Department of Computer Engineering

CENG350 Software Engineering

Software Architecture Description (SAD) for FarmBot

Group 55

By

Barış Can, 2644391

Başar Yılmaz, 2644409

Friday 31st May, 2024

Contents

List of Figures

List of Tables

Revision History

Version	Description	Date	Author
v1.0	Context View finished	08.05.2024	Barış, Başar
v1.1	Functional View finished	13.05.2024	Barış, Başar
v1.2	Information View finished	14.05.2024	Başar
v1.3	Deployment View finished	14.05.2024	Barış
v2.0	Suggestions, Functional and Deployment View finished	15.05.2024	Barış
v2.1	Suggestions, Information view and Context view finished	16.05.2024	Başar

Table 1: Revision Table

1. Introduction

This document is written to describe the Software Architecture Description (SAD) for the open-source project FarmBot. The FarmBot is an open-source CNC (computer numerical control) farming project. It consists of a cartesian coordinate robot farming machine, software, and documentation, which includes a farming data repository.

1.1 Purpose and objectives of FarmBot

The project aims to create open-source and accessible technology for people to grow food easily. In addition to that, the mission of the benefit corporation FarmBot Inc. is to build a community that develops open-source hardware plans, software, data, and documentation available to everyone to build their farming machine. The system, therefore, paves the way for decentralization and democratization of food production.

1.2 Scope

FarmBot is an open-source CNC farming project. The system aims to address the issues that have become prevalent with the ever-growing world population and its food demands. It significantly improves upon the conventional agriculture methods. The scope of the system consists of the following:

- Building a free and open database for farming and gardening knowledge
- A custom operating system, FarmBot OS, for maintaining connection and synchronization between the hardware and the web application using message bro-

kers. Allowing scheduling of events, real-time control, and uploading various sensor data and logs

- A web app for easy configuration and control of the FarmBot, featuring real-time manual control capabilities, logging, drag-and-drop farm designing and a routine builder for FarmBot to execute scheduled routines
- Scalability for all sizes of operation from home-use to industrial-use.
- Big data acquisition and analysis for data-driven agriculture.
- Providing the user with various farm design options with space efficiency in mind
- Fully automated and optimized farming operations such as planting, watering, spraying, weed detection, and seed spacing with mono-crop and poly-crop capabilities
- Offering customizability to the user in adding new sensors, adjusting parameters of their FarmBot to their liking

1.3 Stakeholders and their concerns

The stakeholders of the system range from students, researchers, and robotic artwork creators to home users and people with disabilities. Characteristics of the stakeholders are explained below:

- **Developers:** FarmBot being an open-source project, developers are concerned with improving existing features and implementing new ones, creating documentation for these features. Maintaining the technological ecosystem of FarmBot.
- **Students:** For students in any level of study from K12 (Kindergarten and grades 1-12) to university education, FarmBot provides a practical, fun and hands-on learning experience for STEM (Science, Technology, Engineering and Mathematics) learning objectives such as: **robotics**, **coding**, **soil science**, **biology**, and

much more. They can be called novices in the domain of FarmBot. They do not have the necessary knowledge to comprehend the hardware-software interactions fully. They are naturally curious, which encourages them to come up with new features and ways to use the device. They want to have the freedom to use their creativity. In addition to that, the students do not have a long attention span, and it might be hard to keep them engaged.

- **Researchers:** For researchers, there are various ways to take advantage of the FarmBot. It offers repeatability, scale, speed, and low cost to researchers.

  - **Repeatability**: The human error factor needs to be eliminated, leading to more accurate and repeatable results while experimenting. In addition, they want to schedule events (experiments) easily.
  - **Scale**: Researchers want to easily test multiple groups of crops and run multiple experiments concurrently.
  - **Speed**: Researchers want to create their own sequences and collect data automatically at any frequency and run experiments 24/7.
  - **Cost**: Researchers want to avoid high labor costs

  This class of stakeholders is highly educated and therefore can capitalize on FarmBot being highly customizable and integrable. They would like to create custom tooling (e.g., brush heads for coral farming) and custom regimens (e.g., speeding up coral growth) for their experiments. Collecting accurate data in a timely fashion is of utmost importance to them. Precise and continuous data flow from sensors and cameras frees the researchers from labor-intensive tasks and the potential for error. This class uses FarmBot for **phenotyping research**, **photogrammetry**, **off-world farming research** and such.

- **Robotic Artwork Creators:** This class of stakeholders approach to using FarmBot is different from the others. Their perspective is more creative and artistic. They use the various sensors and cameras of the device, feed the data from them into different algorithms such as **stable diffusion** to create art pieces. For this class, ability to customize farm designs and sequence customizing are crucial features. They have, similar to the researchers class, the required knowledge of technology and they are generally well educated.

- **Home User:** The home user class is usually driven to use FarmBot because of emerging problems in the world such as climate change, environmental pollution due to plastic packaging and excessive pesticide use. This class takes pride in being able to grow its food itself and being able to grow challenging plants. They are not experts on the technology, they have general knowledge. They usually use their backyards, small gardens for farming. They generally do not have a lot of spare time to tend to their garden each day.
- **People with disabilites:** These people suffer from various physical and mental limitations in their day-to-day lives. They are not able to farm on their own using conventional farming techniques (eg. can't clean the garden from weeds). Centres for disabled people can use FarmBot for horticultural therapy (the art or practice of garden cultivation and management) or vocational training. This would help them integrate with the society and give them a sense of autonomy. They do not have the technical expertise due to physical and mental handicaps.
- **Ministry of Agriculture:** This stakeholder is responsible for Agriculture activites and regulations within the country. As FarmBot is a complex agricultural system, the Ministry must audit it and make sure that FarmBot is not in breach of any agriculture regulations.

2. References

[1] "Farmbot express webpage." https://express.farm.bot/. Accessed: 2024-05-10.

[2] "Weather data api." https://openweathermap.org/api. Accessed: 2024-05-10.

[3] "Amazon alexa developer webpage." https://developer.amazon.com/en-US/alexa/alexa-skills-kit/get-deeper/dev-tools-skill-management-api. Accessed: 2024-05-10.

3. Glossary

12 Factor App Twelve-factor app is a methodology for building distributed applications that run in the cloud and are delivered as a service..

Arduino Arduino is an open-source electronics platform based on easy-to-use hardware and software..

CNC Computer numerical control (CNC) is a manufacturing method that automates the control, movement and precision of machine tools through the use of preprogrammed computer software, which is embedded inside the tools..

CRUD The four basic operations (create, read, update, and delete) of data storage, regarded collectively..

Docker Docker is an open platform for developing, shipping, and running applications..

Elixir Elixir is a dynamic, functional language for building scalable and maintainable applications..

GitHub A web-based version control system.

Heroku Heroku is a platform as a service (PaaS) that enables developers to build, run, and operate applications entirely in the cloud..

PostgreSQL PostgreSQL is a powerful, open source object-relational database system with over 35 years of active development that has earned it a strong reputation for reliability, feature robustness, and performance..

RabbitMQ RabbitMQ is a reliable and mature messaging and streaming broker, which is easy to deploy on cloud environments, on-premises, and on your local machine..

Rails Rails is a full-stack framework. It ships with all the tools needed to build amazing web apps on both the front and back end..

Raspberry Pi A small single-board computer developed in the United Kingdom by the Raspberry Pi Foundation.

REST REST, or Representational State Transfer, is an architectural style for providing standards between computer systems on the web, making it easier for systems to communicate with each other..

SAD System Analysis and Design.

SRS Software Requirements Specification.

4. Architectural Views

4.1 Context View

4.1.1 Stakeholders' uses of this view

The key use of the Context View for the stakeholders of [1] FarmBot is to ensure that the overall end-to-end solution indeed sensibly satisfies functionality requirements from a general viewpoint. The different stakeholders' uses of this view are as follows:

- Developers: Use the context view in order to have a better picture of how the system fits into the overall application landscape. They use it to understand the interactions with external systems and entities. In addition to that, for further improvements to the system, they can better assess completeness, consistency and coherence.

- End Users (Students, Researchers, Home Users, Artwork Creators): Use the context view in order to ensure that FarmBot is able to satisfy their requirements and its' scope is correct. Also it enables them to learn how the data flows, the interactions are done throughout the FarmBot system and what external entities or services are part of it.

- Ministry of Agriculture: Use the context view for auditing FarmBot as a system, see what external entities and services are used, where and how data flows.

4.1.2 Context Diagram

The FarmBot is a standalone system that interacts with four external entities as shown in Figure 4.1:

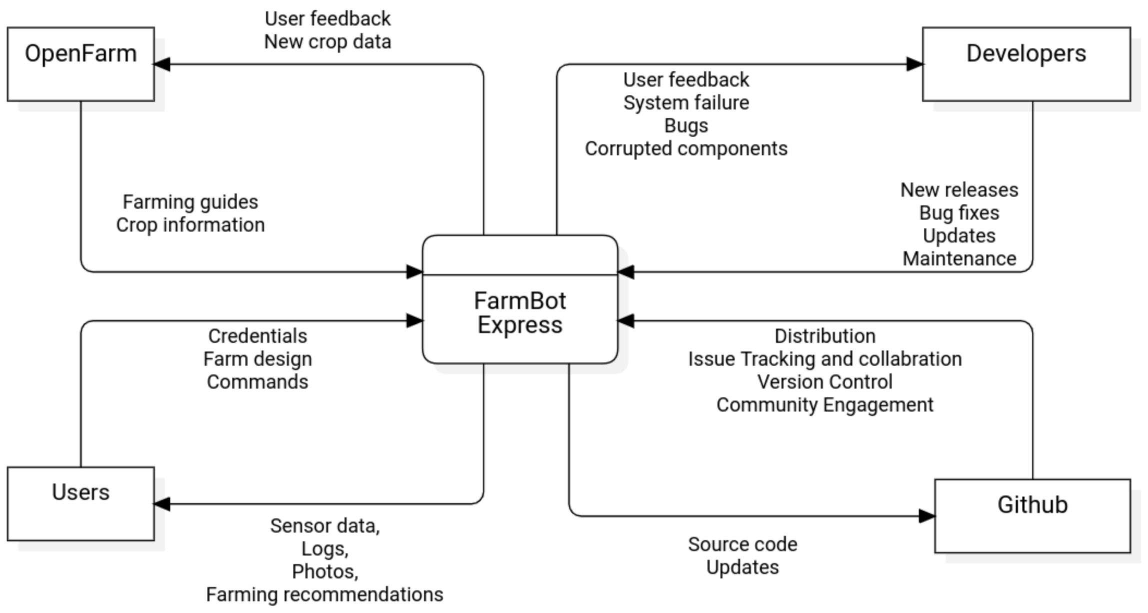


Figure 4.1: FarmBot Context Diagram

- **Users:** The user interacts with the system through the web application. The user can create, edit, and delete sequences, view the status of the FarmBot, and monitor the progress of the FarmBot. The user provides data to the system, such as the location of the FarmBot, the type of plants, and the planting schedule. The user can also receive data from the system such as the status of the FarmBot and the progress of the FarmBot.

- **Developers:** The developers are responsible for maintaining the system and ensuring that it is operational. They can access the system through the web application and perform maintenance tasks. Furthermore, they can update the system and add new features.

- **OpenFarm:** The OpenFarm is an open-source database system that contains information about plants and crops. It can be considered a data store (a shared database) external to the system. The system interacts with OpenFarm to retrieve information about plants and crops. This information is used to determine the planting schedule and the requirements of the plants.

- **GitHub:** The GitHub is a version control system that stores the system's source code. The system interacts with GitHub to retrieve the source code and update the system. The developers can access the source code through GitHub and make changes to the system. In addition, it encourages community engagement and collaboration.

