

A project Report on

Optimal Crop Watering System with Precision Agriculture with Emphasis to the Weather Conditions.

Submitted in partial fulfillment of the requirements for the award of the degree

Bachelor of Technology

in

Computer Science and Engineering

Submitted by

KARRI SUCHITA **20P31A0526**

KARRI HARI RAMA REDDY **20P31A0525**

SAMMIDI APARNA **21P35A0504**

KOVVURI VEERENDRA NADH REDDY **20P31A0532**

Under the esteemed supervision of

Mr. M M Siva Krishna, M. Tech (Ph.D)

Associate Professor



ADITYA COLLEGE OF ENGINEERING & TECHNOLOGY(A)
(Approved by AICTE, New Delhi & Affiliated to JNTUK, Kakinada)

Surampalem, East Godavari District

Andhra Pradesh - 533 437

2020-2024



ADITYA COLLEGE OF ENGINEERING & TECHNOLOGY(A) **(An AUTONOMOUS Institution)**

Approved by AICTE, New Delhi * Permanently Affiliated to JNTUK, Kakinada

Accredited by NBA* Accredited by NAAC A+ Grade with CGPA of 3.40

Recognized by UGC Under Sections 2(f) and 12(B) of the UGC Act, 1956

Aditya Nagar, ADB Road, Surampalem, Gandepalli Mandal, Kakinada District - 533437, A.P

Ph. 99591 76665, Email: office@acet.ac.in, www.acet.ac.in

VISION

To induce higher planes of learning by imparting technical education with

- ✓ International standards
- ✓ Applied research
- ✓ Creative Ability
- ✓ Value based instruction and to emerge as a premiere institute

MISSION

Achieving academic excellence by providing globally acceptable technical education by forecasting technology through

- ✓ Innovative Research And development
- ✓ Industry Institute Interaction
- ✓ Empowered Manpower


PRINCIPAL

PRINCIPAL
Aditya College of
Engineering & Technology
SURAMPALEM- 533 437



ADITYA COLLEGE OF ENGINEERING & TECHNOLOGY(A) (An AUTONOMOUS Institution)

Approved by AICTE, New Delhi * Permanently Affiliated to JNTUK, Kakinada
Accredited by NBA* Accredited by NAAC A+ Grade with CGPA of 3.40
Recognized by UGC Under Sections 2(f) and 12(B) of the UGC Act, 1956
Aditya Nagar, ADB Road, Surampalem, Gandepalli Mandal, Kakinada District - 533437, A.P
Ph. 99591 76665, Email: office@acet.ac.in, www.acet.ac.in

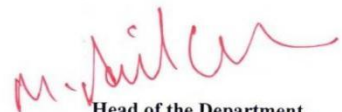
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION:

To become a center for excellence in Computer Science and Engineering education and innovation.

MISSION:

- Provide state of art infrastructure
- Adapt skill-based learner centric teaching methodology
- Organize socio cultural events for better society
- Undertake collaborative works with academia and industry
- Encourage students and staff self-motivated, problem-solving individuals using Artificial Intelligence
- Encourage entrepreneurship in young minds.


Head of the Department
Head of the Department
Dept. of CSE
Aditya College of Engineering
& Technology
SURAMPALEM-533437



ADITYA COLLEGE OF ENGINEERING & TECHNOLOGY(A) (An AUTONOMOUS Institution)

Approved by AICTE, New Delhi * Permanently Affiliated to JNTUK, Kakinada
Accredited by NBA * Accredited by NAAC A+ Grade with CGPA of 3.40
Recognized by UGC Under Sections 2(f) and 12(B) of the UGC Act, 1956
Aditya Nagar, ADB Road, Surampalem, Gandepalli Mandal, Kakinada District - 533437, A.P
Ph. 99591 76665, Email: office@acet.ac.in, www.acet.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM EDUCATIONAL OBJECTIVES

- PEO1: Capability to design and develop new software products as per requirements of the various domains and eligible to take the roles in various government, research organizations and industry
- PEO2: More enthusiastic to adopt new technologies and to improve existing solutions by reducing complexity which serves society requirements as per timeline changes
- PEO3: With good hands-on basic knowledge and ready improve academic qualifications in India or Abroad
- PEO4: Ability to build and lead the team to achieve organization goals.

Head of the Department

Head of the Department
Dept. of CSE
Aditya College of Engineering
& Technology
SURAMPALEM-533437



ADITYA COLLEGE OF ENGINEERING & TECHNOLOGY(A) (An AUTONOMOUS Institution)

Approved by AICTE, New Delhi * Permanently Affiliated to JNTUK, Kakinada
Accredited by NBA * Accredited by NAAC A+ Grade with CGPA of 3.40
Recognized by UGC Under Sections 2(f) and 12(B) of the UGC Act, 1956
Aditya Nagar, ADB Road, Surampalem, Gandepalli Mandal, Kakinada District - 533437, A.P
Ph. 99591 76665, Email: office@acet.ac.in, www.acet.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM SPECIFIC OUTCOMES


PSO 1: The ability to design and develop computer programs for analyzing the data.

PSO 2: The ability to analyze data & develop Innovative ideas and provide solution by adopting

emerging technologies for real time problems of software industry.

PSO 3: To encourage the research in software field that contribute to enhance the standards of

human life style and maintain ethical values.


Head of the Department
Head of the Department
Dept. of CSE
Aditya College of Engineering
& Technology
SURAMPALAM-533437




ADITYA COLLEGE OF ENGINEERING & TECHNOLOGY(A) (An AUTONOMOUS Institution)

Approved by AICTE, New Delhi * Permanently Affiliated to JNTUK, Kakinada
Accredited by NBA * Accredited by NAAC A+ Grade with CGPA of 3.40
Recognized by UGC Under Sections 2(f) and 12(B) of the UGC Act, 1956
Aditya Nagar, ADB Road, Surampalem, Gandepalli Mandal, Kakinada District - 533437, A.P
Ph. 99591 76665, Email: office@acet.ac.in, www.acet.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM OUTCOMES

- 1. ENGINEERING KNOWLEDGE:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
- 2. PROBLEM ANALYSIS:** Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
- 3. DESIGN/DEVELOPMENT OF SOLUTIONS:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
- 4. CONDUCT INVESTIGATIONS OF COMPLEX PROBLEMS:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- 5. MODERN TOOL USAGE:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
- 6. THE ENGINEER AND SOCIETY:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues, and the consequent responsibilities relevant to the professional engineering practice.
- 7. ENVIRONMENT AND SUSTAINABILITY:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
- 8. ETHICS:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- 9. INDIVIDUAL AND TEAM WORK:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- 10. COMMUNICATION:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, give and receive clear instructions.
- 11. PROJECT MANAGEMENT AND FINANCE:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- 12. LIFE-LONG LEARNING:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.


Head of the Department
Head of the Department
Dept. of CSE
Aditya College of Engineering
& Technology
SURAMPALEM-533437

ADITYA COLLEGE OF ENGINEERING & TECHNOLOGY(A)
(Approved by AICTE, New Delhi & Affiliated to JNTUK, Kakinada)

Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the project work entitled, "**OPTIMAL CROP WATERING SYSTEM WITH EMPHASIS TO WEATHER CONDITIONS**", is a bonafide work carried out by **KARRI SUCHITA (20P31A0526), KARRI HARI RAMA REDDY (20P31A0525), SAMMIDI APARNA (21P35A0504), KOVVURI VEERENDRA NADH REDDY (20P31A0532)** in partial fulfillment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING** from **ADITYA COLLEGE OF ENGINEERING AND TECHNOLOGY**, Surampalem, during the academic year 2023-2024.

This project work has not been submitted in full or part to any other University or educational institute for the award of any degree or diploma.

PROJECT SUPERVISOR

Mr. M M SivaKrishna M.Tech(Ph.D)

Associate Professor

HEAD OF THE DEPARTMENT

Dr. M. Anil Kumar, M.Tech.,(Ph.D)

Professor

EXTERNAL EXAMINER

DECLARATION

We hereby declare that this project entitled "**OPTIMAL CROP WATERING SYSTEM FOR PRECISION AGRICULTURE WITH EMPHASIS TO WEATHER CONDITIONS**" has been undertaken by us and this work has been submitted to Department of Computer Science & Engineering, **ADITYA COLLEGE OF ENGINEERING AND TECHNOLOGY (A)**, Surampalem affiliated to JNTUK, Kakinada, in partial fulfillment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING**.

We further declare that this project work has not been submitted in full or part to any other University or educational institute for the award of any degree or diploma.

PROJECT ASSOCIATES

KARRI SUCHITA	(20P31A0526)
KARRI HARI RAMA REDDY SAMMIDI	(20P31A0525)
SAMMIDI APARNA	(21P35A0504)
KOVVURI VEERENDRA NADH REDDY	(20P31A0532)

ACKNOWLEDGEMENT

It is with immense pleasure that we would like to express our indebted gratitude to my **project supervisor, Mr. M M Siva Krishna, M. Tech.,Ph.D.,** who has guided us a lot and encouraged us in every step of project work, his valuable moral support and guidance has been helpful in successful completion of this Project.

We wish to express our sincere thanks to **Dr.M.ANIL KUMAR M.Tech.,Ph.D., Head of the Department of CSE,** for his valuable guidance given to us throughout the period of the project work.

We feel elated to thank **Principal, Dr. Dola S Sanjay M.Tech.,Ph.D.,** of Aditya College of Engineering and Technology for his cooperation in completion of our project and throughout our course.

We feel elated to thank **Dr. A.RAMA KRISHNA M.Tech.,Ph.D., Dean (IQAC)** of Aditya College of Engineering and Technology (A) for his cooperation in completion of our project work.

We feel elated to thank **Dr.Ch V Raghavendran M.Tech.,Ph.D., Dean (Academics)** of Aditya College of Engineering and Technology (A) for his cooperation in completion of our project work.

We wish to express our sincere thanks to all faculty members, and lab programmers for their valuable guidance given to us throughout the period of the project.

We avail this opportunity to express our deep sense and heart full thanks to the **Management of Aditya College of Engineering & Technology (A)** for providing a great support for us by arranging the trainees, and facilities needed to complete our project and for giving us the opportunity for doing this work.

PROJECT ASSOCIATES

KARRI SUCHITA	(20P31A0526)
KARRI HARI RAMA REDDY	(20P31A0525)
SAMMIDI APARNA	(21P35A0504)
KOVVURI VEERENDRA NADH REDDY	(20P31A0532)

ABSTRACT

Scarcity of freshwater around the world is a very serious issue and there is need to come with a solution for the proper utilization of water resources. Smart irrigation technology uses weather data or moisture of soil data to determine the irrigation need of the landscape. The Internet of Things (IoT) based smart irrigation system is employed in the field agriculture to overcome the problem of water resource management. Internet of Things (IoT) and sensors are technologies that further enable farmers to know the exact status of their field, including soil temperature, amount of water required, weather conditions and much more. In this work, a smart irrigation system is used to predict the irrigation requirements of the field using different environmental factors. This system is based on the technology of wireless sensor networks.

Our Optimal Crop Watering System with emphasis to weather conditions is a novel solution for optimizing water usage in agriculture through the development of an automatic on/off water pumping system using Internet of Things (IoT) based on real-time soil moisture and weather conditions such as temperature and humidity. The primary objective is to enhance water efficiency and crop yield by ensuring precise irrigation tailored to the specific needs of various crops.

The system integrates with sensors to continuously monitor soil moisture levels and weather parameters by optimizing watering schedules and durations. All collected data is securely transmitted to a cloud server using Application Programming Interfaces (APIs).

This innovative proposal system aims to contribute to sustainable agriculture practices by reducing water wastage, minimizing manual intervention, and increasing overall crop productivity by offering a scalable and adaptable approach to modern agricultural challenges.

INDEX

CHAPTER	PAGE NO
ABSTRACT	x
LIST OF FIGURES	xiv
LIST OF TABLES	xv
1. INTRODUCTION	1-3
1.1 Purpose of the System	2
1.2 Scope of the System	2
1.3 Current System	3
1.4 Proposed System	3
2. REQUIREMENT ANALYSIS	4-16
2.1 Functional Requirements	4
2.2 Non-Functional Requirements	5
2.2.1 User Interface and Human Factors	5-6
2.2.2 Software Requirements	6-9
2.2.3 Hardware Requirements	9-15
2.2.4 Usability	16
2.2.5 Reliability	16
2.2.6 Performance	16
2.2.7 Supportability	16
2.2.8 Physical Environment	16
2.2.9 Security Requirements	16
2.2.10 Resource Requirements	16

3. SYSTEM ANALYSIS	17-19
3.1 Introduction	17
3.2 Use Cases	17
3.2.1 Actors	18
3.2.2 List of use cases	18
3.3.3 Use case diagrams	19
 4. SYSTEM DESIGN	 20-38
4.1 Introduction	20
4.2 System Architecture	20-21
4.3 System Object Model	21
4.3.1 Introduction	21
4.3.2 Subsystems	22
4.4 Object Description	22
4.4.1 Objects	23
4.4.2 Class Diagrams	23-25
4.5 Object Collaboration	25
4.5.1 Object Collaboration Diagram	26-28
4.6 Dynamic Model	29
4.6.1 Sequence Diagrams	29
4.6.2 Activity Diagrams	30
4.7 Database Design	31
4.7.1 Entity Relationship Diagrams	31-32
4.7.2 Database Tables	33-35
4.8 Static Model	36
4.8.1 Component Diagrams	36-37

4.8.2	Deployment Diagrams	37-38
5.	IMPLEMENTATION	39-63
5.1	Software Used	39-59
5.2	Source Code	60-63
6.	Testing	64-66
6.1	Introduction	64
6.2	Types of testing	64-67
7.	OUTPUT SCREENS	68-70
	CONCLUSION	71
	FUTURE SCOPE	72
	BIBLIOGRAPHY	73

LIST OF FIGURES

S.No	Figure No	NAME OF FIGURES	PAGE No
1	Figure 2.1	Soil Moisture Sensor	10
2	Figure 2.2	DHT11 Sensor	11
3	Figure 2.3	NodeMCU ESP8266	11
4	Figure 2.4	DC Motor	12
5	Figure 2.5	16x2 LCD Display	13
6	Figure 2.6	Buzzer	13
7	Figure 2.7	Relay	14
8	Figure 3.1	Use case Diagram	19
9	Figure 4.1	System Architecture	21
10	Figure 4.2	Class Diagram	25
11	Figure 4.3	Object Collaboration Diagram	28
12	Figure 4.4	Sequence Diagram	29
13	Figure 4.5	Activity Diagram	30
14	Figure 4.6	Entity Relationship Diagram	32
15	Figure 4.7	Process Design of Automatic Irrigation	33
16	Figure 4.8	Component Diagram	37
17	Figure 4.9	Deployment Diagram	38
18	Figure 5.1	Choose the components to install	40
19	Figure 5.2	Choose the installation Directory	40
20	Figure 5.3	Installation in Progress	41
21	Figure 5.4	Configuring the Arduino IDE	42
22	Figure 5.5	Exploring the Arduino IDE	43
23	Figure 5.6	Block Diagram of Embeeded C	47
24	Figure 5.7	Representation of Embeeded C Programming Steps	48
25	Figure 5.8	Checking for Version of Python in your system	52
26	Figure 5.9	Downloading window of VS Code	52
27	Figure 5.10	Setting up VS Code	53
28	Figure 5.11	Completing the Set up of VS Code	53

29	Figure 5.12	Successful installation of VS Code	54
30	Figure 5.13	Installing Flask Framework in VS Code	54
		Terminal	
31	Figure 5.14	Creating the Flask Application	55
32	Figure 5.15	Opening the terminal	56
33	Figure 5.16	Navigating to Directory using commands	57
34	Figure 5.17	Command to run Flask Application	57
35	Figure 5.18	Accessing the Application	57
36	Figure 7.1	Initial LCD Display	68
37	Figure 7.2	Display of T, H, M values	68
38	Figure 7.3	Display of Data through web interface	69
39	Figure 7.4	Graphical view of Temperature	69
40	Figure 7.5	Graphical view of Humidity	70
41	Figure 7.6	Web Interface for Selecting various crops	70
42	Figure 7.7	Prediction result of next water cycle	70

LIST OF TABLES

S.No.	NAME OF TABLE	PAGE No.
1	Database Table	35
2	Test Cases	67

1.INTRODUCTION

Plants are thirsty creatures, needing consistent access to water to thrive. But traditional watering methods, like relying on rainfall or manual scheduling, can be imprecise. This is where crop watering systems come in. These systems are designed to deliver water directly to the roots of plants, ensuring they receive the optimal amount for healthy growth.

Traditionally, irrigation systems have relied on timers or manual controls, which can lead to over or under-watering of crops. Overwatering wastes water and can lead to root rot, while underwatering stresses plants and reduces yields. Automatic irrigation systems based on soil moisture and weather conditions offer a more precise solution. By using sensors to measure soil moisture levels directly, these systems deliver water only when the plants actually need it. Additionally, they can factor in weather data like temperature and humidity to adjust watering schedules. This data helps compensate for increased evaporation during hot and dry periods, or avoid unnecessary watering during rain events. Ultimately, automatic irrigation systems based on real-time conditions promote healthier plants, improve water conservation, and can even boost crop yields.

By utilizing these systems after setting-up and programmed, the system takes over. Sensors monitor soil moisture, and the controller activates the water pump to deliver water when needed. This frees you from manual watering and ensures your plants receive the right amount of water, even when you're away.

These type of systems can save both time and water, allowing farmers to focus on other tasks while ensuring their crops receive the right amount of moisture at the right time. This leads to healthier plants and potentially higher yields. Additionally, automatic systems can reduce water runoff and nutrient loss, making them a more sustainable option for irrigation.

1.1 PURPOSE OF THE SYSTEM

In our daily lives, automatic watering systems become personal gardeners. They eliminate the guesswork and chore of manually remembering to water our plants, flowers, or even lawns. This is especially helpful for busy schedules or forgetful plant owners. By using sensors to monitor soil moisture, these systems deliver water only when plants truly need it, preventing underwatering or overwatering. This promotes healthier plants, avoids potential issues like root rot, and can even save money on water bills by preventing wasteful overwatering. With a programmed schedule and automatic operation, you can enjoy beautiful and thriving greenery without the constant worry of keeping them hydrated.

Automatic watering systems aim to deliver the perfect amount of water to crops, all by itself. This saves farmers time and water, while promoting healthy plants and bigger harvests.

1.1 SCOPE OF THE SYSTEM

Automatic watering systems extend their reach far beyond just household gardens. While they excel at keeping your potted plants and flowerbeds thriving, their scope encompasses a wide range of applications. From domestic use to professional agriculture, these systems can be scaled to fit various needs. On a domestic level, they can manage small vegetable gardens, rooftop gardens, or automate lawn sprinklers.

In professional settings, they play a vital role in greenhouses, nurseries, and large-scale farms, ensuring optimal irrigation for various crops. The ability to customize watering based on soil moisture, weather data, and specific plant needs makes them adaptable to a diverse range of scenarios. Ultimately, the scope of automatic watering systems lies in their ability to deliver water efficiently and precisely, promoting healthy plant growth across various domestic and agricultural applications.

The main scope of an automatic crop watering system is to automate the irrigation process, delivering water based on real-time needs. This optimizes water usage, saves labor, and promotes healthy crop growth.

1.2 CURRENT SYSTEM

In the existing system, weather conditions are not utilized to water the plant. In that system only watering the plants based on soil moisture is happened and it can't regulate watering schedules for a various crops. And also remote monitoring and control is the fault we can get from the system. Many new concepts are need to be developed to allow agricultural automation to flourish and deliver its full potential.

1.3 PROPOSED SYSTEM

In the proposed system we are using microcontroller which is designed to tackle the problems of agricultural sector regarding irrigation system with available water resources.

