FarmAssist

Group #27

Group Members

- **MALHOTRA, Avi (55773896)** – Project Manager
- **GUPTA, Aarnav (55990960)** – Assistant Project Manager
- **BANBAH, Kush (557867405)** – Software Designer
- **JAIN, Utkarsh (55992915)** – Software Designer
- **KASLIWAL, Aryan Girish (55972222)** – Software Designer
- **RAJAGOPALAN, Pratul (55858290)** – Software Designer

Introduction

World Population of about 7.8 Billion

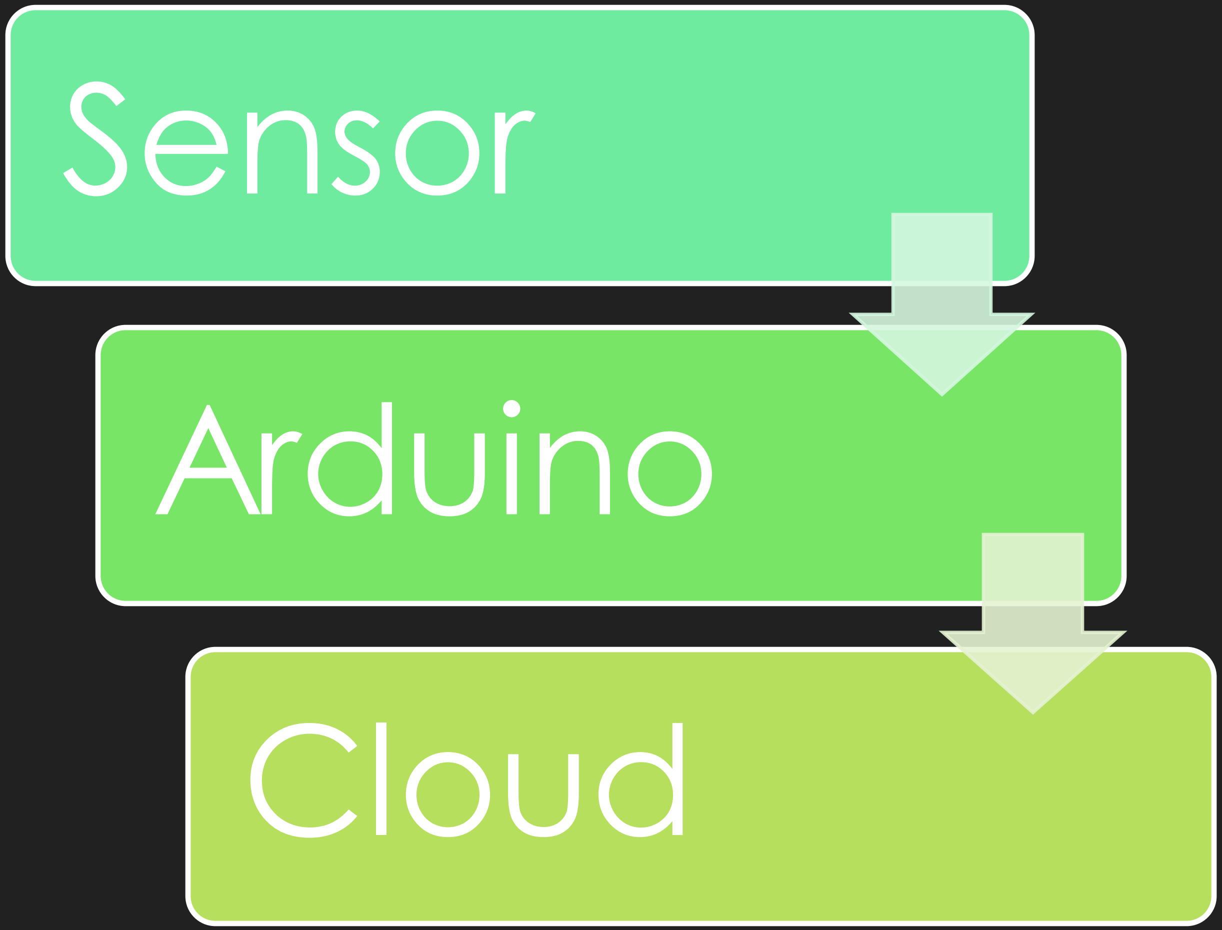
Smart Farming

LINK SENSORS
WITH MOBILE APP

TEMPERATURE

WATER

NUTRIENT LEVELS

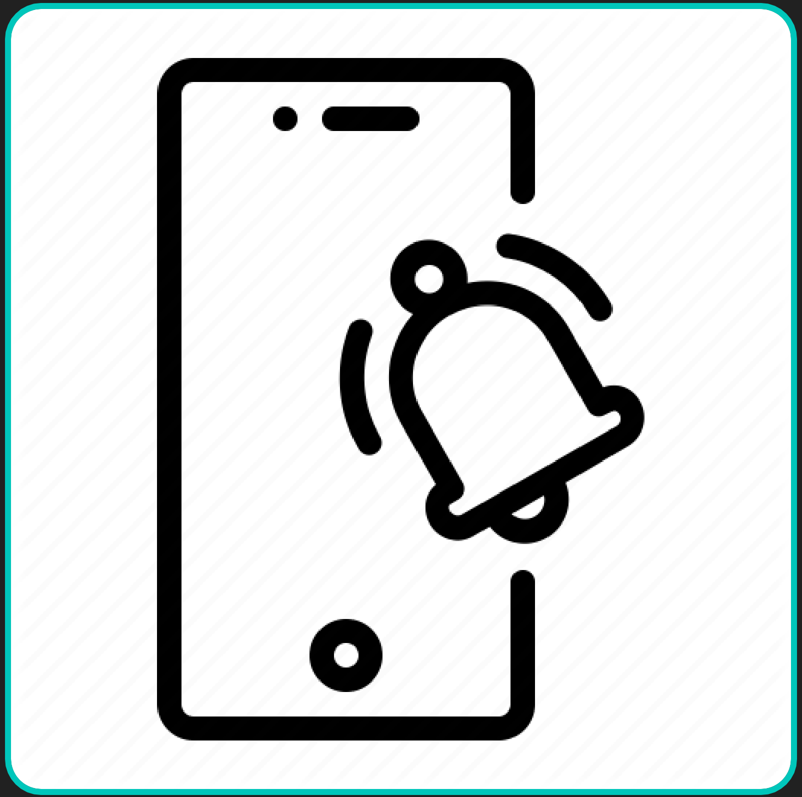


Sensor

Arduino

Cloud

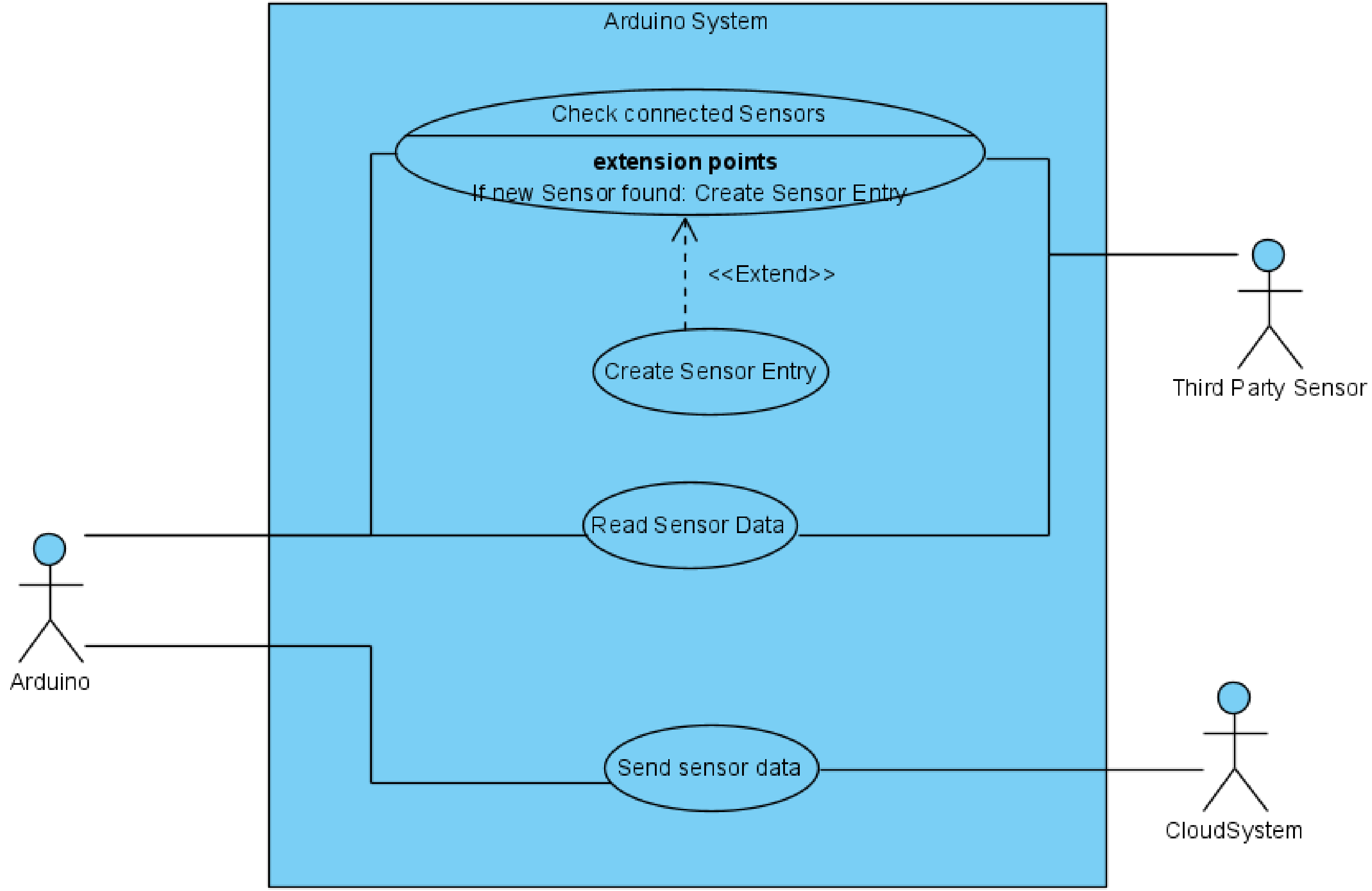
Smartphone Notification

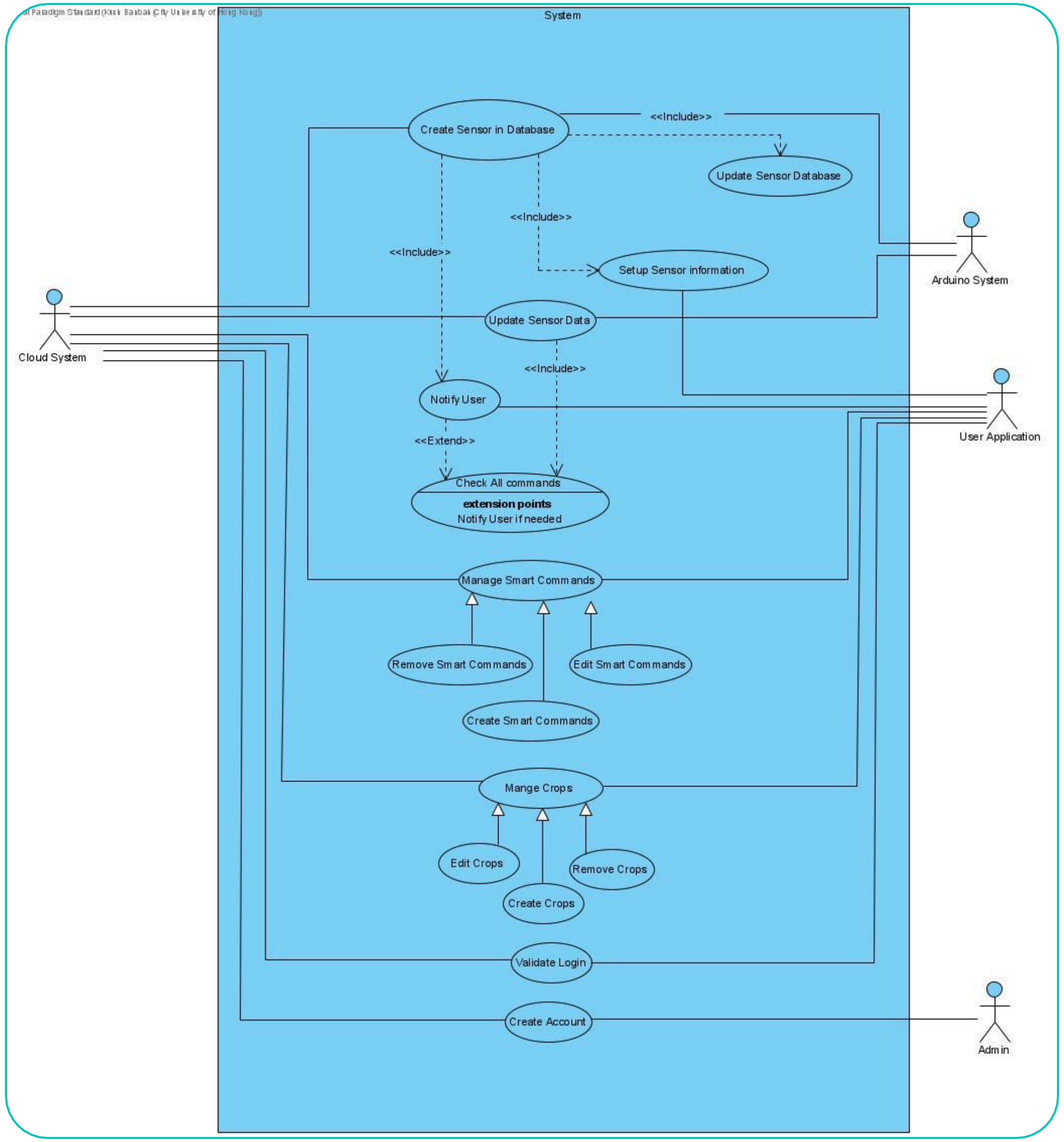


Feasibility

- INITIAL SETUP
- MAINTENANCE
- ADVERTISEMENT
- GOVERNMENT SUPPORT

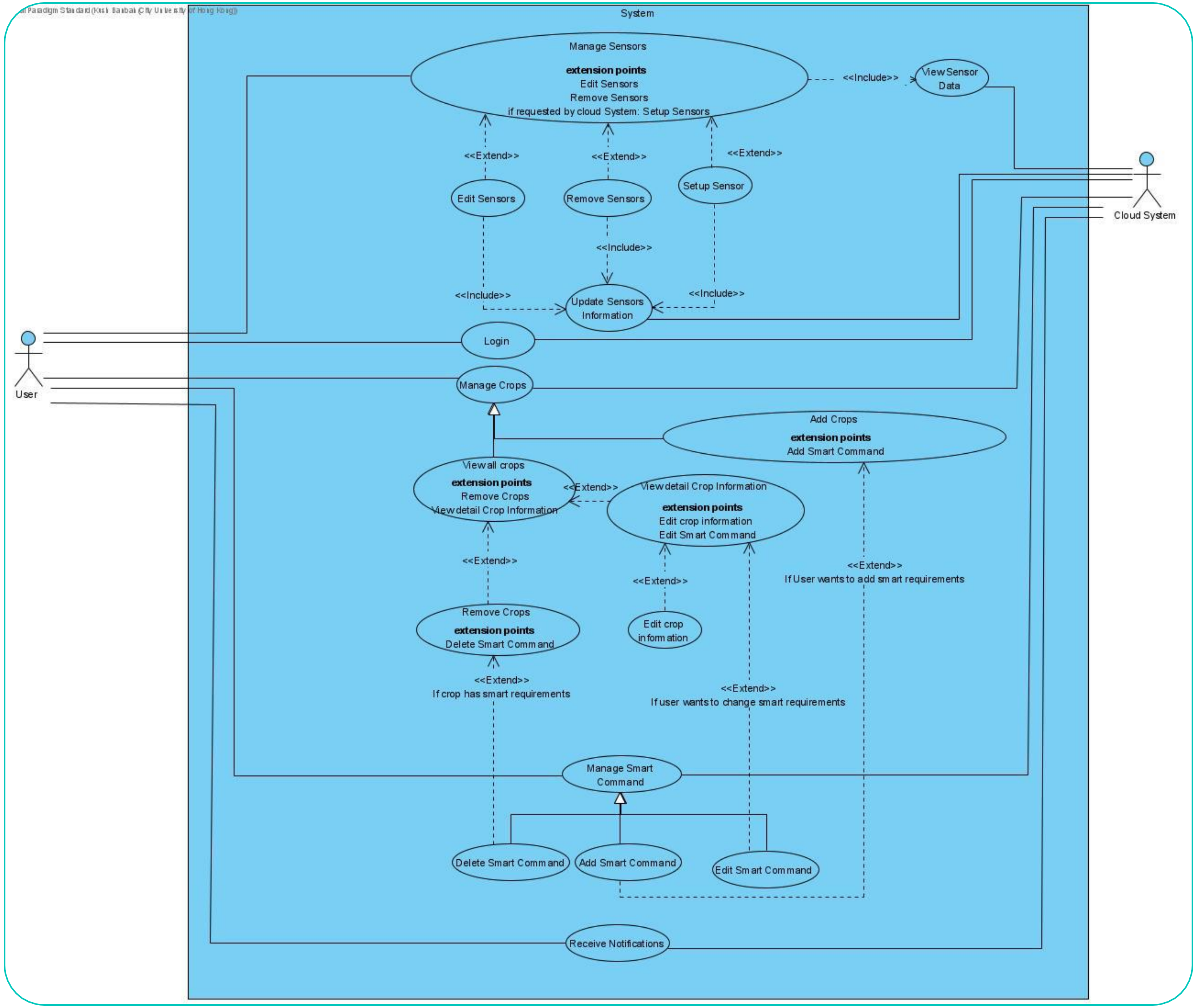
Use Case Modeling Arduino System





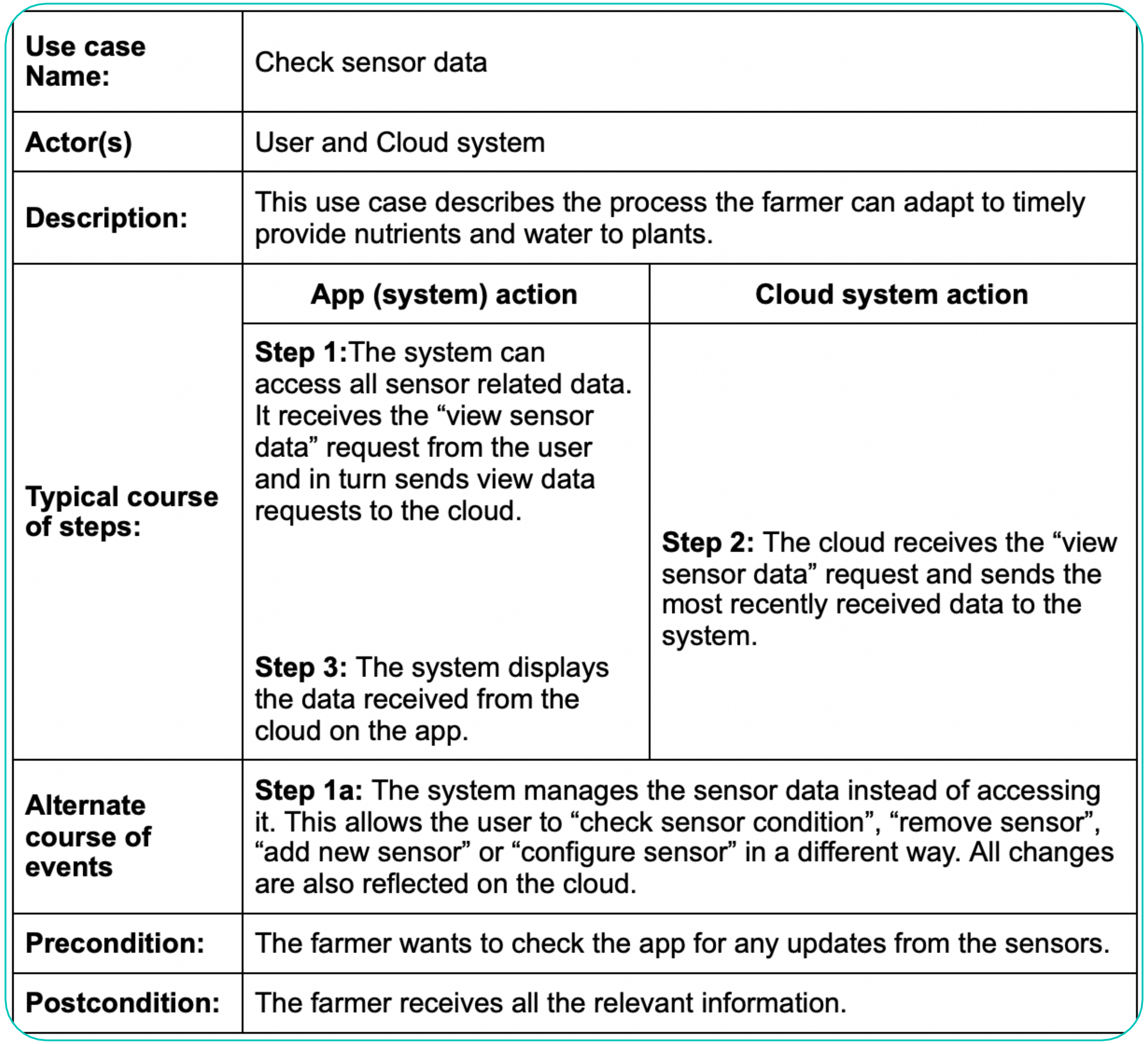
Use Case Modeling Cloud System

Use Case Modeling Cloud System



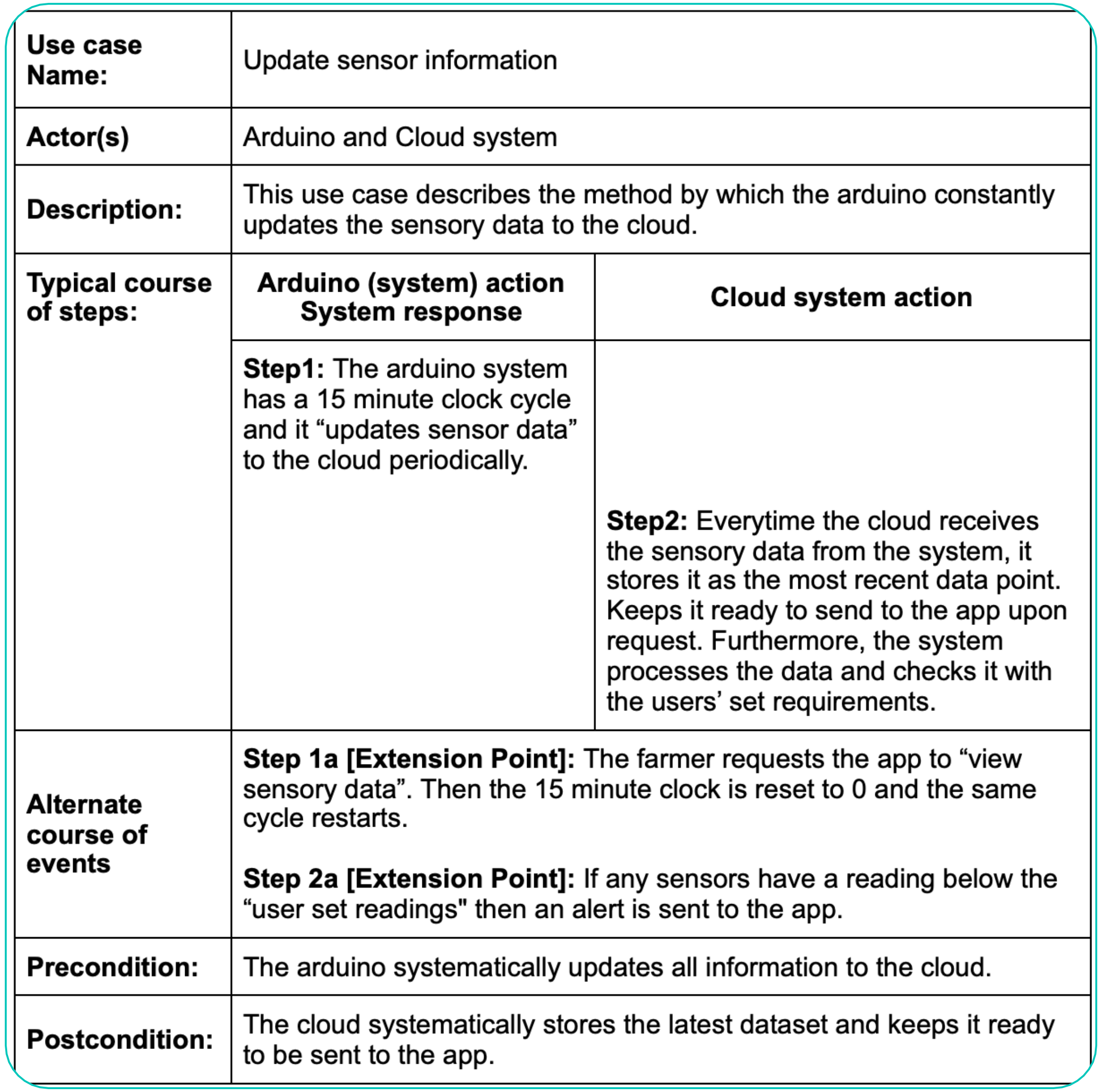
Use Case Requirement Tables

Check Sensor Data from App to Cloud



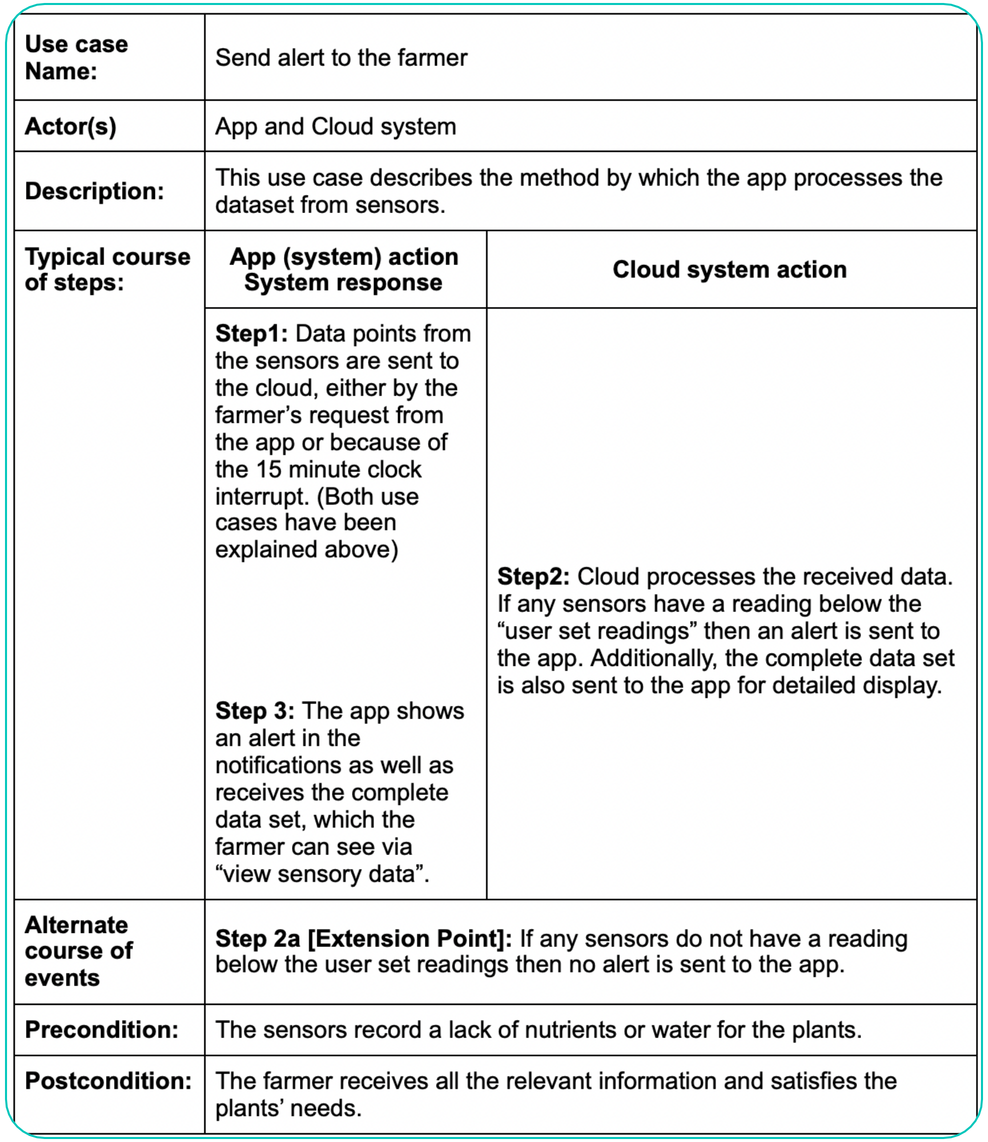
