Software Requirements Specification

farm.bot

Group 17

Ömer Kılınç

2448603

Table of Contents

[**1. Introduction** 5](#_Toc162453508)

[**1.1. Purpose of the System** 5](#_Toc162453509)

[**1.2. Scope** 5](#_Toc162453510)

[**1.3. System Overview** 5](#_Toc162453511)

[**1.3.1. System Perspective** 5](#_Toc162453512)

[**1.3.1.1. System Interfaces** 5](#_Toc162453513)

[**1.3.1.2. User Interfaces** 5](#_Toc162453514)

[**1.3.1.3. Hardware Interfaces** 5](#_Toc162453515)

[**1.3.1.4. Software Interfaces** 5](#_Toc162453516)

[**1.3.1.5. Communication Interfaces** 5](#_Toc162453517)

[**1.3.1.6. Memory Constraints** 5](#_Toc162453518)

[**1.3.1.7. Operations** 5](#_Toc162453519)

[**1.3.2. System Functions** 5](#_Toc162453520)

[**1.3.3. Stakeholder Characteristics** 5](#_Toc162453521)

[**1.3.4. Limitations** 5](#_Toc162453522)

[**1.4. Definitions** 5](#_Toc162453523)

[**2. References** 5](#_Toc162453524)

[**3. Specific Requirements** 5](#_Toc162453525)

[**3.1. External Interfaces** 5](#_Toc162453526)

[**3.2. Functions** 5](#_Toc162453527)

[**3.3. Logical Database Requirements** 5](#_Toc162453528)

[**3.4. Design Constraints** 5](#_Toc162453529)

[**3.5. System Attributes** 5](#_Toc162453530)

[**3.6. Supporting Information** 5](#_Toc162453531)

[**4. Suggestions to Improve the Existing System** 6](#_Toc162453532)

[**4.1. System Perspective** 6](#_Toc162453533)

[**4.2. External Interfaces** 6](#_Toc162453534)

[**4.3. Functions** 6](#_Toc162453535)

[**4.4. Logical Database Requirements** 6](#_Toc162453536)

[**4.5. Design Constraints** 6](#_Toc162453537)

[**4.6. System Attributes** 6](#_Toc162453538)

[**4.7. Supporting Information** 6](#_Toc162453539)

List of Figures

Figure 1: Context Model

Figure 2: farm.bot Web Application Registration Interface

Figure 3: farm.bot Web Application Tutorial I

Figure 4: farm.bot Web Application Tutorial Instructions II

Figure 5: farm.bot Web Application Jobs view

Figure 6: farm.bot Web Application Logs view

Figure 7: farm.bot Web Application Farm Design Interface

Figure 8: Farmduino and Microcomputer schema

Figure 9: Nema 17 stepper motor

Figure 10: Farmbot Genesis

Figure 11: Farmbot Software High Level Overview

Figure 12: External Interfaces Class Diagram

Figure 13: Use Case Model

Figure 14: Sequence Diagram for “Place Crops” use case

Figure 15: Activity Diagram for “Establish Connection Between Systems” use case

Figure 16: Logical Database Requirements Class Diagram

Figure 17: Context Diagram (No change)

Figure 18: External Interfaces Class Diagram for improved system

Figure 19: Use Case Model for improved system

Figure 20: Logical Database Requirements Class Diagram for improved system

List of Tables

Table 1: System Functions and Descriptions

Table 2: Use Case “Set up hardware”

Table 3: Use Case “Add features”

Table 4: Use Case “Listen to feedback”

Table 5: Use Case “Select soil”

Table 6: Use Case “Place crops in virtual farm”

Table 7: Use Case “Set up Schedules”

Table 8: Use Case “Provide feedback”

Table 9: Use Case “Establish connection between systems”

Table 10: Use Case “Provide crops and water”

Table 11: Use Case “Provide information”

Table 12: Use Case for improved system “Insufficient Water/Seed Alert”

Table 13: Use Case for improved system “Check Plant Development”

Table 14: Use Case for improved system “Unstable Power Supply Notification”

Table 15: Use Case for improved system “Inject seed from smartphone”

Revision History

|  |  |
| --- | --- |
| **Author** | **Date** |
| Ömer Kılınç | 06/04/2024 |

# **1. Introduction**

This Software Requirements Specification (SRS) document is prepared for the purpose of providing a clear and structured information about the “farm.bot” open-source project which is an automation software composed of “farm.bot web application”, “farm.bot OS” and “farm.bot Arduino firmware” softwares.

## **1.1. Purpose of the System**

farm.bot project seeks to automate and optimize farming operations to a more efficient and sustainable level. The main purpose of this system is to ease the process of planting, nurturing and harvesting crops for individual gardeners and small-scale famers via an easy-to-use web interface, structured hardware design to reduce labor costs and up to date crop database to increase efficiency.

## **1.2. Scope**

The scope of farm.bot system includes the development and arrangement of an automated robot capable of achieving multiple farming tasks, from seed injection and watering through monitoring crop health.

**Key Features and Capabilities:**

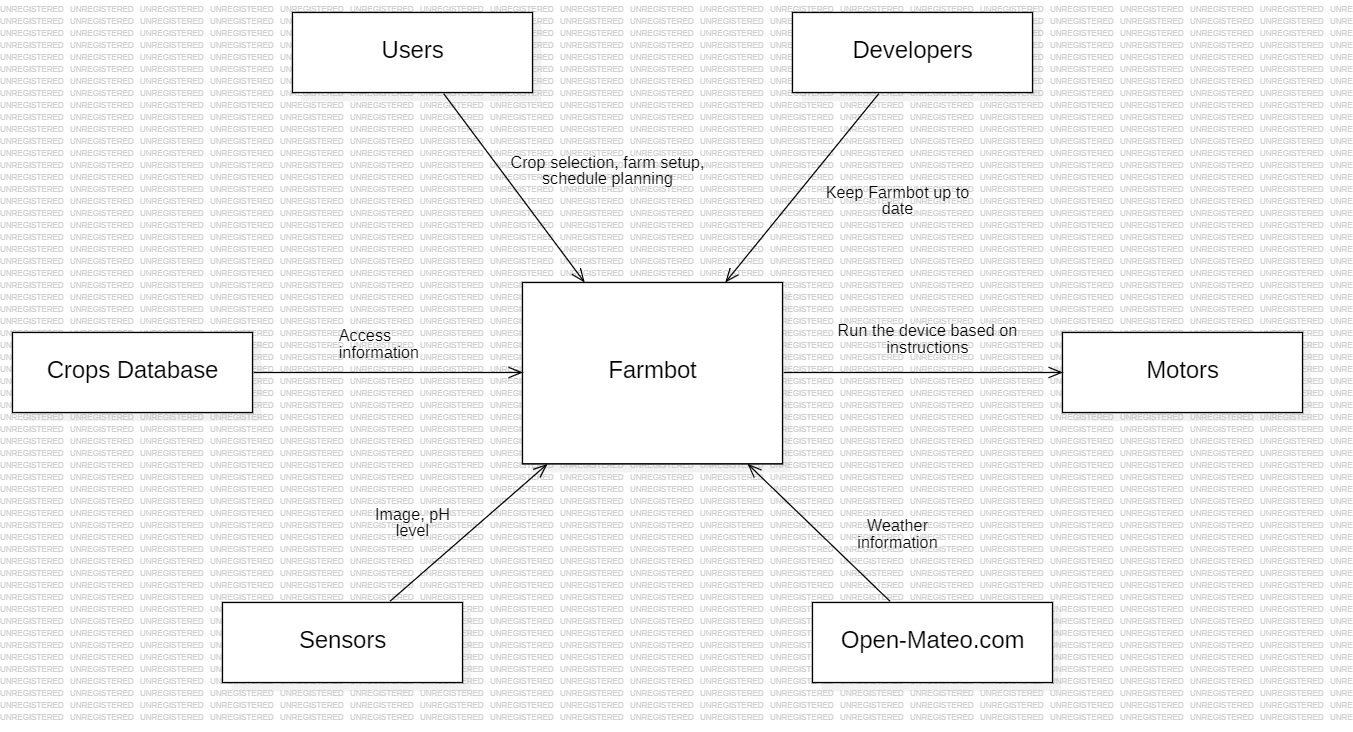
* Farming setup and customization
* Manual movement control
* Seed injection
* Regiment creation
* Weed detection
* Soil moisture measuring and watering
* Environmental monitoring via sensors

**1.3. System Overview**

### **1.3.1. System Perspective**

Farm.bot is a stand-alone system that interacts with external services such as a crop database called OpenFarm.cc and Farm.bot cloud services that uses google cloud storage. Farm.bot utilizes Raspberry Pi as its single board computer to run Farm.bot OS and Arduino as its microcontroller to interact with its sensors to offer a semi-autonomous device.