In this project we are using Moisture sensors, Humidity Sensor, Temperature sensor, AC submersible pump, relay driver. A submersible motor will getswitched ON /OFF depending on the soil moisture condition and status of motor can be displayed on 16X2 LCD. This system also monitor the environment parameters like Humidity Percentage, Temperature etc., so that this system can be very useful to agricultural system. And also this system can supports different kind of plants and pour the water based on the weather conditions.

The proposed system with soil moisture, temperature, and humidity sensors serves two main purposes:

- **Intelligent Irrigation Management:** By collecting real-time data on soil moisture, temperature, and humidity, the system can determine the optimal watering needs for the crops. This data is then uploaded to the cloud for processing and storage. Based on pre-programmed parameters or real-time analysis, the system can even automate irrigation, ensuring efficient water usage and preventing underwatering or overwatering.
- **Predictive Water Management:** The cloud storage of sensor data allows for historical analysis and future predictions. By monitoring trends and environmental factors, the system can anticipate future water needs based on the current conditions and crop growth stage. This "next level" of the water cycle refers to using sensor data to predict watering requirements, optimizing water usage and maximizing crop health.

2.REQUIREMENT ANALYSIS

Requirement analysis for an IoT project involves identifying and prioritizing the functionalities and features the project needs to deliver. It includes understanding user needs, defining system objectives, and specifying technical requirements such as hardware components, communication protocols, and data processing capabilities. Stakeholder consultation, market research, and feasibility studies are crucial in this phase to ensure that the project aligns with business goals, technological constraints, and user expectations. The analysis lays the foundation for the project's design, development, and implementation stages, guiding decision-making and resource allocation throughout the project lifecycle.

2.1 FUNCTIONAL REQUIREMENTS

Water Delivery: The system should deliver water to the plants through a network of drip lines, soaker hoses, or sprinklers (depending on the chosen setup).It should activate a water pump based on pre-programmed schedules or sensor readings.

Soil Moisture Monitoring: The system should utilize soil moisture sensors to measure the moisture level in the root zone of plants. Based on pre-defined thresholds, it should trigger watering when the soil moisture falls below a certain level.

Programming and Control: The system should have a user-friendly interface (either physical buttons or a mobile app) for programming watering schedules.It should allow users to define watering frequency, duration, and adjust settings based on plant needs and weather conditions.

Weather Integration: The system can integrate with weather data sources (temperature, humidity) to adjust watering schedules automatically. For example, it could increase watering frequency during hot and dry periods or skip watering during rain events.

Remote Monitoring and Control: The system might offer smartphone app connectivity to allow users to monitor soil moisture levels, watering history, and remotely adjust settings.

2.2 NON FUNCTIONAL REQUIREMENTS

In software development and engineering, non-functional requirements (NFRs) are specifications that define how a system will operate rather than what specific actions it should perform. These describe the overall qualities and characteristics of the system, often focusing on how well it functions.

Here's a breakdown of why NFRs are important:

1. They define user experience: NFRs consider factors like usability, security, and performance, which directly impact how users interact with the system.
2. They guide development decisions: Knowing the desired level of reliability, scalability, or maintainability helps developers choose the right technologies and approaches.
3. They ensure project success: By outlining quality attributes, NFRs help ensure the final product meets expectations and functions effectively in the real world.

Non-Functional Requirements are crucial for creating a well-rounded system that meets not just functional needs but also delivers a positive user experience and operates effectively in the long run.

2.2.1 USER INTERFACE AND HUMAN FACTORS

The user interface (UI) and human factors for an automatic watering system should be designed with ease of use, clarity, and user comfort in mind.

User Interface (UI):

- **Simplicity:** The interface should be straightforward and easy to navigate, even for users with limited technical experience. Menus and options should be clear and concise, avoiding technical jargon.
- **Visual Representation:** Utilize visual elements like graphs, charts, or icons to represent soil moisture levels, watering schedules, and sensor data. This allows users to quickly grasp system status without relying solely on text.
- **Customization:** The interface should allow users to customize watering

schedules based on plant types, pot sizes, and desired moisture levels. Simple sliders or drop-down menus can provide user control over these parameters.

HUMAN FACTORS

Readability: Ensure text on the display is large enough and uses a clear font for easy reading, especially for users with visual impairments. Adjustable brightness controls can further enhance readability in different lighting conditions.

Tactile Feedback: Physical buttons should provide clear tactile feedback upon pressing, confirming user input. This can be helpful for users who may not be looking directly at the interface while interacting with the system.

Error Prevention: Implement features to prevent accidental configuration changes. Confirmation prompts before saving settings or clear visual indicators for active watering cycles can minimize user errors.

User Guidance: Provide clear instructions and tutorials within the interface or accompanying user manuals. These resources should explain system setup, programming options, and troubleshooting steps in a user-friendly manner.

2.2.2 SOFTWARE REQUIREMENTS

- Embedded C programming language
- Arduino IDE Compiler
- Flask Framework

1) Embedded C programming language:

Embedded C is a set of language extensions for the C programming language by the C Standards Committee to address commonality issues that exist between C extensions for different embedded systems.

Embedded C programming typically requires nonstandard extensions to the C language in order to support enhanced microprocessor features such as fixed-point arithmetic, multiple distinct memory banks, and basic I/O operations. The C Standards Committee produced a Technical Report, most recently revised in 2008 and reviewed in 2013, providing a common standard for all implementations to adhere to. It includes a number of features not available in normal C, such as fixed-point arithmetic, named address spaces and basic I/O hardware addressing. Embedded C

uses most of the syntax and semantics of standard C, e.g., `main()` function, variable definition, datatype declaration, conditional statements (if, switch case), loops (while, for), functions, arrays and strings, structures and union, bit operations, macros, etc.

Embedded C is most popular programming language in software field for developing electronic gadgets. Each processor used in electronic system is associated with embedded software.

Embedded C programming plays a key role in performing specific function by the processor. In day-to-day life we used many electronic devices such as mobile phone, washing machine, digital camera, etc. These all device working is based on microcontroller that are programmed by embedded C.

In embedded system programming C code is preferred over other language.

Due to the following reasons:

- Easy to understand
- High Reliability
- Portability
- Scalability

2) Arduino IDE Compiler:

The Arduino integrated development environment (IDE) is a cross-platform application (for Windows, macOS, Linux) that is written in the programming language Java. It is used to write and upload programs to Arduino compatible boards, but also, with the help of 3rd party cores, other vendor development boards.

The source code for the IDE is released under the GNU General Public License, version 2. The Arduino IDE supports the languages C and C++ using special rules of code structuring. The Arduino IDE supplies a software library from the Wiring project, which provides many common input and output procedures. User-written code only requires two basic functions, for starting the sketch and the main program loop, that are compiled and linked with a program stub `main()` into an executable cyclic executive program with the GNU tool chain, also included with the IDE distribution.

The Arduino IDE employs the program argued to convert the executable code

into a text file in hexadecimal encoding that is loaded into the Arduino board by a loader program in the board's firmware.

Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programmed (referred to as a microcontroller) and a ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the computer code to the physical board.

Arduino provides a standard form factor that breaks the functions of the micro-controller into a more accessible package.

3) Flask Framework :

Flask is a lightweight web application framework written in Python. It's designed to be simple, flexible, and easy to use, making it a popular choice for building web applications, including those that serve machine learning models. Flask provides tools and libraries for handling web requests, managing routes, and generating responses, making it well-suited for creating APIs to deploy machine learning models.

When it comes to deploying machine learning models, Flask serves as a bridge between the model and the end-user application. Machine learning models typically operate in the backend, processing data and making predictions. Flask allows developers to create a RESTful API around these models, enabling seamless communication between the model and the frontend or other applications.

One of the key reasons Flask is favored for deploying machine learning models is its simplicity. Flask has a minimalistic design, allowing developers to quickly set up endpoints to receive input data, pass it to the model for processing, and return the results to the client. Its lightweight nature means that Flask applications have low overhead, making them efficient for handling model inference requests even at scale.

Moreover, Flask integrates seamlessly with popular machine learning libraries such as TensorFlow, PyTorch, and scikit-learn. This compatibility enables developers to easily incorporate trained models into Flask applications without significant modifications. Additionally, Flask's extensive ecosystem of extensions and plugins provides further flexibility, allowing developers to add functionalities like authentication, logging, and caching to their machine learning deployment pipelines.

Another advantage of using Flask for deploying machine learning models is its compatibility with various deployment environments. Whether deploying on a traditional server, a cloud platform like AWS or Google Cloud, or in containerized environments like Docker or Kubernetes, Flask applications can be packaged and deployed with ease.

2.2.3 HARDWARE REQUIREMENTS

- Soil Moisture Sensor
- Temperature and Humidity Sensor (DHT11)
- NodeMCU ESP8266
- Water Pump
- LCD Display
- Buzzer
- Relay
- Remote Power Supply
- Wi-Fi Module
- Voltage Regulators

HARDWARE COMPONENTS DESCRIPTION

SOIL MOISTURE SENSOR

Soil moisture sensors measure the volumetric water content in soil. Since the direct gravimetric measurement of free soil moisture requires removing, drying, and weighing of a sample, soil moisture sensors measure the volumetric water content indirectly by using some other property of the soil, such as electrical resistance, dielectric constant, or interaction with neutrons, as a proxy for the moisture content.

A soil moisture sensor acts like an early warning system for your plants' thirst. These sensors are typically probes inserted directly into the soil near plant roots. They come in two main types:

1. Resistive: These sensors measure the electrical resistance between two electrodes within the probe. Dry soil offers high resistance, while wet soil conducts electricity more easily, resulting in lower resistance. The Arduino can translate these resistance readings into a soil moisture level.

2. Capacitive: These sensors measure the electrical capacitance between the probe and the surrounding soil. Water has a higher dielectric constant than air, so more moisture in the soil leads to a higher capacitance value. The Arduino can interpret these capacitance changes as a measure of soil moisture.

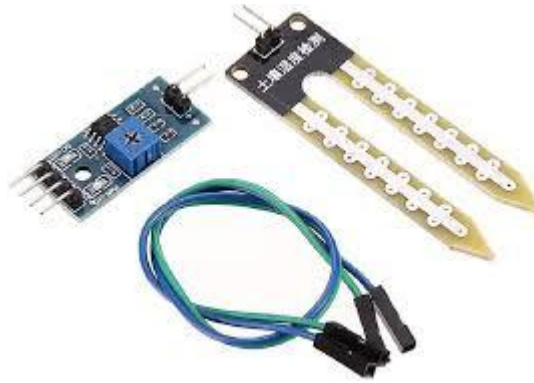


Figure 2.1: Soil Moisture Sensor

TEMPERATURE AND HUMIDITY SENSOR

The DHT11 is a commonly used Temperature and humidity sensor. The sensor comes with a dedicated NTC to measure temperature and an 8-bit microcontroller to output the values of temperature and humidity as serial data. The sensor is also factory calibrated and hence easy to interface with other microcontrollers. The sensor can measure temperature from 0°C to 50°C and humidity from 20% to 90% with an accuracy of $\pm 1^\circ\text{C}$ and $\pm 1\%$.

By incorporating a DHT11 sensor, our system can factor in temperature and humidity readings alongside soil moisture data. This allows for more nuanced watering adjustments. For example, during hot and dry periods (high temperature, low humidity), the system might increase watering frequency to compensate for increased evaporation. Conversely, in cool and humid conditions, it might adjust watering down to avoid oversaturation.

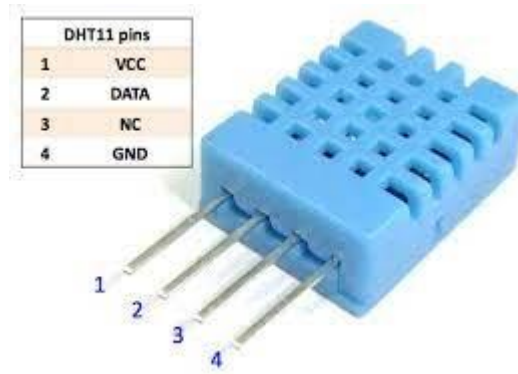


Figure 2.2: DHT11 Sensor

NODEMCU ESP8266

The NodeMCU ESP8266 features a Tensilica 32-bit RISC CPU Xtensa LX106 (80MHz clock speed) which operates at 3.3V. It has 4MB of Flash memory and 64K of SRAM. An on-board voltage regulator allows input voltages from 7-12V. It has 16 digital IO pins, 1 analogue input. It also supports SPI and I2C.

It also features on-board 2.4GHz Wi-Fi and has a PCB antenna giving good range. A micro-USB port enables it to be powered and programmed. There are basically 4 types of memory available on the ESP8266:

RAM: data will be lost after a power cycle or deep sleep. Unlimited write cycles. Amount of storage depends of how much RAM is used by other data.

RTC memory: data will be lost after power cycle but will persist through deep sleep. 768 bytes of storage. Unlimited write cycles.

EEPROM: is actually the flash memory being used to emulate EEPROM. Data will persist through power cycle and deep sleep. 4096 bytes of storage.

SPIFFS: data will persist through power cycle and deep sleep.

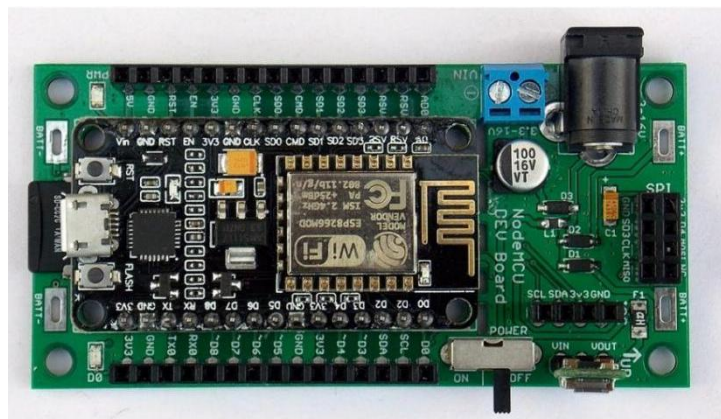


Figure 2.3: NodeMCU ESP8266

DC MOTOR

A DC motor is an electrical motor that uses direct current (DC) to produce mechanical force. The most common types rely on magnetic forces produced by currents in the coils. Nearly all types of DC motors have some internal mechanism, either electromechanical or electronic, to periodically change the direction of current in part of the motor.

DC motors were the first form of motors widely used, as they could be powered from existing direct-current lighting power distribution systems. A DC motor's speed can be controlled over a wide range, using either a variable supply voltage or by changing the strength of current in its field windings. Small DC motors are used in tools, toys, and appliances. The universal motor, a lightweight brushed motor used for portable power tools and appliances can operate on direct current and alternating current. Larger DC motors are currently used in propulsion of electric vehicles, elevator and hoists, and in drives for steel rolling mills. The advent of power electronics has made replacement of DC motors with AC motors possible in many applications.

A coil of wire with a current running through it generates an electromagnetic field aligned with the center of the coil. The direction and magnitude of the magnetic field produced by the coil can be changed with the direction and magnitude of the current flowing through it.



Figure 2.4: DC Motor

LCD DISPLAY

Liquid Crystal Display (LCD) screen is an electronic display module. An LCD has a wide range of applications in electronics. The most basic and commonly used LCD in circuits is the 16x2 display. LCDs are commonly preferred in display because they are cheap, easy to programme and can display a wide range of characters and animations. A 16x2 LCD have two display lines each capable of displaying 16 characters. This LCD has Command and Data registers. The command register stores command instructions given to the LCD while the Data register stores the data to be displayed by the LCD.



Figure 2.5: 16X2 LCD Display

BUZZER

A buzzer is a small electronic device that can produce an audible beep or pulsating sound. In an automatic watering system, it is to be used as a basic alert system.



Figure 2.6: Buzzer

RELAY

A relay is an electrically operated switch. Many relays use an electromagnet to mechanically operate a switch, but other operating principles are also used, such as solid-state relays. Relays are used where it is necessary to control a circuit by a low power signal (with complete electrical isolation between control and controlled circuits), or where several circuits must be controlled by one signal. The first relays were used in long-distance telegraph circuits as amplifiers: they repeated the signal coming in from one circuit and re-transmitted it on another circuit. Relays were used extensively in telephone exchanges and early computers to perform logical operations.

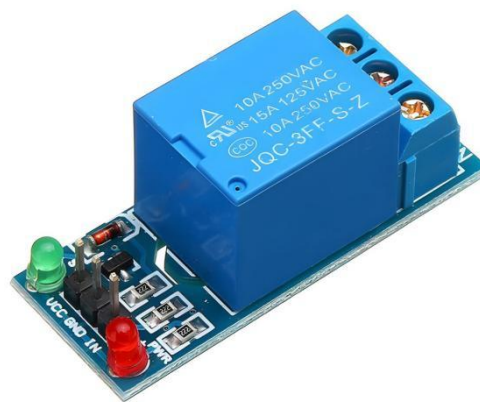


Figure 2.7: Relay

VOLTAGE REGULATOR

A voltage regulator is designed to automatically maintain a constant voltage level. A voltage regulator may be a simple "feed-forward" design or may include negative feedback control loops. It may use an electromechanical mechanism, or electronic components. Depending on the design, it may be used to regulate one or more AC or DC voltages. Electronic voltage regulators are found in devices such as computer power supplies where they stabilize the DC voltages used by the processor and other elements. In automobile alternators and central power station generator plants, voltage regulators control the output of the plant. In an electric power distribution system, voltage regulators may be installed at a substation or along distribution lines so that all customers receive steady voltage independent of how much power is drawn from the line.

TRANSFORMER

A transformer is an electrical device that transfers electrical energy between two or more circuits through electromagnetic induction. Electromagnetic induction produces an electromotive force across a conductor which is exposed to time varying magnetic fields. Commonly, transformers are used to increase or decrease the voltages of alternating current in electric power applications.

WIFI MODULE

In an automatic watering system, a Wi-Fi module is added for improved usability and remote control capabilities. The system can connect to a wireless network and offer one of the advantages such as:

- i. Remote Monitoring and Control: Users can access a mobile app to view real-time sensor data (soil moisture, temperature, humidity), modify watering schedules remotely, and potentially even receive alerts for low water levels or system malfunctions.

REMOTE POWER SUPPLY

A remote power supply in an automatic crop watering system is a critical component that ensures continuous operation without the need for manual intervention or reliance on grid electricity. It typically consists of a rechargeable battery pack or a solar panel coupled with a battery to provide the necessary power to the system.

The remote power supply serves multiple purposes in such systems. Firstly, it powers the sensors responsible for monitoring soil moisture levels, weather conditions, or other parameters relevant to determining when irrigation is needed. Secondly, it powers the actuators or pumps responsible for dispensing water to the crops based on the input from the sensors. Finally, it may also power communication modules like Wi-Fi or cellular connectivity to enable remote monitoring and control of the watering system.

Using a remote power supply offers several advantages. It provides autonomy to the watering system, allowing it to operate in remote or off-grid locations where access to electricity may be limited or unreliable. Additionally, it promotes sustainability by utilizing renewable energy sources such as solar power, reducing the system's environmental impact and operating costs over time. Furthermore, it

enhances scalability and flexibility by enabling the deployment of watering systems in diverse agricultural settings without being constrained by the availability of power infrastructure.

2.2.4 USABILITY

Imagine an intuitive interface with clear controls for setting watering schedules and monitoring plant health - that's what good usability is about in this system.

2.2.5 RELIABILITY

You can trust this system to deliver water based on sensor readings, without random malfunctions. Consistent operation is key.

2.2.6 PERFORMANCE

Delivering the right amount of water, adapting to different plant needs, and reacting quickly to changes in soil moisture - that's all about performance.

2.2.7 SUPPORTABILITY

Easy-to-understand manuals, readily available replacement parts, and helpful customer support ensure you can get help when needed.

