Project Report

ecoWater  
<https://ecowater2017.wordpress.com/>

Raymond Flanagan  
s00081696  
  
PRJ400  
  
B.Sc. (Hons) in Computing (Software Development)

Summer 2017  
  
Supervisors  
Shaun McBrearty  
Colin O’Connor

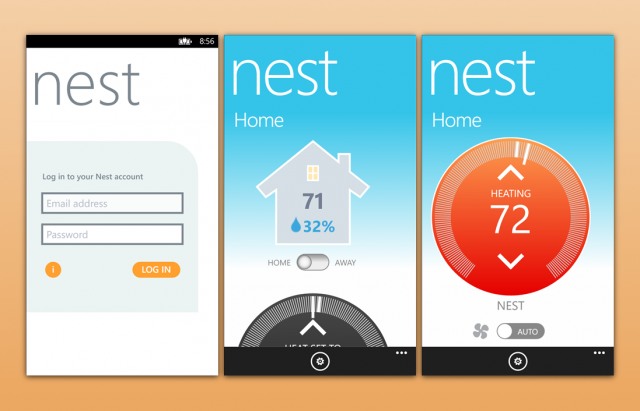
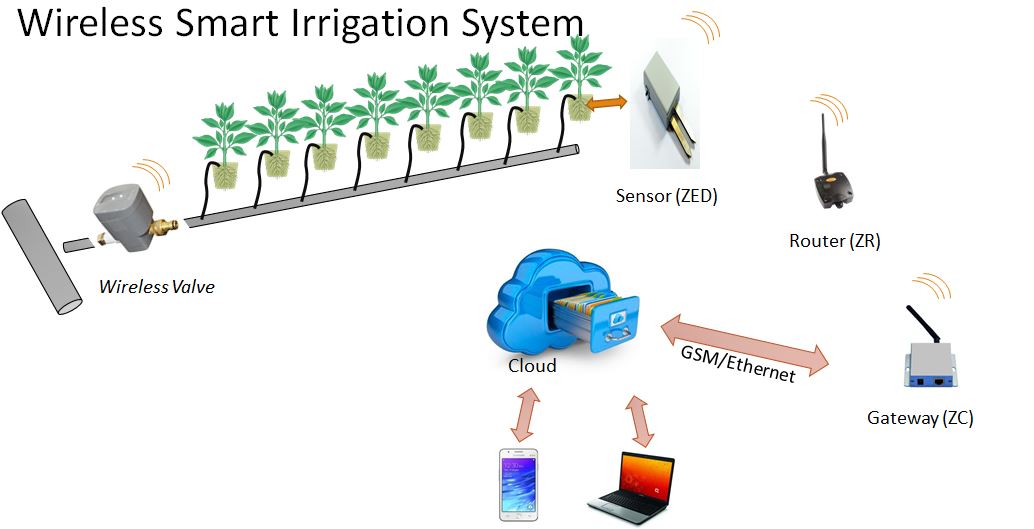
**CHAPTER 1 Overview  
  
1.1 System Synopsis**A system that allows a user to turn off or on a water pump remotely via a smartphone app, as well as monitor and display the water level a given container is currently at and the temperature of the water or other liquids. A web service also provides SMS notifications to users’ phones, SMS control of the system and intelligent automatic switching on or off of the pump if the tank is nearly empty or full.  
  
**1.2 Target Market**Home owners who wish to control or monitor their water pump/water system remotely.

**1.3 Secondary Uses**

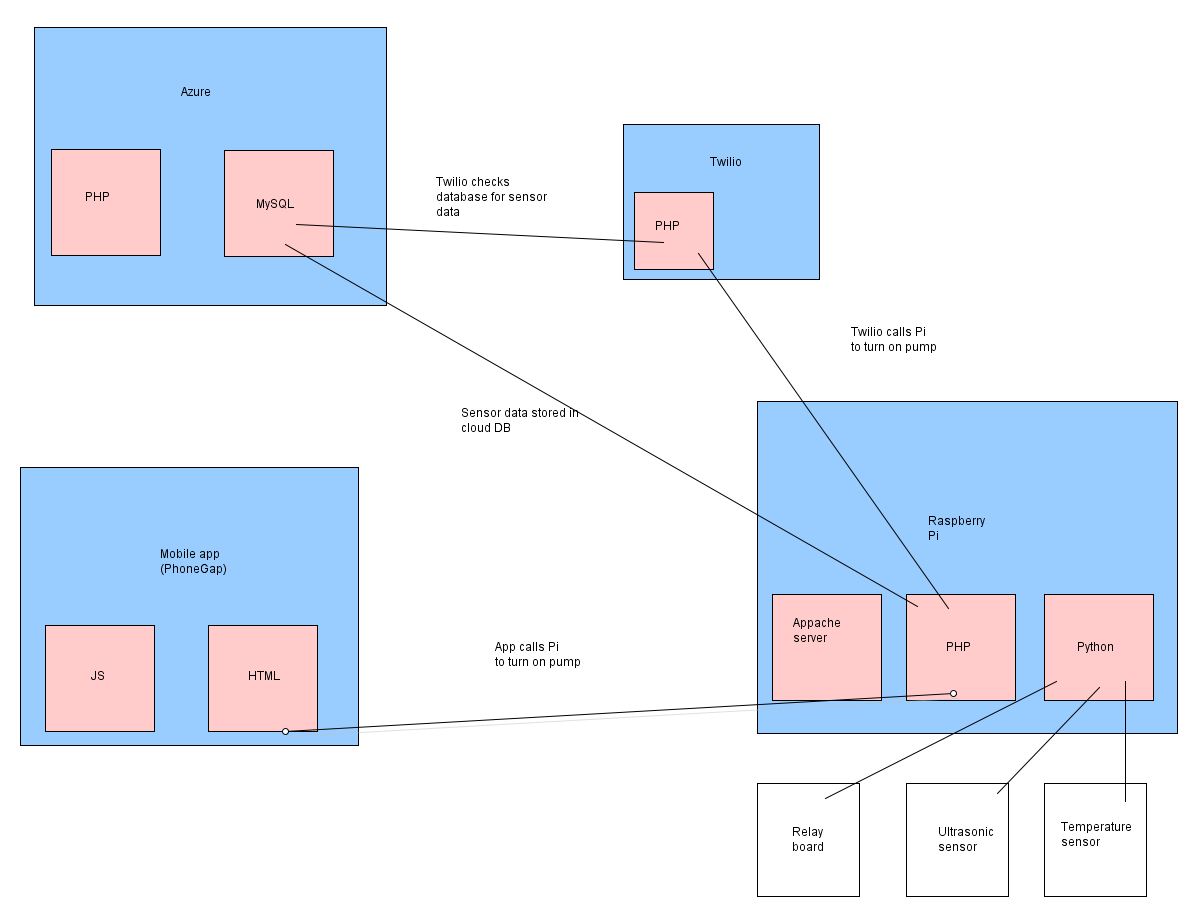
In drought-affected areas, the system can be used to conserve and monitor water supplies – the SMS functionality means the system can operate even in areas with poor internet by leveraging cellular networks.

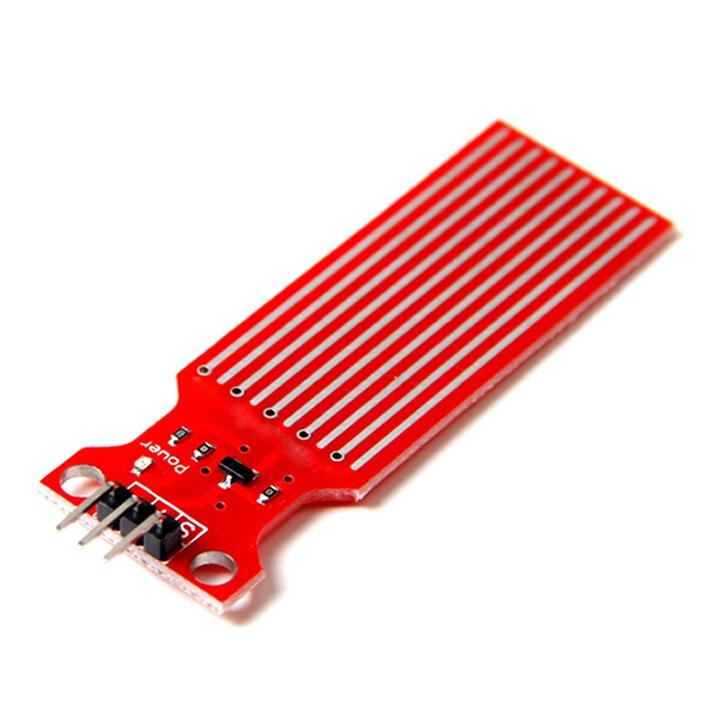
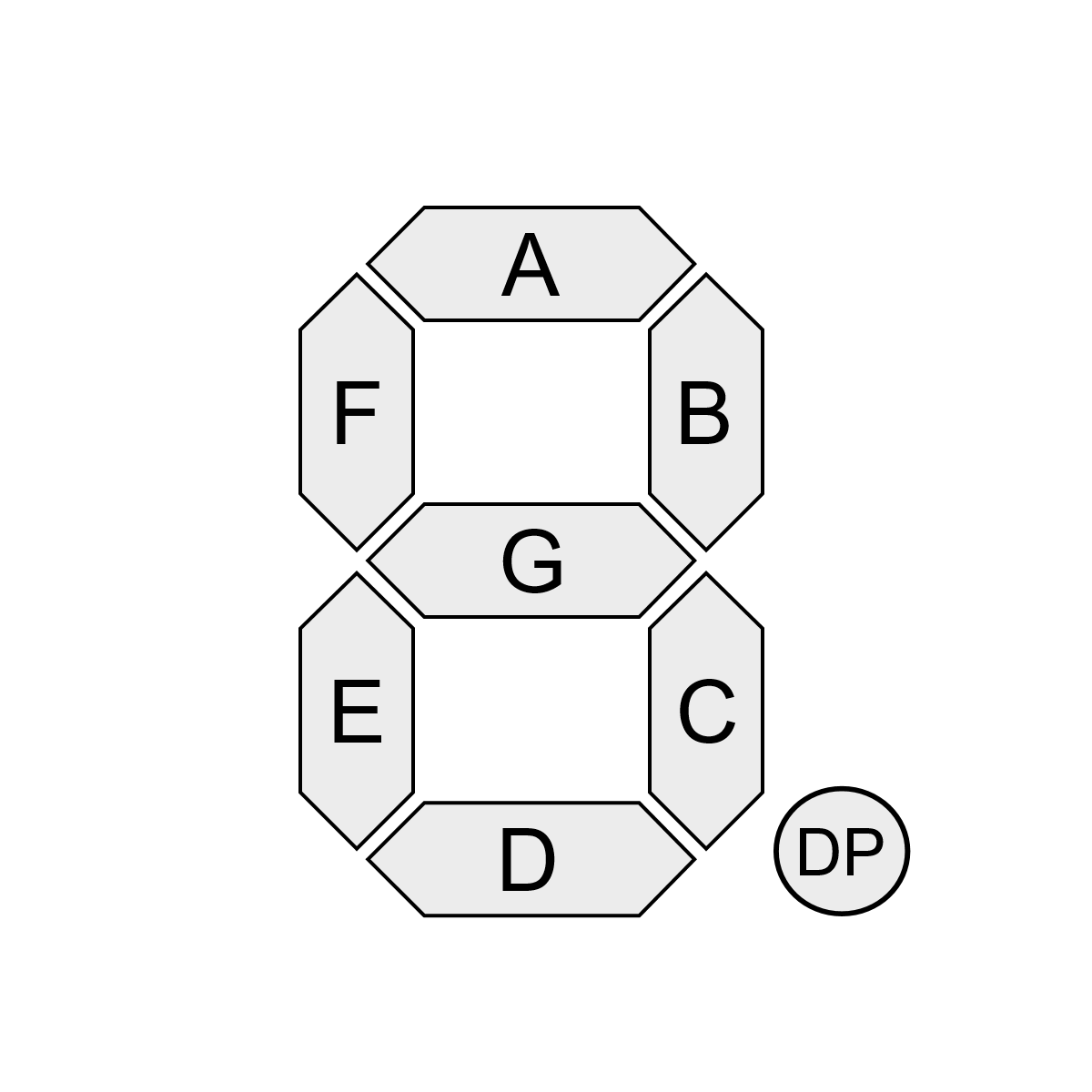
To monitor milk temperatures – if the milk container becomes too hot the system could alert the user of this remotely, thus preventing the milk from being spoiled.   
  
Monitoring cattle or human blood samples that need to be kept at a low enough temperature so as not to spoil the samples  
  
With regards home use specifically, allowing a household to conserve water, monitor water levels and be notified in case of potentially frozen pipes.  
  
  
**1.4 Need for the System**

Only 1% of the total water resources on the planet are available for human use – 97.5% of the world’s surface water is salt water, while over two thirds of the remaining 2.5% that constitute the freshwater supply are frozen in ice caps and glaciers. To put it simply, clean, usable water is scarce.  
  
