Department of Computer Engineering

CENG350 Software Engineering

Software Architecture Description (SAD) for FarmBot

Group 52

By

Emre Çam, 2518843

Tufan Özkan, 2580850

Saturday 18th May, 2024

Contents

List of Figures iv

List of Tables v

1 Introduction 1

1.1 Purpose and objectives of FarmBot 1

1.2 Scope 2

1.3 Stakeholders and their concerns 3

2 References 6

3 Glossary 7

4 Architectural Views 8

4.1 Context View 8

4.1.1 Stakeholders' uses of this view 8

4.1.2 Context Diagram 9

4.1.3 External Interfaces 11

4.1.4 Interaction scenarios 16

4.2 Functional View 18

4.2.1 Stakeholders' uses of this view 18

4.2.2 Component Diagram 19

4.2.3 Internal Interfaces 20

4.2.4 Interaction Patterns 25

4.3 Information View 28
4.3.1 Stakeholders' uses of this view 28
4.3.2 Database Class Diagram 28
4.3.3 Operations on Data 30
4.3.3.1 Operations and Descriptions 30
4.3.3.2 CRUD Effects of Operations 34
4.4 Deployment View 40
4.4.1 Stakeholders' uses of this view 40
4.4.2 Deployment Diagram 41
4.5 Design Rationale 42

5 Architectural Views for Your Suggestions to Improve the Existing System 45

5.1 Context View 45
5.1.1 Stakeholders' uses of this view 45
5.1.2 Context Diagram 46
5.1.3 External Interfaces 47
5.1.4 Interaction scenarios 51
5.2 Functional View 52
5.2.1 Stakeholders' uses of this view 52
5.2.2 Component Diagram 52
5.2.3 Internal Interfaces 54
5.2.4 Interaction Patterns 58
5.3 Information View 58
5.3.1 Stakeholders' uses of this view 58
5.3.2 Database Class Diagram 58
5.3.3 Operations on Data 60
5.3.3.1 Operations and Descriptions 60
5.4 CRUD Effects of Operations 61
5.5 Deployment View 62

5.5.1 Stakeholders' uses of this view 62
5.5.2 Deployment Diagram 63
5.6 Design Rationale 65

List of Figures

4.1 System Context Diagram 10
4.2 Class Diagram for External Interfaces 11
4.3 Identify Plants Activity Diagram 16
4.4 Water Based On Weather Activity Diagram 17
4.5 FarmBot Component Diagram 19
4.6 Internal Interfaces of FarmBot 20
4.7 Mow All Weeds - Sequence Diagram 25
4.8 Implement Event - Sequence Diagram 26
4.9 Water All Plants - Sequence Diagram 27
4.10 FarmBot Database Classes Diagram 29
4.11 Deployment Diagram 42

5.1 System Context Diagram with Suggestions 47
5.2 Suggested Class Diagram For External Interfaces 48
5.3 Alerting Mechanism Activity Diagram 51
5.4 Component Diagram With Suggestion 53
5.5 Internal Interfaces With Suggestion 54
5.6 Sequence Diagram For Suggestion 58
5.7 Database Class Diagram With Suggestion 59
5.8 Deployment Diagram with Suggestion 64

List of Tables

1 Revision History vi

4.1 Operations On Data 34
4.2 CRUD Effects of Operations 40

5.1 Suggested Operations On Data 61
5.2 Crud Effects of Suggested Operations 62

Revision History

Name	Date	Change Reason	Version
SAD First Draft	May 10, 2024	Initial Commit	1.0.0
SAD Final	May 17, 2024	Final Arrangement	1.0.1

Table 1: Revision History

1. Introduction

This document provides the Software Architecture Description (SAD) of an open-source precision agriculture CNC farming project that lets the user grow food from anywhere with a web app. The website for this system is https://farm.bot.

1.1 Purpose and objectives of FarmBot

In the scope of this system, Farmbot is humanity's open-source CNC farming machine that automatically grows food right in your backyard keeping you in complete control, as opposed to the current food production system which no longer has control over how the food is produced.

- Farmbot is provided with 95 pre-assembled elements to be set up quickly and easily and perform the basic functions needed to grow a garden.
- Raspberry Pi computers operate the system with its hardware components (webcam, Arduino firmware microcontroller combined with powerful stepper motors and dynamic devices, ph sensors) and user application requests.
- Farmduino microcontrollers allow positioning of the tool head with millimetre accuracy for sowing, and watering plants in any pattern and frequency with given built-in features of plants based on their type, age, and soil conditions. The sensors also send the condition of soil and weather data of pH, temperature, and moisture.

- The Onboard camera system allows user to monitor their garden, and capture images, and the farmbot can take action by detecting the weeds utilising advanced computer vision.
- The web and mobile application allows users to control the garden remotely, and configure or update the software of Raspberry Pi computers without any need for hardware change.
- The farm designer and sequence editors allow users to lay out their plants with built-in data for plants, creating optimal layouts each season, and taking care of their plants in the way they want without requiring any coding.
- The farmbot software development allows professionals to modify and use it for any purpose (e.g. education, business) in hardware tools or develop completely new versions of the farmbot based on the existing library of components.

1.2 Scope

In the scope of this system, Farmbot is humanity's open-source CNC farming machine that automatically grows food right in your backyard keeping you in complete control, as opposed to the current food production system which no longer has control over how the food is produced.

- Farmbot is provided with 95 pre-assembled elements to be set up quickly and easily and perform the basic functions needed to grow a garden.
- Raspberry Pi computers operate the system with its hardware components (webcam, Arduino firmware microcontroller combined with powerful stepper motors and dynamic devices, ph sensors) and user application requests.
- Farmduino microcontrollers allow positioning of the tool head with millimetre accuracy for sowing, and watering plants in any pattern and frequency with given built-in features of plants based on their type, age, and soil conditions. The

sensors also send the condition of soil and weather data of pH, temperature, and moisture.

- The Onboard camera system allows user to monitor their garden, and capture images, and the farmbot can take action by detecting the weeds utilising advanced computer vision.
- The web and mobile application allows users to control the garden remotely, and configure or update the software of Raspberry Pi computers without any need for hardware change.
- The farm designer and sequence editors allow users to lay out their plants with built-in data for plants, creating optimal layouts each season, and taking care of their plants in the way they want without requiring any coding.
- The farmbot software development allows professionals to modify and use it for any purpose (e.g. education, business) in hardware tools or develop completely new versions of the farmbot based on the existing library of components.

1.3 Stakeholders and their concerns

- **Botanic Students**
  - **Objective**: Utilize FarmBot Express as an educational tool to learn about botany.
  - **Characteristics**:
    - Interested in studying plant biology, growth patterns, and environmental interactions.
    - Seek practical applications to extend understanding of botany.
    - May have varying levels of prior knowledge in botany.
- **Engineering Students**

  - **Objective**: Utilize FarmBot Express as an educational tool to learn about robotics and coding.
  - **Characteristics**:
    * Seek hands-on learning experiences to deepen understanding of robotics and coding.
    * May have varying levels of prior knowledge in robotics and computer science.

- **Scientists**
  - **Objective**: Utilize FarmBot Express for conducting experiments with plants and conducting research.
  - **Characteristics**:
    * Engaged in scientific research related to plant biology and environmental science.
    * Require precise control over experimental conditions, including watering, light exposure, and soil nutrients.
    * May have advanced knowledge of botany, molecular biology, or related fields.
  - **Some applications**:
    * [1]
    * [2]

- **Garden Enthusiasts**
  - **Objective**: Enhance gardening skills and optimize plant growth using automated techniques.
  - **Characteristics**:
    * Passionate about gardening and plant cultivation.
    * Desire to leverage technology for improving crop yields and efficiency.

    - May have varying levels of experience in traditional gardening practices.

- **Educators**
  - **Objective**: Integrate FarmBot Express into educational purposes to teach agricultural concepts and engineering skills.
  - **Characteristics**:
    - Professionals in the field of education with a focus on robotics or agricultural instruction.
    - Seek innovative teaching tools to engage students and enhance learning outcomes.

- **Home Gardeners**
  - **Objective**: Simplify and automate gardening tasks for hobby or sustenance gardening.
  - **Characteristics**:
    - Enjoy gardening as a leisure activity or means of producing food.
    - Value convenience and time-saving solutions for maintaining garden plots.
    - May have experience in traditional gardening methods and programming.

2. References

[1] S. Adebola, M. Presten, R. Parikh, S. Aeron, S. Mukherjee, S. Sharma, M. Theis, W. Teitelbaum, E. Solowjow, and K. Goldberg, "Automated pruning and irrigation of polyculture plants," *IEEE Transactions on Automation Science and Engineering*, pp. 1–12, 2024.

[2] M. Ruppel, S. Nelson, G. Sidberry, M. Mitchell, D. R. Kick, S. Thomas, K. E. Guill, M. J. Oliver, and J. D. Washburn, "Rootbot: High-throughput root stress phenotyping robot," *Applications in Plant Sciences*, vol. 11, no. 6, 2023.

3. Glossary

Arduino A single-board microcontroller for building digital devices..

CNC Computer Numerical Control.

Raspberry Pi A series of small single-board computers.

SAD System Analysis and Design.

4. Architectural Views

4.1 Context View

4.1.1 Stakeholders' uses of this view

The Context View provides stakeholders with a holistic understanding of how FarmBot Express fits into various contexts and fulfils different needs.

- **Botanic Students:** To comprehend how FarmBot Express fits into their research on plant biology and environmental interactions, they would make use of the Context View. They concentrate on elements associated with tracking plant growth patterns and conducting environmental experimentation.

- **Engineering Students:** The Context View would be examined by engineering students to see how FarmBot Express can be used as a useful tool for learning robotics and coding. Technical details like the sensors, hardware, and programming interface may pique their interest.

- **Scientists:** Researchers would examine the Context View to evaluate the accuracy and control that FarmBot Express provides for carrying out plant experiments. They concentrate on features like integration with other scientific instruments, adaptable experimental setups, and data logging.

- **Garden Enthusiasts:** Context View is what gardeners would use to investigate how FarmBot Express can improve gardening practices. Features like crop optimization strategies, soil monitoring, and automated irrigation are of interest to

them.

- **Educators:** Teachers can use the Context View to see how FarmBot Express fits into educational environments. They search for characteristics that support experiential learning and correspond with engineering and agriculture curriculum objectives.
- **Home Gardeners:** Home gardeners would refer to the Context View to evaluate how FarmBot Express can simplify and automate their gardening tasks. They focus on features such as remote monitoring, customizable planting schedules, and compatibility with different garden layouts.

4.1.2 Context Diagram

FarmBot is not part of a large system nor necessarily requires the download of an application. Hence customers shall go into the web browser of the app and configure their FarmBot using their laptop, tablet, or smartphone. The web application features real-time manual controls and logging, a sequence builder for custom routines for FarmBot to execute. The web application keeps the data and receives the built-in data for plants from the database management system. The machine maintains a connection and synchronises with the web application via the message broker. Besides that, Farmbot communicates with the Arduino firmware system over a serial connection to send commands for motor and tool positioning and receive collected from sensors and rotary encoders. Moreover, the FarmBot is connected to a monitoring system through a webcam to provide real-time view, image-capturing, and detecting weeds, taking actions according to it. Moreover, the Lua compiler allows users to write custom scripts to tailor the behaviour of their FarmBot devices to their specific needs and preferences. Furthermore, the system provides an IT staff interface in case of loggings, errors and customary configurations to help customers.

