Contents

[1. Introduction. 4](#_Toc101092274)

[1.1 Acknowledgement 4](#_Toc101092275)

[1.2 Background 4](#_Toc101092276)

[2.Project Research. 5](#_Toc101092277)

[2.1 Review of the Online Literature 5](#_Toc101092278)

[2.2Research Question 5](#_Toc101092279)

[2.3 Methodology 6](#_Toc101092280)

[2.3.1 The Raspberry Pi Powered garden. 6](#_Toc101092281)

[2.3.2 The Automated Garden System Using A Raspberry PI. 6](#_Toc101092282)

[3.Project Proposal 7](#_Toc101092283)

[4.Project Specification. 9](#_Toc101092284)

[4.1 Main Controller. 10](#_Toc101092285)

[4.1.1 Raspberry Pi Vs ODROID XU4. 10](#_Toc101092286)

[4.1.2 Raspberry Pi Vs ASUS Tinker Board. 10](#_Toc101092287)

[4.1.3 Raspberry Pi Vs Arduino. 10](#_Toc101092288)

[4.2Sensors**.** 11](#_Toc101092289)

[4.2.1 Temperature And Humidity. 11](#_Toc101092290)

[4.2.2 Soil Moisture. 12](#_Toc101092291)

[4.2.3 Air Quality Sensor. 12](#_Toc101092292)

[4.2.3 Water Pump. 14](#_Toc101092293)

[4.3Database (Firebase vs Mongo) 15](#_Toc101092294)

[Project Implementation. 17](#_Toc101092295)

[5.1 Python Interface to Firebase Database. 17](#_Toc101092296)

[5.2 Python On Raspberry Pi. 22](#_Toc101092297)

[5.2.1 Sense Hat’s Pressure Temperature and Humidity. 22](#_Toc101092298)

[5.2.2 DHT11 & DHT22. 22](#_Toc101092299)

[5.2.3 Soil Moisture Sensor. 24](#_Toc101092300)

[5.2.4 MQ-135 Sensor. 25](#_Toc101092301)

[5.2.5 Water Pump. 26](#_Toc101092302)

[5.3 Kotlin On Android. 27](#_Toc101092303)

[5.4 Final Kotlin Program. 30](#_Toc101092304)

[5.4.1 Splash Screen. 30](#_Toc101092305)

[5.4.2 Login Screen. 30](#_Toc101092306)

[5.4.2 Register Screen. 32](#_Toc101092307)

[5.4.3 Forgot Password Screen. 34](#_Toc101092308)

[5.4.5 Main Menu Screen. 35](#_Toc101092309)

[5.4.6 User Profile. 36](#_Toc101092310)

[5.4.6 Settings. 37](#_Toc101092311)

[5.4.6 Add a Garden. 38](#_Toc101092312)

[5.4.7 Add a Device. 39](#_Toc101092313)

[5.4.8 List Devices. 40](#_Toc101092314)

[5.4.9 List Gardens. 41](#_Toc101092315)

[5.5 Final Python Program. 42](#_Toc101092316)

[5.5.1 Main Flow Diagram. 42](#_Toc101092317)

[Bibliography 45](#_Toc101092318)

[Figure 1 - The Raspberry Pi Powered Garden 6](#_Toc101092319)

[Figure 2 ‐ Example of a MudPi Circuit Diagram. 7](#_Toc101092320)

[Figure 3 – Example of A MudPi Application 7](#_Toc101092321)

[Figure 4 ‐ Waterfall Diagram 8](#_Toc101092322)

[Figure 5 - Trello Board 9](#_Toc101092323)

[Figure 6 - DHT11 & DHT22 Sensors 11](#_Toc101092324)

[Figure 7 - Soil Moisture Sensors 12](#_Toc101092325)

[Figure 8- Air Quality Sensor 13](#_Toc101092326)

[Figure 9 - Water Pumps 14](#_Toc101092327)

[Figure 10 - Water Pump 14](#_Toc101092328)

[Figure 11 - System Architecture Diagram. 16](file:///C:\Users\tjfitzpatrick\Desktop\P%20Handbook\FInal%20Project.docx#_Toc101092329)

