

Miniaturized Automated Greenhouse

Technical Report

Table of figures

Figure no.	Title	Page
Figure 1.	(Somov et al., 2018)	7
Figure 2.	UML Use Case Diagram	8
Figure 3.	Development Methodology	9
Figure 4.	Work Breakdown Structure	11
Figure 5.	System Architecture Mind-Map	12
Figure 6.	Water Pump	13
Figure 7.	Soil Moisture Sensor	14
Figure 8.	L293D Motor Driver IC pinout	15
Figure 9.	Circuit Diagram	19

Contents

Content	Page
1] Introduction	6
1.1] Brief Project Description	6
1.2] Problem Definition	6
1.3] Project Aims	6
2] Literature Review	6
3] Pre-Design	7
3.1] Overview	7
3.2] Research & Development	8
3.3] Deliverables	9
3.4] Project Planning	9
3.5] Ethical Issues	11
4] Design & Implementation	12
4.1] Design	12
4.2] Implementation	19
5] Evaluation	46
5.1] Testing	46
6] Conclusion	53
6.1] System Limitations	54
6.2] Future Directions	54
7] References	55
8] Appendices	55

Introduction

Brief Project Description:

Our project is an automated greenhouse that actively measures the environmental parameters based on them can provide plants inside with optimal temperature & irrigation based on what the plant exactly needs. This greenhouse can also have its environment requirements set by a user, the irrigation control can be provided to the user as well through an android (mobile) application, making the greenhouse semi-automated. The user can also monitor the current environmental state of the greenhouse. A database stores the requirements of the plant (optimal temperature, irrigation), which is used by the system to automate itself suiting to the needs of the plant. This database keeps receiving updates as part of maintenance & improvements to the system (new plants are added). The setup created in this project can be adapted to large scales

Problem Definition:

Contrary to popular belief, growing a plant isn't only about watering it once or twice a day and exposing to sunlight. There are many factors that come into play to make sure that a plant grows to yield its best properties in the best way. The more optimal conditions provided the better yield a plant gets. Such factors are, water, light, temperature, soil nutrients, water drainage, humidity, sterility/non-sterility. Monitoring all these factors and acting upon them correctly requires dedication and in many cases expertise. When done in large scale, the complexity of growing plants optimally to gain a perfect yield is increased massively. In large scale greenhouses, if manual labour is used, it could also become immensely expensive due to the amount of labour, expertise & planning required.

Project Aims

Our project aims to solve the above-mentioned problems by:

- Making an environment monitoring system that can help monitor the conditions efficiently.
- Making a system that makes it easy to set requirements (by the user) according to what plants the greenhouse consists of.
- Making a database that consists of requirements that help the plant grow optimally.
- Making a system that can act upon the requirements (regulate temperature, irrigate)
- Minimizing labour costs by automating tasks such as irrigation.

Literature Review

In work (Hemming et al., 2019), they have described to us how a competition was held in place where each team was given a greenhouse and control to factors such as ventilation, heating, lighting, screening, lighting, fogging (humidity), nutrients and water supply. It shows us how the teams have manipulated the conditions using AI algorithms to provide optimal growth conditions to increase cucumber yield at the end of the competition duration. A manually grown reference was also set to set a benchmark for the teams. A team out of them had outperformed this manually grown reference by a considerable amount. This goes to show how important it is to control all these factors and how it can be made a lot easier by automatic the greenhouse itself.

In work (Somov et al., 2018), as well, we observe a system overview that describes the IoT enabled greenhouse (for tomatoes) which comprises of a wireless sensor network, AI & Cloud Computing. Below is the system overview that we had observed from this work.

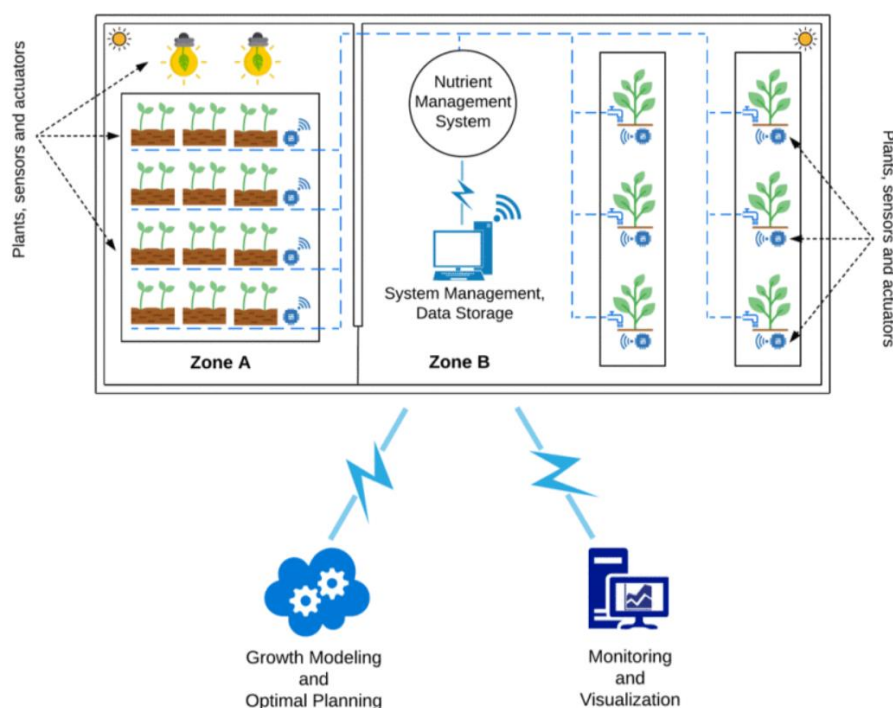


Figure. 1 (Somov et al., 2018)

In work (Ullah et al., 2022) there has been proposed a system that can help optimize control of climate within the greenhouse and the importance of making this system as energy efficient as possible. As mentioned in this work as well, temperature also plays a huge role continually in making or breaking the quality of a plant's life & its yield.

The works we have reviewed had helped us understand what page we're on and provided us with suitable frameworks like which we could develop ours.

Pre-Design

Overview:

This project required us to focus on both hardware & software, our technical requirements were:

- Sense the environment (Soil Moisture level, temperature, humidity)
- Send current sensed values to a cloud non-relational database and store history of values on a relational database that is being stored on a cloud as well.
- Control the environment (Temperature, Irrigation) based on what's optimal for the plant (the optimal conditions and requirements like temperature & irrigation intervals/volume are stored on the non-relational database).
- Store control history (history of when irrigations occurred) on relational database on cloud.
- Automate the above features.
- If needed provide control to user (Irrigate through application) or change the optimal requirements of the plant themselves (if they have expertise) to suit the plants they have.
- Help the user monitor the environment remotely (through Android Application or through the relational & non-relational database).
- Notify the user when the plants need to be watered through the android application.

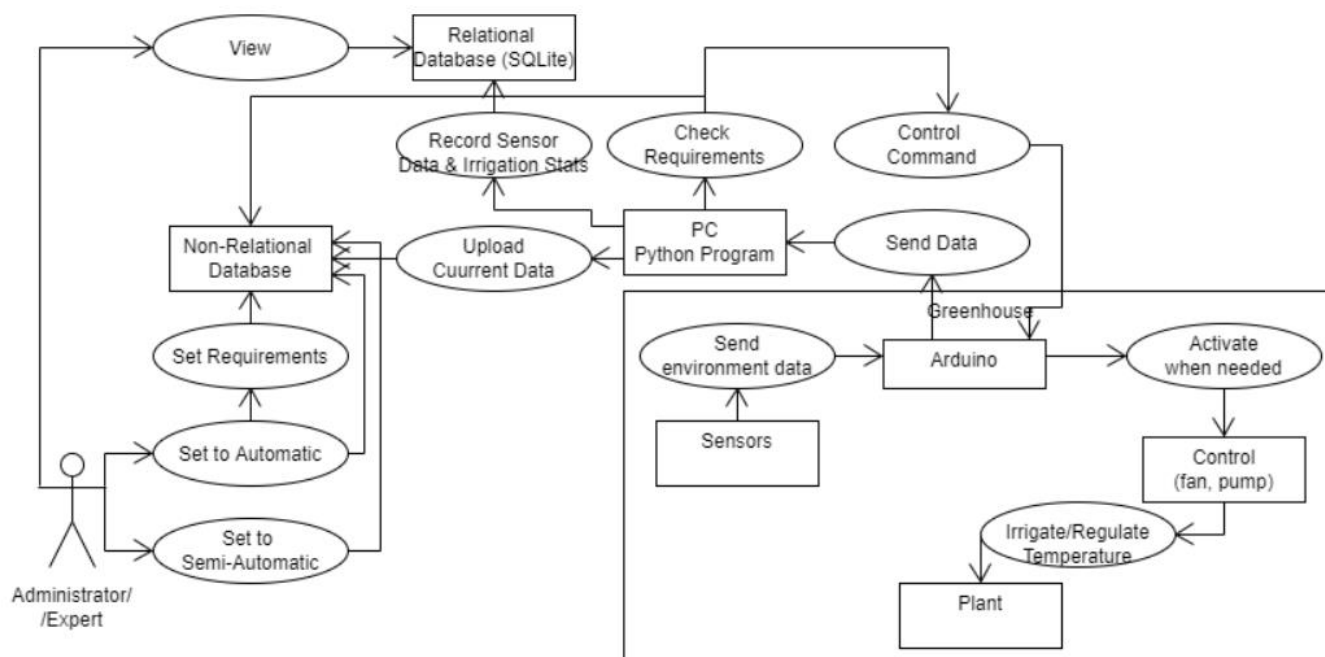


Figure 2. UML use case Diagram

Research & Development

Research Methodology:

Our main subjects of research were:

- Stakeholders and their requirements.
- A basic framework that we can adapt to (From Figure 1.).
- Hardware components used (Arduino UNO R3, Water Pump, Moisture level sensor, DHT11 Sensor, Relays, DC motors & motor drivers)
- Software components used (Arduino IDE, Python, Firebase Non-Relational Cloud Database, Firebase Cloud Storage, SQLite Relational Database, Android Studio)

Qualitative research is done to learn about these subjects and gather information that helps us build this project, which is an initial push. Onwards we must go through trial-error, experimenting with our hardware & software components to know what works best towards our requirements.

Development Methodology:

Our project comprises of the initial build to which updates & improvements are done. For this reason, we adopt the **spiral development methodology** wherein we come back to the same step repeatedly to improve the project we have already completed.

- The database of plants keeps getting updates i.e., new plants are constantly added to the database which allows the user to be able to set what their plant is and decrease the need for them to custom set their optimal growth requirements.
- The application will also keep getting updated (UI, Control features, Monitoring Features etc.)

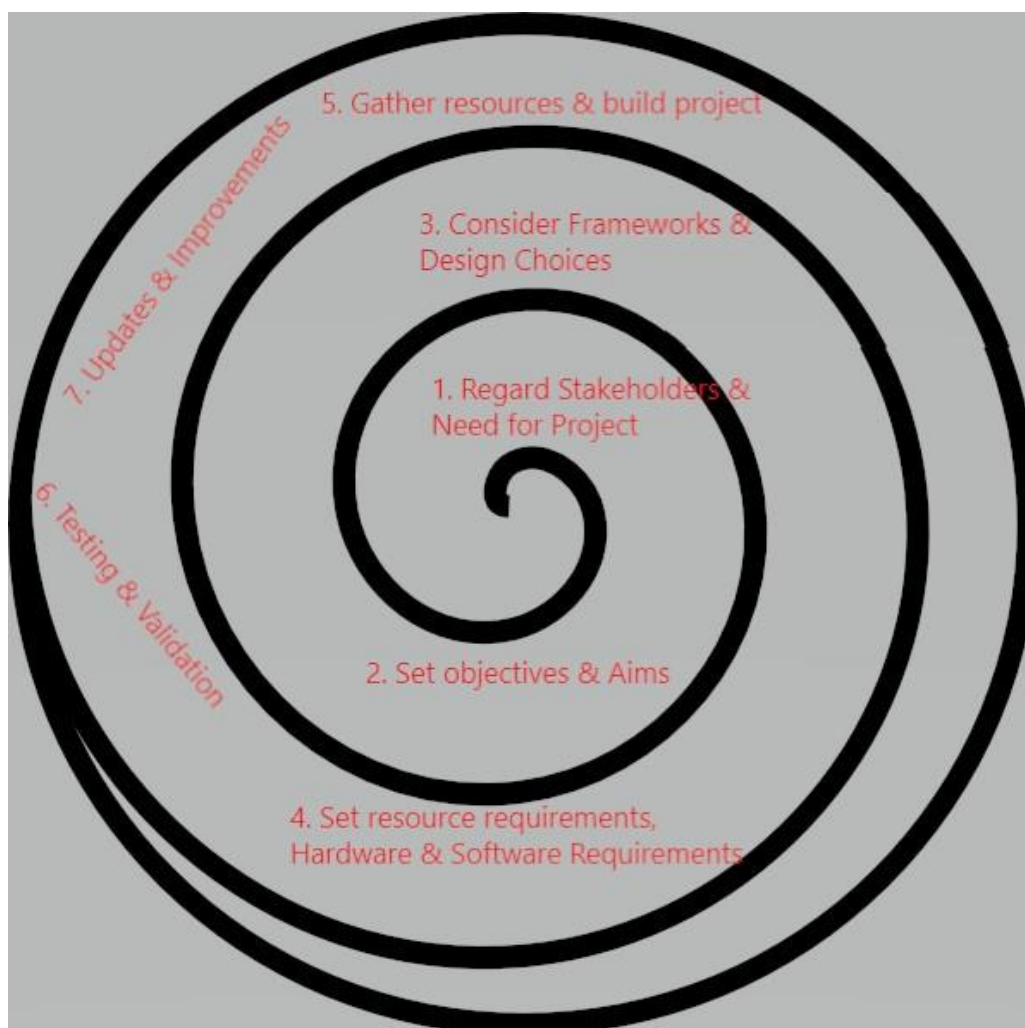


Figure 3. Development Methodology

Deliverables:

Functional Deliverables:

- Environment Sensing (Displayed on Relational & Non-Relational Databases, Android Application)
- Environment Control (Automated through Optimal Requirements Database or User Set)
- Environment Control (Semi-Automated: Irrigate by user input through Android Application and send user notification if plants need to be irrigated)

Non-Functional Deliverables:

- Miniaturized build (small enough to be portable)
- Application UI to be clean.
- Relational Database to be in database file format (.db).

Project Planning (Gantt Chart, WBS):

Our steps to complete this project are as follows, note that the steps start from initializing the idea, content of the project and planning.

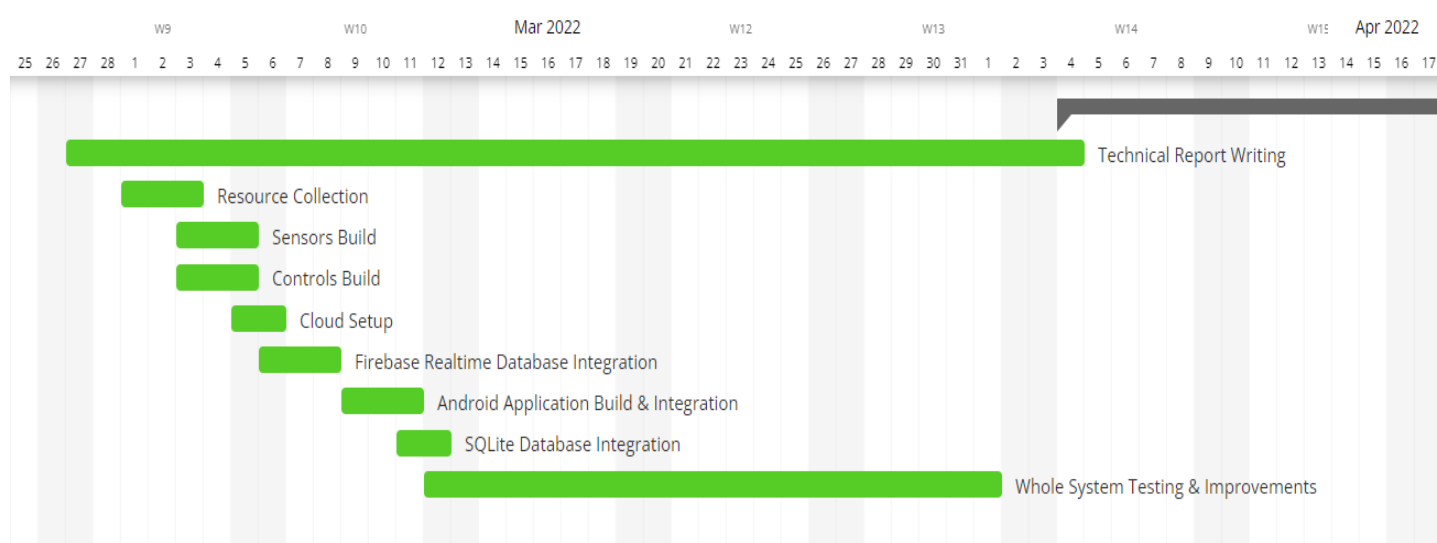
- **Resource Collection:** Resource collection includes research, gathering hardware and required materials. [
- **Sensing Equipment:** Assembling and programming sensors (Moisture Level, DHT11), Testing.
- **Control Equipment:** Assembling and programming controls (Water Pump, DC Motor Fan), Testing

- **Cloud Setup:** In firebase setup non-relational real-time database for data pulling & pushing. Setting up sample optimal requirements (required irrigation duration, volume & temperature). Setup Firebase Storage Bucket for storing the relational database file created later.
- **Non-Relational Database Integration:** Pushing data from Arduino Sensors to Firebase real-time database & pulling optimal requirements, Testing.
- **Android Application Build & Integration:** Build android application and connect it to the Firebase real-time database to monitor the current environment variables. Change variables on the Non-Relational Database which in-turn gives control to the greenhouse (Irrigate) through android application., Testing.
- **Relational Database Integration:** Setting up relational database through SQLite, saving environment stats history and irrigation stats history into relational database (database file .db) & uploading to cloud (Firebase Storage Bucket).
- **Whole System Testing & Improvements**

