

National College of Ireland

BSHCSD4

Software Development

2021/2022

Adam Downey

x18455846

x18455846@student.ncirl.ie

Emanate

Technical Report

Contents

Executive Summary	4
1.0 Introduction	5
1.1. Background	5
1.2. Aims	5
1.3. Technology	6
1.4. Structure	6
2.0 System	7
2.1. Requirements	7
2.1.1. Functional Requirements	7
2.1.1.1. Use Case Diagram	7
2.1.1.2. Requirement 1: Cables and soldered pieces are attached to the corresponding hardware	7
2.1.1.3. Description & Priority	7
2.1.1.4. Use Case	8
2.1.1.5. Use Case Diagram	9
2.1.1.6. Requirement 2: Raspberry Pi with radio bonnet is given internet	9
2.1.1.7. Description & Priority	9
2.1.1.8. Use Case	9
2.1.1.9. Use Case Diagram	11
2.1.1.10. Requirement 3: Code must be executed on microcomputers	11
2.1.1.11. Description & Priority	11
2.1.1.12. Use Case	11
2.1.1.13. Use Case Diagram	12
2.1.1.14. Requirement 4: Firebase writes to database	13
2.1.1.15. Description & Priority	13
2.1.1.16. Use Case	13
2.1.1.17. Use Case Diagram	14
2.1.1.18. Requirement 5: Android Application displays sensor metrics	15
2.1.1.19. Description & Priority	15
2.1.1.20. Use Case	15
2.1.2. Data Requirements	16
2.1.2.1. Requirement 1: Data from sensors should be accurate	16
2.1.2.2. Description & Priority	17
2.1.3. User Requirements	17
	1

2.1.3.1.	Requirement 1: The Android application should be of good performance	17
2.1.3.2.	Description & Priority	17
2.1.3.3.	Requirement 2: Node hardware should be mobile	17
2.1.3.4.	Description & Priority	17
2.1.4.	Environmental Requirements	17
2.1.4.1.	Requirement 1: The nodes should record metrics in an outside environment	17
2.1.4.2.	Description & Priority	17
2.1.4.3.	Requirement 1: The Raspberry Pi with the radio bonnet should be inside	17
2.1.4.4.	Description & Priority	17
2.1.5.	Usability Requirements	18
2.1.5.1.	Requirement 1: Concerning metrics to be made clear	18
2.1.5.2.	Description & Priority	18
2.2.	Design & Architecture	18
2.2.1	Software	18
2.2.2	Operating Systems	19
2.2.3	Languages	19
2.2.4	Key Libraries	19
2.2.5	Raspberry Pi Components	19
2.3.	Implementation	20
2.3.1	File layout (Node)	20
2.3.2	File Layout (Master Radio)	21
2.3.3	rfm69_and_bme688.py - Main Check	21
2.3.4	rfm69_and_bme688.py - bytes_data	21
2.3.5	rfm69_and_bme688.py - send_packet	22
2.3.6	rfm69_and_bme688.py - receiver_check	22
2.3.7	rfm69_and_bme688.py - run	23
2.3.8	rfm69_and_bme688.py - Running program	23
2.3.9	rfm69_bonnet.py - Main Check	24
2.3.10	rfm69_bonnet.py - weather_api_data	25
2.3.11	rfm69_bonnet.py - pressure_to_altitude	26
2.3.12	rfm69_bonnet.py - token_refresh	26
2.3.13	rfm69_bonnet.py - run	27
2.3.14	Firebase - Realtime Database	28
2.3.15	Android Application - Emanate.kt (Application Class)	30
2.3.16	Android Application - SignInActivity.kt	31

2.3.17	Android Application - SignUpActivity.kt	33
2.3.18	Android Application - MainActivity.kt	35
2.3.19	Android Application - NotificationsActivity.kt	37
2.3.20	Android Application - ListAdapter.kt	41
2.3.21	Android Application - WorkerMetrics.kt	42
2.3.22	Record.kt	43
2.3.23	Android Application - MyWorker.kt	44
2.3.24	Android Application - TimeConverter.kt	45
2.4.	Graphical User Interface (GUI)	46
2.4.1	RFM69 Radio Bonnet and Raspberry Pi	46
2.4.2	Node components (BME688 Sensor, RFM69HCW Breakout, LEDs, and resistors)	47
2.4.3	Raspberry Pi	48
2.4.4	Blue LED sending data	49
2.4.5	Red LED receiving data	50
2.4.6	Splash screen	52
2.4.7	Sign in	53
2.4.8	Sign up	54
2.4.9	Main screen - On open	55
2.4.10	Main screen - Enter node id	56
2.4.11	Main screen - Toast message	57
2.4.12	Main screen - Metrics	58
2.4.13	Notifications screen	59
2.4.14	Notification - Notification bar	60
2.4.15	Application Icon	60
2.5.	Testing	60
2.5.1	LED Test	61
2.5.2	Unwire BME688 Sensor	63
2.5.3	RFM69HCW Unwire	65
2.5.4	Internet Robustness	67
2.5.5	Node Distancing	69
2.5.6	bytes_data test	73
2.5.7	pressure_to_altitude and test_weather_api_data	73
2.5.8	Database Rules	74
2.5.9	Android Timestamp Conversion	76
2.6.	Evaluation	78

3.0	Conclusions	78
4.0	Further Development or Research	79
5.0	References	80
6.0	Appendices	80
6.1.	Project Proposal	80
1.0	Objectives	80
2.0	Background	81
3.0	State of the Art	81
4.0	Technical Approach	82
5.0	Technical Details	82
6.0	Special Resources Required	83
7.0	Project Plan	84
8.0	Testing	85
1.1.	Ethics Approval Application (only if required)	86
1.2.	Reflective Journals	102
1.3.	Other materials used	109

Executive Summary

The purpose of this report was to give a broad overview of the current state, and future of Emanate. By doing so, a greater understanding of the overall project was given, while also increasing the potential for new ideas.

I explained the central idea for Emanate in the opening of the report. In this section, I mention all the areas that I was about to cover when designing the project. Outlining these areas that are yet to be done has reinforced the outcome of the project and is useful to see how the project's main idea was shaped over time.

While working through this report and testing with the hardware, I ran into a lot of issues with the physical components. Some of the components were not longer feasible, which slowed down the development of the project. I eventually had to adapt the hardware which in turn made the project less mobile.

In the code for Python and Kotlin I made use of various data structures, I made use of lists and arrays to handle the node data for parsing, writing, and reading from the database with the help of the Pyrebase library. Android also made much of the project easier to work with by auto connecting Firebase to the Android application through my Google account. I also used Gson which was a great implementation to handle the data being read from the database.

