JOMO KENYATTA UNIVERSITY OF AGRICULTURE AND TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

BSc. ELECTRICAL AND ELECTRONICS ENGINEERING (LC)

PROJECT REPORT

PROJECT TITLE:

IoT-BASED SMART IRRIGATION SYSTEM FOR TECHNOLOGY DRIVEN CROP FARMING

Submitted by:

MWANGI TONY BRIAN NGUGI – ENE211-0005/2018

PROJECT SUPERVISOR:

MR. LINUS ALOO

A Final Year Project Report submitted to the Department of Electrical and Electronic Engineering in partial fulfillment of requirements for the award of a Bachelor of Science Degree in Electrical and Electronics Engineering.

DECEMBER 2023

DECLARATION

This project report is my original work except where due acknowledgment is made in the text and to the best of my knowledge has not been previously submitted to Jomo Kenyatta University of Agriculture and Technology or any other institution for the award of a degree or diploma.

SIGNATURE: **……** DATE: **……...**

NAME: **MWANGI TONY BRIAN NGUGI** REG No.: **ENE211-0005/2018**

TITLE OF THE PROJECT:

IoT-BASED SMART IRRIGATION SYSTEM FOR TECHNOLOGY DRIVEN CROP FARMING

SUPERVISOR CONFIRMATION

This project has been submitted to the Department of Electrical and Electronic Engineering, Jomo Kenyatta University of Agriculture and Technology with my approval as the project supervisor:

NAME: **Mr. Linus Aloo**

SIGNATURE: **……** DATE: **……**

ABSTRACT

Irrigation is an important process for crop growth and requires efficient water usage. However, traditional irrigation systems are often water wasteful and inefficient. Moreover, the changing weather conditions make it difficult to maintain an optimum level of irrigation, since unpredictable changes in rainfall and temperature cause direct changes in soil moisture content. Additionally, the existing smart irrigation systems do not take into consideration the different water requirements for different crops, such as maize, cabbage, kales, etc; as well as the different water requirements for crops in different geographical areas, e.g., the different counties in Kenya. Given the above problem, an improved/modified smart irrigation system is needed to optimize the water usage in agriculture and increase crop yield, based on the type of crop and location, and the real-time soil moisture content. The project sought to develop an improved system where the farmer could configure the type of crop and location of their farm (in Kenya) and monitor (remotely) the water supplied to the various crops. The project involved the development of an improved smart irrigation system using IoT and sensors. The system was designed to monitor and control the irrigation process using data obtained from three sources: a database with information on the water requirements of different crops at various stages of growth; a weather API with weather conditions for different geographical areas in Kenya; and soil moisture sensors installed at strategic locations within the farm. The system would then analyse the data using a control algorithm, a microcontroller and a solenoid valve to adjust the amount of water supplied to the crops based on the current soil moisture conditions, and weather in the given location, ensuring the water matches the water requirements of the specific crop. The system was also equipped with a mobile app to allow farmers to remotely control and monitor the improved irrigation system. The improved smart irrigation system optimized water usage and reduced wastage, resulting in increased crop yield and better resource management.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

API – Application Programming Interface

ET_{crop} – Specific crop water requirement

ET_O – Evapotranspiration rate

FDR – Frequency Domain Reflectometry

GIS – Geographic Information System

GPS – Global Positioning System

GSM – Global System for Mobile Communications

IDE – Integrated Development Environment

IoT – Internet of Things

JSON – JavaScript Object Notation (JSON)

K_C – Crop factor

LCD – Liquid Crystal Display

LoRa – Long Range

SMS – Short Messaging Service

TCP/IP – Transmission Control Protocol/Internet Protocol

TDR – Time Domain Reflectometry

VRI – Variable Rate Irrigation

Wi-Fi – Wireless Fidelity

CHAPTER 1: INTRODUCTION

1.1 Background Information

The field of agriculture plays a vital role in sustaining global food production and addressing the increasing demands of a growing population. According to [1], countries with different degrees of economic development have a role in ensuring food security, with the agricultural sector playing an essential role in enhancing food availability. Irrigation, as an essential component of crop farming, ensures the adequate supply of water for optimal plant growth. Additionally, [2] notes that while water is becoming a very scarce natural resource, which thus limits crop production, the demand for food production is increasing continuously due to the rapid rate of human population growth. This situation calls for the development and improvement of efficient agricultural practices such as precision irrigation water management. However, conventional irrigation systems have significant drawbacks in terms of water usage efficiency and adaptability to changing environmental conditions [2]. These limitations have led to the emergence of smart irrigation systems that leverage technology to overcome these challenges.

The IoT (Internet of Things) has emerged as a transformative technology, offering opportunities for advancements in various domains, including agriculture. By connecting physical devices and sensors through the internet, IoT enables the collection and analysis of real-time data, leading to improved decision-making and resource management [3]. In the context of agriculture, IoT-based systems have the potential to revolutionize the way irrigation is conducted, addressing water wastage and enhancing crop productivity.

The proposed project aims to develop an IoT-based smart irrigation system specifically designed for technology-driven crop farming. The system will address the limitations of traditional irrigation methods and existing smart irrigation systems by considering the specific water requirements of

different crop types and geographic locations. By integrating data from multiple sources, that is, a database with crop water requirements and real-time weather conditions, and soil moisture sensors, the system will provide farmers with real-time insights and control over the irrigation process.

The challenges faced in the current application area of smart irrigation systems include water wastage due to inefficient irrigation practices, imprecise estimation of crop water requirements, and the lack of adaptability to varying weather patterns and soil conditions [4]. These issues directly impact crop yield, water resource management, and overall sustainability of agricultural practices. To overcome these challenges, the proposed system will utilize IoT technology and sensors to monitor crucial parameters, such as soil moisture content and weather conditions, and leverage a control algorithm and microcontroller to adjust irrigation based on real-time data. By doing so, the system will optimize water usage, reduce wastage, and ensure that crops receive the appropriate amount of water, tailored to their specific needs.

1.2 Problem Statement

In the field of crop farming, efficient irrigation practices are crucial for maximizing crop yield and conserving water resources. However, traditional irrigation systems often suffer from inefficiencies and inadequate adaptability to changing environmental conditions [5]. Existing smart irrigation systems, while an improvement over conventional methods, still face limitations in addressing the diverse water requirements of different crops and the varying geographic locations in which they are cultivated [6]. The following problem statement outlines the deficiencies in the existing methods and the general issues in the application area, emphasizing the need for an improved IoT-based smart irrigation system.

From the perspective of existing methods, a significant limitation lies in their inability to consider the specific water requirements of different crops at various growth stages. Conventional irrigation

systems lack the intelligence to differentiate between crop types and adjust irrigation accordingly [6]. Similarly, current smart irrigation systems often overlook the fact that different crops have distinct water needs, resulting in suboptimal irrigation practices. This oversight leads to either under-irrigation, leading to stunted growth and reduced yields, or over-irrigation, resulting in water wastage and increased susceptibility to diseases.

Another limitation of existing methods is the inadequate consideration of the varying geographic locations where crops are cultivated. Climate and weather conditions differ across different regions, impacting the irrigation requirements of crops [7]. Further, according to Hatfield and Prueger [8], temperature is one of the primary factors that affect the rate of plant development. Additionally, water deficits and excess soil water also increases the temperature effects on crops. This relationship thus calls for a proper understanding of the interaction of temperature and water in the development of effective irrigation systems that cater for the temperature changes. Yet, existing smart irrigation systems often lack the capability to integrate real-time weather data into the irrigation process, limiting their ability to adjust water supply based on prevailing weather conditions [6]. Consequently, improper irrigation practices persist, leading to inefficient water usage and suboptimal crop growth. Moreover, the lack of remote monitoring and control features in conventional and existing smart irrigation systems hinders effective resource management. Farmers are unable to remotely access and control the irrigation process, making it challenging to respond promptly to changing conditions or adjust irrigation schedules based on real-time data. This limitation results in water wastage, increased labor requirements, and reduced overall efficiency.

To address these limitations, an improved IoT-based smart irrigation system is proposed. By leveraging IoT technology, sensors, and data analytics, the proposed system aims to optimize water

usage by considering the specific water requirements of different crops and the real-time weather and soil moisture conditions in different geographic locations. The system will provide farmers with remote monitoring and control capabilities, allowing them to manage irrigation efficiently, conserve water resources, and enhance crop yields.

1.3 Justification

The proposed IoT-based smart irrigation system for technology-driven crop farming offers significant advantages over existing methods, addressing the limitations identified in the problem statement. The following justifications highlight the technical strengths of the proposed solution and the benefits it brings to the field of agriculture.

Firstly, the integration of IoT technology, sensors, and data analytics in the proposed system enables real-time monitoring and control of the irrigation process. This technical capability overcomes the limitations of conventional irrigation systems, which lack remote access and control features. By allowing farmers to remotely monitor and manage irrigation, the proposed system enhances convenience, efficiency, and resource management [9] . Farmers can respond promptly to changing environmental conditions, such as sudden rainfall or increased temperature, and make necessary adjustments to irrigation schedules and water supply.

Secondly, the utilization of sensors, including soil moisture sensors, in the proposed system ensures accurate and precise monitoring of soil moisture content. This technical feature addresses the shortcomings of existing smart irrigation systems, which often rely on less accurate estimations or measurements of soil moisture levels. The real-time data collected by the sensors enable the system to make informed decisions regarding the amount and timing of water supply to the crops [10]. By avoiding over-irrigation or under-irrigation, the proposed system optimizes water usage, reduces water wastage, and promotes efficient crop growth.

Furthermore, the incorporation of a control algorithm and a microcontroller in the proposed system enables intelligent irrigation control based on data analysis and decision-making. This technical aspect is a significant improvement over existing smart irrigation systems, which may lack effective algorithms for precise irrigation management. The control algorithm, coupled with the integration of a crop database and weather data, allows the system to tailor the water supply to the specific requirements of different crops and geographic locations. This optimization ensures that crops receive the right amount of water at the right time, leading to increased crop yield and improved resource management.

From a technical standpoint, the proposed IoT-based smart irrigation system leverages advanced technologies and data-driven decision-making to address the limitations of existing methods. By combining remote monitoring, accurate sensor measurements, and intelligent control algorithms, the system offers precise irrigation management, efficient water usage, and increased crop productivity. These technical strengths not only contribute to the sustainability of agriculture but also offer economic benefits to farmers through improved crop yield and resource utilization.

1.4 Objectives

1.4.1 Main Objective

- To develop an improved smart irrigation system that optimizes water supply for different types of crops based on their water requirements, weather, and geographical location.

1.4.2 Specific Objectives

i. To develop a control algorithm and integrate it with a microcontroller and valve, to adjust the amount of water supplied to crops based on real-time soil moisture data, weather conditions, and crop-specific water requirements.

ii. To develop a database with information on the water requirements of different crops at various stages of growth and weather conditions for different geographical areas in Kenya.

iii. To develop a mobile app where the farmer can monitor and control the smart irrigation system remotely.

1.5 Scope

The implemented project on IoT-based smart irrigation system for technology-driven crop farming had a defined scope that outlined its focus areas and specified what it would not cover. Additionally, the project concentrated on the irrigation requirements of three specific test crops: maize, cabbage, and kales. The project primarily focused on the development and implementation of the IoT-based smart irrigation system, including the integration of soil moisture sensors, data collection and analysis of soil moisture and weather (temperature, cloud cover, relative humidity, and wind speed), control algorithms, and remote monitoring capabilities. The system was tailored to meet the different water requirements of maize, cabbage, and kales, considering their specific growth stages. Moreover, the project addressed the optimization of water usage based on real-time soil moisture measurements and weather conditions for different geographical locations in Kenya. The primary focus of this project was small-scale farming, which was practised in smallholder farms of an average of 0.47 hectares (approximately 1.2 acres) in Kenya at the time. However, the project was a prototype for demonstration and thus did cover the actual installation of the physical irrigation infrastructure in the farm, such as pipes, valves, or sprinklers. The primary emphasis was on the IoT-enabled components and the software implementation required for data analysis, control, and remote access.

CHAPTER 2: LITERATURE REVIEW

2.0 Introduction to the Chapter

This chapter discusses the significance of effective irrigation in agriculture and the limitations of existing methods in meeting diverse crop water requirements. To address these issues, an improved IoT-based smart irrigation system is proposed. The system's components, such as soil moisture sensors and a microcontroller, are explored, along with existing studies on smart irrigation. Shortfalls of current methods include imprecision, limited responsiveness, and lack of integration. To overcome these, the proposed system integrates real-time data and remote monitoring for optimal water management. The chapter aims to enhance water usage efficiency and increase crop productivity for technology-driven crop farming in Kenya's smallholder farms.

2.1 Existing Technologies/Methods

2.1.1 Manual Methods

Manual irrigation methods have been traditionally used in crop farming, relying on human intervention to estimate and control the water supply. These methods include techniques such as flood irrigation, furrow irrigation, and sprinkler irrigation [11]. However, manual methods lack precision in determining the optimal amount of water required by different crops at specific growth stages. They heavily rely on the experience and judgment of farmers, leading to inconsistencies in water application and potential water wastage.

2.1.2 Automatic Methods

Automatic irrigation systems have been developed to improve water usage efficiency. These systems utilize timers, sensors, and controllers to automate the irrigation process. Timer-based systems enable scheduled irrigation, but they do not consider real-time soil moisture content or

weather conditions [12]. Sensor-based systems, such as soil moisture sensors or tensiometers, measure soil moisture levels to determine irrigation needs. However, they often lack integration with other data sources and may not account for specific crop water requirements or changing weather patterns.

2.1.3 Specific Methods

Certain specific methods have emerged to enhance irrigation practices. IoT-based systems integrate sensors, connectivity, and data analytics to enable intelligent irrigation. By collecting real-time data on soil moisture, weather conditions, and crop water requirements, IoT systems can optimize irrigation scheduling and water supply. Internal control algorithms and microcontrollers are also employed to analyze data and adjust irrigation based on predefined rules and feedback loops.

Despite these existing technologies and methods, there are still limitations in addressing the diverse water requirements of different crops and varying geographical locations. The lack of integration between multiple data sources, limited control over irrigation practices, and the absence of remote monitoring and control features hinder optimal water usage and crop productivity [13]. Therefore, an improved smart irrigation system utilizing IoT, sensors, and advanced data analysis is required to overcome these limitations and enhance irrigation practices for technology-driven crop farming.

2.2 Components

The proposed IoT-based smart irrigation system for technology-driven crop farming incorporates various components that work together to enable efficient irrigation practices. Each component plays a crucial role in the system's operation. The components that were used in the project are explored in greater detail in this section.

2.2.1 Soil Moisture Sensors

Soil moisture sensors collect soil moisture data and feed it into the microcontroller for analysis. Keeping soil moisture balanced is of great importance to the healthy growth of plants since soil moisture has a direct impact on plants' respiration and photosynthesis. High-accuracy soil moisture sensors are essential for precise monitoring of soil moisture content. There are a variety of soil moisture sensors on the market, which are described below.

i. Capacitive soil moisture sensors

The capacitive soil moisture sensor detects moisture through the moisture sensitive capacitor. It typically uses a ceramic material, which changes the permittivity by absorbing the moisture in the environment, and the capacitance value of the capacitor changes. The sensor converts the capacitance value into an electrical signal and outputs a percentage value to monitor moisture. This sensor is often built as a prototype product, so if used as a professional measurement, measurement data may not be accurate and reliable. *Figure 2.1* shows a capacitive soil moisture sensor.

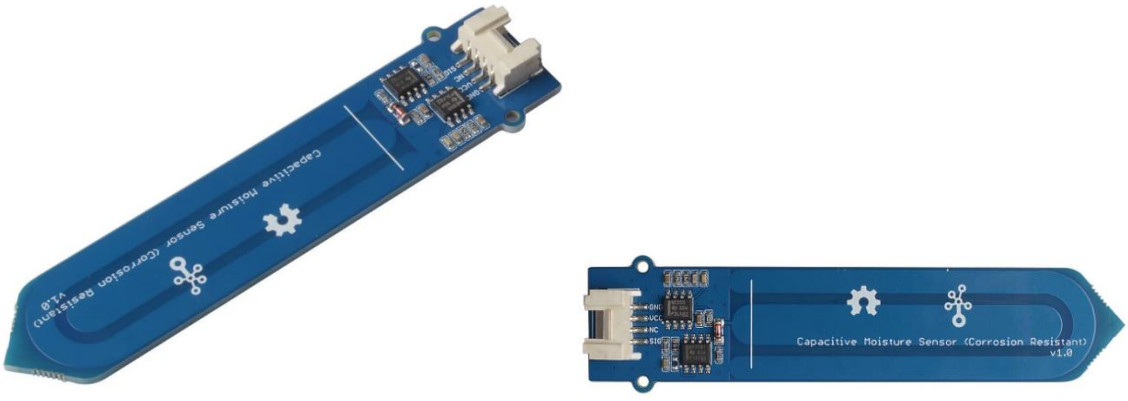

Figure 2.1: Capacitive soil moisture sensor [14]

ii. Time Domain Reflectometry (TDR) soil moisture sensors

Like a radar system, TDR (Time Domain Reflectometry) uses a crystal oscillator to generate high-frequency signals that are transmitted to a metal probe. At this point, the generated signal and the

returned signal will be superimposed, and then the amplitude of the signal will be measured and converted into soil moisture value [15]. The value that is measured in this process is the dielectric constant, and there is a correspondence between the dielectric constant of the soil and the soil moisture. For example, as moisture in soil increases, its dielectric constant increases correspondingly. TDR has strong independence, its results are independent of soil type, density and temperature. Another important point is that TDR can measure soil moisture under freezing.

iii. Frequency Domain Reflectometry (FDR) soil moisture sensors

Since TDR equipment was expensive, in the late 1980s, many companies began to measure the moisture of soil in a simpler way than TDR — FDR (Frequency Domain Reflectometry) [15]. FDR uses the principle of electromagnetic pulse, according to the electromagnetic wave propagation in a medium frequency to measure the apparent dielectric constant (ε) of the soil, so soil volumetric water content is obtained. Compared with TDR test principle, FDR has almost all the advantages of TDR, FDR is not only cheaper than TDR, but also has a shorter measurement time, high measurement accuracy after a specific soil calibration, and the shape of the probe is not limited, can be measured at multiple depths at the same time, data acquisition implementation is easier [15]. FDR has the advantages of simple, safe, fast and accurate, long-term measurement, wide range, and simple calibration.

2.2.2 Microcontroller

The microcontroller acts as the brain of the smart irrigation system. It receives inputs of soil moisture content from the soil moisture sensors and weather information such as temperature and cloud cover from the database and process the data to make irrigation decisions on when to open the outlet valve and for how long. Microcontrollers like Arduino, Raspberry Pi or ESP 32 are commonly used in such systems. The microcontroller will run a control algorithm that considers

inputs such as soil moisture levels, weather data, and crop-specific irrigation parameters stored in the database. Based on these inputs, the microcontroller calculates the required amount of water and controls the opening and closing of the control valves.

Figure 2.2 shows the Arduino, Raspberry Pi and ESP 32 microcontrollers.

Figure 2.2: Arduino, Raspberry Pi or ESP 32 microcontrollers (from left to right) [16]

A comparison of the Arduino, Raspberry Pi or ESP 32 microcontrollers in terms of key features, advantages and disadvantages is presented in *Table 2.1*.

Table 2.1: Comparison of Arduino, Raspberry Pi, and ESP 32 microcontrollers

Microcontroller	***Key Features***	***Advantages***	***Disadvantages***
Arduino	- Simple and user-friendly development platform - Limited computational power - Extensive library support - Easily expandable with shields and modules - Generally lower cost compared to other options	- Easy to learn and suitable for beginners - Extensive library support for various sensors - Easily expandable with shields and modules - Lower cost compared to Raspberry Pi and ESP32	- Limited processing power and memory - Not suitable for resource-intensive tasks - Lack of native Wi-Fi and Bluetooth support
Raspberry Pi	- Full-fledged single-board computer - Linux-based OS - Wi-Fi and Bluetooth built-in - HDMI output for display and audio support	- Significant computational power for complex projects - Runs various programming languages - Supports multitasking and multiple processes - Larger memory and storage capacity	- Higher cost compared to Arduino and some ESP32 models - Requires additional components to function - Higher power consumption compared to Arduino
ESP32	- Dual-core microcontroller - Built-in Wi-Fi and Bluetooth capabilities - Sufficient GPIO pins - Low-power modes for energy efficiency - Compatible with Arduino IDE	- Balance of processing power and power efficiency - Built-in Wi-Fi and Bluetooth for wireless connectivity - Real-time data processing and communication support - Suitable for battery-powered applications	- May still have limitations for highly complex tasks - Limited memory for large datasets or intensive tasks - Costlier than basic Arduino boards - Less powerful than some Raspberry Pi models

2.2.3 Control Valves

Electrically operated control valves regulate the water flow to different crop areas. These valves are controlled by the microcontroller based on the irrigation decisions. Solenoid valves are commonly used in smart irrigation systems due to their ability to provide precise control over water flow. The microcontroller sends signals to the control valves, either opening or closing them, to ensure that the right amount of water is supplied to each crop area.

2.2.4 Database

A comprehensive database stores information on the water requirements of different crops at various growth stages. This database is accessed by the system to retrieve crop-specific irrigation parameters. It contains data such as optimal soil moisture levels, crop water consumption rates, and recommended irrigation schedules. The system uses this data to optimize water supply to each crop based on their specific needs at different growth stages. Additionally, a weather application programming interface (API) integrated to the database provides real-time and forecast weather information, which is essential in making irrigation decisions.

2.2.5 Communication Module

The communication module establishes connectivity between the smart irrigation system and external devices, such as smartphones or computers. It enables remote monitoring and control of the system. These modules allow farmers to access the system through a mobile application or web interface, providing real-time updates on soil moisture levels, weather conditions, and the ability to adjust irrigation settings remotely. In the proposed IoT-based smart irrigation system, two communication modules may be commonly used for data exchange and remote control: the GSM (Global System for Mobile Communications) module and the Wi-Fi module. An overview of the GSM and Wi-Fi modules is as presented in *Table 2.2*.