Use case Name:	Check sensor data	
Actor(s)	User and Cloud system	
Description:	This use case describes the process the farmer can adapt to timely provide nutrients and water to plants.	
Typical course of steps:	**App (system) action**	**Cloud system action**
	Step 1: The system can access all sensor related data. It receives the “view sensor data” request from the user and in turn sends view data requests to the cloud. **Step 3:** The system displays the data received from the cloud on the app.	**Step 2:** The cloud receives the “view sensor data” request and sends the most recently received data to the system.
Alternate course of events	**Step 1a:** The system manages the sensor data instead of accessing it. This allows the user to “check sensor condition”, “remove sensor”, “add new sensor” or “configure sensor” in a different way. All changes are also reflected on the cloud.	
Precondition:	The farmer wants to check the app for any updates from the sensors.	
Postcondition:	The farmer receives all the relevant information.	

Check Sensor Data from Arduino to Cloud



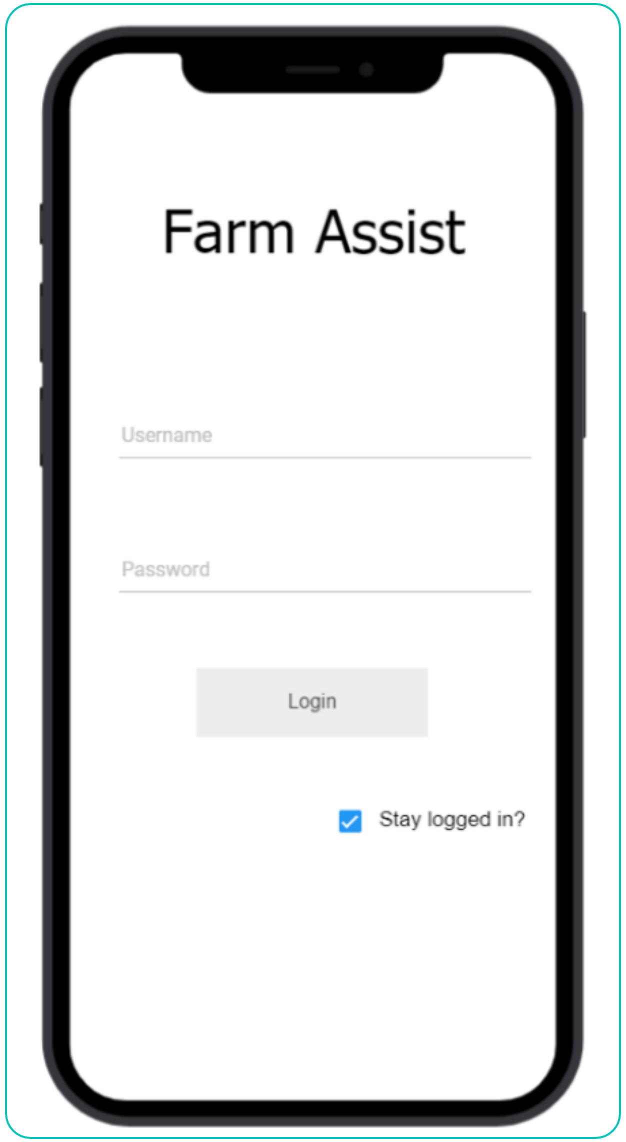
Use case Name:	Update sensor information	
Actor(s)	Arduino and Cloud system	
Description:	This use case describes the method by which the arduino constantly updates the sensory data to the cloud.	
Typical course of steps:	**Arduino (system) action System response**	**Cloud system action**
	Step1: The arduino system has a 15 minute clock cycle and it "updates sensor data" to the cloud periodically.	
		Step2: Everytime the cloud receives the sensory data from the system, it stores it as the most recent data point. Keeps it ready to send to the app upon request. Furthermore, the system processes the data and checks it with the users' set requirements.
Alternate course of events	**Step 1a [Extension Point]:** The farmer requests the app to "view sensory data". Then the 15 minute clock is reset to 0 and the same cycle restarts. **Step 2a [Extension Point]:** If any sensors have a reading below the "user set readings" then an alert is sent to the app.	
Precondition:	The arduino systematically updates all information to the cloud.	
Postcondition:	The cloud systematically stores the latest dataset and keeps it ready to be sent to the app.	

