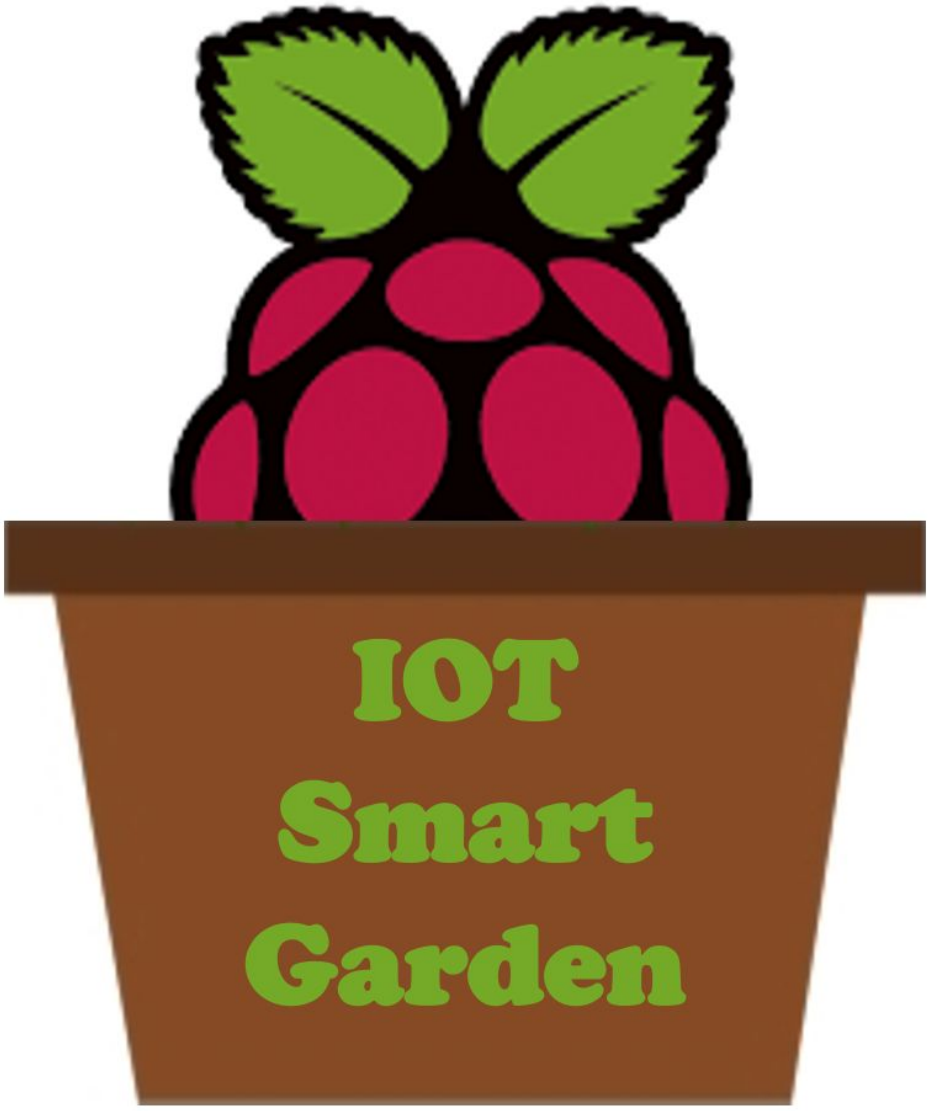
CA400 Final Year Project

IOT Smart Garden

Technical Guide



Daniel Pereira - 15364491
Jacob Byrne - 15492172

Supervisor: Dr. Donal Fitzpatrick

Table of Contents

1. Introduction

1.1 Overview

- ML Powered
    - Our application was designed to be an automatic system, as such, we implemented machine learning to learn what best suits the plant's environment and alter it as such.
- Manual Control from dashboard
    - While we desired to have the automatic functionality present, we also wanted to be able to maintain a degree of control. Users are able to manually control the plants and switch from ML controlled automatic mode to fully manual.
- Check various parameters
    - We wanted detailed information on the plant's environment. To meet this requirement our dashboard can show multiple parameters of the plant's environment graphically.

1.2 Glossary

- Python - interpreted programming language
- JS - javascript (client scripting language)
- EC2 - scalable compute capacity in the cloud (Elastic Compute Cloud)
- HTTP - HyperText Transfer Protocol
- Flask - lightweight web frame for python
- SSH - secure shell (cryptographic network protocol) used for connecting to remove virtual servers.
- Reverse SSH - Used to access systems behind a firewall
- GUI - Graphical User Interface (how the user interacts with the web application)
- Raspberry Pi - Small single board computer
- Sensors - Array of sensors connected to raspberry pi
- RDS - Relational Database Service (relational database in the AWS Cloud)
- DynamoDB - Key Value Document Database
- Lambda - Serverless code execution
- Static IP Address - IP address not renewed and assigned to a specific device (in our case EC2 server)

1.3 Research

Once we were happy with the direction the Final Year Project was going to take, we visited the Botanic Gardens in Glasnevin and spoke to the head gardener. He gave us ideas that were feasible to implement given the time frame for the project such as the option to manually water as well as use the ML automatic watering system given uncontrollable environmental changes for example.

