Documentation

Contents

[PIR Motion Sensor: 1](#_Toc103587515)

[Magnetic Door Sensor Reed Switch: 1](#_Toc103587516)

[MG590S 180 Micro Servo: 1](#_Toc103587517)

[Sound Sensor/Noise Detector 2](#_Toc103587518)

[GPS(Air530): 2](#_Toc103587519)

[Water Level Sensor: 2](#_Toc103587520)

[Soil Moisture Sensor: 2](#_Toc103587521)

[Chainable RGB LED: 2](#_Toc103587522)

[Cooling FAN: 2](#_Toc103587523)

[AHT20 Temp & Humidity Sensor: 2](#_Toc103587524)

[Cloud to Device 3](#_Toc103587525)

[The choice of communication method for controlling actuators. 3](#_Toc103587526)

[A brief explanation for why this method was chosen over other options. 3](#_Toc103587527)

[The keys, values or any necessary message formatting used to control each actuator. 3](#_Toc103587528)

[Examples: 3](#_Toc103587529)

# Team Information:

Team Name: Grass Touchers

Team Number: 2

Members: Magnus Bigras (1840918), Nicholas Chudinov ({id here}) Maxence Roy ({id here}) and Liam Scalzulli ({id here})

# Project Description:

The goal of this project was to implement all the necessary software that managers or farmers required in order to run a container farm. The hardware was divided into three different subsystems: plants/farming, geolocation and security. The plant subsystem is mainly focused on everything that has to with things that can affect the crops such as: the climate in the container, the soil moisture and the level of light. The geo location subsystem focuses on tracking the position of the container, whether it is placed on a sturdy base and if the container has been shaken or bumped which could cause potential issues to the farm. The security subsystem focuses on monitoring who accesses the container, using the hardware to avoid possible break ins. The hardware collects data from sensors, though so hardware items can also be controller such as: lights, fans or even the lock of the container.

Alongside the hardware, our team has developed an application. This provides farmers and managers an easy and practical way to see the data sent by sensors, they can also view the state of actuators. Within the app, users can easily control these actuators, allowing them to easily shut off or turn on devices.

# PIR Motion Sensor:

**Pin:** D16

# Magnetic Door Sensor Reed Switch:

**Pin:** D5

# MG590S 180 Micro Servo:

**Pin:** PWM

# Sound Sensor/Noise Detector

**Pin**: ADC 0

# GPS(Air530):

**Pin:** UART

# Water Level Sensor:

**Pin:** ADC 4

# Soil Moisture Sensor:

**Pin:** ADC 2

# Chainable RGB LED:

**Pin:** UART

# Cooling FAN:

**Pin:** D18

# AHT20 Temp & Humidity Sensor:

**Pin:** I2C bus 4

# Cloud to Device

## The choice of communication method for controlling actuators.

To control our actuators we have chosen to use a set of Direct Methods.

## A brief explanation for why this method was chosen over other options.

Direct methods were chosen as our group believed that it was the simplest solution. The method could be sent from the azure portal and the request-response nature of a direct method made it easy to test. With direct methods all that was required was implementing a handler for requests that would call upon one of the methods for the actuators, passing in an argument for that actuator’s state. We also had to create a format for our request, which basically included the desired state of the actuator.

## The keys, values or any necessary message formatting used to control each actuator.

To communicate with the actuators, a payload needs to be passed. This payload must contain the desired state of the actuator example (on/off).

## Examples:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Command** | **Description** | **Payload** | **Expected Arguments** | **Example** |
| fan | Control the state of the fan | {‘state’ : arg} | **‘**on’ / ‘off’ | {‘state’ : ‘on’} |
| lock | Control the state of the lock | {‘state’ : arg} | **‘**open’ / ‘closed’ | {‘state’ : ‘closed’} |
| buzzer | Control the state of the buzzer | {‘state’ : arg} | **‘**on’ / ‘off’ | {‘state’ : ‘on’} |
| light | Control the state of the light | {‘state’ : arg} | **‘**on’ / ‘off’ | {‘state’ : ‘on’} |

# App Snapshots (Waiting for full feature list)

# Future Work (TBA)

# Contributions:

|  |  |
| --- | --- |
| **Team Member** | **Contributions** |
| Magnus Bigras | **Milestone 2:** Created original Viewmodels, models, repos, security dashboard, login page and basic navigation.  **Milestone 3:** Created the entry model / entry records classes, using what our team decided to use in our payload. Designed the logic for allowing this one model to accommodate all three of our subsystems.  **Milestone 4:** Wrote the project documentation, created the class UML diagrams and coded the logic for allowing users to change the telemetry interval through the app |
| Maxence Roy |  |
| Nicholas Chudinov |  |
| Liam Scalzulli |  |