[Figure 12 - Project Diagram. 17](#_Toc101092330)

[Figure 13 - Firebase Sign Up Authentication in Python . 18](#_Toc101092331)

[Figure 14 - Evaluating Firebase Sign Up Authentication in Python. 18](#_Toc101092332)

[Figure 15 - Firebase Login Authentication in Python. 18](#_Toc101092333)

[Figure 16 - Evaluating Firebase Login Authentication in Python. 18](#_Toc101092334)

[Figure 17 - Verification that Signup and Login Authentication functioned Correctly. 19](#_Toc101092335)

[Figure 18 - Uploading a file to the Firebase Cloud Server. 19](#_Toc101092336)

[Figure 19 - Downloading a File from the Cloud Database 20](#_Toc101092337)

[Figure 20 - Verification that Cloud Upload functions Correctly. 20](#_Toc101092338)

[Figure 21 - Example CRUD of Firebase instance 21](#_Toc101092339)

[Figure 22 - Documents on Firebase Realtime Database. 21](#_Toc101092340)

[Figure 23 - Raspberry Pi's Sensehat 22](#_Toc101092341)

[Figure 24 - Pressure Temperature and Humidity Program. 22](#_Toc101092342)

[Figure 25 - DHT22 Circuit Diagram. 23](#_Toc101092343)

[Figure 26 - DHT 22 Proof of Concept Program. 23](#_Toc101092344)

[Figure 27 - Proof of Concept Temperature and Humidity. 24](#_Toc101092345)

[Figure 28 - A Circuit Diagram To Read Soil Moisture Using an Arduino. 24](#_Toc101092346)

[Figure 29 - A Circuit Diagram To Read Soil Moisture Using an Signal Converter. 25](#_Toc101092347)

[Figure 30 - A Circuit Diagram To read air quality using an Arduino. 25](#_Toc101092348)

[Figure 31- A Circuit Diagram To Read air Quality Using an Signal Converter. 26](#_Toc101092349)

[Figure 32 - Raspberry Pi Interfaced to A Water Pump 27](#_Toc101092350)

[Figure 33 – Storyboard of landing page and Login page. 27](#_Toc101092351)

[Figure 34 - Storyboard of user tying to login. 28](#_Toc101092352)

[Figure 35 - Storyboard of user trying to register. 28](#_Toc101092353)

[Figure 36 - Storyboard Of Everything. 29](#_Toc101092354)

[Figure 37 - Application Splash screen 30](#_Toc101092355)

[Figure 38 - Login Page. 31](#_Toc101092356)

[Figure 39 - Incorrect Username and Password 31](#_Toc101092357)

[Figure 40 - Snapshot of Main Loop at Login 31](#_Toc101092358)

[Figure 41 - Snapshot of the validation of Login Details 32](#_Toc101092359)

[Figure 42 - Snapshot of Code Validating Login Data against the Database. 32](#_Toc101092360)

[Figure 43 - Registration Screen. 33](#_Toc101092361)

[Figure 44 - Terms And Conditions Have Not Been Accepted. 33](#_Toc101092362)

[Figure 45 - Passwords Do Not Match 33](#_Toc101092363)

[Figure 46 - User Already Registered. 33](#_Toc101092364)

[Figure 47 - User Successfully Registered. 33](#_Toc101092365)

[Figure 48 - User Gets Redirected to the login page. 33](#_Toc101092366)

[Figure 49 - User Being Created On Database. 34](#_Toc101092367)

[Figure 50 - Forgot Password Screen. 34](#_Toc101092368)

[Figure 51 - Resetting The Email. 34](#_Toc101092369)

[Figure 52 -24/7 Gardiner Main Menu Screen 35](#_Toc101092370)

[Figure 53 - Users Profile. 36](#_Toc101092371)

[Figure 54 - Device Configuration. 37](#_Toc101092372)

[Figure 55 - Device Configuration on The Realtime Database. 38](#_Toc101092373)

[Figure 56 - Add a Garden Screen 38](#_Toc101092374)

[Figure 57 Add A Device Screen 39](#_Toc101092375)

