National College of Ireland
BSc in Computing
2014/2015

Jack Haymes
13114671
jack.haymes@ncirl.ie
jackhaymes3@gmail.com

House Plant Hydration Notification System

Technical Report

Table of Content

Executive Summary

The problem being addressed is plant dehydration. Nowadays, with cheap travel, people go away often and plants are regularly neglected. Another reason plants are neglected is because of people's busy work schedules or some people are just not aware how often their plants need to be watered. Also as we all know there are different plant families, some requiring a lot of water and some not. For instance succulents, require very little water as they normally live in deserts, whereas basil, the common herb, requires daily watering, if it is to thrive.

I have decided to address this problem by developing this system.

I have been working on a solution. In my opinion this solution involves a Raspberry Pi with a connected moisture sensor that is buried in the soil of the plant. When the moisture level drops, it will be detected by the sensor and this information will be recorded and stored on the Raspberry Pi and a notification will be sent to the owner/user's mobile phone indicating that they need to water their plant.

I have evaluated the need for this system and all the feedback I have received has been positive. As long as costs are realistic/affordable the system would be popular. The feedback I have received has come from private residential homes, companies, banks, hotels and other businesses that have plants in their foyers.

The system could also be used in schools and colleges but I have yet to evaluate their needs and feedback.

1. Introduction

1.1.Purpose

The purpose of this document is to set out the requirements for the development of my 4th year project, a house plant hydration notification system.

The intended customers are individuals, companies, restaurants, hospitals, etc., that are concerned with taking good care of their indoor plants.

1.2.Project Scope

The scope of the project is to develop a system that will notify a user when the moisture level in the soil of a potted house plant drops below a certain level, indicating that it should be watered. The system will use a Raspberry Pi attached to a moisture sensor in the pot. When the moisture level drops below an adequate level for a particular plant, a notification will be sent to the users Android device, informing them they need to water the plant.

I have discussed the idea of the project with several home owners and office workers. Each of them agreed that they think it would be a good idea, and would have a use for it.

This section also details any constraints that were placed upon the requirements elicitation process, such as schedules, costs, or the software engineering environment used to develop requirements.

1.3.Definitions, Acronyms, and Abbreviations

RaspBerry Pi - A Raspberry Pi is a credit card-sized single-board computer. They are used to help yeah computer science in schools and in developing countries. It is used for many computer projects and for a lot of things your desktop computer can do, like spreadsheets, browsing the internet, and playing games. It can also be used in conjunction with sensors that plug into the USB slots to record information, and relay it back.

RPi - Abbreviation for RaspBerry Pi.

Python - Coding language I will be using on the RPi.

AS - Android Studio.

AWS - Amazon Web Services.

GCM - Google Cloud Messaging.

IDE - Integrated Development Environment.

XML - Extensive Markup Language.

MQTT - Message Queue Telemetry Transport

1.4. Background

In my house we have a lot of plants and tend to take care of them quite well, but over the years, I have noticed that we don't always remember to water them as often as they need to be watered. This is due to a busy lifestyle that everyone seems to live these days. There is always an increasing number of applications that remind us with notifications to keep on top of the things we need to do or to gently remind us of certain things we may forget from time to time.

Watering your household plants isn't the top of anyone’s agenda, unless you’re a professional household plant waterer, but I haven't seen many of those around. But, when you do notice one of your plants starting to droop and die you usually tend to it with a pint of water, then you would usually proceed to check and water the other plants around the house.

What if you could be reminded by your mobile phone to water your plants as soon as the moisture level in the soil reaches a certain level? This would keep your plants at optimal health and allow them to flourish as much as possible. And would also save you the guilt and remorse from neglecting your lovely plants.

This is the idea behind the House Plant Hydration Notification System. When the soil reaches a certain level of dehydration the system will notify the owner that it should be watered. There is no specific frequency that works for all indoor plants.

You must determine what kind of plant you have and follow guidelines on how often it should be watered.

There are a variety of online encyclopaedias you can find that will detail how you should care for the specific type of household plant you own, including recommended humidity levels, sunlight exposure guidelines, and watering guides. Since many household plants differ, it’s important to find what is ideal for your specific type of household plant. Most plants come with a tag that has their common and scientific name, and if it doesn't you can ask the florist from where you got it. The system will store generic information about certain categories of plants. Each monitor in each plant will fall into one of these categories. This will enable it to be aware when a certain plant in a certain category falls below a recommended hydration level.

There are hand held moisture monitors available, but, you need to remember to use them in the first place. With this system it will automatically detects when the plant needs to be watered, and sends a push notification to the user’s phone.

Users of the system will download an app that the hydration sensors will be linked to. They can then sync as many hydration sensors to the app as they like. This will be the central place for storing the information on the plants and where the alerts will be sent from. The app could also show real time levels of water in the soil at the moment of checking. Also, there could be information and tips for caring for your plants, but, this will be an add on. First, I will need to get the functionality working.

1.5. Aims

The aim of this system is to provide the owners of plants with a notification when their house plant needs to be watered to save the plant from dehydration or death. It is to help people who forget to water their plants due to busy lifestyles, or other factors.

1.6. Technologies

I will be using a sensor that detects levels of moisture, a RaspBerry Pi computer, and Android Studio Integrated Developing Environment.

The moisture sensor that will be buried in the soil of the plant will be connected to a RaspBerry Pi. A Raspberry Pi is a very small single board computer, about the size of a credit card. It can be used to record information relayed back from sensors monitoring the external environment.

I will also be using Android Studio(IDE) to create an Android application for mobile phones. Android Studio is used to develop mobile and tablet applications based on the Android platform. Android Studio uses a mixture of XML (for layout) and Java (for functionality) to create applications.

AWS will be used to manage messages being sent from The Python code running on the RPi.

GCM will be used to enable push notifications on the user's device.

For database purposes, I will be using MySQL, and PHP MyAdmin.

1.7. Structure

The report is structured in such a way that by reading the first section of the document the reader will be able to get a general overview as to why the project has been developed and exactly what the project aims to do and the problems it aims to address. I have provided examples of this system steps being developed, I have described how my project will aim to provide a solution to a problem home owners may face. I have also described the technologies that will be used to develop my project, from start to finish.

The second section of this technical report will highlight the requirements of the system, ranging from the functional requirements that will describe specific functions of my system. I will also describe the data requirements to show the reader what data I will be working with such as MQTT, XML, etc. The user requirements will describe how the user interacts with the system. Environmental requirements,

such as what operating systems my system can run on and if there are any pre-requisites to running the software. Usability, interface, maintainability requirements, etc., will also be captured.

I will also visualise use cases for the major system requirements showing the scope, description, pre-condition, activation, main flow, alternate flows, exceptional flows, termination, post conditions and their frequency of use.

The third section of this report will detail the design and architecture of my system using UML diagrams, I will then discuss how the project will be implemented using snap shots of my code and any major algorithms to explain how the system works. I will also have some GUI layout detailing how the system will look from a user's point of view. Testing will then be carried out and documented.

The fourth section will evaluate any findings pertaining to the development and completion of my project, along with a conclusion in the fifth section.

I will also be documenting further development opportunities that could be possible, given the time and resources.

Finally I will present an appendix of documentation and sources that contributed to the development of this system.

2. System

2.1.Requirements

2.2.Functional requirements

1. Keep itself running for a long period of time.

2. Sync Android app to the Raspberry Pi.

3. Record moisture level in the soil.

4. Send a notification to the users phone(Android app).

5. Store different categories of plants.

2.2.1.Requirement 1: Keep itself powered on.

2.2.1.1. Description & Priority

The Raspberry Pi needs to be powered constantly as it needs to send a notification when the moisture drops below a predefined level in the soil. This could take a long time, depending on factors like the type of soil, the soil density, the type of plant, or the size of the plant, etc.

This is considered the highest priority as the system would be useless if it cannot detect the moisture level and send the notification.

2.2.1.2. Use Case

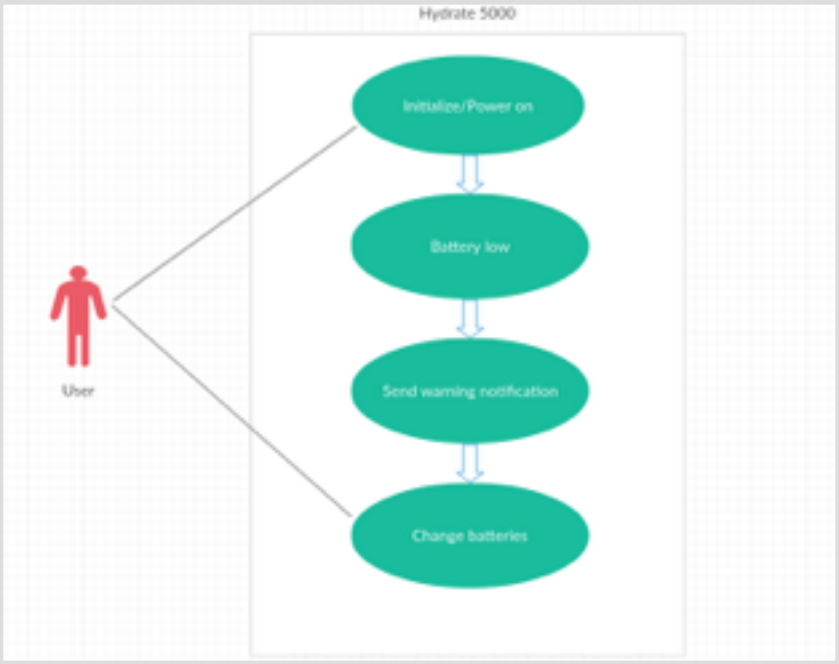
Scope

The scope of this use case is to **i**dentify when the system is running low on power.

Description

This use case describes the the steps involved from first setting up the system and connecting it to power, to when it is running low on power and needs to notify the user of this.

Use Case Diagram



Flow Description

Precondition

The system needs to be physically planted and attached to the pot of the plant it is monitoring.

Activation

This use case starts when an actor initialises the hydration monitor.