Table 2.2: Comparison of the GSM and Wi-Fi modules

Module	***Key Features***	***Advantages***	***Disadvantages***
GSM Module	-GSM modules enable communication over cellular networks, allowing the system to operate in remote areas with cellular coverage. -They support data transfer through SMS (Short Message Service) and GPRS (General Packet Radio Service). -The modules typically have SIM card slots for network authentication and data transfer.	-Wide Coverage: GSM modules can communicate over large areas with cellular network coverage, making them suitable for remote and rural locations where Wi-Fi coverage may be limited or unavailable. - Independent of Wi-Fi: Since GSM modules use cellular networks, they do not rely on Wi-Fi infrastructure, which can be advantageous in areas where Wi-Fi connectivity is unreliable or non-existent. - Simplicity: GSM-based communication is relatively simple to set up, and data transfer via SMS or GPRS is straightforward to implement.	- Data Costs: Data transfer over cellular networks may incur additional costs, depending on the data plan or cellular provider's pricing. - Limited Data Rates: Compared to Wi-Fi, GSM modules generally have slower data transfer rates, which can affect the speed of data exchange and remote-control commands.
Wi-Fi Module	- Wi-Fi modules enable communication over local Wi-Fi networks, providing high-speed data transfer capabilities. - They support TCP/IP communication, allowing seamless integration with the internet and cloud services. - Wi-Fi modules typically require Wi-Fi access points or routers for connectivity.	- High Data Rates: Wi-Fi modules offer faster data transfer rates compared to GSM modules, making them suitable for applications requiring real-time data exchange and responsive remote control. - Cost-Efficient: Wi-Fi communication is often cost-efficient for devices operating within the range of existing Wi-Fi networks, as there are no additional data costs.	- Limited Coverage: Wi-Fi modules are constrained by the range of the local Wi-Fi network, which may limit their use in remote or rural areas without Wi-Fi coverage. - Dependency on Wi-Fi Infrastructure: Since Wi-Fi modules require access to Wi-Fi networks, their functionality may be affected in areas with weak or unstable Wi-Fi signals.

In the context of the smart irrigation system, the choice between GSM and Wi-Fi modules depends on factors such as the location of the farm, availability of Wi-Fi infrastructure, data transfer requirements, and cost considerations. In remote areas without reliable Wi-Fi coverage, GSM modules may be more suitable for data exchange. On the other hand, for farms located within Wi-Fi coverage, Wi-Fi modules offer higher data rates and cost-efficiency for real-time communication and remote-control functionalities.

2.2.6 Mobile Application

A user-friendly mobile application provides farmers with a convenient interface to interact with the smart irrigation system. The application allows farmers to configure crop types, set irrigation schedules, monitor soil moisture levels, and receive notifications and alerts. Through the mobile application, farmers can remotely control and monitor the irrigation process, making adjustments as needed.

By integrating these components into the IoT-based smart irrigation system, farmers can achieve precise and efficient irrigation management. The system collects real-time soil moisture data from the soil moisture sensor and weather conditions from the database, analyzes it using the microcontroller and control algorithm, and controls the water supply through the control valves. This integration optimizes water usage, promotes healthy crop growth, and increases overall crop yield.

2.3 Existing studies/works/solutions

Several existing studies and solutions have been developed in the field of smart irrigation systems. This section presents a compilation of specific studies that have contributed to the advancement of irrigation technologies. Research by Mutambara et al. [5] highlights the challenges faced by smallholder irrigation schemes in Africa and Asia. In Africa, there are few successful and

sustainable farmer-managed smallholder irrigation schemes. Lack of farmer participation, inadequate cost recovery, poor water management reforms, and institutional issues contribute to their inefficiency. In contrast, Asia has a long history of successful smallholder irrigation schemes, attributed to farmer participation, cost recovery, and efficient water management reforms. The article emphasizes the need for African countries to learn from Asia's best practices to improve the sustainability of their irrigation schemes and ensure successful smallholder irrigation. Reforms in land tenure, water sector, and institutional management are crucial for achieving sustainable and resilient irrigation systems.

A study by Brahmanand and Singh [2] discusses precision irrigation water management, defined as providing an optimal quantity of water to crops' root zones at the right time. Precision irrigation aims to maximize crop productivity with minimal water usage, utilizing modern wireless communication, monitoring systems, and automation. Automation, through sensor-based irrigation, enables precise water application. Factors like crop type, climate, soil, and weather parameters are considered for irrigation scheduling. The article highlights the use of drip irrigation, GPS, GIS, and Variable Rate Irrigation (VRI) technologies to achieve efficient water distribution. Precision irrigation relies on satellite data, energy-efficient valves, flow meters, and soil sensors for data transmission. Overall, the article emphasizes using advanced technologies for optimal water management in agriculture. Similarly, another article by Obaideen et al. [9] discusses the use of artificial intelligence (AI), the Internet of Things (IoT), and smart systems in agriculture, specifically focusing on SMART irrigation. SMART irrigation automates irrigation systems based on real-time soil and weather conditions, conserving water usage. The implementation of IoT and sensor technologies allows farmers to monitor and control their fields efficiently, leading to increased productivity with minimal water loss. Various communication technologies, such as Wi-

Fi, GSM, and LoRa are used to enable data transmission in IoT devices for irrigation systems. *Figure 2.3* shows the various elements applied in conventional smart irrigation systems.

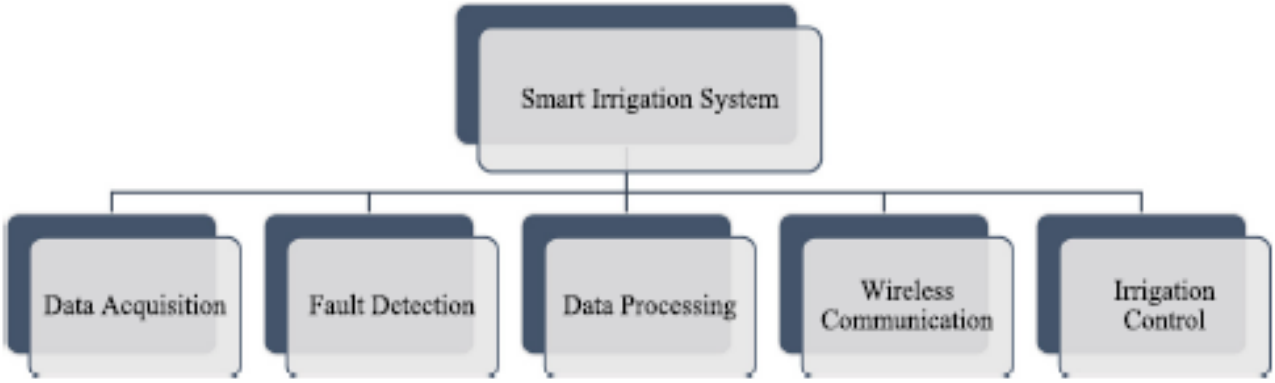

Figure 2.3:Elements of conventional smart irrigation systems [9]

In another study, Baradaran and Tavazoei [17], address the critical issue of water consumption in agricultural regions, particularly arid and desert farms, where water resources are declining due to increasing population and crop demands. Traditional irrigation methods lead to water loss, and farmers' lack of understanding of plant water needs can result in over-irrigation. Intelligent and automated irrigation systems, considering various factors like evapotranspiration and soil moisture, offer a solution to reduce water consumption and improve irrigation efficiency. The study proposes a method using remote sensing estimate water requirements without the need for additional hardware equipment like sensors or IoT-based systems.

Further, according to a study by Meru University of Science and Technology [18] there is a need for increased food production in Kenya due to a growing population and unreliable rain-fed agriculture caused by erratic climate change. The country is classified as water-scarce, and there is a necessity to conserve and efficiently use available water resources for irrigation. The team from Meru University of Science and Technology developed an automatic irrigation system using soil moisture sensors to detect dry soil and channel water to crops, reducing water wastage. The

system could be operated remotely through SMS commands, making irrigation more efficient and reducing labor costs. Plans for further upgrades include real-time soil nutrient evaluation and crop stress monitoring.

Finally, a local solution by Odhiambo and Odhiambo [19] demonstrates the development of a climate-smart, solar-powered water pumping system for irrigating a 25-hectare banana plantation in Garissa, Kenya. The system uses high-efficiency photovoltaic modules to generate 23kW of electrical power from solar energy. It pumps water from the Tana river through a transmission pipe system to the banana plantation, supporting 25 households. The system's easy operation and maintenance lead to a gross income of Kenya shilling 40 million per year, transforming the community from food aid dependence to secure middle-income status. The model showcases a scalable and sustainable livelihood transitioning system. These studies demonstrate the diverse range of approaches that have been applied to enhance smart irrigation systems. They provide valuable insights and serve as references for the development and improvement of the proposed IoT-based smart irrigation system for technology-driven crop farming.

2.4 Shortfalls of existing methods

Existing methods and solutions for smart irrigation systems, as identified in Section 2.1 (Existing Technologies/Methods) and discussed in various studies (Section 2.3), exhibit several notable shortfalls that hinder their effectiveness in optimizing water usage and crop yield. These shortcomings are found in both manual and automatic irrigation approaches. Manual irrigation methods, which rely on human intervention and judgment, suffer from imprecision in determining the optimal amount of water required by different crops at specific growth stages. Farmers often lack the necessary knowledge and expertise to accurately assess crop water needs, leading to inconsistencies in water application [20]. As a result, there is a high risk of over-irrigation, which

not only wastes water but also increases the likelihood of waterlogging and nutrient leaching, adversely affecting plant health and growth.

Automatic irrigation systems, though designed to improve water usage efficiency, still face significant limitations. Timer-based systems, for instance, allow for scheduled irrigation, but they do not consider real-time soil moisture content or weather conditions. This lack of responsiveness to changing environmental factors can lead to both under- and over-irrigation, negatively impacting crop health and yield [12]. Sensor-based systems, such as those using soil moisture sensors or tensiometers, attempt to address the limitations of timer-based methods by measuring soil moisture levels to determine irrigation needs [12]. However, these systems often lack integration with other data sources, such as weather forecasts or specific crop water requirements. As a result, they may not accurately assess the water needs of crops, particularly in the context of varying weather patterns or changing growth stages.

Additionally, many of the existing methods fail to account for the diverse water requirements of different crops and the varying geographical conditions of different regions. The lack of integration between multiple data sources, such as soil moisture, weather forecasts, and crop water requirements, hinders the development of comprehensive and adaptive irrigation strategies. This limitation is especially critical as climate change leads to more unpredictable weather patterns, necessitating more sophisticated irrigation approaches. Moreover, most existing systems lack remote monitoring and control features, making it challenging for farmers to adjust irrigation practices in real-time. Without the ability to remotely access and control irrigation systems, farmers may not be able to respond promptly to changing weather conditions or unexpected events, leading to suboptimal water usage and reduced crop productivity.

The complexity of irrigation scheduling is also not adequately addressed in many of the current methods. Factors like crop type, climate, soil type, and weather parameters all influence the optimal irrigation schedule for maximum crop productivity [21]. However, existing systems often do not consider these factors comprehensively, leading to suboptimal irrigation practices. Furthermore, there is a need for cost-effective solutions that can be readily adopted by smallholder farmers, particularly in developing countries like Kenya where water resources may be limited, and access to advanced technologies may be challenging. Many of the existing methods rely on expensive sensors, data analytics, and sophisticated control systems, making them less accessible to resource-constrained farmers [22].

2.5 Improvements or suggested solutions

The proposed IoT-based smart irrigation system offers some improvements and suggested solutions to overcome the identified shortcomings discussed in Section 2.4 (Shortfalls of Existing Methods). Firstly, to address the lack of consideration for crop-specific water requirements, the system will incorporate a comprehensive database that contains information on the water needs of different crops at various growth stages. By allowing farmers to configure the type of crop being cultivated, the system will accurately determine the appropriate amount of water required for each crop. This will enable precise irrigation practices tailored to the specific needs of individual crops, optimizing water usage and promoting healthy crop growth.

Secondly, the system will integrate real-time data collected from soil moisture sensors and a weather API. By employing a control algorithm, the system will analyse this data to dynamically adjust the irrigation process based on the current soil moisture conditions and weather parameters, such as temperature and cloud cover. This real-time data integration will enable adaptive irrigation control, ensuring that water supply aligns with the immediate requirements of the crops. It will

prevent over-irrigation and water wastage while avoiding under-irrigation and crop stress. To account for geographic variations in water requirements, the system will factor in the farm's location details. This information will be used to access specific weather data for the given location, ensuring that irrigation decisions align with the local climate conditions. By considering the unique characteristics of different regions, the system will optimize irrigation practices for the specific geographic area, ensuring efficient water usage and crop health.

The proposed system will be designed with scalability and flexibility in mind. Although the initial focus of this system targets small-scale farmers owning smallholder farms, it can later be adapted to different farm sizes and configurations. Farmers can customize the system based on their type of crop, including maize, cabbage, and kales, each with their own distinct water requirements. This flexibility will cater to diverse agricultural settings, accommodating various farming scenarios and promoting widespread adoption. Additionally, the system will incorporate a mobile application that will enables farmers to remotely monitor and control the irrigation system. Through the app, the farmer will be able access real-time data, adjust irrigation settings, and receive notifications and alerts. This remote monitoring and control feature will empower farmers to actively manage the irrigation process, facilitating timely interventions and adjustments for optimal crop growth and water resource management. By implementing these improvements and suggested solutions, the proposed IoT-based smart irrigation system will addresses the shortfalls of existing methods. It will provide an improved and comprehensive solution for optimizing water usage in agriculture, increasing crop yield, and promoting efficient resource management in technology-driven crop farming.

CHAPTER 3: METHODOLOGY

3.1 Project Specifications

i. Power Supply

For protype testing and demonstration, the power supply that was used for this project was two rechargeable 18650 Lithium-Ion batteries connected in series, each with a capacity of 5000mAh and a voltage of 3.7V. The key specifications of the power supply (Lithium-Ion batteries) are as listed in *Table 3.1* [23].

Table 3.1: 18650 battery key specifications

Specification	**Rating**
Part Number:	Lithium ion 3.7V 5000mAh 18650 Li-ion Battery
Rated Capacity:	5000mAh
Nominal Voltage:	3.7V
Max Charge Voltage:	4.2V
Discharge Cut Off Voltage:	2.75V
Charging Current:	2.75V
Max. Continuous Discharging Current:	2A
Cycle Life:	500times
Discharge Rate:	≤3%

The keypad was powered by the microcontroller I/O pins, while the real-time clock (RTC) module was powered independently using the CR2032 Lithium coin battery with a nominal voltage rating

of 3V, and a power rating of 235mAh. The key specifications of the power supply (Lithium coin battery) are as listed in *Table 3.2* [24].

Table 3.2: CR2032 battery key specifications

Specification	**Rating**
Battery Type:	Manganese Dioxide Lithium Battery (Li/MnO2)
Designation:	ANSI / NEDA-5004LC, IEC-CR2032
Average Capacity:	225 mAh to 2.0 volts
Voltage:	3.0V
Maximum Reverse Charge Current:	1 microampere

ii. Voltages

The two 3.7V rechargeable Lithium-Ion batteries were connected in series to give a supply voltage of 7.4V. The 7.4V output from the batteries was regulated to 5V using the LM7805 voltage regulator to supply the ESP32 microcontroller, the capacitive soil moisture sensors, the LCD display module, and the relay module. Additionally, the MT3608 buck-boost converter was used to boost the 7.4V output from the batteries to 12 V to power the DC solenoid valve. The voltage levels for the various components used in the project are as summarized in *Table 3.3*.

Table 3.3: Key voltage specifications

Component	**Voltage rating (V)**
18650 batteries	3.7V
CR2032 battery	3.0V
LM7805 regulator	Input: 7.4V; Output: 5.0V
MT3608 buck-boost converter	Input: 7.4V; Output: 12.0V
ESP32 microcontroller	Input (V_{in}): 5.0V; Output (GPIO pins): 3.3V
Real time clock (RTC) module	2.2V - 5.5V
Capacitive soil moisture sensors	5.0V
LCD I2C display module	5.0V
Relay module	5.0V
DC solenoid valve	12.0V

iii. Currents

The approximate current specifications for the components are as listed in *Table 3.4*.

Table 3.4: Key current specifications

Component	**Current rating (A)**
18650 batteries	Max. discharge current: 2A
CR2032 battery	Max. discharge current: 2A
LM7805 regulator	Approx. 1.5A
MT3608 buck-boost converter	Approx. 1.0 - 1.5A
ESP32 microcontroller	95 -240mA
Real time clock (RTC) module	3µA
Capacitive soil moisture sensors	5mA
LCD I2C display module	30mA
Relay module	10A (NO) 5A (NC)
DC solenoid valve	600mA

3.2 Block Diagram

Figure 3.1 shows the block diagram that was adopted for the project.

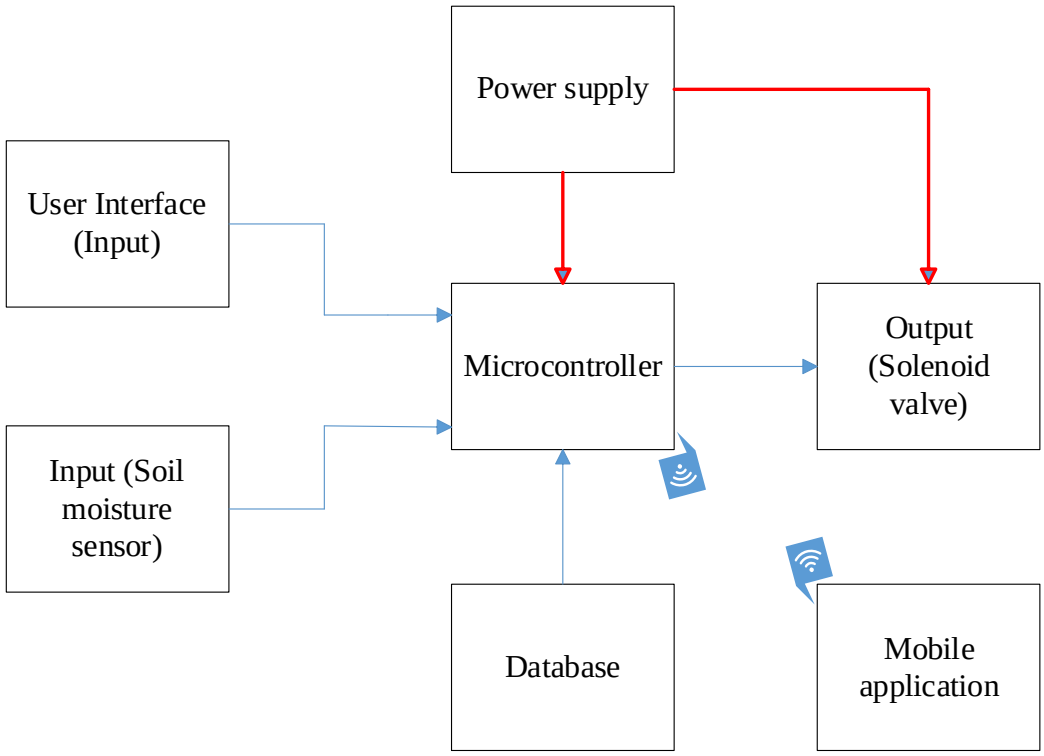

Figure 3.1: Project Block Diagram

A brief explanation of the block diagram is given below, in line with the specific objectives.

i. User interface

The hardware part of smart irrigation system prototype was designed and fabricated to have a 16x02 LCD I2C display module and a 4x4 alphanumeric membrane keypad, which formed the primary interface between the system and the user (farmer). The user interface allowed the user to write and read information to and from the system, respectively. Upon powering the system, a welcome message, and the current date and time were displayed on the LCD. The LCD then informed the user that the system was attempting to establish a Wi-Fi connection, and displayed that the connection had been successful. It then prompted the user to enter the type of crop that

they had planted (maize, kale, or cabbage), and the age of the crop in weeks, using the keypad. The system was internally programmed to only allow valid data for crop type and crop age, and reject invalid information, for example, values of crop age that were beyond the normal range of the three test crops. The user input information was then used to optimize the watering based on the type of crop and its stage of growth. Throughout the rest of the operation of the system, the LCD displayed crucial information, such as the calculated crop water requirements for the planted crop (obtained from the database); the measured soil moisture levels (obtained from the soil moisture sensors); the real-time weather parameters (obtained from the mobile application); and the state of the valve (ON/OFF).

ii. Input (capacitive soil moisture sensors)

The hardware part of the system prototype comprised 3 capacitive soil moisture sensors. The sensors were used to collect soil moisture data from 3 different areas of the soil in the basin (representing the demo farm). The sensors were then calibrated using known amounts of water to map their analog values onto a scale representing soil moisture levels in millimeters (mm) of water. The readings from the three sensors were averaged to obtain the mean soil moisture level in the demo farm. This data would then be used in a control algorithm in the microcontroller to compare the soil moisture level with the water requirements for the specific crop planted, and determine whether to open or close the valve.

iii. ESP32 Microcontroller

The ESP32 microcontroller acted as the brain of the smart irrigation system, and it stored and executed the program/code containing the control algorithm that was used to run the system. The microcontroller performed the following functions in the system:

- It stored and executed the entire program/code that was used to run the system. The elements of this code are briefly listed here, and explained in comprehensive detail in *Section 3.4* of this report.
- It executed program functions that allowed the user to input crop data, that is, the type of crop, and the age of the crop in weeks.
- It executed a program function that obtained specific crop information from the database and real-time weather parameters from the mobile application, and used them to determine the specific crop water requirements.
- It executed a program function to read analogue values from the soil moisture sensors and calibrate the sensors based on known amounts of water.
- It executed a program function to read analogue values from the soil moisture sensors and map them to a scale of soil moisture levels.
- It executed a program function to compare the value of the specific crop water requirements and that of soil moisture levels, and actuate the solenoid valve accordingly to provide optimal water supply.
- It executed the program sections containing various server endpoints to allow for remote wireless communication (via Wi-Fi) between the microcontroller, and the mobile application and database.

iv. Output (solenoid valve)

The 12V DC solenoid valve was used to control the flow of water to the planted crops. The solenoid valve was actuated though the normally open (NO) contacts of a 5V relay module. Based on the signal from the microcontroller, setting the relay pin HIGH energized the relay and closed the NO contacts, which completed the solenoid valve circuit. The solenoid valve coil was thus energized

and the valve opened, allowing water to flow to the crops. Conversely, if the relay received a LOW signal from the microcontroller, it was de-energized, thus the NO contacts opened, and the solenoid valve circuit was opened. The solenoid valve coil was thus de-energized and the valve closed, preventing water from flowing to the crops. This control allowed for supply of water to the crops, ensuring optimum moisture levels were maintained to match the specific crop water requirements.

v. Power supply

As mentioned in *Section 3.1 (i)* of this report, the system was powered using two rechargeable 18650 Lithium-Ion batteries connected in series, each with a capacity of 5000mAh and a rated nominal voltage of 3.7V. This gave a total capacity of 10,000mAh and 7.4V to power the entire hardware system. The 7.4V output from the batteries was regulated to 5V using the LM7805 voltage regulator to supply the ESP32 microcontroller, the capacitive soil moisture sensors, the LCD display module, and the relay module. Additionally, the MT3608 buck-boost converter was used to boost the 7.4V output from the batteries to 12 V to power the DC solenoid valve. The keypad was powered by the microcontroller I/O pins (3.3V), while the real-time clock (RTC) module was powered independently using the CR2032 Lithium coin battery with a nominal voltage rating of 3V, and a power rating of 235mAh.

vi. Database

The database was used to store crop-specific information for each of the three test crops (maize, cabbage, and kales), including the different growth periods of the crops, and the time (in weeks) spent by each crop in each of the growth period. The database also contained static crop data on evapotranspiration rates (ET_O), crop factors (K_C), and specific crop water requirements (ET_{crop}) for each of the crops in their different growth stages. Since these three parameters are affected by variations in weather data, they were first evaluated at specific baseline weather parameters

obtained from meteorological information. The updated weather parameters would then be used to calculate and update the values ET_O, K_C, and ET_{crop} accordingly. Additionally, the database was used to store the user login credentials for the mobile application.

vii. Mobile application

The mobile app allowed the farmer to monitor and control the state of the irrigation system remotely. The application comprised a weather application programming interface (API), which provided real time weather data on temperature, cloud cover, relative humidity, and wind speed, based on the location entered on the app interface through the search functionality, as well as hourly weather forecasts for the day, and daily forecasts for the next 2 days. This real-time data would then be sent to the microcontroller via Wi-Fi, after which the microcontroller would then adjust the crop water requirements and optimize water supply. The application also enabled the user to monitor the state of the smart irrigation system, including the measured value of soil moisture and the state of the relay at any given time. Further, the user could opt to override the valve control algorithm through the mobile app, and thus turn the valve ON/OFF remotely using the application. Finally, in case of any errors or malfunctions in the system, such as flooding, sensor failure, the mobile app would send notifications to the user app to visit the farm and attend to the issue.