2.2.8 PHYSICAL ENVIRONMENT

The system should be built to handle the expected temperatures, humidity, and water pressure in your location.

2.2.9 SECURITY REQUIREMENTS

While security isn't a major concern, some systems might require access controls or secure data practices.

2.2.10 RESOURCE REQUIREMENTS

Low power consumption, efficient water usage, and reliable internet connection (for some features) are important resource considerations.

3.SYSTEM ANALYSIS

3.1 INTRODUCTION

The automatic watering system analyzed here utilizes sensors and a microcontroller to deliver water efficiently to plants. Soil moisture sensors are the core, constantly monitoring moisture levels and triggering the Arduino (microcontroller) to activate the water pump when needed. The Arduino's programmability allows for customization based on plant types and user preferences. Optionally, a DHT11 sensor can provide temperature and humidity data, enabling adjustments for environmental factors. Reliability is a key aspect, with the system functioning consistently based on sensor readings and programmed schedules. Performance is measured by efficient water delivery based on real-time needs and the system's ability to adapt to different watering requirements. User-friendliness is emphasized through a simple interface and the ability to customize settings. Supportability comes from clear documentation and readily available replacement parts. The physical environment needs to be considered, ensuring the system operates within expected temperature and humidity ranges. Security is a minor concern, but some systems might require access controls for remote features. Finally, resource requirements include low power consumption and efficient water usage, with Wi-Fi connectivity being an optional feature for remote monitoring and advanced functionalities. This system analysis highlights the key aspects that contribute to an effective and user-friendly automatic watering solution for plant care.

3.2 USECASES

A use case for the automatic watering system describes a specific situation where it would be beneficial. Imagine you have a thriving vegetable garden, but your busy schedule makes consistent watering difficult. This system acts as a use case solution. Sensors monitor the soil moisture, and the system automatically delivers water when needed, ensuring your plants stay healthy without relying on your memory or time. From houseplants to large farms, any scenario where precise and automated irrigation can benefit plant growth is a use case for this system.

3.2.1 ACTORS

Human Users: These are the individuals who set up, operate, and benefit from the system. This could include homeowners caring for houseplants, gardeners managing vegetable patches, or professional farmers overseeing large-scale irrigation.

Water Pump Motor: This actor is responsible for the physical delivery of water. Based on signals from the Arduino, the water pump motor turns on and draws water from the source to deliver it to the plants.

3.2.2 LIST OF USE CASES

The automatic watering system we discussed has a variety of use cases, both domestic and professional:

Domestic Applications:

Houseplants and Indoor Gardens: Ensure consistent moisture levels for your favorite houseplants, herbs, or indoor vegetable gardens, preventing underwatering or overwatering.

Outdoor Container Gardens: Simplify watering for potted plants on balconies, patios, or decks, giving them the right amount of water regardless of your schedule.

Small Vegetable Gardens: Maintain optimal moisture for your vegetable patch, promoting healthy growth and potentially higher yields.

Rooftop Gardens: Provide efficient irrigation for rooftop gardens, maximizing water usage and plant health in a limited space.

Lawns and Sprinkler Systems: Automate sprinkler systems for your lawn, eliminating the need for manual control and ensuring even watering.

Professional Applications:

Greenhouses and Nurseries: Precisely manage irrigation for a variety of plants in greenhouses and nurseries, optimizing water usage and ensuring consistent growth conditions.

Large-Scale Farms: Implement efficient irrigation systems for extensive agricultural fields, saving water and manpower while promoting crop yields.

Hydroponic Systems: Provide precise control over water delivery in hydroponic setups, ensuring optimal nutrient and moisture levels for plants grown without soil.

3.2.3 USECASE DIAGRAMS

A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses. The actors are often shown as stick figures.

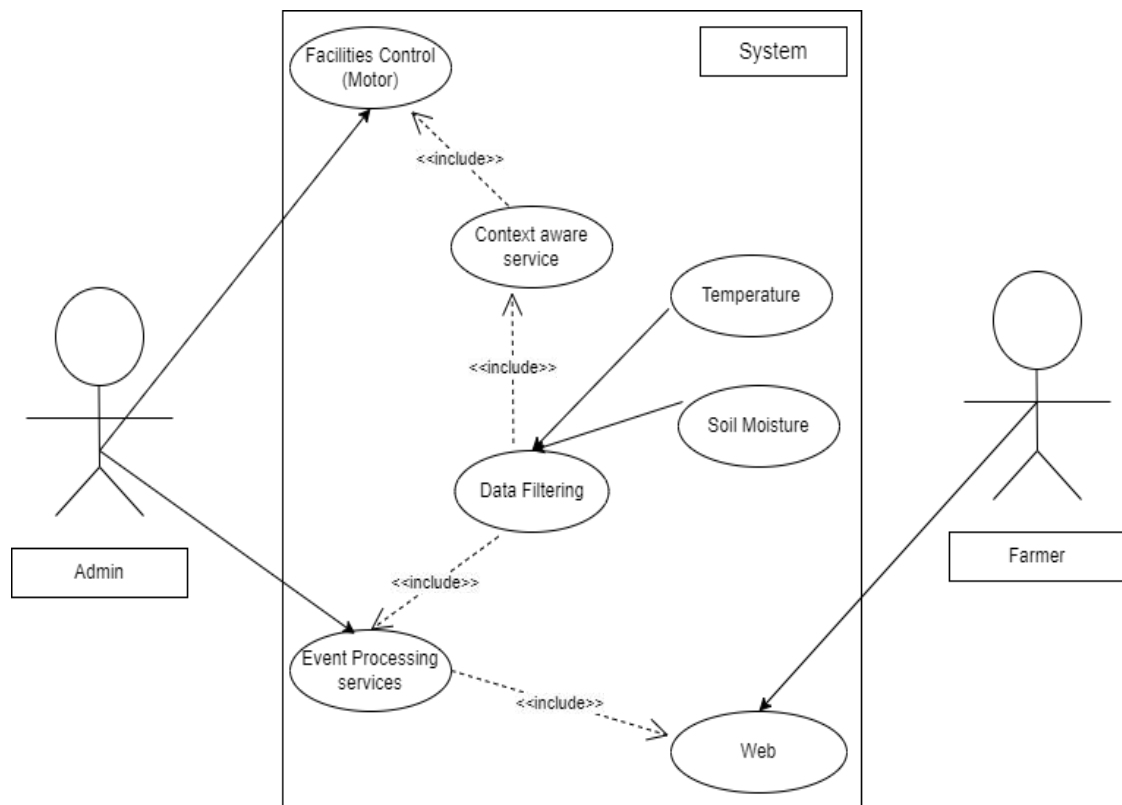


Figure 3.1: Use Case Diagram

4.SYSTEM DESIGN

4.1 INTRODUCTION

The automatic watering system can be designed using an Arduino Uno microcontroller as the central brain. Soil moisture sensors constantly monitor root zone moisture, sending data to the Arduino. Based on pre-programmed schedules and sensor readings, the Arduino triggers a water pump motor to deliver water through drip lines or sprinklers. The system can be scaled to fit various needs, and include a DHT11 sensor to consider temperature and humidity for adjustments. This design offers an efficient and automated way to manage plant irrigation.

4.2 SYSTEM ARCHITECTURE

The system architecture of an automatic crop watering system typically consists of several interconnected components designed to monitor, process, and act upon environmental data to efficiently irrigate crops.

At the core of the architecture are the sensors, which collect data on soil moisture levels, weather conditions, temperature, and possibly other relevant parameters. These sensors feed data into a microcontroller or a central processing unit (CPU), which serves as the brain of the system. The microcontroller processes the sensor data and determines when irrigation is needed based on predefined thresholds or algorithms.

Once the decision to irrigate is made, the microcontroller triggers actuators such as solenoid valves or pumps to deliver water to the crops. These actuators are controlled either directly by the microcontroller or through intermediate relay modules.

Additionally, the system may incorporate a remote monitoring and control module, enabling users to access and manage the watering system remotely via a web interface or a mobile application. This module may utilize Wi-Fi, Bluetooth, or cellular connectivity to communicate with the microcontroller and provide real-time updates on system status and performance.

Overall, the architecture of an automatic crop watering system is designed to efficiently manage water resources, optimize crop growth, and minimize manual intervention, thereby enhancing agricultural productivity and sustainability.

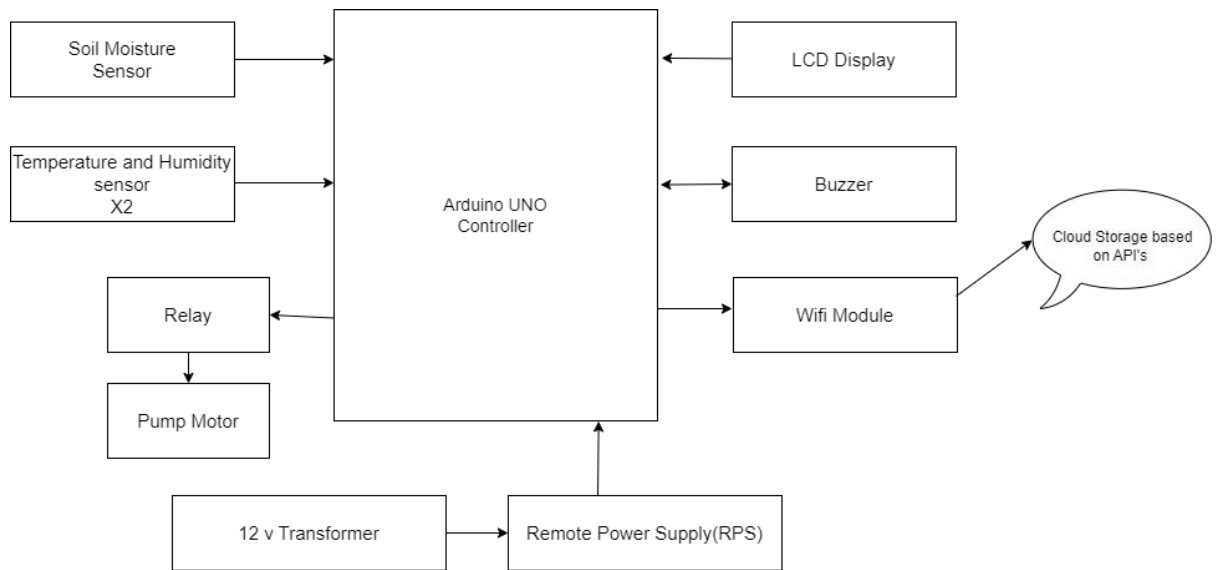


Figure 4.1: System Architecture

4.3 SYSTEM OBJECT MODEL

4.3.1 INTRODUCTION

A system object model refers to a representation of the objects within a system and their relationships. The system object model would outline how these objects interact and connect with each other. It helps developers understand the structure and behavior of the system, enabling them to design, analyze, and modify it effectively. Essentially, it's like a blueprint that guides the creation and management of complex systems.

System object model was intended to be used as a solution to many of the interoperability and reuse problem that occur while sharing class libraries between object- oriented languages.

SOM was designed to be used across IBM's mainframe computers and desktops. It serves as an object-oriented model that can be distinguished from other models contains in object-oriented programming languages. SOM basically includes an interface definition a runtime environment with procedure calls and a set of enabling frameworks.

4.3.2 SUBSYSTEMS

The system comprises several interconnected subsystems, each responsible for specific functionalities and operations. These subsystems work collaboratively to achieve the overarching objectives of the system. Key subsystems within the system object model include:

1.Sensor Subsystem: This subsystem is responsible for collecting data about the environment and plant needs. The primary component is the:

i).Soil Moisture Sensor: Continuously monitors moisture levels in the root zone, sending data to the controller.

ii).DHT11 Component: A DHT11 sensor is added to this subsystem to measure temperature and humidity.

2.Control Subsystem: This is the brain of the system, processing sensor data and making decisions based on programmed logic.

i).Microcontroller (Arduino Uno): Receives data from sensors, compares it to programmed settings, and triggers the water pump accordingly.

3.Water Delivery Subsystem: This subsystem delivers water to the plants based on the controller's instructions.

i).Water Pump Motor: Activated by the controller, it draws water from the source and pumps it through the delivery system.

4.4 OBJECT DESCRIPTION

An object model is a conceptual representation of the entities or objects within a system, along with their attributes and behaviors. It serves as a blueprint for designing and understanding complex systems in software engineering and other disciplines.

In software development, an object model typically consists of classes and objects. A class defines the blueprint for creating objects, specifying their properties (attributes) and behaviors (methods). Objects are instances of classes, representing specific instances or occurrences of the entities described by the class.

4.4.1 OBJECTS

In the system's object-oriented design, various objects encapsulate data and behavior relevant to specific entities within the system. These objects interact with each other to perform tasks and achieve system functionalities.

The objects of an automatic crop watering system include sensors for monitoring soil moisture, weather conditions, and other environmental factors. Actuators such as solenoid valves or pumps are utilized to control the flow of water to the crops based on data from the sensors. A microcontroller or central processing unit processes the sensor data and triggers the actuators accordingly. Additionally, the system may incorporate a remote monitoring and control module for users to manage the watering system remotely. Overall, these components work together to automate the irrigation process, optimize water usage, and promote efficient crop growth.

4.4.2 CLASS DIAGRAM

Class diagrams are a type of static structure diagram in Unified Modeling Language (UML) used to visually represent the structure of a system or software application. They depict the classes, attributes, methods, and relationships between classes within the system.

In a class diagram, classes are represented as boxes with three sections: the top section contains the class name, the middle section lists the class attributes, and the bottom section displays the class methods. Arrows indicate the relationships between classes, such as associations, aggregations, or inheritances.

A class diagram is an illustration of the relationships and source code dependencies among classes in the Unified Modelling Language. In this context, a class defines the methods and variables in an object, which is a specific entity in a program. The unit of code representing that entity. Class diagrams are useful in all forms of object-oriented programming (OOP).

In this there are three classes: data owner, data user and cloud server. Cloud activates owner and user. Data owner uploads files and data user view and download files. UML class is a rectangle divided into: class name, attributes and operations. Our class diagram has three kinds of relationships.

- Association
- Aggregation
- Generalization

What does a class diagram show?

- **Classes:** The building blocks of the system, represented by rectangles.
- **Attributes:** The properties or characteristics of each class, listed within the rectangle.
- **Operations (Methods):** The actions that a class can perform, also displayed inside the rectangle.
- **Relationships:** The connections and interactions between classes, depicted using arrows and specific notations.

Benefits of using class diagrams:

- **Clear Communication:** They provide a clear visual representation of the system's design, promoting better communication among developers.
- **Improved Design:** By visualizing the structure, they can help identify potential issues or areas for improvement in the system's design.
- **Documentation:** They serve as documentation for the software, making it easier to understand and maintain in the long run.

Common uses of class diagrams:

- **Early-stage design:** During the initial phases of software development, class diagrams help establish the core structure and relationships between different parts of the system.
- **Code generation:** In some cases, class diagrams can be used to automatically generate code, reducing development time.
- **Understanding existing code:** They can be used to reverse-engineer existing code, providing a visual understanding of its structure.

In essence, class diagrams are like blueprints for software systems, offering a clear view of the building blocks and how they interact. This understanding is crucial for designing, developing, and maintaining complex software applications.

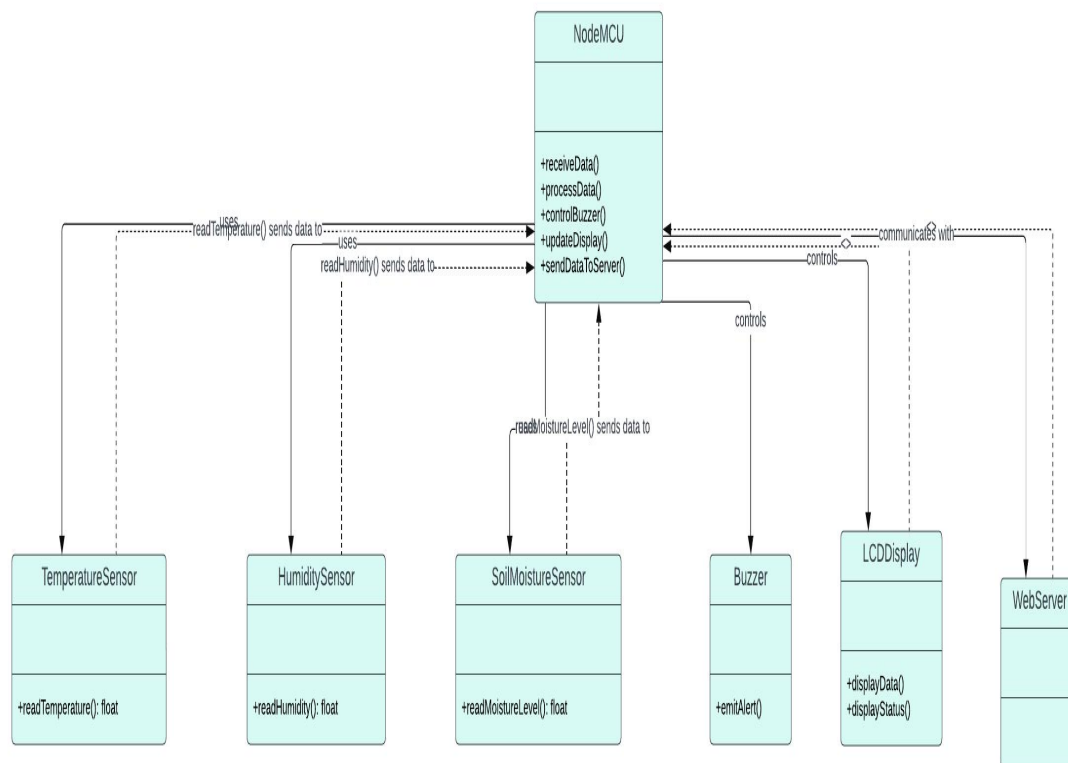


Figure 4.2: Class Diagram

4.5 OBJECT COLLABORATION

Object collaboration refers to the interactions and communication between objects within a software system. It involves objects working together to achieve a specific task or functionality by sending messages to each other. Object collaboration is essential for building complex systems where different objects need to cooperate to accomplish tasks.

During object collaboration, objects may pass messages to request information, invoke methods, or delegate responsibilities to other objects. These messages trigger actions or behavior within the receiving objects, which may involve modifying

internal state, performing calculations, or coordinating with other objects.

4.5.1 OBJECT COLLABORATION DIAGRAM

An object collaboration diagram, also known as a communication diagram, is a type of UML (Unified Modeling Language) diagram used to visualize how objects interact with each other in a system to achieve a specific task or scenario.

Here's a breakdown of what an object collaboration diagram is and what it shows:

1. **Focus on Object Interactions:** Unlike sequence diagrams that focus on the message flow, collaboration diagrams depict the relationships between objects.
2. **Objects and Collaborations:** The diagram represents objects as rectangles with their class name and name (separated by a colon). Arrows connect the objects, representing the messages they send to each other to collaborate and fulfill a particular functionality.

Why use Object Collaboration Diagrams?

Object collaboration diagrams are beneficial for several reasons, including:

- **Modeling Interactions:** They visually represent how objects interact, making it easier to understand how a system works.
- **Clarifying Requirements:** By depicting object interactions for specific functionalities, they help clarify system requirements.
- **Designing Communication Patterns:** They aid in designing communication patterns between objects, including the sequence of messages exchanged.

When to use Object Collaboration Diagrams?

Collaboration diagrams are typically used in the early stages of software development for:

- **Understanding System Behavior:** They help visualize how objects collaborate to achieve specific behaviors within a system.
- **Defining Object Roles:** By showing interactions, they aid in defining the roles and responsibilities of objects in the system.

In essence, object collaboration diagrams are a valuable tool for understanding how objects work together to bring a system to life.

