

# CySecAgri

FINAL REPORT

## Team 17

David Wolfe - IoT Sensors & Base station  
Thomas Ruminski - IoT Sensors & Base station  
Joe Hunter - Frontend Development  
Rian Lamarque - AWS Infrastructure  
Eli Hanson - AWS Infrastructure

## Contact Information

Email: [sdmay23-17@iastate.edu](mailto:sdmay23-17@iastate.edu)  
Website: <https://sdmay23-17.sd.ece.iastate.edu>

## Client/Advisor

Dr. Manimaran Govindarasu

**Revised: 4/29/2023**

# Executive Summary

## Project Objective

CySecAgri's primary goal is to develop an easily accessible, secure, closed loop platform that allows large and small farms to manage and track IoT sensor readings from all across their crops. This objective will take the form of a secure mobile application that communicates with a private AWS instance to provide real time data tracking of sensors deployed in fields anywhere in the world.

## Development Standards & Practices Used

Flutter Application:

- [ISO/IEC 5055:2021](#)
- [ISO/CD 18893](#)

Cloud Computing:

- [ISO/IEC 19944-1 \(2020\)](#)
- [AWS Well-Architected framework](#)

IoT Sensors & Basestation:

- [IEEE 802.15.4](#)
- [IEEE 802.11ac](#)
- [LoRa Alliance Specifications](#)

## Summary of Requirements

**Functional Requirements:**

- IoT soil moisture sensor
  - Data collection
  - Medium-long range wireless transmitter (LoRa)
- IoT basestation
  - Data logging
  - Medium-long range receiver to sensors (LoRa)
  - Wifi data transfer to AWS cloud
  - Secure and consistent data logging and distribution
- AWS cloud
  - Data logging for verified data integrity
  - Data graphing to display to farmers
  - Data transfer using MQTT for IoT and SQS for application
  - Use of Amazon DynamoDB, Lambda, IAM, and Elasticsearch to support data flow and security
- Flutter application
  - Secure login and sign up authentication for each user

- Ability to add/remove/rename sensors
- Data visualization and representation of sensor data
- Data logging and graphing of sensor data
- Send and receive data from to the AWS cloud via Rest API

#### **Non-Functional Requirements:**

- IoT soil moisture sensor
  - Secure and consistent data collection and distribution
  - The IoT sensor will be able to withstand different climate conditions like rain, snow, and wind
- IoT basestation
  - Secure and consistent data logging and distribution
  - The IoT basestation will be able to withstand different climate conditions like rain, snow, and wind
- AWS cloud
  - Data can be stored and accessible from any location and time
  - Stored data will be secure
  - Deploy infrastructure via Terraform
- Flutter application
  - Will be available for both Android and iOS based devices
  - Will display sensor data to the user via two pages, a scroll view and a graph view
  - Users will only be able to see their sensor's data

#### **Physical Requirements:**

- Minimal Footprint (Constraint)
  - Can not take up space needed for crops
  - Can not impede tractor movement.
- Weather resistant (Constraint)
  - Water-proof
  - Hot and cold temperature tolerance
- High Visibility
  - Distinct colors to make it easy to locate in a field
  - Reflector strips at night and inclement weather

#### **Resource Requirements:**

- Batteries should last from planting to harvest
  - Minimal maintenance needed

#### **Environmental Requirements:**

- Batteries don't leak into soil
- Materials don't change soil nutrient levels

#### **UI Requirements:**

- Flutter application
  - Data is only accessible to platform user
  - System performs identically regardless of location
  - Application operates on multiple operating systems
  - Application can handle multiple users at the same time
  - Users can not view other users data

**Security Requirements:**

- IoT Sensors & Basestation
  - Data will be encrypted when stored on disk
  - Data will be encrypted end-to-end in transit
  - Sensors will be authenticated to the basestation using OTAA
- AWS Cloud
  - User roles will be properly separated
  - Minimum permissions will be allowed for all database accesses
- Flutter Application
  - Proper data sanitization will be enforced
  - Users will only be able to access their accounts data

## Applicable Courses from Iowa State University Curriculum

- CprE 309 for creating the frontend and backend connections
- CprE 288 for working with UART and serial interfaces of IoT chips
- CprE 230/231 for understanding security principals
- CprE 489 for networking and data communication principles

## New Skills/Knowledge acquired that was not taught in courses

- LoRaWAN connection protocols
- Flutter application design
- AWS API setup
- All AWS service knowledge

# Table of Contents

<b>Project Objective</b> .....	<b>1</b>
<b>Development Standards &amp; Practices Used</b> .....	<b>1</b>
<b>Summary of Requirements</b> .....	<b>1</b>
<b>Applicable Courses from Iowa State University Curriculum</b> .....	<b>3</b>
<b>New Skills/Knowledge acquired that was not taught in courses</b> .....	<b>3</b>
<b>Figures/Tables/Symbols/Definitions</b> .....	<b>7</b>
<b>1 Team</b> .....	<b>8</b>
1.1 Team Members.....	8
1.2 Required Skill Sets for Your Project.....	8
1.3 Skill Sets covered by the Team.....	8
1.4 Project Management Style Adopted by the team.....	8
1.5 Initial Project Management Roles.....	8
<b>2 Introduction</b> .....	<b>9</b>
2.1 Problem Statement.....	9
2.2 Intended Users and Uses.....	9
2.3 Requirements & Constraints.....	10
2.4 Engineering Standards.....	12
<b>3 Project Plan</b> .....	<b>13</b>
3.1 Project Management/Tracking Procedures.....	13
3.2 Task Decomposition.....	13
3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria.....	14
3.4 Project Timeline/Schedule.....	15
3.5 Other Resource Requirements.....	15
<b>4 Design</b> .....	<b>16</b>
4.1 Design Context.....	16
4.1.1 Broader Context.....	16
Table 2.....	16
4.1.2 Prior/Related Work & Solutions.....	17
4.1.3 Market Survey.....	17
4.1.3 Technical Complexity.....	18
4.2 Design Exploration.....	19
4.2.1 Design Decisions.....	19
4.2.2 Ideation.....	19
4.2.3 Decision-Making and Trade-Off.....	20
Table 3.....	20
4.3 Proposed Design.....	20
4.3.1 Overview.....	20
4.3.2 Detailed Design and Visual(s).....	21
IoT Sensors & Basestation.....	21

Devices.....	21
Cloud Storage & Processing.....	21
Front End Application.....	22
4.3.3 Functionality.....	23
Initial Setup.....	23
Daily Use.....	23
4.3.4 Areas of Concern and Development.....	24
IoT Devices.....	24
Cloud.....	24
Front End.....	24
4.4 Technology Considerations.....	24
LoRA RF and LoRaWAN.....	24
Amazon Web Services (AWS).....	25
Flutter.....	25
4.5 Design Analysis.....	25
IoT Sensors & Basestation.....	25
Cloud.....	26
Front End.....	26
Overview.....	27
5 Testing.....	27
5.1 Unit Testing.....	27
5.2 Interface Testing.....	27
5.3 Integration Testing.....	28
5.4 System Testing.....	28
5.5 Regression Testing.....	29
5.6 Acceptance Testing.....	29
5.7 Security Testing.....	29
5.7.1 Threat Scenarios and Mapping.....	29
5.7.2 Mobile Application Testing.....	30
5.7.3 AWS Infrastructure Testing.....	30
5.7.4 Basestation/Sensor Testing.....	31
5.8 Results.....	31
<b>Table 4</b> .....	<b>31</b>
5.8.1 Mobile App Testing Results.....	32
5.8.2 AWS Testing Results.....	36
5.8.3 Basestation Testing Results.....	36
5.9 Remediations.....	37
6 Implementation.....	38
6.1 Implementation Since 491.....	38
7 Professional Responsibility.....	39
Table 5.....	39

7.1 Areas of Responsibility.....	40
7.2 Project Specific Professional Responsibility Areas.....	41
7.3 Most Applicable Professional Responsibility Area.....	42
8 Future Work.....	42
8.1 IoT Devices Future Work.....	42
8.2 AWS Future Work.....	43
8.3 Mobile App Future Work.....	43
9 Closing Material.....	43
9.1 Discussion.....	43
9.3 Conclusion.....	43

## Figures/Tables/Symbols/Definitions

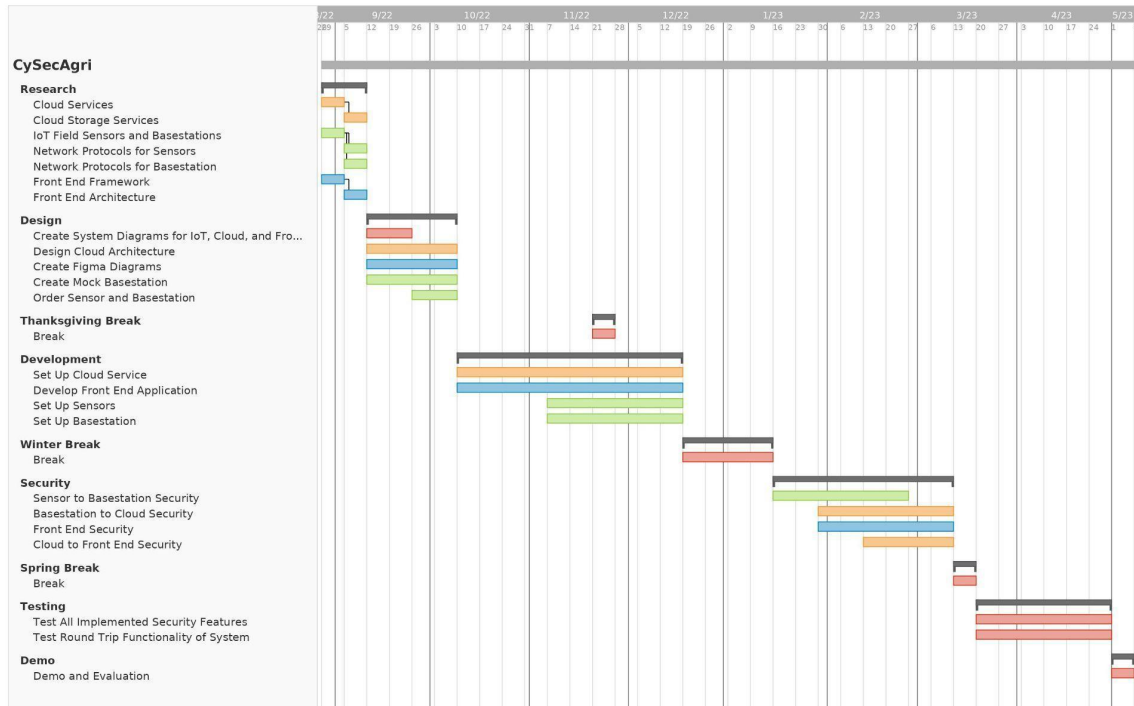


Figure 1. Gantt Chart (Appendix A)

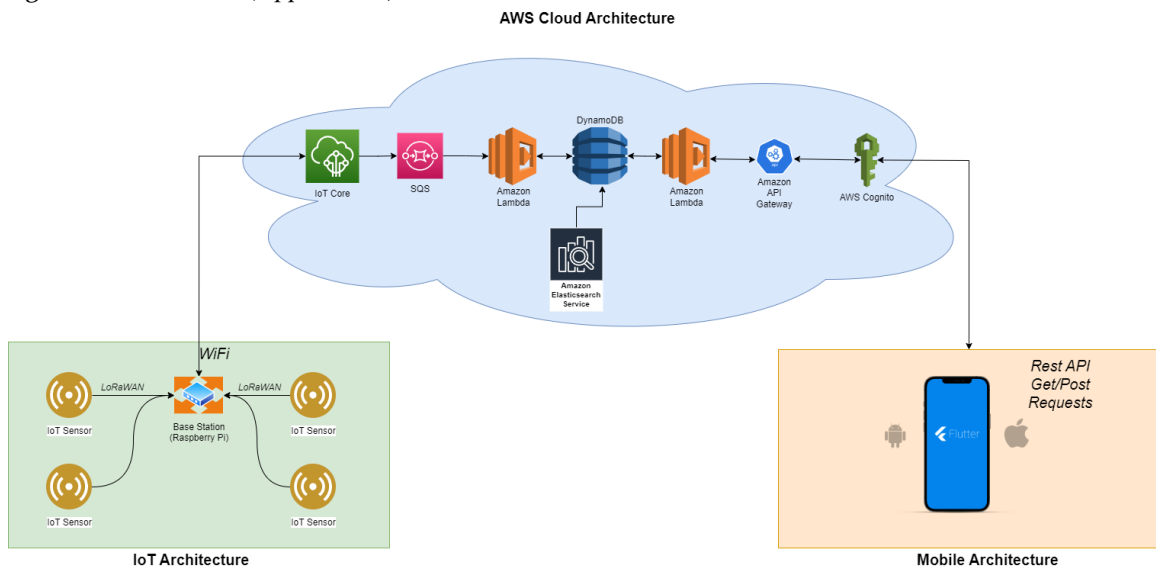


Figure 2. System Diagram (Appendix B)



# 1 Team

## 1.1 TEAM MEMBERS

David Wolfe, Tom Ruminski, Joe Hunter, Rian Lamarque, Eli Hanson

## 1.2 REQUIRED SKILL SETS FOR YOUR PROJECT

AWS service experience

Flutter framework experience

Lambda device management

GPIO programming

Python programming experience

## 1.3 SKILL SETS COVERED BY THE TEAM

AWS service experience is covered by Rian Lamarque

Lambda device management is covered by Eli Hanson and Rian Lamarque

Python programming is covered by David Wolfe

GPIO programming is covered by Tom Ruminski

Flutter framework experience is covered by Joe Hunter and David Wolfe

## 1.4 PROJECT MANAGEMENT STYLE ADOPTED BY THE TEAM

Our team agreed to adopt an agile development project.

## 1.5 INITIAL PROJECT MANAGEMENT ROLES

David Wolfe - IoT Sensors & Basestation, Meeting Note Taker

Tom Ruminski - IoT Sensors & Basestation

Joe Hunter - Frontend development

Rian Lamarque - AWS Infrastructure

Eli Hanson - AWS Infrastructure

## 2 Introduction

### 2.1 PROBLEM STATEMENT

Our project is focused on solving the problem of professional farmers and hobby gardeners needing to manually test, record, and chart IoT sensor data. The solution we will work towards will include the farmers deploying IoT sensors and a basestation to aggregate data. From that basestation, data will be uploaded to the cloud for storage. We will then develop a front-end application for accessing this data. Our solution will include a number of different charts and graphs to help the users track crop progress and health over a selected period of time. We will also allow specific users to run different intrusion detection algorithms on the data that has been collected. These algorithms will produce alerts for the user if there are any issues with the data collection, if there are any data values that are out of the ordinary, or if there has been an indicator of compromise.