[Figure 58 - Device Being Created on the Real Time Database 39](#_Toc101092376)

[Figure 59 - Device Measurements being Displayed to the User. 40](#_Toc101092377)

[Figure 60- Device Measurements On The Database 40](#_Toc101092378)

[Figure 61 - List Gardens Screen. 41](#_Toc101092379)

[Figure 62 - Database Garden Header Documents 41](#_Toc101092380)

[Figure 63 - Main Raspberry Pi Flow Diagram 42](#_Toc101092381)

[Figure 64 - Main Loop Of Raspberry Pi Code 43](#_Toc101092382)

[Figure 65 - Internet Connection Test Flow Diagram 44](#_Toc101092383)

[Figure 66 – Internet Test Connection Code 44](#_Toc101092384)

# 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, I researched 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.

## 2.2Research Question

How can adequate soil moisture for optimal plant growth be maintained, without the need for daily observation from the person caring for the plant? Furthermore how can the impact of the plant on air quality be monitored, in conjunction with the aforementioned maintenance of the plant?

### Methodology

## 2.3.1 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 :

o Real-time feedback of the garden's various sensors o Database of the garden's health status

o 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 to create a smartphone application which is compatible with the Firebase database and the Raspberry Pi.

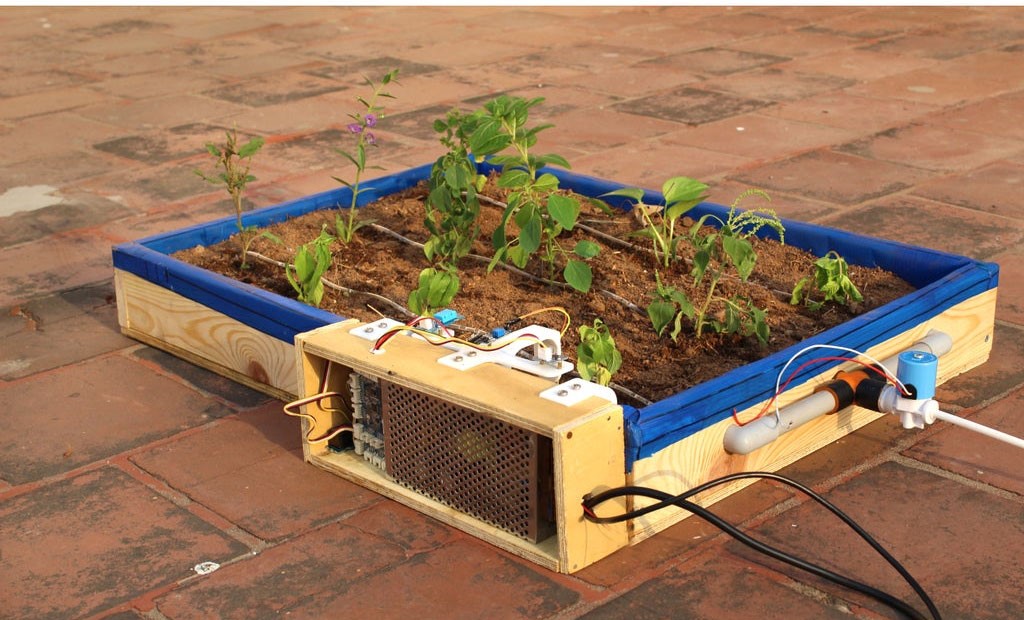


Figure - The Raspberry Pi Powered Garden

## 

## 2.3.2 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.

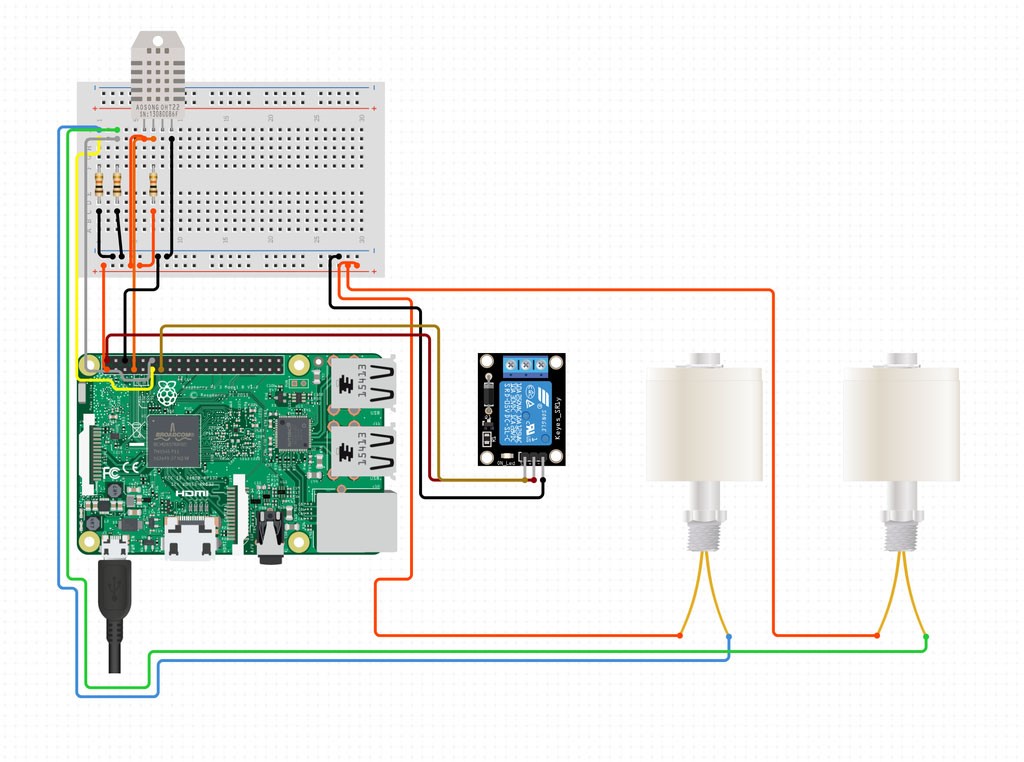


Figure ‐ Example of a MudPi Circuit Diagram.

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



