PROJECT REPORT

Plant Moisture Sensor with Email Notification

Student Name:LiXing Cao

SETU Number:W20109972

Lab ID: Project 2 Plant Moisture Sensor with Email Notification

Email:1269643130@qq.com

Github URL:

Contents

1. the agile process [2](#_bookmark0)
   1. Agile Framework with User Story [2](#_bookmark1)
   2. Scrum & Sprint Summary [4](#_bookmark2)
   3. Burndown Table & Chart [5](#_bookmark3)
2. project introduction [7](#_bookmark6)
   1. Background and Motivation [7](#_bookmark7)
   2. Project Objectives [7](#_bookmark8)
   3. Project Management [7](#_bookmark9)
   4. Project Methodology [8](#_bookmark11)
3. hardware and circuit connection [10](#_bookmark12)
   1. Overview of System Hardware [10](#_bookmark13)
   2. Soil Moisture Sensor Description [10](#_bookmark14)
      1. Soil Moisture Sensor PinOut Details [11](#_bookmark15)
   3. Circuit Connection [11](#_bookmark16)
   4. Moisture Detection Test [12](#_bookmark19)
4. software development methodology [13](#_bookmark20)
   1. Implementation Overview [13](#_bookmark21)
   2. Moisture Sensor Test [13](#_bookmark22)
   3. Email Function Test [15](#_bookmark26)
      1. Email Configuration Setup [15](#_bookmark27)
      2. Email Script Testing [15](#_bookmark28)
   4. Email Notification with Sensor [17](#_bookmark32)
   5. System Testing and Results [19](#_bookmark36)
5. Commit History [21](#_bookmark37)
6. Source Code [22](#_bookmark38)

## THE AGILE PROCESS

This project followed the Scrum Agile framework, employing iterative development as its core methodology. The design and implementation of the "Soil Moisture Monitoring and Email Notification System" were divided into bi-weekly sprint cycles. Through the creation of user stories, I ensured responsiveness to change, and continuous progress. Task tracking was ef- fectively managed using burndown charts, while GitHub was utilized for version control and collaboration. The result was a successful delivery of both hardware integration and software automation, accompanied by complete Agile documentation and demonstrable outcomes.

# Agile Framework with User Story

In the early planning stage, a series of user stories were created to guide the development tasks from both technical and user perspectives, which were shown below.

**SCRUM (Plan Plant Moisture Sensor Project) Duration: 12 Days SPRINT 1 - Duration = 12 Days**:

* User Story 1: As a project manager, I want to define clear user stories so that the tasks are well defined.
* User Story 2: As a Scrum master, I want to establish a two-week sprint plan so that progress can be tracked efficiently.
* User Story 3: As a project manager, I want to create burndown chart and table so that I can track the remaining work.
* User Story 4: As a developer, I want to document agile part so that I can plam efficiently.
* User Story 5: As a developer, I want to connect the soil moisture sensor to the Raspberry Pi correctly so that it can detect moisture levels accurately.
* User Story 6: As a user, I want to perform quick moisture test so that I can verify its functionality.

#### SCRUM (Plan Plant Moisture Sensor Project) Duration: 12 Days

* User Story 7: As a developer, I want to calibrate the sensor’s potentiometer so that it triggers alerts at the right moisture threshold.
* User Story 8: As a developer, I want to create a new repository so that I can manage my code.
* User Story 9: As a developer, I want to write a Python script to read the sensor’s digital output so that I can monitor soil moisture in real time.
* User Story 10: As a user, I want the script to print moisture status messages so that I can manually check plant conditions.
* User Story 11: As a developer, I want to configure SMTP settings for email notifi- cations so that alerts can be sent automatically.
* User Story 12: As a user, I want to create a dedicated sender email with an app password so that my main email account remains secure.
* User Story 13: As a developer, I want to write a Python script to send test emails via SMTP so that I can verify email delivery.
* User Story 14: As a developer, I want to merge the sensor and email scripts so that the system sends alerts based on moisture readings.
* User Story 15: As a user, I want the system to check soil moisture 4 times per day so that I can check healthy status of plant.
* User Story 16: As a maintainer, I want to keep all email alerts for 3 days so that I can review its reliability.
* User Story 17: As a developer, I want to handle exceptions in the Python scripts so that the system fails gracefully.
* User Story 18: As a developer, I want to finish my burndown chart so that I can manage work efficiently.
* User Story 19: As a developer, I want to use Git and Push for version control so that I can track code changes efficiently.
* User Story 20: As a presenter, I want to record a 5-minute video demo showing hardware setup, sensor testing, and email alerts so that others can understand the project.
* User Story 21: As a student, I want to compile a final report so that my work is well-documented.
* User Story 22: As a student, I want to submit the final report so that I can finish my work.

# Scrum & Sprint Summary

According to Section [1.1](#_bookmark1), I have designed the tasks to do in this 12-day sprint.

#### SCRUM (Plan Plant Moisture Sensor Project) Duration: 12 Days SPRINT 1 - Duration = 12 Days:

* Task 1: Define project user stories
* Task 2: Create two-week sprint plan
* Task 3: Prepare burndown chart
* Task 4: Document agile methodology
* Task 5: Connect moisture sensor to Raspberry Pi
* Task 6: Perform initial sensor test
* Task 7: Calibrate sensor threshold
* Task 8: Create Git repository
* Task 9: Develop sensor reading script
* Task 10: Implement moisture status display
* Task 11: Configure SMTP email settings
* Task 12: Set up dedicated sender email
* Task 13: Create email test script
* Task 14: Integrate sensor and email scripts
* Task 15: Schedule 4 daily moisture checks
* Task 16: Maintain 3-day email logs
* Task 17: Add error handling
* Task 18: Finish burndown chart
* Task 19: Manage code version control
* Task 20: Record project demo video
* Task 21: Compile final report
* Task 22: Submit completed work

In Sprint 1, a total of 22 tasks were defined and scheduled over a 12-day development cy- cle. The tasks covered the full life cycle of the project, starting from initial planning to final delivery. Early tasks focused on project setup, including defining user stories, creating sprint plans, and preparing project management tools like the burndown chart. Mid-stage tasks em- phasized technical implementation, such as connecting the soil moisture sensor, testing and calibrating the sensor, and developing Python scripts for sensor reading and email notification. Later tasks involved system integration, error handling, and daily email scheduling. Finally, the

sprint concluded with version control management, a project demonstration video, final report compilation, and submission. This systematic task breakdown ensured efficient progress track- ing and the successful delivery of both hardware and software components within the sprint period.

# Burndown Table & Chart

A burndown chart is a graph that represents the work left to do versus the time it takes to complete it. It can be especially useful for teams working in sprints, as it can effectively show whether deadlines are able to be met along the way.

The Sprint #1 Burndown Chart illustrates the task breakdown and time tracking for the 12-day sprint used in the development of the Plant Moisture Monitoring System. It maps a total of 22 tasks aligned with previously defined user stories, with each task assigned an estimated effort in hours and corresponding completion dates. Figure [1.1](#_bookmark4) presents the burndown chart and full completion of tasks for Sprint 1 (Duration: 12 days).