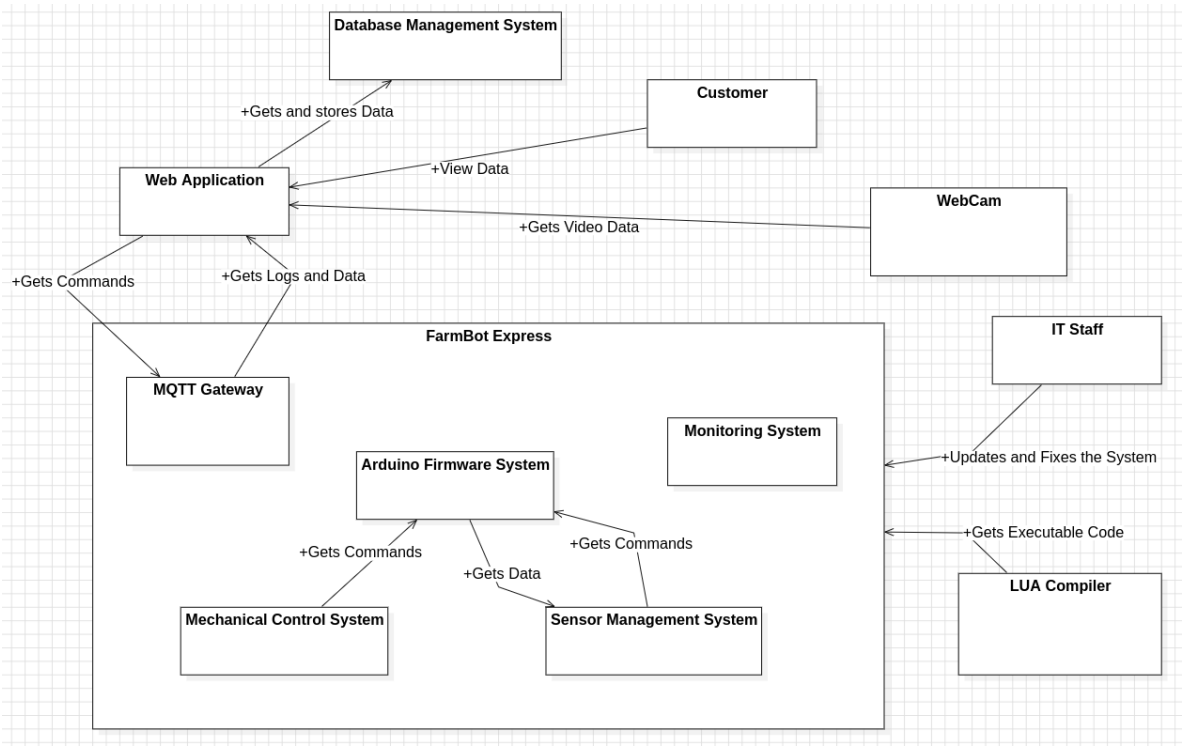

Figure 4.1: System Context Diagram

4.1.3 External Interfaces

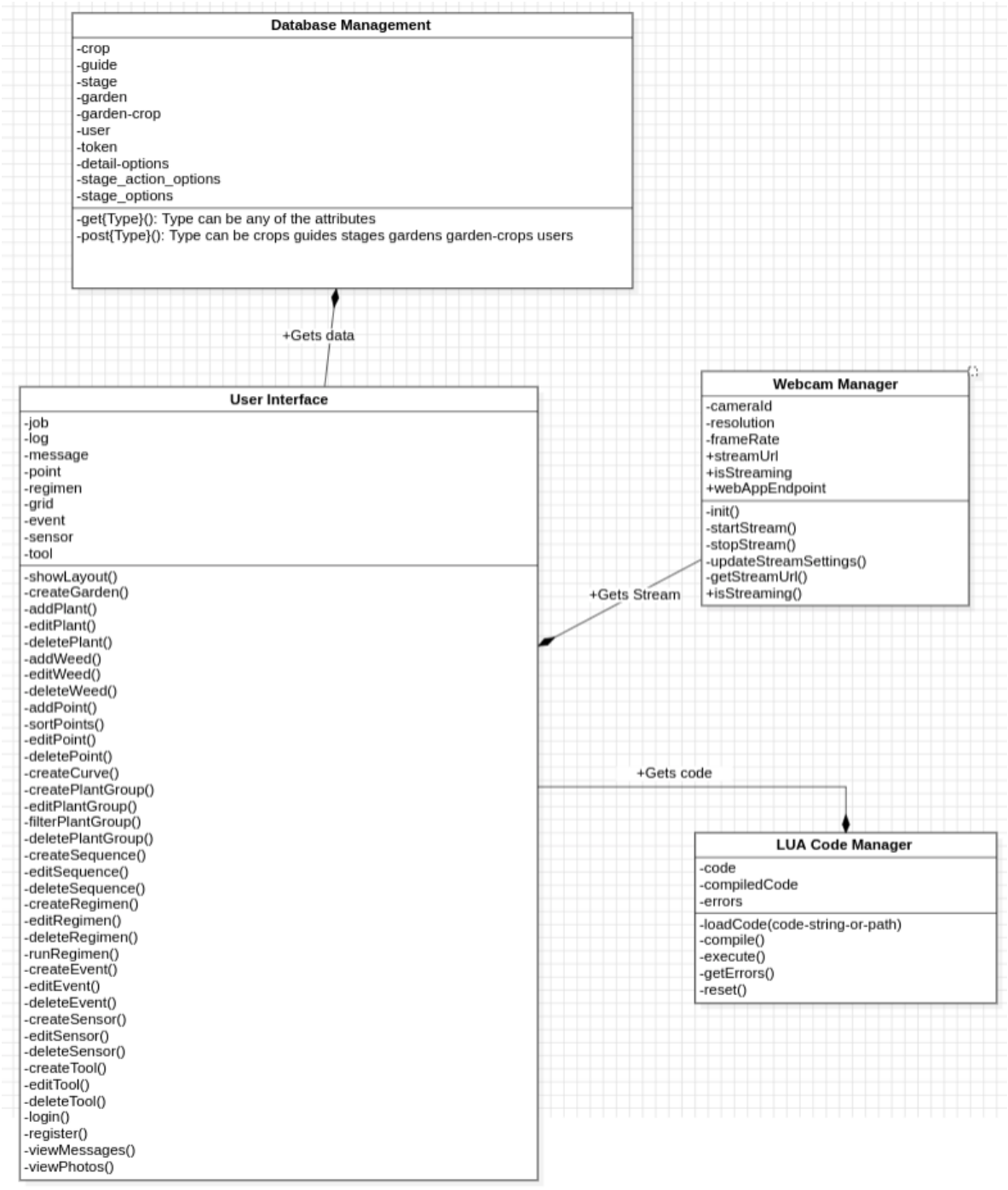

Figure 4.2: Class Diagram for External Interfaces

- **User Interface**

  The user utilizes the user interface to control and monitor the farming processes, such as planting, watering, and harvesting, through a web-based platform or mobile application.

  - ***job*** **:** This field represents a system operation to be executed.
  - ***log*** **:** This field stores the records of activities and events, commands.
  - ***message*** **:** This field manages communication between the user interface and the Raspberry Pi computer.
  - ***point*** **:** This field keeps the 3D position data.
  - ***regimen*** **:** This field stores the schedule for garden operations.
  - ***grid*** **:** This field manages the layout of the garden in a modular manner.
  - ***event*** **:** This field refers to occurrences or commands related to the garden operations.
  - ***sensor*** **:** This field keeps the sensor data that are received from other systems.
  - ***tool*** **:** This field manages mechanical control tools within the system.
  - ***showLayout()*** **:** This method displays the layout or visual representation of the farm.
  - ***createGarden()*****:** Creates a new garden area within the farm layout.
  - ***addPlant()*****:** Adds a new plant to the garden.
  - ***editPlant()*****:** Modifies the details of an existing plant.
  - ***deletePlant()*****:** Removes a plant from the garden.
  - ***addWeed()*****:** Adds a weed to the garden for management.
  - ***editWeed()*****:** Modifies the details of a weed.
  - ***deleteWeed()*****:** Removes a weed from the garden.

- ***addPoint()*:** Adds a new point or location to the farm layout.
- ***sortPoints()*:** Sorts the points within the layout.
- ***editPoint()*:** Modifies the details of a specific point.
- ***deletePoint()*:** Removes a point from the layout.
- ***createCurve()*:** Creates a curved path or line within the layout.
- ***createPlantGroup()*:** Forms a group of plants for collective management.
- ***editPlantGroup()*:** Modifies the details of a plant group.
- ***filterPlantGroup()*:** Filters plants based on specified criteria within a group.
- ***deletePlantGroup()*:** Removes a plant group from the layout.
- ***createSequence()*:** Creates a sequence of actions or tasks for automation.
- ***editSequence()*:** Modifies the details of a sequence.
- ***deleteSequence()*:** Removes a sequence from the system.
- ***createRegimen()*:** Establishes a regimen or schedule for farm operations.
- ***editRegimen()*:** Modifies the details of a regimen.
- ***deleteRegimen()*:** Removes a regimen from the system.
- ***runRegimen()*:** Initiates the execution of a regimen.
- ***createEvent()*:** Creates an event or activity within the system.
- ***editEvent()*:** Modifies the details of an event.
- ***deleteEvent()*:** Removes an event from the system.
- ***createSensor()*:** Adds a new sensor to the farm environment.
- ***editSensor()*:** Modifies the details of a sensor.
- ***deleteSensor()*:** Removes a sensor from the system.
- ***createTool()*:** Adds a new tool or equipment to the farm.
- ***editTool()*:** Modifies the details of a tool.

- ***deleteTool()*:** Removes a tool from the system.
- ***login()*:** Handles the authentication and login process for users.
- ***register()*:** Manages the registration of new users within the system.
- ***viewMessages()***: Displays messages or notifications for the user.
- ***viewPhotos()*:** Displays photos or images related to the farm or operations.

- **Database Management Interface**

  The DBMS stores and manages the vast amount of data generated by the farming processes and sensor readings. It ensures data integrity, facilitates data retrieval for analysis, and supports the historical tracking of farming activities for performance evaluation and planning.

  - ***crop*:** This field keeps the types of plants in the inventory that can be grown inside the garden.
  - ***guide*:** This field provides instructions for the cultivation of crops.
  - ***stage*:** This field keeps the info on various crops' phases of development.
  - ***garden*:** This field specifies the space of the garden for planting crops.
  - ***garden-crop*:** This field keeps the info of planted crops in the garden.
  - ***user*:** This field represents the owner of the system with their registration info.
  - ***token*:** This field provides secure access for authentication purposes of the system.
  - ***detail-options*:** This field specifies the detailed information with options and their definitions about crops, gardens or other system components and elements.
  - ***stage-action-options*:** This field provides the various actions that can be took for the particular development stages of crops.
  - ***stage-options*:** This field provides different options for managing the growth phases of crops planted in the garden.

- ***getType()*:** This method is used for retrieving data of the entered field.
- ***postType()*:** This method is used for updating data of the entered field in the FarmBot database.

- **LUA Code Manager Interface**

  This interface provides a convenient way to manage Lua code compilation and execution, while also allowing for error handling and resource management.

  - ***code*:** This field stores the LUA code string or file paths.
  - ***compiledCode*:** This field stores the compiled bytecode after compilation.
  - ***errors*:** This fields stores any compilation errors encountered.
  - ***loadCode()*:** This method loads the Lua code from a string or file.
  - ***compile()*:** This method compiles the loaded LUA code into LUA bytecode.
  - ***execute()*:** This method executes the compiled LUA bytecode.
  - ***getErrors()*:** This method retrieves any compilation errors.
  - ***reset()*:** This field resets the manager, clearing loaded code and compilation results.

- **Webcam Manager Interface**

  The webcam manager interface provides a simple and extensible way to manage webcam streams for real-time monitoring in the FarmBot web application.

  - ***cameraId*:** A unique identifier for the webcam.
  - ***resolution*:** Specifies the resolution of the video stream.
  - ***frameRate*:** Defines the number of frames per second for the video stream.
  - ***streamUrl*:** The URL where the live video stream can be accessed.
  - ***isStreaming*:** A boolean flag indicating whether the webcam is currently streaming.

- ***webAppEndpoint*:** The base URL or endpoint of the web app where the video feed will be available.
- ***init(self, cameraId, resolution, frameRate, webAppEndpoint)*:** Initializes the Webcam Manager with the given camera ID, resolution, frame rate, and web app endpoint.
- ***startStream(self)*:** Begins the video stream from the webcam.
- ***stopStream(self)*:** Stops the video stream from the webcam.
- ***updateStreamSettings(self, resolution=None, frameRate=None)*:** Updates the resolution and frame rate settings for the video stream.
- ***getStreamUrl(self)*:** Returns the URL where the video stream is available.

4.1.4 Interaction scenarios

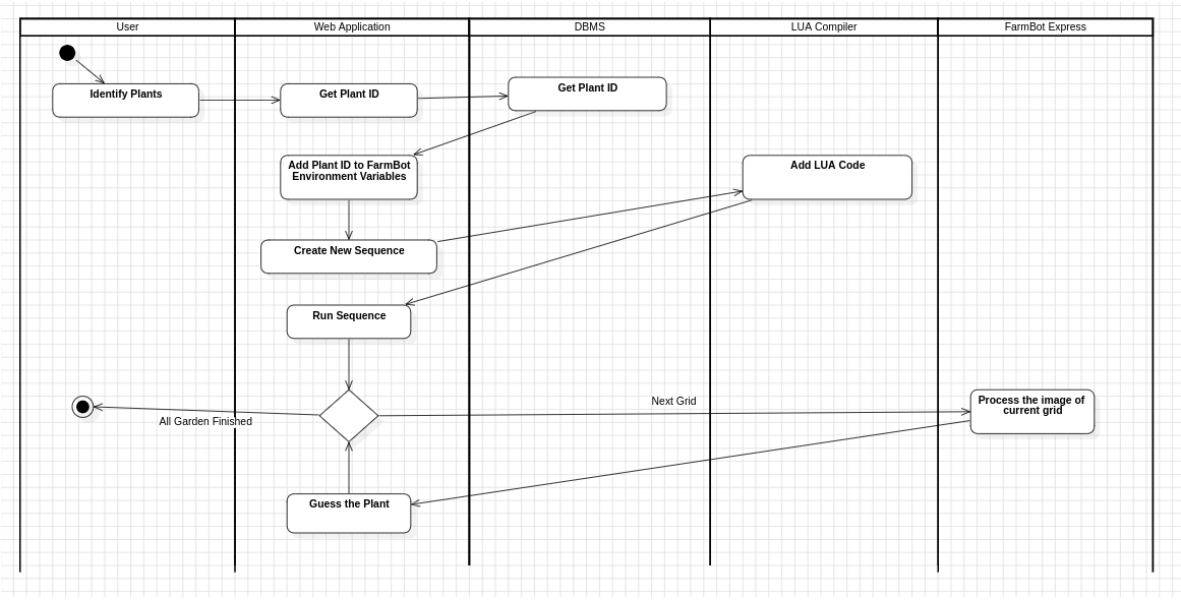

Figure 4.3: Identify Plants Activity Diagram

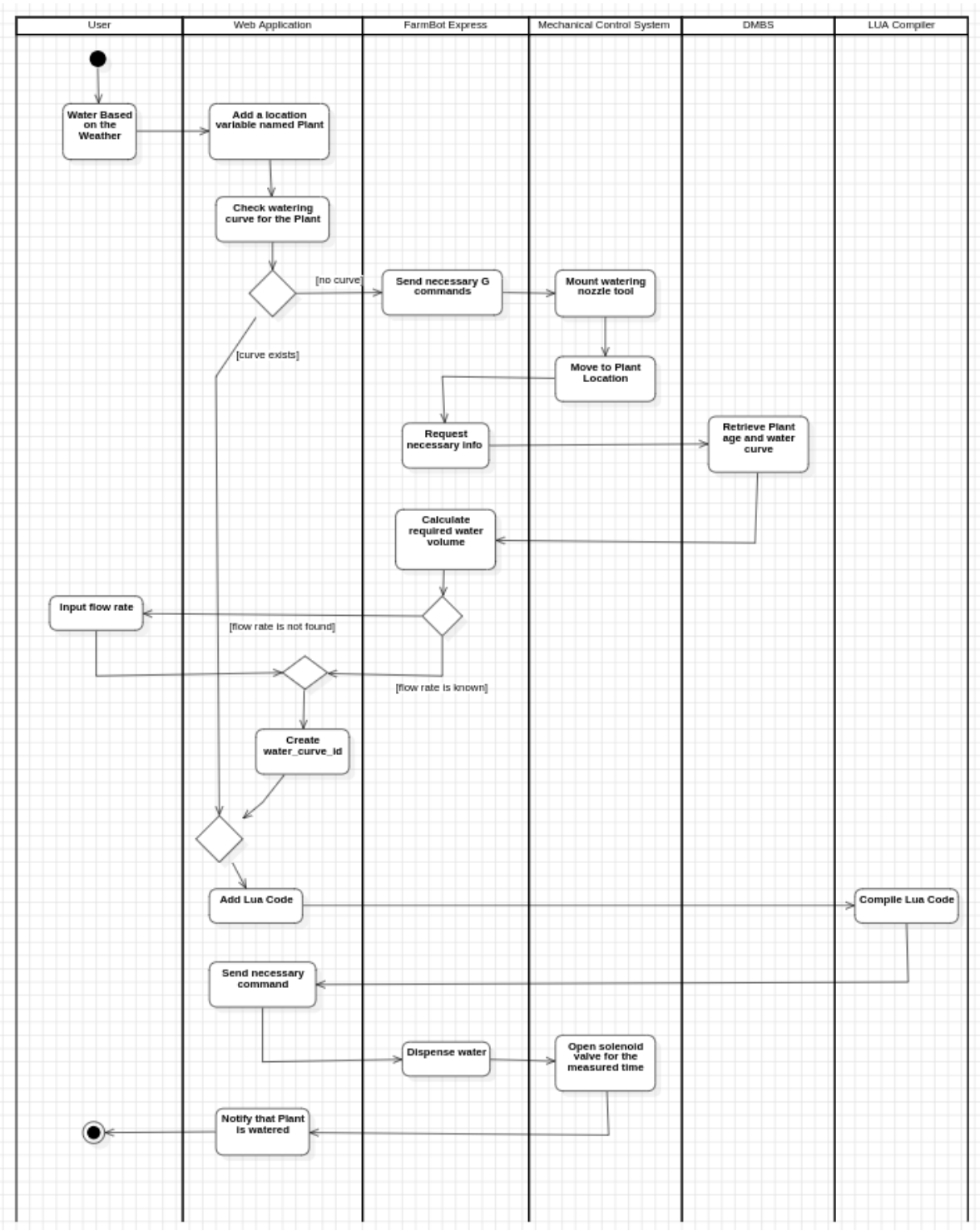

Figure 4.4: Water Based On Weather Activity Diagram

4.2 Functional View

The functional view of the FarmBot Software Architecture Document provides stakeholders with a understanding of how the system's runtime functionalities are organized and interact with each other. It includes detailed descriptions of components, their relationships, and the interfaces through which they communicate.