Deeper analysis is covered in this report with direct influence from real world outdoor activity instructors and reasons why certain metrics were covered and why others were not. References can be found at the bottom of this report addressing these aspects.

Emanate was completed being largely functional with the main limitations being the costs for the hardware, the sensor range, and the block by Android to test the functions implemented into the Android application.

1.0 Introduction

1.1. Background

This project was chosen to bring me out of my own comfort zone. I have been working with web applications for most of my college projects and feel that it would benefit me to design a project with a more diverse use of technologies. Emanate will include three areas in software (IoT, Android Development, and backend database management systems), and add an additional aspect by making use of the outdoors.

The main area that is quite new to me is Android Development. I have only been exposed to this technology this semester and hope to challenge myself further by designing the mobile application in Kotlin. This is also a new language that I have never worked with before, giving the project a greater complexity.

I have worked with a Raspberry Pi in a past project and took a great liking to the Operating System and creative potential for add-on components. This project is also a good opportunity to explore Adafruit's wide array of products that provide networking capabilities for the Raspberry Pi.

This will give me the chance to work with Firebase for the first time. Working with this platform will allow me to bypass a lot of the setup for manually creating a database with software like MySQL Workbench and Heroku.

1.2. Aims

Some rural areas across the world have a lack of cellular data towers for mobile phones, which prevent a reliable communication source from being possible for travellers. This project will try to keep hikers and people on similar expeditions connected through radio communications.

All metrics that a person has on their body or near their environment, will be sent via a radio transmitter to a nearby headquarters with a radio receiver. Emanate will call the on-body technology for a person a 'node' for easier explanation in this report. I will try to make Emanate be able to handle and display this data. There should be warnings from a mobile app if any concerning metrics are found.

The on-body data from the team members will be transferred to a database. A phone application will pull from the database to manipulate and display this data, transmitting emergency warnings to the phone if any team member has concerning metrics.

1.3. Technology

This project is largely hardware-based technology. I will need at least two microcomputers to control Emanate's related components. One for executing the transmission of metrics, and another for executing the receiving of data.

Now two Raspberry Pi's will be controlling all the related components with the CircuitPython and Blinka libraries. In my planning phase for the project, I expected to use a Gemma V2 or Gemma M0. These would be quite mobile electronic platforms, but unfortunately, they do not supply enough electronic connectors for the sensors that Emanate needs. An Adafruit Feather 32u4 with RFM69HCW could replace a Raspberry Pi in future design.

The radio components being used will be two RFM69HCW 433MHz radios, one in a breakout form, and another in a bonnet with an OLED screen. The radio with the bonnet will be the main device at a location within range of the breakout radio signals away from the hikers. The radio breakouts will be part of the node with each hiker.

One sensor that measures a wide range of attributes will be part of the nodes. The Adafruit BME688 will handle measuring temperature, altitude, humidity, barometric pressure, and VOC gas. Heart rate monitors are too expensive for this prototype, no hardware will cover this area.

Emanate will be sending the node metrics to Firebase. This Firebase user information will allow an Android mobile phone application to display and read this data. The mobile phone application will be written in Kotlin and have integrated methods for accessing Android's push notifications. This is in the case of concerning metrics being read, such as a drop in altitude, a dangerously low temperature, or the presence of cancerous compounds.

1.4. Structure

The requirements sections elaborate on the flow of Emanate. Most of the functional requirements rely on the setup of the microcomputers as expected. The rest of the functional requirements will be dependent on how the software is controlled by Firebase and the code I write for the Android application.

Design and architecture include all the attributes that make the system of Emanate. This includes dependencies, languages, operating systems, components, and software being used. Data structures, classes, and everything within the Android application's code now is unknown, which is why it is not included.

Implementation shows all the files being used in the hardware. These are all files that on the Raspberry Pi OS. The files are explained with snippets of code and descriptions of their functionality

Since there is nothing to look at for the GUI, I show the hardware for my project with pictures. The connectors are also included in a table for a clear viewing of the GPIO set up.

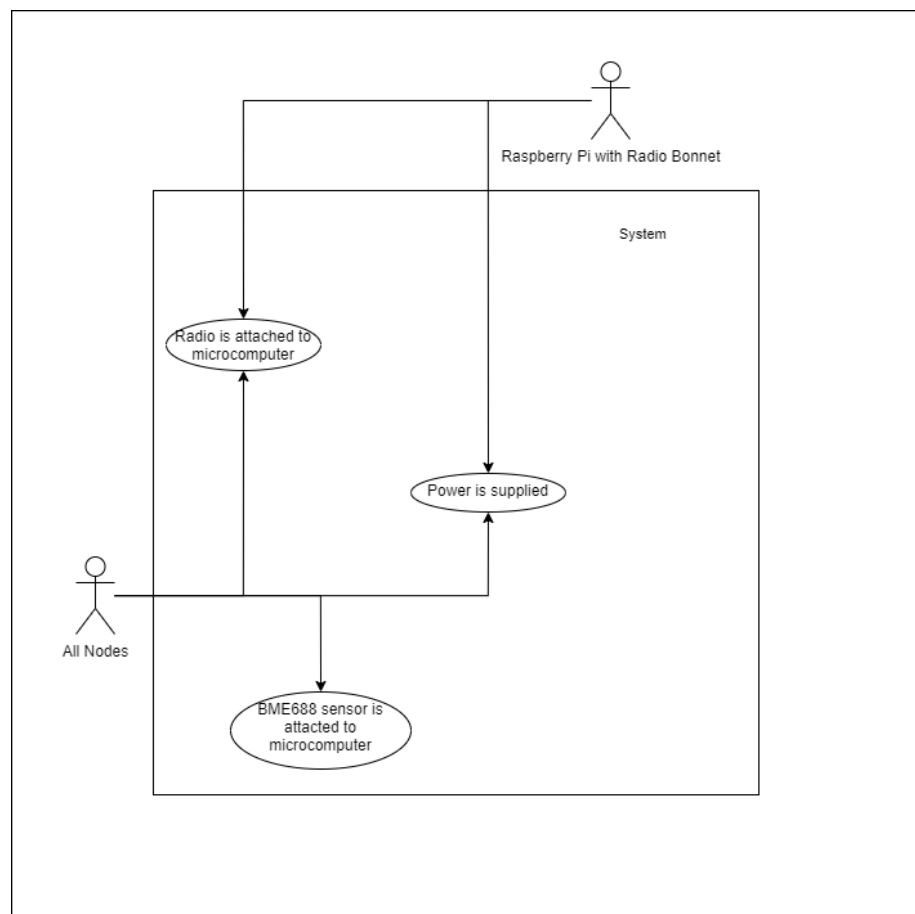
Reflective journals, the project proposal, ethics application, and references are also included at the bottom of the document.