Object collaboration diagrams for an automatic crop watering system depict the interactions and communication between objects involved in the irrigation process. These diagrams illustrate how objects such as sensors, actuators, controllers, and communication modules collaborate to monitor soil moisture levels, analyze data, and regulate water flow to the crops. They showcase the exchange of messages between objects, indicating the flow of control and data within the system. Object collaboration diagrams provide a visual representation of how the various components of the automatic crop watering system work together to automate irrigation and optimize crop growth.

- The diagram focuses on the core functionality of irrigation control.
- You can extend it to include additional objects like a RainSensor that can influence irrigation decisions or a UserInterface for manual control.
- The specific messages and decisions might vary depending on the complexity of the system.

This collaboration diagram provides a visual understanding of how these objects work together to automate the irrigation process based on sensor data and pre-defined rules.

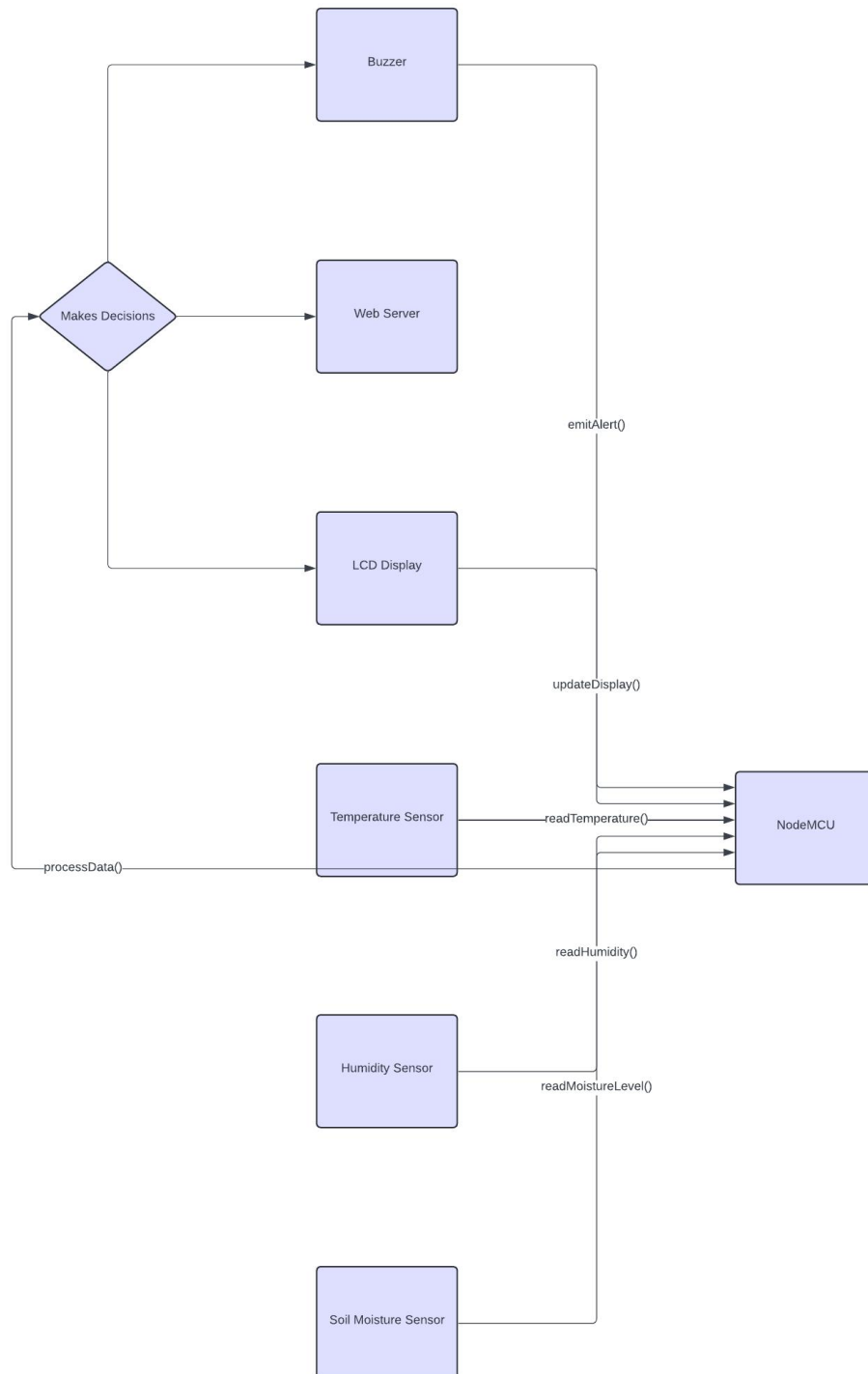


Figure 4.3: Object Collaboration Diagram

4.6 DYNAMIC MODEL

4.6.1 SEQUENCE DIAGRAM

Sequence diagrams for an automatic crop watering system illustrate the chronological flow of messages exchanged between objects during the irrigation process. They depict the sequence of interactions between objects such as sensors, controllers, actuators, and communication modules, showing how they collaborate to automate irrigation tasks.

In a sequence diagram for an automatic crop watering system, each vertical lifeline represents an object, and arrows between lifelines indicate messages being passed between objects. The diagram illustrates the order in which messages are sent and received, along with any conditions or loops involved in the process.

Sequence diagrams capture the dynamic behavior of the system. Sequence diagrams help developers visualize and understand the communication flow within the automatic crop watering system, facilitating effective implementation and maintenance.

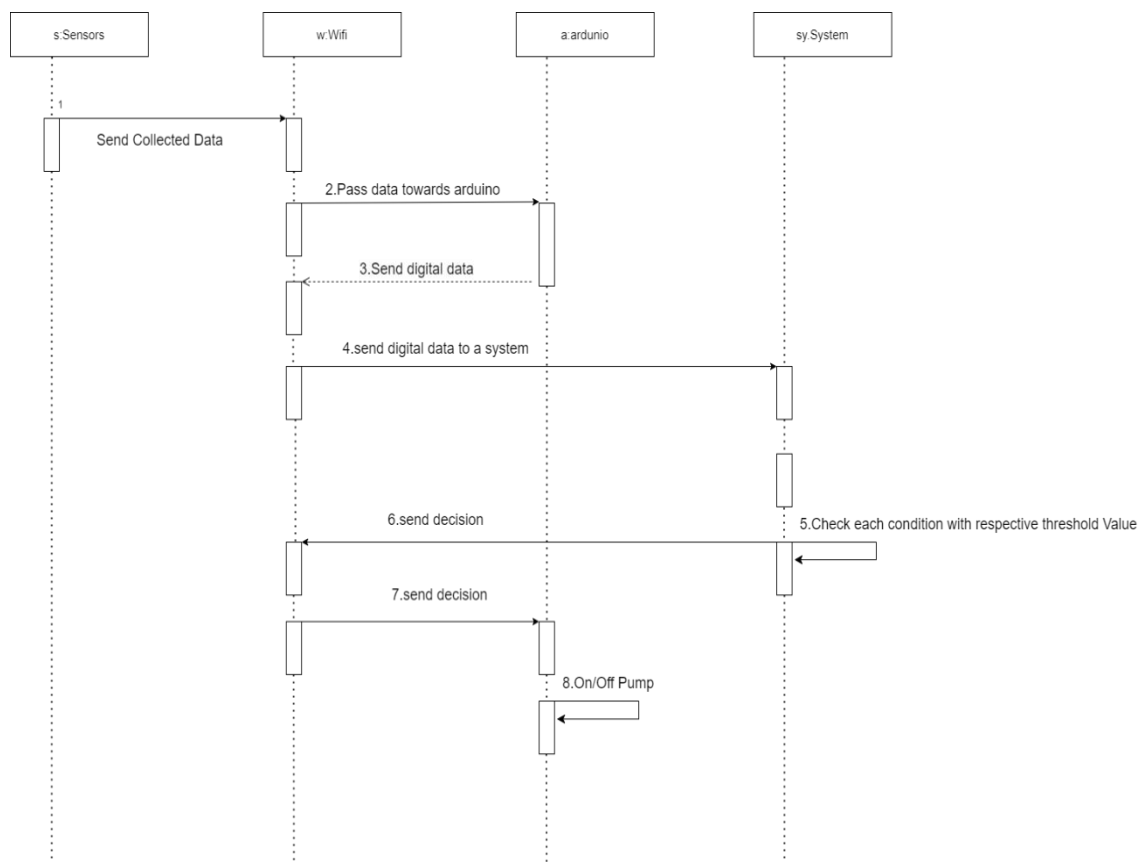


Figure 4.4: Sequence Diagram

4.6.2 ACTIVITY DIAGRAM

Activity diagrams for an automatic crop watering system represent the workflow or sequence of activities involved in the irrigation process. They illustrate the steps and decision points from the initiation of the watering system to the completion of irrigation tasks.

In an activity diagram for an automatic crop watering system, nodes represent different activities or actions, while edges denote the flow of control between them. Nodes can represent actions such as sensor readings, data processing, decision-making, and actuator control. Decision points are depicted using diamond-shaped nodes, indicating branches in the workflow based on conditions or events.

Activity diagrams provide a visual representation of the system's behavior, showing how activities are sequenced, parallelized, or repeated during the irrigation process. They help in understanding the overall logic of the system. Activity diagrams serve as valuable tools for system design, analysis, and documentation in the development of automatic crop watering systems.

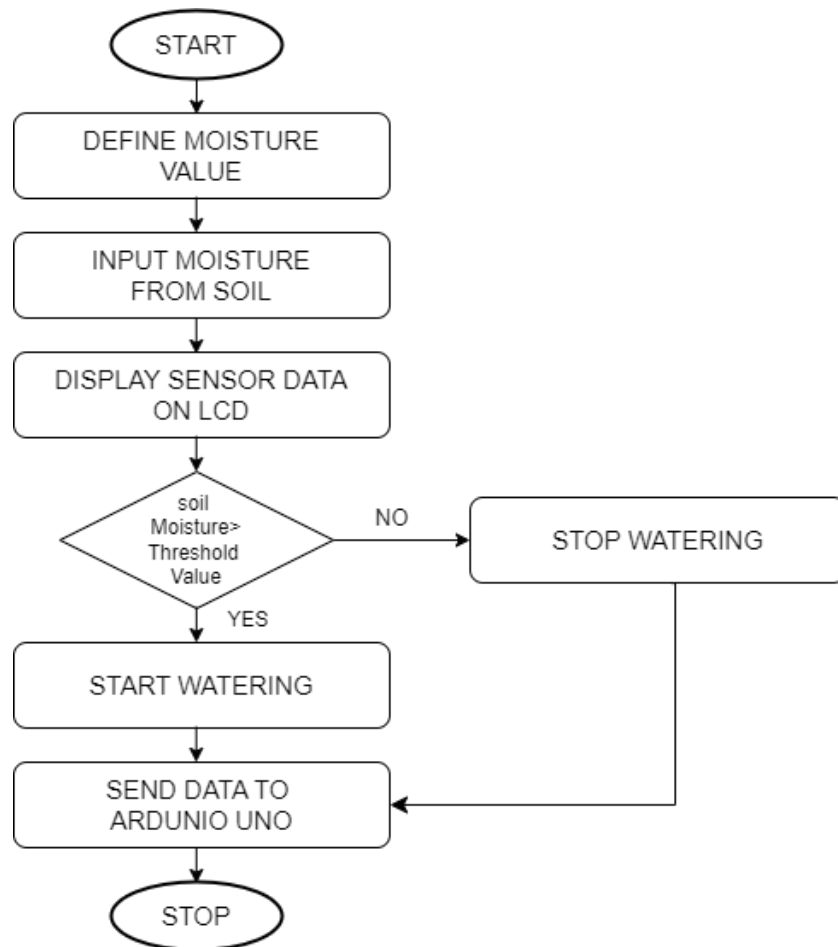


Figure 4.5: Activity Diagram

4.7 DATABASE DESIGN

Database design for IoT projects involves designing the structure and organization of databases to efficiently store, manage, and retrieve data generated by Internet of Things devices. It encompasses defining the schema, tables, relationships, and indexing strategies tailored to the specific requirements of the IoT application. Given the vast volume, variety, and velocity of data generated by IoT devices, database design focuses on scalability, performance, and flexibility. It involves considerations such as data modeling techniques to represent IoT data streams, choosing appropriate database technologies capable of handling high-throughput data ingestion and real-time analytics, and ensuring data integrity, security, and privacy compliance. Additionally, database design for IoT projects often involves implementing strategies for data aggregation, transformation, and archival to optimize storage and processing efficiency. Effective database design is crucial for enabling reliable, efficient, and scalable IoT solutions that can harness the full potential of sensor data to derive actionable insights and support decision-making in various domains.

It typically involves several stages, including conceptual modeling, logical design, and physical implementation, using techniques such as entity-relationship modeling, normalization, and indexing to optimize data storage and access. Overall, database design is critical for building reliable and scalable systems that can meet the needs of users and applications efficiently

4.7.1 ENTITY-RELATIONSHIP DIAGRAM

An Entity-Relationship (ER) diagram is a visual representation of the entities, attributes, and relationships within a database system. It depicts how different entities are related to each other and the attributes associated with each entity. ER diagrams use symbols such as rectangles for entities, ovals for attributes, and lines to represent relationships between entities. They provide a clear and concise way to understand the structure of a database, aiding in the design, communication, and documentation of database systems.

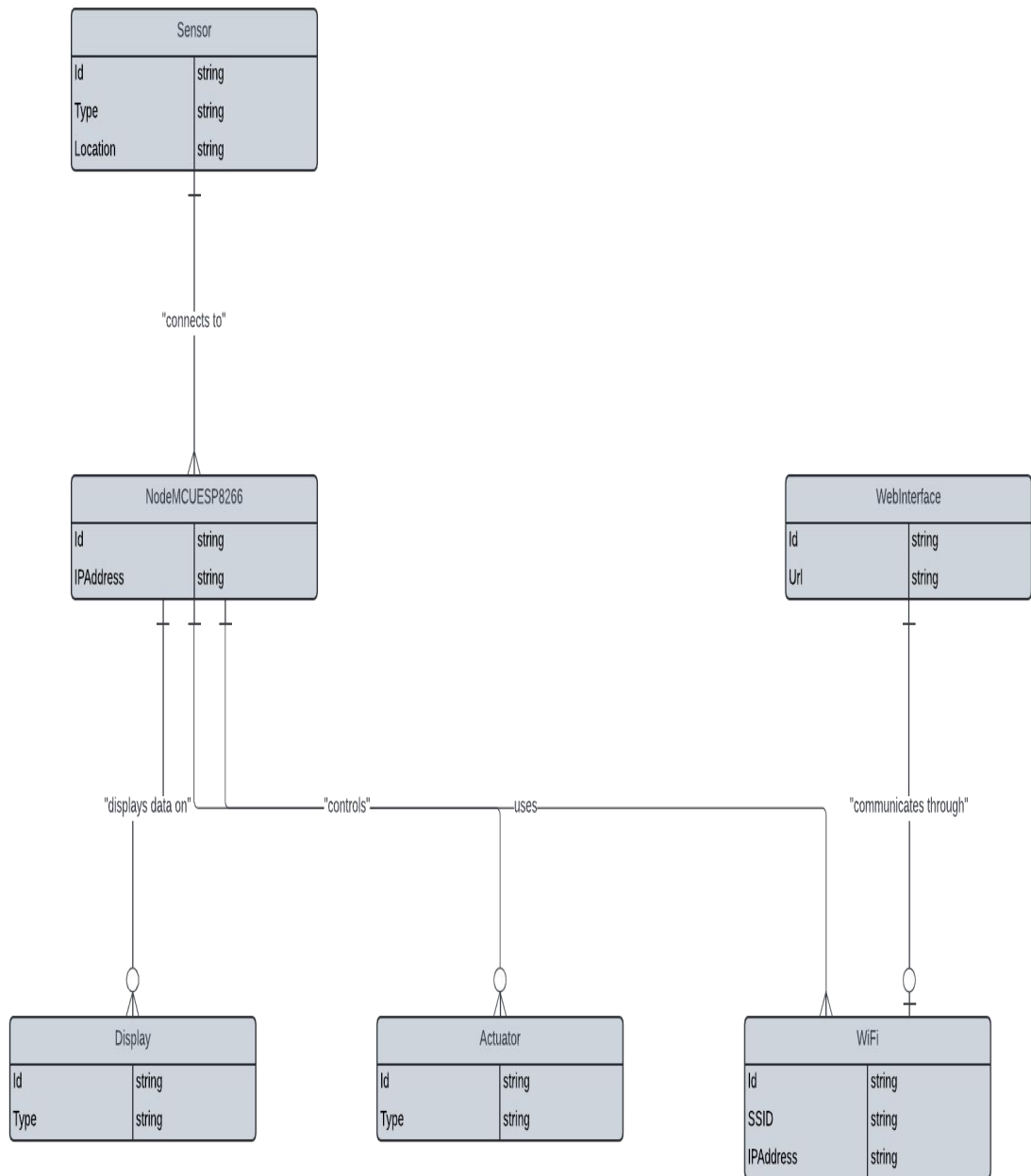


Figure 4.6: Entity Relationship Diagram

4.7.2 DATABASE TABLE

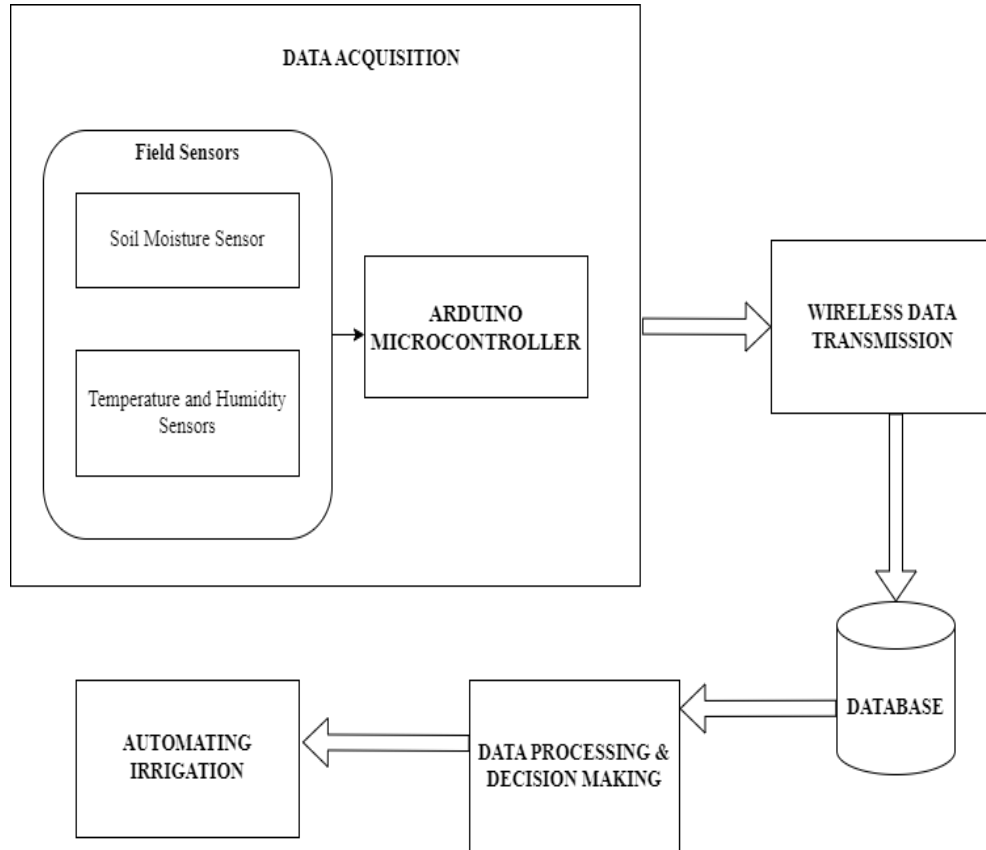


Figure 4.7: Process Design of Automatic Irrigation

The automatic crop watering system with cloud storage can benefit from several database tables to effectively manage sensor data and irrigation control. Here are the main tables to consider:

1.Sensor Data:

- **id (primary key):** Unique identifier for each data entry.
- **sensor_type (text):** Specifies the type of sensor (e.g., "soil_moisture", "temperature", "humidity").
- **timestamp (datetime):** Date and time the data was collected.
- **value (float):** Numeric value representing the sensor reading.