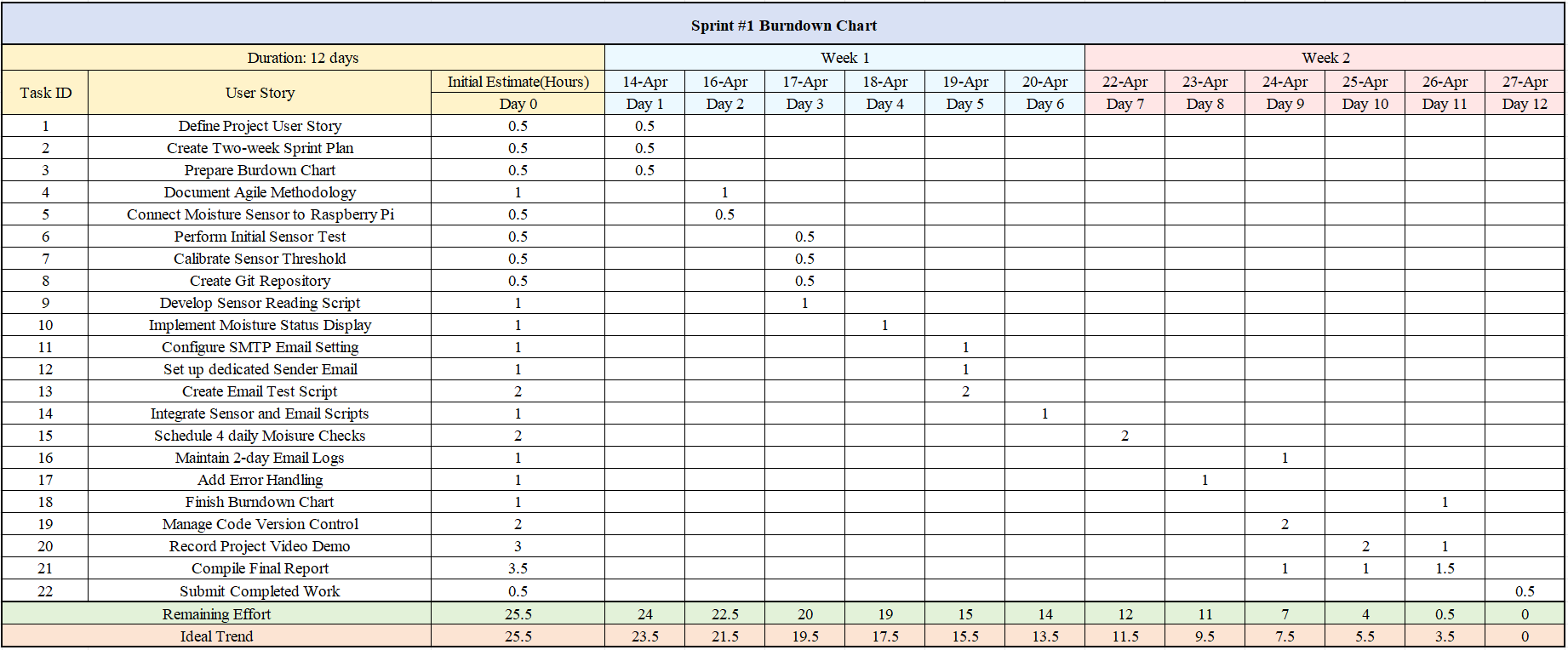


Figure 1.1. Sprint 1 Burndown Table

Figure [1.2](#_bookmark5) shows the ideal trend and remaining effort of completing the tasks.

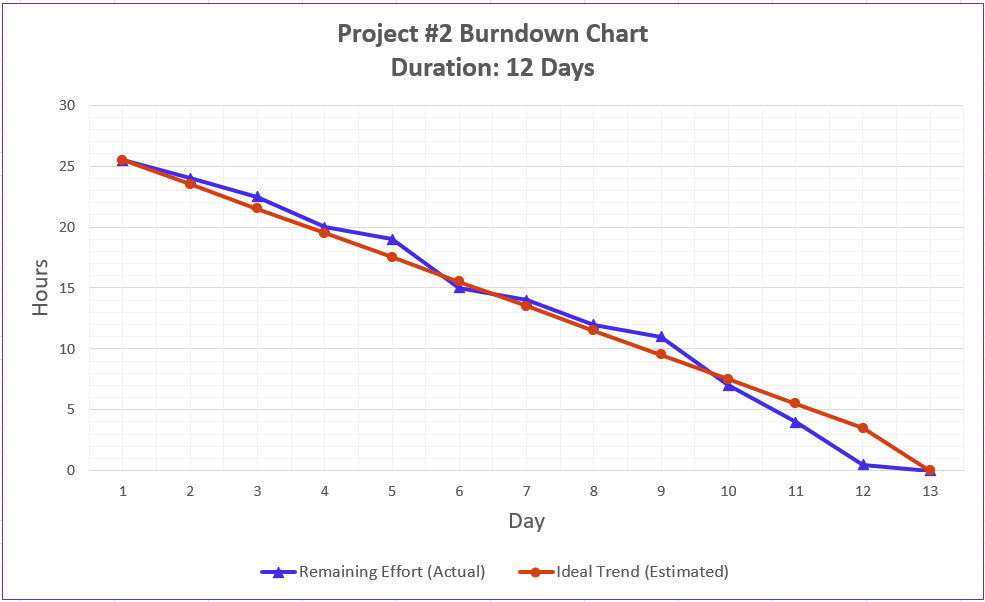


Figure 1.2. Burndown Chart

Figure [1.2](#_bookmark5) presents a comparison between the expected workload and the actual remaining effort throughout the 12-day development sprint. The x-axis indicates the development days (Day 1 to Day 12), while the y-axis represents the remaining workload in hours. The red line denotes the ideal trend, reflecting a uniform, linear decrease if tasks were completed evenly across the sprint. The blue line illustrates the actual remaining effort, showing the day-by-day progress achieved.

From the figure, it can be observed that during the first ten days, the actual progress slightly outpaced the estimated trend, with more than one hour of workload completed on average each day. On Days 11 and 12, the team’s efficiency significantly improved, rapidly catching up with the plan and successfully completing all scheduled tasks. The overall trajectory of the actual effort closely follows the ideal trend, demonstrating effective workload distribution and excellent time management throughout the sprint.

## PROJECT INTRODUCTION

# Background and Motivation

Houseplants are an essential part of modern interior living environments, yet one of the most common reasons for plant deterioration is irregular watering. Many individuals lack the time or consistency to monitor soil moisture regularly, leading to either over-watering or neglect. In light of this, there is a growing demand for automated solutions that ensure plants are watered at appropriate intervals. By integrating low-cost sensors and embedded systems, it becomes possible to design a smart, user-friendly system to assist users in maintaining plant health efficiently.

# Project Objectives

This project aims to develop a soil moisture monitoring system using a Raspberry Pi and a digital soil moisture sensor. The system is designed to read real-time moisture data and automatically notify users via email at regular intervals. The core objectives include:

* Monitoring soil moisture levels using a digital sensor;
* Configuring automated email notifications sent four times daily;
* Ensuring accuracy and responsiveness through GPIO integration and Python scripting;
* Applying Agile methodology to manage the development process efficiently.

# Project Management

Figure [2.1](#_bookmark10) shows the public GitHub repository **Project2\_2025** created for managing the Plant Moisture Monitoring System with Email Notification project. The repository demonstrates clear management of structured Agile project practices and code program version. The repository includes multiple Python scripts, each reflecting different stages of the system’s evolution:

* SoilSensor.py for initial sensor testing,
* send\_email.py for verifying SMTP email functionality,
* SoilSensorEmail.py for the complete integrated solution

The README.md file at the root of the repository provides a well-documented project descrip- tion, including its introduction, hardware requirements, steps to run the script, and project features. It also highlights that the system was built using Scrum-based Agile methodology, incorporating a sprint cycle and burn-down chart to track progress.

