Waterford Institute *of* Technology

The 24/7 Gardener

Student: TJ Fitzpatrick.

Student Number: 20027865.

Date: April 18h 2022.

Supervisor: Mr Richard Lacey.

Award: Higher Diploma in Computer Science.

Table of Contents

List Of Figures

Table of Tables

Declaration.

I declare that the work which follows is my own, and that any quotations from any sources (e.g., books, journals, the internet) are clearly identified as such by the use of ‘single quotation marks’, for shorter excerpt and identified italics for longer quotations. All quotations and paraphrases are accompanied by (date, author) in the text and a fuller citation is the bibliography. I have not submitted the work represented in this report in any other course of study leading to an academic award.

Student: TJ FITZPATRICK Date: 18/04/2022

Workplace Mentor:

1. Introduction.

This final year project is submitted as part requirement for the award of an academic level 8 Higher Diploma in Computer Science. This project is being submitted to the third level University Institution - Waterford Institute of Technology.

1.1 Acknowledgement

First and foremost, I would like to thank my project supervisor, Mr. Richard Lacey, who guided me throughout this project. Richard provided invaluable advice and insight at difficult times, allowing me to develop my skills to complete project milestones. His motivation and guidance contributed tremendously to the successful completion of this project.

Additionally, I would like to thank all the lecturers in the Department of Computer Science who helped not just me, but the whole class, through a difficult but rewarding journey. With much of the course being held online during the COVID-19 pandemic, and through multiple countrywide lockdowns, the department's ability to quickly adapt and still provide quality learning opportunities is inspirational. I am grateful for the opportunity that has been provided to me though this trying time.

Finally, last but not in least, I would like to thank everyone who helped and motivated me to work on this project. Family, friends, and classmates together provided me with the drive and determination to push myself and develop my technical ability.

1.2 Background

The rationale behind my project, an Internet of Things (IOT) project, was derived from the COVID-19 pandemic. The pandemic shifted both work and social aspects of life in a direction never seen before. As a society during this time, we began to rely more and more heavily of technology, to not only function from a work perspective, but a social one too.

With many people utilising existing spaces like bedrooms and utility closest as home offices, new challenges began to emerge. In my own case, I worked 8-9 hours a day from my office desk in my bedroom, another 3-4 hours completing college work, and I would of course sleep for 7-8 hours. Any remaining time would be spent keeping fit, talking with friends, or watching television. Much of this social time was again spent at home, and sometimes in the same room as my work and college studies. As such, I found oxygen levels in the room to be problematic. Fatigue, irritability, trouble breathing, and poor sleeping patterns were some of the symptoms of this environment.

From this, in conversation with one of my work colleagues I found others were experiencing similar problems. Their recommendation was to invest in good quality house plants to clean the air. Not having much experience with plants, I feared that maintenance would be a problem. However following advice from my colleague, I

quickly became proficient in caring for the plants that were used in my home to improve air quality. Water, at required intervals, based on soil conditions was the key component of care and was easily monitored when spending large portions of time at home.

Recently, with the COVID-19 pandemic coming to an end and given endemic status instead, a sudden shift occurred with people going back to the office, booking holidays abroad, and socialising in person. With this came a challenge in order to maintain these newly acquired plants. From this I was inspired to bridge my interest in home automation technologies with plant care. As such I invented in parts to create a smart IOT system that would monitor and feed the plants when I am not around. This final project for the Higher Diploma in Computer Science was the perfect opportunity for this, and with this in mind the next step was to research different ideas in order to make a project proposal.

2. Project Research.

2.1 Review of the Online Literature

To start with, There were 2 different online projects that aimed to automate the care of their plants, using an IOT approach “The Raspberry Pi Powered Garden” (Technovation, 2022), and “The Automated Garden System Built on Raspberry Pi for Outdoors or Indoors” (mudpi, 2022). The main problem that both projects encountered, revolved around the creation of a system that would monitor current moisture levels in the soil, to determine if water was needed. An additional problem was encountered in the monitoring of oxygen levels in the air near or around the plants. Both projects considered the use of a standalone Arduino, however this limited what the gardener could see in terms of data. The gardener would only be able, through the use of complicated ICT software and interfaces, what the Arduino was currently seeing through its sensors. Both projects, after exploring their options, decided to introduce a Raspberry Pi, which was able to act as intermediator between the Sensors and a self-hosted database, allowing potential gardeners to view historical data on their plants watering schedule and influence on air quality. In one of the projects the garden was outdoors.

2.2 Methodology

2.2.1 The Raspberry Pi Powered garden.

This IOT system functions using the following processes:

A Raspberry Pi is used to relay useful information of the garden, such as luminosity, and humidity from various sensors and relay this information into a cloud database.

Once the information is in the cloud, it can be accessed from anywhere using a smartphone app that the author built.

The following are some of the key features of the garden:

- Real-time feedback of the garden's various sensors
- Database of the garden's health status
- Global monitoring and operating capacities

In this project they used Google's Firebase as the intermediary of their IOT system, to create their own free cloud database.

They then used “MIT's App Inventor” (MIT, 2022) to create a smartphone application which is compatible with the Firebase database and the Raspberry Pi.

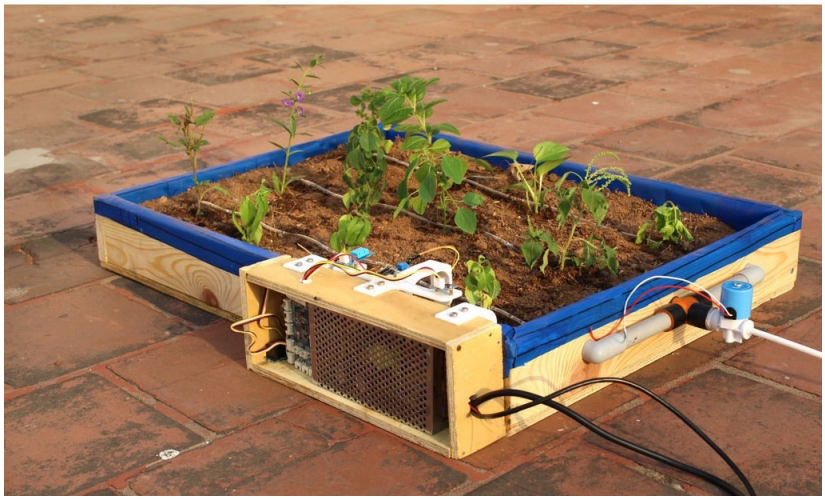


Figure 1 - The Raspberry Pi Powered Garden

(Raspberry Pi Powered IOT Garden, 2022)

2.2.2 **The** Automated Garden System Using a Raspberry PI.

This type of system uses a program called a MudPi.

A MudPi is an open-source garden system the author made to manage and maintain garden resources; it is built on a Raspberry Pi. A Debian operating is loaded onto the raspberry pi and MudPi application is then downloaded, the user can then add specific sensors to specific pins.

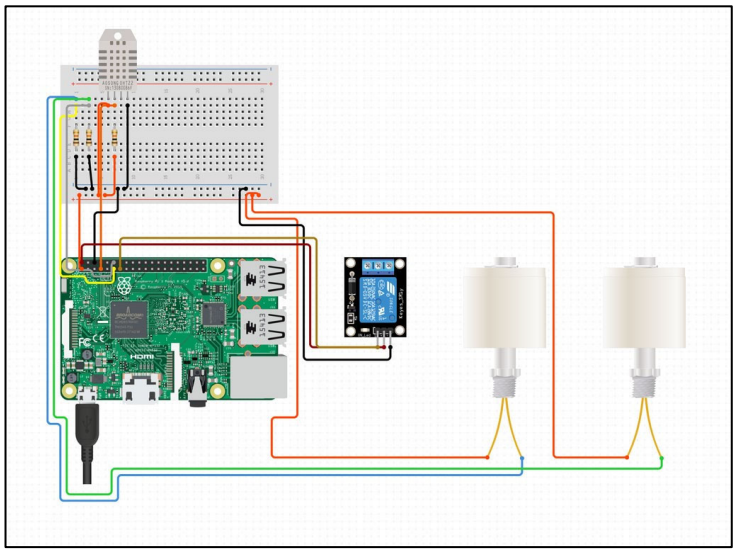


Figure 2 - Example of a MudPi Circuit Diagram.

(mudpi, 2022)

The sensors then relay information back to the user over the Wifi.

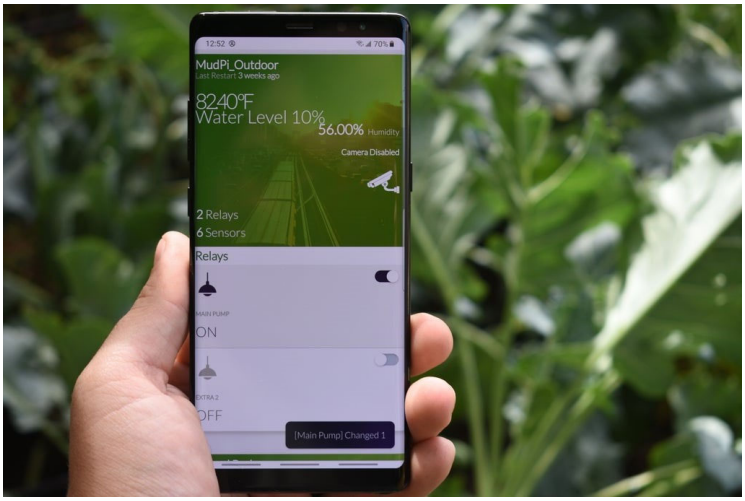


Figure 3 – Example of A MudPi Application

(mudpi, 2022)

Upon researching these 2 systems a project proposal was put together.

3. Project Proposal

This project is an application of green technologies for sustainable living. An indoor garden will be created, where plants (Snake Plant, Peace Lilly and Spider) will help clean and recycle the air. The technological solution will measure the oxygen and carbon dioxide levels in the air, and display this using an android application. Building on this idea, other fruit and vegetables will be grown with the aid of robots to assist with irrigation by using thresholds for dryness and wetness.
This project is broken into 2 parts, the hardware, and the software.
The hardware includes different sensors to measure different quantities in the garden then the first piece of software will run on a Raspberry Pi that will interface and read the sensors then a native Android app will be built to monitor and display these values of the garden.

An analysis of green technologies based on IOT solutions will be carried to identify potential solutions and features for my project. These include:

1. The Raspberry Pi Powered Garden.
2. The Automated Garden System Built of Raspberry PI for Outdoors or Indoors.

3.1 Technologies.

Hardware Requirements

1. A main mother board e.g. (Raspberry Pi, Arduino).
2. Sensors (light sensor, soil/moisture sensor, CO2 sensor).
3. Water pump.
4. LCD screen.

Software Requirements

1. Android Studio (Kotlin or Java).
2. Database (Firebase or MongoDB).

There are many different software lifecycles that could be used in this project, but for the purpose of this project the decision was either Kanban or SCRUM and implement a Trello board to monitor the progress of this project.
The whole idea is that all parts of the project be broken into can smaller tasks where I plan, build, test, and review, then put all finished pieces together at the end to create the finished product.
The software design methodology I will use in this case is the Waterfall method.

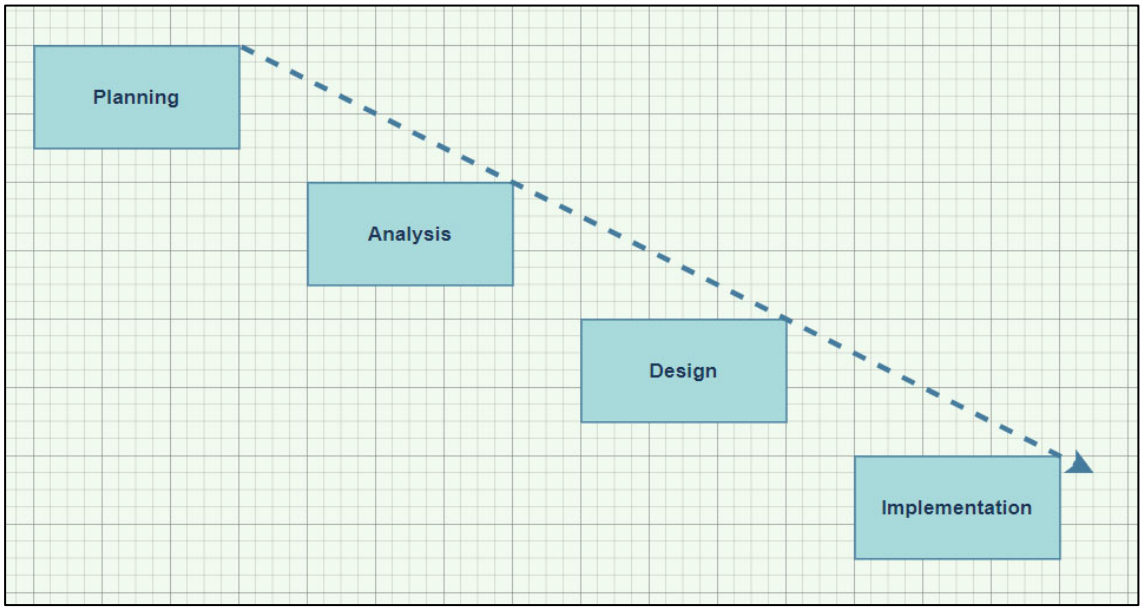


Figure 4 - Waterfall Diagram

4. Project Specification.

The next step once the project proposal was made and accepted was to think of a way to implement this idea. In Order to track this project's deliverables Trello was used as a Kanban board to track and keep this project moving, this allowed me to keep track of everything in once place.

The different columns meant the following:

Backlog: This was a backlog of works to be completed based on ideas from project meetings.
Doing: This is works and ideas that I am currently working on.
Testing: This is works and Ideas that I have competed but just putting them through a basic test.
Done: This was works completed and tested.
Weekly Reports: This was just my own weekly log.
Weekly Meetings: This was a photo of the project meetings notes that I had made with my supervisor.

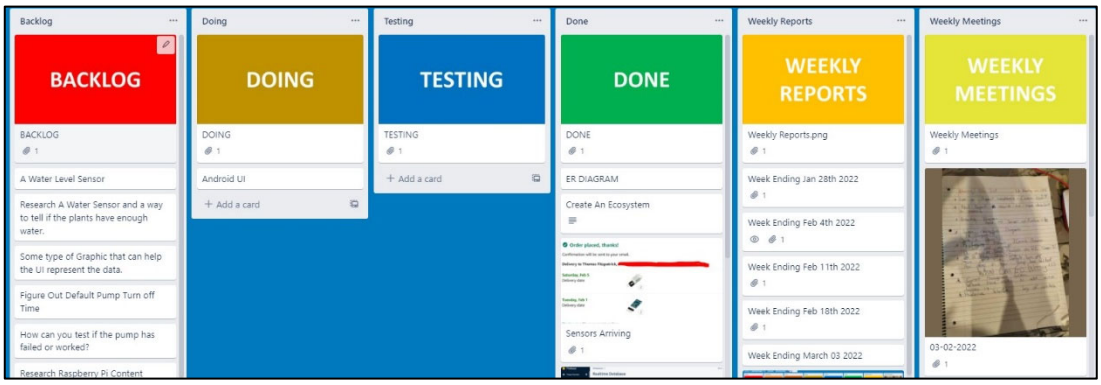


Figure 5 - Trello Board

Once I had figured a system to manage this project my next step was to research different technologies that would help me build a similar system to that I had researched.

4.1 Main Controller.

Upon researching and evaluating 4 different single board computer technologies It was decided to use the Raspberry Pi as the main controller. The reason being was when we compared the Pi to other boards there was a lot more information and sensors available online.

4.1.1 Raspberry Pi Vs ODROID XU4.

I chose the Raspberry Pi in this case, for one reason is for the greater RAM will help run applications faster but the main reason is the Pi has a huge global community which is unmatched.
This means there's ample information and supports for new users as well as continued development and maintenance of software.
Although the ODROID community is growing fast.

4.1.2 Raspberry Pi Vs ASUS Tinker Board.

Overall, both the Raspberry Pi 4 and the Asus Tinker Board have strong online communities, and great support, for open-source projects available to try out.

However, the Tinker Board is from 2017 and definitely shows its age in comparison to the connectivity options of the Pi 4.

Also, the current price of the Tinker board far outweighs my budget compared to the Raspberry Pi which is more affordable in my case.

4.1.3 Raspberry Pi Vs Arduino.

The Raspberry Pi can do everything that an Arduino can do, but it does need a little help in the form of HATs and add on boards, because certain features like analogue-to-digital conversion aren't built in there are a lot more libraries available online and a lot more tools available for the PI compared to the Arduino.
The Arduino is a truly versatile board, but the Raspberry Pi is a full computer.
If you need wireless communication, raw processing power and access to the GPIO then the Raspberry Pi gives you all of that in a small package.

Controller	Cost	RAM	GPIO	Bluetooth	WIFI
Raspberry Pi 4B	$35.00	8GB	40 pins	V5.0	Wi-F 802.11b/g/n
ODROID XU4	$95.00	2GB	N/A	N/A	Wi-F 802.11b/g/n
ASUS Tinker Board	$105.00	2GB	28 pins	BLE	Wi-F 802.11b/g/n
Arduino	$24.05	2K SRAM 1K EEPROM	20 pins	Add on Board	Add on board

Table 1- Main Controller Comparison.