4.1.3 External Interfaces

The FarmBot interacts with four external entities as shown in Figure 4.2:

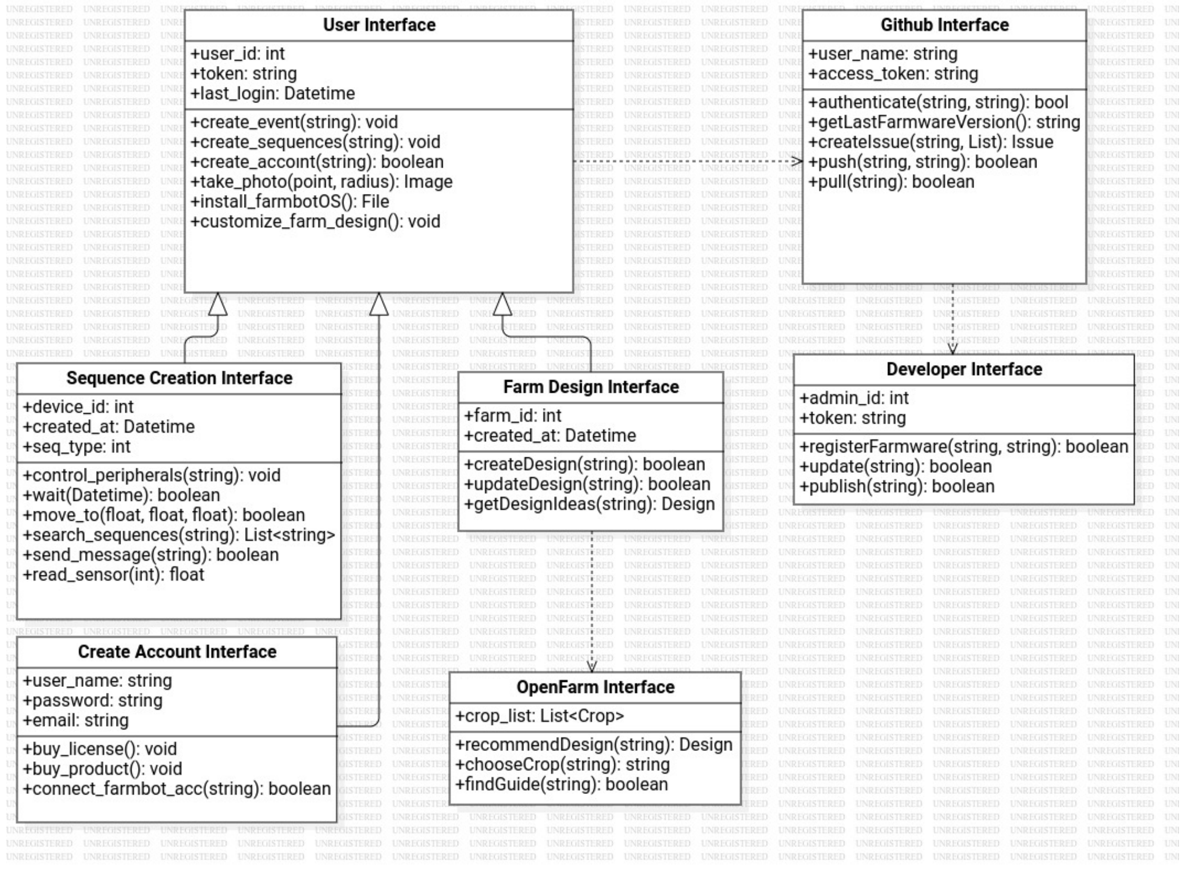

User Interface
+user_id: int
+token: string
+last_login: Datetime
+create_event(string): void
+create_sequences(string): void
+create_accoınt(string): boolean
+take_photo(point, radius): Image
+install_farmbotOS(): File
+customize_farm_design(): void
Github Interface
+user_name: string
+access_token: string
+authenticate(string, string): bool
+getLastFarmwareVersion(): string
+createIssue(string, List): Issue
+push(string, string): boolean
+pull(string): boolean
Sequence Creation Interface
+device_id: int
+created_at: Datetime
+seq_type: int
+control_peripherals(string): void
+wait(Datetime): boolean
+move_to(float, float, float): boolean
+search_sequences(string): List<string>
+send_message(string): boolean
+read_sensor(int): float
Farm Design Interface
+farm_id: int
+created_at: Datetime
+createDesign(string): boolean
+updateDesign(string): boolean
+getDesignIdeas(string): Design
Developer Interface
+admin_id: int
+token: string
+registerFarmware(string, string): boolean
+update(string): boolean
+publish(string): boolean
Create Account Interface
+user_name: string
+password: string
+email: string
+buy_license(): void
+buy_product(): void
+connect_farmbot_acc(string): boolean
OpenFarm Interface
+crop_list: List<Crop>
+recommendDesign(string): Design
+chooseCrop(string): string
+findGuide(string): boolean

Figure 4.2: FarmBot External Interfaces Class Diagram

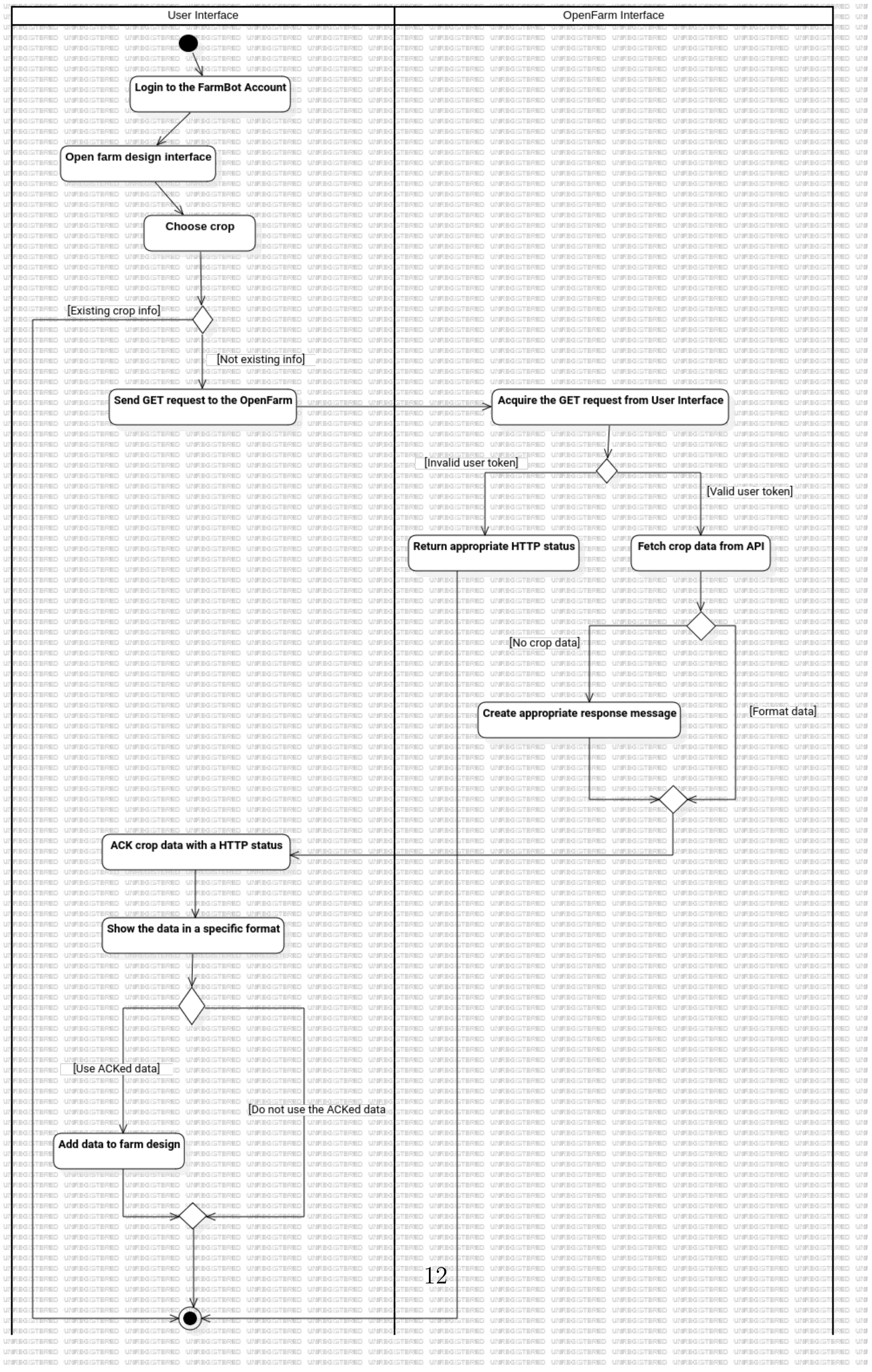
- **User Interface:** The User Interface is responsible for handling the user interactions with the system. It provides the user with a graphical user interface (GUI) that allows the user to interact with the system. The User Interface receives input from the user and sends it to the system. It also receives output from the system and displays it to the user. The User Interface is implemented as a web application that can be accessed through a web browser. It can be considered as an event provider or consumer in our case.

- **Developer Interface:** The developer interface is used to interact with the system for development purposes. Developers can create new features, fix bugs, etc. Developers use this interface to contribute to the system. With further investigation, one can see that users depend on this interface to get the latest version of the software since the maintenance of the GitHub interface depends on developers.

- **OpenFarm Interface:** The OpenFarm Interface is responsible for interacting with the OpenFarm database. It retrieves information about plants and crops from the OpenFarm database. The OpenFarm Interface sends requests to the OpenFarm database and receives responses from the OpenFarm database. The OpenFarm Interface is implemented as a RESTful API that can be accessed over the internet. It can be considered as a data or service provider in our case.

- **GitHub Interface:** The GitHub Interface is responsible for interacting with the GitHub version control system. It retrieves the source code of the system from the GitHub repository. The GitHub Interface sends requests to the GitHub repository and receives responses from the GitHub repository. Furthermore, it provides users and developers with a forum to collaborate and contribute to the system. The GitHub Interface is implemented as a web application that can be accessed through a web browser. It can be considered as a data or service provider in our case.

4.1.4 Interaction scenarios


User Interface
OpenFarm Interface
Login to the FarmBot Account
Open farm design interface
Choose crop
[Existing crop info]
[Not existing info]
Send GET request to the OpenFarm
Acquire the GET request from User Interface
[Invalid user token]
[Valid user token]
Return appropriate HTTP status
Fetch crop data from API
[No crop data]
Create appropriate response message
[Format data]
ACK crop data with a HTTP status
Show the data in a specific format
[Use ACKed data]
[Do not use the ACKed data
Add data to farm design

The activity diagram in Figure 4.3 shows the interaction between the User Interface and the OpenFarm Interface. The User Interface sends a request to the OpenFarm Interface to retrieve information about plants and crops. The User Interface can request the OpenFarm Interface to retrieve information about a specific plant or crop. The OpenFarm Interface responds to the User Interface with information about the specific plant or crop. The User Interface displays the information to the user.

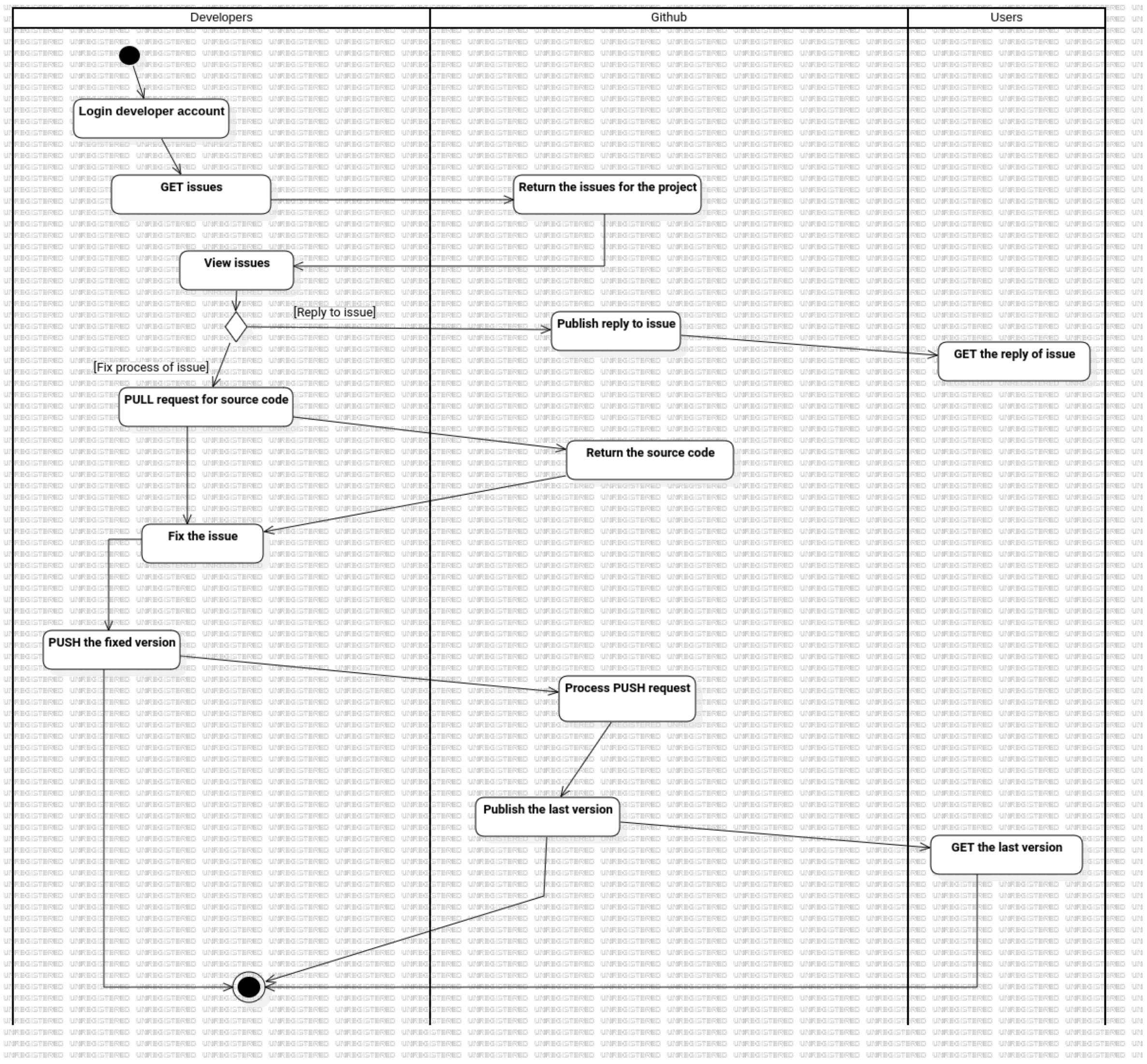


Figure 4.4: Activity diagram of a Maintenance Pipeline

The activity diagram in Figure 4.4 shows the interaction between the Developer, GitHub, and the user interface; it shows a basic instance of a maintenance action. The developer sends a request to the GitHub Interface to retrieve the system's source code. The GitHub Interface responds to the Developer with the system's source code. The Developer changes the source code and sends a request to the GitHub Interface to update the system. The GitHub Interface updates the system with the changes made by the Developer. The User Interface displays the updated system to the user.