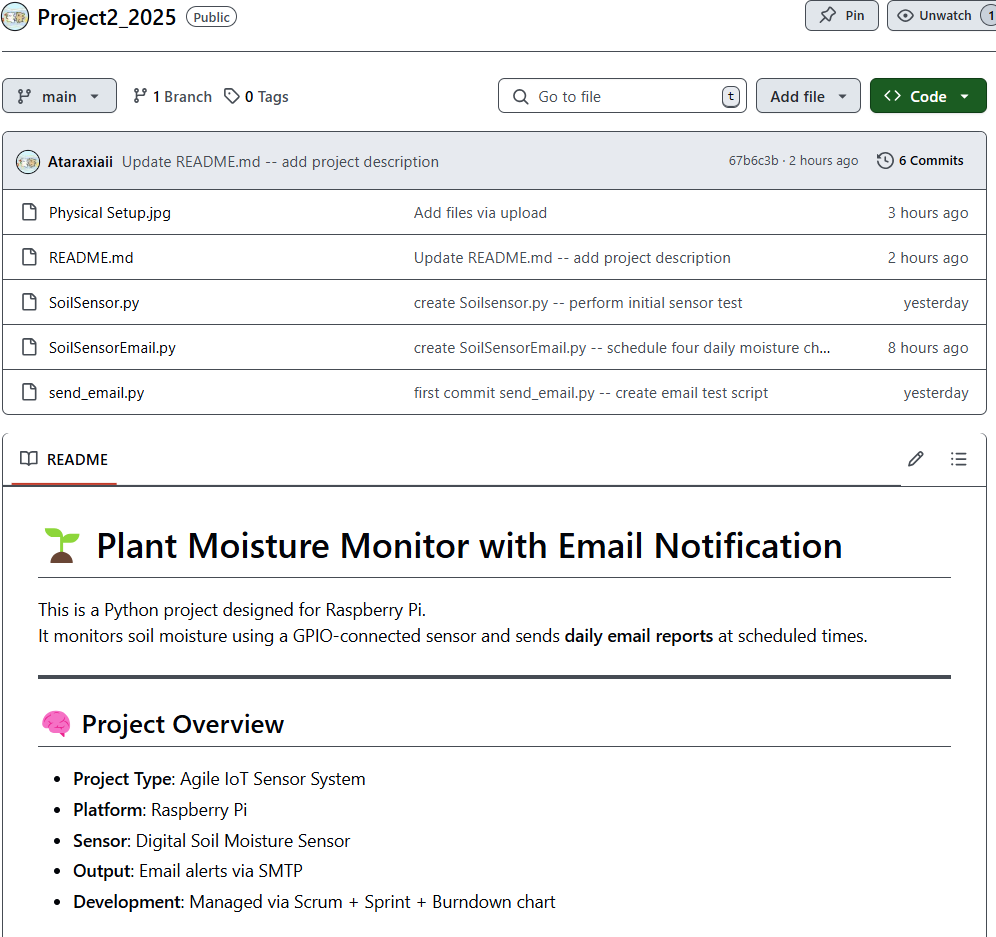


Figure 2.1. Github Repository Management

# Project Methodology

In this project, I designed and implemented a plant moisture monitoring system based on a Raspberry Pi and a digital soil moisture sensor. I began by conducting background research and planning the system using Agile methodology, including defining user stories and prepar- ing a burndown table. I then completed the hardware wiring, sensor calibration, and moisture

detection testing.

On the software side, I wrote Python scripts to read moisture data via GPIO and send auto- mated email notifications using SMTP. I scheduled the system to send daily reports at 07:00, 10:00, 13:00, and 16:00. All code was version-controlled through GitHub, and I documented the full development process, from planning to testing. Through this hands-on project, I gained experience in sensor integration, embedded programming, automation, and agile project man- agement. During the development process, a major challenge was encountered: the default

GPIO4 pin on the Raspberry Pi was already occupied by the OneWire protocol, which conflicted with the moisture sensor input. To address this, the sensor was reassigned to GPIO17, and the software libraries were updated accordingly to ensure compatibility and proper functionality.

## HARDWARE AND CIRCUIT CONNECTION

# Overview of System Hardware

The hardware for this project is centered around the Raspberry Pi, which acts as the main con- troller for the soil moisture monitoring system. The sensor used is the FC-28 Soil Moisture Sensor, which is capable of providing both analog and digital outputs. For this application, the digital output is utilized due to its simplicity and suitability for threshold-based detection.

Additional components include jumper wires for connections and a potted plant for real-world simulation. Power to the sensor is supplied through the 5V and GND pins on the Raspberry Pi.

# Soil Moisture Sensor Description

The FC-28 sensor consists of two primary parts: the probe and the signal conditioning board. The probe is inserted into the soil to measure conductivity, which varies with moisture content. The onboard potentiometer allows the user to calibrate the sensor’s trigger point for dry or wet soil. When the soil moisture level drops below the preset threshold, the digital output goes low, triggering a response in the system.

There are many benefits to using a soil moisture sensor. One benefit is that it can help to save water. The sensor can be used to monitor the moisture content in the soil and then the irrigation system can be adjusted accordingly. This can help to reduce the amount of water that is used and can also help to reduce the amount of water that is wasted. Another benefit of using a soil moisture sensor is that it can help to improve the health of the plants.

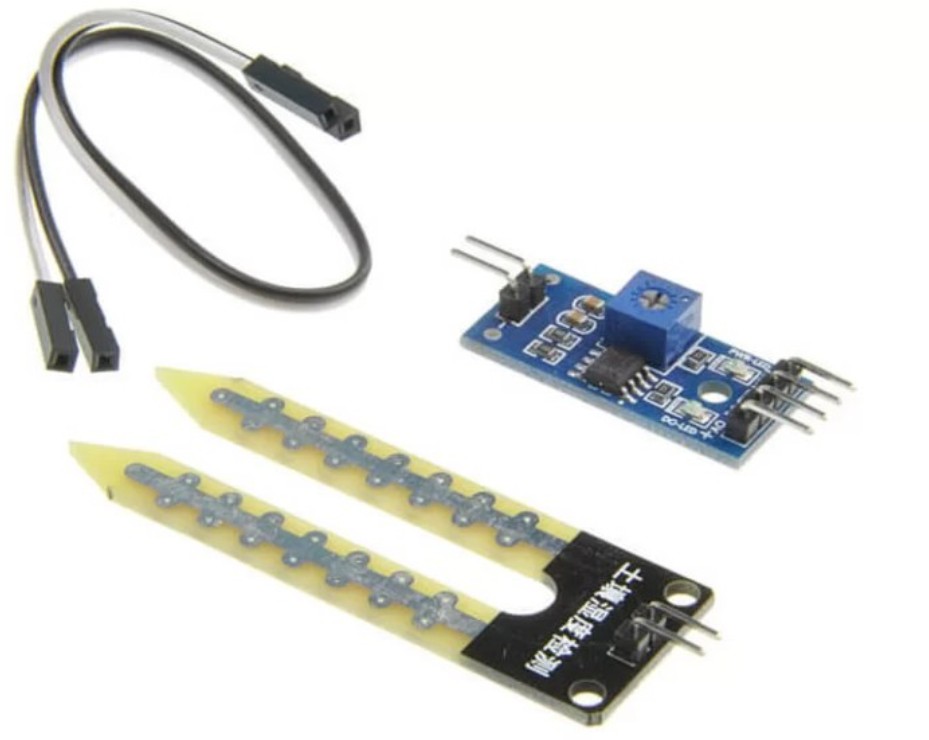


Figure 3.1. Soil Moisture Sensor

### Soil Moisture Sensor PinOut Details

The soil Moisture sensor FC-28 has four pins:

* + - * + VCC: Connected to Raspberry Pi 5V pin
        + GND: Connected to Raspberry Pi GND pin
        + D0 (Digital Output): Connected to Raspberry Pi GPIO
        + A0 (Analog Output): Not used in this project

# Circuit Connection

The wiring is simple but essential for correct functionality. The VCC and GND pins of the FC- 28 sensor are connected directly to the Raspberry Pi’s 5V and GND pins. The D0 pin (digital output) is connected to GPIO 17. This connection allows the Raspberry Pi to read the moisture status as a binary value (wet/dry). The ideal connection of components should be like in Figure [3.2](#_bookmark17).