4.2 Sensors.

The sensors I found online include soil moisture, air quality, temperature and humidity.

4.2.1 Temperature and Humidity.

When researching temperature and humidity sensors, I came across two different family related sensors the DHT11 and DHT22, I decided to evaluate both.

The benefits of these type of sensors include great long-term stability and low consumption of power.
In addition, you can get relatively high accuracy in measurement at an affordable rate.
Both use the same family of internal chips but only one is more accurate than the other.

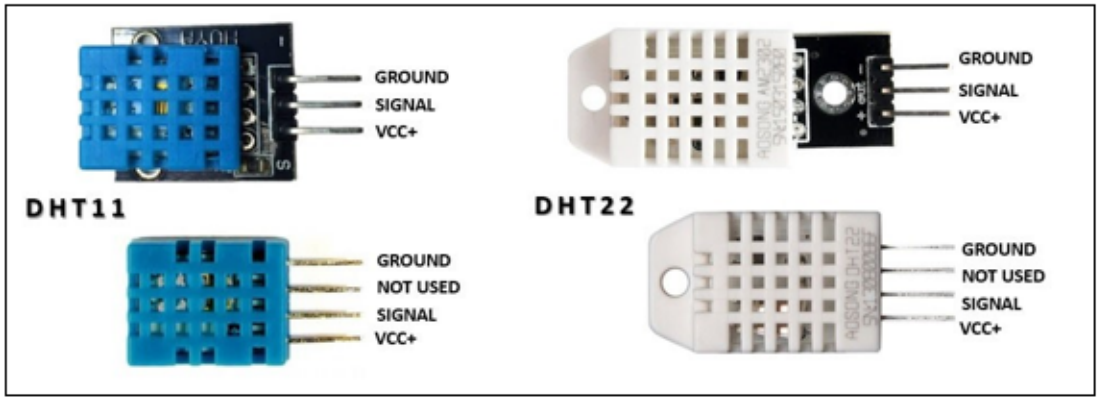


Figure 6 - DHT11 & DHT22 Sensors

	Temperature Range	Temperature Accuracy	Humidity Range	Humidity Accuracy	Cost
DHT11	-20 to 60°C	±2%	5 to 95% RH	±5%	$5.90
DHT22	-40 to 80°C	±0.5%	0 to 100%RH	±2%	$9.90

Table 2 - DHT11 vs DHT22

The DHT22 outshines the DHT11 in every aspect from temperature range, temperature accuracy, humidity range to humidity accuracy. The only downside of the DHT22 is, of course, the slightly higher price but you are paying for the better specs.

4.2.2 Soil Moisture.

I evaluated two different soil moisture sensors, but the sensor I'm going to go with is the YL69 Sensor. The reason being is that both send back an analogue reading for the soil moisture, but because I decided to use the Raspberry Pi it cannot read an analogue measurement.

So, the YL-69 has an extra built on board that will allow me to set the sensitivity value once this value has been passed the board will send a logic one to the Raspberry Pi. If I was going to use the Aideepen I would have to use an Arduino or some other Microcontroller that reads analogue signals and then send this signal to the PI over some type of bus or even wirelessly.

Aideepen	YL-69 Sensor
	

Figure 7 - Soil Moisture Sensors

4.2.3 Air Quality Sensor.

The next and last sensor I investigated was an is MQ-135.

The MQ-135 Gas sensor can detect gases such as Ammonia (NH3), Sulphur (S), Benzene (C6H6), CO2, and other harmful gases and smoke. There are other MQ gas sensors in this series, but unlike them this sensor also has a digital and analogue output pin. When the level of these gases goes beyond a threshold limit in the air the digital pin goes high. This threshold value can be set by using the on-board potentiometer. The analogue output pin, outputs an analogue voltage which can be used to approximate the level of these gases in the atmosphere.

Figure 8- Air Quality Sensor

4.2.4 Water Pump.

The idea of this project was to try keeps the plants alive, I needed to invest in some type of irrigation system, I investigated multiple different pumps but settled on a single one this was due to budget.

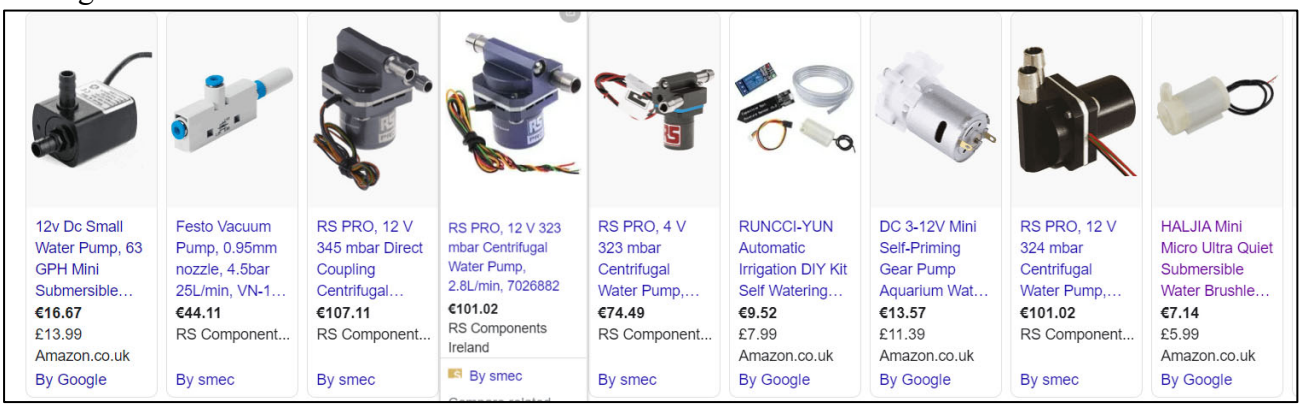


Figure 9 - Water Pumps

The following is a 5V water pump that can easily be interfaced to the Raspberry Pi.

Figure 10 - Water Pump

Once I had chosen the sensors, I was going to try interface to the Raspberry Pi the next step was to evaluate some type of database that was going to hold and store data.

4.3 Database (Firebase vs Mongo)

There were 2 choices, this was only due to studying them during the course.

5. Name	Firebase Realtime Database	MongoDB
Description	Cloud-hosted Realtime document store. iOS, Android, and JavaScript clients share one Realtime Database instance and automatically receive updates with the newest data.	One of the most popular document stores available both as a fully managed cloud service and for deployment on self-managed infrastructure
Primary database model	Document store	Document store
SQL	no	Read-only SQL queries via the MongoDB Connector for BI
APIs and other access methods	Android iOS JavaScript API RESTful HTTP API	proprietary protocol using JSON

Table 3 - Evaluation of Databases

I have chosen to the firebase Realtime Database only because there are API's available for the Android operating system, and it will make the project development life cycle a lot more efficient, in the future there could be a possibility to change to a Mongo DB, but for now it will be a Firebase Realtime Database.

The next step of this project was to draw up a system diagram of what I thought the system functioned, this allowed me to be able to explain to others what this project was about.

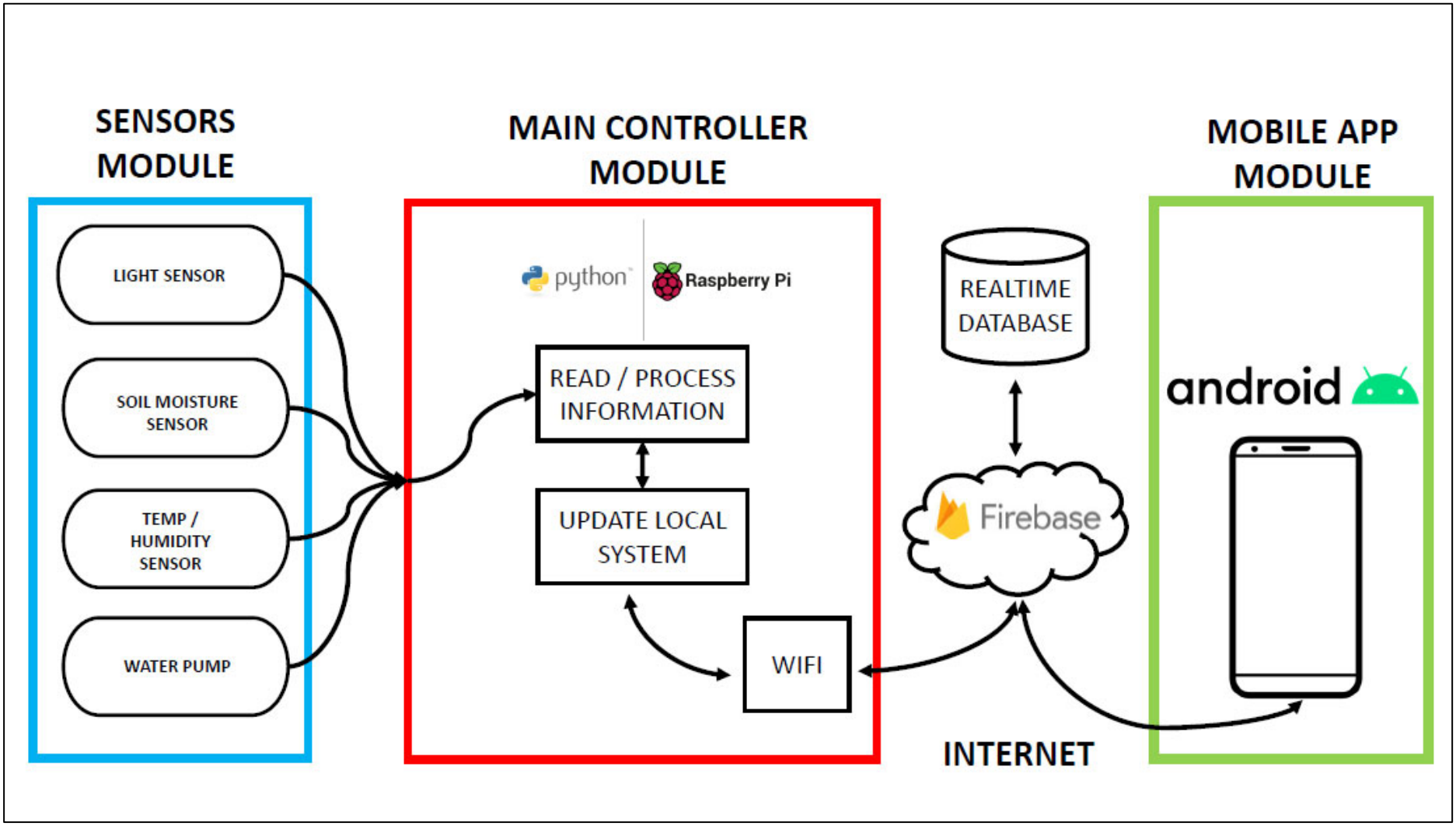


Figure 11 - System Architecture Diagram.

5. Project Implementation.

5.1 Python Interface to Firebase Database.

A very abstract idea of this project is to have Android communicating with a Raspberry Pi and vice versa using firebase as an intermediary.

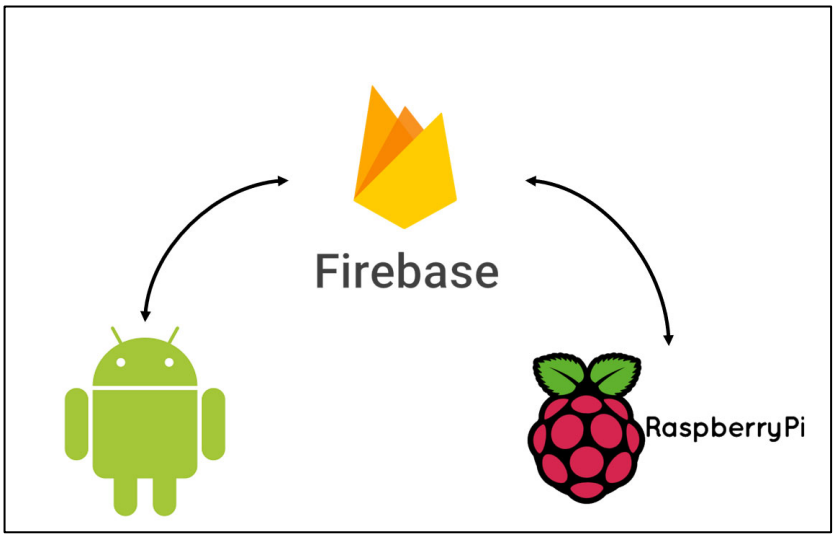


Figure 12 - Project Diagram.

In order to complete this I had to find a python library that would allow the Raspberry Pi to communicate with firebase.

The only python library that I could find at this time that would allow this, is a library called “Pyrebase”.

I evaluated the Pyrebase library using the following criteria:

- Authentication.
- Storage.
- Database (Create Read, Update, Delete).

5.1.1 Firebase Authentication using Pyrebase.

Firebase Authentication provides backend services, ready-made UI libraries and easy-to-use SDKs to authenticate users to an application without using server-side code. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter.

Firebase Authentication integrates tightly with other Firebase services, and it implements industry standards like OAuth 2.0 and OpenID Connect, so it can be easily integrated with a custom backend.

The “sign_in_with_email_and_password ()” method will return user data including a token you can use to adhere to security rules.

A developer can use this code for creating of users in a firebase console.

```python
#Pyrebase Authentication
import pyrebase

firebaseConfig = {
[redacted]
};

firebase = pyrebase.initialize_app(firebaseConfig)

auth=firebase.auth()

#Signup
email = input("Enter your Email: ")
password = input("Enter Your Password: ")
confirmPassword = input("Confirm Your Password: ")
if password == confirmPassword:
    try:
        auth.create_user_with_email_and_password(email, password)
        print("Succesfully signed Up")
    except Exception as e:
        print(e)
```

Figure 13 - Firebase Sign Up Authentication in Python.

```
PS C:\Users\tjfitzpatrick\Desktop\P Handbook\Programs\Proof of Concept Programs\Python\PyrebasePrograms>  c:;
s'; & 'C:\Users\tjfitzpatrick\AppData\Local\Microsoft\WindowsApps\python3.10.exe' 'c:\Users\tjfitzpatrick\.vs
s\tjfitzpatrick\Desktop\P Handbook\Programs\Proof of Concept Programs\Python\PyrebasePrograms\Authentication_
Enter your Email: tjfitzster@gmail.com
Enter Your Password: Limerick1
Confirm Your Password: Limerick1
Succesfully signed Up
PS C:\Users\tjfitzpatrick\Desktop\P Handbook\Programs\Proof of Concept Programs\Python\PyrebasePrograms>
```

Figure 14 - Evaluating Firebase Sign Up Authentication in Python.

```python
#Pyrebase Authentication
import pyrebase

firebaseConfig = {
[redacted]
};

firebase = pyrebase.initialize_app(firebaseConfig)

auth=firebase.auth()

"""
#Login
"""
email = input("Enter your Email: ")
password = input("Enter Your Password: ")
try:
    auth.sign_in_with_email_and_password(email, password)
    print("Succesfully Signed In")
except Exception as e:
    print(e.message)
```

Figure 15 - Firebase Login Authentication in Python.

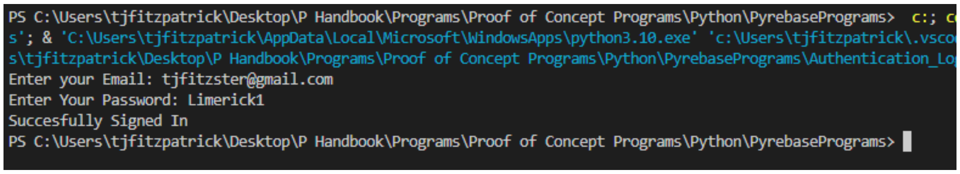


Figure 16 - Evaluating Firebase Login Authentication in Python.

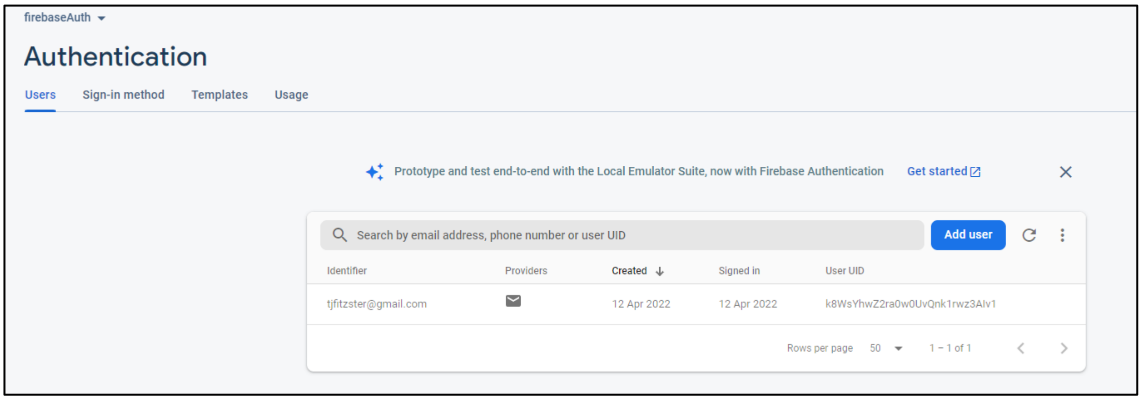


Figure 17 - Verification that Signup and Login Authentication functioned Correctly.