User interacts with Farm.bot Web Application to create the farm they desire and sends instructions to the controller. Farm.bot OS then follows the instructions according to the crop information from its database. System’s main hardware “Farmbot Genesis” injects seeds and waters them.



**Figure 1:** Context Model

#### **1.3.1.1. System Interfaces**

**External Data Interfaces:**

Farm.bot utilizes external data interfaces to manage the user’s local farm and to offer the best actions. It uses a crop database to gather information to utilize operations.

**Sensor Management Interface:**

This system allows Farm.bot to take actions according to the information present in the local farm such as the crops location, weeds, pH levels of the soil, age of plants and more.

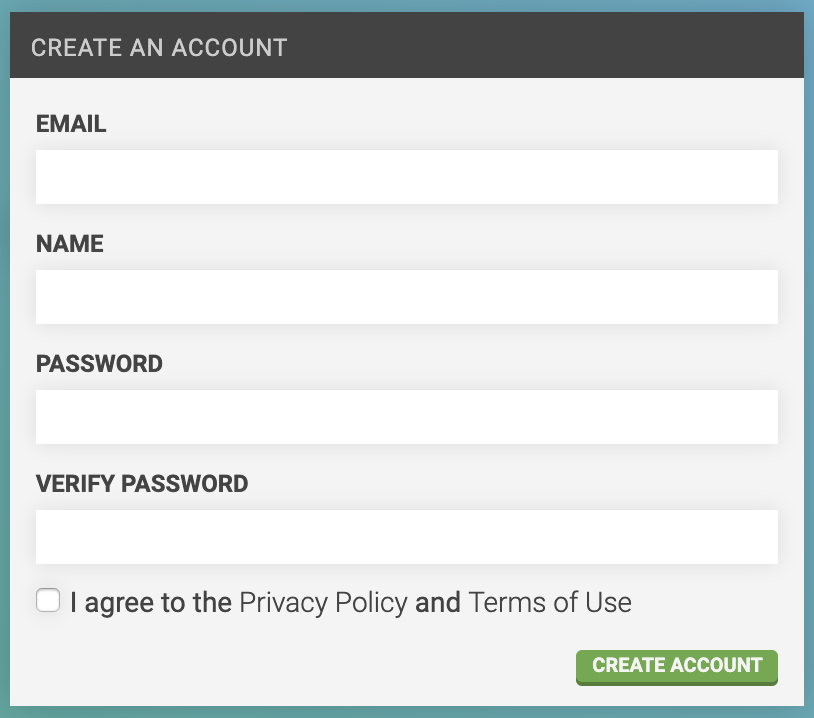
**Farm.bot Web Application:**

This application allows users to design and manipulate their farm and set up schedules. Then selected preferences directed to the Farm.bot OS to start the process.

#### **1.3.1.2. User Interfaces**

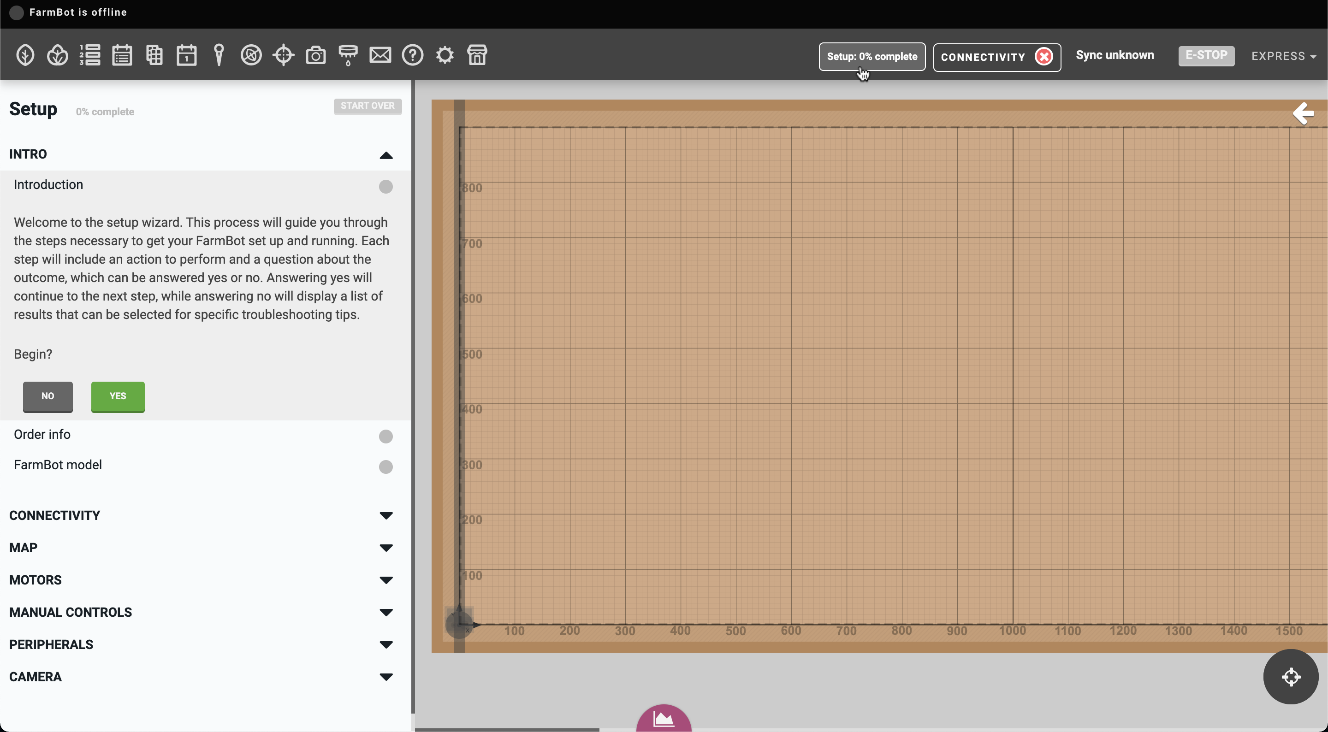
The user interface is handled mostly within Farm.bot Web Application, which is an easy-to-use drag and drop based system, illustrating the basic map of the garden with progress bars and various settings. Farmbot web application is available for any device with browser and internet access.

**Registration Interface:** To manage a farm using Farm.bot, users first need to create an account. Users must simply enter their email address, their name and then set up a password to get started.

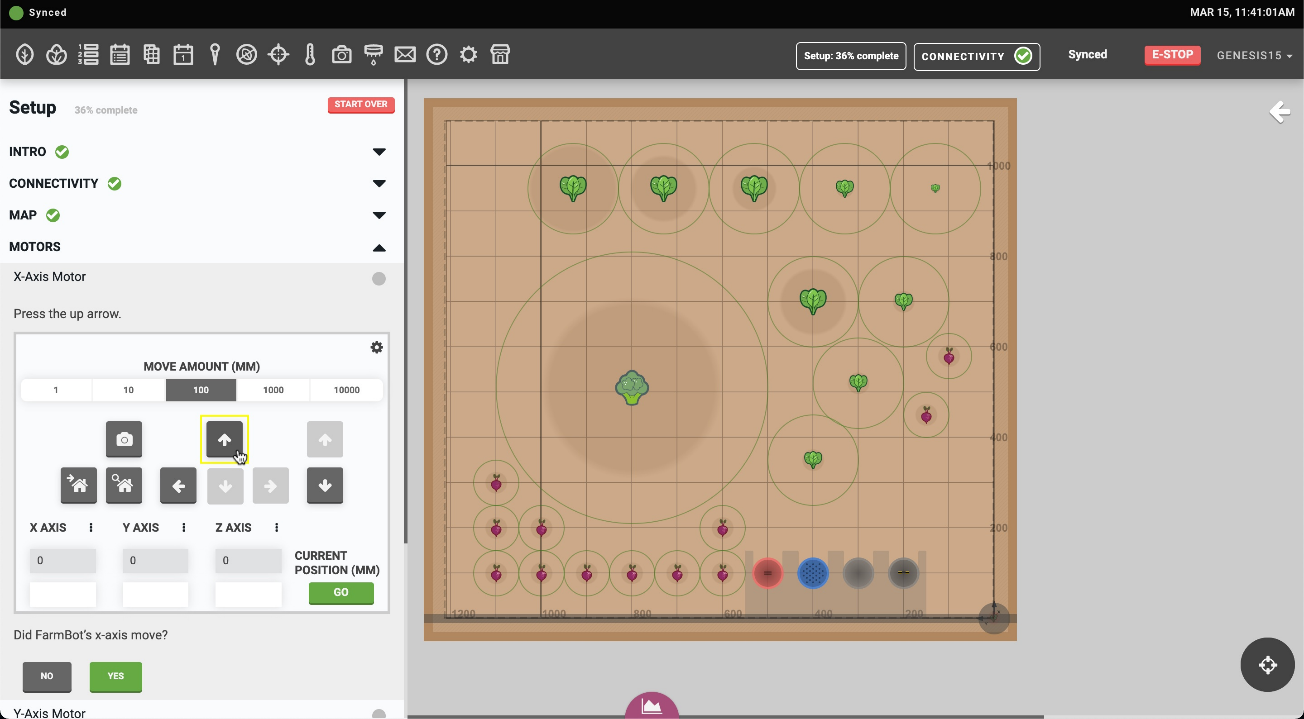


**Figure 2:** farm.bot Web Application Registration Interface

Users then should follow the easy instructions to get started.