Even if we focus exclusively on Ireland, instances of water shortages are a regular occurrence – summer 2017 left 70,000 residents in Meath and Louth without running water for over a week – while water rationing is a favoured tactic of many county councils in summer. This situation is only becoming worse with climate change reducing fresh water supplies.  
  
In Ireland, the attempts by the Government body Irish Water to install a nationwide household water metering and charge-per-usage system indicates a future where the more water household uses, the costlier it will be.  
  
With all this in mind, there is an opportunity for a system that can monitor water tanks in the home, or elsewhere, to provide power benefits. If a system can measure how full a water tank is and then intelligently turn off a pump when the tank is nearly full it will conserve the amount of water the system uses – this benefits the environment as a whole, but in a future where water usage charges are inevitable it can also save a home owner money.  
  
Likewise, if a system can check the water temperature and notify the user if the water or pipes are in danger of freezing it can allow the user to remedy the situation before it becomes critical and, potentially, costly and time consuming to do so.   
  
Providing SMS notifications and SMS control allow any phone – smart or otherwise – that can send texts interact with the system. This means a system that is robust and build for users and environments where internet connectivity can vary wildly. A more user-friendly mobile application for smartphones will also provide control over the pump and surface data about the water system that is being monitored.  
  
To summarise, the stated requirements for a successful system are as follows:  
  
- An IoT device connected to a pump and sensor that can record sensor data and control the pump  
  
- A cloud-based web service that can host a database that contains user and sensor data, as well as serve that data to other applications.  
  
- A mobile application that users can log into to control their pump and see sensor data  
  
- SMS notifications, allowing the system to notify users of issues that need their attention  
  
- A way to control and request information from the system via SMS so the system is accessible in almost all situations.

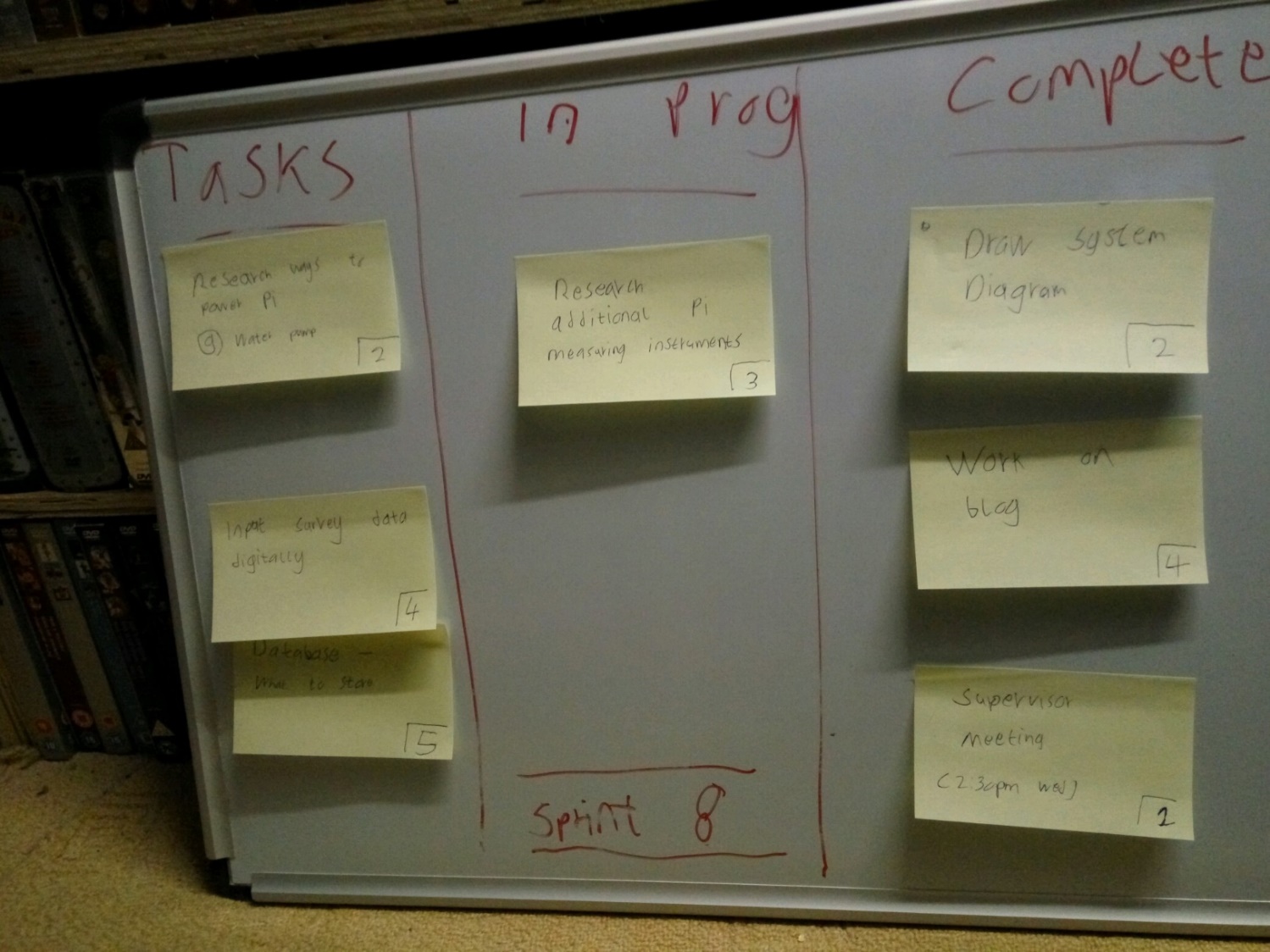
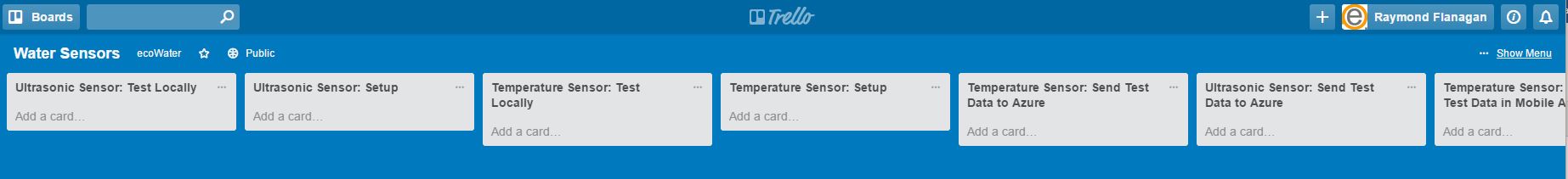
**Chapter 2 – Environmental Research  
  
2.1 Choosing a Project – FundAssist**I started thinking seriously about what I would base my fourth-year project towards the end of last summer, as I was winding down my work experience as a developer at FundAssist in Dundrum. FundAssist are a financial services company that serve the investment fund industry.  
  
In fact, one of the two fleshed out ideas I considered stemmed from my work at FundAssist – I’d mentioned to my manager, Philip McKee, that I was doing a fourth year project module and would be interested in any project idea that the company had that might fit the brief of the project module, with the intent being to work with the company throughout the development process.  
  
In my final weeks at work in early September, Philip and I had a meeting where he outlined a few different ideas he had that fit with what I was looking for and the amount of time available to complete it. The idea that stuck was a one-way company-wide notification system – a program that would be pre-installed on all employees’ work computers and would notify them when a ticket was relevant to that person or if they were referenced in it, as well as notify them of other business requirements such as meetings or filling in timesheets. The company has a custom-built in-house ticketing system to manage issues that arise, be it anything from technical support to client requests.  
  
When a person’s department is associated with a ticket everyone in that department is CC’ed on the email the ticketing system sends out whenever the ticket is updated in any way, resulting in a lot of noise and delays in people responding to issues that need their sign-off or input.  
  
Philip suggested I find an independent project idea before committing to one of the ones he had, and to get back to him later when I’d chosen. I had planned to look into some of the ideas I had before committing to anything, but I was glad to have Philip’s understanding in that regard.  
  
Over the next few weeks or so I did a lot of research into both my own ideas and the FundAssist-sponsored idea that I thought had potential, the notification system. As the data FundAssist handles for clients is confidential, the company host their own servers in Baldonnel rather than utilise cloud or third party servers for security reasons. This was relevant to the notification system because it meant the use of cloud-based services would be precluded, and that was something I was keen to immerse myself in with my project.  
  
I did competitor research - how intra-company notifications systems worked in other companies as well – Everbridge was a popular option, and utilises an interesting system that tracked users locations on work phones or laptops and sends alerts that are relevant for the area the user is it. With two separate office buildings, something like that would have value for FundAssist.  
  
I had a follow-up meeting with Philip to get the go-ahead for surveying a few of the employees at FundAssist – I selected people of different competencies with regards computers as well as people in each department of FundAssist so I could get a decent overview of what the notifications system would need to actually work as intended and be useful as a productivity tool.  
  
The take-home points for me were that the ticket email notifications that were being sent were overwhelming and impossible for employees to keep track of. Power users in the Software Development office had used Outlook formulas to neatly organise their ticket notifications, separating ones sent to the department’s email address and ones directly addressed to them, but other more novice users had no clue how to do this. Secondly, the interviews brought to light the issue of who would have access to the ‘admin’ of such a system – would it be just department heads? In the Software Development department I knew, for example, that the systems administrator routinely emailed about server downtime or other infrastructure issues and the senior secretary emailed about general office issues – these tasks were a natural fit for the prospective system. I felt establishing a concrete list of who was going to be sending notifications would allow me to design a system that catered to their needs directly.  
  
I took all this information and came to a few conclusions by the time of my next Skype meeting with Philip, as by then the college year had begun and I was back in Sligo and October was fast approaching. I’d also been working on the idea that would become my fourth-year project through this period, and it was something that had captured my imagination.  
  
I told Philip of my decision – to work on my own idea – and he was incredibly gracious and offered any help and advice he could give me when I outlined what it was. With regards to the notification system for FundAssist, I recommended that in lieu of designing a whole new piece of software, the majority of the problems it intended to solve could be addressed by using the Outlook inbox formulas the development team had been using. This fact also played a role in my decision as I felt it might prove to be a case of change for change’s sake if the formulas approach was not at least tried before committing significant resources to development of a new software solution.  
  
Though nothing tangible came of my work for FundAssist with regards my fourth-year project it did provide a great opportunity to engage with a business and develop requirements and design solutions to a real problem. I had to accumulate and present data to my superiors, something that speaks to the project module’s learning outcomes that related to written and verbal communication, and working in co-ordination with others. The month or so I spent working on the notifications system improved many of the skills I would need to undertake in the following months, not least of which was the focus on designing software that keeps the intended users in mind and general requirements gathering techniques.  
 **2.2 Background information and research**Growing up on a farm, I’d always had an interest in farming to go with my passion for computers. When I was trying to think of an idea for my project the first place I started to look for inspiration was at home on my family’s small farm. We have suckler cows and I’d become accustomed to the water holes that we have in each field overflowing regularly. It seemed like a waste of water, and if a tap was overflowing a container the only solution was to trek down the field and turn it off manually.  
  
I started thinking about what a system that controlled the water pump on the farm would look like. On a the most basic level, it would need to be able to turn the pump on and off. I then looked at the types of pumping systems used on farms – some were connected to boreholes, some used solar pumps. In a few of these cases, my idea of using a Pi to control the pump wasn’t viable because of where the pump was and how it operated.  
  
That meant I looked at home uses of the potential system – could I monitor water storage tank levels in the home, for example, or control a domestic pump? The answer to both of these questions were more affirmative, and because this would place the system in a house, a solid internet connection would be quite like to exist already, making the system easier to set up. **2.3 Similar Products**While there isn’t a product that would compete directly with ecoWater, there are analogous products in other fields that gave insight into how a system like this could work.  
  
  
*Third generation Nest thermostat*  
  
Nest, now owed by Alphabet (Google’s parent company), built their company on a smart thermostat that connects to a house’s heating system and allows for control of it via the thermostat’s touchscreen interface or via a smartphone app. A unique selling point of the Nest thermostat is its machine learning – it monitors the occupants of a house and tries to graph the hours in a day when no one is in the house so it can put the heating system into energy-saving mode and save the user money. This has proven quite popular – the company generated revenue of around $340 million (Bergen, 2017), with the smart thermostat their most popular product.  
  
  
*Nest smartphone app interface*Tebiti’s wireless smart irrigation system is a good example of the ways in which smart technology is replacing manual or timer-based tasks on farms – it collects soil moisture data via an on-site and then transmits that data to the cloud for interpretation and manipulation. The end-user web app that can be accessed through a browser allows commands to be sent to the wireless value, letting the farmer decide when and for how long they water their crops. *Diagram of Tebiti’s smart irrigation system* **2.4 Requirements Gathering**At first I’d focused on farming uses for the system, I decided to research the farming community. The average age of farmers in Ireland is 57 (European Commission, 2017) – this fact impacted how I approached designing the end-user app. I then set out to survey as many farmers as I could so I could get a sense for what their needs were, what their comfort level with technology was, pricing, and how likely they would be to consider the final product were it to come to market. I emailed the Irish Farmers’ Association for, partly to query them about existing smart technology on farms and partly to ask them to email their members about completing a survey, be it digitally or over email or telephone. They forwarded my query to their Connacht-based emails, but honestly responses were nearly non-existent. The IFA were helpful, but ultimately the survey had little pick up.  
  