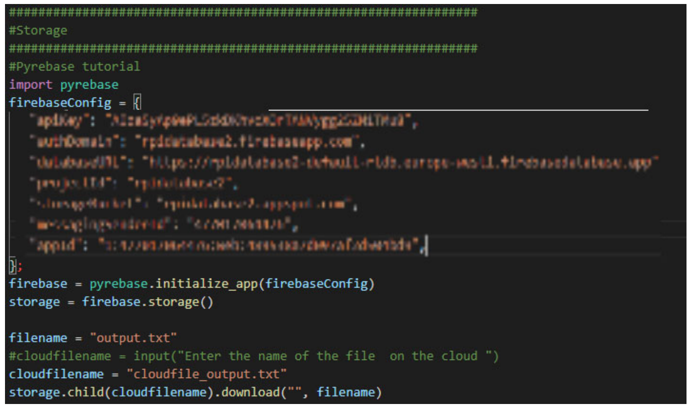
Once I verified Authentication, I then moved onto evaluating storage on the database, how I did this was I sent text files up to Firebase.

5.1.2 Firebase Storage using Pyrebase.

Cloud Storage for Firebase is a powerful, simple, and cost-effective object storage service built for Google scale. The Firebase SDKs for Cloud Storage add Google security to file uploads and downloads for Firebase apps, regardless of network quality.

Developers can use SDKs to store images, audio, video, or other user-generated content. In the below example and for proof of concept I used python to upload and download a simple text file.

Figure 18 - Uploading a file to the Firebase Cloud Server.



```
####################################################################
#Storage
####################################################################
#Pyrebase tutorial
import pyrebase
firebaseConfig = {
  [redacted]
};
firebase = pyrebase.initialize_app(firebaseConfig)
storage = firebase.storage()

filename = "output.txt"
#cloudfilename = input("Enter the name of the file  on the cloud ")
cloudfilename = "cloudfile_output.txt"
storage.child(cloudfilename).download("", filename)
```

Figure 19 - Downloading a File from the Cloud Database

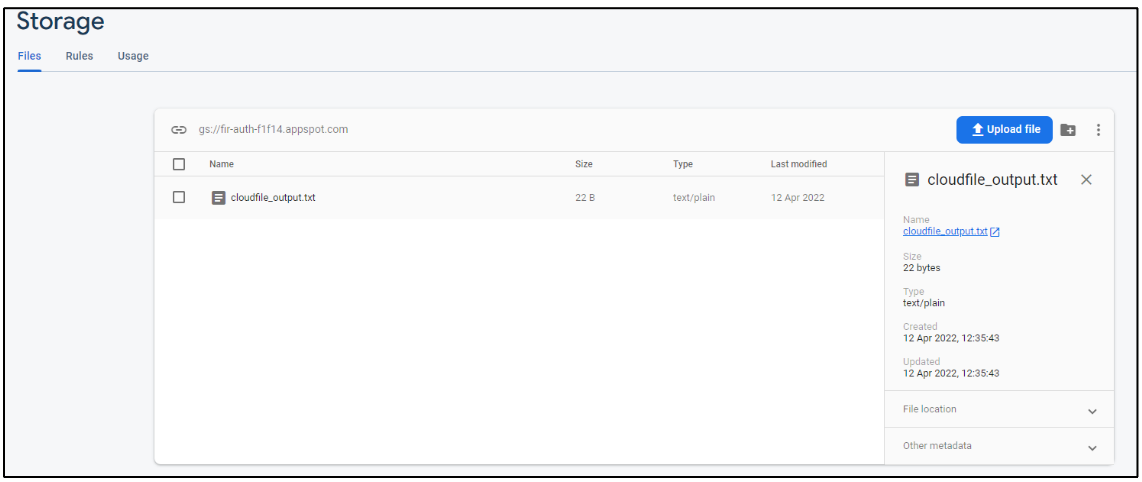


Figure 20 - Verification that Cloud Upload functions Correctly.

5.1.3 Firebase Database (Create Read, Update, Delete) using Pyrebase.

The last part of this proof of concept was implementing a program that would create read update and delete data on a Firebase Database.
The Firebase Realtime Database is a cloud-hosted database. Data is stored as a JSON document and synchronized in real time to every connected client. When a developer builds cross-platform apps or web applications, all of their clients share one Realtime Database instance and automatically receive updates with the newest data.
Documents are designed based on Key Value pair, so when a developer has access to a key, a value can be read back using it.

In the below example, I am creating a user, I am adding this user to a “Users” document,

I then am reading all users in the “Users” document looking for the username “TJFITZSTER” once I find it, I print up a confirmation message. I then delete another user based on user ID. Document IDs are automatically generated in Firebase and are completely unique.

```
################################################################
#Database
################################################################
#create
data ={"age": "50","firstName": "Jim York","lastName": "true", "userName": "Jim Smith"}
#db.push(data)
db.child("Users").push(data)
#db.child("Users").child("TomTom").set(data)

#update
#db.child("Users").child("TomTom").update({"age": "60"})
people = db.child("Users").get()

for person in people.each():
    #print(person.val())
   # print(person.key())
   if person.val()["userName"] == "TJFITZSTER":
       print("We found you TJ")
       #db.child("Users").child(person.key()).update({"name":"Jane"})
db.child("users").child("-LzqIcMVMPaQKVLLjK5d").remove()
```

Figure 21 - Example CRUD of Firebase instance.

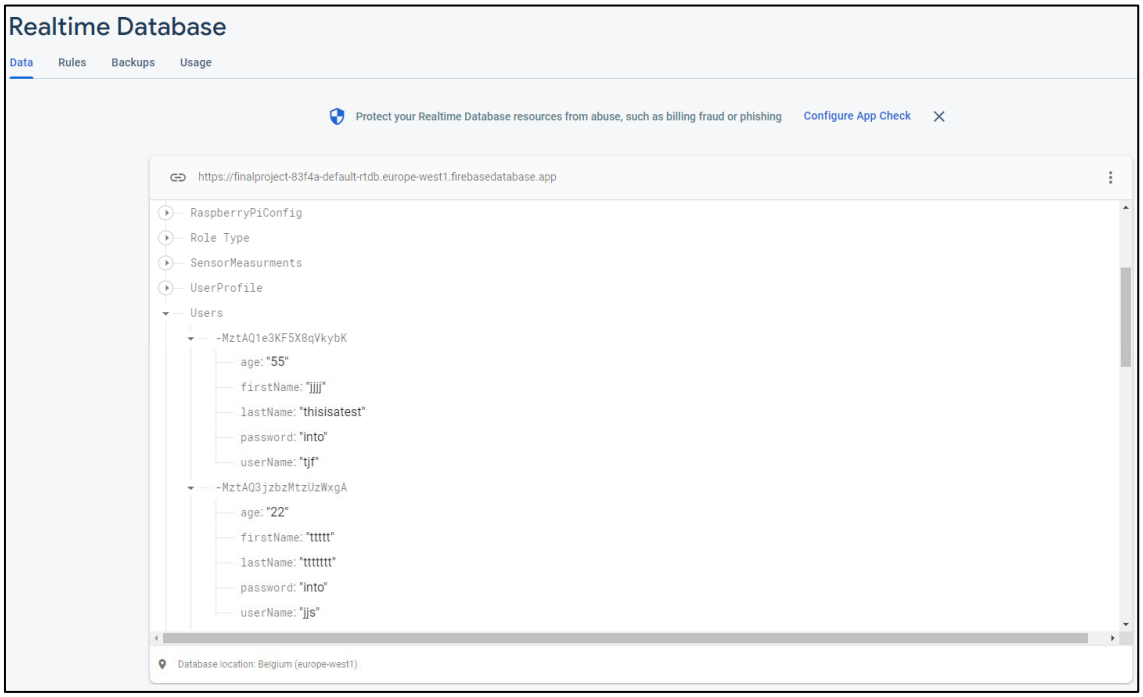


Figure 22 - Documents on Firebase Realtime Database.

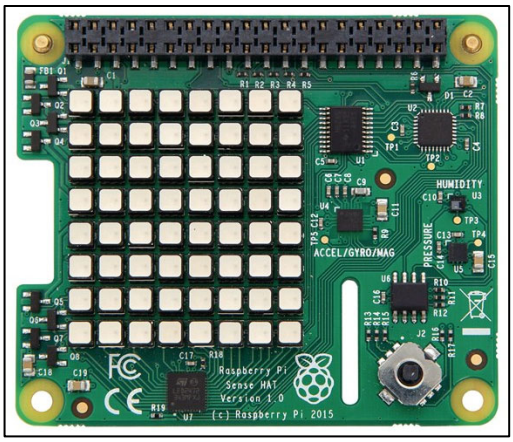
Once the proof-of-concept idea was complete I could now connect to the database using Python running on the Raspberry Pi the next step was to build onto the sensors part of this project.

5.2 Python on Raspberry Pi.

Using the Raspberry Pi's Sensehat I wrote very simple programs, these programs allowed me to evaluate the potential of the Pi and also allowed me to familiarise myself with the Python language.

5.2.1 Sense Hat's Pressure Temperature and Humidity.

This first program I wrote on this project was implementing Raspberry Pi's Sensehat, the Sensehat is a plugin module that allows the Raspberry Pi to read different sensors. This program was designed to read the temperature, humidity and air pressure and print it up to the terminal.



(raspberrypi.dk, 2022)

Figure 23 - Raspberry Pi's Sensehat

```
C: > Users > tjfitzpatrick > Desktop > P Handbook > Programs > Proof of Concept Programs > Python > hunidity.py > ...
1   from sense_hat import SenseHat
2
3   sense = SenseHat()
4   sense.clear()
5
6   humidity = sense.get_humidity()
7   temp = sense.get_temperature()
8   pressure = sense.get_pressure()
9   print(pressure)
10  print(humidity)
11  print(temp)
```

Figure 24 - Pressure Temperature and Humidity Program.

5.2.2 DHT11 & DHT22.

Interfacing the DHT11 and DHT22 to the Raspberry Pi was more difficult than the SenseHat. Researching these devices led me to discover they use an interface called "One Wire", so there are 3 pins on each DHT device, Power Pin, Ground Pin and a Signal pin. The signal pin can be connected to any pin on the Raspberry Pi provided the Raspberry Pi is programmed correctly and used the correct libraries.

The biggest downside to the DHT11 and DHT22 sensors is that they are quite slow sensors. They have a sampling rate of once every second for the DHT11 and once every 2 seconds for the DHT22.
The circuit is as follows:

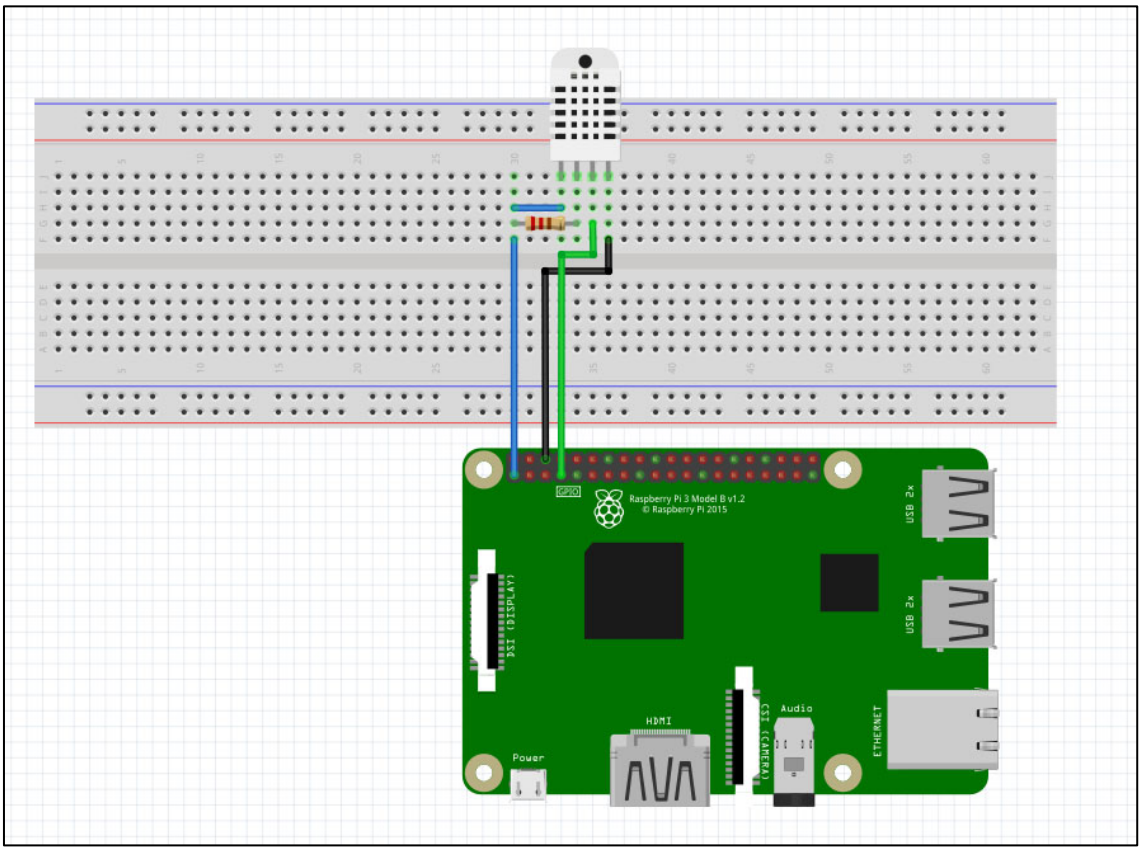


Figure 25 - DHT22 Circuit Diagram.

When I executed the program there were errors in reading the sensors, so I put a clause that the program was not to print the temperature or humidity id there were a read error.

```
import Adafruit_DHT

DHT_SENSOR = Adafruit_DHT.DHT22
DHT_PIN = 4

while True:
    humidity, temperature = Adafruit_DHT.read_retry(DHT_SENSOR, DHT_PIN)

    if humidity is not None and temperature is not None:
        print("Temp={0:0.1f}*C  Humidity={1:0.1f}%".format(temperature, humidity))
    else:
        print("Failed to retrieve data from humidity sensor")
```

Figure 26 - DHT 22 Proof of Concept Program.

```
Current Temprature is: 11.178527849347283 *C
Current Humidity is: 82.20193515187538 %

Current Temprature is: 11.316456691044898 *C
Current Humidity is: 82.39909884691939 %

Current Temprature is: 11.858347057983554 *C
Current Humidity is: 82.6517177388197 %

Current Temprature is: 11.711341669238633 *C
Current Humidity is: 82.69386687563612 %

Current Temprature is: 11.510431542171952 *C
Current Humidity is: 82.15585352631764 %

Current Temprature is: 11.99499716816878 *C
Current Humidity is: 82.44478779159768 %

Current Temprature is: 11.6055020951682 *C
Current Humidity is: 82.01271759482276 %

Current Temprature is: 11.048986789737976 *C
Current Humidity is: 82.12437259963296 %
```

Figure 27 - Proof of Concept Temperature and Humidity.

5.2.3 Soil Moisture Sensor.

There are 2 ways in which the soil moisture sensor can be wired to the Raspberry Pi, the first way is the most accurate, but the trade-off is there are more hardware and more code to be, but the user would get an analogue reading rather than a digital one.

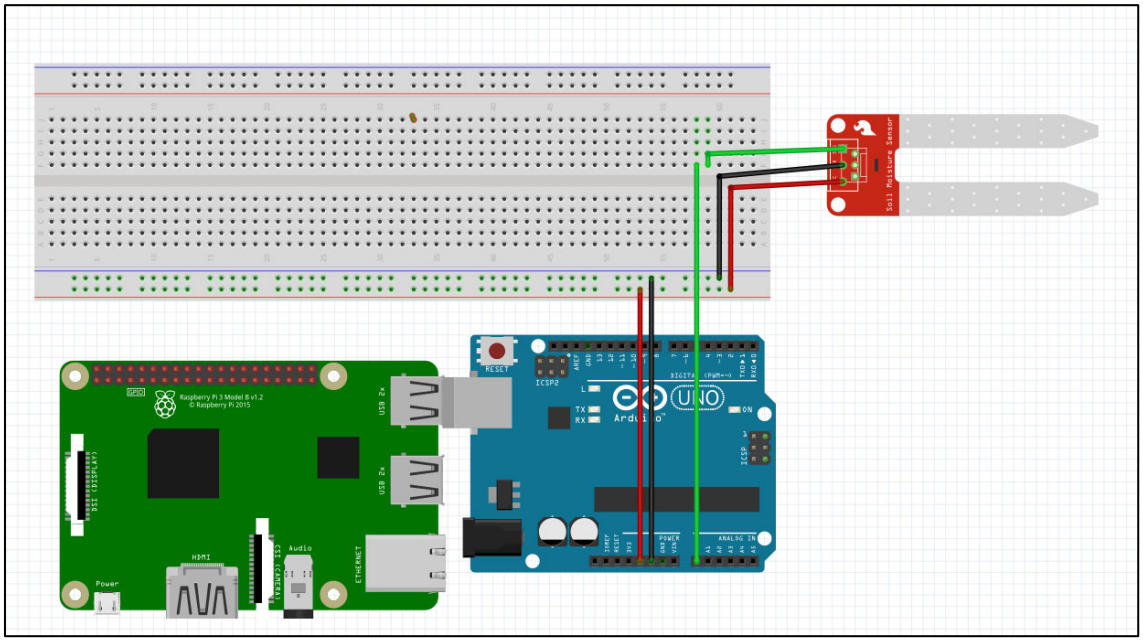


Figure 28 - A Circuit Diagram to Read Soil Moisture Using an Arduino.