Figure – Example of A MudPi Application

Upon researching these 2 systems I able to put a project proposal 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.

3.1Technologies.

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 document I am going use 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 a.



Figure ‐ 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 myself I used Trello 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 - 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 I 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 analog-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 pin | V5.0 |  | | | Wi-F | i | | |
|  | 802.11b/g/n | | | | |  |
| ODROID XU4 | $95.00 | 2GB | N/A | N/A |  | | | Wi-F | i | | |
|  | 802.11b/g/n | | | | |  |
| ASUS Tinker  Board | $105.00 | 2GB | 28 pin | BLE |  | | | Wi-F | i | | |
|  | 802.11b/g/n | | | | |  |
|  | | | | |
| Arduino | $24.05 | 2K SRAM  1K  EEPROM | 20 pin | Add on Board |  | | Add on | | |  | |
|  | board |  |
|  |

Table 1- Main Controller Cost Benefit Analysis

## 4.2Sensors**.**

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.

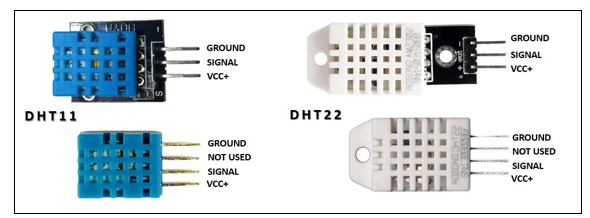


Figure - DHT11 & DHT22 Sensors

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

**Table 1 ‐ 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.

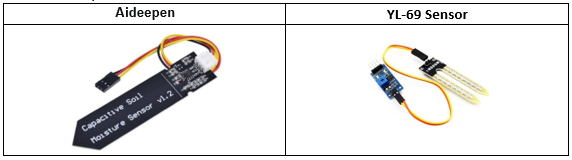


Figure - Soil Moisture Sensors

### 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 go 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.

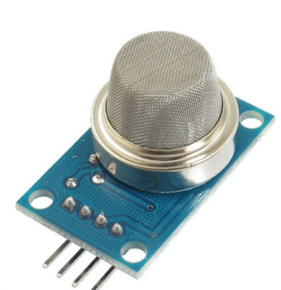


Figure 8- Air Quality Sensor

### 4.2.3 Water Pump.

If 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.