Main flow

1. The system is attached to the plant and the monitor placed in the soil.
2. The actor initialises the system by planting the sensor and connecting the Raspberry Pi to a power source. (See A1)
3. The system remains on the whole time. If using a battery pack, the user should be notified that the remaining power is low.(See E1)
4. The actor can then change the batteries and check that everything is in working order.

Alternate flow

A1 : The system is disconnected from a power source

1. The system is disconnected from a power source or runs out of battery without the user being aware.
2. The actor will have to realise that there is no power running the system.
3. The use case continues at position 4 of the main flow.

Exceptional flow

E1 : The system dies or stops working.

4. The system is no longer running due to a technical issue.
5. The actor will have to troubleshoot the issue and resolve it.
6. The use case continues at position 1 of the main flow

Termination

The system is discontinued by the user.

Post condition

There is no post condition for this use case.

2.2.3. Requirement 2: Allow communication between the Raspberry Pi and the Mobile Application.

2.2.3.1. Description & Priority

After setting up and powering on the Raspberry Pi the user will need to download the Android Hydration Monitor App.

This is ranked second in terms of priority, as, if the Raspberry Pi and Mobile App can not communicate, the Pi will not be able to automatically notify the user when the plant needs to be watered.

2.2.3.2. Use Case

Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.

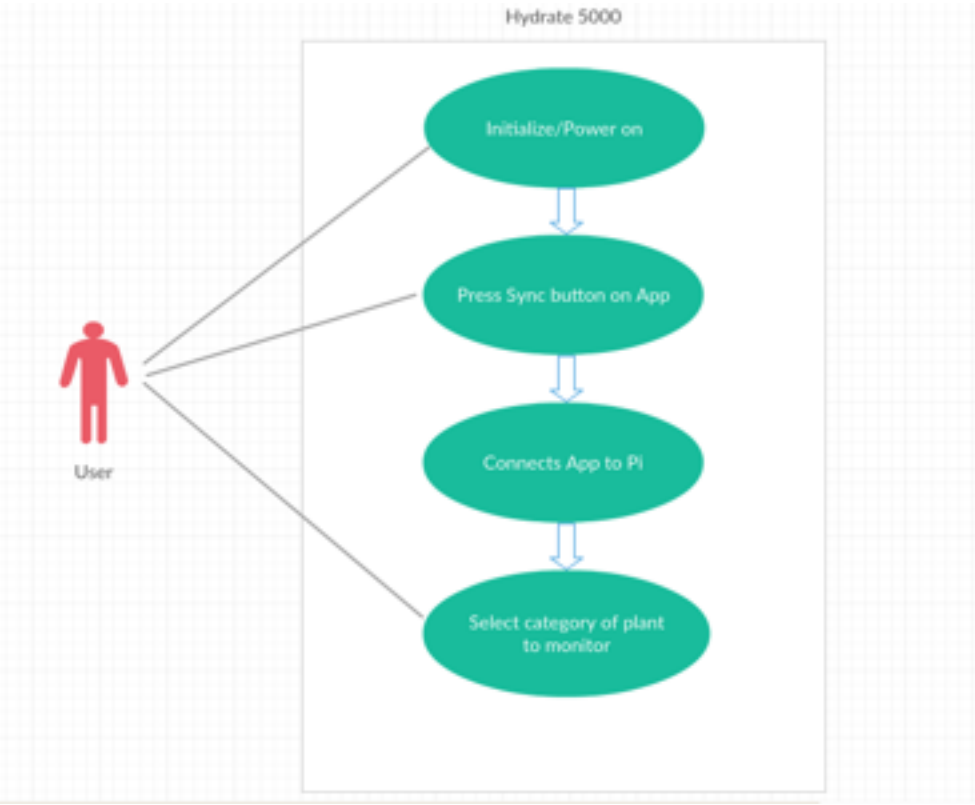
Scope

The scope of this use case is to demonstrate the steps of connecting the Raspberry Pi to the Mobile App.

Description

This use case describes the how to sync the downloaded mobile app to the hydration system so that they may communicate.

Use Case Diagram



Flow Description

Precondition

The system has been initialised, is connected to a power source, and is turned on.

Activation

This use case starts when an actor has the mobile app open and running and presses the Sync button.

Main flow

5. The system is planted in the soil of the plant and is being powered.
6. The actor has the app open and presses the Sync button. (See A1)
7. The system connects to the app. (See E1)
8. The actor is then able to select a category of plant to monitor.

Alternate flow

A1 : The app crashes

7. The system is still running but the app has crashed.
8. The actor can try to restart the app.
9. The use case continues at position 2 of the main flow

Exceptional flow

E1 : The app won't work/open.

10. The actor can delete the app completely from their phone.
11. The actor can re-download the app and try again.
12. The use case continues at position 2 of the main flow

Termination

The user has successfully connect to the Pi, selected the category of plant to monitor and has closed the app.

Post condition

The system goes into a “wait state” while monitoring the moisture level of the soil.

2.2.4.Requirement 3: Record Moisture Level of Soil

2.2.4.1. Description & Priority

This requirement is based on the sensor recording the moisture level of the soil.

It is 3rd in the priority list as there is no point in having it record anything if the previous two requirements are not functioning properly.

2.2.4.2. Use Case

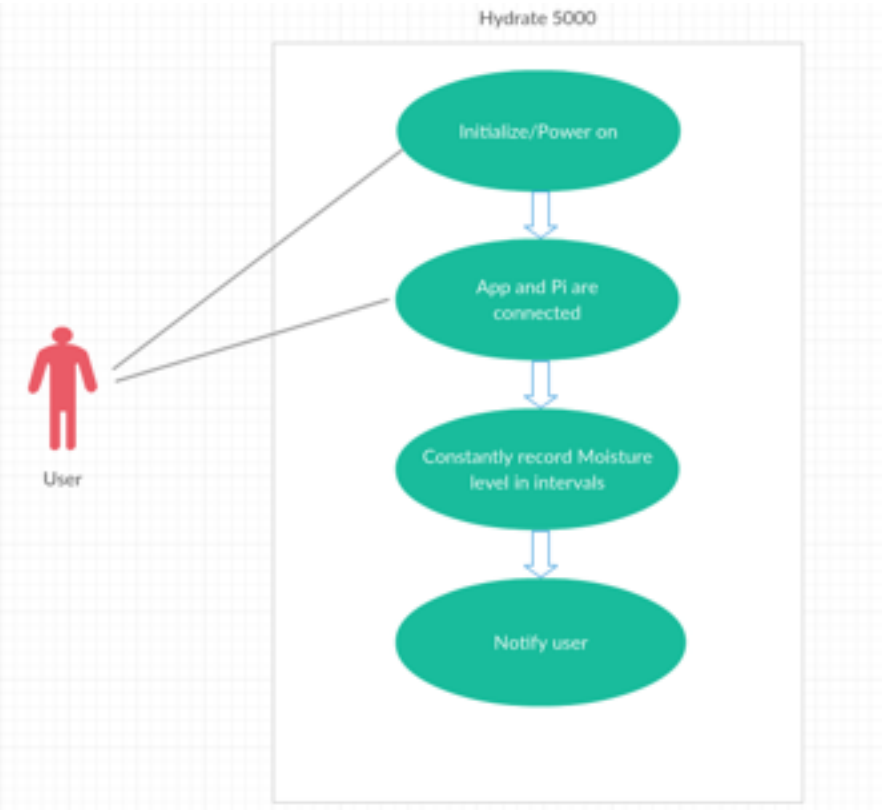
Scope

The scope of this use case is to ensure the moisture sensor is working as planned.

Description

The whole time the Raspberry Pi is powered on and being used, the sensor needs to be recording the moisture of the soil. Once the moisture drops below a certain level it will send a notification to the users phone(this is the next functional requirement).

Use Case Diagram:



Flow Description

Precondition

The system has been initialised, is connected to a power source, and is turned on. It must also be synced with the app and a category of plant selected.

Activation

This use case starts when an actor has the mobile app open, running, synced the devices, and has selected a category of plant to monitor.

Main flow

1. The system is running.

2. The actor syncs the devices. (See A1)
3. The system remains on the whole time. The sensor intermittently records the moisture data.(See E1)
4. The system notifies the user if the moisture becomes to low.

Alternate flow

A1 : The system cannot be synced.

1. The system is cannot be synced due to technical issues.
2. The actor will have troubleshoot the problem and try to resolve it.
3. The use case continues at position 2 of the main flow.

Exceptional flow

E1 : The system dies or stops working.

4. The system is no longer running due to a technical issue.
5. The actor will have to troubleshoot the issue and resolve it.
6. The use case continues at position 1 of the main flow

Termination

The system is discontinued by the user.

Post condition

The user gets a notification that the plant needs to be watered.

2.2.5.Requirement 4: Send a notification to the user's phone

2.2.5.1. Description & Priority

When the sensor detects that the moisture of the soil has dropped below the recommended level it sends a notification to the app on the users phone. This is 4th in the priority list as it can't be possible without the previous functional requirements, but, it is pretty much the whole point of this project.