### 2.2 INTENDED USERS AND USES

#### Intended Users:

1. Large Scale Farmers
2. Gardeners (Smaller family farms)
3. System Administrators
4. People who need to monitor plants from a distance (Corporate or Co-Op)
5. Owners of only house plants (Students)
6. People learning to take care of plants

#### Beneficial User Groups:

All the users listed above will benefit from our project since it will allow them to easily see the conditions of their plants and soil. Some of the benefits will include increased free time and less monetary overhead for monitoring their crops.

#### Intended User breakdown by persona and needs:

**Large Scale Farmers Persona:** Majority are white, older males. Their hobbies and interests include farming. They are motivated to make a living and feed the world. Their personalities are usually more business-oriented and value quantity and value of their produce.

**Need For this Product:** This product gives large-scale farmers the ability to collect more data about their field, get immediate alerts if problems occur, and track data trends over time.

**Gardeners (Smaller family farms) Persona:** Majority are male and own their family farm. Their hobbies include gardening and farming. Their motivations to do this are for retirement and their own personal interests. They are also very personally interested in the farm's success with a community focus.

**Need For this Product:** This product will allow gardeners to monitor water levels while away from the field, get notification reminders, and reduce time spent collecting data.

**System Administrators Persona:** SA's demographic would be technically oriented people making a living off this product. Hobbies for these people would be technology and agriculture. Their motivations would be to help farmers and earn a paycheck. Their personality would be very technical and problem-oriented. They would value customer satisfaction.

**Need For this Product:** This product will give System Administrators the ability to set permission levels for different users.

**People who need to monitor plants from a distance (Corporate or Co-Op) Persona:**

These people would be businessmen or women who may not be on the farm too often. Their hobbies would include analyzing data, discovering new products and ideas, and being around people. Their work motivation would be business oriented and growing their farms. Their personality would be outgoing and personable. They would value company satisfaction and profit.

**Need For this Product:** This product will give Corporate/Co-Op users the ability to analyze their farms and fields over time and get consistent, immediate data from the fields.

**Owners of only house plants Persona:** This demographic of people would be homeowners or students who want to improve their living conditions with plants. Their hobbies are very open-ended, but all would at least have some interest in plants. Their work motivation wouldn't be very relevant since they aren't making money from our product. They could have a wide range of personalities since the demographic is so broad. They would value their time and ease of information.

**Need For this Product:** This product will give house plant owners the ability to monitor water levels while being away from the house and get notification reminders.

**People learning to take care of plants Persona:** This demographic leans toward students. Their hobbies would include learning about agriculture and horticulture. Their work motivation would be related to school and their own interests. Their personalities would be very broad, but all would be eager to learn about plants and soil conditions. They would value and have aspirations of bettering the field of agriculture and horticulture.

**Need For this Product:** This product will give people learning about plants the ability to monitor water levels, get notification reminders, and easily see what needs the plants have as well as how to fulfill those needs.

### 2.3 REQUIREMENTS & CONSTRAINTS

#### Functional Requirements:

- IoT soil moisture sensor
  - Data collection
  - Medium-long range wireless transmitter (LoRa)
- IoT basestation
  - Data logging
  - Medium-long range receiver to sensors (LoRA)
  - Wifi data transfer to AWS cloud
  - Secure and consistent data logging and distribution

- AWS cloud
  - Data logging for verified data integrity
  - Data graphing to display to farmers
  - Data transfer using MQTT for IoT and SQS for application
  - Use of Amazon RDS, IAM, and Elasticsearch to support data flow and security
- Flutter application
  - Secure login and sign up authentication for each user
  - Ability to add/remove/rename sensors
  - Data visualization and representation of sensor data
  - Data logging and graphing of sensor data
  - Send and receive data from to the AWS cloud via Rest API

#### Non-Functional Requirements:

- IoT soil moisture sensor
  - Secure and consistent data collection and distribution (**constraint**)
  - The IoT sensor will be able to withstand different climate conditions like rain, snow, and wind
- IoT basestation
  - Secure and consistent data logging and distribution
  - The IoT basestation will be able to withstand different climate conditions like rain, snow, and wind
- AWS cloud
  - Data can be stored and accessible from any location and time
  - Stored data will be secure (**constraint**)
  - Deploy infrastructure via Terraform and potentially Drone
- Flutter application
  - Will be available for both Android and iOS based devices
  - Will display sensor data to the user via two pages, a scroll view and a graph view
  - Users will only be able to see their sensor's data

#### Physical Requirements:

- Minimal Footprint
  - Can not take up space needed for crops (**constraint**)
  - Can not impede tractor movement (**constraint**)
- Weather resistant
  - Water-proof (**constraint**)
  - Hot and cold temperature tolerance (**constraint**)
- High Visibility
  - Distinct colors to make it easy to locate in a field
  - Reflector strips at night and inclement weather

#### Resource Requirements:

- Batteries should last from planting to harvest
  - Minimal maintenance needed

**Environmental Requirements:**

- Batteries don't leak into soil
- Materials don't change soil nutrient levels

**UI Requirements:**

- Flutter application
  - Data is only accessible to platform user
  - System performs identically regardless of location
  - Application operates on multiple operating systems
  - Application can handle multiple users at the same time
  - Users can not view other users data

**Security Requirements:**

- IoT Sensors & Basestation
  - Data will be encrypted when stored on disk
  - Data will be encrypted end-to-end in transit
  - Sensors will be authenticated to the basestation using OTAA
- AWS Cloud
  - User roles will be properly separated
  - Minimum permissions will be allowed for all database accesses
- Flutter Application
  - Proper data sanitization will be enforced
  - Users will only be able to access their accounts data

**2.4 ENGINEERING STANDARDS****Flutter Application:**

- ISO/IEC 5055:2021: Software quality standards for source code to be measured upon. The standard considers the reliability, security, performance efficiency, and maintainability of source code.
- ISO/CD 18893: Applies to mobile elevating work platforms to protect users from personal injury, property damage, and accidents. It also establishes criteria for inspection, maintenance, and operation.

**Cloud Computing:**

- ISO/IEC 19944-1 (2020): Provides guidance on ensuring data flow and use as well as making cloud service agreements. Helps guide transparent use of data across the cloud services. Doing so will maintain trust between use parties and protects personally identifiable information (PII)
- AWS Well-Architected framework: Defines six pillars to help create the best design possible in the cloud. Assists with efficiency, cost-optimization, and sustainability.

**IoT Sensors & Basestation:**

- IEEE 802.15.4 is a standard that defines the operation of a low-rate wireless personal area network (LR-WPAN). It specifies the physical layer and media access control for LR-WPANs.

- IEEE 802.11ac is the most recent update to ac wi-fi standards. Our devices may use wi-fi, so following the IEEE standards will be important for compatibility purposes.
- LoRa Alliance is the governing body controlling specifications for the three classes of LoRa devices (A, B, C). It controls regional frequency bands, data rates, and security standards for LoRa devices.

## 3 Project Plan

### 3.1 PROJECT MANAGEMENT/TRACKING PROCEDURES

Our team plans on using an Agile project management style with the addition of a Gantt chart to help keep track of long-term goals. This is the most effective way to manage our team due to the fact that each aspect of our project is closely connected to one another. In order for the AWS team to complete their weekly goals, the IoT team must also be making strides in goals related to sending data to AWS. The same goes for the front end requiring AWS data to be used for visualization and display. Having quick weekly meetings and Agile-like sprints will allow us to ensure that teamwork is happening and that nobody is being seriously bottlenecked by anybody else.

The use of a Gantt chart on top of our standard Agile development will be useful to ensure we aren't spending more time than we have on specific aspects of the project. Each of the components of our system will require constant tweaking and development. However, we plan to track several major milestones with our Gantt chart. These milestones include a full connection between our IoT basestation and AWS, a full connection between AWS and our frontend application, and different data visualization goals.

The software we plan on using to manage these goals is a combination of GitLab for code management and version control as well as a shared KanBan board hosted on Mattermost to help keep our teams working efficiently and following our Agile development goal.

### 3.2 TASK DECOMPOSITION

Our team has broken all major projects down into more manageable tasks that can be delegated to individual team members. These decomposed tasks are shown below. They are grouped by overarching sprint goal. These tasks are also displayed in Figure 1: Gantt Chart.

- IoT Basestation connection to AWS
  - Being able to send and receive data securely
  - Storing data securely and making it easily accessible
  - Ensure access to user-specific data
- Flutter App connection to AWS
  - Being able to send and receive data securely
  - Allow transmission of data to legitimate users
- Frontend data visualization
  - Correctly displaying our data in a way that Farmers can understand and use
- Connect the IoT Sensors to the Basestation

- Setting up LoRaWAN protocol to connect the sensors and basestation
- Secure the AWS Cloud
  - Limited permissions
- Securing the IoT sensors and Basestation
  - End to end encryption
  - Exhaustive penetration testing
- Secure the Flutter App
  - End to end encryption
  - Exhaustive penetration testing

### 3.3 PROJECT PROPOSED MILESTONES, METRICS, AND EVALUATION CRITERIA

Major milestones for our project are grouped by large features in the end application. We've included specific and measurable metrics for success to ensure each of these milestones are accomplished to a satisfactory level.

- **Round trip data connection from IoT sensor to frontend**
  - Metric: Live updates when new sensor readings come in
  - Metric: Delay of no greater than 5 minutes
- **Secure login system for frontend**
  - Metric: Offensive security testing is unable to find flaws in login page
- **AWS data logging**
  - Metric: Data stored in the cloud
  - Metric: Data separated by user/group
- **Data analysis implemented in the cloud**
  - Metric: Data is being fed into AWS machine learning algorithm
  - Metric: Accurate assessments of the input data is returned from the algorithm
- **Front end data graphing**
  - Metric: Users are able to see useful visualizations of their data

### 3.4 PROJECT TIMELINE/SCHEDULE

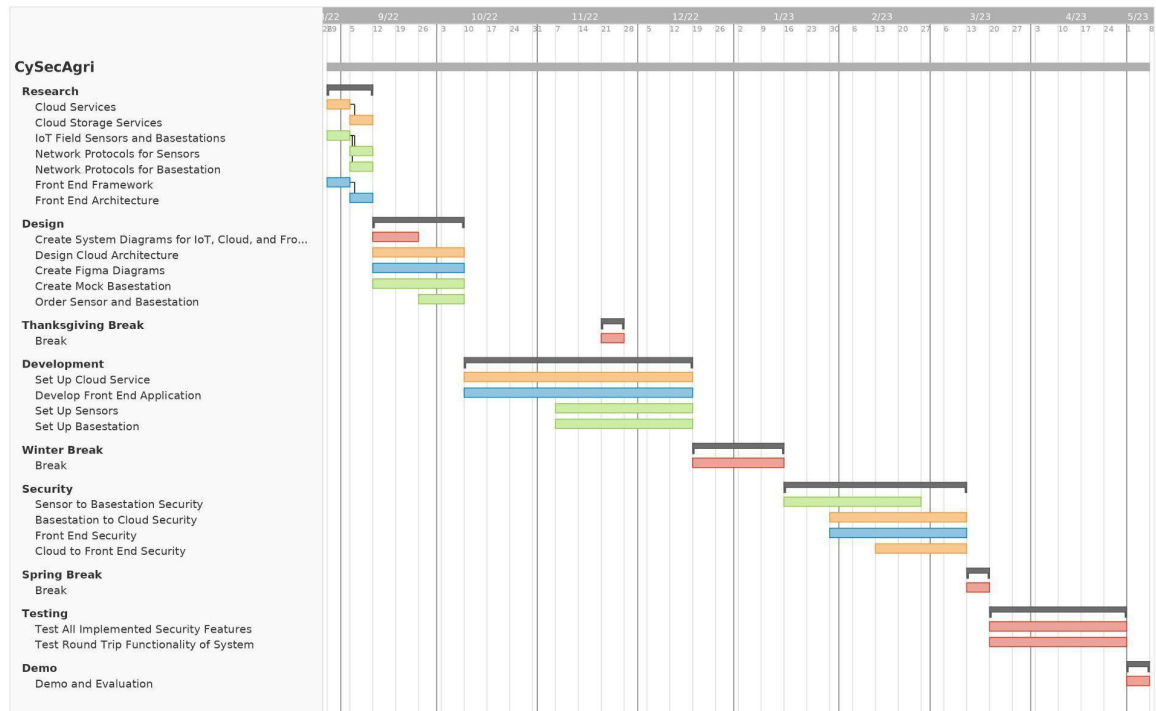


Figure 1. Gantt Chart (Appendix A)

### 3.5 OTHER RESOURCE REQUIREMENTS

#### SenseCAP S2104 - LoRaWAN® Wireless Soil Moisture and Temperature Sensor:

- Cost: \$130 (x2)
- 2 Sensors
- Battery Type: Standard D-size
- LoRaWAN Device Class A (least power consuming)
- No required modifications

#### Basestation:

- RaspberryPi Model 4B- x1
  - Cost: \$180 (market price)
- LoRaWAN Concentrator Chip RAK5146 Pi HAT- x1
  - Cost: \$120



## 4 Design

### 4.1 DESIGN CONTEXT

#### 4.1.1 Broader Context

Table 2

Area	Description	Examples
Public health, safety, and welfare	How does your project affect the general well-being of various stakeholder groups? These groups may be direct users or may be indirectly affected (e.g., a solution is implemented in their communities)	Increasing/reducing exposure to pollutants and other harmful substances, increasing/reducing safety risks, and increasing/reducing job opportunities.
Global, cultural, and social	How well does your project reflect the values, practices, and aims of the cultural groups it affects? Groups may include but are not limited to specific communities, nations, professions, workplaces, and ethnic cultures.	The development or operation of the solution would violate a profession's code of ethics, and implementation of the solution would require an undesired change in community practices.
Environmental	What environmental impact might your project have? This can include indirect effects, such as deforestation or unsustainable practices related to materials manufacture or procurement.	Increasing/decreasing energy usage from nonrenewable sources, increasing/decreasing usage/production of non-recyclable materials.
Economic	What economic impact might your project have? This can include the financial viability of your product within your team or company, cost to consumers, or broader economic effects on communities, markets, nations, and other groups.	The product needs to remain affordable for target users, the product creates or diminishes opportunities for economic advancement, and high development cost creates risk for the organization.

- Public health, safety, and welfare
  - IoT systems reduce the need for manual data collection, which can be physically taxing. This will increase safety for those who are injured or otherwise incapable of performing this task.
  - Securing our devices means increased safety and privacy for those who operate them.
  - Increased agricultural data collection and analysis could lower the barrier to entry for people interested in pursuing a career in agriculture.
  - IoT smart agriculture could increase crop yields and improve food security globally.
- Global, cultural, and social
  - Through securing agricultural IoT devices, we can increase trust with farmers and the general public.

