Final Project Documentation

Table of Contents

Team Information

Team Name: Grass Touchers

Team Number: 2

Members:
Magnus Bigras (1840918)
Nicholas Chudinov (1423131)
Maxence Roy (1957042)
Liam Scalzulli (1947334)

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 do 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 controlled 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.

IoT System

Sensors

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: D22
RX ⇒ GPIO pin 22
TX ⇒ GPIO pin 23

Cooling FAN
Pin: D18

AHT20 Temp & Humidity Sensor
Pin: I2C bus 4

Cloud to Device

Communication method for controlling actuators

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

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.

Message formatting for actuators

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'}

Contributions

Team Member	Contributions
Magnus Bigras	**Milestone 1:** Developed team contract and plan for the project **Milestone 2:** Responsible for writing the scripts for collecting data and controlling actuators from the security subsystems **Milestone 3:** Sending security telemetry to IOT Hub and desired properties. **Milestone 4:** Reported properties, fixing bugs, helping integrate all the subsystems into one script, direct methods. **Milestone 5:** Cleaning up code, separating classes into separate modules and documentation
Maxence Roy	**Milestone 1:** Wrote team contract, Created the Jira board and added epics and user stories, Programmed plant subsystem sensors **Milestone 2:** Made plant subsystem class **Milestone 3:** Code review, Implemented plant subsystem to collective class **Milestone 4:** Code review **Milestone 5:** Made the payload send all entries at once instead of individually, Added ‘dummy’ argument to send data for each subsystem without hardware, Made it so all entries are sent by default (if no specific entry is specified), Changed binary entries to send more specific data like ‘on’ and ‘off’ instead of True and False, Completed the ‘Getting Started’ section of the document
Nicholas Chudinov	**Milestone 1:** Helped with the team contract **Milestone 2:** Made the GPS subsystem scripts and got the GPS data running **Milestone 3:** Modified the scripts to have the suggested changes as well as Code Cleanup **Milestone 4:** Code Cleanup
Liam Scalzulli	**Milestone 1:** Did everything in a different group (wireframing, team contract, jira board, readme). **Milestone 2:** Implemented the geo-location subsystem, read and outputted all required data. **Milestone 3:** Implemented sending data to IOT hub and controlling desired properties in the geo-location subsystem. **Milestone 4:** Combined all subsystems into a single python script, added direct method and desired property handlers **Milestone 5:** Minor bug fixes and code cleanup

Getting Started

Run **farm.py** to connect your device to IoTHub and start sending data.

Every library that cannot be installed with pip3, such as chainable_rgb_direct, is provided in the Hardware folder.

Command Line Arguments

--dummy: Send dummy data to IoTHub without using any hardware component (prevents relying on hardware)

--all: Send all data at once (this is automatically done when running without specifying any sensors)

Arguments for specific sensors: --angles, --buzzer, --door, --fan, --gps, --light, --lock, --luminosity, --moisture, --motion, --noise, --temp_humi, --vibration, --water

--verbose: Add tracebacks to error messages

Running without a Raspberry Pi

With dummy data, it is possible to use farm.py from any device. To run python scripts on a Windows computer, you must complete the following steps.

- Install python if not already done
- Add the python folder location in your Path environment variable
- Install ms-python.python and tht13.python and formulahendry.code-runner extensions on Visual Studio Code
- Restart your computer
- You should now be able to run farm.py from the command line or powershell by calling '**python .\farm.py --dummy**'

Mobile App

App Purpose

The purpose of our application is to provide a user interface where farm technicians and fleet managers can remotely track the status of all their respective sensors in real time. The app also allows the users to change the state of their actuators and telemetry interval. Finally, it contains a page to let the fleet manager track the location of their subsystem on a map.

App Functionality

Login Page

The login page lets the user select which dashboard they want to access: farm technician or fleet manager. Selecting either one will navigate to the

Dashboard Page

This page loads different records depending on the dashboard type selected on the login screen. It displays every record desired and updates them in real-time. Clicking any record navigates to the details screen. On top of the page the user can select an actuator, type a desired state, and click ‘Run’ to run a direct method which updates the specified actuator remotely. There is also a toolbar item with an ‘upload’ symbol. Clicking it navigates to the telemetry form page. If the fleet manager dashboard is being displayed, there is a second toolbar item with a ‘globe’ symbol. Clicking it navigates to the Location page. Finally, at the bottom of the page, it is indicated whether the app is currently connected to the internet or not to let the user know if their records are being updated.