3.3 Flow Chart

Figure 3.2 shows the flow chart that was adopted in the software design.

3.3.1 Flow chart explanation

Upon powering the smart irrigation system/prototype, the LCD screen displayed a welcome message and the current date and time. The microcontroller then established a Wi-Fi connection to allow for remote wireless communication with the mobile application and database. It then

prompted the user to enter the type of crop they planted, and the age of the crop in weeks. The microcontroller then determined the stage of growth of the crop based on crop age entered by the user. It then obtained the real-time weather parameters from the mobile app, and the baseline crop water requirements from the database. Using this information, the microcontroller calculated the specific crop water requirements for the growth stage of the selected crop.

The microcontroller then obtained the real-time soil moisture content through the soil moisture sensors. By comparing the calculated crop water requirements and the measured soil moisture content, the microcontroller controlled the opening and closing of the solenoid valve, allowing water to flow to the crops until the measured soil moisture content was greater than or equal to the calculated crop water requirements. Once this was established, the microcontroller would then wait for a given duration, say 15 minutes, before repeating the control algorithm. Thus, as long as the microcontroller remained powered, the smart irrigation system ensured that the crops received an optimum level of water supply.

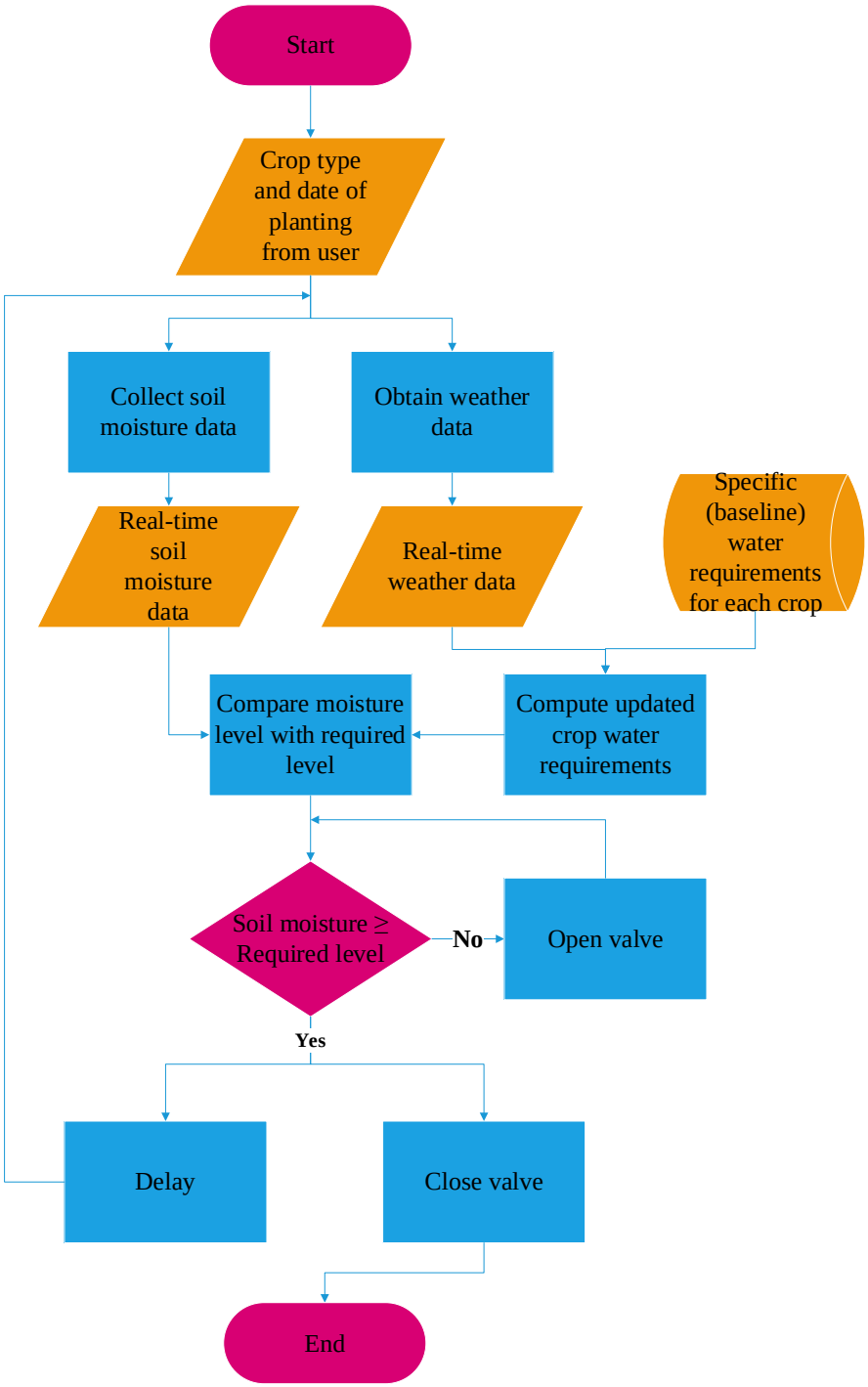

Figure 3.2: Project flow chart

3.4 Hardware and Software Design

This section explains in comprehensive detail, the methodology that was used to achieve each of the specific objectives of the project.

3.4.1 Methodology for realization of specific objective i: Control algorithm and hardware implementation

The realization of the first specific objective involved the development of a control algorithm, implemented though hardware, to supply an optimal amount of water to the planted crops.

a. Control algorithm

As earlier mentioned, the specific crop water requirements of a crop are affected by the following variables:

- The type of crop, e.g., maize, crop, or cabbage
- The stage of growth of the crop, e.g., initial stage, crop development stage, midseason stage, or late season stage
- The prevailing weather conditions, e.g., temperature, cloud cover, relative humidity, and wind speed

For this reason, it was necessary to develop a control algorithm that could take in the crop type and age, determine its stage of growth, obtain the real time weather conditions, and thus use all this information to calculate the specific crop water requirements of the crop at the particular stage of growth. The algorithm would then need to compare the measured level of soil moisture in the farm with the calculated/required value, and then control the supply of water to the crop to attain an optimum level of irrigation. This control algorithm, as established in the flow chart in *Figure 3.2*, was executed through a program in the ESP32 microcontroller, using the code sections and

functions explained the in following steps. The development of the program employed the best practices of good programming, such as modularity by using custom methods, and incorporating error handling capabilities.

The elements of the program that actualized the algorithm are explained in the subsections below.

i. Choice of programming language and integrated development environment (IDE)

The program was written and compiled using the Arduino Integrated Development Environment (IDE). Arduino IDE employs C/C++ for programming the ESP32 microcontroller, leveraging the language's simplicity and wide usage. Therefore, in this project, the Arduino language simplified tasks crucial for microcontroller programming, such as managing GPIO pins and reading analog inputs. This approach fostered a supportive community, providing abundant resources and libraries. The use of C/C++ ensured cross-platform compatibility, resource efficiency, and fine-grained hardware control, allowing for optimal performance and flexibility. Overall, the combination of Arduino and C/C++ for the program development struck a balance between ease of use for beginners and versatility for advanced developers in the realm of microcontroller programming.

ii. Initializing the program

This step comprised including the necessary libraries, initializing user defined variables as well as assigning the GPIO pins of the ESP32 microcontroller, and creating relevant objects that would be used in the program, or by the hardware modules in the circuit. This is as explained in the code snippet below. The single line comments (//) and multiline comments (/* …*/) have been used to annotate and explain the purpose of the code.

```
//Libraries to be used in the program
```

```
#include <Wire.h> //Wire library for communicating with I2C devices
#include <LiquidCrystal_I2C.h> //Library for the I2C LCD module
#include <Keypad.h> //Library for the 4x4 membrane keypad module
#include <RTClib.h> //Library for the real time clock (RTC) module
#include <WiFi.h> //Library for the Wi-Fi communication function of ESP32
#include <ESPAsyncWebServer.h> //Library for the ESP32 server for communication

#define relayPin 26 // ESP32 pin GPIO26, which connects to the DC solenoid valve
via the relay

//Defining capacitive soil moisture sensor pins in the ESP32 microcontroller
#define moistureSensorPin1 13 //Pin 13 connected to AOUT pin of sensor 1
#define moistureSensorPin2 14 //Pin 14 connected to AOUT pin of sensor 2
#define moistureSensorPin3 27 //Pin 27 connected to AOUT pin of sensor 3

RTC_DS3231 rtc; // Creatin/defining the RTC object
LiquidCrystal_I2C lcd(0x27, 16, 2); // I2C address and LCD size (16x02)

//Keypad variables
const byte ROWS = 4; //four rows of the keypad
const byte COLS = 4; //four columns of the keypad
char keys[ROWS][COLS] = {//mapping the 4x4 keypad matrix to an array in the
program
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
byte rowPins[ROWS] = {19, 18, 5, 17}; //connect to the row pinouts of the keypad
byte colPins[COLS] = {16, 4, 23, 25}; //connect to the column pinouts of the
keypad
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

//RTC Variables
int SDA_2 = 33; //Pin 33 of ESP32 used as a serial data pin for the RTC
int SCL_2 = 32; //Pin 32 of ESP32 used as a serial clock pin for the RTC
int freq = 100000; //Clock frequency of the RTC module
```

iii. Preliminary functions in the program

These are the custom functions that were called in the void setup upon powering the system. They were not essential components of the algorithm, yet they served an important purpose of enhancing

the interaction between the user and the system. These functions are explained briefly in the code snippets below.

```
void setup() {
  //Initialize analogue pins 13, 14, 27 as inputs, and digital pin 26 as an
output
  pinMode(moistureSensorPin1, INPUT);
  pinMode(moistureSensorPin2, INPUT);
  pinMode(moistureSensorPin3, INPUT);
  pinMode(relayPin, OUTPUT);

  lcd.init(); //Setting up the LCD display module
  lcd.backlight(); //Switching ON the LCD display backlight

  welcomeMessage();
  configureRTC();
  displayCurrentDateTime();

}

//Displaying a message on the LCD to welcome the user to the project:
void welcomeMessage() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Welcome to");
  lcd.setCursor(0, 1);
  lcd.print("Tony's Project!");
  delay(5000); // Display the message for 5 seconds
}

//Configuring and setting the time on the real time clock module
void configureRTC(){
  // SETUP RTC MODULE
  Wire1.begin(SDA_2, SCL_2, freq);
  if (! rtc.begin(&Wire1)) {
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("RTC NOT found");
    delay(200);
    lcd.clear();
    while (1);
  }
```

```
  // automatically sets the RTC to the date & time on PC this sketch was
compiled
  rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));

}

//Displaying the current date and time on the LCD from the real time clock
module
void displayCurrentDateTime() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Date & Time:");
  DateTime now = rtc.now();
  lcd.setCursor(0, 1);
  lcd.print(now.timestamp(DateTime::TIMESTAMP_FULL));
  delay(5000); // Display the date and time for 5 seconds
}

//Informing the user that the crop name and age have been entered
successfully
void configSuccessful (){
  lcd.clear();
  lcd.print("Setup complete!");
  lcd.setCursor(0, 1);
  lcd.print(selectedCrop == 1 ? "Maize" : (selectedCrop == 2 ? "Kales" :
"Cabbage"));
  lcd.setCursor(8, 1);
  lcd.print(selectedAge);
  lcd.setCursor(11, 1);
  lcd.print("weeks");
  delay(3000); //Display for 3 seconds
}
```

All the functions described above are only executed once at the beginning of the program.

iv. Obtaining the crop type (Maize, Kales, or Cabbage) from the user

This was achieved by first initializing the crop type as an integer variable in the program. The integer value would then be set from the keypad according the type of crop as: 1 to represent maize, 2 to represent kales, and 3 to represent cabbage.

```
int selectedCrop = -1; // Initialize selectedCrop with an invalid value
```

The variable was initialized with an invalid value to ensure that no crop type would be selected accidentally, or erroneously. In case the user failed to enter anything on the crop type, the program would display an invalid value, which is an important error handling feature and good programming practice.

Next, the crop type was obtained by calling the custom function `selectCropType()` in the `void setup()` part of the program. The function was used to prompt the user (through the LCD) to select the crop that they had planted by pressing the corresponding value on the keypad. This function would only be executed once, since the crop type only needed to be entered in the program once. The function that was used for collecting crop type is as explained in the code snippets below.

```
void setup() {

//User input functions
  selectCropType(); //Function called within the void setup

}

//User input function
void selectCropType() {
//Displaying the prompt on the LCD: Please enter crop type:
// 1. Maize 2. Kales 3. Cabbage
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Please enter");
  lcd.setCursor(0, 1);
  lcd.print("crop type:");
  delay(3000);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("1.Maize, 2.Kales");
  lcd.setCursor(0, 1);
  lcd.print("3.Cabbage");
```

Additionally, if the user entered an invalid value the function was set to display that an invalid value had been entered, as explained in the code snippet below.

```
// If the key pressed was not 1, 2, or 3, an invalid message would be displayed
on the LCD for 1 second
  while (selectedCrop == -1) {
    char key = keypad.getKey();
    if (key) {
      if (key >= '1' && key <= '3') {
        selectedCrop = key - '0';
      } else {
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Invalid choice!");
        delay(1000); // Displaying the invalid message for 1 second

// Prompting the user again to enter the correct crop type
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("1.Maize, 2.Kales");
        lcd.setCursor(0, 1);
        lcd.print("3.Cabbage");
      }
    }
  }
```

Once the user entered a valid crop type, the name of the crop was then displayed on the LCD, indicating that the crop type has been selected successfully, as explained in the code snippet below.

```
// Displaying the selected crop name on the LCD
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Selected Crop:");
  lcd.setCursor(0, 1);
  lcd.print(selectedCrop == 1 ? "Maize" : (selectedCrop == 2 ? "Kales" :
"Cabbage")); // Ternary operator that checks the value of the integer
variable selectedCrop, and then prints the corresponding name of the LCD
  delay(3000); // Displaying the crop name on the LCD for 3 seconds

}
```

v. Obtaining the age of the crop (in weeks) from the user

This was achieved by first initializing the crop age as an integer variable in the program. The integer value would then be set by entering the age of the crop in weeks from the keypad.

```
int selectedAge = -1; // Initialize selectedAge with an invalid value
```

Similarly, as part of error handling, the variable was initialized with an invalid value to ensure that no crop age would be selected accidentally, or erroneously. In case the user failed to enter anything on the crop age, the program would display an invalid value.


```
int selectedAge = -1; // Initialize selectedAge with an invalid value
```


Next, the crop age was obtained by calling the custom function `selectCropAge()` in the `void setup()` part of the program. The function was used to prompt the user (through the LCD) to enter the age crop that they had planted by pressing numeric key on the keypad. Similarly, this function would only be executed once, since the crop type only needed to be entered in the program once. The function that was used for collecting crop age data is as explained in the code snippets below.

```
void setup() {

//User input functions
  selectCropAge(); //Function called within the void setup

}

//User input function
void selectCropAge() {
// Displaying the prompt on the LCD: Enter crop age: (0 - 20 weeks)
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Enter crop age:");
  lcd.setCursor(0, 1);
  lcd.print("0-20 weeks");
  delay(2500); // Displaying the prompt for 2.5 seconds

// Displaying the prompt on the LCD: Press C when done
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Press 'C'");
```

```
  lcd.setCursor(0, 1);
  lcd.print("when done");
  delay(2500); // Displaying the prompt for 2.5 seconds

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Crop age (weeks):");
  lcd.setCursor(0, 1);
```

Due to the possibility of the user entering double digit values for crop age, that is, for crop ages between 10 and 20 weeks, the function incorporated lines of code to enable this functionality. This was established using integer variables with the tens and ones positions, which would then be summed to obtain the crop age. This functionality is as as explained in the code snippets below.

```
// Initializing local integer and Boolean variable to be used within the
function
  int cropAge = 0;
  bool ageEntered = false;

  while (!ageEntered) {
    char key = keypad.getKey();
    if (key >= '0' && key <= '9') {
      lcd.print(key); // Displaying the key entered from the keypad to the
LCD
      cropAge = cropAge * 10 + (key - '0');
      if (cropAge > 20) {
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Invalid age!      ");
        lcd.setCursor(0, 1);
        lcd.print("0-20 weeks");
        delay(2000);
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Crop age (weeks):");
        lcd.setCursor(0, 1);
        cropAge = 0;
      }
```

Once the user had entered a valid crop age and pressed character ‘C’ on the keypad to indicate completion, the age of the crop was then displayed on the LCD, indicating that the crop age has been entered successfully, as explained in the code snippet below.

```
// Displaying the entered crop age on the LCD
} else if (key == 'C') {
      if (cropAge >= 0 && cropAge <= 20) {
        selectedAge = cropAge;
        lcd.clear();
        lcd.print("Crop age success!");
        lcd.setCursor(0, 1);
        lcd.print(selectedAge);
        lcd.setCursor(3, 1);
        lcd.print("weeks");
        delay(3500); // // Displaying the crop age on the LCD for 3 seconds
        ageEntered = true;
      }
```

The range of crop age variable was limited to 0 to 20 weeks since each of the three test crops (Maize, Kales and Cabbage) typically reach maturity and harvesting within this period of time. Therefore, the program’s error handling capability would prevent the user from entering a value that was outside the allowable range. If the user entered an invalid value the function was set to display that an invalid value had been entered, as explained in the code snippet below.

```
      else {
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Invalid age!      ");
        lcd.setCursor(0, 1);
        lcd.print("0-20 weeks");
        delay(2000);
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Crop age (weeks):");
        lcd.setCursor(0, 1);
        cropAge = 0;
      }
```

```
    }
  }
}
```

vi. Obtaining real-time weather data

As earlier mentioned, the specific crop water requirements are affected by variations in weather parameters, namely temperature, cloud cover, relative humidity, wind speed, and the proportion of annual daytime hours [25]. These factors were used to calculate the crop water requirements.