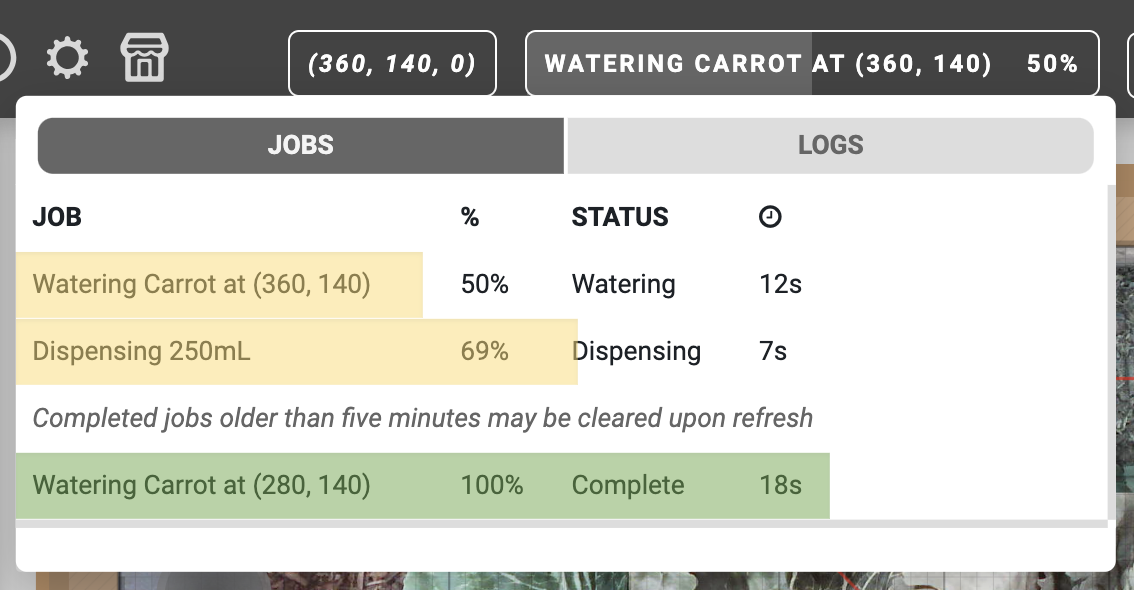


**Figure 3:** farm.bot Web Application Tutorial I



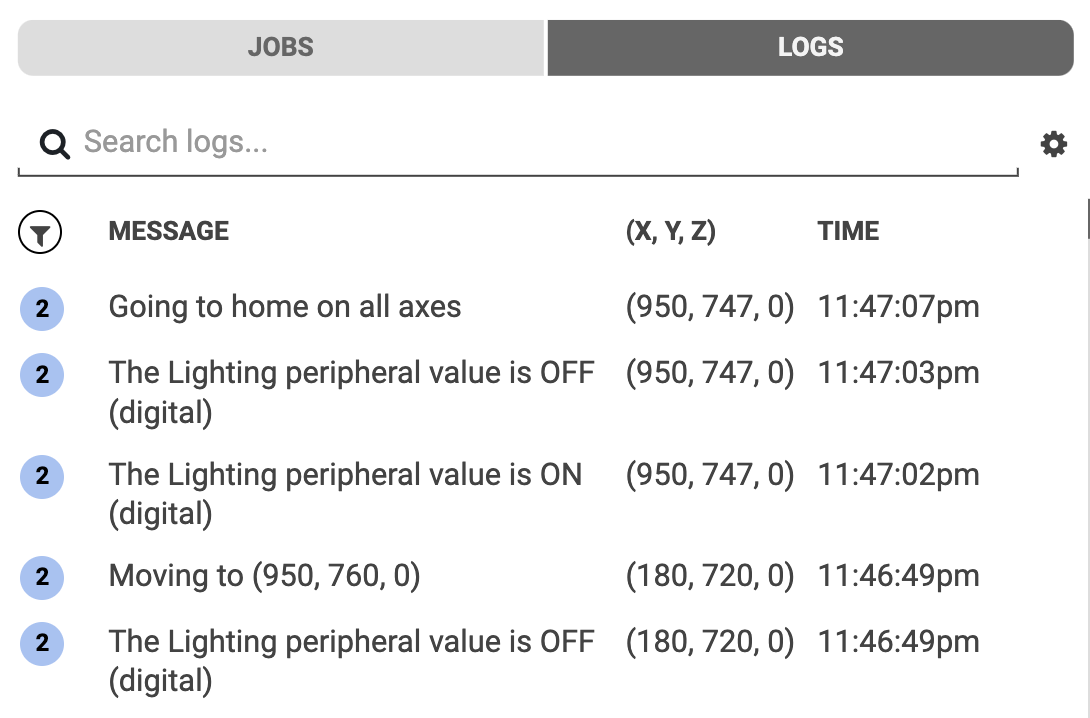
**Figure 4:** farm.bot Web Application Tutorial Instructions II

#### **Jobs and Logs Interface:** Jobs are long-running actions that are tasked by the users. Jobs and Logs Popup in Farm.bot Web Application allows users to check the status and duration of the jobs given.



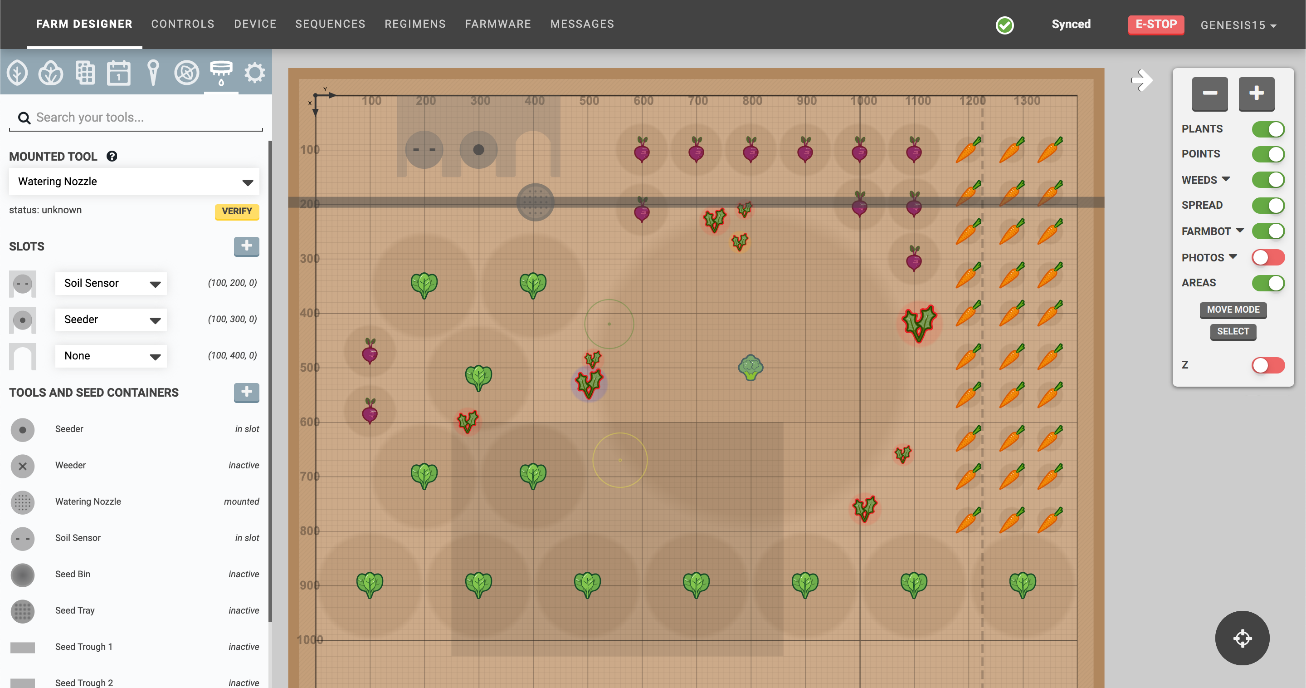
**Figure 5:** farm.bot Web Application Jobs view

Logs view functions like Jobs view, but they are low level single operations that are not intended to be used by general user but available as an option to debug or check individual processes.