I changed tact after that failure, deciding to solicit in-person interviews with farmers I knew personally. I figured some solid environmental information was still much better than none, even if the sample size was likely to be small – it ended up being twelve farmers. I created a survey sheet that I used to write down responses, as well as miscellaneous notes from the interviews that stood out to me. I found I was able to explore the idea of the system more meaningfully talking one-to-one with a potential user, so even though the sample size means the survey isn’t terribly scientific, it still gave some useful insights into who I was designing a system for.  
  
The take-aways from these interviews were that the age profile of the farmers was indeed quite old, and that most owned smartphones or PCs (meaning they could get value from a system they could interface with via a web app), but they didn’t download many apps for their phones. Full-time farmers tended to be older than part-time, but this is partly explained by some being retired from the jobs. The tabulated data is below, and some examples of raw survey sheets are in the appendices.

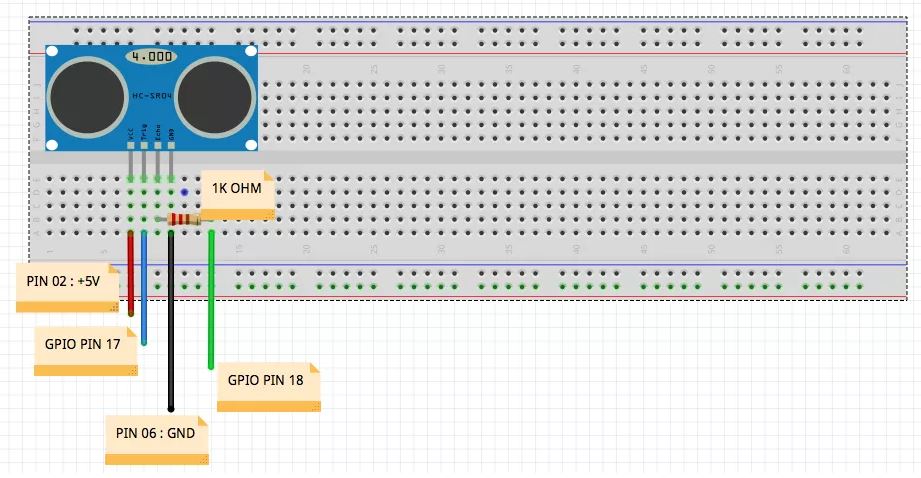
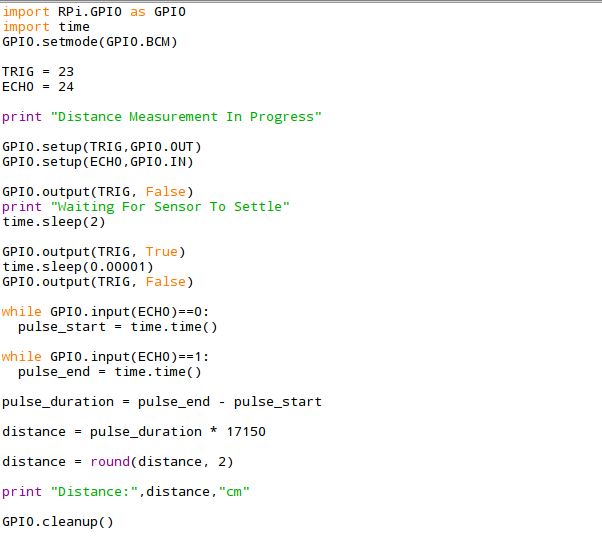
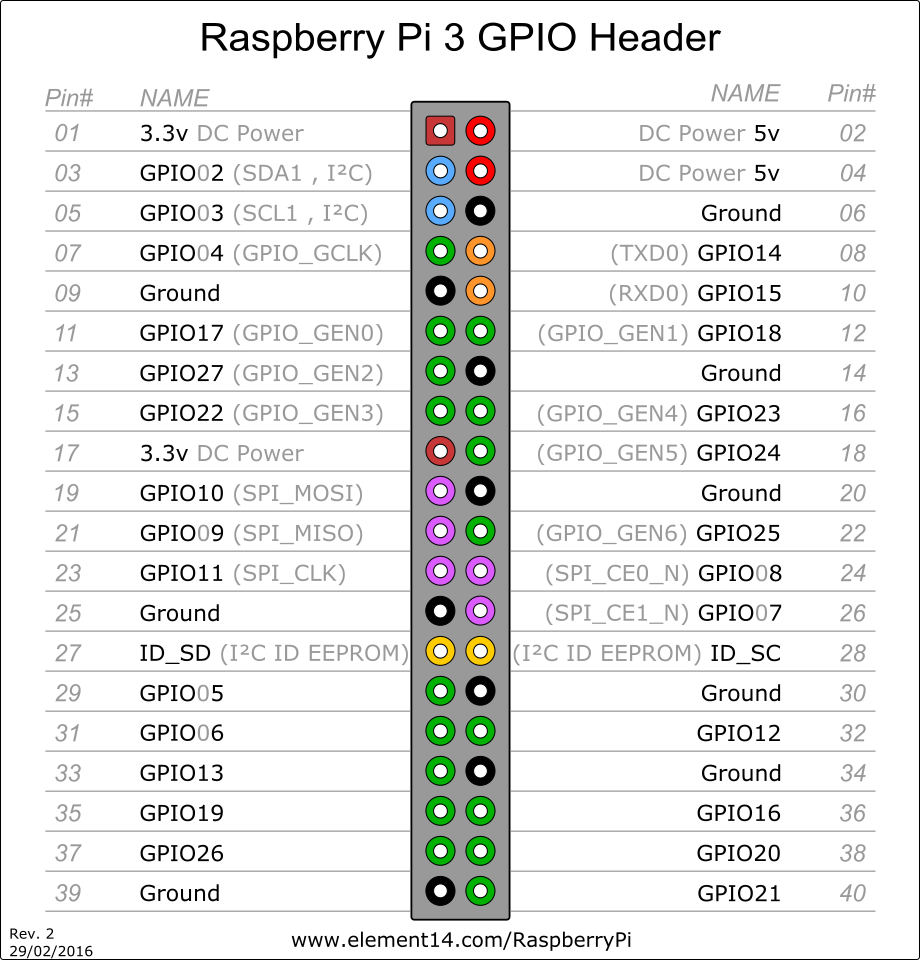
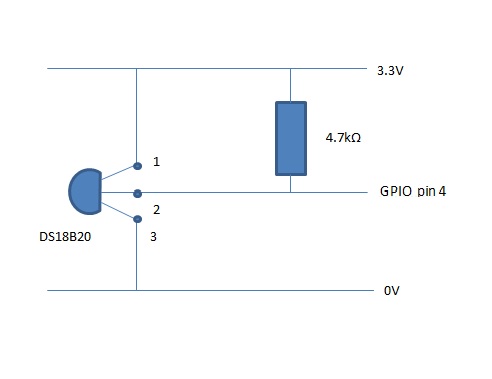
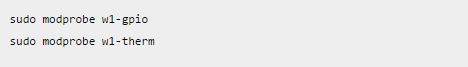
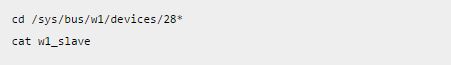
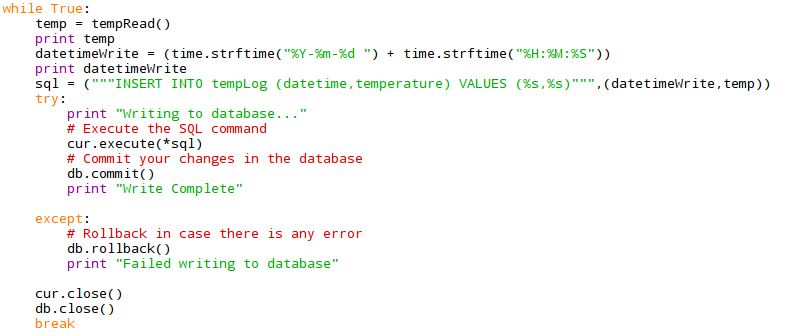
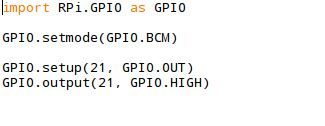
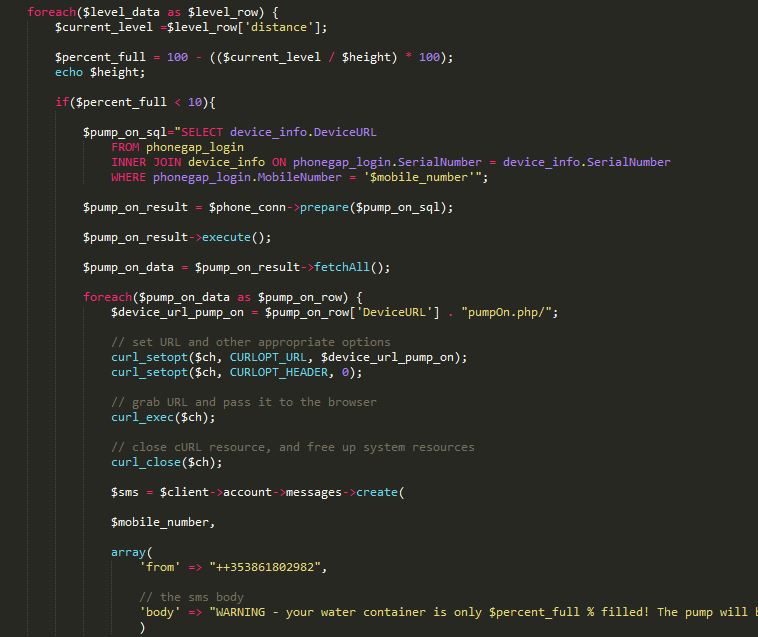
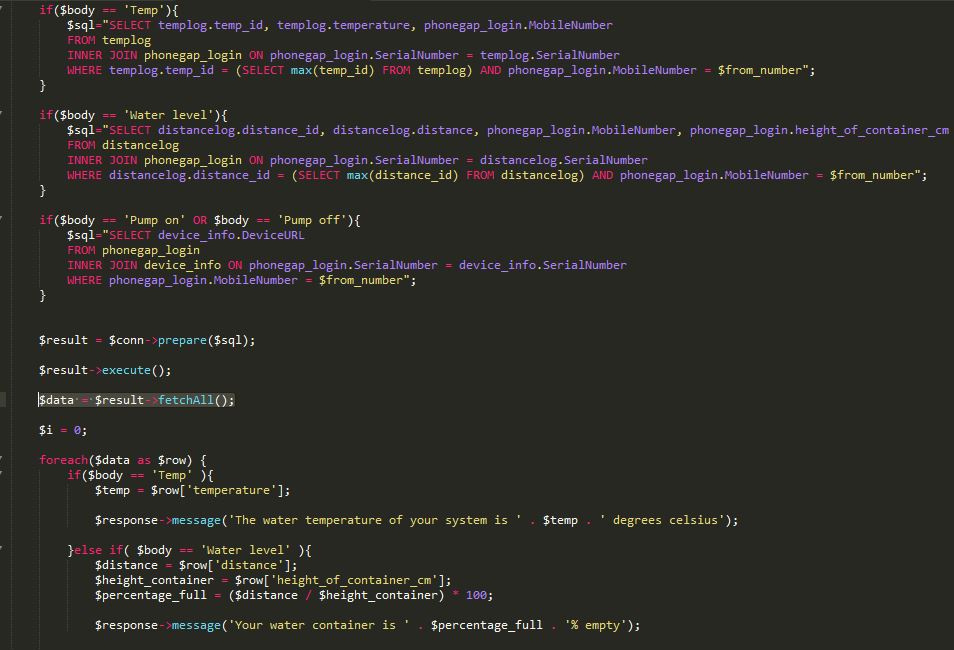
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Person** | **Age** | **Role** | **Own Smartphone?** | **How often download Apps (per month)** | **How likely**  **Are you to use system? (1-10)** | **Distance to**  **pump from dwelling**  **(meters)** |
|  |  |  |  |  |  |  |
| **A** | 62 | Full-time farmer | Yes | 2 | 8 | 100 |
| **B** | 58 | Part-time farmer | No | N/A | N/A | 80 |
| **C** | 47 | Part-time farmer | Yes | 0 | 9 | 50 |
| **D** | 69 | Part-time farmer | Yes | 2 | 8 | 15 |
| **E** | 53 | Part-time farmer | Yes | 4 | 7 | 17 |
| **F** | 61 | Full-time farmer | Yes | 5 | 6 | 45 |
| **G** | 54 | Full-time  farmer | Yes | 1 | 8 | 150 |
| **H** | 51 | Part-time farmer | Yes | 0 | 9 | 75 |
| **I** | 67 | Full-time farmer | No | N/A | N/A | 100 |
| **J** | 65 | Full-time farmer | Yes | 0 | 10 | 30 |
| **K** | 72 | Part-time farmer | Yes | 2 | 8 | 50 |
| **L** | 52 | Part-time farmer | Yes | 3 | 8 | 40 |
| **M** | 59 | Part-time farmer | Yes | 1 | 8 | 60 |
|  |  |  |  |  |  |  |
| **Averages** | 59 |  | 84.62% Yes | 1.8 apps a month | 8 | 62 meters |

