CSE 447
SOFTWARE ENGINEERING

Agri Assist

Solution for Agricultural Automation

Course Instructor: Associate Prof. Dr. Mert Özkaya

Group Members:
Mertkan İşcan
Yalın Gürsoy

Yeditepe University
Computer Engineering Department
Istanbul, 2023

1. Background (Introduction of the Problem Domain)

Navigating the Future of Automation:
In the dynamic landscape of modern technology, the role of automation systems is becoming increasingly pivotal. Across diverse sectors - from the precision-dependent realms of agriculture and industrial manufacturing to the convenience-centric world of smart home management - the demand for sophisticated, integrated automation solutions is at an all-time high. This era is marked by a transition from manual, labor-intensive processes to intelligent, data-driven systems.

Unveiling a New Era of Integrated Automation:
At the heart of this transformative wave is our project, a groundbreaking initiative that intertwines advanced sensors, a highly capable Automation Engine (AE), and agile actuators into a seamless, interconnected network. This integration goes beyond mere functionality; it represents a harmonious fusion of precision, efficiency, and adaptability. Our system is not just an array of devices; it's a finely tuned orchestra of technological symphony, orchestrated to anticipate and respond to the ever-evolving demands of the modern world.

Problem Domain:
The domain involves a network of sensors that collect data about their environment (e.g., temperature, humidity), an Automation Engine (AE) that processes this data and makes decisions, and actuators that execute actions based on commands from the AE. This system is intended to automate tasks, improve efficiency, and respond rapidly to changing conditions.

2. Problem Statement

Redefining the Automation Landscape:
In the traditional automation landscape, challenges such as fragmented system architectures, latency in data processing and reaction, and limited user interfaces are prevalent. These challenges often lead to inefficiencies, reduced productivity, and a disconnect between user intent and system performance. Our vision was to transform these challenges into a canvas of opportunity, painting a new picture of what automation can achieve.

Our Pioneering Solution:
We introduce a system that transcends the conventional boundaries of automation:

Intuitive and Seamless Integration:
Where others see a jigsaw of complexities, we see a puzzle waiting to be solved. Our system ensures a fluid conversation between sensors and actuators, mediated by the AE. This level of integration ensures that every component works in unison, creating a tapestry of operational harmony.

Real-Time Data Processing and Proactive Adaptation:
We turn mere data into insightful, actionable intelligence. Our AE is engineered to not just analyze but to foresee and adapt, making real-time decisions that optimize outcomes instantaneously. This proactive approach ensures that the system is always a step ahead, ready to respond to any change with precision and agility.

Empowering Users with Unprecedented Control:
We believe that true power lies in control. Our user-centric Desktop App is more than a window into the system; it's a command center that empowers users with unparalleled control and customization. Users can monitor, manage, and manipulate the system, tailoring it to their precise needs and preferences.

Crafting the Future of Automated Intelligence:
Our system is a beacon of innovation in the automation arena. By leveraging the latest in technology, we offer not just a solution, but a transformation - a system that embodies reliability, scalability, and intelligence. We're not just constructing an automation system; we're shaping the future where efficiency, accuracy, and user empowerment converge to create a new standard in automated intelligence.

Current Challenges:
Integration of diverse sensors and actuators with the AE.
Real-time processing and analysis of sensor data.
Reliable communication between sensors, the AE, and actuators.
User interaction and control over the automated system.

Project Aim:
Develop a robust system where sensors can send data to the AE, which then analyzes this data and sends appropriate commands to actuators. The system should also allow user intervention for direct control of actuators via a Desktop App.

3. Requirements Gathering and Analysis

3.1.1. Functional Requirements

1. Sensor Data Collection
Requirement: Sensors must continuously collect environmental data.
Implementation: Soil and weather sensors will collect specific data such as soil humidity, nitrogen, phosphorus, potassium levels for soil sensors, and humidity, temperature, wind speed, wind direction, and UV index for weather sensors.

2. Data Transmission to AE
Requirement: Sensors should transmit data to the Automation Engine (AE) at regular intervals.
Implementation: Both soil and weather sensors will send data to the AE using TCP connections. Data will be formatted in JSON for consistency and ease of processing.

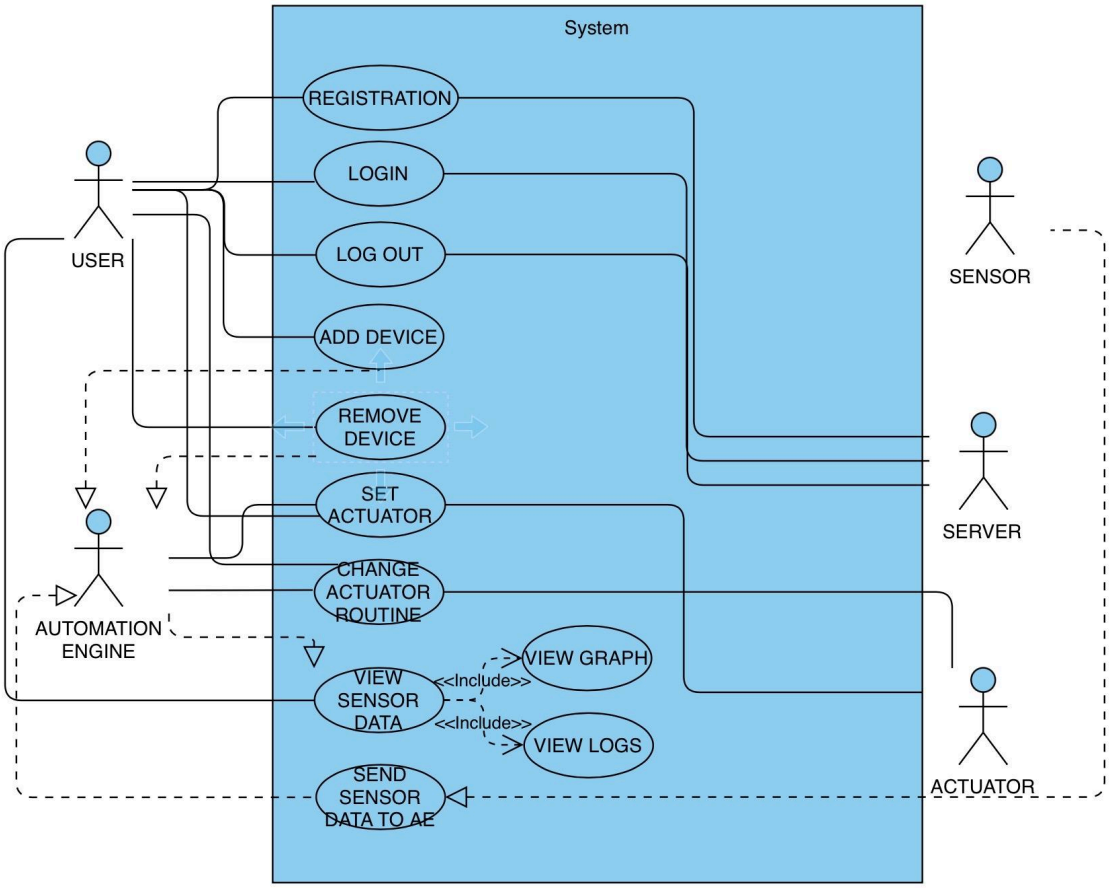
3. Real-Time Data Processing by AE
Requirement: The AE must analyze sensor data in real time.
Implementation: The AE will process incoming data immediately upon receipt, using predefined logic to evaluate and respond to environmental conditions.

