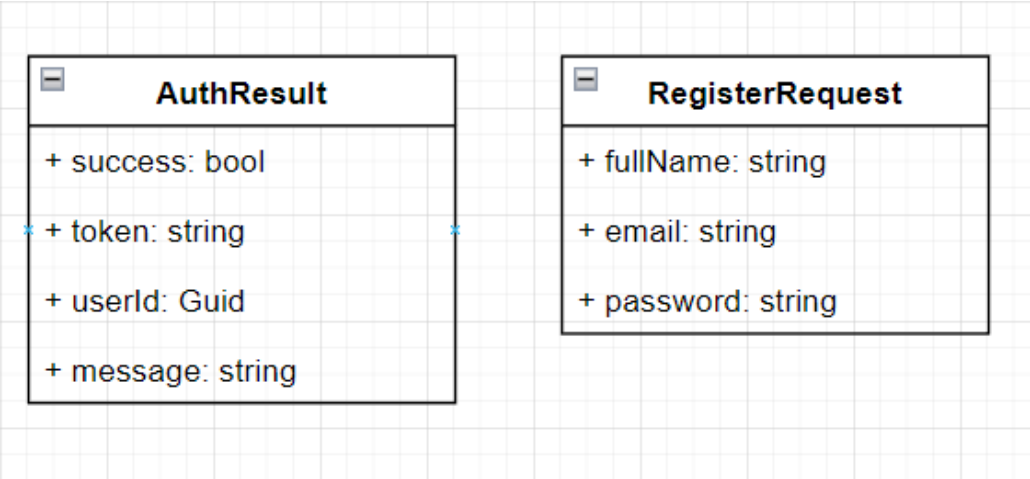

Figure 4.15: AuthResult and RegisterRequest objects

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.

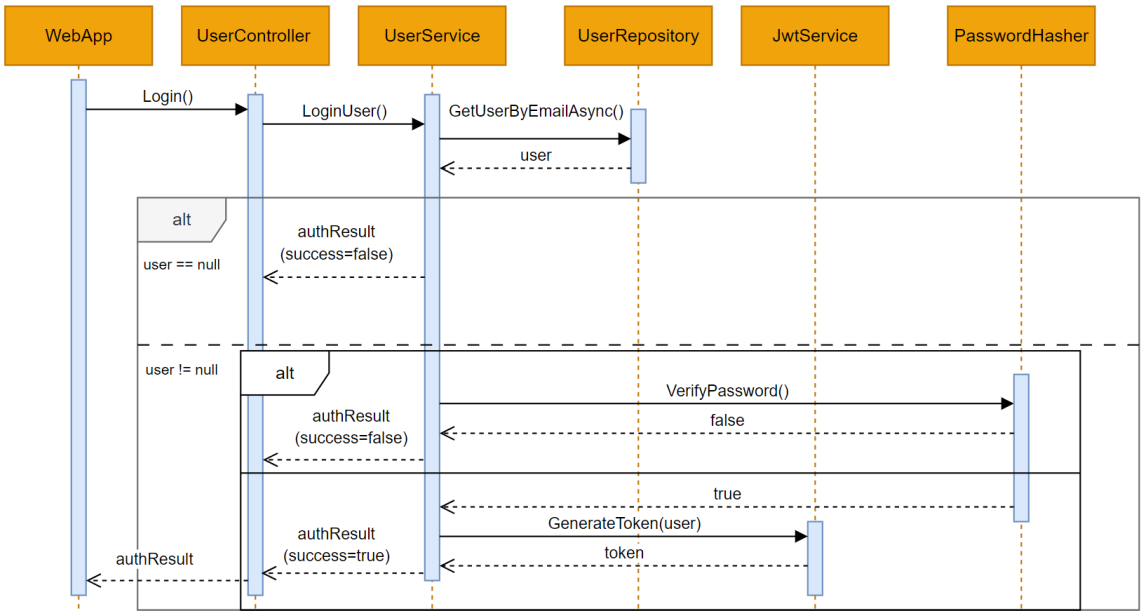

Figure 4.16: Login flow

5. CONCLUSIONS

5.1 BIBLIOGRAPHY

5.2 AUTHENTICITY DECLARATION

6. 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] B. Lutkevich, "Github definition," 2024. [Online]. Available: https://www.techtarget.com/searchitoperations/definition/GitHub.

[9] TheThingsNetwork, "What are lora and lorawan?," [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/what-is-lorawan/.

[10] MSDN, "Http," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

DECLARAȚIE DE AUTENTICITATE A LUCRĂRII DE FINALIZARE A STUDIILOR *

Subsemnatul **ION POPESCU**, legitimat cu **CI** seria **TZ** nr. **100772**, CNP **5000102355779**, autorul lucrării **TITLUL LUCRĂRII DE FINALIZARE A STUDIILOR** elaborată în vederea susținerii examenului de finalizare a studiilor de **LICENȚĂ** organizat de către Facultatea **AUTOMATICĂ ȘI CALCULATOARE** din cadrul Universității Politehnica Timişoara, sesiunea **IUNIE** a anului universitar **2022**, coordonator **dr.ing. MIHAI IONESCU**, luând în considerare conținutul art. 34 din *Regulamentul privind organizarea și desfășurarea examenelor de licență/diplomă și disertație*, aprobat prin HS nr. 109/14.05.2020 și cunoscând faptul că în cazul constatării ulterioare a unor declarații false, voi suporta sancțiunea administrativă prevăzută de art. 146 din Legea nr. 1/2011 – legea educației naționale și anume anularea diplomei de studii, declar pe proprie răspundere, că:

- această lucrare este rezultatul propriei activități intelectuale,
- lucrarea nu conține texte, date sau elemente de grafică din alte lucrări sau din alte surse fără ca acestea să nu fie citate, inclusiv situația în care sursa o reprezintă o altă lucrare/alte lucrări ale subsemnatului.
- sursele bibliografice au fost folosite cu respectarea legislației române şi a convențiilor internaționale privind drepturile de autor.
- această lucrare nu a mai fost prezentată în fața unei alte comisii de examen de licență/diplomă/disertație.

Timișoara,

Data Semnătura

_____ _____

* Declarația se completează „de mână” şi se inserează în lucrarea de finalizare a studiilor, la sfârşitul acesteia, ca parte integrantă.