As we progressed through the planning phase, we researched various methods to deploy our project and landed on AWS EC2, a virtual server in the cloud where we had granular control over networking. After this, we required the server to connect to a raspberry pi on a local network. Initially we were going to use a third party service, it became apparent in the later stages of the project this was not feasible which we will discuss further in section 5.

The last element was sensors, based on feedback we gained from the Botanic Gardens, we selected an array of sensors that would be connected in series to the raspberry pi. They came in the form of a kit that included most of the necessary components we required.

From an accessibility point of view, we also wanted some other way to communicate with the application and have results spoken to the user. The clear contender for this was to use an Alexa Custom Skill as we were already using amazon products to interact and host the application.

2. System Architecture

2.1 System Architecture Description

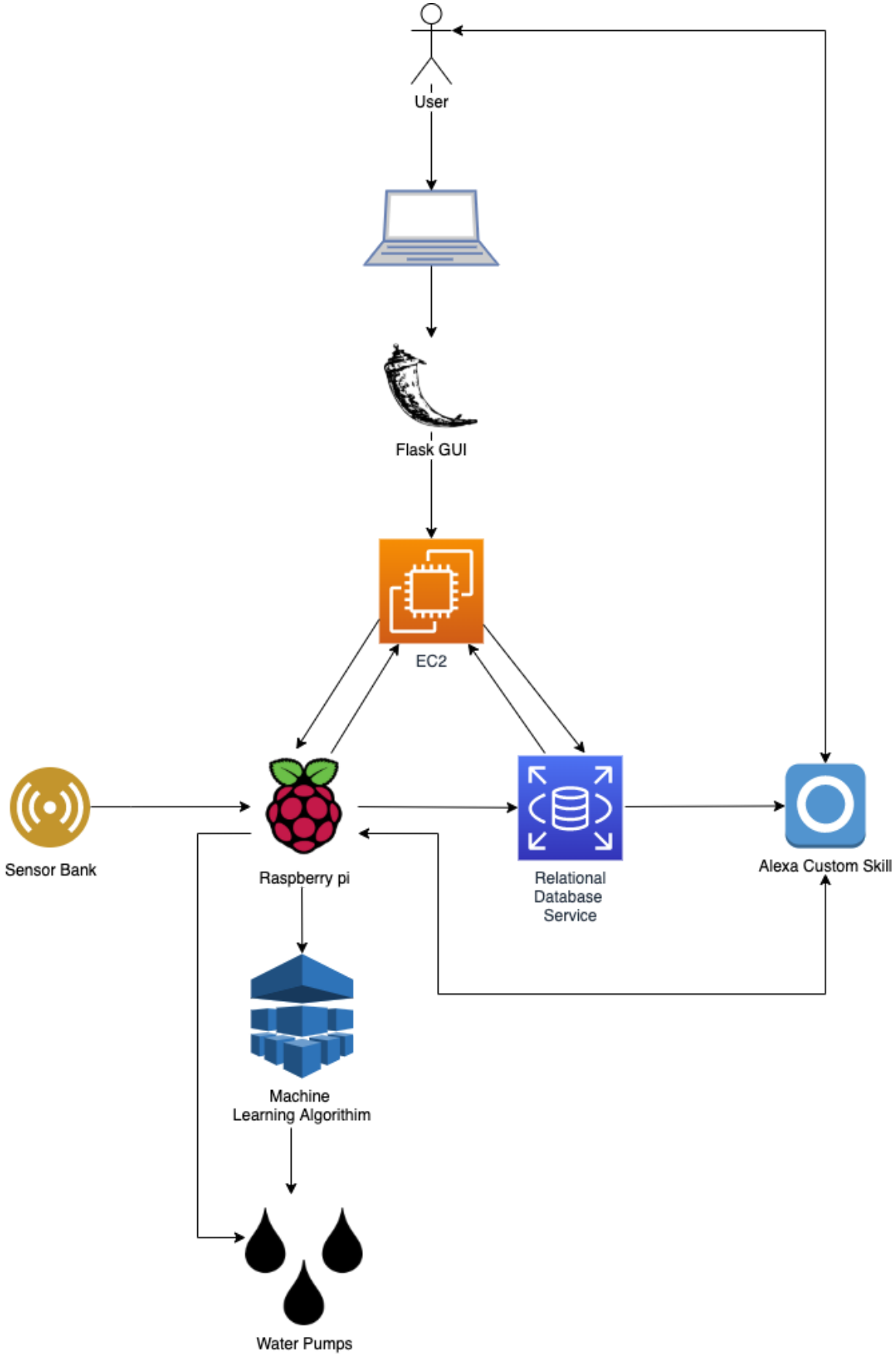
For our final year project we built a multi-layered architecture that consists of a GUI running on the flask framework. The flask server runs on AWS EC2 instance the application to be accessed from anywhere.