**Figure 6:** farm.bot Web Application Logs view

**Designer Interface:** This interface will be the most used part of user experience. Users interact with the virtual map to manage their crops type and location. Users can also edit soil preferences, its moisture, groups of the crops, weeds and many more details.



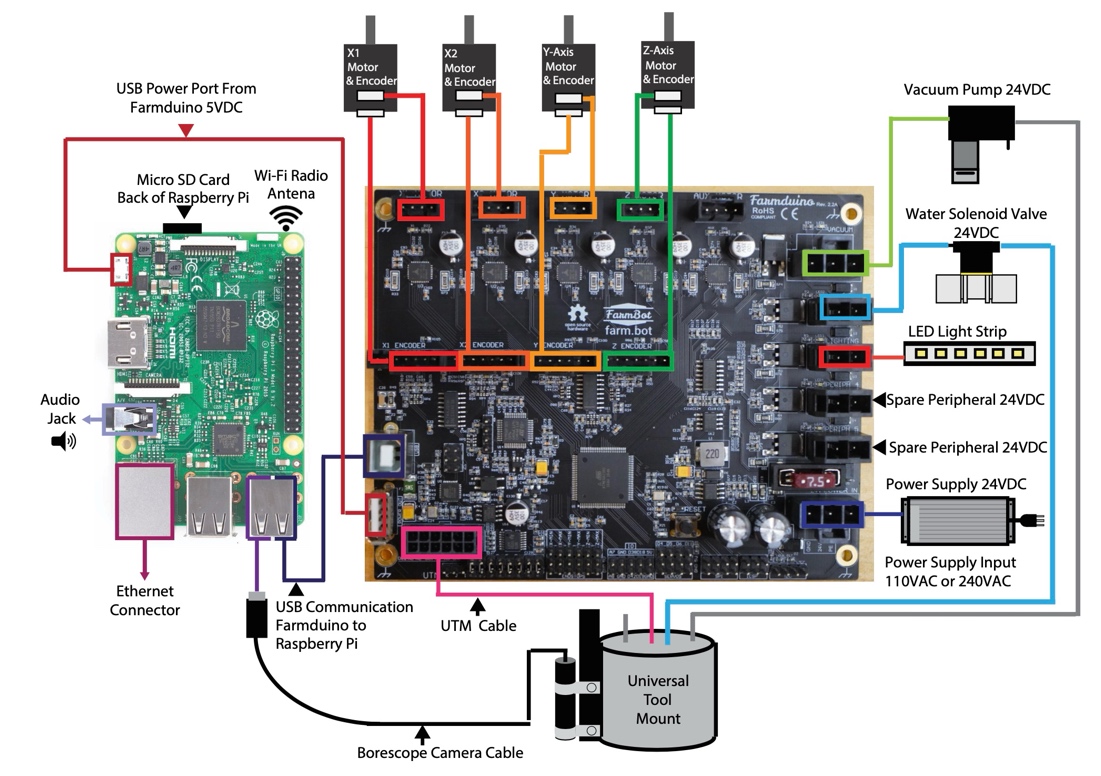
**Figure 7:** farm.bot Web Application Farm Design Interface

#### **1.3.1.3. Hardware Interfaces**

farm.bot is a hardware dependent project. After users set up the farm they want to use in the application, farm.bot then transfers the information gathered to its hardware components.

* **Raspberry PI Controller:** This controller runs on farm.bot OS. It executes scheduled events, uploads logs and sensor data. OS is connected to the microcontroller Arduino to send commands.
* **Arduino Microcontroller:** Arduino or farmduino is responsible for operating the motors, sensors, tools, and other electronics. It takes commands from Raspberry Pi and executes them.

farm.bot also requires a medium to run farm.bot Web Application on. It can be many available devices in our current times including phones, tablets, PCs, or laptops. It can be easily set up and it is then ready to use.



**Figure 8:** Farmduino and Microcomputer schema



**Figure 9:** Nema 17 stepper motor



**Figure 10:** Farmbot Genesis

#### **1.3.1.4. Software Interfaces**

farm.bot executes its functions on Raspberry Pi running on farmbot OS. Raspberry Pi communicates in both directions with arduino microcontroller to get sensor information and control the movement via stepper motors. On the other side, Raspberry Pi gets instructions from user via farmbot web application interface which is available on any OS with a web browser and internet connection.

#### **1.3.1.5. Communication Interfaces**

farm.bot relies on external data. That means farm.bot requires internet connection to perform accurately. Data transit is encrypted via SSL and HTTPS. This data is then passed through MQTT gateway to safely transmit the message to the hardware. Raspberry Pi is connected to Arduino Firmware by USB connection or serial connection to send G and F code commands.

#### **1.3.1.6. Memory Constraints**

There are no memory constraints to run farm.bot.

#### **1.3.1.7. Operations**

#### **User Operations:**

### Login

### Setup virtual farm

### Setup hardware

### Connect the systems accordingly

**Crops Database Operations:**

* Provide information to farm.bot

**Sensor Operations:**

* Receive information on physical farm
* Send data to Arduino

**Open-Mateo.com Operations:**

* Regularly update recent weather information in detail
* Provide information to farm.bot

**Motor Operations:**

* Move the watering head
* Help sensors to gather information

**Developer Operations:**

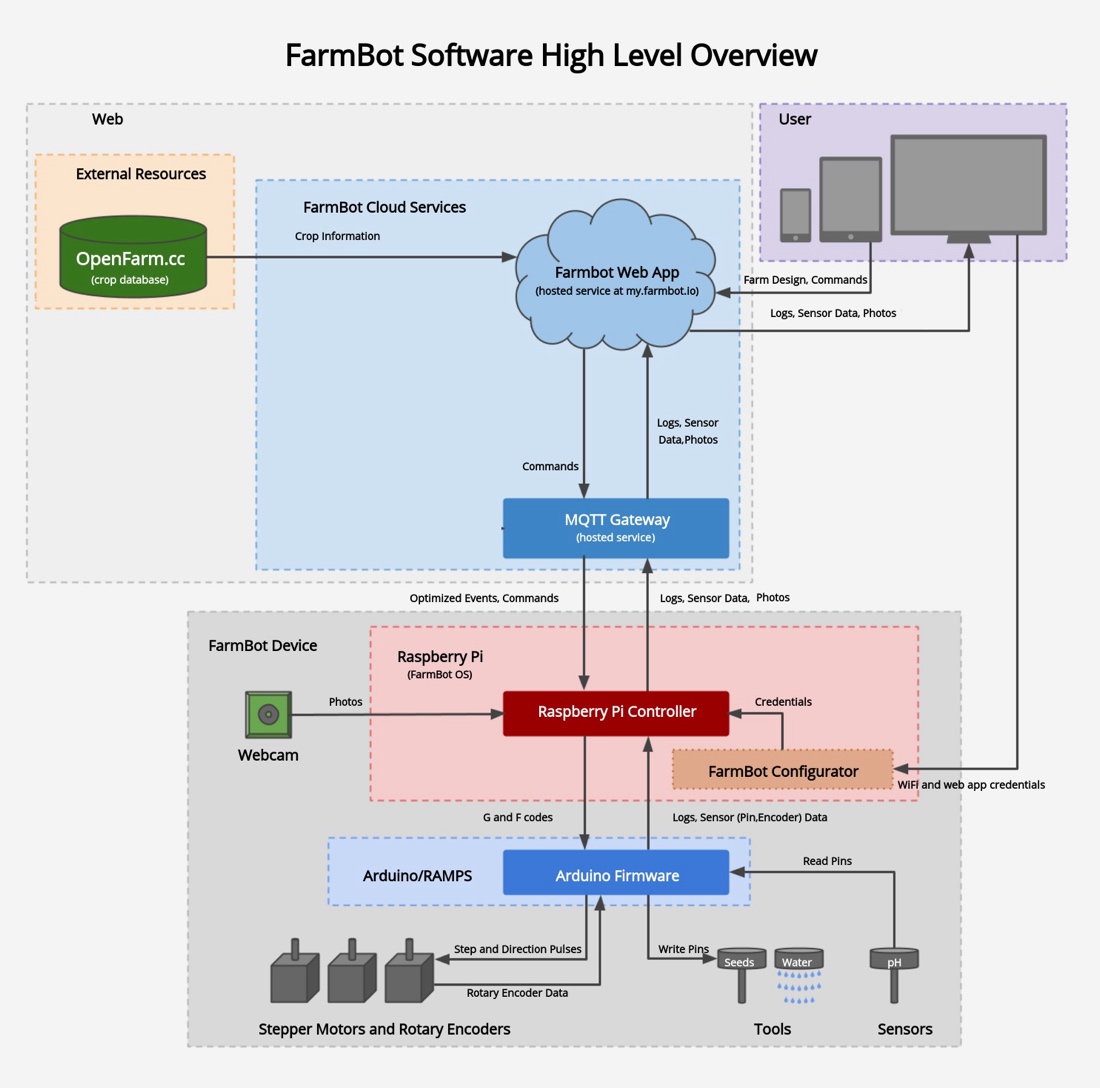
* Listen to user feedback
* Keep farm.bot up to date and resolve bugs
* Provide open source for users