2.0 System

2.1. Requirements

2.1.1. Functional Requirements

2.1.1.1. Use Case Diagram



2.1.1.2. Requirement 1: Cables and soldered pieces are attached to the corresponding hardware

2.1.1.3. Description & Priority

Rank 1

All the hardware needs to be connected to a power source, and microcomputer components need to be connected to the microcomputer with jumper cables. This is the top requirement because it makes recording and sending the sensor data possible.

2.1.1.4. Use Case

Hardware Connection, Rank 1

Scope

The scope of this use case is to provide electrical power with a USB power cable. All additional components are to be connected, like radio and environmental sensors.

Description

This use case describes the setup for hardware connectivity.

Use Case Diagram

Actors: All Nodes, Raspberry Pi with Radio Bonnet

Flow Description

Precondition

The user has the hardware, cables, and a power supply.

Activation

The use case starts when the hardware is being setup in the location

Main flow

1. The hardware is in the user's vicinity.
2. Cables are connected from microcomputers to sensors and radios.
3. Power is supplied to the microcomputers with batteries and a power outlet.

Alternate flow

1. The hardware is in the user's vicinity.
2. Cables are connected from microcomputers to sensors and radios.
3. Power is supplied to the microcomputers with only power banks

Exceptional flow

1. The hardware is in the user's vicinity.
2. Cables are not connected in the right way to sensors and radios.
3. Power is supplied but the flow is broken.

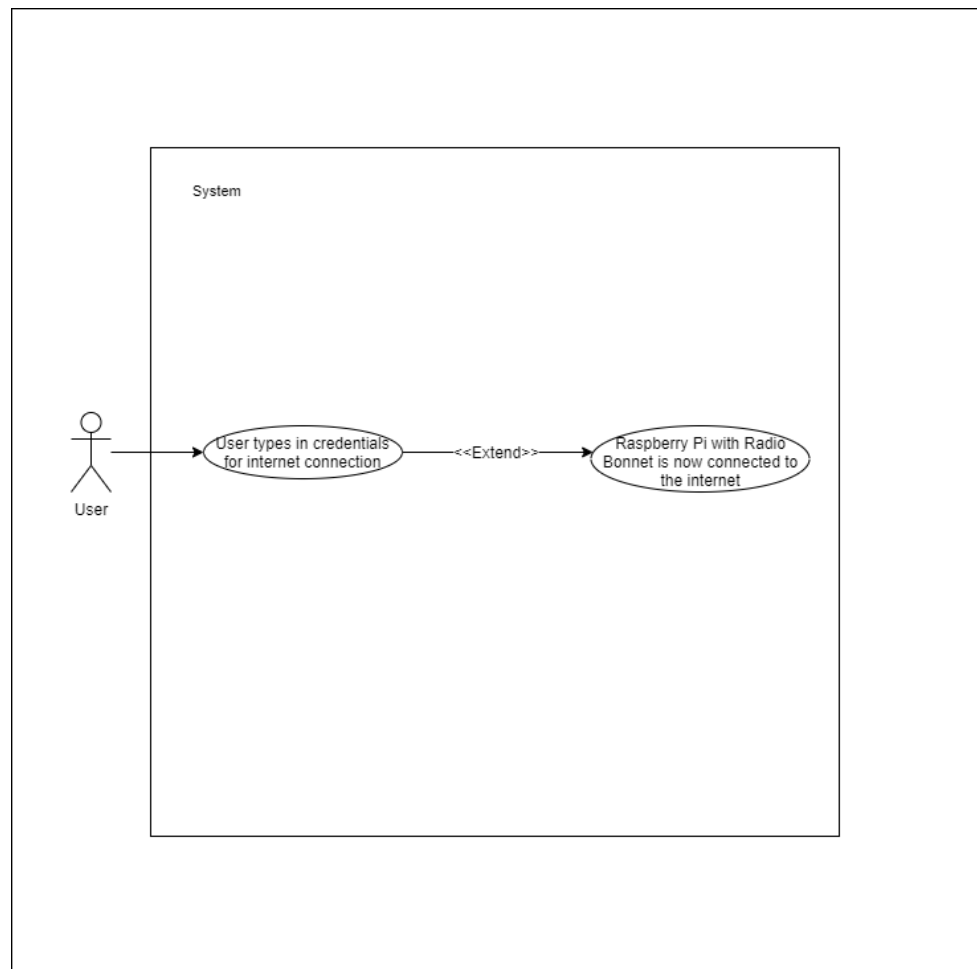
Termination

The system has all of the hardware operating and should continue to function. When power or connector cables are detached, the system will terminate.

Postcondition

The system is in an idle state, waiting for code to be executed.

2.1.1.5. Use Case Diagram



2.1.1.6. Requirement 2: Raspberry Pi with radio bonnet is given internet

2.1.1.7. Description & Priority

Rank 2

Internet is supplied to the Raspberry Pi. This allows for the metrics from the sensors to be sent to Firebase. This is an important requirement because it allows for the Raspberry Pi to transport data beyond its own scope.

2.1.1.8. Use Case

Internet-Connected, Rank 2

Scope

The scope of this use case is to bring an internet connection to the Raspberry Pi with the radio bonnet.

Description

This use case describes the process of connecting to the internet in the Raspbian OS.

Use Case Diagram

Actors: User

Flow Description

Precondition

The user has an internet router, with a password for it.

Activation

The use case starts when the user has the Raspbian OS loaded on a screen.

Main flow

1. The Raspbian OS is being viewed by a user on a computer screen.
2. The user selects the internet connection icon.
3. The user selects the router from the wireless options.
4. The user inputs the router's password.
5. The Raspberry Pi is connected to the internet.

Alternate flow

1. The Raspbian OS is being viewed by a user on a TV screen.
2. The user connects an Ethernet cable from a router to the Raspberry Pi.
3. The user configures the Ethernet cable that is connected.

Exceptional flow

1. The Raspbian OS is being viewed by a user on a computer screen.
2. The user selects the internet connection icon.
3. The user attempts to input a password for the selected router option.
4. The password has failed to authenticate.
5. The Raspberry Pi cannot connect to the internet.