Details Page

This page displays information about a desired record. On top, a graph is displayed showing the last 8 values sent. At the bottom, the user can scroll through the list of every value sent, with the latest one on top. The graph and value list update in real-time.

Telemetry Form Page

This page allows the user to change the frequency at which their subsystem sends telemetry data. To do so, they simply need to type a new value and click the ‘Update’ button.

Location Page

This page displays a Google map centered on the current location of the geo-location subsystem (based on the latest latitude and longitude data). A pin highlights the subsystem location.

Note: The Google map will only load on an Android device that meets certain conditions. See the ‘Google Maps API Constraints’ on the Mobile App section.

OOP Design

SensorRecord

This class represents a record of entries for a field. Its purpose is to store a list of every value sent by IoTHub for a single field.

SensorEntry

This class represents a value sent at a certain time. Its purpose is to be collected by a SensorRecord and represent a temporary value for a field.

Payload

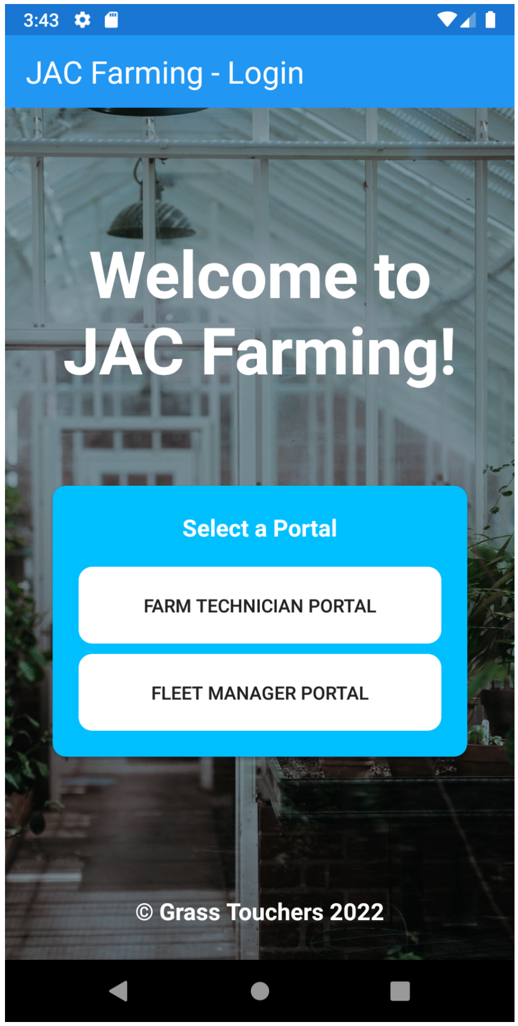
This class represents the raw keys and values sent by a IoTHub payload. Its purpose is to easily parse the string data received into a usable SensorEntry.