Cloud sends alert to the Farmer

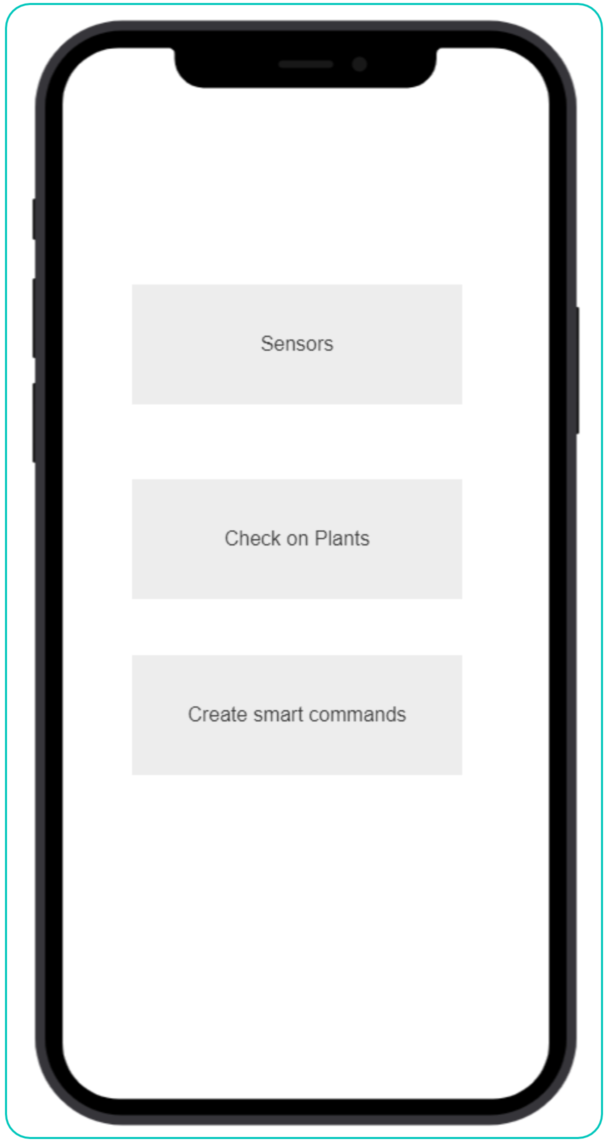


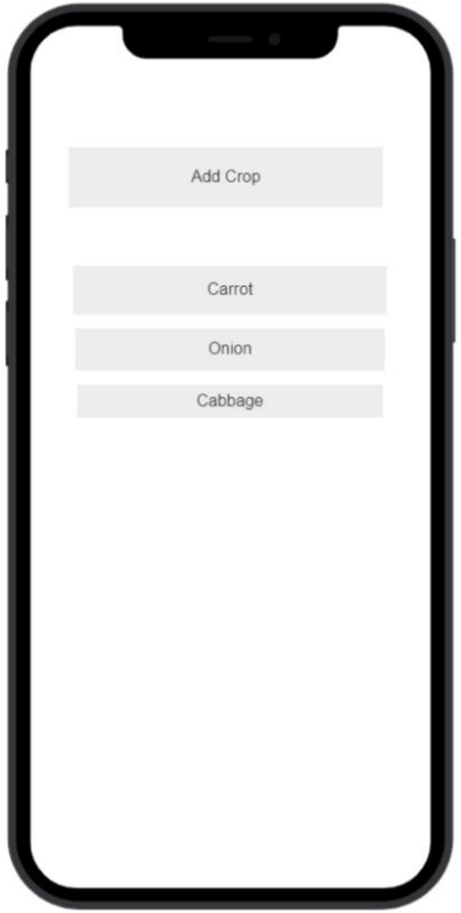
Use case Name:	Send alert to the farmer	
Actor(s)	App and Cloud system	
Description:	This use case describes the method by which the app processes the dataset from sensors.	
Typical course of steps:	**App (system) action System response**	**Cloud system action**
	Step1: Data points from the sensors are sent to the cloud, either by the farmer's request from the app or because of the 15 minute clock interrupt. (Both use cases have been explained above)	
		Step2: Cloud processes the received data. If any sensors have a reading below the "user set readings" then an alert is sent to the app. Additionally, the complete data set is also sent to the app for detailed display.
	Step 3: The app shows an alert in the notifications as well as receives the complete data set, which the farmer can see via "view sensory data".	
Alternate course of events	**Step 2a [Extension Point]:** If any sensors do not have a reading below the user set readings then no alert is sent to the app.	
Precondition:	The sensors record a lack of nutrients or water for the plants.	
Postcondition:	The farmer receives all the relevant information and satisfies the plants' needs.	

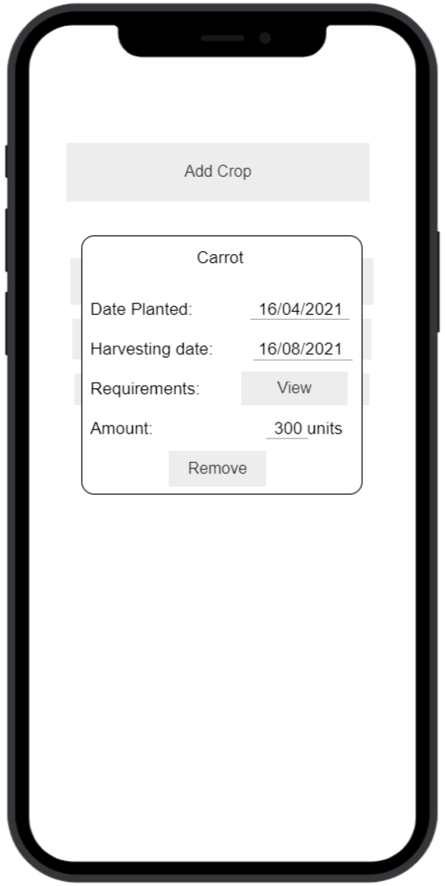
Prototype Demo- Login Screen



Prototype Demo- Main Menu







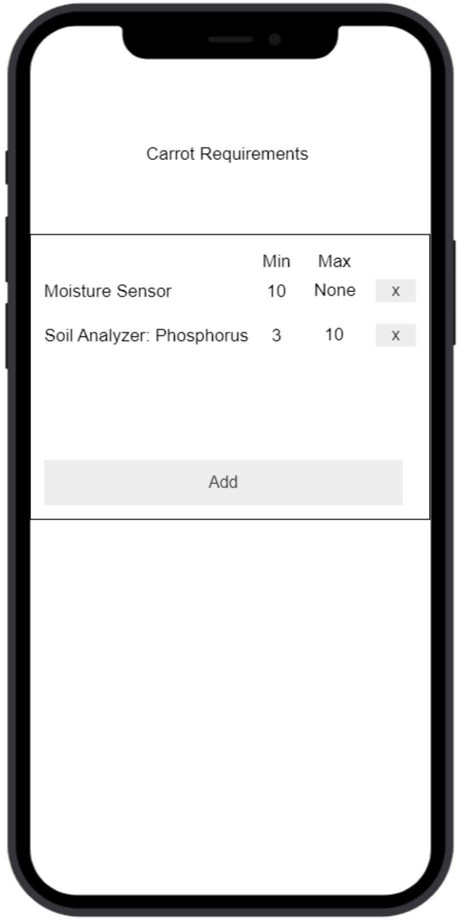
Carrot

Date Planted: 16/04/2021

Harvesting date: 16/08/2021

Requirements: View

Amount: 300 units

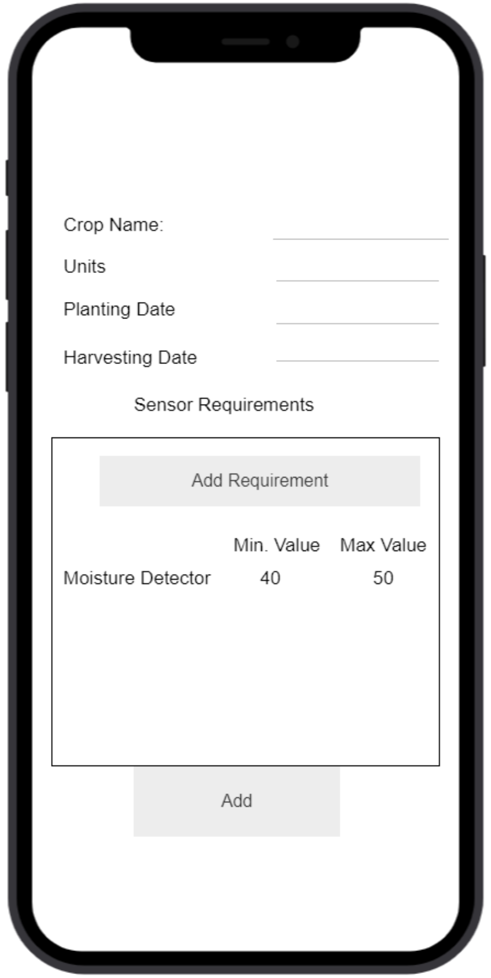


Carrot Requirements

	Min	Max	
Moisture Sensor	10	None	x
Soil Analyzer: Phosphorus	3	10	x

Prototype Demo View Crops

Prototype Demo- Adding Crops



Crop Name:

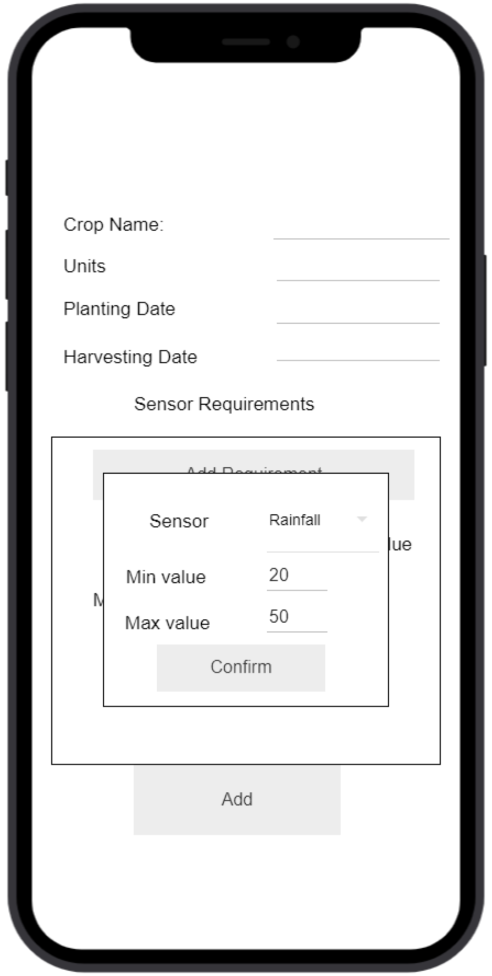
Units

Planting Date

Harvesting Date

Sensor Requirements

	Min. Value	Max Value
Moisture Detector	40	50



Crop Name:

Units

Planting Date

Harvesting Date

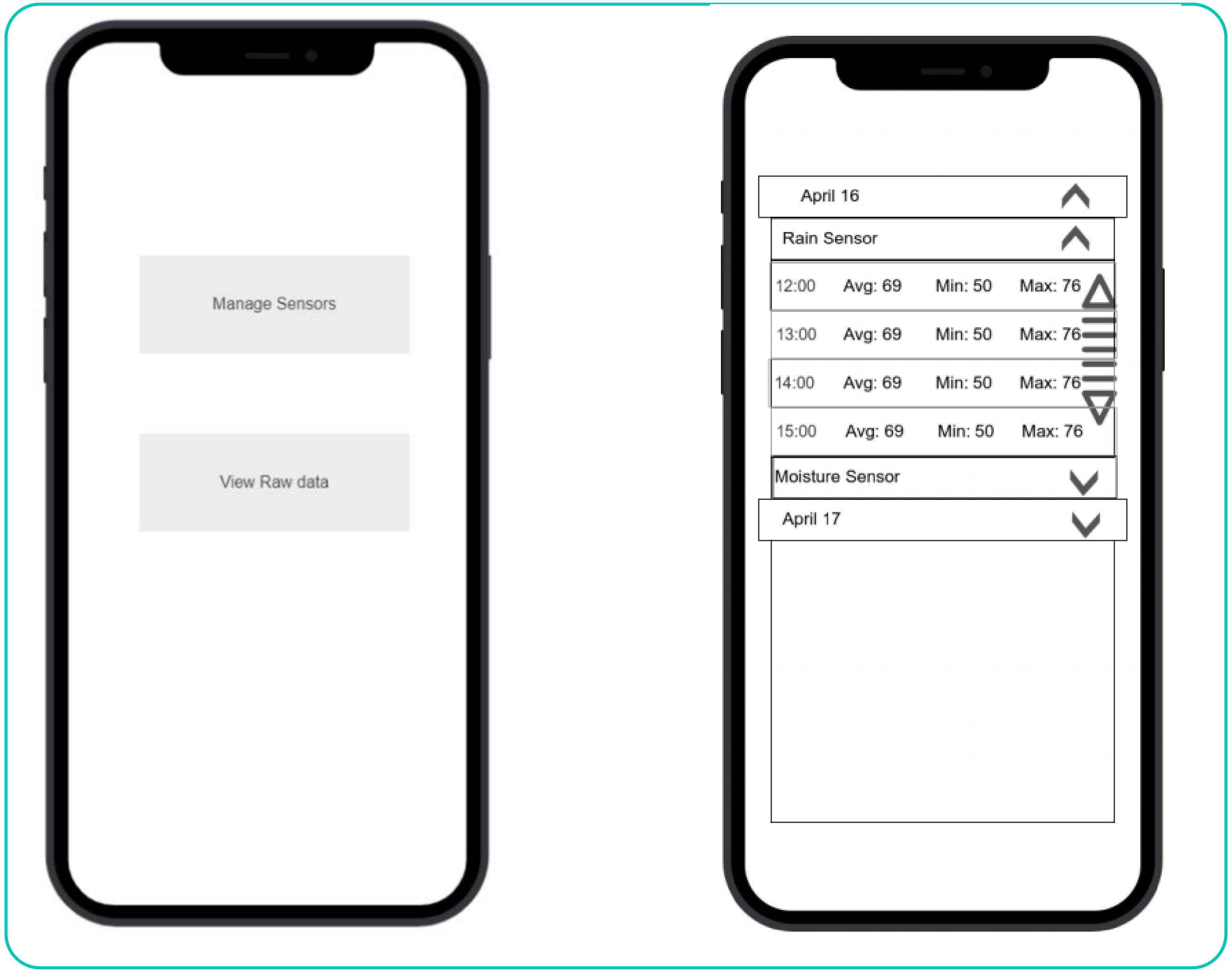
Sensor Requirements

Sensor: Rainfall

Min value: 20

Max value: 50

Prototype Demo- Sensor



April 16

Rain Sensor

12:00 Avg: 69 Min: 50 Max: 76

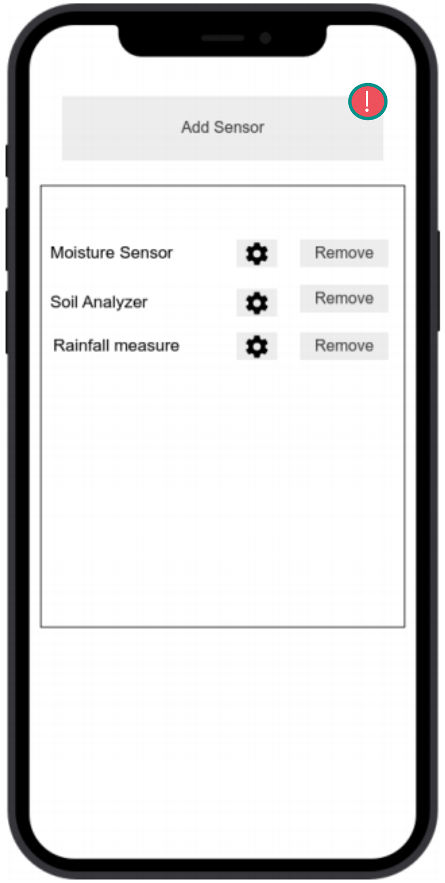
13:00 Avg: 69 Min: 50 Max: 76

14:00 Avg: 69 Min: 50 Max: 76

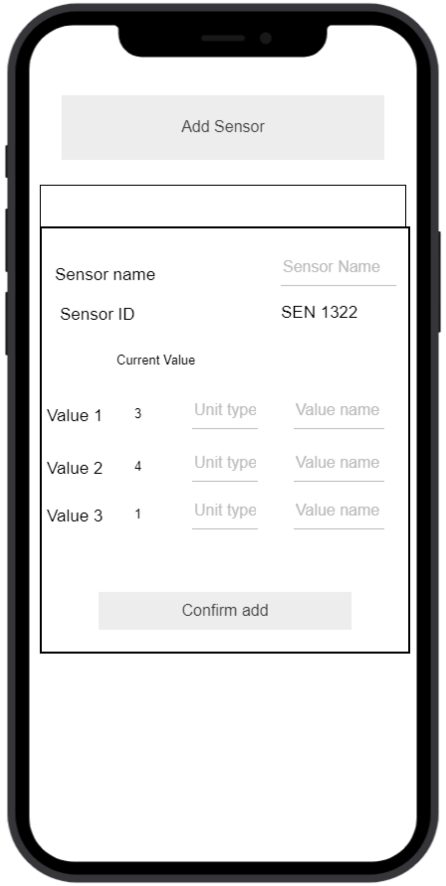
15:00 Avg: 69 Min: 50 Max: 76

Moisture Sensor

April 17



Moisture Sensor	⚙	Remove
Soil Analyzer	⚙	Remove
Rainfall measure	⚙	Remove

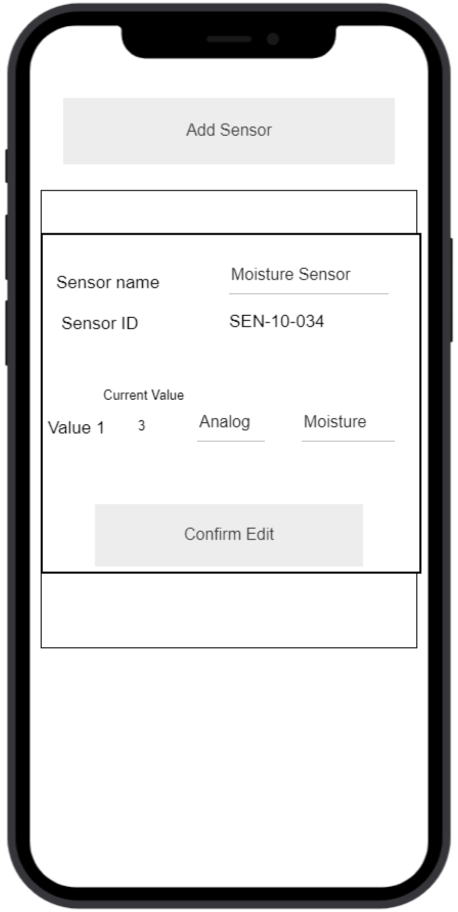


Sensor name Sensor Name

Sensor ID SEN 1322

Current Value

Value 1	3	Unit type	Value name
Value 2	4	Unit type	Value name
Value 3	1	Unit type	Value name