### **1.3.2. System Functions**

The FarmBot system is designed to automate and optimize the agricultural process from planting to monitoring. This section outlines the key functionalities of how each functionality of farmbot system contributes to achievement of creating an accessible, open-source, small scale farming solutions.

|  |  |
| --- | --- |
| **Function** | Description |
| **Moisture Levels and Watering** | The system adjusts watering amount depending on the current moisture and whether information |
| **Growth Tracking and Logging** | Stores plant growth data, sensor data and system operations to help users analyze their farming strategies. |
| **Manual Device Control** | Through the web application, users can manually control the FarmBot hardware motion. |
| **Database Integration** | Enables the OpenFarm database access to a large storage of crop information to support users with useful, related information |
| **Crop Planning** | Users can graphically design their farming area on the web application, determining the most effective area to plant various crops. |
| **Data Mapping** | software can create data maps that the farmer and the decision support  system can be used to make smarter, data-driven decisions |
| **Precision Weeding** | Make use of computer vision to differentiate crops from weeds. FarmBot detects weeds as targets and uproots them without disturbing nearby plants, relies on pre-set or user-defined criteria for weed identification. |
| **Weather Integration** | Adjusts operations based on local weather data. |
| **Growth and Health Monitoring** | Keeps track of plant growth progress and health status for each individual plant by logging sensor readings and photographing changes over time. |

**Table 1:** System Functions And Descriptions



**Figure 11:** Farmbot Software High Level Overview

### **1.3.3. Stakeholder Characteristics**

**Gardeners Hobbyists and Families:** Individuals or families whose focus is to use an automated system to keep their small-scale farm or garden in good shape and need a user-friendly interface and easy setup for automation.

**Entrepreneurs and Tech** **Hobbyists:** Individuals who are interested in developing or integrating additional features to the existing farmbot system and become a part of this project and/or tinker with it.

### **1.3.4. Limitations**

The Farmbot system faces several limitations that stakeholders should take into account when planning for implementation or integration. These limitations mainly stem from the nature of open source projects and hardware resistancy to physical factors.

**Software Integration**: FarmBot software integration with external farm management systems and third-party applications can require extensive customization or more time fixing the compatibility problems.

**Hardware Compatibility**: The system may face limitations in scaling or adapting to new agricultural technologies without significant hardware upgrades

**Software Maintenance**: Regular maintenance is required to ensure FarmBot's operational efficiency, including regular checks of the system notifications and software updates, and hardware checks. May be challenging for regular users.

**Hardware Maintenance**: System’s hardware components should be regularly checked and cleaned for optimal efficiency in long term usage.

**Durability:** System's reliance on internet connectivity and cloud services and a stable power source requirement may be affected by geography, low quality infrastructure and whether conditions

**Climate and Weather Resistance**: The electronic and mechanical components have limited resistance to extreme weather conditions, requiring additional protection or isolated spaces.

**Soil and Terrain Variability**: FarmBot's efficiency can be influenced by soil composition and different types of terrains. May require additional calibration or hardware modifications to the installation site.

**Data Privacy and Security**: Given the system's reliance on internet connectivity and cloud services, ensuring the privacy and security of user data is important. Catching up with global data protection standards is necessary to protect user information.

## **1.4. Definitions**

|  |  |
| --- | --- |
| **Term** | **Definition** |
| **CNC** | **Computer Numerical Control**  A method to control machine tools with software and an embedded microcomputer. |
| **MQTT** | **Message Queuing Telemetry Transport**  A lightweight messaging protocol designed for small sensors and mobile devices, bridging efficient communication between hardware and software components |
| **API** | **Application Programming Interface**  A set of libraries, protocols, and tools for building software and applications. |
| **G-Code** | A language used to tell computerized machine tools how to perform functions. |
| **IoT** | **Internet of Things**  The network of hardware that uses sensors and APIs to connect and exchange data over the Internet |

## **2. References**

ISO/IEC/IEEE 29148 International Standard Systems and software engineering

Life cycle processes Requirements engineering (Second Edition-2018)

Farmbot Whitepaper, Rory Landon Anderson, 19 September 2013

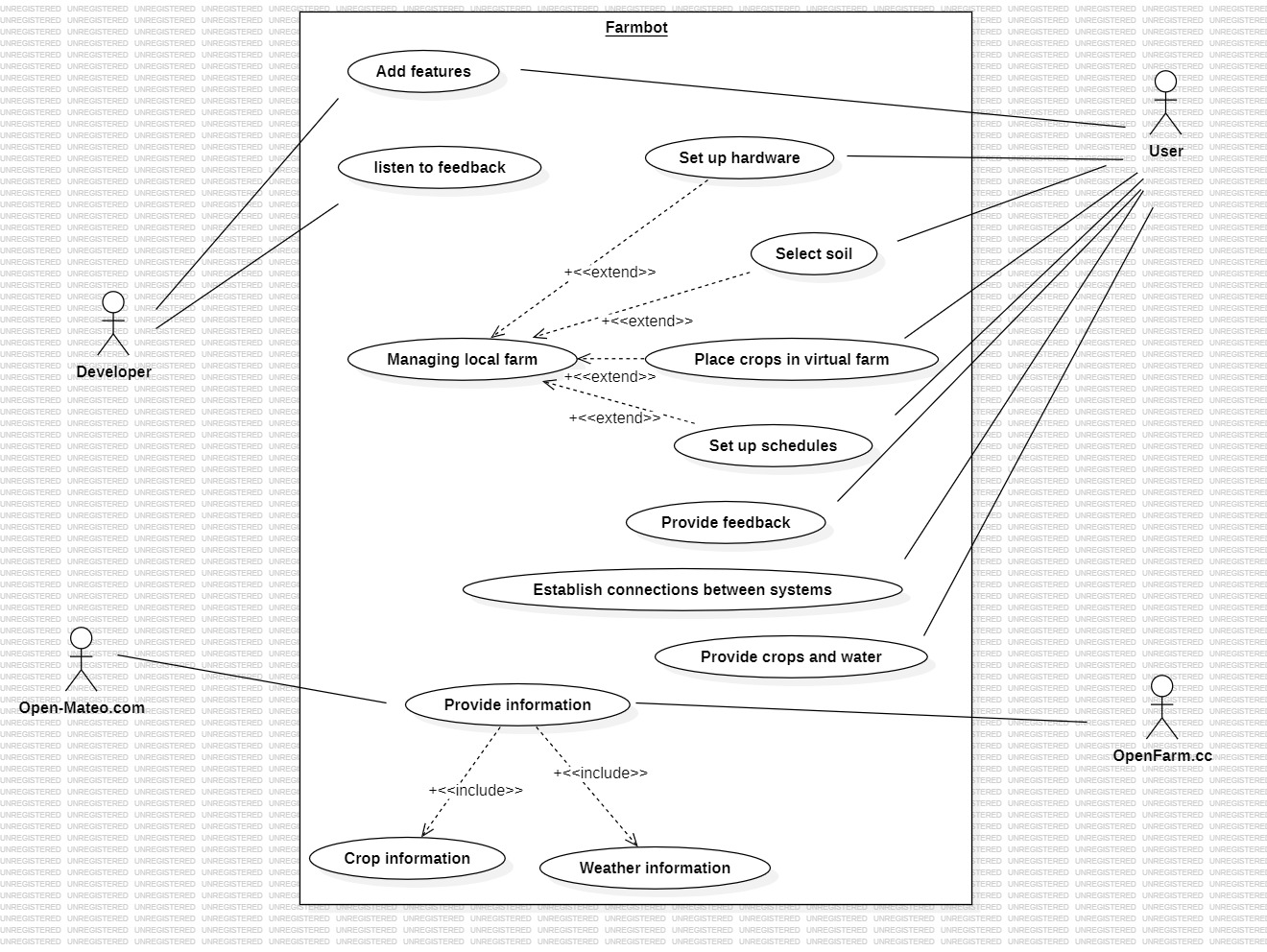
# **3. Specific Requirements**

## **3.1. External Interfaces**

## 

**Figure 12:** External Interfaces Class Diagram

## **3.2. Functions**



**Figure 13:** Use Case Model

|  |  |
| --- | --- |
| **User Case Name** | Set up hardware |
| **Actors** | User |
| **Description** | Setting up the hardware part of farm.bot project. |
| **Data** | - |
| **Preconditions** | User has the working components of the device.  User has farm.bot Web Application open.  User has farm.bot manual. |
| **Stimulus** | - |
| **Basic Flow** | Step 1: Log in to the application and start setup.  Step 2: follow the instructions written.  Step 3: Set up Raspberry Pi and connect it to Arduino  Step 4: Place local farm to an appropriate location and place the device as instructed. |
| **Alternative Flow** | - |
| **Exception Flow** | If an error occurs during setup, retrace steps to find the possible error or check FAQ on website. |
| **Post Conditions** | farm.bot is ready to take action. |