2.2.5.2. Use Case

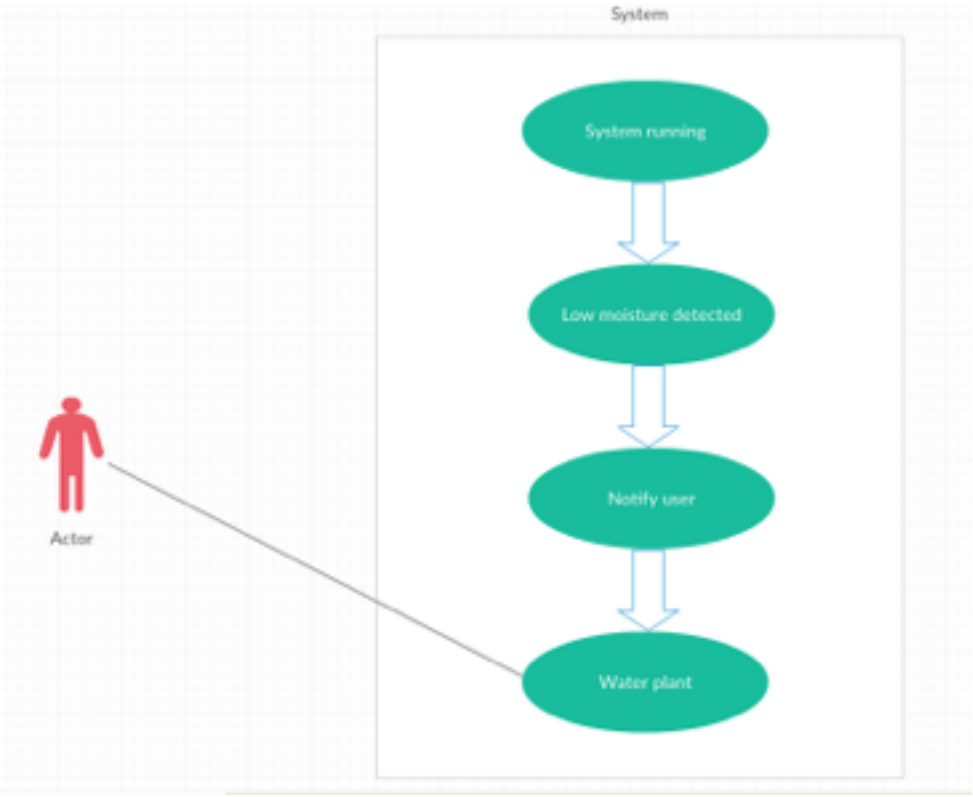
Scope

The scope of this use case is to demonstrate how the Pi notifies the user that a certain plant needs to be watered.

Description

This use case describes the steps from the sensor detecting low moisture levels to the user being notified about it.

Use Case Diagram:



Precondition

The system has been initialised, is connected to a power source, is running and waiting to detect low moisture levels.

Activation

This use case is active while the system is running, the app has been synced and the sensor has been planted.

Main flow

1. The system is on and running.
2. The actor places the sensor in the soil. (See A1)
3. The system remains on while the sensor records the moisture level of the soil. (See E1)
4. The system detects the moisture has dropped below a certain level.
5. The system notifies the user of this.
6. The user waters the plant.

Alternate flow

A1 : Sensor malfunction or loss of power.

1. The system is disconnected from a power source or the sensor stops working.
2. The actor will have to realise that there is no power running the system or that they have not received a notification around when they expected one.
3. The use case continues at position 1 of the main flow.

Exceptional flow

E1 : The system dies or stops working.

4. The system is no longer running due to a technical issue.
5. The actor will have to troubleshoot the issue and resolve it.
6. The use case continues at position 1 of the main flow

Termination

The system is discontinued by the user.

Post condition

The user waters the plant.

2.2.6.Requirement 5: The app must store categories of plants

2.2.6.1. Description & Priority

After the user has synced the system and the application, they need to select a category of plant that the one they are monitoring will fall into.

The app will be capable of storing these different types of categories for the user to choose from.

2.2.6.2. Use Case

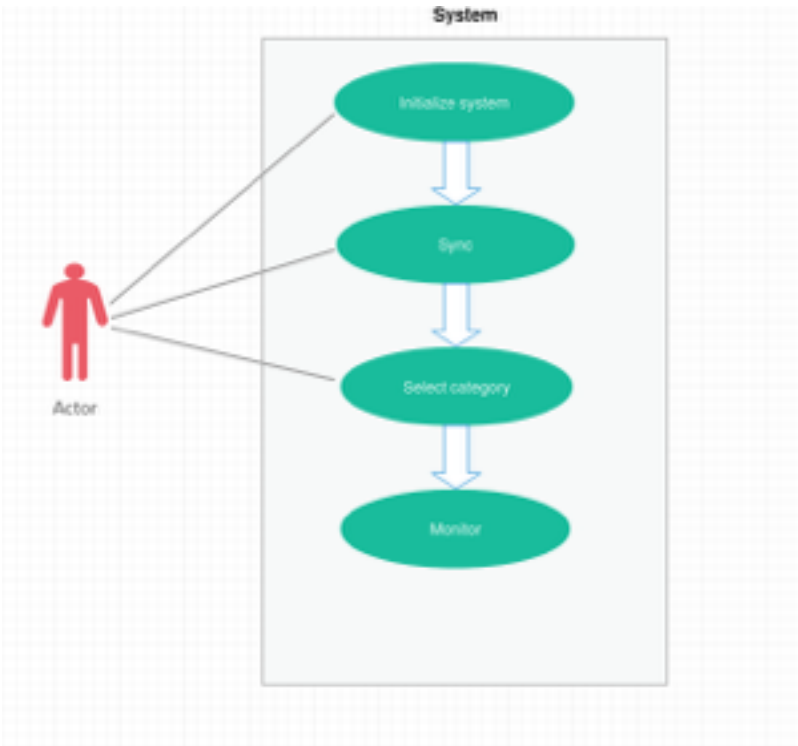
Scope

The scope of this use case is to demonstrate the steps of selecting a category of plant to monitor.

Description

This use case describes the steps in selecting a category of plant.

Use Case Diagram:



Flow Description

Precondition

The system has been initialised, is connected to a power source, is turned on, and synced to the app.

Activation

This use case starts when an actor has the mobile app open and running and has just synced the system and the app.

Main flow

1. The system is attached to the plant and the monitor placed in the soil.
2. The actor initialises the system by planting the sensor and connecting the Raspberry Pi to a power source. (See A1)
3. The system is synced with the mobile application.(See E1)
4. The actor selects a category of plant to monitor..

Alternate flow

A1 : The system is disconnected from a power source

1. The system is disconnected from a power source or runs out of battery without the user being aware.
2. The actor will have to realise that there is no power running the system.
3. The use case continues at position 4 of the main flow.

Exceptional flow

E1 : The system won't or can't find the app to sync with .

4. Unable to connect with mobile app.
5. The actor can restart the system.
6. The use case continues at position 3 of the main flow

Termination

The system is discontinued by the user.

Post condition

The system is ready to monitor the soil.

2.3.Non-Functional Requirements

2.3.1.Performance/Response time requirement

The system must successfully send a notification to the users phone within a suitable time period after the sensor has detected low moisture levels of the soil.

The slowest part of the system is when booting up the Raspberry Pi. When tested, times would vary in seconds but all tests were less than 60 seconds. The testing the response time of running the script on the RPi it would usually take about 12 seconds to run, connect to AWS, and start listening for requests.

The Android application has a minimal response time of about 2 seconds when booting up on a client device. When connecting to AWS, response time varies a little but generally connects within 5 seconds.

The sending of messages from Android to the Python script through AWS is almost instant.

Performance isn't really an issue as the Raspberry Pi is only monitoring one sensor while lying idle waiting to send or receive a message. This doesn't put any stress on this side of the system. The performance of the Android application would depend on a users device that it is installed on. But the application itself only handles the sending of MQTT messages to AWS so there is not much worry for coping with large loads of functionality.

2.3.2. Availability requirement

The system must be on, running and the sensor properly acting, as it should, for the system to be able to send the message to trigger the notification when it needs to. The app also needs to be functioning properly and the users phone switched on for it to be available to receive the notification.

The Raspberry Pi will need to be available 24/7 to record the moisture of the soil, it will require power and internet connection. The notification is triggered by a process running in the background of the user's device, so it does not matter if the application is open or closed, as long as the device it is installed on has power and internet connection.

2.3.3. Recovery requirement

The Raspberry Pi uses a micro SD card to run it's operating system and store all of it's necessary files and libraries on. I will have a backup micro SD card with a replica image of that initial operating system and files incase of a disaster that renders the card corrupt, not allowing the system to function as it should.

Code for the Python script and the Android Application will also be backed up on my machine using the source version control application, Git. This is then pushed to the cloud version control on GitHub. So in the case that code becomes corrupted, broken, or lost, I can easily re-install it from a particular version. The system can recover to a specific point in time, assuming that the backups are complete up to that point in time.

2.3.4. Robustness requirement

The code on the Raspberry Pi must be able to withstand system crashes. It must not be damaged or corrupted. This may be the case if there is a high electrical surge through the RPi which causes it to short out. In the

case of this happening there is no option but to replace the physical system and re-install the operating system with the appropriate code.

The Android application is extremely robust as it was developed to excellent standards, following professional guidelines in the latest version of the Android IDE, Android Studio.

The physical system should be enclosed in a case to prevent any damage being caused to it, either by being knocked or by getting wet.

2.3.5. Security requirement

The system maintains security through AWS API in terms of passing messages from Android to the Python script and visa versa.

To authenticate to AWS IoT Android used Cognito in conjunction with AWS IoT to create an identity (client certificate and private key) and store it in a Java key-store. After certificate and private key are added to the key-store the app will use these for future connections. This is implemented when configuring the AWS Mobile SDK for Android to allow the application to communicate with AWS IoT.

To allow connection between the Python script and AWS I used X.509 certificate credentials for certificate-based mutual authentication. I used the AWS console to generate and download the certificate and private key. I then specified where these were stored when initialising the Python client, to allow the secure connection between them.

If a user is accessing the Raspberry themselves they are required to enter a username and password to gain access.

The android application does not require any sign up or login functionality to gain access.

2.3.6.Reliability requirement

The system needs to be reliable enough to run on its own without any user interference. This requires a dedicated power source for the Raspberry Pi, allowing it to run 24/7. The Android application will be as reliable as the software on the user's phone, once it is installed.

If there are any issues with either code base, they can be backed up from the online source code version control, GitHub.

As there will be a backed up micro SD card with a full recovery model, file backup would be consistent.

2.3.7.Extendibility requirement

After some time I will look at adding additional sensors to additional plants to be monitored. I will need to extend the capability of the app and add extra Raspberry Pi's to the system.

Maintainability Requirement

The system will be easy to maintain and modify as long as the user has the correct access privileges and rights. Maintainability will only be nesse cary for modifications that are made between releases of major new ver sions of the application or component. These will include minor defect and enchantments throughout the systems lifetime.

If any critical corruption or failures occur, I will be able to roll back or restore the the source code with minimal work loss.

Downtime expected from any form of maintenance should be less than 15 minutes and will be performed at any time of the day as it would affect the user.