UML Diagram

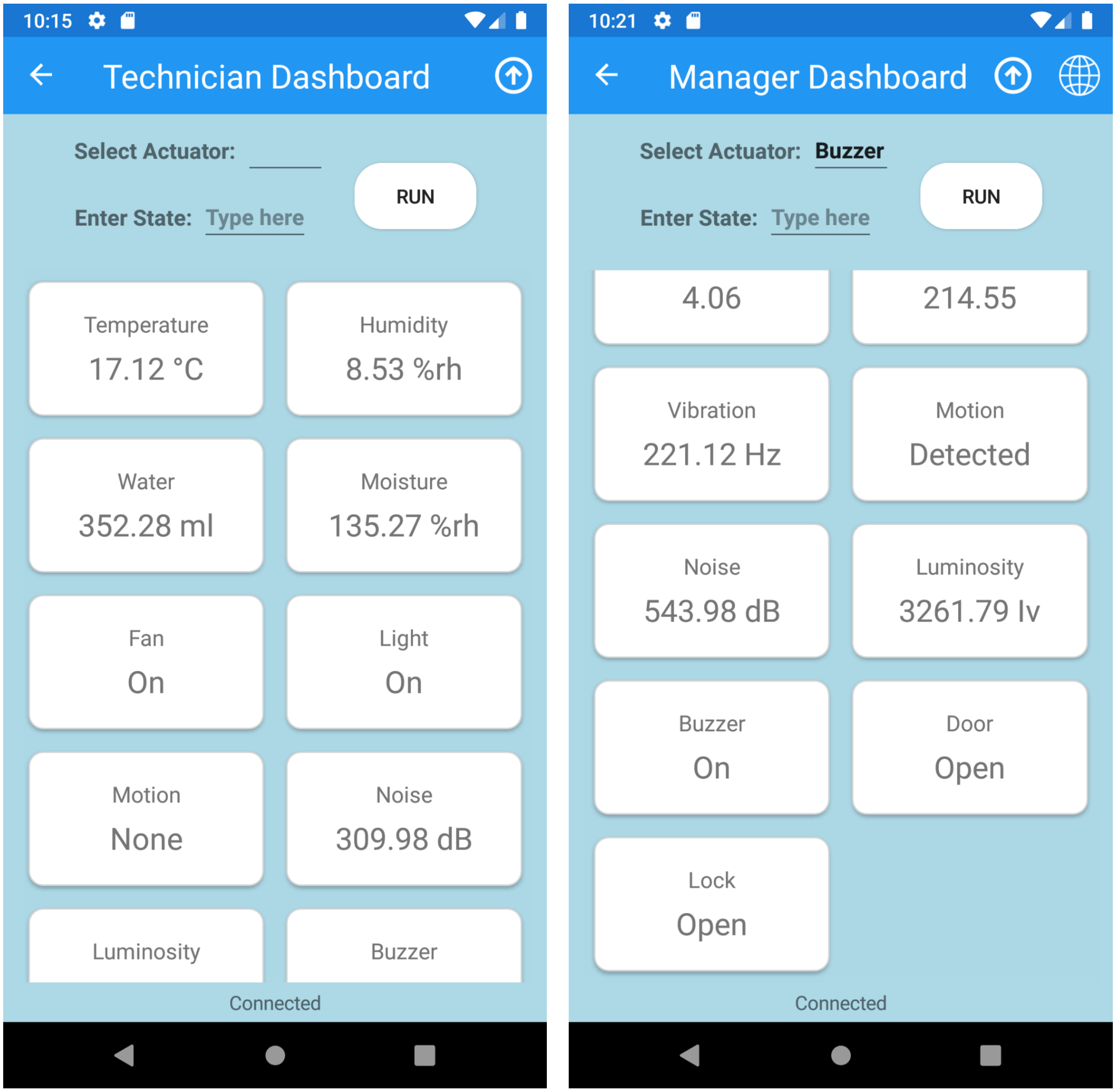
The UML Diagram is too large to display clearly on the document. To see it, open the JPEG file named UML.jpeg in the root of the GitHub repo.