4.2 Functional View

4.2.1 Stakeholders' uses of this view

The Functional View offers an overview of how architectural elements work together to provide the various functionalities of FarmBot. Below, you can find the FarmBot Stakeholders' uses of this view:

- **Developers**: The developers' main concern is the quality of design and the internal structure of the FarmBot system. The internal structure is crucial to meet the desired quality requirements such as availability, ability to scale, security during development. Also developers can use this view to see the external interfaces.
- **End Users** (Students, Researchers, Home Users, Artwork Creators): End users can use this view to learn the functionalities provided by FarmBot and how they are provided, what external interfaces are used.
- **Ministry of Agriculture**: Their use of this view is to have a more in-depth knowledge of the overall system, internal structure, external interfaces, and functionality for auditing.

4.2.2 Component Diagram

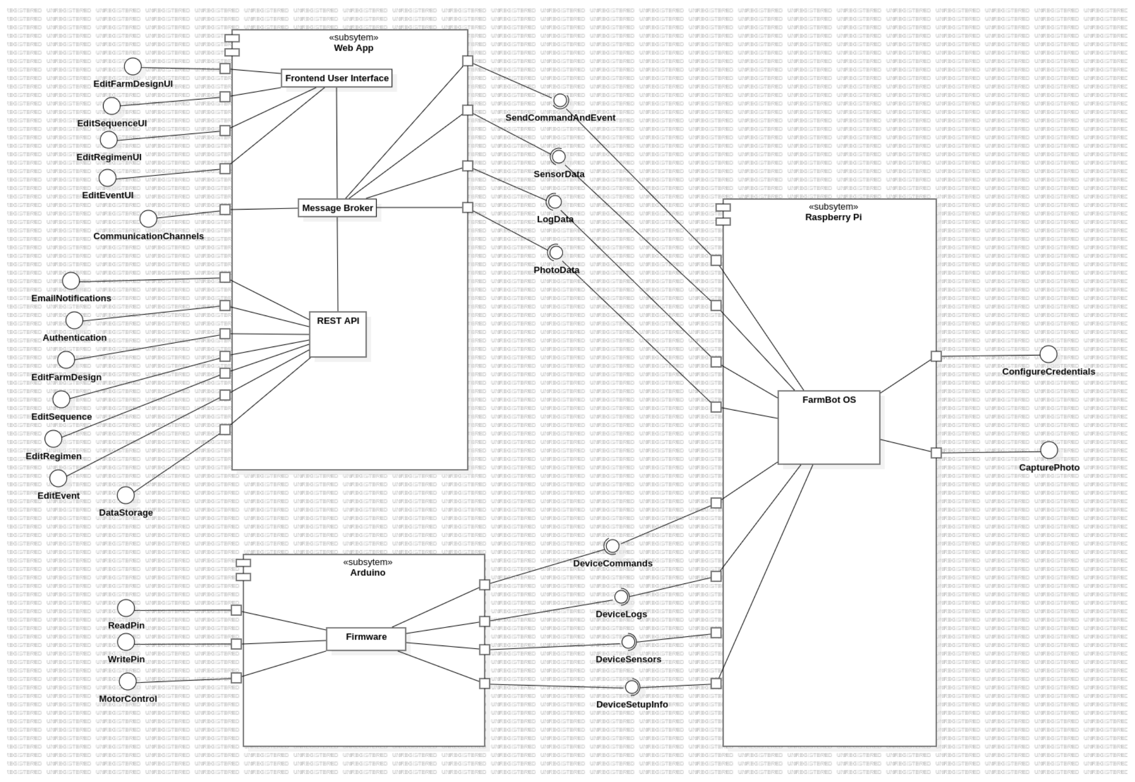


Figure 4.5: Component Diagram for FarmBot

The Component Diagram for FarmBot consists of three subsystems: Web App, Raspberry Pi, and Arduino.

- **Web App:** The Web App has three parts in Frontend User Interface, Message Broker, and REST API.
  - **The Frontend User Interface** provides UI elements for the end user to design their farm, edit, and create sequences, regimens, or events.
  - **The Message Broker** provides communication between FarmBot device and WebApp. The device is controlled through the message broker. It handles photo, sensor and log data flow. The commands, events of user are sent through it. Provides an external interface for different communcation channels.

– **The REST API** provides the functionality for user actions with the Web App such as editing farm design, sequences, regimens, and events. It provides long term data storage and authentication.

- **Raspberry Pi:** The Raspberry Pi componenet consists of FarmBot OS. FarmBot OS provides external interfaces for device configuration (WIFI and account credentials) and capturing photos. In addition, it provides interfaces to the message broker for the flow of photo, log, and sensor data. It generates G and F code commands from high level user commands then passes them to the FarmBot device.

- **Arduino:** The Arduino consists of the firmware. It provides external interfaces for reading and writing device pins, and controlling the motors. In addition, it executes commands that were passed by FarmBot OS and provides device logs, sensor data and setup information of pins and encoders back.

4.2.3 Internal Interfaces

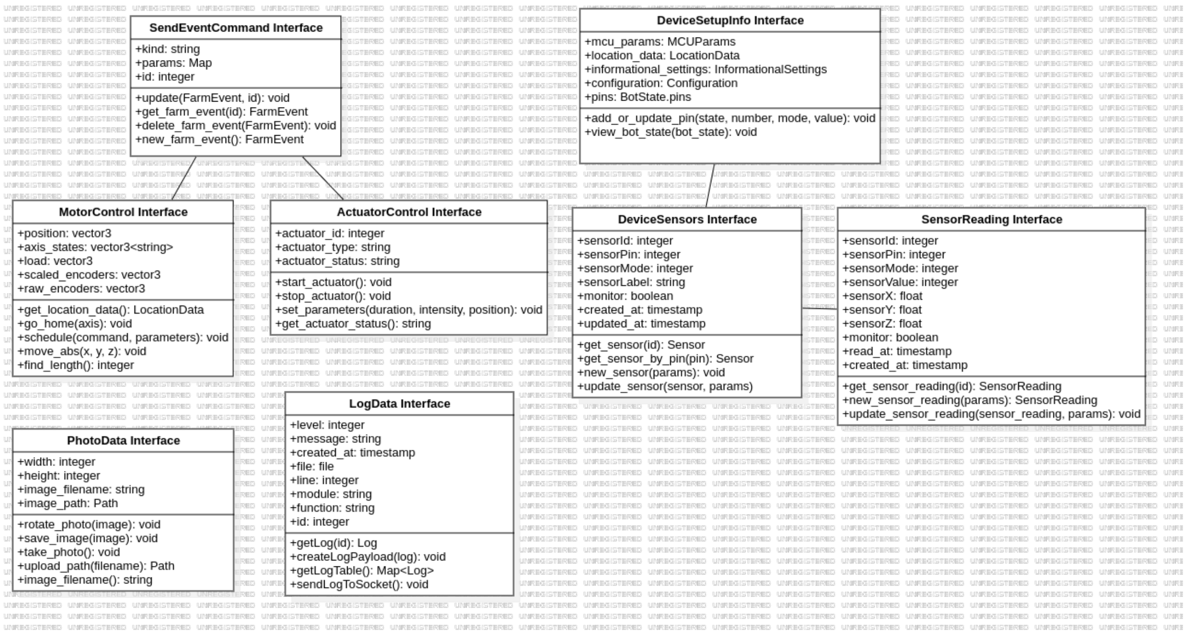


Figure 4.6: FarmBot Internal Interface Diagram

- **PhotoData Interface:**
  - **width**: The width of the image
  - **height**: The height of the image
  - **image_filename**: The stored filename of the image with timestamp
  - **image_path**: The path of the stored image used for uploading the image
  - **rotate_photo(image)**: Rotates the captured if the calibration data exists.
  - **save_image(image)**: Saves the captured image to a file after modifying operations
  - **take_photo()**: Takes a photo using the connected camera
  - **upload_path(filename)**: Returns the path for the image file to be uploaded.
  - **image_filename()**: Prepares and returns the filename with timestamp.
- **SendCommandAndEvent Interface:**
  - **kind**: The type of the command asset (FarmEvent, Regimen etc.)
  - **params**: A key value map of changes to be made with the command.
  - **id**: The id of the remote database that stores events and commands.
  - **update(FarmEvent, id)**: Updates the FarmEvent stored in the database.
  - **get_farm_event(id)**: Returns the FarmEvent with the given id.
  - **delete_farm_event(id)**: Deletes the FarmEvent with the given id from the database.
  - **new_farm_event()**: Creates and returns a new FarmEvent.
- **MotorControl Interface:**
  - **position**: A vector with three components storing the position of the FarmBot

- **axis_states**: A vector with three components storing the position of the FarmBot with string representation
- **load**: A vector with three componenets that stores the load on each axis of the FarmBot
- **scaled_encoders**: A vector with three components that hold the rotation of each motor axis, scaled with the correct motor_resolution.
- **raw_encoders**: A vector with three components that hold the raw rotation values of each motor axis.
- **get_location_data()**: Returns the location data of the FarmBot device.
- **go_home(axis)**: Commands the FarmBot device to move to its home position regarding to the given axis.
- **move_abs(x, y, z)**: Move to the given location with absolute coordinates.
- **schedule(command, parameters)**: Communicates with the Arduino firmware in G Code to start executing the command.
- **find_length(axis)**: Find the length of the given axis.

• **ActuatorControl Interface:**

- **actuator_id**: The ID of the actuator
- **actuator_type**: An indicator of the type of actuator (e.g., watering pump, linear actuator, LED light strip).
- **actuator_status**: Indicates the current state of the actuator (e.g., on, off, idle, busy).
- **start_actuator()**: Starts the actuator
- **stop_actuator()**: Stops the actuator
- **set_parameters(duration, intensity, position)**: Sets the parameters of actuator for executing its action
- **get_actuator_status()**: Returns the current state of the actuator

- **SensorReading Interface:**
  - **sensorId**: The ID of the sensor.
  - **sensorPin**: The pin of the sensor on the FarmBot device.
  - **sensorMode**: The I/O mode of the sensor.
  - **sensorValue**: The value of the sensor.
  - **sensorX**: The X position of the sensor.
  - **sensorY**: The Y position of the sensor.
  - **sensorZ**: The Z position of the sensor.
  - **monitor**: A boolean value set if the sensor will be monitored.
  - **read_at**: The timestamp of last reading from sensor.
  - **created_at**: The timestamp when the sensor was first created.
  - **get_sensor_reading(id)**: Returns the latest reading from the sensor with the given id.
  - **new_sensor_reading(params)**: Creates a new sensor reading from the sensor using the params.
  - **update_sensor_reading(sensor_reading, params)**: Updates an existing sensor_reading using the given params.
- **DeviceSensors Interface:**
  - **sensorId**: The ID of the sensor.
  - **sensorPin**: The pin of the sensor
  - **sensorMode**: The I/O mode of the sensor
  - **sensorLabel**: The string label attached to the sensor.
  - **monitor**: A boolean value set if the sensor will be monitored.
  - **created_at**: The creation timestamp of sensor.
  - **updated_at**: The update timestamp of sensor.

- **get_sensor(id)**: Returns the sensor with the given id.
- **get_sensor_by_pin(pin)**: Returns the sensor that is connected to the given pin.
- **new_sensor(params)**: Creates a new sensor using the given parameters.
- **update_sensor(sensor, params)**: Updates the given sensor with the given parameters.

- **DeviceSetupInfo Interface:**
  - **mcu_params**: Holds hardware parameter information regarding the device such as encoders, pins, movement settings of motors.
  - **location_data**: Holds the current location data of the device.
  - **informational_settings**: Holds general information about the hardware device status such as firmware version, controller version, WiFi level, uptime, memory usage.
  - **configuration**: Holds the configuration settings of firmware such as firmware hardware, input-output logs, debug logs, auto-update settings.
  - **pins**: Holds the pin information for the device.
  - **add_or_update_pin(state, number, mode, value)**: Updates the value of a given pin with number, state and mode. In the case it does not already exist, adds new pin.
  - **view_bot_state(bot_state)**: Returns a view of the current state of the FarmBot such as position, pins, stall status.

- **LogData Interface:**
  - **level**: The level of the log indicating the type such as debug, info, error, warn, success.
  - **message**: The message string to be logged.
  - **created_at**: The timestamp of the log creation.

– **file**: The file associated with the log entry.

– **line**: The line of file associated with the log entry.

– **module**: The software module associated with the log entry.

– **function**: The function that is associated with the log entry.

– **id**: The id of the log.

– **get_log(id)**: Returns the log with the given id.

– **create_log_payload(log)**: Creates a log payload to be sent over the network.

– **get_log_table()**: Returns a map of id and logs for all logs.

– **send_log_to_socket()**: Sends the created log payload over the network to the web socket.