- NIST security standards for IoT systems will need to be of utmost importance.
- Environmental
  - IoT sensors and gateways can be powered by renewable energy. As technology improves, our design can be adapted to renewable energy sources.
  - Without the need to produce batteries, less toxic pollutants will be released as a byproduct of production.
  - Our IoT devices are considered low-power and use as little energy as possible.
  - By taking advantage of distributed sensor networks, agriculturalists can better target specific field areas that may need additional resources instead of applying potentially harmful materials throughout the entire field.
- Economic
  - Product cost can be tailored to consumers depending on their needs (e.g. a small farm will require fewer sensors and cloud resources than a corporate farm).
  - Our IoT security implementations will increase trust in such systems and could expand their adoption into more farms.
  - With real-time access to sensor data in their fields, farmers could potentially increase their yields which might reduce the cost of food to consumers.

#### 4.1.2 Prior/Related Work & Solutions

A. Mishra, "Cloud-based multi-sensor remote data acquisition system for precision agriculture (CSR-DAQ)," Thesis, 2020.

- Mishra details a cloud infrastructure using MQTT as the key protocol for data transmission to and from the cloud. The primary focus of the paper was on cloud architecture. Our design is similar at the cloud level but will be a full physical implementation of an IoT system with data collection on soil moisture and temperature.
  - Our work will also be a combination of the LoRa RF/LoRaWAN protocols and MQTT. LoRa is our physical layer transmission from our sensor to our gateway and was not a component of Mishra's design
- Mishra created a web application as well as an Android application for data representation to the customer. The web application was developed using Python and the Flask framework, and the Android application using Java and Android Studio. Both were connected to AWS through an Elastic Beanstalk Container and Amplify, respectively.
  - Instead of creating a web application, our group decided to create a mobile application. We designed a Flutter application, to allow us to build an iOS and Android application simultaneously. Our application will also be connected to AWS via Amplify.

### 4.1.3 Market Survey

Below is a comparison of our product against two other products on the market. We compare the price of each product for two sensors and a basestation as well as the key features and drawbacks of each product.

Solution	Price	Key Features	Drawbacks
CySecAgri (our product)	\$300 (Basestation) \$260 (2 Sensors)  Total: <b>\$560</b>	<ul style="list-style-type: none"> <li>• Instant Access</li> <li>• Configurable</li> <li>• Scalable</li> <li>• Comm. range = ~1km</li> <li>• Versatile</li> </ul>	<ul style="list-style-type: none"> <li>• Beta (bugs)</li> <li>• Limited to LoRaWAN compatible devices</li> </ul>
HOBO	\$375(Gateway) \$680 (2 Sensors)  Total: <b>\$1,055</b>	<ul style="list-style-type: none"> <li>• Robust</li> <li>• Stable</li> <li>• High Data Rate</li> <li>• Plug-and-play</li> </ul>	<ul style="list-style-type: none"> <li>• Not customizable</li> <li>• Annual data plan</li> <li>• Comm. range = ~30m</li> </ul>
KaaloT (Just Basestation)	\$187 (Gateway)  Total: <b>\$187</b>	<ul style="list-style-type: none"> <li>• Versatile</li> <li>• Supports multiple MAC protocols</li> </ul>	<ul style="list-style-type: none"> <li>• Not user friendly</li> <li>• Doesn't include sensors</li> <li>• Out of reach of average user to set up</li> <li>• Not weatherproof</li> </ul>

### 4.1.3 Technical Complexity

Our design satisfies the following two benchmarks for technical complexity:

1. Our design consists of multiple components/subsystems that each utilize distinct scientific, mathematical, or engineering principles
2. The problem scope contains multiple challenging requirements that match or exceed current solutions or industry standards.

These benchmarks are met in the following ways:

- At an infrastructure level, our system combines a variety of network protocols. We will be using LoRaWAN for communication between sensors and gateways, the TCP/IP suite for communication with the network server (The Things Network), MQTT from the network server to AWS IoT Core, and AWS IoT Core's set of rules for communication with AWS Cloud Services

- After implementation, extensive testing will be performed on security protocols between sensors, gateways, the cloud, and our user application
- We will be employing different kinds of hardware/computing elements. LoRa capable sensors, a RaspberryPi with a LoRaWAN transceiver, and AWS Cloud Services will be our core components.
- Our cloud infrastructure will be built using the AWS services mentioned above. Each service will require appropriate scoping of permissions, cost optimization, and full functionality testing. The infrastructure will be deployed to AWS via Terraform and a GitLab CI/CD pipeline. Services used will include Lambda, DynamoDB, IoT Core, SQS, and IAM.
- Challenging requirements include:
  - Managing several different protocols between the elements of our infrastructure
  - Low-power authentication for sensors and gateways
  - Data analysis and anomaly detection at the cloud level
  - UX design for farmers viewing their data through our application

## 4.2 DESIGN EXPLORATION

### 4.2.1 Design Decisions

**The following are three crucial design decisions we have made:**

1. We chose a star topology and LoRa RF physical medium for sensor-to-gateway communication. This is less complex than mesh-network systems but reduces weak points in our future security protocols. LoRa RF was chosen as our communication medium because it operates on low power and has a range of ~1km. This decision will enable us to better protect against “imposter” attacks and implement fewer nodes across a large area.
2. We chose Flutter to develop our user interface. We chose Flutter because it reduces development time yet increases access to our application. This is possible due to Flutter providing the ability to have an iOS and Android application be generated from a single code base. With Flutter, we only have to develop a singular application, reducing development time, which can be used by both major mobile platforms, increasing access.
3. Cloud computing platforms were another area where our team had to reach a decision. We decided to use AWS to service our cloud computing needs. This decision is important to our project's success because cloud computing can be difficult to learn. If the platform we chose did not provide the necessary tools to complete our project, we would be at a loss of significant development time.

### 4.2.2 Ideation

One decision that needed to be made was: deciding which front-end framework we will use to develop our application. To make this decision, we used the ideation process of brainstorming. From our brainstorming, we came up with five different frameworks that we could use to build our application. Those frameworks were:

- React Native
- Android (Java)
- Flutter
- IOS (Swift)
- Creating an application for all platforms, i.e. web and mobile

After we brainstormed these five different choices, we decided to research each choice to see which framework would work best for our project. This research step took about a week, and once we had finished it, we moved on to our next step, which was creating a decision-making matrix.

#### 4.2.3 Decision-Making and Trade-Off

From our matrix, Flutter was the clear decision for us. Our criteria included how much time would be spent developing compared to our other options, how much experience we have with each option, and how easy it would be for our users to access our application. Flutter was tied in the time category, as most other options would require the same amount of development time. Flutter was also tied for first in the experience category, as our team as a whole had the same amount of experience with Flutter as React. Where Flutter took the lead was in the access category. With Flutter, we would be able to create an application for all major mobile platforms, allowing our users to easily access our application from their smartphones. This made Flutter our best option to use as a front-end framework.

*Table 3*

Selection Criteria	Criterion Weight	React		Android		Flutter		IOS		All	
		Score	Total	Score	Total	Score	Total	Score	Total	Score	Total
Time	.25	3	.75	3	.75	3	.75	3	.75	1	.25
Experience	.5	3	1.5	2	1	3	1.5	1	.5	1	.5
Access	.25	3	.75	2	.5	4	1	2	.5	5	1.25
<b>Total</b>	<b>1</b>	<b>9</b>	<b>3</b>	<b>7</b>	<b>2.25</b>	<b>10</b>	<b>3.25</b>	<b>6</b>	<b>1.75</b>	<b>7</b>	<b>2</b>

### 4.3 PROPOSED DESIGN

#### 4.3.1 Overview

Our project is focused on solving the problem of professional farmers and hobby gardeners needing to manually test, record, and chart IoT sensor data. The solution we will work towards will include the farmers deploying IoT soil sensors and a RaspberryPi basestation to aggregate data. From the basestation, data will be uploaded to Amazon Web Services for storage and processing. Users then access this data and information from our front-end application. This front-end app will give the user access to different charts and graphs to help the users track crop progress and health over a selected period of time. We will also allow specific users to run different intrusion detection algorithms on the data that has been collected. These algorithms will produce alerts for the user if

there are any issues with the data collection, if any data values that are out of the ordinary, or if there has been an indicator of compromise from a cyber security perspective. Below is a more detailed breakdown of the individual components and technologies we plan on using for our first deployment.

#### 4.3.2 Detailed Design and Visual(s)

Our senior design system consists of three key components: IoT Sensors & Basestation, cloud storage & processing, and the front-end application.

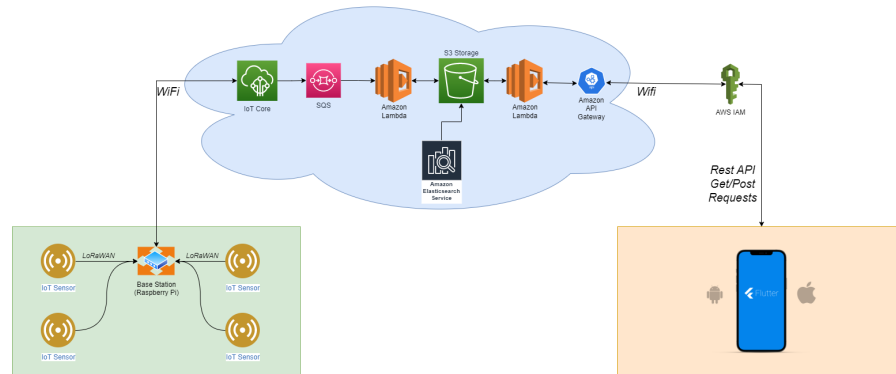


Figure 2. System Diagram (Appendix B)

#### IoT Sensors & Basestation

Starting with the IoT Sensors & Basestation, we have two sensor nodes connected to a basestation in a star network topology. This basestation will connect to a network server (AWS IoT Core) which will then interface with other AWS services utilized by our Cloud Development team. This system will be scalable by adding more basestations and nodes to the design.

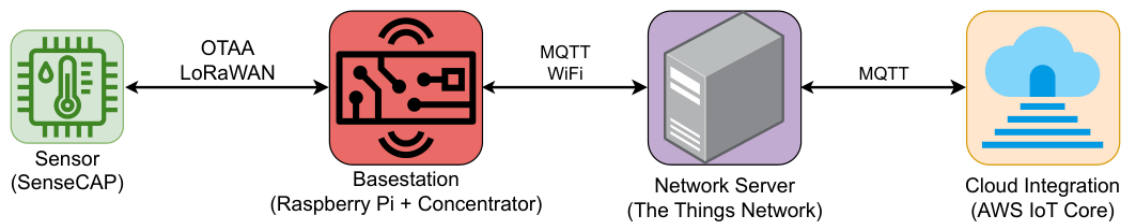


Figure 3. IoT Sensor and Basestation Diagram.

#### Devices

We have purchased two SenseCAP S2104 sensors which collect soil temperature and moisture data. These sensors are LoRaWAN enabled and will broadcast data to the basestation. The basestation is a RaspberryPi with a LoRaWAN receiver connected to it which aggregates the data and uses the CRC-16/Kermit algorithm to check for corruption. The RaspberryPi is WiFi enabled and will connect with AWS IoT Core via TCP/IP. AWS IoT Core is responsible for handling uplinks and downlinks from the sensors to the AWS Services the cloud team is using.

### Cloud Storage & Processing

Currently, the cloud storage option uses s3 with specified path names for each farm. The data travels from the basestation to the IoT Rule, which passes the message through a Simple Queue Service, reaching a Lambda function to break down the message. The Lambda function breaks down the message per hex values and converts them to decimal values that are then pushed into the database. In the future, this process may include the AWS ELK machine learning stack to ensure the data is validated.

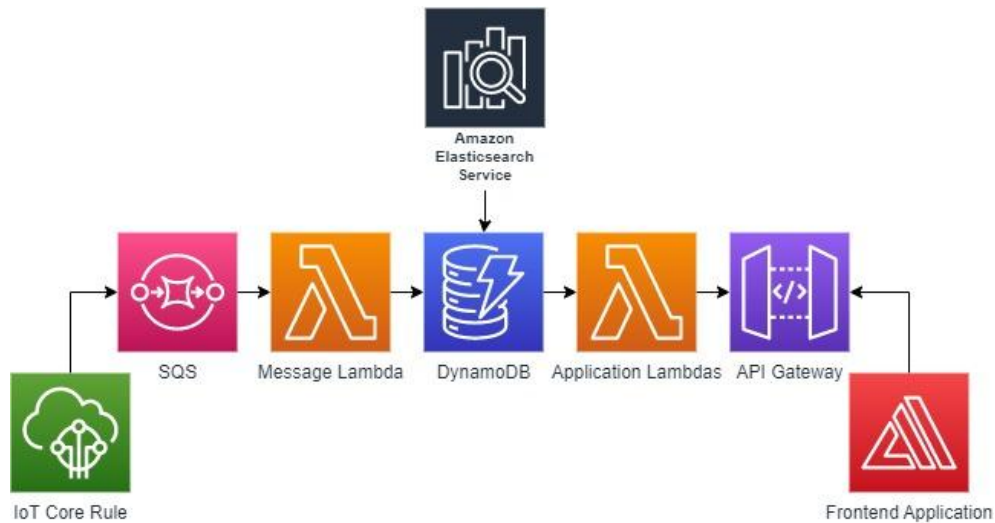


Figure 4. Cloud Storage & Processing Diagram.

### Front End Application

For our application we decided to use a Flutter application to build our application layer. After researching multiple different ways to develop, including making a decision matrix, we came to the conclusion that Flutter would be our best choice for development. To connect our app to Amazon Web Services, we used AWS Amplify, a backend service that connects our frontend Flutter app to AWS. To connect to our backend data, we will use Rest API which will allow for easy and efficient communication. We will use multiple different Widget libraries to build our application, including AppBar, BottomNavigationBar, and Containers. These libraries will allow us to build a quality application that looks modern and performs efficiently. We will display information about our sensor on multiple different pages that we will design for our users. These pages will be pre-designed using Figma.

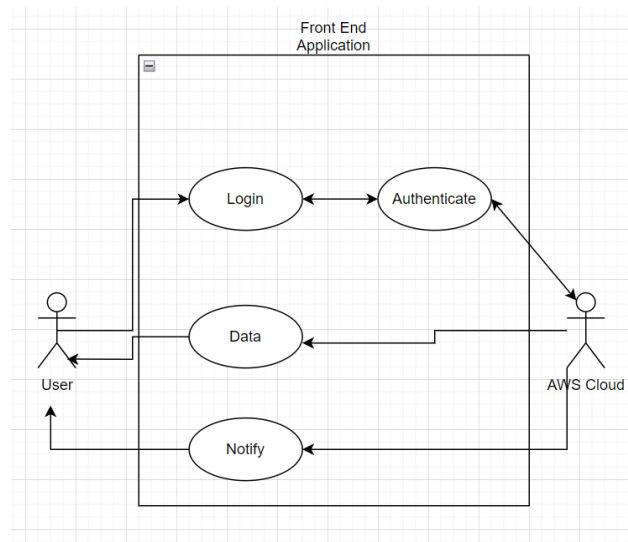


Figure 5. User & AWS flow diagram

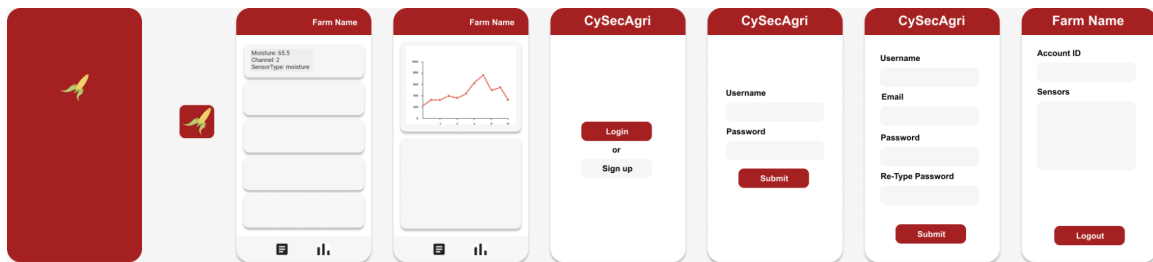


Figure 6. Figma designs for app pages (Appendix D).