Project Timeline

Task	Start (DD/MM/YY)	End (DD/MM/YY)	Duration (Days)
Project	27/02/2022	04/04/2022	37
Technical Report Writing	27/02/2022	04/04/2022	37
Resource Collection	01/03/2022	03/03/2022	3
Sensors Build	03/03/2022	05/03/2022	3
Controls Build	03/03/2022	05/03/2022	3
Cloud Setup	05/03/2022	06/03/2022	2
Firebase Realtime Database Integration	06/03/2022	08/03/2022	3
Android Application Build & Integration	09/03/2022	11/03/2022	3
SQLite Database Integration	11/03/2022	12/03/2022	2
Whole System Testing & Improvements	12/03/2022	01/04/2022	21

Gantt Chart



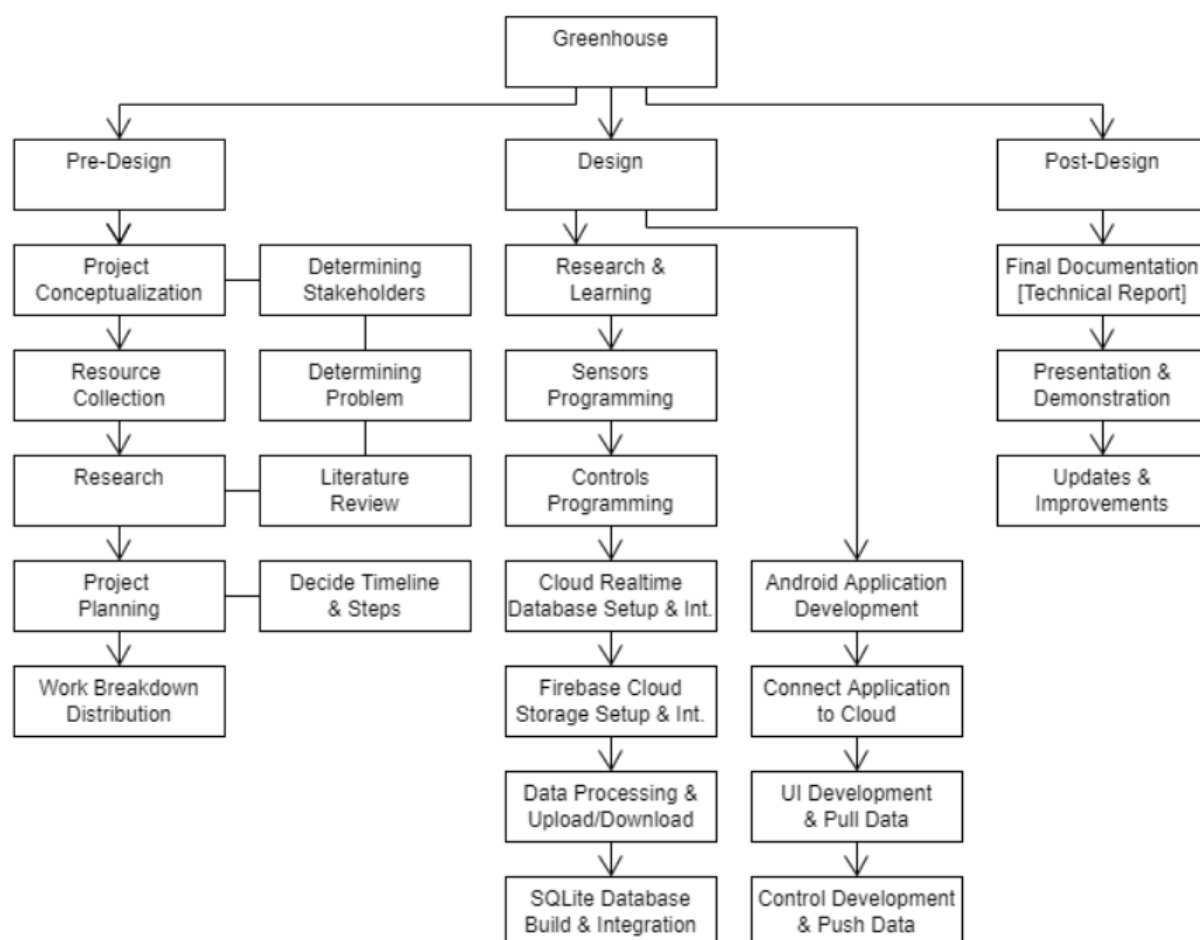


Figure 4. Work Breakdown Structure

Ethical Issues:

Our project provides us with many positive outcomes, however, not all are positive for everyone. Our project focused on automating the greenhouse which in a way also means reduction of manual labour. This reduction could lead to unemployment & financial disadvantages to the labourers. It is hard to distinguish which is more important, increased yield if automation techniques are used or increased employment for the people who can work in these greenhouses & farms if they were not automated. We still choose to however make a adaptation to the solution to support the idea and framework if they were to be ever used.

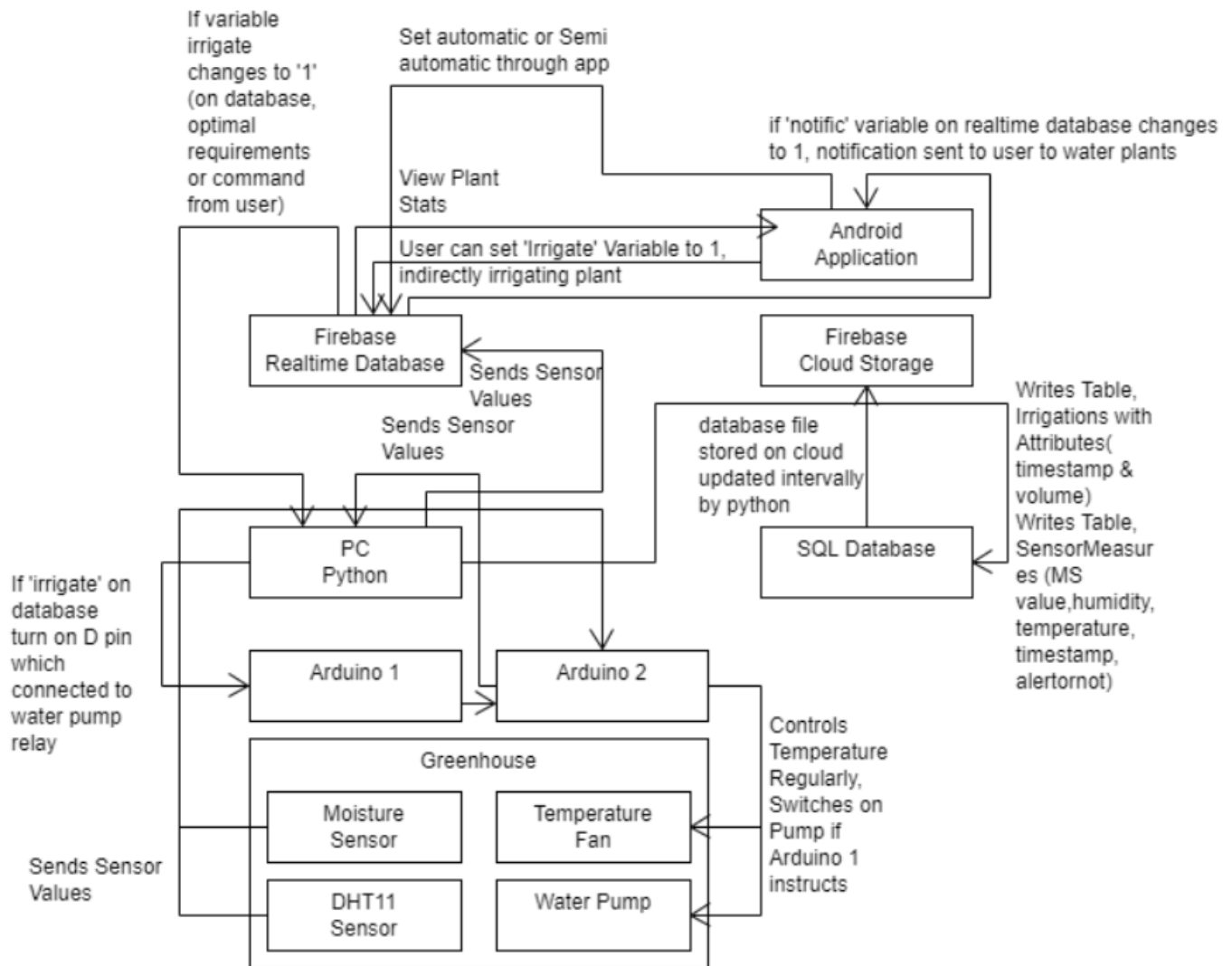


Figure 5. System Architecture Mind-Map

Hardware Requirements:

- **Arduino UNO R3**
- **Greenhouse [Housing]**
- **Water Pump**

The water pump used in our project is small and submersible (Fig. 5). It takes 3 to 5 volts DC voltage.

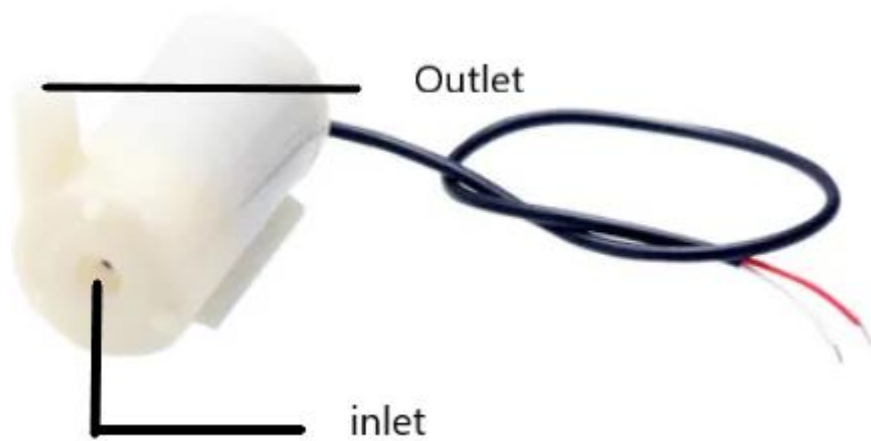
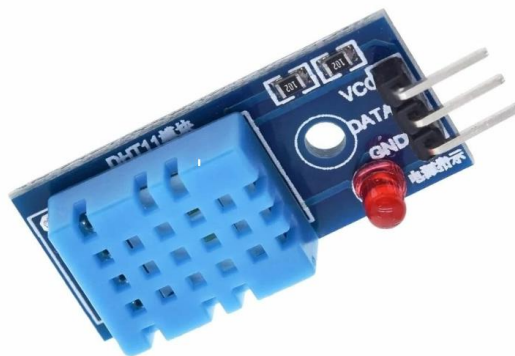


Figure 6. Water Pump

- **Relay**
- **DHT11 Sensor**

This is the sensor that we used to detect air temperature & humidity. The working principles are a thermistor (material that changes resistance when temperature changes) and a capacitive humidity sensor coupled with an IC. The pinouts can be seen in the picture below, there's VCC, DATA, GND pins.

- 5V is required for VCC.
- The GND is connected to Ground
- DATA connected to a digital pin.



DHT11 Sensor

- **Soil Moisture Sensor**

Soil Moisture sensor can estimate or measure the level of water within the soil. The working principle behind this is that dielectric permittivity is a function of the water content, the sensor ends up creating voltage proportional to the dielectric permittivity i.e. the water content of the soil. A comparator is used in addition to this making it compatible to use with our Arduino UNO R3.

- Sensor + to Comparator +
- Sensor – to Comparator –
- Comparator Analog to Arduino Analog pin

- Comparator VCC to 3V
- Comparator GND to Ground

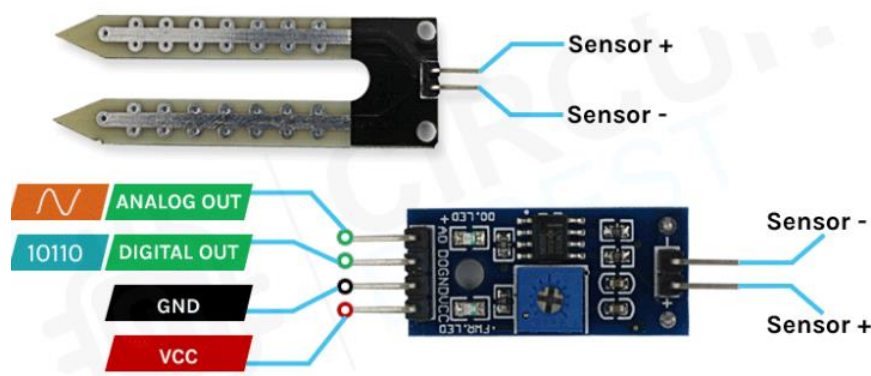


Figure 7. Soil Moisture Sensor

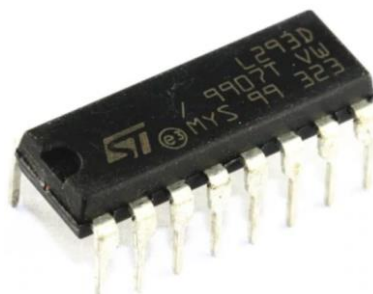
• DC Stepper Motor & L293D Motor Driver IC

The DC Motor used takes voltage of up to 6 volts and requires a L293D Motor Driver in order to make it work. PWM is a technique that's used to control the speed of the motor and an 'H' bridge configuration is used to specify the direction of rotation. The motor driver is used for PWM.

- Motor + to Motor Driver IC OUT1
- Motor – to Motor Driver IC OUT2
- Motor Driver IC VCC to 5V
- Motor Driver IC GND to Ground
- Motor Driver ENA to Arduino Digital Pin X
- Motor Driver IN1 to Arduino Digital Pin Y
- Motor Driver IN2 to Arduino Digital Pin Z

The ENA pin simply enables all the pins we have connected just now

The OUT1 and OUT2 pins should be turned on (through Arduino Code By turning on either only IN1 or either only IN2) one at a time, the specify the direction in which the motor will rotate.



L293D Motor Driver IC

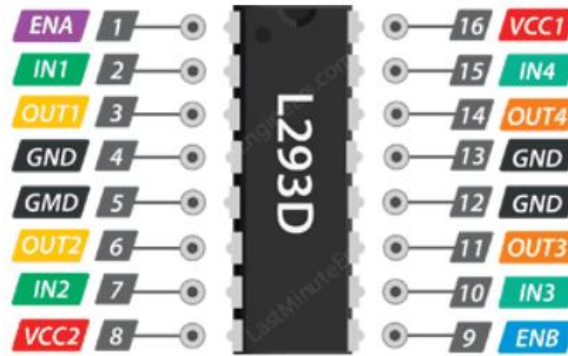
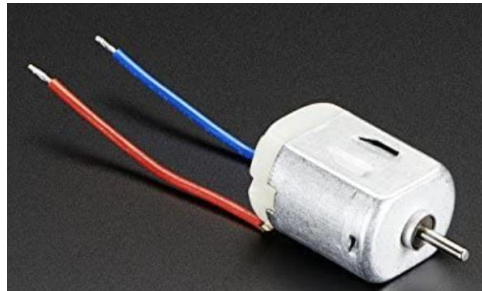


Figure 8. L293D Motor Driver IC pinout



4.5-6V Stepper Motor

- **Breadboard**

Software Requirements:

- **Arduino IDE (DHT11 Library, FirmataExpress)**
- **Python3 (Pymata4, sqlite3, datetime, serial, pyrebase, time)**

Python is an open-source high-level programming language mostly used to automate tasks, build software or websites and in our case conduct data analysis, upload to cloud, retrieving from cloud, write SQL database, Control greenhouse. The properties and ease of use in the language makes it suitable for our project. The Default IDLE will be used to build scripts

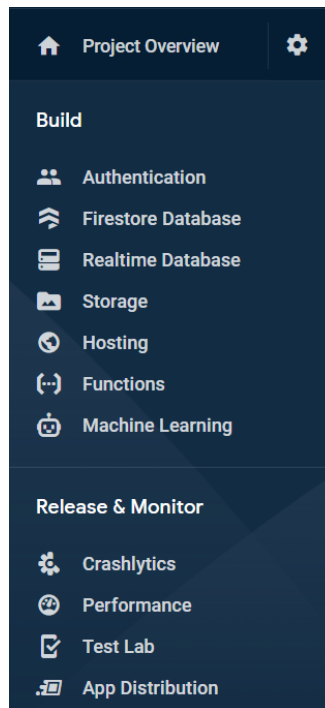
- **Google Firebase (Realtime Database, Storage Bucket)**

Firebase Realtime Database:

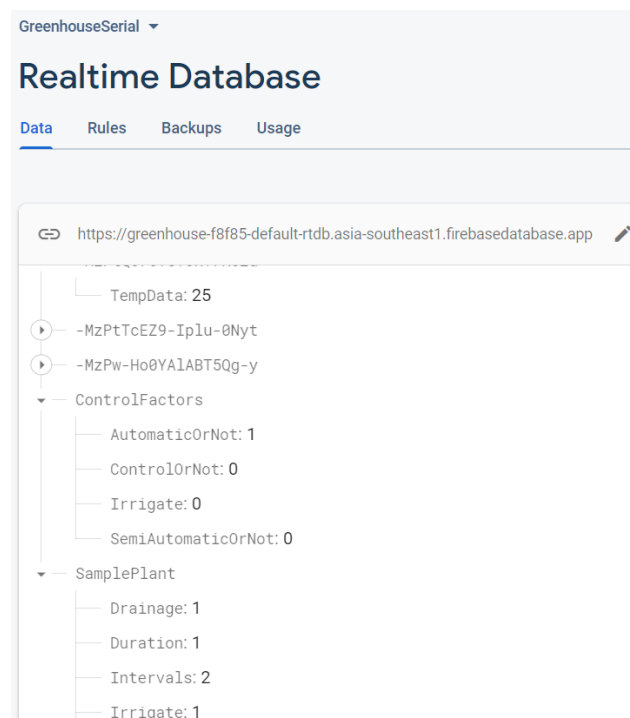
Google's firebase platform allows developers to build mobile & web applications. Founded in 2011, it was initially an independent company. Google acquired the platform in 2014, and now its their flagship web app development platform.



The firebase console (main interface) consists of many applications of which we use the Realtime Database & Storage Bucket.



Below is a snip of the database structure in the realtime database tab.



- **SQLite DB Browser**

The SQLite DB browser is used in our case to open database files (.db) as shown below an example where we observe the table of Sensor Measures (history).