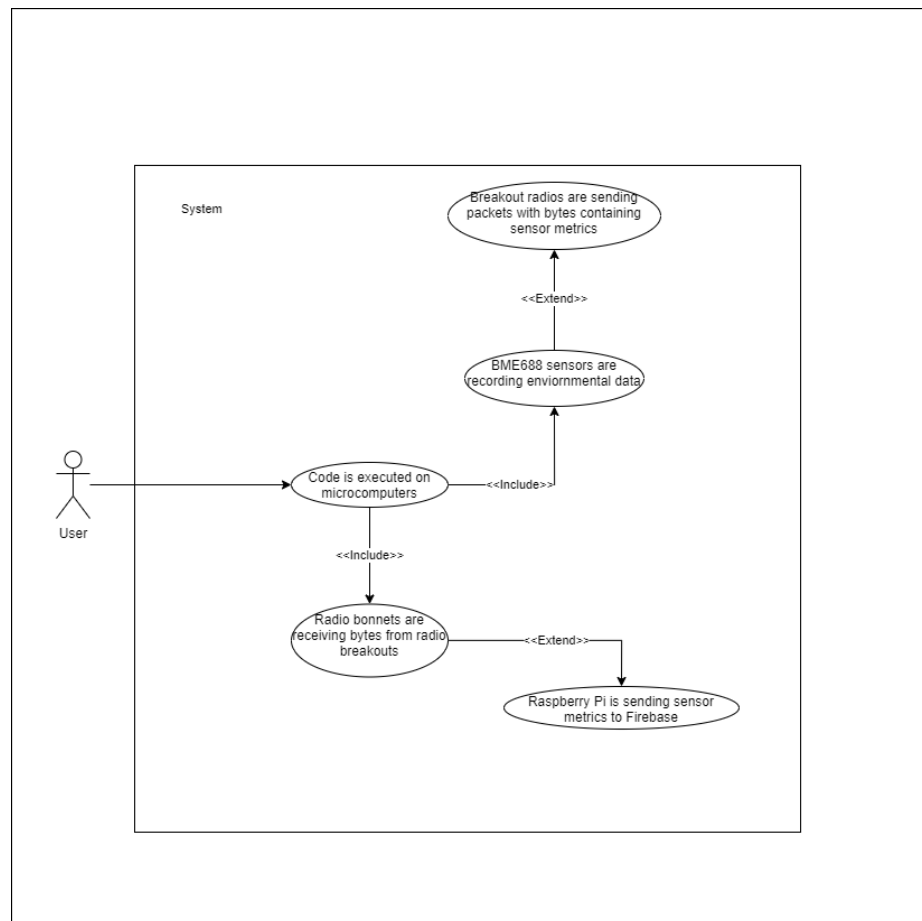
Termination

The Raspberry Pi in the system can access the internet. The system will be terminated when the internet connection is ceased.

Postcondition

The Raspberry Pi is idle while connected to the internet.

2.1.1.9. Use Case Diagram



2.1.1.10. Requirement 3: Code must be executed on microcomputers

2.1.1.11. Description & Priority

Rank 3

For the radio signals and sensors to operate in the desired way, the Python code for it must be executed. This is crucial for the components to function; the hardware will not perform any action without this requirement.

2.1.1.12. Use Case

Code Execution, Rank 3

Scope

The scope of this use case is to make the microcomputers perform the actions we want them to, dependant on each component.

Description

This use case describes the execution of the software inside the microcomputer's hardware.

Use Case Diagram

Actors: User

Flow Description

Precondition

The user knows how to execute Python code in a terminal.

Activation

The use case starts when the hardware is running all of its software correctly.

Main flow

1. The Raspbian OS is being viewed by a user on a computer screen.
2. The user attempts to run the code in a terminal.
3. Components are receiving and transmitting data over a radio frequency.

Alternate flow

1. The Raspbian OS is being viewed by a user on a TV screen.
2. The user attempts to run the code in an IDE with the Python files.
3. Components are receiving and transmitting data over a radio frequency.

Exceptional flow

1. The Raspbian OS is being viewed by a user on a computer screen.
2. The user attempts to run the code in a terminal but runs into errors with a physical hardware issue.
3. Code failed to execute; the system has failed.

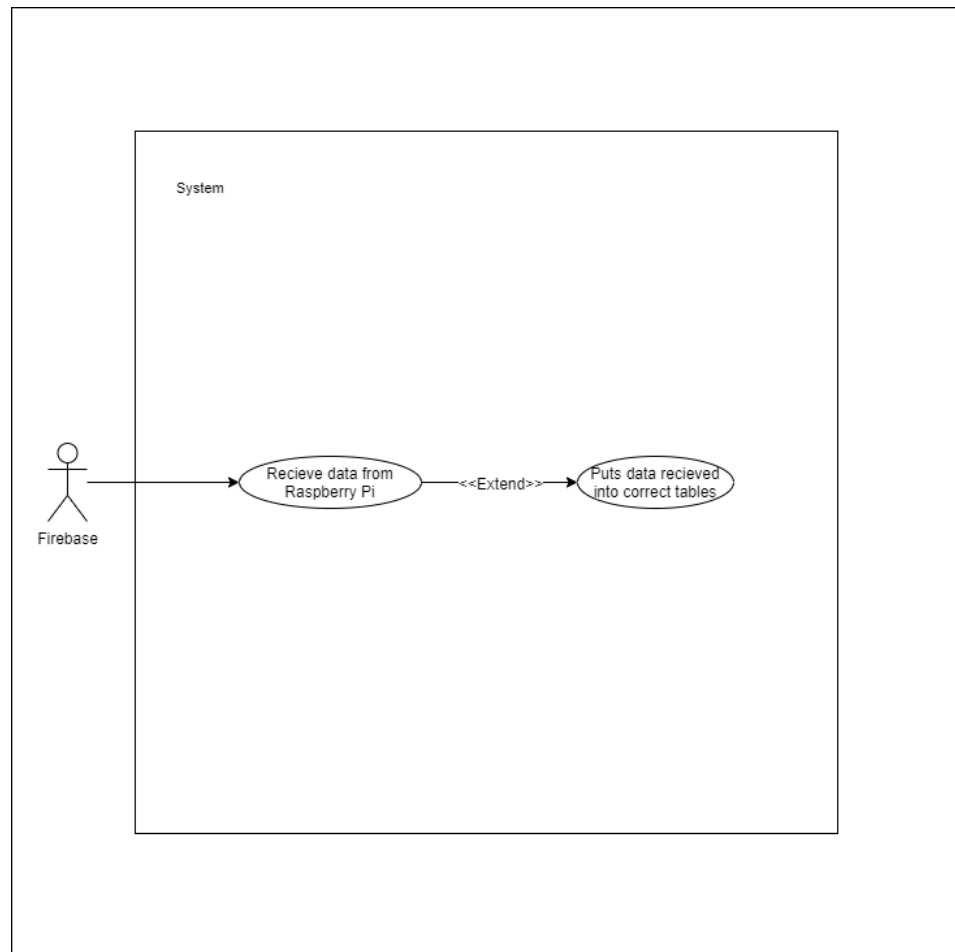
Termination

The system records environmental data and transmits it over a frequency, this will be constantly done until the executable file is terminated.