### 4.3.3 Functionality

#### Initial Setup

Our end users would purchase our solution as a kit with a basestation and select sensors included. They would then place these sensors in different positions around their farm, garden, or home. Users will then download our front-end application to initialize their sensors for monitoring. Users will be able to name all of the sensors they have placed. There will be two different ways for users to view their sensor data. The first will be a scroll view that shows all the most recent sensor values as a scrollable list. The second way will be through a line graph where users will be able to see trends over time from their field.

#### Daily Use

After the initial setup, users will simply have an application on their phone, tablet, or computer that allows them to monitor the sensor's reading at any time of day from anywhere. Users will open the mobile application and be greeted with a list of the different sensors and locations they are monitoring. From there, they will be able to tap into any specific sensor and get live readings of the



sensor values as well as a graphical representation of the sensor readings from the last week. This time scale for historical data will be customizable by the user.

#### 4.3.4 Areas of Concern and Development

The current design satisfies physical requirements by keeping a small profile in sensor distribution. Sensors can be sparsely placed due to the use of a LoRa RF, star topology rather than a mesh network. The sensors we purchased are also weather and temperature resistant, with a rating between -40 and 185 degrees Fahrenheit. This will be sufficient for a majority of climates, including Iowa.

Our primary concern for our user and client needs is security. Security is very important to both our clients and our user. We know we can meet the main functionality of our product by delivering data from our sensors to user devices. Securing the data being transferred will be a little trickier. We must secure our data on all three levels of our product (Hardware, Cloud, User Interface), each of which will require unique security solutions.

#### *IoT Devices*

Currently we use CRC16/Kermit to ensure data integrity between our IoT sensors and the basestation. We also employ asymmetric encryption between our basestation and AWS to ensure data confidentiality when transmitting across the internet. Each sensor has a unique key and EUI that can be registered to the basestation to help verify sensors. These sensor nodes are simple to install, and farmers would not need a technician to add more sensors to their farms.

#### *Cloud*

To mitigate cloud security concerns, we plan to implement AWS Identity and Access Management policies so that users will not have unlimited privileges and are able to see other users' data. Also, we plan to use AWS Cognito to handle user authentication, this will help to make sure that only authorized users are able to access our cloud resources.

#### *Front End*

The front end will employ proper data sanitization to ensure users' security. On top of only allowing a strict set of characters to be entered for any user input field, user password and account details will be stored securely in AWS. This includes hashing passwords and transmitting all data over secure, encrypted channels.

### 4.4 TECHNOLOGY CONSIDERATIONS

We have made key technology choices within each of our subteams. This includes the protocol stack between gateways and sensors, our cloud service, and front end application framework.

#### *LoRa RF and LoRaWAN*

At the perception level, Sensor-Basestation communication is performed through LoRaWAN and LoRa RF. The link layer and physical protocol have a key advantage over most other low power transfer options: range. Bluetooth Low Energy (BLE) and Zigbee are both popularly used in IoT

applications for their low power usage. However, these protocol stacks are typically used in a mesh network configuration where a long signal range can be overkill. Limited to a maximum of 100 meters, BLE and Zigbee are not suited for the sparse dispersion of sensor nodes on a moderately large farm. Enter LoRa RF. LoRa RF has similar low-power usage at a significantly longer range of 1km. LoRaWAN is the accompanying link layer to LoRa RF, and our purchased sensor node and transceiver (for the basestation) are already LoRaWAN enabled. In addition, LoRa devices cost about the same BLE or Zigbee, making it financially viable for our project. In conclusion, LoRa-enabled devices were chosen for their low-power usage, long range, and affordability.

### *Amazon Web Services (AWS)*

There were three main options we thought about when choosing our cloud platform, AWS, Google Cloud, and Azure. We chose to go with AWS for a number of reasons. One of them was the fact that one of the members of the group is very familiar with AWS. Also, our client already had an AWS account, so it would be simple to give us access to the platform. Our client also mentioned the idea of using machine learning to implement an intrusion detection system, and AWS has a service that supports that functionality called ELK. We briefly entertained the idea of using other cloud platforms, but it quickly became clear that AWS was the best fit for our project due to previous experience and the services it offers.

### *Flutter*

When first considering what we wanted to create for farmers, we considered multiple different options. At the start, we debated whether we wanted a web or mobile application. After talking to farmers about their wants we decided that we would go with a mobile application. Once we decided on a mobile application, we had to figure out how we would create our mobile application. We looked at mobile frameworks that were easy to develop for both Android and iOS operating systems. And after making a decision matrix, we went with a Flutter mobile application. We chose Flutter because it is a fast, reliable framework that can simultaneously develop an Android and IOS mobile application.

## 4.5 DESIGN ANALYSIS

### *IoT Sensors & Basestation*

A survey of the marketplace revealed the SenseCAP S2104 was both fully compatible with our protocol needs, but was also waterproof and temperature resistant. This made these soil and temperature sensors the ideal choice for our design. While fabricating our own sensors may have allowed more flexibility, purchasing configurable sensors allowed more time to focus on security practices and round trip communication.

Our basestation design went through more trial and error. We needed to purchase a LoRaWAN concentrator module that could listen for and collect LoRa RF packets from the sensors. Through experimenting with different LoRaWAN concentrator chips, we settled on the RAK Wireless 5146 PiHAT kit for its reliability and form factor. The concentrator chip connected to a baseboard that was compatible with a RaspberryPi Model 4B. Using a RaspberryPi allowed us more control over the basestation and also expanded the testing and debugging we could perform.

Lastly, we chose to use The Things Network (TTN) as a network server to aggregate data and schedule downlink packet delivery to individual sensors. Since both the LoRaWAN concentrator chip and the sensors were compatible with TTN, we established student accounts and created an application to accept data from our end devices. We placed an integration with AWS IoT Core in our TTN application which allowed our sensor data to reach our cloud infrastructure.

### *Cloud*

Our cloud implementation has evolved multiple times throughout our development. We began with basic lambda functions and a DynamoDB for simple prototyping. Then, we briefly used S3 for storing all information but quickly realized the limits to scalability and accessibility. Finally, using AWS Amplify we were able to design a data model which is deployed using DynamoDB. This allows us to update the data model dynamically such as adding new fields while also managing data relationships such as one-to-one or one-to-many. Along with this, amplify assists with connecting this data model to the application through AWS AppSync using GraphQL.

### *Front End*

In the beginning, front-end development has been focused on establishing a connection between the app and AWS. Getting that established connection took longer than expected. We had originally planned to use AWS's Amplify service, but after spending the timespan of a week and a half with no success, we decided to move away from Amplify and create our own RESTful API system to connect to AWS. However, this was not a long-term solution. Once that was complete, our attention was turned to displaying the sensor data in a user-friendly way. We were able to build two different screens to display data. The first screen is a graph view that tracks the sensor's data values of time. The second is a scroll view where users can see their sensor data with timestamps to know exactly when their data has been sent. This was further expanded on by adding a filter system where users can filter their sensor data to any timespan they desire.

The next step for the front end was to figure out how to securely set up an account system for the users. We tried a few different ideas but did not want to implement an entire authentication system when there are already made and more secure APIs than anything we could build. This led us back to AWS's Amplify service. We decided to use its authentication system to authenticate users. This allowed us to speed up our development process significantly and gave us useful API calls to change usernames, passwords, and emails securely. When we got the Amplify authentication set up, we decided to move our RESTful API over to Amplify. This allowed our users to securely make their API requests and allowed for better integration for our AWS team.

The final step that we were able to implement on our front end was sensor addition, deletion, and rename functionality. We added a profile page where, along with seeing their username, email, and account ID, users could see their list of sensors. They can edit any of their sensors at any point and see each sensor's battery percentage. We also have implemented a downlinking feature that allows users to change the interval rate at which their sensor will send data measurements. This downlinking feature is the first step to more responsive app functionality that could be developed further in future research.

## Overview

We took many approaches to data storage before landing on using AWS DynamoDB through designing a data model. This process included monitoring the packets we were receiving and understanding the need for our data to be used and shared between users.

We are also being very cautious about the types of IoT radios and sensors we are selecting for the project. There are a lot of subtle differences between the parts, and we need to make sure they all work together. These issues will become harder to resolve as the semester goes on, as part orders can take several weeks to fill.

## 5 Testing

### 5.1 UNIT TESTING

Our CRC-16/KERMIT checking class will be tested for accurate packet validation. This will be tested by using a distribution of valid and invalid packets to give the class. CRC validation needs to operate at 100% accuracy, so evaluating accuracy is actually rather simple.

Inside our AWS infrastructure, ensuring that the data is passed between services is crucial. By running basic test payloads in the AWS console, we can see if the data is successfully passing through. Our packet analyzer lambda needs to receive information from the IoT Core through the SQS so we can run our mock basestation script to send information to the IoT Core. From there, using CloudWatch to view the logs, we can see whether the data reaches the analyzer lambda successfully or what errors may have occurred. As well as testing in the cloud, these scripts can be tested locally to see how they react because we know what the packet format will look like.

Additionally, our front-end application incorporates unit testing to ensure our design's accuracy. The Flutter framework has a nice directory in the main application directory to create and run unit tests as well as widget tests called `/test`. All that is needed to do to be able to run these tests is to add the test or flutter\_test dependency to the pubspec, Flutter's package manager file. To add even more functionality to our tests, we also use the Mockito dependency. One downside to unit and widget testing for Flutter is that widget trees can become very large. This problem means we must mock many widget functionalities within each test, making them long and abstract. Because of this downside, we have decided to only create tests for only certain widgets and focus more on manual testing the larger widgets. This will save us time by not having to mock up so many different functionalities to the point where some tests become meaningless.

### 5.2 INTERFACE TESTING

Our two biggest interfaces for this application would be our IoT sensor network and our AWS infrastructure. Due to the interconnectedness of these two interfaces, it will be essential to not only test them in their own unit test but also test them together as one large interface. These tests will include reverse engineering our basestation code and attempting to send malicious or malformed packets to it from our sensors. If we are able to get one of these modified packets through the basestation it will be put into our AWS database and could potentially cause more damage to the stability of our application.

Just as we test the data flow from our IoT sensors to AWS, it will be important to understand the ways in which data from AWS can affect our sensor network. We will be attempting to implement a naming system where users of our application can name and digitally place sensors over a map image of their land to better keep track of the positions their readings are coming from. This feature will require data to be sent from our user configuration database in AWS to the sensor and interact with them. This will require a large amount of manual testing to ensure AWS is not able to put the sensors in some state that is unrecoverable or requires physical interaction with the device (hard resetting it).

### 5.3 INTEGRATION TESTING

One significant integration path on the hardware end of our IoT system is Sensor-to-Basestation communication. Due to constructing our own basestation, a lot of testing will need to be completed on both communication paths of the basestation to ensure our implementation is functioning properly. Due to the nature of the LoRaWAN protocol, we can expect about a 10% packet loss rate as a reasonable amount for our system and its applications. Communication testing outside with obstacles (or the actual crops they're meant to simulate) should be completed for reliability. If our packet loss rate is consistent with the 10% margin, it will be considered suitable for our agricultural application. Since the packet structure created by our sensors have a CRC code attached to it, accurate transmission can also be examined.

The next integration path is between the basestation and the AWS IoT core. This will operate over the TCP/IP stack and use an MQTT application protocol. Reliability testing will need to be completed at this stage to evaluate the functionality of our MQTT implementation. Again, due to constructing our own basestation, testing will be more focused on the basestation and its configuration with the AWS IoT Core. Due to our IoT system's infrequent data collection, speed and latency are not as important to consider during testing. MQTT (particularly Quality of Service 2) is considered very reliable, and our pathway will need to be held to that standard. To test this, we can manually disrupt the network connection of the basestation and evaluate how it recovers. If, after the disruption, the basestation still accurately sends its data and it is properly received by the IoT core, then our system can be considered reliable for data transmission to the cloud.

The final integration path is between the AWS API gateway and our Flutter front-end application. This connection is made possible through AWS Amplify's GraphQL queries and received by our front-end application. To test the functionality and correctness of this connection, we created widget tests in our front-end application. These tests are similar to how our unit tests for our front-end application are implemented. We added the Flutter widget\_test dependency to our pubspec, package manager file, and after that, all we have to do is run the tests we created to verify that they have passed. This will allow us to test the functionality of our application to see functionality like if buttons are being triggered correctly and pages are being displayed properly.

### 5.4 SYSTEM TESTING

Our full loop system consists of three smaller subsystems in the IoT sensors, AWS infrastructure, and frontend application. For system testing, we will need to ensure that we can have data transferred through our entire system. Once we can do that, we will be able to see how the system interacts as a whole. This form of testing will be manual in nature because of the size of our system. Each form of testing prior to this will be integral to getting a foundation for our system testing. Unit

testing will allow us to look at a specific aspect of our system, interface testing will help us see how two parts connect to each other, and integration testing will show us how those paths of dataflow in our system move.

## 5.5 REGRESSION TESTING

Since many of our development changes will be centered on the basestation, consistent regression testing to determine that new security additions have not broken our sensor-basestation-cloud communication path for uplinks and downlinks. A few uplink and downlink transmissions should be sufficient for smaller modifications and can be expanded for larger design improvements for more thoroughness.

The next way our project will regression test is by creating a Continuous Integration and Continuous Deployment pipeline. This will allow us to test our new code on our old unit and integration tests. We will create these CI/CD pipes in Gitlab, and once they are set up, they will run every time we push our code to Gitlab. If one of our old tests fails, the CI/CD pipeline will also fail, indicating to us that there has been a regression. This will help us with the AWS side of our project as well as in our front-end application.

## 5.6 ACCEPTANCE TESTING

We plan on keeping track of whether the test was successful or not and, if not, what specifically the error was. In order to monitor which areas of our project we have tested, we will categorize the attacks and mitigation techniques based on the relevant area of the project that they apply to, such as IoT devices, AWS infrastructure, or frontend applications.