Graphical user interface, application

Description automatically generated

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.

## 4.3Database (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 2‐ 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.

Diagram

Description automatically generated

Figure 11 - System Architecture Diagram.

# 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.

A picture containing company name

Description automatically generated

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 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.

Text

Description automatically generated

Figure - Firebase Sign Up Authentication in Python .

Text

Description automatically generated

Figure - Evaluating Firebase Sign Up Authentication in Python.

Text

Description automatically generated

Figure - Firebase Login Authentication in Python.

Text

Description automatically generated

Figure - Evaluating Firebase Login Authentication in Python.

Graphical user interface, text, application, email

Description automatically generated

Figure - 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 uploaded and download a simple text file.

Text

Description automatically generated

Figure - Uploading a file to the Firebase Cloud Server.

Text

Description automatically generated

Figure - Downloading a File from the Cloud Database

Graphical user interface, text, application, email

Description automatically generated

Figure - 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 JSON 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 ID’s are automatically generated in Firebase and are completely unique.

Text

Description automatically generated

Figure 21 - Example CRUD of Firebase instance

Graphical user interface, text, application, email

Description automatically generated

Figure 22 - Documents on Firebase Realtime Database.

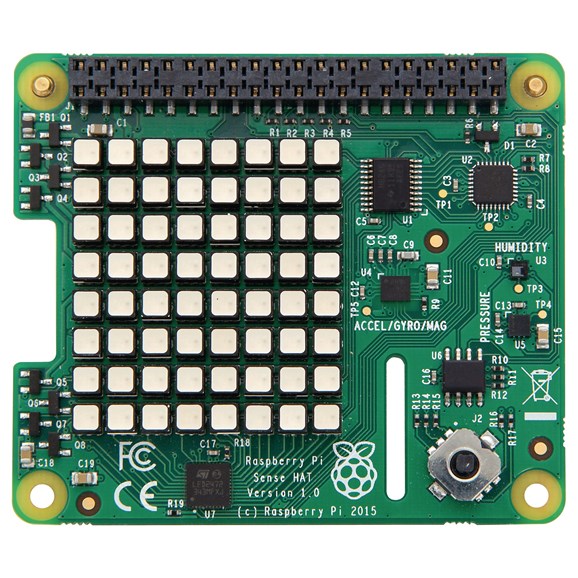
Once the proof of concept idea was complete I could now connect to the database using python running on the Raspberry Pi the next step was to build onto the sensors part of this project.

## 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

Text

Description automatically generated

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 sense hat. 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.

Our circuit is as follows:

A picture containing graphical user interface

Description automatically generated

Figure - 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.

A screenshot of a computer

Description automatically generated with medium confidence

Figure - DHT 22 Proof of Concept Program.

Text

Description automatically generated

Figure - 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.

Diagram

Description automatically generated with medium confidence

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

The second way is to use a converter road, 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.

Schematic

Description automatically generated with medium confidence

Figure - A Circuit Diagram To Read Soil Moisture Using an 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.

A picture containing diagram

Description automatically generated

Figure - 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.

Diagram, schematic

Description automatically generated

Figure - A Circuit Diagram To Read air Quality Using an 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.

A picture containing text, clock

Description automatically generated

Figure - 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 would look like:

Graphical user interface, application

Description automatically generated

Figure 33 – Storyboard of landing page and Login page.

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

A group of cell phones

Description automatically generated with medium confidence

Figure 34 - Storyboard of user tying to login.

The next storyboard is of a user trying to register:

Graphical user interface, application

Description automatically generated

Figure 35 - Storyboard of user trying to register.

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

Graphical user interface, application

Description automatically generated

Figure 36 - Storyboard Of Everything.