Portability Requirement

Over time, if there is more of a demand for the system, it could be de signed to run on IOS devices as well as windows, and other capable devices. The code could be written in different languages required by different devices. The physical device could also be changed for another, similar micro controller computer.

2.3.8.Reusability requirement

The system will be able to be reused in different plant pots if the user wishes to do so. They will just need to make sure that the system and the application are still in sync, and to select the appropriate category from the app for the new plant. The android application can be downloaded and used on any device that can host Android apps.

2.3.9.Data requirements

In this section I will describe the various data types which will be essential implementing the functionality of the system, allowing it to work, and seamlessly pass messages through components.

Data is first gathered and recorded by the moisture sensor buried in the soil of the plant. The sensor then relays this information back to the connected RaspBerry Pi. This data is recorded and stored in a variable in Integer format. This integer variable is then used to compare the readings to the particular threshold levels of the adequate moisture levels.

The message sent to AWS from the Python script is sent in the form of a light weight MQTT message.

“MQTT is a simple and lightweight messaging protocol for M2M communications that enables the transfer of telemetry-style data in the form of messages from pervasive devices, along high latency or constrained networks, to a server or small message broker.” - prismtech.com (2017).

The messages, sent from Android to AWS, determining which category of plant to monitor are also sent as MQTT messages.

For the development of the Android application a mixture of XML(for layout and design) and Java(for functionality) is used.

Both the Python and Android applications need to store the keys for connecting to AWS. These keys are accessed using string variables containing, the path, name, password, and certificate. When connecting to AWS, both applications need to distinguish themselves. They do this by using a client ID in string format.

2.3.10. User requirements

Users first set the system up which involves connecting the sensor to the RaspBerry Pi. These components are then placed in the plant pot, with the sensor being place directly into the soil. After it is set up and switched on the user can download the accompanying app from the Play Store, if they have not already done so. All that is left for them to do then, is to follow the steps after opening the application on their phone, or tablet, to sync and select the category of plant.

There is no requirement for the user to sign up or log in to the Android application. For the Raspberry Pi they will need a username and password to gain access.

To receive the push notification the user will need to ensure that the RPi and their mobile device with have internet connection.

2.3.11. Environmental requirements

The following requirements are vital to ensuring the systems runs and works as expected.

- An Android comparable device to run the application on.

- Internet connection for both the Raspberry Pi and the Android application.
- AWS IoT platform to handle the exchanging of messages.
- Dedicated power source for the Raspberry Pi.
- Moisture sensor connected to the RPi.
- House plants growing in pots with soil, to embed the sensor in.

2.3.12. Usability requirements

Understandability: The UI elements should be clear and easy to use and understand.

Operability: The system should be consistent and always functioning correctly. The interface actions and elements of the device application should be consistent. It should be easy enough for a child, or an elderly person to use. Users must be able to download the application associated with the system. This includes access to the Play Store and having a compatible device. They must also be able to insert the sensor in the soil and turn it on.

Attractiveness: The screen layout and colour should be appealing to users.

Integrity: The system should ensure that no data is manipulated unless permission has been granted for the data to be manipulated. The data should be consistent and correct at all times.

Learnability: The system should be self-explanatory and easy to use.

3.Interface requirements

User Interface: The interface on the Android application will first show buttons to select that correspond to different Raspberry Pis that monitor separate plants. After the user selects a Raspberry Pi they will be shown a page containing four buttons. The first button will enable them to connect to the selected Raspberry Pi. The other three buttons correspond to the three categories of plants they can monitor. When they click a category to monitor they will be shown a popup box contains information on that category and another two buttons, giving them the options to confirm monitoring or to close the popup window. When they choose a category to monitor the popup window will close and a message at the bottom of the screen will appear confirming that the plant is being monitored.

Hardware Interfaces: The hardware I will be using during the development of my system will be my own Apple Macbook Pro, the Raspberry Pi computers, Grove shields, connected to the Raspberry Pis, the grove moisture sensors, and the college computers. I will be doing the majority of the development on my own laptop and the Raspberry Pis.

Software Interfaces: The system requires Android Studio to develop, write, and compile the Java and XML languages to create the Android application, and Python IDLE to develop, write, and compile the Python code on the Raspberry Pi. For backup, and code version control purposes Git will be configured with these development environments.

Communication Interfaces: All data and messages transferred between applications used by the system will be done so through the AWS IoT platform.

3.1.GUI Mockups

These are the mock-ups I have created to illustrate the functional requirements and the flow of the Android application. These will be a guide for me when the development of the application begins, and are subject to change.

Screen 1

Screen 2

Screen 3

Screen 4

The Android application starts and the sync screen (screen1) appears. The user presses the sync button and waits for the app to identify available systems to connect to. On the next screen(screen 2) a list of the available devices will appear. The user may select one of these systems to connect to. If the app can successfully connect to the selected system, the next screen(screen 3) will confirm this with a message box at the top of the screen. Underneath this a list of the categories of plant available for the user to select will appear. The user then selects the appropriate category to monitor. The last screen to appear is another confirmation box with information telling the user that they will be notified when the plant needs to watered.

3.2.Application Programming Interfaces (API)

After a bit of research and deliberating I decided the most robust way to allow the RPi and Android to communicate would be to use a MQTT broker like AWS IoT. AWS has a rules engine whereby you could set up a rule to inform the Android device if the moisture readings being transmitted by the Pi drop below a certain threshold, and then the Android listener could push a notification to the user. AWS is also free to sign up for and to use the basic functionality.

MQTT is an M2M(Machine-to-machine)/IoT protocol for connectivity. It was designed as an extremely lightweight publish-subscribe messaging transport system. It is useful for connections with remote locations where a small code footprint is required and/or bandwidth is at a premium. It is used in a range of home automation and scenarios involving small devices. It is also ideal for mobile applications due to its small size, low power usage, minimised data packets, and efficient distribution of information to one or many receivers.

AWS IoT helps to collect and send data to the cloud from connected devices. It makes it easy to load and analyse the recorded information and provides the ability to manage IoT devices. The AWS IoT platform is a managed cloud plat-

form that lets connected devices easily and securely interact with cloud applications and other devices. AWS IoT has the capability of supporting billions of devices and trillions of messages, and can process and route those messages to AWS endpoints and to other devices reliably and securely.

The Raspberry Pi and Android application will communicate through an intermediate server, AWS. Both the Python script and the Android application will be configured to allow publishing to and subscribing from messages on AWS, through it's API.

When a user clicks to monitor a certain category of plant, Android will publish this category to AWS. When AWS receives this message the Python script will be able to read it as it is subscribed to the messages. Whatever category is sent by the Android application will determine which method the Python should run. The methods correspond to the three different categories of plants it is capable of monitoring.

When the readings from the sensor drop below a certain threshold, associated with that particular method, the Python script will publish a message back to AWS, which will invoke the push notification to the user's phone.

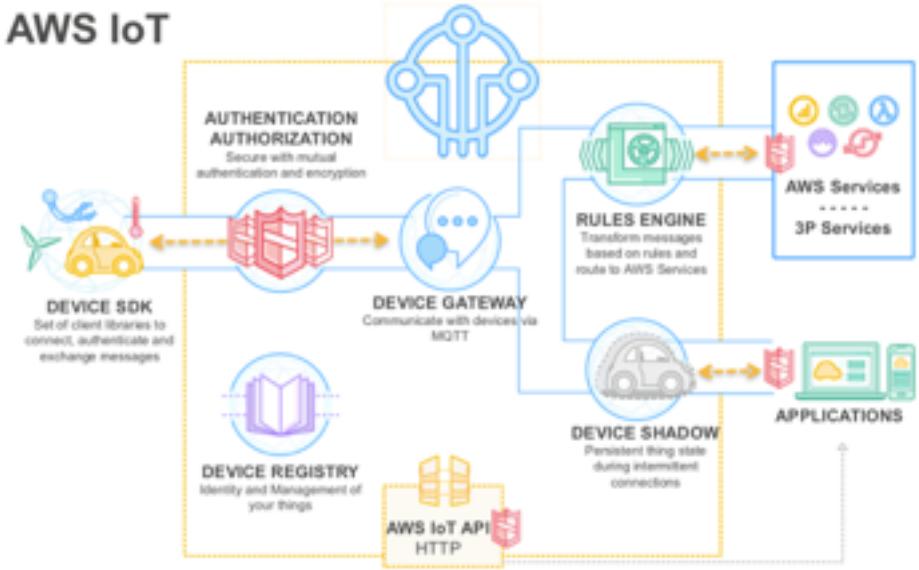
4.Design and System Architecture

This system uses a Raspberry Pi micro controller computer to monitor the hydration levels in a contained environment, i.e. the pots of household plants, and flags low readings to the user through the use of an online intermediate server, direct to an Android mobile application.