To make sure that our client is aware of our testing progress, we have been providing live demonstrations during our weekly meetings. Going forward, we will continue this but also increase communication so that our client is always aware of our progress. Our client has stressed that for a successful project, we should ensure that the data communication is sent securely from the sensors to the user through Amazon Web Services. To make sure that our client's needs are satisfied, we focus not only on securing the inter-component communication but also making sure data is secure inside each component, such as information stored in databases or adding more sensors.

## 5.7 SECURITY TESTING

The bulk of our second-semester testing falls into the category of security testing. In order to ensure comprehensive coverage of our application's attack surface, we created several supporting documents to help map out vulnerabilities and security concerns in all stages of our application. These documents can be found in our appendix.

### 5.7.1 Threat Scenarios and Mapping

In order to most effectively all aspects of our physical and cloud-based systems we broke attacks up into more approachable scenarios and then mapped those scenarios onto a risk impact matrix as shown in Figure 7. The scenarios covered are as follows:

- S1. An attacker is able to break authentication flow and access data for arbitrary user accounts
- S2. An attacker modifies account information between the app and AWS
- S3. Our API Endpoints are flooded with requests resulting in a DoS/Fees
- S4. A user modifies their account permissions inside of AWS for privilege escalation
- S5. Unauthorized data is sent from imitation sensors/basestations to AWS
- S6. AWS Developer Account is compromised
- S7. Basestation user account is compromised
- S8. Common vulnerability scans are run against assets






		Impact		
		LOW	MEDIUM	HIGH
Likelihood	HIGH	 Medium risk (3)	High risk (4)	 Highest risk (5)
	MEDIUM	Low risk (2)	Medium risk (3)	 High risk (4)
	LOW	Lowest risk (1)	 Low risk (2)	 Medium risk (3)

Figure 7. Risk Impact Matrix for CySecAgri (Appendix E?)

These scenarios cover all of our application's attack surfaces and provide us with an actionable plan for testing and remediation of any vulnerabilities. With this risk impact matrix, we decided anything colored yellow or above would need to be tested and anything orange or above must be remediated if any vulnerabilities are found.

Using this risk matrix we concluded anything medium or higher would need to be tested and anything discovered to be high risk or worse should be remediated.

### 5.7.2 Mobile Application Testing

For testing scenarios S1 through S3, we took a multipronged approach to mobile application testing. We first carried out dynamic analysis by compiling a debug version of our application that allows us to proxy all API and authentication requests through an analysis app of our choosing. We then performed static analysis through the decompilation of our final released application. This gave us further insight into the metadata and hardcoded values that an attacker would have access to after downloading our app. The results of these tests are detailed in section 5.8.1.

### 5.7.3 AWS Infrastructure Testing

To test scenario S4 we examined the existing roles and policies across our infrastructure including Lambda, API Gateway, IoT, DynamoDB, and more. Many of our roles and policies were created with full access to whatever action and whatever service it needs.



For scenario S6 we did an assumed breach of a developer AWS account. Given access to a developer account, an attacker has full access to every service in AWS. Upon logging in, the attacker can view recently used services, cost analysis, and more.

#### 5.7.4 Basestation/Sensor Testing

For our basestation testing we performed a black box penetration test against the raspberry pi running our basestation software. In this test, we followed a standard four-step testing plan that included: scanning, enumeration, exploitation, and documentation. The details of this test are provided in section 5.8.4. Due to the highly specialized equipment and physical access requirements needed to attack the sensors in our network, it was deemed not worth the time investment to test them manually.

## 5.8 RESULTS

The results of all of our different tests have been compiled into table 4.

*Table 4*

Test Name	Description	Results	Actions Taken
GraphQL Injection Test	This test ensures that an unauthenticated user cannot arbitrarily access user data stored inside of GraphQL	Critical Vulnerability Found	Immediate remediation via authentication changes
Mobile Application Reverse Engineering	This test is used to understand what data is available to users that download our app. Reverse engineering and disassembling the compiled APK for our app will reveal any hardcoded credentials or insecure secret storage that may not be obvious during normal development.	Medium Vulnerability Found	Noted for future development
Basestation External Penetration Test	This penetration test is used to confirm all necessary security precautions have been taken to ensure an attacker can't gain a foothold on our IoT networks basestation. The steps involved in this test are as follows: scanning, versions enumeration, exploitation, and privileged escalation.	Critical Vulnerability Found	Immediate remediation via SSH configuration changes
Basestation Data Sniffing Attack	In this scenario we would validate that data transmitted between the sensors and the basestation is encrypted. This test requires specialized radios	Incomplete	Requires additional hardware resources to perform correctly



	to be able to capture the frequencies sent out from our sensors.		
AWS Developer Account Breach	This test assumed a breach of our AWS developer accounts by a malicious actor. The breach enabled us to understand what damage could be done to our infrastructure if a user could access the accounts.	Medium Vulnerability Found	Current actions included adding MFA to developer accounts. In the future, dividing developers into scoped user groups would further ensure developer account and infrastructure security
AWS Lambda Privilege Testing	Testing included examining roles and policies attached to resources such as Lambda to understand the scope. Often time these resources are able to take action across resources and attackers could leverage this to query or destroy other data.	Medium Vulnerability Found	Scoping of policies to the only resources in which they must interact with

### 5.8.1 Mobile App Testing Results

#### Dynamic:

Detailed instructions for setting up this testing environment can be found in Appendix F. After establishing a proxy server for all of our mobile app API requests we began seeing what data was sent as a user browsing the app. We found that the login service provided by AWS offered robust security in the form of challenge-response based authentication. After the login, it was noticed that there was a query request being sent to our GraphQL API. The body of the request is shown below in Figure 8. This query was being made using a static API key stored within the app's source code.

```
query GetUserSensorReadings {  
  getUserData(id: \"465d1771-b947-4d9a-a966-72f21f3b1f60\") {  
    UserSensors {  
      items {  
        friendly_name  
        SensorReadings(limit: 10000) {  
          items {  
            type  
            value  
            time  
          }  
        }  
      }  
    }  
  }  
}
```

Figure 8. GraphQL sensor readings query

This key was the same for all accounts and all login sessions, meaning there was no way to distinguish between different user accounts or permission levels. Because of this, we were able to alter the query being sent to print out the email and account information of all of the users on our app. The output of this malicious query is shown in Figure 9.

```
"data":{
  "listUserData":{
    "items":[
      {
        "id":"z",
        "email",
        "farm_name":null,
        "preferred_name":"test884"
      },
      {
        "id":"a",
        "email",
        "farm_name":"test_Farm",
        "preferred_name":"testing"
      },
      {
        "id":",
        "email",
        "farm_name":null,
        "preferred_name":"rian1"
      },
      {
        "id":",
        "email",
        "farm_name":"IceBear Farms",
        "preferred_name":"IceBear"
      },
      {
        "id":",
        "email":"jo",
        "farm_name":null,
        "preferred_name":"Joe2"
      }
    ]
  }
}
```

Figure 9. Sensitive information dump through GraphQL

This vulnerability fell into scenario S1 in figure 7 and was required to be remediated immediately as per our previous discussion on risk tolerance.

#### Static:

The frontend application can be downloaded on Android devices in the form of an APK file. One attack vector is decompiling the source code contained in the file and examining the contents for potential vulnerabilities. To initiate the reverse engineering, we used the open-source tool JADX to decompile the latest version of our APK. The first interesting piece of information we were able to find was contained in the AndroidManifest.xml file, which can be seen in Figure 10.

```

<?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1001" android:versionName="1.0.0" android:compileSdkVersion="31"
7 <uses-sdk android:minSdkVersion="21" android:targetSdkVersion="31"/>
11 <uses-permission android:name="android.permission.INTERNET"/>
14 <uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
15 <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
17 <application android:label="CySecAgri" android:icon="@mipmap/launcher_icon" android:name="android.app.Application" android:usesCleartextTraffic="true"
23 <activity android:theme="@style/LaunchTheme" android:name="com.example.flutter_application_3.MainActivity" android:exported="true" android:launch
38 <meta-data android:name="io.flutter.embedding.android.NormalTheme" android:resource="@style/NormalTheme"/>
42 <intent-filter>
43 <action android:name="android.intent.action.MAIN"/>
45 <category android:name="android.intent.category.LAUNCHER"/>
42 </intent-filter>
23 </activity>
52 <meta-data android:name="flutterEmbedding" android:value="2"/>
56 <uses-library android:name="androidx.window.extensions" android:required="false"/>
59 <uses-library android:name="androidx.window.sidecar" android:required="false"/>
63 <service android:name="com.amazonaws.mobileconnectors.s3.transferutility.TransferService" android:enabled="true"/>
66 <service android:name="com.amplifyframework.storage.s3.service.AmplifyTransferService" android:enabled="true"/>
70 <activity android:theme="@style/OverlayActivity" android:name="com.amplifyframework.devmenu.DeveloperMenuActivity"/>
73 <activity android:theme="@android:style/Theme.Translucent.NoTitleBar" android:name="com.amazonaws.mobileconnectors.cognitoauth.activities.CustomTs
79 <uses-library android:name="org.apache.http.legacy" android:required="false"/>
17 </application>
2 </manifest>

```

Figure 10. *AndroidManifest.xml*

Various library and SDK versions can be found here, which an attacker could look up in order to find pre-existing vulnerabilities with those specific versions. It also shows that our application is communicating in cleartext with HTTP API endpoints so that traffic can be sniffed and read without having to decrypt the data.

Additionally, we found a more pressing issue in the `libapp.so` file. It contained the URL of our API endpoint, our static API key, as well as a bunch of other information about our AWS resources. A segment can be seen in Figure 11.

```

"UserAgent": "aws-amplify-cli/2.0",
"Version": "1.0",
"auth": {
  "plugins": {
    "awsCognitoAuthPlugin": {
      "UserAgent": "aws-amplify-cli/0.1.0",
      "Version": "0.1.0",
      "IdentityManager": {
        "Default": {}
      },
      "AppSync": {
        "Default": {
          "ApiUrl": "https://6glq2r7ccvebnki5bi5mz2wm2e.appsync-api.us-east-1.amazonaws.com/graphql",
          "Region": "us-east-1",
          "AuthMode": "API_KEY",
          "ApiKey": "da2-4r2ch2nqpfodpdmis5krp3cav4",
          "ClientDatabasePrefix": "flutterapplication3_API_KEY"
        },
        "flutterapplication3_AWS_IAM": {
          "ApiUrl": "https://6glq2r7ccvebnki5bi5mz2wm2e.appsync-api.us-east-1.amazonaws.com/graphql",
          "Region": "us-east-1",
          "AuthMode": "AWS_IAM",
          "ClientDatabasePrefix": "flutterapplication3_AWS_IAM"
        }
      }
    }
  }
}

```

Figure 11. *Output of libapp.so*

Using the data discovered, an attacker would not even need to create an account in order to send the malicious query seen in the previous section. This vulnerability also falls into scenario S<sub>1</sub> since it

is a continuation of the GraphQL misconfiguration and was fixed with the same remediations. Additionally, we can compile our APK in an obfuscated mode to prevent users from being able to see all of the source code in plaintext, which would alleviate many of the issues seen here.

### 5.8.2 AWS Testing Results

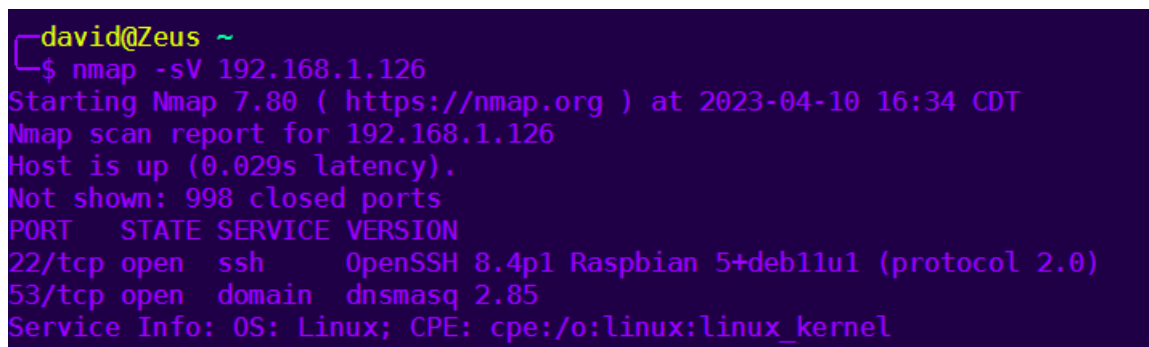
The outcome of our AWS testing was that we needed to review the scoping of our users, roles, and policies. The current configuration of AWS accounts for team members allows full unscoped access to every service. To correct this, users should be divided into the three divisions of the project; IoT, AWS, Application. From here, access can be narrowed down to the services related to the division. For example, IoT users would be able to access AWS IoT, Lambda, CloudWatch, AWS users would have more broad access, and Application users would have access to Amplify, API Gateway, Lambda, DynamoDB.

To solve issues with interservice permissions, we examined our existing policies attached to resources. Many of our services were able to invoke any Lambda function or access any IoT rule. The solution was to examine the Amazon Resource Name (ARN) of the resources that needed to interact. By using the ARN in the access policy of the resource, we can implement the concept of least privilege for our resources.