4. Actuator Control
Requirement: The AE must send control commands to actuators based on the sensor data.
Implementation: Based on the analysis of combined soil and weather sensor data, the AE will send appropriate commands to actuators to perform actions like irrigation control or environmental adjustments.

5. User Interface Interaction
Requirement: Users should be able to monitor real-time data and control the system through a Desktop App.
Implementation: The Desktop App will provide a real-time overview of sensor data and system status. It will allow users to manually control actuators and customize settings.

7. Data Storage and Caching
Requirement: The AE should temporarily store sensor data for historical analysis and backup.
Implementation: The AE will store received sensor data in a temporary storage system while processing it in real time. This data can be used for historical analysis or as a backup.

3.1.2. Use Case Diagram



3.1.3. Use Case Specifications

USE CASE ID: UC1
USE CASE:REGISTRATION
ACTORS: USER, SERVER
RELATED USE CASES:
PRECONDITION:
MAIN FLOW: 1.User opens registration page. 2.User chooses and enters username and password. 3.User completes registration.
POST CONDITION: User is registered to the system.

USE CASE ID: UC2
USE CASE: LOGIN
ACTORS: USER, SERVER
RELATED USE CASES:
PRECONDITION: User already registered.
MAIN FLOW: 1.User opens the application. 2.User enters username and password. 3.User completes logging in.
POST CONDITION: User has logged in.

USE CASE ID: UC3
USE CASE: LOG OUT
ACTORS: USER,SERVER
RELATED USE CASES:
PRECONDITION: User has already logged in.

MAIN FLOW: 1.User clicks the log out button. 2.Confirms to log out.
POST CONDITION: User has logged out.

USE CASE ID: UC4
USE CASE: ADD DEVICE
ACTORS: USER, AUTOMATION ENGINE
RELATED USE CASES:
PRECONDITION: User has registered and logged in.
MAIN FLOW: 1.User clicks on “Add new device”. 2.User enters device ID. 3.User completes adding and returns to the main page.
POST CONDITION: Device is added.

USE CASE ID: UC5
USE CASE: REMOVE DEVICE
ACTORS: USER, AUTOMATION ENGINE
RELATED USE CASES:
PRECONDITION: User has added device(s).
MAIN FLOW: 1.User clicks on “Remove device”. 2.User either browses the devices and clicks the “Remove Device” button or Enters the ID of the device. 3.User completes removal and returns to the main page.
POST CONDITION: Device is removed.

USE CASE ID: UC6
USE CASE: SET ACTUATOR

ACTORS: USER/AUTOMATION ENGINE, ACTUATOR
RELATED USE CASES:
PRECONDITION: User already added actuator(s).
MAIN FLOW: 1.User clicks on “Start Actuator”/”Stop Actuator”. 2.User chooses an actuator. 3.User completes setting and returns to the main page. OR 1.AE sends a signal to start/stop the actuator.
POST CONDITION: ACTUATOR IS SET.

USE CASE ID: UC7
USE CASE: CHANGE ACTUATOR ROUTINE
ACTORS: USER, ACTUATOR, AUTOMATION ENGINE
RELATED USE CASES:
PRECONDITION: User has added actuator(s).
MAIN FLOW: 1. User clicks on “Change Actuator Routine”. 2.User chooses an actuator. 3.User views the actuator's current routine. 4.User can manipulate the current routine. 5.User completes changes and returns to the main page.
POST CONDITION: ACTUATOR ROUTINE IS CHANGED.

USE CASE ID: UC8
USE CASE: VIEW SENSOR DATA
ACTORS: USER, AUTOMATION ENGINE
RELATED USE CASES: Includes: VIEW GRAPH, VIEW LOGS
PRECONDITION: User has added sensor(s).
MAIN FLOW:

1.User clicks on “View Sensor Data”.
2.User chooses a sensor.
3.User can choose to see a graph of data or logs.
4.User views the data or logs.
5.User completes viewing and returns to the main page.

POST CONDITION: NO CHANGES.

USE CASE ID: UC9

USE CASE: VIEW GRAPH

ACTORS: USER, AUTOMATION ENGINE

RELATED USE CASES:

PRECONDITION: User chose to view the data graph of a sensor.

MAIN FLOW:

1.User chooses to view the data graph of a sensor.
2.A graph of data is generated by AE.
3.User completes viewing and returns to the main page.

POST CONDITION: NO CHANGES

USE CASE ID: UC10

USE CASE: VIEW LOGS

ACTORS: USER, AUTOMATION ENGINE

RELATED USE CASES:

PRECONDITION: User chose to view the logs of a sensor.

MAIN FLOW:

1.User chooses to view the logs of a sensor.
2.Logs are provided by AE.
3.User completes viewing and returns to the main page.

POST CONDITION: NO CHANGES

USE CASE ID: UC11

USE CASE: SEND SENSOR DATA TO AE

ACTORS: SENSOR, AE
RELATED USE CASES:
PRECONDITION: User has added sensor(s).
MAIN FLOW: 1.Sensors keep collecting data for a specified time. 2.Sensors send data they collected to AE. 3.Sensors keep repeating the first 2 steps.
POST CONDITION: Data is sent to AE.

