A

Project Report

On

Automated Garden

Submitted by

Akhilesh Mangesh Sathe

Roll No.: 22356

MCA–I

SEM–II

Under the guidance of

Prof. Meenakshi Jadhav

For the Academic Year 2022-23

Sinhgad Institutes

Sinhgad Technical Education Society's

Sinhgad Institute of Management

Vadgaon Bk Pune 411041

(Affiliated to SPPU Pune & Approved by AICTE New Delhi)

Date:

CERTIFICATE

This is to certify that Mr Akhilesh Mangesh Sathe has successfully completed his project work entitled **"Automated Garden"** in partial fulfillment of MCA – I SEM –II Mini Project for the year 2022-2023. He has worked under our guidance and direction.

Prof. Meenakshi Jadhav
Project Guide

Dr. Chandrani Singh
Director, SIOM-MCA

Examiner 1

Examiner 2

Date:

Place: Pune

DECLARATION

I certify that the work contained in this report is original and has been done by meunder the guidance of my supervisor(s).

- The work has not been submitted to any other Institute for any degree or diploma.
- I have followed the guidelines provided by the Institute in preparing the report.
- I have conformed to the norms and guidelines given in the Ethical Code ofConduct of the Institute.
- Whenever I have used materials (data, theoretical analysis, figures, and text) fromother sources, I have given due credit to them by citing them in the text of the report and giving their details in the references.

Name and Signature of Project Team Members*:*

Sr. No.	**Seat No.**	**Name of students**	**Signature of students**
1		**Akhilesh Mangesh Sathe**	

ACKNOWLEDGEMENT

It is very difficult task to acknowledge all those who have been of tremendous help in this project. I would like to thank my respected guide **Prof. Meenakshi Jadhav** for providing me necessary facilities to complete my project and also for their guidance and encouragement in completing my project successfully without which it wouldn't be possible. I wish to convey my special thanks and immeasurable feelings of gratitude towards **Dr. Chandrani Singh, Director SIOM-MCA.** I wish to convey my special thanks to all teaching and non-teaching staff members of **Sinhgad Institute of Management, Pune** for their support.

Thank You

Yours Sincerely,

Akhilesh Mangesh Sathe

INDEX

CHAPTER 1: INTRODUCTION

1.1Abstract

The Automated Garden project aims to provide an innovative solution for plant care by automating the watering process based on real-time data from soil moisture and temperature sensors. By combining the power of Raspberry Pi, sensor technology, and Python programming, users can create a smart and efficient garden management system.

Traditionally, garden watering is often done manually or based on predetermined schedules, which may not take into account the specific needs of each plant or the prevailing environmental conditions. This manual approach can lead to overwatering or underwatering, resulting in poor plant health and wastage of water resources.

The Automated Garden system addresses these challenges by continuously monitoring the soil moisture levels and ambient temperature in the garden. The soil moisture sensor accurately measures the moisture content in the soil, providing crucial information about when watering is required. The temperature sensor helps in understanding the environmental conditions that affect plant growth.

1.2 Existing System and Need for System

At present, there may be manual methods of watering plants or basic irrigation systems that do not consider real-time soil moisture data. The Automated Garden system offers an automated and intelligent approach to watering plants, leveraging sensor data and advanced control algorithms for efficient garden management.

1.3 Scope of System

1. Automated Watering: Develop a system that automatically waters plants based on real-time soil moisture readings. The system should detect when the soil becomes dry and trigger the watering mechanism to ensure plants receive adequate hydration.
2. Sensor Integration: Connect and calibrate the soil moisture sensor and temperature sensor to accurately measure the moisture content and ambient temperature in the garden. Ensure the sensors are properly integrated with the Raspberry Pi for data collection.
3. Watering Control: Develop a Python script that reads sensor data and controls the watering system. The script should analyze the soil moisture readings and activate the water pump or irrigation system when the moisture level falls below a predetermined threshold.
4. User Interface: Implement a user interface, either web-based or desktop-based, that allows users to monitor and manage the garden system. The interface should display real-time sensor data, provide options to set watering schedules, and offer manual control over the watering process if desired.
5. Data Logging and Analysis: Incorporate data logging capabilities to record sensor readings and watering events over time. This will enable users to analyze the data and gain insights into plant behavior, moisture patterns, and watering efficiency.

1.4 Operating Environment Hardware and Software

Hardware Requirements

- Processor: Pentium IV or above
- RAM: 1 GB or above
- HDD:5 GB or above
- Raspberry Pi: A single-board computer used as the main control unit.
- Soil Moisture Sensor: Measures the moisture level in the soil.
- Temperature Sensor: Measures the ambient temperature in the garden.

Software Requirement

- Operating System: Windows, Linux or MacOS
- A browser which supports JavaScript

1.5 Brief Description of Technology used

- Front End: HTML, CSS, JavaScript,
- Back End: Python Django
- Database: Sqlite3

CHAPTER 2: PROPOSED SYSTEM

2.1 Feasibility Study

Feasibility Study: The Automated Garden Project

1. Technical Feasibility:
    - Raspberry Pi: The use of Raspberry Pi as the central control unit provides a reliable and flexible platform for the project.
    - Sensor Technology: Soil moisture and temperature sensors are readily available and can be easily integrated with the Raspberry Pi.
    - Python Programming: Python is a widely-used programming language with extensive libraries and resources for IoT applications, making it suitable for developing the automation logic.
2. Economic Feasibility:
    - Cost of Components: The cost of Raspberry Pi, sensors, and other necessary components is relatively affordable and within reach for most garden enthusiasts.
    - Maintenance and Upkeep: The Automated Garden system requires periodic maintenance and occasional sensor calibration, which can be performed by the user without significant expense.
3. Environmental Feasibility:
    - Water Conservation: By automating the watering process based on soil moisture levels, the Automated Garden system promotes water conservation by avoiding overwatering and ensuring efficient water usage.
    - Energy Efficiency: Raspberry Pi is known for its low power consumption, making it an energy-efficient choice for running the garden automation system.
4. Operational Feasibility:
    - User-Friendly Interface: The system can be designed with a user-friendly interface, allowing users to easily set up and customize watering schedules, view sensor data, and receive notifications.
    - Integration with Existing Gardens: The Automated Garden system can be implemented in both existing and new gardens, providing flexibility and convenience for users.
    - Scalability: The system can be scaled up or expanded to accommodate multiple sensors, zones, or different types of plants in larger gardens.

In conclusion, the Automated Garden project demonstrates strong technical feasibility, economic viability, and environmental benefits. With careful design and implementation, it has the potential to provide an efficient and convenient solution for automated plant care, enhancing plant health and water conservation.

2.2 Objectives of the proposed system

The main objectives of the Automated Garden project are as follows:

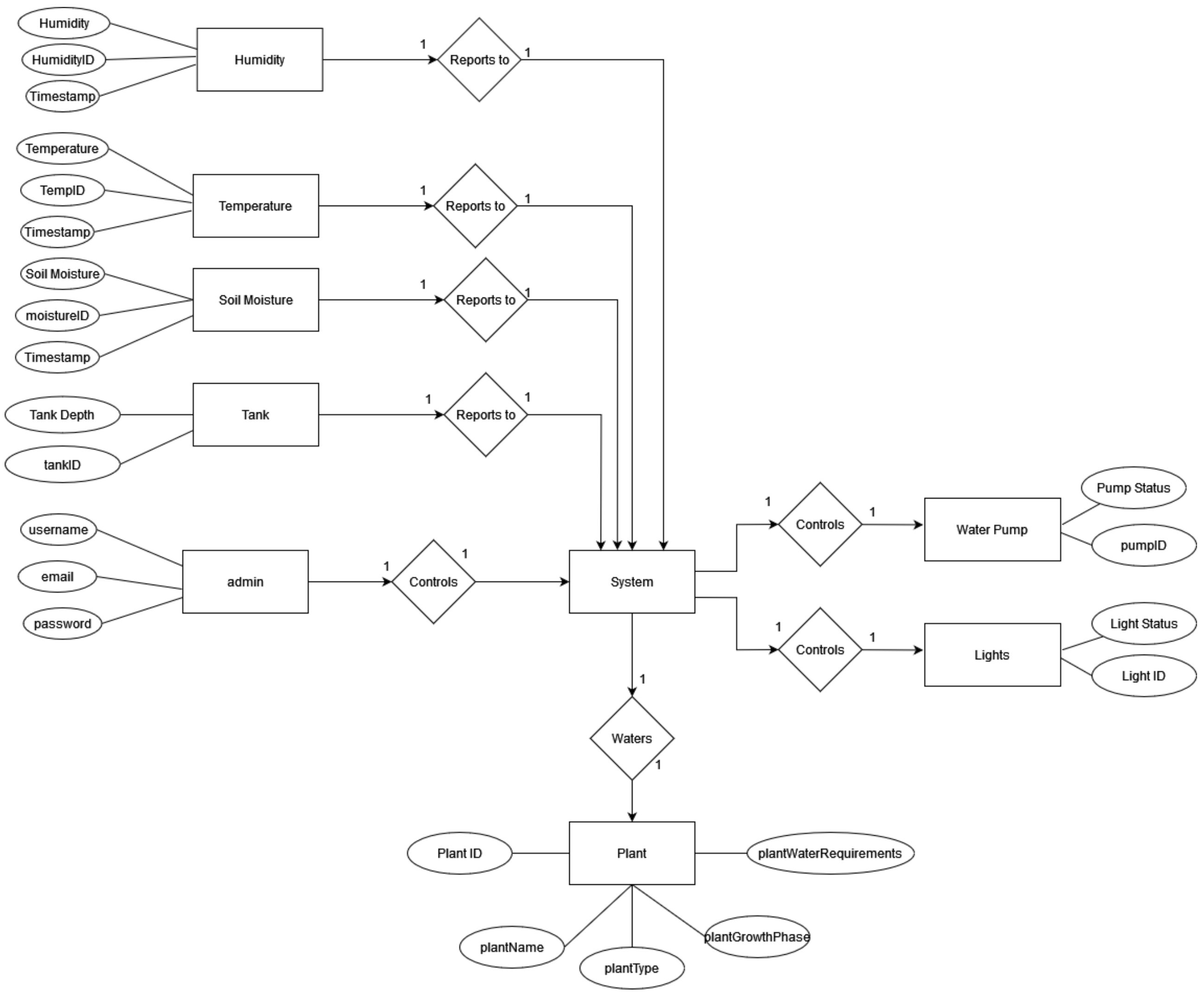
- Develop a system to monitor soil moisture levels and temperature in a garden.
- Automate the process of watering plants when the soil becomes dry.
- Provide a user-friendly interface to control and monitor the garden system.
- Improve plant health and growth
- Ensuring proper watering based on environmental conditions.