4.2.1 Stakeholders' uses of this view

- **Botanic Students:** Botanic students may utilize the Functional View to understand the underlying mechanisms and technologies that enable FarmBot Express to interact with plants and the environment.
- **Engineering Students:** Engineering students would likely explore the Functional View to comprehend the system's architecture from a technical perspective. They may examine components such as controllers, and communication protocol. This understanding can support their learning in the field of robotics and automation.
- **Scientists:** Scientists may analyze the Functional View to assess the system's capabilities for conducting plant experiments. They might focus on components related to data acquisition, and integration with scientific instruments to evaluate the suitability of FarmBot Express for their research needs.
- **Garden Enthusiasts and Home Gardeners:** Garden enthusiasts could refer to the Functional View to understand how FarmBot Express facilitates automated gardening tasks. This understanding can help them leverage the system's features to maintain healthy and productive gardens with minimal manual intervention.
- **Educators:** Educators might use the Functional View to illustrate the concepts of robotics, automation, and software architecture in educational settings. They may focus on explaining system components, their interactions, and the principles underlying their operation to engage students in hands-on learning experiences.

4.2.2 Component Diagram

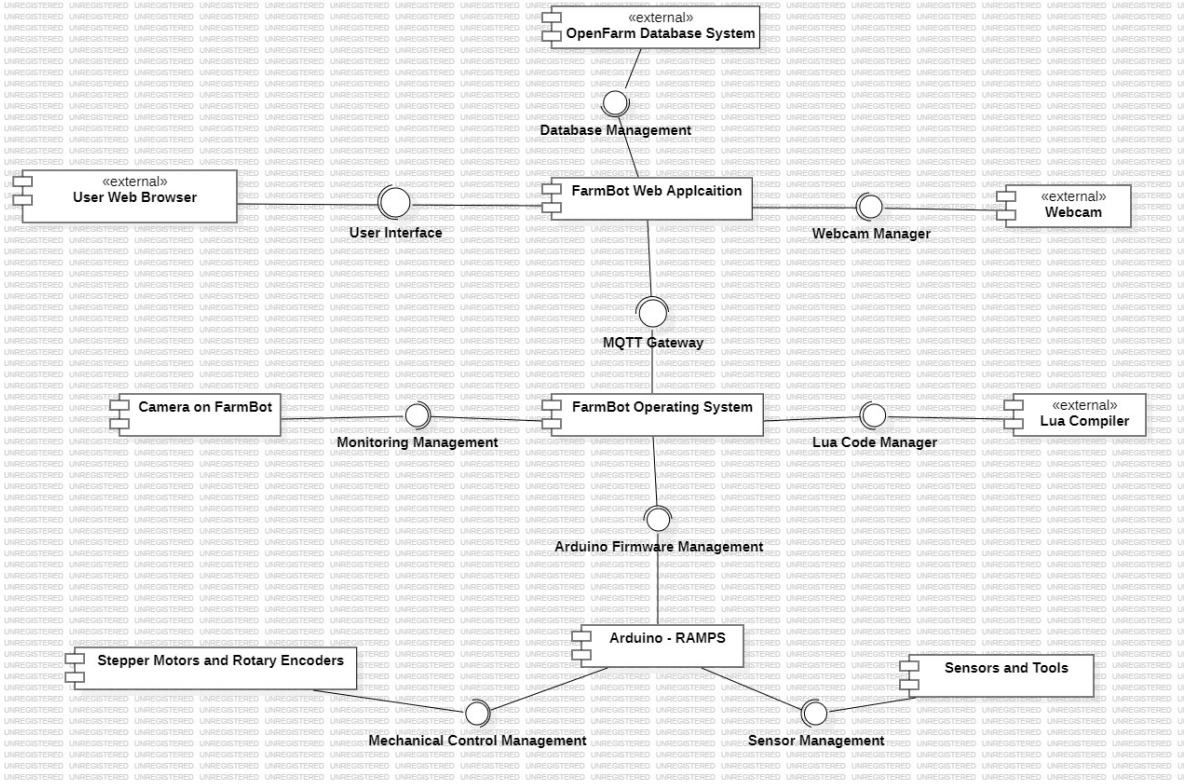

Figure 4.5: FarmBot Component Diagram

The FarmBot system comprises internal components like the FarmBot OS, Arduino-RAMPS, sensors, tools, stepper motors, encoders, and cameras, which work together to automate gardening tasks. The FarmBot OS serves as the central control hub, managing operations, processing user commands, and coordinating hardware actions. Sensors gather environmental data, while tools execute tasks like planting and watering under the control of Arduino-RAMPS. Stepper motors ensure precise movement, guided by feedback from encoders. User interactions primarily occur through the web application and browser, facilitating remote monitoring and control. External components like the Lua compiler enable customization through scripting, while the OpenFarm database provides essential plant care information. This comprehensive design ensures efficient, accurate, and user-friendly automated gardening, optimizing plant health and growth.

4.2.3 Internal Interfaces

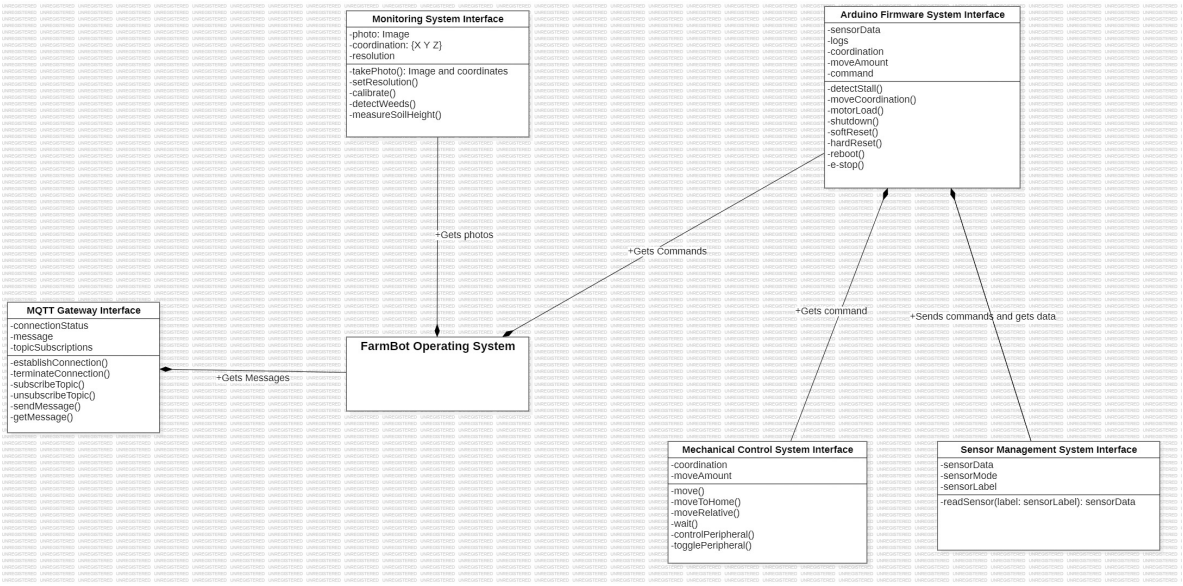

Figure 4.6: Internal Interfaces of FarmBot

- **Arduino Firmware System Interface**

  This firmware controls the operation of various actuators and sensors, executing commands sent from the user interface.

  - ***sensorData:*** This field keeps the data collected from various sensors such as temperature, pH, and moisture.
  - ***logs:*** This field stores the records of system events, sensor readings, errors and more for troubleshooting and maintenance.
  - ***coordination:*** This field keeps the data of spatial positions to be set accordingly.
  - ***moveAmount:*** This field keeps the data of steps for different motors on the mechanical system to take upon later.
  - ***command:*** This field refers to the action to be took later that is sent by the user interface.

- ***detectStall():*** This method alerts the system in case of stalling of the motors during an action.
- ***moveCoordination():*** This method moves the stepper drivers, needles, or sensor reader heads to the position that is kept in the coordination field.
- ***motorLoad():*** This method provides information about the load that motors are experiencing before an action to prevent potential failures.
- ***shutdown():*** This method powers down the Arduino and its associated systems.
- ***softReset():*** This method applies a software-based reset of the system, restoring to the predefined states of software components without affecting hardware.
- ***hardReset():*** This method applies a hardware-based reset of the system, restarting the microcontroller and associated hardware components.
- ***reboot():*** This method is similar to the softReset, additionally it performs some initialization on some components.
- ***e-stop():*** This method halts all system operations and motors in case of emergency to prevent worse damages.

- **Mechanical Control System Interface**

  It includes motors, actuators, and mechanisms for moving the robotic arm, controlling water flow, and manipulating tools for soil cultivation and plant care based on instructions received from the Arduino firmware.

  - ***coordination:*** This field keeps the data of spatial positions to be set accordingly.
  - ***moveAmount:*** This field keeps the data of steps for different motors on the mechanical system to take upon later.
  - ***move():*** This method moves the mechanical components to the position that is kept in coordination.

- ***moveToHome():*** This method makes mechanical components to return to the predefined 'Home' position.
- ***moveRelative():*** This method makes a movement on the mechanical components relative to the current position.
- ***wait():*** This method pauses the execution of motors.
- ***controlPeripheral():*** This method manages the components such as stepper drivers and encoders.
- ***togglePeripheral():*** This method powers up or down the peripherals of the mechanical control system.

- **Sensor Management System Interface**

  This interface ensures seamless integration of sensor data into the overall farming operation, enabling real-time monitoring and analysis of environmental conditions to optimize crop growth and resource utilization.

  - ***sensorData:*** This field keeps the data collected from various sensors.
  - ***sensorMode:*** This field refers to the configuration settings for the sensors.
  - ***sensorLabel:*** This field provides an identifier for each sensor.
  - ***readSensor():*** This method is used for retrieving the data from the sensors within the system.

- **Monitoring System Interface**

  It collects data and provides real-time feedback to users, enabling informed decision-making for optimized crop growth.

  - ***photo*:** This field keeps the image data.
  - ***coordination*:** This field refers to the spatial positioning by all three dimensions.

- ***resolution*:** This field provides a range of quality of the image data based on the processor's and camera's power.
- ***takePhoto()*:** This method captures image data of the current position's area of view with configured resolution settings.
- ***setResolution()*:** This method updates the resolution field to capture photos with new adjustments from now on.
- ***calibrate()*:** This method adjusts the parameters for optimal image data.
- ***detectWeeds()*:** This method utilises an algorithm to identify undesirable or harmful weeds for the garden within the monitored area.
- ***measureSoilHeight()*:** This method measures the depth of the soil within the monitored area to assess its usability.

- **MQTT Gateway Interface**

  This interface defines the operations available for interacting with the MQTT Gateway, which serves as the communication bridge between the FarmBot Web Application and the FarmBot embedded device operating System.

  - ***establishConnection()***: This operation establishes a communication link between the MQTT Gateway and either the Web Application or the Operating System running on the embedded device.
  - ***terminateConnection()***: When invoked, this operation closes the existing connection between the MQTT Gateway and its counterpart, whether it's the Web Application or the Operating System.
  - ***subscribeTopic()***: This operation allows the MQTT Gateway to subscribe to a specific topic on the MQTT broker. By subscribing to a topic, the Gateway indicates its interest in receiving messages published to that topic by other clients connected to the broker.
  - ***unsubscribeTopic()***: When called, this operation instructs the MQTT Gateway to stop receiving messages from a previously subscribed topic. It

removes the subscription from the Gateway's configuration, halting message delivery for that topic.

- ***sendMessage()***: This operation facilitates the transmission of a message from a counterpart to MQTT Gateway. It stores the message in the desired topic.
- ***getMessage()***: When invoked, this operation retrieves the next pending message from the Gateway's message buffer.

4.2.4 Interaction Patterns

These are sequence diagrams of some of the main functionalities of FarmBot System.