3.2.1 Non-Functional Requirements

Reliability and Availability
Implementation: The system's software, particularly the AE, will be designed for continuous operation with robust error handling and recovery mechanisms. Regular updates and patches will be scheduled during low-usage periods to minimize downtime.
Fail-Safes: Sensors and actuators will have built-in fail-safes. For example, in case of communication loss, actuators will default to a safe state to prevent any hazardous operations.

Scalability
Implementation: The system architecture, especially the AE, will support modular scalability. The use of standardized communication protocols (TCP/IP) and data formats (JSON) allows for the addition of new sensor and actuator types without significant reconfiguration.

Usability
Implementation: The Desktop App will feature an intuitive user interface, designed for ease of use. The interface will provide clear and straightforward controls for system monitoring and management.
User Training: Minimal training will be required. User-friendly documentation and tooltips within the app will guide the users.

Performance
Implementation: Real-time processing will be achieved through efficient software algorithms in the AE. The system will be optimized for low-latency responses, ensuring timely analysis of sensor data and command execution.
Data Handling: The system will be capable of handling high volumes of sensor data without performance degradation, utilizing efficient data processing and storage techniques.

Security
Implementation: Secure communication channels will be established between sensors, AE, and actuators. This may include the use of SSL/TLS for data encryption.
Data Protection: Sensitive data will be encrypted both in transit and at rest. The system will implement strong authentication and authorization mechanisms to prevent unauthorized access.

Maintainability and Support
Implementation: The system will be designed for easy maintenance, with modular components that can be independently updated or replaced.
Documentation: Comprehensive documentation will be provided, covering system setup, troubleshooting guides, and maintenance procedures.

3.2.2 Volere Templates

REQUIREMENT: RELIABILITY AND AVAILABILITY	**REQUIREMENT ID:**1
DESCRIPTION: The system's software, particularly the AE, will be designed for continuous operation with robust error handling and recovery mechanisms. Regular updates and patches will be scheduled during low-usage periods to minimize downtime.	
RATIONALE: To ensure uninterrupted operation of the system and minimize disruptions to users.	
FIT CRITERION:The system should have a downtime of less than 1% of the total operational hours in a month due to software updates and patches.	

REQUIREMENT: FAIL-SAFE	**REQUIREMENT ID:**2
DESCRIPTION: Sensors and actuators will have built-in fail-safes. For example, in case of communication loss, actuators will default to a safe state to prevent any hazardous operations.	
RATIONALE: To mitigate risks and ensure the safety of the system in the event of failures or communication issues.	
FIT CRITERION: Actuators must default to a safe state within 2 seconds of communication loss with sensors.	

REQUIREMENT: SCALABILITY	**REQUIREMENT ID:**3
DESCRIPTION: The system architecture, especially the AE, will support modular scalability. The use of standardized communication protocols (TCP/IP) and data formats (JSON) allows for the addition of new sensor and actuator types without significant reconfiguration.	
RATIONALE: To accommodate future growth and the addition of new sensor and actuator types with ease.	
FIT CRITERION: The system should support the integration of at least five new sensor and actuator types without requiring a significant change in the system architecture or causing downtime.	

REQUIREMENT: USABILITY	**REQUIREMENT ID:**4
DESCRIPTION: The Desktop App will feature an intuitive user interface, designed for ease of use. The interface will provide clear and straightforward controls for system monitoring and management.	
RATIONALE: To ensure that users can interact with the system efficiently and without confusion.	
FIT CRITERION:User testing results should indicate that 90% of users find the Desktop App interface easy to use.	

REQUIREMENT: USER TRAINING	**REQUIREMENT ID:**5
DESCRIPTION:Minimal training will be required. User-friendly documentation and tooltips within the app will guide the users.	
RATIONALE: To minimize the learning curve for users and provide adequate support.	
FIT CRITERION: Users should be able to perform basic system tasks without external assistance after reading the provided documentation.	

REQUIREMENT: PERFORMANCE	**REQUIREMENT ID:**6
DESCRIPTION: Real-time processing will be achieved through efficient software algorithms in the AE. The system will be optimized for low-latency responses, ensuring timely command execution.	
RATIONALE: To provide timely and efficient responses to sensor data and user commands.	
FIT CRITERION: The system should process sensor data and execute commands with a latency of less than 100 milliseconds.	

REQUIREMENT: DATA HANDLING	**REQUIREMENT ID:**7
DESCRIPTION: The system will be capable of handling high volumes of sensor data without performance degradation, utilizing efficient data processing and storage techniques.	
RATIONALE: To ensure that the system can effectively manage and process large volumes of data without impacting performance.	
FIT CRITERION: The system should be able to handle a sustained data input rate of at least 1,000 data points per second without performance degradation.	

REQUIREMENT: SECURITY	**REQUIREMENT ID:**8
DESCRIPTION: Secure communication channels will be established between sensors, AE, and actuators.	
RATIONALE: To protect sensitive data and prevent unauthorized access to the system.	
FIT CRITERION: All communication between sensors, AE, and actuators should be encrypted using and no unauthorized access should be permitted.	

REQUIREMENT: DATA PROTECTION	**REQUIREMENT ID:**9
DESCRIPTION: Sensitive data will be encrypted both in transit and at rest. The system will implement strong authentication and authorization mechanisms to prevent unauthorized access.	
RATIONALE: To safeguard sensitive data from unauthorized access and protect data integrity.	
FIT CRITERION: All sensitive data should be encrypted during transmission and storage, and unauthorized access attempts should be logged and blocked.	

REQUIREMENT: MAINTAINABILITY AND SUPPORT	**REQUIREMENT ID:**10
DESCRIPTION: The system will be designed for easy maintenance, with modular components that can be independently updated or replaced.	
RATIONALE: To facilitate system maintenance and minimize disruptions during updates and component replacements.	
FIT CRITERION: Individual system components should be replaceable or upgradable without affecting the overall system's functionality or causing extensive downtime.	

REQUIREMENT: DOCUMENTATION	**REQUIREMENT ID:**11
DESCRIPTION: Comprehensive documentation will be provided, covering system setup, troubleshooting guides, and maintenance procedures.	
RATIONALE: To assist system administrators and users in setting up, troubleshooting, and maintaining the system.	
FIT CRITERION: The documentation should cover all essential aspects of system setup, troubleshooting, and maintenance, and user feedback should indicate its usefulness and comprehensiveness.	