The weather parameters were first initialized as floating-point variables with baseline values based on meteorological information on the weather elements of Juja Town, Kiambu County, Kenya [25] [26], which was the initial test location for the project. This is as shown in the code snippet below.

```
// Weather parameters
float temperature = 20.000;      // in °C
float cloudCover = 50.000;       // in %
float relativeHumidity = 75.000;  // in %
float windSpeed = 10.000;         // in km/h
float sunshineDuration = 0.275;
```

The subsequent real-time values of local weather were then sent to the microcontroller and updated from the mobile application via Wi-Fi communication. The mobile application contained a weather application programming interface (API) that periodically updated the weather information in the microcontroller after every 5 seconds, allowing for near real-time weather updates. To achieve this functionality, the microcontroller code contained a function that enabled the ESP32 to connect to a specified Wi-Fi network by passing the previously initialized network credentials to the inbuilt Wi-Fi functionality of the microcontroller. The Wi-Fi connection was established by calling the custom function `connectWiFi()` in the `void setup()` part of the program. This function would only be executed once, since the Wi-Fi connection only needed to established once in the program,

upon powering the circuit. In case the connection was temporarily lost and then recovered, the microcontroller would automatically reconnect to the Wi-Fi network. This function is as explained in the code snippet below.

```
//Wi-Fi network credentials
const char* ssid = "TECNO CAMON 19 Pro";
const char* password = "Tony@2023";

void setup() {
  //Wi-Fi function called in the void setup
  connectWiFi();
}

void connectWiFi(){
  WiFi.mode(WIFI_STA); //Optional - ESP32 was used in station mode (default mode)
  WiFi.begin(ssid, password); //Wi-Fi connection set up with network credentials

//Displaying the message on the LCD: Connecting to WiFi
    while(WiFi.status() != WL_CONNECTED){
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Connecting to");
      lcd.setCursor(0, 1);
      lcd.print("WiFi ...");
      delay(500);
    }
//Displaying the message on the LCD: WiFi Connected
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("WiFi Connected");
    lcd.setCursor(0, 1);
    lcd.print(ssid);
    delay(4000);
//Displaying the local IP Address of the ESP on the LCD after connecting. This
would be useful in connecting to the ESP32 server from the mobile app
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("ESP32 Local IP");
    lcd.setCursor(0, 1);
    lcd.print(WiFi.localIP());
    delay(5000);

}
```

To establish communication between the ESP32 and the mobile application and allow for the updating of the weather data, the program created an asynchronous web server on the ESP32 and established an endpoint for updating the weather data. Once the mobile application (client) called the endpoint on the ESP32 (server) with the updated weather data from the API, a connection was established, and the weather parameters were automatically updated. The code snippets for this functionality are as explained below.

```
AsyncWebServer server(80); Asynchronous ESP32 web server using port 80 for HTTP
communication

//Sever endpoint for mobile app communication

  // Route to update weather parameters
  server.on("/receiveSendData", HTTP_GET, [](AsyncWebServerRequest *request) {
    temperature = request->arg("temperature").toFloat();
    cloudCover = request->arg("cloudCover").toFloat();
    relativeHumidity = request->arg("relativeHumidity").toFloat();
    windSpeed = request->arg("windSpeed").toFloat();
  });

// Start server
  server.begin();
```

vii. Calculating the specific crop water requirements of the selected crop

The specific crop water requirements of the planted crop were determined using the custom function `calculateCropWaterRequirements()` in the `void loop()` part of the program. The function was used to calculate the crop water requirements by considering the type and age of crop, its stage of growth, and the real time weather parameters. Since the weather parameters sent from the mobile app to the microcontroller varied in real time, the function needed to calculate the updated crop water requirements in real time; hence it was called in the `void loop()` so as to run

repeatedly as long as the system was powered. The operation of the function was as explained in the code snippets below.

```
void loop() {
  // Function called from the void loop to run repeatedly
  calculateCropWaterRequirements();

}
```

According to the Food and Agriculture Organization (FAO) [25], most crops typically go through four major stages of growth, that is, the initial stage, crop development stage, midseason stage, or late season stage. However, the duration that a crop take in each of the stages varies for different types of crops. For instance, maize at 9 weeks and kales at 9 weeks would be in different stages of growth, hence their water requirements would be considerably different. The durations of the different growth stages for maize, kales, and cabbage are as shown in *Table 3.5*.

Table 3.5:Growth stages of maize, kales, and cabbage

Crop type	**Crop age (weeks)**			
Growth stage	***Initial stage***	***Crop development stage***	***Midseason stage***	***Late season stage***
Maize	Week 0 – Week 3 (3 weeks)	Week 4 – Week 9 (6 weeks)	Week 10 – Week 16 (7 weeks)	Week 17 – Week 21 (5 weeks)
Kales	Week 0 – Week 3 (3 weeks)	Week 4 – Week 8 (5 weeks)	Week 9 – Week 11 (3 weeks)	Week 12 – Week 13 (2 weeks)
Cabbage	Week 0 – Week 3 (3 weeks)	Week 4 – Week 7 (4 weeks)	Week 8 - Week 16 (4 weeks)	Week 18 - Week 19 (3 weeks)

The information in *Table 3.5* was used determine and assign the growth stage of the crop based on the crop type and crop age. The `calculateCropWaterRequirements()` function first checked to

see the particular crop type and the age of the crop that the user had entered. (Recall that the crop type and crop age were entered by the user on through the keypad, using custom functions `selectCropType()` and `selectCropAge()`. These functions then set the values of the variables `int selectedCrop` and `int selectedAge`, to the respective values of crop type and crop age.) Using the values of `selectedCrop` and `selectedAge`, the `calculateCropWaterRequirements()` function determined the growth stage and set it an integer value of 1, 2, 3, or 4, using an *if else… else if… else statement* with nested ternary operators as explained in the code snippet below.

```
float calculateCropWaterRequirements() {
  // Determine the growth stage based on the selectedCrop and selectedAge
  int growthStage;
  if (selectedCrop == 1) {  // Maize
      //Nested ternary operators to set growth stage if the crop type is
maize
    growthStage = (selectedAge >= 0 && selectedAge <= 3) ? 1 :
                  (selectedAge >= 4 && selectedAge <= 9) ? 2 :
                  (selectedAge >= 10 && selectedAge <= 16) ? 3 : 4;
  } else if (selectedCrop == 2) {  // Kales
      //Nested ternary operators to set growth stage if the crop type is
kales
    growthStage = (selectedAge >= 0 && selectedAge <= 3) ? 1 :
                  (selectedAge >= 4 && selectedAge <= 8) ? 2 :
                  (selectedAge >= 9 && selectedAge <= 11) ? 3 : 4;
  } else if (selectedCrop == 3) {  // Cabbage
      //Nested ternary operators to set growth stage if the crop type is
cabbage
    growthStage = (selectedAge >= 0 && selectedAge <= 3) ? 1 :
                  (selectedAge >= 4 && selectedAge <= 7) ? 2 :
                  (selectedAge >= 8 && selectedAge <= 16) ? 3 : 4;
  }
```

At this point, both the type of crop, and the stage of growth that the crop is in, had been established. Additionally, the weather information had already been updated from the mobile application. With this information, the specific water requirements of the crop were calculated using the formula in *Equation 1*.

$$ETcrop = ETo * Kc$$ …… *Equation 1*

where *[25]:*

- ET_{crop} is the specific water requirement of the crop in mm/day
- ET_O is the evapotranspiration rate in mm/day
- K_C is the crop factor (no units)

ET_O is dependent on weather parameters, specifically temperature (T), and daily proportion of annual daytime hours (p). Eto is calculated using the Blaney Criddle formula [25] as in *Equation 2*.

$$ETO = p\,(0.6T + 8)$$ …… *Equation 2*

where [25]:

- T = temperature
- p = proportion of annual daytime hours

K_C is dependent on the growth stage of the crop (initial stage, crop development stage, midseason stage, or late season stage), and weather parameters (cloud cover, relative humidity, and wind speed).

According to the Food and Agriculture Organization (FAO) [25], the values of K_C for maize, kales, and cabbage in each of the four stages of growth as shown in *Table 3.6*.

Table 3.6: Crop factor K_C values for maize, kales

Crop	**Initial/baseline value of Kc**			
	Initial stage	***Crop development stage***	***Midseason stage***	***Late season stage***
Maize	0.40	0.80	1.15	0.70
Kales	0.45	0.60	1.00	0.90
Cabbage	0.45	0.75	1.05	0.90

The values of K_C are provided by FAO [25] at baseline values of weather parameters. Based on this information, the baseline values of K_C were initialized as floating-point variables in the `calculateCropWaterRequirements()` function using an *if else… else if… else statement* with nested ternary operators as explained in the code snippet below.

```
// Initializing baseline Kc based on crop and growth stage
  float baselineKc = 0.000;
  if (selectedCrop == 1) {  // Maize
      //Nested ternary operators to set baselineKc if the crop type is
maize
    baselineKc = (growthStage == 1) ? 0.40 : ((growthStage == 2) ? 0.80 :
((growthStage == 3) ? 1.15 : 0.70));
  } else if (selectedCrop == 2) {  // Kales
      //Nested ternary operators to set baselineKc if the crop type is
kales
    baselineKc = (growthStage == 1) ? 0.45 : ((growthStage == 2) ? 0.60 :
((growthStage == 3) ? 1.00 : 0.90));
  } else if (selectedCrop == 3) {  // Cabbage
      //Nested ternary operators to set baselineKc if the crop type is
cabbage

    baselineKc = (growthStage == 1) ? 0.45 : ((growthStage == 2) ? 0.75 :
((growthStage == 3) ? 1.05 : 0.90));
  }
```

Further, according to FAO [25], the value of K_C should then be modified to the final value based on the values of the weather parameters (cloudCover, relativeHumidity, and windSpeed), using the following conditions:

- Cloud Cover (CC): If the cloud cover is high (CC > 80%), Kc should be reduced by 0.05; else if the cloud cover is low (CC < 50%), Kc should be increased by 0.05; otherwise, if cloud cover is average (between 50% and 80%), Kc should remain as it was in the initial/baseline value.
- Relative Humidity (RH): If the relative humidity is high (RH > 80%), Kc should be reduced by 0.05; else if the relative humidity is low (RH < 50%), Kc should be increased by 0.05; otherwise, if RH is average (between 50% and 80%), Kc should remain as it was in the initial/baseline value.
- Wind Speed (u): If the windspeed is low (u < 7.2 km/h), Kc should be reduced by 0.05; else if the wind speed is high (u > 18km/h), Kc should be increased by 0.05; otherwise, if the wind speed is average (between 7.2km/h and 18 km/h), Kc should remain as it was in the initial/baseline value.

Using this information, the conditions above were used to create *if ... else if statements* for cloud cover, relative humidity, and windspeed, within the `calculateCropWaterRequirements()` function. The final value of the crop factor, K_C was then calculated from these statements in the function as shown the code snippets below.

```
// Modifying Kc to final value based on weather parameters
  if (cloudCover > 80.000) {
    baselineKc -= 0.050;
  } else if (cloudCover < 50.000) {
    baselineKc += 0.050;
  }

  if (relativeHumidity > 80.000) {
```

```
    baselineKc -= 0.05;
  } else if (relativeHumidity < 50.000) {
    baselineKc += 0.050;
  }

  if (windSpeed < 7.200) {
    baselineKc -= 0.050;
  } else if (windSpeed > 18.000) {
    baselineKc += 0.050;
  }
```

Finally, the values of ET_O and ET_{crop} were calculated in the function using *Equations 1* and *2* as shown in the code snippet below.

```
// Calculate Eto
  float eto = sunshineDuration * (0.460 * temperature + 8.000);

 // Calculate ETcrop
  float etcrop = eto * baselineKc; // The value of Kc here is the final value
```

Note that the value of ET_{crop} that was obtained here was the amount of water in mm that the crop requires. However, not all of this water is available for the crop to absorb. The percentage of available water varies based on the type of soil. Since the effect of different types of soil on water availability levels was not within the scope of this project, a fixed availability level of 30% was assumed. Selecting a low percentage of water availability is a safer choice since it ensures that more water will need to be supplied to the crop, thus preventing it from the risk of water stress and wilting. From this, the value of the required soil moisture level was calculated and displayed on the LCD as shown in the code snippet below.

```
 requiredSoilMoisture = 3.333 * etcrop;

//Display required soil moisture level
  lcd.clear();
  lcd.print("Moisture need");
  lcd.setCursor(0, 1);
  lcd.print(requiredSoilMoisture);
```

```
delay(3000);
```

Additionally, in order to check and confirm that the `calculateCropWaterRequirements()` function was performing as intended, the weather parameters and the calculated values of ET_O, K_C and ET_{crop} were displayed on the LCD. The code for this display purpose was quite lengthy and repetitive, and was solely for function testing, and not part of the operational purpose of the function in the project. Thus, the code is listed in *Appendix A* of this report.

Finally, the `calculateCropWaterRequirements()` function returned the value of the specific crop water requirement (ET_{crop}) as shown in the line of code below.

```
return etcrop; // Return calculated value of ETcrop
```

As a more efficient alternative, the baseline values of ET_O, K_C and ET_{crop} were also sent from the database to the mobile app and the microcontroller, resulting in faster execution, along with the benefits of using the database to easily modify the parameters as opposed to hardcoding them. However, in this case, there was also an increased redundancy between the values of ET_O, K_C and ET_{crop} obtained from the database, and those calculated from the `calculateCropWaterRequirements()` function, which improved the fault tolerance of the entire system in case of loss of connection. The role of the database is covered in more comprehensive detail in *Subsection 3.4.2* of this report.

viii. Measuring the real time soil moisture

The real-time soil moisture levels were determined using the custom function `readSoilMoisture()` in the `void loop()` part of the program. The function was used to obtain analogue values of soil moisture using the soil moisture sensors. Since the soil moisture kept on

changing with time, the function was called in the `void loop()` so as to run repeatedly as long as the system was powered. The operation of the function was as explained in the code snippets below.

```
void loop() {

  // Function called from the void loop to run repeatedly
  readSoilMoisture();

}

float readSoilMoisture() {
  // Read soil moisture values from each sensor
  int soilMoisture1 = analogRead(moistureSensorPin1);
  int soilMoisture2 = analogRead(moistureSensorPin2);
  int soilMoisture3 = analogRead(moistureSensorPin3);

  // Calculate the average soil moisture content
  float averageSoilMoisture = (soilMoisture1 + soilMoisture2 + soilMoisture3) /
3.000;

  // Map the soil moisture value to a scale of 0 to 100
  float mappedSoilMoisture = map(averageSoilMoisture, 3050.000, 1035.000, 0.000,
30.000);

  // Print the mapped value to the LCD for testing
  lcd.clear();
  lcd.setCursor(0, 0); // Set cursor to the first row
  lcd.print("Soil Moisture:");
  lcd.setCursor(0, 1); // Set cursor to the second row
  lcd.print(mappedSoilMoisture, 2); //2 decimal places
  lcd.setCursor(5, 1);
  lcd.print("mm");
  delay(5000);

  return mappedSoilMoisture; // Return the mapped soil moisture value
}
```

ix. Controlling the solenoid valve

The actuating of the valve to turn it ON or OFF was accomplished using the custom function `controlValve()` in the `void loop()` part of the program. The function was used to compare the measured value of soil moisture from the soil moisture sensors with the required level of soil moisture. (Recall that the variables `requiredSoilMoisture` and `measuredSoilMoisture` were updated by the `calculateCropWaterRequirements()` and `readSoilMoisture()` functions respectively.) Since the comparison of these levels, and the corresponding actuation of the valve, needed to be done in real time, the `controlValve()` function was called in the `void loop()` so as to run repeatedly as long as the system was powered. The operation of the function was as explained in the code snippets below.

```
void loop() {

  // Function called from the void loop to run repeatedly
  controlValve();

}
```

If the measured value of soil moisture was found to be lower than the required level, it implied that the soil was drier than required, hence the valve needed to be opened to allow water to flow to the crops. This was achieved by setting the relay pin of the microcontroller HIGH, which energized the solenoid valve's coil and thus opened the valve. The state of the relay after the actuation was also displayed on the LCD and updated to be read by the mobile application. This functionality was as shown in the code snippet below.

```
void controlValve() {

    // Compare water requirements and measured soil moisture
  if (measuredSoilMoisture < requiredSoilMoisture) {
    // Turn ON the valve
    lcd.clear();
```

```
    lcd.print("Valve Status:");
    lcd.setCursor(0, 1);
    lcd.print("ON");
    digitalWrite(relayPin, HIGH); //  HIGH turns ON the valve
    relayState = HIGH; // Dynamically update relay state to the app
  }
```

On the other hand, if the measured value of soil moisture was found to be higher than or equal to the required level, it implied that the soil was just at the right wetness level for the crop or wetter than required, hence the valve needed to be closed to prevent water from flowing to the crops. This was achieved by setting the relay pin of the microcontroller LOW, which de-energized the solenoid valve's coil and thus closed the valve. The state of the relay after the actuation was also displayed on the LCD for 10 seconds and updated to be read by the mobile application. This functionality was as shown in the code snippet below.

```
else {
    // Turn OFF the valve
    lcd.clear();
    lcd.print("Valve Status:");
    lcd.setCursor(0, 1);
    lcd.print("OFF");
    digitalWrite(relayPin, LOW); //  LOW turns OFF the valve
    relayState = LOW; // Dynamically update relay state to the app
  }
  delay(10000); // Display valve status for 10 seconds

}
```

The valve control algorithm ensured that the state of the valve changed based on the real-time and dynamic changes in the required versus measured values of soil moisture.

b. Hardware implementation

The hardware circuit that was used to implement the project's functionalities was implemented as follows:

i. Selection/choice of components

The circuit comprised the following components:

- *ESP32 Microcontroller*: The ESP32 was chosen for its dual-core processing, Wi-Fi capabilities, and GPIO pins, enabling seamless integration with the IoT system. Its low power consumption suits battery-operated setups.
- *12V DC Solenoid Valve:* The 12V valve aligned with common irrigation systems, ensuring compatibility and ease of integration. Its robust build and power efficiency made it suitable for agricultural applications.
- *4X4 Membrane Keypad:* The keypad provided a user-friendly interface for input. Its simplicity, cost-effectiveness, and durability made it a suitable choice for manual control input.
- *Capacitive Soil Moisture Sensors:* Chosen for their sensitivity to soil moisture levels, capacitive sensors offered accurate readings. They were more resilient in varied soil conditions and had a longer lifespan compared to alternatives like resistive sensors which corroded easily.
- *I2C LCD Display Module:* The I2C LCD streamlined wiring and conserved GPIO pins, crucial in a compact setup. Its compatibility with the ESP32 and ease of programming made it a practical choice for displaying real-time data.
- *5V Relay Module (1 Channel):* This relay interfaced seamlessly with the ESP32 and allowed control of higher voltage devices, like the solenoid valve. Its simplicity, reliability, and ease of use in interfacing with microcontrollers made it an ideal choice.
- *Real-Time Clock Module:* The RTC module provided accurate timekeeping, crucial for irrigation events. Its low power consumption and ability to operate independently of the

microcontroller enhanced system reliability. It also helped to increment the age of the crop as time elapsed.

- *3.7V Lithium-Ion Batteries:* These batteries offered a balance between voltage requirements and portability. Their rechargeable nature and energy density made them suitable for powering the system efficiently while allowing for a compact design.

ii. ESP32 pin assignments to components

The 30-pin version of the ESP32 Dev Kit Module, whose pinout is shown in *Figure 3.3*, was used in this project.

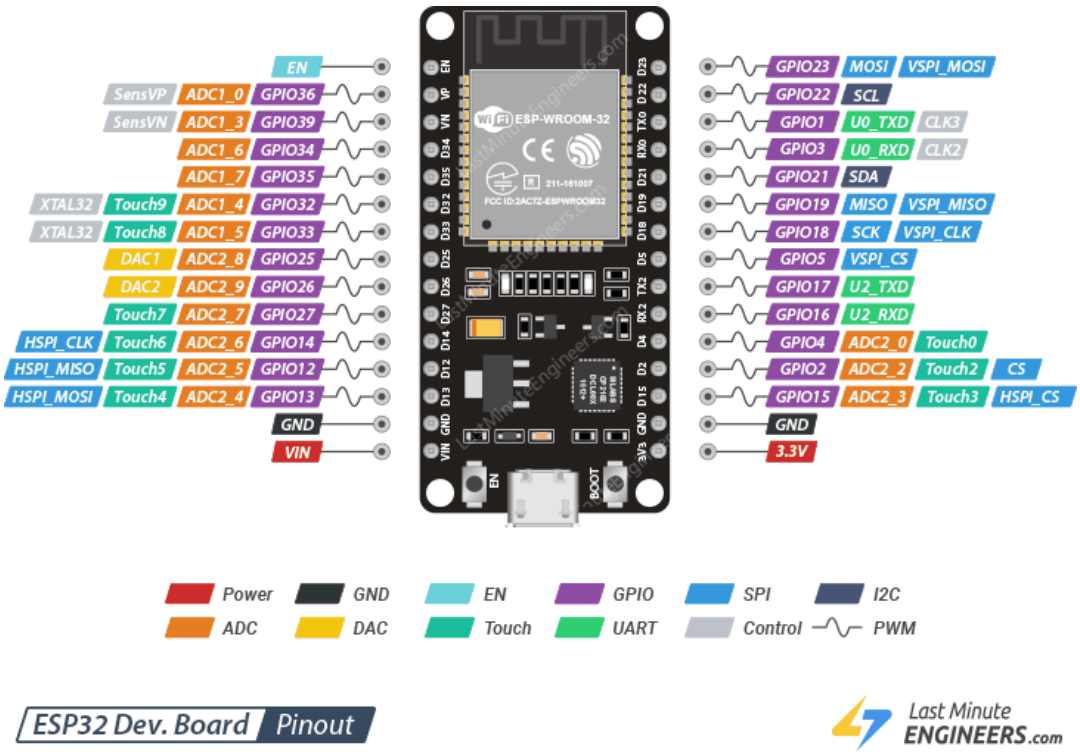

Figure 3.3: ESP32 Dev Kit Module Pinout

The pins of the ESP32 microcontroller were assigned to the programmable components of the hardware circuit that were connected to the microcontroller as shown in *Table 3.7*.

Table 3.7: ESP32 pin assignment to components

Component	***Component pins***	***ESP32 Pins***
LCD 12C module	SDA	GPIO 21
	SCL	GPIO 22
Keypad	R1	D19
	R2	D18
	R3	D5
	R4	D17
	C1	D16
	C2	D4
	C3	D23
	C4	D25
Capacitive soil moisture sensors	AOUT (Sensor 1)	D13
	AOUT (Sensor 2)	D14
	AOUT (Sensor 3)	D27
Relay module	Signal pin	D26
RTC module	SDA	GPIO 33
	SCL	GPIO 32

iii. Power circuit/schematic diagram

The power circuit of the project was designed in *NI Multisim* software as shown in *Figure 3.4*. This was used in conjunction with the pin assignments specified in *Table 3.7* to create the schematic diagram from which the assembled circuit was implemented.