The second way is to use a converter board, where I set the threshold manually and if the moisture in the soil goes past this threshold value it sends the Raspberry Pi a signal saying this value is passed. The measurement is correct, but we only use the digital measurement which is on or off this is not as accurate. The trade-off is there is less circuitry and less wiring.

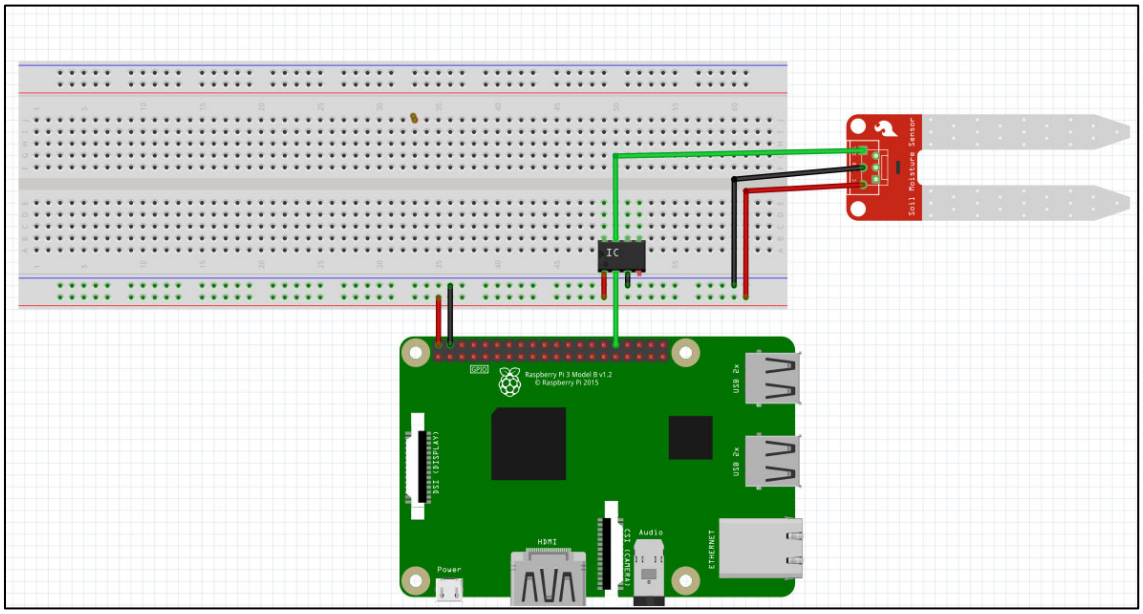


Figure 29 - A Circuit Diagram to Read Soil Moisture Using a Signal Converter.

5.2.4 MQ-135 Sensor.

There are 2 ways in which the air quality sensor can be wired to the Raspberry Pi, the first way is the most accurate but again the trade-off is there is more hardware and more code to be, but the user would get an analogue reading rather than a digital one.

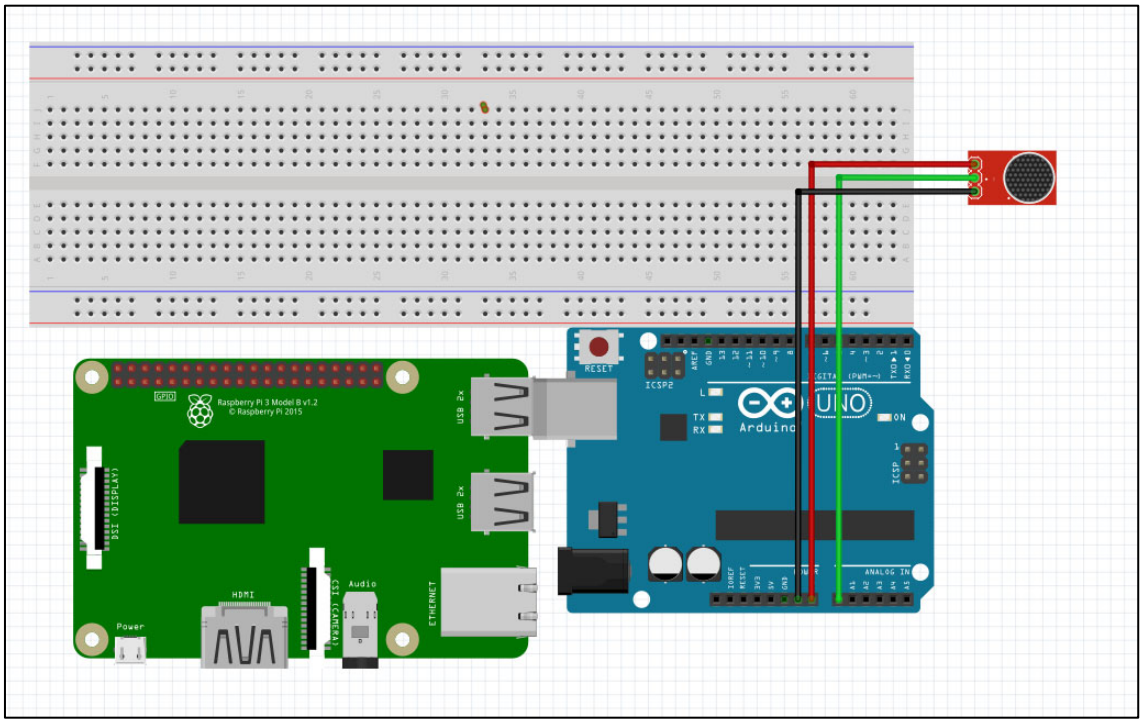


Figure 30 - A Circuit Diagram To read air quality using an Arduino.

The second way is to use a converter board, where I set the threshold manually and if the moisture in the soil goes past this threshold value it sends the raspberry pi a signal saying this value is passed. The measurement is correct, but we only use the digital measurement which is on or off this is not as accurate. The trade-off is there is less circuitry and less wiring.

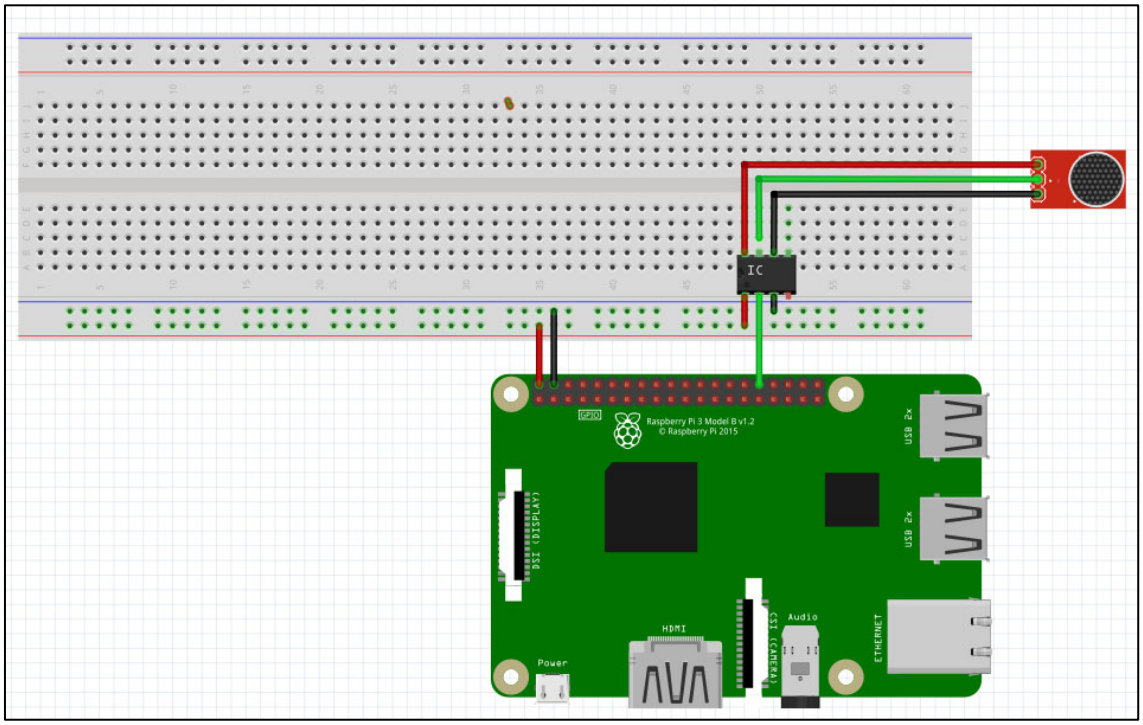


Figure 31- A Circuit Diagram to Read air Quality Using a Signal Converter.

5.2.5 Water Pump.

The last device I had to interface was a water pump. A water pump can be connected to any digital pin on the Raspberry Pi, once the pin's state on the Raspberry Pi is set to a high state the pump will turn on, and vice versa when the pump is connected to a pin that is set to a low state.

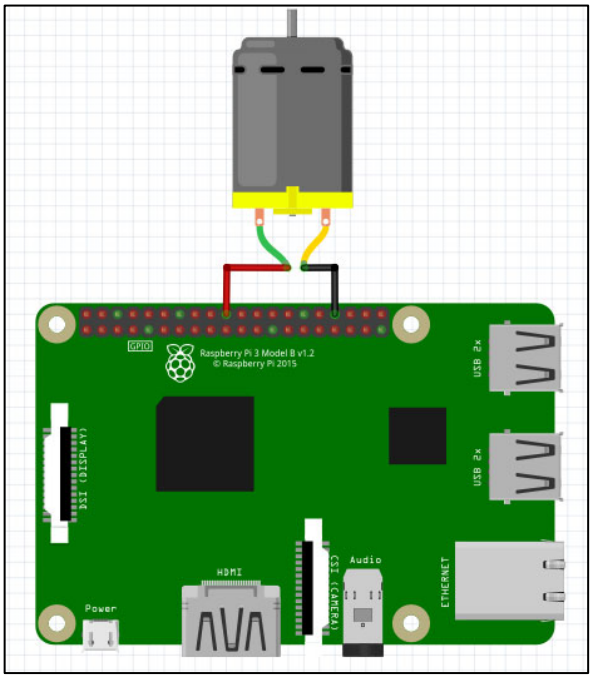


Figure 32 - Raspberry Pi Interfaced to A Water Pump

5.3 Kotlin on Android.

When researching the Kotlin program on Android I had to draw up some storyboards of what I thought the overall program would look like.

The following image is of what thought the landing page and login would look like:

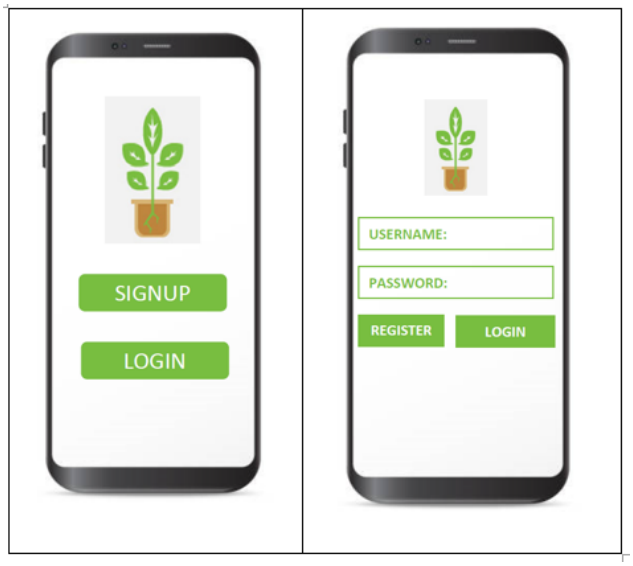


Figure 33 – Storyboard of landing page and Login page.

The next storey board is of a user trying to login:

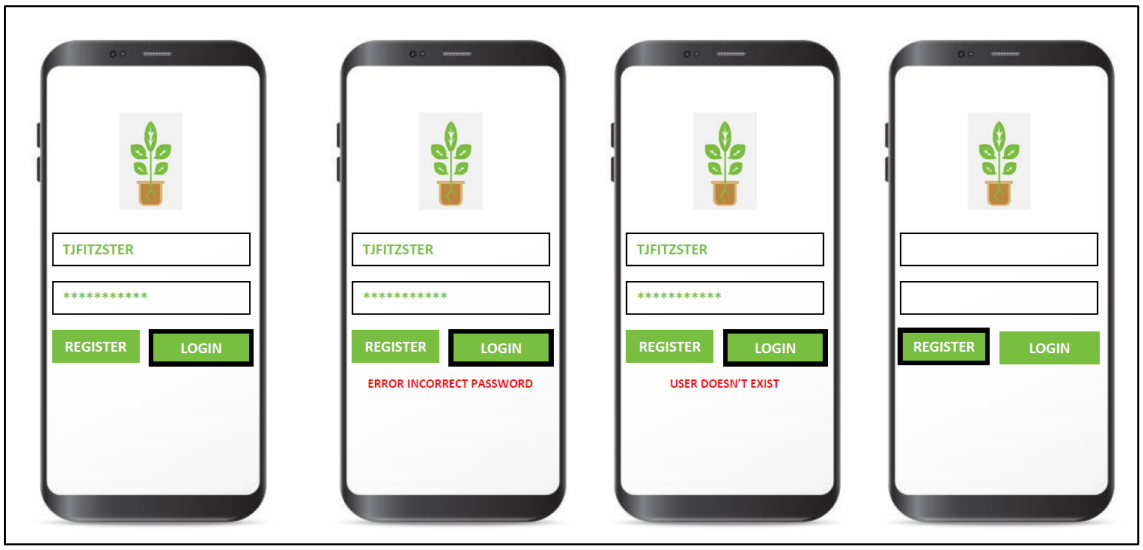


Figure 34 - Storyboard of user tying to login.

The next storyboard is of a user trying to register:

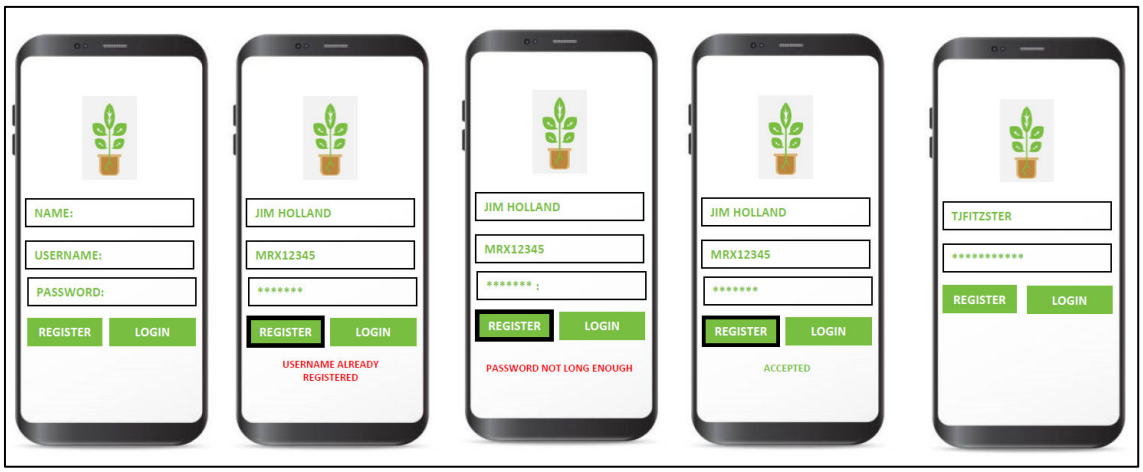


Figure 35 - Storyboard of user trying to register.

The next storey board is of the complete program all together including splash screen and design graphs.