DB Browser for SQLite - C:\Users\zayed\Downloads\FAAIZ\FirebaseConnection2\database2.c

File Edit View Tools Help

New Database Open Database Write Changes Revert Changes Open P

Database Structure Browse Data Edit Pragma Execute SQL

Table: SensorMeasures

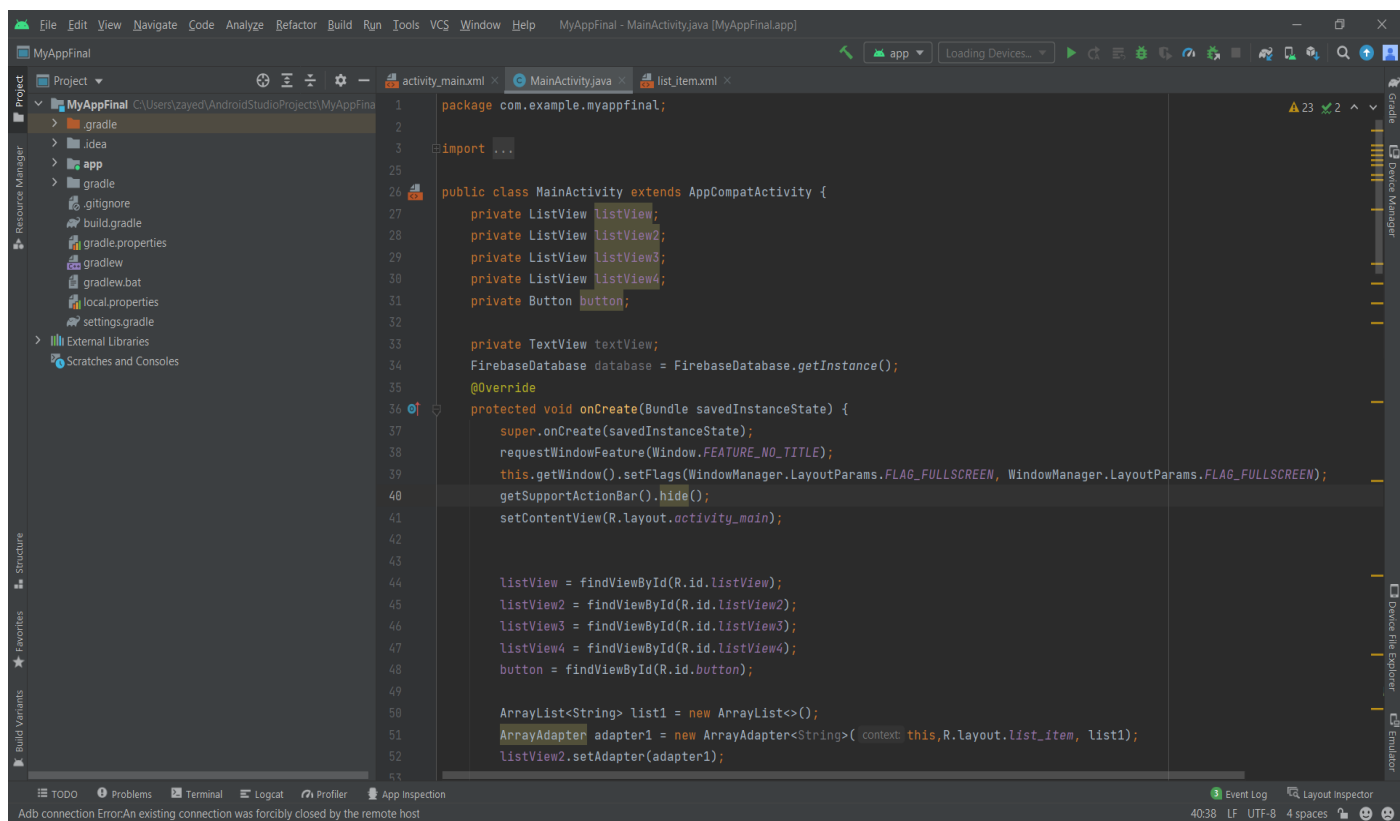
	MSValue	humidity	temperature	alert	datetime
	Filter	Filter	Filter	Filter	Filter
1	1000	51	30	0	22-1-3
2	958	53	24	0	2022-04-14 12:30:00
3	958	53	24	0	2022-04-14 12:30:05
4	958	53	24	0	2022-04-14 12:30:08
5	958	53	24	0	2022-04-14 12:30:11
6	958	53	24	0	2022-04-14 12:30:13
7	959	52	24	0	2022-04-14 12:30:16
8	975	53	24	0	2022-04-14 12:30:18
9	974	53	24	0	2022-04-14 12:30:21
10	975	53	24	0	2022-04-14 12:30:23
11	972	53	24	0	2022-04-14 12:30:26
12	967	53	25	0	2022-04-14 12:30:28
13	974	52	25	0	2022-04-14 12:30:31
14	973	52	24	0	2022-04-14 12:30:33
15	974	52	25	0	2022-04-14 12:30:36
16	967	52	25	0	2022-04-14 12:30:38
17	971	52	24	0	2022-04-14 12:30:41
18	975	52	25	0	2022-04-14 12:30:43
19	973	52	25	0	2022-04-14 12:30:47
20	972	52	24	0	2022-04-14 12:30:52
21	974	52	25	0	2022-04-14 12:30:55
22	973	52	24	0	2022-04-14 12:30:59

Firebase provides with an SDK consisting of the project and the database, this SDK allows us to implement firebase into our Android Studio environment and application.

- **Android Studio**

Android Studio is an official integrated development environment for Google's Android operating system. Built on JetBrains IntelliJ IDEA software, it is designed specifically for Android development. The software is free to download & use.

Android Studio includes a virtual device manager that helps us emulate an android device within the environment, makes application development a lot easier and faster.



Circuit Diagram:

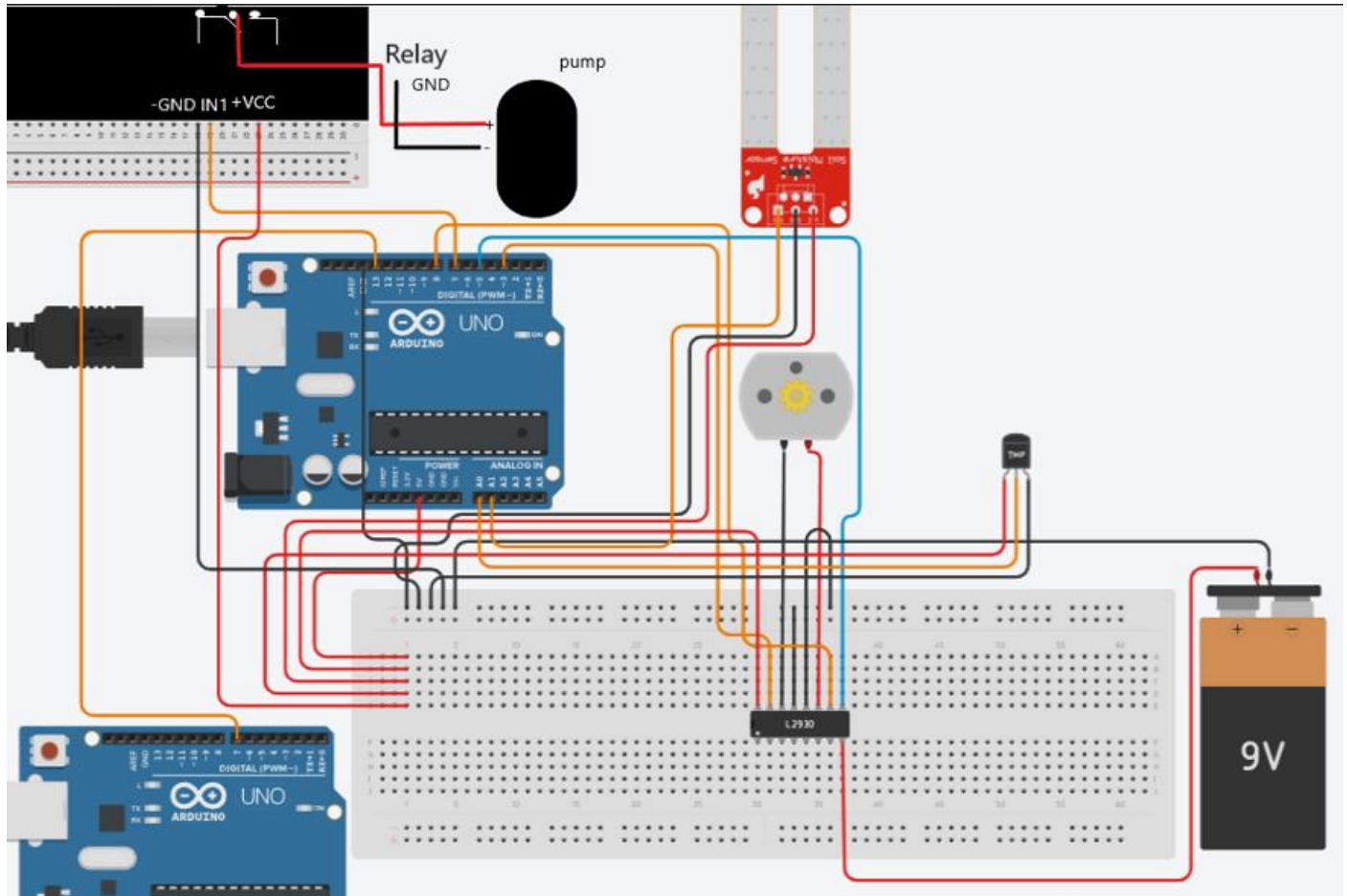


Figure 9. Circuit

Connections Summary:

- Pump +ve to Relay COM [K1] (middle pin)
- Pump -ve to Relay NC [K1]
- DHT11 +ve to 5V
- DHT11 GND to GND
- DHT11 DATA to Arduino1 A0
- DC Motor +ve to L293D OUTPUT3
- DC Motor -ve to L293D OUTPUT4
- L293D ENABLEB (3,4) to Arduino1 D5
- L293D INPUT3 to Arduino1 D8
- L293D INPUT4 to Arduino1 D3
- L293D GND to GND
- L293D POWER1 to 5V
- L293D POWER2 to 9V (if needed)
- Moisture Sensor +VCC to 5V
- Moisture Sensor -ve or GND to GND
- Moisture Sensor DATA or SIGNAL to Arduino1 A1
- Arduino2 D7 to Arduino1 D13

- **Arduino D7 to Relay IN1**
- **Relay VCC to 5V**
- **Relay GND to GND**

Prototype:



Arduino Code for Arduino1 [Arduino IDE]

```

1. #include <dht.h>
2. #define dht_apin A0 // Analog Pin sensor is connected to
3. #define relay 7 // D pin connected to the IN1 of the relay
4. #define pinpump 13 // D pin connected to D7 of the Arduino2
5. #define ENABLE 5 //ENB(3,4) pin of the L293D Motor Driver IC to D5 of the Arduino
6. #define DIRA 3 // Keep this pin LOW & D8 HIGH to spin counterclockwise
7. #define DIRB 8 // Keep this pin HIGH & D3 LOW to spin clockwise
8.
9. int i;
10.
11. dht DHT; // DHT sensor instance
12. int msensor = A1; // Moisture sensor Signal Pin connected to A1 on Arduino1
13. int msvalue = 0;
14. int switchautomatic = 0;
15. int pumpstate =0;
16. int fanswitch =0;
17.
18.
19. void setup(){ // This code runs once
20.   Serial.begin(9600); // Begin Serial Communication with Arduino1 at 9600 bits/second
21.   pinMode(msensor, INPUT); // setting pinmode to input (receives on/off)
22.   pinMode(blueled, OUTPUT); // sets pinmode to output (sends on/off)
23.   pinMode(relay, OUTPUT);

```

```

24. pinMode(pinpump, INPUT);
25. digitalWrite(relay, HIGH);
26. pinMode(ENABLE, OUTPUT);
27. pinMode(DIRA, OUTPUT);
28. pinMode(DIRB, OUTPUT);
29.
30. delay(500); // Delay to let system boot
31. delay(1000); // Wait before accessing Sensor
32.
33. } // end "setup()"
34.
35. void loop(){ // This code runs on continuous loop
36.     digitalWrite(ENABLE, LOW);
37.     digitalWrite(DIRB, LOW); // Initializing Fans\ to be OFF
38.     digitalWrite(DIRA, LOW);
39.     DHT.read11(dht_apin); // Read data from DHT pin
40.     msvalue = analogRead(msensor); // Read analog value from moisture sensor
41.     Serial.print("MS ="); // print-
42.     Serial.print(msvalue); // -Moisture sensor value
43.
44.     pumpstate = digitalRead(pinpump); // Checks if signal is received from Arduino2
45.     If(pumpstate==HIGH){
46.         digitalWrite(relay, LOW); // if signal HIGH, relay HIGH
47.
48.         if (DHT.temperature>40){ // Algorithm to prevent overheating in greenhouse
49.             fanswitch = 1;
50.             digitalWrite(ENABLE, HIGH);
51.
52.             digitalWrite(DIRB, HIGH);
53.             digitalWrite(DIRA, LOW);
54.
55.         }
56.         if (DHT.temperature>40 && fanswitch==1){
57.             digitalWrite(ENABLE, LOW);
58.
59.             digitalWrite(DIRB, HIGH);
60.             digitalWrite(DIRA, LOW);
61.
62.             fanswitch =0;
63.
64.         }
65.         Serial.print("HM ="); // printing relative humidity & temperature
66.         Serial.print(DHT.humidity);
67.         Serial.print("TP =");
68.         Serial.print(DHT.temperature);
69.         Serial.println("Celcius");
70.
71.         delay(2100);}

```

Arduino Code for Arduino2 [Arduino IDE]

We upload the default FirmataExpress code provided in the examples for FirmataExpress library [Refer to Appendices]

```

1. #include <Servo.h>
2. #include <Wire.h>
3. #include <FirmataExpress.h>
4. #include <Ultrasonic.h>
5. #include <Stepper.h>
6. #include <DHTStable.h>
7.
8. #if defined(__AVR__)
9. #include <avr/wdt.h>
10. #endif
11.
12. #define ARDUINO_INSTANCE_ID 1
13.
14. #define I2C_WRITE B00000000
15. #define I2C_READ B00001000
16. #define I2C_READ_CONTINUOUSLY B00010000
17. #define I2C_STOP_READING B00011000
18. #define I2C_READ_WRITE_MODE_MASK B00011000
19. #define I2C_10BIT_ADDRESS_MODE_MASK B00100000

```

```

20. #define I2C_END_TX_MASK          B01000000
21. #define I2C_STOP_TX              1
22. #define I2C_RESTART_TX           0
23. #define I2C_MAX_QUERIES           8
24. #define I2C_REGISTER_NOT_SPECIFIED -1
25.
26. // the minimum interval for sampling analog input
27. #define MINIMUM_SAMPLING_INTERVAL 1
28.
29. #define INTER_PING_INTERVAL 40 // 40 ms.
30.
31. extern void printData(char *id, long data);
32.
33. // SYSEX command sub specifiers
34.
35. #if defined(__AVR__)
36. #define TONE_TONE 0
37. #define TONE_NO_TONE 1
38. #endif
39.
40. #define STEPPER_CONFIGURE 0
41. #define STEPPER_STEP 1
42. #define STEPPER_LIBRARY_VERSION 2
43.
44. // DHT Sensor definitions
45. #define DHT_INTER_PING_INTERVAL 2200 // 2000 ms.
46. #define DHTLIB_OK                0
47.
48.
49. /*=====
50.   GLOBAL VARIABLES
51.   =====*/
52.
53. #ifdef FIRMATA_SERIAL_FEATURE
54. SerialFirmata serialFeature;
55. #endif
56.
57. /* analog inputs */
58. int analogInputsToReport = 0; // bitwise array to store pin reporting
59.
60. /* digital input ports */
61. byte reportPINS[TOTAL_PORTS]; // 1 = report this port, 0 = silence
62. byte previousPINS[TOTAL_PORTS]; // previous 8 bits sent
63.
64. /* pins configuration */
65. byte portConfigInputs[TOTAL_PORTS]; // each bit: 1 = pin in INPUT, 0 = anything else
66.
67. /* timer variables */
68. unsigned long currentMillis; // store the current value from millis()
69. unsigned long previousMillis; // for comparison with currentMillis
70. unsigned int samplingInterval = 19; // how often to run the main loop (in ms)
71. #if defined(__AVR__)
72. unsigned long previousKeepAliveMillis = 0;
73. unsigned int keepAliveInterval = 0;
74. #endif
75.
76. /* i2c data */
77. struct i2c_device_info {
78.     byte addr;
79.     int reg;
80.     byte bytes;
81.     byte stopTX;
82. };
83.
84. /* for i2c read continuous more */
85. i2c_device_info query[I2C_MAX_QUERIES];
86.
87. byte i2cRxData[64];
88. boolean isI2CEnabled = false;
89. signed char queryIndex = -1;
90. // default delay time between i2c read request and Wire.requestFrom()
91. unsigned int i2cReadDelayTime = 0;
92.
93. Servo servos[MAX_SERVOS];
94. byte servoPinMap[TOTAL_PINS];
95. byte detachedServos[MAX_SERVOS];

```

```

96. byte detachedServoCount = 0;
97. byte servoCount = 0;
98.
99. boolean isResetting = false;
100.
101. // Forward declare a few functions to avoid compiler errors with older versions
102. // of the Arduino IDE.
103. void setPinModeCallback(byte, int);
104.
105. void reportAnalogCallback(byte analogPin, int value);
106.
107. void sysexCallback(byte, byte, byte *);
108.
109. /* utility functions */
110. void wireWrite(byte data) {
111.   #if ARDUINO >= 100
112.     Wire.write((byte)data);
113.   #else
114.     Wire.send(data);
115.   #endif
116. }
117.
118. byte wireRead(void) {
119.   #if ARDUINO >= 100
120.     return Wire.read();
121.   #else
122.     return Wire.receive();
123.   #endif
124. }
125.
126. // Ping variables
127. int numLoops = 0;
128. int pingLoopCounter = 0;
129.
130. int numActiveSonars = 0; // number of sonars attached
131. uint8_t sonarPinNumbers[MAX_SONARS];
132. int nextSonar = 0; // index into sonars[] for next device
133.
134. // array to hold up to 6 instances of sonar devices
135. Ultrasonic *sonars[MAX_SONARS];
136.
137. uint8_t sonarTriggerPin;
138. uint8_t sonarEchoPin;
139. uint8_t currentSonar = 0; // Keeps track of which sensor is active.
140.
141. uint8_t pingInterval = 33; // Milliseconds between sensor pings (29ms is about the min to avoid
142. // cross- sensor echo).
143. byte sonarMSB, sonarLSB;
144.
145.
146. // Stepper Motor
147. Stepper *stepper = NULL;
148.
149. // DHT sensors
150. int numActiveDHTs = 0; // number of DHTs attached
151. uint8_t DHT_PinNumbers[MAX_DHTS];
152. uint8_t DHT_WakeUpDelay[MAX_DHTS];
153. uint8_t DHT_TYPE[MAX_DHTS];
154.
155. DHTStable DHT; // instance of dhtstable
156.
157. uint8_t nextDHT = 0; // index into dht[] for next device
158. uint8_t currentDHT = 0; // Keeps track of which sensor is active.
159.
160. int dhtNumLoops = 0;
161. int dhtLoopCounter = 0;
162.
163. uint8_t dht_value[4]; // buffer to receive data
164.
165. /*=====
166.   FUNCTIONS
167.   =====*/
168.
169. void attachServo(byte pin, int minPulse, int maxPulse) {
170.   if (servoCount < MAX_SERVOS) {
171.     // reuse indexes of detached servos until all have been reallocated