App Snapshots

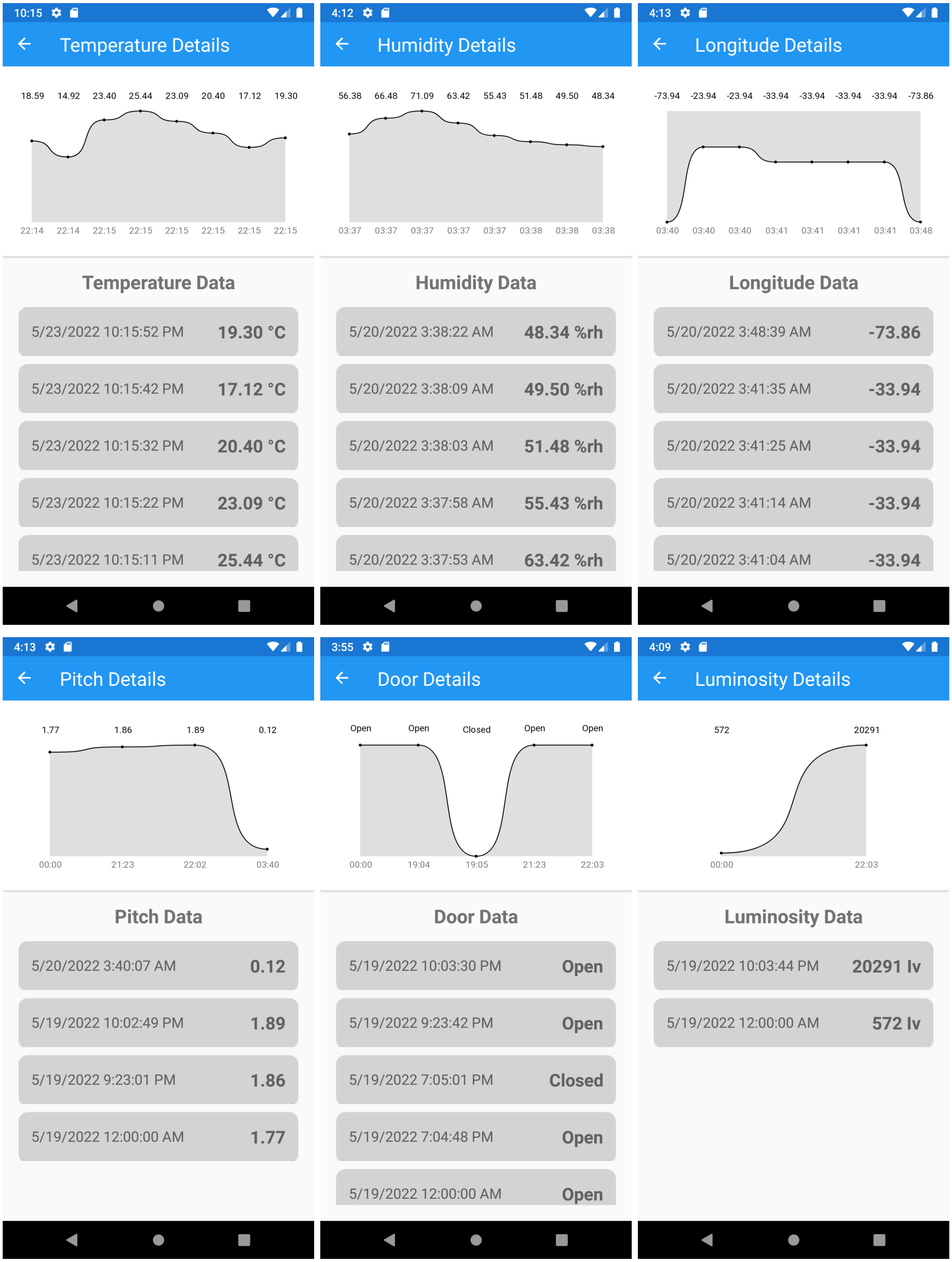
Login Page



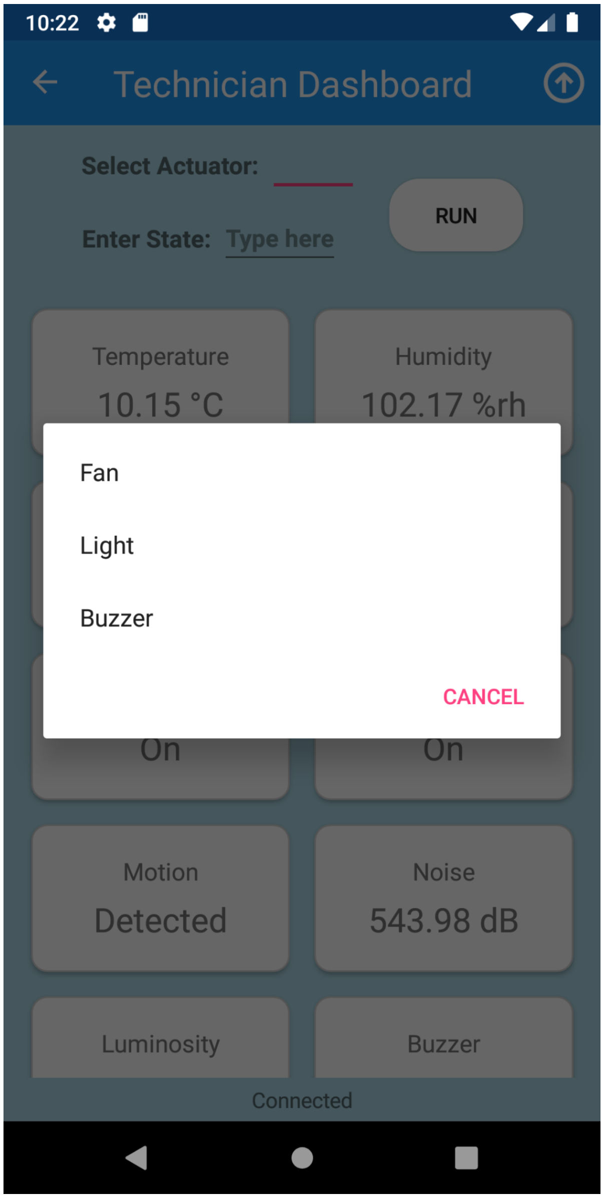
Farm Technician and Manager Dashboard Pages