2.Crop Information:

- **id (primary key):** Unique identifier for each crop type.
- **crop_name (text):** Name of the crop being monitored.
- **ideal_moisture_range (text):** Preferred soil moisture range for the crop (e.g., "30%-50%").
- **watering_frequency (text):** Typical watering schedule for the crop (e.g., "Daily", "Every other day").

3.Irrigation Log:

- **id (primary key):** Unique identifier for each irrigation event.
- **timestamp (datetime):** Date and time of the irrigation cycle.
- **duration (integer):** Duration of the watering cycle in minutes (seconds, etc.).
- **triggered_by (text):** Indicates what triggered the irrigation (e.g., "Manual", "Automated rule").

Here's how these tables work together:

1. The SensorData table stores all the readings from the soil moisture, temperature, and humidity sensors.
2. The CropInformation table (optional) can be used to define ideal watering conditions for different crops, allowing the system to make informed decisions based on sensor data and crop type.
3. The IrrigationLog table (optional) keeps track of past irrigation events, providing a record of watering activity.

By utilizing these database tables, the system can effectively manage sensor data, make informed irrigation decisions (potentially automated based on pre-defined rules), and maintain a history of watering events for analysis and future reference.

Database tables represent structured collections of data related to various aspects of the system's operation. These tables store information such as sensor readings, irrigation schedules, historical data, and system configurations. Each table typically corresponds to a specific entity or concept within the watering system, and

its columns define the different attributes or properties associated with that entity.

There will be a "SensorData" table to store readings from soil moisture sensors, with columns for timestamp, sensor ID, and moisture level. Another table could be "IrrigationSchedule" to manage the timing and duration of watering sessions, with columns for start time, end time, and targeted area.

Database tables play a crucial role in organizing and managing the data generated by the automatic crop watering system. They facilitate efficient data storage, retrieval, and manipulation, enabling the system to effectively monitor soil conditions, regulate irrigation, and analyze historical trends for optimized crop management.

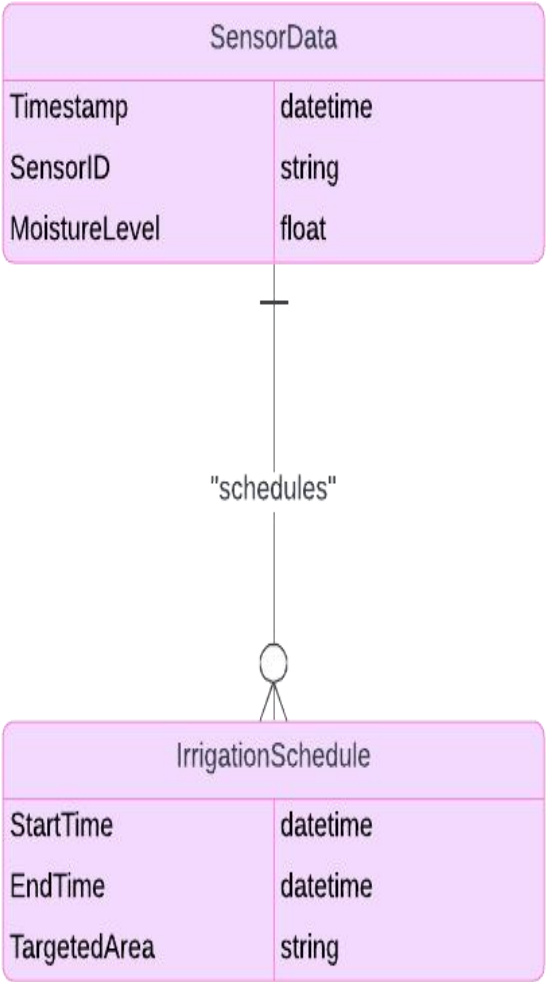


Table 4.1: Database Table

4.8 STATIC MODEL

A static model diagram is a visual representation used in software engineering and system design to illustrate the structure of a system or software application at a specific point in time. It captures the static relationships and dependencies between various components, such as classes, modules, interfaces, and their attributes or properties. These diagrams, such as class diagrams in object-oriented design or component diagrams in component-based development, help stakeholders understand the architecture and organization of the system, facilitating communication and analysis during the design and development process. Static model diagrams provide a blueprint for developers to implement the system and serve as a reference for maintenance and documentation throughout the software lifecycle.

4.8.1 COMPONENT DIAGRAM

Component diagrams for an automatic crop watering system illustrate the physical and logical components of the system and their interconnections. These diagrams provide a visual representation of the system's architecture, showing the individual components and their relationships, dependencies, and interactions.

In a component diagram for an automatic crop watering system, each component represents a modular unit of functionality, such as sensors, actuators, controllers, communication modules, and user interfaces. Components are depicted as rectangles with labeled names and can include interfaces, ports, and dependencies.

The connections between components depict the interactions and dependencies between them, indicating how data and control flow within the system. Component diagrams help in understanding the system's structure, modularization, and component reuse, facilitating system design, analysis, and documentation. They aid developers in identifying potential bottlenecks, optimizing system performance, and ensuring scalability and maintainability in the development of automatic crop watering systems.

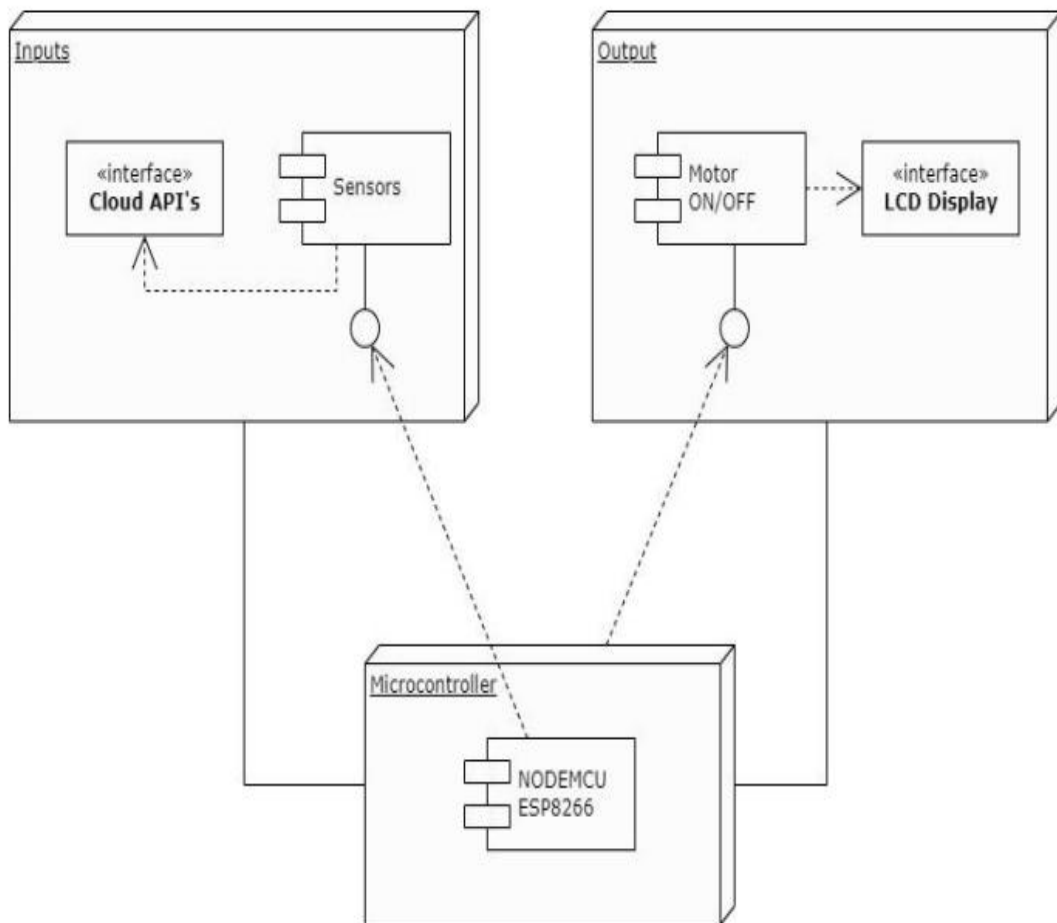


Figure 4.8: Component Diagram

4.8.2 DEPLOYMENT DIAGRAM

Deployment diagrams for an automatic crop watering system depict the physical deployment of software components and hardware devices within the system's environment. These diagrams illustrate how the system's software and hardware elements are distributed across different nodes or devices, including servers, sensors, actuators, and communication modules.

In a deployment diagram for an automatic crop watering system, nodes represent the physical or virtual computing devices where software components are

deployed. Software components, represented by rectangles, are deployed onto these nodes, showcasing their placement and relationships within the system architecture.

Deployment diagrams help in visualizing the system's physical layout, facilitating understanding of its distributed nature and resource allocation.

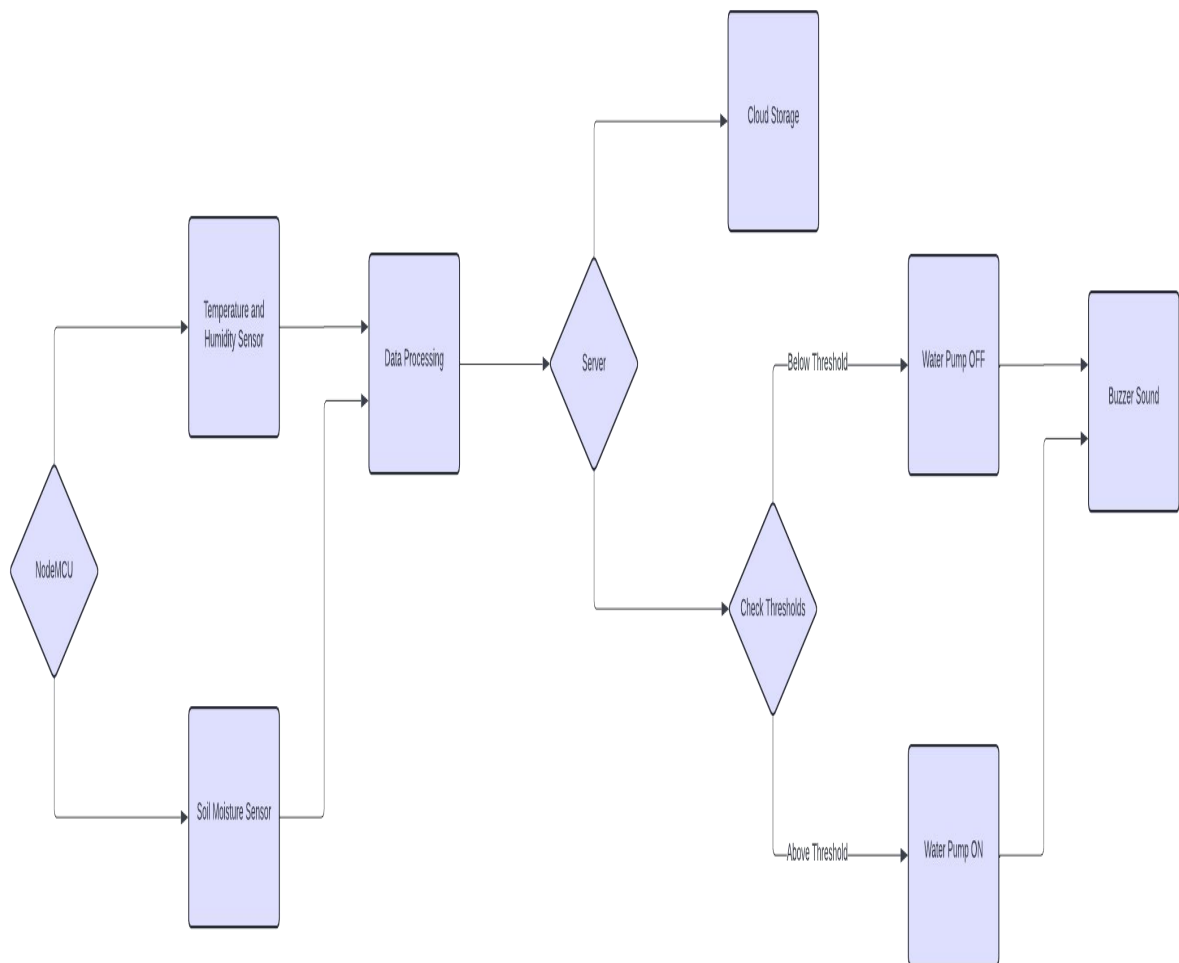


Figure 4.9: Deployment Diagram

5.IMPLEMENTATION

5.1 SOFTWARE USED

After downloading, installing, and testing the Arduino software available at [Arduino.cc](https://www.arduino.cc) (click on the Download link near the top of the page. This program is known as the Arduino IDE - short for Integrated Development Environment. Before you jump to the page for your operating system, make sure you've got all the right equipment. The programming environment outlined in this document is provided by [Arduino.cc](https://www.arduino.cc) free of charge and is the recommended programming environment for your ArduinoKit.

What you will need:

- A computer (Windows, Mac, or Linux)
- An Arduino-compatible microcontroller (DuinoKits use Arduino NANO w/ ATmega328 chip)
- A USB cable is required to connect the NANO microprocessor to your computer for programming

Download the Arduino Software (IDE)

- Get the latest version from the download page. You can choose between the Installer (.exe) and the Zip packages. We suggest you use the first one that installs directly everything you need to use the Arduino Software (IDE), including the drivers. With the Zip package you need to install the drivers manually. The Zip file is also useful if you want to create a portable installation.
- When the download finishes, proceed with the installation and please allow the driver installation process when you get a warning from the operating system

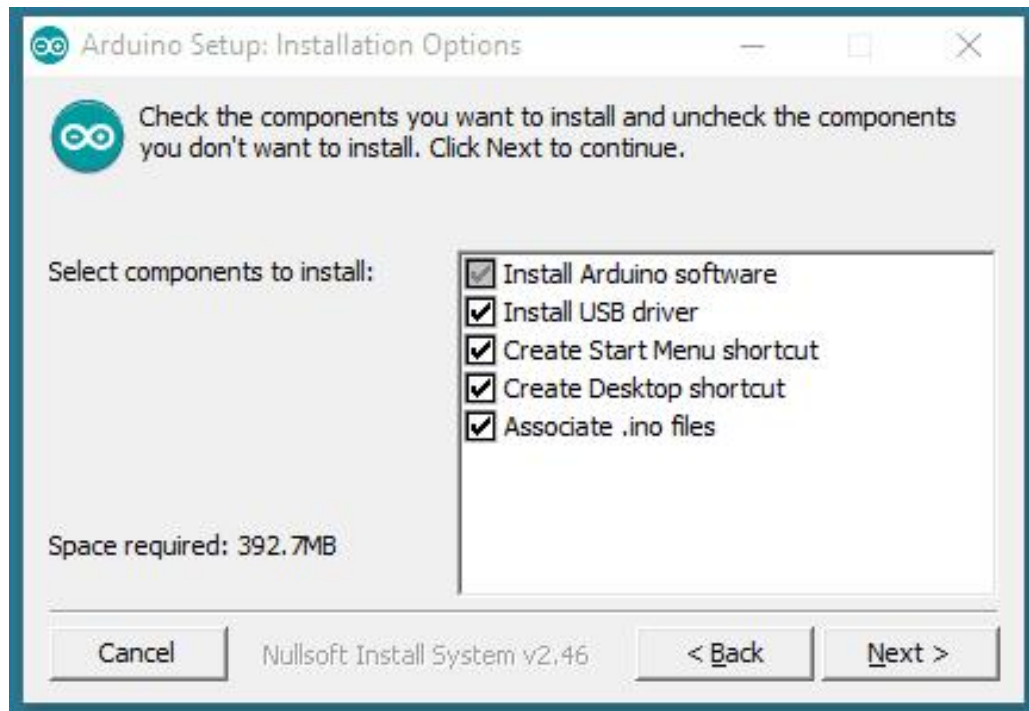


Figure 5.1: Choose the components to install.

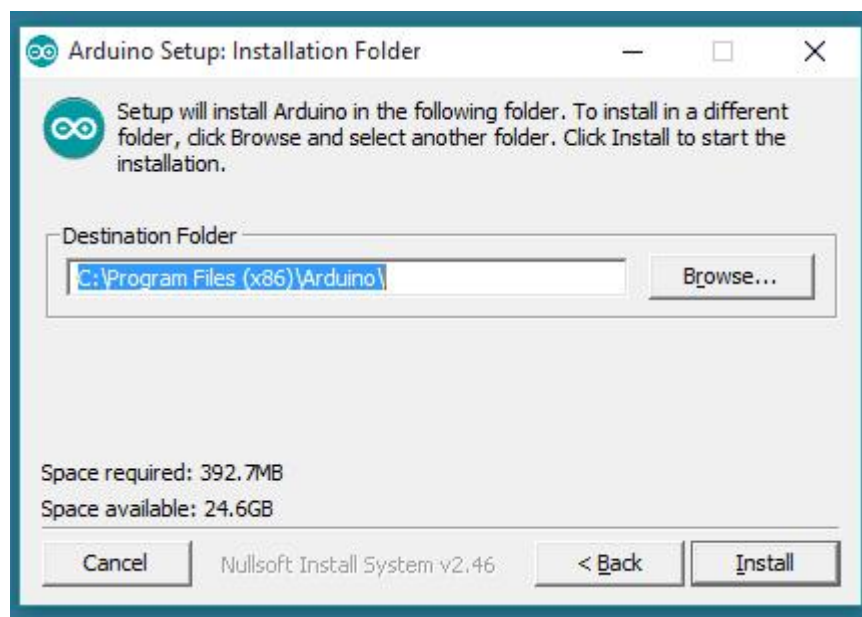


Figure 5.2: Choose the installation directory.

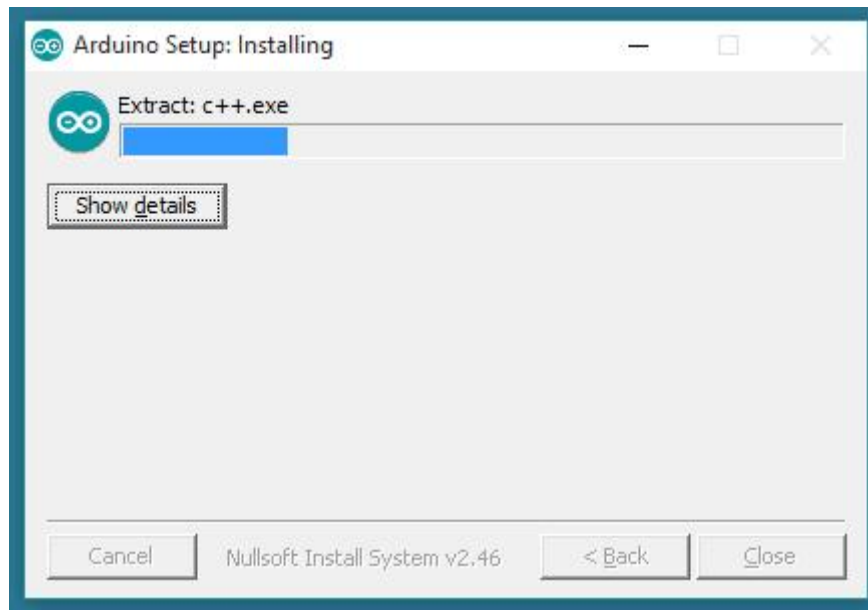


Figure 5.3: Installation in progress.