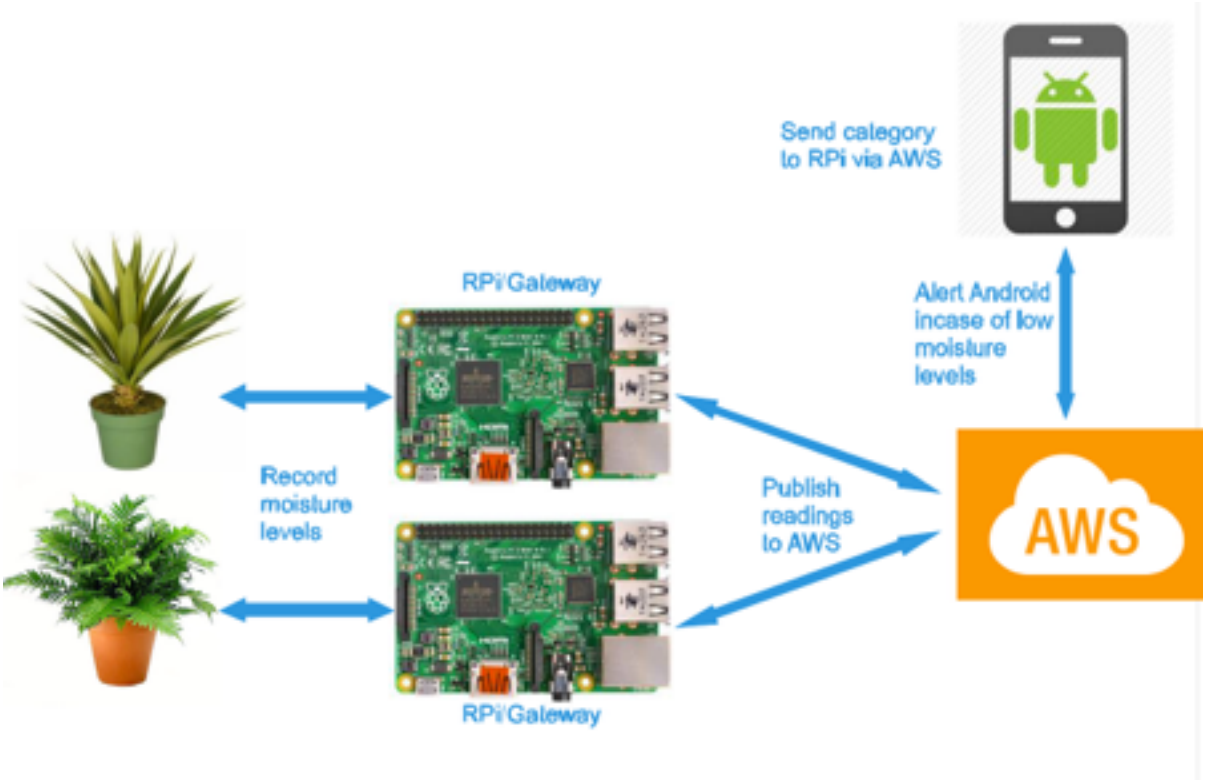
The Raspberry Pi is connected to a Seeed GrovePi+ shield, which sits on top of the Raspberry Pi and connects through the GPIO pins. This allows a GrovePi moisture sensor to be used to record the moisture content and feed it back to the Raspberry Pi.

If the readings recorded by the sensor reach a certain threshold on the Raspberry Pi, the Pi will then send an MQTT message to the intermediate server. The intermediate server that allows the Raspberry Pi and Android application to communicate is AWS IoT(Amazon Web Services Internet of Things).

I registered a “Thing” with this service called “JackThing”. This allowed me to publish messages to JackThing from my Android application and subscribe to JackThing from the Python script on the RPi to read these messages.



High Level Architecture Diagram:



Structure of the steps involved for using the system:

1. The first step of the process is to set up the system. This involves burying the moisture sensor into the soil of the plant pot. The sensor should be placed relatively close to the bottom of the pot, where the roots of the plant end.
2. Connect the sensor to a sensor input port on the RaspBerry Pi. This will allow the sensor to send information to the Pi to be compared with stored information.
3. Turn the system on. Connect the RaspBerry Pi to a power source, either a wall outlet using a wire and plug, or to a power bank us a micro USB cable.

4. Download the application that allows communication between the users device and the monitoring system. The application is only be available to Android users. The user must have an Android compatible device to download the app.
5. Once the application has been downloaded and successfully installed on the users device they will be prompted to sync the app to the system.
6. On completion of the sync, the app will be ready to receive information from the system monitoring the soil.
7. Users will then have an option to choose a category of plant from the application that suits the one being monitored. This will allow the RPi to send the notification appropriately.
8. Users wait for a notification to be pushed to their device from the Android application when the moisture level drops below a certain level, indicating their plant needs to be watered.

4.1.Implementation

To develop the Android application I used Android Studio. Android Studio is the official IDE(integrated development environment) for the Android platform.

The first thing I needed to do was to configure the AWS Mobile SDK for Android to allow the application to communicate with AWS IoT. To authenticate to AWS IoT Android used Cognito in conjunction with AWS IoT to create an identity (client certificate and private key) and store it in a Java key-store. After certificate and private key have been added to the key-store the app will use these for future connections.

There is a lot of code involved for configuring AWS with Android but here is a snippet of code showing the initial code and variables used in the configuration:

```
// ********************************** AWS CONFIGURATION ********************************** //
// IoT endpoint
// AWS Iot CLI describe-endpoint call returns: XXXXXXXXXX.iot.<region>.amazonaws.com
private static final String CUSTOMER_SPECIFIC_ENDPOINT = "aavki6aun7jrw.iot.eu-west-1.amazonaws.com";
// Cognito pool ID. For this app, pool needs to be unauthenticated pool with
// AWS IoT permissions.
private static final String COGNITO_POOL_ID = "eu-west-1:cc4c1b4b-57de-418e-a1a8-f6bd5f7a47d9";
// Name of the AWS IoT policy to attach to a newly created certificate
private static final String AWS_IOT_POLICY_NAME = "JackThing-Policy";

// Region of AWS IoT
private static final Regions MY_REGION = Regions.EU_WEST_1;
// Filename of KeyStore file on the filesystem
private static final String KEYSTORE_NAME = "iot_keystore2";
// Password for the private key in the KeyStore
private static final String KEYSTORE_PASSWORD = "password";
// Certificate and key aliases in the KeyStore
private static final String CERTIFICATE_ID = "default";
```

The main activity of the android app lets a user select between two Raspberry Pis to connect to(plants to be monitored).

```
Button first_plant;
Button second_plant;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_plant_pot_select);


    first_plant = (Button) findViewById(R.id.first_plant);
    first_plant.setOnClickListener(first);

    second_plant = (Button) findViewById(R.id.second_plant);
    second_plant.setOnClickListener(second);
}


View.OnClickListener first = onClick(v) → {
        Intent i = new Intent(MainActivity.this, first_plant.class);
        MainActivity.this.startActivity(i);
};

View.OnClickListener second = onClick(v) → {
        Intent i = new Intent(MainActivity.this, second_plant.class);
        MainActivity.this.startActivity(i);
};
```

When a user selects one they are shown a page allowing them to connect to that RPi, and to select a category of plant to monitor. Linked to these UI buttons, I created methods for connecting, and publishing the category of plant.

```
// Connect to AWS. Displays connection status in a text view on the UI.
View.OnClickListener connectClick = (v) -> {

        Log.d(LOG_TAG, "clientId = " + clientId);

        try {
            mqttManager.connect(clientKeyStore, (status, throwable) -> {
                    Log.d(LOG_TAG, "Status = " + String.valueOf(status));


                    runOnUiThread(() -> {
                            if (status == AWSIotMqttClientStatus.Connecting) {
                                tvStatus.setText("Connecting...");

                            } else if (status == AWSIotMqttClientStatus.Connected) {
                                tvStatus.setText("Connected");
                            } else if (status == AWSIotMqttClientStatus.Reconnecting) {
                                if (throwable != null) {
                                    Log.e(LOG_TAG, "Connection error.", throwable);
                                }
                                tvStatus.setText("Reconnecting");
                            } else if (status == AWSIotMqttClientStatus.ConnectionLost) {
                                if (throwable != null) {
                                    Log.e(LOG_TAG, "Connection error.", throwable);
                                }
                                tvStatus.setText("Disconnected");
                            } else {
                                tvStatus.setText("Disconnected");

                            }
                    });
            });
        } catch (final Exception e) {
            Log.e(LOG_TAG, "Connection error.", e);
            tvStatus.setText("Error! " + e.getMessage());
        }
};
```

The connection method corresponds to the “Sync to Pi” button on the UI. It allows connection to AWS IoT, using MQTT Manager. It renders the connection status in a text view on the UI, and throws and prints an exception if there is an error.

The send category methods send an MQTT message to the topic “JackThing” on the AWS platform with the value of “cat1”, “cat2”, or “cat3”. These methods correspond to the UI buttons for “Monitor” that appear in the popup windows. The RPi then reads this from AWS, runs the appropriate code and starts monitoring the moisture of a plant. A toast message is rendered to confirm that the message is sent and the plant is being monitored. If the method can not publish the message an exception is thrown and the error is logged.

Similarly to the Android application, the first thing I did with the Python script was to configure the AWS SDK for connecting to the AWS IoT platform from a device.

“The AWS IoT Device SDK for Python allows developers to write Python script to use their devices to access the AWS IoT platform through MQTT or MQTT over

```
View.OnClickListener cat1Click = (v) → {
        LayoutInflater layoutInflater = (LayoutInflater) getBaseContext()
                .getSystemService(LAYOUT_INFLATER_SERVICE);

        View popupView = layoutInflater.inflate(R.layout.popup, null);
        final PopupWindow popupWindow = new PopupWindow(
                popupView,
                ActionBar.LayoutParams.WRAP_CONTENT,
                ActionBar.LayoutParams.WRAP_CONTENT);

        final String topic = "JackThing";
        final String msg = "cat1";

        Button btnSend = (Button) popupView.findViewById(R.id.btnSend);
        Button btnClose = (Button) popupView.findViewById(R.id.btnClose);

        btnSend.setOnClickListener((v) → {

                try {
                    mqttManager.publishString(msg, topic, AWSIotMqttQos.QOS0);
                    Context context = getApplicationContext();
                    CharSequence text = "Monitoring plant category 1";
                    int duration = Toast.LENGTH_SHORT;

                    Toast toast = Toast.makeText(context, text, duration);
                    toast.show();
                    popupWindow.dismiss();
                } catch (Exception e) {
                    Log.e(LOG_TAG, "Publish error.", e);
                }
        });
        btnClose.setOnClickListener((v) → { popupWindow.dismiss(); });

        popupWindow.showAtLocation(popupView, Gravity.CENTER, 0, 0);
};
```

the WebSocket protocol. By connecting their devices to AWS IoT, users can securely work with the message broker, rules, and the device shadow (sometimes referred to as a thing shadow) provided by AWS IoT and with other AWS services like AWS Lambda, Amazon Kinesis, Amazon S3, and more.” - Unknown, GitHub(2017).