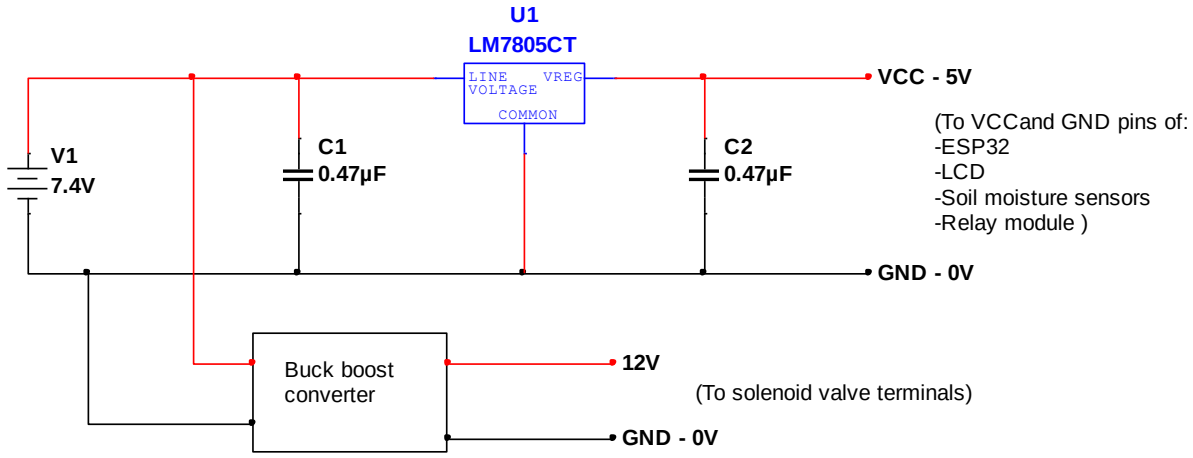

Figure 3.4: Power circuit

iv. Assembled circuit

The components listed above were assembled into the hardware circuit in *Figure 3.5*.

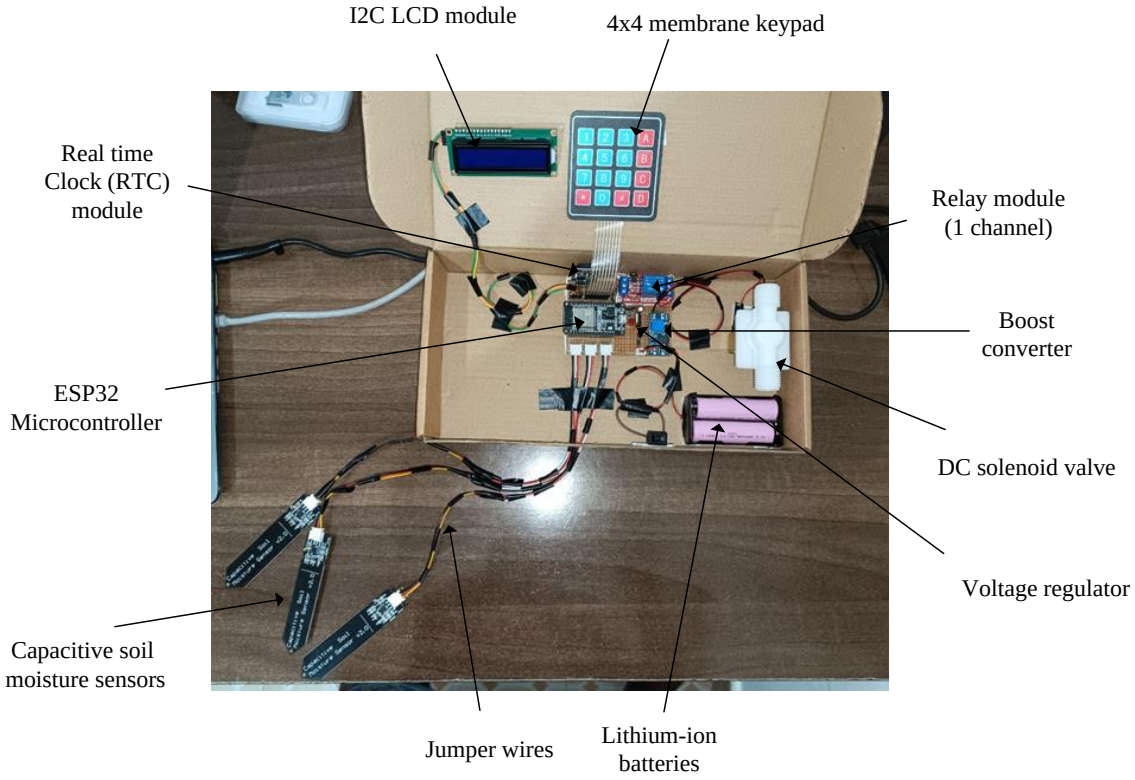

Figure 3.5 Assembled circuit

3.4.2 Methodology for realization of specific objective ii: Database development

i. Description of the intended data in the database and its role

Prior to developing the database, it was first necessary to establish the role of the database and the nature of data that was to be stored in it. This informed the selection of the database type and development platform. The role of the database in the project was to store the baseline crop data, that is, as the evapotranspiration rates and crop factors that affect the specific crop water requirements. As earlier mentioned, these requirements are dependent on the type of crop, the age or growth stage of the crop, and weather conditions. Additionally, recall that the three different test crops (maize, kales, and cabbage) take different durations in the different growth stages, that is, the initial stage, crop development stage, midseason stage, or late season stage (tabulated in *Table*

3.8), resulting in varying crop water requirements. Therefore, the database would need to store the names of the different growth stages of each of the three crop types, and the corresponding ages of the crops (in weeks) in each growth stage, for each crop type.

Table 3.8: Growth stages of maize, kales, and cabbage [25]

Crop type	**Crop age (weeks)**			
Growth stage	***Initial stage***	***Crop development stage***	***Midseason stage***	***Late season stage***
Maize	Week 0 – Week 3 (3 weeks)	Week 4 – Week 9 (6 weeks)	Week 10 – Week 16 (7 weeks)	Week 17 – Week 21 (5 weeks)
Kales	Week 0 – Week 3 (3 weeks)	Week 4 – Week 8 (5 weeks)	Week 9 – Week 11 (3 weeks)	Week 12 – Week 13 (2 weeks)
Cabbage	Week 0 – Week 3 (3 weeks)	Week 4 – Week 7 (4 weeks)	Week 8 - Week 16 (4 weeks)	Week 18 - Week 19 (3 weeks)

Further, the crop factors (the specific water requirement of the crop in mm/day - ET_{crop}; the evapotranspiration rate in mm/day - ET_O; and the crop factor - K_C) were different for each of the crop types in the different stages of growth. These values also needed to be stored in the database. *Table 3.9* shows the baseline K_C values of the different crop types in their various stages of growth.

Table 3.9: Crop factor K_C values for maize, kales

Crop	**Initial/baseline value of Kc**			
	Initial stage	***Crop development stage***	***Midseason stage***	***Late season stage***
Maize	0.40	0.80	1.15	0.70
Kales	0.45	0.60	1.00	0.90
Cabbage	0.45	0.75	1.05	0.90

Since ET_O and K_C were also affected by weather changes, their baseline values were evaluated at the baseline weather conditions for Juja town, shown in the *Table 3.10*. These baseline values of ET_O, K_C and ET_{crop} also needed to be stored in the database. Their values were static, since they were evaluated at fixed, known weather parameters.

Table 3.10: Weather parameters for evaluating baseline values of ET_O, K_C and ET_{crop}

Baseline weather parameters:			**Notes**
Parameter	***Value***	***Units***	These are the baseline/static values to be used if the microcontroller cannot connect to the internet. The static values are calculated and set for a farm located in Juja, based on averaged weather data for the area. This offline mode is hardcoded in the microcontroller, and serves as a backup in case of lost internet connection.
temperature	20	°C	
cloudCover	50%	-	
relativeHumidity	75%	-	
windSpeed	10	km/h	
Sunshine duration, p	0.275	-	

Based on the descriptions above, the comprehensive data that needed to be stored in the database for each of the three crops was evaluated and summarized as shown in *Tables 3.11, 3.12* and *3.13.* This information was then fed into the database upon its creation.

Table 3.11:Crop data for maize

Crop Age (weeks)	***Growth Period***	***ET_O (mm/day)***	***K_C***	***Baseline water consumption, ET_{crop} (mm/day)*** ***$ET_{crop} = ET_O * K_C$***
0	Initial stage (3 weeks)	4.73	0.40	1.89
1				
2				
3				
4	Crop development stage (6 weeks)	4.73	0.80	3.78
5				
6				
7				
8				
9				
10	Midseason stage (7 weeks)	4.73	1.15	5.44
11				
12				
13				
14				
15				
16				
17	Late season stage (5 weeks)	4.73	0.70	3.31
18				
19				
20				
21				

Table 3.12: Crop data for kales

Crop Age (weeks)	***Growth Period***	***ET_O (mm/day)***	***K_C***	***Baseline water consumption, ET_{crop} (mm/day)*** ***$ET_{crop} = ET_O * K_C$***
0	Initial stage (3 weeks)	4.73	0.45	2.13
1				
2				
3				
4	Crop development stage (5 weeks)	4.73	0.60	2.84
5				
6				
7				
8				
9	Midseason stage (3 weeks)	4.73	1.00	4.73
10				
11				
12	Late season stage (2 weeks)	4.73	0.90	4.26
13				

Table 3.13 Crop data for cabbage

Crop Age (weeks)	***Growth Period***	***ET_O (mm/day)***	***K_C***	***Baseline water consumption, ET_{crop} (mm/day)*** ***$ET_{crop} = ET_O * K_C$***
0	Initial stage (3 weeks)	4.73	0.45	2.13
1				
2				
3				
4	Crop development stage (4 weeks)	4.73	0.75	3.55
5				
6				
7				
8	Midseason stage (9 weeks)	4.73	1.05	4.97
9				
10				
11				
12				
13				
14				
15				
16				
17	Late season stage (3 weeks)	4.73	0.90	4.25
18				
19				

Finally, in addition to crop data, the database also stored the login credentials (name, phone number, and PIN) of users who logged into the mobile application.

ii. Database development platform

Firebase Realtime Database was chosen as the database platform for this project due to its seamless integration with IoT applications and real-time data requirements. The decision was influenced by Firebase's ability to handle dynamic, continuously updated data, which aligned with the real-time nature of the soil moisture and weather parameters in agricultural settings. The platform's scalability and ease of use made it a suitable choice for managing the diverse dataset that included crop-specific water requirements, growth stages, and baseline weather parameters.

Firebase Realtime Database is a NoSQL database. It's a cloud-hosted database that stores data as JavaScript Object Notation (JSON) and allows for real-time synchronization across clients. Unlike traditional relational databases, NoSQL databases like Firebase are designed to be flexible and scalable, making them well-suited for applications with dynamic or evolving data structures, such as those in IoT and real-time data scenarios. The NoSQL model allowed for easy and efficient handling of unstructured or semi-structured data, making it a popular choice for modern, scalable, and flexible database solutions.

Additionally, Firebase's cloud-based architecture ensured accessibility from anywhere, enabling remote monitoring and control—a crucial feature for a project aiming to provide farmers with the ability to manage irrigation systems remotely. The platform's support for mobile and web applications simplified the development of the accompanying mobile app, enhancing the overall user experience. In retrospect, Firebase Realtime Database served as an efficient and reliable solution for handling the agricultural data essential for the project's irrigation control algorithm.

3.4.3 Methodology for realization of specific objective iii: Mobile app development

a. Description of the mobile application's role and functionalities

Prior to developing the mobile application, it was first necessary to establish the role of the app, its intended functionalities, and the app's platform (Android or iOS). This informed the mobile app development process and tools. The role of the mobile app in the project was to enable the user (farmer) to monitor and control the smart irrigation system remotely using a mobile device. To achieve this purpose, the mobile app was developed with the following functionalities:

i. Enabling the user to create their profile and login to the app
ii. Obtaining real-time weather data and weather forecasts based on location
iii. Updating the real-time weather data to the microcontroller
iv. Displaying the crop data parameters (ET_O, K_C, and ET_{crop}) from the database
v. Displaying the selected crop type, crop age, valve status and real-time soil moisture levels
vi. Controlling/ overriding the state of the solenoid valve
vii. Notifying the user of extremely high soil moisture levels
viii. Other functionalities such as saving the IP address of host/server, enabling/disabling notifications, saving last-used credentials, animating the logo upon app startup

b. Mobile app development platform and tools

Dart, Flutter, and Android Studio were chosen for the mobile app development based on their synergy and efficiency in creating a robust, cross-platform application tailored for the specific needs of the irrigation system project. Dart, as the programming language, offered a concise and expressive syntax, facilitating faster development and easy maintenance of the application's backend logic. Flutter, being an open-source UI toolkit, provided a rich set of pre-designed and

customizable widgets, ensuring a consistent and visually appealing user interface across different platforms. This was crucial for delivering a seamless user experience.

Android Studio, as the official IDE for Android development, seamlessly integrated with Dart and Flutter, streamlining the development, testing, and deployment processes. The combination of these technologies allowed for the creation of a single codebase that could efficiently target both Android and potentially other platforms. The choice was driven by the project's goal of developing a user-friendly and responsive mobile application capable of real-time communication with the microcontroller, weather data retrieval, and seamless integration with the irrigation system's database.

c. Implementation of the mobile application's functionalities

The implementation of the functionalities of the mobile app listed in *Part (a)* above is explained in more details in this part. As is usually the case with mobile app development, the codes that were developed to achieve the entire range of functionalities of the app in this project were extremely lengthy. Thus, only some code snippets are explained here for the app's core functionalities where necessary.

i. Enabling the user to create an account and login to the app

This was achieved by creating a "*Login*" page on the app and launching it as the first page when the app was opened for the first time. For first-time users, this page allowed the user to enter their details, that is, name, phone number, and a 4-digit PIN. These credentials were then be stored in the database in the backend. Once the account was created successfully, the user was prompted to enter their login credentials (phone number and PIN). The app would then query the database in the backend to check if the credentials entered by the user were valid, so as to determine whether

to grant or deny login access. Once a successful login was initiated, the app's homepage was displayed.

ii. Obtaining real-time weather data and weather forecasts based on location

To achieve this core functionality, the mobile app was integrated with a weather application programming interface (API). An Application Programming Interface (API) is a set of protocols, tools, and definitions that allows different software applications to communicate with each other. It defines the methods and data formats that applications can use to request and exchange information. APIs enable developers to access specific features or data from a service, library, or platform without exposing the internal workings. They serve as intermediaries that facilitate seamless integration between diverse software systems.

A weather API, specifically, is an API that provides access to real-time or historical weather data. It allows developers to retrieve information such as temperature, humidity, wind speed, and more for a given location and time. Weather APIs are valuable for applications that require up-to-date weather information, such as weather forecasts, climate monitoring, or any system where weather conditions play a role in decision-making, such as smart irrigation in this case. By integrating a weather API, developers can enhance their applications with accurate and timely meteorological data without having to manage the complex infrastructure required for collecting and processing such information.

The weather API used in the app was known OpenWeather. The platform provided accurate and reliable historical, real time, as well as forecast weather information for any location across the world. Through the weather platform's API key, the mobile application made API calls to the platform, which responded by sending the current weather data at the given time and selected

location, as well as hourly forecasts for the rest of that day and daily forecasts for the next two days. The weather parameters that were of interest from the API were temperature, cloud cover, relative humidity, and wind speed, since those parameters directly affected crop water requirements. The weather API platform allowed for 1000 free API calls per day. Since the mobile app was programmed to automatically make 12 API calls per hour to query for weather data (to obtain updated weather after every 5 minutes), the total API calls made per day were 288, which was well within the free limit. The obtained weather data was then displayed on the mobile app's *"Weather"* page, and refreshed immediately after an API call was made.

iii. Updating the real-time weather data to the microcontroller

The weather parameters obtained from the weather API were sent to the microcontroller after every 5 seconds for calculation of crop water requirements, to achieve near real-time processing. This was established by creating an endpoint in the microcontroller program as shown in the code snippet below.

```
AsyncWebServer server(80); Asynchronous ESP32 web server using port 80 for HTTP
communication

//Sever endpoint for mobile app communication

  // Route to update weather parameters
  server.on("/receiveSendData", HTTP_GET, [](AsyncWebServerRequest *request) {
    temperature = request->arg("temperature").toFloat();
    cloudCover = request->arg("cloudCover").toFloat();
    relativeHumidity = request->arg("relativeHumidity").toFloat();
    windSpeed = request->arg("windSpeed").toFloat();
  });

// Start server
  server.begin();
```

When the mobile app (client) called that endpoint on the microcontroller (server), the current values of temperature, cloud cover, relative humidity, and wind speed were sent to the microcontroller, which then used the updated values to calculate the specific crop water requirements. On the mobile app side, the mobile app called the endpoint to update the weather parameters by accessing the microcontroller's asynchronous web server through the ESP32's local IP address, as shown with the URL below.

- http://192.168.132.154/updateWeather?temperature=25.5&cloudCover=60.0&relativeHumidity=70.0&windSpeed=15.0

The IP address and the weather values used in the URL were examples to test the weather updating functionality in the microcontroller. In the mobile app however, the IP address in the URL was the actual local IP address of the microcontroller, and the weather values were the current values obtained from the weather API.

iv. Displaying the crop data parameters (ET_O, K_C, and ET_{crop}) from the database

The process of retrieving crop data parameters (ETO, KC, and ETcrop) from the Firebase Realtime Database and displaying them on the crops page of the app involved several steps. First, the Flutter framework was utilized for building the app's user interface, providing a structured and visually appealing layout. Dart, as the programming language for Flutter, was employed to implement the app's logic and functionality. Firebase SDK for Flutter was integrated into the project to establish a connection between the mobile app and the Firebase Realtime Database. This integration allowed the app to send queries to the database and receive real-time updates. Specifically, the app used Firebase's data retrieval methods, such as `once` or `on`, to fetch the current values of ET_O, K_C, and ET_{crop} associated with the selected crop. Upon retrieving the data, the app dynamically updated

the information on the crops page, ensuring that users could view the most recent and accurate crop parameters. This seamless integration of Flutter, Dart, and Firebase SDK enabled efficient communication with the database, providing users with valuable insights into their selected crops' water requirements. The retrieved crop details were displayed on the "*Database*" page of the app

v. Displaying the selected crop type, crop age, valve status and real-time soil moisture levels

The details of selected crop type, selected crop age, status of valve (ON/OFF), the required (calculated) soil moisture level, and the measured (real-time) soil moisture level were read by the mobile app from the microcontroller and displayed on the "*Home*" page of the app. This was established by creating endpoints in the microcontroller program for each of the values that was to be sent.

When the mobile app (client) called that endpoint on the microcontroller (server), the current values of selected crop type, selected crop age, status of valve (ON/OFF), the required soil moisture level, and the measured soil moisture level were read by the mobile app and displayed on the "*Home*" page of the application. The endpoint was called every 5 seconds to ensure seamless synchronization between the mobile application and the microcontroller, ensuring that the data was transferred with minimal lag in the operation. On the mobile app side, the mobile app called the endpoint to read the values of the listed parameters by accessing the microcontroller's asynchronous web server through the ESP32's local IP address, as shown with the URLs below.