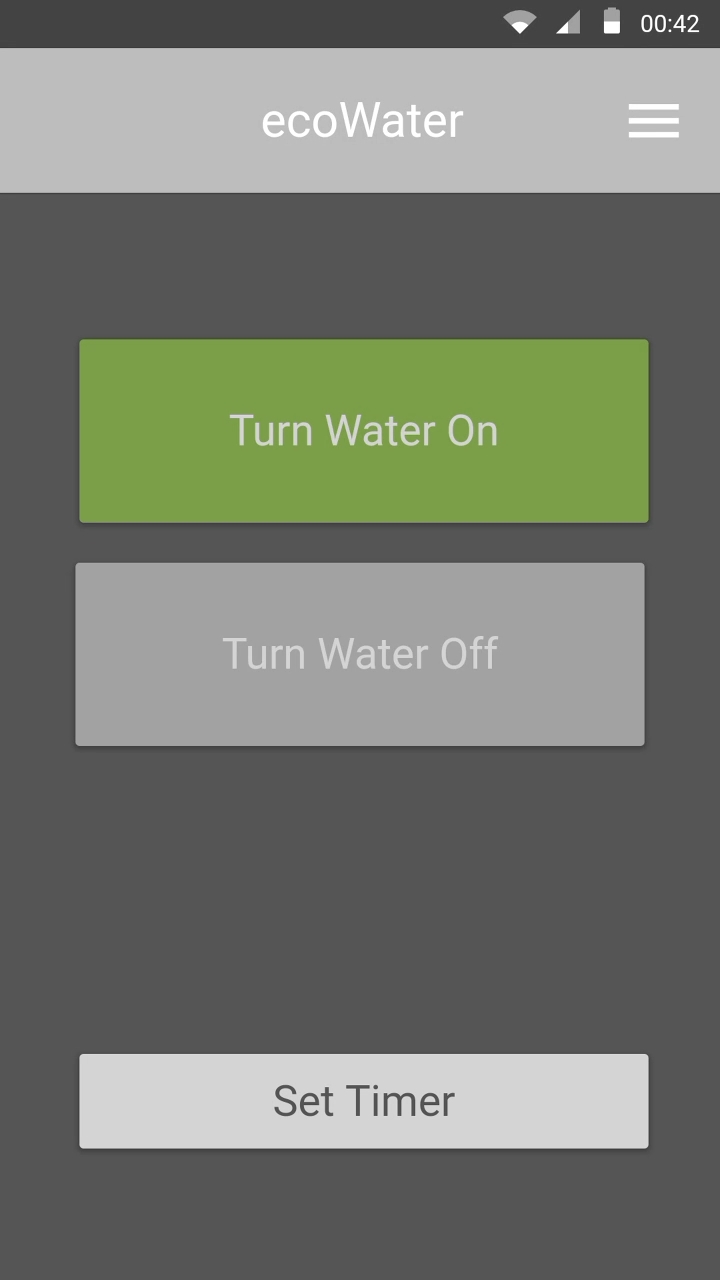
**CHAPTER 3 Technical Research and Technology Selection**This chapter has been split into three parts of easy of reading – one for research related to the physical Pi system, one related to the back-end technologies and one for the research done towards selecting technologies to surface the data.  
  
  
*Software architecture diagram for ecoWater* **3.1 Technical Research – Pump Control System  
  
3.1.1 Micro-computer**I knew when I chose this project that there were two very well supported options for an IoT device, namely Raspberry Pis and Arduinos. Both share a lot of similarities – open source, large communities developing for each and a focus on being a good introductory platform for people wanting to develop prototypes or home automation systems.  
  
The Arduino was cheaper than (most models of) the Raspberry Pi, which was appealing when taking into account the stated aim of choosing technology partly on the grounds of keeping the cost of a built system down, but it also had a major disadvantage over the Raspberry Pi – it is a micro-controller rather than a fully-fledged system-on-a-chip like a Pi. The Pi was simply a much more powerful machine when compared to an Arduino.  
  
What this also meant when compared to Pi was a lack of onboard features – the Raspberry Pi 3 has four USB ports, the Arduino none, the Pi 3 has Bluetooth, Wi-Fi and an ethernet port built in, the Arduino has none of those comforts. The price of a raw Pi 3 board and an Arduino UNO was only €38 compared to €22 euro on Amazon.co.uk, and for that extra 16 euro I felt I was getting a lot of added networking functionality that would come in useful when developing the project.  
  
I also purchased a ribbon cable, breadboard and T cobbler, which allowed me much more flexibility with where I connected the sensors and modules to the Pi – the ribbon cable connects the GPIOs pins to the T cobbler, which in turn connects to the breadboard. This allows you more space to connect jumper cables from the GPIO, and set up resistors to regulate current where necessary. Without it, testing the components would have involved a lot of plugging in and plugging out, as well as a serious amount of soldering resistors to jumper cables.  
  
Finally, I had some experience with a Raspberry Pi from my third year project, which only increased my comfort with selecting it as the core hardware of my project. **3.1.2 Pump Control Module**A domestic water pump is invariably connected to mains electricity and can utilise anything from 120v to 230v of power depending on a given pump, how it was originally wired and what part of the world . Immediately that brings up an issue with using the Pi to control a pump – the Pi is usually driven by a micro-USB cabled power supply rated at 5v. I had to check if connecting such a powerful power source to a Pi would even be feasible.  
  
The best way to control a ‘dumb’ electrical device such as a pump with a Pi is to connect a relay module to the Pi’s GPIO pins. In the simplest of terms, a relay switch has three inputs for cables, NC (Normally Closed), COM (Common) and NO (Normally Open). The relay either connects two positive wires (turning the device on) or breaks the circuit (turns the device off).

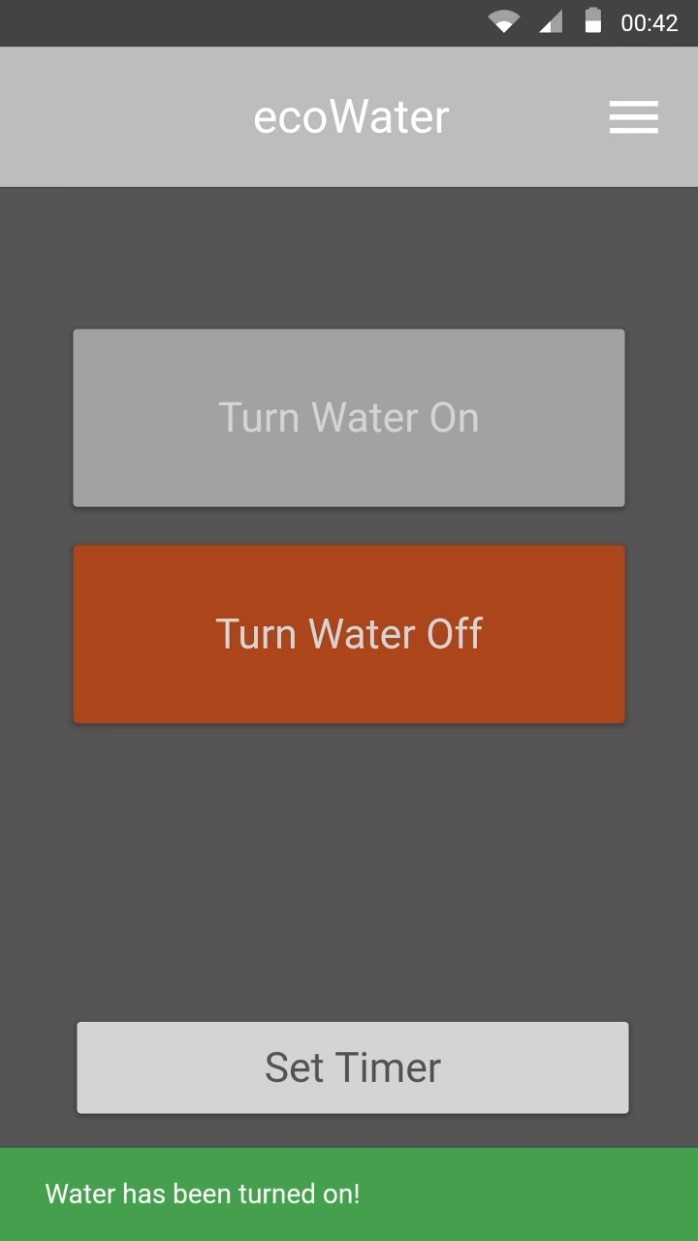
To make this work, one must connect the positive wire from the power supply to the COM input and the positive wire from the pump to the NC or NO channels depending on if you want the default state of the switch to allow the power to the pump, or to break the circuit and wait for a signal to complete the circuit and power the pump.  
  
I had bought a starter kit with my Raspberry Pi 3 that contained a relay module that was powered by the Pi – a max of 5v. This could be used to control low-powered devices (say, a set of LED lights), but it didn’t have the capacity to support something more powerful. With that in mind, I bought a 12v 16 channel relay board that was powered separately to the Pi – this would have the horsepower to do what I wanted.  
  
This would work for a domestic pump, but a newcomer to electrical engineering testing on mains power with a domestic pump was both impractical and incredibly dangerous (the voltages involved can kill, so I was rightly cautious). For testing purposes, I settled on a 12v peristaltic pump, powered by its own 12v power supply. The pump was selected as it would mimic the set-up of a domestic pump, but at a much safer voltage. I had to do some soldering of electrical wires to this pump as well as hook up wiring to connect the pump to an adaptor for the 12v power supply.  
  
 **3.1.3 Temperature Sensor**The criteria for a sensor to measure the temperature of water (or any liquid) was that it had to be waterproof, otherwise it would just short the entire system. The most popular temperature sensor for the Pi and Arduino is the DS18b20:  
  
The DS18b20 isn’t water-proof, but a version of the same sensor has been water-proofed and released, and it suited my needs quite well. It comes with an extended cable, which came in useful while testing it. **3.1.4 Container Level Sensor**A requirement was a sensor that could tell the level of water in a container, be it a container to give livestock water on a farm, the level of grain in a silo, or water boxes in the home (common in countries with warm weather). Despite the multiple uses for the sensor, the process would be the same – get a measurement of the container while empty, take another when the level has changed and then do some quick math to calculate the percentage filled or empty.  
  
  
*Generic water level sensor*  
  
  
I researched a sensor that could measure the level of a liquid when it touched it – see above . This meant liquid would have to be touching the sensor for it to measure the level, and given its small size it would make for quite limited sensory data as it wouldn’t know the level unless the container was very smaller or nearly full.  
  
  
*HR-SR04 ultra-sonic sensor*  
  
An alternative I found was much more robust – the HC-SR04 ultra-sonic sensor. This sensor uses an ultra-sonic sound to measure levels – it sends out a signal and waits for the ‘echo’ to bounce off whatever it hits. Using these values and the speed of sound (340m/s!) you can calculate distances from the sensor. Comparing the size of an empty container verses how much of it is currently filled you are able to estimate the percentage of the container that is filled or empty with a high degree of accuracy. **3.1.5 Other Sensors Considered**I spent some time researching the idea of displaying water quality information in the web app. Despite extensive searches, there wasn’t much in the way of digital, pi-friendly sensors that I could use. I did find examples of water quality devices that tested waters for its composition, toxicity and so on, and then displayed this information via a seven-segment display.   
  
*Layout of a seven-segment display*  
  
When I told this to my supervisor, he suggested there may be a way to determine the values being displayed by the pins that were active at a given time – a seven-segment display has a pin for each of its seven segments, and if current was going to that pin you could surmise that that segment was active. A regular seven-segment display has 128 unique states it can be in, and even though you would only be looking for displays that were showing a number (0-9) or one of the hex letters it can also display (A-F), accounting for situations where the ‘G’ segment was lit to display a hyphen would also be necessary. Given the displays I had seen had multiple seven-segment displays to show long numbers, the number of wires and connections needed to actually capture meaningful data quickly multiplied to a stage where I felt it was both beyond my capabilities to solder or connect to so many pins as well as the sort of time-intensive task that I scarcely had time to complete. As such, it was a tempting idea but one which I left at that.

**3.2 Back-end**This section covers technologies and research relating to utilising the Pi system to collect/send data from the sensors to the cloud. The Pi operating system that was chosen was Raspian, the most popular and most well documented Pi-supporting Linux kernel. **3.2.1 SSH/Python**Starting out working with the PI, the most effective way to get the Pi’s sensors and modules to do something was to set up SSH on the Pi so I could run it in headless mode from my development windows PC – a relatively straightforward process of setting the Pi’s config file to enable SSH, and then making note of the Pi’s IP address via the terminal command ‘ifconfig’.  
  
Once this was done I downloaded Putty on my Windows machine, which allowed me to specify the Pi’s IP address and then SSH into it so I could feed it command line prompts, be it writing or executing scripts.  
  