To allow connection I used X.509 certificate credentials for certificate-based mutual authentication. I used the AWS console to generate and download the certificate and private key. I then needed to specify where these were stored when initialising the Python client.

```
# //  *************** AWS CONFIGURATION ************************ //
# Code sourced from: https://github.com/aws/aws-iot-device-sdk-python
myAWSIoTMQTTClient = AWSIoTMQTTClient("basicPubSub")
myAWSIoTMQTTClient.configureEndpoint("aavki6aun7jrw.iot.eu-west-1.amazonaws.com",
8883)
myAWSIoTMQTTClient.configureCredentials("/home/pi/GrovePi/Software/Python/root-CA.crt",
"/home/pi/GrovePi/Software/Python/JackThing.private.key",
"/home/pi/GrovePi/Software/Python/JackThing.cert.pem")

# AWSIoTMQTTClient connection configuration
myAWSIoTMQTTClient.configureAutoReconnectBackoffTime(1, 32, 20)
myAWSIoTMQTTClient.configureOfflinePublishQueueing(-1)  # Infinite
offline Publish queueing
myAWSIoTMQTTClient.configureDrainingFrequency(2)  # Draining: 2 Hz
myAWSIoTMQTTClient.configureConnectDisconnectTimeout(10)  # 10 sec
myAWSIoTMQTTClient.configureMQTTOperationTimeout(5)  # 5 sec

# Connect and subscribe to AWS IoT
myAWSIoTMQTTClient.connect()
myAWSIoTMQTTClient.subscribe("JackThing", 1, customCallback)
time.sleep(2)
```

When a message is received from AWS the customCallback method is run. This method stores the message(which will be the category of plant to monitor sent from the Android app) in a global variable called “category”. Using this variable it would be able to determine which method to run.

```
# Custom MQTT message callback. Called when a message is recieved
# Stores the message in a global variable and calls the method for
# determining which method to run.
def customCallback(client, userdata, message):
        print("Received a new message: ")
        print(message.payload)
        print("from topic: ")
        print(message.topic)
        print("--------------\n\n")
        global category
        category = message.payload
```

I then created a method for recording moisture readings from the sensor for a plant that required very little water, the sudo code for this was as follows:

sensor = 0

moisture = readings from sensor

if moisture < recommended level for a plant that needs very little water

send message to AWS

```
def cat1(): # Low moisture required
    sensor = 0
    while category != "cat2" and category != "cat3":
        moisture = grovepi.analogRead(sensor)
        print "cat 1"
        time.sleep(5)
        if moisture < 200:
            myAWSIoTMQTTClient.publish("JackThing/water", "water", 1)
```

Then I created similar methods for plants that required a medium amount of water and plants that needed a lot of water. The sudo code is as follows:

sensor = 0

While category is not equal to “cat2” and category is not equal to “cat3”

moisture = readings from sensor

if moisture < recommended level for a plant that needs very little water

send message to AWS

Verifying that this worked, I created the other two methods for categories 2 and 3. 2 requiring a medium amount of water and 3 requiring a lot of water.

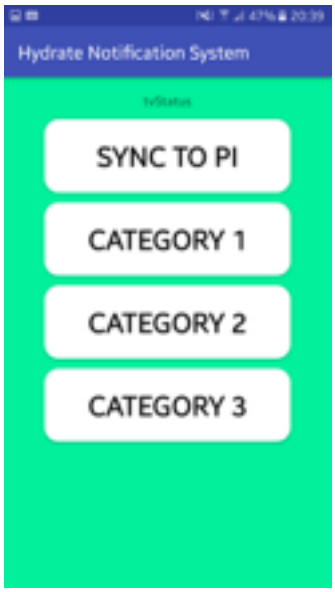
```
def cat2():
    sensor = 0
    while category != "cat1" and category != "cat3":
        moisture = grovepi.analogRead(sensor)
        print "cat 2"
        time.sleep(5)
        if moisture <= 400:
            myAWSIoTMQTTClient.publish("JackThing/water", "water", 1)

def cat3():
    sensor = 0
    while category != "cat1" and category != "cat2":
        moisture = grovepi.analogRead(sensor)
        print "cat 3"
        time.sleep(5)
        if moisture < 600:
            myAWSIoTMQTTClient.publish("JackThing/water", "water", 1)
```

4.2. Graphical User Interface (GUI) Layout

This first screenshot shows what a user sees when they first open the application. It contains two buttons, each representing a plant to monitor.

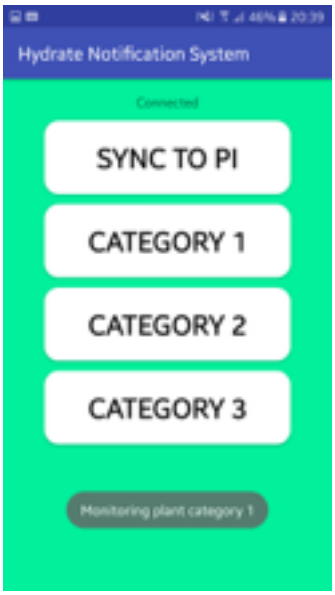
After they click one of these UI buttons they are presented with a page to connect to the Raspberry Pi that is linked with a particular plant.



They are also presented with a choice of the three categories they can choose from to monitor.

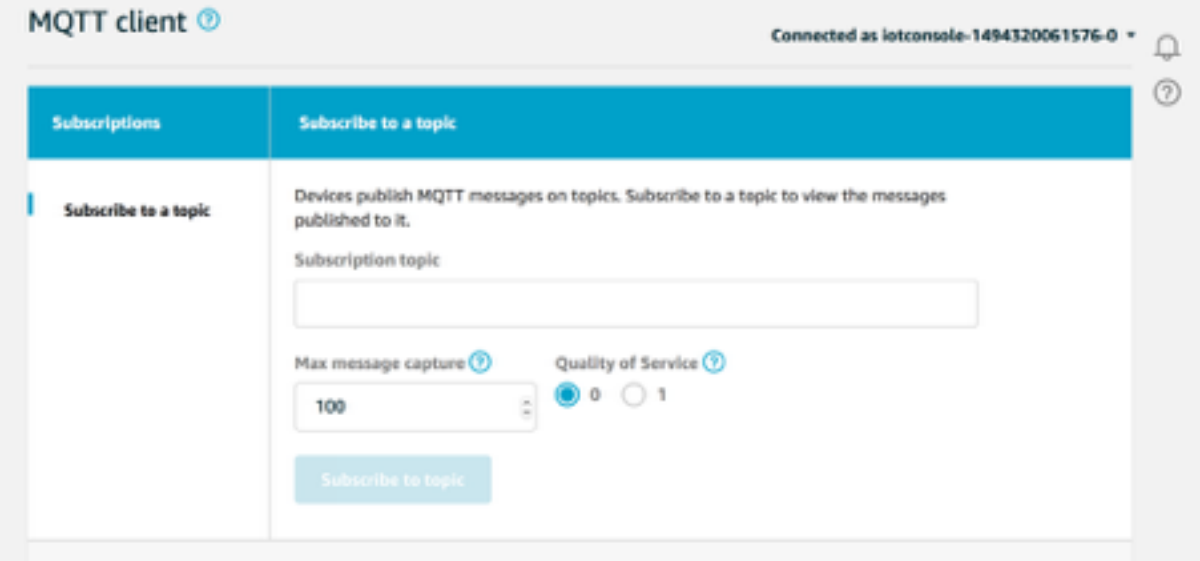
On clicking a category option a popup window will appear describing the category and a choice of buttons for monitoring that category of plant or closing the popup window.

If they choose to monitor that category of plant the popup window will disappear and a message will appear at the bottom of the screen confirming that the system is now monitoring.



4.3. Testing

For testing of messages being successfully sent and passed from Android to AWS, I was able to use the testing console provided by the AWS IoT platform. This test console acted as an MQTT client. I could subscribe to the “Thing” topic I had created to receive the messages sent by Android. This functionality was very simple to use, and provided great use for testing. Here is an image of the testing interface.



The process was the same for testing messages sent from the Python script.

When testing a message was successfully received by Android(read from AWS), I logged the incoming message and printed it to the output console. This allowed me to visually see the message was being received and had been converted to the correct format to be used by the Android application.

For testing the successful receiving of message on the Python script I printed the message read from AWS in the custom callback method, along with the topic it was sent from, to the terminal window.

For testing purposes to ensure the moisture sensor was working I wrote the recordings from the sensor to a txt file using a buffered writer implement in the

Python script. The Python script stored the recordings on a separate file on the RaspBerry Pi.

Testing average application response time and peak response time of methods and the system was done using a timer in milliseconds. The average response time gives you a sense of performance from the user's perspective, the peak response time metric will help identify areas where performance could be improved.

Testing for the midpoint:

This was the initial testing of the system recording moisture sensor readings and posting them to www.dweet.io. The following code was implement for the midpoint presentation prototype, but was subsequently changed.

The Python script stored on the Raspberry Pi will contain a method for accessing the port to which the sensor is connected, a method to store the readings in an array, a method to post to dweet.io and a method to write the readings to a txt file among other methods.

For testing purposes this method shows how to access the sensor and to return readings:

```
def getMoisture():
    moisture_sensor = 0
    moisture_value = grovepi.analogRead(moisture_sensor)
    return(moisture_value)
```

This method will post an array of the stored readings from the sensor to dweet.io. Dweet will be listening for in coming data sent from 'JacksReadings':

```
19
20 def post(dic):
21     thing = 'JacksReadings'
22     print dweepy.dweet_for(thing, dic)
23
24 def getReadings():
```

This method is made up of a while loop. Each time it scans for readings it will first store the new readings into an array called dict, then it will buffer this to a comparable data type to be written to the txt file. It opens the backup text file, and stores the data along with a date and time of the recording. Then it closes the backup file. After this it posts the same array to dweet.io then it repeats the process every 2 seconds.

```
while True:
    dict = getReadings()
    #Saves data from the senors to a text file
    stringToWrite = str(dict)
    backup = open('raspberryReadings.txt', 'a')
    timeToWrite = datetime.datetime.now().strftime(dateString)
    backup.write(timeToWrite + " " + stringToWrite + '\n')
    backup.close()
    #Post the data to Dweet
    post(dict)
    time.sleep(2)
```

4.4. Evaluation

Throughout the project I will be evaluating the time / cost of the idea and it's viability for home use, and compiling this information for a report on the benefits of its implementation set against the drawbacks. I have set milestones for the different phases of the project, and will review each phase on its completion date. These milestones are indicated on my Gantt chart (attached to the Appendix in the project plan section) as the end dates of the phases; initiation, planning, execution, performance and monitoring. The key indicators for progress towards the objectives of the project will be the milestones. At each milestone I will gauge the percentage of the phases completion, and can asses whether the objective has been met.