The EC2 instance then communicates with a raspberry pi via reverse SSH. This allows a connection to be made to the raspberry pi behind a private, secure network to ensure security. The raspberry pi is constantly reading multiple outputs from an array of sensors monitoring various parameters of a plant's environment. Additionally the raspberry pi commits the sensor outputs to an AWS RDS SQL database which we are then able to query for the various graphs on our dashboard.

The EC2 instance queries the database and receives up to date values on the environment prior to starting the ML module. The

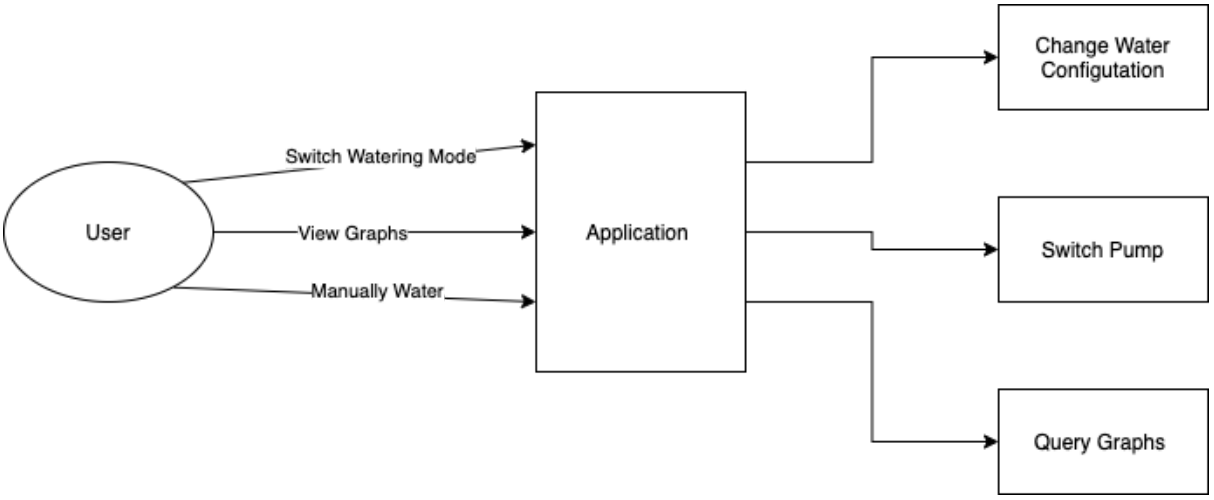
The same RDS instance mentioned above also provides outputs to our Amazon Alexa Custom skill.