Postcondition

The Python code in the system is about to reach Firebase code.

2.1.1.13. Use Case Diagram



2.1.1.14. Requirement 4: Firebase writes to database

2.1.1.15. Description & Priority

Rank 4

All of the metrics from the Raspberry Pi with the radio bonnet are received. It is then sorted by Firebase, inserting the metrics into the corresponding tables. This is required for the phone application to read and display the data from the sensors.

2.1.1.16. Use Case

Database Writing, Rank 4

Scope

The scope of this use case is to let Firebase manage all of the metrics being sent to it, writing to the correct tables that they correspond to.

Description

This use case describes the process of Firebase working with the sensor metrics.

Use Case Diagram

Actors: Firebase

Flow Description

Precondition

Firebase has received the sensor metrics.

Activation

The use case starts when the sensor metrics are being written.

Main flow

1. Firebase looks at the sensor metrics.
2. Firebase writes these metrics to tables that match the categories.
3. The database is available to be accessed.

Alternate flow

1. Firebase looks at the sensor metrics.
2. Firebase writes these metrics to the same table inside the database, ignoring the categories.
3. The database is available to be accessed.

Exceptional flow

1. Firebase looks at the sensor metrics.
2. Firebase writes the metrics in random areas across the database.
3. The database is incorrect when read.

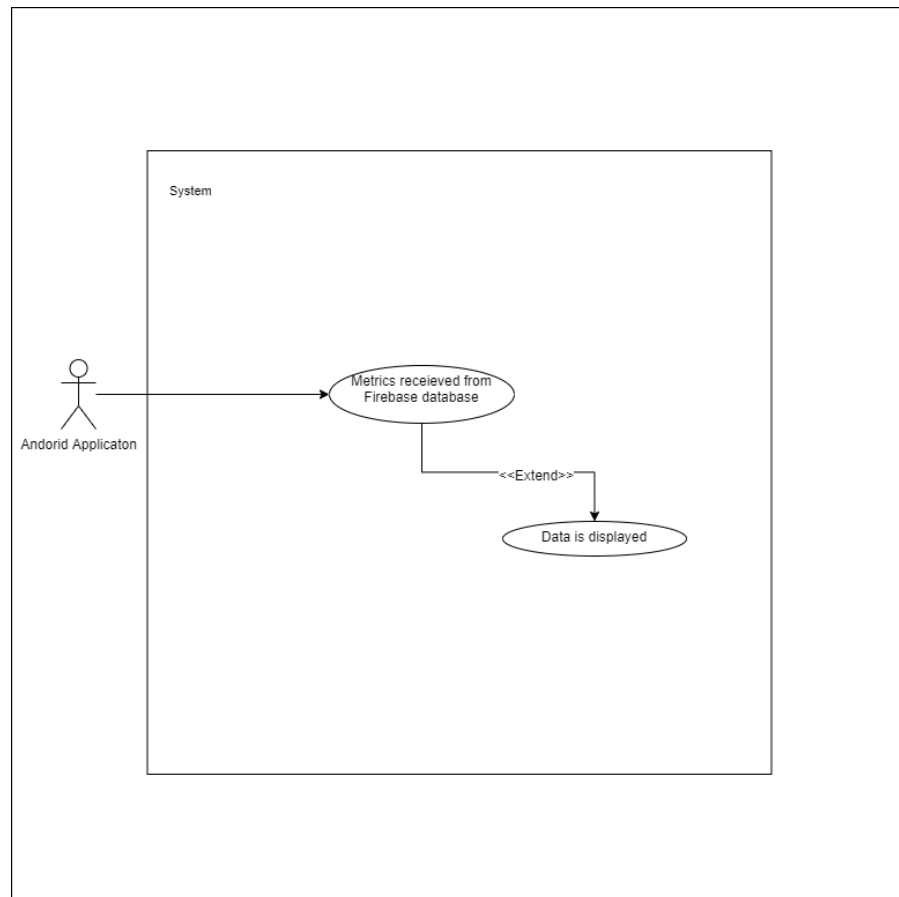
Termination

The database in the system is ready to be read, updating with more metrics that it receives.

Postcondition

Firebase waits for more metrics coming from the Raspberry Pi bonnet.

[2.1.1.17. Use Case Diagram](#)



2.1.1.18. Requirement 5: Android Application displays sensor metrics

2.1.1.19. Description & Priority

Rank 5

Metrics are displayed in the mobile app for the user. The formatting will be handled in Kotlin code with the Android App. This is the only way for the user to see the metrics without looking at the back-end database.

2.1.1.20. Use Case

Android Sensor Metric Viewing, Rank 5

Scope

The scope of this use case is to display all of the sensor metrics available to the user in the Android application.

Description

This use case describes the projection of the sensor metrics onto the Android app's display.

Use Case Diagram

Actors: Android Application

Flow Description

Precondition

Android application has received the sensor metrics.

Activation

The use case starts when the sensor metrics are being written.

Main flow

1. The Android application looks at the sensor metrics that it has from the database.
2. The Android application sorts the sensor metrics accordingly with the UI.
3. The Android application has the sensor metrics displayed to the user.

Alternate flow

1. The Android application looks at the sensor metrics that it has from the database.
2. The Android application assigns a node number to the metrics that it has available.
3. The Android application sorts the sensor metrics accordingly with the UI, along with attaching the corresponding node number.

Exceptional flow

1. The Android application looks at the sensor metrics that it has from the database.
2. The Android application reads the sensor metrics incorrectly, giving skewed metrics for the UI.
3. The UI displays data that is incorrect.

Termination

The system will continually display new metrics for the user until the application is closed.

Postcondition

The system will monitor the metrics in case of concerning metrics.

2.1.2. Data Requirements

2.1.2.1. Requirement 1: Data from sensors should be accurate

2.1.2.2. Description & Priority

The metrics recorded by the BME688 sensors should give readings that are accurate within each category. For example, the temperature metrics of a sensor's environment must give a reading that is not inaccurate by 2 degrees Celsius.

2.1.3. User Requirements

2.1.3.1. Requirement 1: The Android application should be of good performance

2.1.3.2. Description & Priority

I do not want the metrics and code that I implement to slow down the Android application. This is to create a smoother experience for the user, but also to avoid running into any issues within the Android application.