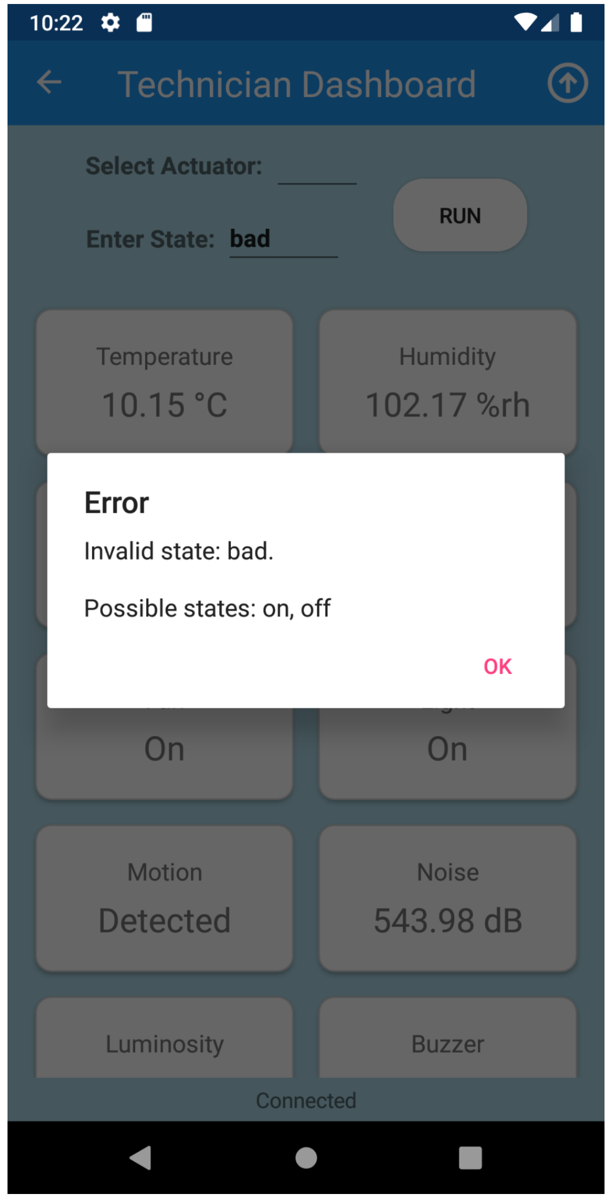


Details Page examples

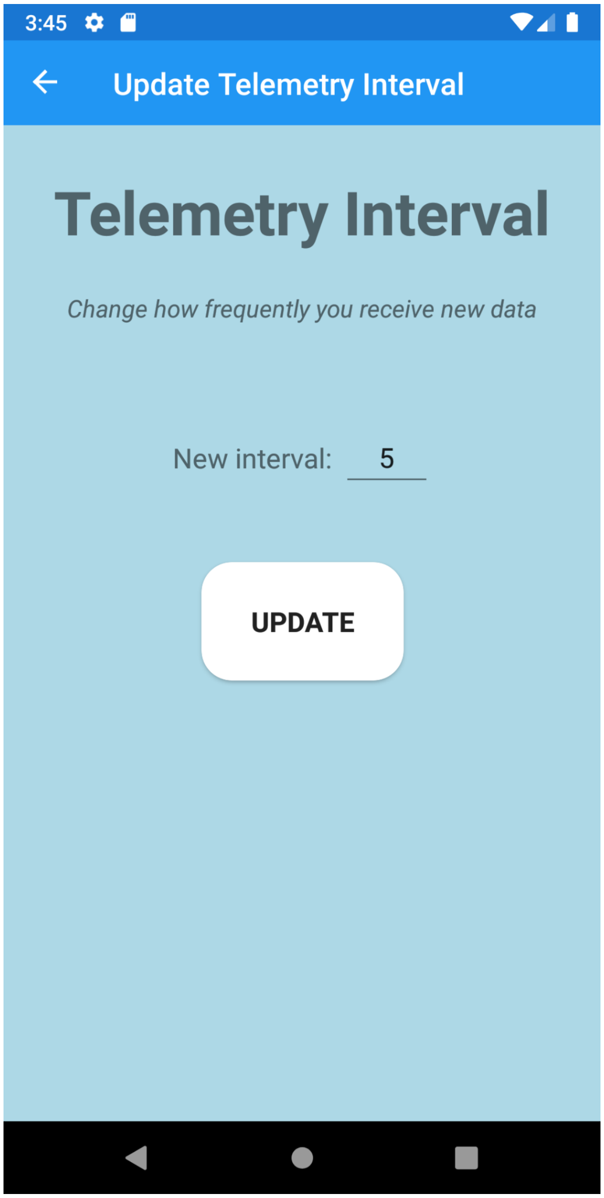