2.2 System Architecture Diagram

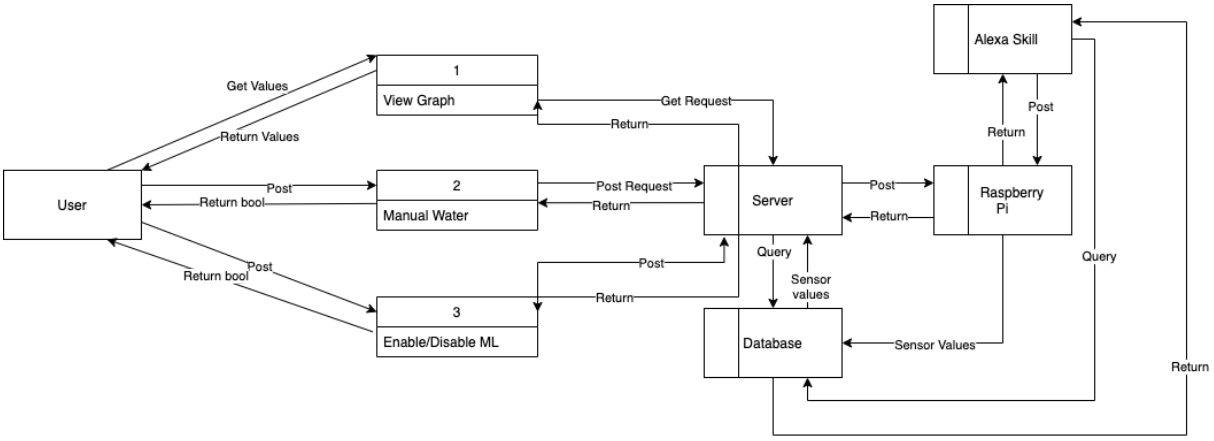


3. High Level Design

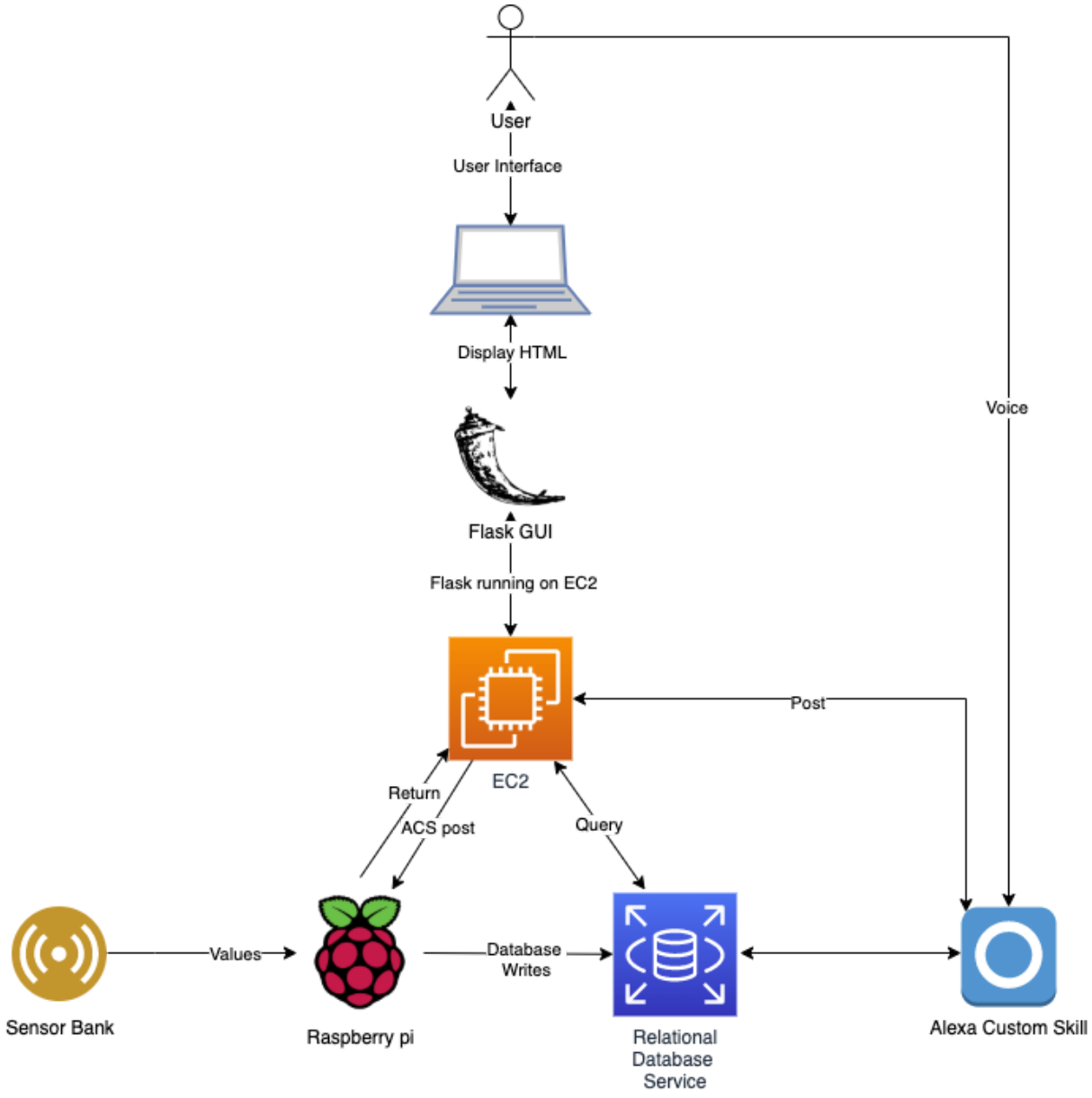
3.1 Context Diagram



3.2 Data Flow Diagram



3.3 Overall System Design



4. Development Workflow

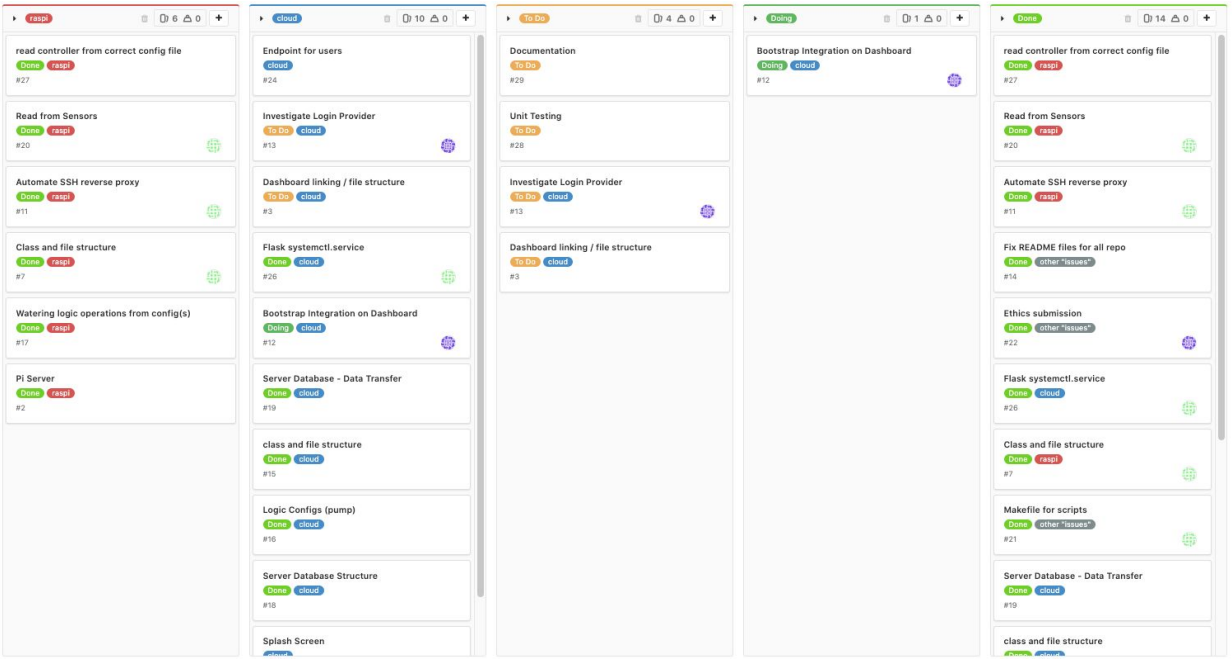
As we developed the project, we followed an agile approach to development, we utilised regular meetings, sprint planning and various tools to ensure the project was developed incrementally and pair programming was used to the extent that was possible for the later stages of the project.

4.1 Issues Board

Gitlab comes as standard with issue tracking, we utilised the issue board in particular. It was an extremely useful tool that helped to visualise the various features we had decided to implement, what we had done to date and what we had completed. Each issue is assigned a category with a number, in our case the categories were the raspberry pi elements and the cloud infrastructure.

During each commit we would associate it with an issue number, this further aided the tracking of issues.

```
~/Documents/college/FYP/2020-ca400-byrnj233-pereird2/src/server  develop !3
❯ git commit -m "#20 update SQL query"
```



4.2 Commit Practice

During each stage of the project, we would regularly commit work as we incrementally added features. Most features were given dedicated branches where they could be worked on with smaller features and bug fixes left to develop.

Once a feature on a branch was developed and testing performed, a merge request was created and the group member not responsible for the request would inspect and approve or deny the request if there were any apparent issues or implications for affecting existing code in the code base.

Any issues with this process were kept to a minimum by regular communication, on the occasion that code was being merged to a branch that the other team member was working on, the member in question would pull the new code and rebase accordingly to avoid issues further down the development process.