4.2.4 Interaction Patterns

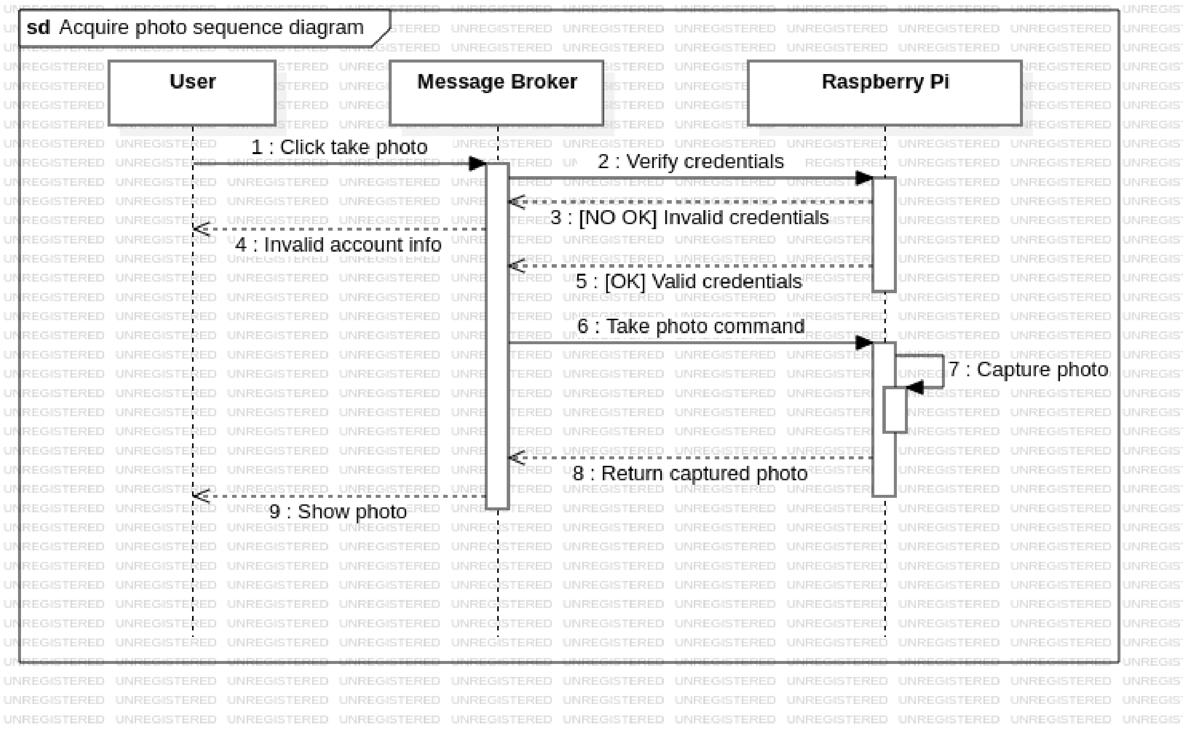


Figure 4.7: Acquire Photo Sequence Diagram

The sequence diagram in Figure 4.7 shows the interaction between the **User**, **Message Broker** (FarmBot Web App) and **Raspberry Pi**. The **User** sends a request to the **Message Broker** to acquire a photo. The **Message Broker** sends a request to the **Raspberry Pi** to acquire a photo. The **Raspberry Pi** acquires the photo and sends it to the **Message Broker**. The **Message Broker** sends the photo to the **User**.

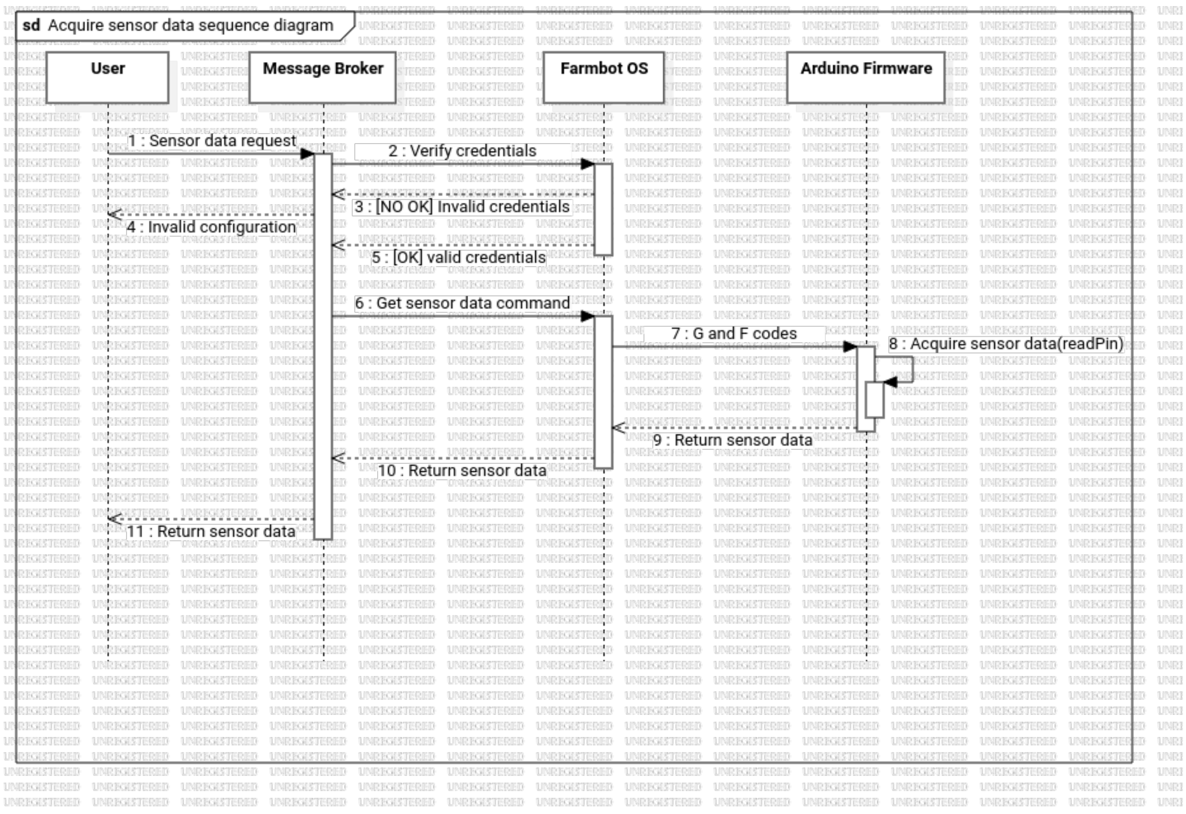

sd Acquire sensor data sequence diagram
User
Message Broker
Farmbot OS
Arduino Firmware
1 : Sensor data request
2 : Verify credentials
3 : [NO OK] Invalid credentials
4 : Invalid configuration
5 : [OK] valid credentials
6 : Get sensor data command
7 : G and F codes
8 : Acquire sensor data(readPin)
9 : Return sensor data
10 : Return sensor data
11 : Return sensor data

Figure 4.8: Acquire Sensor Data Sequence Diagram

The sequence diagram in Figure 4.8 shows the interaction between the **User**, **Message Broker** (FarmBot Web App) and **FarmBot OS** (Raspberry Pi), and the **Arduino Firmware**. The **User** sends a request to the **Message Broker** to acquire sensor data. The **Message Broker** sends a converted command to the Raspberry Pi's OS to acquire sensor data. The OS creates related G and F codes that can be encoded/decoded in the **Arduino Firmware**. The **Arduino Firmware** acquires the sensor data and sends it to the **User**.

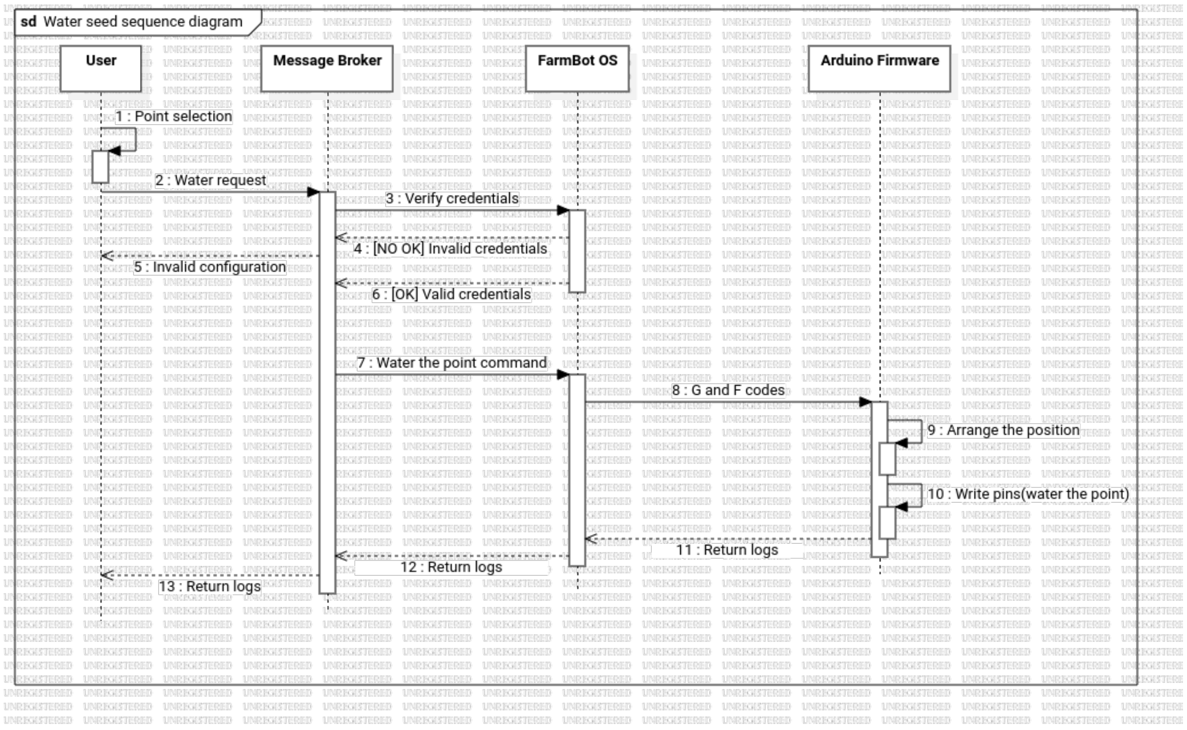


Figure 4.9: Watering Crop Sequence Diagram

The sequence diagram in Figure 4.9 shows the interaction between the **User**, **Message Broker** (FarmBot Web App), **FarmBot OS** (Raspberry Pi), and the **Arduino Firmware**. The **User** sends a request to the **Message Broker** to water the crops at the specified coordinates (point). The **Message Broker** sends a converted command to the Raspberry Pi's OS to water the crops at the specified coordinates. The OS creates related G and F codes that can be encoded/decoded in the **Arduino Firmware**. The **Arduino Firmware** waters the crops at the specified coordinates and sends the status (logs) to the **User**.

4.3 Information View

4.3.1 Stakeholders' uses of this view

The key use of the **Information View** for the stakeholders of FarmBot is to ensure that the data requirements are correctly understood and that the data is stored and handled correctly. The different stakeholders' uses of this view are as follows:

- **End users:** The end users primarily interact with the information view to understand the data being collected by FarmBot and how it relates to their gardening activities. They may use this view to monitor plant health, track growth progress, and receive alerts or recommendations.
- **Developers:** Developers use the information view to design and implement features related to data collection, storage, and processing. They ensure that the data is collected efficiently, stored securely, and can be accessed and manipulated as needed for various functionalities.
- **Ministry of Agriculture:** The Ministry of Agriculture utilizes the information view to oversee and regulate the use of FarmBot data for agricultural purposes. They may use this view to monitor farming practices, analyze trends, and make informed decisions regarding agricultural policies and practices.

4.3.2 Database Class Diagram

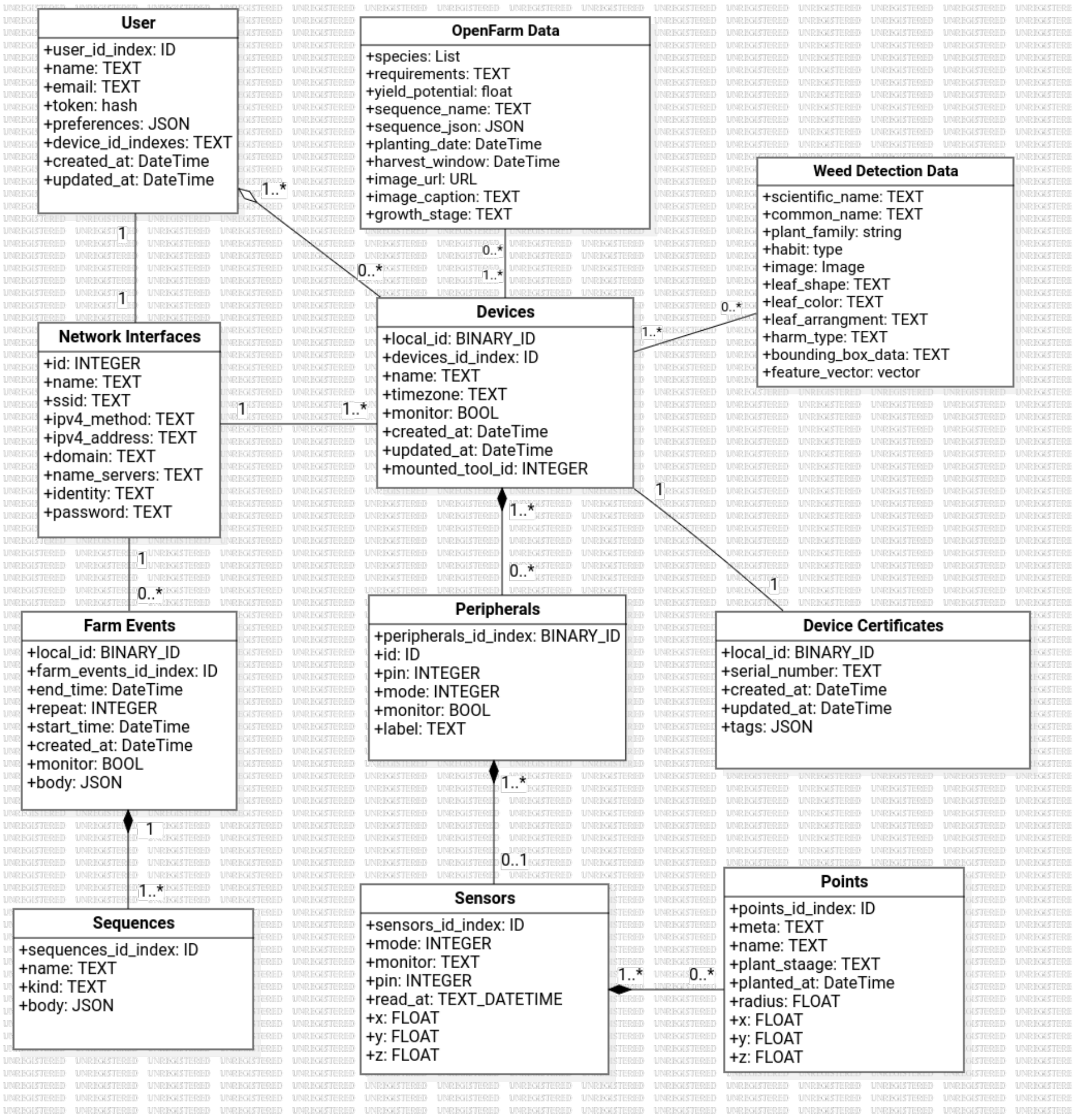


Figure 4.10: FarmBot Database Class Diagram

The **Database Class Diagram** in Figure 4.10 shows the key database objects used by the FarmBot system. The database objects include **user, device, network interface, farm events, sequences, etc.**. Some of the main tables explained in the diagram are as follows:

- **User:** The **user** object represents the user of the system. It contains information about the user, such as the user's name, email address, and password.
- **Device:** The **device** object represents the FarmBot device. It contains information about the device, such as its name, location, and status.
- **Network Interface:** The **network interface** object represents the network interface of the FarmBot device. It contains information about the network interface, such as the network interface's name, IP address, and status.
- **Farm Events:** The **farm events** object represents the events that occur on the farm. It contains information about the farm events, such as the event's name, date, and description.
- **Sequences:** The **sequences** object represents the sequences of actions that the FarmBot device can perform. It contains information about the sequences, such as the sequence's name, description, and actions.
- **OpenFarm Data:** The **open farm data** object represents the data retrieved from the OpenFarm database. It contains information about the plants and crops, such as the plant's name, description, and requirements.
- **Sensors:** The **sensors** object represents the sensors on the FarmBot device. It contains information about the sensors, such as their name, type, and status.
- **Points:** The **points** object represents the points on the farm. It contains information about the points, such as their name, location, and status.
- **Weed Detection Data:** The **weed detection data** object represents the data collected by the weed detection system. It contains information about the weed detection data, such as the data's name, location, and status. It can be used to detect and remove weeds from the farm. The data stored in this table can be used in image processing algorithms to detect and remove weeds from the farm. Furthermore, using bounding box data and feature vectors of the detected weeds, some AI algorithms can be used to classify and remove the weeds.

4.3.3 Operations on Data

Operation	Description
CreateAccount	Create: User Read:- Update:- Delete:-
DeleteAccount	Create:- Read:- Update:- Delete:User
AddNewDevice	Create:Devices Read:- Update:User Delete:-
DeleteDevice	Create:- Read:- Update: User Delete: Devices
GetDevices	Create:- Read: Devices,User Update:- Delete:-
CreateFarmEvent	Create: FarmEvents, Sequences Read:- Update:User Delete:-

Table 4.1 continued from previous page

Operation	Description
GetPinReport	Create: - Read: Sensors Update:Sensors Delete:-
MoveAbsolute	Create:- Read: Points Update:Points, Sensors Delete:-
GetPosition	Create:- Read:Points Update:- Delete:-
NewSensorReading	Create:- Read:- Update: Sensor Delete:-
ReadStatus	Create:- Read:Devices Update:- Delete:-
UpdateSequence	Create:- Read: FarmEvents Update: Sequences Delete:-

Table 4.1 continued from previous page

Operation	Description
GetSequence	Create:- Read: Sequences, FarmEvents Update: - Delete: -
NewSequence	Create: Sequences Read: - Update: - Delete: -

Table 4.1: CRUD Operations

4.4 Deployment View

4.4.1 Stakeholders' uses of this view

The stakeholders uses of the Deployment view are described as follows:

- Developers: They use this view to better understand how they should orchestrate their development environment, what should their coverage be for deployment.
- End Users: They use this view to learn what tools, software or applications they need to get the system up and running.
- Ministry of Agriculture: They use this view to better understand dependencies, relations and interactions in the system. It can offer a inside view to how the system is orchestrated, aiding them in auditing.

4.4.2 Deployment Diagram

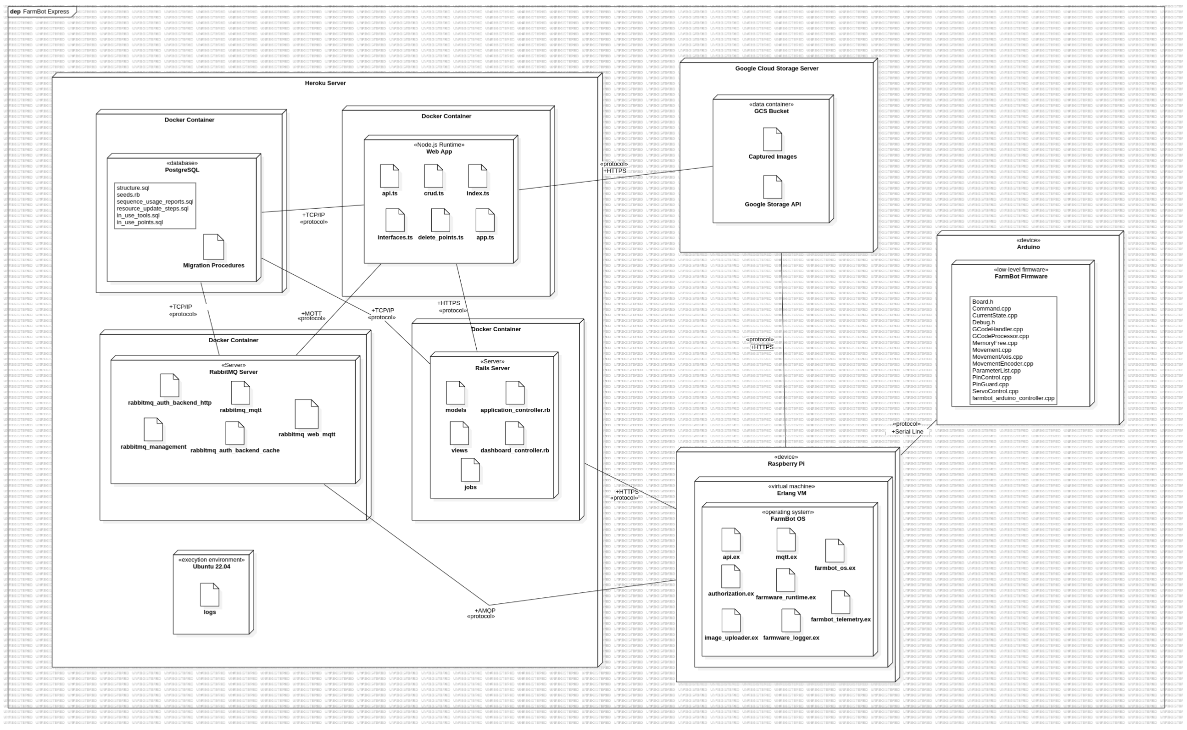


Figure 4.11: FarmBot Deployment Diagram

The deployment diagram depicts the cloud-based FarmBot system architecture in which a central server manages updates, a Raspberry Pi controller runs the core application and a web application provides the user interface. The web application and the servers are deployed on Heroku. It's highly containerized. Docker containers are used for the PostgreSQL database, the Node.js runtime for the frontend, the RabbitMQ server used as the message broker, and Rails server used for backend. The containers run on a Ubuntu based machine. Besides the FarmBot Web App, Google Cloud Storage is used as an external datastore for storing images. FarmBot OS, an Elixir based operating system, is deployed on a Raspberry Pi Controller using Erlang VM. It communicates with the FarmBot firmware running on Arduino using low-level G and F codes. The firmware is responsible for controlling the hardware such as motors, sensors, actuators.

4.5 Design Rationale

- **Context View:** In the context of **FarmBot Express**, the Context View plays a particularly important role in elucidating how various external entities interact with the system. This view sheds light on how users engage with **FarmBot Express**, how developers contribute to its maintenance and updates, how the system fetches vital plant data from OpenFarm to inform agricultural decisions, and how it seamlessly integrates with GitHub for retrieving and incorporating the latest source code. This transparency empowers stakeholders to grasp the complete user experience, development workflow, data acquisition process, and software update mechanisms within the **FarmBot Express** ecosystem.

- **Functional View:** The Functional View, divides the system into three components. The Web app provides external interfaces to the end user to interact with, communicates with the FarmBot hardware through internal interfaces. FarmBot OS and Arduino firmware control hardware actions. The maintainability of the system is achieved with this separation of concerns.

- **Information View:** The Information View plays a critical role in deciphering FarmBot Express's data landscape. This view delves into the system's data requirements, outlining how data is stored, managed, and processed. Stakeholders gain valuable insights into the system's data model, allowing them to comprehend the structure and organization of information. Additionally, the Information View clarifies data flow throughout the system, providing transparency into how data is exchanged, manipulated, and utilized within FarmBot Express. This understanding empowers stakeholders to make informed decisions regarding data management strategies and ensures the system effectively handles the information crucial for its operation.

- **Deployment View:** The deployment conforms with the methodology of 12 Factor App. This ensures that the application is suitable to be deployed on modern cloud platforms, enables continuous deployment, can scale-up and offers maximum portability. For particular decision rationales:

- Google Cloud Storage is used because uploading large files such as image files can take longer than 30 seconds on slow internet connections and for web applications that block requests, it can result in timout. Therefore, Heroku suggests directly uploading to a data bucket.
- A message broker is used because some interactions do not work well with the request/response pattern. We do not want to constantly check the API for messages. Instead, messages shall be received as soon as they are created without explicitly asking for them. Also, message brokers offer remote procedure calls and real-time data syncing.

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

5.1 Context View

5.1.1 Stakeholders' uses of this view

The key use of the Context View for the stakeholders of FarmBot is to ensure that the overall end-to-end solution indeed sensibly satisfies functionality requirements from a general viewpoint. The different stakeholders' uses of this view are as follows:

- Developers: Use the context view in order to have a better picture of how the system fits into the overall application landscape. They use it to understand the interactions with external systems and entities. In addition to that, for further improvements to the system, they can better assess completeness, consistency, and coherence.

- End Users (Students, Researchers, Home Users, Artwork Creators): Use the context view in order to ensure that FarmBot is able to satisfy their requirements and that its scope is correct. Also, it enables them to learn how the data flows, the interactions are done throughout the FarmBot system, and what external entities or services are part of it.

- Ministry of Agriculture: Use the context view for auditing FarmBot as a system, see what external entities and services are used and where and how data flows.

5.1.2 Context Diagram

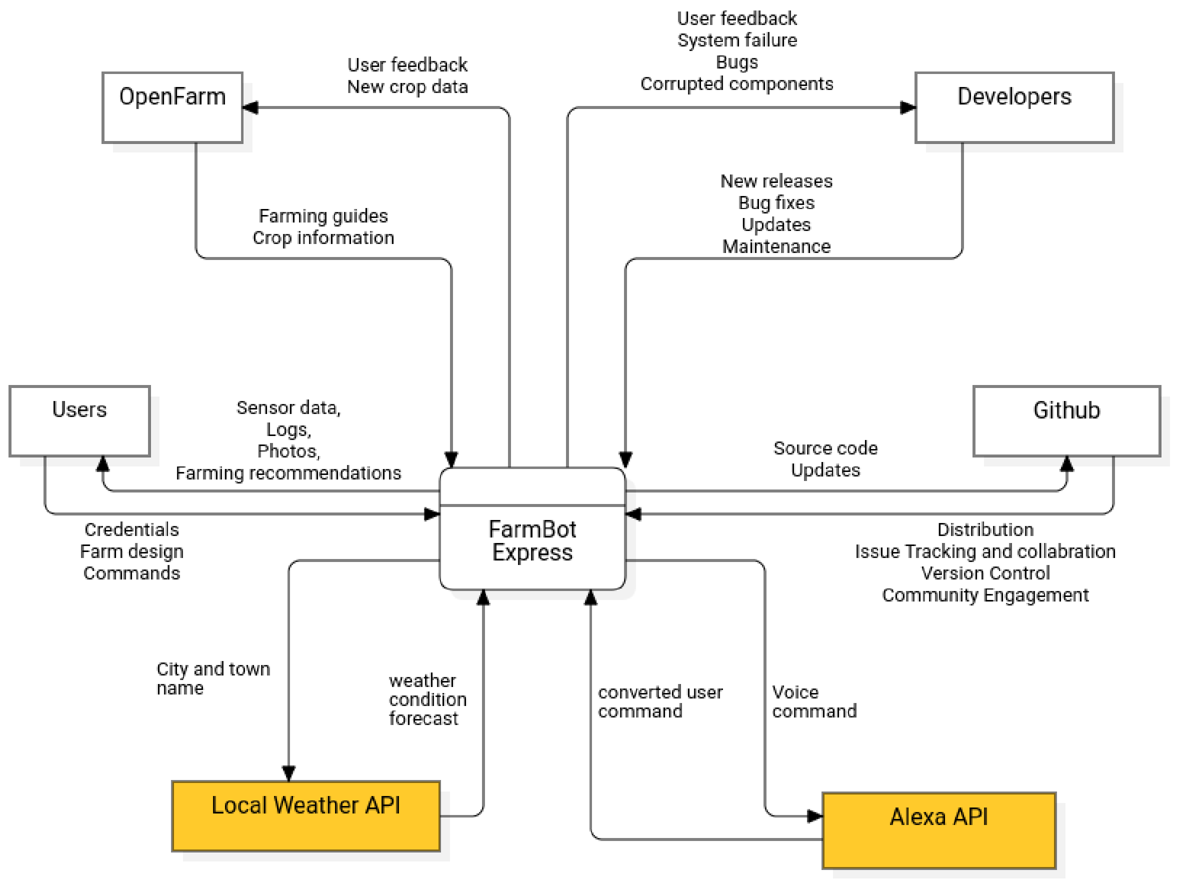


Figure 5.1: Suggested Context Diagram for FarmBot

The suggested context diagram for FarmBot is shown in Figure 5.1. The diagram shows the external entities that may interact with the system. It now includes [2] **Local Weather API** and [3] **Alexa API** as new external system entities. The **Local Weather API** is used to get the weather information of the FarmBot location. The **Alexa API** is used to control the FarmBot using voice commands.