**Table 2:** Use Case “Set up hardware”

|  |  |
| --- | --- |
| **User Case Name** | Add features |
| **Actors** | Developer, User |
| **Description** | Since the project is open source and hardware devices are available, user can add desired features without waiting for developer. |
| **Data** | A new function for farm.bot |
| **Preconditions** | Knowledge on coding |
| **Stimulus** | - |
| **Basic Flow** | Step 1: Decide on desired feature  Step 2: Implement accordingly |
| **Alternative Flow** | - |
| **Exception Flow** | It is wise to debug regularly and have an extensive understanding of software engineering. In the event of an unresolved error, it is best to set up the developer’s version. |
| **Post Conditions** | New features are added. If it is by developer, users can access it. |

**Table 3:** Use Case “Add features”

|  |  |
| --- | --- |
| **User Case Name** | Listen to feedback |
| **Actors** | Developer |
| **Description** | Check user feedback to see if there is an unnoticed bug or complication to create a better user experience. |
| **Data** | Feedback |
| **Preconditions** | - |
| **Stimulus** | An issue or suggestion presented by a user. |
| **Basic Flow** | - |
| **Alternative Flow** | - |
| **Exception Flow** | - |
| **Post Conditions** | If the developers decide to act on it, they will add a feature or edit an existing feature. |

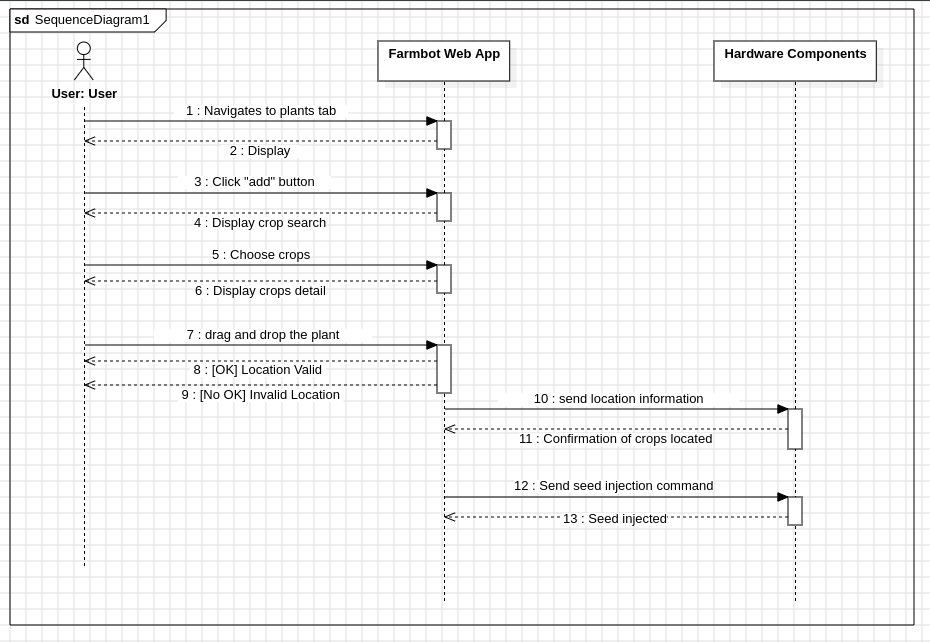
**Table 4:** Use Case “Listen to feedback”

|  |  |
| --- | --- |
| **User Case Name** | Select soil |
| **Actors** | User |
| **Description** | For crops to grow better, user decides on the type of soil and its details. |
| **Data** | Soil information |
| **Preconditions** | User must be logged in to the website |
| **Stimulus** | - |
| **Basic Flow** | Step 1: User navigates to soil tab in the application  Step 2: User edits the information given  Step 3: User saves the data |
| **Alternative Flow** | - |
| **Exception Flow** | There might not be specific data about soil that the user is searching for. In this case the user should simply inform the developer. |
| **Post Conditions** | Soil is ready to be planted on. |

**Table 5:** Use Case “Select soil”

|  |  |
| --- | --- |
| **User Case Name** | Place crops in virtual farm |
| **Actors** | User |
| **Description** | By using the virtual map in the application, the user manages the plants and their location. |
| **Data** | Crop information on map |
| **Preconditions** | User must be logged in to the website |
| **Stimulus** | -mq |
| **Basic Flow** | Step 1: User navigates to designer tab in the application  Step 2: User selects crops and their location. Then edits the remaining data as desired  Step 3: User saves the data |
| **Alternative Flow** | - |
| **Exception Flow** | There might not be specific data about crops that the user is searching for. In this case the user should simply inform the developer. |
| **Post Conditions** | Plants are ready to be planted if user issues command to the hardware |

**Table 6:** Use Case “Place crops in virtual farm”

**Figure 14:** Sequence Diagram for “Place Crops” use case

|  |  |
| --- | --- |
| **User Case Name** | Set up Schedules |
| **Actors** | User |
| **Description** | Schedule is needed to make a watering plan for the crops. User is responsible for schedules. |
| **Data** | Schedule information |
| **Preconditions** | User must be logged in to the website |
| **Stimulus** | - |
| **Basic Flow** | Step 1: User navigates to schedules tab  Step 2: User selects the type of schedule. Then edits the rest of the data  Step 3: User saves the data |
| **Alternative Flow** | - |
| **Exception Flow** | - |
| **Post Conditions** | Schedule is ready to be followed once the user sends the information to the hardware. |

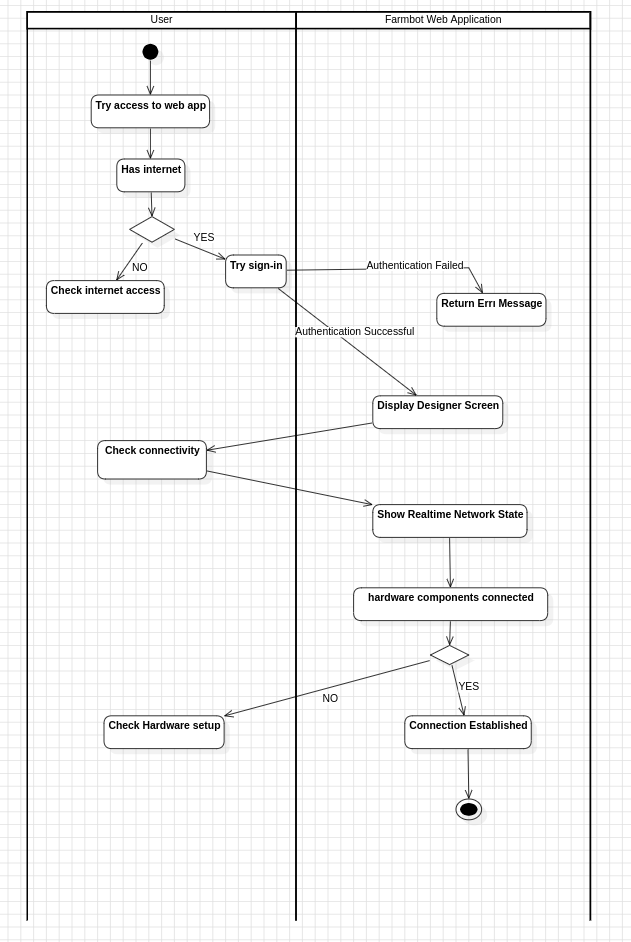
**Table 7:** Use Case “Set up Schedules”

|  |  |
| --- | --- |
| **User Case Name** | Provide feedback |
| **Actors** | User |
| **Description** | User shares an issue or suggests a feature to the developer. |
| **Data** | Feedback |
| **Preconditions** | - |
| **Stimulus** | - |
| **Basic Flow** | - |
| **Alternative Flow** | - |
| **Exception Flow** | - |
| **Post Conditions** | Developers might take the feedback into consideration to improve farm.bot project. |

**Table 8:** Use Case “Provide feedback”

|  |  |
| --- | --- |
| **User Case Name** | Establish connection between systems |
| **Actors** | User |
| **Description** | farm.bot project requires communication between its application and its hardware. User is responsible for the action. |
| **Data** | - |
| **Preconditions** | A reliable internet connection |
| **Stimulus** | - |
| **Basic Flow** | Step 1: User checks the internet connection  Step 2: User checks the data available from crops database and weather forecast  Step 3: User checks the uploaded data to the hardware is accurate and all physical connections are done  Step 4: Battery is charged |
| **Alternative Flow** | - |
| **Exception Flow** | - |
| **Post Conditions** | farm.bot is ready to take action. |