2.3 Users of the system

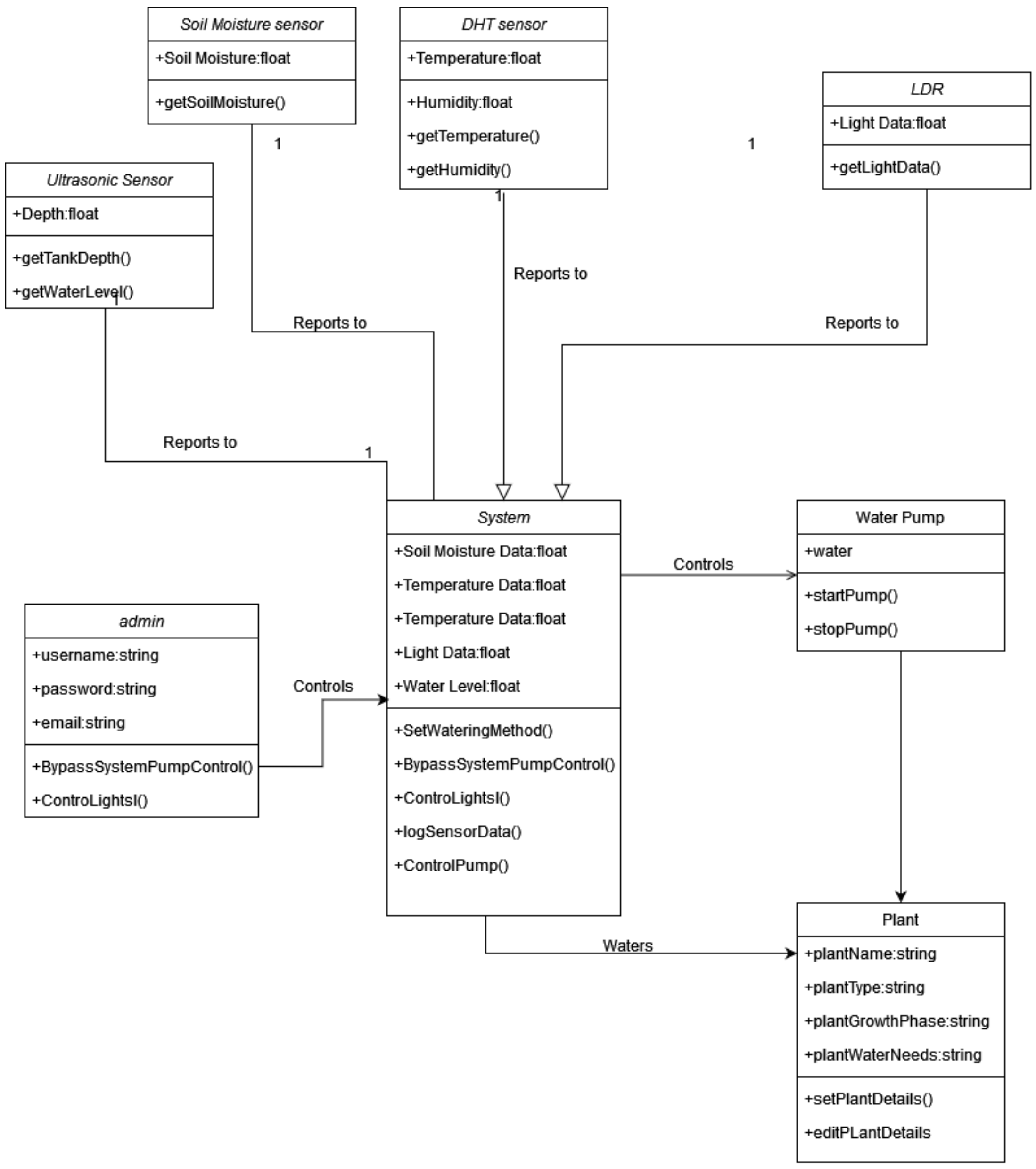
- User
  The user is the owner of the garden
  The user is able to add plant to the system
  The user is able to choose watering method
  The user is able to see status of the plant
  The user is able to visualize the stats of the plants