4. Architecture Specifications

Sensor to Automation Engine Interaction:
Description: Sensors act as data providers, continuously collecting environmental information such as temperature, humidity, etc. This data is transmitted to the Automation Engine (AE) at regular intervals or immediately in case of threshold breaches or emergency conditions.
Data Format: Sensor data is encapsulated in JSON format for universal compatibility and ease of parsing.

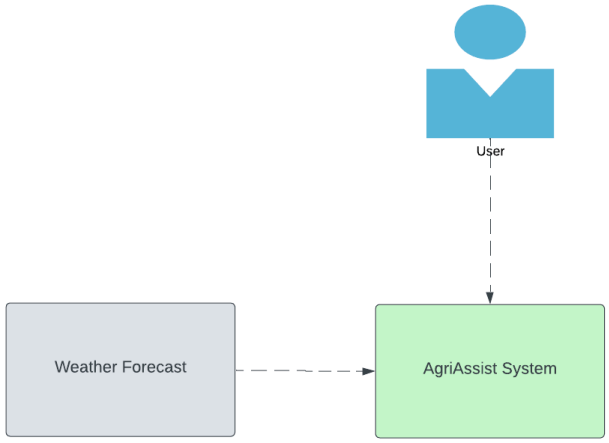
Automation Engine to Actuator Interaction:
Description: The AE processes sensor data and, based on predefined logic or real-time analysis, sends commands to actuators to perform specific actions.
Technology: TCP/IP protocol for robust and persistent communication.
Command Structure: Commands are sent in JSON format, containing fields like command (e.g., start, stop, updateSchedule) and parameters for detailed instruction.

System Sensor Devices:
Our system's sensor devices can be added in any desired quantity and type. There are two main types of sensors: Soil and Weather. Apart from the data they transmit, the code structures of these sensors are completely identical.

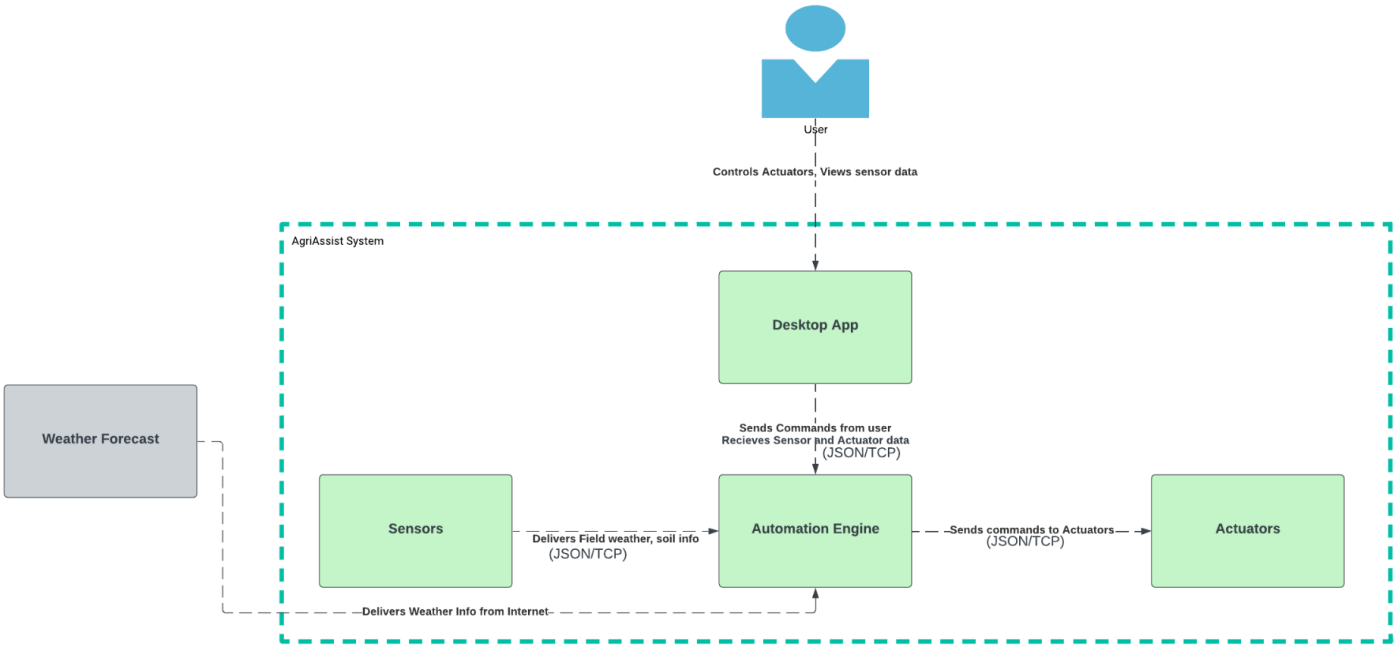
SensorTcpSender:
This component transmits the collected data to the Automation Engine (AE). It can use either wireless or wired network connections.