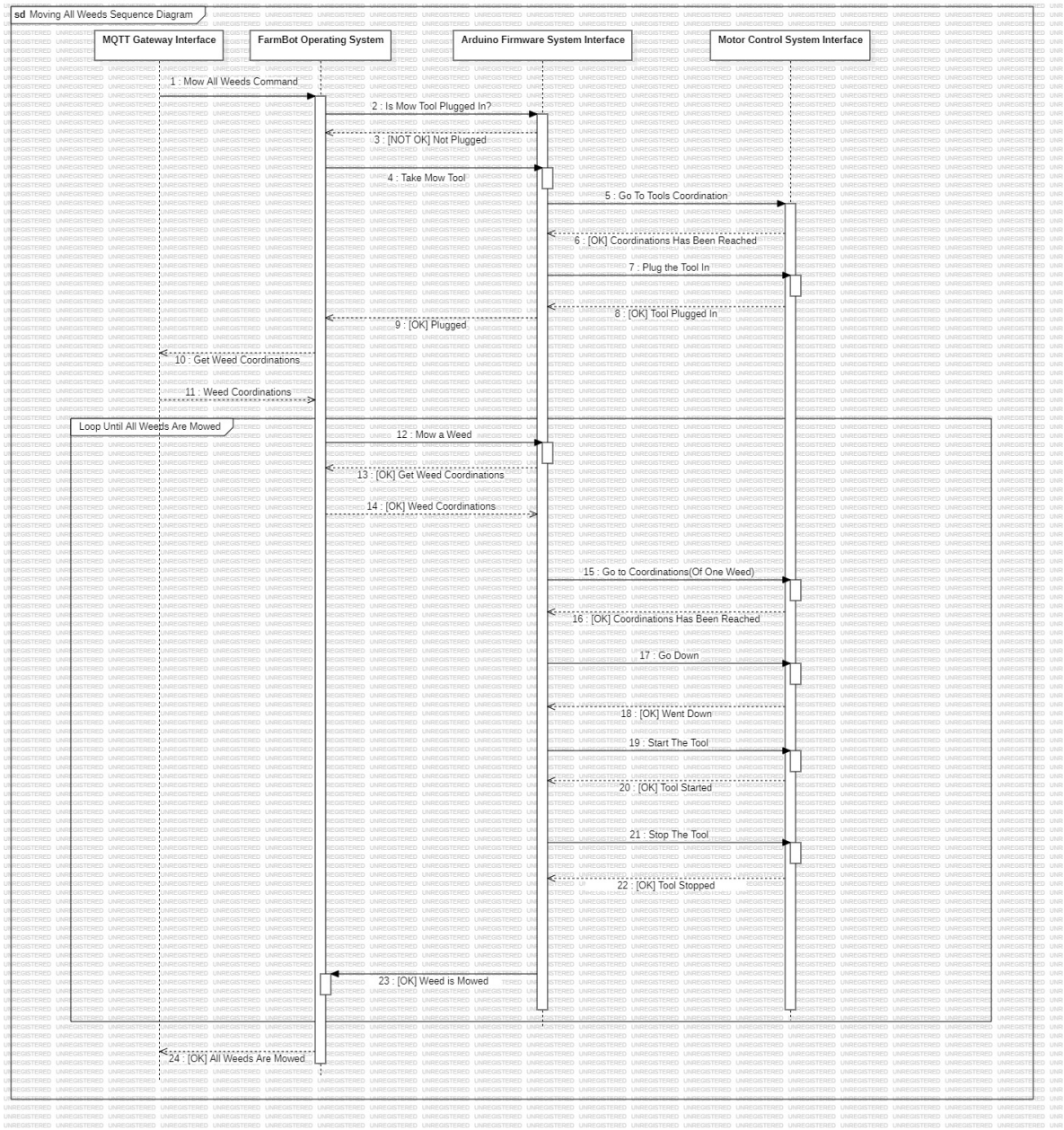

Figure 4.7: Mow All Weeds - Sequence Diagram

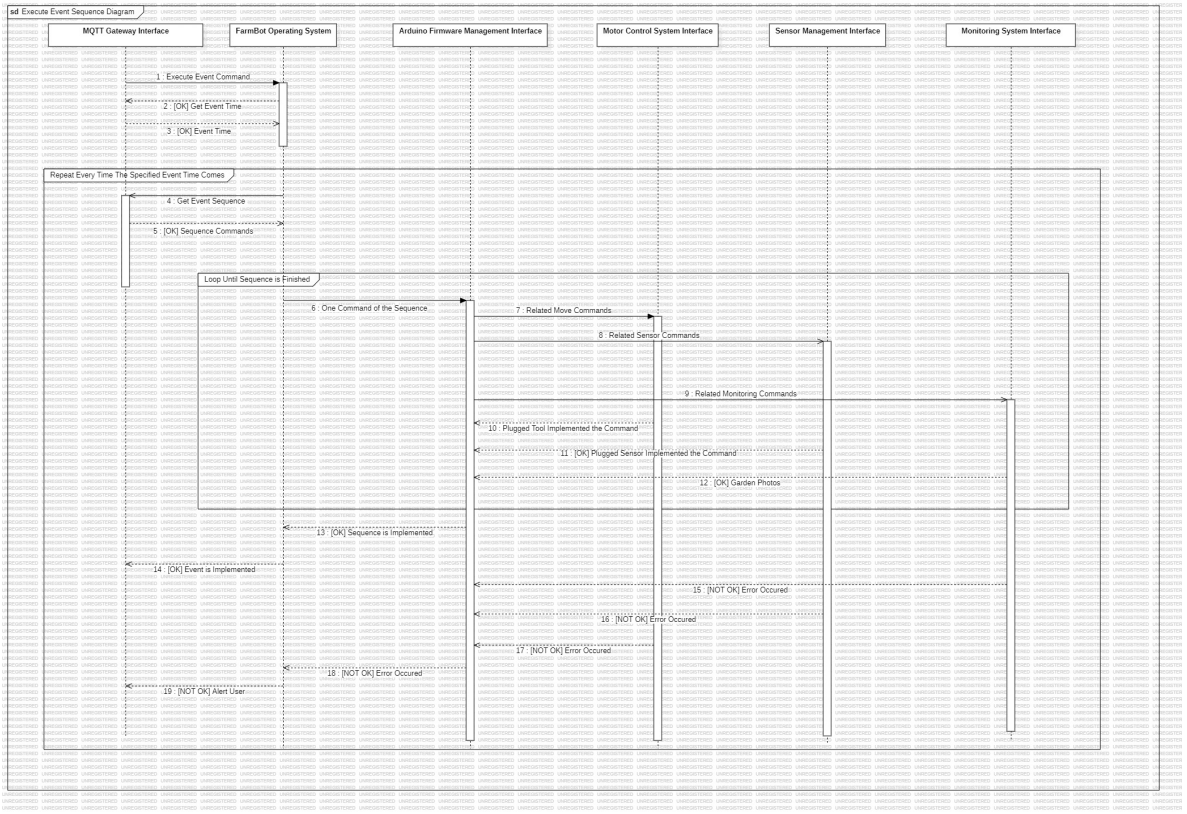

Figure 4.8: Implement Event - Sequence Diagram

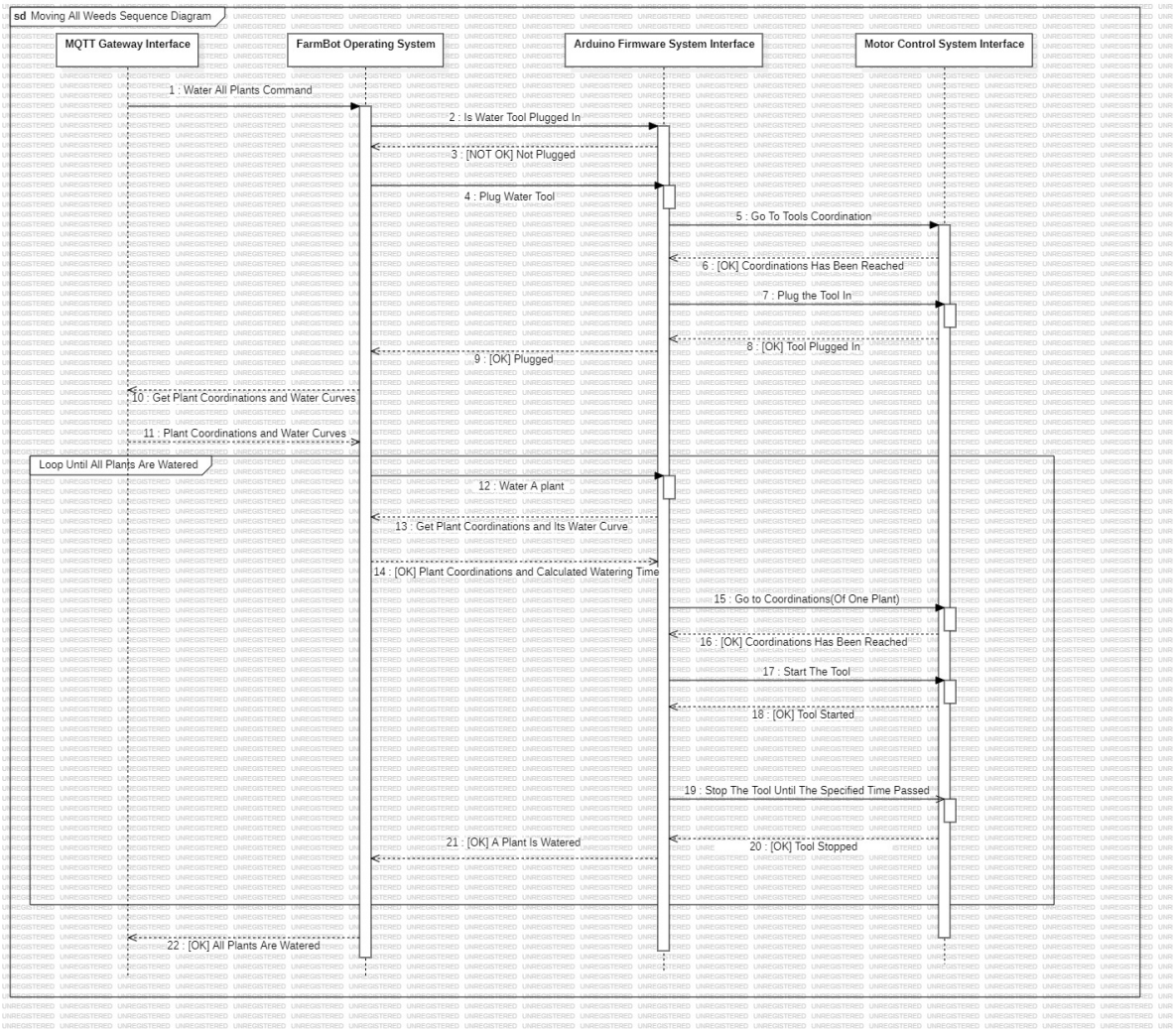

Figure 4.9: Water All Plants - Sequence Diagram

4.3 Information View

The section provides a depiction of the data entities, their attributes, relationships, and the flow of information within a system. It also provides a depiction of operations that operates on data entities. It includes the structure and organization of data, offering insights into how information is stored, accessed, and manipulated.

4.3.1 Stakeholders' uses of this view

- A user who hasn't purchased yet can check this view and can get an idea of what one can do with FarmBot.
- Developers can check that to understand general data structures and operations of the FarmBot.
- Developers can use this view to troubleshoot user-reported issues related to data manipulation operations.
- An experienced user in computer sciences can check this to understand data classes to use them in a data analysis.
- A botany student or scientist may need to get the information about the garden that they conduct their experiment on. Looking this view facilitates the process. Furthermore, they can use this information to collaborate with other researchers, share findings.

4.3.2 Database Class Diagram

- Points represents coordinates in a garden. It also holds relevant information of that coordinate such as a plant, a weed or a point (to use in a sequence).
- Device represents the whole FarmBot device.
- All hardware, such as tools and sensors, have to be added to use in sequences.

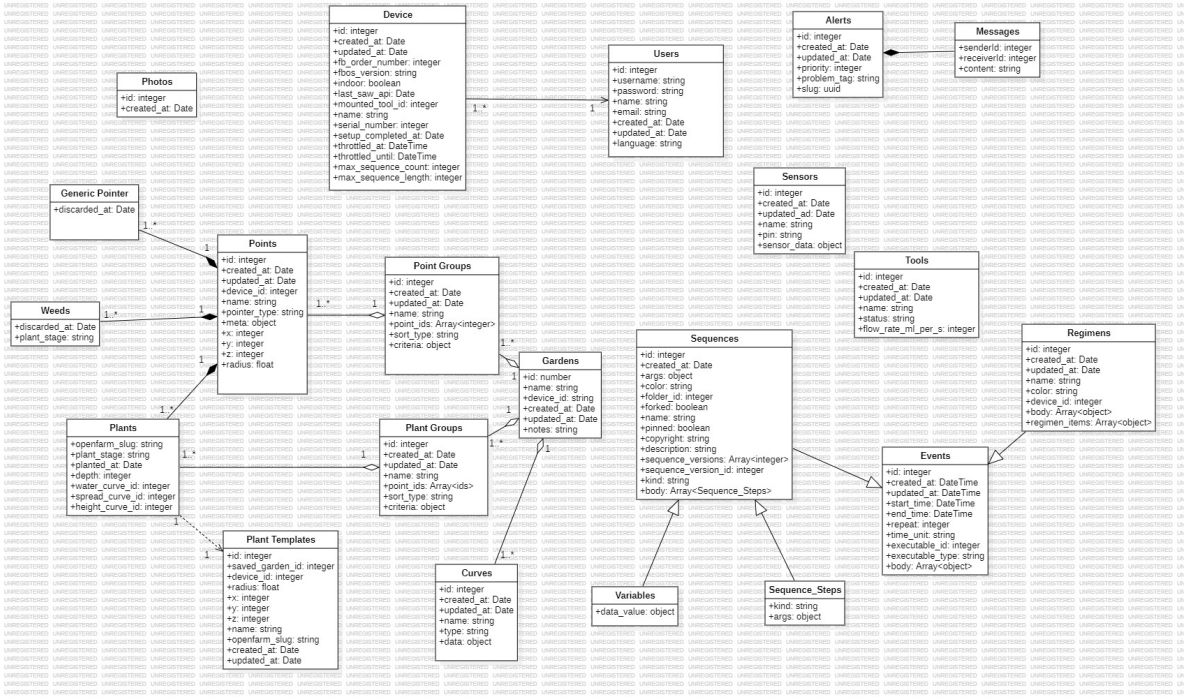

Figure 4.10: FarmBot Database Classes Diagram

- Curves are generic templates to use in plant care. They can store water curve that determines the amount of water to be given, can store spread curve that limits the amount of spread a plant or a weed makes, can store height curve that limits the height a plant makes.

- Plants, weeds and generic pointers are stored as Points, they have additional information in their own classes.

- Points can be included in Point Groups and Plants can be included in Plant Groups in order to implement commands on all of the points or plants in a group.

- Sequences includes step by step operations to be done by FarmBot Hardware.

- Messages includes jobs that are done by FarmBot, logs that are resulted from jobs, and messages a user has gotten.

- Regimens are plant care recipes for FarmBot to follow.

- Events are used to schedule FarmBot actions. It includes sequences and their implementation times.
- Variables are user defined variables to reuse in command codes.
- Each sequence step is a simple command that can be followed by FarmBot.