Python was the easiest language to use when learning to control the Pi’s GPIO pins – Python 3 is pre-installed on newer Raspian versions, as it is a readable scripting language for beginners. There were many other languages that can interface with the GPIO pins – JavaScript, in part – but Python was the obvious choice for me because it allowed for rapid prototyping and testing of scripts to run the sensors and relay boards. **3.2.2 Server**Node was researched also to service as the basis for the server that would run on the Pi, serving data from the sensors to an API on demand. Node is a JavaScript-based technology that allows the use of JavaScript on the back-end. Issues with poorly supported Node packages for my sensors led me towards PHP and MySQL.  
  
Using PHP meant I could edit and call my previously created python scripts relatively easily, while MySQL is a lightweight database that can sit on the Pi server and collect data - specifically temperature data in this case. Being able to use SQL commands increased my comfort level as I was already well accustomed to using SQL in college and at work. Storing this data locally rather than in the cloud prevents excess bandwidth usage, particularly useful when a user has a poor or metered internet connection. Likewise, it saves on the cost to the developer because there is no need to store the data in the cloud. **3.2.2 Microsoft Azure**I created a MySQL database on Azure to store the user account data securely and remotely, with PHP hosted on Azure being used to communicate between the database and the web app.

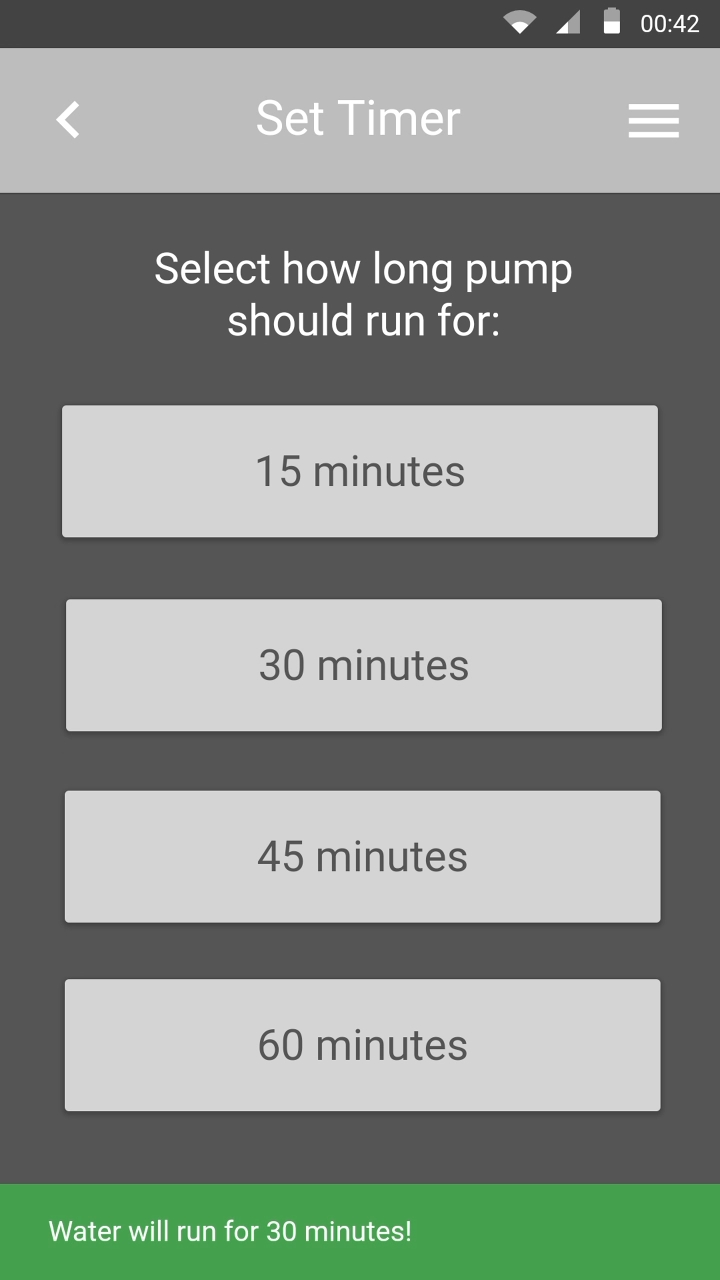
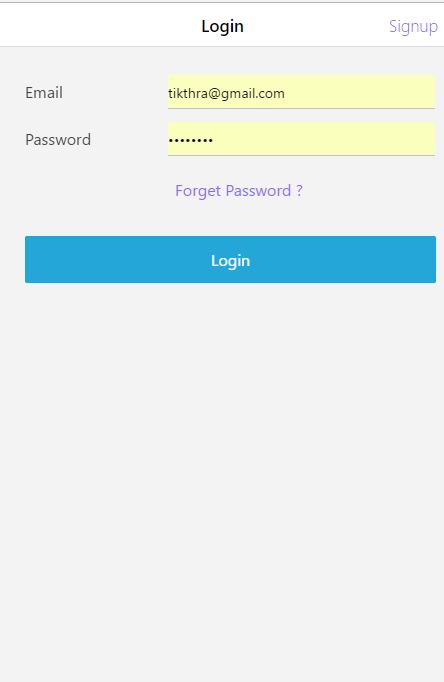
**3.3 Front-end  
  
3.3.1 HTML + JavaScript**HTML and JavaScript were selected for the web app, as it allowed me to build something that was able to be viewed in a browser, and if there was time then ported to a phone via PhoneGap. My previous experience with HTML and JavaScript played a role in the choice as well, as I had already undertaken learning Python, some electrical engineering and PHP to complete the project and I wanted to use languages I had a comfort level with. **3.3.2 PhoneGap**PhoneGap is based on the Cordova framework and allows you to develop HTML + JavaScript web apps and then use it to compile it as a native Android or iOS app, providing you have the relevant SDKs for either platform.  
  
As iOS compilation required a Mac, and I was not familiar with iOS development, I focused on producing a native .apk that could run on Android**.  
  
  
  
  
  
  
CHAPTER 4 Development  
  
4.1. Requirements Specification - Target Users**By the time it came to development, I’d adjusted my target users to home owners as well as farmers. As a system that controls a pump would involve installation work **4.2 Development**Much like the technological research, there were three distinct parts to the development of the system – work done on the Pi itself to get sensors to operate, the back-end databases and servers and the front-end user-facing app. **4.2.1 Approach to Software Development**I took an agile approach to develop, making product backlog items and organising them into sprints, doing MOSCOW analysis of what I wanted the system to have, and what it needed to have. I made use of a physical sprint planning board:  
  
  
****  
*Whiteboard for sprint planning. Not wall hung.*  
  
I also used Trello for managing a few areas of the project as I was intrigued by often I was hearing about the service:  
  


*Trello board for water sensor research/development*  
  
Ultimately, Trello wasn’t for me and I enjoyed the more tactile feel of a physical board as so much of my work involved technology that it was enjoyable to have an excuse to take a break from a computer screen at times. **4.2.1 Pi Development**The very first thing I needed to do was put an image of Raspian on a SD card and install it on the Pi – the SD card would be acting as the storage device for the Pi. This was relatively straightforward, with good tutorials guiding me through the process on the Raspberry Pi Foundation’s website. I also used the opportunity to enable SSH on the Pi and make note of its IP address so I could use Putty on my Windows machine to send commands to the Pi without having to dedicate my monitor screen and a mouse and keyboard to the Pi.  
  
Next I connected up my ribbon cable, T cobbler and breadboard as described in the previous section – this would be the basis for all my testing and development against the sensors and modules.  
  
**4.2.1.1 Ultra-sonic sensor**  
The first sensor that arrived was the ultra-sonic level module, and I searched for a wiring guide that would show me, an absolute novice at circuitry, how to safely power and transfer data from the sensor and the Pi itself. A 1k resistor from the ground pin and the data pin (GPIO 18 below) was used to regulate the current.  
  
  
*Wiring diagram for the ultra-sonic level sensor, with breadboard in background*  
  
Then I got to work testing the sensor with Python – using the command ‘python SCRIPT\_NAME\_HERE.py’ in a terminal window execute the python I’d written.  
  
  
*The Python script that controls the execution of the level sensor*  
  
A few things about the snippet above – the first thing that needs to be done in all Python scripts related to GPIO use is import the RPi.GPIO library, or else you won’t have control over the pins.   
  
The ‘time’ library has some functions that are useful when you want to delay execution of a command for a certain amount of time, or simply to record the time. In this script, the GPIO pins for the TRIG and ECHO inputs on the ultra-sonic sensor are setup with the GPIO.setup command. In simple terms, the script fires an ultra-sonic sound for a fraction of a second with GPIO.output(TRIG,TRUE), sets the variable pulse\_start to the current time and then waits for the ECHO pin to return a 1 – that is to say the sound has bounced off the bottom of the container and hit the ECHO sensor on the module. Another variable, pulse\_duration, is used to record the time when the echo is received.  
  
Then it is a simple case of subtracting the two recorded pulse times and using math to calculate the distance in CM – I take no credit for idea behind the math, but it essentially uses the speed of sound (340m/s) to calculate the distance.  
  
A note on the GPIO.setmode(GPIO.BCM) command – it sets the numbering of the GPIO pins to the GPIO numbering, which is different to the physical numbering of the pins on the board - GPIO.BOARD would set the pin numbering to those. Most users utilise BCM mode and consult GPIO numbering guides such as this one for the Raspberry Pi 3:  
  
  
*GPIO pin layout for Raspberry Pi 3*  
  
**4.2.1.2 Temperature sensor**  
The basics of the setup of the temperature sensor is very much like the ultra-sonic sensor – here’s the wiring diagram for it:  
  
*Wiring diagram for the DS18b20 temperature sensor*  
  
Where they differ is in enabling them for use. Unlike the level sensor, it is necessary to load two kernel modules in a terminal window on the Pi to allow it to interpret the data the temperature sensor is sending it:  
  
  
  
Then you need to navigate to the address of the temperature sensor:  
  
  
  
The star in the address is because all DS18b20 sensors have serial numbers that start with 28 – this number can change so doing it this way ensure you always go to the right folder. Cat w1\_slave in that folder is how we get a reading from the sensor – it returns something along these lines:  
  
C:\Users\Ray\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ggdf.jpg  
  
We don’t need to worry about much besides the final five numbers in the second line – this is the raw temperature reading from the sensor. This is important know to understand what the Python script that gets the temperature sensor data does – it’s essentially loading the above modules and then navigating to the file that contains the temperature data, opening it, and reading the line that contains the data. Dividing the raw number by 1000 will give you the temperature in degrees Celsius.  
  
  
  
The Python script above executes to grab the reading gets the raw data, does the math to convert to Celsius, and then logs the value in a MySQL database on the Pi that’s already been configured. You can see the credential for the database in the screenshot above, as well as opening a connection to said database.  
  
Recording the value into the MySQL database in Python is quite similar to writing to a SQL database in C# - passing the value to a function/method, opening a connection to the database, and then ensuring to close it again when the value has been successfully committed to the database:  
  
  
  
The scripts that controls the relay (and by extension the pump, or any electrical device wired to it) is the simplest of all three functions– they simply import the GPIO library as before and then set up the pin the relay is connected to on the GPIO, in this instance pin 21:  
  
  
The last line turns the switch on with the command GPIO.HIGH, while another script that’s nearly identical turns it off again with the command GPIO.LOW, making the relay switch break the circuit and thus turning off the pump or another electrical device – the code is agnostic as regards what is connected to the switch, all it does is send on/off values, meaning the code is easily reusable for other electrical devices.  
 **4.2.2 Pi Back-end**Installing a web server on the Pi began with installing Apache and PHP, achieved with this command on the Pi:  
  
C:\Users\Ray\AppData\Local\Microsoft\Windows\INetCache\Content.Word\mysql1.jpg  
  
A similar command installs the MySQL client on the Pi, and the interface between Python and MYSQL that will allow the temperature script to contact the database:  
  
C:\Users\Ray\AppData\Local\Microsoft\Windows\INetCache\Content.Word\mysl2.jpg  
  
C:\Users\Ray\AppData\Local\Microsoft\Windows\INetCache\Content.Word\mysql3.jpg  
  
Then I used regular SQL syntax used to create a database and table in that database:  
  
C:\Users\Ray\AppData\Local\Microsoft\Windows\INetCache\Content.Word\mysql5.jpg  
  
  
  
  
As I’d already written the scripts for controlling the relay and the sensors, the only remaining thing was to get the web server to serve the data publicly so the client application could access it, and send commands in the case of the on/off functionality for the pump or other electrical device.So now I have a script that gets the temperature and saves it, but nothing to run it – this is where Crontab comes in, a unix tool that will schedule jobs to be done. I decided to run the temperature check every five minutes to not overly tax the Pi, and on the assumption temperature changes would take time to occur in the first place.  
  