```



```

172.         if (detachedServoCount > 0) {
173.             servoPinMap[pin] = detachedServos[detachedServoCount - 1];
174.             if (detachedServoCount > 0) detachedServoCount--;
175.         } else {
176.             servoPinMap[pin] = servoCount;
177.             servoCount++;
178.         }
179.         if (minPulse > 0 && maxPulse > 0) {
180.             servos[servoPinMap[pin]].attach(PIN_TO_DIGITAL(pin), minPulse, maxPulse);
181.         } else {
182.             servos[servoPinMap[pin]].attach(PIN_TO_DIGITAL(pin));
183.         }
184.     } else {
185.         Firmata.sendString("Max servos attached");
186.     }
187. }
188.
189. void detachServo(byte pin) {
190.     servos[servoPinMap[pin]].detach();
191.     // if we're detaching the last servo, decrement the count
192.     // otherwise store the index of the detached servo
193.     if (servoPinMap[pin] == servoCount && servoCount > 0) {
194.         servoCount--;
195.     } else if (servoCount > 0) {
196.         // keep track of detached servos because we want to reuse their indexes
197.         // before incrementing the count of attached servos
198.         detachedServoCount++;
199.         detachedServos[detachedServoCount - 1] = servoPinMap[pin];
200.     }
201.
202.     servoPinMap[pin] = 255;
203. }
204.
205. void enableI2CPins() {
206.     byte i;
207.     // is there a faster way to do this? would probaby require importing
208.     // Arduino.h to get SCL and SDA pins
209.     for (i = 0; i < TOTAL_PINS; i++) {
210.         if (IS_PIN_I2C(i)) {
211.             // mark pins as i2c so they are ignore in non i2c data requests
212.             setPinModeCallback(i, PIN_MODE_I2C);
213.         }
214.     }
215.
216.     isI2CEnabled = true;
217.
218.     Wire.begin();
219. }
220.
221. /* disable the i2c pins so they can be used for other functions */
222. void disableI2CPins() {
223.     isI2CEnabled = false;
224.     // disable read continuous mode for all devices
225.     queryIndex = -1;
226. }
227.
228. void readAndReportData(byte address, int theRegister, byte numBytes, byte stopTX) {
229.     // allow I2C requests that don't require a register read
230.     // for example, some devices using an interrupt pin to signify new data available
231.     // do not always require the register read so upon interrupt you call Wire.requestFrom()
232.     if (theRegister != I2C_REGISTER_NOT_SPECIFIED) {
233.         Wire.beginTransaction(address);
234.         wireWrite((byte) theRegister);
235.         Wire.endTransmission(stopTX); // default = true
236.         // do not set a value of 0
237.         if (i2cReadDelayTime > 0) {
238.             // delay is necessary for some devices such as WiiNunchuck
239.             delayMicroseconds(i2cReadDelayTime);
240.         }
241.     } else {
242.         theRegister = 0; // fill the register with a dummy value
243.     }
244.
245.     Wire.requestFrom(address, numBytes); // all bytes are returned in requestFrom
246.
247.     // check to be sure correct number of bytes were returned by slave

```

```

248.     if (numBytes < Wire.available()) {
249.         Firmata.sendString("I2C: Too many bytes received");
250.     } else if (numBytes > Wire.available()) {
251.         Firmata.sendString("I2C: Too few bytes received");
252.     }
253.
254.     i2cRxData[0] = address;
255.     i2cRxData[1] = theRegister;
256.
257.     for (int i = 0; i < numBytes && Wire.available(); i++) {
258.         i2cRxData[2 + i] = wireRead();
259.     }
260.
261.     // send slave address, register and received bytes
262.     Firmata.sendSysex(SYSEX_I2C_REPLY, numBytes + 2, i2cRxData);
263. }
264.
265. void outputPort(byte portNumber, byte portValue, byte forceSend) {
266.     // pins not configured as INPUT are cleared to zeros
267.     portValue = portValue & portConfigInputs[portNumber];
268.     // only send if the value is different than previously sent
269.     if (forceSend || previousPINS[portNumber] != portValue) {
270.         Firmata.sendDigitalPort(portNumber, portValue);
271.         previousPINS[portNumber] = portValue;
272.     }
273. }
274.
275. /* -----
276.    check all the active digital inputs for change of state, then add any events
277.    to the Serial output queue using Serial.print() */
278. void checkDigitalInputs(void) {
279.     /* Using non-looping code allows constants to be given to readPort().
280.        The compiler will apply substantial optimizations if the inputs
281.        to readPort() are compile-time constants. */
282.     if (TOTAL_PORTS > 0 && reportPINS[0]) outputPort(0, readPort(0, portConfigInputs[0]), false);
283.     if (TOTAL_PORTS > 1 && reportPINS[1]) outputPort(1, readPort(1, portConfigInputs[1]), false);
284.     if (TOTAL_PORTS > 2 && reportPINS[2]) outputPort(2, readPort(2, portConfigInputs[2]), false);
285.     if (TOTAL_PORTS > 3 && reportPINS[3]) outputPort(3, readPort(3, portConfigInputs[3]), false);
286.     if (TOTAL_PORTS > 4 && reportPINS[4]) outputPort(4, readPort(4, portConfigInputs[4]), false);
287.     if (TOTAL_PORTS > 5 && reportPINS[5]) outputPort(5, readPort(5, portConfigInputs[5]), false);
288.     if (TOTAL_PORTS > 6 && reportPINS[6]) outputPort(6, readPort(6, portConfigInputs[6]), false);
289.     if (TOTAL_PORTS > 7 && reportPINS[7]) outputPort(7, readPort(7, portConfigInputs[7]), false);
290.     if (TOTAL_PORTS > 8 && reportPINS[8]) outputPort(8, readPort(8, portConfigInputs[8]), false);
291.     if (TOTAL_PORTS > 9 && reportPINS[9]) outputPort(9, readPort(9, portConfigInputs[9]), false);
292.     if (TOTAL_PORTS > 10 && reportPINS[10]) outputPort(10, readPort(10, portConfigInputs[10]), false);
293.     if (TOTAL_PORTS > 11 && reportPINS[11]) outputPort(11, readPort(11, portConfigInputs[11]), false);
294.     if (TOTAL_PORTS > 12 && reportPINS[12]) outputPort(12, readPort(12, portConfigInputs[12]), false);
295.     if (TOTAL_PORTS > 13 && reportPINS[13]) outputPort(13, readPort(13, portConfigInputs[13]), false);
296.     if (TOTAL_PORTS > 14 && reportPINS[14]) outputPort(14, readPort(14, portConfigInputs[14]), false);
297.     if (TOTAL_PORTS > 15 && reportPINS[15]) outputPort(15, readPort(15, portConfigInputs[15]), false);
298. }
299.
300. // -----
301. /* sets the pin mode to the correct state and sets the relevant bits in the
302.    two bit-arrays that track Digital I/O and PWM status
303.    */
304. void setPinModeCallback(byte pin, int mode) {
305.
306.     if (Firmata.getPinMode(pin) == PIN_MODE_IGNORE)
307.         return;
308.
309.     if (Firmata.getPinMode(pin) == PIN_MODE_I2C && isI2CEnabled && mode != PIN_MODE_I2C) {
310.         // disable i2c so pins can be used for other functions
311.         // the following if statements should reconfigure the pins properly
312.         disableI2CPins();
313.     }
314.     if (IS_PIN_DIGITAL(pin) && mode != PIN_MODE_SERVO) {
315.         if (servoPinMap[pin] < MAX_SERVOS && servos[servoPinMap[pin]].attached()) {
316.             detachServo(pin);
317.         }
318.     }
319.     if (IS_PIN_ANALOG(pin)) {
320.         reportAnalogCallback(PIN_TO_ANALOG(pin), mode == PIN_MODE_ANALOG ? 1 : 0); // turn on/off
321.         reporting
322.     }
323.     if (IS_PIN_DIGITAL(pin)) {

```



```

323.         if (mode == INPUT || mode == PIN_MODE_PULLUP) {
324.             portConfigInputs[pin / 8] |= (1 << (pin & 7));
325.         } else {
326.             portConfigInputs[pin / 8] &= ~(1 << (pin & 7));
327.         }
328.     }
329.     Firmata.setPinState(pin, 0);
330.     switch (mode) {
331.         case PIN_MODE_ANALOG:
332.             if (IS_PIN_ANALOG(pin)) {
333.                 if (IS_PIN_DIGITAL(pin)) {
334.                     pinMode(PIN_TO_DIGITAL(pin), INPUT);    // disable output driver
335. #if ARDUINO <= 100
336.                     // deprecated since Arduino 1.0.1 - TODO: drop support in Firmata 2.6
337.                     digitalWrite(PIN_TO_DIGITAL(pin), LOW); // disable internal pull-ups
338. #endif
339.                 }
340.                 Firmata.setPinMode(pin, PIN_MODE_ANALOG);
341.             }
342.             break;
343.         case INPUT:
344.             if (IS_PIN_DIGITAL(pin)) {
345.                 pinMode(PIN_TO_DIGITAL(pin), INPUT);    // disable output driver
346. #if ARDUINO <= 100
347.                 // deprecated since Arduino 1.0.1 - TODO: drop support in Firmata 2.6
348.                 digitalWrite(PIN_TO_DIGITAL(pin), LOW); // disable internal pull-ups
349. #endif
350.                 Firmata.setPinMode(pin, INPUT);
351.             }
352.             break;
353.         case PIN_MODE_PULLUP:
354.             if (IS_PIN_DIGITAL(pin)) {
355.                 pinMode(PIN_TO_DIGITAL(pin), INPUT_PULLUP);
356.                 Firmata.setPinMode(pin, PIN_MODE_PULLUP);
357.                 Firmata.setPinState(pin, 1);
358.             }
359.             break;
360.         case OUTPUT:
361.             if (IS_PIN_DIGITAL(pin)) {
362.                 if (Firmata.getPinMode(pin) == PIN_MODE_PWM) {
363.                     // Disable PWM if pin mode was previously set to PWM.
364.                     digitalWrite(PIN_TO_DIGITAL(pin), LOW);
365.                 }
366.                 pinMode(PIN_TO_DIGITAL(pin), OUTPUT);
367.                 Firmata.setPinMode(pin, OUTPUT);
368.             }
369.             break;
370.         case PIN_MODE_PWM:
371.             if (IS_PIN_PWM(pin)) {
372.                 pinMode(PIN_TO_PWM(pin), OUTPUT);
373.                 analogWrite(PIN_TO_PWM(pin), 0);
374.                 Firmata.setPinMode(pin, PIN_MODE_PWM);
375.             }
376.             break;
377.         case PIN_MODE_SERVO:
378.             if (IS_PIN_DIGITAL(pin)) {
379.                 Firmata.setPinMode(pin, PIN_MODE_SERVO);
380.                 if (servoPinMap[pin] == 255 || !servos[servoPinMap[pin]].attached()) {
381.                     // pass -1 for min and max pulse values to use default values set
382.                     // by Servo library
383.                     attachServo(pin, -1, -1);
384.                 }
385.             }
386.             break;
387.         case PIN_MODE_I2C:
388.             if (IS_PIN_I2C(pin)) {
389.                 // mark the pin as i2c
390.                 // the user must call I2C_CONFIG to enable I2C for a device
391.                 Firmata.setPinMode(pin, PIN_MODE_I2C);
392.             }
393.             break;
394.         case PIN_MODE_SERIAL:
395. #ifdef FIRMATA_SERIAL_FEATURE
396.             serialFeature.handlePinMode(pin, PIN_MODE_SERIAL);
397. #endif
398.             break;

```

```

399.  #if defined(__AVR__)
400.      case PIN_MODE_TONE:
401.          Firmata.setPinMode(pin, PIN_MODE_TONE);
402.          break ;
403.  #endif
404.      case PIN_MODE_SONAR:
405.          Firmata.setPinMode(pin, PIN_MODE_SONAR);
406.          break;
407.      case PIN_MODE_DHT:
408.          Firmata.setPinMode(pin, PIN_MODE_DHT);
409.          break;
410.      case PIN_MODE_STEPPER:
411.          Firmata.setPinMode(pin, PIN_MODE_STEPPER);
412.          break;
413.      default:
414.          Firmata.sendString("Unknown pin mode"); // TODO: put error msgs in EEPROM
415.          break;
416.  }
417.  // TODO: save status to EEPROM here, if changed
418.  }
419.
420.  /*
421.   * Sets the value of an individual pin. Useful if you want to set a pin value but
422.   * are not tracking the digital port state.
423.   * Can only be used on pins configured as OUTPUT.
424.   * Cannot be used to enable pull-ups on Digital INPUT pins.
425.   */
426.  void setPinValueCallback(byte pin, int value) {
427.      if (pin < TOTAL_PINS && IS_PIN_DIGITAL(pin)) {
428.          if (Firmata.getPinMode(pin) == OUTPUT) {
429.              Firmata.setPinState(pin, value);
430.              digitalWrite(PIN_TO_DIGITAL(pin), value);
431.          }
432.      }
433.  }
434.
435.  void analogWriteCallback(byte pin, int value) {
436.      if (pin < TOTAL_PINS) {
437.          switch (Firmata.getPinMode(pin)) {
438.              case PIN_MODE_SERVO:
439.                  if (IS_PIN_DIGITAL(pin))
440.                      servos[servoPinMap[pin]].write(value);
441.                  Firmata.setPinState(pin, value);
442.                  break;
443.              case PIN_MODE_PWM:
444.                  if (IS_PIN_PWM(pin))
445.                      analogWrite(PIN_TO_PWM(pin), value);
446.                  Firmata.setPinState(pin, value);
447.                  break;
448.          }
449.      }
450.  }
451.
452.  void digitalWriteCallback(byte port, int value) {
453.      byte pin, lastPin, pinValue, mask = 1, pinWriteMask = 0;
454.
455.      if (port < TOTAL_PORTS) {
456.          // create a mask of the pins on this port that are writable.
457.          lastPin = port * 8 + 8;
458.          if (lastPin > TOTAL_PINS) lastPin = TOTAL_PINS;
459.          for (pin = port * 8; pin < lastPin; pin++) {
460.              // do not disturb non-digital pins (eg, Rx & Tx)
461.              if (IS_PIN_DIGITAL(pin)) {
462.                  // do not touch pins in PWM, ANALOG, SERVO or other modes
463.                  if (Firmata.getPinMode(pin) == OUTPUT || Firmata.getPinMode(pin) == INPUT) {
464.                      pinValue = ((byte) value & mask) ? 1 : 0;
465.                      if (Firmata.getPinMode(pin) == OUTPUT) {
466.                          pinWriteMask |= mask;
467.                      } else if (Firmata.getPinMode(pin) == INPUT && pinValue == 1 &&
468.                          Firmata.getPinState(pin) != 1) {
469.                          // only handle INPUT here for backwards compatibility
470.                          pinMode(pin, INPUT_PULLUP);
471.                      }
472.                      // only write to the INPUT pin to enable pullups if Arduino v1.0.0 or earlier
473.                      pinWriteMask |= mask;

```

```

474. #endif
475.     }
476.     Firmata.setPinState(pin, pinValue);
477.     }
478.     }
479.     mask = mask << 1;
480. }
481. writePort(port, (byte) value, pinWriteMask);
482. }
483. }
484.
485.
486. // -----
487. /* sets bits in a bit array (int) to toggle the reporting of the analogIns
488. */
489. //void FirmataClass::setAnalogPinReporting(byte pin, byte state) {
490. //}
491. void reportAnalogCallback(byte analogPin, int value) {
492.     if (analogPin < TOTAL_ANALOG_PINS) {
493.         if (value == 0) {
494.             analogInputsToReport = analogInputsToReport & ~(1 << analogPin);
495.         } else {
496.             analogInputsToReport = analogInputsToReport | (1 << analogPin);
497.             // prevent during system reset or all analog pin values will be reported
498.             // which may report noise for unconnected analog pins
499.             if (!isResetting) {
500.                 // Send pin value immediately. This is helpful when connected via
501.                 // ethernet, wi-fi or bluetooth so pin states can be known upon
502.                 // reconnecting.
503.                 Firmata.sendAnalog(analogPin, analogRead(analogPin));
504.             }
505.         }
506.     }
507.     // TODO: save status to EEPROM here, if changed
508. }
509.
510. void reportDigitalCallback(byte port, int value) {
511.     if (port < TOTAL_PORTS) {
512.         reportPINS[port] = (byte) value;
513.         // Send port value immediately. This is helpful when connected via
514.         // ethernet, wi-fi or bluetooth so pin states can be known upon
515.         // reconnecting.
516.         if (value) outputPort(port, readPort(port, portConfigInputs[port]), true);
517.     }
518.     // do not disable analog reporting on these 8 pins, to allow some
519.     // pins used for digital, others analog. Instead, allow both types
520.     // of reporting to be enabled, but check if the pin is configured
521.     // as analog when sampling the analog inputs. Likewise, while
522.     // scanning digital pins, portConfigInputs will mask off values from any
523.     // pins configured as analog
524. }
525.
526. /*=====
527. SYSEX-BASED commands
528. =====*/
529.
530. void sysexCallback(byte command, byte argc, byte *argv) {
531.     byte mode;
532.     byte stopTX;
533.     byte slaveAddress;
534.     byte data;
535.     int slaveRegister;
536.     unsigned int delayTime;
537.     byte pin;
538.     int frequency;
539.     int duration;
540.
541.     switch (command) {
542.
543.         case RU_THERE:
544.             Firmata.write(START_SYSEX);
545.             Firmata.write((byte) I_AM_HERE);
546.             Firmata.write((byte) ARDUINO_INSTANCE_ID);
547.             Firmata.write(END_SYSEX);
548.             break;
549.