4.3.3 Operations on Data

4.3.3.1 Operations and Descriptions

Operation	**Description**
userSignIn	This operation allows users to sign in to the FarmBot system, creating a new user account if one does not already exist.
userLogin	This operation allows registered users to log in to the FarmBot system, granting access to their personalized settings and data.
registerPurchase	This operation registers a purchase made by a user, associating the order with their account by using order number of the purchase.
connectFarmBot	This operation establishes a connection between the FarmBot device and the user's account, enabling remote control and monitoring of the device.
addTool	This operation adds a new tool to the FarmBot system, allowing it to be used in automated gardening tasks.
addSensor	This operation adds a new sensor to the FarmBot system, expanding its capabilities for monitoring environmental conditions.
addPlant	This operation creates a new plant to the garden, allowing users to care it and track its growth from the FarmBot system.

movePlant	This operation moves a plant to a new location within the garden, updating its coordinates and associated data accordingly.
removePlant	This operation removes a plant from the garden, deleting its data and freeing up coordinates for other plants.
addWeed	This operation adds a new weed to the garden, allowing users to track its presence and take appropriate action to remove it or care it.
moveWeed	This operation moves a weed to a new location within the garden, updating its coordinates and associated data accordingly.
removeWeed	This operation removes a weed from the garden, deleting its data and freeing up coordinates for other plants.
addPoint	This operation adds a new point to the garden, allowing users to mark specific locations for various purposes such as planting, maintenance or meisure tasks.
movePoint	This operation moves a point to a new location within the garden, updating its coordinates and associated data accordingly.
removePoint	This operation removes a point from the garden, deleting its data and freeing up coordinates for other purposes.
createPointGroup	This operation creates a new group of points, allowing users to organize and manage them collectively for specific tasks or analyses.
updatePointGroup	This operation updates the properties of an existing point group, such as its name, description, or membership.
deletePointGroup	This operation deletes a point group from the system, associated points continue to be stored after this operation.

createPlantGroup	This operation creates a new group of plants, allowing users to organize and manage them collectively for specific care or analysis purposes.
updatePlantGroup	This operation updates the properties of an existing plant group, such as its name, description, or membership.
deletePlantGroup	This operation deletes a plant group from the system, associated points continue to be stored after this operation.
createWeedGroup	This operation creates a new group of weeds as a point group, allowing users to organize and manage them collectively for specific control or analysis purposes.
updateWeedGroup	This operation updates the properties of an existing point group related to weeds, such as its name, description, or membership.
deleteWeedGroup	This operation deletes a point group related to weeds from the system, associated points continue to be stored after this operation.
createSequence	This operation creates a new sequence of actions to be executed by the FarmBot hardware, defining the steps and their order. A sequence includes events, regimens, variables and primitive sequence steps.
executeSequence	This operation executes a previously created sequence, triggering the FarmBot hardware to perform the defined actions in the specified order.
createRegimen	This operation creates a new regimen, which is a predefined set of sequences and their scheduling for automated plant care.

scheduleEvent	This operation schedules an event, such as the execution of a regimen or sequence, based on predefined criteria and timing parameters.
addWebcamFeed	This operation adds a webcam feed to the system, allowing users to monitor the garden remotely through live video streaming.
setHome	This operation sets the home position for the FarmBot device within the garden, defining a reference point for navigation and operation.
getJobs	This operation retrieves the list of jobs that recently done by FarmBot device, providing information on the current state of operations.
getLogs	This operation retrieves logs and records of past activities and events performed by the FarmBot device, allowing users to review historical data.
saveGarden	This operation saves the current state of the garden, including all data related to points, plant groups, curves, and other parameters, for future reference and restoration.
deleteGarden	This operation deletes the entire garden and its associated data, removing all records of points, plants, and configurations.
applyGarden	This operation applies a saved garden configuration to the current system, restoring previously saved data and settings.
addCurve	This operation adds a new curve to the system, which can define various parameters such as water usage, plant spread, or height, affecting the behavior of the FarmBot device during plant care.

updateCurve	This operation updates an existing curve in the system, allowing users to modify parameters and settings to adjust plant care behavior.
deleteCurve	This operation deletes a curve from the system, removing its influence on plant care behaviors and configurations.
detectWeeds	This operation initiates the detection of weeds in the garden using photos, enabling the FarmBot application to identify and manage unwanted plant growth.
filterPhotos	This operation applies filters and criteria to photos captured by the FarmBot device, allowing users to refine and select images for analysis or documentation purposes.
calibrateCamera	This operation calibrates the camera system of the FarmBot device, ensuring accurate image capture and plant monitoring.

Table 4.1: Operations On Data

4.3.3.2 CRUD Effects of Operations

Operation	**Affected Data Entities**
userSignIn	Create: User Read: User Update: - Delete: -
userLogin	Create: - Read: User Update: - Delete: -

registerPurchase	Create: - Read: Order number Update: User Delete: -
connectFarmBot	Create: Device Read: - Update: - Delete: -
addTool	Create: Tool Read: - Update: - Delete: -
addSensor	Create: Sensor Read: - Update: - Delete: -
addPlant	Create: Plant Read: - Update: Garden Delete: -
movePlant	Create: - Read: Points Update: Point Delete: -
removePlant	Create: - Read: Points Update: Garden Delete: Point

addWeed	Create: Point Read: - Update: Garden Delete: -
moveWeed	Create: - Read: Points Update: Point Delete: -
removeWeed	Create: - Read: Points Update: Garden Delete: Point
addPoint	Create: Point Read: - Update: Garden Delete: -
movePoint	Create: - Read: Points Update: Point Delete: -
removePoint	Create: - Read: Points Update: Garden Delete: Point
createPointGroup	Create: Plant Group Read: Generic Pointers, Points Update: Garden Delete: -

updatePointGroup	Create: - Read: Generic Pointers, Points, Point Groups Update: Point Group Delete: -
deletePointGroup	Create: - Read: - Update: Garden Delete: Point Group
createPlantGroup	Create: Point Group Read: Plants, Points Update: Garden Delete: -
updatePlantGroup	Create: - Read: Plants, Plant Groups Update: Plant Group Delete: -
deletePlantGroup	Create: - Read: - Update: Garden Delete: Plant Group
createWeedGroup	Create: Point Group Read: Weeds, Points Update: Garden Delete: -
updateWeedGroup	Create: - Read: Weeds, Points, Point Groups Update: Point Group Delete: -

deleteWeedGroup	Create: - Read: - Update: Garden Delete: Point Group
createSequence	Create: Sequence Read: Event, Regimen, Sequence Step, Variable Update: Variable Delete: -
executeSequence	Create: - Read: Event, Regimen, Sequence Step, Variable Update: Points, Point Groups, Messages Delete: Points, Point Groups
createRegimen	Create: Regimen Read: Variables, Sequences Update: - Delete: -
scheduleEvent	Create: Event Read: Regimens, Sequences, Variables Update: - Delete: -
addWebcamFeed	Create: Tool Read: - Update: - Delete: -
setHome	Create: Point Read: Gardens Update: - Delete: -

getJobs	Create: - Read: Device, Messages Update: - Delete: -
getLogs	Create: - Read: Device, Messages Update: - Delete: -
saveGarden	Create: Garden Read: Point Groups, Plant Groups, Curves, Points Update: - Delete: -
deleteGarden	Create: - Read: - Update: - Delete: Garden
applyGarden	Create: - Read: Gardens Update: Garden Delete: -
addCurve	Create: Curve Read: - Update: - Delete: -
updateCurve	Create: - Read: - Update: Curve Delete: -

deleteCurve	Create: - Read: - Update: - Delete: Curve
detectWeeds	Create: - Read: Photos Update: Point, Point Groups Delete: Points
filterPhotos	Create: - Read: Date, Time Update: Photos Delete: -
calibrateCamera	Create: - Read: Sequences Update: Messages Delete: -

Table 4.2: CRUD Effects of Operations

4.4 Deployment View

4.4.1 Stakeholders' uses of this view

The Deployment View offers stakeholders a comprehensive insight into the practical implementation of FarmBot Express, detailing how the system is installed, configured, and managed within operational environments.

- **Engineering Students:** Engineering students would be interested in the deployment view to understand the technical setup and configuration of FarmBot Express. This would include aspects such as the deployment of sensors, actuators,

and other hardware components, as well as the deployment of software systems such as the FarmBot control software and any additional coding frameworks utilized for customization or integration.

- **Scientists:** Researchers would also find the deployment view valuable as they assess the practical implementation of FarmBot Express for their experiments. They would be interested in how the device is deployed in different experimental setups, including its integration with other scientific instruments and how it is configured for specific experimental conditions.

- **Educators:** Teachers incorporating FarmBot Express into educational settings would benefit from understanding the deployment view to effectively set up and manage the device within the classroom or learning environment. This would involve considerations such as installation, calibration, and maintenance procedures, as well as any safety protocols or guidelines for student use.

4.4.2 Deployment Diagram

The FarmBot system's deployment view involves various components deployed across different layers of infrastructure, utilizing protocols tailored to their functionalities. The web application, likely deployed on web servers, enables user interaction via HTTP/HTTPS protocols, ensuring accessibility and ease of use. FarmBot OS, deployed on hardware, potentially utilizes MQTT for real-time communication, facilitating seamless control and monitoring of hardware components. The database management system, deployed on servers or cloud platforms, supports efficient data storage and retrieval, contributing to overall system performance. The Lua compiler, integrated within the FarmBot OS environment, enhances flexibility and customization options through script execution. External components, such as the OpenFarm Database, are accessed via standard APIs or web services protocols, enabling seamless integration and access to additional plant care information.

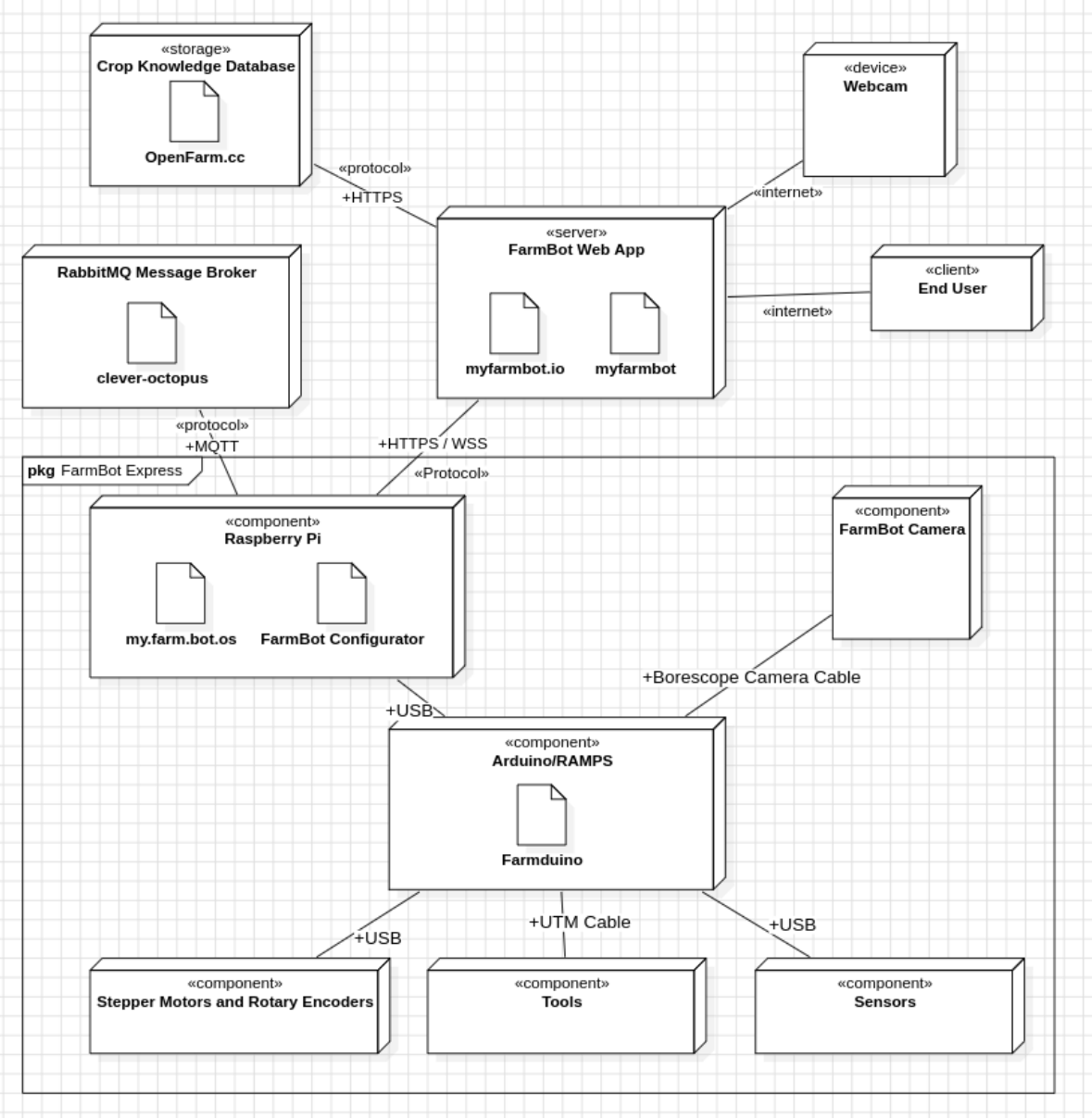

Figure 4.11: Deployment Diagram

4.5 Design Rationale

Design Rational of Context View

In context view, FarmBot utilises various external entities to reduce manual interventions and ease remote usage. Therefore, it utilises three external entities besides the web application which is the primary user interface to interact with the device. The

database management stores all data related to the FarmBot's operations. It allows to enable the system to quickly access the necessary information to perform tasks and generate reports. The Lua Compiler allows the system to gain flexibility, enabling users to extend the capabilities of FarmBot without modifying the core system code. Finally, the webcam provides real-time visual monitoring, allowing users to see the status of their plants and detect any issues such as pets or diseases.

Design Rational of Functional View

In terms of functional view, the FarmBot OS acts as the central controller, managing data from sensors and sending commands to mechanical components like stepper motors via the Arduino - RAMPS interface. The web application serves as the primary interface for users to remotely monitor and control the FarmBot, providing real-time visualization of data and status. The camera on the FarmBot and an external webcam enable comprehensive visual monitoring of the garden. Sensor data and gardening task execution are managed by the FarmBot OS, ensuring precise control and automation. The Lua compiler allows for customizable scripting to extend functionalities, while the OpenFarm database integrates expert plant care information to optimize gardening practices. This design ensures coordinated monitoring, precise control, and flexible customization for efficient automated gardening.

Design Rational of Information View

OpenFarm.cc's information view design aims to ensure accessibility, collaboration, and comprehensiveness in agricultural knowledge sharing. The platform offers a user-friendly interface with multimedia elements like images and videos to engage users and enhance understanding. Through a crowdsourced approach, users contribute their expertise, fostering collaboration and keeping the database relevant and up-to-date. The platform covers a wide range of crops and cultivation practices, catering to users with varying interests and expertise levels. Search functionality enables efficient navigation, while a structured format organizes information logically for easy comprehension. Overall, OpenFarm.cc's design prioritizes accessibility, collaboration, comprehensive-

ness, searchability, and structured presentation to provide a valuable resource for agricultural knowledge.