4.3 Scrum style meetings

Each week we would set aside time to plan the work that would be carried out as well as setting aside time to work through and attempt to minimise backlog from the previous features. At times this was not practical as we tried to balance work between assignments and exam preparation.

After the university was closed due to covid-19, communication was naturally hindered. However we made every effort to ensure regular communication by utilising the likes of discord to facilitate planning meetings and used the screen share feature of this software to implement a form of pair programming.

5. Implementation and Testing

5.1 Machine Learning Automation

The IOT Smart garden system utilized a Q-learning ML model to provide automation to the watering of the plants. It was chosen because it used a model free reinforcement policy to learn what actions to take. This meant that we were able to implement it without prior learning from a training set. At first this meant it performed poorly in its decisions, but with the use of a custom designed rewards system it made rapid improvements.

For developing the ML model, it was first applied in a development environment with a small q table of 3 states and 3 actions. By putting certain values through the program, we could inspect the resulting q table to determine if the ML was functioning correctly. The q table was then scaled up until it was applied to the live environment.

5.2 Continuous Integration

Development took place locally on separate branches, as a feature was implemented, the branch was merged to ‘develop’. As the code was merged, we would ensure the application was working locally by performing manual ad hoc testing to ensure any functionality that had been implemented up to that point was unaffected.