```

```

550.     case I2C_REQUEST:
551.         mode = argv[1] & I2C_READ_WRITE_MODE_MASK;
552.         if (argv[1] & I2C_10BIT_ADDRESS_MODE_MASK) {
553.             Firmata.sendString("10-bit addressing not supported");
554.             return;
555.         } else {
556.             slaveAddress = argv[0];
557.         }
558.
559.         // need to invert the logic here since 0 will be default for client
560.         // libraries that have not updated to add support for restart tx
561.         if (argv[1] & I2C_END_TX_MASK) {
562.             stopTX = I2C_RESTART_TX;
563.         } else {
564.             stopTX = I2C_STOP_TX; // default
565.         }
566.
567.         switch (mode) {
568.             case I2C_WRITE:
569.                 Wire.beginTransmission(slaveAddress);
570.                 for (byte i = 2; i < argc; i += 2) {
571.                     data = argv[i] + (argv[i + 1] << 7);
572.                     wireWrite(data);
573.                 }
574.                 Wire.endTransmission();
575.                 delayMicroseconds(70);
576.                 break;
577.             case I2C_READ:
578.                 if (argc == 6) {
579.                     // a slave register is specified
580.                     slaveRegister = argv[2] + (argv[3] << 7);
581.                     data = argv[4] + (argv[5] << 7); // bytes to read
582.                 } else {
583.                     // a slave register is NOT specified
584.                     slaveRegister = I2C_REGISTER_NOT_SPECIFIED;
585.                     data = argv[2] + (argv[3] << 7); // bytes to read
586.                 }
587.                 readAndReportData(slaveAddress, (int) slaveRegister, data, stopTX);
588.                 break;
589.             case I2C_READ_CONTINUOUSLY:
590.                 if ((queryIndex + 1) >= I2C_MAX_QUERIES) {
591.                     // too many queries, just ignore
592.                     Firmata.sendString("too many queries");
593.                     break;
594.                 }
595.                 if (argc == 6) {
596.                     // a slave register is specified
597.                     slaveRegister = argv[2] + (argv[3] << 7);
598.                     data = argv[4] + (argv[5] << 7); // bytes to read
599.                 } else {
600.                     // a slave register is NOT specified
601.                     slaveRegister = (int) I2C_REGISTER_NOT_SPECIFIED;
602.                     data = argv[2] + (argv[3] << 7); // bytes to read
603.                 }
604.                 queryIndex++;
605.                 query[queryIndex].addr = slaveAddress;
606.                 query[queryIndex].reg = slaveRegister;
607.                 query[queryIndex].bytes = data;
608.                 query[queryIndex].stopTX = stopTX;
609.                 break;
610.             case I2C_STOP_READING:
611.                 byte queryIndexToSkip;
612.                 // if read continuous mode is enabled for only 1 i2c device, disable
613.                 // read continuous reporting for that device
614.                 if (queryIndex <= 0) {
615.                     queryIndex = -1;
616.                 } else {
617.                     queryIndexToSkip = 0;
618.                     // if read continuous mode is enabled for multiple devices,
619.                     // determine which device to stop reading and remove it's data from
620.                     // the array, shifting other array data to fill the space
621.                     for (byte i = 0; i < queryIndex + 1; i++) {
622.                         if (query[i].addr == slaveAddress) {
623.                             queryIndexToSkip = i;
624.                             break;
625.                         }

```

```

626.         }
627.
628.         for (byte i = queryIndexToSkip; i < queryIndex + 1; i++) {
629.             if (i < I2C_MAX_QUERIES) {
630.                 query[i].addr = query[i + 1].addr;
631.                 query[i].reg = query[i + 1].reg;
632.                 query[i].bytes = query[i + 1].bytes;
633.                 query[i].stopTX = query[i + 1].stopTX;
634.             }
635.         }
636.         queryIndex--;
637.     }
638.     break;
639. default:
640.     break;
641. }
642. break;
643. case I2C_CONFIG:
644.     delayTime = (argv[0] + (argv[1] << 7));
645.
646.     if (argc > 1 && delayTime > 0) {
647.         i2cReadDelayTime = delayTime;
648.     }
649.
650.     if (!isI2CEnabled) {
651.         enableI2CPins();
652.     }
653.
654.     break;
655. case SERVO_CONFIG:
656.     if (argc > 4) {
657.         // these vars are here for clarity, they'll optimized away by the compiler
658.         byte pin = argv[0];
659.         int minPulse = argv[1] + (argv[2] << 7);
660.         int maxPulse = argv[3] + (argv[4] << 7);
661.
662.         if (IS_PIN_DIGITAL(pin)) {
663.             if (servoPinMap[pin] < MAX_SERVOS && servos[servoPinMap[pin]].attached()) {
664.                 detachServo(pin);
665.             }
666.             attachServo(pin, minPulse, maxPulse);
667.             setPinModeCallback(pin, PIN_MODE_SERVO);
668.         }
669.     }
670.     break;
671. #if defined(__AVR__)
672.     case KEEP_ALIVE:
673.         keepAliveInterval = argv[0] + (argv[1] << 7);
674.         previousKeepAliveMillis = millis();
675.         break;
676. #endif
677. case SAMPLING_INTERVAL:
678.     if (argc > 1) {
679.         samplingInterval = argv[0] + (argv[1] << 7);
680.         if (samplingInterval < MINIMUM_SAMPLING_INTERVAL) {
681.             samplingInterval = MINIMUM_SAMPLING_INTERVAL;
682.         }
683.         /* calculate number of loops per ping */
684.         numLoops = INTER_PING_INTERVAL / samplingInterval;
685.         /* calculate number of loops between each sample of DHT data */
686.         dhtNumLoops = DHT_INTER_PING_INTERVAL / samplingInterval;
687.     } else {
688.         //Firmata.sendString("Not enough data");
689.     }
690.     break;
691. case EXTENDED_ANALOG:
692.     if (argc > 1) {
693.         int val = argv[1];
694.         if (argc > 2) val |= (argv[2] << 7);
695.         if (argc > 3) val |= (argv[3] << 14);
696.         analogWriteCallback(argv[0], val);
697.     }
698.     break;
699. case CAPABILITY_QUERY:
700.     Firmata.write(START_SYSEX);
701.     Firmata.write(CAPABILITY_RESPONSE);

```

```

702.         for (byte pin = 0; pin < TOTAL_PINS; pin++) {
703.             if (IS_PIN_DIGITAL(pin)) {
704.                 Firmata.write((byte) INPUT);
705.                 Firmata.write(1);
706.                 Firmata.write((byte) PIN_MODE_PULLUP);
707.                 Firmata.write(1);
708.                 Firmata.write((byte) OUTPUT);
709.                 Firmata.write(1);
710.                 Firmata.write((byte) PIN_MODE_STEPPER);
711.                 Firmata.write(1);
712.                 Firmata.write((byte) PIN_MODE_SONAR);
713.                 Firmata.write(1);
714.                 Firmata.write((byte) PIN_MODE_DHT);
715.                 Firmata.write(1);
716.
717.             #if defined(__AVR__)
718.                 Firmata.write((byte) PIN_MODE_TONE);
719.                 Firmata.write(1);
720.             #endif
721.             }
722.             if (IS_PIN_ANALOG(pin)) {
723.                 Firmata.write(PIN_MODE_ANALOG);
724.                 Firmata.write(10); // 10 = 10-bit resolution
725.             }
726.             if (IS_PIN_PWM(pin)) {
727.                 Firmata.write(PIN_MODE_PWM);
728.                 Firmata.write(DEFAULT_PWM_RESOLUTION);
729.             }
730.             if (IS_PIN_DIGITAL(pin)) {
731.                 Firmata.write(PIN_MODE_SERVO);
732.                 Firmata.write(14);
733.             }
734.             if (IS_PIN_I2C(pin)) {
735.                 Firmata.write(PIN_MODE_I2C);
736.                 Firmata.write(1); // TODO: could assign a number to map to SCL or SDA
737.             }
738.             #ifndef FIRMATA_SERIAL_FEATURE
739.                 serialFeature.handleCapability(pin);
740.             #endif
741.             Firmata.write(127);
742.         }
743.         Firmata.write(END_SYSEX);
744.         break;
745.     case PIN_STATE_QUERY:
746.         if (argc > 0) {
747.             byte pin = argv[0];
748.             Firmata.write(START_SYSEX);
749.             Firmata.write(PIN_STATE_RESPONSE);
750.             Firmata.write(pin);
751.             if (pin < TOTAL_PINS) {
752.                 Firmata.write(Firmata.getPinMode(pin));
753.                 Firmata.write((byte) Firmata.getPinState(pin) & 0x7F);
754.                 if (Firmata.getPinState(pin) & 0xFF80)
755.                     Firmata.write((byte) (Firmata.getPinState(pin) >> 7) & 0x7F);
756.                 if (Firmata.getPinState(pin) & 0xC000)
757.                     Firmata.write((byte) (Firmata.getPinState(pin) >> 14) & 0x7F);
758.             }
759.             Firmata.write(END_SYSEX);
760.         }
761.         break;
762.     case ANALOG_MAPPING_QUERY:
763.         Firmata.write(START_SYSEX);
764.         Firmata.write(ANALOG_MAPPING_RESPONSE);
765.         for (byte pin = 0; pin < TOTAL_PINS; pin++) {
766.             Firmata.write(IS_PIN_ANALOG(pin) ? PIN_TO_ANALOG(pin) : 127);
767.         }
768.         Firmata.write(END_SYSEX);
769.         break;
770.     case SERIAL_MESSAGE:
771.         #ifndef FIRMATA_SERIAL_FEATURE
772.             serialFeature.handleSysex(command, argc, argv);
773.         #endif
774.         break;
775.     #if defined(__AVR__)

```

```

776.         case TONE_DATA:
777.             byte toneCommand, pin;
778.             int frequency, duration;
779.
780.             toneCommand = argv[0];
781.             pin = argv[1];
782.
783.             if (toneCommand == TONE_TONE) {
784.                 frequency = argv[2] + (argv[3] << 7);
785.                 // duration is currently limited to 16,383 ms
786.                 duration = argv[4] + (argv[5] << 7);
787.                 tone(pin, frequency, duration);
788.             }
789.             else if (toneCommand == TONE_NO_TONE) {
790.                 noTone(pin);
791.             }
792.             break ;
793.     #endif
794.         // arg0 = trigger pin
795.         // arg1 = echo pin
796.         // arg2 = timeout_lsb
797.         // arg3 = timeout_msb
798.         case SONAR_CONFIG :
799.             unsigned long timeout;
800.             if (numActiveSonars < MAX_SONARS) {
801.                 sonarTriggerPin = argv[0];
802.                 sonarEchoPin = argv[1];
803.
804.                 timeout = argv[2] + (argv[3] << 7);
805.                 sonarPinNumbers[numActiveSonars] = sonarTriggerPin;
806.
807.                 setPinModeCallback(sonarTriggerPin, PIN_MODE_SONAR);
808.                 setPinModeCallback(sonarEchoPin, PIN_MODE_SONAR);
809.                 sonars[numActiveSonars] = new Ultrasonic(sonarTriggerPin, sonarEchoPin, timeout);
810.
811.                 numActiveSonars++;
812.             } else {
813.                 Firmata.sendString("PING_CONFIG Error: Exceeded number of supported ping devices");
814.             }
815.             break;
816.
817.         case STEPPER_DATA:
818.             // determine if this a STEPPER_CONFIGURE command or STEPPER_OPERATE command
819.             if (argv[0] == STEPPER_CONFIGURE) {
820.                 int numSteps = argv[1] + (argv[2] << 7);
821.                 int pin1 = argv[3];
822.                 int pin2 = argv[4];
823.                 if (argc == 5) {
824.                     // two pin motor
825.                     stepper = new Stepper(numSteps, pin1, pin2);
826.                 } else if (argc == 7) // 4 wire motor
827.                 {
828.                     int pin3 = argv[5];
829.                     int pin4 = argv[6];
830.                     stepper = new Stepper(numSteps, pin1, pin2, pin3, pin4);
831.                 } else {
832.                     Firmata.sendString("STEPPER CONFIG Error: Wrong Number of arguments");
833.                     printData((char *) "argc = ", argc);
834.                 }
835.             } else if (argv[0] == STEPPER_STEP) {
836.                 long speed = (long) argv[1] | ((long) argv[2] << 7) | ((long) argv[3] << 14);
837.                 int numSteps = argv[4] + (argv[5] << 7);
838.                 int direction = argv[6];
839.                 if (stepper != NULL) {
840.                     stepper->setSpeed(speed);
841.                     if (direction == 0) {
842.                         numSteps *= -1;
843.                     }
844.                     stepper->step(numSteps);
845.                 } else {
846.                     Firmata.sendString("STEPPER OPERATE Error: MOTOR NOT CONFIGURED");
847.                 }
848.             } else if (argv[0] == STEPPER_LIBRARY_VERSION) {
849.                 if (stepper != NULL) {
850.                     int version = stepper->version();
851.                     Firmata.write(START_SYSEX);

```

```

852.         Firmata.write(STEPPEER_DATA);
853.         Firmata.write(version & 0x7F);
854.         Firmata.write(version >> 7);
855.         Firmata.write(END_SYSEX);
856.     } else {
857.         // did not find a configured stepper
858.         Firmata.sendString("STEPPER FIRMWARE VERSION Error: NO MOTORS CONFIGURED");
859.     }
860.     break;
861. } else {
862.     Firmata.sendString("STEPPER CONFIG Error: UNKNOWN STEPPER COMMAND");
863. }
864. break;
865. case DHT_CONFIG:
866.     int DHT_Pin = argv[0];
867.     int DHT_type = argv[1];
868.
869.     if (numActiveDHTs < MAX_DHTS) {
870.         if (DHT_type != 22 && DHT_type != 11) {
871.             Firmata.sendString("ERROR: UNKNOWN SENSOR TYPE, VALID SENSORS ARE 11, 22");
872.             break;
873.         } else {
874.             // test the sensor
875.             DHT_PinNumbers[numActiveDHTs] = DHT_Pin;
876.             DHT_TYPE[numActiveDHTs] = DHT_type;
877.
878.             setPinModeCallback(DHT_Pin, PIN_MODE_DHT);
879.             numActiveDHTs++;
880.             dhtNumLoops = dhtNumLoops / numActiveDHTs;
881.             break;
882.         }
883.     } else {
884.         Firmata.sendString("DHT_CONFIG Error: Exceeded number of supported DHT devices");
885.         break;
886.     }
887. }
888. }
889.
890. /*=====
891.     SETUP()
892.     =====*/
893.
894. void systemResetCallback() {
895.     isResetting = true;
896.
897.     // initialize a default state
898.     // TODO: option to load config from EEPROM instead of default
899.
900. #ifdef FIRMATA_SERIAL_FEATURE
901.     serialFeature.reset();
902. #endif
903.
904.     if (isI2CEnabled) {
905.         disableI2CPins();
906.     }
907.
908.     for (byte i = 0; i < TOTAL_PORTS; i++) {
909.         reportPINS[i] = false;    // by default, reporting off
910.         portConfigInputs[i] = 0;  // until activated
911.         previousPINS[i] = 0;
912.     }
913.
914.     for (byte i = 0; i < TOTAL_PINS; i++) {
915.         // pins with analog capability default to analog input
916.         // otherwise, pins default to digital output
917.         if (IS_PIN_ANALOG(i)) {
918.             // turns off pullup, configures everything
919.             setPinModeCallback(i, PIN_MODE_ANALOG);
920.         }
921. #if defined(__AVR__)
922.         else if (IS_PIN_TONE(i)) {
923.             noTone(i);
924.         }
925. #endif
926.         else {
927.             // sets the output to 0, configures portConfigInputs

```



```

928.         setPinModeCallback(i, OUTPUT);
929.     }
930.
931.     servoPinMap[i] = 255;
932. }
933. // stop pinging
934. numActiveSonars = 0;
935. for (int i = 0; i < MAX_SONARS; i++) {
936.     sonarPinNumbers[i] = PIN_MODE_IGNORE;
937.     if (sonars[i]) {
938.         sonars[i] = NULL;
939.     }
940. }
941. numActiveSonars = 0;
942.
943. // by default, do not report any analog inputs
944. analogInputsToReport = 0;
945.
946. detachedServoCount = 0;
947. servoCount = 0;
948.
949. // stop pinging DHT
950. numActiveDHTs = 0;
951.
952. /* send digital inputs to set the initial state on the host computer,
953.    since once in the loop(), this firmware will only send on change */
954. /*
955.     TODO: this can never execute, since no pins default to digital input
956.         but it will be needed when/if we support EEPROM stored config
957.     for (byte i=0; i < TOTAL_PORTS; i++) {
958.         outputPort(i, readPort(i, portConfigInputs[i]), true);
959.     }
960. */
961. isResetting = false;
962. }
963.
964. void setup() {
965.     Firmata.setFirmwareVersion(FIRMATA_FIRMWARE_MAJOR_VERSION, FIRMATA_FIRMWARE_MINOR_VERSION);
966.
967.     Firmata.attach(ANALOG_MESSAGE, analogWriteCallback);
968.     Firmata.attach(DIGITAL_MESSAGE, digitalWriteCallback);
969.     Firmata.attach(REPORT_ANALOG, reportAnalogCallback);
970.     Firmata.attach(REPORT_DIGITAL, reportDigitalCallback);
971.     Firmata.attach(SET_PIN_MODE, setPinModeCallback);
972.     Firmata.attach(SET_DIGITAL_PIN_VALUE, setPinValueCallback);
973.     Firmata.attach(START_SYSEX, sysexCallback);
974.     Firmata.attach(SYSTEM_RESET, systemResetCallback);
975.
976.     // to use a port other than Serial, such as Serial1 on an Arduino Leonardo or Mega,
977.     // Call begin(baud) on the alternate serial port and pass it to Firmata to begin like this:
978.     // Serial1.begin(115200);
979.     // Firmata.begin(Serial1);
980.     // However do not do this if you are using SERIAL_MESSAGE
981.
982.     Firmata.begin(115200);
983.     while (!Serial) {
984.         ; // wait for serial port to connect. Needed for ATmega32u4-based boards and Arduino 101
985.     }
986.
987.     systemResetCallback(); // reset to default config
988. }
989.
990. /*=====
991.     LOOP()
992.     =====*/
993. void loop() {
994.     byte pin, analogPin;
995.
996.     /* DIGITALREAD - as fast as possible, check for changes and output them to the
997.        FTDI buffer using Serial.print() */
998.     checkDigitalInputs();
999.
1000.    /* STREAMREAD - processing incoming message as soon as possible, while still
1001.       checking digital inputs. */
1002.    while (Firmata.available())
1003.        Firmata.processInput();