To set a job, all that needs to be done is call Crontab from a terminal window, give it a value for when to repeat the job and the address of what you want it to run:  
  
\*/5 \* \* \* \* /home/pi/Log/TempSensorToSQL.py  
  
The script also needs this directive (called a shebang line) added to it to help the job do its task:  
  
#!/usr/bin/env python  
  
Now all that’s left is to create some PHP that will produce usable data from the MySQL table:  
  
  
  
Essentially, it queries the database that was created, and then iterates through the results to create an output. The ’or die’ command provides an error message if the database cannot be reached. When you spin up the web server and navigate the PHP page above, the output will be something like this – JSON data that’s easily transferable to a web app on the client end of the system:  
  
*[{“Date”:”2017-04-20 19:35:12″,”Temp”:”15.40″},{“Date”:”2017-04-20 19:40:15″,”Temp”:”15.40″}]*  
  
Finally, this is a photo of my prototype system, with the sensors and the test pump connected to it:  
  
 **4.2.3 Web App Back-end**The design of the web app is covered in the next section, but the web app has hooks in it to tie it to the Azure-based database MySQL database that stores user information and sensor data. The MySQL database was created using the GUI interface on the Azure website. PHP hosted on Azure is used to communicate between the database and the web app, with AJAX calls posting (new users being added) or getting data (temperature values, turning the pump on/off).  
  
The database contains four tables – a User table with user information, a Device table with the URL of the Pi system and its serial number, and log tables for the sensor data of the ultrasonic senor and temperature sensor respectively. The Pi records its serial number alongside each sensor data write so that it can be easily paired to the correct device URL and user for notification and control on the front end.  
  
This is the PHP that handles the requests to login, sign up, change passwords and so on: **  
  
  
  
  
4.3.1 Twilio and SMS Support**Twilio is a company that provides a programable SMS service, accessible via a multitude of APIs in popular languages. It was my choice as it had great documentation and the support staff were very responsive to my questions when I was deciding between it and similar services such as Bandwidth.com or Nexmo. As I was already using PHP to interface between the Pi, mobile app and Azure, the PHP Twilio API was the obvious choice for the development of the ecoWater SMS notifications system. You can rent a mobile number from Twilio for a fee that can then be used to send and receive SMS messages.  
  
There are two distinct parts to the Twilio implementation – one is a PHP application (hosted on Azure) what is scheduled to run every 10 minutes on the Azure web service that is attached to the Azure-hosted MySQL database for ecoWAter. This application uses a SQL command to get user data and each user’s sensor data and then runs it through a series of checks – to see if the water temperature is too hot, too cold, if the water tank is nearly empty or in danger of over-filling. The application checks for this buy finding the latest sensor data for the user’s system’s serial number (stored in the database) and then if a given check is true, it acts and notifies the user via an SMS text to the user’s mobile number which is also stored in the database. A specific example is if the water level is below 10%, the PHP application calls the Pi and turns on the pump. A sister check to see if the water level is above 90% full prevents the pump from remaining on endlessly, or overflowing and wasting water.  
  
  
*Example of PHP application turning the pump when it sees the tank is less than 10% full*  
The second PHP application deals with situations where the user texts the Twilio mobile number to receive information about the system, be it the water temperature (by texting ‘Temp’), water level or to turn on or off the water pump. When the user texts the Twilio number, Twilio POSTs the text and its originating mobile number to a PHP application you can specify in the Twilio console. By using this mobile number, the application queries the users table in the Azure database to find the associated user data, serial number and sensor data for that number. Then the application processes the given text-based request and sends an SMS reply to the user’s phone.*How the PHP application deals with user SMS messages and then generates a response* **CHAPTER 5 – Web App UX  
  
5.1 Users**Home owners and farmers were identified as the target users. **5.2 Devices**The web app could be viewed through a PC browser – at its core it is just HTML and JavaScript - but it is intended to be used on Android mobile phones. My test device was a Xiaomi Redmi Note 2, which is a 5.5 inch 1080p screen that runs a variant of Android called MIUI. No other devices were tested for the purposes of this project. **5.3 Heuristics and Interface**My original design for the app can be best reflected in the proto.io high fidelity mock-ups I made. When considering my target market of farmers (average age in Ireland of 57) and their relative lack of comfort with complex actions on smartphone I settled on this as the main screen:

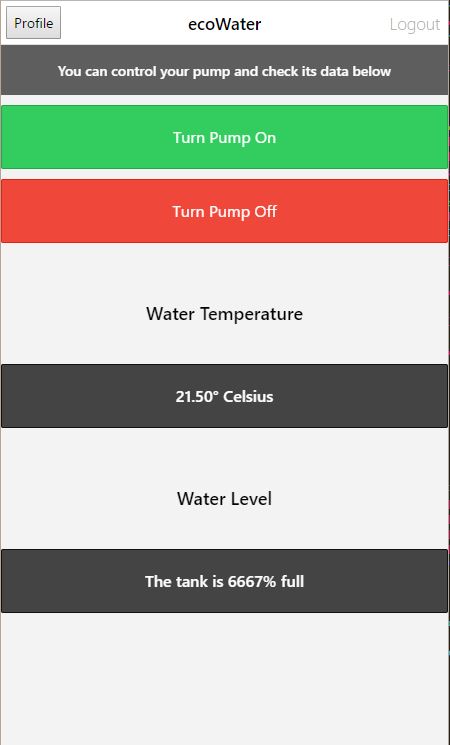


Two big buttons for on and off, colour-coded. The action that can be performed at that moment (on or off) is the one in colour, with the other button greyed out to take user attention away from it. I original had it as a toggle switch but this was small and hard to hit for the users I tested this on.  
  


Finally, I envisaged a page where the pump can be set for a specific amount of time, after which it then turns off. with similar info prompts to the main screen:

  
  
  
When I moved on to making the HTML/JS, things necessarily changed, and my focus had shifted away from simply farmers and towards home users so the design had to reflect this also.  
  
The final design has a login page with associated functions:  
  


And now on the main screen is the on/off and the sensor data:

  
 **5.5 Testing Design**The Proto.io mock-ups were very useful when it came to testing the design with users. My first tests were with some of the same farmers I interviewed in my requirements gathering phase to test out the Proto.io mock-up. When I pivoted the focus of the app and open open the target audience to home owners as well I got some home owners to test it – I selected testers at different stages of their life, one in their late 20s, one in their 40s and one that was retired and in their  
  
This allowed me to rapidly user test the design, rather than waiting for the HTML/JavaScript version to be completed, which would be much more technical and time-consuming – I believe getting this early input into the design helped me avoid wasting time when it came to developing the final app. I used the Proto.io Android app on the Redmi test device to mimic the finished user experience as much as possible  
  
For all user testing I had set tasks that I wanted the user to achieve – be it register an account, or turn on the pump and off again. All tasks would start at the login screen, the first screen a user would encounter unless they had already logged into the app. My ground rules were that I wouldn’t guide the tester in any way, unless they were suck and asked for my help. As the tester completed tasks, I noted the pain points in the design and where they had trouble. I timed the length it took to complete the tasks, to see if after a few tests any area of the design was proving universally difficult or time-consuming. When all task were completed, I would do a quick interview with the tester, quizzing them on the issues I’d noticed as well as asking more general questions as to what they would change or add to the app.  
  
Some of the issues with the original designs were the size of the size of buttons on the pump on/off screen made them harder to hit – and there seemed to be a correlation between age and this issue. With the average age of farmers being 49, and homeowners 2 the level of content noise in the app – as this target audience tended to be older than average and less comfortable with technology, simplicity above all else became the basis for the design. **CHAPTER 6 – Problems Encountered  
  
6.1 Electrical Circuits**I had absolutely no experience with circuitry before undertaking this project – I wasn’t even able to tell the difference between a live and a ground wire. When I went to assemble my sensors I quickly realised despite the Pi being sold as a learning platform, a decent knowledge of circuits and power regulation (resistors) was necessary. I also soon found out the usefulness of wiring diagrams after a few failed attempts to get the ultra-sonic sensor working.  
  
One of the more spectacular problems I encountered with the circuitry was when I first got the breadboard connected to the Pi and started out by trying the breadboard equivalent of a ‘Hello World’ program, connecting an LED pin and turning it on and off via Python. This worked fine at first, but the LED later melted, smoked and stopped working. Looking up what went wrong, I omitted the use of a resistor, which meant the LED was able to draw the maximum amount of voltage the power supply could output, way above what was needed to safely run a small LED. This is existentially defined in Ohm’s Law, i.e. voltage equals current times resistance. This was news to me!  
  
When this happened I’d the temperature sensor connected to the breadboard too, and the LED ‘blowing up’ ruined that sensor. Thankfully the Pi and breadboard were fine and luckily enough the temperature sensor came in a pack of five so I had a replacement on hand. I quickly learnt the value of resistors and to ensure the current was always regulated for whatever module I connected to the breadboard.  
  
I found the members of the official Raspberry Pi forum incredibly helpful and patient when it came to helping me troubleshoot what to them would have been very basic electrical engineering queries. **6.2 Raspberry Pi GPIO Pins**The layout for the Raspberry PI’s GPIO pins changed from a 26 pin layout on the original Raspberry Pi Models A and B, to a 40 pin layout on subsequent revisions of the hardware. This meant many of the schema and scripts that manipulated GPIO pins were not compatible with my Raspberry Pi 3 Model B board.  
  
  
The differences between newer and older versions of the Pi’s GPIO pins  
  
A lot of scripts and guides rarely stated which version of the Pi they were designed for – in a lot of cases I had to work backwards from the dates the articles were published and try to divine which generation of Raspberry Pi they were developed for, sometimes with success and sometimes simply dragging me down a fruitless rabbit hole. The biggest danger was attaching a jumper to a pin that was a I/O on an older model, but was a positive or ground pin on my newer model. If I wired it wrong, the whole system could easily have been destroyed. This slowed me down and made testing new components more deliberate than I would have liked, but I did manage to avoid any catastrophic errors. **6.3 Time Management**In the frankest terms possible, I don’t think I did the best job in planning my work on this project out. It was partly due to the lead time with component delivery – some parts came from China, others from the US and others still from the UK, and in many cases without all pieces testing and development was impossible, disrupting the rhythm of the project.  
  
With that said, it doesn’t go towards fully explaining my failures with planning. Looking back at previous assignments, I always had a very rigid structure I could rely on – do x, with x technology and in a compressed time-frame. Even in the third-year project I had three other team members pressing me and always helping keep the project front and centre. With the fourth-year project, it was me and me alone, and without much time constraints to hem me in and that lead me to meander between topic and technologies too much. I created test apps for different types of servers on the Pi – Apache, Node, Flash – didn’t select what one I wanted to use for weeks afterwards.  
  
The sheer length of the project’s duration allowed me to indulge my tendency towards procrastination much more than I would have liked. I know I could have done even more with the idea I chose had I been more decisive and willing to sit down for more than an hour or two at a time and really build the ground floor of the project, so to speak.  
  
If I did a project again I would be fast to get my technologies set, as I found the biggest time sink was not having those set in stone. Once I had settled on them, development was relatively rapid.  
  
I did try to organise my project work into month-long sprints, and this at least gave me goals I could target in each month, but again inconsistent component deliveries made some of those goals hard to even attempt to hit, and it was some-what discouraging to look at a sprint item that had been on my board for weeks or in some cases months without being resolved.  
  
I made the error of trying to apply a minimum viable product approach to the project at first – get a sensor connected, a server on the Pi running, a web app to display data and a cloud database to store user data set up. This seemed correct in practice but, in reality, trying to develop for one sensor when I hadn’t even tested if the language I was using would support another one I would be using just meant I had to scrap work half-way through. If I had a second chance I would have utilised a more waterfall-esque approach at the start of my project, collecting all the components I needed to hook up all sensors and modules to a Pi and then concurrently test them to ensure they played nice together. If that much worked, I could then look to implement a more agile approach as I would have known the other sensors I had to develop for were viable in the language I was using. **6.4 Microsoft Azure**I had some issues with Microsoft Azure when trying to set up databases on it – namely that I couldn’t at first. I’d signed up for a student Azure account in third year for that year’s project, but when I went to create a DocumentDB to test its viability as a store for the web app’s user data I realised I couldn’t create one, despite it being a Microsoft developed database.  
  
When I looked into this I realised others in the course had similar issues with paltry Azure options – you could make a MSSQL database with a student account, but not much else. This issue was also effecting another project I was working on for Colm Davey’s Mobile Development module and I contacted him about the issue. He got back to me with information about a new ‘Virtual Studio Essentials’ package Microsoft were offering students – this contained a code for 25 euros of Azure credit a month for 12 months. Unlike the limited free student account I had before, I could create any resource Azure offered (as long as I had enough credit) meaning I could now test the databases and services I had wanted to.

**6.5 Project Complexity**Despite knowing full well that one person could only ever do so much with a project that was being developed alongside other college modules, there was a real sense of times of feature creep with the project, I would develop one aspect of the project and it would lead to ideas for more features or improvements to existing functionality. Looking back, at times I would have been better served treating the project less like a passion project and more professionally.  
  
I added significant parts to the web app after it begun development, namely logins (which added more PHP to my slate of technology to learn) and using APIs such as Twilio to give secondary functionality such as SMS notifications. I don’t think I fully appreciated how complex a system I’d designed was, and having to support these other functions meant less time spent on the core functionality of the system.  
  
  
**6.6 Pivoting the Intended Target Market**I had originally intended to focus solely on farmers as a target market, but as I researched about how applicable the system would be to the different types of pumping systems I realised I needed to broaden my focus, while retaining the work that related to farm uses of the system. A pump system like a borehole pump wouldn’t be applicable to a system that relied on electrical wires to control the pump, for example. In home uses, however, where pumps of different sizes were used to pump water through main pipes, control aquariums, or even to combat flooding in flood-prone areas – a concern in my homeland near the Shannon.  
  