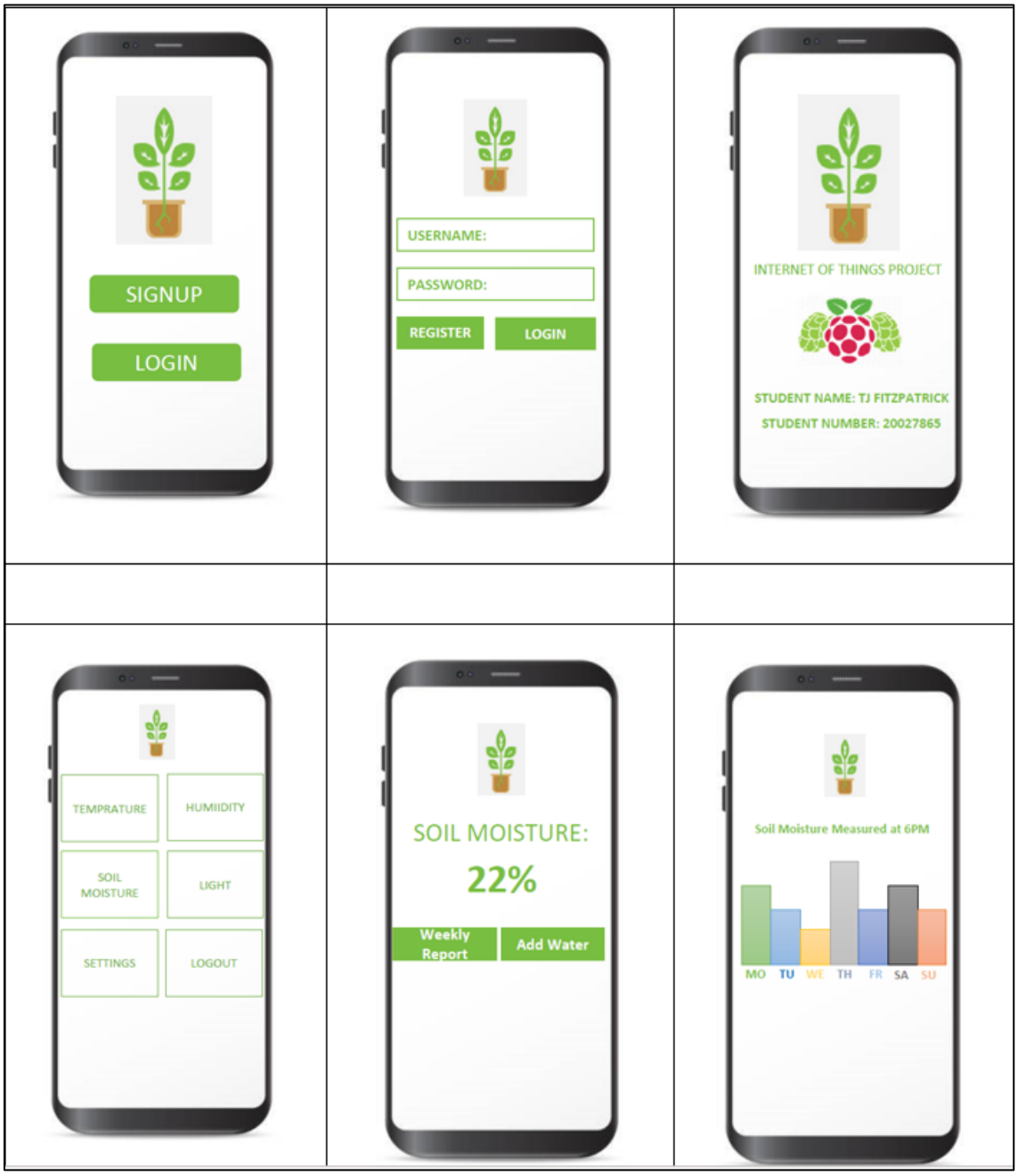


Figure 36 - Storyboard of The Proposed App.

Once I had an idea in my head of how I wanted the software to look I then went on to design the program in Android Studio.

5.4 Final Kotlin Program.

5.4.1 Splash Screen.

The following is a typical splash screen that is shown to the user and lasts 5 seconds before the user is brought to the login page.

Figure 37 - Application Splash screen

5.4.2 Login Screen.

The next screen the user is brought to is a login screen, I originally designed the login screen for the user to input an email, but ten changed my mind to username, the reason behind this is that I wanted to build more functionality for processing the username later in the code. There are 2 text fields that are username and password, there are three buttons, “Login”, “Register”, and “Forgot Password”. Should the user try login, the software performs some validation on the username and password. For example, the username and password cannot be empty, the username and password must match a username and password given in the database.

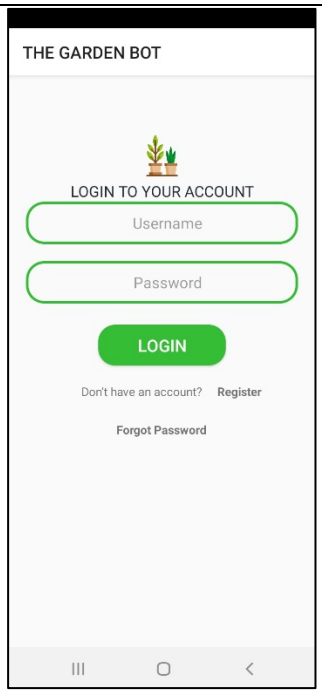


Figure 38 - Login Page.

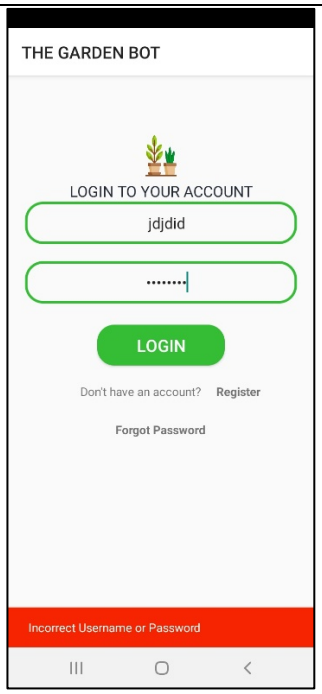


Figure 39 - Incorrect Username and Password

```
@RequiresApi(Build.VERSION_CODES.O)
override fun onClick(v: View?) {
    if (v != null) {
        when (v.id) {

            R.id.tv_forgot_password -> {
                // START
                // Launch the forgot password screen when the user clicks on the forgot password text.
                val intent = Intent( packageContext: this@LoginActivity, ForgotPasswordActivity::class.java)
                startActivity(intent)
                // END
            }

            R.id.btn_login -> {
                logInRegisteredUser()
            }

            R.id.tv_register -> {
                // Launch the register screen when the user clicks on the text.
                val intent = Intent( packageContext: this@LoginActivity, RegisterActivity::class.java)
               startActivity(intent)
            }
        }
    }
}
```

Figure 40 - Snapshot of Main Loop at Login

```
 * A function to validate the login entries of a user.
 */
private fun validateLoginDetails(): Boolean {
    return when {
        TextUtils.isEmpty(et_login_username.text.toString().trim { it <= ' ' }) -> {
            showErrorSnackBar("Please enter email.", errorMessage: true)
            false
        }
        TextUtils.isEmpty(et_login_password.text.toString().trim { it <= ' ' }) -> {
            showErrorSnackBar("Please enter password.", errorMessage: true)
            false
        }
        else -> {
            true
        }
    }
```

Figure 41 - Snapshot of the validation of Login Details

```
@RequiresApi(Build.VERSION_CODES.O)
private fun readData(userName: String, password: String) {

    database = FirebaseDatabase.getInstance().getReference( path: "Users")
    database.child(userName).get().addOnSuccessListener {  it: DataSnapshot!

        if (it.exists()){

            val readusername = it.child( path: "userName").value
            val id = it.child( path: "uid").value
            val readpassword = it.child( path: "password").value

                if((readusername?.equals(userName) == true) &&  (readpassword?.equals(password) == true)){
                    showErrorSnackBar( message: "User Logged In",  errorMessage: false)
                    hideProgressDialog()

                    updateLoginTable(id as String, userName)
                    intent.flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
                    val intent = Intent( packageContext: this@LoginActivity, MainActivity::class.java)
                    intent.putExtra( name: "Username", userName)
                    startActivity(intent)
                }
            else{
                    hideProgressDialog()
                    showErrorSnackBar( message: "Incorrect Username or Password",  errorMessage: true)


            }
        }else{

            showErrorSnackBar( message: "Incorrect Username or Password",  errorMessage: true)
            hideProgressDialog()
        }
    }.addOnFailureListener{  it: Exception
        showErrorSnackBar( message: "Incorrect Username or password",  errorMessage: true)
    }
```

Figure 42 - Snapshot of Code Validating Login Data against the Database.

5.4.2 Register Screen.

Whilst registering a user there is also some type of validation performed which the username has to be unique, the user must accept the terms and conditions and the user must enter the exact same password twice.

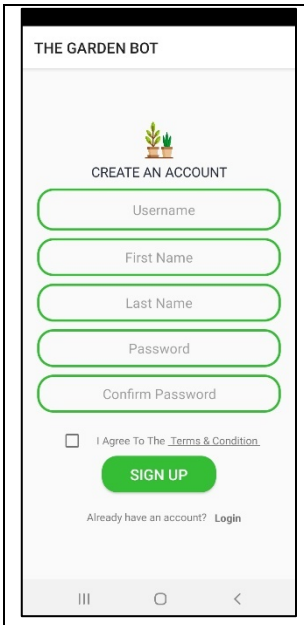


Figure 43 - Registration Screen.

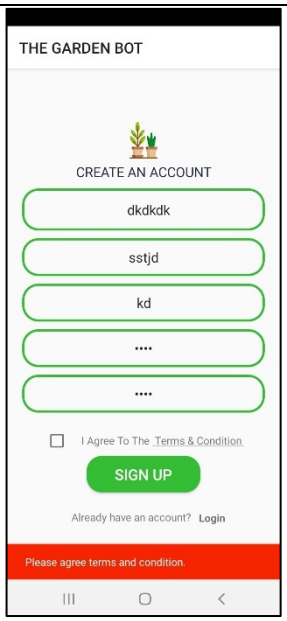


Figure 44 - Terms and Conditions Have Not Been Accepted.

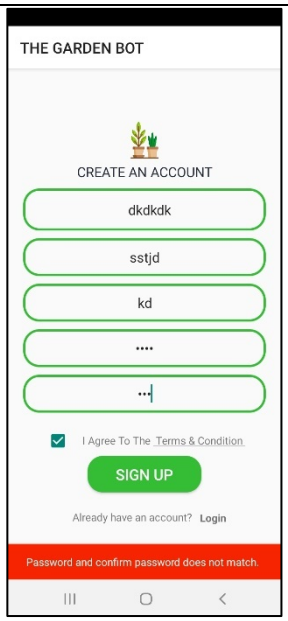


Figure 45 - Passwords Do Not Match

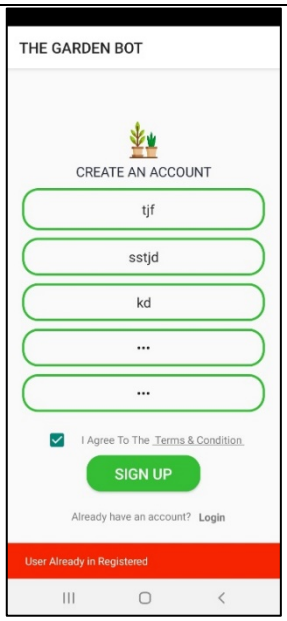


Figure 46 - User Already Registered.

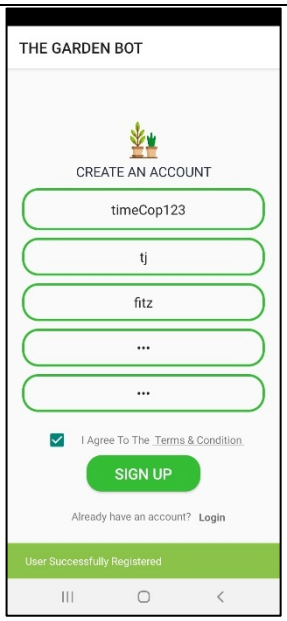


Figure 47 - User Successfully Registered.

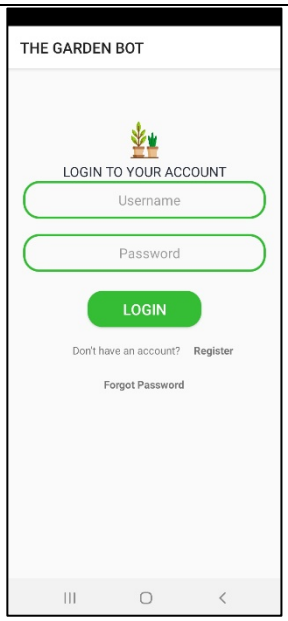


Figure 48 - User Gets Redirected to the login page.

Once the use has registered successfully a document will get created on the Realtime Database under the user's username.

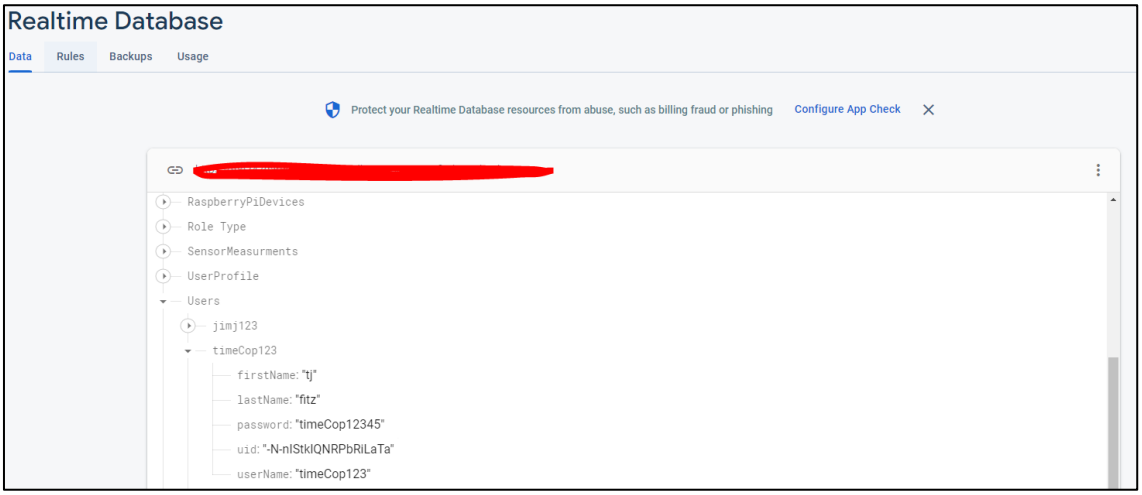


Figure 49 - User Being Created on Database.

5.4.3 Forgot Password Screen.

This screen was originally designed for when we were using google authentication against email addresses. There is an inbuilt function that a user can from firebase, that we can call and pass it an email address, if that email address is registered with firebase, a reset email would be sent.

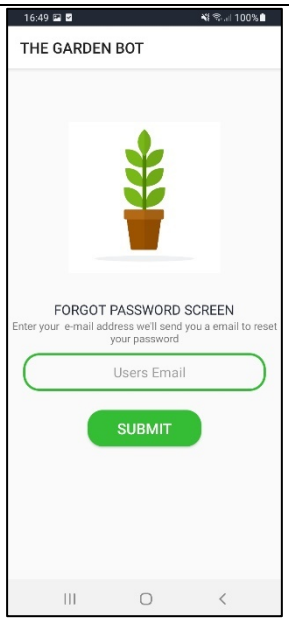


Figure 50 - Forgot Password Screen.

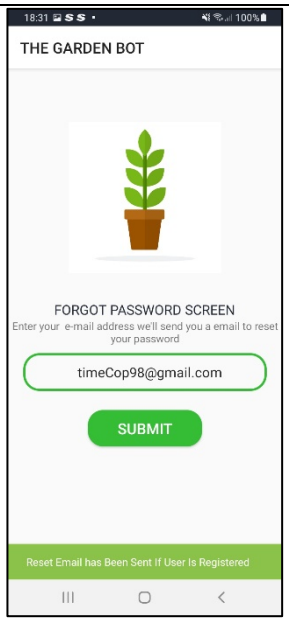


Figure 51 - Resetting the Email.

5.4.5 Main Menu Screen.

I designed many different versions on this screen, but upon consulting with my project supervisor we decided on a design. In the design there are 8 buttons, there are 3 on the menu bar and 5 on linear list. The menu buttons are “Logout”, “User Profile”, and “Settings”. The logout button log’s the user out of the device and re-directs the user to the login page. The settings button brings the user to the settings on the Raspberry Pi and the user profile button brings the user to their profile details.

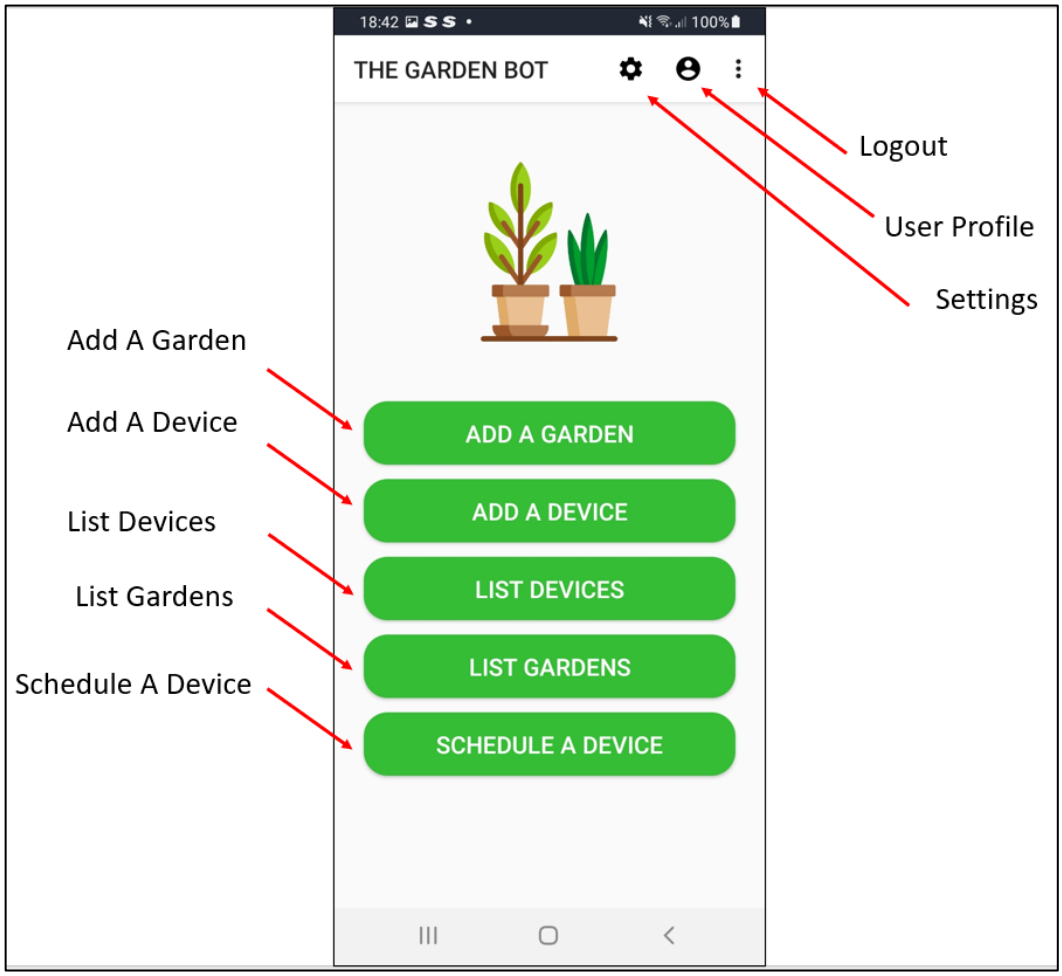


Figure 52 -24/7 Gardiner Main Menu Screen

The next set of buttons are all associated with the Raspberry Pi.