4.0.1) Early c4 Design

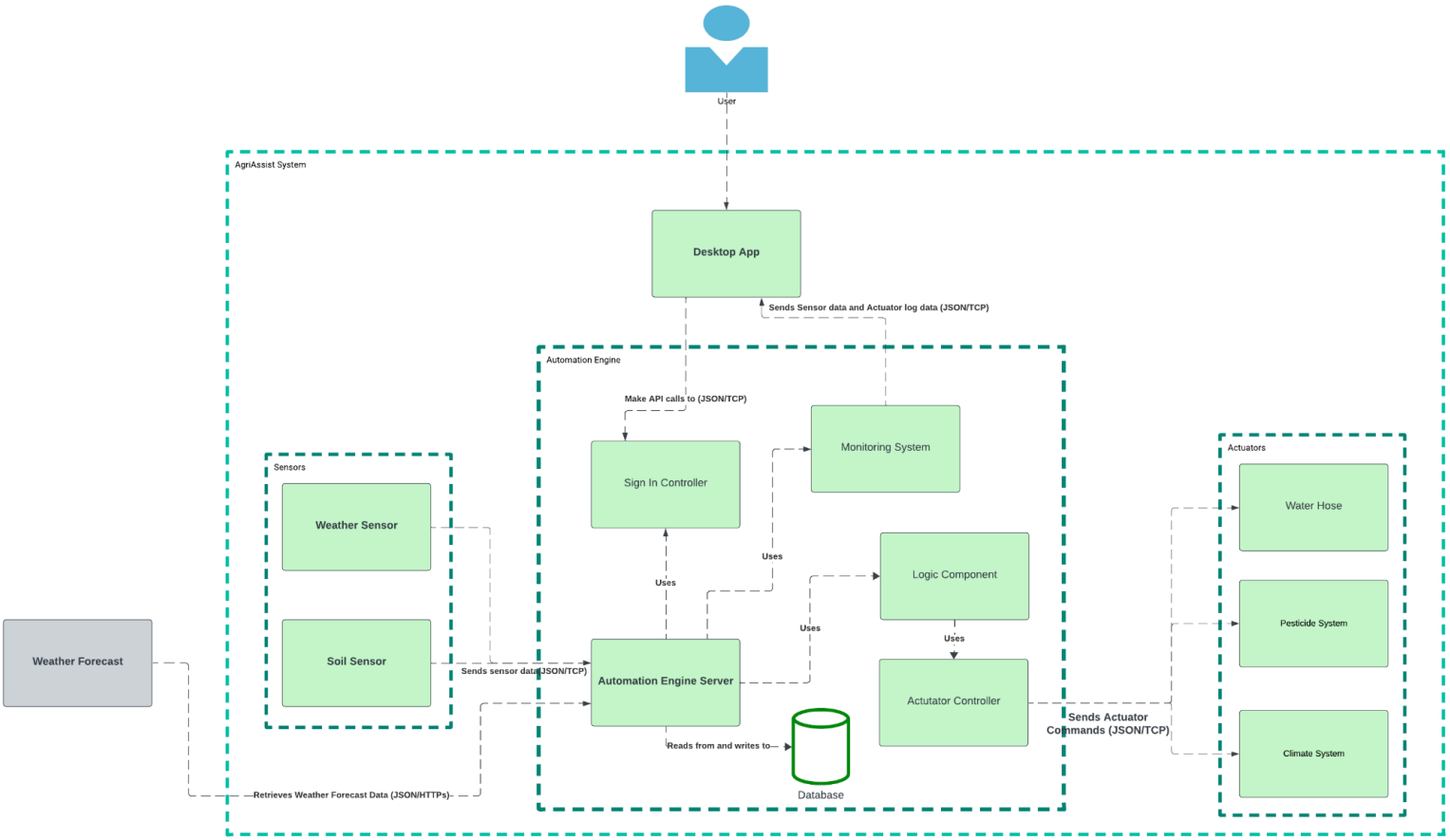
4.0.1.1) System Context diagram



4.0.1.2) Container diagram



4.0.1.3) Component diagram



4.1. Sensor Types:

4.1.1. Soil Sensor:

The Soil Sensor sends data to the AE at predetermined intervals via a TCP connection.
It transmits identity data related to the sensor, including sensorType, sensorId, sensorLocation, and timestamp. It measures and sends values related to soil components such as soil humidity, nitrogen, phosphorus, and potassium.
The data is sent to the AE in JSON format.

Example SoilSensor JSON Data:


```
{
  "sensorType": "Soil Sensor",
  "sensorId": "SoilSensor123",
  "sensorLocation": "field1",
  "timestamp": "2023-12-15T10:00:00Z",
  "soilHumidityValue": 32.5,
  "nitrogenValue": 10.00,
  "phosphorusValue": 5.00,
  "potassiumValue": 15.00
}
```


4.1.2. Weather Sensor:

The Weather Sensor also sends data to the AE at set intervals via a TCP connection.
It transmits identity data such as sensorType, sensorId, sensorLocation, and timestamp.
It measures and sends weather-related values including humidity, temperature, windSpeed, windDirection, and uvIndex. The data is formatted in JSON and sent to the AE.

Example WeatherSensor JSON Data:


```
{
  "sensorType": "Weather Sensor",
  "sensorId": "WeatherSensor456",
  "sensorLocation": "field1",
  "timestamp": "2023-12-15T14:00:00Z",
  "humidityValue": 60.0,
  "temperatureValue": 24.0,
  "windSpeedValue": 15.0,
  "windDirection": "NE",
  "uvIndex": 5
}
```

4.2. Actuator Devices

Water Hose Actuator
Role: Controls the application of water in fields or gardens.
Commands: Start/Stop water flow, Adjust flow rate, Set watering duration.

Example Command:

```
{
  "actuatorType": "Water Hose",
  "actuatorId": "WaterHose001",
  "command": "Start",
  "parameters": {
    "flowRate": "Medium",
    "duration": "20 minutes"
  }
}
```

Climate Control Actuator
Role: Manages temperature and humidity in controlled environments like greenhouses.
Commands: Set temperature, Adjust humidity, Activate/Deactivate climate control.

Example Command:

```
{
  "actuatorType": "Climate Control",
  "actuatorId": "ClimateControl002",
  "command": "Set",
  "parameters": {
    "temperature": "22°C",
    "humidity": "60%"
  }
}
```

Pesticide Sprayer Actuator
Role: Automates the spraying of pesticides in agricultural fields.
Commands: Start/Stop spraying, Adjust spray intensity, Schedule spraying.
Example Command:


```
{
  "actuatorType": "Pesticide Sprayer",
  "actuatorId": "PesticideSprayer003",
  "command": "Start",
  "parameters": {
    "intensity": "High",
    "duration": "15 minutes"
  }
}
```


4.3. Interaction with Automation Engine:
Actuators receive commands from the AE based on sensor data analysis.
The TCP/IP protocol is used for communication, ensuring reliability.
Command Structure:
Commands are structured in JSON, containing actuator identification, command type, and relevant parameters.
Consistent with the system's data format for ease of integration and parsing.
Communication and Data Transmission:
Actuators acknowledge command receipt and execution.
In advanced systems, they can send back status updates to the AE for a feedback loop, improving system responsiveness and efficiency.

Example Actuator Status JSON:


```
{
  "actuatorType": "Water Hose",
  "actuatorId": "WaterHose001",
  "status": "Running",
  "lastCommandExecuted": {
    "command": "Start",
    "parameters": {
      "flowRate": "Medium",
      "duration": "20 minutes"
    }
  },
  "timestamp": "2023-12-16T10:00:00Z"
```

```
}
```

4.4. Overview:

Our system is architected as a symphony of interconnected devices and software, all harmonizing to create an efficient, intelligent automation environment. At its core, sensors gather crucial environmental data, acting as the ears and eyes of the system. This data is the lifeblood that flows through our system's veins, reaching the AE, which acts as the brain, analyzing, deciding, and directing.