### 5.8.3 Basestation Testing Results

#### Scanning:

The scanning process for this test involved running the command line tool *nmap* against the IP address of the pi. The full results of this will be included as Figure 12, but the only port of interest we identified was TCP/22 running SSH.



```
david@Zeus ~
└─$ nmap -sV 192.168.1.126
Starting Nmap 7.80 ( https://nmap.org ) at 2023-04-10 16:34 CDT
Nmap scan report for 192.168.1.126
Host is up (0.029s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.4p1 Raspbian 5+deb11u1 (protocol 2.0)
53/tcp    open  domain   dnsmasq 2.85
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Figure 12. Nmap Scan Results

#### Enumeration:

In the enumeration phase we began by identifying the version of SSH and the authentication mechanisms available to an attacker. The SSH version identified was 8.4p1 which is a standard up-to-date version of SSH with no public exploits. Next, it was noted that SSH was configured to accept both passwords and keys.

## Exploitation:

With the information that SSH accepts standard password authentication we set out to build a custom wordlist for password spraying against the basestation login. We used the publicly available tool *Cewl* to scrape our team website and put together a simple list of possible passwords for our login. Given that this was running on a raspberry pi we decided the username for spraying would be the default *pi* user which comes set up on all raspberry pi installations. With these pieces of information, we used *Hydra* for performing the brute force. More specifically, the command run was `hydra -l pi -P cewl-passwords.txt ssh://192.168.1.205`. After allowing to run for a few minutes it was able to find the correct login was *pi:cysecagri* which is an incredibly weak password for standard ssh access. After logging in, we found that the default pi user also had root access to everything, so the basestation was completely compromised. This attack falls into *S7* in *Figure 7*, so the decision was made to remediate this vulnerability immediately. Details of this remediation can be found in section 5.9.

## 5.9 REMEDIATIONS

### GraphQL Injection Vulnerability

This vulnerability was the most severe issue found during our security testing. Because of this, we decided not only to remediate it as quickly as possible but also to ensure our GraphQL API is using the industry's best practices for user authentication. Our proposed remediation centers around switching our AWS infrastructure from using a statically generated API key for user authentication to our Cognito user pool. This change will give us the granularity to control permissions for API queries on a user-by-user basis. With this in place, an individual user could only ever run a single query against their own account to load their own sensor values. Any attempts to access other user account data would not only be denied but would also generate security logs for our team to analyze and determine appropriate action for the offending account.

As of writing, this remediation has not been implemented yet. The procedure for making these changes has been documented extensively, however, the actual implementation has required more troubleshooting than our team currently has the bandwidth for. Any further development on this project will require this remediation to be fully implemented for scaling to any additional users.

### Mobile App Reverse Engineering

While nothing incredibly pressing came out of the static testing, a few good practices could have been followed that we missed. The first and most important would be to obfuscate our code when creating the APK to prevent users from easily being able to see the underlying source code. This would stop an attacker from seeing all of the frameworks and libraries we are using for the application which should also be regularly checked to make sure there are no known vulnerabilities contained in them. This can easily be implemented by adding the obfuscate option when compiling the APK file. Additionally, in the future binary protection could be implemented to prevent an attacker from changing the applications code and enabling unapproved functionality. This can be done with things like tamper detection, encoding, and encryption.

### Basestation SSH Vulnerability

Due to the severity and possible impact of this vulnerability it was decided that immediate remediation would be required. There are a plethora of ways to go about mitigating the risk posed by a weak SSH login on a server. For this situation, we decided on a three-pronged approach to reduce risk as much as possible while still maintaining maintenance access to the basestation. The steps needed for our solution are as follows:

1. Move SSH to an ephemeral port
2. Disable password authentication and replace it with key-based authentication
3. Generate a secure maintenance key using asymmetric cryptography

We began by moving SSH off of its standard port (TCP/22) and having it run on TCP port 12317. This will cause common network scans to report no SSH port open, reducing the likelihood of bots attacking this server with brute force attacks. After this change, we edited the *ssh.conf* file on the basestation to only accept trusted SSH keys. This will completely remove the ability to attempt online brute force attacks as demonstrated in section 5.8.3. Finally, we generated a strong public private key pair and placed the public key onto our basestation. This allows our developers with the appropriate private key to access the server to perform updates or changes without opening the server up to further attacks against SSH.

## 6 Implementation

Current implementation of our design includes progress towards round-trip communication between the different layers of our IoT system. We established communication between our Front-End application, AWS Cloud Service, and a mock-basestation written in Python. Due to delayed part orders, we were unable to fully implement a custom LoRaWAN basestation during the Fall 2022 semester. Thus, our first task in the spring will be to fabricate the basestation to enable full round-trip communication. The LoRaWAN Concentrator chip we are waiting for is a Hardware Attached to Top (HAT) design that will facilitate easy setup and low noise with our RaspberryPi. Furthermore, we have manufacturer-provided instructions on how to set up the basestation using the HAT chip in conjunction with our RaspberryPi model. With this in mind, we estimate this to have a quick turnaround time. At the conclusion of the basestation's integration into our IoT system, we will begin security implementation along both communication paths attached to the basestation.

In the meantime, our other subteams will begin implementing our security designs in the cloud and front-end application. These tasks do not rely on having complete round-trip communication implemented. A specific breakdown of these tasks can be found in Section 3.

### 6.1 IMPLEMENTATION SINCE 491

Our overall design has changed slightly since the completion of CprE 491. The first and most major change came in the form of our user data storage mechanism. Originally we believed Amazon's S3 storage system would be sufficient to store all of our user data and sensor readings. However, after running a test instance of this solution for a few weeks, we realized that this approach would not be able to scale well with our applications' full deployment size. To address this, we pivoted to using another set of AWS services, AWS AppSync and DynamoDB. AWS AppSync uses GraphQL in AWS for quick and efficient queries of large data sets and is often used to load social media posts for a

given user account. We use it to load all sensor readings for all the sensors owned by a single account in our app. This not only increased query response time but also provided a much easier path for scaling up our deployment to 10's and 100's of sensors and accounts.

Our frontend architecture has also changed since the prototype we demonstrated at the end of 491. We used to handle the AWS API calls and authentication manually through native Flutter HTTP requests. As we began to scale up the number of requests and users on our platform, we realized this method was not maintainable. To correct this, we switched over to using the AWS Amplify library developed for Flutter. This library has built-in methods for automatically authenticating and requesting a plethora of different AWS resources. It has saved us countless hours of duplicate work as well as providing centralized handling for security measures such as secret storage and user account authentication.

The IoT infrastructure design has not changed significantly from our proposed design in CprE 491. We have experimented with different versions of the basestation's concentrator chip, but all protocol choices have remained the same. We did add downlink functionality (i.e., messaging from the front end application to end devices) after the usefulness of this feature was pointed out by an industry expert. Our end devices can be rebooted or change their data collection interval from downlinks. However, this functionality translates to any end device in the system. For example, a useful feature would be to remotely activate an irrigation mechanism when soil moisture is measured under a certain threshold. This concept could be greatly expanded to incorporate several automated processes to monitor and maintain ideal growing conditions for farms of any size.

## 7 Professional Responsibility

This discussion is with respect to the paper titled "Contextualizing Professionalism in Capstone Projects Using the IDEALS Professional Responsibility Assessment," *International Journal of Engineering Education* Vol. 28, No. 2, pp. 416–424, 2012

Table 5

Area of Responsibility	Definition	NSPE Canon
Work Competence	Perform work of high quality, integrity, timeliness, and professional competence.	Perform services only in areas of their competence; Avoid deceptive acts
Financial Responsibility	Deliver products and services of realizable value and at reasonable costs.	Act for each employer or client as faithful agents or trustees
Communication Honesty	Report work truthfully, without deception, and is understandable to stakeholders	Issue public statements only in an objective and truthful manner; Avoid deceptive acts.
Health, Safety, Well-Being	Minimize risks to safety, health, and well-being of	Hold paramount the safety, health, and welfare of the



	stakeholders	public.
Property Ownership	Respect property, ideas, and information of clients and others.	Act for each employer or client as faithful agents or trustees.
Sustainability	Protect environment and natural resources locally and globally.	
Social Responsibility	Produce products and services that benefit society and communities.	Conduct themselves honorably, responsibly, ethically, and lawfully so as to enhance the honor, reputation, and usefulness of the profession.

### 7.1 AREAS OF RESPONSIBILITY

Pick one of IEEE, ACM, or SE code of ethics. Add a column to Table 1 from the paper corresponding to the society-specific code of ethics selected above. State how it addresses each of the areas of seven professional responsibilities in the table. Briefly describe each entry added to the table in your own words. How does the IEEE, ACM, or SE code of ethics differ from the NSPE version for each area?

Comparing to IEEE Computer Society

Work Competence:

NSPE Canon: "Perform services only in areas of their competence; Avoid deceptive acts".

IEEE: Perform services adhering to own level of competence while being honest about that level of competence.

Comparison: They both say the same thing.

Financial Responsibility:

NSPE Canon: "Act for each employer or client as faithful agents or trustees".

IEEE: Do not participate in immoral financial activities. Provide services that do not detriment your employer.

Comparison: They are similar but IEEE focuses on making sure your actions don't hinder the employer.

Communication Honesty:

NSPE Canon: "Issue public statements only in an objective and truthful manner; Avoid deceptive acts".

IEEE: Be honest and make sure to communicate with affected parties should problems arise.

Comparison: IEEE focuses more on daily conduct while NSPE focuses on organization conduct.

Health, Safety, Well-being:

NSPE Canon: "Hold paramount the safety, health, and welfare of the public".

IEEE: Health, safety and welfare of the public should be the number one focus.

Comparison: They both say the same thing.

Property Ownership:

NSPE Canon: "Act for each employer or client as faithful agents or trustees."

IEEE: Provide a fair agreement in terms of ownership.

Comparison: NSPE talks about ownership of property under an organization while IEEE talks about ownership of property in general.

Sustainability:

NSPE Canon:

IEEE: Make sure not to harm the environment.

Comparison: Cannot compare.

Social Responsibility:

NSPE Canon: "Conduct themselves honorably, responsibly, ethically, and lawfully so as to enhance the honor, reputation, and usefulness of the profession."

IEEE: Identify and report issues of social concern.

Comparison: This differs from the NSPE Canon in the fact that it talks less about perception of the profession and more about what the public expects

## 7.2 PROJECT SPECIFIC PROFESSIONAL RESPONSIBILITY AREAS

Work Competence is an important part of every team, ours included. We currently function at a medium level in relation to work competence. Every team member brings unique expertise to the team that allows us to perform at a high level. However, since we are still young in our careers there is much that we do not know, which we realize and continue to strive to get better every day.

Financial Responsibility is a low level of importance to us. We do not have a budget for our project, but we still recognize that we need to be financially responsible with the money we are given. If we are needing to buy a part for our system, we spend time making sure we are making a reasonable and correct purchase so as to not waste our client's money.

Communication Honesty, for our group, is a medium level of importance to us. As a team, we have multiple ways to communicate with each other. If one team member is falling behind with their work, we communicate with them to figure out the problem and how to go about a solution to the problem.