Design Rational of Deployment View

The design rationale for the deployment view of the FarmBot is to ensure scalability and flexibility in managing the system's diverse components and functionalities. By deploying the web application on web servers and using standard protocols like HTTP/HTTPS, the system provides users with easy and remote access from various devices. The FarmBot OS, deployed on the hardware and using MQTT for communication, allows for real-time control and monitoring, ensuring efficient operation of the physical components. The DBMS, hosted on servers or cloud platforms, ensures efficient and secure data storage and retrieval. Integrating the Lua compiler within the FarmBot OS enables customization and automation through script execution. External resources like the OpenFarm Database are accessed via APIs, facilitating seamless integration and enhancing the system's functionality with additional plant care data. This design approach not only supports scalable and flexible deployment but also ensures reliable and accessible operations, accommodating the needs of various stakeholders effectively.

5. Architectural Views for Your Suggestions to Improve the Existing System

5.1 Context View

5.1.1 Stakeholders' uses of this view

The Context View provides stakeholders with a holistic understanding of how FarmBot Express fits into various contexts and fulfils different needs.

- **Botanic Students:** To comprehend how FarmBot Express fits into their research on plant biology and environmental interactions, they would make use of the Context View. They concentrate on elements associated with tracking plant growth patterns and conducting environmental experimentation.
- **Engineering Students:** The Context View would be examined by engineering students to see how FarmBot Express can be used as a useful tool for learning robotics and coding. Technical details like the sensors, hardware, and programming interface may pique their interest.
- **Scientists:** Researchers would examine the Context View to evaluate the accuracy and control that FarmBot Express provides for carrying out plant experiments. They concentrate on features like integration with other scientific

instruments, adaptable experimental setups, and data logging.

- **Garden Enthusiasts:** Context View is what gardeners would use to investigate how FarmBot Express can improve gardening practices. Features like crop optimization strategies, soil monitoring, and automated irrigation are of interest to them.

- **Educators:** Teachers can use the Context View to see how FarmBot Express fits into educational environments. They search for characteristics that support experiential learning and correspond with engineering and agriculture curriculum objectives.

- **Home Gardeners:** Home gardeners would refer to the Context View to evaluate how FarmBot Express can simplify and automate their gardening tasks. They focus on features such as remote monitoring, customizable planting schedules, and compatibility with different garden layouts.

5.1.2 Context Diagram

In terms of context view, the integration of the Suggestion and Feedback System into FarmBot enhances the platform by acting as a virtual assistant and data analyzer. It leverages data analytics and machine learning models to provide personalized guidance and insights to users. This system analyzes current garden and environmental data, generating practical advice and predictions that improve gardening efficiency and accelerate the user's learning process. By interacting with both the FarmBot software and the OpenFarm.cc database, it enriches the user experience with tailored recommendations and feedback, leading to better gardening outcomes. The Warning System enhances FarmBot by providing robust security and real-time monitoring, protecting the farm from animal intrusions and theft. It integrates seamlessly with FarmBot's infrastructure, automating responses and notifying users instantly via a web application. The system includes sensors, sirens, and strobe lights for immediate deterrence and alerting, along with configurable settings to match specific security needs. By enabling

remote management and providing event logs, it ensures efficient operation and peace of mind for farmers, allowing them to focus on other tasks while maintaining a secure environment.

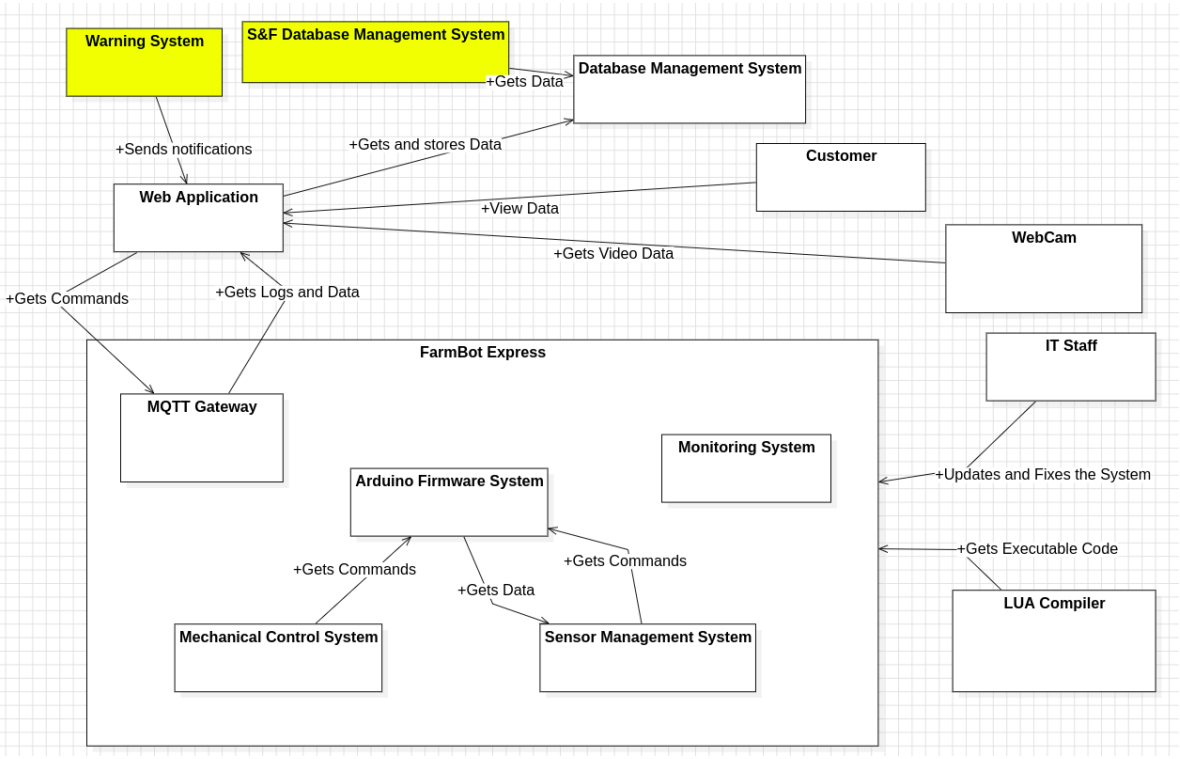

Figure 5.1: System Context Diagram with Suggestions

5.1.3 External Interfaces

- **SF Database Management**

  The SF Database Management interface ensures seamless interaction between the FarmBot Suggestion and Feedback System and the OpenFarm.cc FarmBot DBMS, thereby enhancing the overall functionality and efficiency of the system.

  - ***plantData*:** Contains detailed information about plant species, including optimal growing conditions, care instructions, and common issues.
  - ***userGardenData*:** Contains user-specific garden data stored in the OpenFarm.cc database, such as plant growth stages, environmental conditions,

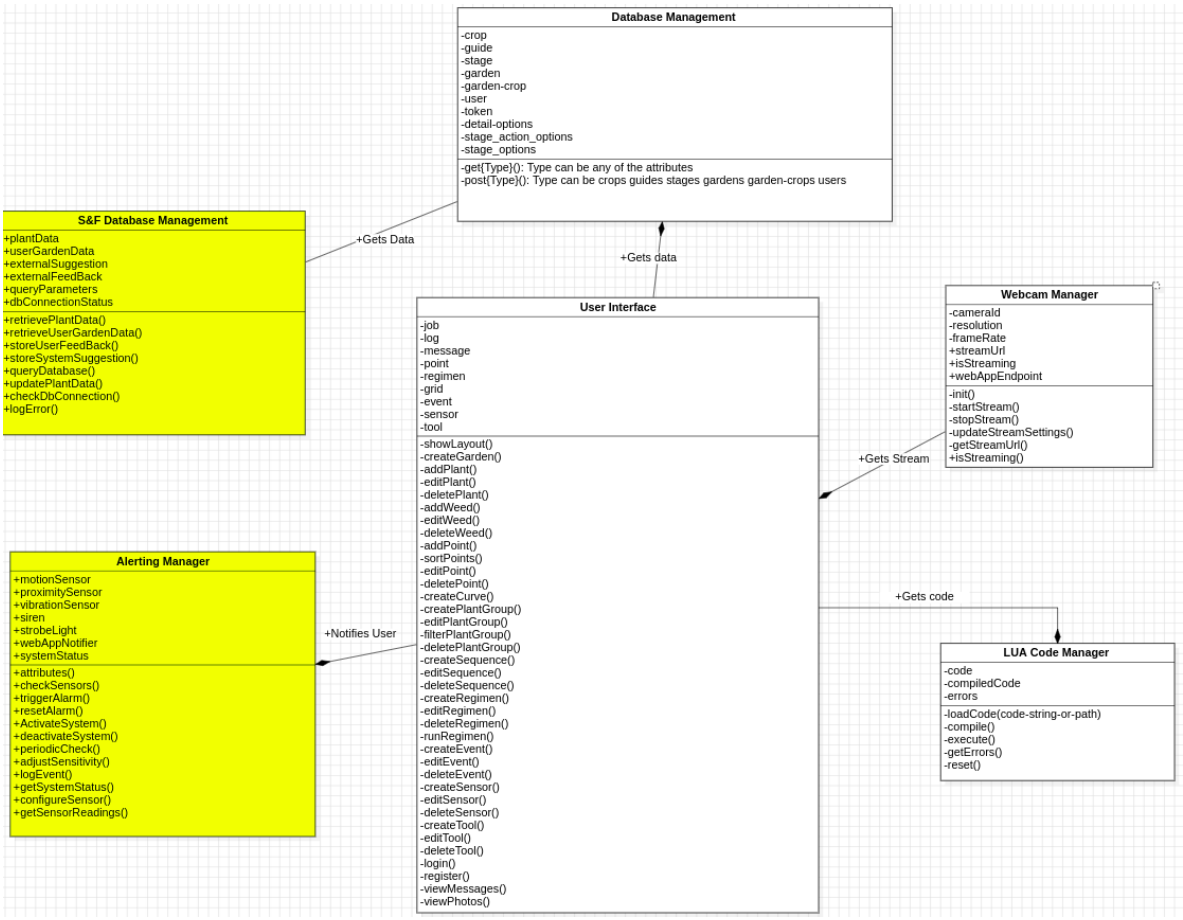

Figure 5.2: Suggested Class Diagram For External Interfaces

and historical data.

- ***externalSuggestion*:** Represents suggestions retrieved from the external database, which can be used to supplement the system-generated suggestions.
- ***externalFeedback*:** Represents feedback retrieved from the external database, which can be used to inform system-generated feedback
- ***queryParameters*:** Parameters used for querying the external database to retrieve relevant plant and garden data.
- ***dbConnectionStatus*:** Indicates whether the connection to the OpenFarm.cc database is active.
- ***retrievePlantData(plantId)*:** Fetches data about a specific plant species

from the OpenFarm.cc database.

- ***retrieveUserGardenData(userId)*:** Fetches garden data specific to a user from the OpenFarm.cc database.
- ***storeUserFeedback(userId, feedback)*:** Stores user-generated feedback into the OpenFarm.cc database.
- ***storeSystemSuggestion(userId, suggestion)*:** Stores system-generated suggestions into the OpenFarm.cc database.
- ***queryDatabase(queryParameters)*:** Performs a query on the OpenFarm.cc database based on specified parameters to retrieve relevant data.
- ***updatePlantData(plantId, plantData)*:** Updates plant-specific data in the OpenFarm.cc database.
- ***checkDbConnection()*:** Checks the connection status to the OpenFarm.cc database.
- ***logError(errorMessage)*:** Logs an error message related to database interactions.

- **Alerting Manager**

  This interface allows for integrating sensor checks and handling notifications.

  - ***motionSensor*:** Represents the motion sensor component.
  - ***proximitySensor*:** Represents the proximity sensor component.
  - ***vibrationSensor*:** Represents the vibration sensor component.
  - ***siren*:** Represents the siren component.
  - ***strobeLight*:** Represents the strobe light component.
  - ***webAppNotifier*:** Manages notifications to the web application
  - ***systemStatus*:** Indicates the current status of the system (e.g., "Idle", "Active", "Triggered").
  - ***attributes()*:** Returns the current status of all components and the system.

- ***checkSensors()*:** Checks all sensors for any detection.
- ***triggerAlarm()*:** Activates the siren and strobe light, and sends a notification.
- ***resetAlarm()*:** Deactivates the siren and strobe light, resetting the system status.
- ***activateSystem()*:** Sets the system status to "Active" and sends a notification.
- ***deactivateSystem()*:** Resets the alarm and sets the system status to "Idle", sending a notification.
- ***periodicCheck()*:** Periodically checks sensors if the system is active
- ***adjustSensitivity()*:** Adjusts the sensitivity level of a specified sensor.
- ***logEvent*:** Logs an event with the current timestamp.
- ***getSystemStatus()*:** Retrieves the current status and log of the system
- ***configureSensor()*:** Configures a specified sensor with provided settings.
- ***getSensorReadings()*:** Retrieves current readings from all sensors.

5.1.4 Interaction scenarios

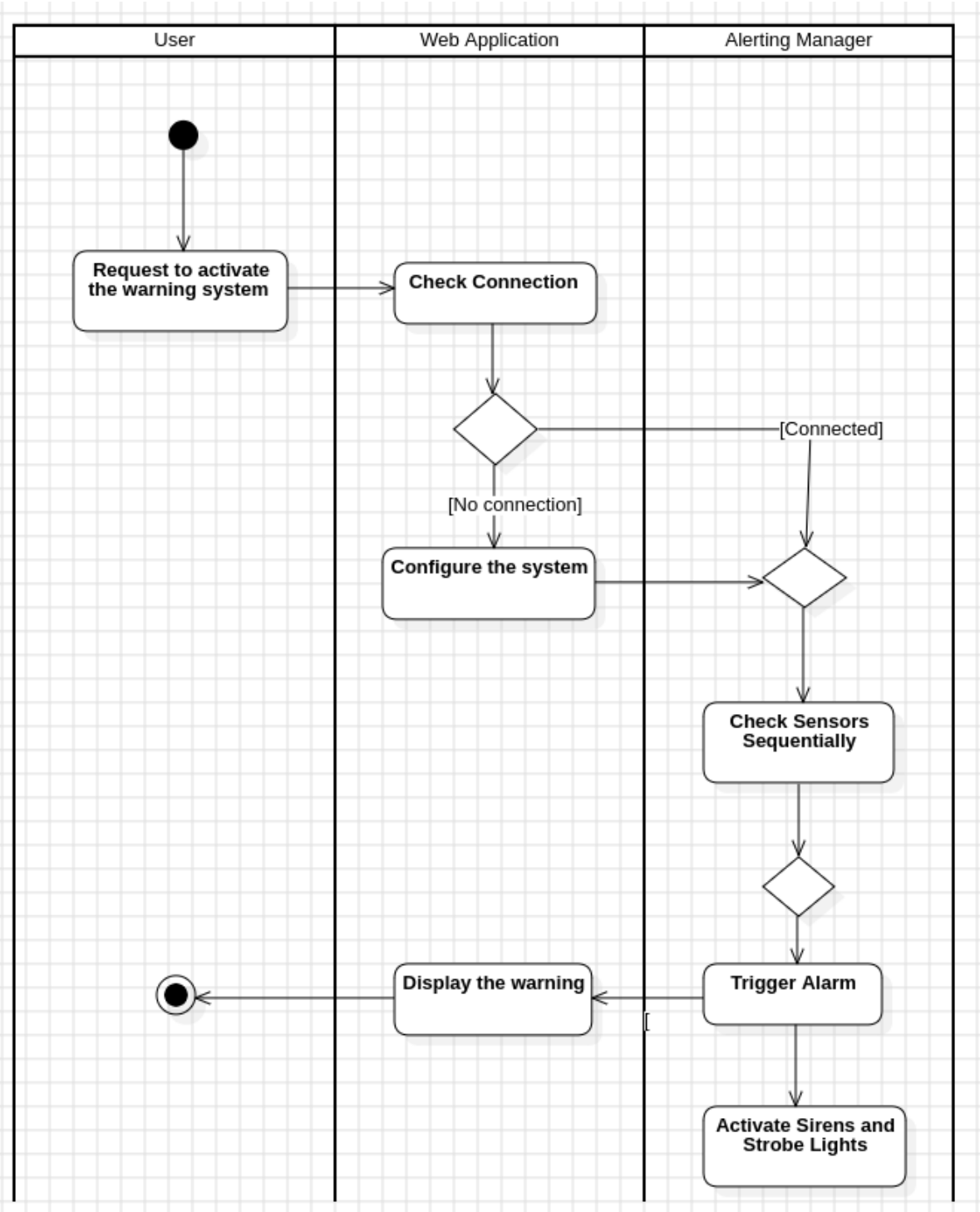

Figure 5.3: Alerting Mechanism Activity Diagram

5.2 Functional View

This section extends functional view document of FarmBot with the suggestions made by us.