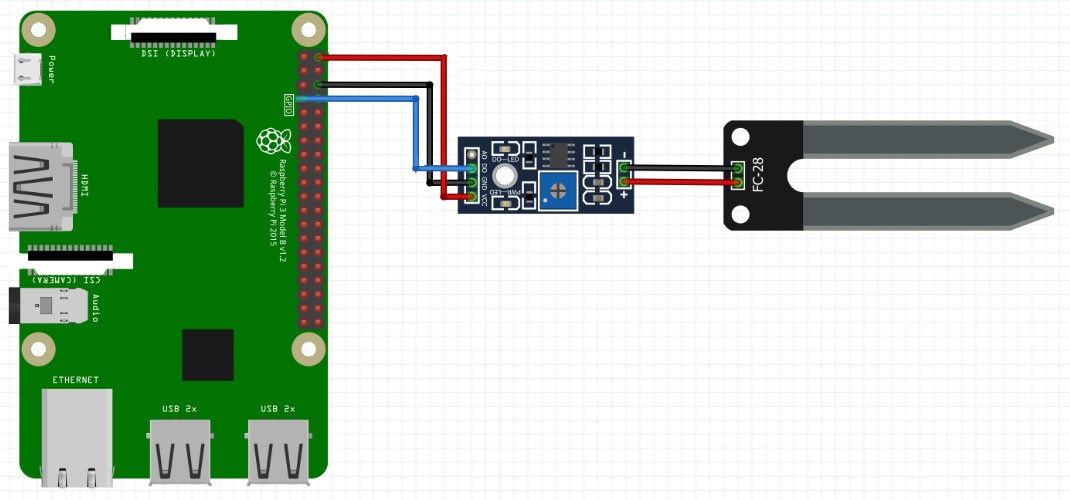


Figure 3.2. Ideal Connection of Circuit

Build the circuit according to the requirements and diagram in Figure [3.2](#_bookmark17). The final physical connection is shown in Figure [3.3](#_bookmark18).



Figure 3.3. Physical Connection Diagram

# Moisture Detection Test

To confirm that the sensor was working properly, a simple moisture test was conducted. The probe was inserted into a dry plant pot, and the absence of LED illumination indicated a low moisture level. When the probe was inserted into a water cup, the LED turned on, verifying high moisture detection. Additionally, users can adjust the potentiometer to fine-tune sensitivity.

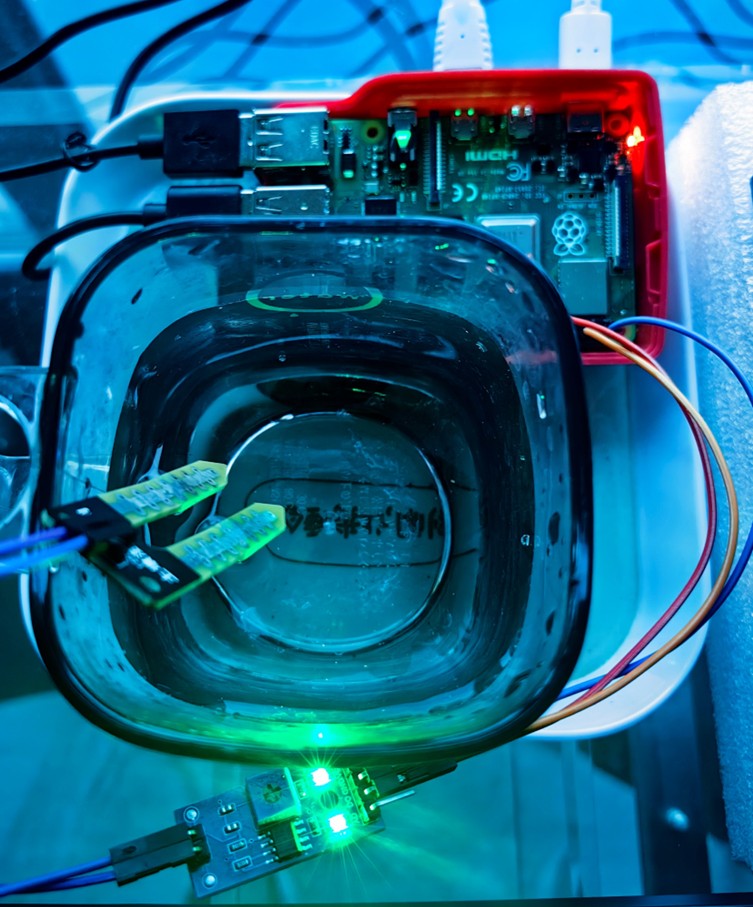


Figure 3.4. Moisture Detection Test

## SOFTWARE DEVELOPMENT METHODOLOGY

# Implementation Overview

The software developed for this project consists of three Python scripts that together form the complete moisture monitoring and notification system. The scripts were written and tested on a Raspberry Pi running Raspberry Pi OS. The primary components include GPIO-based sen- sor data acquisition, condition evaluation logic, and automated email sending through SMTP. These components work together to monitor the plant’s status and deliver real-time updates.

Due to the fact that GPIO pin 4 was already occupied by the OneWire protocol on the Raspberry Pi, the sensor input pin was changed to GPIO17. Additionally, the GPIO library was updated accordingly to ensure compatibility with the new pin configuration.

# Moisture Sensor Test

This section focuses on the initial sensor testing using a standalone Python script. The FC- 28 digital output pin (D0) is connected to GPIO17. The software uses the gpiozero library to monitor the pin status. The GPIO pin was configured to detect whether the soil is dry or wet. The sensor’s LED was used as a visual indicator, and a simple print message was displayed when the moisture level changed, as shown in Listing [1](#_bookmark23).

**Listing 1** Moisture Reading Test

1 from gpiozero import Button

2 import time

3 gpiopin=17

4 class GPIO4InterruptDetector:

5 def init (self):

6 self.button = Button(gpiopin, pull\_up=True, bounce\_time=0.2)

7 # Bind callback functions for press and release events

8 self.button.when\_pressed = self.pin\_low\_callback

**Listing 1** Moisture Reading Test (Continued)

|  |  |  |
| --- | --- | --- |
| 1 |  | self.button.when\_released = self.pin\_high\_callback |
| 2 | def | pin\_low\_callback(self): |
| 3 |  | print("Water Detected") |
| 4 |  |  |
| 5 | def | pin\_high\_callback(self): |
| 6 |  | print("No Water Detected") |
| 7 |  |  |
| 8 | def | run(self): |
| 9 |  | try: |
| 10 |  | # Keep the program running |
| 11 |  | while True: |
| 12 |  | time.sleep(0) |

13 except KeyboardInterrupt:

14 print("Program interrupted by user")

15

16 if name == " main ":

17 detector = GPIO4InterruptDetector()

18 detector.run()

Specifically, when moisture is detected, the message "Water Detected" is printed to the terminal; when no moisture is present, "No Water Detected" is shown. The script continuously runs and updates the sensor status in real time, as shown in Figure [4.1](#_bookmark24).

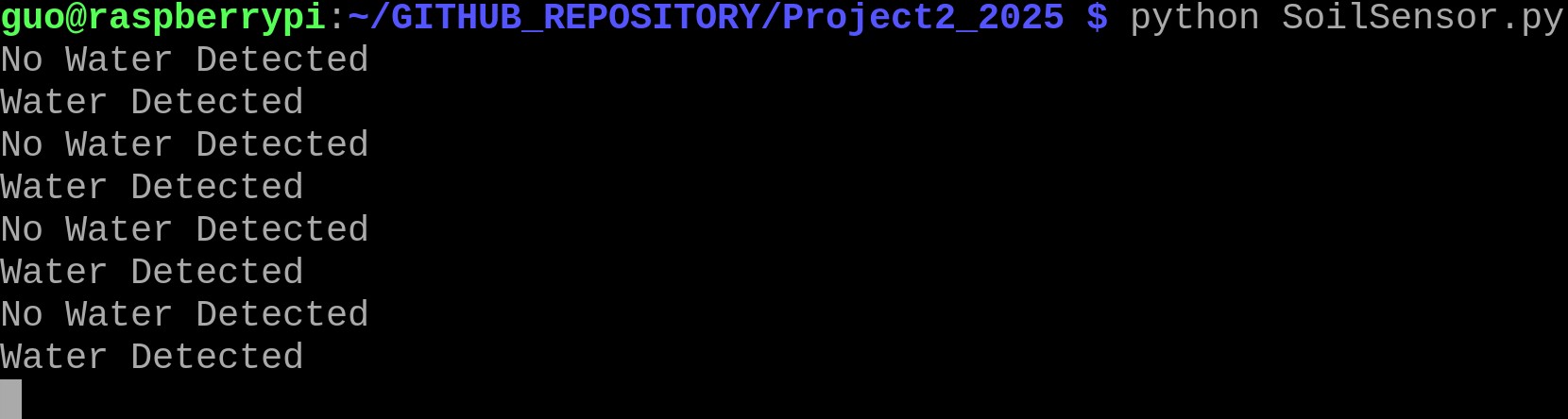


Figure 4.1. Script Testing

After executing the git push command, the terminal displayed a message indicating that all changes were successfully uploaded to the remote repository on GitHub, as shown in Figure [4.2](#_bookmark25).