4.4.1. The Brain - Automation Engine:

The AE stands as the central decision-maker. It's not just about data collection; it's about making sense of the data. The AE processes information, applying complex algorithms and machine learning models to make informed decisions. It then communicates these decisions to the actuators, effectively translating data into action.

4.4.2. The Limbs - Actuators:

Actuators receive their directives from the AE and spring into action. They are the hands and feet of our system, interacting with the physical world, be it adjusting a thermostat, controlling a water valve, or managing the lights.

4.4.3. The Human Touch - User Interface:

The Desktop App is the face of our system, where users can see, understand, and interact with their environment. It's designed not just for functionality but for user engagement and satisfaction. It bridges the gap between human intention and automated execution.

4.5. XCD formal software

4.5.1. Sensor Components

```
component SoilSensor(byte sensorId){
   byte soilHumidity, nitrogen, phosphorus, potassium;

   provided port dataOut{
      @functional{
         requires: true;
         ensures: \result := ConvertDataToJson(soilHumidity, nitrogen, phosphorus, potassium);
      }
      string sendData();
   }
}

component WeatherSensor(byte sensorId){
   byte humidity, temperature, windSpeed, windDirection, uvIndex;

   provided port dataOut{
      @functional{
         requires: true;
         ensures: \result := ConvertDataToJson(humidity, temperature, windSpeed,
windDirection, uvIndex);
      }
      string sendData();
```

4.5.2. Actuator Components

```
component WaterHoseActuator(byte actuatorId){
    string command;
    byte flowRate;
    string duration;
    byte status;

    required port commandIn{
        @functional{
            promises: \nothing;
            requires: command == "Start" || command == "Stop";
            ensures: command := incomingCommand;
                    adjustFlowRateAndDuration(incomingParameters);
        }
        void receiveCommand(string incomingCommand, string incomingParameters);
    }

    emitter port statusOut{
        @functional{
            requires: true;
            ensures: \result := ConvertToJSON("Water Hose", actuatorId, status, flowRate, duration);
        }
        string emitStatus();
    }

    private void adjustFlowRateAndDuration(string parameters){
        // Implementation for adjusting flow rate and duration based on incoming parameters
    }
}
```

4.5.3. Automation Engine (AE)

```
component AutomationEngine(){
    string sensorDataBuffer;
    string actuatorCommandBuffer;

    required port sensorDataIn{
        @functional{
            promises: \nothing;
            requires: true;
            ensures: sensorDataBuffer := incomingData;
        }
        void receiveSensorData(string incomingData);
    }

    emitter port actuatorCommandOut{
        @functional{
            requires: true;
            ensures: \result := GenerateActuatorCommand(sensorDataBuffer);
        }
        string emitActuatorCommand();
    }

    private string GenerateActuatorCommand(string sensorData){
        // Implementation for generating specific actuator commands based on sensor data
    }
}
```

4.5.4. User Interface Component

```
component UserInterface(){
   byte uiCommand;
   byte sensorDataDisplay;
   byte actuatorStatusDisplay;

   required port uiCommandIn{
      @functional{
         promises: \nothing;
         requires: true;
         ensures: uiCommand := incomingCommand;
      }
      void receiveCommand(byte incomingCommand);
   }

   provided port dataDisplayOut{
      @functional{
         requires: true;
         ensures: sensorDataDisplay := sensorData;
                  actuatorStatusDisplay := actuatorStatus;
      }
      void displayData(string sensorData, string actuatorStatus);
   }
}
```

4.5.5. Connectors

```
// Connectors for Sensor Data to AE
connector SensorDataToAE(SoilSensor{dataOut}, WeatherSensor{dataOut},
AutomationEngine{sensorDataIn}){
   // define connector processing and data flow
}

// Connectors for Actuator Command from AE
connector AEToActuator(AutomationEngine{actuatorCommandOut}, Actuator{commandIn}){
   // define connector processing and command flow
}

// Connectors for UI to AE and vice versa
connector UIToAE(UserInterface{uiCommandIn, dataDisplayOut},
AutomationEngine{sensorDataIn, actuatorCommandOut}){
   // define connector for UI interactions
}
```

5. Traceability between Requirements and Architecture

Microservice-Based Layered Architecture

Architectural Decision: The system employs a microservice-based layered architecture, comprising Data Collection, Processing, Actuator Control, and User Interface layers, each designed to handle specific functions within the system.

Data Collection Layer

Requirement: Efficient and reliable data collection by sensors.
Architectural Decision: Sensors operate as individual microservices in this layer, focusing solely on collecting environmental data and transmitting it to the Processing Layer (AE). This design enhances the reliability and efficiency of data collection.

Processing Layer (Automation Engine)

Requirement: Real-time data processing and decision-making based on sensor inputs.
Architectural Decision: The AE, central to this layer, processes incoming data from sensors, applies conditional logic, and sends commands to actuators when necessary. This setup ensures timely and accurate responses to environmental changes.

Actuator Control Layer

Requirement: Responsive and precise control of actuators based on AE commands.
Architectural Decision: Actuators function as microservices, receiving commands from the AE and performing physical actions. This layer's focus on direct actuator control enables quick and precise responses to AE commands.

User Interface Layer

Requirement: Intuitive user interface for monitoring and controlling the system.
Architectural Decision: The Desktop App interfaces with the AE, providing users with real-time data visualization and control over sensors and actuators. This layer enhances user interaction with the system, ensuring a user-friendly experience.

Network and Communication Infrastructure

Requirement: Flexible and robust communication between sensors, AE, and actuators.
Architectural Decision: The system supports both wired and wireless communication methods, utilizing TCP/IP protocol and JSON format for data transmission. This infrastructure ensures reliable and adaptable connectivity across the system.

Real-Time Data Processing and Concurrent Storage in AE
Requirement: Simultaneous processing and storage of incoming sensor data.
Architectural Decision: The AE is configured to process sensor data in real-time as soon as it is received, ensuring immediate decision-making and action. Concurrently, this data is also written to a storage system. This dual approach allows for real-time responsiveness while also maintaining a record of the data for historical analysis, auditing, or backup purposes.

Existing Points
This configuration enhances the system's capability to not only react promptly to environmental changes but also to maintain a historical record of sensor data. The real-time processing ensures immediate action based on current data, while the simultaneous storage provides a comprehensive dataset for future analysis, trend observation, or system optimization. The choice of storage technology would be crucial here, balancing between write efficiency and data retrieval performance.