Makefile to push local configs

- We developed locally and when we were happy with the standard of the feature we used a makefile that would use SCP to push the local configuration to a mirror of the configuration of the EC2 instance which we would then apply the same ad hoc testing we performed locally.

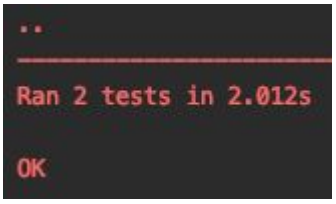
5.3 Testing

All testing scripts can be found in the testing directory.

We used pythons built in development tools unittest.mock for mocking calls to and from dependent modules. Our aim was to ensure that individual components of developed code were tested to the highest level. Unfortunately due to the current circumstances we were not able to perform the user testing that we desired. Reasons for this came down to the need to reapply for ethical approval as our current ethical approval would not cover the changes needed to test.

5.3.1 Raspberry Pi Testing results

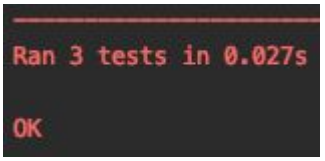
Test Controllers unit & integration tests:




```
pi@raspberrypi:~/src $ python3 controllers.py
pump 0 run for 0.15 seconds
pump 1 run for 0.45 seconds
pump 2 run for 0.15 seconds
```

- Ad Hoc testing:
    - All functions imported and testing individually to ensure connection to relays was correct.

Test ReadSensors unit & integration tests:

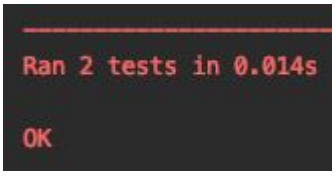



```
pi@raspberrypi:~/src $ python3 readSensors.py
DHT,fin
mcp,fin
DS1,fin

airTemp 21.0
humidity 51.0
mcp00 74.16951636541279
mcp01 31.34261520924933
mcp02 58.69972317212182
mcp03 100
mcp04 100
mcp05 100
mcp06 100
mcp07 0
soilTemp 16.375
```

- Ad Hoc testing:
    - All functions imported and testing individually to ensure connection to sensors was correct and to ensure that the sensors returned their raw values.

Test Store unit & integration tests:

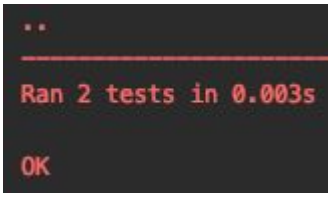



```
pi@raspberrypi:~/src $ python3 store.py
DHT,fin
mcp,fin
DS1,fin
{'2020-05-15 22:22:00': {'airTemp': 21.0, 'humidity': 51.0, 'mcp00': 71.238397
65510503, 'mcp01': 31.668295065950176, 'mcp02': 58.21120338707051, 'mcp03': 10
0, 'mcp04': 100, 'mcp05': 100, 'mcp06': 100, 'mcp07': 0, 'soilTemp': 16.187}}
['logs']
total time taken: 6.789089918136597
```

- Ad Hoc testing:
    - All functions imported and tested individually to ensure reading/writing of the correct files. Connection to the database was tested, including database connect failure, where files would be saved locally (on Pi) and pushed at a later date.

5.3.2 EC2 instance Testing results

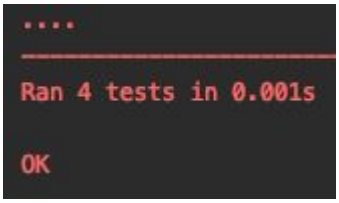
Database Query unit & integration tests:




```
ubuntu@ip-172-31-34-5:~/src$ python3 database_query.py
[(75.3094, 31.3426, 60.9795, 21.0, 51.0), (75.4722, 30.8541, 58.2112, 21.0, 51.0)]
['2020-05-15 22:40:00', '2020-05-15 22:35:00']
[75.3094, 75.4722]
(['2020-05-15 22:40:00', '2020-05-15 22:35:00'], [75.3094, 75.4722])
```

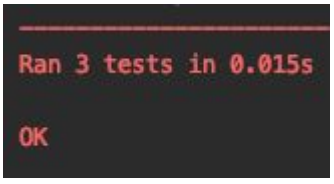
- The test for database query mocks the imported mysql library and ensures the correct arguments.

Sensor Learn test:



- Ad Hoc testing:
    - ML model as discussed earlier 5.1 mainly used ad hoc testing through its development.

Manual Water:



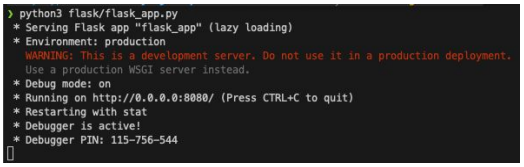

```
ubuntu@ip-172-31-34-5:~/src$ python3 manualWater.py
try
b'watered 0'
--------plant water connected!!----------
after
```

- Ad Hoc testing:
    - All functions imported and testing individually to ensure correct endpoints are reached and serve the correct values.