5.2.1 Stakeholders' uses of this view

The integration of a Suggestion and Feedback System into the FarmBot Software Architecture enhances the educational experience and gardening efficiency by providing personalized guidance and insights derived from data analytics and machine learning models. This system acts as a virtual assistant and data analyzer, offering practical advice and predictions to users based on the current state of the garden and environmental conditions. By taking advantage data-driven suggestions and feedback, users can accelerate their learning process and their garden's efficiency.

5.2.2 Component Diagram

- **Suggestion and Feedback System:** The Suggestion and Feedback System is a sophisticated feature designed to enhance the operational efficiency and user experience of the FarmBot. This system provides intelligent recommendations and feedback based on the analysis of garden data and user interactions. It generates various attributes such as plant-specific suggestions, feedback on implemented actions by considering garden data, environmental conditions, and user requests. By generating actionable recommendations for future, and feedback for previous actions, this system helps users to optimize their gardening activities, protect plant health, and improve learning experience. Key operations include generating suggestions and events for a subject plant or a written request, providing feedback for previous actions of the users.

- **Backup Storage System:** The Backup Storage System is a component designed to ensure the continuous operation of the FarmBot even in the absence of

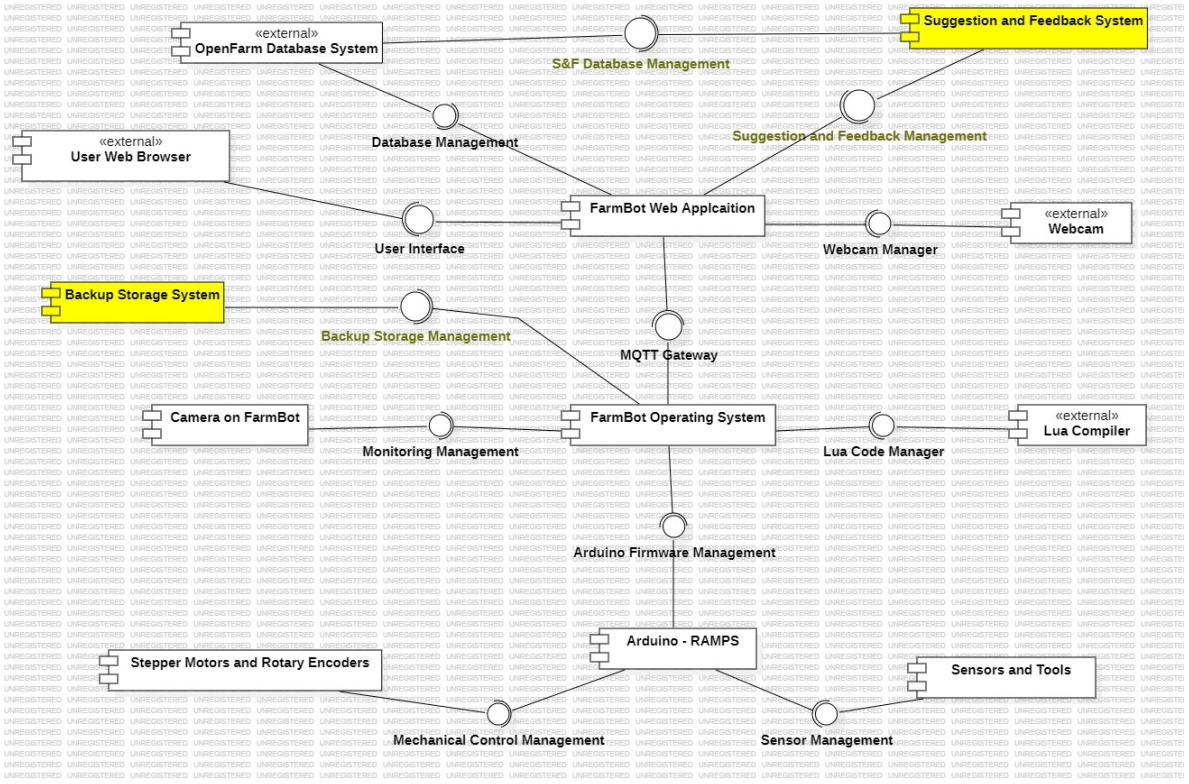

Figure 5.4: Component Diagram With Suggestion

an internet connection. This system is connected to the FarmBot via a cable and serves as a secure repository for critical data, including system configurations, scheduled events, sequences, water curves and other important information. By storing this data locally, the Backup Storage System provides a reliable mechanism that allows the FarmBot to maintain its functionalities and access necessary information during internet outages. Key features of this system include data backup initiation, restoration, and data retrieval.

5.2.3 Internal Interfaces

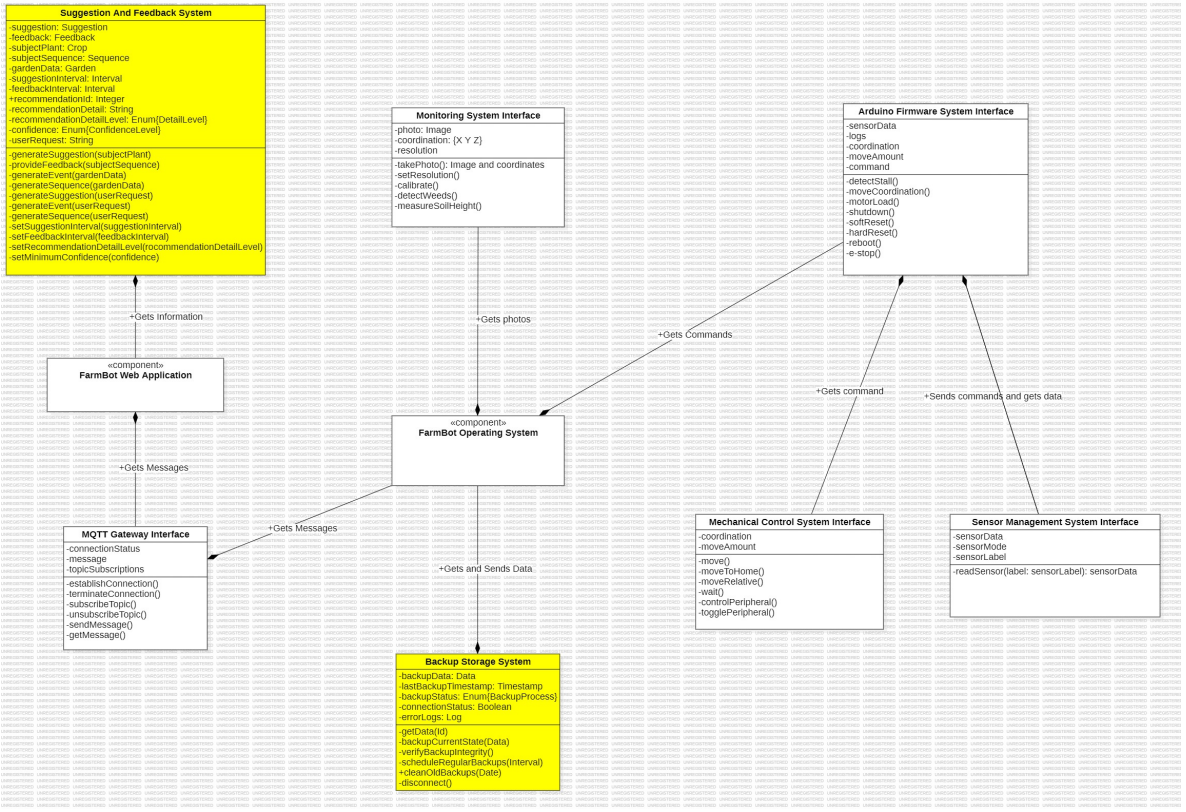

Figure 5.5: Internal Interfaces With Suggestion

- **Suggestion and Feedback Management**
  - **Attributes:**
    - **suggestion**: Represents a suggested action or recommendation provided by the system.
    - **feedback**: Represents a feedback on a user action such as an implemented sequence or a scheduled event.
    - **subjectPlant**: Refers to the specific crop or plant species related to a suggestion or feedback.
    - **subjectSequence**: Represents a sequence of actions or tasks related to a gardening activity related to a suggestion or feedback.

    - **gardenData**: Contains information about the current state of the garden, including environmental conditions, plant health, and growth progress.
    - **suggestionInterval**: Defines the frequency at which suggestions are generated and presented to the user.
    - **feedbackInterval**: Specifies the interval at which feedback are generated and presented to the user.
    - **recommendationId**: Unique identifier for a recommendation or suggested action.
    - **recommendationDetail**: Additional details or information accompanying a recommendation, such as explanations or rationale.
    - **recommendationDetailLevel**: Enumerates the level of detail provided in recommendations, allowing users to adjust the depth of information presented.
    - **confidence**: Enumerates the confidence level associated with a recommendation, indicating the system's certainty in its recommendation.
    - **userRequest**: Stores user-initiated requests or queries, which may trigger the generation of suggestions or events.
- **Operations:**
    - **generateSuggestion(subjectPlant)**: Generates a suggestion tailored to a specific plant or crop, based on current garden data and system parameters.
    - **provideFeedback(subjectSequence)**: Generates a feedback tailored to a specific sequence implementation or scheduled event, based on current garden data and system parameters.
    - **generateEvent(gardenData)**: Generates the most urgent and important event based on the analysis of the current state of the garden and predefined gardening objectives.
    - **generateSequence(gardenData)**: Creates the most urgent and important sequence based on the analysis of the current state of the garden

and predefined gardening objectives.

  - **generateSuggestion(userRequest)**: Generates a suggestion in response to a user-initiated request or query, providing actionable recommendations.
  - **generateEvent(userRequest)**: Generates a ready-to-use event in response to a user-initiated request or query.
  - **generateSequence(userRequest)**: Creates a ready-to-use sequence based on a user-initiated request or query.
  - **setSuggestionInterval(suggestionInterval)**: Allows users to adjust the frequency at which suggestions are generated and presented within the system.
  - **setFeedbackInterval(feedbackInterval)**: Allows users to adjust the frequency at which feedback are generated and presented within the system.
  - **setRecommendationDetailLevel(recommendationDetailLevel)**: Adjusts the level of detail provided in recommendations, allowing users to customize their experience based on preference or objectives.
  - **setMinimumConfidence(confidence)**: Sets a minimum confidence threshold for recommendations, ensuring that only suggestions meeting the specified confidence level are presented to users.

- **Backup Storage Management**
  - **Attributes:**
    - **backupData**: Contains the essential data required for the FarmBot's operations.
    - **lastBackupTimestamp**: Records the date and time when the last backup was performed.
    - **backupStatus**: Reflects the current status of the backup process.

  - **connectionStatus**: Indicates whether the backup system is currently connected to the FarmBot.
  - **errorLogs**: Contains logs of any errors or issues encountered during the backup process.

- **Operations:**
  - **getData(Id)**: Retrieves data from the backup storage system using the provided identifier (Id). This operation allows the FarmBot to access important information such as scheduled events, sequences, or water curves directly from the backup system. By using this function, the FarmBot can continue to operate without an internet connection.
  - **backupData(Data)**: Backups essential data such as scheduled events, sequences, water curves, from the FarmBot to the backup storage system.
  - **verifyBackupIntegrity()**: Verifies the integrity of the backed-up data when internet connection is established with database.
  - **scheduleRegularBackups(Interval)**: Sets up a regular backup schedule, specifying the interval at which backups should be performed automatically.
  - **cleanOldBackups(Date)**: Deletes backups older than a specified date, ensuring that the storage does not get filled with outdated data.
  - **disconnect()**: Safely disconnects the backup storage system from the FarmBot.