Health, Safety, and Well-Being are very important for us in a professional context. As a team, we are currently at a low level but plan on focusing more on this area next semester when we start our security implementation.

Property Ownership is a low level of importance to our team. We were given an AWS account by our client to us, and we respect that. We will not use it for any personal project or manage it irresponsibly.

Sustainability is an important aspect for us. We are currently at a high level of performance as a team in this area. At the start of our project, we focused a large portion of our time on making sure our system would be sustainable for farmers so that they can use it from planting season to harvest.

Social Responsibility does not apply to our group. At the moment, we are not developing a system that is important socially speaking.

### 7.3 MOST APPLICABLE PROFESSIONAL RESPONSIBILITY AREA

The most applicable professional responsibility area for this project would be sustainability. Our products will be deployed in fields that require constant maintenance and specific balances of nutrient levels. If our product does not respect this environment and take proactive steps to prevent damage, it will fail as none of our users will want to trade our product for the health of their crops.

We plan on ensuring our systems hold sustainability at the highest standard by regularly testing the durability and composition of our products before deploying them in fields. We only buy and test parts that are rated for outdoor weather conditions and have a long life span. We also research the materials used for these products to ensure there is a minimal amount of possibly toxic materials

## 8 Future Work

### 8.1 IOT DEVICES FUTURE WORK

There are a few points in the perception and networking elements of the design that can be continued. One example is to redesign the basestation to be more cost and energy efficient. The RaspberryPi 4B was excellent for prototyping because its extra computing power allowed for more analysis and testing. However, for deployment in agriculture, the computing power of the RaspberryPi is not necessary. Redesigning the basestation to strictly support packet forwarding and its configuration would be worthwhile.

Alongside creating a new device, evaluating alternative network servers to The Things Network would be a good next step. In this project, the LoRaWAN concentrator chip we selected had limited network servers it could interact with (i.e., The Things Network and Chirpstack). Ideally, a new basestation could be configured to interact with any LoRaWAN network server. This would allow for cost efficiency analysis between LoRaWAN network servers, cloud services, and even investing in a server for the project.

Finally, fabricating renewably powered end devices would be better for environmental concerns. The introduction of renewable or even batteryless sensors adds a level of complexity that was deemed out of scope during this project. However, with a fully functioning infrastructure in place, it would be a reasonable next step to see how the energy supply to sensors could be moved away from batteries.

## 8.2 AWS FUTURE WORK

There are a few aspects of AWS that may be continued. First, as the project grows migrating to Infrastructure as Code would enable developers to manage versioning more easily. This may include using Terraform and Drone in conjunction to manage a CI/CD pipeline. Second, implementing anomaly detection is a big task that could be integrated into our existing AWS infrastructure. By using OpenSearch or the ELK stack, machine learning can be used to add to the data collection monitoring. Finally, as discussed above in section 5.8.2, a division of developer permissions to align with the subsections of the project will allow for security best practices.

## 8.3 MOBILE APP FUTURE WORK

Adding more advanced features to support power users of our platform is the most important future work. There are a plethora of different additional configuration options that could be added for users to better control their sensors. We've discussed including options for optimal water/temperature levels on a per sensor bases so users can more easily understand if their plants are receiving too much or not enough resources. Another feature discussed was the ability to overlay the sensor positions on Google Maps so that users could see at a glance where each sensor was located on their property.

# 9 Closing Material

## 9.1 DISCUSSION

Thus far, our project has yielded a usable mobile application and simple single-user AWS infrastructure. Users are able to download our application and receive static data from AWS. The only portion of our project that still requires significant development is the IoT sensors & basestations. Moving forward into the next semester, this will be our primary focus, as we hope to have as much time as possible set aside for security and testing. By the end of this year or January of next year, we will have sensors reading and uploading data to our cloud infrastructure. There are a number of functional requirements that still need to be addressed, mainly surrounding the access of live and up-to-date data. However, we will also be dedicating a significant amount of time to developing our intrusion detection systems going forward. This will require additional research into possible machine-learning techniques and other data processing techniques.

## 9.3 CONCLUSION

At the end of our two semester pursuit, we have accomplished almost all of the major goals set out for us at the beginning of the school year. We have a scalable solution that allows end users to buy and deploy as many sensors and basestations as needed for their project. These sensors produce raw data that is securely collected and stored in our AWS infrastructure. Finally, our frontend mobile

application contextualizes that raw data and produces information in a user-friendly, digestible manner. The only major milestones that have yet to be accomplished for this project are the use of cloud-based machine learning models for anomaly detection, and the integration of some of the more advanced user controls for sensor deployment.

Our IoT team has put together a seamless pipeline from purchase to deployment of end user basestation and sensor hardware, allowing for easy scaling no matter the size of the project. Basestations are made of an easy to use Raspberry Pi 4 and a common LoRAWAN Pi Hat. These parts keep costs low while providing the same reliability as larger commercial solutions. There is no future work planned for the IoT team after this project concludes.

The AWS infrastructure that has been spun up for this project is secure in it's handling of customer data as well as efficient in it's use of AWS resources. We followed the principles of least privilege when designing these systems in order to minimize damages in the case that a malicious user gains control of an internal AWS service account. At peak load, our testing deployments never reached more than \$10 a month in server costs. This is owed to the use of low cost serverless computing and minimal redundant resource usage. There are several pieces of future work to consider for the AWS team after this project concludes. As mentioned above, the transition to a proper CI/CD pipeline using Terraform is a must if this project is going to continue to be successful. There is also the optional addition of machine learning for the purposes of anomaly detection, this would increase user awareness and control over their crops.

Our mobile application has been the cornerstone of our success when demonstrating the power of this technology. It allows us to demonstrate the responsiveness of our project through the real time graphical representation of data from user sensors. Users have the ability to filter data collection down to a specific day and sensor. This level of filtering gives users unparalleled granularity when managing their crops. Future work for the app involves reimplementing a more complete state management system to enable complex features as well as adding additional configuration choices for sensors during deployment.

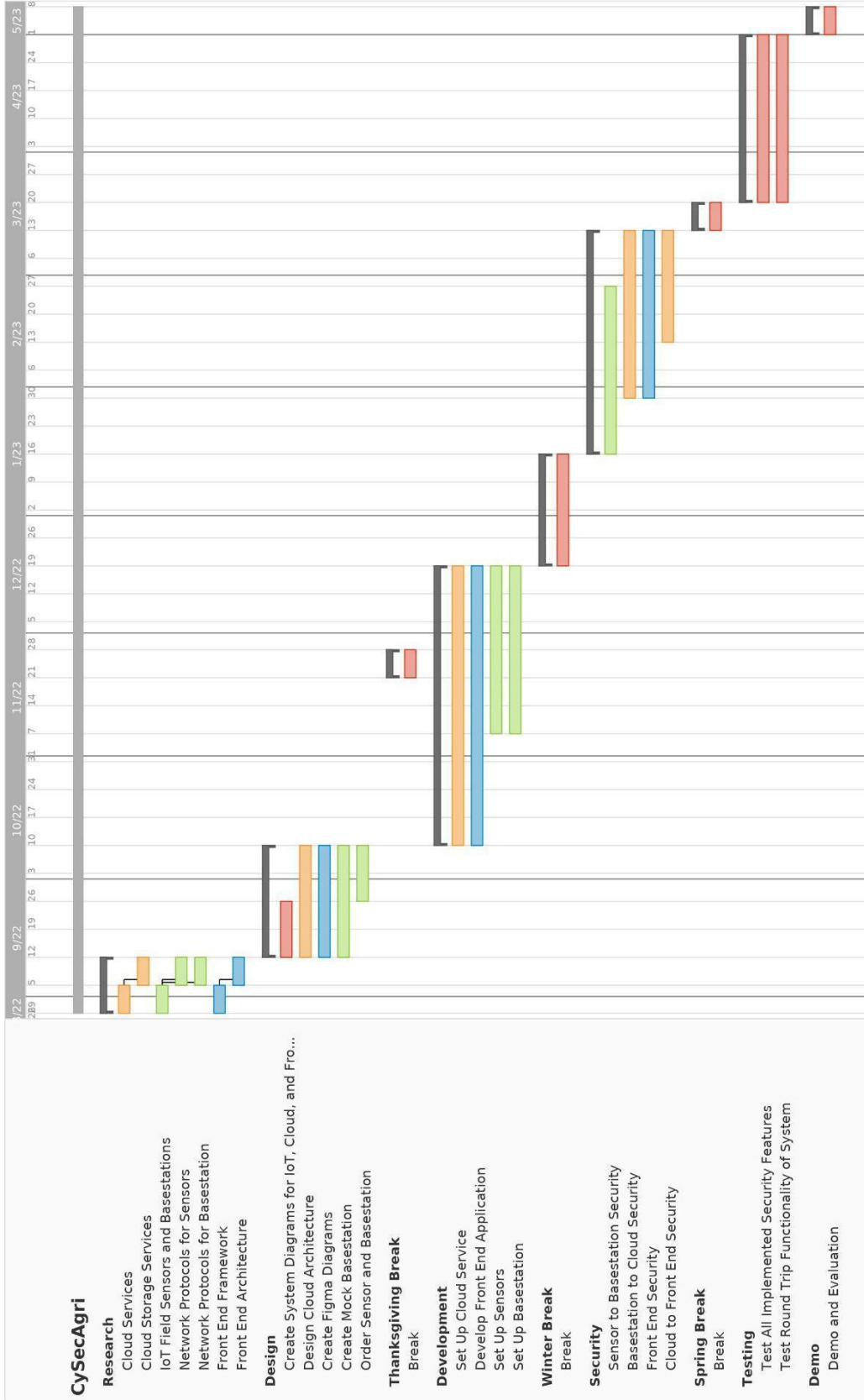
This project was a lofty undertaking; however, it's allowed our team to push beyond our limits and discover new and interesting possibilities in the realm of IoT and smart agriculture. We've produced a polished final product that peers and professionals alike agree will benefit consumer agriculture as a whole. We've produced detailed and insightful documentation for each step of this projects development and deployment. Sdmay23-17 hopes to see this project continue to flourish under the care of another senior design or research group, we know this has the potential to make a real impact on real people

## References

- [1] N. Selimović, "ABP vs OTAA," [www.thethingsindustries.com](http://www.thethingsindustries.com), Nov. 17, 2017.  
<https://www.thethingsindustries.com/docs/devices/abp-vs-otaa/>
- [2] A. Mishra, "Cloud-based multi-sensor remote data acquisition system for precision agriculture (CSR-DAQ)," Thesis, Iowa State University, 2020.
- [3] N. Selimović, "Best Practices," [www.thethingsindustries.com](http://www.thethingsindustries.com), 2022.  
<https://www.thethingsindustries.com/docs/devices/best-practices/>
- [4] K. Lee and J. Lu, "The New Generation LoRaWAN Sensors of SenseCAP S201X Sensors User Guide.," Nov. 2022. [Online]. Available:  
<https://files.seeedstudio.com/products/SenseCAP/S210X/SenseCAP%20S210X%20LoRaWAN%20Sensor%20User%20Guide.pdf>
- [5] M. El-Hajj, A. Fadlallah, M. Chamoun, and A. Serhrouchni, "A Survey of Internet of Things (IoT) Authentication Schemes," *Sensors*, vol. 19, no. 5, p. 1141, 2019, doi: 10.3390/s19051141.
- [6] J. McCormack et al., "Contextualizing Professionalism in Capstone Projects Using the IDEALS Professional Responsibility Assessment," *International Journal of Engineering Education*, vol. 28, no. 2, pp. 416–424, 2012.

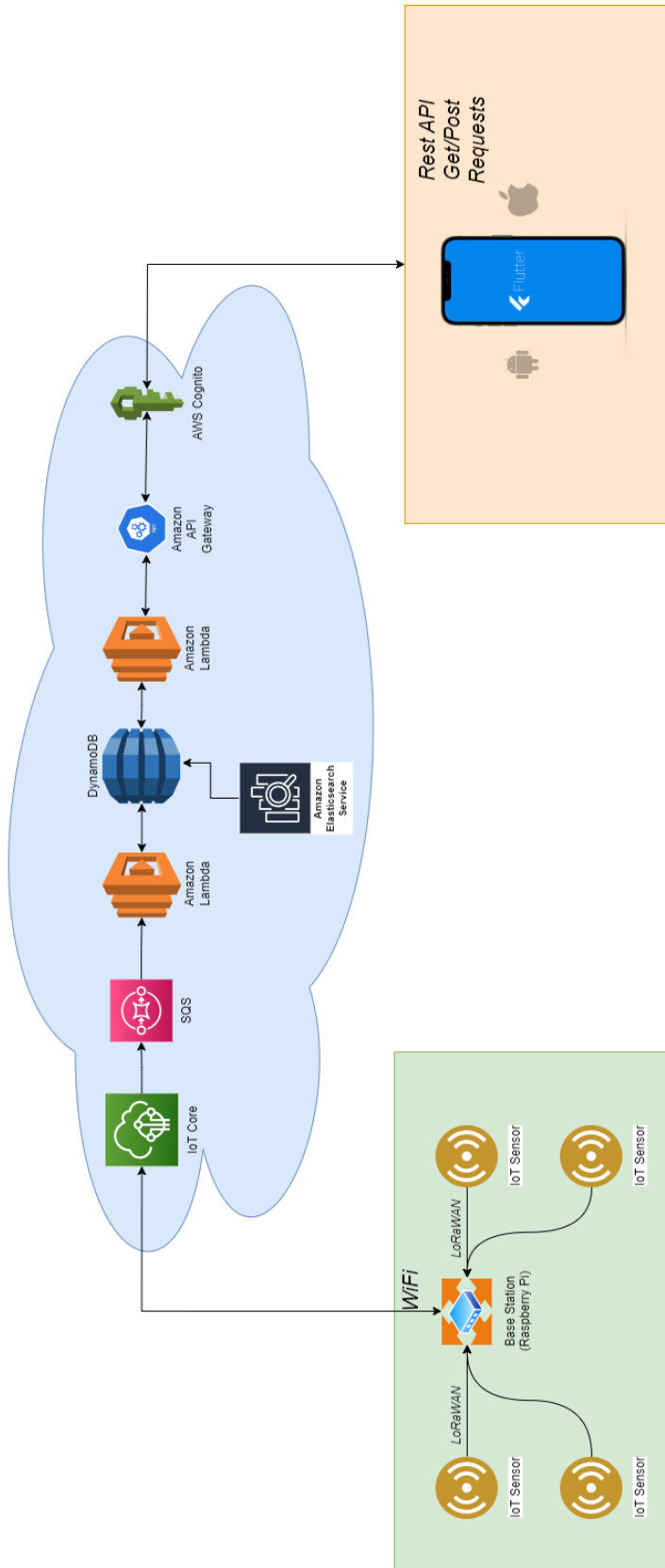
# Appendix A

## Gantt Chart



# Appendix B

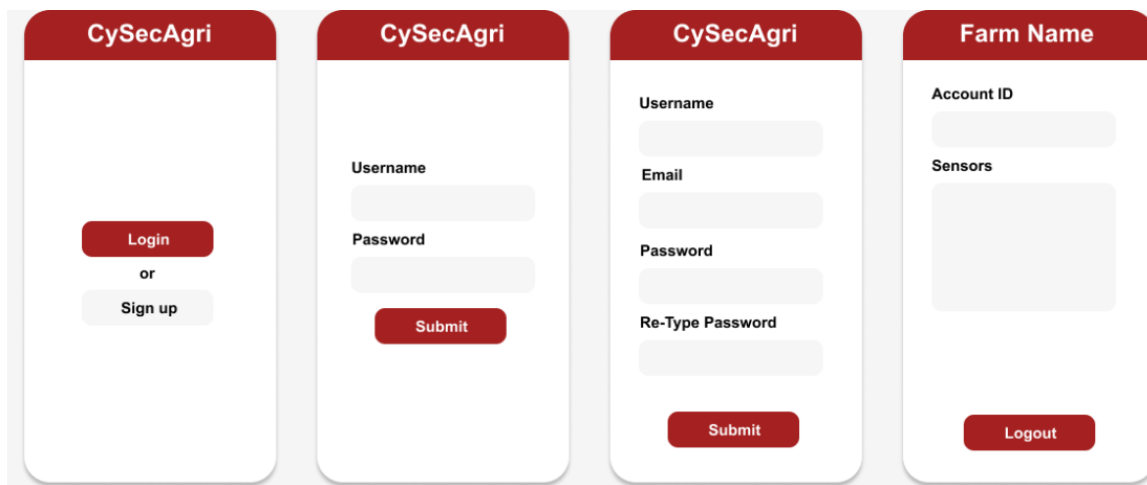
## System Diagram





# Appendix C

## Figma Page Designs



## Appendix D

Risk Severity Table

Probability	Impact				
	Very Low	Low	Medium	High	Very High
Highly Probable	5 - Moderate	10 - Major	15 - Major	20 - Severe	25 - Severe
Probable	4 - Moderate	8 - Moderate	12 - Major	16 - Major	20 - Severe
Possible	3 - Minor	6 - Moderate	9 - Moderate	12 - Major	15 - Major
Unlikely	2 - Minor	3 - Moderate	6 - Moderate	8 - Moderate	10 - Major
Rare	1 - Minor	2 - Minor	3 - Minor	4 - Moderate	5 - Moderate

## Appendix E

### Operation Manuals

# CySecAgri: Basestation Set-up Manual

## Materials & Prerequisites

- A registered *The Things Stack* (TTS) account (<https://console.cloud.thethings.network>)
- A Raspberry Pi Model 4B Starter Kit
  - [https://www.amazon.com/GeeekPi-Raspberry-4GB-Starter-Kit/dp/B0B3M2HKN6?ref\\_=ast\\_slp\\_dp](https://www.amazon.com/GeeekPi-Raspberry-4GB-Starter-Kit/dp/B0B3M2HKN6?ref_=ast_slp_dp)
- A RAK 5146 PiHAT Kit for LoRaWAN
  - <https://store.rakwireless.com/products/rak5146-kit?variant=41577988194502>
- A monitor, TV, or other screen with an HDMI port

Note: This manual shows frequency plans for LoRaWAN implementations in North America. If you do not live in a North American country please check the link below to see what frequency plan you should substitute during configuration.

<https://www.thethingsnetwork.org/docs/lorawan/frequencies-by-country/>

## Flashing Raspbian to the Pi

Install the operating system for the basestation onto the Raspberry Pi.

1. Follow the instructions on the Raspberry Pi website to download and run *imager* on your chosen platform (<https://www.raspberrypi.com/software/>)
2. Put Micro-SD card into USB adapter (provided in the Starter Kit)
3. Insert USB into the computer where *imager* was installed
4. Open *imager* application
  - a. Select Default Raspbian OS (32 bit)
  - b. Select your USB (mass storage device)
  - c. Write



5. Wait for this process to finish
6. Remove the USB adapter from your computer and Micro SD card from the adapter. Insert the Micro SD card into the bottom side of the Raspberry Pi

## Install the RAK 5146 Pi Hat

Now, install the concentrator chip into the Raspberry Pi to enable LoRaWAN signal aggregation.

1. Insert 4 brass standoffs through the bottom 4 screw holes of the 4 corners of the Raspberry Pi
2. Screw 4 more brass standoffs from on top of the Pi into the 4 from the bottom
3. Press the RAK 5146 Base Board through the 4 brass standoffs and ensure all pins are inserted into the GPIO pins of the Raspberry Pi
4. Insert the RAK concentrator chip into the slot on the top of the RAK 5146 base board
  - a. NOTE: Insert at a 45 degree angle until you hear a click. The concentrator module should stay at that 45 degree angle if you take your hands off
5. Press down on the raised part of the concentrator chip to align the screw holes with the base board
6. Use the smallest included screws to secure the concentrator module to the base board
7. Clip the GPS antenna into the GPS port on the base board
8. Clip the LoRa antenna into the LoRa port on the base board

## Powering on the Pi

Next, turn the basestation on for the first time.

1. Insert power cable (included in the Starter Kit) into USB C port on Pi
  - a. Ensure the switch position has the raised line side NOT depressed
  - b. Other side into wall outlet
2. Insert micro-HDMI cable into micro-HDMI port on Pi and the other end into full side HDMI on a monitor
3. Insert keyboard/mouse into USB on Pi
4. Flip power switch on Raspberry Pi
5. Monitor should display booting screens and status
6. Complete onscreen setup
  - a. Write down username and password created during setup
  - b. Connect the Pi to a reliable WiFi
7. Once in, open a command prompt and run `sudo systemctl enable ssh`. (This will enable the ability to remotely connect the Pi in the future. Which removed the need for a dedicated monitor, mouse, and keyboard for future configuration.)

## Installing Basestation Software

With the preparation work complete, enable LoRaWAN gateway functionality on the basestation. The following link may be helpful for reference:

[https://github.com/RAKWireless/rak\\_common\\_for\\_gateway/tree/1dcdc30e1c7ec21f9d8071eb45e59171e56b3718](https://github.com/RAKWireless/rak_common_for_gateway/tree/1dcdc30e1c7ec21f9d8071eb45e59171e56b3718)

1. On the Raspberry Pi run the command:
 