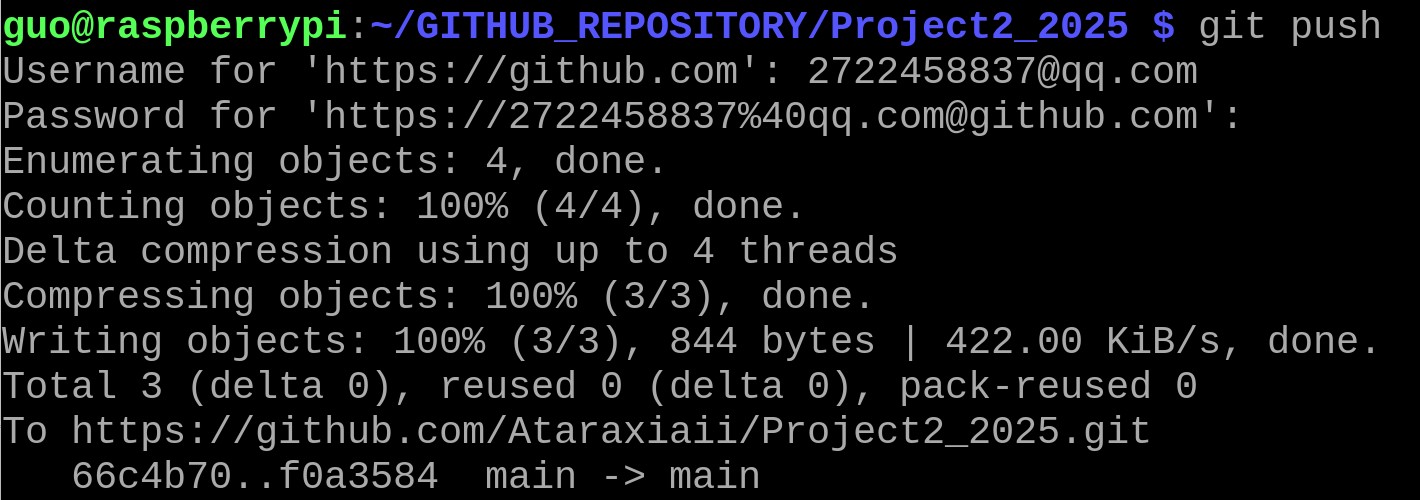


Figure 4.2. Git Push Command Output

# Email Function Test

### Email Configuration Setup

To enable email functionality, a dedicated QQ Mail account was configured. Since QQ Mail requires two-factor authentication for SMTP access, an App Password was generated in the QQ Mail settings (Settings > Account > Enable SMTP and generate app password). This app- specific password replaces the regular login password in the script.



Figure 4.3. App Password Setup

### Email Script Testing

Before integrating with the sensor, the email function was tested independently. The code below demonstrates how an email is constructed and sent using SMTP with QQ Mail.

**Listing 2** Email Sending Test

1 import smtplib

2 from email.message import EmailMessage

3 # Set the sender email and password and recipient email

4 from\_email\_addr = ["1241428411@qq.com](mailto:1241428411@qq.com)"

5 from\_email\_pass = "qpxzanjnrhwejejh"

6 to\_email\_addr = ["2722458837@qq.com"](mailto:2722458837@qq.com)

7 # Create a message object

8 msg = EmailMessage()

9

10 # Set the email body

11 body = "Hello from Raspberry Pi"

12 msg.set\_content(body)

13 # Set sender and recipient

14 msg['From'] = from\_email\_addr

**Listing 2** Email Sending Test

1 msg['To'] = to\_email\_addr

2

3 # Set your email subject

4 msg['Subject'] = 'TEST EMAIL'

5

6 # Connecting to server and sending email

7 # Provider*'*s SMTP server details

8 server = smtplib.SMTP('smtp.qq.com', 587)

9

10 # Use TLS

11 server.starttls()

12

13 # Login to the SMTP server

14 server.login(from\_email\_addr, from\_email\_pass)

15

16 # Send the message

17 server.send\_message(msg)

18 print('Email sent')

19

20 # Disconnect from the server

21 server.quit()

This Python script performs a basic functionality test of the SMTP-based email notification sys- tem. It connects to the QQ Mail SMTP server (smtp.qq.com) over port 587 using the STARTTLS encryption protocol. After establishing a secure connection, the script logs in to the sender email account using an application-specific password. A simple email message is created, containing

the text “Hello from Raspberry Pi” as the body content and “TEST EMAIL” as the subject, as shown in Figure [4.5](#_bookmark30). The message is then sent from the configured sender address to the re- cipient address. Upon successful transmission, a confirmation message “Email sent” is printed on the terminal which is shown in Figure [4.4](#_bookmark29), and the server connection is properly closed.

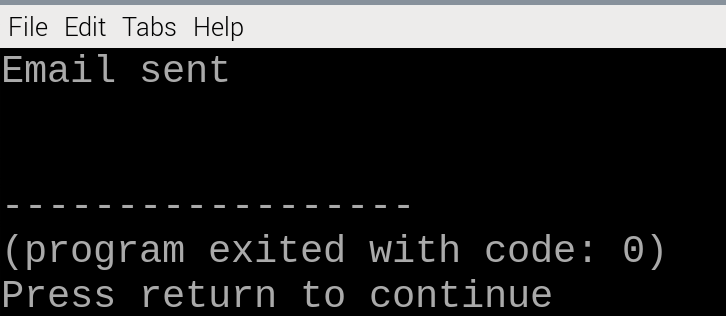


Figure 4.4. Terminal Output



Figure 4.5. Received Email

This test confirms that the SMTP configuration, login credentials, and email transmission func- tionality are all operating correctly before integration with the main system. After testing, I made a successful git push of this script to the main branch, as shown in Figure [4.6](#_bookmark31).

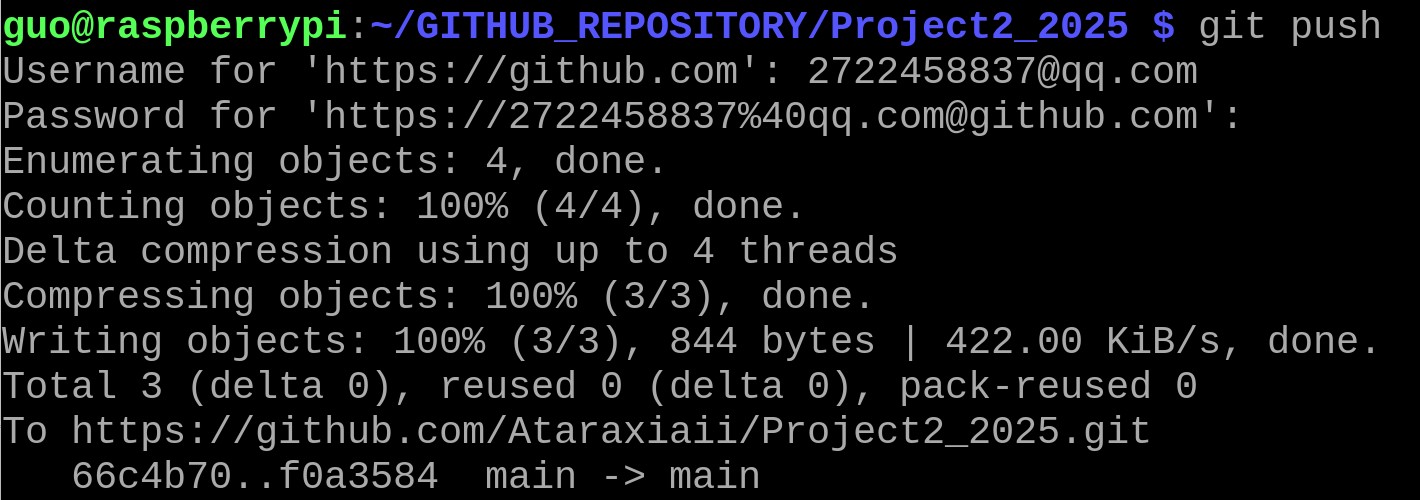


Figure 4.6. Git Push Outcome

# Email Notification with Sensor

After verifying the sensor and email modules independently, they were integrated into a sin- gle script. This code reads the moisture status in real time, stores the status flag, and sends an automated email report four times daily. This section focuses on the development of the scheduled sending module. **The complete source code is provided in Appendix** [**B**](#_bookmark38).