- Reading the crop type: Maize, Kales or Cabbage (from microcontroller to app)
    - http://192.168.132.154/cropType
- Reading the age of the crop (from microcontroller to app)

- http://192.168.132.154/cropAge

- Reading measured soil moisture level (from microcontroller to app)
  - http://192.168.132.154/measuredSoilMoisture
- Reading required soil moisture level (from microcontroller to app)
  - http://192.168.132.154/requiredSoilMoisture

The IP address used in the URL was an example to test the reading functionality from the microcontroller. In the mobile app however, the IP address in the URL was the actual local IP address of the microcontroller. To ensure a more efficient means of data transfer, one common endpoint was created to send and receive all the necessary data across the microcontroller and mobile app simultaneously and using one route, as shown in the code snippet below.

```
// Route to update weather parameters, serve crop type, crop age, moisture
levels, and valve state
  server.on("/receiveSendData", HTTP_GET, [](AsyncWebServerRequest *request) {
    temperature = request->arg("temperature").toFloat();
    cloudCover = request->arg("cloudCover").toFloat();
    relativeHumidity = request->arg("relativeHumidity").toFloat();
    windSpeed = request->arg("windSpeed").toFloat();

    // now calculate values to send back to user
    String crop;
    switch (selectedCrop) {
      case 1:
        crop = "Maize";
        break;
      case 2:
        crop = "Kales";
        break;
      case 3:
        crop = "Cabbage";
        break;
      default:
        crop = "Not selected";
    }

    String age = String(selectedAge);
    String measuredMoisture = String(measuredSoilMoisture, 2); // Display with
2 decimal places
    String requiredMoisture = String(requiredSoilMoisture, 2); // Display with
2 decimal places
    String state = (relayState == HIGH) ? "ON" : "OFF";
```

```
    String response = "cropType," + crop
    + "\ncropAge," + age
    + "\nmeasuredSoilMoisture," + measuredMoisture
    + "\nrequiredSoilMoisture," + requiredMoisture
    + "\nrelayState," + state;

    request->send(200, "text/plain", response);
  });
```

When this endpoint was called, the microcontroller responded by all the both updating the weather parameters with those received from the app, and by simultaneously sending all the required values at once to the mobile application through a single route, as opposed to the many routes shown in the URLs above. The common URL used in the mobile application was of the form:

- Reading required soil moisture level (from microcontroller to app)

  http://192.168.132.154/receiveSendData

As explained with the other endpoints, in the mobile app side, the IP address in the URL was the actual local IP address of the microcontroller.

vi. Controlling/ overriding the state of the solenoid valve

The remote control of the solenoid valve in the circuit using the mobile app was established by creating an endpoint in the microcontroller program as shown in the code snippet below.

```
// Route to control the relay from the app
  server.on("/controlRelay", HTTP_GET, [](AsyncWebServerRequest *request) {
    String value = request->arg("value");

    if (value == "ON") {
      // Turn ON the relay
      digitalWrite(relayPin, HIGH);
      relayState = HIGH;
      request->send(200, "text/plain", "Relay turned ON");
    } else if (value == "OFF") {
      // Turn OFF the relay
      digitalWrite(relayPin, LOW);
      relayState = LOW;
      request->send(200, "text/plain", "Relay turned OFF");
    } else {
```

```
        request->send(400, "text/plain", "Invalid command");
    }
});
```

When this endpoint was called, the microcontroller responded by setting the relay pin to the HIGH or LOW value sent from the mobile, to set switch the valve ON or OFF, respectively. The URL used in the mobile application to control the state of the relay, and thus the solenoid valve was of the form:

- Controlling the relay ON/OFF (from the app to the microcontroller)
    - http://192.168.132.154/controlRelay?value=ON
    - http://192.168.132.154/controlRelay?value=OFF

As explained with the other endpoints, in the mobile app side, the IP address in the URL was the actual local IP address of the microcontroller.

vii. Notifying the user of extremely high soil moisture level

The mobile application code was configured to trigger an alert once the measured soil moisture level reached a set high threshold of 25mm of water, at which the soil was taken to be waterlogged or flooded. The alert from the mobile app used the mobile device's built-in vibration functionality and caused the device to vibrate periodically, which displayed an alert to the user to attend to the farm due to the excessively high soil moisture levels being recorded. The rationale for this notification was that other than natural causes such as rainfall and flooding, which would result in excessively high soil moisture levels, other man-made situations would cause flooding. This would include conditions such leaking or burst pipes, the valve opening erroneously or failing in the ON position and allowing excess water to flow, or erroneous readings from the soil moisture sensors

causing a false alarm. All of these scenarios would require the farmer to physically attend to the farm, establish the cause of the alert, and thus rectify it.

viii. Other functionalities such as saving the IP address of host/server, enabling/disabling notifications, animating the logo upon app startup

These extra functionalities of the mobile application were put in place for the convenience of the user, to enhance user experience and improve user-friendliness of the app. They are explained here briefly as follows:

- ***Saving the IP address of host/server*** – this functionality enabled the user to enter the IP address of the ESP32 microcontroller (server) through the settings of the mobile app in the "*Profile*" page, thus establishing the connection between the app (client) and the microcontroller (server). This feature would be essentially helpful in case of Dynamic Host Configuration Protocol (DHCP), which is a management protocol used on IP networks for automatically assigning IP addresses to devices connected to the network using a client-server architecture. In this protocol, there was a possibility that the IP address of the of the ESP32 microcontroller (server) would change. Since the microcontroller program was set to display the IP address of the ESP32 address on the LCD upon connecting to the network, the user would then enter that IP address into the mobile application to establish the connection between the app and the microcontroller. This would only be necessary in cases where the IP address of the microcontroller changed, otherwise if it remained the same as from the previous connection, the connection would be established automatically without having to enter the IP address again.
- ***Enabling/disabling notifications*** – this functionality enabled the user to enable or disable the vibration and visual alert from the app when the soil moisture exceeds the 25mm

threshold for waterlogging or flooding. This functionality was to enable the user to set their preference conveniently.

- ***Saving last-used credentials*** – once the user logged into the app, they remained signed in, even after leaving the application interface. This feature was important since the app needed to stay running in the background so as to update the weather information to the microcontroller; hence the user needed to stay logged in even in the background. Additionally, upon logging out the application would retain the username (phone number) of the most recent user on the "*Login*" page. Therefore, if the same user wanted to log back in, they would only have to enter their PIN, and not both the phone number and PIN; thus, the user experience was enhanced.
- Animating the logo on the splash screen upon app startup – upon stating the application, the app's splash screen was programmed to zoom the logo into the screen and then gently fade out and give way to the landing page ("*Login*" page or "Home" page if the user was already logged in). This feature was placed to improve the UI/UX interface and the aesthetics of the application.

3.5 Implementation and testing

The different modules constituting the entire project were assembled into a prototype based on the software and hardware design (and fabrication) that was carried out. The prototype was then tested for functionality. In the testing, the hardware setup was used to calculate the specific crop water requirements based on the crop's growth stage and weather conditions, measure the real time soil moisture levels, and thus actuate the valve accordingly to maintain an optimum soil moisture level for the crop at the given age and weather parameters. *Figure 3.6* shows the setup and testing process for the different crops.

Figure 3.6: Project setup implementation and testing (left) and soil moisture sensors reading the values of soil moisture for the crop(s) under test (right)

CHAPTER 4: RESULTS

The main objective of this project was to develop an improved smart irrigation system that optimized water supply for different types of crops based on their water requirements, weather conditions, and geographical location. This chapter presents the results obtained from the design, implementation and testing of the smart irrigation system, in line with the specific objectives.

4.1 Results for specific objective i: Control algorithm and hardware implementation

The first specific objective of the project was to develop a control algorithm and integrate it with a microcontroller and valve, thus adjust the amount of water supplied to crops based on real-time soil moisture data, weather conditions, and crop-specific water requirements. This objective was achieved by compiling and executing the algorithm/program using the hardware setup that were both explained in *Subsection 3.4.1*. The results for this objective are explained below.

4.1.1 Preliminary functions in the program

i. Welcome message, and current date and time

The welcome message, current date and time were displayed on the LCD as shown in *Figure 4.1*.

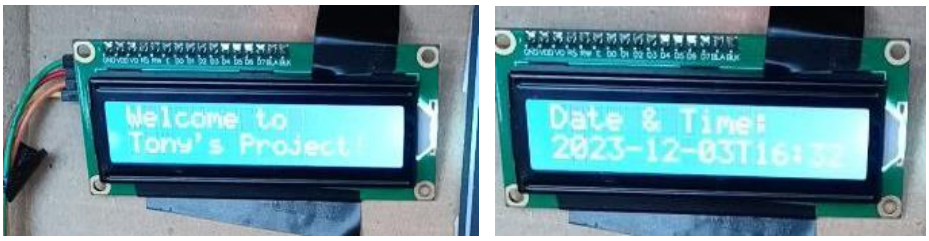

Figure 4.1: Welcome message, current date and time

ii. Wi-Fi connection

The Wi-Fi connection process, and the local IP of the ESP32 microcontroller were displayed on the LCD as shown in *Figure 4.2*.

Figure 4.2: Wi-Fi connection, network SSID and ESP32 local IP address

iii. Setup complete

Upon successful entry of crop data (crop type and crop age), the success message was displayed as shown in *Figure 4.3*.

Figure 4.3: Setup complete

4.1.2 Obtaining the crop type (Maize, Kales, or Cabbage) from the user

The user was prompted to enter the crop type as shown in *Figure 4.4*.

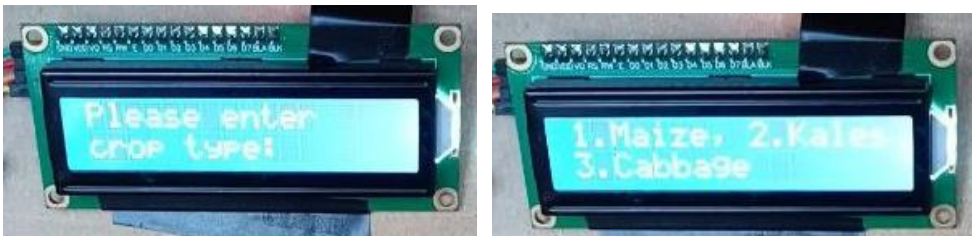

Figure 4.4: Prompt for the user to enter the crop type

If the user pressed any other key other than 1, 2, or 3, an error message was displayed on the LCD. Otherwise the correct selected crop type was displayed on the LCD. For example, if the user pressed key '*1*', the crop "*Maize*" was displayed, as shown in *Figure 4.5*.

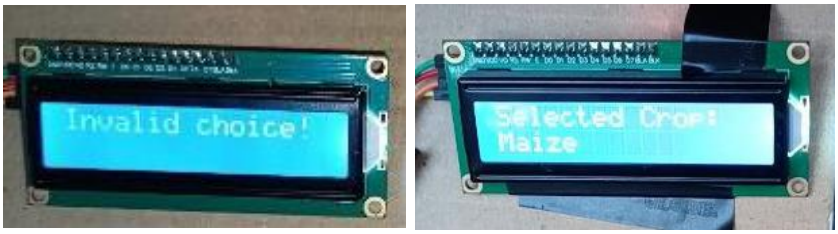

Figure 4.5: Displaying "Invalid choice" error, or the selected crop

4.1.3 Obtaining the age of the crop (in weeks) from the user

The user was prompted to enter the crop type as shown in *Figure 4.6*.

Figure 4.6: Prompt for the user to enter the crop age

If the user entered a value that was beyond the allowed range of crop age (20 weeks), an error message was displayed on the LCD. Otherwise the correct entred crop age was displayed on the LCD. For example, if the user entred key '7', the age "*7 weeks"* was displayed, as shown in *Figure 4.7*.

Figure 4.7: Displaying "Invalid age" error, or the correct entered age

4.1.4 Obtaining real-time weather data

The real time weather data (temperature, cloud cover, relative humidity, and wind speed) obtained from the weather API was updated and displayed on the LCD as shown in *Figure 4.8.* Note that this weather was obtained from the API, and the same information could be seen on the mobile application.

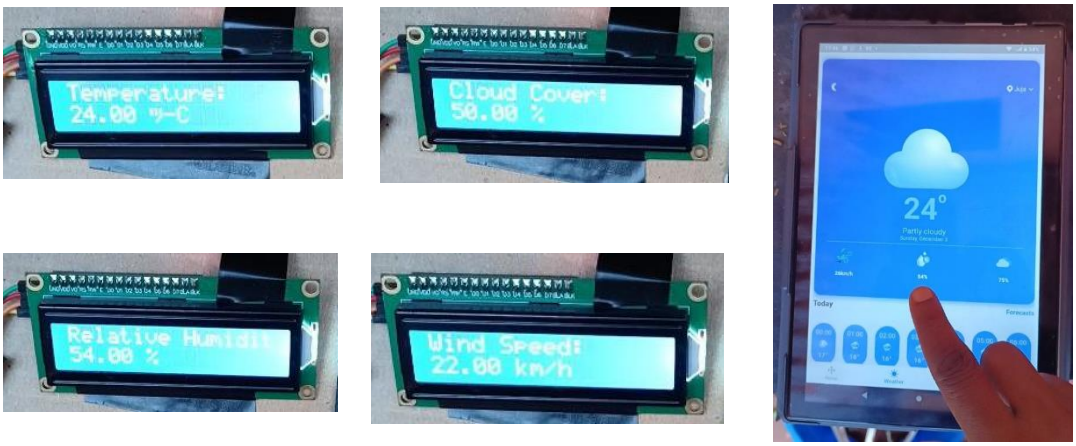

Figure 4.8: Real time weather data (temperature, cloud cover, relative humidity, and wind speed) from the weather API and on the LCD

4.1.5 Calculating the specific crop water requirements of the selected crop

The crop factors (ET_O, K_C and ET_{crop}) and specific crop water requirements were calculated and displayed on the LCD, as shown in *Figure 4.9.* Note that the required soil moisture could also be read on mobile application.

Figure 4.9: ET_O, K_C, ET_{crop}, and moisture need values displayed on the LCD

4.1.6 Measuring the real time soil moisture

The real time soil moisture levels read from the soil moisture sensors and calibrated onto a millimeter (mm) scale were displayed on the LCD as shown in *Figure 4.10.* This value of measured soil moisture could also be read from the mobile application as earlier shown in *Figure 4.9*.

Figure 4.10:Real time soil moisture level mapped onto the mm scale

4.1.7 Controlling the solenoid valve

Based on the calculated value of real-time required soil moisture and the real-time measured soil moisture level, the valve turned ON or OFF automatically to allow water to flow to the crops and thus maintain an optimum water supply to the crops. At the calculated required value of soil moisture of 14.83 mm, and the measured value of 5.19 mm, the valve status was ON, and water

could flow to the crops as shown in *Figure 4.11.* However, if the user wished to control the valve manually, they could do this using the mobile app.

Figure 4.11:Automatic valve control with changing water levels

4.1.8 Calibration and response of the soil moisture sensors to changes in soil moisture

Prior to their use in the circuit, the soil moisture sensors were calibrated with known amounts of water in the soil (known soil moisture). The raw analogue values read from the sensors were recorded and mapped onto the millimeter scale. The calibration data was as shown in *Table 4.1.*

Table 4.1: Calibration data for the soil moisture sensors

State of soil moisture	***Water amount (depth) in the soil (mm)***	***Soil Moisture Sensor 1 value***	***Soil Moisture Sensor 2 value***	***Soil Moisture Sensor 2 value***	***Average soil moisture value***
Dry	0 mm	1038	1039	1039	1039
	2 mm	1042	1043	1042	1042
Moist	10 mm	2052	2051	2053	2052
Wet	20mm	3023	3022	3023	3023
Flooded	27mm	3043	3042	3041	3042
	30mm	3046	3048	3045	3046

Upon calibration, the soil moisture sensors recorded fairly accurate readings for dry, moist, and wet soil conditions. When the sensors were not placed into any soil sample, they gave the results shown in *Figure 4.12.* The results for different soil moisture conditions are tabulated in *Table 4.3*.

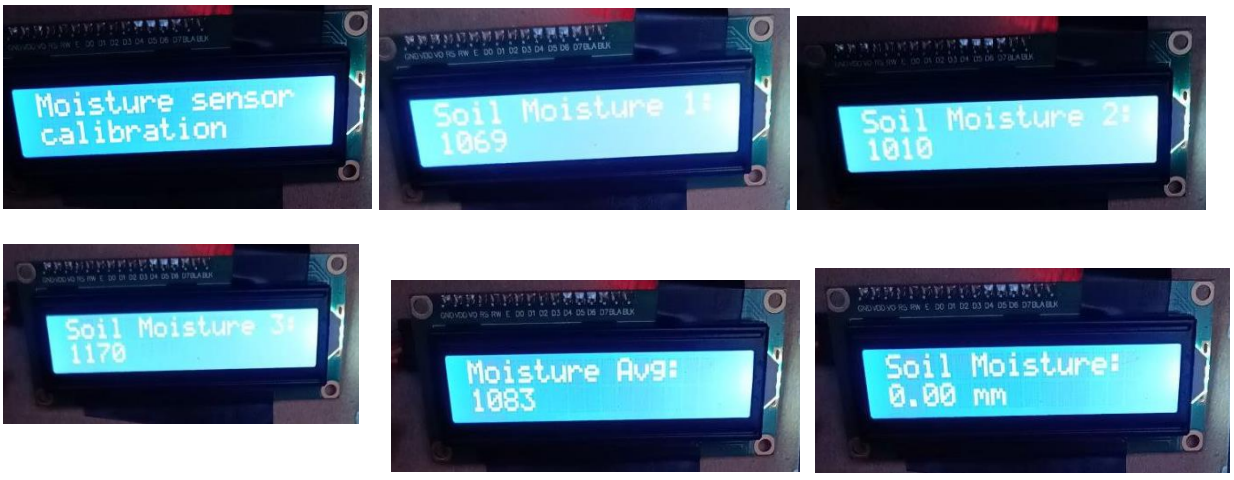

Figure 4.12: Soil moisture readings for dry conditions (no soil)

4.1.9 Summary of results from the hardware implementation

i. Test results using kales and cabbage

Similar tests were carried out with Kales at 5 weeks, and Cabbage at 5 weeks. For ease of observing the test results at a glance and keeping the report concise, the results from these tests have been tabulated in *Table 4.2*, avoiding the use of repetitive photos showing similar nature of data.

Table 4.2: Test results with maize, kales and cabbage

Test parameter	**Value**		
Crop type (Name)	***Maize***	***Kales***	***Cabbage***
Crop age (weeks)	7	5	5
Temperature (°C)	24.00	24.00	24.00
Cloud cover (%)	50.00	50.00	50.00
Relative humidity (%)	54.00	54.00	54.00
Wind speed (km/h)	22	22.00	25.90
ET_O (mm/day)	5.24	5.24	5.24
K_C	0.85	0.65	0.80
ET_{crop}	4.45	3.40	4.19
Required soil moisture (mm)	14.83	11.34	13.96
Measured soil moisture (mm)	5.19	23.59	28.47
Status of the solenoid valve	ON	OFF	OFF

The measured value of 25mm in the test with cabbage triggered a notification alert since the level was beyond the flooding threshold set at 30mm.

ii. Results from the calibration of the soil moisture sensors

The values obtained from the soil moisture sensors under different test soil conditions are tabulated in *Table 4.3*.

Table 4.3: Soil moisture sensor values under different test soil conditions

Visual nature of soil	***Dry/No soil***	***Moist***	***Wet***
Sensor 1 (raw value)	1069	1684	2991
Sensor 2 (raw value)	1010	1792	2877
Sensor 3 (raw value)	1170	1711	2815
Average raw value	1083	1729	2895
Calibrated average value	0.00mm	10mm	27mm

4.2 Results for specific objective ii: Database development

The second specific objective of the project was to develop a database with information on the water requirements of different crops at various stages of growth and weather conditions. This objective was achieved by creating the database on Firebase Realtime Database and updating the information as explained in *Subsection 3.4.2*. The results for this objective are explained using the screenshots below.

4.2.1 Overall project analytics on downloads and storage

Figure 4.13 shows the overall project analytics on downloads and storage of the IoT Smart Irrigation Project for 1 week prior to project presentation (November 25th, 2023 to December 1st, 2023), as provided by Firebase.

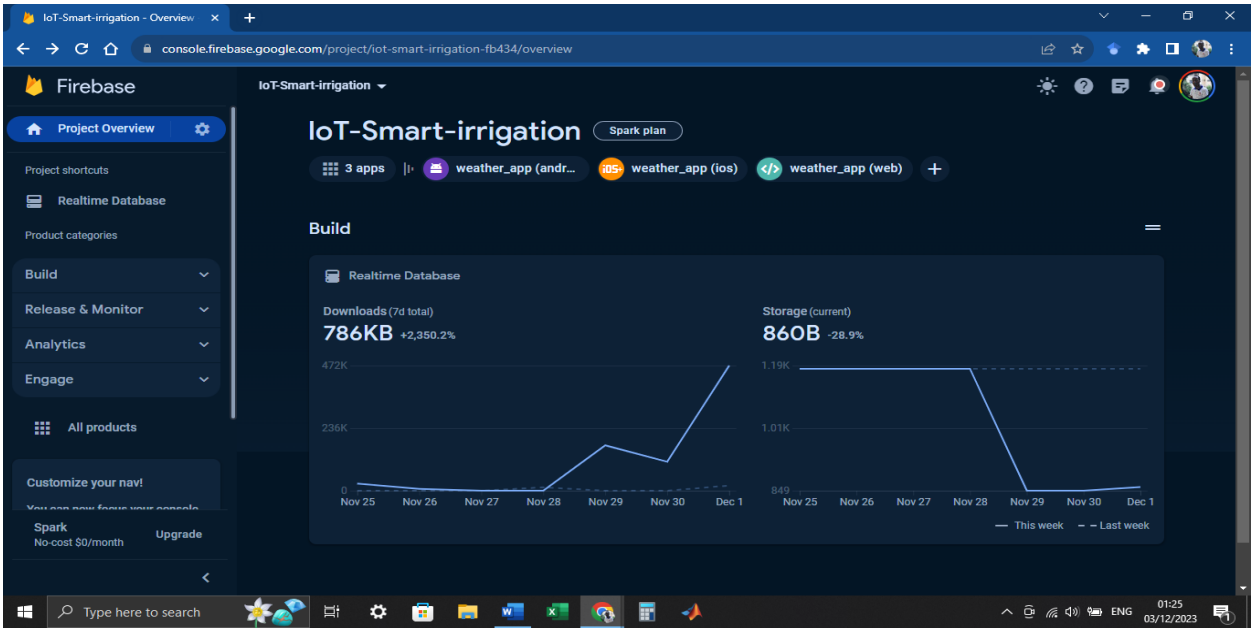

Figure 4.13: Snapshot of project analytics over time

4.2.2 Storing different crop types on the database

Figure 4.14 shows the three types of crops (maize, kales and cabbage) stored in the database.

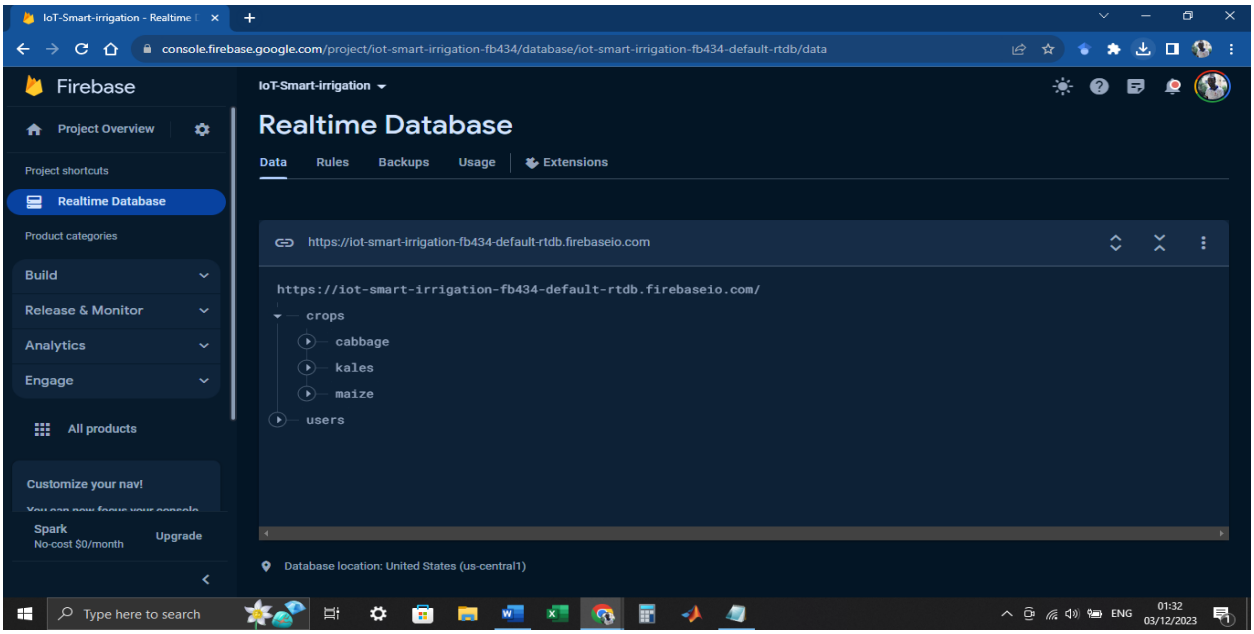

Figure 4.14: Crop types stored in the database

4.2.3 Storing the four stages of growth of each crop

Figures 4.15 and *4. 16* shows the four growth stages (initial stage, crop development stage, midseason stage, or late season stage) of the three types of crops (maize, kales and cabbage) stored in the database.