5.3.3 Flask Apps Testing

For the testing of the flask apps flask.py was run locally on "http://127.0.0.1:5000/". Testing was performed by passing different variable cases through the url.

Sample from ec2 flask being hit with /water/3 and passing the test.



```
❯ python3 flask/flask_app.py
 * Serving Flask app "flask_app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 115-756-544
```

```
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 196-598-593
-----++ 2
b'watered 2'
--------plant watered!!----------
b'watered 2'
188.141.103.91 - - [15/May/2020 22:46:54] "GET /water/3 HTTP/1.1" 302 -
```

GUI testing

- Phone Test
- Different Browsers
- Manual Test (keyboard walkthrough)

Endpoints

- Postman for manual watering
- Change ports to avoid sudo

6. Problems and Resolution

6.1 Code Duplication

Due to the multiple elements through our stack querying our database for example, code replication became apparent early on in the development process. We made an effort to minimise this as we progressed.

We combatted this for the early stages by making an effort to only transplant necessary methods pertaining to the function being carried out.

6.2 Time Management

Time management was difficult in every stage of the project given the high workload associated with a Final Year Project as well as maintaining a high standard of work in other modules. This was particularly difficult around the exam periods of both semesters as well as coming up to assignment due dates. This was exacerbated by the current pandemic, adjusting to working from home as well as modifying our workflow to account for not being able to work in the same room was difficult.

We were able to mitigate this by heavily reducing the work expected to be done in the days/weeks leading up to exams and assignments as well as using various software to keep in close contact as we adjusted to working from home.

6.3 Covid-19

DCU was closed in March due to the covid-19 pandemic. This undoubtedly had an impact on our workflow. Adjusting to working from home was a major challenge.

We used various software to maintain our workflow to the extent that was possible with software like zoom to hold meetings with our supervisor.

6.4 Git

The third year project was useful for learning Git at a high level for version control. However, this year we decided from the beginning of the development process to use an industry-like approach. The use of branches for example introduced an extra level of complexity.

To combat this we communicated clearly when creating merge requests and working off different branches as standard.

6.5 EC2 Security Groups

During the configuration process of the project we had significant issues maintaining connections via SSH and allowing connections via other protocols (HTTP/HTTPS).

EC2 has the option to configure a specific/range of IP addresses (Security Groups) and protocols that are allowed to interact with it. As such we were forced to reconfigure our Security Groups to allow SSH from all IP's to account for the fact that there would be incoming connections from behind the DCU network and home connections from both project workers.

6.5 Raspberry Pi Integration

As we began to connect the EC2 instance running our application to the raspberry pi backend we realised there would be issues due to network connections, in order to avoid opening a port on a home network and jeopardising said network, we researched and subsequently found a third party service, remote.it that would allow us to perform these connections securely. However, as we progressed we realised their API was not returning the necessary information to perform the desired functionality reliably.

To combat this we decided to do this work manually with a reverse ssh tunnel. This was timely and difficult to implement but we were able to create it in such a way that connections between EC2 and the raspberry pi operating behind a private home network was more beneficial in the long term despite the significant time commitment to get it working as we had full, granular control on operating listeners that would allow us to get and make desired information/changes to the raspberry pi local configurations.

6.6 Alexa Developer Console - Packaging

A part of our project was to use an alexa custom skill to check sensor outputs from the raspberry pi. We were able to reuse code from the graph that made use of the SQL Connector Python module. It's possible to integrate an AWS Lambda function to the skill to carry out some custom logic which is where we queried our database. However Lambda running the Python 3.X comes with minimal packages and didn't come with the necessary packages to use the Python MySQL connector module.

As such we were forced to create a 'layer' which allows users to package libraries and upload them to the Lambda platform. This was a time consuming process given that we had to create and ensure installations were correct locally and package them, ensuring to use a virtual environment to isolate the specific required package. Additionally, since the Alexa Custom skill platform comes with an integrated lambda function editing environment, a lot of time was spent trying to add the layer. What became apparent after some time and research however was the fact that there is a requirements section where you can specify necessary libraries for the function to run. This felt like a lot of time wasted given the easy nature of using the requirements file that was not immediately apparent to us.

6.7 Chart js Libraries integrated with Flask

We decided on the charting library that uses javascript ‘chart.js’. We went with this as it interacts well with the flask framework and html. However, we ran into pretty significant issues with options associated with this, namely colouring and style. To debug this issue we mainly used the inspect element feature to check request headers and responses.

Despite this we weren’t able to overcome it, as such we were forced to place different values on individual graphs.