**Listing 3** Scheduled Execution

1 import schedule

2 import time

3 send\_times = ["07:00", "10:00", "13:00", "16:00"]

4 for t in send\_times:

5 schedule.every().day.at(t).do(send\_email)

6 print(f"Scheduled email at {t} every day.")

7

8 while True:

9 schedule.run\_pending()

10 time.sleep(1)

To ensure that users receive updates at regular intervals, the project utilizes the schedule Python library to automate email dispatch. Four specific times each day are configured: 07:00, 10:00, 13:00, and 16:00. Once these times are reached, the send\_email function is called to generate and send the current plant moisture status report automatically without user intervention.

After correctly wiring the circuit, a temporary test mode was configured by adjusting the sched- ule to send an email every two minutes, as shown in Figure [4.7](#_bookmark33). This high-frequency testing helped verify that the system could consistently detect moisture status, construct email content, and dispatch notifications without errors.

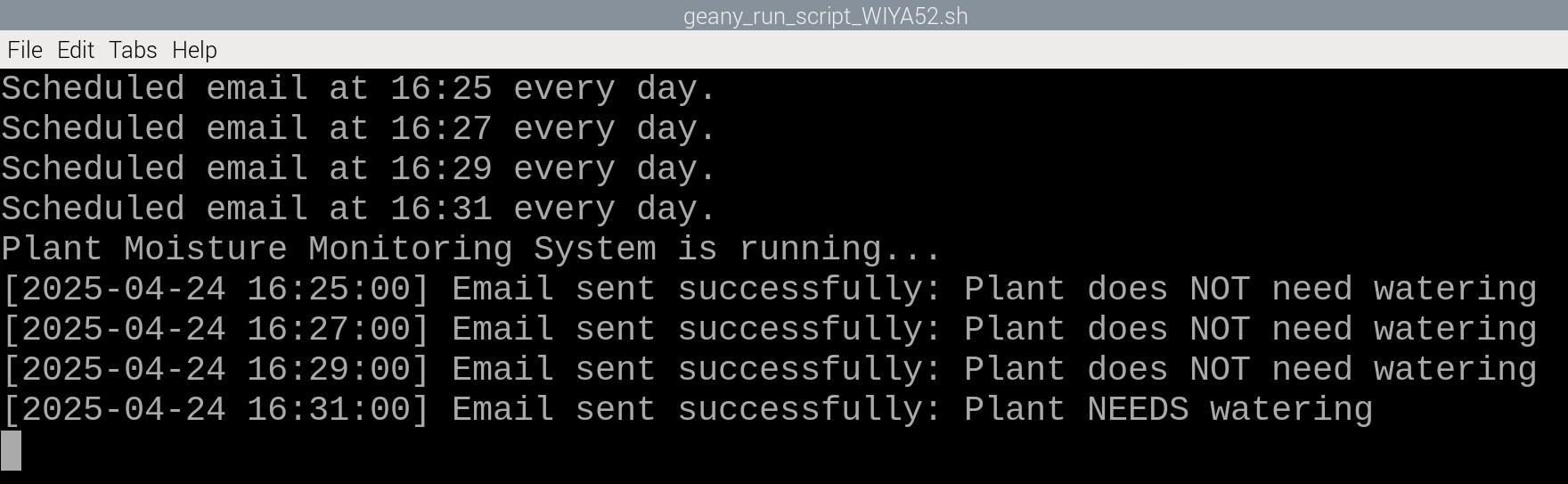


Figure 4.7. Testing Final Script

I received four emails successfully finally, as shown in Figure [4.8](#_bookmark34). The system successfully passed the test, demonstrating reliable operation under intensive sending conditions.

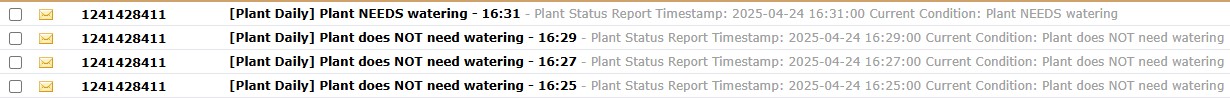


Figure 4.8. Email Receiving Result

After testing, I made a successful git push of this script to the main branch, as shown in Figure [4.9](#_bookmark35).

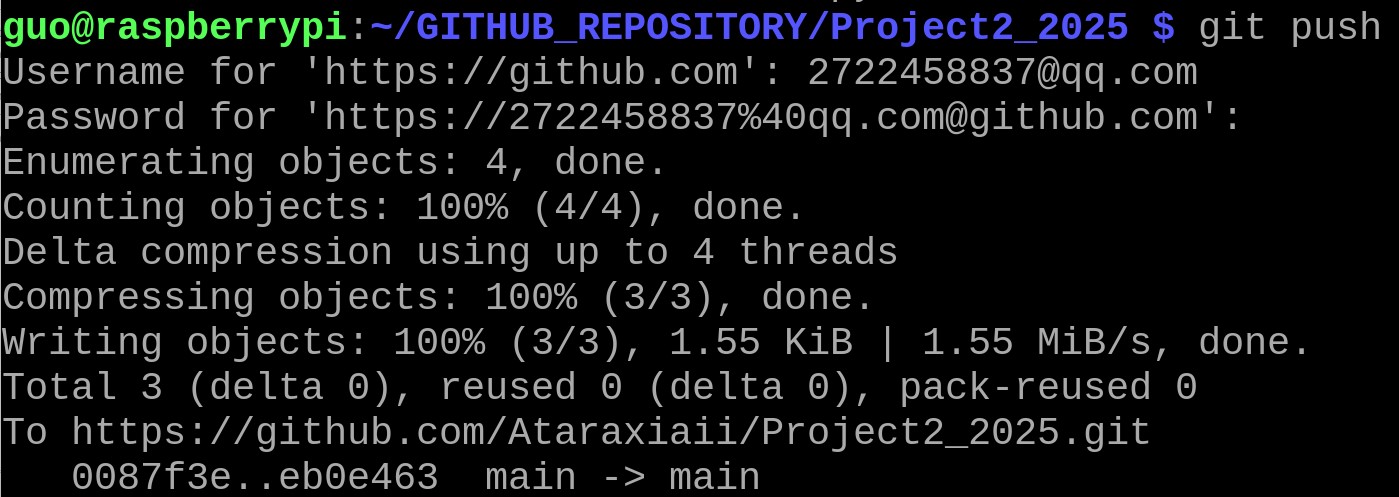


Figure 4.9. Email Receiving Result

# System Testing and Results

After completing hardware connection and software integration, the system was subjected to functional testing. The Raspberry Pi was powered on, and the Python script was executed to continuously monitor soil moisture status. For testing purposes, the email sending schedule was adjusted to dispatch a report every three hours.