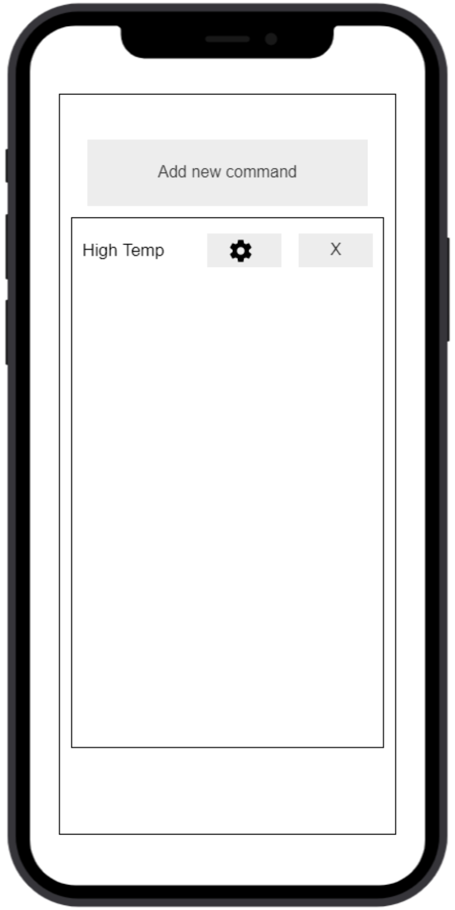
Sensor name Moisture Sensor

Sensor ID SEN-10-034

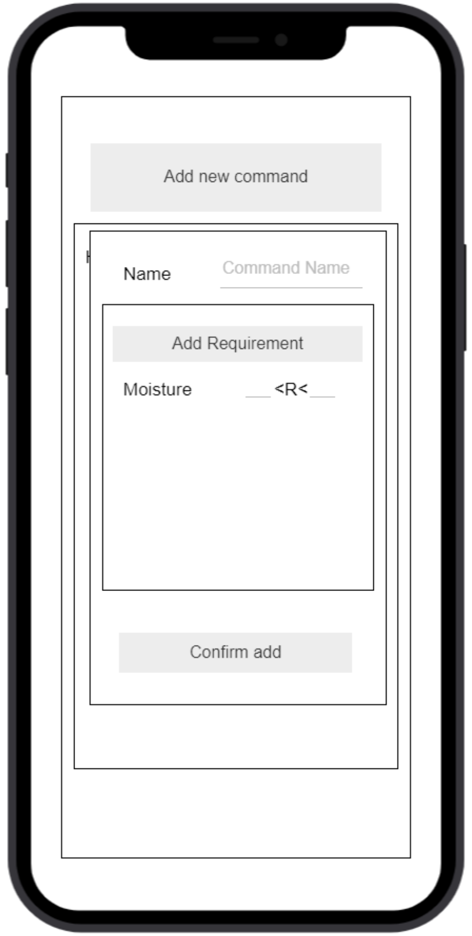
Current Value

Value 1 3 Analog Moisture

Prototype Demo-Sensor

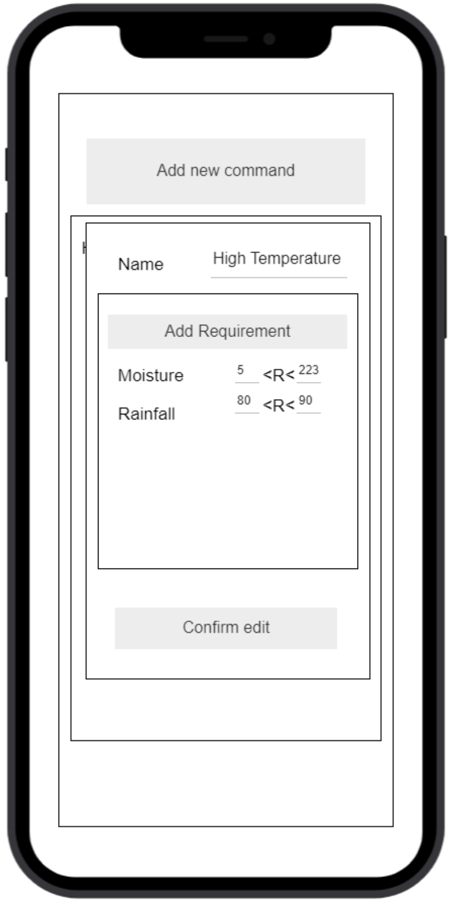


High Temp



Name Command Name

Moisture ___ <R< ___

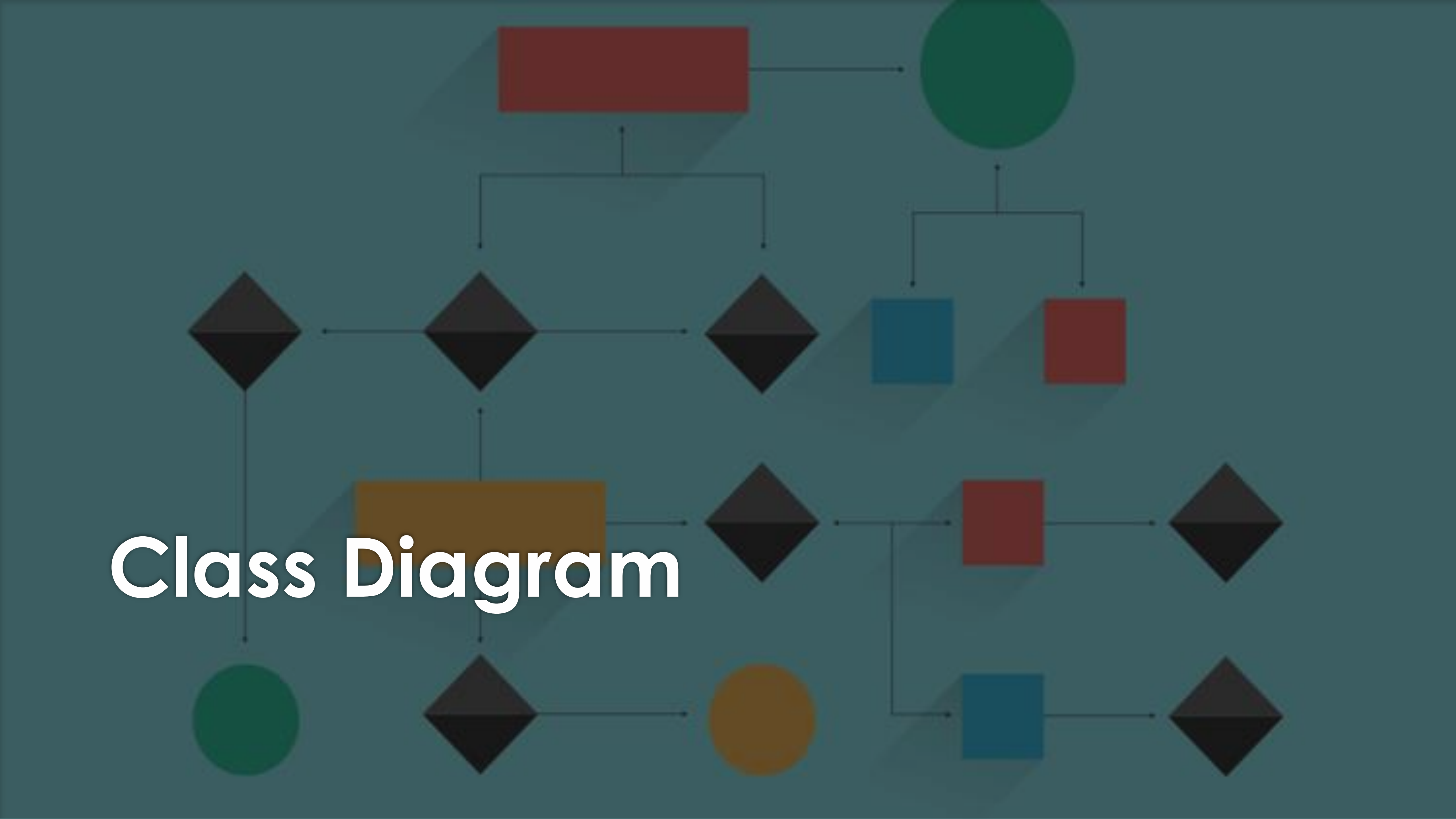


Name High Temperature

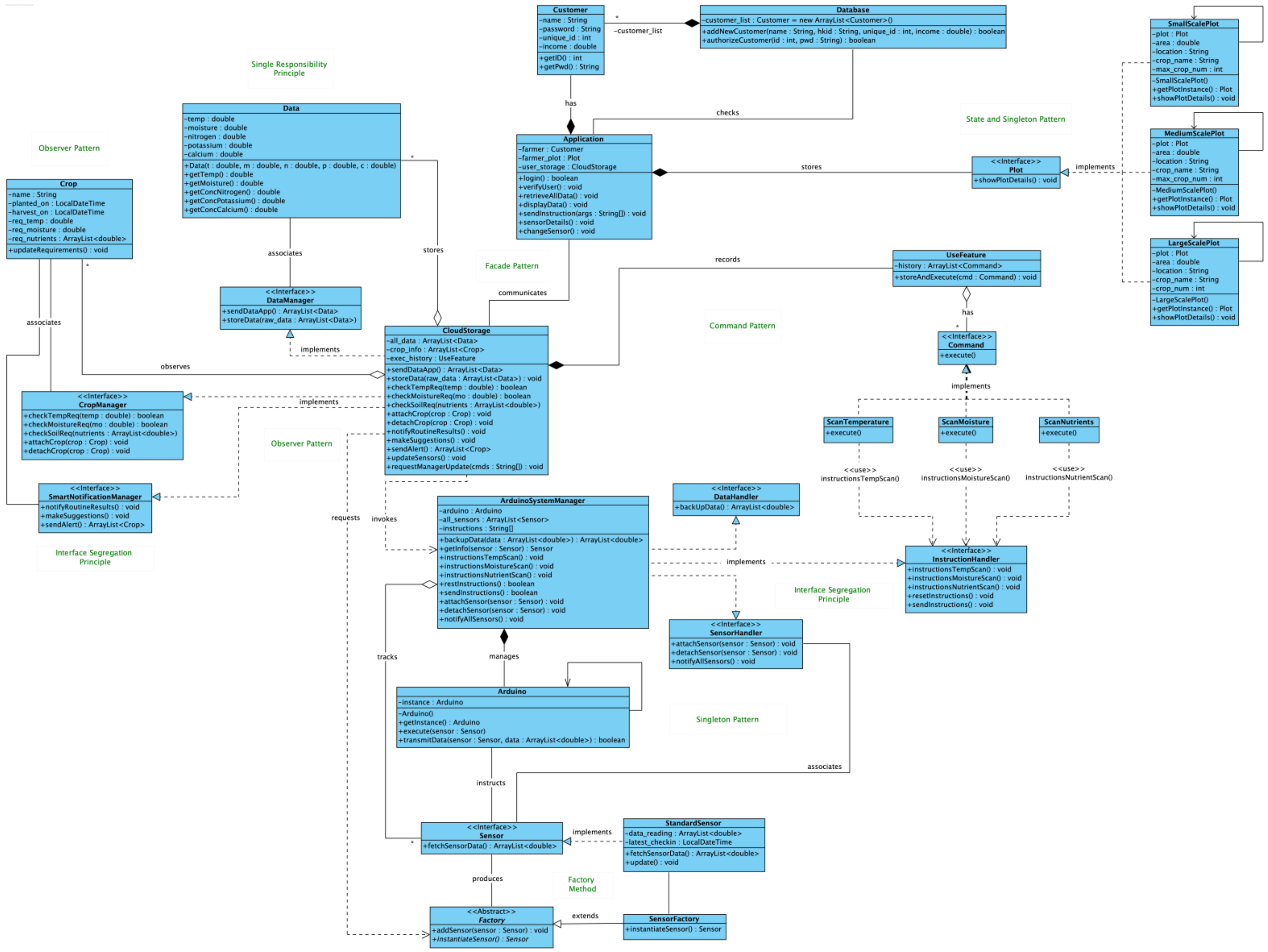
Moisture 5 <R< 223

Rainfall 80 <R< 90

Prototype Demo-Commands



Class Diagram


Single Responsibility Principle
Observer Pattern
Crop
-name : String
-planted_on : LocalDateTime
-harvest_on : LocalDateTime
-req_temp : double
-req_moisture : double
-req_nutrients : ArrayList<double>
+updateRequirements() : void
Data
-temp : double
-moisture : double
-nitrogen : double
-potassium : double
-calcium : double
+Data(t : double, m : double, n : double, p : double, c : double)
+getTemp() : double
+getMoisture() : double
+getConcNitrogen() : double
+getConcPotassium() : double
+getConcCalcium() : double
associates
stores
*
<<Interface>>
DataManager
+sendDataApp() : ArrayList<Data>
+storeData(raw_data : ArrayList<Data>)
implements
associates
observes
Customer
-name : String
-password : String
-unique_id : int
-income : double
+getID() : int
+getPwd() : String
-customer_list
Database
-customer_list : Customer = new ArrayList<Customer>()
+addNewCustomer(name : String, hkid : String, unique_id : int, income : double) : boolean
+authorizeCustomer(id : int, pwd : String) : boolean
has
checks
Application
-farmer : Customer
-farmer_plot : Plot
-user_storage : CloudStorage
+login() : boolean
+verifyUser() : void
+retrieveAllData() : void
+displayData() : void
+sendInstruction(args : String[]) : void
+sensorDetails() : void
+changeSensor() : void
stores
State and Singleton Pattern
<<Interface>>
Plot
+showPlotDetails() : void
implements
SmallScalePlot
-plot : Plot
-area : double
-location : String
-crop_name : String
-max_crop_num : int
-SmallScalePlot()
+getPlotInstance() : Plot
+showPlotDetails() : void
MediumScalePlot
-plot : Plot
-area : double
-location : String
-crop_name : String
-max_crop_num : int
-MediumScalePlot()
+getPlotInstance() : Plot
+showPlotDetails() : void
LargeScalePlot
-plot : Plot
-area : double
-location : String
-crop_name : String
-crop_num : int
-LargeScalePlot()
+getPlotInstance() : Plot
+showPlotDetails() : void
Facade Pattern
communicates
records
UseFeature
-history : ArrayList<Command>
+storeAndExecute(cmd : Command) : void
has
Command Pattern
<<Interface>>
Command
+execute()
implements
ScanTemperature
+execute()
ScanMoisture
+execute()
ScanNutrients
+execute()
<<use>>
instructionsTempScan()
<<use>>
instructionsMoistureScan()
<<use>>
instructionsNutrientScan()
CloudStorage
-all_data : ArrayList<Data>
-crop_info : ArrayList<Crop>
-exec_history : UseFeature
+sendDataApp() : ArrayList<Data>
+storeData(raw_data : ArrayList<Data>) : void
+checkTempReq(temp : double) : boolean
+checkMoistureReq(mo : double) : boolean
+checkSoilReq(nutrients : ArrayList<double>)
+attachCrop(crop : Crop) : void
+detachCrop(crop : Crop) : void
+notifyRoutineResults() : void
+makeSuggestions() : void
+sendAlert() : ArrayList<Crop>
+updateSensors() : void
+requestManagerUpdate(cmds : String[]) : void
<<Interface>>
CropManager
+checkTempReq(temp : double) : boolean
+checkMoistureReq(mo : double) : boolean
+checkSoilReq(nutrients : ArrayList<double>)
+attachCrop(crop : Crop) : void
+detachCrop(crop : Crop) : void
implements
Observer Pattern
<<Interface>>
SmartNotificationManager
+notifyRoutineResults() : void
+makeSuggestions() : void
+sendAlert() : ArrayList<Crop>
Interface Segregation Principle
requests
invokes
ArduinoSystemManager
-arduino : Arduino
-all_sensors : ArrayList<Sensor>
-instructions : String[]
+backupData(data : ArrayList<double>) : ArrayList<double>
+getInfo(sensor : Sensor) : Sensor
+instructionsTempScan() : void
+instructionsMoistureScan() : void
+instructionsNutrientScan() : void
+restInstructions() : boolean
+sendInstructions() : boolean
+attachSensor(sensor : Sensor) : void
+detachSensor(sensor : Sensor) : void
+notifyAllSensors() : void
<<Interface>>
DataHandler
+backUpData() : ArrayList<double>
implements
<<Interface>>
InstructionHandler
+instructionsTempScan() : void
+instructionsMoistureScan() : void
+instructionsNutrientScan() : void
+resetInstructions() : void
+sendInstructions() : void
Interface Segregation Principle
<<Interface>>
SensorHandler
+attachSensor(sensor : Sensor) : void
+detachSensor(sensor : Sensor) : void
+notifyAllSensors() : void
tracks
manages
Arduino
-instance : Arduino
-Arduino()
+getInstance() : Arduino
+execute(sensor : Sensor)
+transmitData(sensor : Sensor, data : ArrayList<double>) : boolean
Singleton Pattern
instructs
associates
<<Interface>>
Sensor
+fetchSensorData() : ArrayList<double>
implements
StandardSensor
-data_reading : ArrayList<double>
-latest_checkin : LocalDateTime
+fetchSensorData() : ArrayList<double>
+update() : void
produces
Factory Method
<<Abstract>>
Factory
+addSensor(sensor : Sensor) : void
+instantiateSensor() : Sensor
extends
SensorFactory
+instantiateSensor() : Sensor