1. The process will extract and install all the required files to execute properly the Arduino Software (IDE)
2. The text of the Arduino getting started guide is licensed under a Creative Commons Attribution-ShareAlike 3.0 License. Code samples in the guide are released into the public domain.

CONFIGURING THE ARDUINO IDE

The next thing to do is to make sure the software is set up for your particular Arduino board. Go to the “Tools” drop-down menu, and find “Board”. Another menu will appear, where you can select from a list of Arduino models. I have the Arduino Uno R3, so I chose “Arduino Uno”.

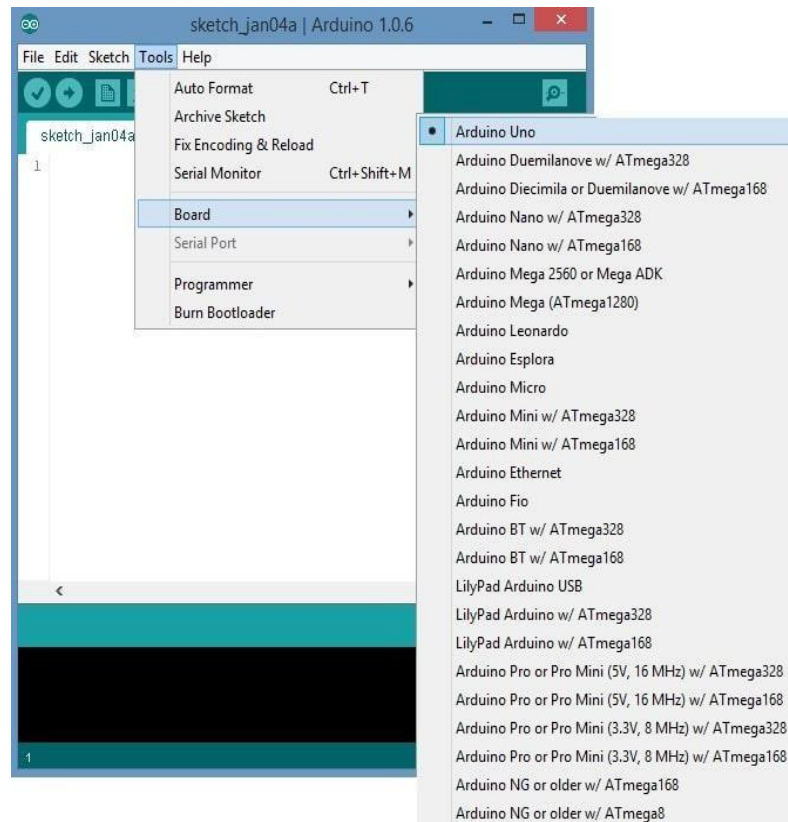


Figure 5.4: Configuring the Arduino IDE

EXPLORING THE ARDUINO IDE

If you want, take a minute to browse through the different menus in the IDE. There is a good variety of example programs that come with the IDE in the “Examples” menu. These will help you get started with your Arduino right away without having to do lots of research.

The Arduino IDE (Integrated Development Environment) is a software application you install on your computer. It provides a user-friendly workspace with several key areas:

1. **Editor:** This is where you write your code, which consists of text instructions that tell the Arduino board what to do.
2. **Text Console:** This window displays messages and feedback when you compile (convert your code) or upload (send your code) to the Arduino board.
3. **Menu Bar and Toolbar:** These provide access to various functionalities like

creating new projects, opening existing ones, uploading code, and debugging errors.

4. **Board Selection and Serial Monitor:** You can choose the specific Arduino board you're using and view data transmitted from the board to your computer in real-time.

Learning by Doing:

The Arduino IDE philosophy is centered around learning through hands-on experience.

Here's what you might encounter as you explore:

- **Example Sketches:** The IDE comes with a library of pre-written code examples (sketches) for various functionalities like blinking an LED, reading sensor data, or controlling motors. These serve as a great starting point to understand basic coding concepts and hardware interactions.
- **Step-by-Step Tutorials:** Numerous online tutorials and resources guide you through building specific projects using Arduino. These tutorials typically involve following instructions, writing code snippets, and uploading them to your board to see the results come to life.

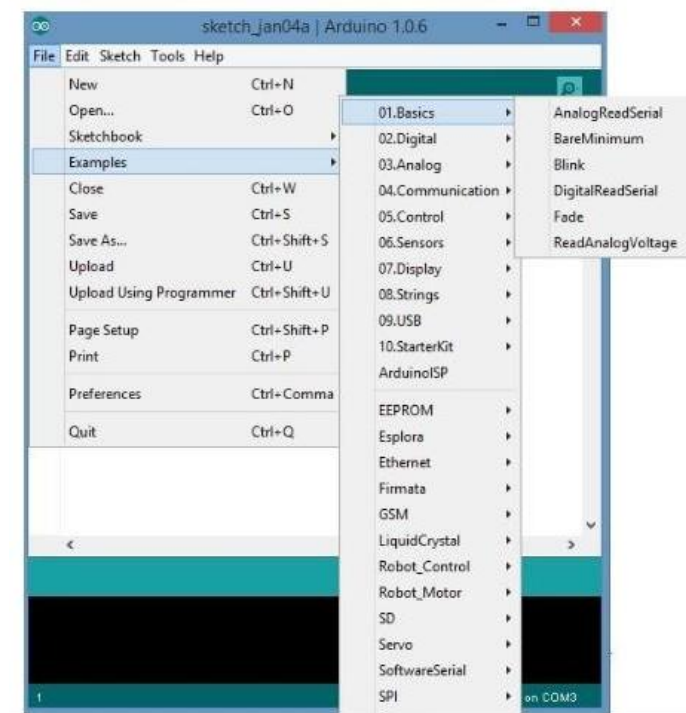


Figure 5.5: Exploring the Arduino IDE

Embedded System

An embedded system is a computer system—a combination of a computer processor, computer memory, and input/output peripheral devices—that has a dedicated function within a larger mechanical or electronic system. It is embedded as part of a complete device often including electrical or electronic hardware and mechanical parts. Because an embedded system typically controls physical operations of the machine that it is embedded within, it often has real-time computing constraints. Embedded systems control many devices in common use today. In 2009 it was estimated that ninety-eight percent of all microprocessors manufactured were used in embedded systems. Modern embedded systems are often based on microcontrollers (i.e. microprocessors with integrated memory and peripheral interfaces), but ordinary microprocessors (using external chips for memory and peripheral interface circuits) are also common, especially in more complex systems. In either case, the processor(s) used may be types ranging from general purpose to those specialized in a certain class of computations, or even custom designed for the application at hand. A common standard class of dedicated processors is the digital signal processor (DSP). Since the embedded system is dedicated to specific tasks, design engineers can optimize it to reduce the size and cost of the product and increase the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale.

Embedded systems range in size from portable personal devices such as digital watches and MP3 players to bigger machines like home appliances, industrial assembly lines, robots, transport vehicles, traffic light controllers, and medical imaging systems. Often they constitute subsystems of other machines like avionics in aircraft and spacecraft. Large installations like factories, pipelines and electrical grids rely on multiple embedded systems networked together. Generalized through software customization, embedded systems such as programmable logic controllers frequently comprise their functional units.

Embedded systems range from those low in complexity, with a single microcontroller chip, to very high with multiple units, peripherals and networks, which may reside in equipment racks or across large geographical areas connected via long- distance communications lines.

Applications of Embedded System

- Embedded systems are commonly found in consumer, industrial, automotive, home appliances, medical, telecommunication, commercial, and aerospace and military applications.
- Telecommunications systems employ numerous embedded systems from telephone switches for the network to cell phones at the end user. Computer networking uses dedicated routers and network bridges to route data.
- Consumer electronics include MP3 players, television sets, mobile phones, video game consoles, digital cameras, GPS receivers, and printers. Household appliances, such as microwave ovens, washing machines and dishwashers, include embedded systems to provide flexibility, efficiency and features.
- Advanced HVAC systems use networked thermostats to more accurately and efficiently control temperature that can change by time of day and season. Home automation uses wired- and wireless networking that can be used to control lights, climate, security, audio/visual, surveillance, etc., all of which use embedded devices for sensing and controlling.
- Transportation systems from flight to automobiles increasingly use embedded systems. New airplanes contain advanced avionics such as inertial guidance systems and GPS receivers that also have considerable safety requirements. Spacecraft rely on avionics systems for trajectory correction. Various electric motors brushless DC motors, induction motors and DC motors use electronic motor controllers. Automobiles, electric vehicles, and hybrid vehicles increasingly use embedded systems to maximize efficiency and reduce pollution. Other automotive safety systems using embedded systems include anti-lock braking system (ABS), Electronic Stability Control (ESC/ESP), traction control (TCS) and automatic four-wheel drive.
- Medical equipment uses embedded systems for monitoring, and various medical imaging (PET, Single-photon emission computed tomography (SPECT), CT, and MRI) for non-invasive internal inspections. Embedded systems within medical equipment are often powered by industrial computers.
- Embedded systems are used for safety-critical systems in aerospace and defense industries. Unless connected to wired or wireless networks via on-chip 3G cellular or other methods for IoT monitoring and control purposes, these systems

can be isolated from hacking and thus be more secure.[citation needed] For fire safety, the systems can be designed to have a greater ability to handle higher temperatures and continue to operate. In dealing with security, the embedded systems can be self-sufficient and be able to deal with cut electrical and communication systems.

- Miniature wireless devices called motes are networked wireless sensors. Wireless sensor networking makes use of miniaturization made possible by advanced IC design to couple full wireless subsystems to sophisticated sensors, enabling people and companies to measure a myriad of things in the physical world and act on this information through monitoring and control systems. These motes are completely self contained and will typically run off a battery source for years before the batteries need to be changed or charged.Embedded C is a set of language extensions for the C programming language by the C Standards Committee to address commonality issues that exist between C extensions for different embedded systems.

Embedded C

Embedded C is a set of language extensions for the C programming language by the C Standards Committee to address commonality issues that exist between C extensions for different embedded systems.

Embedded C programming typically requires nonstandard extensions to the C language in order to support enhanced microprocessor features such as fixed-point arithmetic, multiple distinct memory banks, and basic I/O operations. The C Standards Committee produced a Technical Report, most recently revised in 2008 and reviewed in 2013, providing a common standard for all implementations to adhere to. It includes a number of features not available in normal C, such as fixed-point arithmetic, named address spaces and basic I/O hardware addressing. Embedded C uses most of the syntax and semantics of standard C, e.g., `main()` function, variable definition, datatype declaration, conditional statements (`if`, `switch case`), loops (`while`, `for`), functions, arrays and strings, structures and union, bit operations, macros, etc.

Embedded C is most popular programming language in software field for developing electronic gadgets. Each processor used in electronic system is associated with embedded software.

Embedded C programming plays a key role in performing specific function by the processor. In day-to-day life we used many electronic devices such as mobile phone, washing machine, digital camera, etc. These all device working is based on microcontroller that are programmed by embedded C.

Let's see the block diagram representation of embedded system programming:

The Embedded C code written in above block diagram is used for blinking the LED connected with Port0 of microcontroller.

In embedded system programming C code is preferred over other language. Due to the following reasons:

- Easy to understand
- High Reliability
- Portability
- Scalability

Basic Declaration

Let's see the block diagram of Embedded C Programming development:

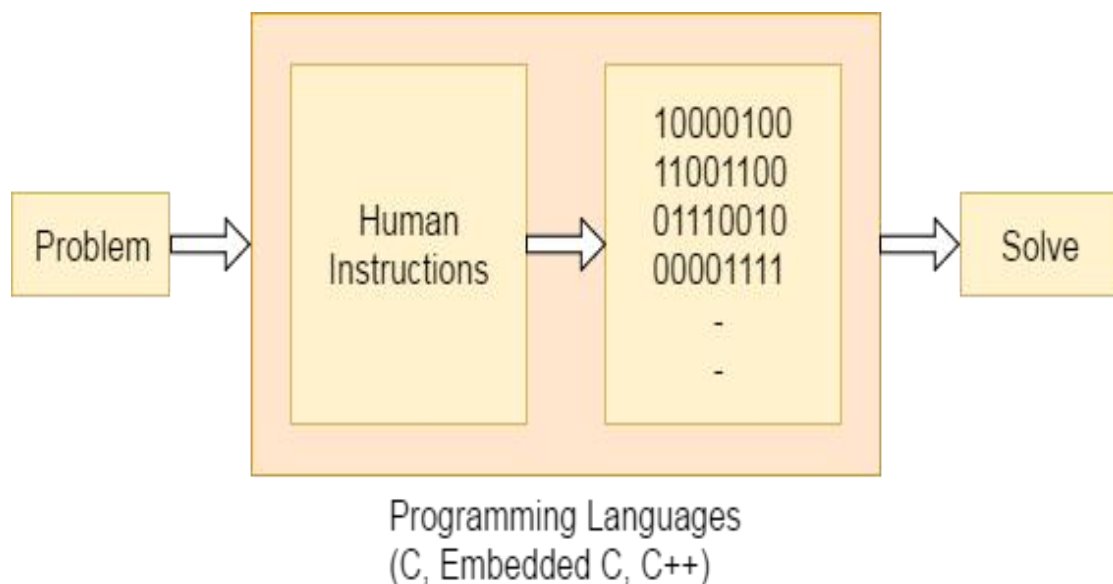


Figure 5.6: Block Diagram of Embedded C

Function is a collection of statements that is used for performing a specific task and a collection of one or more functions is called a programming language. Every language is consisting of basic elements and grammatical rules. The C language programming is designed for function with variables, character set, data types, keywords, expression and so on are used for writing a C program. The extension in C language is known as embedded C programming language. As compared to above the embedded programming in C is also have some additional features like data types, keywords and header file etc.

Basic Embedded C Programming Steps

Let's see the block diagram representation of Embedded C Programming Steps:

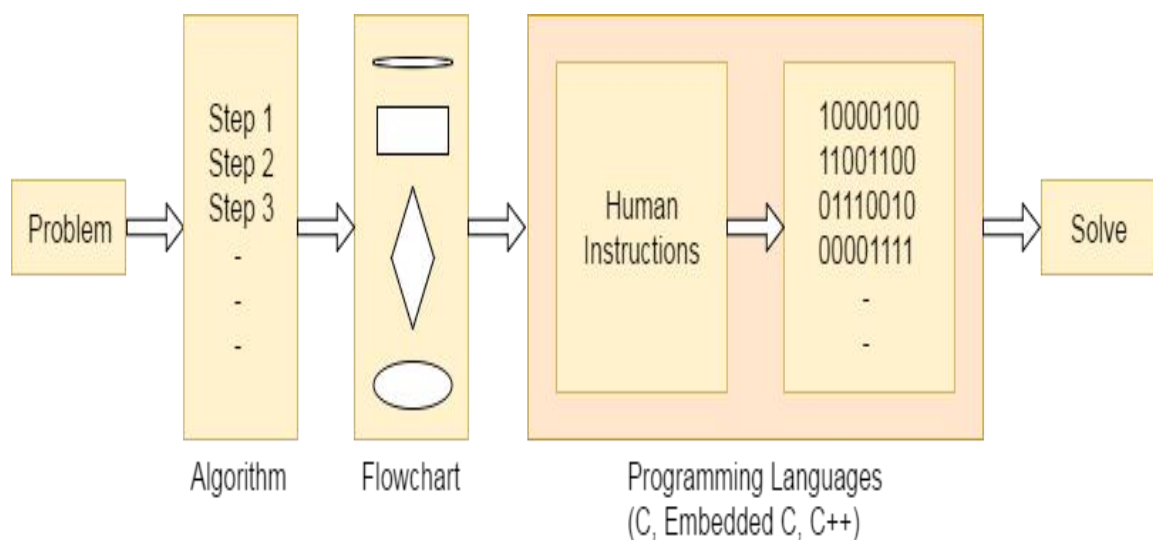


Figure 5.7: Representation of Embedded C programming steps

- The microcontroller programming is different for each type of operating system. Even though there are many operating system are exist such as Windows, Linux, RTOS, etc but RTOS has several advantage for embedded system development.

Flask Framework

Flask, a lightweight Python web framework, shines in machine learning projects by enabling you to deploy your models as web applications or APIs. Here's how Flask comes into play:

Key Uses of Flask:

- **Model Deployment:** Flask provides a simple and efficient way to deploy your trained machine learning models. You can create a web application where users can interact with the model by providing input data and receive predictions as output.
- **API Development:** Flask is excellent for building RESTful APIs (Application Programming Interfaces) that expose your models' functionalities to other applications or systems. This allows for integration with mobile apps, web dashboards, or other software components.
- **Model Serving:** Imagine having a model that predicts customer churn. With Flask, you can develop a web service that takes customer data as input, uses your model to predict churn probability, and returns the results. This can be integrated into a customer relationship management system for proactive actions.

Implementation Steps:

1. **Import Libraries:** You'll import Flask and your machine learning library (e.g., scikit-learn, TensorFlow) into your Python script.
2. **Load the Model:** Load your pre-trained machine learning model using the appropriate library functions.
3. **Define Routes:** Flask uses decorators to define routes (URLs) that map to specific functions in your code.
4. **Create Input Form:** Design an HTML form in your templates directory where users can provide input data for the model.
5. **Handle User Input:** In your Flask routes, write functions that capture user input from the form and preprocess it for your model.

6. **Make Predictions:** Use the loaded model to make predictions on the preprocessed data.
7. **Return Output:** Flask allows you to return the model's predictions as a web page response or JSON data for API consumption.

By leveraging Flask, you can effectively bridge the gap between your machine learning models and the real world, allowing users to interact with your models and benefit from their predictions.

Functionalities of using Flask in our system:

Data Collection:

- 1) Users can enter values for soil moisture, temperature, and humidity through a web interface built with Flask.
- 2) Alternatively, these values can be collected from real-time sensors connected to an Arduino board and uploaded to the web server.

Machine Learning Model:

- 1) A machine learning model will be trained on historical data of soil moisture, temperature, humidity, and the corresponding optimal watering times for your specific crop.
- 2) This model could be a regression model that predicts the time until the next watering cycle based on the input sensor data.

Flask Web Application:

The Flask application will:

Provide a web page for users to input sensor data (or display real-time sensor readings).

Use the trained model to predict the next watering time based on the user input.

Display the predicted watering time on the web page.

Integrate with the hardware components to trigger the water pump/valve for automated watering based on the predicted schedule.

Benefits:

- **Water Conservation:** This system helps optimize water usage by watering plants only when necessary based on real-time data and predictions.
- **Improved Crop Health:** By providing the right amount of water at the right time, you can promote healthy plant growth and potentially increase yield.
- **Remote Monitoring and Control:** The web interface allows you to monitor sensor data and predicted watering schedules remotely.