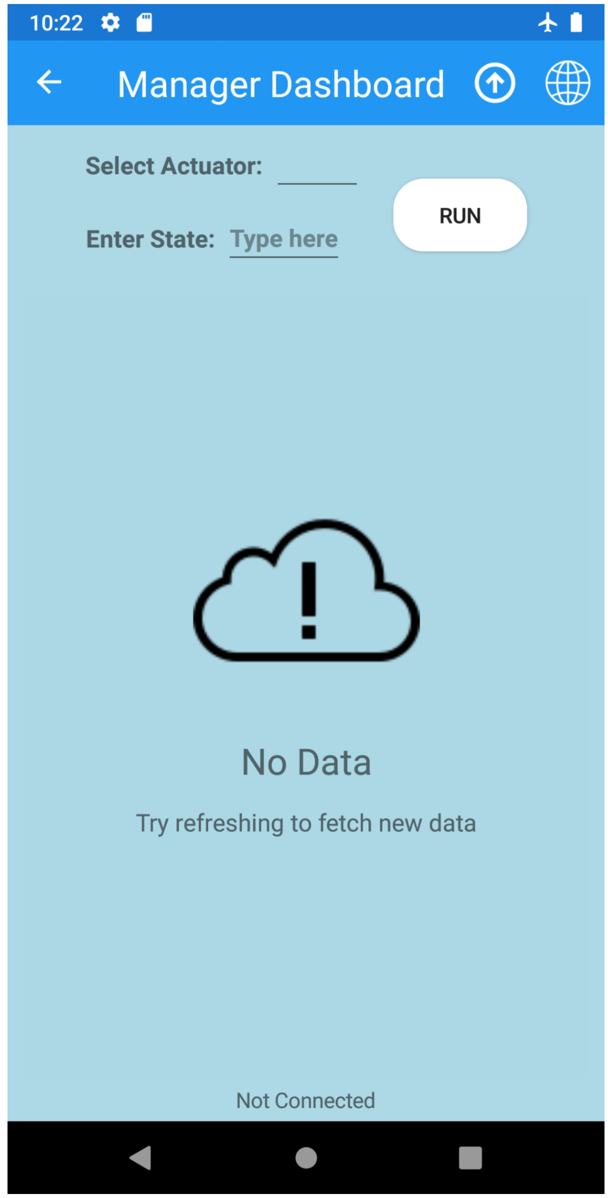
Selecting an Actuator / Trying to run a Direct Method with an invalid state