5.1.3 External Interfaces

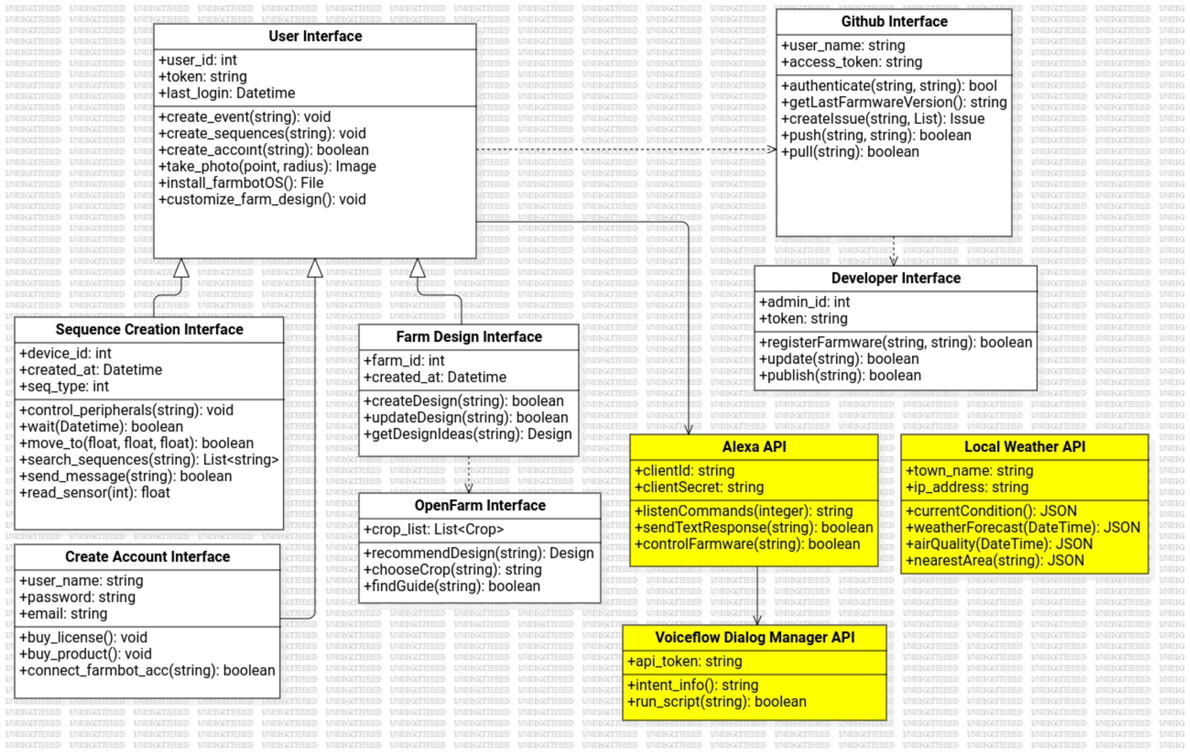

User Interface
+user_id: int
+token: string
+last_login: Datetime
+create_event(string): void
+create_sequences(string): void
+create_accoint(string): boolean
+take_photo(point, radius): Image
+install_farmbotOS(): File
+customize_farm_design(): void
Github Interface
+user_name: string
+access_token: string
+authenticate(string, string): bool
+getLastFarmwareVersion(): string
+createIssue(string, List): Issue
+push(string, string): boolean
+pull(string): boolean
Developer Interface
+admin_id: int
+token: string
+registerFarmware(string, string): boolean
+update(string): boolean
+publish(string): boolean
Sequence Creation Interface
+device_id: int
+created_at: Datetime
+seq_type: int
+control_peripherals(string): void
+wait(Datetime): boolean
+move_to(float, float, float): boolean
+search_sequences(string): List<string>
+send_message(string): boolean
+read_sensor(int): float
Farm Design Interface
+farm_id: int
+created_at: Datetime
+createDesign(string): boolean
+updateDesign(string): boolean
+getDesignIdeas(string): Design
OpenFarm Interface
+crop_list: List<Crop>
+recommendDesign(string): Design
+chooseCrop(string): string
+findGuide(string): boolean
Create Account Interface
+user_name: string
+password: string
+email: string
+buy_license(): void
+buy_product(): void
+connect_farmbot_acc(string): boolean
Alexa API
+clientId: string
+clientSecret: string
+listenCommands(integer): string
+sendTextResponse(string): boolean
+controlFarmware(string): boolean
Local Weather API
+town_name: string
+ip_address: string
+currentCondition(): JSON
+weatherForecast(DateTime): JSON
+airQuality(DateTime): JSON
+nearestArea(string): JSON
Voiceflow Dialog Manager API
+api_token: string
+intent_info(): string
+run_script(string): boolean

Figure 5.2: Suggested External Interfaces Class Diagram for FarmBot

The suggested external interfaces class diagram for FarmBot is shown in Figure 5.2. The diagram shows the external interfaces of the system. It now includes **Local Weather API** and **Alexa API** with **Voiceflow Dialog Manager API** as new external interfaces of the system.

- **Local Weather API**: The system uses this API to get the weather information of the FarmBot location. The acquired data shows that current temperature, humidity, and weather conditions are used to adjust many functionalities of the FarmBot, such as watering the plants, turning on the lights, etc. Furthermore, the system can also use this data to predict the weather conditions for the upcoming days and adjust the FarmBot's schedule accordingly.

- **Alexa API**: The system uses this API to control the FarmBot using voice commands. The users can interact with the FarmBot using voice commands through the Alexa API. The system can understand the voice commands and execute the corresponding actions, such as watering the plants, turning on the lights, etc. The vocal commands of the users go into Alexa API; then it is redirected to the Voiceflow Dialog Manager API to understand the commands and execute the corresponding actions.

- **Voiceflow Dialog Manager API**: The system uses this API to understand the users' voice commands. Internally, it runs a voice recognition algorithm that understands the commands and executes the corresponding actions. The commands are then redirected to the FarmBot system to execute the corresponding actions.

5.1.4 Interaction scenarios

The activity diagram for the voice command pipeline is shown in Figure 5.3. The diagram shows the interaction sequences taking place over the external interfaces for the suggested system. The diagram illustrates the process of storing a voice command in the system, recognizing the command using the Voiceflow Dialog Manager API, and executing the corresponding action in the FarmBot system. The interaction starts with the user issuing a voice command through the Alexa API. The command is then processed by the Voiceflow Dialog Manager API, which recognizes the command and sends it to the FarmBot system for execution.

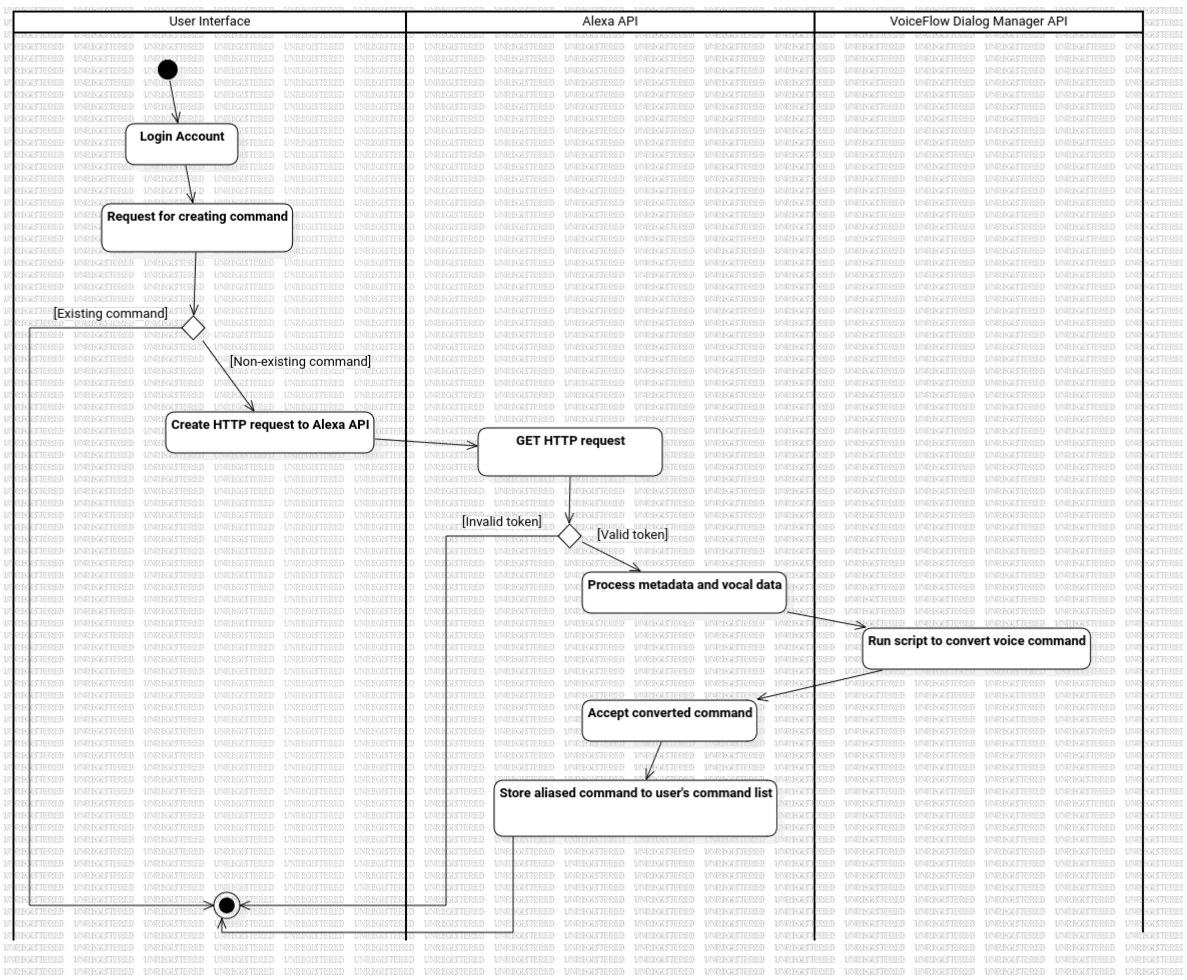

User Interface
Alexa API
VoiceFlow Dialog Manager API
Login Account
Request for creating command
[Existing command]
[Non-existing command]
Create HTTP request to Alexa API
GET HTTP request
[Invalid token]
[Valid token]
Process metadata and vocal data
Run script to convert voice command
Accept converted command
Store aliased command to user's command list

Figure 5.3: Activity Diagram for Voice Command Pipeline

5.2 Functional View

5.2.1 Stakeholders' uses of this view

The Functional View offers an overview of how architectural elements work together to provide the various functionalities of FarmBot. Below, you can find the FarmBot Stakeholders' uses of this view:

- **Developers**: The developers' main concern is the quality of design and the internal structure of the FarmBot system. The internal structure is crucial to meet the desired quality requirements such as availability, ability to scale, security during development. Also developers can use this view to see the external interfaces.

- **End Users** (Students, Researchers, Home Users, Artwork Creators): End users can use this view to learn the functionalities provided by FarmBot and how they are provided, what external interfaces are used.
- **Ministry of Agriculture**: Their use of this view is to have a more in-depth knowledge of the overall system, internal structure, external interfaces, and functionality for auditing.