With that in mind, I lessened the focus on farming, and more on the multiple uses for such a system, be it in the home, on the farm or elsewhere. I wanted ensure it had enough use cases that it was a viable product, if fully developed.  
 **CHAPTER 7 – Critical Evaluation**Overall, I am glad I chose such a challenging topic for my project, but there were times when I wished I’d chosen a nice, comfortable Dot Net web app. Open source, be it the Raspberry Pi, Python, PHP, Apache or even plain JavaScript is an incredibly exciting realm to learn after four years of near total sub-mergence in Microsoft technologies from end-to-end.  
  
But by its nature open source breaks very easily – the Raspian kernel could be updated and entirely break your system’s functionality (as happened to me!), for example – and there are no ready-made solutions as many times you’re using a combination of languages, modules and chipsets few if anyone else has. For a novice, this was discouraging at times. There were moments when I wondered if I would produce a working prototype in any form. But through it all I learnt a little bit about the value of perseverance, and quite a bit about how a Raspberry Pi and IoT in general.  
  
In terms of new languages learnt, the two I was exposed to the most were Python and PHP – both languages with a lot of uses outside of IoT devices. I believe what I’ve learnt in terms of individual research and development and self-motivation will serve me well in the future. I’m proud of what I did produce, even if there’s always more that could have been done.

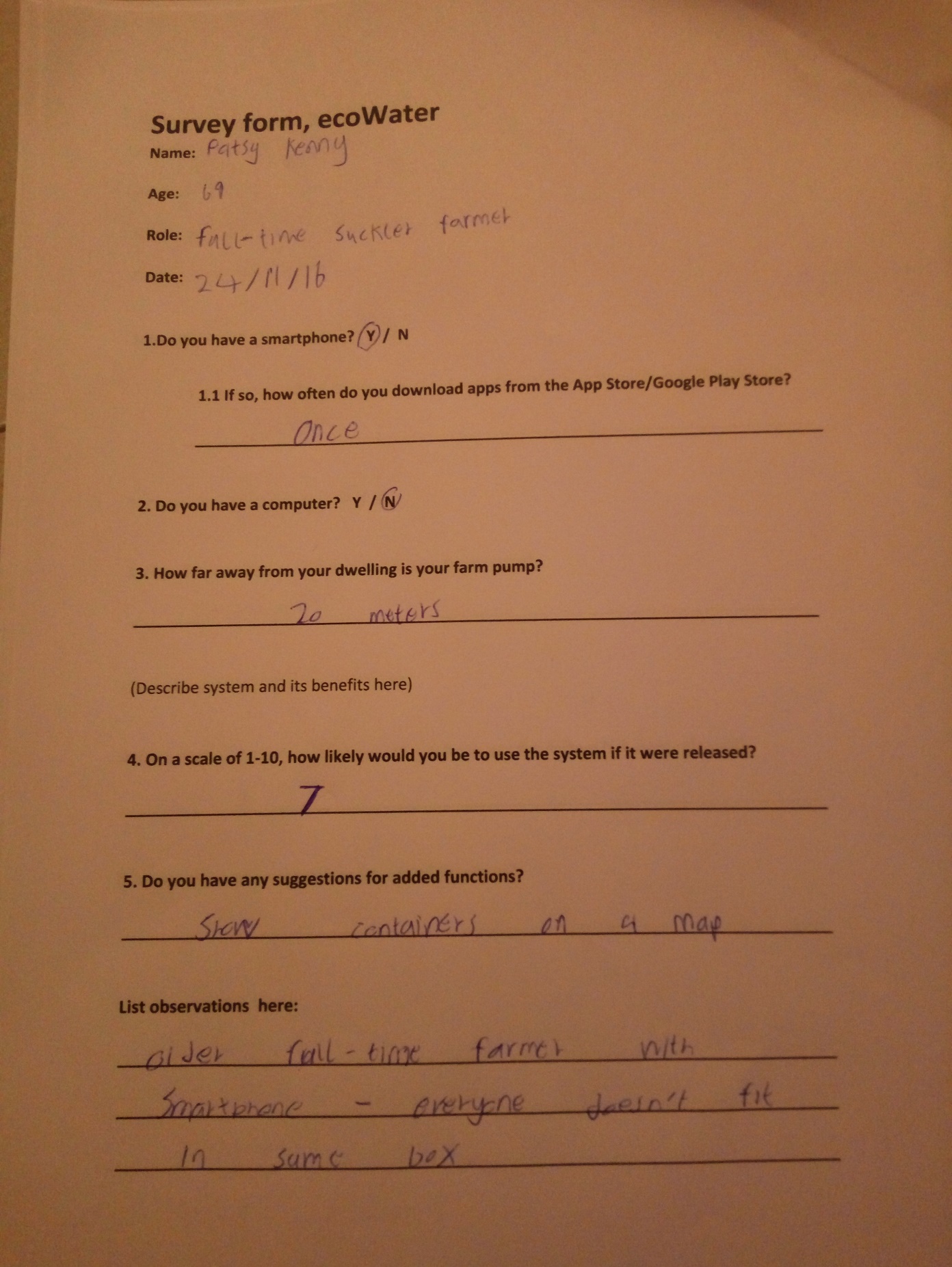
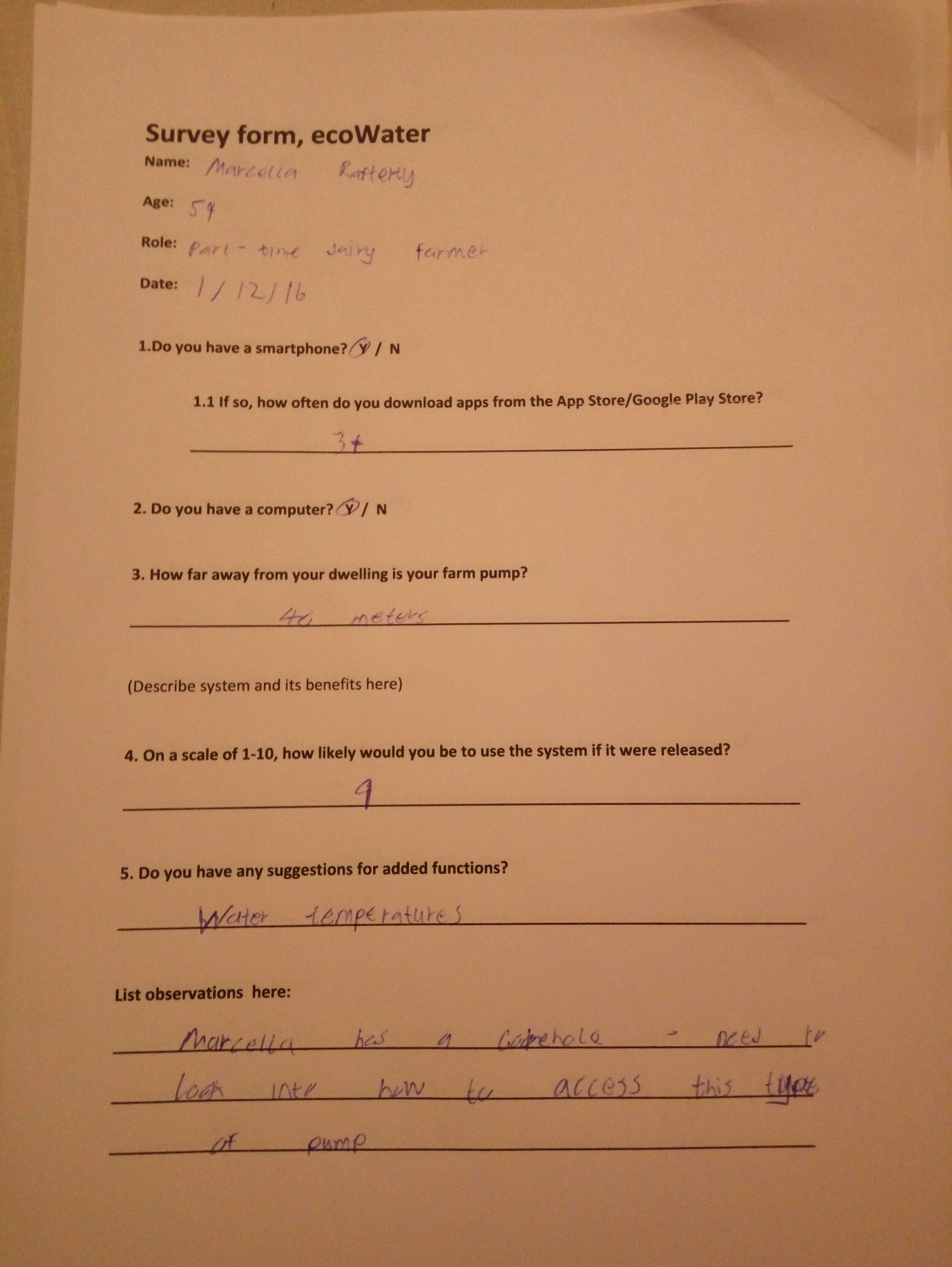
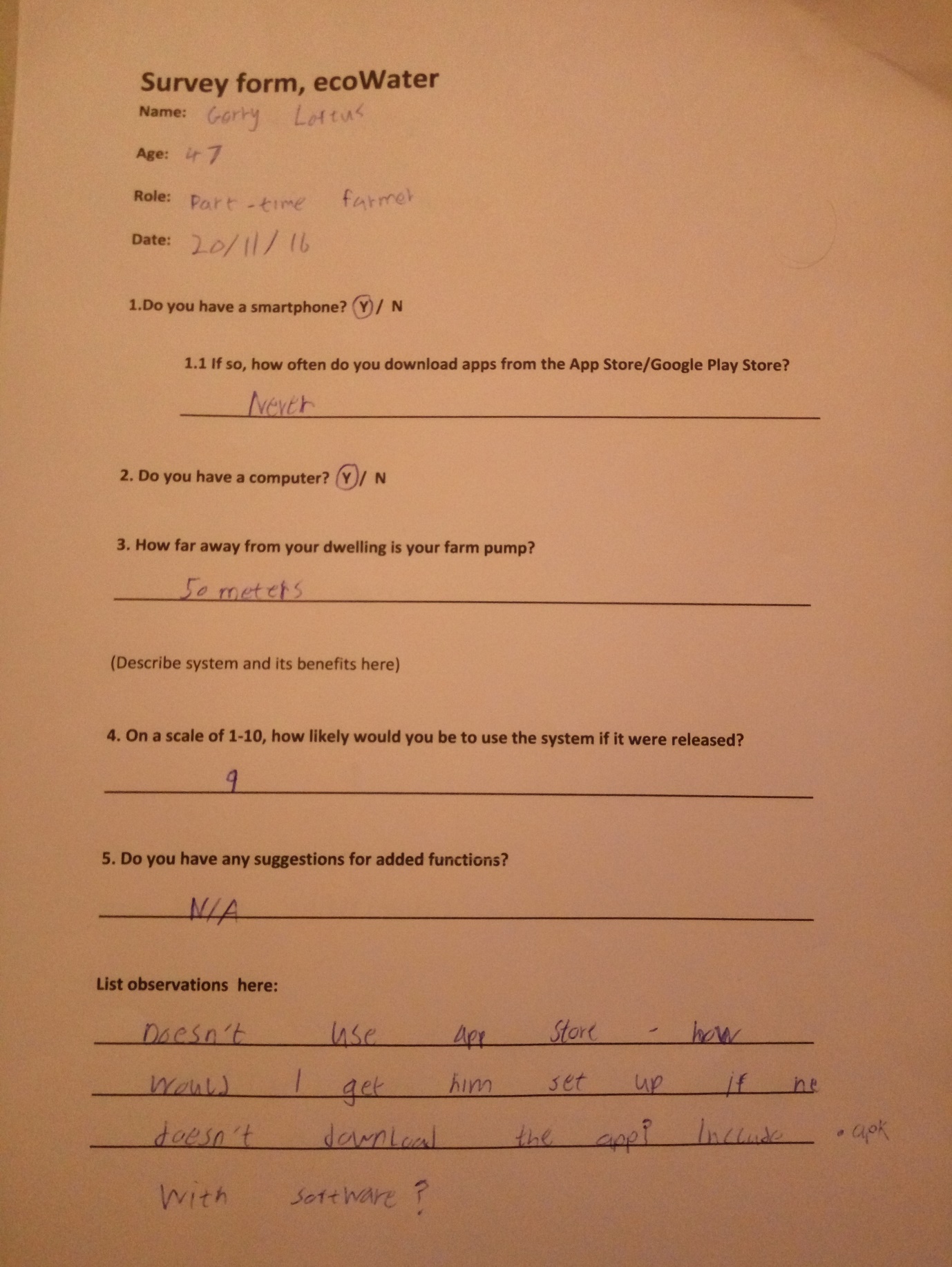
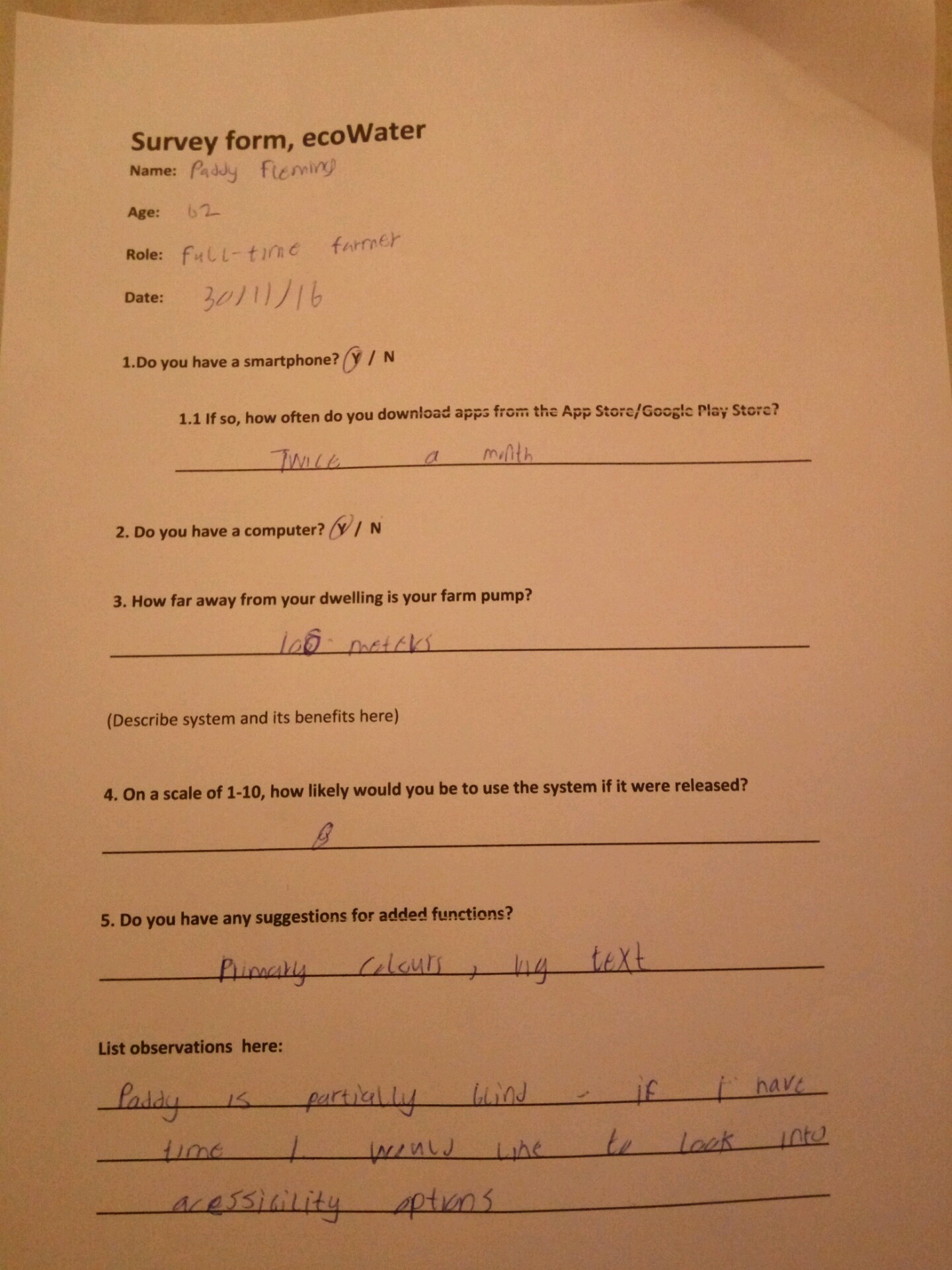
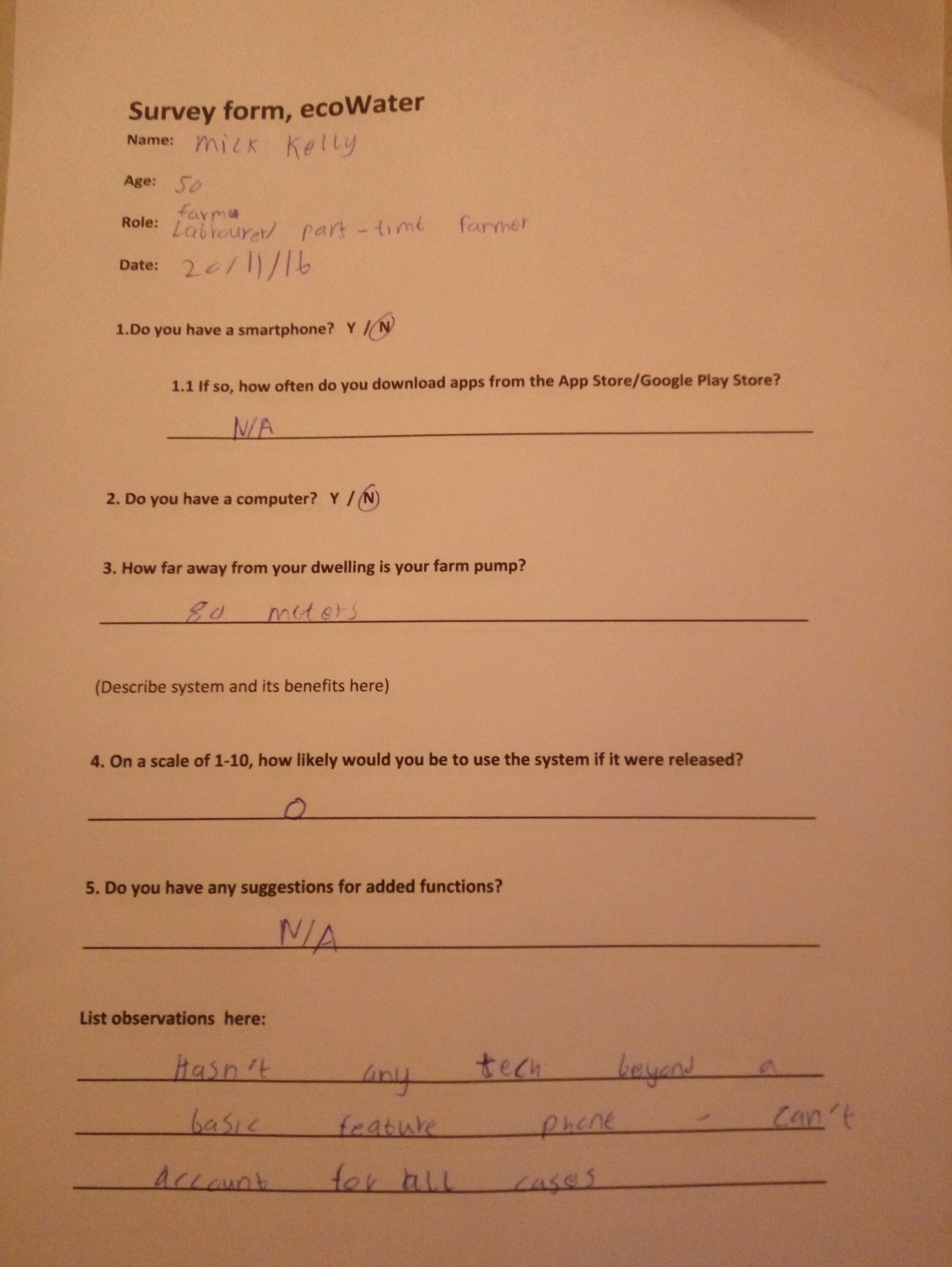
Returning to this project for the repeats, I met with Shaun, established what the criticisms of the project were the first time around and endeavoured to address them. To that end, I moved the storage of the water temperature and water level sensor data to an Azure-hosted MySQL database to address concerns about data being lost in case of a catastrophic event locally.