2.1.3.3. Requirement 2: Node hardware should be mobile

2.1.3.4. Description & Priority

Currently, the nodes are not mobile. This is an early development decision that I want to change in the future. Using a power bank as the power source for the nodes will allow them to be mobile but may make the nodes overly cumbersome.

2.1.4. Environmental Requirements

2.1.4.1. Requirement 1: The nodes should record metrics in an outside environment

2.1.4.2. Description & Priority

For Emanate's true functionality to be tested, the nodes will need to be outside. Recording the sensors inside might only be possible now but will have to be tested outside with an improvised method later in development. At that point, this requirement will be able to be satisfied.

2.1.4.3. Requirement 1: The Raspberry Pi with the radio bonnet should be inside

2.1.4.4. Description & Priority

To provide more protection for the components, and create a more stable internet connection, this device will be inside. There are no sensors within this device so it does not need to be exposed to the weather.

2.1.5. Usability Requirements

2.1.5.1. Requirement 1: Concerning metrics to be made clear

2.1.5.2. Description & Priority

I will need to display the metrics that may reveal threats a hiker is in through a noticeable alert. A push notification may not be very effective in notifying people of an emergency, additional audible alerts might need to be integrated also.

2.2. Design & Architecture

In the current design of Emanate, it is hard to determine the functions and data structures that will be used. The prototype is being developed in Python with the use of the CircuitPython and Blinka libraries, which allow for software to hardware communication.

2.2.1 Software

The software that I'm working with will vary across multiple platforms. The software may vary slightly in the future to adapt to unprecedented problems in the project. Below is the software expected to be used:

Thonny

A Python IDE is being used to develop the Python files for the Raspberry Pi hardware.

Terminal in Raspbian

This is a Linux terminal on the Raspberry Pi that is being used to install libraries and configure the hardware components.

Firebase

This software is a Google platform that will handle all the database related needs for the sensor metrics.

Android Studio

This is an IDE for Google's Android operating system. The Android application will be developed with this software and will be connected to Firebase.

Git Bash

The Linux shell will likely be used on my Windows machines for the additional installation of libraries for Kotlin.

2.2.2 Operating Systems

Raspbian

These will be the microcomputers for the hardware. There is a possibility of Arduino microcomputers replacing the Raspberry Pi microcomputers for the nodes.

Windows

Development is being done on two Windows machines to design Raspberry Pi's, while also being the future development environment for Firebase and the Android application.

Android

The Android application will likely be deployed for use on Android smartphones.

2.2.3 Languages

Python

Language for coding on the microcontrollers.

Kotlin

Language for coding the Android application

2.2.4 Key Libraries

CircuitPython

Blinka

Adafruit component libraries

Gson

Firebase

2.2.5 Raspberry Pi Components

Two Raspberry Pis

RFM69HCW Radio Breakout

RFM69HCW Radio Bonnet

BME688 Sensor

Breadboard

Jumper Cables

Improvised Wire Antenna

Copper Coil Antennae

MicroUSB Raspberry Pi Power Cables

Two resistors

One red LED

One blue LED

2.3. Implementation

There are multiple sections of the project where work has been implemented. This includes the hardware, the Firebase database, and the Android application. I will explain how they all operate in the screenshots below. Code is commented throughout, which should help bring a further understanding.

There is one file that controls each hardware component. Inside these files, there are Blinka and font file dependencies. Below are pictures of the layout of the files and some code snippets.

2.3.1 File layout (Node)

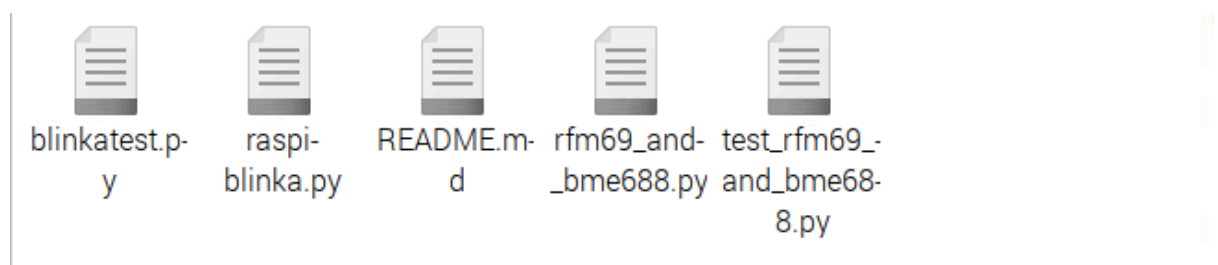


Figure 2.3.1.1: File Layout(Node)

Figure 2.3.1.1 is all the files that are needed to operate the node components. The Blinka files are needed for each of the files to work, providing libraries to the components like board and busio. There is a test with the two components as well as the file that controls all the hardware that is attached to the Raspberry Pi#

2.3.2 File Layout (Master Radio)

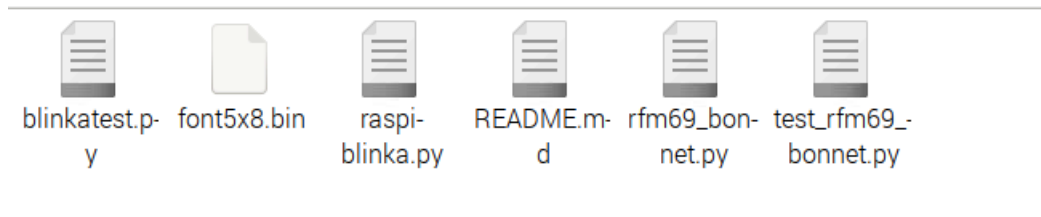


Figure 2.3.2.1: File Layout(Master Radio)

Figure 2.3.2.1 is the files that are needed to operate the master radio on the Raspberry Pi. Note that the font file is for the OLED to display the text in a desired format. There is the test, Blinka files and the main file which interacts with the hardware

2.3.3 rfm69_and_bme688.py - Main Check

```
if __name__ == "__main__":  
    # Defining GPIO pins  
    spi = busio.SPI(board.SCK, MOSI = board.MOSI, MISO = board.MISO)  
    cs = digitalio.DigitalInOut(board.CE1)  
    reset = digitalio.DigitalInOut(board.D25)  
  
    # Configure the packet radio (Frequency 433MHz)  
    rfm69 = adafruit_rfm69.RFM69(spi, cs, reset, 433.0)  
  
    # Configuring the BME688 sensor  
    i2c = board.I2C()  
    sensor = adafruit_bme680.Adafruit_BME680_I2C(i2c)  
  
    # Reciever packet  
    packet = rfm69.receive(timeout = 3.0)  
  
    run()
```

Figure 2.3.3.1: Main Check

Figure 2.3.3.1 is a section of the code some defining of the variables as well as setting up the sensor.