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 a 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.

|  |  |
| --- | --- |
| Graphical user interface, text  Description automatically generated  Figure - Login Page. | Figure 39 - Incorrect Username and Password |

Text

Description automatically generated

Figure 40 - Snapshot of Main Loop at Login

Text

Description automatically generated

Figure 41 - Snapshot of the validation of Login Details

Text

Description automatically generated

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 users username.

Graphical user interface, text, application, email

Description automatically generated

Figure 49 - User Being Created On Database.

### 5.4.3 Forgot Password Screen.

This screen was originally designed for when we using google authentication against email address’s. 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.

|  |  |
| --- | --- |
| Graphical user interface, text, application  Description automatically generated  Figure - 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.

Diagram

Description automatically generated

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.

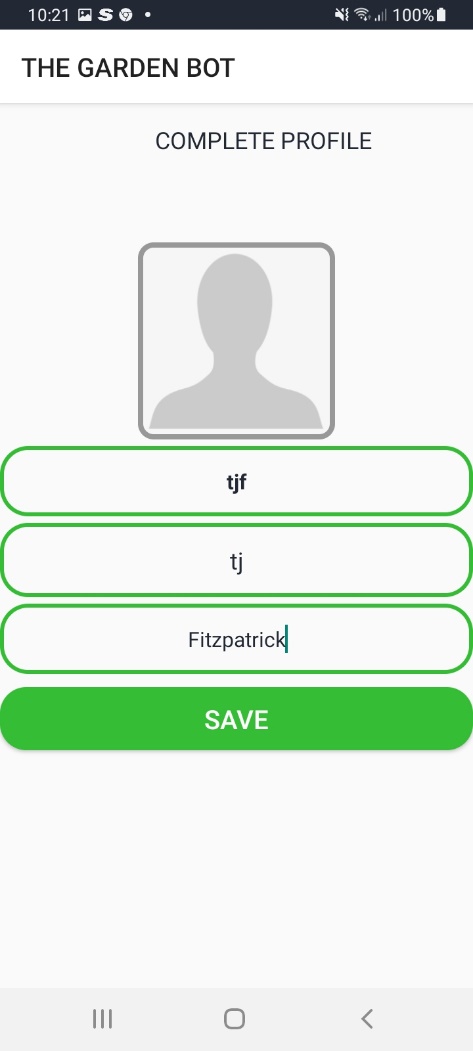


Figure - Users Profile.

### 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.

Graphical user interface, application

Description automatically generated

Figure - 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.

Graphical user interface, text, application

Description automatically generated

Figure - 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.

A screenshot of a phone

Description automatically generated with low confidence

Figure - 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.

Graphical user interface, application

Description automatically generated

Figure Add A Device Screen

Graphical user interface, text, application, email

Description automatically generated

Figure - Device Being Created on the Real Time Database

### 5.4.8 List Devices.

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

Graphical user interface, text, application

Description automatically generated

Figure - Device Measurements being Displayed to the User.

Graphical user interface, text, application, email

Description automatically generated

Figure - 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.

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure - List Gardens Screen.

Graphical user interface, text, application

Description automatically generated

Figure - Database Garden Header Documents

## 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.

Diagram

Description automatically generated

Figure - 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 its 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).

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.

Text

Description automatically generated

Figure - 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.

Diagram

Description automatically generated

Figure - Internet Connection Test Flow Diagram

A screenshot of a computer

Description automatically generated with medium confidence

Figure 66 – 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.

Text

Description automatically generated

Figure - 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”.

Text

Description automatically generated

Figure - Pushing Data to the Database.

Text

Description automatically generated

Figure - 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 4 seconds, if something goes wrong then the default frequency is 10 seconds.

Text

Description automatically generated

Figure - Reading an Attribute on the Configuration Document.

Text, letter

Description automatically generated

Figure 71 - 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.

### Text Description automatically generated

Figure - Writing A Configuration File.

### 

A picture containing text

Description automatically generated

Figure - 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.

Graphical user interface, table

Description automatically generated with medium confidence

Figure - Devices Firebase Document.

Text

Description automatically generated

Figure 75 - Devises Log File Code.

Text

Description automatically generated

Figure 76 - Devices JSON File

# Bibliography

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/

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/