```
git clone
https://github.com/RAKWireless/rak_common_for_gateway.git
```
2. In the same command prompt, run:
 

```
cd rak_common_for_gateway
```
3. Then run:
 

```
sudo ./install.sh
```
4. When prompted select the option to install the gateway model: RAK5146(SPI)
5. After installation, reboot the Pi (`sudo shutdown -r`). It is now officially a gateway.

## Registering the Gateway with TTS

Finally, the basestation needs to be registered with the network server to manage uplinks and downlinks in the IoT system.

1. Open a command prompt on the basestation and run:
 

```
sudo gateway-config
```
2. From the config menu select: Setup RAK Gateway Channel Plan
3. Select: Server is TTN
4. Finally, select US\_902\_928. Hit OK.
 

(\*) This might vary depending on the region a basestation is deployed in. Consult the Materials section for details.
5. At the top of the main configuration window record the **Gateway ID** for later use. An example might look like: E45F01FAAE8B23D8
6. On a separate computer login to the TTS and navigate to the dashboard
7. At the top of the dashboard, click on Gateways
8. Click “Register Gateway” in the top right corner of the Gateways page
9. Enter the Gateway ID into the Gateway EUID field and click confirm
10. On the next page set a unique Gateway Name and ensure the FSB 2 frequency plan is selected. Finally uncheck share status within network and uncheck share location within network.
11. Click Register Gateway
12. Go back to the Gateways tab in the dashboard and select the newly created Gateway
13. Then go to the API Keys tab on the left side of the Gateway page
14. From here give the API key a name and the following individual permission:

- a. Link as Gateway to A Gateway Server (Write uplinks read downlinks)
15. In a few minutes, the basestation should appear as a connected gateway on your TTS dashboard

# CySecAgri: Sensor Set-up Manual

## Materials & Prerequisites

- A registered *The Things Stack* (TTS) account (<https://console.cloud.thethings.network>)
- A gateway registered to your TTS account (see *CySecAgri: Gateway Set-up Manual*)
- A SenseCAP S2104 sensor

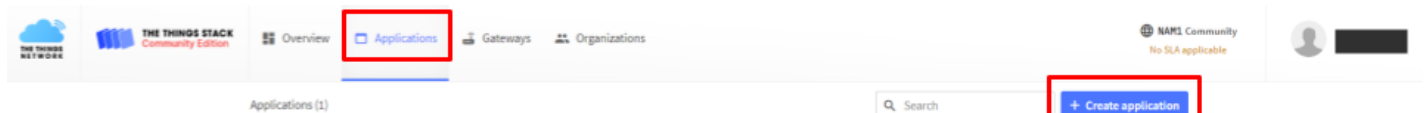
Note: This manual shows frequency plans for implementations in North America. If you do not live in a North American country please check the link below to see what frequency plan you should substitute during configuration.

<https://www.thethingsnetwork.org/docs/lorawan/frequencies-by-country/>

## Creating a TTS Application

Before connecting the sensors, an application needs to be created to receive data from them and approve sensor join-requests.

1. In the The Things Stack, navigate to the Applications tab and select “Create Application.” Select a Application ID, Application Name, and provide an optional description. Then select “Create Application” again to finish creating your

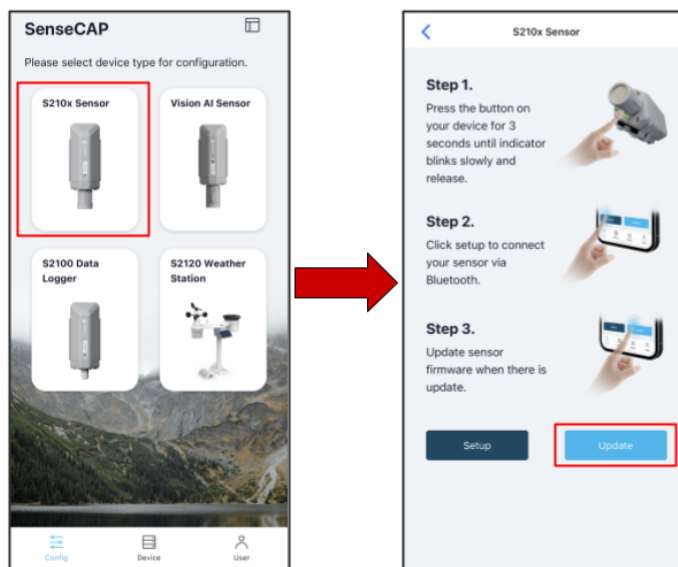


- application.
2. On the left, select “API Keys.” Then, in the upper right, select “Add API Key.” Give a name to your new key, leave the expiry date blank, and select “All current and future rights.” Then “Create API Key.”
3. Your TTS application is now ready to receive sensor data.

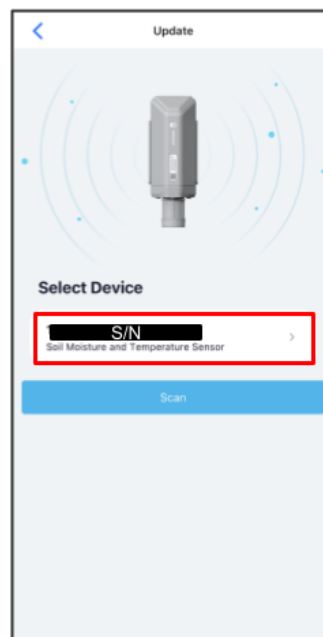
## Connecting S2104 Sensors to TTS

Next, we will turn on and configure the sensors as well as configure our application to accept data from them.

1. To begin setup, remove the sensor from the box and download the SenseCAP Mate mobile application. Enable bluetooth on your mobile device. Open the app and select "S210X Sensor." Then select "Update."



2. Hold the button/indicator LED (shown below) on the sensor for 3 seconds. You should see the LED flashing slowly. This means it is now ready to pair with the mobile application. Select the device that appears on the mobile application. The ID number on the screen should match the S/N number on the sensor's sticker.



3. Select the Settings tab on the top menu bar (right). You will then see several options. Choose the following options:
  - a. Platform = The Things Network
  - b. Frequency Plan\* = US915
  - c. Uplink Interval = 5
  - d. Activation Type = OTAA
  - e. Packet Policy = 1N

Note: For the DeviceEUI, APP EUI, and APP Key you can use the default settings provided or create your own. However, these values **must** be unique for every sensor in your implementation. Regardless, record these values in a **secure location** for later use.

(\*) Adjust this option based upon your geographical region.

4. Set the mobile application and sensor aside for the moment. Now, navigate to the “Overview” menu of the TTS application you created previously. On the overview screen, click the button labeled “Register end device”.
5. Choose “Select end device in the LoRaWAN Device Repository” and select the options given below.

## Register end device

Does your end device have a LoRaWAN® Device Identification QR Code? Scan it to speed up onboarding.

📷 Scan end device QR code
📄 [Device registration help](#)

---

### End device type

Input method ⓘ

Select the end device in the LoRaWAN Device Repository

Enter end device specifics manually

End device brand ⓘ *	Model ⓘ *	Hardware Ver. ⓘ *	Firmware Ver. ⓘ *	Profile (Region) *
SenseCAP   ▾	SenseCAP S2104 - L...   ▾	1.0   ▾	1.0   ▾	US_902_928   ▾

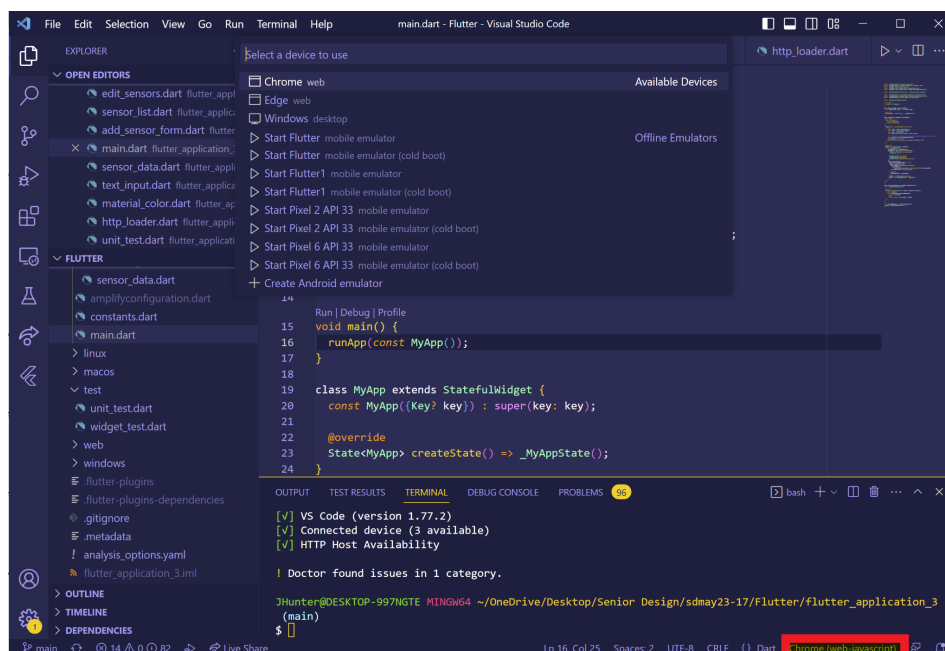
6. Select the “FSB 2” frequency plan. Then, using the credentials obtained from pairing the sensor, fill in the Join EUI, Dev EUI, and AppKey information you recorded. [NOTE: In this instance, JoinEUI and AppEUI mean the same thing.] The other settings can be left to their defaults. Once completed, select “Register End Device.”
7. Return to the mobile application, select “Send” on the bottom right. The LED indicator will begin a “breathing” pattern and when the sensor connects to the gateway it will flash several times. Your sensor is now connected to your network!



# CySecAgri: Application Development Environment Set-Up Manual

## Setup Windows Environment

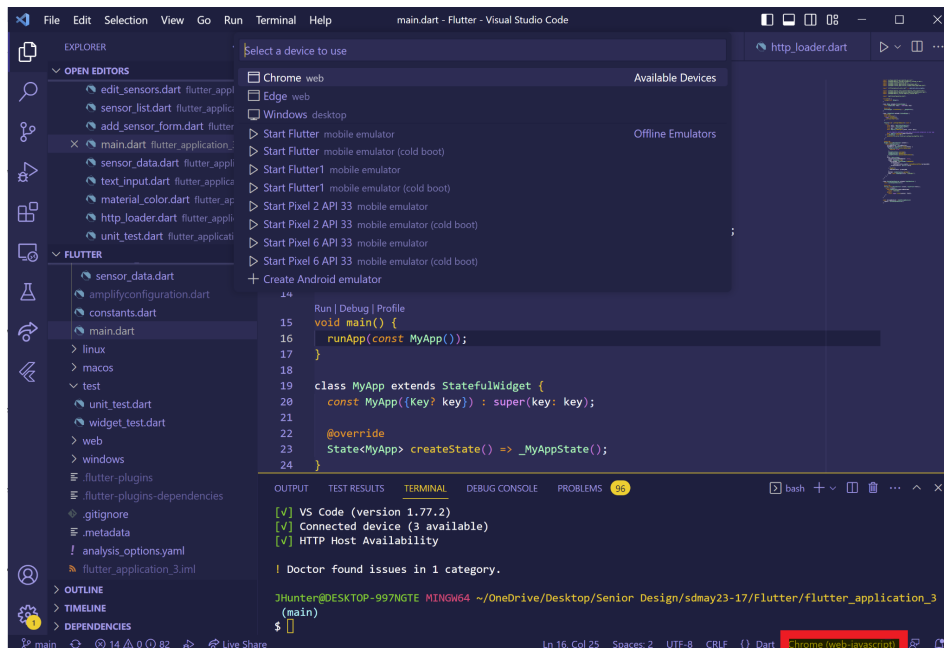
1. Download Flutter: <https://docs.flutter.dev/get-started/install/windows>
  - a. Follow the install for Windows as well as the [set up editor](#) section on the next page to set up VScode for Flutter.
    - i. This should set up an Android emulator to use for VScode through Android Studio.
2. Clone repository from GitLab: <https://git.ece.iastate.edu/sd/sdmay23-17>
  - a. Need to be on Iowa State Wifi or VPN to access GitLab.
3. Open the project through VScode.
  - a. `cd` into the `flutter_application_3` directory.
    - i. `cd sdmay23-17/Flutter/flutter_application_3`
4. Next run `flutter pub get` to get all of the applications dependencies.
5. Start the emulator through VScode.
  - a. Click the highlighted button in the picture below. Then select the emulator you created from Android Studio.



6. In your terminal type the command `flutter run` to run the application.
7. In your terminal type the command `flutter test` to run the unit and widget tests for the application.

## Setup MacOS Environment

1. Download Flutter <https://docs.flutter.dev/get-started/install/macos>
  - a. Follow the install for Windows as well as the [set up editor](#) section on the next page to set up VScode for Flutter.
2. Clone repository from GitLab: <https://git.ece.iastate.edu/sd/sdmay23-17>
  - a. Need to be on Iowa State Wifi or VPN to access GitLab.
3. Open the project through VScode.
  - a. cd into the flutter\_application\_3 directory.
    - i. cd sdmay23-17/Flutter/flutter\_application\_3
4. Next run `flutter pub get` to get all of the applications dependencies.
5. Start an iPhone or iPad through Simulator.
6. Start the emulator through VScode.
  - a. Click the highlighted button in the picture below. Then select the emulator you are currently running from Simulator.



7. In your terminal type the command `flutter run` to run the application.
8. In your terminal type the command `flutter test` to run the unit and widget tests for the application.

# CySecAgri: Mobile App Proxy Set-Up Manual

1. Install Android Studio
2. Install BurpSuite
3. Export BurpSuite Proxy CA
4. Copy CA (drag and drop) to Android Emulator
5. Import CA inside android settings (security/trust/add ca)
  - a. This will put it in the `user` certificate space in android
6. Add following to the *android/app/src/main/res/xml/network\_security\_config.xml* file inside of the flutter project

```
<network-security-config>  
  <base-config cleartextTrafficPermitted="true">  
    <trust-anchors>  
      <certificates src="system" />  
      <certificates src="user" />  
    </trust-anchors>  
  </base-config>  
  <domain-config cleartextTrafficPermitted="true">  
    <domain includeSubdomains="true">127.0.0.1</domain>  
  </domain-config>  
</network-security-config>
```

7. Add *android:networkSecurityConfig="@xml/network\_security\_config"* after *<application* in the AndroidManifest.xml file
8. Login to the mobile app through the emulator and see the login request captured in BurpSuite