**Table 9:** Use Case “Establish connection between systems”



**Figure 15:** Activity Diagram for “Establish Connection Between Systems” use case

|  |  |
| --- | --- |
| **User Case Name** | Provide crops and water |
| **Actors** | User |
| **Description** | farm.bot is a farming project and therefore is need of physical farming requirements. |
| **Data** | - |
| **Preconditions** | Clean water source and seeds |
| **Stimulus** | User wants to plant crops |
| **Basic Flow** | Step 1: User places required objects to their respected places |
| **Alternative Flow** | - |
| **Exception Flow** | - |
| **Post Conditions** | Hardware is ready to follow instructions. |

**Table 10:** Use Case “Provide crops and water”

|  |  |
| --- | --- |
| **User Case Name** | Provide information |
| **Actors** | OpenFarm.cc, Open-Mateo.com |
| **Description** | farm.bot relies on a crops database and weather information to accurately function. These external systems provide this information. |
| **Data** | Crops and weather data |
| **Preconditions** | Internet access |
| **Stimulus** | User wants to plant crops |
| **Basic Flow** | Step 1: farm.bot requests information from these available external data systems.    Step 2: They provide requested data. |
| **Alternative Flow** | - |
| **Exception Flow** | - |
| **Post Conditions** | - |

**Table 11:** Use Case “Provide information”

## **3.3. Logical Database Requirements**

**hardware\_configuration:** Stores hardware configuration information along with hardware identification number and connected devices.

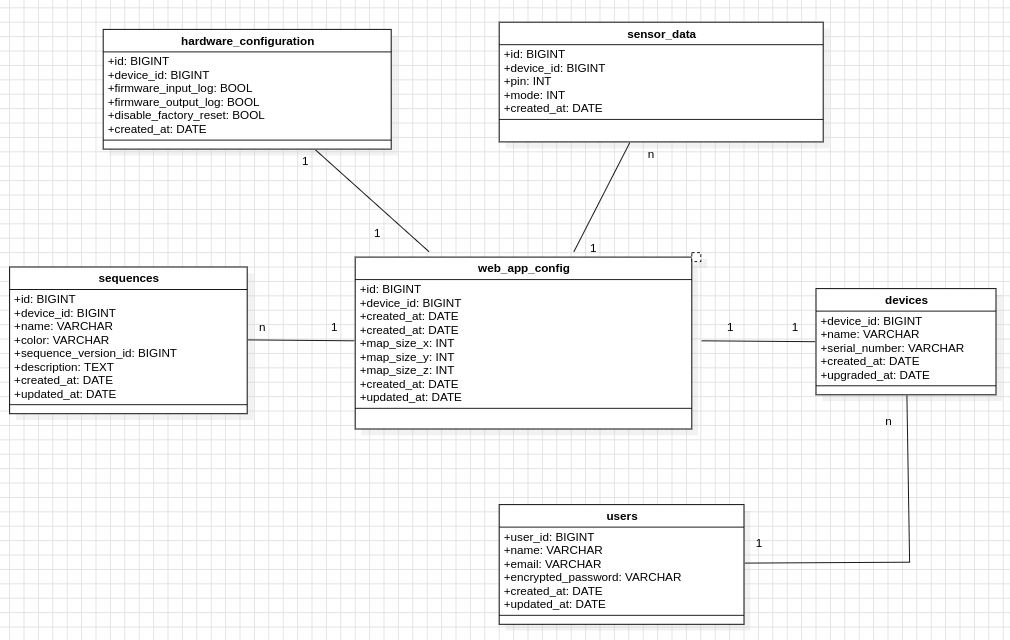
**sensor\_data:** Stores sensor and pin information with sensor mode.

**sequences:** Stores sequence information and customized setting properties such as color code with a description.

**web\_app\_config:** Stores user information, map layout information and connected device identification number.

**devices:** Stores device information and device serial number which is used for authentication by the web application.

**users:** Stores user communication information and user login information.

**Figure 16:** Logical Database Requirements Class Diagram

## **3.4. Design Constraints**

**Open-Source License**: Adherence to open-source software principles do not allow integrating proprietary technologies that may be beneficial for the overall project but against the project's ethos and accessibility.

**Local Restrictions and Regulations on Agricultural Activities:** Compliance with local agricultural regulations and practices may limit the use of FarmBot in certain regions or require ineffective modifications of design and functionality of the system.

**Hardware Compatibility:** Ensuring all hardware components are in contact and work properly together may not be easy to scale and create additional implementation problems.

**Power Consumption:** Modular nature and open-source nature of the project may result in less efficient and more energy consuming result. Optimization on the energy consumption side may require extra attention.

**Software Compatibility:** Even though the components are all open source and regularly updated, new modifications from the public side may result in software compatibility problems among different hardware components using different software versions such as farmbot OS and the Web application.

**Internet & Energy Connection Requirement:** To properly work, farmbot components require consistent energy and internet access 7/24, which limits the portability of the whole design.

## **3.5. System Attributes**

**Reliability**

**Durable Hardware Design**: The hardware components are selected and designed durable and resistant to regular use and exposure to external physical factors.

**Availability**

**Cloud-Based Services**: The FarmBot web application and helping cloud services are hosted on platforms provided by trusted systems ensuring users a consistent, high-quality access.

**Data Security**: All data transmitted between the hardware components, the web application, and the cloud services is encrypted using industry-standard protocols to protect user data privacy.

**Regular Software Updates**: The software receives regular updates to protect against vulnerabilities and ensure the system is safe.

Maintainability

**Modularity:** Both the hardware and software components of FarmBot have modular design, making it easier to update, repair, or replace parts of the system without affecting the entire system.

**Portability**

**Cross-Platform Web Application**: The FarmBot web application is designed to be cross-platform, accessible from any device with internet connectivity and a web browser.

**Scalable Cloud Services**: The backend infrastructure supporting FarmBot's web services can scale to handle a growing number of users, ensuring system’s performance remains stable as the user base expands without any additional requirements for adjustment or disks.

**Portable Hardware:** FarmBot system, along with the Farmbot Genesis hardware, is completely modular and lightweight which makes it easy to displace via disintegration and integration of the system.

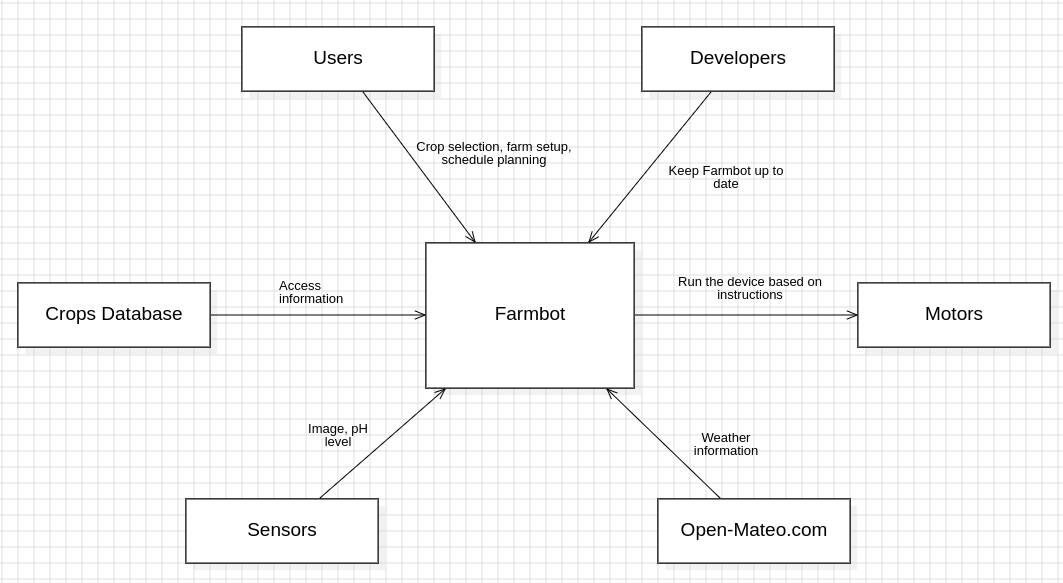
## **3.6. Supporting Information**

The farmbot project is an open-source agriculture project with the source code available on GitHub which is accessible on farmbot website, “Learn more” section. Contributions to the system and more information about the system’s requirements are accessible on both the same tab and the GitHub repository, which is accessible from the same tab.