CHAPTER 3: ANALYSIS AND DESIGN

3.1 Entity Relationship Diagram (ERD)

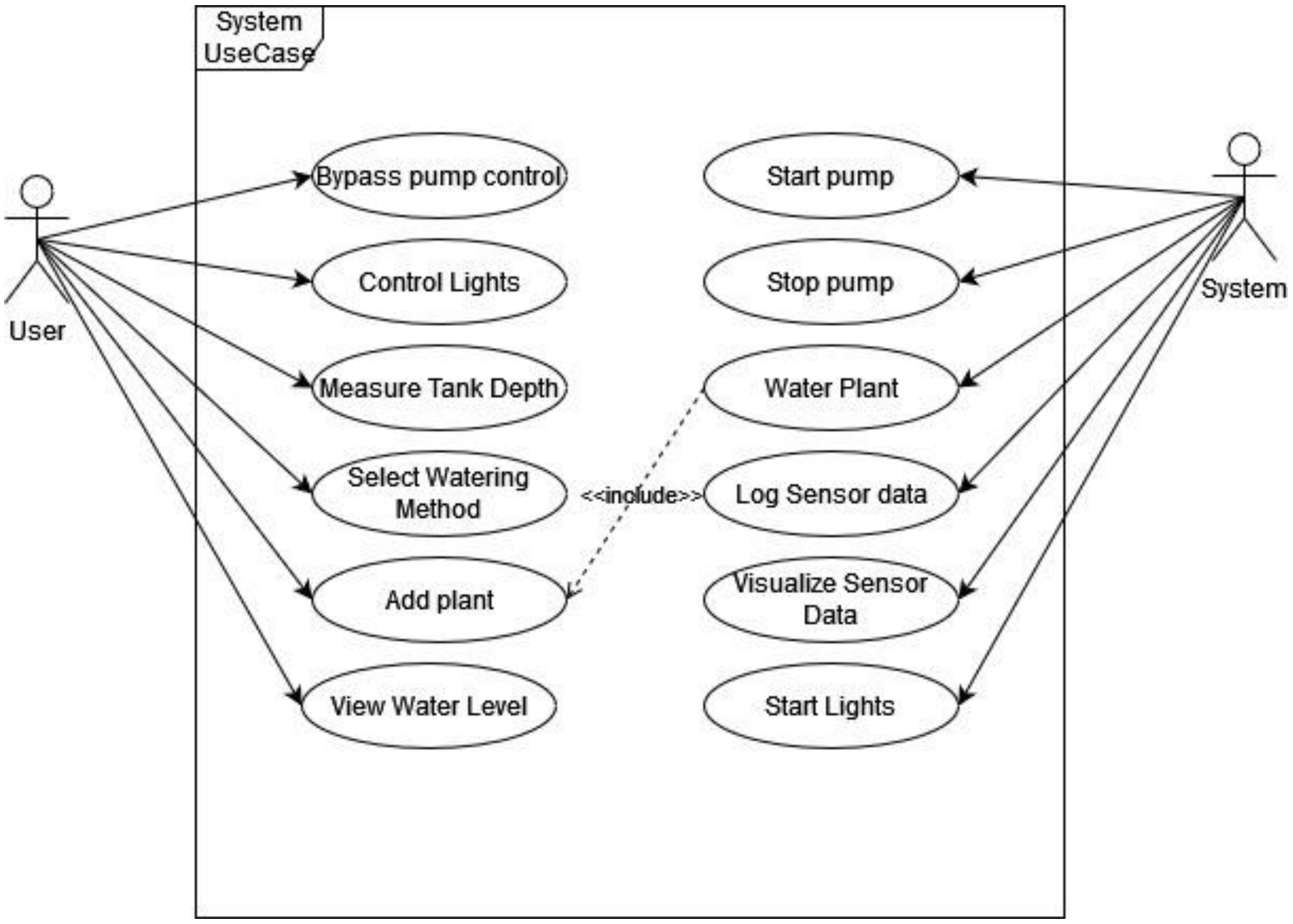


3.2 Class Diagram

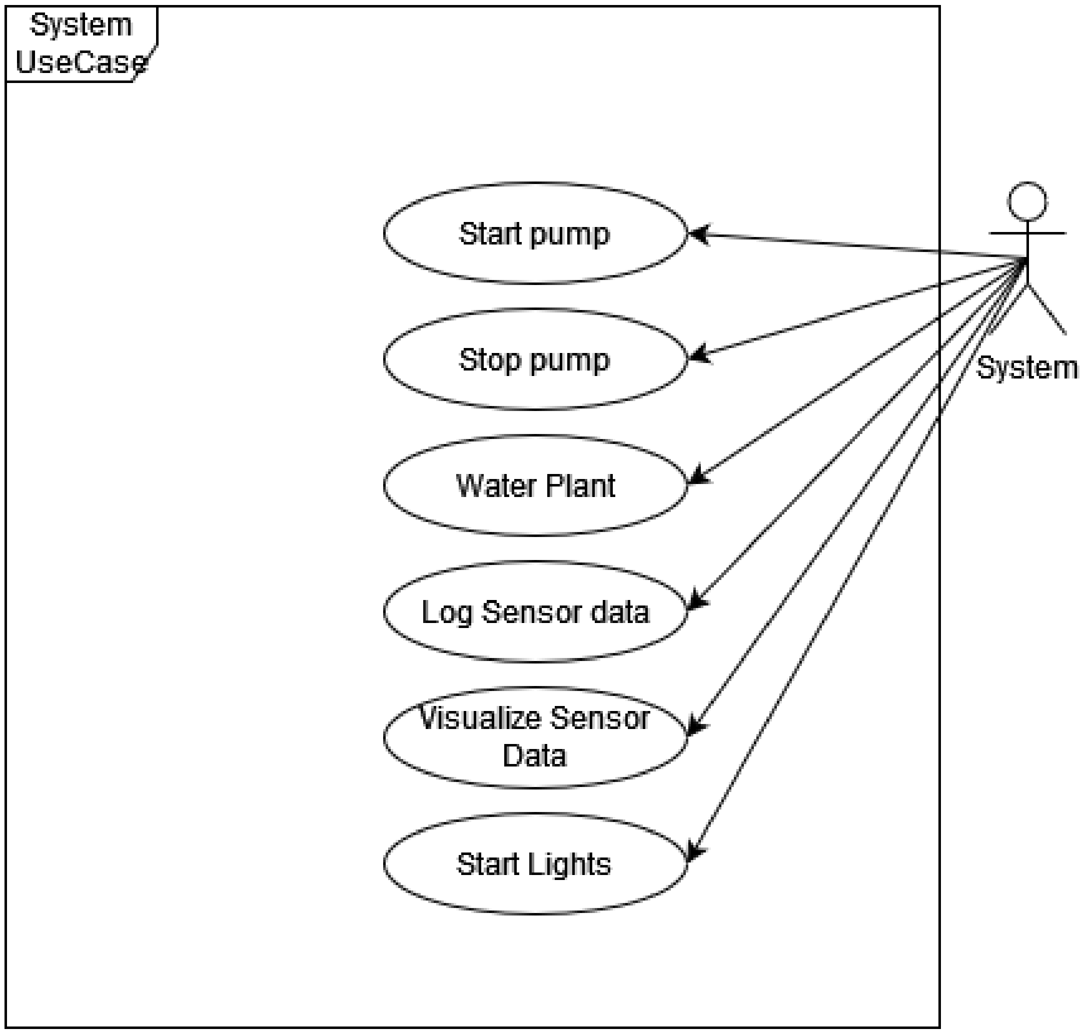


3.3 Use Case Diagrams

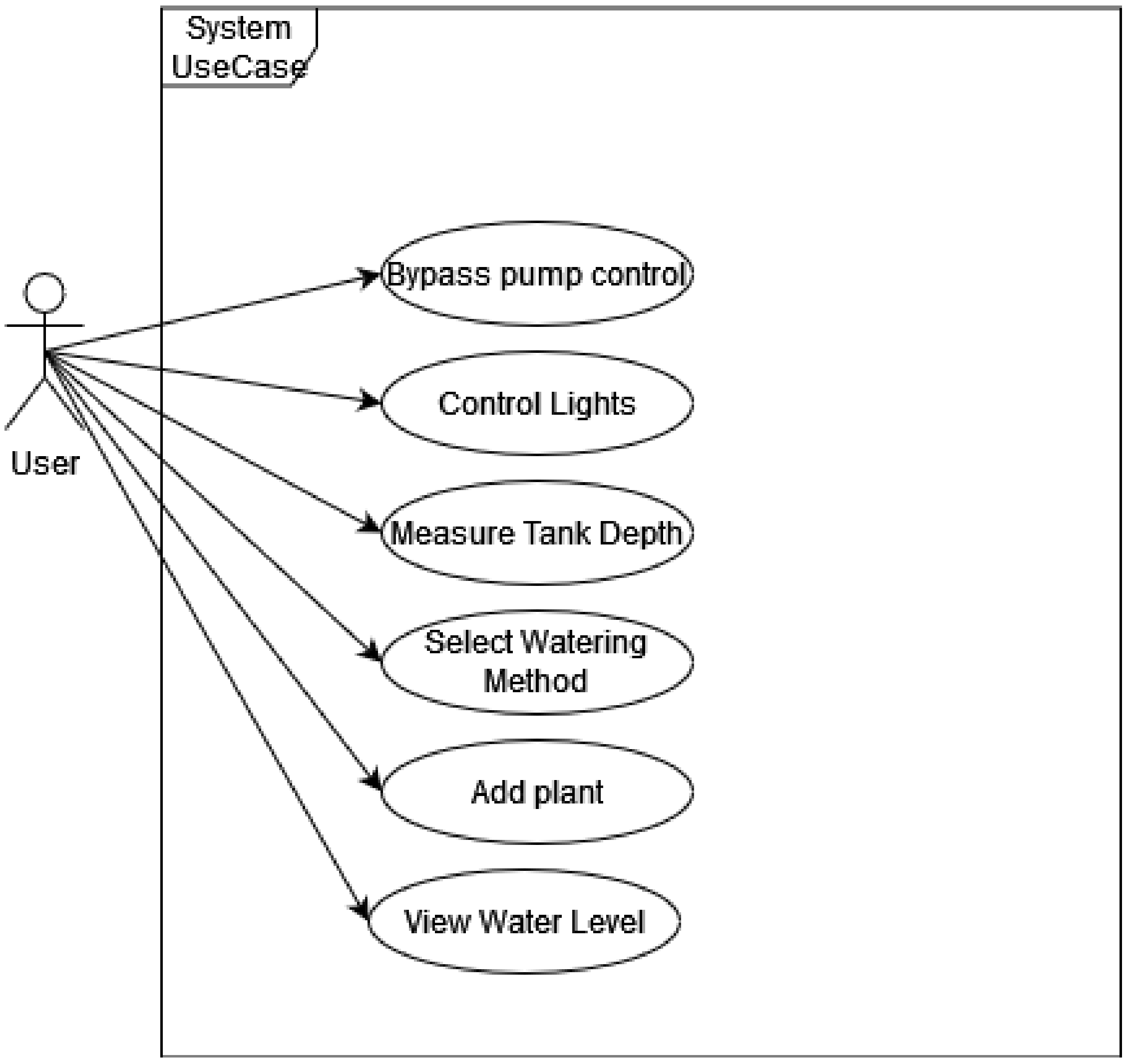
System Use Case Diagram:



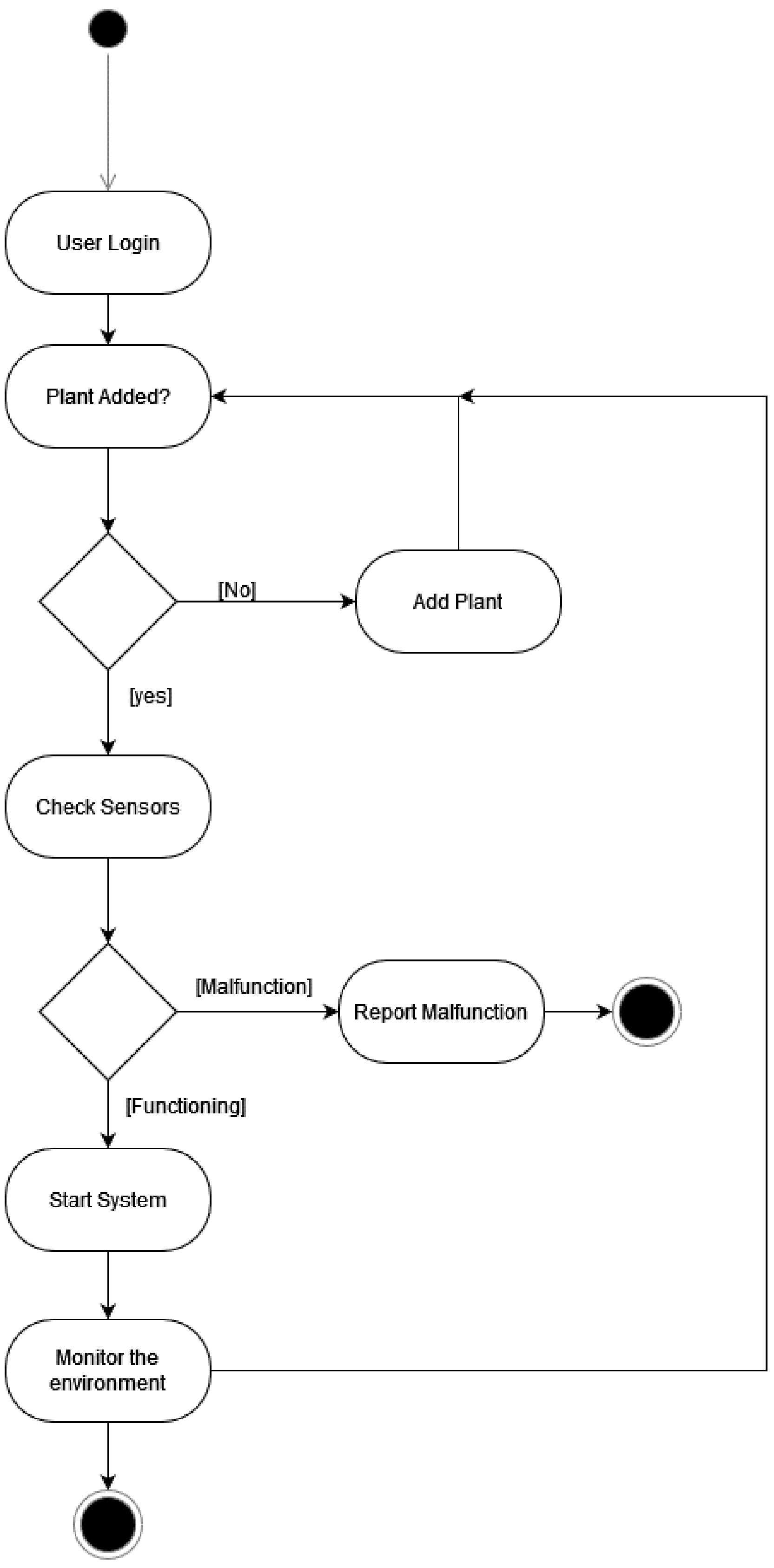
User Use Case Diagram:



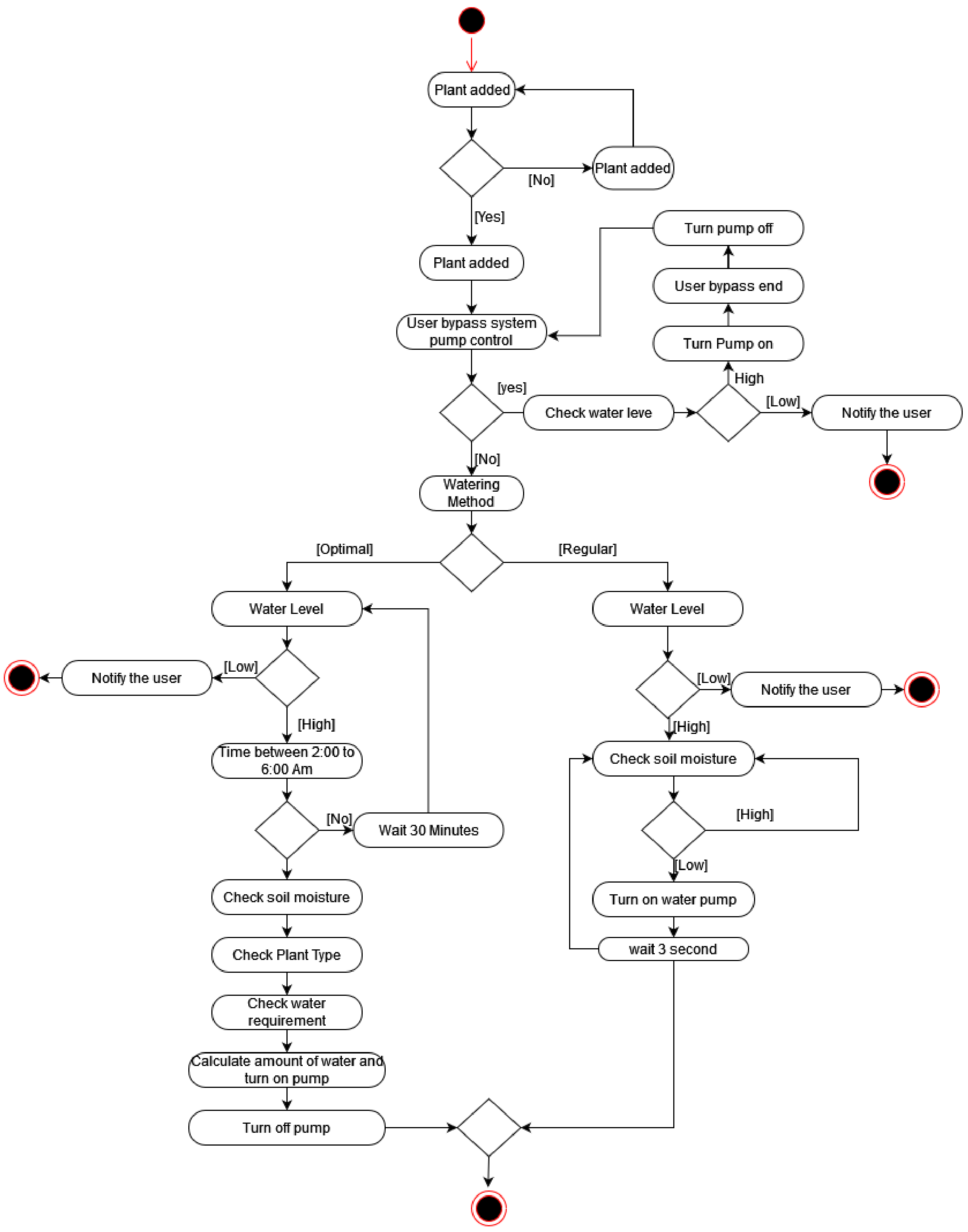
Parent Use Case Diagram:



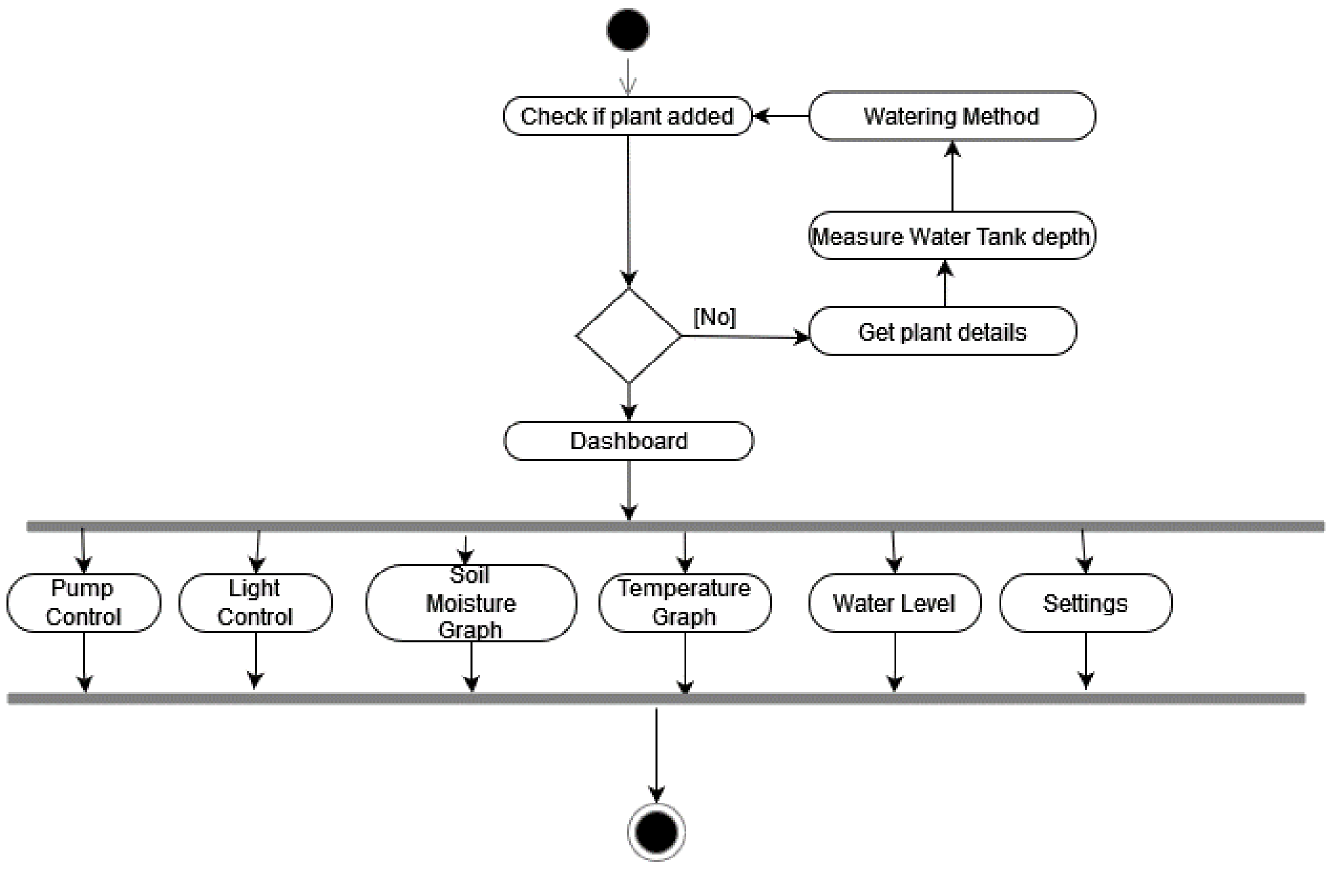
3.4 Activity Diagram



Water Supply Activity Diagram:

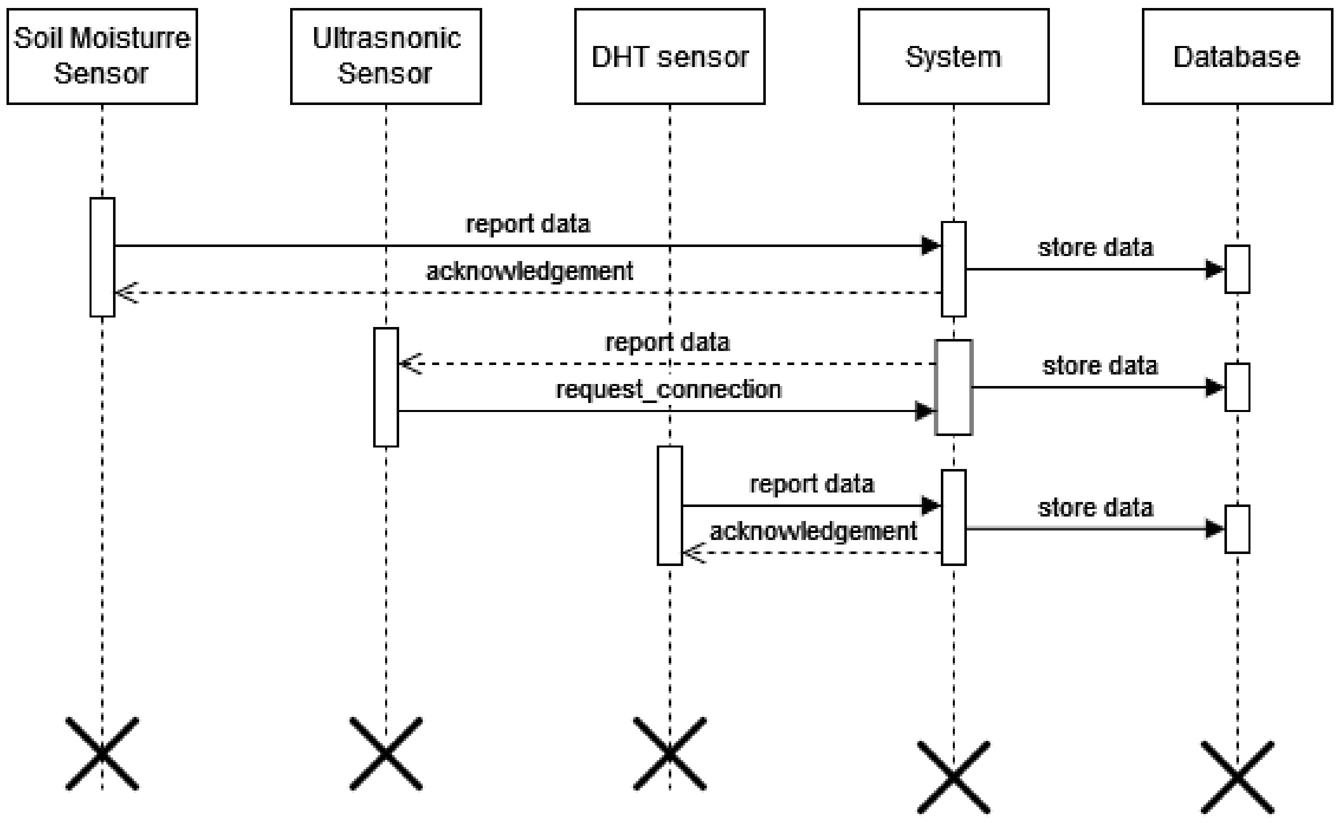


User Activity Diagram:

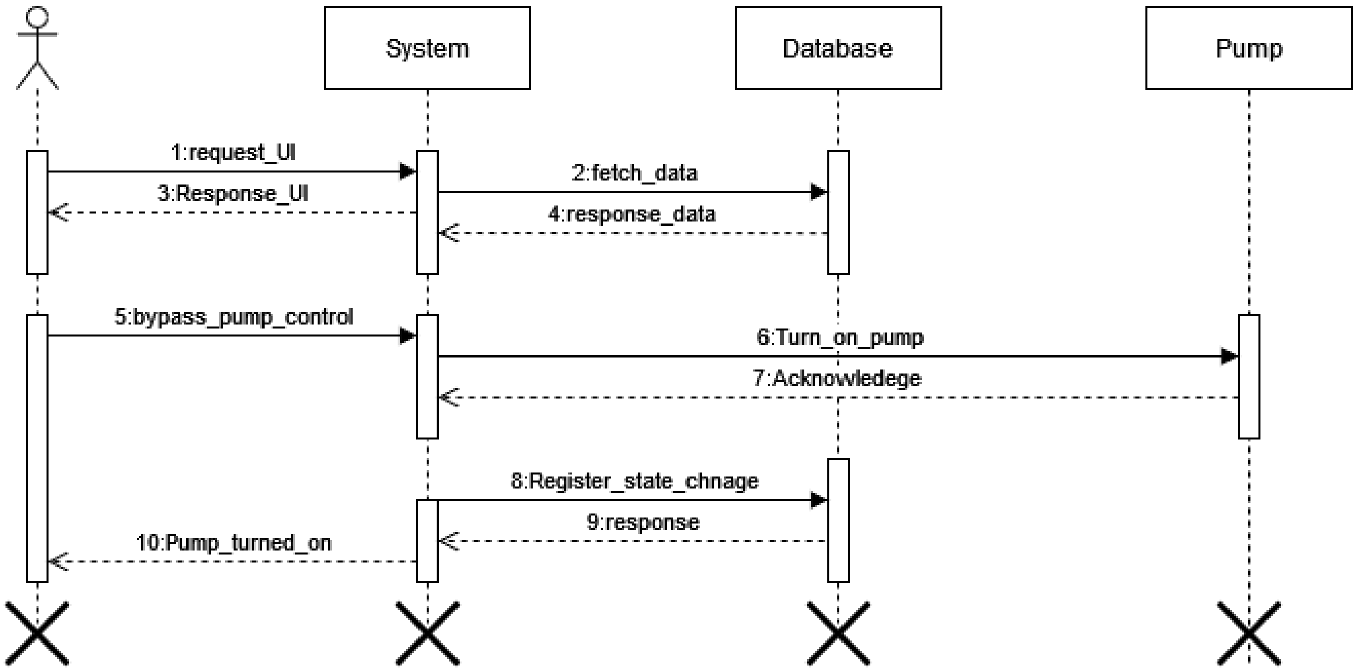


3.5 Sequence Diagram

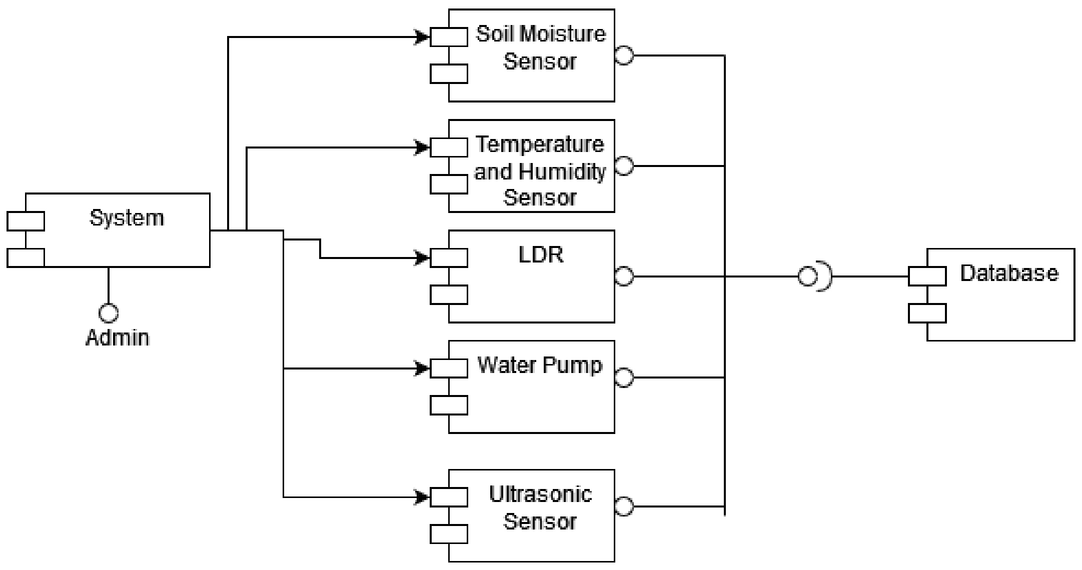
Sensor Sequence Diagram:



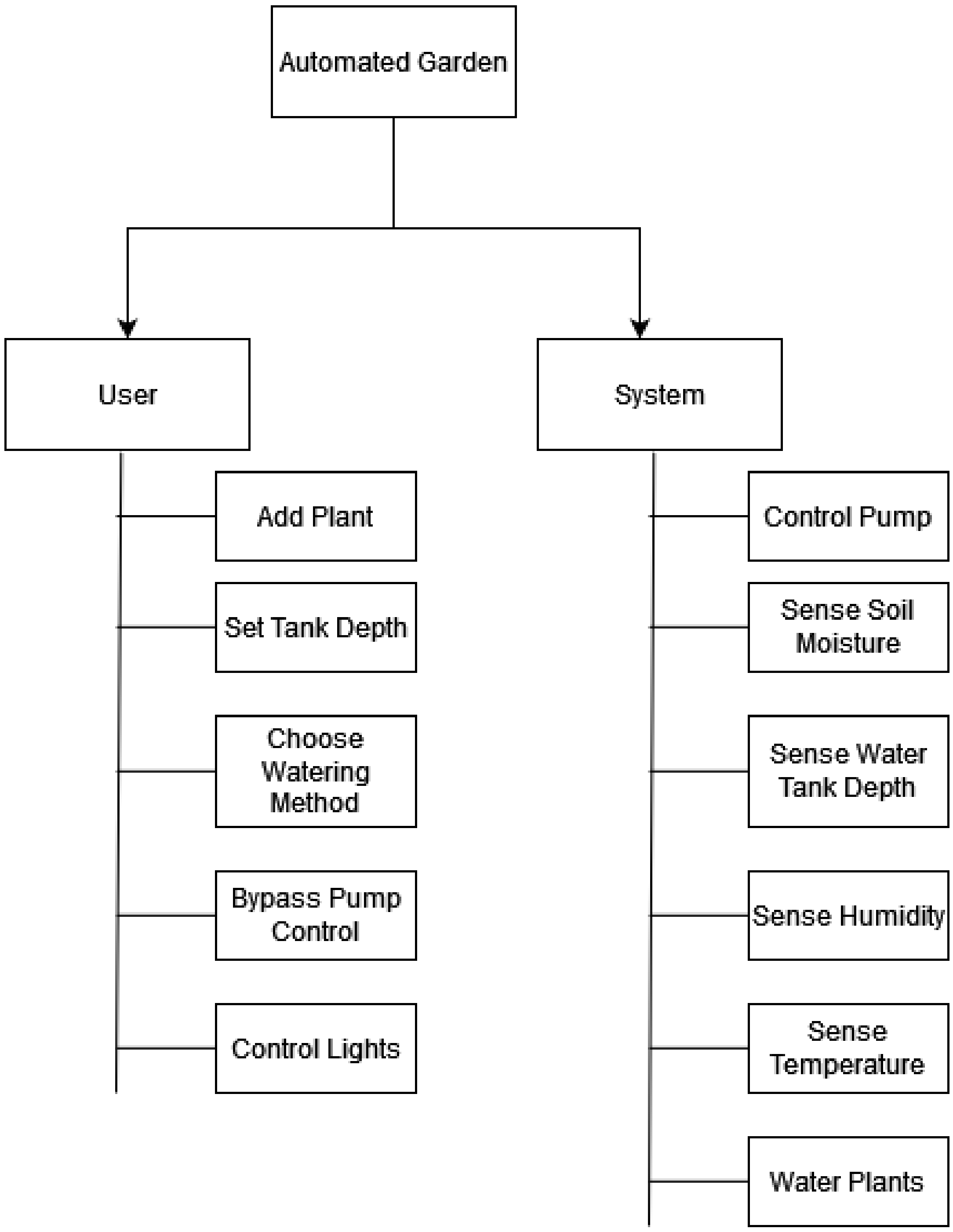
User Sequence Diagram:



3.6 Component Diagram



3.7 Module and Hierarchy Diagram



3.8 Table Design

3.9 Data Dictionary

Plant

Column	Data Type	Description
id	AutoField	Primary key for the Plant table
name	CharField	Name of the plant
plant_type	CharField	Type of the plant (herb, shrub, perennial, etc.)
growth_phase	CharField	Growth phase of the plant
water_requirement	CharField	Level of water requirement for the plant
placement	CharField	Placement of the plant (indoors or outdoors)

Tank

Column	Data Type	Description
id	AutoField	Primary key for the Tank table
value	FloatField	Value representing the tank measurement

Watering Method

Column	Data Type	Description
id	AutoField	Primary key for the WateringMethod table
method	CharField	Watering method (e.g., regular)

Humidity

Column	Data Type	Description
value	FloatField	Humidity value
time	DateTimeField	Timestamp for the humidity data

Soil Moisture

Column	Data Type	Description
value	FloatField	Soil Moisture value
time	DateTimeField	Timestamp for the Soil Moisture data

Temperature

Column	Data Type	Description
value	FloatField	Temperature value
time	DateTimeField	Timestamp for the temperature data

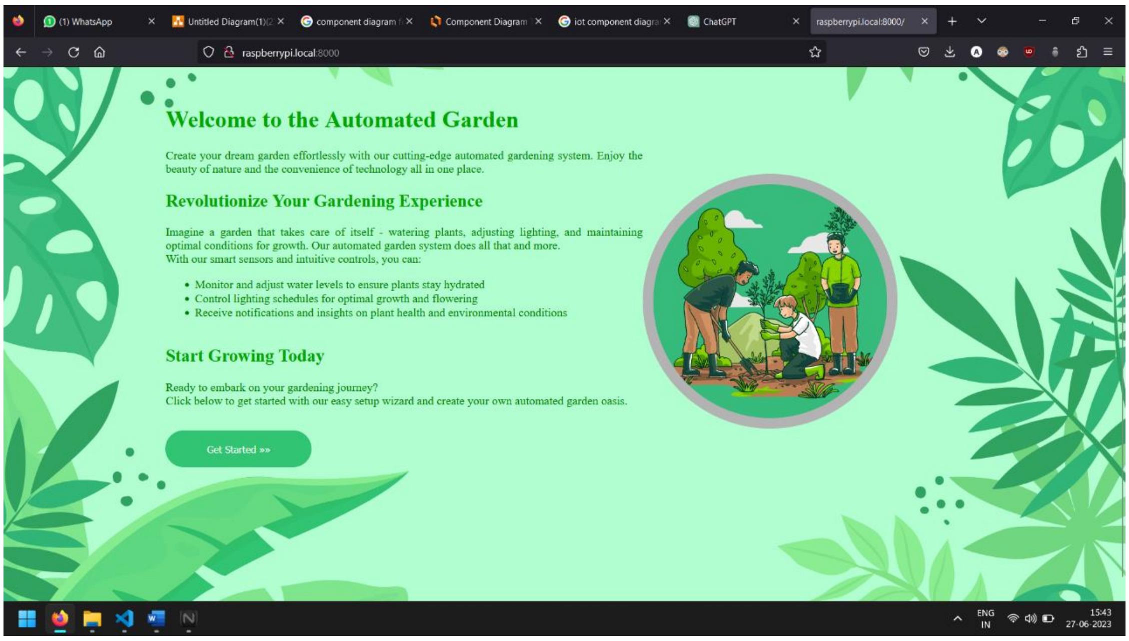
Pump State

Column	Data Type	Description
state	BooleanField	State of the pump (On/Off)

Light State

Column	Data Type	Description
state	BooleanField	State of the light (On/Off)

3.10 Sample Input and Output Screens:



Welcome to the Automated Garden

Create your dream garden effortlessly with our cutting-edge automated gardening system. Enjoy the beauty of nature and the convenience of technology all in one place.

Revolutionize Your Gardening Experience

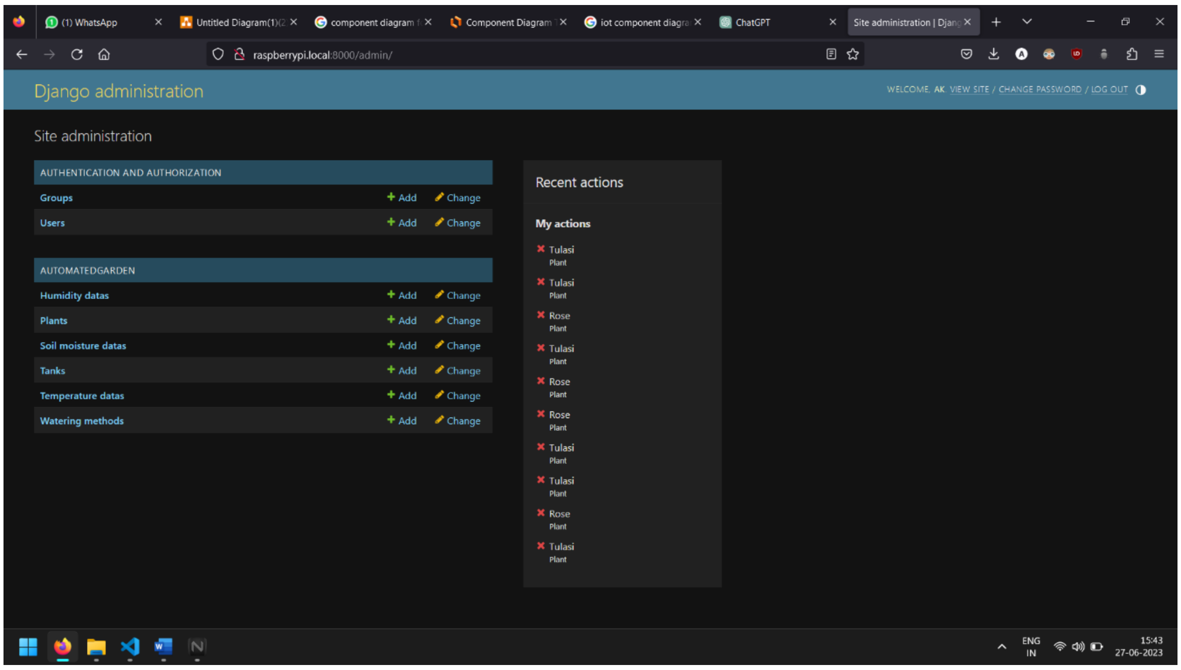
Imagine a garden that takes care of itself - watering plants, adjusting lighting, and maintaining optimal conditions for growth. Our automated garden system does all that and more. With our smart sensors and intuitive controls, you can:

- Monitor and adjust water levels to ensure plants stay hydrated
- Control lighting schedules for optimal growth and flowering
- Receive notifications and insights on plant health and environmental conditions

Start Growing Today

Ready to embark on your gardening journey?
Click below to get started with our easy setup wizard and create your own automated garden oasis.

Home Page


raspberrypi.local:8000/admin/
Django administration
WELCOME, AK. VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

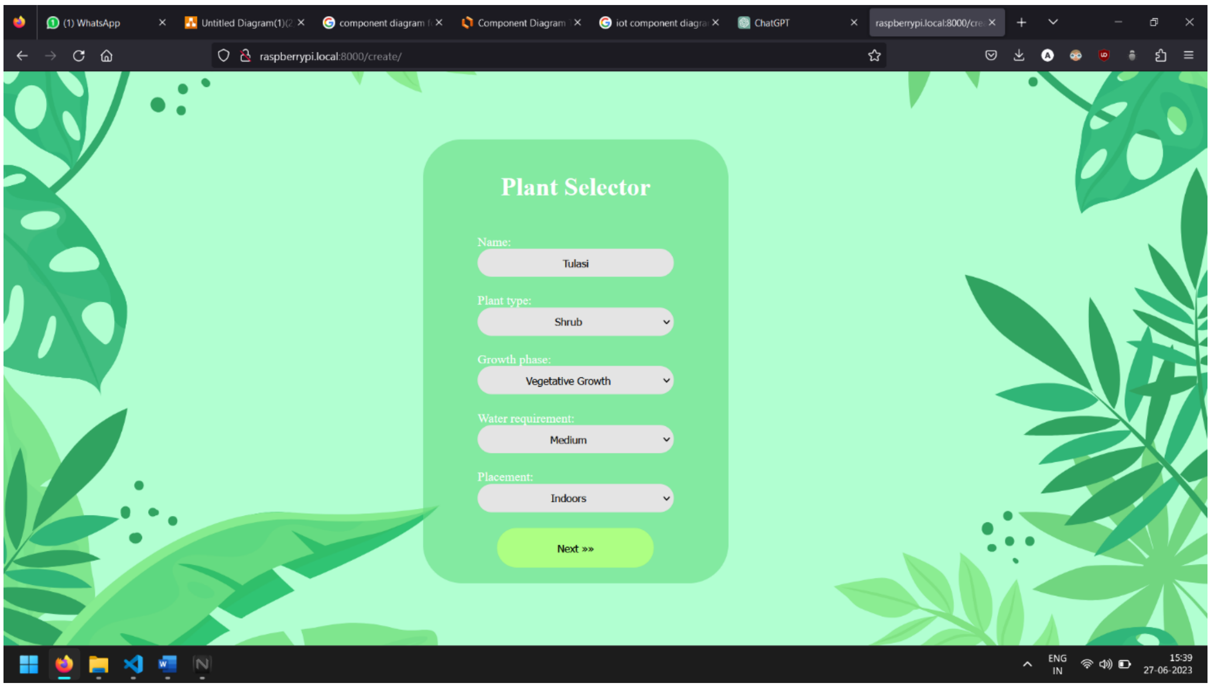
AUTOMATEDGARDEN		
Humidity datas	+ Add	Change
Plants	+ Add	Change
Soil moisture datas	+ Add	Change
Tanks	+ Add	Change
Temperature datas	+ Add	Change
Watering methods	+ Add	Change