During the test period, the system was able to detect changes in soil moisture in real-time and generate correct email notifications. The scheduled emails were delivered successfully to the recipient address without transmission failures. The following screenshots capture a sample of the email notification received and the terminal log output:

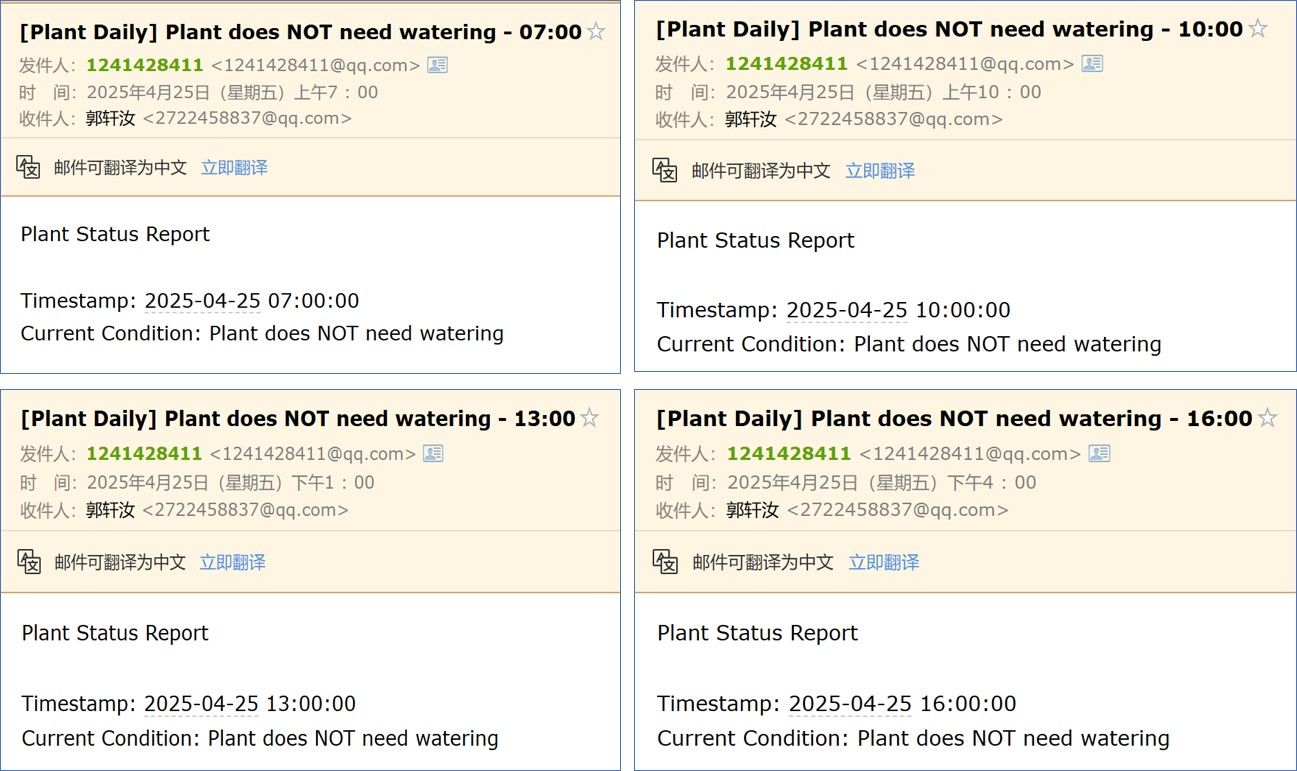


Figure 4.10. First Day Email

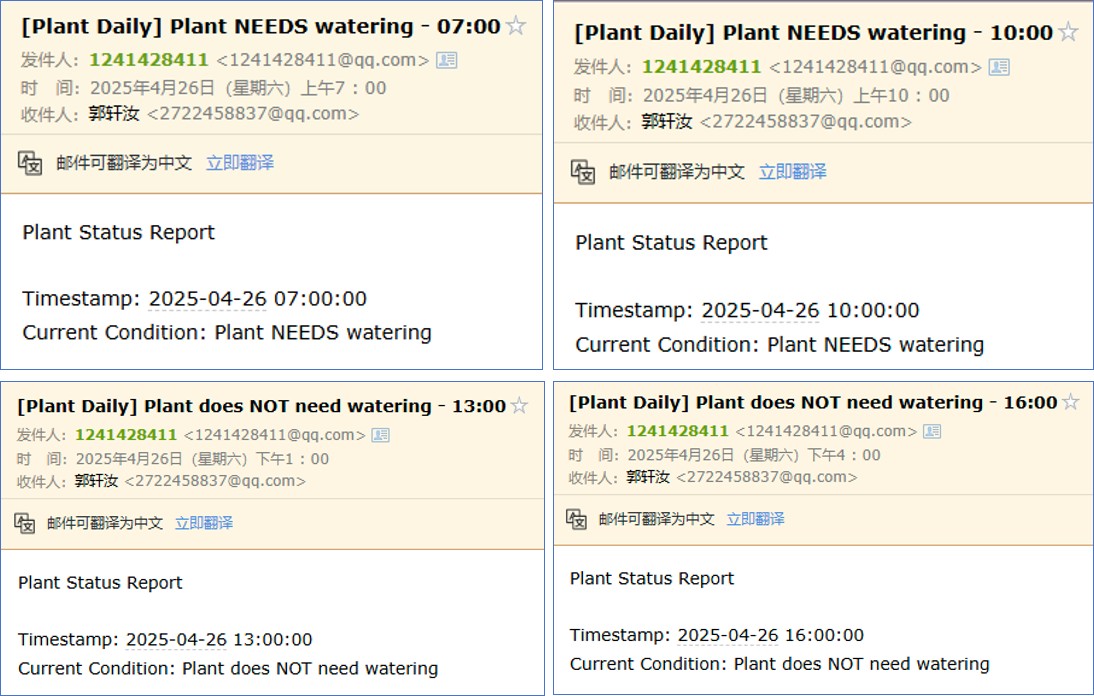


Figure 4.11. Second Day Email

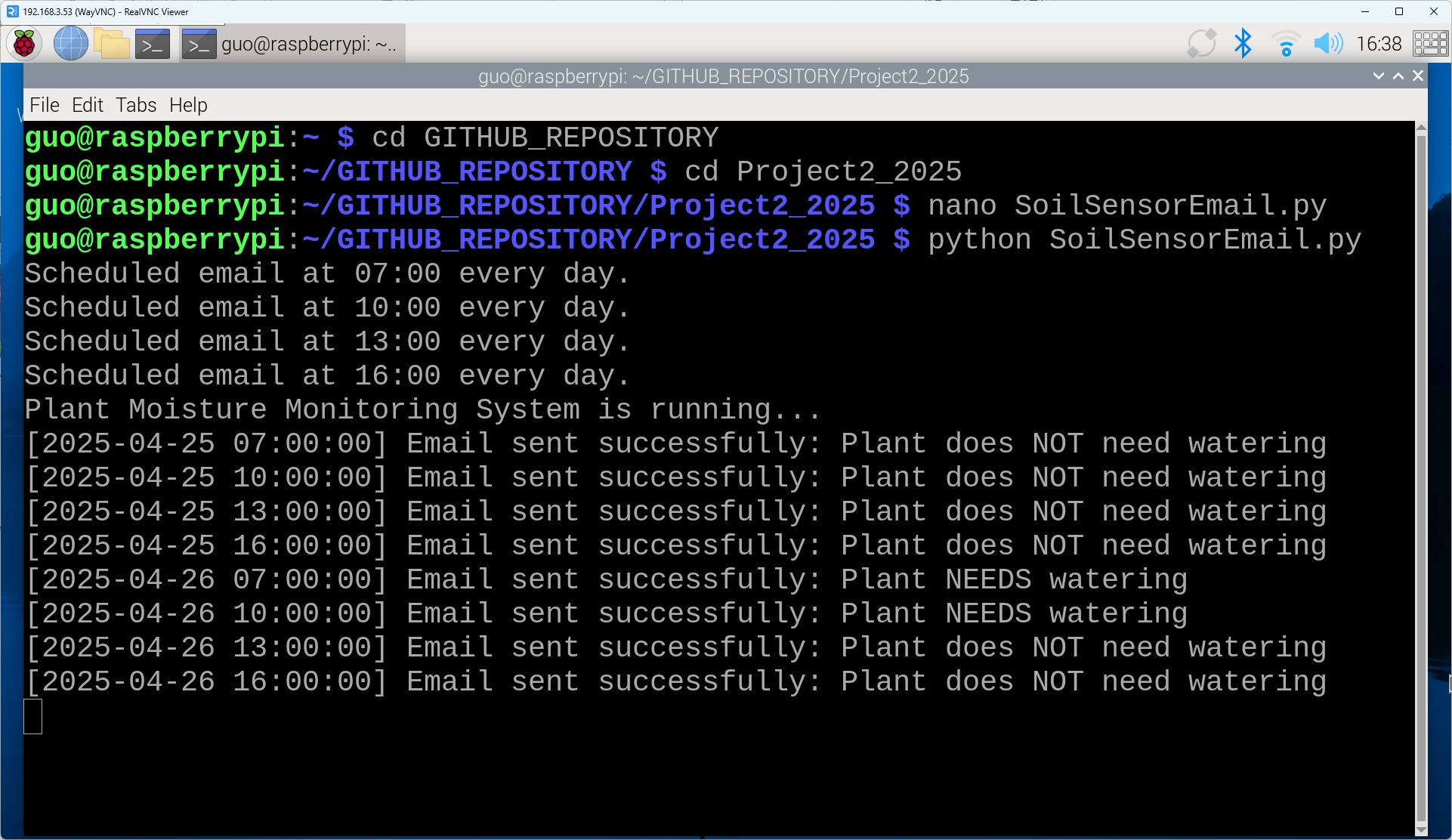
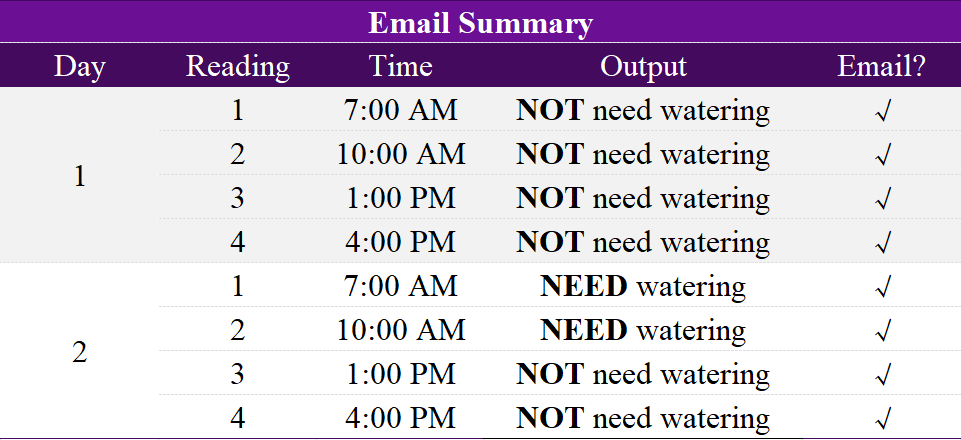