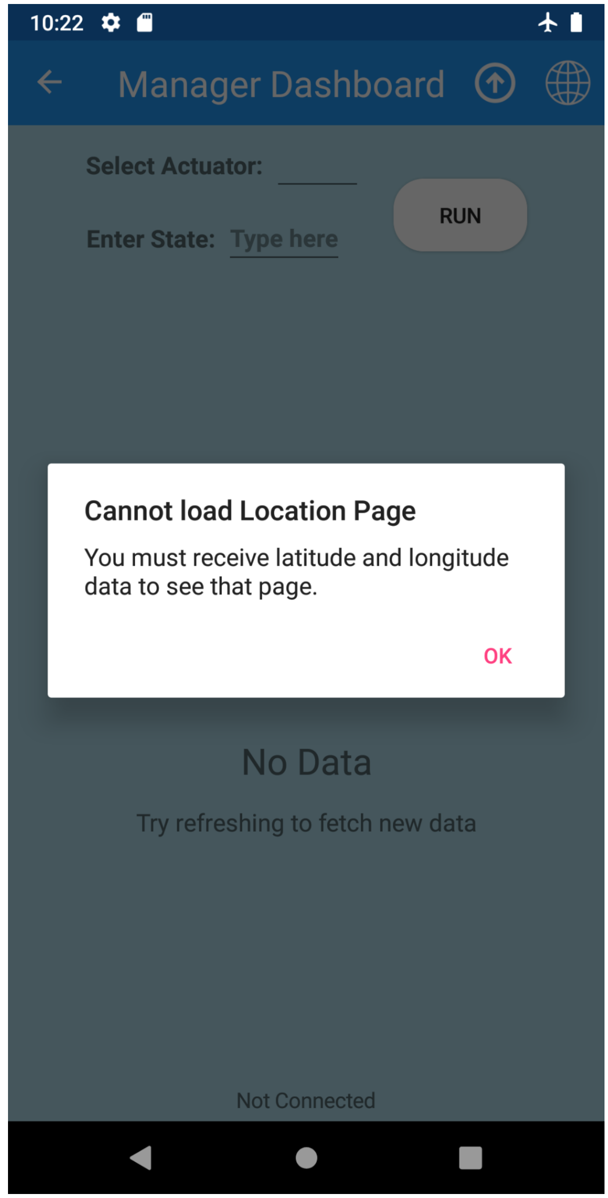


Telemetry Interval Page / Empty Dashboard Page





Location Page / Trying to open the location page with no latitude and longitude record



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	**Milestone 1:** Wrote team contract, Created the Jira board and added epics and user stories **Milestone 2:** Reviewed code quality (added XML comments and headers), Recorded the demo video **Milestone 3:** Code review **Milestone 4:** Revamped ViewModels, Refactored user roles, Details view (Graph and entry list), Added Location page, Implemented auto refresh for Dashboard and Details screen, Polished the design (ie added app icon, images for toolbar, reorganizing controls on the dashboard), Added popup message to show direct method result, Remade the UML diagram, Completed the 'App Purpose', 'App Functionality', 'OOP Design', and 'Test Payload' sections of the document
Nicholas Chudinov	**Milestone 1:** Did the designing and wireframing of the app **Milestone 2:** Created the App UI for dashboards, login and details based off of the wireframe **Milestone 3:** Designed the whole UI side of the project as well as formatted the incoming data to display for the user. Formatted some code for the app and bug fixes overall. **Milestone 4:** Code cleanup
Liam Scalzulli	**Milestone 1:** Did everything in a different group (wireframing, team contract, jira board, readme) **Milestone 2:** (Late integration into this team) Implemented navigation for the app, also the geo-location subsystem. **Milestone 3:** Added functionality for reading and deserializing data from Azure Event Hubs in the mobile app. **Milestone 4:** Added functionality for controlling actuators in the app, minor refactors and bug fixes.