```

```

1004.
1005. // TODO - ensure that Stream buffer doesn't go over 60 bytes
1006.
1007. currentMillis = millis();
1008. if (currentMillis - previousMillis > samplingInterval) {
1009.     previousMillis += samplingInterval;
1010.
1011.     if (pingLoopCounter++ > numLoops) {
1012.         pingLoopCounter = 0;
1013.         if (numActiveSonars) {
1014.             unsigned int distance = sonars[nextSonar]->read();
1015.             currentSonar = nextSonar;
1016.             if (nextSonar++ >= numActiveSonars - 1) {
1017.                 nextSonar = 0;
1018.             }
1019.             sonarLSB = distance & 0x7f;
1020.             sonarMSB = distance >> 7 & 0x7f;
1021.
1022.             Firmata.write(START_SYSEX);
1023.             Firmata.write(SONAR_DATA);
1024.             Firmata.write(sonarPinNumbers[currentSonar]);
1025.             Firmata.write(sonarLSB);
1026.             Firmata.write(sonarMSB);
1027.             Firmata.write(END_SYSEX);
1028.
1029.         }
1030.     }
1031.
1032.     if (dhtLoopCounter++ > dhtNumLoops) {
1033.         if (numActiveDHTs) {
1034.             int rv;
1035.             float humidity, temperature;
1036.
1037.             uint8_t current_pin = DHT_PinNumbers[nextDHT];
1038.             uint8_t current_type = DHT_TYPE[nextDHT];
1039.             dhtLoopCounter = 0;
1040.             currentDHT = nextDHT;
1041.             if (nextDHT++ >= numActiveDHTs - 1) {
1042.                 nextDHT = 0;
1043.             }
1044.             // clear out the data buffer
1045.             for (int i = 0; i < 4; i++) {
1046.                 dht_value[i] = (uint8_t) 0;
1047.             }
1048.             if (current_type == 22) {
1049.                 rv = DHT.read22(current_pin);
1050.             } else {
1051.                 rv = DHT.read11(current_pin);
1052.             }
1053.
1054.             if (rv == DHTLIB_OK) {
1055.                 float i, f;
1056.                 humidity = DHT.getHumidity();
1057.                 f = modff(humidity, &i);
1058.
1059.                 dht_value[0] = (uint8_t)i;
1060.                 dht_value[1] = (uint8_t)(f * 100);
1061.
1062.
1063.                 temperature = DHT.getTemperature();
1064.
1065.                 f = modff(temperature, &i);
1066.
1067.                 dht_value[2] = (uint8_t)i;
1068.                 dht_value[3] = (uint8_t)(f * 100);
1069.             }
1070.
1071.             // send the message back with an error status
1072.             Firmata.write(START_SYSEX);
1073.             Firmata.write(DHT_DATA);
1074.             Firmata.write(current_pin);
1075.             Firmata.write(current_type);
1076.             Firmata.write(abs(rv));
1077.             if (humidity >= 0.0) {
1078.                 Firmata.write(0);
1079.             }

```

```

1080.         else {
1081.             Firmata.write(1);
1082.         }
1083.         if (temperature >= 0.0) {
1084.             Firmata.write(0);
1085.         }
1086.         else {
1087.             Firmata.write(1);
1088.         }
1089.
1090.         for (uint8_t i = 0; i < 4; ++i) {
1091.             Firmata.write(dht_value[i]);
1092.         }
1093.         Firmata.write(END_SYSEX);
1094.     }
1095. }
1096.
1097. /* ANALOGREAD - do all analogReads() at the configured sampling interval */
1098. for (pin = 0; pin < TOTAL_PINS; pin++) {
1099.     if (IS_PIN_ANALOG(pin) && Firmata.getPinMode(pin) == PIN_MODE_ANALOG) {
1100.         analogPin = PIN_TO_ANALOG(pin);
1101.         if (analogInputsToReport & (1 << analogPin)) {
1102.             Firmata.sendAnalog(analogPin, analogRead(analogPin));
1103.         }
1104.     }
1105. }
1106. // report i2c data for all device with read continuous mode enabled
1107. if (queryIndex > -1) {
1108.     for (byte i = 0; i < queryIndex + 1; i++) {
1109.         readAndReportData(query[i].addr, query[i].reg, query[i].bytes, query[i].stopTX);
1110.     }
1111. }
1112.
1113. #if defined(__AVR__)
1114.     if ( keepAliveInterval ) {
1115.         currentMillis = millis();
1116.         if (currentMillis - previousKeepAliveMillis > keepAliveInterval * 1000) {
1117.             systemResetCallback();
1118.             wdt_enable(WDTO_15MS);
1119.             // systemResetCallback();
1120.             while (1)
1121.                 ;
1122.         }
1123.     }
1124. #endif
1125. }
1126.
1127. #ifdef FIRMATA_SERIAL_FEATURE
1128.     serialFeature.update();
1129. #endif
1130. }
1131.
1132. void printData(char *id, long data) {
1133.     char myArray[64];
1134.
1135.     String myString = String(data);
1136.     myString.toCharArray(myArray, 64);
1137.     Firmata.sendString(id);
1138.     Firmata.sendString(myArray);
1139. }
1140.

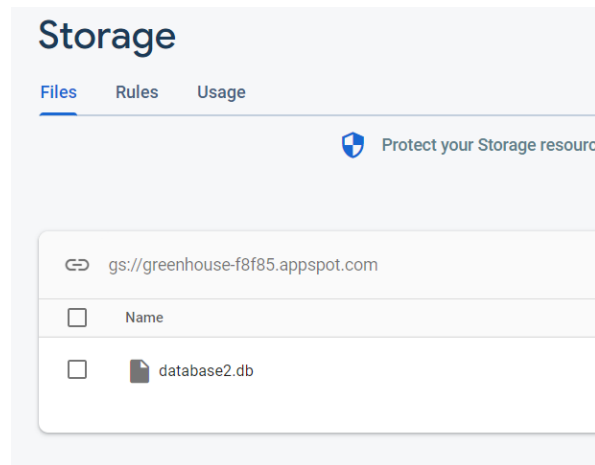
```

Firebase Realtime Database

Through the firebase console we created a real-time database instance that we connected to our project using the Web App SDK provided by firebase.

Firebase Cloud Storage

A simple storage bucket on the cloud provided by firebase, created from the firebase console as well.



SQLite Database:

We create the SQLite database through python code. Below is a little snippet on how we created the table SensorMeasures in the database file database2.db.

```
1. import sqlite3 # Library containing functions for working with SQLite (pip3 install sqlite3)
2. conn = sqlite3.connect('database2.db') #Connects to database2.db file or creates database2.db file if-
3. # -doesn't exist already
4. a = conn.cursor()
5. a.execute("""CREATE TABLE SensorMeasures(
6. MSValue text,
7. humidity integer,
8. temperature integer,
9. Alert text,
10. datetime text
11. )""")# Creating table
12. conn.commit() #saving changes
13.
14. conn.close() # closing the database file.
15.
```

We use SQLite DB browser to view the database file. Alternatives like Microsoft Access can also be used to view. This database file keeps getting stored on the cloud (updating the existing file on the cloud storage) at regular intervals.

Python Code [Arduino1 is connected to COM6 (USB) and Arduino2 is connected to COM8(USB)]

```
1. from pymata4 import pymata4 # pip3 install pymata4
2. import sqlite3 #pip3 install sqlite3
3. conn = sqlite3.connect('database2.db')
4. a = conn.cursor()
5. import datetime # pip3 install datetime
6. import serial # pip3 install serial
7. import pyrebase #pip3install pyrebase or pip3install pyrebase4
8. import time #Arduino1 connected to COM6
9. ser = serial.Serial("COM6",9600) #Connect python to take serial data from COM6 port at 9600 baud rate
10. ser.flushInput() #Clear
11. automaticswitch =0 #switch set to off
12. semiautomaticswitch=0 #switch set to off
13. i =0
14. alert =0
15. firebaseConfig = { #SDK to interact with the realtime Database
16.     "apiKey": "AIzaSyBvKEa3vLxHKUEIzrrNUGkR6ojbWnNpHU",
17.     "authDomain": "greenhouse-f8f85.firebaseio.com",
18.     "databaseURL": "https://greenhouse-f8f85-default-rtdb.asia-southeast1.firebaseio.com",
19.     "projectId": "greenhouse-f8f85",
20.     "storageBucket": "greenhouse-f8f85.appspot.com",
21.     "messagingSenderId": "328574079196",
22.     "appId": "1:328574079196:web:9ea3726a2d046d2e502419",
23.     "measurementId": "G-5HQ89J4WDY"
24. }
25. firebase = pyrebase.initialize_app(firebaseConfig) #initializing our realtime database
```

```

26. db = firebase.database() #setting our database instance as db
27. storage = firebase.storage() #setting our storage bucket instance as storage
28. poc = "database2.db"
29. pol = "database2.db"
30. board = pymata4.Pymata4('COM8') #Arduino2 connected to COM8
31. digital_pinpump = 7 # this signal indirectly activates the relay (refer to Arduino code)
32.
33. board.set_pin_mode_digital_output(digital_led) #setting pinmode output
34. board.set_pin_mode_digital_output(digital_pinpump)
35. while True:
36.     try: #we want the code to keep running on an infinite loop and when error prints '*' on console
37.         board.digital_write(digital_pinpump, 1) # initializing water pump to be set off
38.         ser_bytes = ser.readline() # reads a line out of the serial data
39.         decoded_bytes = ser_bytes[0:len(ser_bytes)-2].decode("utf-8") #decodes data to readable
            information
40.         now = datetime.datetime.now() #creating timestamp
41.         now = now.strftime("%Y-%m-%d %H:%M:%S") #formatting time stamp to Hours:Minutes:Seconds
42.         data = str( "{}'{}\r\n".format(now,decoded_bytes) ) #data = string (timestamp ,
            decodedbytes
43.         t = time.localtime() # Another timestamp
44.         current_time = time.strftime("%H:%M:%S", t)
45.         #Now we will create time stamps of seconds, minutes, hours
46.         seconds = time.localtime() #
47.         seconds_time = time.strftime("%S",seconds)
48.         usableseconds = int(seconds_time) #the second of current time
49.         minutes = time.localtime()
50.         minutes_time = time.strftime("%M",minutes)
51.         usableminutes = int(minutes_time) #the current minute
52.         hours = time.localtime()
53.         hours_time = time.strftime("%H",hours)
54.         usablehours = int(hours_time) #The current hour
55.         print(data)
56.
57.         index1 = data.find("MS")
58.         index2 = data.find("HM")
59.         msvalue = int(round(float(data[index1+4:index2]))) #finding Moisture Sensor Value in data
60.         #print(msvalue)
61.         index3 = data.find("HM =")
62.         index4 = data.find("TP =")
63.         index5 = data.find("Celcius")
64.         humidity = int(round(float(data[index3+4:index4]))) #Finding humidity value in data
65.         #print(humidity)
66.         temperature = int(round(float(data[index4+4:index5]))) #Finding temperature value in data
67.         #print(temperature)
68.         #print(usableseconds)
69.         if (usableseconds%20==0): #every 20 seconds updating the sensor values on non-relational DB
70.             #print("X")
71.             db.child('-MzPw-Ho0YALABT5Qg-y').update({"MSData":msvalue})
72.             db.child('-MzPtQo78YGTUxYrKC2u').update({"TempData":temperature})
73.             db.child('-MzPtTcEZ9-Iplu-0Nyt').update({"HumData":humidity})
74.             storage.child(poc).put(pol) #Updating the Relational Database on cloud storage
75.         else:pass
76.
77.         ATORNOTDatabase = db.child('ControlFactors').child('AutomaticOrNot').get()
78.         automaticornot= int(ATORNOTDatabase.val()) #Variable that says if greenhouse is set to
            automated or not
79.         SATORNOTDatabase = db.child('ControlFactors').child('SemiAutomaticOrNot').get()
80.         semiautomaticornot = int(SATORNOTDatabase.val()) #Variable that says if greenhouse is set to
            semi automated or not
81.         IrrigateDatabase = db.child('ControlFactors').child('Irrigate').get()
82.         irrigate = int(IrrigateDatabase.val()) #Variable that says if user is wanting to irrigate or
            not, if irrigate = 1 then the pump will turn on for duration amount of seconds.
83.         DurationDatabase = db.child('SamplePlant').child('Duration').get()
84.         duration = int(DurationDatabase.val()) #duration that the pump will be on for (seconds)
85.         IntervalsDatabase = db.child('SamplePlant').child('Intervals').get()
86.         intervals = int(IntervalsDatabase.val()) #intervals between irrigations
87.         TempReqDatabase = db.child('SamplePlant').child('TempReq').get()
88.         tempreq = int(TempReqDatabase.val()) #temperature requirement
89.
90.         #board.digital_write(digital_pinpump, 1)
91.         now1 = str(now) #String timestamp
92.         print("")
93.
94.         if ((usableminutes%30)==0): #every 30 mins check if temperature is too high or not if too
            high alert (stored into relational database)
95.             if (temperature>(tempreq+2)):

```

```

96.             alert = 1
97.         else:
98.             alert = 0
99.             a.execute("INSERT INTO SensorMeasures (MSValue, humidity, temperature , alert,
datetime) VALUES (?, ?, ?, ?, ?)", (msvalue, humidity, temperature, alert, now1 )) #sensor measures history
placed into relational database (SQLite)
100.             conn.commit() #Commit changes to relational database
101.             if ((usablehours%intervals)==0) and automaticornot==1 and automaticswitch==0:
102.                 print("") #Irrigating plants according to optimal requirements (intervals, volume)
103.                 board.digital_write(digital_pinpump, 0)
104.                 board.digital_write(digital_pinpump, 0)
105.                 board.digital_write(digital_pinpump, 0)
106.                 time.sleep(duration)
107.
108.                 board.digital_write(digital_pinpump, 1)
109.                 board.digital_write(digital_pinpump, 1)
110.                 board.digital_write(digital_pinpump, 1)
111.                 a.execute("INSERT INTO Irrigations (datetime, duration) VALUES(?,?)", (now1,
duration)) #Putting the timestamp of when irrigation happened into the relational database
112.                 conn.commit()
113.                 automaticswitch =1
114.
115.             else:
116.                 pass
117.
118.                 print("")
119.
120.                 if ((usablehours%intervals)==0) and semiautomaticornot==1 and semiautomaticswitch==0:
121.                     db.child('SamplePlant').update({"Notific":1})
122. # the notific variable on the non relational database is the variable which changes to 1 when its time
to water the plants, when this variable changes to 1 the application sends a notification to the user
through the android application
123.                     semiautomaticswitch =1
124.                     time.sleep(2)
125.                     db.child('SamplePlant').update({"Notific":0})
126.             else:
127.                 pass
128.                 if ((usablehours%intervals)!=0) and semiautomaticswitch==1:
129.                     semiautomaticswitch =0
130.             else:
131.                 pass
132.                 if ((usablehours%intervals)!=0) and automaticswitch==1:
133.                     automaticswitch =0
134.             else:
135.                 pass
136.                 print("")
137.                 if irrigate ==1: #irrigation
138.                     board.digital_write(digital_pinpump, 0)
139.                     board.digital_write(digital_pinpump, 0)
140.                     board.digital_write(digital_pinpump, 0)
141.                     print("!@")
142.                     time.sleep(duration)
143.                     board.digital_write(digital_pinpump, 1)
144.                     board.digital_write(digital_pinpump, 1)
145.                     board.digital_write(digital_pinpump, 1)
146.                     print("@!")
147.                     a.execute("INSERT INTO Irrigations (datetime, duration) VALUES(?,?)", (now1,
duration)) #irrigation recording with timestamp & volume
148.                     conn.commit()
149.                     db.child('ControlFactors').update({"Irrigate":0})
150.                 print("")
151.                 if irrigate ==0:
152.                     board.digital_write(digital_pinpump, 1)
153.                     board.digital_write(digital_pinpump, 1)
154.                     board.digital_write(digital_pinpump, 1)
155.                 print("")
156.                 if semiautomaticornot ==0 and automaticornot==1:
157.                     print("automatic rn")
158.
159.
160.                 print("")
161.
162.         except:
163.             print("")
164.             conn.close()
165.