5.2.2 Component Diagram

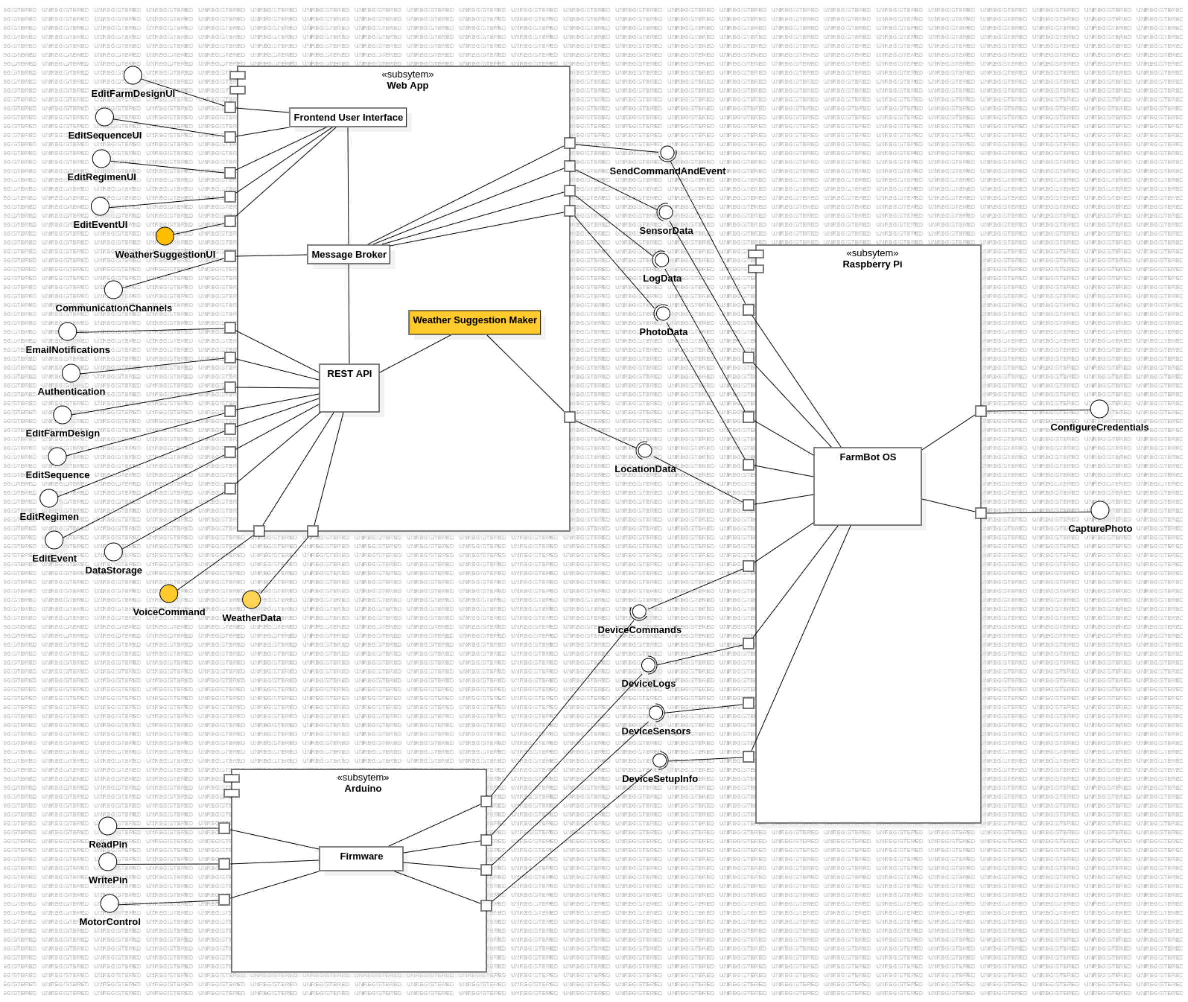


Figure 5.4: Suggested Component Diagram for FarmBot

The suggested component diagram for FarmBot extends the original one with three external interfaces, a part inside the web app which makes the suggestions based on weather and a interface between FarmBot device to gather the device location. The Frontend User Interface provides an external interface for the user to add voice commands. The REST API provides two external APIs for voice commands and weather data respectively. Voice commands are processed and extracted using external services. Therefore, the interaction between these services are provided through this API. Lastly, the weather data interface allows the web app to communicate with the external weather service providers.

5.2.3 Internal Interfaces

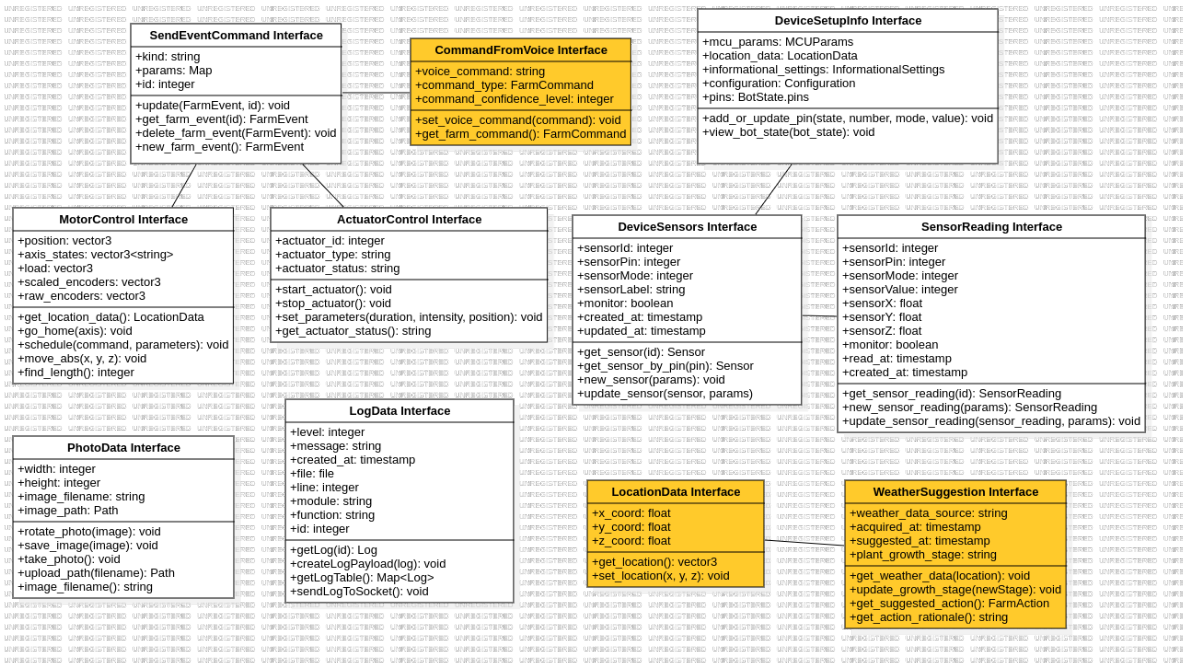

SendEventCommand Interface
+kind: string
+params: Map
+id: integer
+update(FarmEvent, id): void
+get_farm_event(id): FarmEvent
+delete_farm_event(FarmEvent): void
+new_farm_event(): FarmEvent
CommandFromVoice Interface
+voice_command: string
+command_type: FarmCommand
+command_confidence_level: integer
+set_voice_command(command): void
+get_farm_command(): FarmCommand
DeviceSetupInfo Interface
+mcu_params: MCUParams
+location_data: LocationData
+informational_settings: InformationalSettings
+configuration: Configuration
+pins: BotState.pins
+add_or_update_pin(state, number, mode, value): void
+view_bot_state(bot_state): void
MotorControl Interface
+position: vector3
+axis_states: vector3<string>
+load: vector3
+scaled_encoders: vector3
+raw_encoders: vector3
+get_location_data(): LocationData
+go_home(axis): void
+schedule(command, parameters): void
+move_abs(x, y, z): void
+find_length(): integer
ActuatorControl Interface
+actuator_id: integer
+actuator_type: string
+actuator_status: string
+start_actuator(): void
+stop_actuator(): void
+set_parameters(duration, intensity, position): void
+get_actuator_status(): string
DeviceSensors Interface
+sensorId: integer
+sensorPin: integer
+sensorMode: integer
+sensorLabel: string
+monitor: boolean
+created_at: timestamp
+updated_at: timestamp
+get_sensor(id): Sensor
+get_sensor_by_pin(pin): Sensor
+new_sensor(params): void
+update_sensor(sensor, params)
SensorReading Interface
+sensorId: integer
+sensorPin: integer
+sensorMode: integer
+sensorValue: integer
+sensorX: float
+sensorY: float
+sensorZ: float
+monitor: boolean
+read_at: timestamp
+created_at: timestamp
+get_sensor_reading(id): SensorReading
+new_sensor_reading(params): SensorReading
+update_sensor_reading(sensor_reading, params): void
LogData Interface
+level: integer
+message: string
+created_at: timestamp
+file: file
+line: integer
+module: string
+function: string
+id: integer
+getLog(id): Log
+createLogPayload(log): void
+getLogTable(): Map<Log>
+sendLogToSocket(): void
PhotoData Interface
+width: integer
+height: integer
+image_filename: string
+image_path: Path
+rotate_photo(image): void
+save_image(image): void
+take_photo(): void
+upload_path(filename): Path
+image_filename(): string
LocationData Interface
+x_coord: float
+y_coord: float
+z_coord: float
+get_location(): vector3
+set_location(x, y, z): void
WeatherSuggestion Interface
+weather_data_source: string
+acquired_at: timestamp
+suggested_at: timestamp
+plant_growth_stage: string
+get_weather_data(location): void
+update_growth_stage(newStage): void
+get_suggested_action(): FarmAction
+get_action_rationale(): string

Figure 5.5: Suggested Internal Interfaces Diagram for FarmBot

- **CommandFromVoice Interface:**
  - **voice_command**: The recognized text of voice command.

  - **command_type**: The type of the command such as WATERING, PLANTING, SCANNING_FOR_WEEDS
  - **command_confidence**: The level of confidence for the recognized text of voice command, acquired from external service Voiceflow.
  - **set_voice_command(command)**: Sets the recognized voice command and the type of it.
  - **get_farm_command()**: Returns the FarmCommand to be executed

- **LocationData Interface:**
  - **x_coord**: X coordinate of the FarmBot device
  - **y_coord**: Y coordinate of the FarmBot device
  - **z_coord**: Z coordinate of the FarmBot device
  - **get_location()**: Returns a three dimensional vector storing the coordinates of FarmBot device
  - **set_location(x, y, z)**: Updates the current coordinates of FarmBot device with the given parameters

- **WeatherSuggestion Interface:**
  - **weather_data_source**: Specifies the source of the weather data
  - **acquired_at**: The timestamp when latest weather data was acquired
  - **suggested_at**: The timestamp when suggestion was made
  - **plant_growth_stage**: Specifies the current growth stage of the plants being cultivated by the FarmBot (e.g., seedling, vegetative, flowering).
  - **get_weather_data(location)**: Acquires the current weather data through the REST API.
  - **update_growth_stage(newStage)**: Updates the growth stage according to the crop information

- **get_suggested_action()**: Returns a FarmAction that is computed by taking into account the current weather data and the growth stage of the crop
- **get_action_rationale()**: Returns a human friendly string of rationale, to be displayed to the user, explaining the suggested action

5.2.4 Interaction Patterns

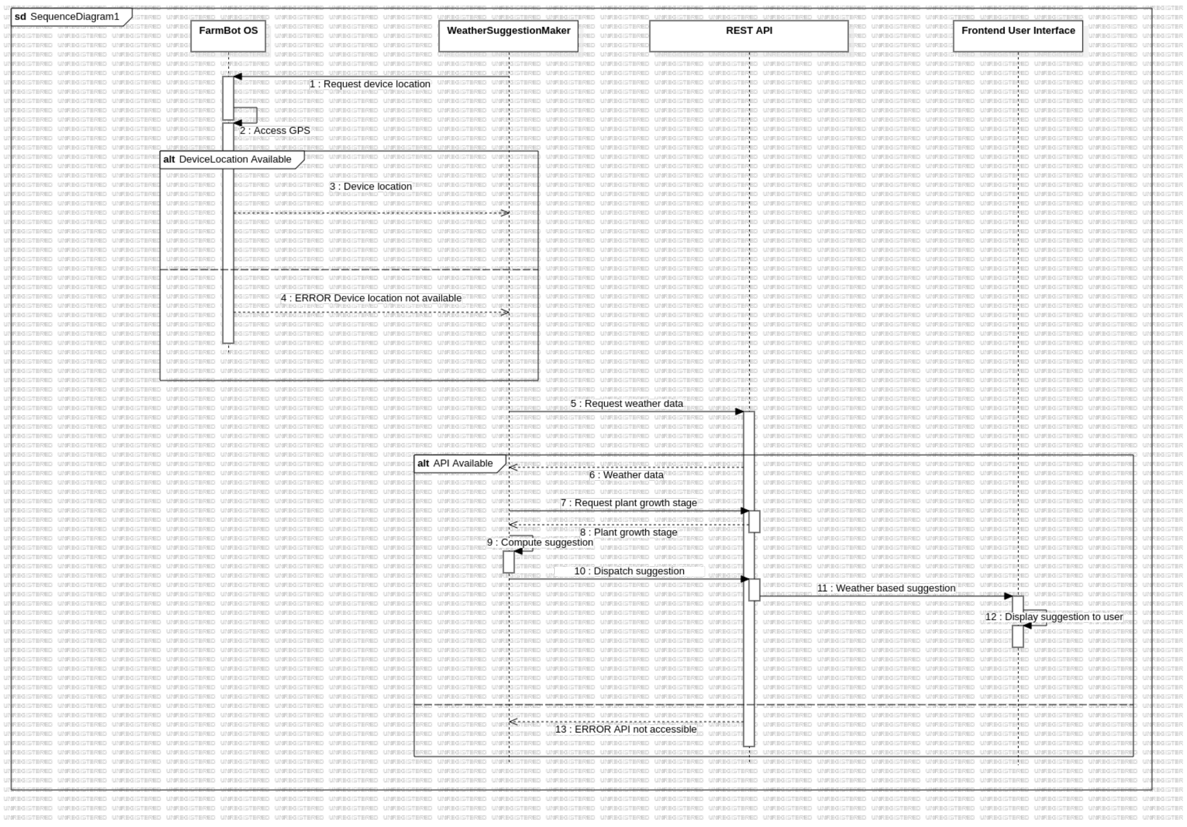


Figure 5.6: Sequence Diagram for Weather Based Suggestion of FarmBot

The sequence diagram in Figure 5.5, shows the weather based suggestion of FarmBot. WeatherSuggestionMaker, requests current location data from FarmBot OS. FarmBot OS, access the GPS device and returns the location data to the WeatherSuggestion-Maker or in case of failure returns an error. Consequently, WeatherSuggestionMaker requests weather data from the REST API using the location data it acquired. The REST API returns the weather data, after that another request for growth stage of plant is made. Using these data, WeatherSuggestionMaker computes a suggestion and dispatches it to the web app through the REST API. Frontend User Interface displays it to the user. In the case that the API is not available, an error is returned with the requests.