2.3.4 rfm69_and_bme688.py - bytes_data

```
rfm69_and_bme688.py % test_rfm69_and_bme688.py %  
1 import RPi.GPIO as GPIO  
2 from time import sleep  
3 import board  
4 import busio  
5 import digitalio  
6 import adafruit_rfm69  
7 import board  
8 import adafruit_bme680  
9  
10  
11 # Bytes to be sent in the packet to the master radio  
12 def bytes_data(temp_values, pressure_values, gas_values, humidity_values, node_id):  
13     data = bytes(f'{float(temp_values):.0f},{float(pressure_values):.0f},{float(gas_values):.0f},{float(humidity_values):.0f},{node_id}', "utf-8")  
14     return data
```

Figure 2.3.4.1: rfm69_and_bme688

```
".format(float(gas_values)), "{:.1f}".format(float(humidity_values)), {(node_id)}', "utf-8")
```

Figure 2.3.4.2: rfm69_and_bme688 extended

Both figures above show a function take in six parameters and puts them into a ByteArray to be sent over the radio frequency of 433MHz.

2.3.5 rfm69_and_bme688.py - send_packet

```
16 # Sending the packet with a series of blue LED calls signaling that the packet is sent outside of the code
17 def send_packet(rfm69br_data):
18     rfm69.send(rfm69br_data)
19     GPIO.setup(12,GPIO.OUT)
20     print("Packet sent")
21     GPIO.output(12,GPIO.HIGH)
22     sleep(0.2)
23     GPIO.output(12,GPIO.LOW)
```

Figure 2.3.5.1: rfm69_and_bme688.py – send_packet

Figure 2.3.5.1 shows a function take in the ByteArray given and sends it over the radio frequency, blinking the blue LED as it does so.

2.3.6 rfm69_and_bme688.py - receiver_check

```
25 # Ensuring that master radio recieved the packet from the node. Brings bidirectional data transfer to project
26 def receiver_check(packet, node_id):
27     if packet is not None:
28         packet_text = str(packet, 'utf-8')
29
30         # This is an additional step checking that the packet is the one sent from this node with a red LED
31         if packet_text == "Packet received from node " + node_id:
32             GPIO.setup(13,GPIO.OUT)
33             print("Master radio received packet")
34             GPIO.output(13,GPIO.HIGH)
35             sleep(0.2)
36             GPIO.output(13,GPIO.LOW)
37             print('Received: {}'.format(packet_text))
38         else:
39             print('No packet received!')
```

Figure 2.3.6.1: rfm69_and_bme688.py – receiver_check

Figure 2.3.6.1 shows a function taking in the packet variable which is received from the master radio. It blinks the red LED if the packet that was received is the node id of the packet and device. If that is not the case, the packet will not be received. This is to bring a wider connection between the nodes and the master radio

Here is the start of the code for the breakout. We have defined the GPIO pins and brought in the sensor file to read the environmental variables.

2.3.7 rfm69_and_bme688.py - run

```
rfm69_and_bme688.py test_rfm69_and_bme688.py
40     print('No packet received!')
41
42 def run():
43     # Will continuously send and receive packets
44     while True:
45
46         # Defining environmental metrics from BME688 sensor (These are defined in the loop so metrics will update)
47         temp_values = '{}'.format(sensor.temperature)
48         gas_values = '{}'.format(sensor.gas)
49         humidity_values = '{}'.format(sensor.humidity)
50         pressure_values = '{}'.format(sensor.pressure)
51         humidity_values = '{}'.format(sensor.humidity)
52         node_id = 'xEm-4'
53
54         rfm69br_data = bytes_data(temp_values, pressure_values, gas_values, humidity_values, node_id)
55
56         print(rfm69br_data)
57
58         send_packet(rfm69br_data)
59
60         # Reciever packet
61         packet = rfm69.receive(timeout = 3.0)
62
63         receiver_check(packet, node_id)
64
65         # There is a sleep of 10 seconds at the end of the program for the sensors to give more accurate readings
66         sleep(10)
```

Figure 2.3.7.1: rfm69_and_bme688.py - run

Figure 2.3.7.1 is the run function where all the code is executed. This will happen constantly while the code is being run, unless an error is found, which will cause the code to stop.

2.3.8 rfm69_and_bme688.py - Running program

```
b'23,1012,6552,47.1,xEm-4'
Packet sent
Master radio received packet
Received: Packet received from node xEm-4
```

Figure 2.3.8.1: rfm69_and_bme688.py - Running program

Figure 2.3.8.1 is the output of running the program.

2.3.9 rfm69_bonnet.py - Main Check

```
test_rfm69_bonnet.py x rfm69_bonnet.py x
149 if __name__ == "__main__":
150
151     # Variable for testing purposes
152     isTest = False
153
154     print("Running")
155
156     config = {
157         "apiKey": [REDACTED],
158         "authDomain": [REDACTED],
159         "databaseURL": [REDACTED],
160         "storageBucket": [REDACTED]
161     }
162
163     # Reference to database service
164     firebase = pyrebase.initialize_app(config)
165
166     # Reference to authentication service
167     auth = firebase.auth()
168
169     # Credentials
170     email = "[REDACTED]"
171     password = "[REDACTED]"
172
173     # Applying credentials to Firebase
174     user = auth.sign_in_with_email_and_password(email, password)
175
176     db = firebase.database()
177
178     # Current Locale of Expedition. Must be in format "City,Country code"
179     locale = "Dublin,IE"
```

Figure 2.3.9.1: rfm69_bonnet – Main Check

```
180
181     # Open Weather Map API key
182     open_weather_map_api_key = [REDACTED]
183
184     # Creating the I2C interface
185     i2c = busio.I2C(board.SCL, board.SDA)
186
187     # OLED display
188     reset_pin = DigitalInOut(board.D4)
189     display = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c, reset=reset_pin)
190
191     # Clearing the display.
192     display.fill(0)
193     display.show()
194     width = display.width
195     height = display.height
196
197     # Configure the packet radio (Frequency 433MHz)
198     CS = DigitalInOut(board.CE1)
199     RESET = DigitalInOut(board.D25)
200     spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
201     rfm69 = adafruit_rfm69.RFM69(spi, CS, RESET, 433.0)
202     prev_packet = None
203
204     run()
```

Figure 2.3.9.2: rfm69_bonnet – Main Check extended

Both figures show the main function which has all of variables needed for when the program is being run. The blacked-out values can be found in the Emanate Credentials and API keys.txt file.