Monitoring and evaluating the project's progress and eventual outcome will be done throughout the project by review of the sub-phase milestones, as indicated on my Gantt chart; design, procurement, etc. A review of the sub-phase at each

milestone date will indicate the progress of the project and give a good indication of the outcome of the project as a whole.

5.System Evolution

This system will be capable of evolving over time. The user will be able to add extra Pi's with attached moisture sensors to monitor individual plants around their home, office etc. The Android application will be capable of handling more than two systems. Also, there will be a capability for adding different sensors to the system that a user may want to use, such temperature or humidity. Using such sensors I would be able to determine optimal conditions for certain plants to grow.

6. Conclusions

The advantages of this system are that people could potentially save money by not throwing out their house plants because they have died, due to dehydration. This is also an advantage for the plant itself, as long as they are placed in an area where they can receive sufficient amounts of sunlight, they will be able to thrive due to this constant adequate levels of water.

The system could be manufactured in Ireland potentially creating a numerous amount of jobs, depending on the demand for the system.

The system could be further developed to be used on farms in the agricultural industry, to monitor friends of crops such as potatoes. It could also be used commercially in glasshouses for growing tomatoes and other such plants.

The only disadvantage of the system that I have concluded would be the initial cost to develop it. But, if the demand for the system grows and mass production is required, the cost would drop substantially, both for developing it and for the user to buy it off the shelf.

7. Further development or research

With further research, the Android application could contain and list as many types of plants as required to cover the vast majority of house plants. This would allow a user to pick the exact plant that they wish to monitor. Each plant on the application could contain interesting information, fun facts, and optimal conditions for growth.

Using extra sensors to scale the project, and to record more information relating to the growth of plants, such as temperature sensors, humidity sensors, light sensors, and PH level sensors, the system could be further developed to monitor the optimal growing conditions of particular plants in houses, or commercial glasshouses. Farmers could also use it for monitoring and notification purposes for optimal crop growth conditions.

8. References

Unknown. (2017). *DDS and MQTT.* Available: http://www.prismtech.com/vortex/technologies/dds-and-mqtt?gclid=CMXa1-774NMCFaO87QodUkAAug. Last accessed 12th March 2017.

Unknown. (2000). *Recommended methods: 5. Usability requirements .* Available: http://www.usabilitynet.org/trump/methods/recommended/requirements.htm. Last accessed 19th Dec 2016.

Unknown. (2017). *AWS IoT.* Available: https://aws.amazon.com/iot-platform/how-it-works/. Last accessed 20 Jan 2017.

Unknown. (2017). *Android PubSub Sample.* Available: https://github.com/awslabs/aws-sdk-android-samples/tree/master/AndroidPubSub. Last accessed 15th Jan 2017.

Unknown. (2017). *Android Studio.* Available: https://en.wikipedia.org/wiki/Android_Studio. Last accessed 22 Apr 2017.

AWS. (2017). *Publish/Subscribe Policy Examples.* Available: http://docs.aws.amazon.com/iot/latest/developerguide/pub-sub-policy.html. Last accessed 12th Feb 2017.

Unknown. (2017). *SDK for connecting to AWS IoT from a device using Python..* Available: https://github.com/aws/aws-iot-device-sdk-python. Last accessed 2nd Dec 2016.

Unknown. (2012). *Make 2 functions run at the same time.* Available: http://stackoverflow.com/questions/2957116/make-2-functions-run-at-the-same-time. Last accessed 15th Feb 2017.

Unknown. (2017). *grove_moisture_sensor.py.* Available: https://github.com/DexterInd/GrovePi/blob/master/Software/Python/grove_moisture_sensor.py. Last accessed 10th Nov 2016.

Unknown. (Unknown). *Python.* Available: https://www.raspberrypi.org/documentation/usage/python/. Last accessed 21st Jan 2017.

9. Appendix

9.1.Project Proposal

1. *Objectives*

1. Ensure house plants have an adequate amount of water, all the time.

2. Monitor moisture levels of a household plant's soil.

3. To notify the user of the system when a house plant needs to be watered.

2. *Background*

I have noticed many house plants are neglected due to the busy lifestyles that we all lead nowadays. Often a lot of money is spent on beautiful house plants which are then, unfortunately, thrown out after a year or so due to over or under watering. So I decided to develop the House Plant Hydration Notification System to tackle this problem.

3. *Technical Approach*

To build and create this system I will be using a mixture of Android Studio, a Raspberry Pi, and GrovePi+ sensors. Android Studio will be used to develop the user interface, for storing information, and for sending notifications to the users phone. The Raspberry Pi will be used to record information from the GrovePi+ sensors that are embedded in the plants, and the GrovePi+ sensors are sensors that are enabled to interact with the Raspberry Pi.

Android Studio is an Integrated Development Environment(IDE) for Android platform development. It is free to use under the Apache License 2.0.

A Raspberry Pi is a credit card-sized single-board computer. They are used to help yeah computer science in schools and in developing countries. It is used for many computer projects and for a lot of things your desktop computer can do, like spreadsheets, browsing the internet, and playing games. It can also be used

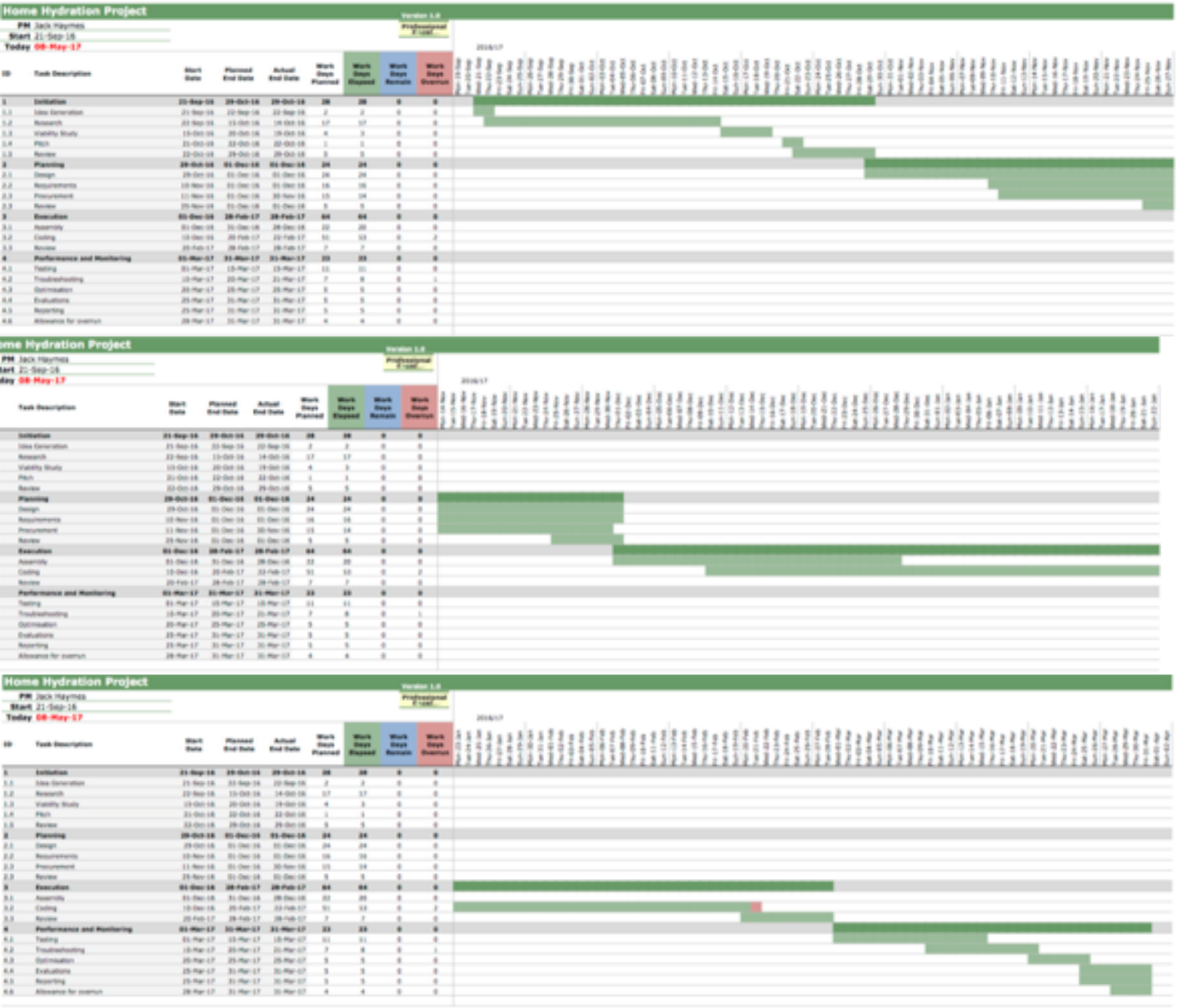
in conjunction with sensors that plug into the USB slots to record information, and relay it back.

GrovePi+ is a plugin kit for Raspberry Pi. It includes a variety of sensors that can be used, including the humidity/temperature sensors I will be using for this project. These sensors allow you to turn any object into an internet-connected device and relays live information.

4. Special resources required

- Raspberry Pi.
- GrovePi+ shield and sensors.
- Android Studio.
- Python programming language.
- Java programming language.
- Amazon Web Services
- XML.

5. *Project Plan*



6. Technical Details

Developing the user interface part of the project and the application itself will be created and managed with Android Studio. The languages I'll be using here are XML and Java. XML is used for the application layout and presentation, while Java is used for the functionality and API calls. Java has an extensive amount of libraries available to use with it.

For the Raspberry Pi and for capturing data from the sensors ill be using Python and the associated libraries.