5.3 Information View

5.3.1 Stakeholders' uses of this view

The key use of the **Information View** for the stakeholders of FarmBot is to ensure that the data requirements are correctly understood and that the data is stored and handled correctly. The different stakeholders' uses of this view are as follows:

- **End users:** The end users primarily interact with the information view to understand the data being collected by FarmBot and how it relates to their gardening activities. They may use this view to monitor plant health, track growth progress, and receive alerts or recommendations.

- **Developers:** Developers use the information view to design and implement features related to data collection, storage, and processing. They ensure that the data is collected efficiently, stored securely, and can be accessed and manipulated as needed for various functionalities.

- **Ministry of Agriculture:** The Ministry of Agriculture utilizes the information view to oversee and regulate the use of FarmBot data for agricultural purposes. They may use this view to monitor farming practices, analyze trends, and make informed decisions regarding agricultural policies and practices.

5.3.2 Database Class Diagram

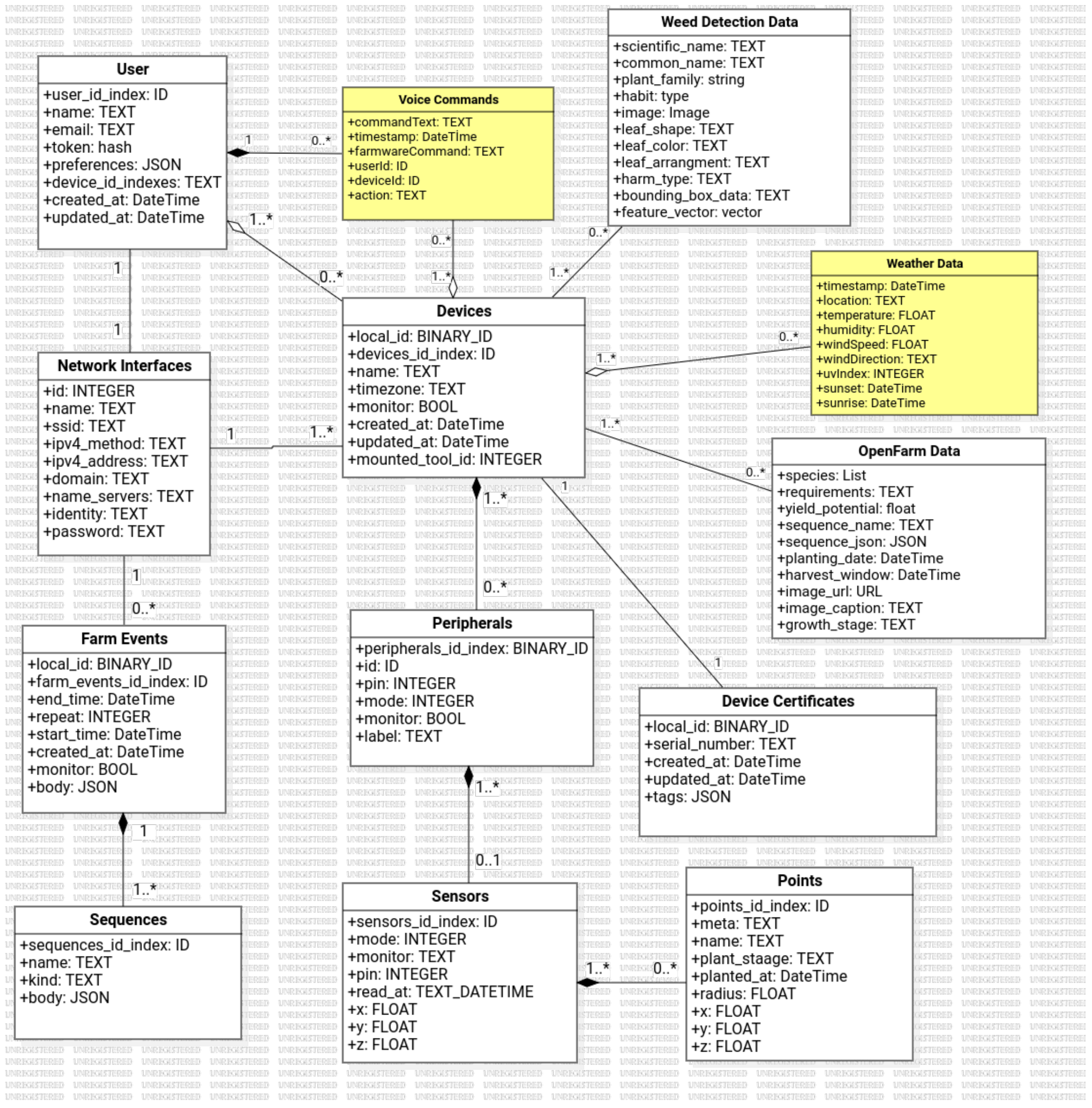


Figure 5.7: Suggested Database Class Diagram for FarmBot

The suggested database class diagram for FarmBot is shown in Figure 5.7. The diagram shows the key database objects of the system. It now includes **Weather Data** and **Voice Command** as new database objects of the system.

- **Weather Data:** This object stores the weather information of the FarmBot location. Using the location information, the interface fetches the current temperature, humidity, and weather conditions from the Local Weather API. The related data is also stored in this object for future use, such as predicting the weather conditions for the upcoming days and adjusting the FarmBot's schedule accordingly.

- **Voice Command:** This object stores the voice commands of the users processed by the VoiceFlow API. This table has columns for storing voice command data, such as command text and the corresponding action to be executed by the FarmBot system. The voice commands are used to control the FarmBot using voice commands through the Alexa API.

5.3.3 Operations on Data

Operation	**Description**	**Object**
CreateWeatherData	Creates a new weather data object	Weather Data
ReadWeatherData	Reads the weather data of the location	Weather Data
UpdateWeatherData	Updates the weather data of the location	Weather Data
DeleteWeatherData	Deletes the weather data of the location	Weather Data
CreateVoiceCommand	Creates a new voice command object	Voice Command
ReadVoiceCommand	Reads the voice command data	Voice Command
UpdateVoiceCommand	Updates the voice command data	Voice Command
DeleteVoiceCommand	Deletes the voice command data	Voice Command

Table 5.1: CRUD Operations on the Suggested System

Some of the operations on data for FarmBot are listed in Table 5.1. These operations include CRUD (Create, Read, Update, Delete) operations for the Weather Data and Voice Command objects. The operations are used to create, read, update, and delete the system's weather data and voice command data. The operations are related to other objects as mentioned in Table 4.1.

5.4 Deployment View

5.4.1 Stakeholders' uses of this view

The stakeholders uses of the Deployment view are described as follows:

- Developers: They use this view to better understand how they should orchestrate their development environment, what should their coverage be for deployment.
- End Users: They use this view to learn what tools, software or applications they need to get the system up and running.
- Ministry of Agriculture: They use this view to better understand dependencies, relations and interactions in the system. It can offer a inside view to how the system is orchestrated, aiding them in auditing.

5.4.2 Deployment Diagram

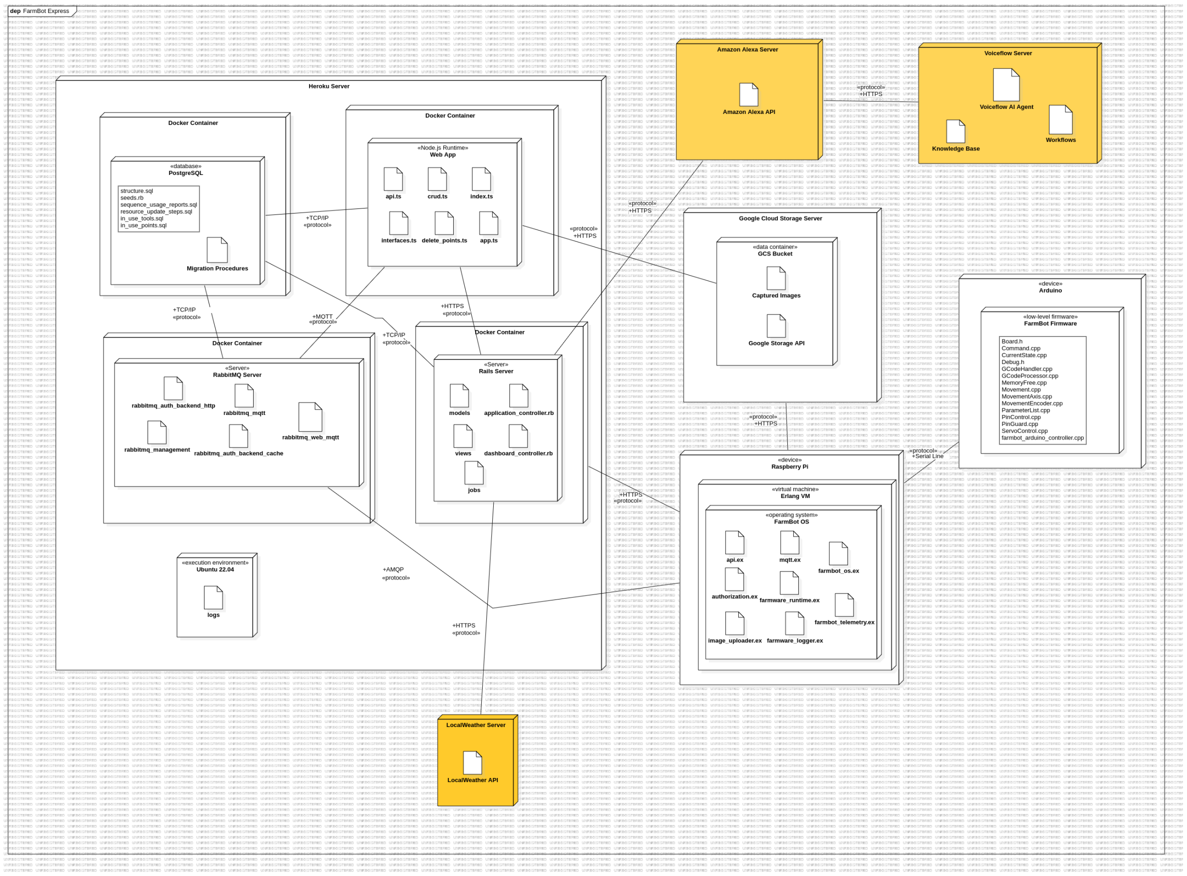


Figure 5.8: Suggested Deployment Diagram for FarmBot

The deployment diagram for the suggestions extends the original deployment diagram with three nodes. These are the Amazon Alexa servers used for capturing the voice and the Voiceflow servers for processing the voice to acquire the command. For the weather feature, a node for the LocalWeather Servers is added to the diagram. This node holds the LocalWeather API artifact. The REST API interacts with this server to acquire the weather data for the day and use it to make suggestions based on it.

5.5 Design Rationale

- **Context View:** The suggested context view includes the Local Weather API and Alexa API as new external system entities.
  - Local Weather API: Integrating the Local Weather API into the context view allows the FarmBot system to adapt its operations based on real-time weather conditions. This improves automation by enabling the FarmBot to adjust watering schedules, activate frost protection measures, or optimize growth lamp usage based on temperature, humidity, and precipitation data.
  - Alexa API: The inclusion of the Alexa API in the context view facilitates voice-controlled interaction with the FarmBot. This enhances user experience by providing a more intuitive and hands-free way to control the system. Users can issue commands to perform actions like starting/stopping irrigation, adjusting nutrient delivery, or receiving status updates using simple voice prompts.
- **Functional View:**
  - Weather Suggestions: An internal interface for the location data is used for the operation of the weather based suggestions internal interface. This suggestion interface is used for acquiring weather data from REST API and computing a farm action to be taken based on the current status. Seperation of concerns is achieved and interface segregation leads to better maintainability.
  - Voice Commands: A seperate internal interface for commands is added following the interface segregation principle. This interface keeps additional data for the captured voice and recognized command. The confidence score can be used to decide whether executing or not.
- **Information View:**

- Weather Data: Integrating the Local Weather API expands the information view by providing real-time weather data alongside existing FarmBot sensor readings. This comprehensive view allows for a more holistic understanding of the farm environment. Users can easily correlate sensor data with weather conditions to identify potential issues or optimize growing conditions. Database stores the necessary information that users can access and analyze the weather data.
- Voice Command: The inclusion of the Alexa API in the information view offers the ability to display voice command options directly within the FarmBot interface. To support user customization, the FarmBot system will need to store data related to user-defined aliases, favorite commands, and potentially custom command sets. The information view ensures that the user can easily understand the stored voice commands and their corresponding actions.

• **Deployment View:**

- LocalWeather API: This is an external service provided by an external service provider. Therefore, the app must interact with it over the network. This interaction follows a request-response pattern since for each day, the web-app makes a request to acquire the weather data and gets the weather data in response. Therefore, the app interacts with the Local Weather API through the Rails server REST API.
- Voice Command: The voice commands require two external services, Amazon Alexa and Voiceflow. By using these external services, the complexity of FarmBot project is minimized. It provides better maintainability. In addition, since these services are not on FarmBot servers, they cost less to scale. Providing better availability. Commands are acquired through the REST API and sent over to the physical FarmBot device via the MQTT.