- Add a Garden: Created a Garden associated with the user.
- Add a Device: Allows the user to add a device to a garden.
- List Devices: List all devices measurements.
- List Gardens: List all Gardens.
- Schedule A Device: Schedule a device to take a sample or turn on.

5.4.6 User Profile.

In the user's profile there are 3 text fields, username, which is un-editable, first name and second name which are editable, the reason this is done like this is because I wanted to implement the read and update functions of a CRUD. The user can edit first name and second name and once they press save, they can update the database.

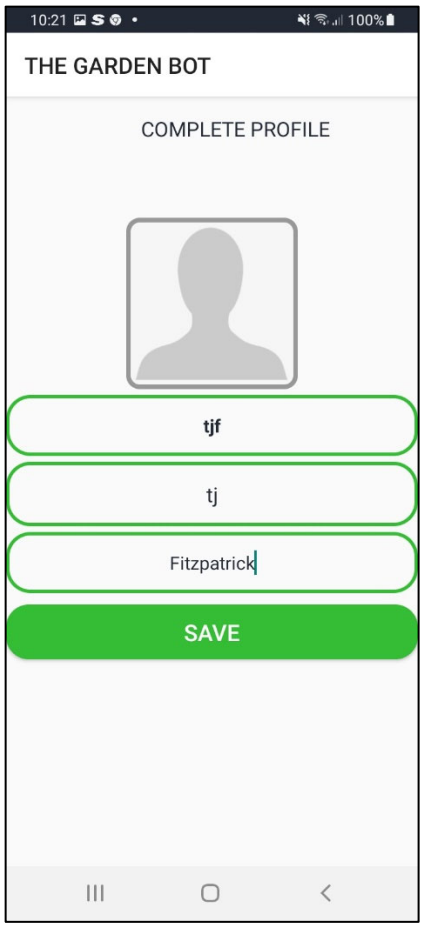


Figure 53 - Users Profile.

5.4.6 Settings.

The option is for the user to control some settings on the Raspberry Pi, the text box is a number only text box that allows the user to set the measurement frequency in seconds.

Then there are 3 more options, the execute program option allows the use to choose to allow the raspberry pi to continue program execution or to shut it down, by default its set to execute program.

Figure 54 - Device Configuration.

The next is button is for the raspberry pi to upload a configuration file from the Raspberry Pi's end to the firebase server. The last one allows the Pi to write a configuration file in case the raspberry pi ever loses its connection to the database.

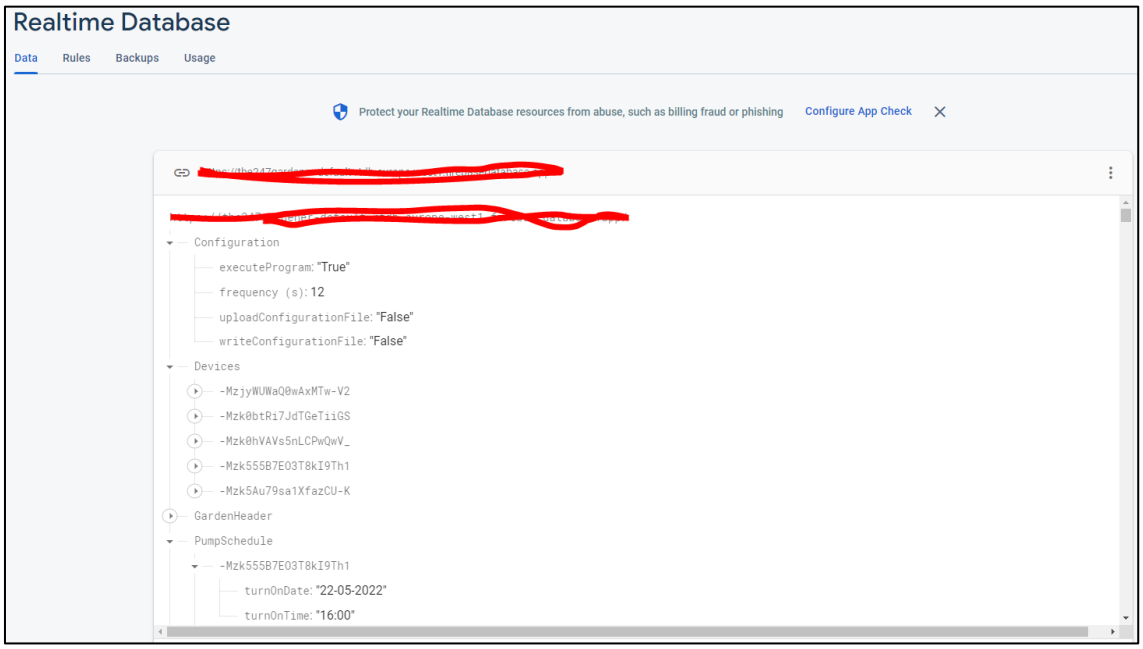


Figure 55 - Device Configuration on The Realtime Database.

5.4.6 Add a Garden.

The is basically the same as registering a user the user can add a garden to their account.

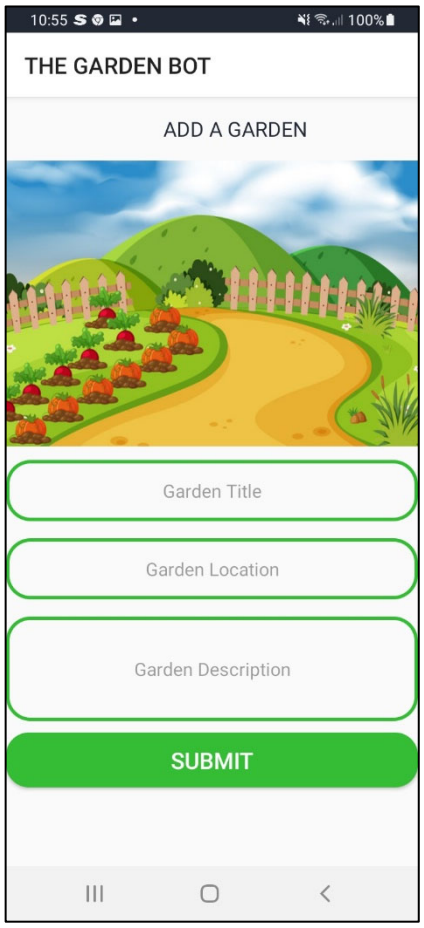


Figure 56 - Add a Garden Screen

5.4.7 Add a Device.

This screen allows the user to add a device to a garden via its garden id the user can name the device, they can set the service time in months, they tell the database which pin the raspberry pi has to read this device from and finally they set the device type.

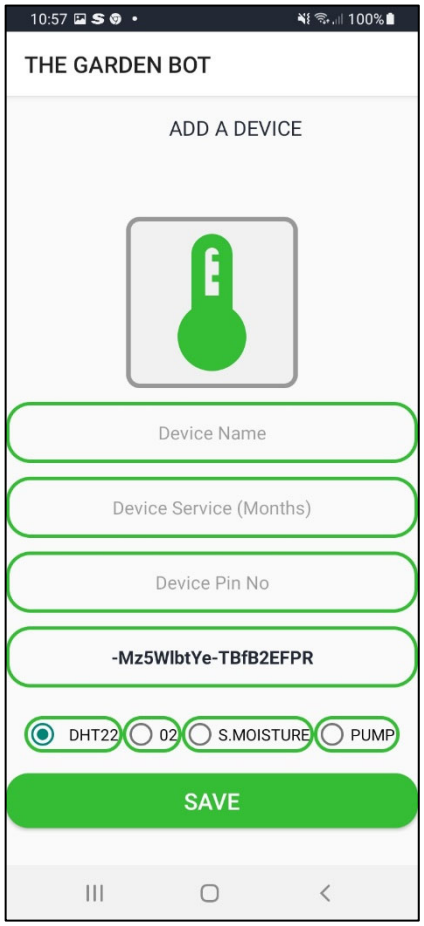


Figure 57 Add A Device Screen

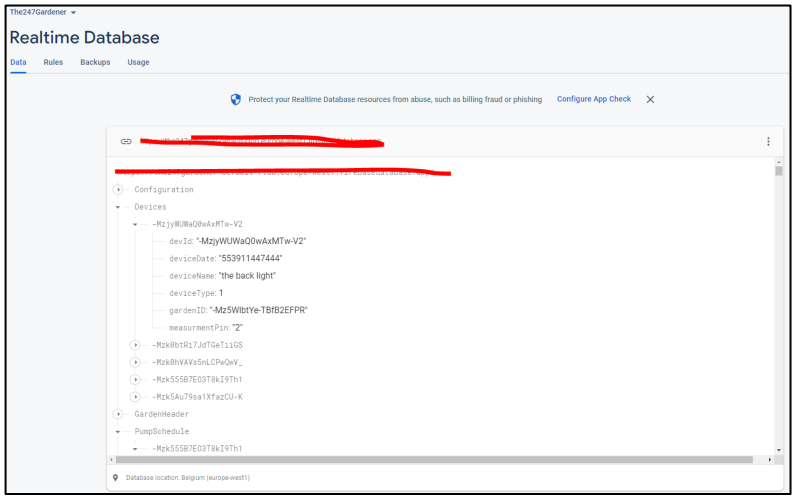


Figure 58 - Device Being Created on the Real Time Database

5.4.8 List Devices.

This screen just lists all devices measurements that were measured on the Raspberry Pi, listing the latest measurement first its displayed to the user using a recycler view.

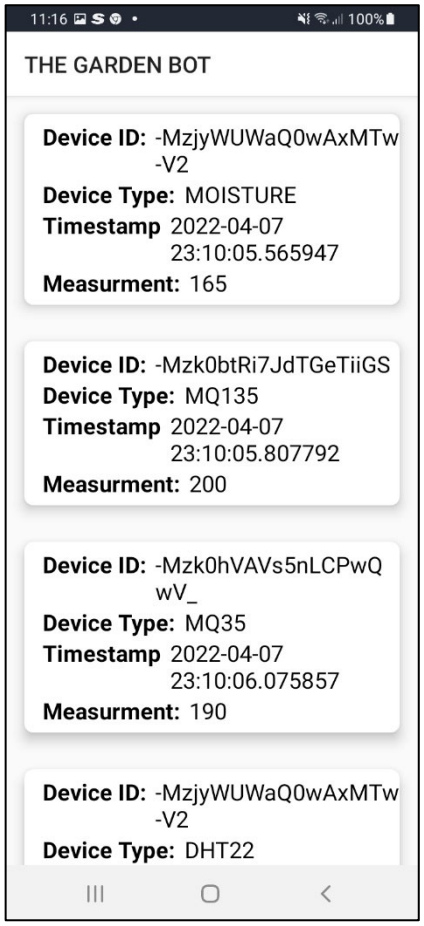


Figure 59 - Device Measurements being Displayed to the User.

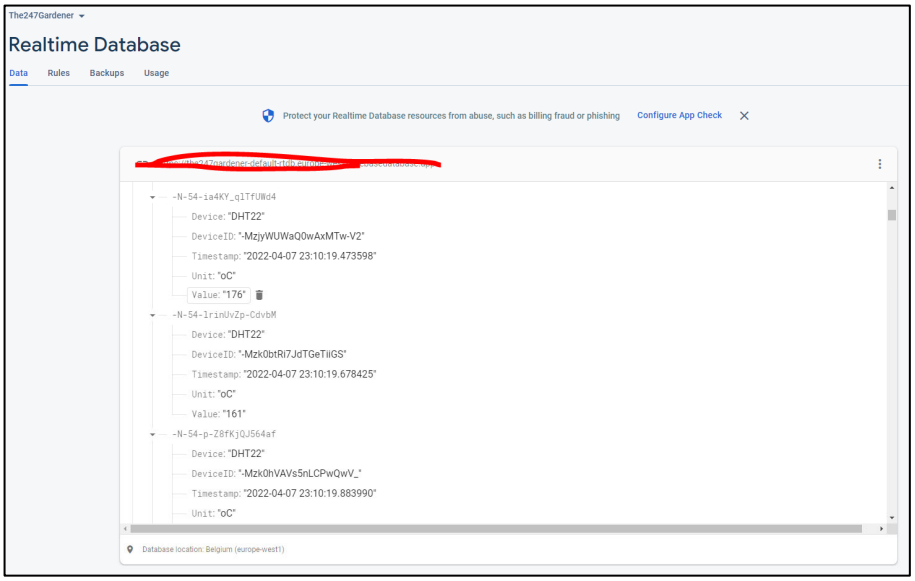


Figure 60- Device Measurements on The Database

5.4.9 List Gardens.

This is one of the later screens that was in development basically for every garden that ever was created for a user on the ER diagram there would be a header and a details table, once the user clicked the view details button all the garden details would appear, but it is unfinished.

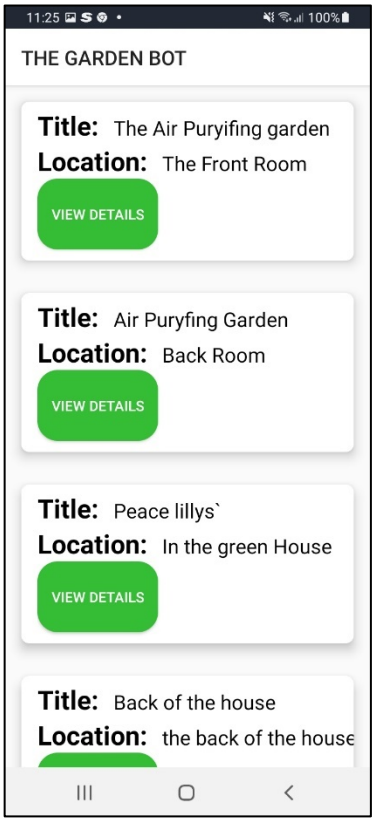


Figure 61 - List Gardens Screen.

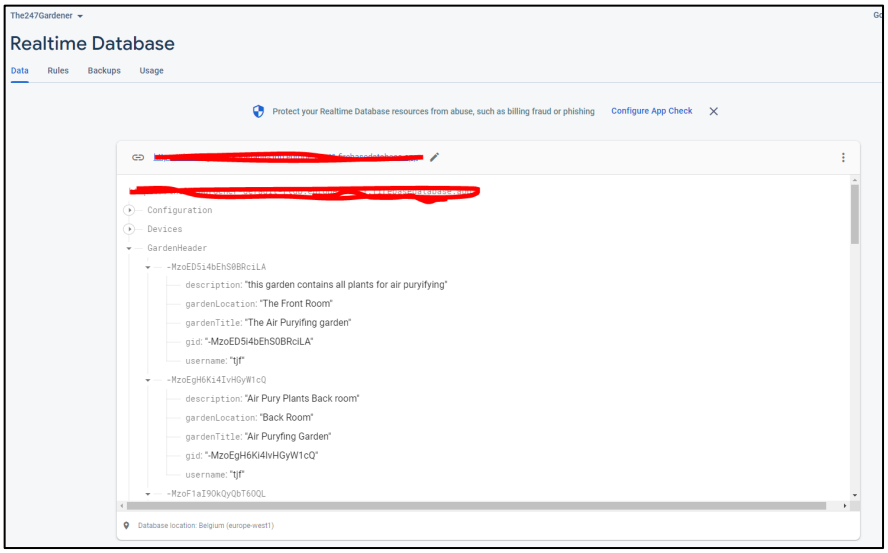


Figure 62 - Database Garden Header Documents

5.4.10 Schedule A Device.

Figure 63 - Scheduling A Device Pump To Turn On

This screen consists of a device ID field, a date picker and a 24 hour time picker once the user clicks save, A schedule is made in the Database schedule table for the device to turn on.

5.4.11 App Icon.

The last piece of design work completed on the Android side of things in this project was the application icon.