Secondly, the direct link between the mobile app and the system has been broken, with the system now having a serial number that is used to identify it and then entered when the user signs up on the mobile app, linking the user’s account with their specific system.   
  
Thirdly, full Twilio support was delivered, adding an interface that allows the system to be interacted with via SMS by the end user. This benefits a user who is travelling and may not have a steady internet connection to use the mobile app – with SMS, any cellular network at all can be leveraged to check up on the health of the user’s water system.  
  
Finally, the Twilio integration also queries the sensor data for each user and intelligently controls the system and sends SMS notifications. If the water level is too low, the system’s pump is turned on automatically and the user is notified of this via SMS, and the reverse happens if the water level is too high. If the water is close to freezing or too hot an SMS is also sent to notify the user so that they can correct the issue. This addresses concerns about the system not doing much with the data that was collected.  
  
Referring back to what the requirements of this system were at the beginning –   
  
- An IoT device connected to a pump and sensor that can record sensor data and control the pump  
  
- A cloud-based web service that can host a database that contains user and sensor data, as well as serve that data to other applications.  
  
- A mobile application that users can log into to control their pump and see sensor data  
  
- SMS notifications, allowing the system to notify users of issues that need their attention  
  
- A way to control and request information from the system via SMS so the system is accessible in almost all situations.  
  
I have addressed each requirement. While more work can always be done, I have both achieved the stated requirements for a successful system and addressed most if not all of the criticisms of the first design.   
  
With that said I will close the report with an overview of the cost involved in assembling the system, as well a checklist of what was achieved, and what wasn’t achieved. **7.1 Viability of a Built System**With the ultimate objective being to produce a system that was sellable to farm or home users, I tried at all times to select hardware and technology that was cost-effective. I calculated the cost of building a complete unit from scratch, using off the shelf components:  
  
**Raspberry Pi 3 board and wiring:** €38.53  
**Casing for Pi:** €5.50  
**Ultra-sonic sensor:** €2.12  
**Water-proof temperature sensor:** €2.34  
**12v Relay module:** €14.34  
**Power supply:** €7.95  
**Misc. cables/resistors:** €1.75  
  
**Total:** €72.53  
  
In addition, an Azure database would be run to collect and maintain user account data, but as this would not be storing constant temperature data (this is stored on the MySQL database on the Pi), this database would not be receiving heavy traffic and a standard S1 single database on the North Europe cluster with a maximum storage of 250GB would cost approximately €25.29 a month (Microsoft, 2017). A Twilio phone number costs $6 a month to rent, while individual texts cost €0.0075 to receive or send. All other technologies are open source.  
  
I believe when compared to similar products that connect to existent ‘dumb’ technology in the home such as Nest – the ecoWater system could easily be marketed at an attractive and competitive price-point. The Nest’s third generation thermostat costs €249, for example, so if the ecoWater system was developed further there is potential to undercut market leaders and establish a foothold in a niche market that larger competitors have no yet entered into. **7.1 Summary of what was achieved**- Learnt how to build electrical circuits and regulate power with resistors  
  
- Created working prototype of the Pi-based IoT system  
  
- Learnt Python scripting  
  
- Control of an electrical device – pump – via the relay module  
  
- Reading temperature data and storing it in a local MySQL database  
  
- Created an Apache web server on the Pi  
  
- Wrote PHP app to serve temperature data as easy-to-parse JSON  
  
- Iterative UX design, utilising high-fidelity mock-ups and user testing to get early feedback on design  
  
- Azure-hosted database and PHP web service for user accounts on the mobile app  
  
- SMS notifications when water level is low/too high and when water temperature is too cold/hot, leveraging Twilio’s API

- Constant monitoring of the system via PHP with the aid of an Azure web job that is scheduled to run every 10 minutes to check water level and temperature and then intelligently turning on or off the pump if necessary, preventing the tank from either going empty or over-filling and wasting water.

- Support for text message control of the system – if the user texts ‘Pump on/off’, ‘Water level’ or ‘Temp’ to the provided phone number they can get key information about their system, or turn it on or off remotely.  
  
 **7.2 Summary of what was not achieved**- Machine learning – analysing the times a pump or other device is on and adapting to the user’s needs by automatically scheduling on/off cycles

- Converting digital inputs from seven-segment display on water quality meter for display in-app  
  
- Water-proofed version of prototype system  
  
- Web app works in a PC browser, but optimizations for different screen size not implemented  
 **Additional Info**Project blog: <https://ecowater2017.wordpress.com/>   
GitHub Repo: <https://github.com/Syferus/4thYearProject/tree/master>

**Appendices***Survey form examples***  
  
  
  
  
  
  
  
  
References**  
  
*Bergen, M. (2017). With $340 million in revenue, Nest is underperforming, and its future at Google is at risk. [online] Recode. Available at: https://www.recode.net/2016/3/30/11587388/nest-2015-sales-budget [Accessed 17 Apr. 2017].  
  
European Commission. (2017). Agriculture - Ireland - European Commission. [online] Available at: http://ec.europa.eu/ireland/news/key-eu-policy-areas/agriculture\_en [Accessed 6 Apr. 2017].  
  
Tebiti.com. (2017). Smart-Precision Farming – TEBITI. [online] Available at: http://tebiti.com/our-smart-farm [Accessed 19 Apr. 2017].  
  
Azure.microsoft.com. (2017). Pricing - SQL Database | Microsoft Azure. [online] Available at: https://azure.microsoft.com/en-us/pricing/details/sql-database/ [Accessed 18 Apr. 2017].*