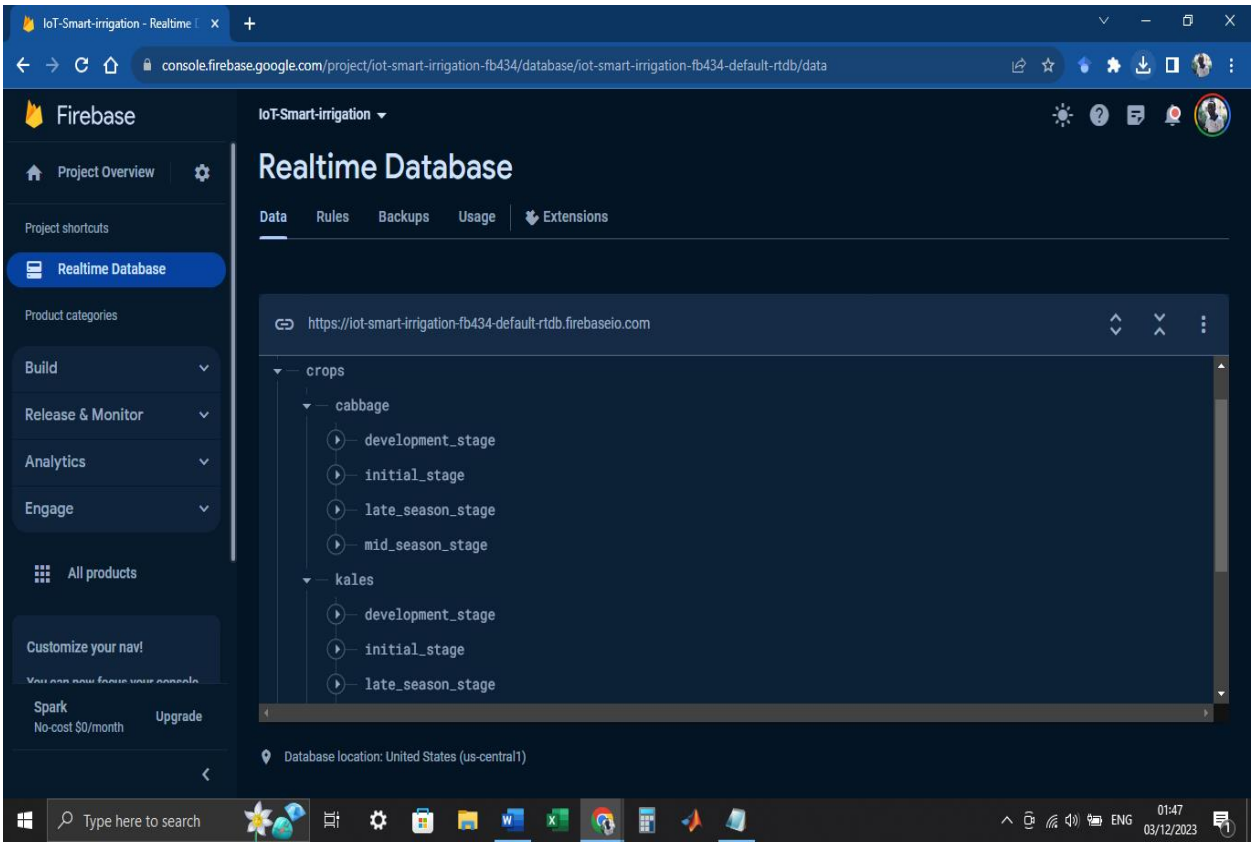

Figure 4.15: Four stages of growth of each stored in the database

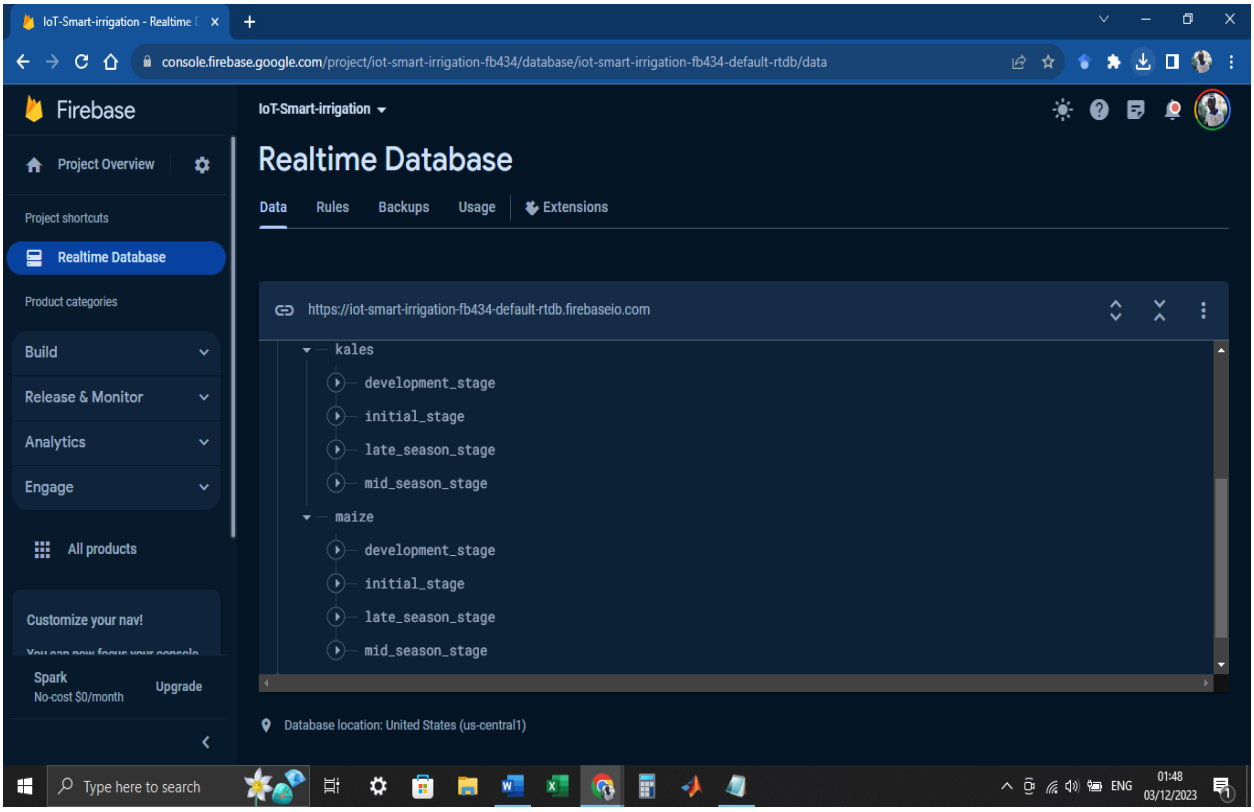

Figure 4.16: Four stages of growth of each stored in the database

4.2.4 Storing the specific crop water requirements of the three crops

The baseline values of ET_O, K_C and ET_{crop} were stored in the database, and also sent from the database to mobile app and the microcontroller, resulting in faster execution, along with the benefits of using the database to easily modify the parameters as opposed to hardcoding them. This is as shown in the results presented in *Section 4.1* and *Section 4.3*.

Figures 4.17 to 4. 22 shows the baseline crop water requirements and crop factors (ET_O, K_C and ET_{crop}) in each of the four growth stages (initial stage, crop development stage, midseason stage, or late season stage) of the three types of crops (maize, kales and cabbage) stored in the database.

Figure 4.17

Figure 4.18

Figure 4.19

Figure 4.20

Figure 4.21

Figure 4.22: Crop water requirements and crop factors of the three crops stored in the database

4.2.5 Storing the user credentials of the mobile app users

Figure 4.23 shows the login credentials (name, phone number, and PIN) of users who logged into the mobile application as stored in the database. These details have been redacted in this report for privacy reasons.

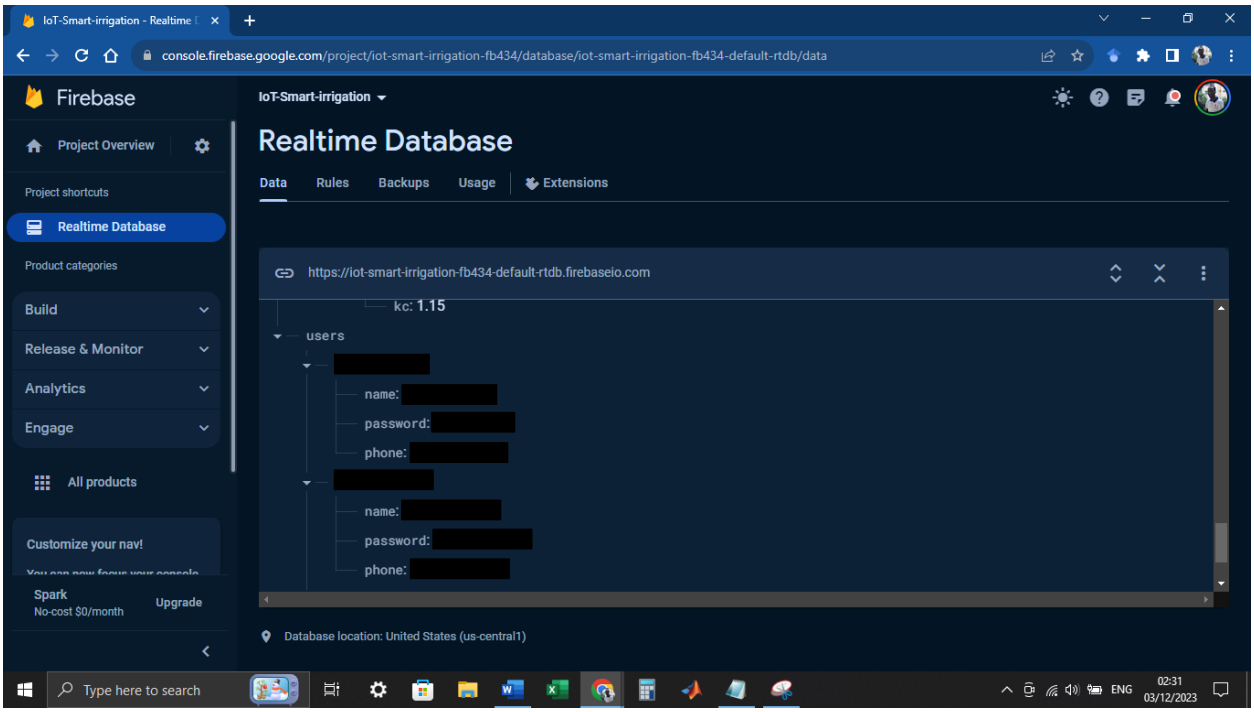

Figure 4.23: Login credentials of mobile application users

4.2.6 Overall results of the database information

Since Firebase Realtime Database is a NoSQL database, the entire information can be exported as JSON using the Export JSON Command in Firebase, as shown in *Figure 24.*

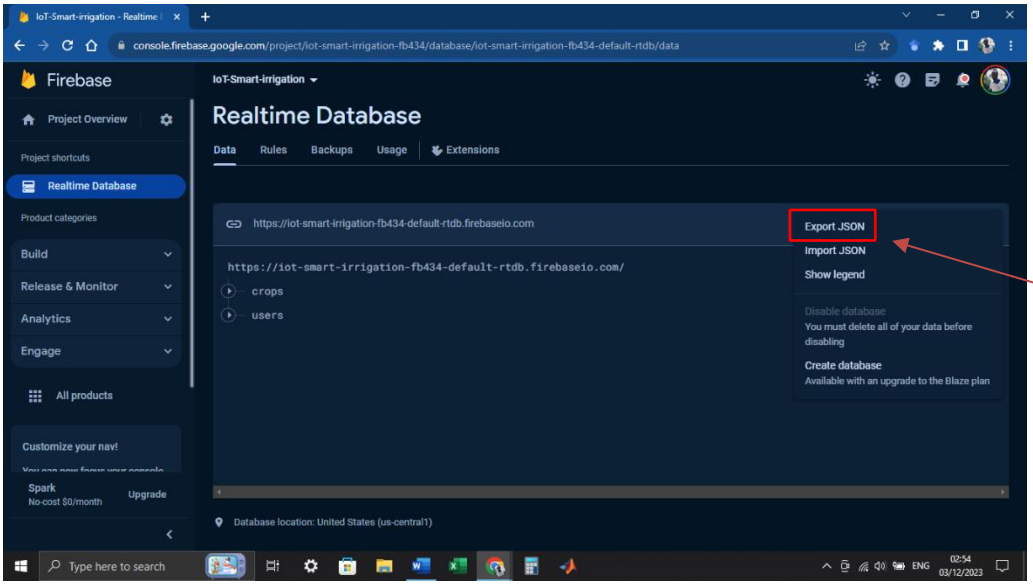

Figure 4.24: Exporting the entire data from the database as JSON

The data is similar in formatting to the one presented in the screenshots above, since the entire database information is huge, only a snippet of the JSON statements has been displayed here.

```
{
  "crops": {
    "cabbage": {
      "development_stage": {
        "etcrop": 3.55,
        "eto": 4.73,
        "kc": 0.75
      },
… data continues up to user credentials.
```

4.3 Results for specific objective iii: Mobile app development

The third specific objective of the project was to develop a mobile app where the farmer could monitor and control the smart irrigation system remotely. This objective was achieved by developing the mobile app using Dart programming language and Flutter UI software development

toolkit on the Android Studio IDE, as explained in *Subsection 3.4.3*. The results for this objective are explained using the screenshots below.

4.3.1 Enabling the user to create an account and login to the app

Figure 4.25 shows the mobile application's pages used to create an account for a new user, or to login for an existing user.

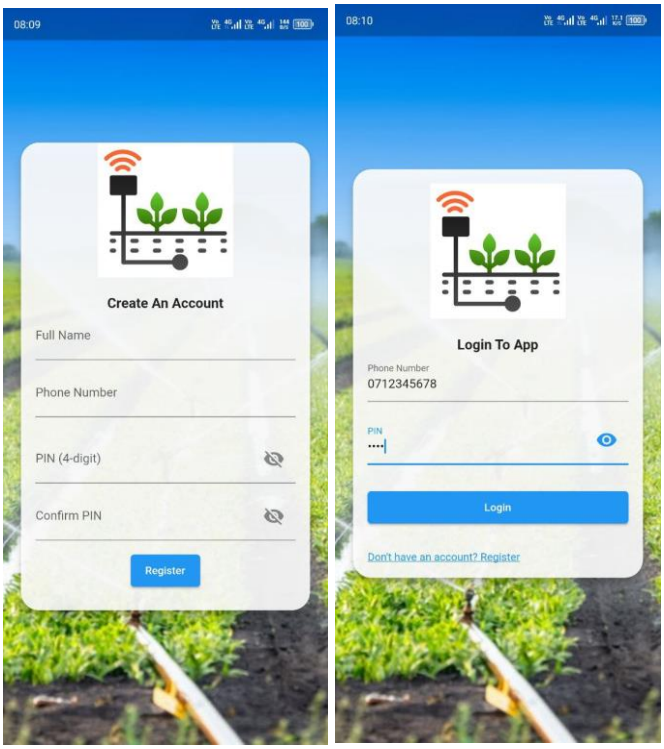

Figure 4.25: Registration and login pages

4.3.2 Obtaining real-time weather data and weather forecasts based on location

Figure 4.26 shows the real-time weather data and weather forecasts based on location (Juja, Kenya).

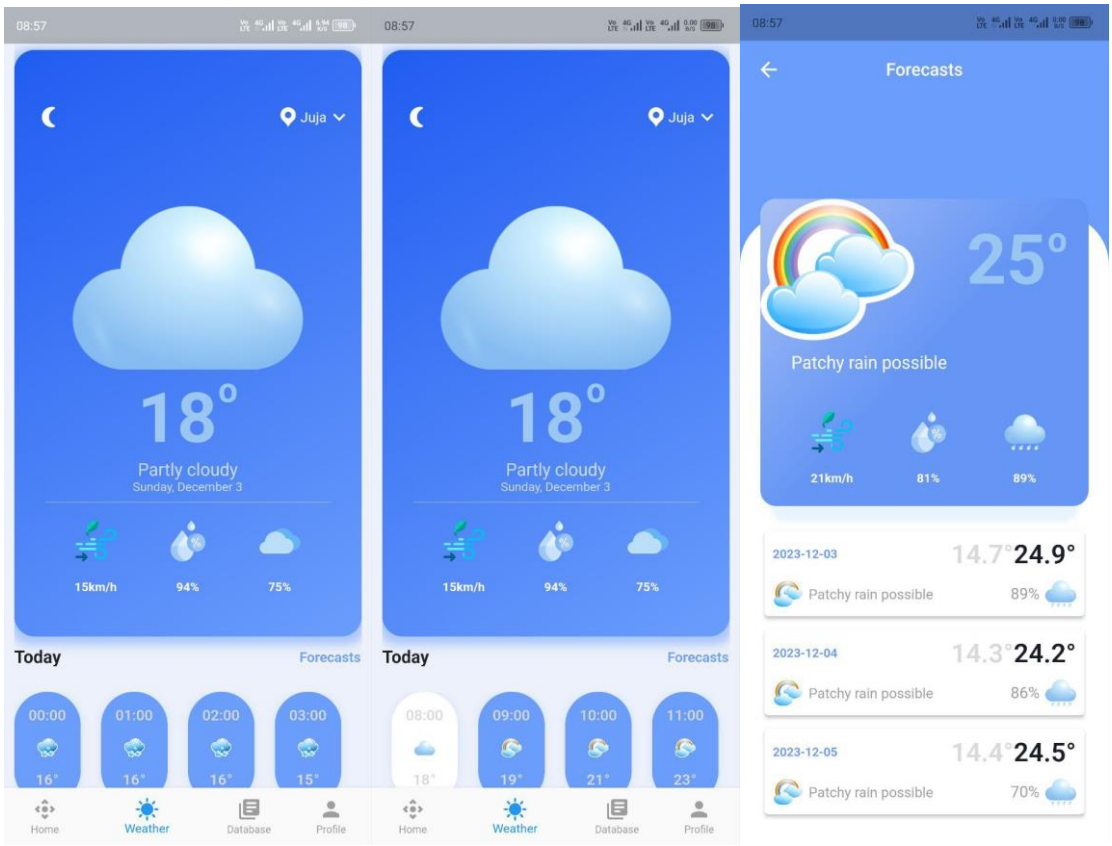

Figure 4.26: Real-time weather data and weather forecasts for Juja, Kenya

4.3.3 Updating the real-time weather data to the microcontroller

Figure 4.27 shows the real-time weather data updated from the mobile app to the microcontroller and displayed on the LCD.

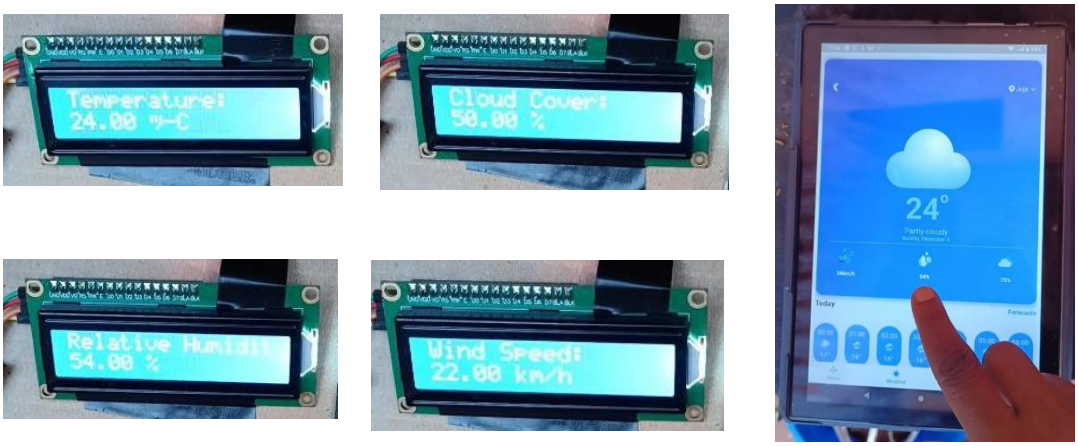

Figure 4.27: Real time weather data (temperature, cloud cover, relative humidity, and wind speed) from the weather API to the microcontroller

4.3.4 Displaying the crop data parameters (ET_O, K_C, and ET_{crop}) from the database

Figure 4.28 shows the crop data parameters (ET_O, K_C, and ET_{crop}) retrieved from the database and displayed on the app.