# **4. Suggestions to Improve the Existing System**

## **4.1. System Perspective**

The improved version of farmbot system contains a mobile application, available on all mobile devices from smartphones to tablets on all operating systems. This improvement makes farmbot control easier to use and make alerts regarding the overall system way more eye-catching.



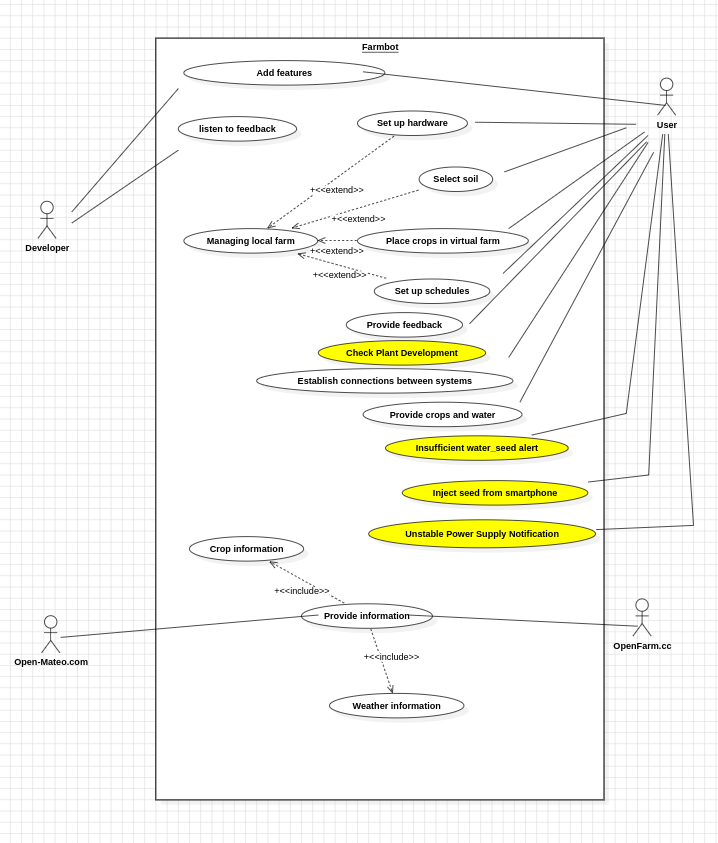
**Figure 17:** Context Diagram (No change)

## **4.2. External Interfaces**

## 

**Figure 18:** External Interfaces Class Diagram for improved system

## **4.3. Functions**



**Figure 19:** Use Case Model for improved system

|  |  |
| --- | --- |
| **User Case Name** | Insufficient Water/Seed Alert |
| **Actors** | User |
| **Description** | An alert with either noise or phone light to alert the user that water or seed is insufficient. |
| **Data** | Water or seed amount |
| **Preconditions** | Internet access and power supply |
| **Stimulus** | Noise/led signal |
| **Basic Flow** | Step 1: Raspberry checks the amount of water and seed of genesis  Step 2: Water or seed shortage  Step 3: Send notification to mobile device |
| **Alternative Flow** | - |
| **Exception Flow** | - |
| **Post Conditions** | - |

**Table 12:** Use Case for improved system “Insufficient Water/Seed Alert”

|  |  |
| --- | --- |
| **User Case Name** | Check plant development |
| **Actors** | User |
| **Description** | Send plant photo on user demand |
| **Data** | Picture of the plant |
| **Preconditions** | Internet access and power supply |
| **Stimulus** | - |
| **Basic Flow** | Step 1: User demands a photo via mobile application  Step 2: Related hardware components and integrated camera relocates according to the demanded plant’s position.  Step 3: Camera takes the picture and system sends the file to mobile app. |
| **Alternative Flow** | - |
| **Exception Flow** | - |
| **Post Conditions** | - |

**Table 13:** Use Case for improved system “Check Plant Development”

|  |  |
| --- | --- |
| **User Case Name** | Unstable Power Supply Notification |
| **Actors** | User |
| **Description** | Unexpected stability on power supply notifies the user via mobile app. |
| **Data** | Power connection |
| **Preconditions** | Internet connection |
| **Stimulus** | - |
| **Basic Flow** | Step 1: Raspberry checks the stability of power supply  Step 2: Abnormality on power input  Step 3: Send notification to mobile device |
| **Alternative Flow** | - |
| **Exception Flow** | - |
| **Post Conditions** | - |

**Table 14:** Use Case for improved system “Unstable Power Supply Notification”

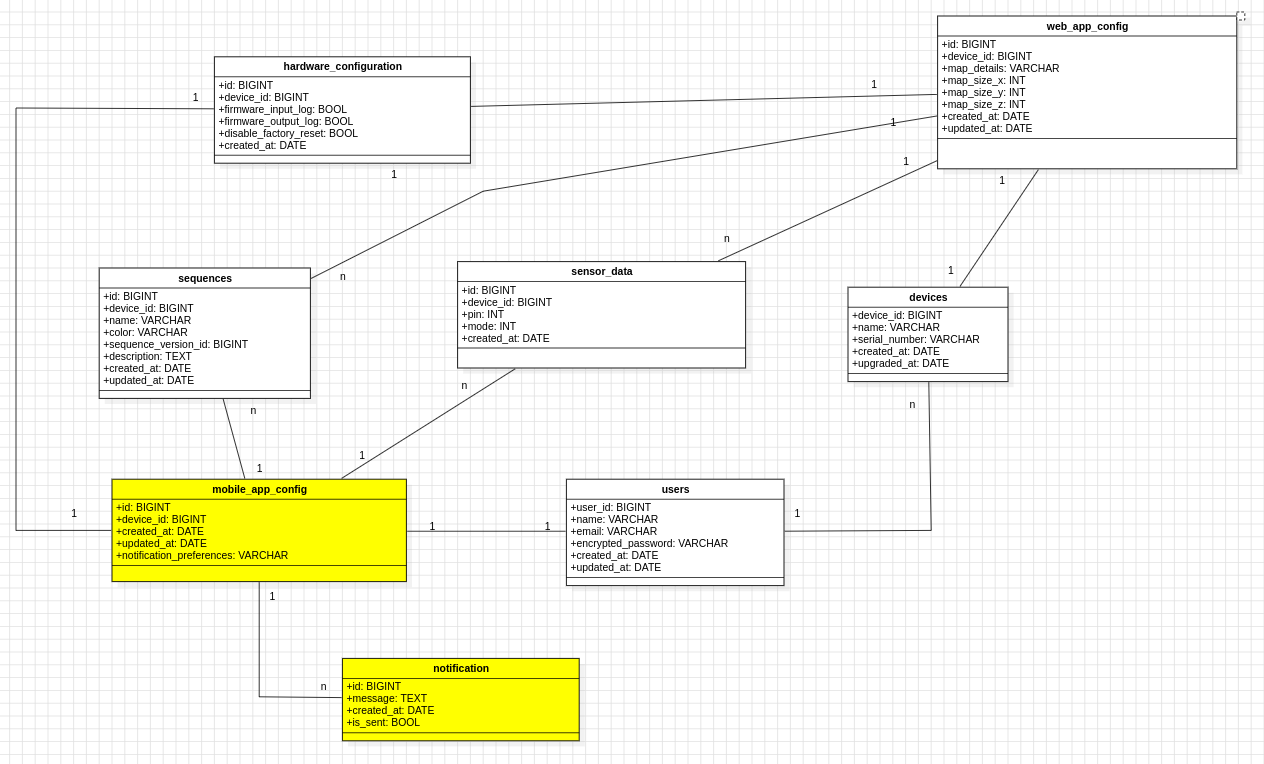
|  |  |
| --- | --- |
| **User Case Name** | Inject seed from smartphone |
| **Actors** | User |
| **Description** | Plant seed from mobile |
| **Data** | Seed type, location |
| **Preconditions** | Internet access |
| **Stimulus** | - |
| **Basic Flow** | Step 1: User locates the plant via drag and drop on the mobile application interface  Step 2: Raspberry checks the necessary conditions and sends command to arduino  Step 3: Arduino executes the injection process by its control system and genesis hardware. |
| **Alternative Flow** | - |
| **Exception Flow** | - |
| **Post Conditions** | - |

**Table 15:** Use Case for improved system “Inject seed from smartphone”

## **4.4. Logical Database Requirements**

**mobile\_app\_config:** Stores user information, map layout information and connected device identification number alogside with mobile device information.

**notification:** Keeps notification information to be sent to mobile device.

**Figure 20:** Logical Database Requirements Class Diagram for improved system

## **4.5. Design Constraints**

Constraints for web application is persistent in this case with additional constraints that may occure due to the nature of mobile platform such as OS version and device configuration.

## **4.6. System Attributes**

Mobile applications share the same attributes as web application. However, decreases the unnecessary details of the web application and more appropriate to mobile usage.