5.2.4 Interaction Patterns

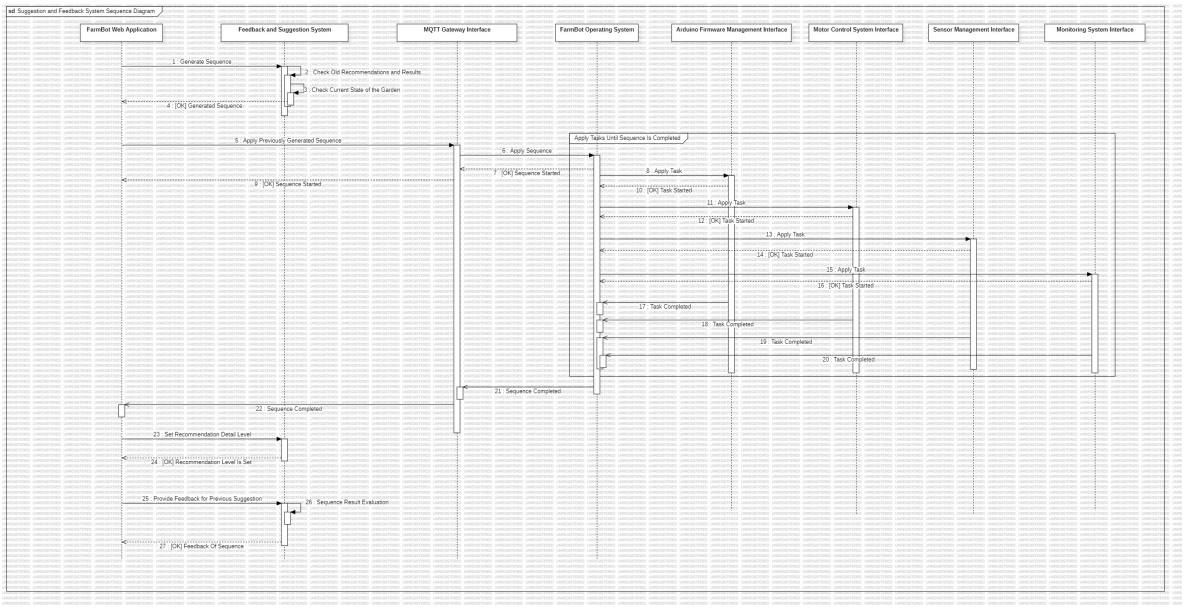

Figure 5.6: Sequence Diagram For Suggestion

5.3 Information View

This section extends the information view document of FarmBot with the suggestions made by us.

5.3.1 Stakeholders' uses of this view

A potential user can explore this view to understand how the suggestion and feedback system operates within FarmBot, gaining insights into the capabilities and benefits of personalized guidance and feedback.

5.3.2 Database Class Diagram

- Recommendation is a generalization for feedback and suggestion. It includes generic information of both.

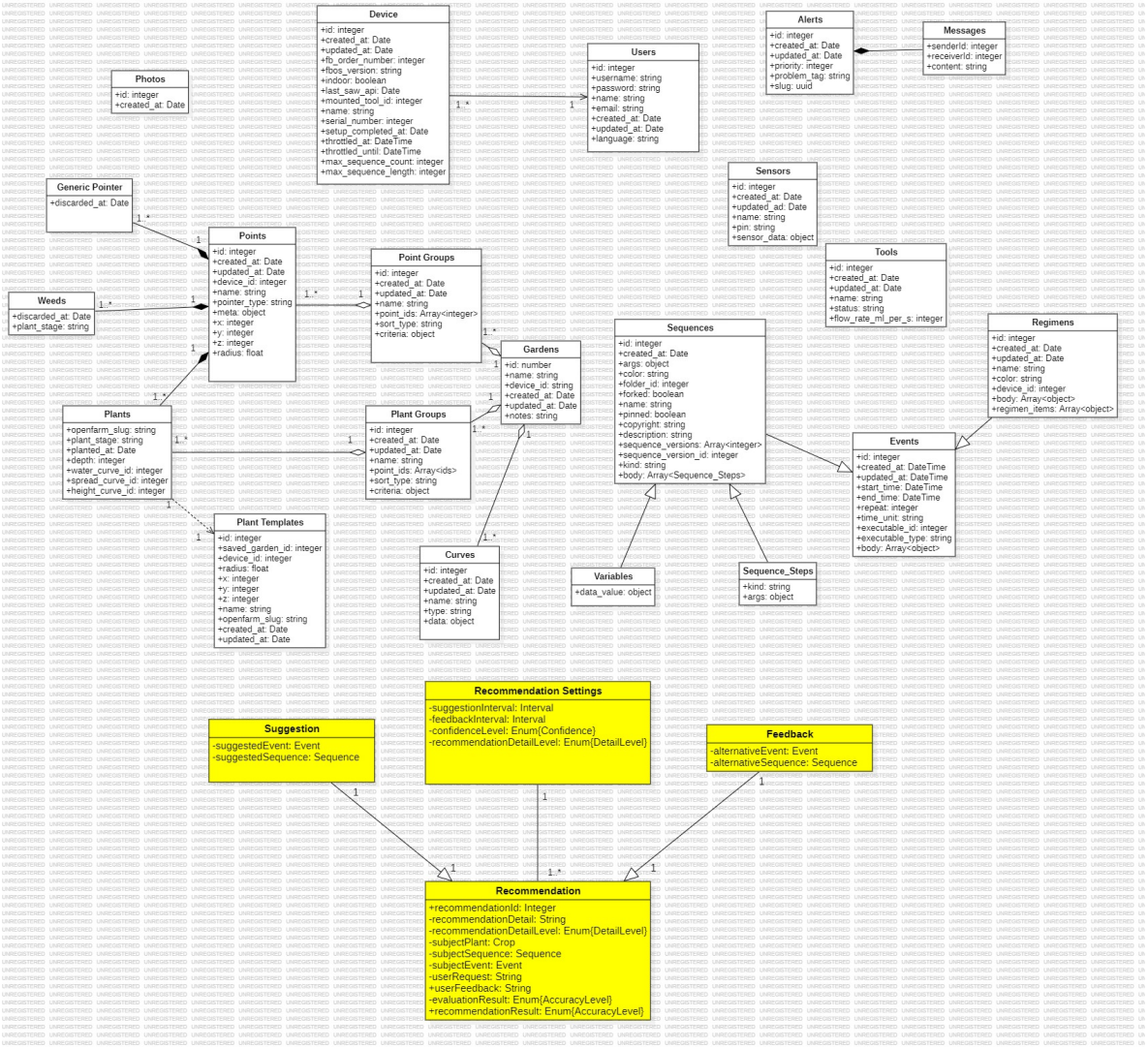

Figure 5.7: Database Class Diagram With Suggestion

- Suggestion represents a suggested action or recommendation provided by the system.
- Feedback represents a feedback on a user action such as an implemented sequence or a scheduled event.
- Recommendation Settings includes general configurations of the Suggestion and Feedback system.

5.3.3 Operations on Data

5.3.3.1 Operations and Descriptions

Operation	Description
getSuggestion	This operation allows users to get the most urgent and information suggestion from suggestion and feedback system based on the analysis of the current state of the garden.
getFeedback	This operation allows users to get the most urgent and information feedback from suggestion and feedback system based on the analysis of the current state of the garden.
makeRequest	This operation allows users to make a sequence or event request from suggestion and feedback system with a well structured message.
listRecommendations	This operation allows users to list all suggestion and feedback history.
deleteRecommendation	This operation allows users to delete a recommendation from recommendation history. This operation affects the data analysis of the suggestion and feedback system.
setSuggestionInterval	This operation allows users to adjust the frequency at which suggestions are generated and presented within the system.
setFeedbackInterval	This operation allows users to adjust the frequency at which feedback are generated and presented within the system.

setMinimumConfidenceLevel	This operation sets a minimum confidence threshold for recommendations.
setRecommendationDetailLevel	This operation adjusts the level of detail provided in recommendations.

Table 5.1: Suggested Operations On Data

5.4 CRUD Effects of Operations

Operation	**Affected Data Entities**
getSuggestion	Create: Suggestion Read: Suggestion Update: Recommendations Delete: -
getFeedback	Create: Feedback Read: Feedback Update: Recommendations Delete: -
makeRequest	Create: - Read: Recommendation Update: Recommendations Delete: -
listRecommendations	Create: - Read: Recommendations Update: - Delete: -

deleteRecommendation	Create: - Read: Recommendations Update: - Delete: Recommendation
setSuggestionInterval	Create: - Read: Recommendation Settings Update: Recommendation Settings Delete: -
setFeedbackInterval	Create: - Read: Recommendation Settings Update: Recommendation Settings Delete: -
setMinimumConfidenceLevel	Create: - Read: Recommendation Settings Update: Recommendation Settings Delete: -
setRecommendationDetailLevel	Create: - Read: Recommendation Settings Update: Recommendation Settings Delete: -

Table 5.2: Crud Effects of Suggested Operations

5.5 Deployment View

5.5.1 Stakeholders' uses of this view

The Deployment View offers stakeholders a comprehensive insight into the practical implementation of FarmBot Express, detailing how the system is installed, configured,

and managed within operational environments.

- **Engineering Students:** Engineering students would be interested in the deployment view to understand the technical setup and configuration of FarmBot Express. This would include aspects such as the deployment of sensors, actuators, and other hardware components, as well as the deployment of software systems such as the FarmBot control software and any additional coding frameworks utilized for customization or integration.

- **Scientists:** Researchers would also find the deployment view valuable as they assess the practical implementation of FarmBot Express for their experiments. They would be interested in how the device is deployed in different experimental setups, including its integration with other scientific instruments and how it is configured for specific experimental conditions.

- **Educators:** Teachers incorporating FarmBot Express into educational settings would benefit from understanding the deployment view to effectively set up and manage the device within the classroom or learning environment. This would involve considerations such as installation, calibration, and maintenance procedures, as well as any safety protocols or guidelines for student use.

5.5.2 Deployment Diagram

In terms of deployment view, the Suggestion and Feedback System enhances the FarmBot deployment by providing a seamless integration with the OpenFarm.cc FarmBot DBMS. This integration facilitates real-time access to extensive plant and garden data, enabling the system to generate more accurate suggestions and feedback. By leveraging cloud-based infrastructure, the system ensures scalability and reliability, accommodating varying user demands and ensuring consistent performance across different deployment environments. Additionally, it enables efficient deployment updates and maintenance, as improvements to the system can be rolled out centrally and seamlessly integrated into the FarmBot platform. The Warning System for FarmBot is deployed

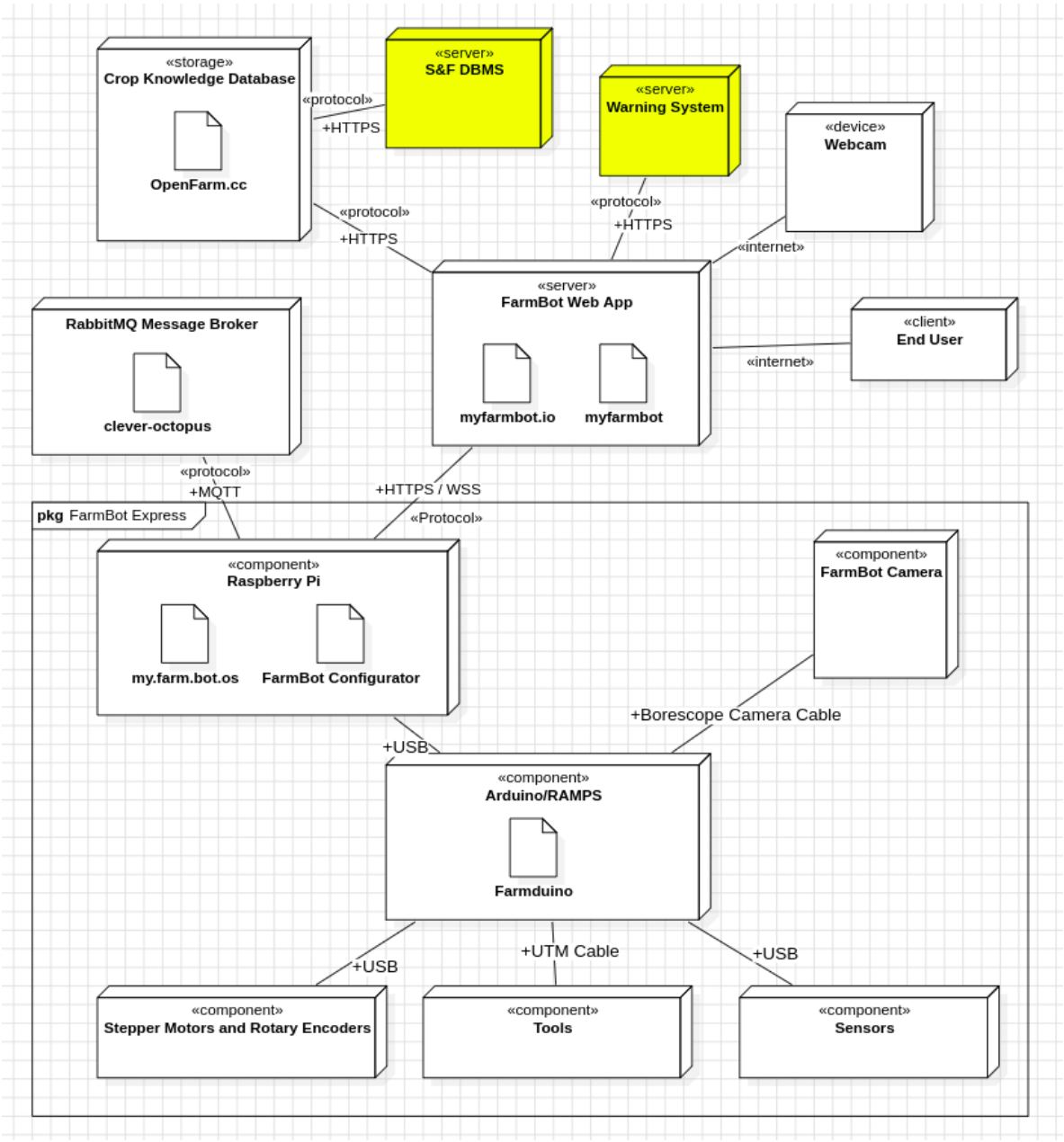

Figure 5.8: Deployment Diagram with Suggestion

with strategic sensor placement and seamless integration with existing software infrastructure. It relies on reliable internet connectivity and secure protocols for data exchange between FarmBot, sensors, and the web application interface. The system's scalability and redundancy ensure adaptability to farm growth and robustness against failures, while its ruggedized components and remote monitoring capabilities ensure

continuous operation and ease of maintenance in outdoor agricultural environments.

5.6 Design Rationale

Design Rational of Context View

Suggestion and Feedback System in terms of context view is to centralize and streamline data management for enhanced decision-making and user experience. This centralization simplifies the data architecture, reduces redundancy, and enhances data consistency, ultimately leading to more accurate and actionable insights for FarmBot users. The warning system protects valuable agricultural assets and offers peace of mind to farmers, knowing that their farm is under constant surveillance and that they can respond promptly to any threats.

Design Rational of Functional View

The system minimizes the need for manual input and expertise from users using S&F DBMS. This also educates users by providing actionable insights and recommendations. The integration with the OpenFarm.cc database ensures that the suggestions are backed by comprehensive and up-to-date agricultural knowledge, making the system a reliable virtual assistant for users.

Design Rational of Information View

In terms of the information view, the suggested systems ensure data integrity and accessibility. By structuring the database to include clearly defined attributes and relationships, such as plant data, user garden data, and feedback records, the system ensures that all relevant information is easily accessible and logically organized.

Design Rational of Deployment View

By deploying the S&F System on a cloud platform, it can dynamically scale resources to handle varying loads and user demands, ensuring optimal performance even during peak usage times.