Integrated Data Processing in AE
Requirement: Combined analysis of data from multiple sensor types (soil and weather) for comprehensive decision-making.
Architectural Decision: The AE is designed to integrate data from both soil and weather sensors. It will compile and analyze this combined data to gain a holistic understanding of environmental conditions. Based on this integrated analysis, the AE will make informed decisions and send appropriate commands to actuators.

Existing Points
This integrated approach to data processing enables the AE to make more accurate and context-aware decisions. By considering a broader range of environmental factors, the system can optimize its responses, enhancing efficiency and effectiveness. For instance, the AE might analyze soil moisture levels in conjunction with weather predictions to make more intelligent irrigation decisions. This integration not only improves immediate operational responses but also contributes to long-term resource management and environmental sustainability.

Use of Linux Operating System in AE
Architectural Decision: The Automation Engine (AE) will operate on the Linux operating system. This decision is based on several key advantages that Linux provides, suitable for the needs of our system.

Advantages of Linux:

Stability and Reliability: Linux is known for its stability and reliability, which are crucial for continuous, uninterrupted operation of the AE. This ensures that the system remains operational and efficient over long periods.

Security: Linux offers strong security features, which is vital for protecting the data processed by the AE, especially considering the sensitivity of environmental data from sensors.

Customizability: The open-source nature of Linux allows for greater customizability. We can tailor the operating system to meet the specific needs of our AE, optimizing its performance and functionality.

Resource Efficiency: Linux is generally more resource-efficient compared to other operating systems, making it suitable for systems where optimal resource utilization is important.

Existing Points
The choice of Linux as the operating system for AE aligns with our goal of creating a robust, secure, and efficient automation system. It supports our system's requirements for reliability, security, and performance. Furthermore, the ability to customize and optimize the OS allows for a more tailored solution, ensuring that the AE operates optimally within the context of our specific system requirements.

Requirement: Precise and Timely Execution of Actions
Architectural Decision: The system's actuators are designed to respond accurately and promptly to commands from the AE. This is achieved through real-time communication protocols and efficient command-processing algorithms within each actuator.
Justification: This decision ensures that the system's actions are carried out exactly as intended, minimizing delays and inaccuracies, which are crucial for maintaining operational efficiency and effectiveness.

Requirement: Adaptability to Various Operational Conditions
Architectural Decision: Actuators are equipped with mechanisms to adjust their operations based on varying environmental conditions and command parameters. This includes the ability to modulate factors like flow rate in water hoses, temperature and humidity settings in climate control, and spray intensity in pesticide sprayers.
Justification: This flexibility allows the system to adapt to different scenarios, optimizing performance and resource utilization in diverse conditions.

Requirement: Remote and Automated Control
Architectural Decision: Integration of actuators with a centralized control system (the AE), allowing for both automated control based on sensor inputs and remote manual control via the user interface.
Justification: This approach provides a balance between automation for efficiency and manual override for human discretion, enhancing the system's versatility and user control.

Requirement: Continuous Operation and Durability
Architectural Decision: Selection of robust and durable materials for actuators, along with design choices that prioritize long-term operation and minimal maintenance.
Justification: Ensuring the longevity and reliability of actuators is essential for a system that aims to operate continuously with minimal downtime.

Requirement: Feedback and Status Monitoring
Architectural Decision: Actuators are designed to provide feedback on their status and the execution of commands, which is relayed back to the AE and, if necessary, to the user interface.
Justification: Feedback mechanisms are crucial for monitoring the system's performance, diagnosing issues, and enabling proactive maintenance.

Requirement: Energy Efficiency and Environmental Consideration
Architectural Decision: Actuators are designed with energy efficiency in mind, using technologies that minimize power consumption and environmental impact.
Justification: This aligns with sustainability goals and reduces operational costs, making the system more eco-friendly and cost-effective.

Requirement: Safety and Compliance with Regulations
Architectural Decision: Ensuring that all actuators comply with relevant safety standards and regulations, incorporating necessary safety features and fail-safes.
Justification: Compliance with safety standards is non-negotiable for protecting both the system and its users, as well as for legal adherence.

Software and Hardware Compatibility
While our architectural design decisions aim to establish a robust and efficient system, one area where we face constraints is in the integration of hardware with the software, particularly regarding the sensors and actuators' integration with the Automation Engine (AE).

Limitation Explanation: Financial and Time Resources: Our project is currently limited by financial and temporal resources, which impacts our ability to fully explore and implement hardware integration. Specifically, these constraints affect:

Hardware Acquisition: The budgetary constraints restrict the range and sophistication of hardware (sensors and actuators) that we can procure. This limitation means we might not be able to access the latest or most advanced devices compatible with our software architecture.

Development Time: Our project timeline is tight, which limits the extent to which we can develop and test custom integration solutions for hardware. With more time, we could potentially create more sophisticated integration protocols that would enhance the interaction between our software (AE) and the hardware components.

Testing and Optimization: Limited resources also impact our ability to extensively test and optimize the hardware-software interface. Comprehensive testing is crucial for ensuring seamless integration, but our current constraints mean that this aspect might not be as thorough as desired.

Impact: These limitations primarily affect the 'Software and Hardware Compatibility' aspect of our system. While we strive to ensure the AE can effectively communicate with and control the sensors and actuators, the full potential of hardware-software integration might not be realized due to these constraints. However, we are committed to optimizing within these limits and ensuring that the system remains functional and effective for its intended purposes.