```

Android Application

This is the main (main.dart) page (Plant Select Page), where you can select which plant out of your greenhouse you want to monitor/control

Page |

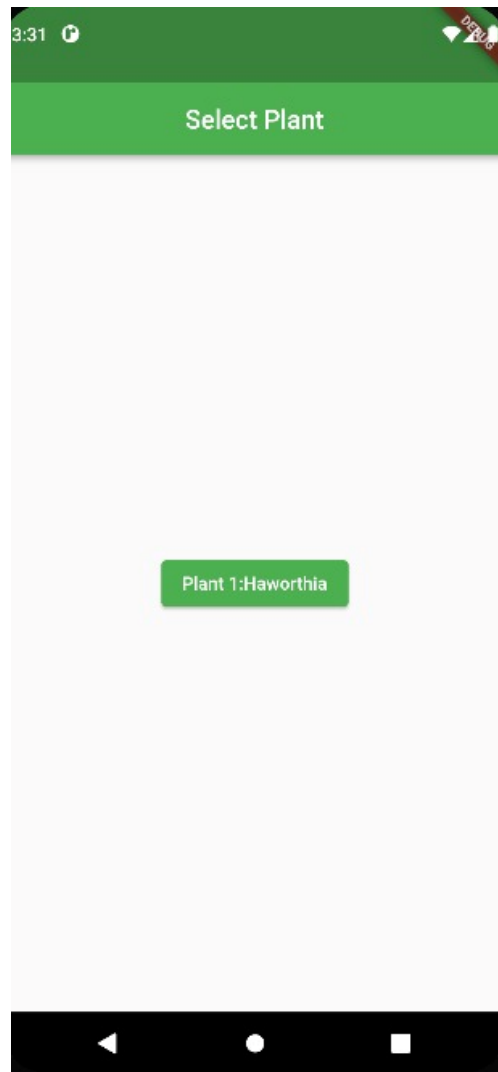
38

```
import 'package:flutter/material.dart';
import 'package:firebase_database/firebase_database.dart';
import 'package:firebase_core/firebase_core.dart';
// ignore_for_file: prefer_const_constructors
import 'plantstats.dart';

void main() {
  WidgetsFlutterBinding.ensureInitialized();
  runApp(MaterialApp(
    home: Home(),
  ));
}

class Home extends StatelessWidget {
  const Home({Key? key}) : super(key: key);

  get notificationService => null;
  @override
  Widget build(BuildContext context) {
    Firebase.initializeApp();
    return Scaffold(
      appBar: AppBar(
        title: Text('Select Plant'),
        centerTitle: true,
        backgroundColor: Colors.green,
      ),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(
                builder: (context) => PlantStats(),
              )
            );
          },
          style: ButtonStyle(backgroundColor:
            MaterialStateProperty.all<Color>(Colors.green)),
          child: Text('Plant 1:Haworthia'),
        ),
      ),
    );
  }
}
```



When plant is selected the plant stats, an irrigate button, and a settings button shows up.

```
import 'package:flutter/material.dart';
import 'package:testapp/Setting.dart';
import 'package:flutter_local_notifications/flutter_local_notifications.dart';
import 'package:firebase_database/firebase_database.dart';
import 'package:firebase_core/firebase_core.dart';
// ignore_for_file: prefer_const_constructors

class PlantStats extends StatefulWidget {
  const PlantStats({Key? key}) : super(key: key);

  @override
  _PlantStatsState createState() => _PlantStatsState();
}

class _PlantStatsState extends State<PlantStats> {
  String temp = ''; // Initializing temperature value to be shown
  String hum= ''; // Initializing humidity value to be shown
  String ms = ''; // Initializing moisture sensor value to be shown
  String notif = '';
  final _database = FirebaseDatabase.instance.ref(); // sets instance of database

  @override
  void initState() {
    super.initState();
  }
}
```



```
_activateListeners(); // Looks for changes in values
}

void _activateListeners(){
  _database.child('-MzPtQo78YGTUxYrKC2u/TempData').onValue.listen((event) {
    final Object? temperature = event.snapshot.value;
    setState(() {
      temp = '$temperature'; // detects changes in temperature and updates variable
in- -app
    });
  });
  _database.child('-MzPtTcEZ9-Iplu-0Nyt/HumData').onValue.listen((event) {
    final Object? humidity = event.snapshot.value;
    setState(() {
      hum = '$humidity'; // detects change in humidity and updates variable in app
    });
  });
  _database.child('-MzPw-Ho0YAlABT5Qg-y/MSData').onValue.listen((event) {
    final Object? moisture = event.snapshot.value;
    setState(() {
      ms = '$moisture'; // detects change In soil moisture and updates variable in
app
    });
  });
  _database.child('SamplePlant/Notific').onValue.listen((event) {
    final Object? notification = event.snapshot.value;
    setState(() {
      notif = '$notification'; // detects when notific variable turns to 1 to send-
-notification to user (code given below)
    });
  });
}

void sendNotification({String? title,String? body}) async{ // Algorithm for sending-
// -notification
FlutterLocalNotificationsPlugin flutterLocalNotificationsPlugin =
FlutterLocalNotificationsPlugin();
const AndroidInitializationSettings initializationSettingsAndroid =
AndroidInitializationSettings('@mipmap/ic_launcher');
const InitializationSettings initializationSettings = InitializationSettings(
  android: initializationSettingsAndroid,
);
await flutterLocalNotificationsPlugin.initialize(initializationSettings,
);
AndroidNotificationChannel channel = AndroidNotificationChannel(
  'high_channel',
  'High Importance Notification',
  description: "this is the channel description",
  importance: Importance.max);
flutterLocalNotificationsPlugin.show(0, title, body, NotificationDetails(
  android: AndroidNotificationDetails(channel.id, channel.name,
    channelDescription: channel.description )
));
}

@override
Widget build(BuildContext context) {
  final controldataref = _database.child('ControlFactors');
  Firebase.initializeApp();
  return Scaffold(
    appBar: AppBar(title: Text('Haworthia Stats'),
      backgroundColor: Colors.green,
    ),
    body: Column(
      children: [
```

```

        Center(
            child: getNotifStatus(),
        ),
        Container(
            padding: EdgeInsets.all(20.0),
            child:Text ('Temperature:$temp', //update view in app
            style: TextStyle(fontSize: 25)),),
    ),
    Container(
        padding: EdgeInsets.all(20.0),
        child:Text ('Humidity:$hum', //update view in app
        style: TextStyle(fontSize: 25)),
    ),
    Container(
        padding: EdgeInsets.all(20.0),
        child:Text ('Moisture:$ms',
            style: TextStyle(fontSize: 25//update view in app)),
    ),
    ElevatedButton(
        onPressed: (){
            controldataref.update({'Irrigate': 1});
        }, // Sends 1 to irrigate variable on cloud which indirectly irrigates the
plants in the greenhouse
        child: Text('Water Plant'),
    ),

]
),

floatingActionButton: FloatingActionButton(
    onPressed: (){

        Navigator.push(
            context,
            MaterialPageRoute(
                builder:(context) => Setting()
            )
        );

    },
    backgroundColor: Colors.green,
    child:const Icon(IconData(0xe57f, fontFamily: 'MaterialIcons')),
),
);

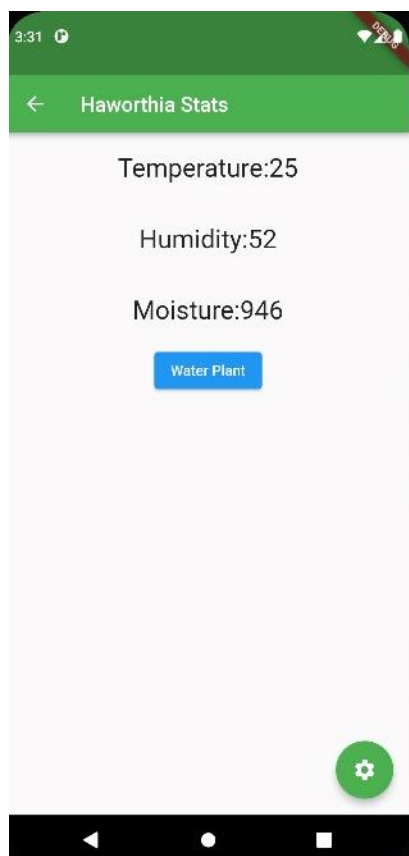
}

getNotifStatus() {
    final controldataref = _database.child('SamplePlant');
    Firebase.initializeApp();
    if(notif=='1'){// Sending notification
        sendNotification(title: "Plant Alert",body: "Water your plant");
        controldataref.update({'Notific': 0});
    }
}

```

```
}  
  
}
```

Page |
42



If you click the settings button, the settings page shows up which consists of the settings for the current plant, here you can set if its automated (automatic irrigation according to optimal requirements) or semi-automated (receive notification when to irrigate according to optimal requirements & irrigate yourself)

```
import 'package:flutter/material.dart';  
import 'package:firebase_database/firebase_database.dart';  
import 'package:firebase_core/firebase_core.dart';  
// ignore_for_file: prefer_const_constructors  
import 'plantstats.dart';  
import 'main.dart';  
  
class Setting extends StatefulWidget{  
  const Setting({Key? key}) : super(key: key);  
  
  @override  
  SettingState createState() => SettingState();  
}  
  
class SettingState extends State<Setting>{  
  final _database = FirebaseDatabase.instance.ref();  
  @override  
  Widget build(BuildContext context) {  
    final controlfactorsref = _database.child('ControlFactors');  
    Firebase.initializeApp();  
    return Scaffold(  
      appBar: AppBar(  
        title: Text('Settings'),
```

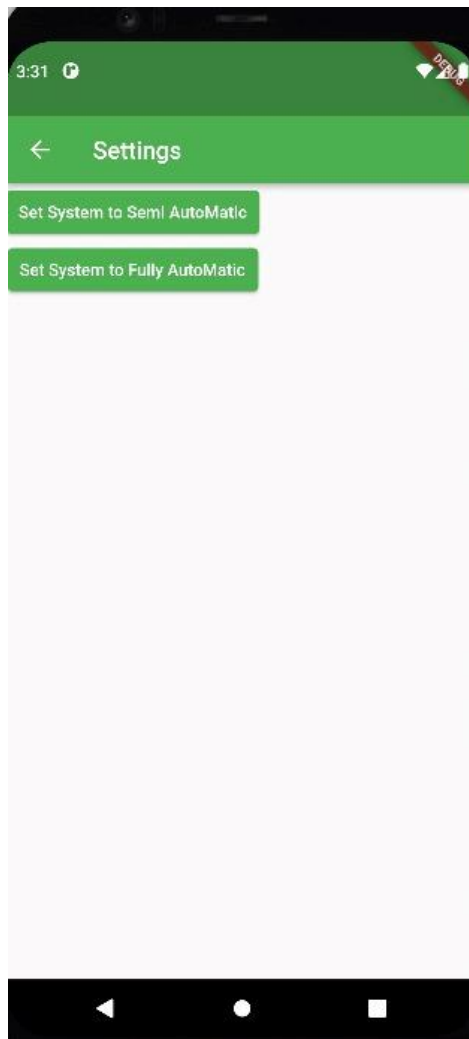
```

        backgroundColor: Colors.green,
      ),
      body: Column(
        children: <Widget>[
          ElevatedButton(
            onPressed: () {
              controldataref.update({'SemiAutomaticOrNot': 1}); // Setting to semi
automatic
              controldataref.update({'AutomaticOrNot': 0});
            },
            style: ElevatedButton.styleFrom(
              primary: Colors.green,
              padding: EdgeInsets.all(10)
            ),

            child: Text('Set System to Semi AutoMatic'),
          ),
          ElevatedButton(
            onPressed: () {
              controldataref.update({'SemiAutomaticOrNot': 0});
              controldataref.update({'AutomaticOrNot': 1});
            }, // Setting to automatic
            style: ElevatedButton.styleFrom(
              primary: Colors.green,
              padding: EdgeInsets.all(10)
            ),

            child: Text('Set System to Fully AutoMatic'),
          ),
        ],
      ),
    );
  }
}

```



Evaluation

Testing:

The components of the system undergo unit testing for each component built, and after integrating, integration testing will be done. As per the project build steps, we started with the sensing equipment first.

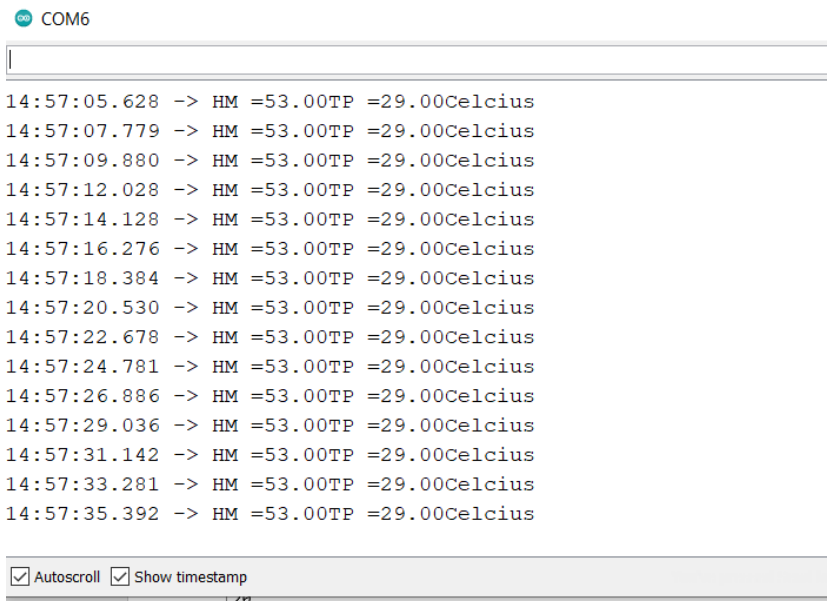
DHT11 Sensor:

Note: Circuit Connections as per the Connections summary given above.

```
1. #include <dht.h>
2. #define dht_apin A0
3. dht DHT;
4.
5. void setup(){
6.   Serial.begin(9600);
7.   delay(500);
8.   delay(1000);
9.
10. }
11.
12. void loop(){
13.   DHT.read11(dht_apin);
14.   Serial.print("HM =");
15.   Serial.print(DHT.humidity);
16.   Serial.print("TP =");
17.   Serial.print(DHT.temperature);
18.   Serial.println("Celcius");
19.   delay(2100);
20.
21. }
```

```
22.  
23. }  
24.
```

Output:



The screenshot shows a serial monitor window titled 'COM6'. It displays a series of 15 lines of data, each representing a timestamp followed by a right arrow, then 'HM' (humidity), an equals sign, 'TP' (temperature), an equals sign, and 'Celcius'. The humidity values are consistently 53.00 and the temperature values are consistently 29.00. At the bottom of the window, there are two checked checkboxes: 'Autoscroll' and 'Show timestamp'.

```
14:57:05.628 -> HM =53.00TP =29.00Celcius  
14:57:07.779 -> HM =53.00TP =29.00Celcius  
14:57:09.880 -> HM =53.00TP =29.00Celcius  
14:57:12.028 -> HM =53.00TP =29.00Celcius  
14:57:14.128 -> HM =53.00TP =29.00Celcius  
14:57:16.276 -> HM =53.00TP =29.00Celcius  
14:57:18.384 -> HM =53.00TP =29.00Celcius  
14:57:20.530 -> HM =53.00TP =29.00Celcius  
14:57:22.678 -> HM =53.00TP =29.00Celcius  
14:57:24.781 -> HM =53.00TP =29.00Celcius  
14:57:26.886 -> HM =53.00TP =29.00Celcius  
14:57:29.036 -> HM =53.00TP =29.00Celcius  
14:57:31.142 -> HM =53.00TP =29.00Celcius  
14:57:33.281 -> HM =53.00TP =29.00Celcius  
14:57:35.392 -> HM =53.00TP =29.00Celcius
```

Conclusion:

The DHT Sensor senses the temperature & relative humidity and prints it to the Serial Port.

Soil Moisture Sensor:

Circuit connected as per the connections summary & circuit diagram.

```
1. int msensor = A1;  
2. int msvalue = 0;  
3. void setup(){  
4.   Serial.begin(9600);  
5.  
6.   pinMode(msensor, INPUT);  
7.  
8.   delay(500);  
9.   delay(1000);  
10.  
11. }  
12. void loop(){  
13.  
14.   msvalue = analogRead(msensor);  
15.   Serial.print("MS =");  
16.   Serial.println(msvalue);  
17.   delay(2000);  
18. }  
19.
```

Output:

The moisture sensor was dipped into water.

```
COM6

15:04:05.222 -> MS =1020
15:04:07.212 -> MS =1020
15:04:09.212 -> MS =423
15:04:11.249 -> MS =372
15:04:13.241 -> MS =371
15:04:15.237 -> MS =475
15:04:17.237 -> MS =354
15:04:19.232 -> MS =286
15:04:21.228 -> MS =264
15:04:23.228 -> MS =492
15:04:25.273 -> MS =1020
15:04:27.259 -> MS =1020
15:04:29.249 -> MS =260
15:04:31.289 -> MS =1020
15:04:33.285 -> MS =1020
15:04:35.279 -> MS =1021

☐ Autoscroll ☒ Show timestamp
```

Conclusion:

The moisture sensor sensed the moisture its in contact with and the same gets printed on the Serial Port.

DC Motor & Motor Driver IC:

Connections made as per circuit diagram & connections summary.

```
1. #define ENABLE 5
2. #define DIRA 3
3. #define DIRB 8
4.
5. int i;
6.
7. void setup(){
8.   pinMode(ENABLE, OUTPUT);
9.   pinMode(DIRA, OUTPUT);
10.  pinMode(DIRB, OUTPUT);
11.  Serial.begin(9600);
12. }
13. void loop(){
14.   digitalWrite(ENABLE, HIGH);
15.   digitalWrite(DIRB, HIGH);
16.   digitalWrite(DIRA, LOW);
17. }
18.
```

Output:

The motor starts rotating (attached to a fan creating air flow).



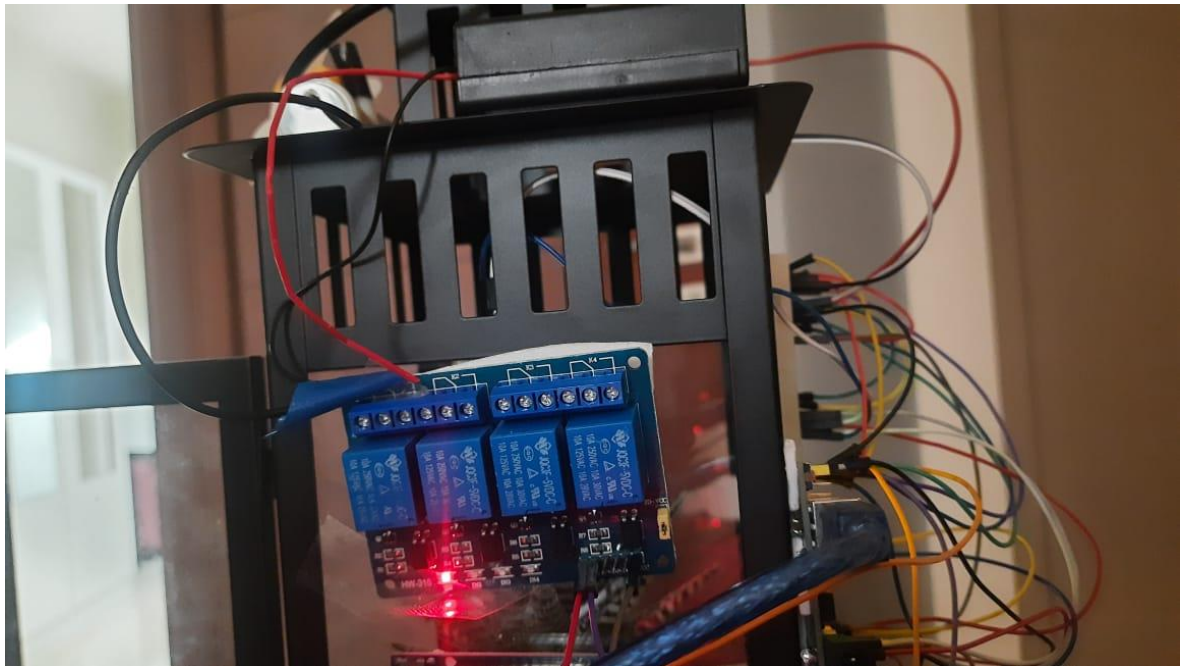
Conclusion: The motor starts rotating hence creating airflow which can be used as temperature control.