Implementation Steps:

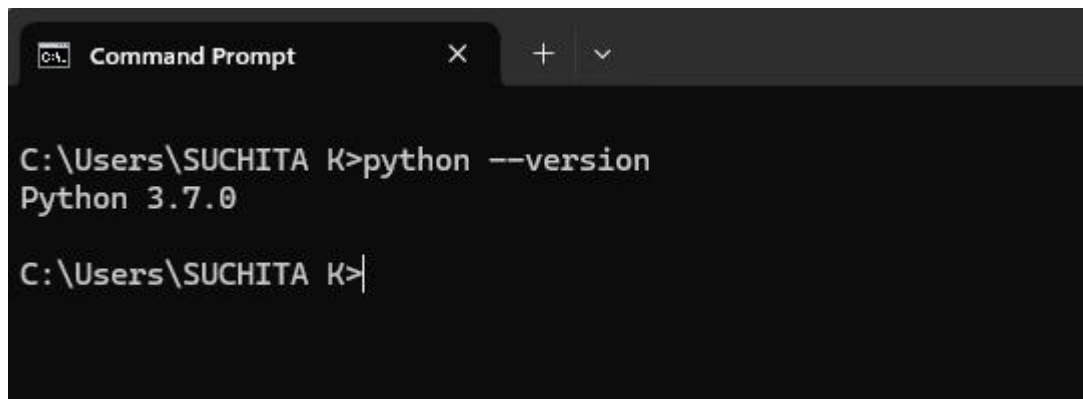
- **Data Collection:** Prepare historical data of sensor readings and corresponding watering times for your crop. Alternatively, develop the sensor data collection part using Arduino.
- **Model Training:** Train a machine learning model using your prepared data. Choose a model suitable for predicting time-based values (e.g., regression model).
- **Flask Application Development:** Build a Flask application with functionalities for user input, model prediction, and displaying results. Optionally, integrate with hardware for automated watering.
- **Deployment:** Deploy the Flask application on a web server to make it accessible remotely.

How to Implement FLASK in VSCode:

Here's a breakdown of how you can use Flask in Visual Studio Code (VS Code) for implementing automatic crop watering system project:

Prerequisites:

Python: Ensure you have Python installed on your system. You can check this by running `python --version` in your terminal.



```
C:\Users\SUCHITA K>python --version
Python 3.7.0
C:\Users\SUCHITA K>
```

Figure 5.8: Checking for version of python in your system

VS Code: Download and install VS Code from the official website <https://code.visualstudio.com/download>.

Download the Visual Studio Code installer for Windows. Once it is downloaded, run the installer (VSCodeUserSetup-{version}.exe). Then, run the file – it will only take a minute.

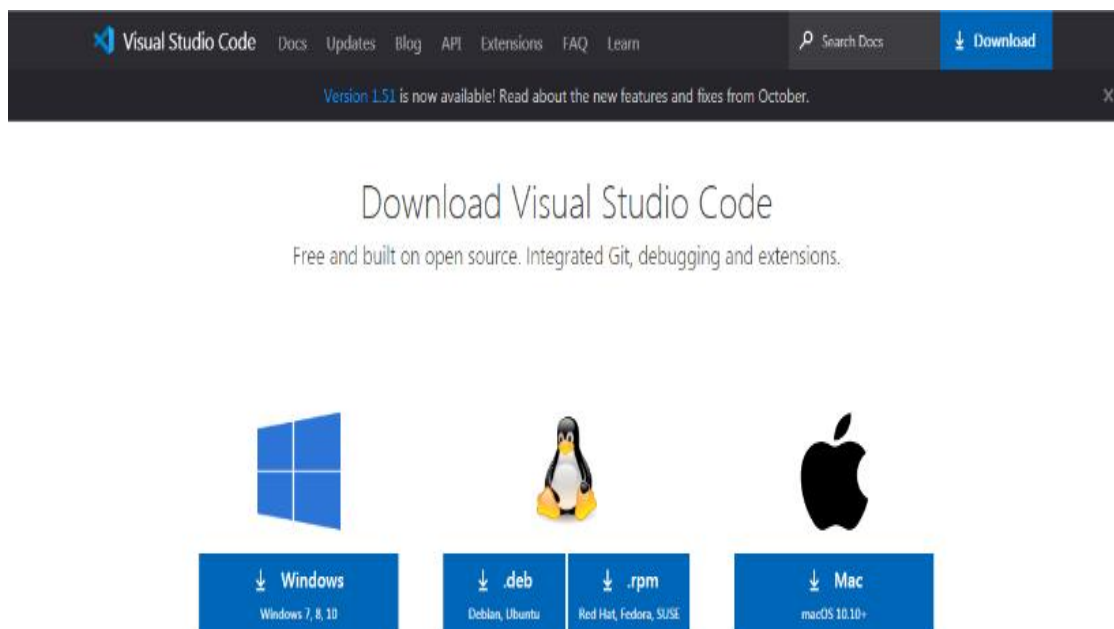


Figure 5.9: Downloading window of VSCode

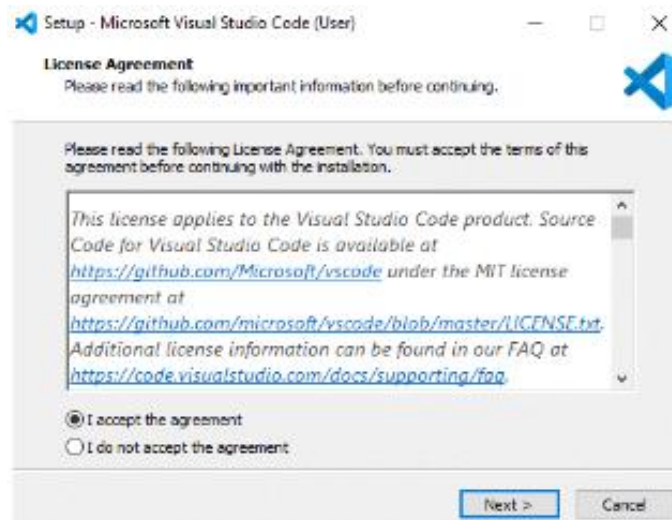


Figure 5.10: Setting up VSCode

After accepting all the requests press finish button. By default, VS Code installs under: “C:\users{username}\AppData\Local\Programs\Microsoft VS Code.”

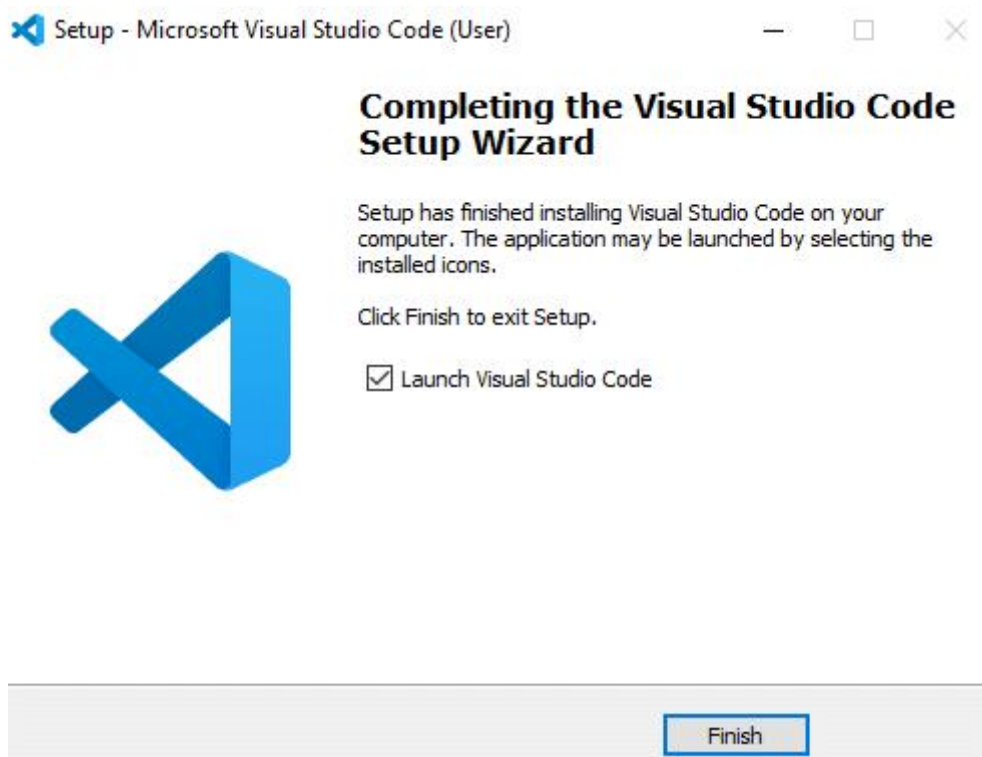


Figure 5.11: Completing the setup of VSCode

After Successful installation you will get the following window:

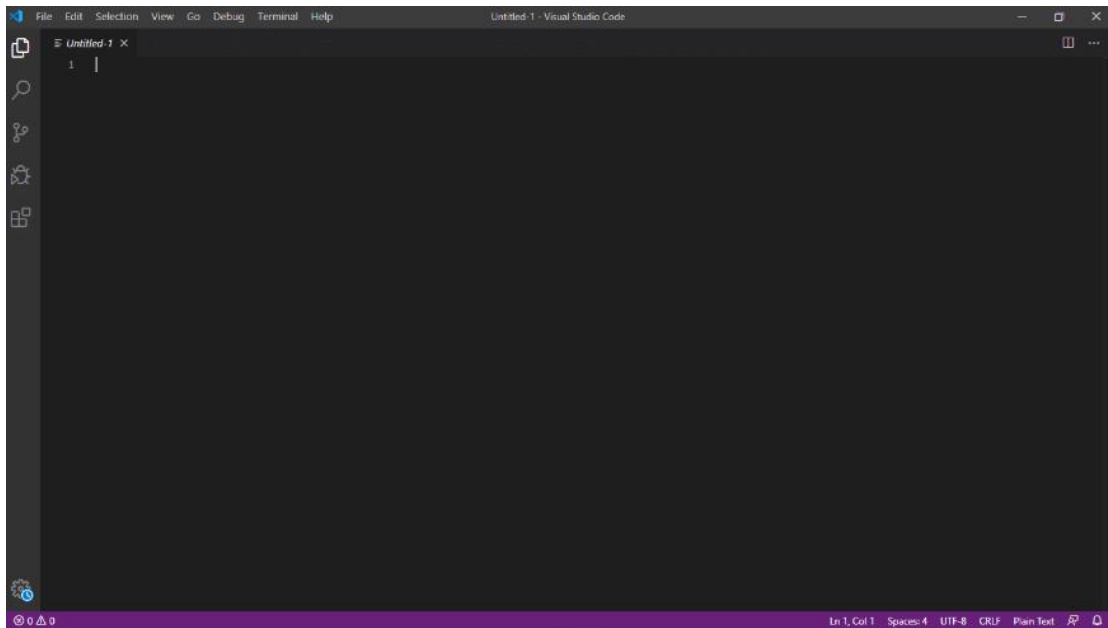


Figure 5.12: Successful installation of VS Code

Flask: Install Flask using pip by running the command `pip install Flask` in your terminal.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\water (2)\water\moisture> pip install flask
Collecting flask
  Using cached Flask-2.2.5-py3-none-any.whl.metadata (3.9 kB)
Requirement already satisfied: Werkzeug>=2.2.2 in c:\users\suchita k\appdata\local\programs\python\python37\lib\site-packages (from flask) (2.2.3)
Requirement already satisfied: Jinja2>=3.0 in c:\users\suchita k\appdata\local\programs\python\python37\lib\site-packages (from flask) (3.1.3)
Requirement already satisfied: itsdangerous>=2.0 in c:\users\suchita k\appdata\local\programs\python\python37\lib\site-packages (from flask) (2.1.2)
Requirement already satisfied: click>=8.0 in c:\users\suchita k\appdata\local\programs\python\python37\lib\site-packages (from flask) (8.1.7)
Requirement already satisfied: importlib-metadata>=3.6.0 in c:\users\suchita k\appdata\local\programs\python\python37\lib\site-packages (from flask) (6.7.0)
Requirement already satisfied: colorama in c:\users\suchita k\appdata\local\programs\python\python37\lib\site-packages (from click>=8.0->flask) (0.4.6)
Requirement already satisfied: zipp>=0.5 in c:\users\suchita k\appdata\local\programs\python\python37\lib\site-packages (from importlib-metadata>=3.6.0->flask) (3.15.0)
Requirement already satisfied: typing-extensions>=3.6.4 in c:\users\suchita k\appdata\local\programs\python\python37\lib\site-packages (from importlib-metadata>=3.6.0->flask) (4.7.1)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\suchita k\appdata\local\programs\python\python37\lib\site-packages (from Jinja2>=3.0->flask) (2.1.5)
Using cached Flask-2.2.5-py3-none-any.whl (101 kB)
Installing collected packages: flask
Successfully installed flask-2.2.5
```

Figure 5.13: Installing Flask Framework in VSCode terminal

1) Setting Up the Project in VS Code:

Create a Project Folder: Create a new folder for your project

Open in VS Code: Open the project folder in VS Code.

2) Creating the Flask Application:

New Python File: Create a new Python file named **app.py** in your project directory. This will be the main script for your Flask application.

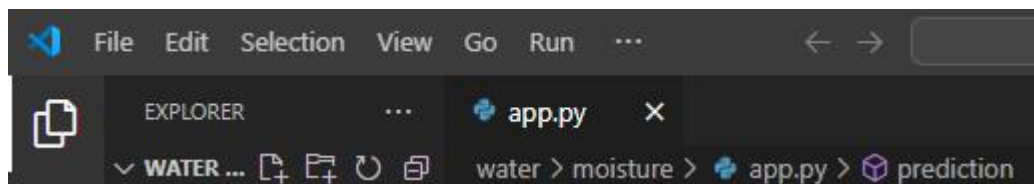


Figure 5.14: Creating Flask Application

3) Writing the Flask Code:

Here's a basic structure of the Flask code within app.py to get you started Python.

```
from flask import Flask, render_template, request
import your_ml_model # Replace with your ML model library
```

```
# Initialize Flask application
```

```
app = Flask(__name__)
```

```
# Route for the main page@app.route("/")
```

```
def index():
```

```
    # Display a form to collect sensor data
```

```
    return render_template("index.html")
```

```
# Route to handle form submission@app.route("/", methods=["POST"])def predict_watering():
```

```
    # Get form data (soil moisture, temperature, humidity)
```

```
    moisture = float(request.form["moisture"])
```

```
    temperature = float(request.form["temperature"])
```

```
    humidity = float(request.form["humidity"])
```

```
# Use your ML model to predict next watering time
predicted_time = your_ml_model.predict(moisture, temperature, humidity)

# Display the predicted watering time on the web page
return render_template("results.html", predicted_time=predicted_time)

if __name__ == "__main__":
    app.run(debug=True) # Run the Flask application in debug mode
```

Explanation of how it works:

1. We import Flask and libraries for templating (render_template) and form handling (request).
2. The app object is our Flask application instance.
3. The index() function defines the route for the main page (/). It renders an HTML template (index.html) that likely includes a form for users to input sensor data.
4. The predict_watering() function handles form submissions (POST method). It retrieves the user-entered sensor data and calls your machine learning model to predict the next watering time. Finally, it renders another template (results.html) displaying the predicted time.
5. The if __name__ == "__main__": block ensures the app runs only when the script is executed directly (not imported as a module). Here, we run the app in debug mode for easier development.

Running the Application:

- **Open Terminal in VS Code:** Open an integrated terminal within VS Code (Terminal > New Terminal).

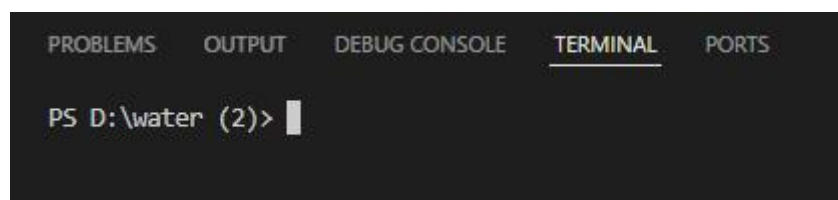
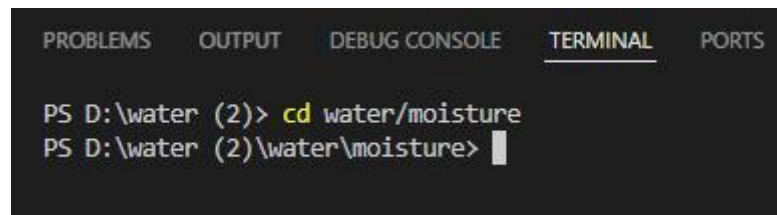


Figure 5.15: Opening the terminal

- **Navigate to Project Directory:** Use the cd command to navigate to your project

directory in the terminal.

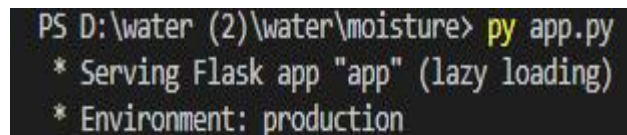


```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\water (2)> cd water/moisture
PS D:\water (2)\water\moisture> |
```

Figure 5.16: Navigating to Directory using commands

- **Run the Flask App:** Run the command `python app.py` to start the Flask development server.



```
PS D:\water (2)\water\moisture> py app.py
* Serving Flask app "app" (lazy loading)
* Environment: production
```

Figure:5.17: Command to run the flask application

Accessing the Application:

By default, the Flask development server typically runs on `http://127.0.0.1:5000/`. Open this URL in your web browser to access the web interface of your application.

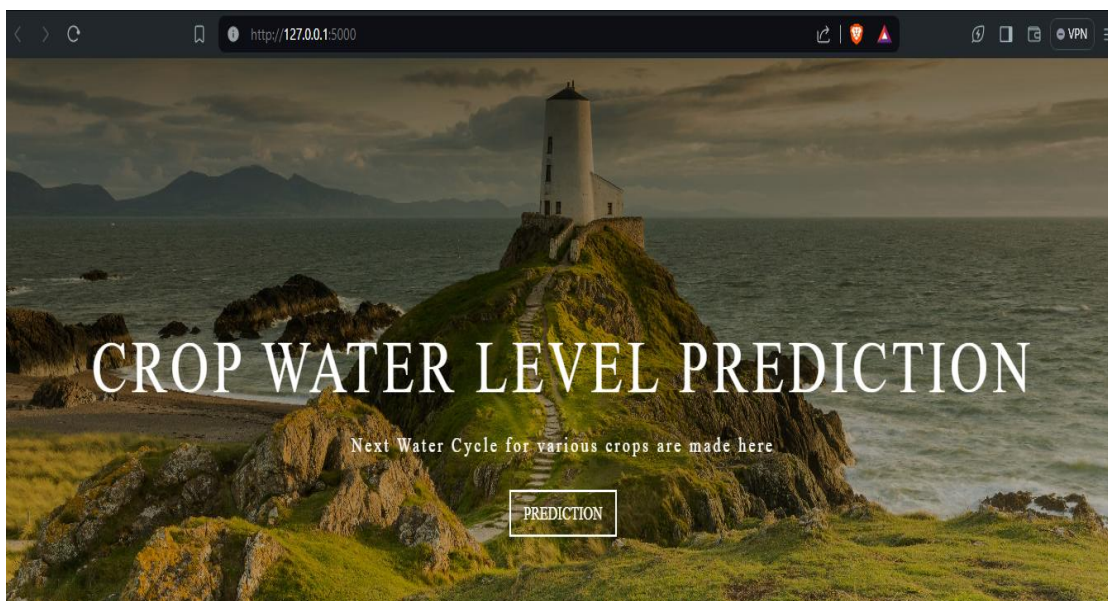


Figure 5.18: Accessing the application

VS Code offers various features to enhance your development experience with Flask, including syntax highlighting, code completion, debugging tools, and extensions for templating languages like HTML.

Flask offers several benefits beyond its usefulness in deploying machine learning models.

- Here are some key advantages of using Flask for web development projects:

Lightweight and Easy to Learn: Compared to more heavyweight frameworks like Django, Flask has a minimalist design and is known for its simplicity. This makes it easier to learn, especially for beginners in web development.

Flexibility and Control: Flask provides a high degree of flexibility in how you structure your application. You have more control over the project architecture and can choose the libraries and tools that best suit your needs.

Rapid Prototyping: Due to its simplicity, Flask is ideal for rapid prototyping. You can quickly build a basic web application to test ideas and validate concepts before diving into more complex development.

API Development: Flask excels at building RESTful APIs (Application Programming Interfaces). These APIs allow your application to communicate with other applications or systems, making it suitable for building backend services or microservices architectures.

Integration with Other Technologies: Flask integrates well with various libraries and frameworks for tasks like database interaction, user authentication, templating engines, and more. This flexibility allows you to build feature-rich web applications.

Scalability: While Flask is primarily designed for smaller to medium-sized web applications, it can also be scaled to handle a moderate amount of user traffic. There are techniques and best practices to improve scalability for growing applications.

Community and Resources: Flask has a large and active community of developers. This translates to readily available online resources, tutorials, and forums for learning and troubleshooting.

- Here are some specific use cases where Flask shines outside of machine learning:
Simple Web Applications: Need a basic website to showcase your portfolio, create a landing page for a project, or display static content? Flask can handle it effectively.

CRUD Applications: Building applications for basic Create, Read, Update, and Delete (CRUD) operations on data can be accomplished efficiently with Flask. Imagine a simple to-do list app or a basic note-taking application.

RESTful APIs: If you need to expose functionalities of your backend system to mobile apps, web dashboards, or other applications, Flask is a great choice for building robust APIs.

Microservices Architecture: Flask can be used to build individual microservices that can be combined to create a larger application. This modular approach promotes maintainability and scalability.

Overall, Flask offers a powerful and versatile framework for web development, making it a valuable tool beyond just deploying machine learning models. Its simplicity, flexibility, and ease of use make it a great choice for a variety of web application development needs.

5.2 SOURCE CODE