6. Model-to-Code Transformation Algorithm Specifications

6.1. Sensor Data Collection

```
Algorithm: CollectSensorData
Input: Sensor parameters (e.g., type, range, frequency)
Output: Sensor data

Begin
   InitializeSensor(sensor parameters)
   While true do
      If CheckSensorStatus() is OK Then
         data = ReadSensorData(sensor parameters)
         If ValidateData(data) is True Then
            StoreDataInBuffer(data)
         Else
            LogError("Invalid sensor data")
         End If
      Else
         LogError("Sensor malfunction")
      End If
      Wait for the specified frequency duration
   End while
End
```

6.2. Data Transmission to AE

```
Algorithm: TransmitDataToAE
Input: Sensor data, AE address
Output: Confirmation of data transmission

Begin
   If EstablishTCPConnection(AE address) is Successful Then
      For each data in sensor data buffer do
         jsonData = ConvertDataToJson(data)
         If SendDataToAE(jsonData) is not Successful Then
            LogError("Data transmission failed")
         End If
      End for
      CloseTCPConnection()
      Return "Transmission successful"
   Else
      LogError("Failed to establish connection")
```

```
        Return "Transmission failed"
    End If
End
```

6.3. Real-Time Data Processing by AE

```
Algorithm: ProcessSensorData
Input: Incoming sensor data
Output: Decision/action

Begin
    While AE is Receiving Data do
        parsedData = ParseJsonData(incoming data)
        If CheckDataValidity(parsedData) is True Then
            decision = AnalyzeData(parsedData)
            ExecuteActionBasedOnDecision(decision)
        Else
            LogError("Invalid data received")
        End If
    End while
End
```

6.4. Actuator Control

```
Algorithm: ControlActuator
Input: Control command, actuator specifications
Output: Actuator status

Begin
    actuator = RetrieveActuator(specifications)
    If actuator is Not Null Then
        If control command is "Start" Then
            ActivateActuator(actuator)
        Else If control command is "Stop" Then
            DeactivateActuator(actuator)
        End If
        Return GetActuatorStatus(actuator)
    Else
        LogError("Actuator not found")
        Return "Actuator operation failed"
    End If
End
```

6.5. User Interface Interaction

```
Algorithm: UserInterfaceInteraction
Input: User commands
Output: System response

Begin
   InitializeDesktopApp()
   While Desktop App is Running do
      command = GetUserCommand()
      If ValidateCommand(command) is True Then
         response = HandleCommand(command)
         DisplayResponseOnUI(response)
      Else
         DisplayError("Invalid command")
      End If
   End while
End
```

6.6. Data Storage and Caching

```
Algorithm: StoreSensorData
Input: Sensor data
Output: Storage confirmation

Begin
   If ConnectToDataStorageSystem() is Successful Then
      For each data in sensor data do
         If StoreDataInStorageSystem(data) is not Successful Then
            LogError("Data storage failed")
         End If
      End for
      Return "Data storage successful"
   Else
      LogError("Failed to connect to storage system")
      Return "Data storage failed"
   End If
End
```

6.7. Integrated Data Processing in AE

```
Algorithm: IntegratedDataProcessing
Input: Soil sensor data, Weather sensor data
Output: Comprehensive decision

Begin
   combinedData = CombineData(soil sensor data, weather sensor data)
   If ValidateCombinedData(combinedData) is True Then
      analysisResult = AnalyzeCombinedData(combinedData)
      decision = MakeDecisionBasedOnAnalysis(analysisResult)
      SendCommandToActuator(decision)
   Else
      LogError("Invalid combined data")
      Return "Analysis failed"
   End If
End
```

6.8. Sensor Data Evaluation in AE

```
Algorithm: EvaluateSensorData
Input: SoilSensorData, WeatherSensorData
Output: EvaluationResult

Begin
   // Evaluate Soil Sensor Data
   soilHealthStatus = EvaluateSoilSensor(SoilSensorData)

   // Evaluate Weather Sensor Data
   weatherCondition = EvaluateWeatherSensor(WeatherSensorData)

   // Combine evaluations
   combinedEvaluation = CombineEvaluations(soilHealthStatus, weatherCondition)

   Return combinedEvaluation
End

Function EvaluateSoilSensor(SoilSensorData)
   // Define thresholds for soil conditions
   OptimalSoilHumidity = ... // Define optimal range or value
   OptimalNutrients = ... // Define optimal range or value for nitrogen, phosphorus, potassium

   If SoilSensorData.soilHumidityValue is within OptimalSoilHumidity and
      SoilSensorData.nitrogenValue, phosphorusValue, potassiumValue are within
OptimalNutrients Then
       Return "Soil Health: Good"
   Else
       Return "Soil Health: Needs Attention"
   End If
End Function

Function EvaluateWeatherSensor(WeatherSensorData)
   // Define thresholds or conditions for weather
   OptimalTemperature = ... // Define optimal range or value
   OptimalHumidity = ... // Define optimal range or value
   HighWindSpeedThreshold = ... // Define threshold for high wind speed

   If WeatherSensorData.temperatureValue is within OptimalTemperature and
      WeatherSensorData.humidityValue is within OptimalHumidity and
      WeatherSensorData.windSpeedValue < HighWindSpeedThreshold Then
       Return "Weather Condition: Favorable"
   Else
```

```
        Return "Weather Condition: Unfavorable"
    End If
End Function

Function CombineEvaluations(SoilHealthStatus, WeatherCondition)
    // Combine soil and weather evaluations for a comprehensive view
    Return SoilHealthStatus + "; " + WeatherCondition
End Function
```

7. GUI Demonstration

https://www.youtube.com/channel/UCQ1prPPYelA83CT4dtzIi_w

8. Lessons Learned

Requirements and Planning

We have encountered several disagreements during the design process of the development. We had different opinions on how to handle the requirements and from which perspective to approach them. Some solutions we've come up with were very hard to model, some solutions had very simple models but lacked functionality. We had some changes along the way. There were times we realized we had a different view of the project entirely because we had left out planning the fundamentals thoroughly. Every little part we left out at the beginning became harder to deal with after every single progress. We kept going back and fixing the foundation and this taught us the necessity of thoroughness during the planning phase of a project.

Effective Communication:

We have witnessed the value of effective communication in the team to avoid any conflicts of perspectives in the future of the development. We have experienced that when a tiny detail is left undiscussed, it becomes a lot harder to deal with later because the entire image of the project in our minds may have to change due to that detail. If we try to create similar images about the project as much as we can, there will be a lot less problems and conflicts to deal with as we progress and we can achieve the best result.

Time Management

We experienced the challenges of time management since we are both senior students who are about to graduate and we have several different projects and have to manage our time so that we can make the best of each one.

Collaboration and Teamwork

We have experienced issues in this area. We could not collaborate effectively at the beginning of the project. We have had to make changes of plan after we learned that our third group member was not going to participate in the project. Then we both had to work harder to get this project done. We started to take more responsibility and played through our strengths.

Documentation and Knowledge Transfer

We learned that we can communicate and understand each other's perspectives a lot better with proper documentation. As we progressed through our report, we began to really realize what points we were trying to make to each other. Whether it is a diagram or a paragraph, the documentation made what we were thinking a lot more clear for each other to see.