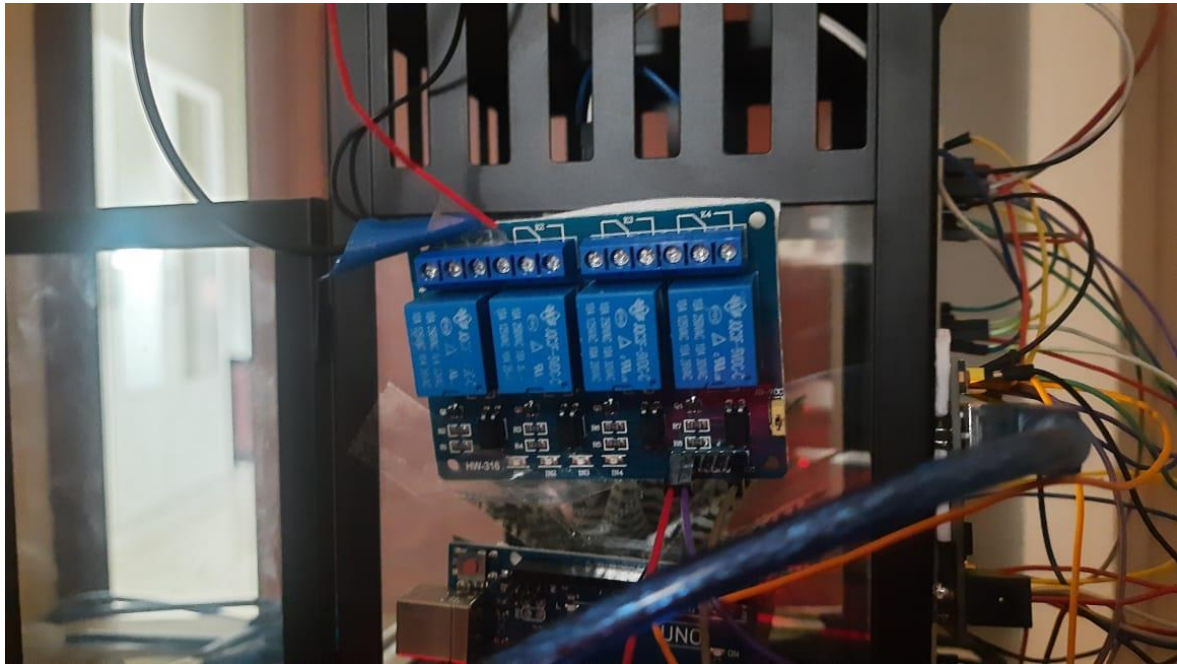
Water Pump w/ Relay:

```
1. #define relay 7
2. #define pinpump 13
3. int pumpstate =0;
4. void setup(){
5.     Serial.begin(9600);
6.     pinMode(relay, OUTPUT);
7.     pinMode(pinpump, INPUT);
8.     digitalWrite(relay, LOW);}
9. void loop(){
10.    pumpstate = digitalRead(pinpump);
11.    digitalWrite(relay, pumpstate);
12.    digitalWrite(relay, HIGH);
13.    delay(2000);
14.    digitalWrite(relay, LOW);
15.    delay(2000);
16. }
17.
```

Output:

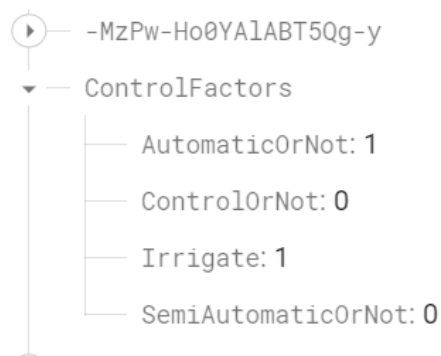
The relay turns on for two seconds & turns off for 2 seconds.





Conclusion: The water pump turns on which will dispense (irrigate) for a required amount of time (optimal volume) between intervals (when controlled by Arduino2 which has intervals set through python code).

Cloud Testing with Relay (Irrigate):



```
1. import datetime
2. import serial
3. import pyrebase
4. import time
5.
6. automaticswitch =0
7. semiautomaticswitch=0
8. firebaseConfig ={
9.     "apiKey": "AIzaSyBvKEa3vxLxHKUEIzrrNUGkR6ojbWnNpHU",
10.    "authDomain": "greenhouse-f8f85.firebaseio.com",
11.    "databaseURL": "https://greenhouse-f8f85-default-rtdb.asia-southeast1.firebaseio.com",
12.    "projectId": "greenhouse-f8f85",
13.    "storageBucket": "greenhouse-f8f85.appspot.com",
14.    "messagingSenderId": "328574079196",
15.    "appId": "1:328574079196:web:9ea3726a2d046d2e502419",
16.    "measurementId": "G-5HQ89J4WDY"
17. }
18. firebase = pyrebase.initialize_app(firebaseConfig)
19. db = firebase.database()
20. board = pymata4.Pymata4('COM8')
21. digital_pinpump = 7
22. board.set_pin_mode_digital_output(digital_pinpump)
23. while True:
24.     try:
25.         ATORNOTDatabase = db.child('ControlFactors').child('AutomaticOrNot').get()
26.         automaticornot= int(ATORNOTDatabase.val())
27.         SATORNOTDatabase = db.child('ControlFactors').child('SemiAutomaticOrNot').get()
28.         semiautomaticornot = int(SATORNOTDatabase.val())
29.         IrrigateDatabase = db.child('ControlFactors').child('Irrigate').get()
```

```

30.         irrigate = int(IrrigateDatabase.val())
31.         DurationDatabase = db.child('SamplePlant').child('Duration').get()
32.         duration = int(DurationDatabase.val())
33.         if ((usablehours%intervals)==0) and automaticornot==1 and automaticswitch==0:
34.             print("")
35.             board.digital_write(digital_pinpump, 0)
36.             board.digital_write(digital_pinpump, 0)
37.             board.digital_write(digital_pinpump, 0)
38.             time.sleep(duration)
39.
40.             board.digital_write(digital_pinpump, 1)
41.             board.digital_write(digital_pinpump, 1)
42.             board.digital_write(digital_pinpump, 1)
43.
44.             automaticswitch =1
45.
46.         else:
47.             pass
48.         if irrigate ==1:
49.             board.digital_write(digital_pinpump, 0)
50.             board.digital_write(digital_pinpump, 0)
51.             board.digital_write(digital_pinpump, 0)
52.             print("!@")
53.             time.sleep(duration)
54.             board.digital_write(digital_pinpump, 1)
55.             board.digital_write(digital_pinpump, 1)
56.             board.digital_write(digital_pinpump, 1)
57.             print("@!")
58.         db.child('ControlFactors').update({"Irrigate":0})
59.         print("")
60.         if irrigate ==0:
61.             board.digital_write(digital_pinpump, 1)
62.             board.digital_write(digital_pinpump, 1)
63.             board.digital_write(digital_pinpump, 1)
64.         print("")
65.         if semiautomaticornot ==0 and automaticornot==1:
66.             print("automatic rn")
67.     except:
68.         print("*")
69.

```

Output:

```

type help , copyright , credits or license() for more information.
>>>
= RESTART: C:\Users\zayed\Downloads\FAAIZ\FirebaseConnection2\CloudRelayTest.py
!@
@!
>>> |

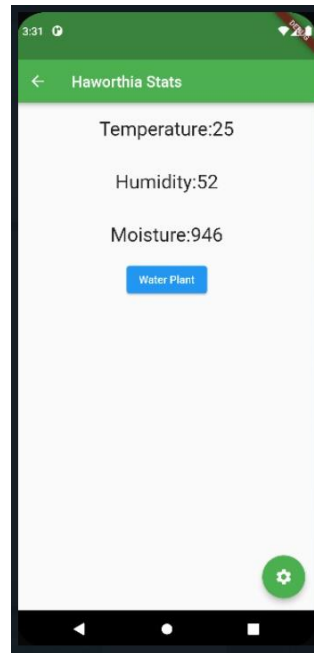
```

‘Irrigate’ Changes to 1 the relay turns on when irrigate on cloud changes to 1 and after duration python turns irrigate variable on cloud back to 0+*

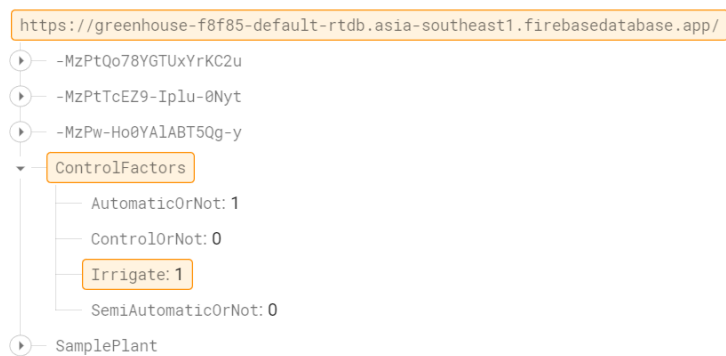
Conclusion: Whenever irrigate on cloud changes to 1, or it is time to irrigate (according to optimal requirements) the water pump (relay) will turn on.

Application Testing:

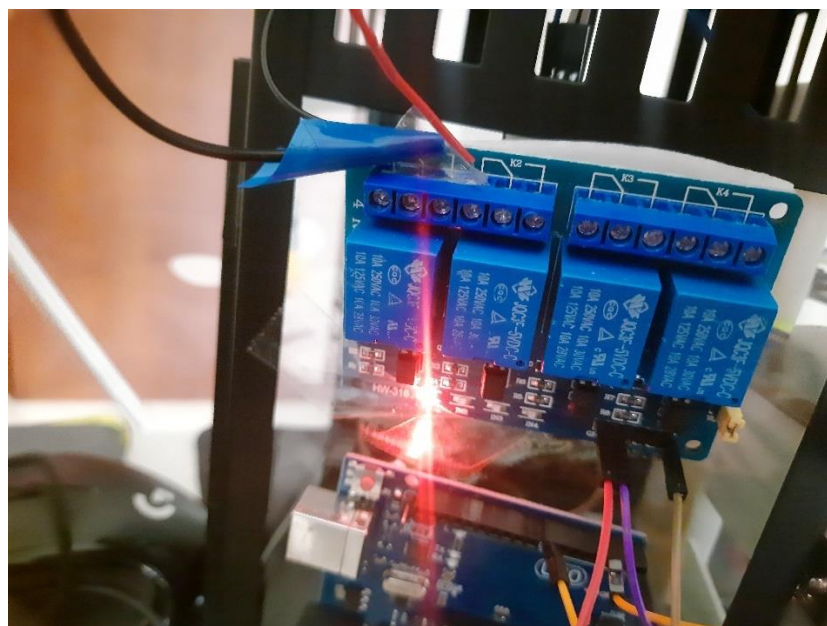
Whenever the irrigate button is pressed the irrigate variable on the cloud should change to 1.



Button pressed



Irrigate Value Changing to 1



Relay (pump) turns on

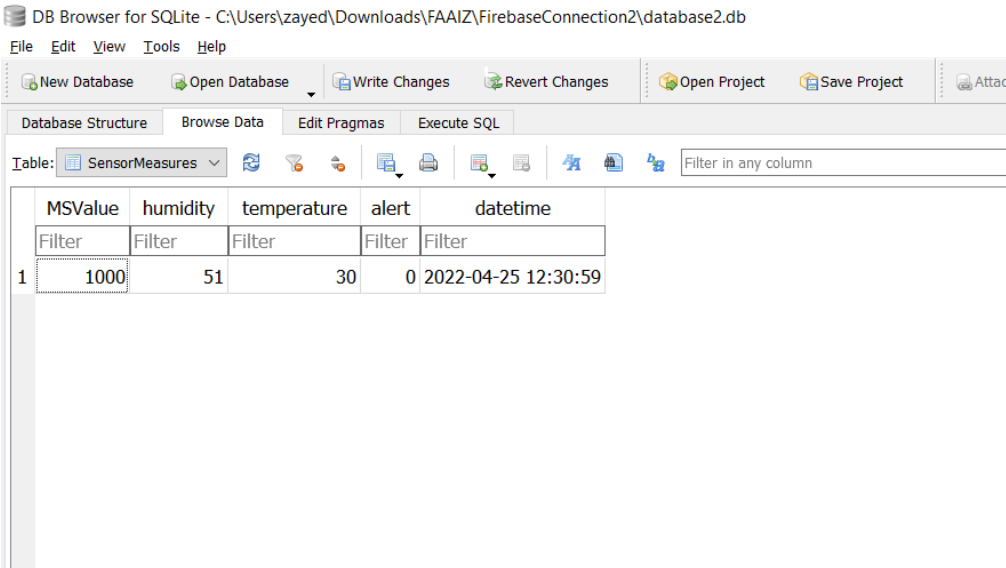
Conclusion: We are able to indirectly turn on the water pump just by pressing the Irrigate button on the android application.

Relational Database Testing:

After initially creating the relational database and the tables (with the attributes) we move forward to test it.

```
1. import sqlite3
2. conn = sqlite3.connect('database2.db')
3. a = conn.cursor()
4. msvalue = 1000
5. humidity = 51
6. temperature = 30
7. alert = 0
8. now1= str("2022-04-25 12:30:59")
9. a.execute("INSERT INTO SensorMeasures (MSValue, humidity, temperature , alert, datetime) VALUES
  (?, ?, ?, ?, ?)", (msvalue, humidity, temperature, alert, now1 ))
10. conn.commit()
11. conn.close()
12.
```

Output:



The screenshot shows the DB Browser for SQLite interface. The title bar indicates the file path: C:\Users\zayed\Downloads\FAAIZ\FirebaseConnection2\database2.db. The menu bar includes File, Edit, View, Tools, and Help. The toolbar contains buttons for New Database, Open Database, Write Changes, Revert Changes, Open Project, Save Project, and Attach. The main window has tabs for Database Structure, Browse Data, Edit Pragmas, and Execute SQL. The 'Browse Data' tab is active, showing a table named 'SensorMeasures'. The table has five columns: MSValue, humidity, temperature, alert, and datetime. Each column has a 'Filter' button. The table contains one row of data with the following values: 1000, 51, 30, 0, and 2022-04-25 12:30:59.

	MSValue	humidity	temperature	alert	datetime
1	1000	51	30	0	2022-04-25 12:30:59

Conclusion: We can add data to the relational database through python script this will be integrated into the main python script.

Conclusion

We began this project to show how the problems given in the problem definition can be solved.

Our project aims were:

- Making an environment monitoring system that can help monitor the conditions efficiently.
- Making a system that makes it easy to set requirements (by the user) according to what plants the greenhouse consists of.
- Making a database that consists of requirements that help the plant grow optimally.
- Making a system that can act upon the requirements (regulate temperature, irrigate)
- Minimizing labour costs by automating tasks such as irrigation.

We proceeded to solve these problems by creating a system that:

- Sense the environment (Soil Moisture level, temperature, humidity)
- Send current sensed values to a cloud non-relational database and store history of values on a relational database that is being stored on a cloud as well.

- Control the environment (Temperature, Irrigation) based on what's optimal for the plant (the optimal conditions and requirements like temperature & irrigation intervals/volume are stored on the non-relational database).
- Store control history (history of when irrigations occurred) on relational database on cloud.
- Automate the above features.
- If needed provide control to user (Irrigate through application) or change the optimal requirements of the plant themselves (if they have expertise) to suit the plants they have.
- Help the user monitor the environment remotely (through Android Application or through the relational & non-relational database).
- Notify the user when the plants need to be watered through the android application.

System Limitations:

- Since the system is always connected to the PC through wires, in a real-world scenario, the internet connection and connection to the PC needs to always be stable otherwise interruptions in the system functions may occur.
- In the current system, the methods of temperature control are only by using fan, this method would not work well in an environment that's too hot since it would only be circulating the hot air (solution explained in future directions)
- Apart from lowering labour costs, in large scale, electrical costs could go up when using such a system.
- There is a little delay between the irrigate button in the application pressed and the water pump turning on.
- This system for now cannot monitor things like soil nutrition, light & sterility.
- The person operating the whole system (if in large scale) needs to have expertise in plants and needs to be trained (or read the technical report) to use the system since it might be a quite complicated.
- In the current prototype (miniaturization) the pipe for water flow is too short.

Future Directions:

- The android application UI to be made cleaner and more features to be included within the application such as temperature control, show biodata of plant.
- Add ways of sensing soil nutrition, light (sunlight) & sterility.
- Greenhouse cooling system could be improved by using air conditioning or by adding a water mist sprayer along with fans which can decrease the temperature by a lot (evaporation off surfaces will leave the surface cooler).
- Make the communication between the system and the non-relational database wireless (Wi-Fi, MQTT).
- Use solar panels to power the whole system making it a lot less expensive to run and efficient.
- Add more plants for users to choose from along with their respective optimal requirements.
- Create a User Guide to help new users.
- Add an authentication (login) page to the android application (Firebase Authentication + Android Studio).
- Add more tables and fields to the relational database that can help track more stats (Electricity & Water consumption per unit time)
- To the current prototype, add a sprinkler to the end of the pipe to further improve irrigation.

References

[1] Hemming, S., de Zwart, F., Elings, A., Righini, I. & Petropoulou, A. 2019, "Remote Control of Greenhouse Vegetable Production with Artificial Intelligence-Greenhouse Climate, Irrigation, and Crop Production", *Sensors (Basel, Switzerland)*, vol. 19, no. 8, pp. 1807.

Page | Somov, A., Shadrin, D., Fastovets, I., Nikitin, A., Matveev, S., Seledets, I. & Hrinchuk, O. 2018, "Pervasive
53 Agriculture: IoT-Enabled Greenhouse for Plant Growth Control", *IEEE Pervasive Computing*, vol. 17, no. 4, pp. 65-75.

Ullah, I., Fayaz, M., Aman, M. & Kim, D. 2022, "An optimization scheme for IoT based smart greenhouse climate control with efficient energy consumption", *Computing*, vol. 104, no. 2, pp. 433-457

Appendices

All files related to the project & project codes uploaded to this google drive in zip file:

<https://drive.google.com/file/d/1DTT4OfzuLuzjb5DKGWe97SJAIWXYtU9h/view?usp=sharing>