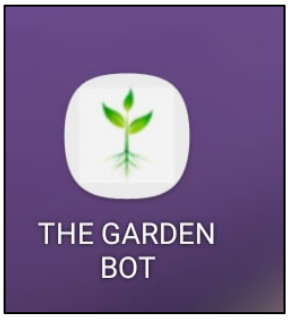


Figure 64 - Android Application Icon.

5.5 Final Python Program.

Once the android program was written the next step was to write the python program.

5.5.1 Main Flow Diagram.

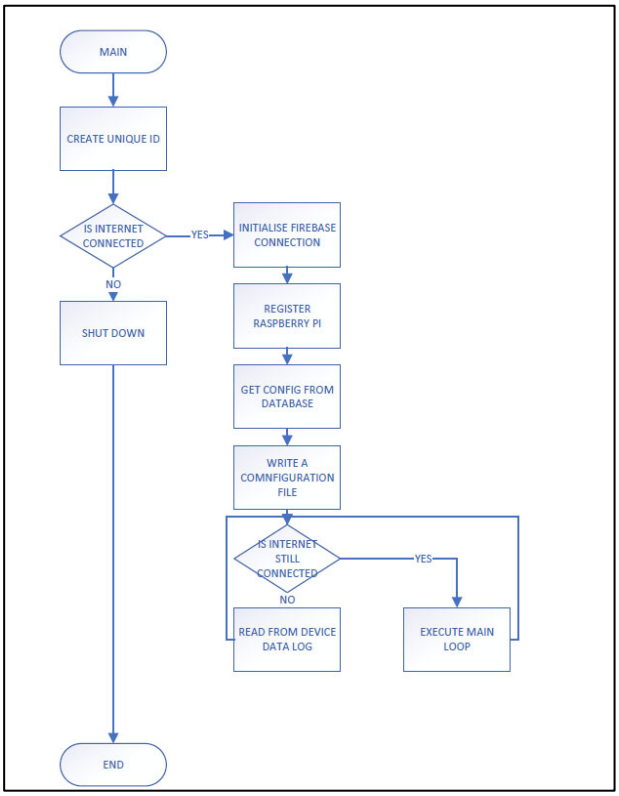


Figure 65 - Main Raspberry Pi Flow Diagram

Coming near the end of the project I was researching a way to generate a unique ID for the Raspberry Pi; the idea was that maybe one user can select which pi the android application connects to and for this task the library that I was researching was called UUID and it's based on RFC 4122. RFC 4122 is a specification that defines a Uniform Resource Name for UUIDs (Universally Unique IDentifier), also known as GUIDs (Globally Unique IDentifier). (P. Leach, 2022)

It was decided to keep things simple at this stage of the project, so the function within the UUD library I used was UUID1, UUID1 creates an ID using device Mac Address, and the current time the address is created the benefits of this include there is no need for a registration process, and I can use the UUID generated as a transaction ID.

```
##############
#   Main Program
##############
def main():
    uuidOne = uuid.uuid1()
    measurment_cycle_time = 0
    if(testInternet_connection()):
        db = firebaseInit()
        registerPiDevice(db, uuidOne)
        measurment_cycle_time = devConfig(db, "config")
        devConfig(db, "writeConfig")
        while(True):
            if (testInternet_connection()):
                uploadDevicessMeasurments(db)
                measurment_cycle_time = devConfig(db, "config")
                readDevices(db)
                exeProg = devConfig(db, "progVal")
                if(str(exeProg) == "False"):
                    devConfig(db, "writeConfig")
                    shutDown()
                time.sleep(measurment_cycle_time)
            else:
                readFromDeviceDatalog()
                time.sleep(measurment_cycle_time)

    else:
            shutDown()
            input("Exiting Program")
            exit()

if __name__ == "__main__":
    main()
```

Figure 66 - Main Function of Raspberry Pi Code

5.5.2 Internet Connection.

The next step was to test the internet connection.
The idea behind the testing of the internet connection was that if I ping the database endpoint, and if “OK” came back which is the response code that indicated a request was successful the database is connected and there is internet, but if anything, else came back there is an error with the internet.
If something out of the ordinary happens then there is an exception generated.

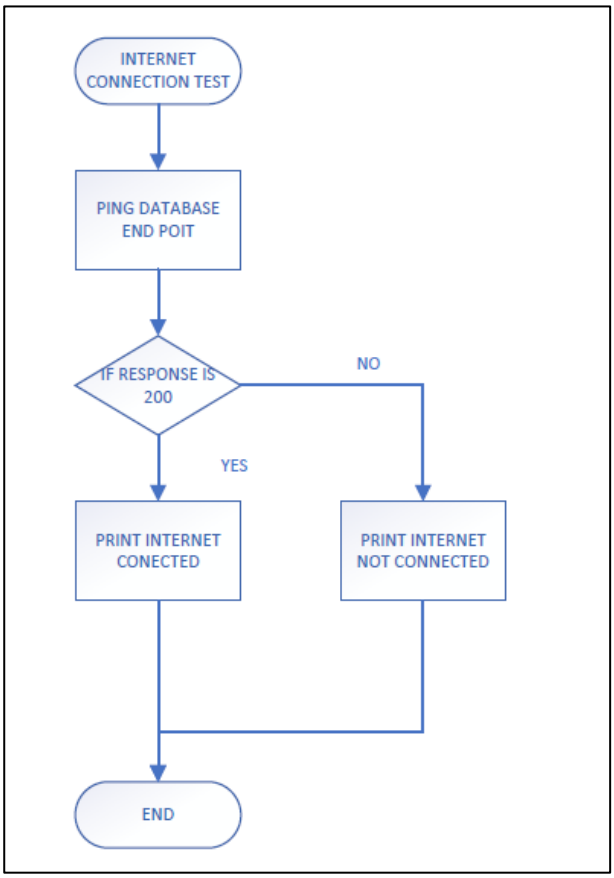


Figure 67 - Internet Connection Test Flow Diagram

```
def testInternet_connection():
    os.system('color')
    url = '[redacted]'
    timeout = 2
    op = None
    now = datetime.now()
    try:
        op = requests.get(url, timeout=timeout).status_code
        if op == 200:
            print(now, colored("Internet Connected", "green"))
            return True
        else:
            print(now, colored("Database is down", "red"))
            return False
    except:
        print(now, colored("No Internet Connected", "red"))
        return False
```

Figure 68 – Internet Test Connection Code

5.5.3 Firebase Initialisation.

The next step is to initialise the firebase database connection, this code is the same as the proof of concept.

```
def firebaseInit():

    firebaseConfig = {
[redacted]
};

    firebase = pyrebase.initialize_app(firebaseConfig)
    db=firebase.database()
    return db
```

Figure 69 - Initialising the Firebase Connection in Python.

5.5.4 Registering Raspberry Pi Device.

This piece of the code is basically uploading a Raspberry Pi device and ID to firebase. The code connects to the database and updates a document (Child) "RaspberryPiDevices".

```
def registerPiDevice(db, uuid):
    data = {"Timestamp": getTimestamp(),"RaspberryPi": str(uuid)}
    db.child("RaspberryPiDevices").push(data)
```

Figure 70 - Pushing Data to the Database.

Figure 71 - Verifying Data is in the database.

5.5.5 Reading the Measurement Cycle.

The Raspberry Pi reads the document Configuration, it reads the attribute “frequency (s)”

If the frequency is less than 4 seconds, then Raspberry Pi defaults to 4 seconds, the reason behind this number is that the DHT22 sensor only sends data every 2 seconds, if something goes wrong then the default frequency is 10 seconds.

```
if (configType == "config"):
    try:
        objfreq = db.child("Configuration").child("frequency (s)").get()
        freq = objfreq.val()
        if (freq < 4):
            freq = 4
        else:
            pass
    except:
        freq = 10
    return freq
```

Figure 72 - Reading an Attribute on the Configuration Document.

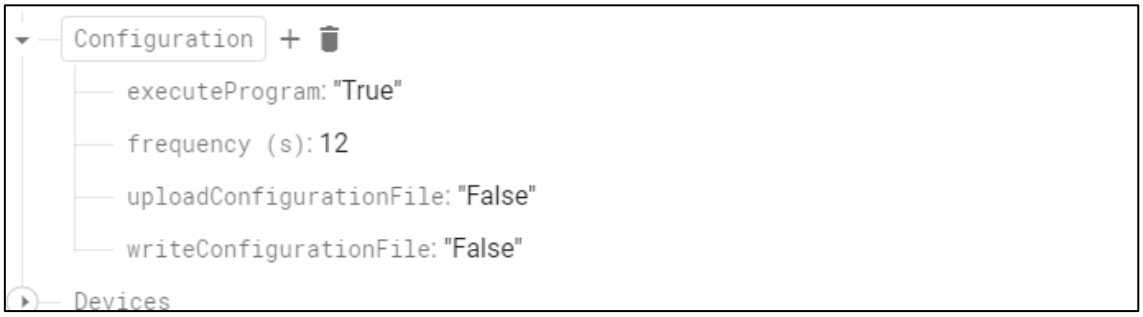


Figure 73 - Datable Configuration Document

5.5.6 Writing Configuration File.

The user of the Android application can instruct the Raspberry Pi to write its own configuration file, the reason this function is built in is that if the pi ever loses internet connection it can read a configuration from this file and take measurements.

So, it reads the Configuration Document is the “WriteConfigurationFile” flag is set to true then the Raspberry Pi will write its own configuration file.

```
elif(configType == "writeConfig"):
    try:
     wConfig = db.child("Configuration").child("writeConfigurationFile").get()
     wConfigVal = wConfig.val()
    except:
       wConfigVal = False
       print(datetime.now(), colored("Writing Config File Database Error Has Occured", "red"))
       return False

    if(wConfigVal == str(True)):
          print(datetime.now(), colored("Writing Config File", "green"))
          writeDeviceLog(db)
          return True
```

Figure 74 - Writing A Configuration File.

Figure 75 - Database Configuration Flags.

The configuration file is written using the devices document.
So, any device in the Devices document is written to a JSON file.

Figure 76 - Devices Firebase Document.

```python
def writeDeviceLog(db):
    devData = dict()

    allDevices = db.child("Devices").get()
    for device in allDevices.each():
        devData[device.key()] = device.val()

    json_object = json.dumps(devData, indent = 1)
    with open("deviceData.json", "w") as outfile:
        outfile.write(json_object)
```

Figure 77 – Devices Log File Code.

Figure 78 - Devices JSON File

5.5.7 Main Loop.

The Raspberry Pi is now inside the main loop of its code, where its constantly testing the internet connection, if the internet connection goes the raspberry pi can read a configuration from a configuration file and document the measurements file, when the internet returns it uploads all measurements to the server, reads the configuration document on the server, takes a measurement reading and checks that the user has not initiated a device shutdown.

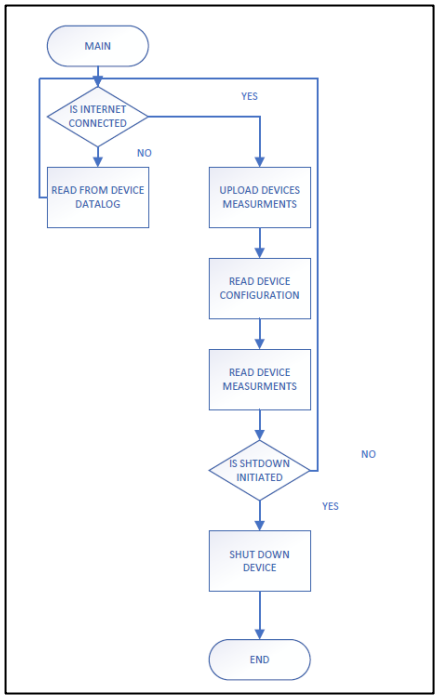


Figure 79 – Main Loop Flow Diagram.

```
#############
#   Main Program
#############
def main():
 uuidOne = uuid.uuid1()
 measurment_cycle_time = 0
 if(testInternet_connection()):
     db = firebaseInit()
     registerDevice(db, uuidOne)
     measurment_cycle_time = devConfig(db, "config")
        while(True):
            if (testInternet_connection()):
                uploadDevicessMeasurments(db)
                measurment_cycle_time = devConfig(db, "config")
                readDevices(db)
                exeProg = devConfig(db, "progVal")
                if(str(exeProg) == "False"):
                    devConfig(db, "writeConfig")
                    shutDown()
                time.sleep(measurment_cycle_time)
            else:
                readFromDeviceDatalog()
                time.sleep(measurment_cycle_time)

    else:
            shutDown()
            input("Exiting Program")
            exit()

if __name__ == "__main__":
    main()
```

Figure 80 - Main Loop Code.

5.5.8 Read from Device Data Log

This function gets executed when there is no internet connection. It opens the device data JSON file; it then loops through all devices in that JSON file and takes a measurement from the pin these devices are configured on and documents this measurement in a csv file to be uploaded when the internet connection is returned.

```
def readFromDeviceDatalog():
    try:
        with open('deviceData.json') as json_file:
            data = json.load(json_file)

            devId =""
            devicDate = ""
            deviceName = ""
            deviceType = ""
            gardenID = ""
            measurmentPin = ""

            for key in data:
                f_devId = data[key]
                for devAttr in f_devId:
                    if (devAttr == "devId"):
                        devId = str(f_devId[devAttr])
                    elif(devAttr == "deviceDate"):
                        devicDate = str(f_devId[devAttr])
                    elif(devAttr == "deviceName"):
                        deviceName = str(f_devId[devAttr])
                    elif(devAttr == "deviceType"):
                        deviceType = str(f_devId[devAttr])
                    elif(devAttr == "gardenID"):
                        gardenID = str(f_devId[devAttr])
                    elif(devAttr == "measurmentPin"):
                        measurmentPin = str(f_devId[devAttr])

                measureDeviceValue(devId, deviceType, measurmentPin)

    except:
        print(datetime.now(), colored("Error with config file", "red"))
```

Figure 81 - Read from Device Datalog Code.

```
def documentDeviceMeasurment(deviceName, devId, value):
    with open("deviceMeasurments.csv", "a") as f:
        if(deviceName == "1"):
            deviceName = "DHT22"
        elif(deviceName == "2"):
            deviceName = "MQ135"
        elif(deviceName == "3"):
            deviceName = "Moisture Sensor"
        elif(deviceName == "4"):
            deviceName = "Water Pump"
        f.write(str(deviceName) + "," + str(devId) + "," + getTimestamp() + "," +  "UNIT" + ","  + str(value) +  "\n")
    f.close()
```

Figure 82 - Writing CSV File of Device Measurement.

```
def measureDeviceValue(devId, deviceType, measurmentPin):

    if(deviceType == "1"):   #DHT22
        value = readDevice(deviceType, measurmentPin)
        documentDeviceMeasurment(deviceType, devId, value)
    elif(deviceType == "2"): # MQ135
        value = readDevice(deviceType, measurmentPin)
        documentDeviceMeasurment(deviceType, devId, value)
    elif(deviceType == "3"):  # Soil Moisture
        value = readDevice(deviceType, measurmentPin)
        documentDeviceMeasurment(deviceType, devId, value)
    elif(deviceType == "4"): #Water pump
        value = readDevice(deviceType, measurmentPin)
        documentDeviceMeasurment(deviceType, devId, value)
```

Figure 83 - Measure Device Value Code

5.5.10 Upload Device Measurements.

The next function only gets executed when there is internet, the program checks weather there is a csv file created, is there a file the program opens the file and checks the data in it, the program reads the data as a string, so it has to index the data anywhere where there is a carriage return, a blank file will always have one line in it, so the program checks if the length of the data is greater than 1, if it is greater than one there is data in it and it begins to split the data by its comma and upload it to the SensorMeasurments document.

```python
def uploadDevicessMeasurments(db):
    file_exists = os.path.exists("deviceMeasurments.csv")

    if(file_exists == True):
        try:
            datalog = open("deviceMeasurments.csv", "r")
            data = datalog.read()
            data = data.strip()
            datalog.close()
            indexedData = data.split("\n")
            if(len(indexedData) > 1):
                for x in range(len(indexedData)):
                    comma_seperated_indexed_data = indexedData[x].split(",")
                    data = {"Timestamp": comma_seperated_indexed_data[2],"DeviceID": comma_seperated_indexed_data[1],
                            "Device": comma_seperated_indexed_data[0], "Unit": comma_seperated_indexed_data[1], "Value": comma_seperated_indexed_data[4]}
                    db.child("SensorMeasurments").push(data)
                open("deviceMeasurments.csv", "w").close()
            else:
                pass
        except Exception as e:
            print(e)
    else:
        pass
```

Figure 84 - Upload Devices Measurements.

5.5.11 Read Devices.

This function only ever gets executed when the there is an internet connection.