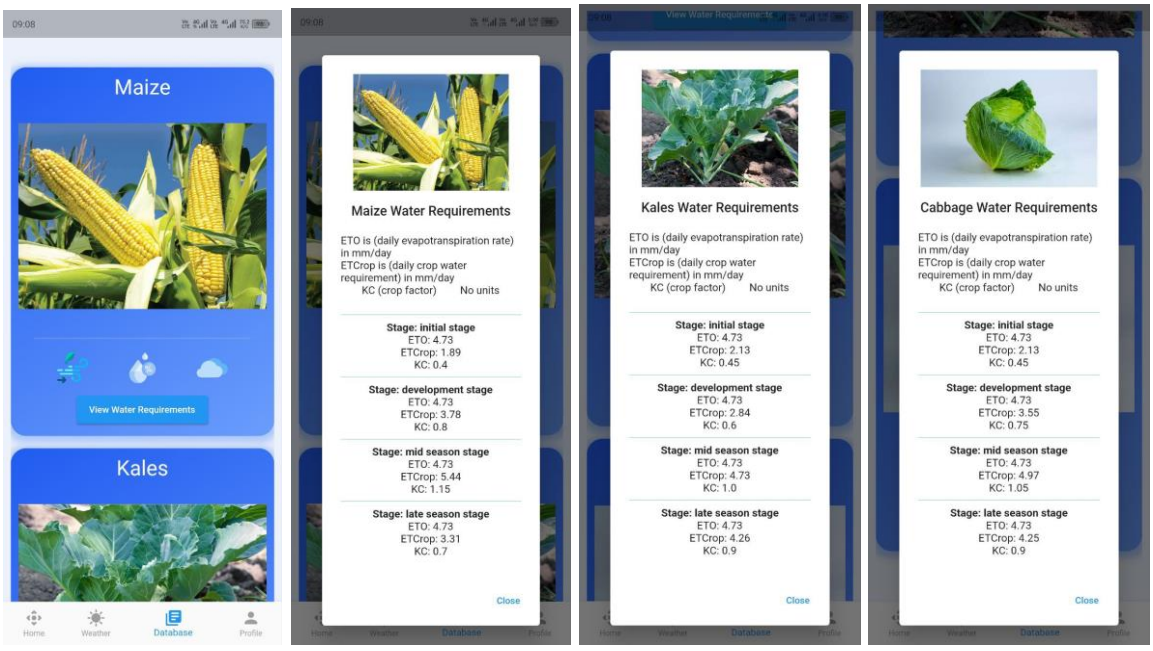

Figure 4.28: Crop data parameters (ET_O, K_C, and ET_{crop}) retrieved from the database

4.3.5 Displaying the selected crop type, crop age, valve status and real-time soil moisture levels

Figure 4.29 shows the selected crop type, crop age, valve status and real-time soil moisture levels displayed on the "*Home*" page of the app.

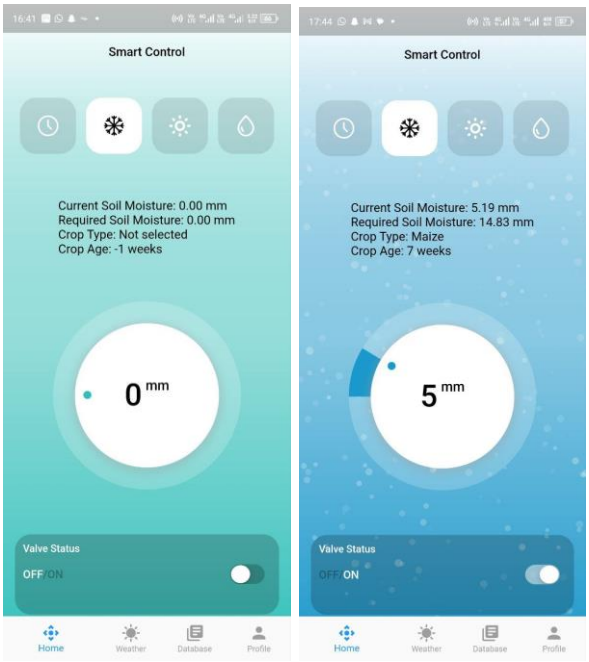

Figure 4.29: Selected crop type, crop age, valve status and real-time soil moisture levels displayed on the app

4.3.6 Controlling/ overriding the state of the solenoid valve

Figure 30 shows the app's functionality to control/override the state of the solenoid valve. Initially, the valve was automatically ON since the measured soil moisture was lower than the required moisture level. The manual override feature allowed the user to turn the valve OFF from the mobile app.

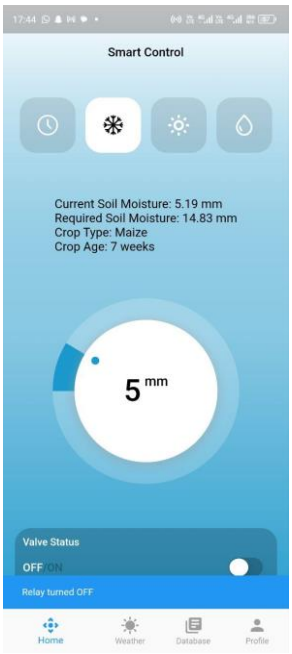

Figure 4.30: Selected crop type, crop age, valve status and real-time soil moisture levels displayed on the app

4.3.7 Notifying the user of extremely high soil moisture level

Figure 4.31 shows the app's functionality to notify the user of extremely high soil moisture levels (above 25mm of water).

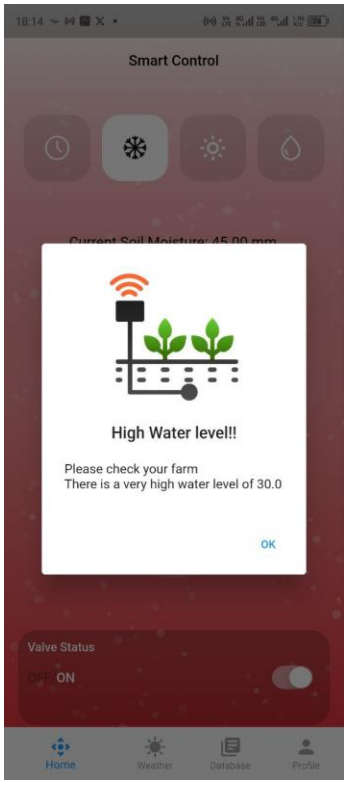

Figure 4.31: Notification for extremely high soil moisture level

4.3.8 Other functionalities such as saving the IP address of host/server, enabling/disabling notifications, saving last-used credentials, animating the logo upon app startup

- *Figure 32* shows the app's functionality of saving the saving the IP address of host/server, enabling/disabling notifications, and saving last-used credentials.

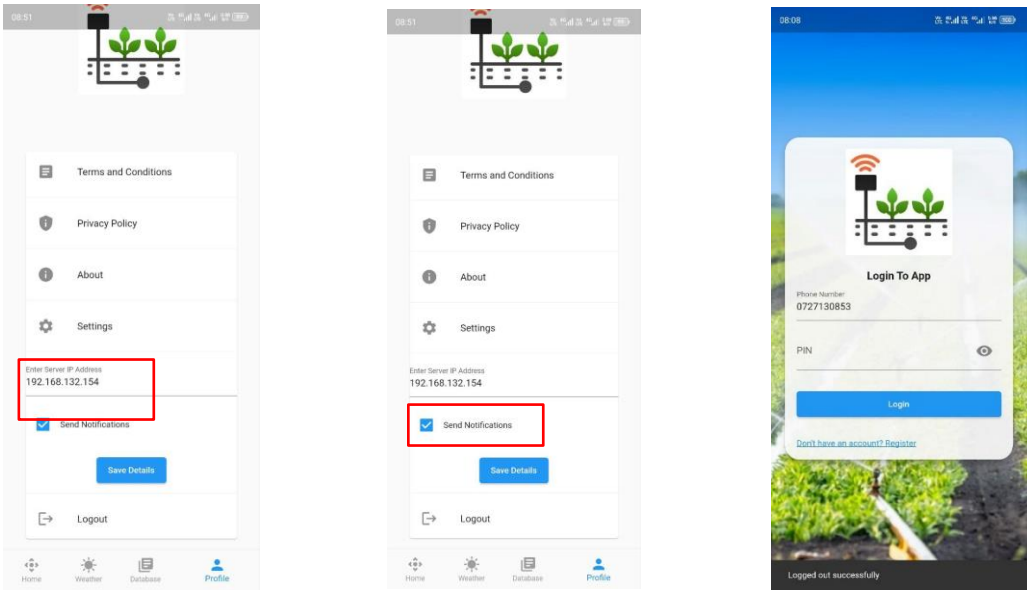

Figure 4.32: Saving the IP address of host/server, enabling/disabling notifications, saving last-used credentials (left to right)

- Splash screen logo effect – *Figure 33* shows the app's splash screen logo animation, shown by the logo zooming in before displaying the "*Login*" page.

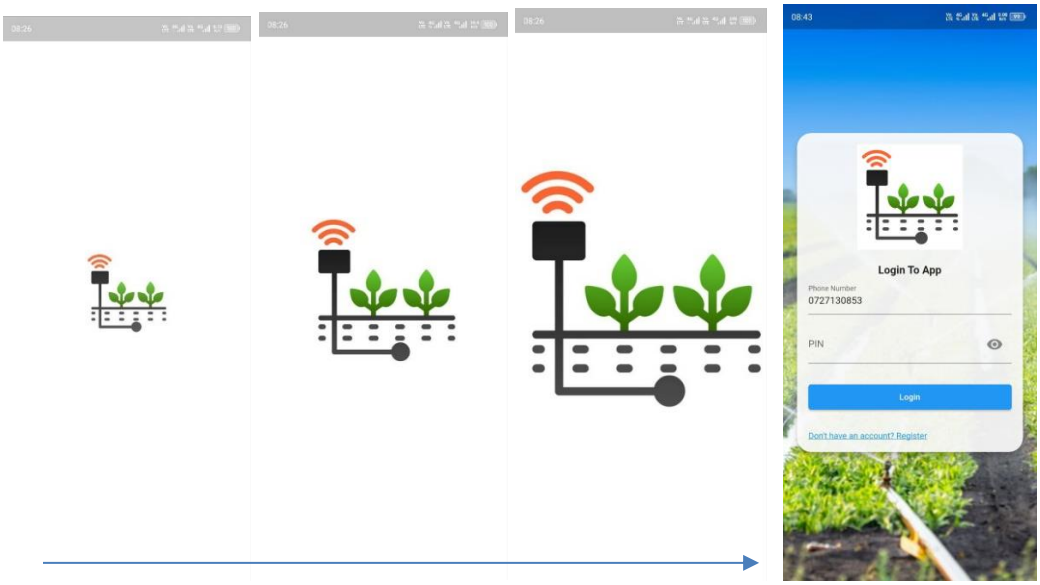

Figure 4.33: Splash screen logo animation (left to right transitions)

4.4 Combined/Overall results

The integration of all the specific objectives helped to achieve the project's overarching goal of developing the fully functional IoT-based smart irrigation system prototype. *Figure 4.34* shows the complete project setup.

Figure 4.34: Fully functional IoT-based smart irrigation system prototype

CHAPTER 5: CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

In conclusion, this project successfully achieved its primary goal of developing an advanced smart irrigation system, incorporating a sophisticated control algorithm and hardware integration. The control algorithm, executed on an ESP32 microcontroller, efficiently adjusted water supply to crops based on real-time soil moisture data, weather conditions, and crop-specific water requirements. The implemented hardware demonstrated accurate functionality, encompassing key features such as Wi-Fi connection setup, user prompts for crop type and age, real-time weather data retrieval, and automatic solenoid valve control.

The development of a Firebase Realtime Database facilitated the storage of crucial crop information, weather conditions, and user credentials. This database played a pivotal role in achieving seamless communication between the microcontroller and the mobile app, enabling real-time updates of weather data and crop parameters. The mobile app, developed using Dart and Flutter on Android Studio, provided an intuitive interface for users to create accounts, monitor weather forecasts, control the irrigation system, and receive notifications, showcasing a holistic approach to remote irrigation management.

The comprehensive results demonstrate the successful integration of the control algorithm, database, and mobile app, validating the project's effectiveness in optimizing water supply for diverse crops in response to varying environmental conditions. The combined efforts resulted in a fully functional IoT-based smart irrigation system prototype, poised to enhance agricultural practices through intelligent and resource-efficient irrigation management.

5.2 Challenges

The project encountered challenges such as calibration intricacies for soil moisture sensors, ensuring seamless communication between the microcontroller and Firebase, and optimizing the mobile app for real-time data synchronization. Overcoming these hurdles required iterative testing, debugging, and refining of both hardware and software components. Additionally, the limited funding available for the project was a significant constraint, leading to improvisation or selection of alternative resources to meet the project's objectives.

5.2 Recommendations/ Improvements

Future improvements for the project could involve enhancing the accuracy of soil moisture sensors through advanced calibration techniques. Integration of machine learning algorithms could refine the control algorithm for more precise irrigation decisions. Expanding the crop database to include a broader range of crops would enhance the system's versatility. Additionally, implementing a robust security framework for user authentication and data transmission in the mobile app could fortify the system against potential cyber threats. Continuous updates to the weather API and incorporating predictive analytics might further optimize irrigation decisions based on evolving climatic patterns, contributing to sustainable and efficient agricultural practices.

PROJECT BUDGET

Table 6.1 shows the budget used for the completion of the project.

Table 6.1: Project Budget

Item S/No.	Description	Quantity	Price (Ksh.)	Amount (Ksh.)
1.	ESP32 Microcontroller	1	1,200	1,200
2.	Soil moisture sensors	3	200	600
3.	LCD display module	1	500	500
4.	4x4 keypad	1	250	250
5.	Jumper wires	1	150	150
6.	Real Time Clock (RTC) module	1	450	450
7.	Boost converter	1	250	250
8.	Relay module (1 channel)	1	150	150
9.	Power supply (Lithium-Ion Batteries)	2	400	800
10.	Solenoid valve	1	1,000	1,000
11.	Fabrication	1	1,000	1,000
12.	Pipes, tanks, and fittings	Package	1,650	1,650
13.	Internet connectivity/bundles	8 months	150	1,250
14.	Stationery and printing	4	250	1,000
	TOTAL			**10,250**

PROJECT TIMEPLAN

Table 7.1 shows the timeplan that was followed for the completion of the project.

Table 7.1: Project Timeplan

Activity	Semester 1 - 2023				Semester 2 - 2023			
	May	***June***	***July***	***Aug.***	***Sep.***	***Oct.***	***Nov.***	***Dec.***
Documentation								
Project research/literature review								
Proposal writing								
Proposal presentation								
Design and coding								
Hardware configuration, testing, trouble-shooting and adjustment								
Final report writing								
Final project presentation								

APPENDICES

Appendix A

Code snippet for testing the values in the `calculateCropWaterRequirements()` function

```
// Display weather parameters and calculated values on LCD

  lcd.clear();
  lcd.print("Temperature: ");
  lcd.setCursor(0, 1);
  lcd.print(temperature);
  delay(2000);

  lcd.clear();
  lcd.print("Cloud Cover: ");
  lcd.setCursor(0, 1);
  lcd.print(cloudCover);
  delay(2000);

  lcd.clear();
  lcd.print("Relative Humidity: ");
  lcd.setCursor(0, 1);
  lcd.print(relativeHumidity);
  delay(2000);

  lcd.clear();
  lcd.print("Wind Speed: ");
  lcd.setCursor(0, 1);
  lcd.print(windSpeed);
  delay(2000);

  lcd.clear();
  lcd.print("Eto: ");
  lcd.setCursor(0, 1);
  lcd.print(eto);
  delay(2000);

  lcd.clear();
  lcd.print("Kc: ");
  lcd.setCursor(0, 1);
  lcd.print(baselineKc);
  delay(2000);
```

```
  lcd.clear();
  lcd.print("ETcrop: ");
  lcd.setCursor(0, 1);
  lcd.print(etcrop);
  delay(3000);
}
```

REFERENCES

[1] K. Pawlak and M. Kołodziejczak, "The role of agriculture in ensuring food security in developing countries: Considerations in the context of the problem of sustainable food production," *Sustainability,* vol. 12, no. 13, p. 5488, 2020.

[2] P. S. Brahmanand and A. K. Singh, "Precision irrigation water management-current status, scope and challenges.," *Indian J. Fertil,* vol. 18, pp. 372-380, 2022.

[3] M. Andronie, G. Lăzăroiu, M. Iatagan, C. Uță, R. Ștefănescu and M. Cocoșatu, "Artificial intelligence-based decision-making algorithms, internet of things sensing networks, and deep learning-assisted smart process management in cyber-physical production systems.," *Electronics,* vol. 10, no. 21, p. 2497, 2021.

[4] Z. Zhai, J. Martínez, V. Beltran and N. L. Martínez, "Decision support systems for agriculture 4.0: Survey and challenges.," *Computers and Electronics in Agriculture,* vol. 170, p. 105256, 2020.

[5] S. Mutambara, M. B. Darkoh and J. R. Atlhopheng, "A comparative review of water management sustainability challenges in smallholder irrigation schemes in Africa and Asia," *Agricultural Water Management,* vol. 171, pp. 63-72, 2016.

[6] N. Khan, R. L. Ray, G. R. Sargani, M. Ihtisham, M. Khayyam and S. Ismail, "Current progress and future prospects of agriculture technology: Gateway to sustainable agriculture," *Sustainability,* vol. 13, no. 9, p. 4883, 2021.

[7] U. Adhikari, A. P. Nejadhashemi and S. A. Woznicki, “Climate change and eastern Africa: a review of impact on major crops,” *Food and Energy Security,* vol. 4, no. 2, pp. 110-132, 2015.

[8] J. L. Hatfield and J. H. Prueger, “Temperature extremes: Effect on plant growth and development,” *Weather and climate extremes,* vol. 10, pp. 4-10, 2015.

[9] K. Obaideen, B. A. Yousef, M. N. AlMallahi, Y. C. Tan, M. Mahmoud, H. Jaber and M. Ramadan, “An overview of smart irrigation systems using IoT,” *Energy Nexus,* p. 100124, 2022.

[10] R. Togneri, C. Kamienski, R. Dantas, R. Prati, A. Toscano, J. P. Soininen and T. S. Cinotti, “Advancing IoT-based smart irrigation,” *IEEE Internet of Things Magazine,* vol. 2, no. 4, pp. 20-25, 2019.

[11] B. K. Jha, S. S. Mali, S. K. Naik and T. Sengupta, “Yield, water productivity and economics of vegetable production under drip and furrow irrigation in eastern plateau and hill region of India.,” *International Journal of Agricultural Science and Research,,* vol. 7, no. 3, pp. 43-50, 2017.

[12] R. E. Schattman, H. Jean, J. W. Faulkner, R. Maden, L. McKeag, K. C. Nelson and T. Ohno, “Effects of irrigation scheduling approaches on soil moisture and vegetable production in the Northeastern USA.,” *Agricultural Water Management,* vol. 287, p. 108428, 2023.

[13] O. Adeyemi, I. Grove, S. Peets and T. Norton, “Advanced monitoring and management systems for improving sustainability in precision irrigation.,” *Sustainability,* vol. 9, no. 3, p. 353, 2017.

[14] “Grove - Capacitive Moisture Sensor (Corrosion Resistant),” Seeed Studio, July 2022. [Online]. Available: https://wiki.seeedstudio.com/Grove-Capacitive_Moisture_Sensor-Corrosion-Resistant/. [Accessed 29 June 2023].

[15] Y. Zhu, S. Irmak, A. J. Jhala, M. C. Vuran and A. Diotto, “Time-domain and frequency-domain reflectometry type soil moisture sensor performance and soil temperature effects in fine-and coarse-textured soils,” *Applied Engineering in Agriculture,,* vol. 35, no. 2, pp. 117-134., 2019.

[16] L. A. Jean, “Know the Differences between Raspberry Pi, Arduino, and ESP8266/ESP32,” CNX Software, 24 March 2020. [Online]. Available: https://www.cnx-software.com/2020/03/24/know-the-differences-between-raspberry-pi-arduino-and-esp8266-esp32/. [Accessed 11 July 2023].

[17] A. A. Baradaran and M. S. Tavazoei, “Fuzzy system design for automatic irrigation of agricultural fields.,” *Expert Systems with Applications,* vol. 210, p. 118602, 2022.

[18] M. U. o. S. a. Technology, “Sensor based Automatic Irrigation System,” 9 June 2016. [Online]. Available: https://www.must.ac.ke/sensor-based-automatic-irrigation-system/. [Accessed 27 July 2023].

[19] J. O. Odhiambo and B. O. Odhiambo, “Development of Climate Smart Solar Powered Irrigation System for Improved Livelihoods in Garissa, Kenya.,” *Development,* vol. 6, no. 8, 2019.

[20] J. O'Keeffe, W. Buytaert, A. Mijic, N. Brozović and R. & Sinha, “The use of semi-structured interviews for the characterisation of farmer irrigation practices.,” *Hydrology and Earth System Sciences,* vol. 20, no. 5, pp. 1911-1924, 2016.

[21] A. Daccache, J. W. Knox, E. K. Weatherhead, A. Daneshkhah and T. M. Hess, “Implementing precision irrigation in a humid climate–Recent experiences and on-going challenges,” *Agricultural water management,* vol. 147, pp. 135-143, 2015.

[22] A. R. de Araujo Zanella, E. da Silva and L. C. P. & Albini, “Security challenges to smart agriculture: Current state, key issues, and future directions,” *Array,* vol. 8, p. 100048, 2020.

[23] Himax, “Rechargeable 18650 Lithium Ion Battery pack 3.7V 5000mAh,” Himax, [Online]. Available: https://himaxelectronics.com/product-item/18650-lithium-ion-battery-pack-3-7v-5000mah/. [Accessed 29 11 2023].

[24] Panasonic, “Lithium Coin Battery CR-2032,” Panasonic, [Online]. Available: https://www.panasonic.com/mea/en/consumer/batteries/dry-batteries/coin-batteries/cr-2032_5be.specs.html. [Accessed 29 11 2023].

[25] J. Doorenbos, W. Pruitt, A. Aboukhaled, J. Damagnez, N. Dastane, C. Van den Berg, P. Rijtema, O. Ashford, M. Frere and L. a. W. Division, “Crop water requirements,” Food and Agriculture Organization of the United Nations, 1992. [Online]. Available:

https://www.fao.org/documents/card/en?details=43024473-a2c0-515b-9bf2-0fa4d1dfbb19%2f. [Accessed 25 11 2023].

[26] AccuWeather, [Online]. Available: https://www.accuweather.com/en/ke/juja/828638/weather-forecast/828638. [Accessed 25 11 2023].