Recent actions

My actions

- ✖ Tulasi Plant
- ✖ Tulasi Plant
- ✖ Rose Plant
- ✖ Tulasi Plant
- ✖ Rose Plant
- ✖ Rose Plant
- ✖ Tulasi Plant
- ✖ Tulasi Plant
- ✖ Rose Plant
- ✖ Tulasi Plant

Admin Page



Plant Selector

Name: Tulasi

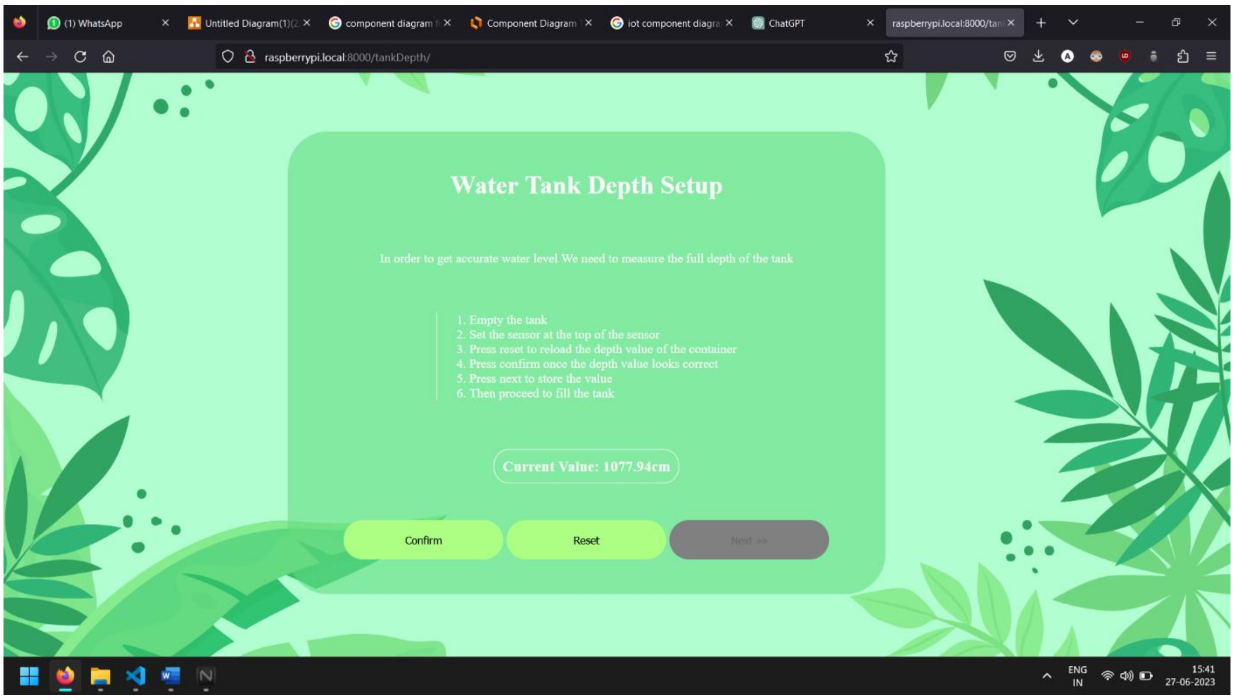
Plant type: Shrub

Growth phase: Vegetative Growth

Water requirement: Medium

Placement: Indoors

Plant Selection Page



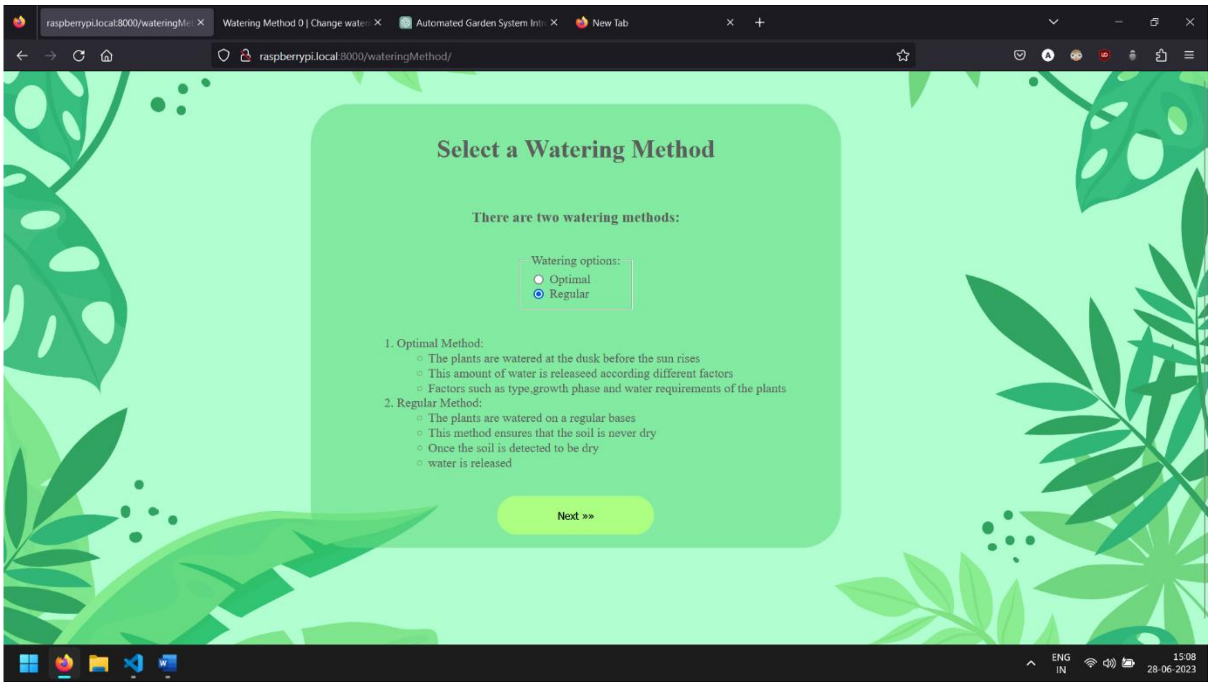
Water Tank Depth Setup

In order to get accurate water level We need to measure the full depth of the tank

1. Empty the tank
2. Set the sensor at the top of the sensor
3. Press reset to reload the depth value of the container
4. Press confirm once the depth value looks correct
5. Press next to store the value
6. Then proceed to fill the tank

Current Value: 1077.94cm

Water Tank Depth Setup



raspberrypi.local:8000/wateringMethod/

Select a Watering Method

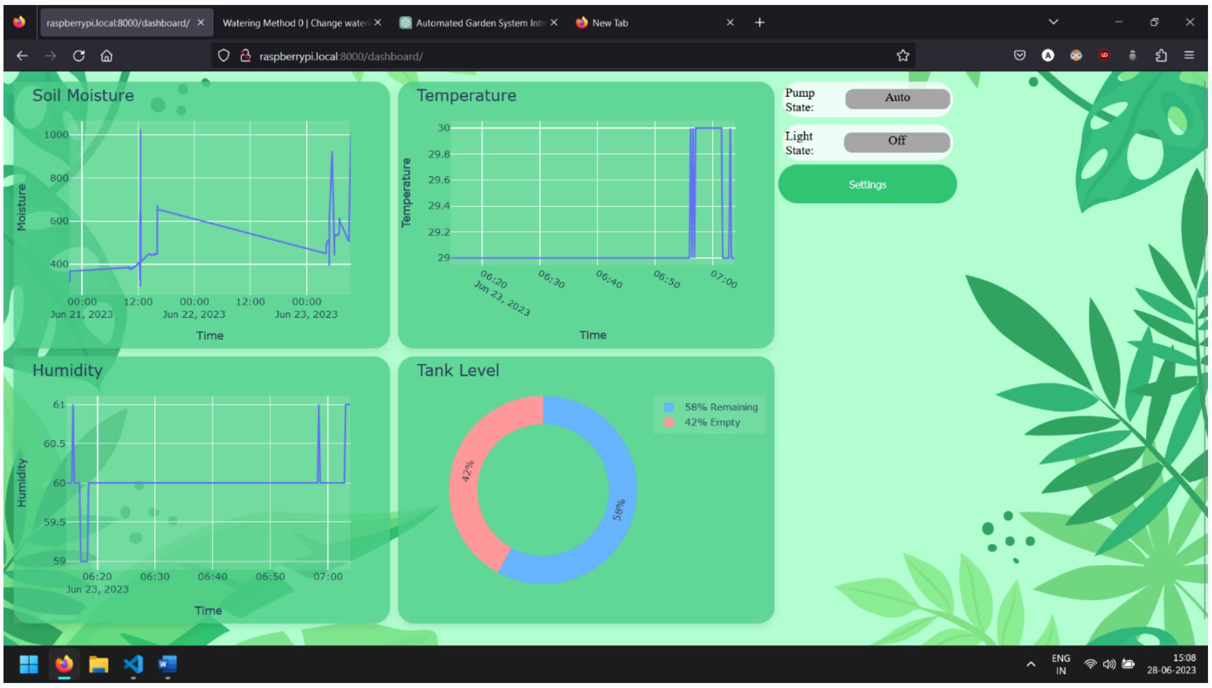
There are two watering methods:

Watering options:
Optimal [✓ Regular]

1. Optimal Method:
   - The plants are watered at the dusk before the sun rises
   - This amount of water is releaseed according different factors
   - Factors such as type,growth phase and water requirements of the plants
2. Regular Method:
   - The plants are watered on a regular bases
   - This method ensures that the soil is never dry
   - Once the soil is detected to be dry
   - water is released

Next »»