It reads the devices from devices document o the firebase database it sorts them into their sensor type on the and takes a measurement, it then uploads the measurement to the database.

```python
def readDevices(db):
    try:
      allDevices = db.child("Devices").get()
      for device in allDevices.each():

            devkey = str(device.key())
            devAttrs = db.child("Devices").child(devkey).get()

            for attrib in devAttrs.each():
                if(str(attrib.key()) == "deviceType"):
                    if(str(attrib.val()) == str(1)): #DHT22
                            devId = str(devAttrs.key())
                            value = readDht22Device(db, devId)
                            data = {"Timestamp": getTimestamp(),"DeviceID": str(devId), "Device": "DHT22", "Unit": "oC" , "Value": str(value)}
                            db.child("SensorMeasurments").push(data)

                    elif(str(attrib.val()) == str(2)): # MQ135
                        devId = str(devAttrs.key())
                        value = readMQ135Device(db, devId)
                        data = {"Timestamp": getTimestamp(),"DeviceID": str(devId), "Device": "MQ135", "Unit": "" , "Value": str(value)}
                        db.child("SensorMeasurments").push(data)

                    elif(str(attrib.val()) == str(3)): # Soil Moisture
                        devId = str(devAttrs.key())
                        value = readSoilMoistDevice(str(attrib.val()))
                        data = {"Timestamp": getTimestamp(),"DeviceID": str(devkey), "Device": "Moisture Sensor", "Unit": "Moisture" , "Value": str(value)}
                        db.child("SensorMeasurments").push(data)

                    |

    except:
        print(datetime.now(), colored("No Devices Configured", "red"))

def lightSensor():
    return random.randint(0, 255)
```

Figure 85 - Read Devices Code.

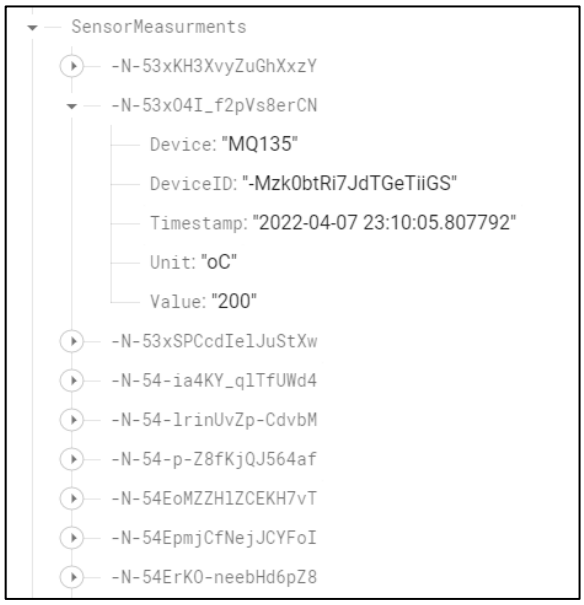


Figure 86 - Sensor Measurements Document.

With this last function the project was complete, the user of the app is able to measure garden.

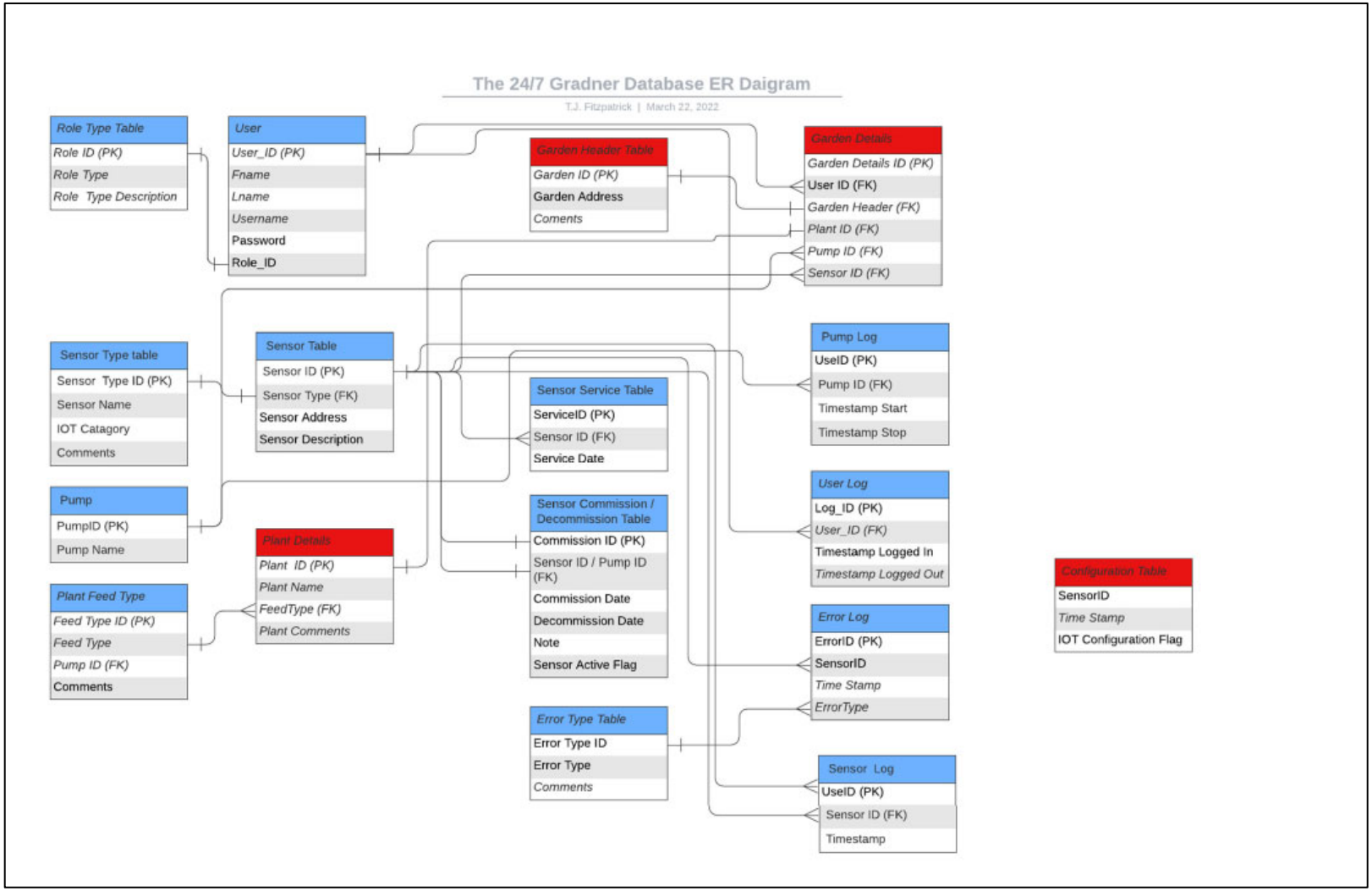


Figure 87 - Entity ER Diagram Of What the Initial System Was Supposed To Do.

6. Deployment.

During this project there were times when I had to build and evaluate different hardware technologies, I did this in my own home.

I deployed a Raspberry Pi and created my own garden once this was done, I then tested various different sensors.

Figure 88 - Creating My Own Garden for Deployment of System

Once I had my own system up and running the next step was to deploy water pumps.

Once these were deployed, I needed tube holders, so I used old cloth hangers and some pliers to bend them into a holder.

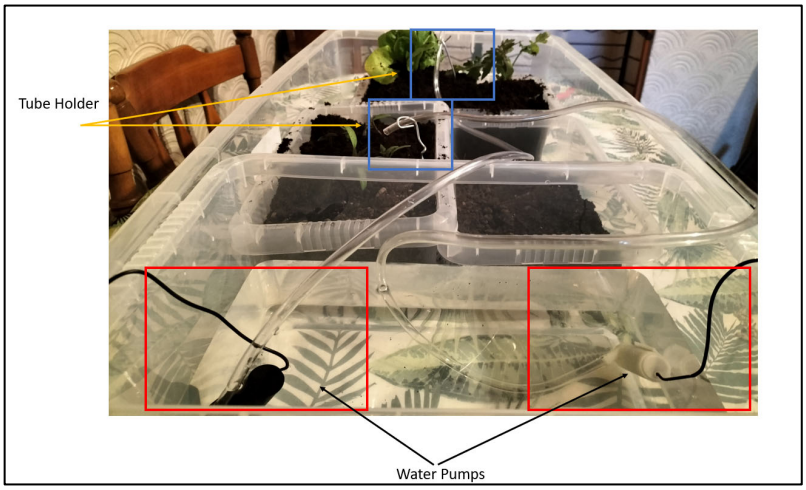


Figure 89 - Creating an Irrigation System.

I then tried my proof of concepts on the different sensors.

Figure 90 - Soil Moisture Sensors.

Figure 91 - DHT22 Measuring Temperature and Humidity.

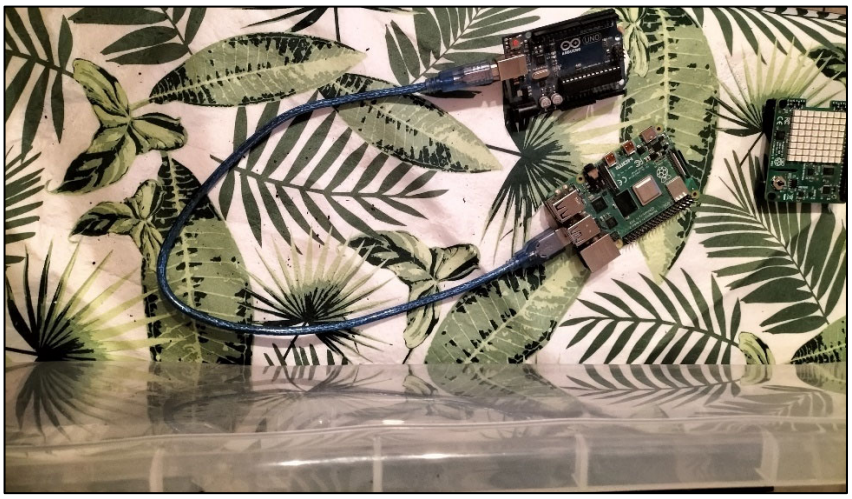


Figure 92 - Raspberry Pi and Arduino.

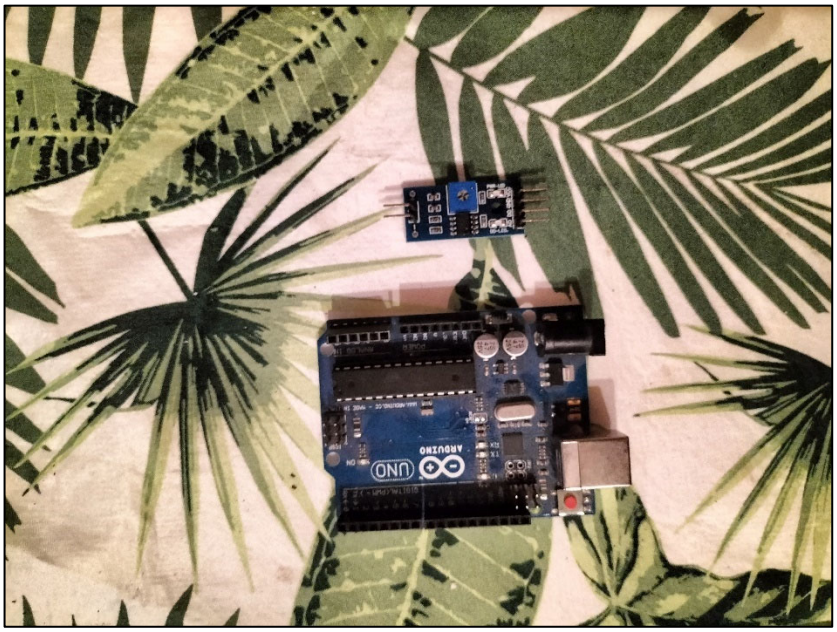


Figure 93 - Arduino Vs Converter Board

The last step was I needed to keep track of all the wiring, in order to do that I used cable labels.

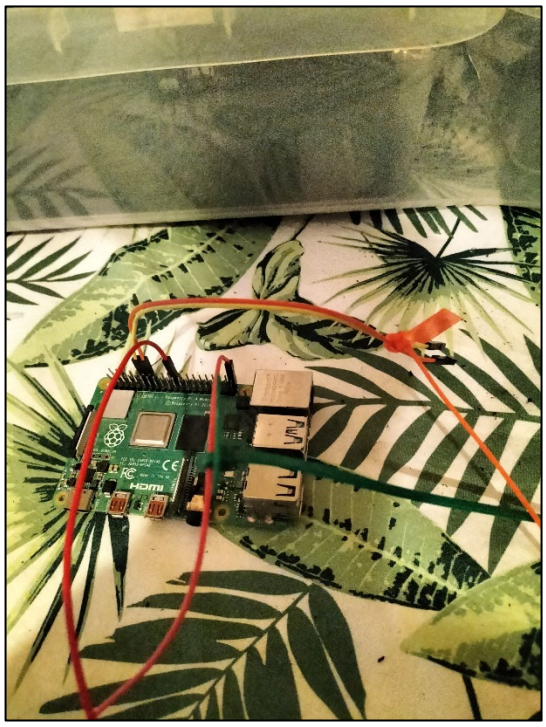


Figure 94 - Raspberry Pi with Wiring and Cable Labels.

7. Conclusion.

This project really did test my own ability think and be creative. As the weeks went by slowly but surely the jigsaw puzzle came together as the work, I put into this project began to show, it does have its downfalls, but it completed the initial specification.

7.1 Personal Reflection.

During this project there were up's down, and various different problems, if I was to reflect and say "would I apply a different thought process" I really would.

For example, I would of evaluating the Firebase Realtime Database against the Firestore database, the difference here is that if I chose the Firestore database which is a more professional database I would have lost my Realtime updates for my android application.

But I would of having a more secure and professional database.

Another problem with the Realtime Database was it would not allow special characters,

So, I had designed A Login Page, A Register Page and a "Forgot Password Page" using Goole Authentication and then found out I could not store an email address in the database.

Figure 95 - Errors with Special Characters.

This was the reason why originally, I had login page based on Email/Password Authentication, but I had to switch to username because of this error.

Thinking on it, I should have just split the username and domain name into 2 different fields, but I guess I just didn't think of them at the time.

Another problem I came across whilst working with Firebase is that I used a free monthly license, but the project spanned over the course of 3 months, so I had to renew and start a new database every month.

It didn't occur to me originally, but my Android Studio program would generate config files without my knowledge and then try connect to previous databases that didn't exist anymore, when it happened first I just started again, but on the third time I debugged the situation and found this out I needed to find these config files and erase them.

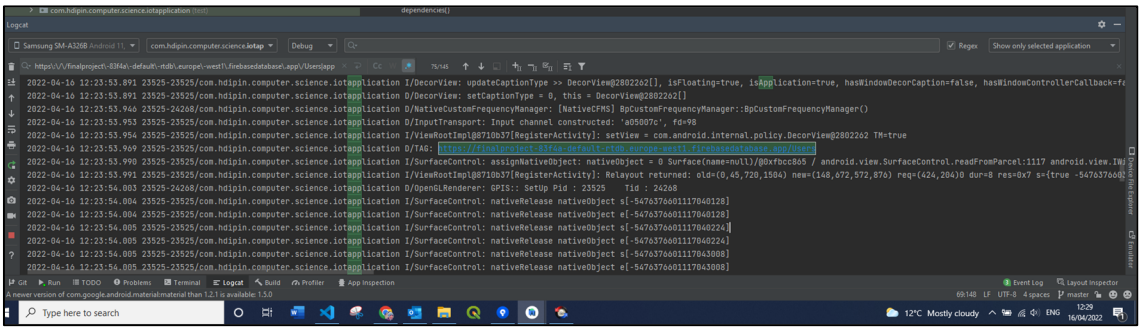


Figure 96 - Connecting to the Wrong Database.

Another thing I wouldn't do again I would not use digital sensors like the DHT's again, they are very buggy and generate a lot of errors and very slow, I would like to use digital that I can put on a bus like SPI or I2C.

I would also like more accuracy in my sensors for this reason I would try use the Arduino and an extension of the Raspberry Pi.

8. Bibliography

MIT. (2022, 04 18). *App Inventor*. Retrieved from AppInventor: https://appinventor.mit.edu/

mudpi. (2022, 04 18). *Automated Garden System Built on Raspberry Pi for Outdoors or Indoors - MudPi.* Retrieved from https://www.instructables.com/: https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.instructables.com%2FAutomated-Garden-System-Built-on-RaspberryPi-for-O%2F&psig=AOvVaw3VS1aNjZKlXYVAjbqmG5KC&ust=1650361459927000&source=images&cd=vfe&ved=0CAwQjRxqFwoTCPCo2depnfcCFQAAAAAdAAAAABAD

mudpi. (2022, 04 07). *Automated-Garden-System-Built-on-RaspberryPi-for-O/*. Retrieved from Instructables: https://www.instructables.com/Automated-Garden-System-Built-on-RaspberryPi-for-O/

P. Leach. (2022, 04 17). *rfc4122.* Retrieved from datatracker.ietf.or: https://datatracker.ietf.org/doc/html/rfc4122

Raspberry Pi Powered IOT Garden. (2022, 04 18). Retrieved from instructables.com: https://www.instructables.com/Raspberry-Pi-Powered-IOT-Garden/#:~:text=A%20Raspberry%20Pi%20is%20used,smartphone%20app%20that%20we%20built.

raspberrypi.dk. (2022, April 14). */en/product/raspberry-pi-sense-hat/*. Retrieved from https://raspberrypi.dk/en/product/raspberry-pi-sense-hat/: https://raspberrypi.dk/en/product/raspberry-pi-sense-hat/

Technovation. (2022, 04 07). *Raspberry-Pi-Powered-IOT-Garden*. Retrieved from Instructables: https://www.instructables.com/Raspberry-Pi-Powered-IOT-Garden/