Future Work

Here are the list of features we would like to implement if time permits:

- **Device notification:** Include a label that mentions if the device is currently online (actively sending data or not)
- **Actuator state dropdown:** Instead of making the user type the new state for an actuator, make them select the state from a dropdown for convenience
- **Customizable Dashboard tiles:** Allowing users to modify the size, color and shape of tiles on their dashboard.
- **Storage:** Obtain IOT telemetry data from blob storage, then display past data in the app.
- **User Authentication:** Having accounts for our two types of user roles (managers and farmers), logging in via Google
- **Sharing:** Allow users to easily copy and send sensor data (SMS, Email, ect...)
- **Security notification:** Highlight records that exceed a certain threshold. Receive notifications and emails when it happens.
- **Auto-refresh map:** Have the Location Page automatically update every time new GPS data is sent
- **Expanded design:** Adding a custom UI style to the full app (like dark mode)

Google Maps API Constraints

To access the Google map on the Location page, the Android device must have the Google Services SDK installed.

On top of this, the user must retrieve a SHA-1 certificate fingerprint from their device, send it to the Google Maps API owner (in this case, Maxence) and then the owner must add that key to their list of allowed devices.

Source:
https://docs.microsoft.com/en-us/xamarin/android/platform/maps-and-location/maps/obtaining-a-google-maps-api-key

Aside from this, the application does not need any prior modification to run properly.

Test Payload

Note: Dummy test data can be sent by running farm.py with the --dummy argument. See the ‘Getting Started’ section for Hardware.


```
[
    {'SubSystem': 'location', 'Field': 'pitch', 'Value': '4.379662128747901', 'EntryDate': '05/24/2022, 16:30:04'},
    {'SubSystem': 'location', 'Field': 'roll', 'Value': '206.2093777328896', 'EntryDate': '05/24/2022, 16:30:04'},
    {'SubSystem': 'security', 'Field': 'buzzer', 'Value': 'on', 'EntryDate': '05/24/2022, 16:30:04'},
    {'SubSystem': 'location', 'Field': 'buzzer', 'Value': 'on', 'EntryDate': '05/24/2022, 16:30:04'},
    {'SubSystem': 'security', 'Field': 'door', 'Value': 'open', 'EntryDate': '05/24/2022, 16:30:04'},
    {'SubSystem': 'plant', 'Field': 'fan', 'Value': 'off', 'EntryDate': '05/24/2022, 16:30:04'},
    {'SubSystem': 'location', 'Field': 'latitude', 'Value': '-1.1747667677956173', 'EntryDate': '05/24/2022, 16:30:04'},
    {'SubSystem': 'location', 'Field': 'longitude', 'Value': '-27.120174309568533', 'EntryDate': '05/24/2022, 16:30:04'},
    {'SubSystem': 'plant', 'Field': 'light', 'Value': 'off', 'EntryDate': '05/24/2022, 16:30:04'},
    {'SubSystem': 'security', 'Field': 'lock', 'Value': 'closed', 'EntryDate': '05/24/2022, 16:30:04'},
    {'SubSystem': 'security', 'Field': 'luminosity', 'Value': '6027.381584400036', 'EntryDate': '05/24/2022, 16:30:04'},
    {'SubSystem': 'plant', 'Field': 'moisture', 'Value': '181.0367148209547', 'EntryDate': '05/24/2022, 16:30:04'},
    {'SubSystem': 'security', 'Field': 'motion', 'Value': 'detected', 'EntryDate': '05/24/2022, 16:30:04'},
    {'SubSystem': 'security', 'Field': 'noise', 'Value': '542.6282268368966', 'EntryDate': '05/24/2022, 16:30:04'},
    {'SubSystem': 'plant', 'Field': 'temperature', 'Value': '14.954746520733263', 'EntryDate': '05/24/2022, 16:30:04'},
    {'SubSystem': 'plant', 'Field': 'humidity', 'Value': '67.64603951971893', 'EntryDate': '05/24/2022, 16:30:04'},
    {'SubSystem': 'location', 'Field': 'vibration', 'Value': '155.0604756599238', 'EntryDate': '05/24/2022, 16:30:04'},
    {'SubSystem': 'plant', 'Field': 'water', 'Value': '67.5364057973784', 'EntryDate': '05/24/2022, 16:30:04'}
]
```