7. *Evaluation*

After I have a finished working system, I will test it with real technical data. I will install the app, I will sync the sensors, I will match each sensor with a category of plant. I will then use a plant with very dry soil, a plant that is in need of being watered. I'll record a reading from it, then I'll slowly water it. If the sensor is working correctly, the moisture reading on the app should rise to a suitable level. I'll need to wait for the soil to start drying out and for the moisture level to drop under the suggested level before I can properly test that the automatic notification is working. Once i have tested with one I will add another sensor with another plant and test again.

If there are any flaws in the system I will address them and test again until it is working perfectly. Also, if any room for improvement is noticed, this will also be addressed.

10.Monthly Journals

September:

Student name: Jack Haymes

Programme: BSc in Computing

Month: September

This month, I was told my original project idea was a bit simple to be undertaken as a 4th year project. I haven't been able to come up with anything suitable myself yet, so I was looking through the suggested projects that were posted on my Moodle page. I like the look of a project that monitored dampness in a room using a Raspberry Pi. I especially liked this because it would kind of tie in with my IoT specialisation. Sadly though, this project was already taken by another student. I have been brainstorming ideas by myself, and also I will be meeting with Lisa Murphy to come up with an idea. Hopefully I will settle on one this week.

Unfortunately I haven't been able to start anything yet, so no need for any changes.

Supervisor meeting is yet to be held. I will be Meeting with Lisa Murphy this week.

October:

Student name: Jack Haymes

Programme: BSc in Computing

Month: October

This month, I was able to decide on a project. After a lot of brainstorming and research to see what would be achievable I've decide to attempt a project that will notify a user when their house plants need to watered.

I managed to get the original draft of my project proposal uploaded. This was a challenge install to get it uploaded on time as I only decided on my project late the night before the submission date for the proposal.

I decided on the software and hardware that will be used to develop this project.

I'm yet to attend my first supervisor meeting as my first one that was suppose to be help on Friday evening before the reading week seemed to be cancelled. Once I can sit down with Ralph and get some direction on how to tackle starting my project I can start making some progress.

No intended changes yet as I haven't started anything solid, just thoughts and ideas.

November:

Student name: Jack Haymes

Programme: BSc in Computing

Month: November

This month was mostly spent thinking about how I would develop my project. I thought the best thing to do first, before I started the development process, would be to ask prospective customers and users what they thought of the concept and if they liked it, how they think it should function.

I received some great feedback. This gave me a basis to start. I researched different devices capable of monitoring levels of moisture by means of a sensor. The best and most practical solution was to use a RaspBerry Pi computer, a Grove Pi moisture sensor, and a mobile application built on the Android Platform using Android Studio.

After deciding how I would create and implement the system I needed to evaluate the functional and non-functional requirements. I had a requirement specification to complete to upload to Moodle. This would then be added to my technical report along with all of my other documentation. Most of this documentation will need to be updated as time goes on as I suspect there will be changes and unforeseen obstacles and challenges that may require changes to the system. I could also need to add functionality, it is only in it's preliminary stages.

I received the date for my mid-point presentation, it will be on the 14th of December at 11:30am. For this, I need to have a prototype developed and I need to prepare a powerpoint presentation. I'm not quite sure how much functionality will be expected for the prototype, I will need to speak with Ralph (my supervisor) about it. The technical report is also due the weekend before my presentation. Again, this is a live in document that can be updated during the course of the year.

The next step, after the prototype, would be to first set up a GitHub account so I can easily manage my code and rollback to previous versions if needed. Then in the new year I can actually start the development of the application that will communicate with the RaspBerry Pi.

I felt, it was definitely a good idea to talk to potential users of the system before I started any development, it gave me great guidance in tackling the project from the outset. I was able to determine by their suggested needs and wants what software and hardware should be used.

Date of Supervisor Meeting: 01-12-16

Items discussed: Structure of the system and Requirements Specification documentation.

Action Items: Market research and research components to use in the development.

December:

Student name: Jack Haymes

Programme: BScH in Computing

Month: December

This month was a busy month for me. I had to complete a prototype for my project to present, as well as uploading the up to date technical report containing everything relating to my plant hydration notification system.

For the prototype I wrote a simple Python script that recorded data from a moisture sensor and stored it to a text file on the Raspberry Pi. I added functionality in this script so that the Pi could connect to dweet.io and post the readings live. Dweet allows you to post findings and readings and view them live from devices capable of doing so. Coming up to the date of my

presentation I managed to accidentally break the moisture sensor I was using for the prototype. I went online to order another one and check the appropriate shops around the city but couldn't acquire one before the date of my presentation. This left me in a bit of a panicked state. I decide I would just have to use another type of sensor for testing purposes and explain my misfortune on the day. Thankfully, this was OK and with the prototype and the slides I had prepared, the presentation seemed to go well and I received some valuable, constructive feedback to help me in the next steps of the project.

I didn't have much time to concentrate on this project after the presentation as I had a lot other deadlines I needed to meet and also my exams were starting the week after Christmas, so I need to dedicate my time to meeting these deadlines and studying for the fast approaching exams. I did however manage to set up a Git Hub account which I will be using to manage my Python code.

The next step is to create the final wireframes for the mobile application. I will be using an open source online application to complete them. After this, I will begin the development of the mobile app with Android Studio while simultaneously updating the Python script written for the prototype. I will need to research in depth how i can allow the Raspberry Pi and the Android application communicate with one another.

There were no meetings held this month between myself and my project supervisor, Ralph Beirg.

January:

Student name: Jack Haymes

Programme: BScH in Computing

Month: January

This month was busy with exams but I managed to find time to start developing the actual production system. Using the feedback from the presentation I decided on a couple of small changes I would make. After some further research I decided there would also be some bigger changes from the working prototype. One of these was to use the AWS IoT platform to manage the sending and receiving of messages between the Raspberry Pi and the Android Application. This seemed a lot more robust and appropriate for my needs. I spent a lot of time researching the AWS IoT platform and discovering how I could make it work for my needs.

I started to develop the Python code on the Raspberry Pi first. I needed to configure the AWS connection settings to allow the Python script to subscribe and publish messages to AWS. After a while and a bit of trouble shooting I managed to get it working. I was able to test this by sending and receiving test messages to and from the AWS IoT console. I then needed to think about how I would approach the coding for the methods for handling a message from AWS and determining what method a particular message would run. I needed a different method associated with each category of plant. The message received from AWS would determine which method to run.

I had ordered a new moisture sensor after the incident before the presentation. This had now arrived and was ready to be used. After testing it was working the I wanted I decided it was time to start developing the Android side of the project.

I know the Python code will need to be revised and altered as the project proceeds, but for now I am happy that it is working as expected.

I will use the previously created wireframes to model the design for the Android application. The app will be developed using Android Studio and configured with AWS IoT. I will also need to think about how to create the push notification functionality.

February:

Student name: Jack Haymes

Programme: BScH in Computing

Month: February

February was mostly dedicated to designing and building the Android application. I had a bit of experience building basic Android applications from my Multimedia and Application Development module in the first semester of fourth year, so I used this experience as a starting point. After I was happy with the designs I had sketched up I began the development in Android Studio. Similarly to the Python script on the Raspberry Pi, I first needed to configure AWS with the application to allow publishing and subscribing, to and from messages on the platform.

After I had successfully configured AWS and tested it using the AWS IoT platform, I began to develop the other functionalities of the app.

The Android application consisted of a welcome page with the option to monitor the first plant - linked to the first Raspberry Pi, or the second plant - linked to the second Raspberry Pi. When either option was clicked, a new page is shown with the options of different categories of plants to monitor. Each category is related to how much water a particular plant

needs. When a category is click, a popup appears describing the category and certain plants that would be covered by it. The user can then choose to click the monitor button which will send the request to AWS and in return will be read by The Python script, which will start the monitory process of a particular plant.

While I was developing, I decided I would use an extra Raspberry Pi in the project so the system would have the capability of monitoring two plants at once. When I had the functionality to monitor two plants configured it would be simple to add more Raspberry Pis to the system to allow monitoring of any number of plants, if I wished.

My IoT lecturer, Dominic Carr, had supplied me with the first Raspberry Pi so I asked him if he would have a second one available I could borrow, to save me buying another one. He was in a position to lend me the second one, which was great, but he did not have another WiFi dongle that would be needed by the Raspberry Pi for internet connection to AWS, so I would have to buy one from Maplins.

While testing the second Raspberry Pi for internet connection with the first WiFi dongle I already had, I realised that it wouldn't work. After hours and hours or trouble shooting and following online solutions, to no avail, I concluded that there must be a problem with the software installed on the SD card of the Raspberry Pi. I decided to copy the software from the first card onto the new card, completely replacing the software installed on it. This process wasn't as straight forward as I thought it would be, but after a while and with some persistence I managed to successfully copy and replace the operating system. After this, I retested the WiFi dongle, and to my delight, it was working perfectly.

March:

Student name: Jack Haymes

Programme: BScH in Computing

Month: March

March was mostly spent bringing everything together and testing the system worked as planned. After I was happy with the final design of the Android application and the structure of the Python code on the Raspberry Pi I spent time with potential users to receive feedback. The feedback I received was all very good. I did receive some suggestions which could help in further development of the project but this was not in the scope of this initial project and I would not have time to implement the suggestions.

I still need to complete the testing and analysis design report, which will go with the overall technical report to be printed and bound. This will be tackled in the text month using feedback I have already received, research and gathering specifications online, and physical testing reports conducted by myself.

I needed to design the poster for our project showcase. I first approached this by drawing mockups on an A4 piece of paper. The poster needed to briefly explain my project and get the idea across succinctly, it need an image of myself, and images of the logos to represent the technologies I used in the development of the system. I also need a pictures to represent a dead plant(what would happen without using this system), and a nice flourishing plant(how the plant would thrive while using this system). I took images of the technologies used, and of the plants from Google to use on the poster. I had a picture of myself already that was taken by the college for my Linked In account. To create the poster I used a free version of Photoshop. All that is left to do is to get the poster and technical report printed and bound.

All that is left to do, after the final testing, is to prepare a presentation to present the system. This will consist of several slides showing and describing my project.

Overall the whole process of researching, designing, and developing this project has been a very valuable and rewarding experience.