Watering Method Selection



raspberrypi.local:8000/dashboard/

Pump State: Auto

Light State: Off

User Dashboard

CHAPTER 4: CODING Sample code

BackEnd Server

```
from django.shortcuts import render,redirect
from .functions import read_depth
from .forms import PlantForm,WateringOptionsForm
from .models import 
Plant,Tank,WateringMethod,HumidityData,TemperatureData,LightState
from .models import SoilMoistureData
from django.http import HttpResponse
import plotly.graph_objects as go

import plotly.graph_objs as go
import plotly.io as pio


def create_plant(request):
    instance, created = Plant.objects.get_or_create(id=0)

    if request.method == 'POST':
        form = PlantForm(request.POST, instance=instance)
        if form.is_valid():
            form.save()
            # Redirect to the desired URL after saving the form
    else:
        form = PlantForm(instance=instance)

    return render(request, 'create_plant.html', {'form': form, 'created': 
created})

def tank_depth(request):
    value = 0
    confirmed=False
    tank, created = Tank.objects.get_or_create(id=0)

    if request.method == 'POST':
        if 'confirm' in request.POST:
            value = read_depth()
            confirmed=True
        elif 'reset' in request.POST:

            value = read_depth()
        elif 'save' in request.POST:
            value = read_depth()
            tank.value = value
            tank.save()
        else:
            value = None
    else:
        value = read_depth()

    return render(request, 'tank_depth.html', {'value':
value,'confirmed':confirmed})
def watering_method(request):
    wateringMethod, created = WateringMethod.objects.get_or_create(id=0)
    if request.method == 'POST':
        form = WateringOptionsForm(request.POST)
        if form.is_valid():
            selected_option = form.cleaned_data['watering_options']
            wateringMethod.method = selected_option
            wateringMethod.save()
    else:
        form = WateringOptionsForm()

    return render(request, 'watering_method.html', {'form': form})
def index(request):
    return render(request, 'index.html')
    try:
        obj = Plant.objects.get(id=0)
        return redirect('dashboard')
    except Plant.DoesNotExist:
        # If the ID does not exist, redirect to the create page
        return redirect('create')
def dashboard(request):
    try:
        # Query the database using the ID
        obj = Plant.objects.get(id=0)

        # If the ID exists, redirect to the dashboard page
        return redirect('create')

        return render(request, 'index.html')
    except Plant.DoesNotExist:
        # If the ID does not exist, redirect to the create page
        return render(request, 'index.html')



def soilgraph(request):

    data = SoilMoistureData.objects.only('value', 'time').order_by('-time')
    values = [entry.value for entry in data]
    timestamps = [entry.time for entry in data]

    fig = go.Figure(data=go.Scatter(x=timestamps, y=values, mode='lines'))

    # Customize the plot layout
    fig.update_layout(
        xaxis=dict(title='Time'),
        yaxis=dict(title='Moisture'),
        title='Soil Moisture',
        plot_bgcolor='rgba(255, 255, 255, 0.1)',
        paper_bgcolor='rgba(255, 255, 255, 0.1)',
        showlegend=False
    )

    # Convert the plot to an image and return it as the response
    img_bytes = fig.to_image(format='png')


    return HttpResponse(img_bytes, content_type='image/png')



def humiditygraph(request):
    # Retrieve the soil moisture data from the database
    data = HumidityData.objects.only('value', 'time').order_by('-time')

    # Extract the values and timestamps from the queryset
    values = [entry.value for entry in data]
    timestamps = [entry.time for entry in data]

    # Create a scatter plot
    fig = go.Figure(data=go.Scatter(x=timestamps, y=values, mode='lines'))

    # Customize the plot layout
    fig.update_layout(
        xaxis=dict(title='Time'),
        yaxis=dict(title='Humidity'),
        title='Humidity',
        plot_bgcolor='rgba(255, 255, 255, 0.1)',
        paper_bgcolor='rgba(255, 255, 255, 0.1)',
        showlegend=False
    )

    # Convert the plot to an image and return it as the response
    img_bytes = fig.to_image(format='png')

    # Return the image as the response
    return HttpResponse(img_bytes, content_type='image/png')
```

```python
def temperaturegraph(request):
    # Retrieve the soil moisture data from the database
    data = TemperatureData.objects.only('value', 'time').order_by('-time')

    # Extract the values and timestamps from the queryset
    values = [entry.value for entry in data]
    timestamps = [entry.time for entry in data]

    # Create a scatter plot
    fig = go.Figure(data=go.Scatter(x=timestamps, y=values, mode='lines'))

    # Customize the plot layout
    fig.update_layout(
        xaxis=dict(title='Time'),
        yaxis=dict(title='Temperature'),
        title='Temperature',
        plot_bgcolor='rgba(255, 255, 255, 0.1)',
        paper_bgcolor='rgba(255, 255, 255, 0.1)',
        showlegend=False
    )

    # Convert the plot to an image and return it as the response
    #img_bytes = fig.to_image(format='png')
    config = {'displayModeBar': False}
    temperature = pio.to_html(fig, include_plotlyjs=False,
full_html=False,config=config)

    # Pass the plot_div as context to the template for rendering

    return temperature
    # Return the image as the response
    #return HttpResponse(img_bytes, content_type='image/png')

def waterlevel(request):
    scale_percentages = [0, 20, 40, 60, 80, 100]
    tankDepth = Tank.objects.first().value
    percentage = int(read_depth() / tankDepth * 100)

    if percentage >= 101:
        percentage = 100
    remaining = 100 - percentage
    slices = [percentage, remaining]
    labels = [f'{percentage}% Empty', f'{remaining}% Remaining']
    colors = ['#ff9999', '#66b3ff']

    # Create the pie chart
    fig = go.Figure(data=[go.Pie(labels=labels, values=slices, hole=0.7,
marker=dict(colors=colors))])
```

```
    # Set aspect ratio to equal to ensure a circular pie chart
    fig.update_layout(
        title='Tank Level',
        showlegend=True,
        plot_bgcolor='rgba(255, 255, 255, 0.1)',
        paper_bgcolor='rgba(255, 255, 255, 0.1)',
        # height=400,
        # width=600
    )

    # Convert the plot to an image and return it as the response
    img_bytes = fig.to_image(format='png')

    # Return the image as the response
    return HttpResponse(img_bytes, content_type='image/png')



def waterlevel2(request):

    scale_percentages = [0, 20, 40, 60, 80, 100]
    tankDepth = Tank.objects.first().value
    percentage = int(read_depth() / tankDepth * 100)

    if percentage >= 101:
        percentage = 100
    remaining = 100 - percentage
    slices = [percentage, remaining]
    labels = [f'{percentage}% Empty', f'{remaining}% Remaining']
    colors = ['#ff9999', '#66b3ff']

    # Create the pie chart
    fig = go.Figure(data=[go.Pie(labels=labels, values=slices, hole=0.7,
marker=dict(colors=colors))])

    # Set aspect ratio to equal to ensure a circular pie chart
    fig.update_layout(
        title='Tank Level',
        showlegend=True,
        plot_bgcolor='rgba(255, 255, 255, 0.1)',
        paper_bgcolor='rgba(255, 255, 255, 0.1)',
        # height=400,
        # width=600
    )

    # Convert the plot to a div string
    config = {'displayModeBar': False}
    plot_div = pio.to_html(fig, include_plotlyjs=False,
full_html=False,config=config)

    # Pass the plot_div as context to the template for rendering
    # context = {'waterlevel': plot_div}
    return plot_div
    #return render(request, 'dashboard.html', context)




def dashboard(request):
    context={'waterlevel':waterlevel2(request),
             "temperature":temperaturegraph(request)}
    return render(request, 'dashboard.html',context)

#pump_state, created = PumpState.objects.get_or_create(pk=1)

# Change the values of the state attribute
  # Set the state to True (On)
# pump_state.state = False  # Set the state to False (Off)

# # Save the changes

# pump_state.save()
def switch_lights_view(request):
    if request.method == 'POST':
        light_switch = request.POST.get('light_switch', False)
        #light_state, created = LightState.objects.get_or_create(pk=1)
        light_state = LightState.objects.first()

        # Change the value of the state attribute
        light_state.state = light_switch  # Set the state to True (On)
        light_state.save()  # Save the changes


    return redirect('dashboard')

def switch_pump_view(request):
    if request.method == 'POST':
        pump_switch = request.POST.get('pump_switch', False)
        # Perform actions based on the pump_switch state, e.g., control pump
operation
        # Your logic here
    return redirect('dashboard')
```

CHAPTER 5: LIMITATIONS OF SYSTEM

Some limitations of the Automated Garden project are as follows:

- The system relies on the accuracy of the soil moisture sensor and temperature sensor readings.
- The system may not account for other factors affecting plant growth, such as sunlight and nutrient levels.

CHAPTER 6: PROPOSED ENHANCEMENTS

The Automated Garden project has potential for further enhancements and expansions, including:

- Integration with weather forecast data to adjust watering schedules based on upcoming weather conditions.
- Adding additional sensors to monitor sunlight levels, humidity, and nutrient levels in the soil.
- Incorporating machine learning techniques to optimize

CHAPTER 7: CONCLUSION

In conclusion, the Automated garden is a valuable tool for gardeners and farmers to ensure that their plants receive timely water. With the aid of this project, gardeners can keep track of their plant’s watering schedules, get timely notifications, access stats. It is a beneficial resource for gardeners to maintain the health of their dear plants, Since this project automates the water of plant the gardeners can be worry free that their plants might get dry because of dehydration.

CHAPTER 8: BIBLIOGRAPHY

1. Raspberry Pi Documentation Raspberry Pi. "Raspberry Pi Documentation." Accessed June 10, 2023. https://www.raspberrypi.com/documentation/.
2. Arduino Documentation Arduino. "Arduino Documentation." Accessed June 20, 2023. https://docs.arduino.cc/.
3. CircuitPython Neopixel Library Documentation CircuitPython. "Neopixel Library Documentation." Accessed June 5, 2023. https://docs.circuitpython.org/projects/neopixel/en/latest/index.html.
4. Django Documentation Django Software Foundation. "Django Documentation." Accessed June 25, 2023. https://docs.djangoproject.com/en/.
5. Python Documentation Python Software Foundation. "Python Documentation." Accessed June 15, 2023. https://docs.python.org.
6. NodeMCU Documentation NodeMCU Team. "NodeMCU Documentation." Accessed June 8, 2023. https://nodemcu.readthedocs.io/en/release/.