Figure 4.12. Terminal Log Output

Table below shows the email summary table collected over two days of system operation. Each day was divided into four scheduled readings at 7:00 AM, 10:00 AM, 1:00 PM, and 4:00 PM. The "Output" column records the plant’s moisture status determined by the sensor at each time point, while the "Email?" column indicates whether an email was successfully sent. On Day 1, all four readings reported "NOT need watering," indicating sufficient soil moisture. On Day 2, the readings at 7:00 AM and 10:00 AM showed "NEED watering," suggesting the soil had dried overnight. Afterward, the moisture returned to a healthy level by the afternoon. This demonstrates that the system can accurately detect moisture changes and send real-time notifications according to schedule.



## COMMIT HISTORY

Appendix A shows the git commits of this project. There are four commits in total.

commit eb0e463da6085498424076ef7ae427703744dc76 (HEAD -> main, origin/main, origin/HEAD)

Author: Xuanru Guo <[2722458837@qq.com](mailto:2722458837@qq.com)>

Date: Thu Apr 24 16:34:54 2025 +0800

create SoilSensorEmail.py – schedule four daily moisture checks

commit 0087f3ed0ad9b7cf4d9811bb792d0d3d98dd7de7 Author: Xuanru Guo <[2722458837@qq.com](mailto:2722458837@qq.com)>

Date: Wed Apr 23 14:59:41 2025 +0100

first commit send\_email.py – create email test script

commit f0a3584bc9971eaf35a2aa4bfa90df5d71e70f3f Author: Xuanru Guo <[2722458837@qq.com](mailto:2722458837@qq.com)>

Date: Wed Apr 23 14:47:55 2025 +0100

create Soilsensor.py – perform initial sensor test

commit 66c4b70a96d8dfe61cfe9ce3a117d277a43dd550

Author: Xuanru GUO <[121843241+Ataraxiaii@users.noreply.github.com](mailto:121843241%2BAtaraxiaii@users.noreply.github.com)>

Date: Mon Apr 14 11:35:21 2025 +0800

Initial commit

## SOURCE CODE

**Listing 4** Final Source Code

1 # Program: Plant Moisture Sensor with Email Notification

2 # Author: Xuanru Guo

3 # Student Number: W20109677

4 # Date: 23/4/2025

5 # Description: send daily email reports using GPIO moisture sensor input

6 # Course & Year: Project Semester 3

7

8 import smtplib

9 from email.message import EmailMessage

10 from gpiozero import Button

11 from datetime import datetime

12 import schedule

13 import time

14

15 # ===== Email Configuration =====

16 FROM\_EMAIL = ["1241428411@qq.com"](mailto:1241428411@qq.com)

17 FROM\_PASSWORD = "qpxzanjnrhwejejh" # QQ Mail app password

18 TO\_EMAIL = "[2722458837@qq.com"](mailto:2722458837@qq.com)

19 SMTP\_SERVER = "smtp.qq.com"

20 SMTP\_PORT = 587

21

22 # ===== Moisture Sensor Configuration =====

23 MOISTURE\_PIN = 17

24 sensor = Button(MOISTURE\_PIN, pull\_up=True, bounce\_time=0.2)

25

26 # ===== Water Status (updated in real-time) =====

27 # True = Dry (no water detected): watering needed

28 # False = Wet (water detected): no watering needed

29 water\_needed = not sensor.is\_pressed # Initial status on startup

**Listing 4** Final Source Code (Continued)

1 # Callback when water is detected (button pressed)

2 def on\_water\_detected():

3 global water\_needed

4 water\_needed = False

5 # print("Water detected: Plant does NOT need watering.")

6

7 # Callback when no water is detected (button released)

8 def on\_no\_water():

9 global water\_needed

10 water\_needed = True

11 # print("No water detected: Plant NEEDS watering.")

12

13 # Attach sensor callbacks

14 sensor.when\_pressed = on\_water\_detected

15 sensor.when\_released = on\_no\_water

16

17 # ===== Email Sending Function =====

18 def send\_email():

19 now = datetime.now()

20 status = "Plant NEEDS watering" if water\_needed

21 else "Plant does NOT need watering"

22 timestamp = now.strftime('%Y-%m-%d %H:%M:%S')

23

24 body = f"""Plant Status Report

25

26 Timestamp: {timestamp}

27 Current Condition: {status}

28 """

29

30 msg = EmailMessage()

31 msg.set\_content(body)

32 msg['From'] = FROM\_EMAIL

33 msg['To'] = TO\_EMAIL

34 msg['Subject'] = f"[Plant Daily] {status} - {now.strftime('%H:%M')}"

35

36 server = smtplib.SMTP(SMTP\_SERVER, SMTP\_PORT)

37 try:

38 server.starttls()

39 server.login(FROM\_EMAIL, FROM\_PASSWORD)

40 server.send\_message(msg)

41 print(f"[{timestamp}] Email sent successfully: {status}")

42 except Exception as e:

43 print(f"[{timestamp}] Error during email send: {e}")

44 finally:

45 server.quit()

**Listing 4** Final Source Code (Continued)

1 # ===== Schedule Email Times =====

2 send\_times = ["07:00", "10:00", "13:00", "16:00"]

3 for t in send\_times:

4 schedule.every().day.at(t).do(send\_email)

5 print(f"Scheduled email at {t} every day.")

6

7 # ===== Main Loop =====

8 print("Plant Moisture Monitoring System is running...")

9

10 try:

11 while True:

12 schedule.run\_pending()

13 time.sleep(1)

14 except KeyboardInterrupt:

15 print("Program manually terminated.")