Sequence Diagrams

Diagram-1: Registration

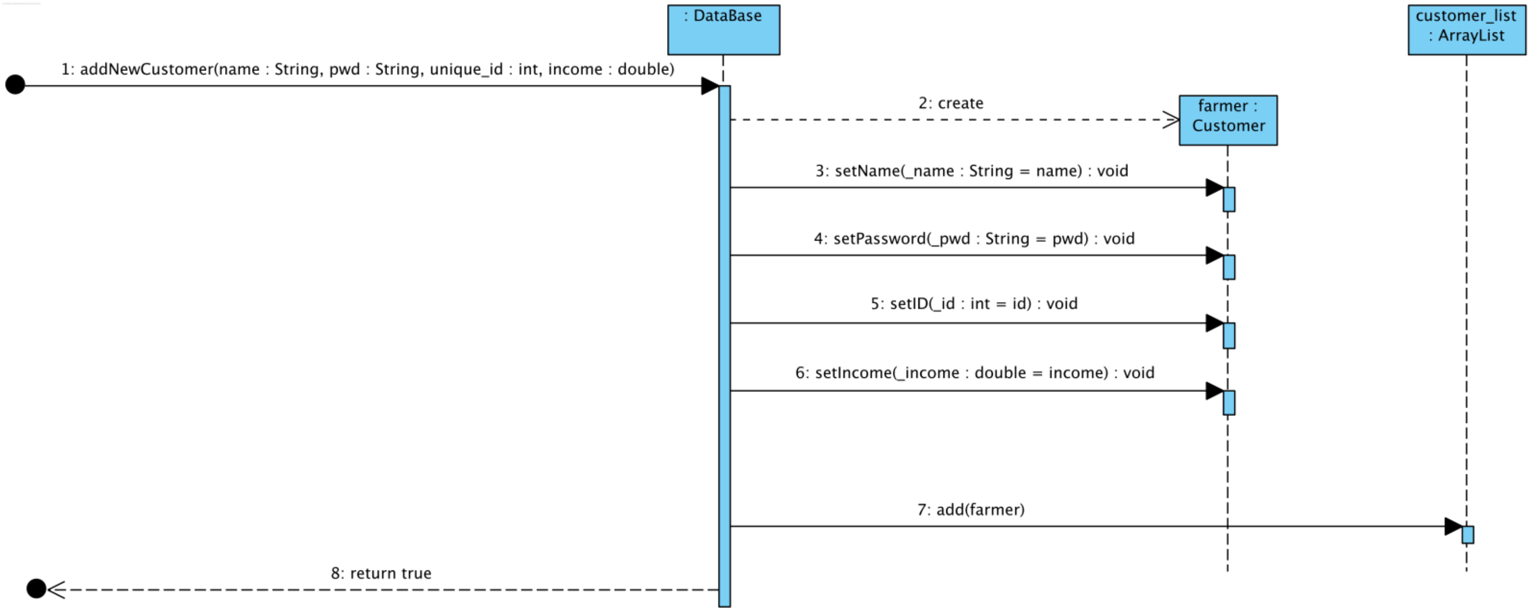
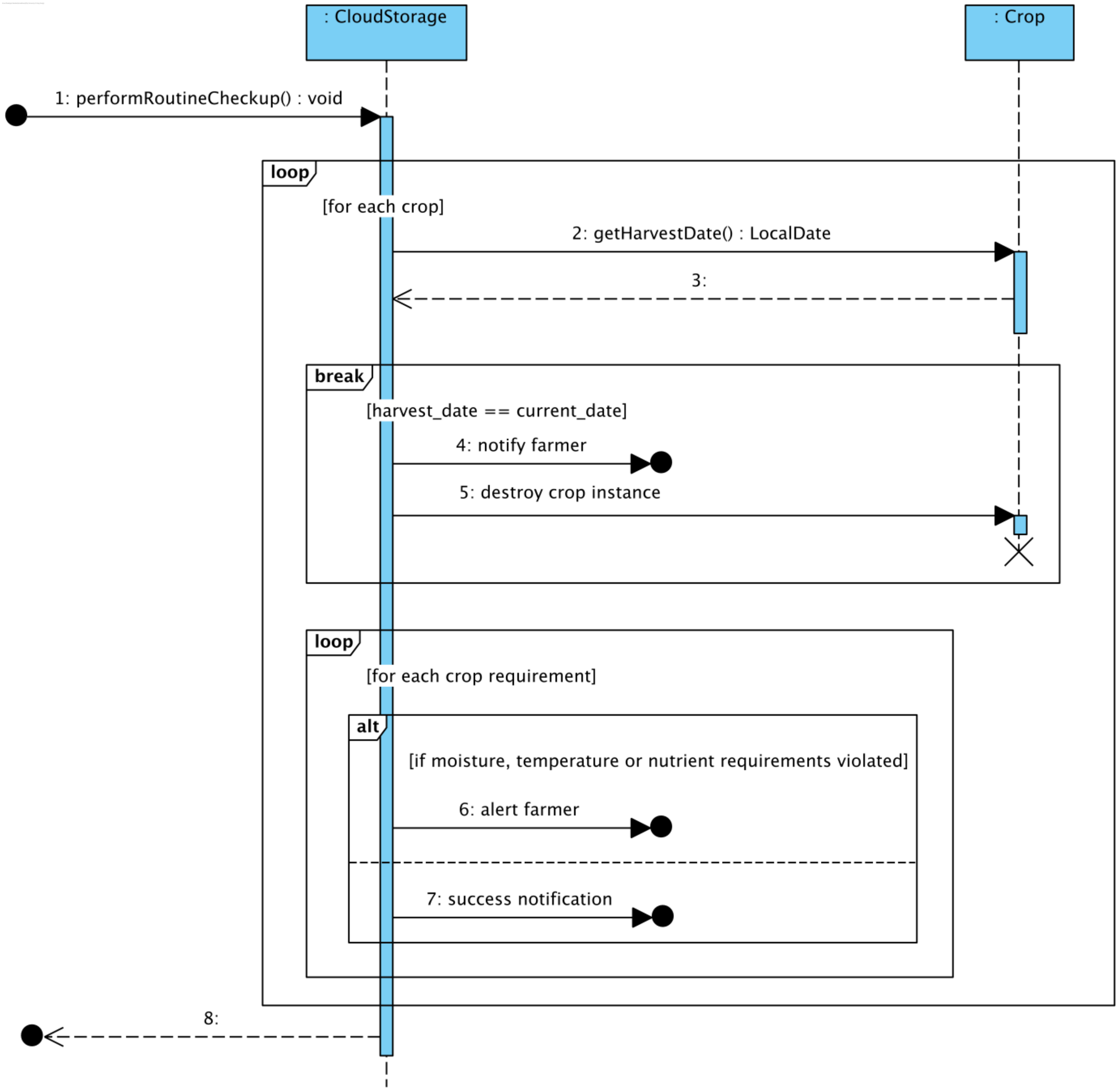

: DataBase
customer_list : ArrayList
1: addNewCustomer(name : String, pwd : String, unique_id : int, income : double)
2: create
farmer : Customer
3: setName(_name : String = name) : void
4: setPassword(_pwd : String = pwd) : void
5: setID(_id : int = id) : void
6: setIncome(_income : double = income) : void
7: add(farmer)
8: return true

Diagram-2: Smart Notification



: CloudStorage
: Crop
1: performRoutineCheckup() : void
loop
[for each crop]
2: getHarvestDate() : LocalDate
3:
break
[harvest_date == current_date]
4: notify farmer
5: destroy crop instance
loop
[for each crop requirement]
alt
[if moisture, temperature or nutrient requirements violated]
6: alert farmer
7: success notification
8:

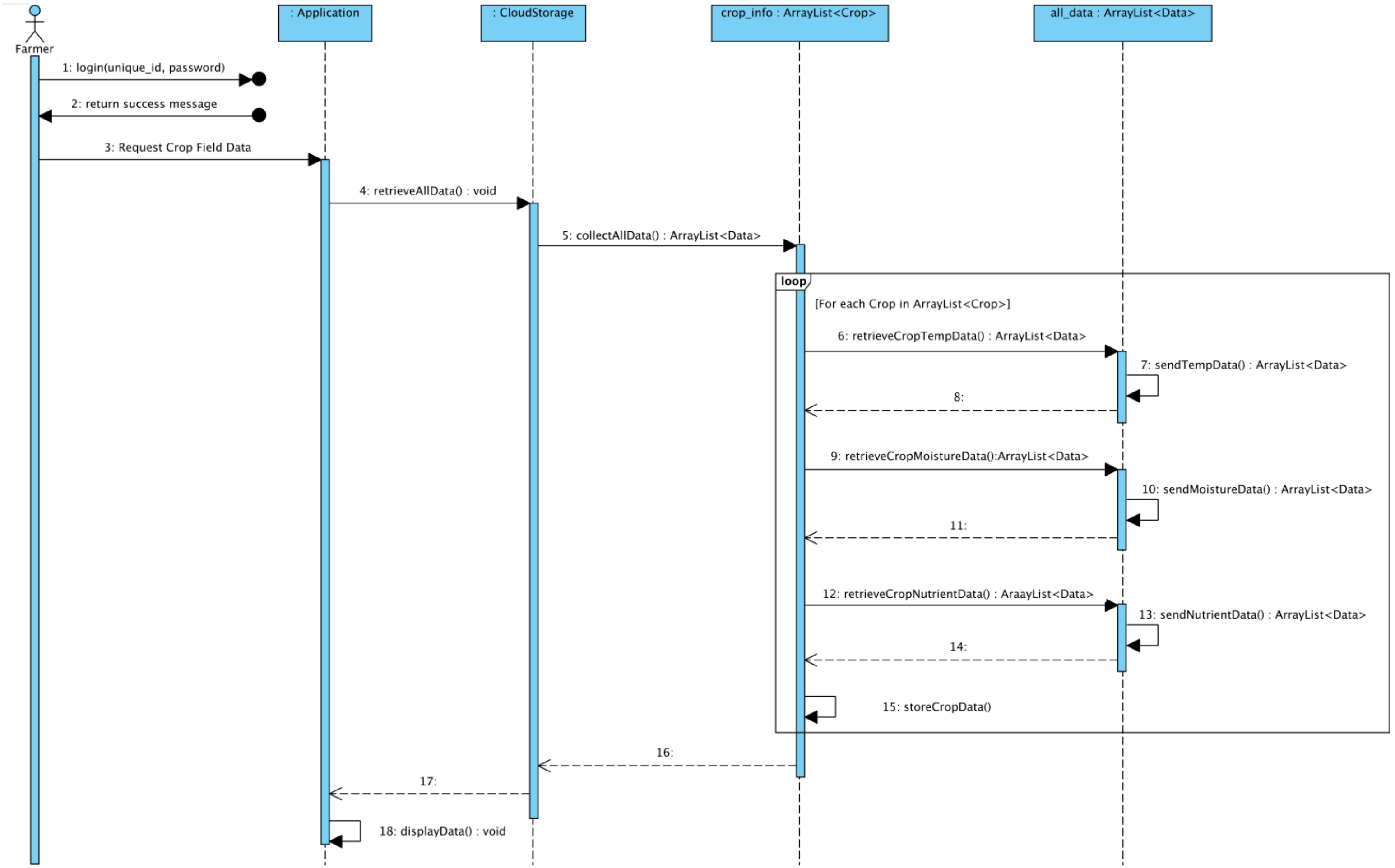
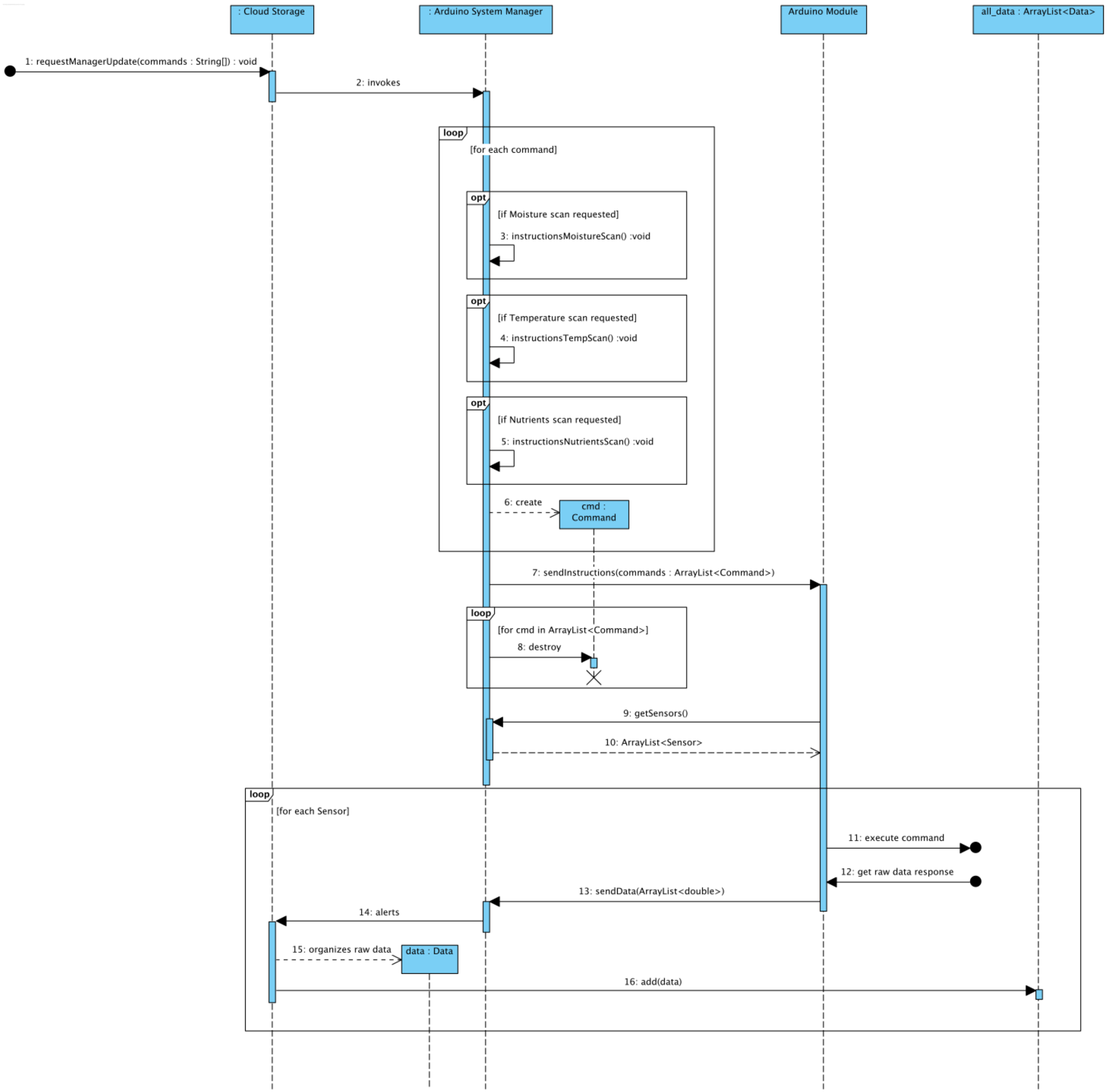
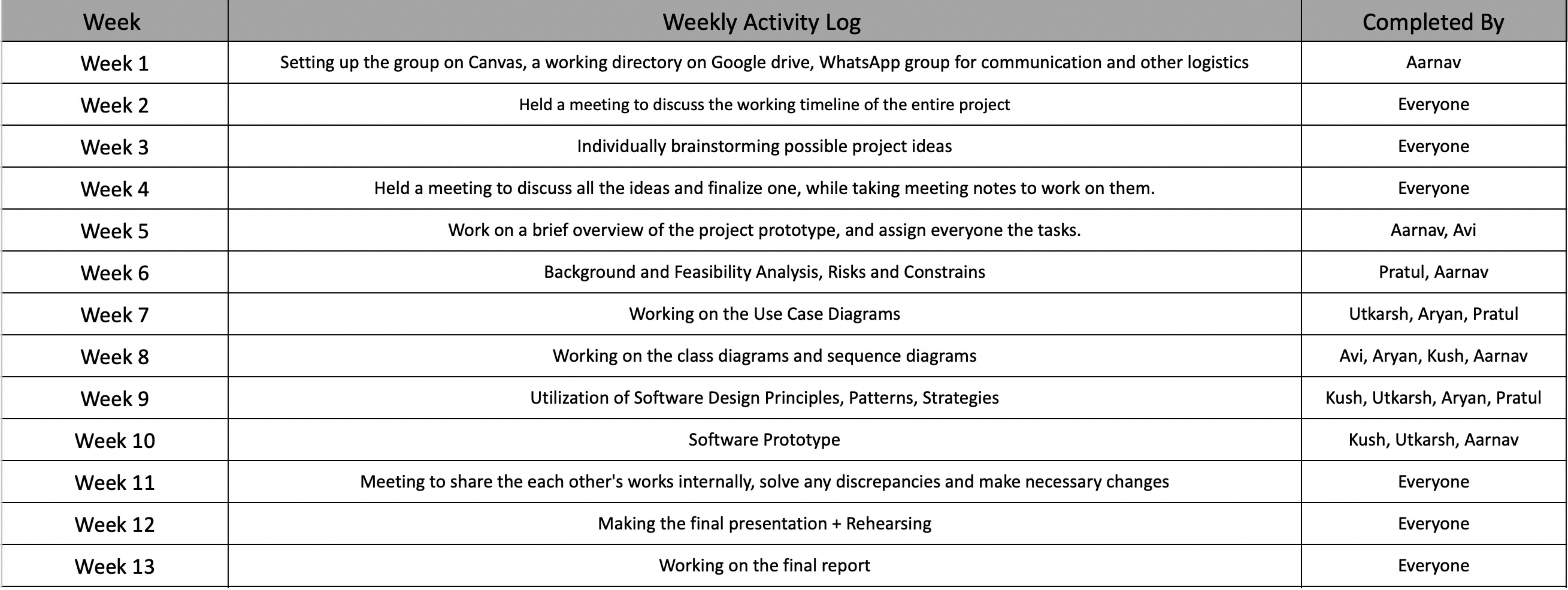

Farmer
: Application
: CloudStorage
crop_info : ArrayList<Crop>
all_data : ArrayList<Data>
1: login(unique_id, password)
2: return success message
3: Request Crop Field Data
4: retrieveAllData() : void
5: collectAllData() : ArrayList<Data>
loop
[For each Crop in ArrayList<Crop>]
6: retrieveCropTempData() : ArrayList<Data>
7: sendTempData() : ArrayList<Data>
8:
9: retrieveCropMoistureData():ArrayList<Data>
10: sendMoistureData() : ArrayList<Data>
11:
12: retrieveCropNutrientData() : AraayList<Data>
13: sendNutrientData() : ArrayList<Data>
14:
15: storeCropData()
16:
17:
18: displayData() : void

Diagram-3: Data Retrieval

Diagram-4: Backend Interaction



: Cloud Storage
: Arduino System Manager
Arduino Module
all_data : ArrayList<Data>
1: requestManagerUpdate(commands : String[]) : void
2: invokes
loop
[for each command]
opt
[if Moisture scan requested]
3: instructionsMoistureScan() :void
opt
[if Temperature scan requested]
4: instructionsTempScan() :void
opt
[if Nutrients scan requested]
5: instructionsNutrientsScan() :void
6: create
cmd : Command
7: sendInstructions(commands : ArrayList<Command>)
loop
[for cmd in ArrayList<Command>]
8: destroy
9: getSensors()
10: ArrayList<Sensor>
loop
[for each Sensor]
11: execute command
12: get raw data response
13: sendData(ArrayList<double>)
14: alerts
15: organizes raw data
data : Data
16: add(data)



Week	Weekly Activity Log	Completed By
Week 1	Setting up the group on Canvas, a working directory on Google drive, WhatsApp group for communication and other logistics	Aarnav
Week 2	Held a meeting to discuss the working timeline of the entire project	Everyone
Week 3	Individually brainstorming possible project ideas	Everyone
Week 4	Held a meeting to discuss all the ideas and finalize one, while taking meeting notes to work on them.	Everyone
Week 5	Work on a brief overview of the project prototype, and assign everyone the tasks.	Aarnav, Avi
Week 6	Background and Feasibility Analysis, Risks and Constrains	Pratul, Aarnav
Week 7	Working on the Use Case Diagrams	Utkarsh, Aryan, Pratul
Week 8	Working on the class diagrams and sequence diagrams	Avi, Aryan, Kush, Aarnav
Week 9	Utilization of Software Design Principles, Patterns, Strategies	Kush, Utkarsh, Aryan, Pratul
Week 10	Software Prototype	Kush, Utkarsh, Aarnav
Week 11	Meeting to share the each other's works internally, solve any discrepancies and make necessary changes	Everyone
Week 12	Making the final presentation + Rehearsing	Everyone
Week 13	Working on the final report	Everyone

Teamwork

- Had a really good working dynamic
- Zoom and different timezones acted as a limitatiom

Overall Reflection

01

Opportunity to implement Software Design principles

02

Apply different concepts and integrate them altogether.

03

There are a lot of opportunities to grow the project. But we did meet the basic requirements that we wanted to implement.

04

Add functionalities for FarmAssist to make decisions on its own, and carry them out, such as watering the soil, etc.

Conclusion

Better understanding of the crops at any time of the day.

Can take more informed decisions

The app can be connected to as many sensors as needed.