6.8 Database/Relearning SQL

Originally we had hoped to adopt a NoSQL approach to the data storage element of our project. As such we used AWS DynamoDB originally, however as we progressed we realised that as the database grew in terms of the amount of values being committed. Due to the nature of dynamo we would have had to pull all the values and parse them for the required values client side.

As such we made the switch to a SQL style database. This allowed us more granular control over queries and removed a lot of work to be carried out client side in order to make the application as responsive as possible

6.9 Static IP Requirement

In order to access the deployed dashboard we would be affected by the fact that our EC2 instance has the ability to change IP addresses.

To mitigate this we had to research a way to allocate a static IP to the instance, to do this we used the Elastic IP feature of AWS EC2. This allowed us to reliably connect to the stable EC2 endpoint and removed any ambiguity surrounding the ability to connect to the dashboard.

6.10 Hardware Inconsistencies

The ML program calculated its desired action based upon the current state of the environment. Eg. the values from the sensors. We had issues from time to time where the ML made a decision to overwater. After some hunting we found that the sensors would occasionally jump and send in incorrect values. In order to combat this, we reworked the ML so that it normalized 5 values into 1. The change greatly improved the performance, and stopped over watering issues.

6.11 Human Interference

Similar to when we had hardware inconsistencies, we noticed some odd values in being set by the ML. The system was setting the water output for one of the plants substantially lower than the others. It was soon discovered that one of our mothers

had been watering that specific plant so she could plant it in her garden later. It was very interesting to see how the ML model adapted to the situation and made sure that the plant was being watered at the same level as the others.

7. Future Work

7.1 Development

Going forward, there are several features that would be useful to implement.

- Graph
    - Use the graphing library to display the various parameters on one graph that dynamically changes Y value coordinates in order to reduce the size of the complexity of the dashboard.
    - Expand the options for display, currently the graph shows the 5 most recent sensor outputs, in the future we would like to be able to specify with, for example, a wider range of historical sensor outputs. While this information is accessible to use and the ML model, it wasn't feasible in our timeframe to implement this custom query model to our dashboard.
- Login Feature
    - Using a service such as AWS Cognito to serve as an oauth2 provider for the purposes of login would be useful.
- Manual Watering
    - Currently the manual watering of our raspberry pi is set to a static value, we would like to implement an option that would allow the user to specify how much water can be delivered by the feature.
- HTTPS Connection
    - Reflecting the login feature previously mentioned, securing HTTPS valid certification would be desirable.

7.2 Expansion of sensor/pump bank

Expanding the number of plants that can be monitored and environment's altered by the application would be extremely useful. Another option would be to be able to expand the sensor bank to include for example soil PH.

Additionally the alterations made could be expanded. Currently watering is the only parameter change, it would be feasible to implement a heat lamp for example assuming it could be wired to the mains as the main 5 volt GPIO header may not be useful.

7.3 Move to Neural Network

The Q-Learning Ml model that we used mostly worked for our purposes. However we quickly came to the limitations of this design. We found that when we passed in all parameters as a state and all actions cubed, the resulting q-table would have over a billion cells. The q-table is pre build in our design and then filled with random values, the downside to this is many of the states are unrealistic and won't ever be reached leading to wasted space. The first solution we had to this would be to dynamically build the a q-table. Our preferred and long term solution would be to implement a simple Neural Network. We were initially turned off by the idea because the training

would take too long. The goal would be to build a neural network around our current reinforcement model.This would solve the issue of having every state and action stored, and instead pass data through to the nn.

8. Conclusion

Overall, the project was an enjoyable learning experience. Neither project members had any prior experience working with IOT devices and as such it was a challenging experience. Iit was a steep learning curve but a rewarding endeavour nonetheless to create a system that integrated with web technologies we were more familiar with.

There were undoubtedly hindrances along the way, especially considering the shutdown of DCU and the major challenges it presented to our workflow, however we felt we were able to continue working and produce a feature-rich application that met to the extent possible our goals.

Below are the plants our system worked with over the course of the project as of the date of submission (16/05/20).

9. References

1. https://www.chartjs.org/docs/latest/getting-started/
2. https://www.patricksoftwareblog.com/creating-charts-with-chart-js-in-a-flask-application/
3. https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/
4. https://medium.com/@data.corgi9/real-time-graph-using-flask-75f6696deb55
5. https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world
6. http://web.mst.edu/~gosavia/neural_networks_RL.pdf
7. https://docs.aws.amazon.com/ec2/index.html?nc2=h_ql_doc_ec2
8. https://docs.aws.amazon.com/rds/index.html?nc2=h_ql_doc_rds
9. https://developer.amazon.com/en-US/docs/alexa/custom-skills/steps-to-build-a-custom-skill.html
10. https://docs.aws.amazon.com/lambda/latest/dg/configuration-layers.html