```
#include <LiquidCrystal.h>
#include <stdio.h>
LiquidCrystal lcd(13, 12, 14, 27, 26, 25);
#include <WiFi.h>
#include <HTTPClient.h>
HTTPClient http;

const char *ssid = "iotserver";
const char *password = "iotserver123";

int httpResponseCode;
String servername = "http://projectsfactoryserver.in/storedata.php?name=";
String accountname = "iot813";
String field1 = "&s1=";
String field2 = "&s2=";
String field3 = "&s3=";
String field4 = "&s4=";
String field5 = "&s5=";
String payload="";

#include <Wire.h>
#include "DHT.h"
#define DHTPIN 21
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
int buzzer = 23;
int mos = 19;
int motor = 18;
float tempc=0,humc=0;
unsigned char rcv,count,gchr,gchr1,robos='s';
int sti=0;
String inputString = "";    // a string to hold incoming data
```

```

boolean stringComplete = false; // whether the string is complete
int i=0,k=0,lop=0;
unsigned char gv=0,msg1[10],msg2[11];
int ii=0,rchkr=0;
void beep()
{
    digitalWrite(buzzer, LOW);delay(2000);digitalWrite(buzzer, HIGH);
}
void okcheck()
{
    unsigned char rcr;
    do{
        rcr = Serial.read();
    }while(rcr != 'K');
}
int cntlmk=0;int stn=0; int hbv=0;int cntlmk=0;
int stn=0; int hbv=0;
void iot_send()
{
    lcd.setCursor(15,0);lcd.print("U");
    http.begin(servername + accountname + field1 + String(tempc) + field2 +
String(humc) + field3 + mos_string);
    httpStatusCode = http.GET();
    if(httpStatusCode>0)
    {
        payload="";
        payload = http.getString();
    }
    else
    { ; }
    delay(3000);
    lcd.setCursor(15,0);lcd.print(" ");
}
void setup()

```

```

{
  Serial.begin(9600); //serialEvent();
  pinMode(motor, OUTPUT); pinMode(mos, INPUT);pinMode(buzzer, OUTPUT);
  digitalWrite(buzzer, HIGH);digitalWrite(motor, LOW);

  lcd.begin(16, 2);
  lcd.print("  IOT Based");
  lcd.setCursor(0,1);
  lcd.print("  Agriculture");
  delay(2500);
  WiFi.begin(ssid, password);
  Serial.println("Connecting");
  while(WiFi.status() != WL_CONNECTED)
  {
    delay(500);
  }
  //Serial.println(WiFi.localIP());
  delay(3000);
  //dht.setup(DHTpin, DHTesp::DHT11);
  dht.begin();

  lcd.clear();
  lcd.print("T:");
  lcd.setCursor(8,0);
  lcd.print("H:");

  lcd.setCursor(0,1);
  lcd.print("M:"); //2
}

void loop()
{
  delay(500);
  humc = dht.readHumidity();

```

```

    tempc = dht.readTemperature();

    if(tempc == 295 || humc == 295)
    {
        goto mnp;
    }
    lcd.setCursor(2,0);convertl(tempc);
    lcd.setCursor(10,0);convertl(humc);

    if(tempc >= 38 || humc > 70)
    {
        beep();
        iot_send();
    }
mnp:
    mos_string="";
    if(digitalRead(mos) == LOW)
    {
        lcd.setCursor(2,1);lcd.print("Wet");
        mos_string="Wet";digitalWrite(motor, LOW);
        // iot_send();
    }
    if(digitalRead(mos) == HIGH)
    {
        lcd.setCursor(2,1);lcd.print("Dry");
        mos_string="Dry";digitalWrite(motor, HIGH);
    }
    delay(1000);
    cntlmk++;
    if(cntlmk >= 40)
    {cntlmk = 0;
        iot_send();
    }
}

```

6.TESTING

6.1 INTRODUCTION

A test case is a specification of the inputs, execution condition, testing procedure, and expected results that define a single test to be executed to achieve particular software testing objective. The mechanism for determining whether software program or system has passed or failed such a test is known as a test oracle. In some settings, an oracle could be a requirement or use case. It may take a test case to determine that a software program or system is functioning correctly. Test cases are often referred to as test scripts, particularly which written. Written test cases are usually collected into test suites.

6.2 TYPES OF TESTING

1.UNIT TESTING

Unit Testing is a level of software testing where individual units components of the software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software Symptoms Based Intelligent Disease Prediction System Using DT. It usually has one or a few inputs and usually a single output. In procedural programming, a unit may be an individual program, function, procedure, etc. In object-oriented programming, the smallest unit is a method, which may belong to a base/ super class, abstract class or derived child class. (Some treat a module of an application as a unit. This is to be discouraged as there will probably be many individual units within that module.) Unit testing frameworks, drivers, stubs, and mock/ fake objects are used to assist in unit testing.

2.INTEGRATION TESTING

Integration Testing is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist integration Testing Integration Testing is the second level of testing performed after Unit Testing and before System Testing.

3.SYSTEM TESTING

System testing is a level of software testing where a complete and integrated software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements. System testing is performed on the entire system in context of either functional requirement specifications (FRS) or system requirement specification (SRS), or both. System testing tests not only design but also behavior and even the believed expectations of customers. It is also intended to test up to and beyond bounds defined in the software or hardware requirements specification.

4.PERFORMANCE TESTING: Simulate user load by sending multiple requests concurrently to assess the system's performance under stress. This helps identify bottlenecks and ensure the application can handle expected user traffic.

5.DATA VALIDATION: Test the quality of your training data. Ensure the data is accurate, complete, and free from biases that might affect the model's performance. This can involve data cleaning and visualization techniques.

6.MODEL VALIDATION: Evaluate the performance of your machine learning model on unseen data. Split your data into training and testing sets. Train the model on the training data and assess its accuracy on the testing data.

7.DEPLOYMENT TESTING: After deployment to a web server, perform testing to ensure the application functions as expected in the production environment. This might involve testing database connectivity, user authentication (if implemented), and overall system stability.

Exploration of the system



S.No	Soil Moisture(%)	Temp (°C)	Humidity (%)	Expected Output	Output Display	Water Pump(ON/OFF)	Result
1	30(Dry)	25	50	Soil Dry, Pump on	Dry	ON	Pass
2	50(Moist)	25	50	Soil Moist, Pump on	Moist	OFF	Fail
3	70(Wet)	25	50	Soil wet, Pump off	Wet	OFF	Pass
4	30(Dry)	35(Hot)	50	Soil Dry, pump on	Dry	ON (Longer time run)	Pass
5	50(Moist)	35(Hot)	50	Soil Moist, Pump off	Moist	OFF	Pass
6	70(Wet)	35(Hot)	50	Soil wet, Pump off	Wet	OFF	Pass
7	30(Dry)	15(Cold)	50	Soil wet, Pump off	Dry	ON (Shorter time run)	Fail
8	50(Moist)	15(Cold)	50	Soil Moist, Pump off	Moist	OFF	Pass
9	70(Wet)	15(Cold)	50	Soil wet, Pump off	Wet	OFF	Pass

Table 6.1:Test Cases

7.OUTPUT SCREENS

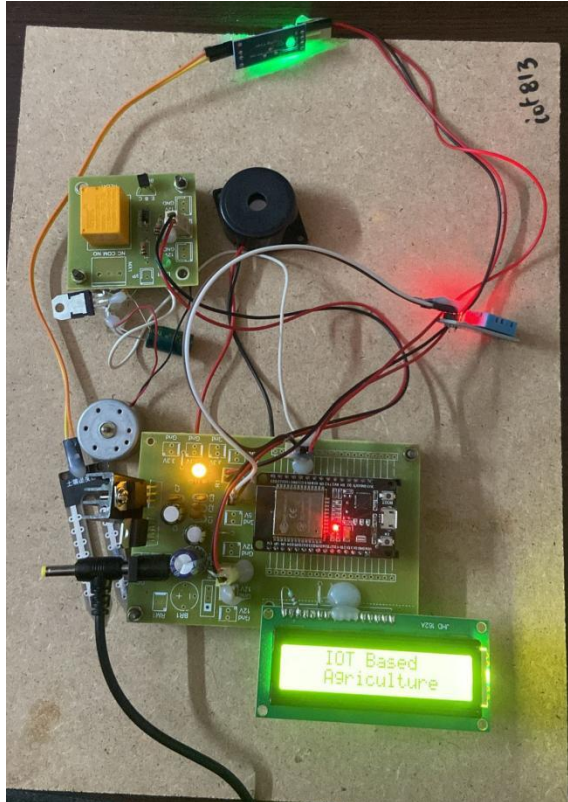


Figure 7.1: Initial LCD Display

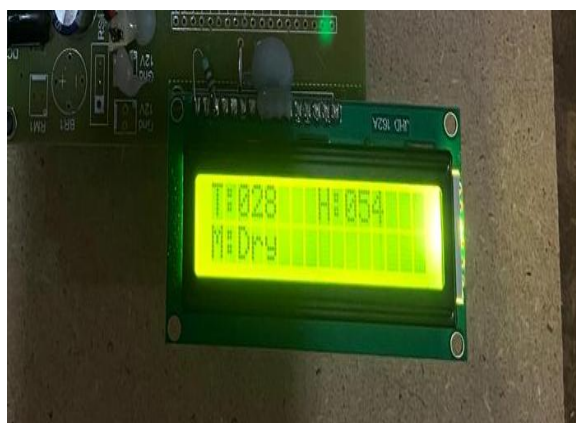


Figure 7.2: Display of T, H, M Values

S.No	Temperature	Humidity	Moisture	Date
1	31.30	67.00	Dry	2024-04-03 17:58:16
2	31.40	67.00	Dry	2024-04-03 17:57:12
3	31.30	66.00	Wet	2024-04-03 17:56:08
4	31.30	67.00	Dry	2024-04-03 17:55:04
5	31.30	66.00	Dry	2024-04-03 17:53:59
6	31.20	67.00	Dry	2024-04-03 17:52:55
7	31.20	66.00	Dry	2024-04-03 17:51:51
8	31.20	66.00	Dry	2024-04-03 17:50:47
9	31.20	66.00	Dry	2024-04-03 17:49:43
10	38.00	22.00	Dry	2024-04-01 16:06:57
11	38.00	21.00	Dry	2024-04-01 16:06:50
12	38.30	21.00	Dry	2024-04-01 16:06:43
13	38.60	21.00	Dry	2024-04-01 16:06:36
14	39.00	21.00	Dry	2024-04-01 16:06:29
15	39.00	21.00	Dry	2024-04-01 16:06:19

Figure 7.3: Display of Data Through Web Interface

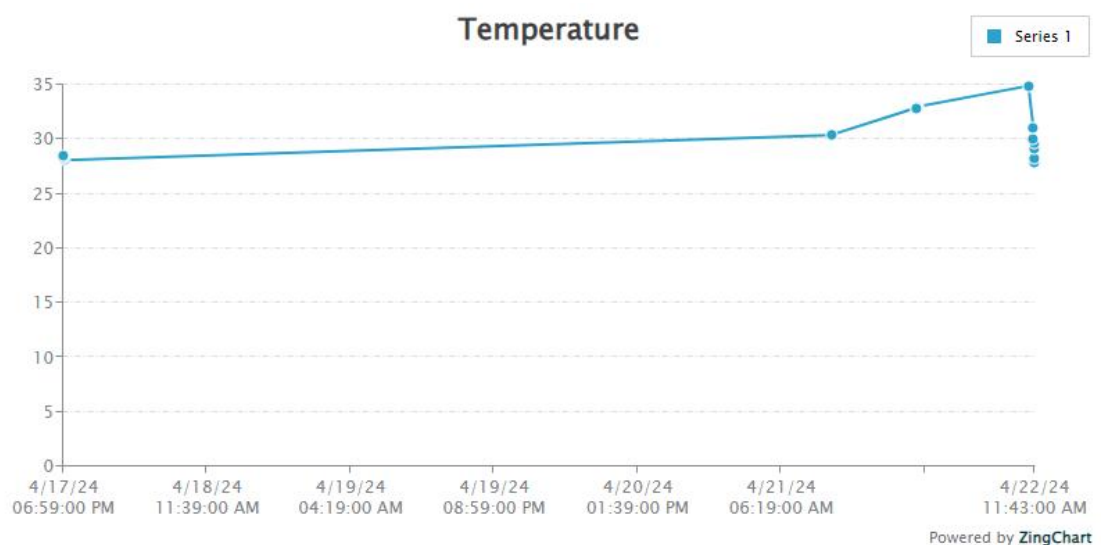


Figure 7.4: Graphical View of Temperature

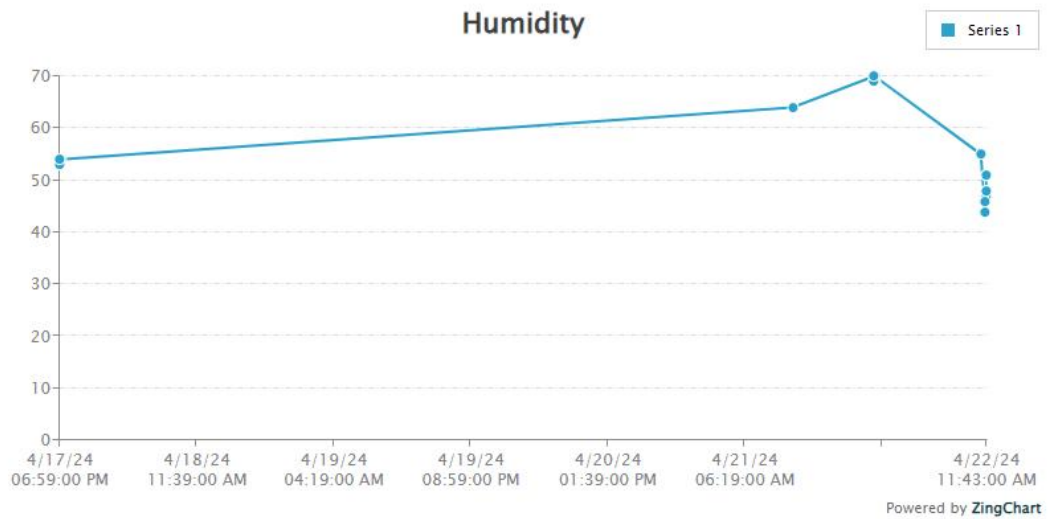


Figure 7.5: Graphical View of Humidity

CROP PREDICTION

Crop: rice

soil moisture level: 500 to 600

Temperature: 30 to 40

Humidity: above 40

---select re ---select humidity----

- below 25
- 25 to 30
- 30 to 40
- above 40

Figure 7.5: Web Interface for selecting various crops

Crop Prediction
Home Prediction

PREDICTION RESULT

We can water RICE CROP two times for the next day and it helps us to conserve water.

Figure 7.6: Prediction Result of next water cycle

8.CONCLUSION

This project has developed an automatic watering system using Arduino. The prototype of the model worked properly when tested on different soils. The components we use in the system are readily available and easy to operate. Thus, this system acts as an efficient irrigation method. This is much better than the manual watering process, which is labor intensive and time-consuming. This project is primarily intended for farmers and gardeners who do not have time to water their plants. It also applies to those farmers who waste water during irrigation. The project can be extended to greenhouses, where manual supervision is practically nonexistent. This principle can be elaborate to create fully automated gardens and farmland.

In conclusion, the integration of IoT technology, APIs for web interfaces, and machine learning in automatic crop watering systems represents a significant advancement in agricultural practices. By leveraging IoT sensors to monitor soil moisture levels, weather conditions, and other environmental parameters, farmers can efficiently manage water resources and optimize crop growth. The use of APIs enables seamless communication between the watering system and web interfaces, allowing users to remotely monitor, control, and receive updates on crop watering schedules. It plays a crucial role in predicting the optimal timing for water cycles based on historical data, weather forecasts, and crop characteristics. This predictive capability enhances the efficiency and effectiveness of irrigation, ensuring that crops receive the right amount of water at the right time to maximize yields while conserving resources. Overall, the combination of IoT, APIs, and machine learning offers a sophisticated solution for automated crop watering, driving improvements in agricultural productivity, sustainability, and profitability.

9.FUTURE SCOPE

- In future this system can be developed with IoT based mobile system for a user by sending the notification whether motor pump is on/off, so that they can notify with the timings when does the water is given to a particular plant.
- Solar energy can be used for power supply.
- Machine learning can be used to further include N, P, K values to our datasets which are gathered through web interface and can recommend crop.
- In future, by integrating advanced sensors and AI-driven analytics could enable more precise monitoring and management of crop health, leading to targeted interventions such as variable-rate irrigation and fertilization.

BIBLIOGRAPHY

1. Nayyar, Anand & Puri, Vikram. (2016). Smart farming: IoT based smart sensors agriculture stick for live temperature and moisture monitoring using Arduino, cloud computing & solar technology. The international conference on communication and computing (ICCCS-2016)
2. Gorli, Ravi & Yamini G. (2017). Future of Smart Farming with Internet of Things. Journal of Information technology and Its Applications.
3. S. jegadeesan, dr. g. k. d. Prasanna venkatesan Smart cow health monitoring, farm environmental monitoring and control system using wireless sensor networks, International journal of advanced engineering technology, Jan-March 2016.
4. Vaibhavraj S. Roham, Ganesh Pawar, Abhijit Patil & Prasad Rupnar, Smart Farm using Wireless Sensor Network, International Journal of Computer Applications, National Conference on Advances in Computing, NCAC 2015.
5. Anushree M K & Krishna R. (2018). A smart farming using Arduino based technology. International Journal of Advance Research,Ideas and Innovations in Technology.
6. Yuthika Shekhar F.: Intelligent IoT Based Automated Irrigation System. International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 18 (2017) pp. 7306-7320.
7. R. Assaf F., I. Ishaq S.: Improving Irrigation by Using a Cloud Based IoT System. International Conference on Promising Electronic Technologies (ICPET), 2020, pp. 28-31, doi: 10.1109/ICPET51420.2020.00014.
8. T. Thaher F., I. Ishaq S.: "Cloud-based Internet of Things Approach for Smart Irrigation System: Design and Implementation,". International Conference on Promising Electronic Technologies (ICPET), 2020, pp. 32-37, doi: 10.1109/ICPET51420.2020.00015.