2.3.10 rfm69_bonnet.py - weather_api_data

```
import time
import datetime
import busio
from digitalio import DigitalInOut, Direction, Pull
import board
import adafruit_ssd1306
import adafruit_rfm69
import requests
import pyrebase

def weather_api_data(locale, open_weather_map_api_key, isTest):
    weather_response = ''

    if isTest:
        weather_response = 'locale=Ontario,CA,"pressure":1020'

    else:
        # Open Weather Map API data
        response = requests.get("https://api.openweathermap.org/data/2.5/weather?q=" + locale + "&appid=" + open_weather_map_api_key)
        weather_response = response.text

    # Putting the response into a list, where the first item is text before pressure and the second item is after pressure
    weather_list = weather_response.split("pressure:")

    # Removing first item from list
    weather_list.pop(0)

    # Converting list to string
    weather_text_string = ''.join(weather_list)
```

Figure 2.3.10.1: rfm69_bonnet.py - weather_api_data

```
! + "&appid=" + open_weather_map_api_key)
```

```
31 # Converting list to string
32 weather_text_string = ''.join(weather_list)
33
34 # Dividing string by comma, which converts the string to list
35 text_split = weather_text_string.split(",", 1)
36
37 # Printing the pressure, the first item in the list
38 sea_level_pressure = text_split[0]
39
40 # Returning sea level pressure for use in altitude calculation
41 return sea_level_pressure
```

Figure 2.3.10.2: rfm69_bonnet.py - weather_api_data extended

Both figures above show a function take in the Open Weather Map API key and performs a request on the locale that is given, with a test check that is explained in testing. The sea level pressure is used later in another function.

2.3.11 rfm69_bonnet.py - pressure_to_altitude

```
44 # Using this formula: 44,300 * [1-(local_pressure/sea_level_pressure)^(1/5.255)] altitude is received from pressure
45 def pressure_to_altitude(local_pressure, sea_level_pressure):
46
47     # Dividing the sea level pressure
48     pressure_divided = int(local_pressure) / int(sea_level_pressure)
49
50     # Bringing in power
51     power = 1 / 5.255
52
53     # Setting pressure to the power
54     pressure_to_the_power = pressure_divided**power
55
56     # Minus pressure from one
57     minus_from_one = 1 - pressure_to_the_power
58
59     # Multiplying by constant
60     altitude = 44330 * minus_from_one
61
62     altitude = round(altitude, 0)
63
64     # Converting altitude int to string
65     altitude = str(altitude)
66
67     # Returning altitude to use in database
68     return altitude
69
```

Figure 2.3.1.1: rfm69_bonnet.py - pressure_to_altitude

Figure 2.3.1.1 Shows conversion of the sea level pressure and local pressure into a variable that is altitude. This follows the formula I found online for getting altitude from pressure.

2.3.12 rfm69_bonnet.py - token_refresh

```
71 def token_refresh(data: dict, user, node_id, epoch_time):
72
73     # The token expires every hour, it will fail to write. Once refreshed it will work again
74     try:
75         # sasgPRBqdwQdgr9jCXIF82qFYjA2 is the account id and a cell, which is used for Firebase authentication
76         db.child("BaseCamps").child("sasgPRBqdwQdgr9jCXIF82qFYjA2").child(node_id).child(epoch_time).set(data, user['idToken'])
77     except:
78         user = auth.refresh(user['refreshToken'])
79         db.child("BaseCamps").child("sasgPRBqdwQdgr9jCXIF82qFYjA2").child(node_id).child(epoch_time).set(data, user['idToken'])
80
```

Figure 2.3.12.1: rfm69_bonnet.py - token_refresh

```
user['idToken'])
```

```
user['idToken'])
```

Figure 2.3.12.2: rfm69_bonnet.py - token_refresh extended

Both figures demonstrate that token_refresh is needed so Firebase can verify the user's token every hour. This is called if the token is false.

2.3.13 rfm69_bonnet.py - run

```

82 def run():
83     while True:
84         packet = None
85         # draw a box to clear the image
86         display.fill(0)
87         display.text('Emanate Radio', 35, 0, 1)
88
89         # check for packet rx (receive)
90         packet = rfm69.receive()
91         if packet is None:
92             display.show()
93             display.text('Waiting for packet..', 1, 20, 1)
94         else:
95             # Display the packet text and rssi
96             display.fill(0)
97             prev_packet = packet
98             packet_text = str(prev_packet, "utf-8")
99             packet_array = packet_text.split(",")
100
101             # Packet Data
102             temperature = packet_array[0]
103             local_pressure = packet_array[1]
104             gas = packet_array[2]
105             humidity = packet_array[3]
106             node_id = packet_array[4]
107
108             sea_level_pressure = weather_api_data(locale, open_weather_map_api_key, isTest)
109
110             altitude = pressure_to_altitude(local_pressure, sea_level_pressure)
111
112             packet_received = bytes("Packet received from node " + node_id, "utf-8")

```

Figure 2.3.13.1:rfm69_bonnet.py - run

```

112     packet_received = bytes("Packet received from node " + node_id, "utf-8")
113     print(packet_received)
114     rfm69.send(packet_received)
115
116     print("_____")
117     print("Data is being sent to Firebase with the master radio")
118     print("_____")
119
120     # This is the data that is being written to Firebase
121     data = {
122         "locale": locale,
123         "altitude": altitude,
124         "temperature": temperature,
125         "localPressure": local_pressure,
126         "gas": gas,
127         "humidity": humidity,
128         "nodeId": node_id
129     }
130
131     # This Will be the time of the packet event. It is a timestamp to be set as a cell in the database
132     epoch_time = int(time.time())
133     date_time = datetime.datetime.fromtimestamp(epoch_time)
134
135     # These are metrics that I think are important from the sensor, it may be subjective
136     important_metrics = altitude + " " + temperature + " " + humidity
137
138     token_refresh(data, user, node_id, epoch_time)
139
140     # Display the important metrics on the master radio display when they are recieved
141     display.text("NODE ID " + node_id + " METRICS", 0, 0, 1)
142     display.text(important_metrics, 1, 20, 1)

```

Figure 2.3.13.2:rfm69_bonnet.py – run extended part 1

```

140     # Display the important metrics on the master radio display when they are recieved
141     display.text("NODE ID " + node_id + " METRICS", 0, 0, 1)
142     display.text(important_metrics, 1, 20, 1)
143     print(important_metrics)
144     time.sleep(0.5)
145
146     display.show()
147     time.sleep(0.1)

```

Figure 2.3.13.3:rfm69_bonnet.py – run extended part 2

Three of the figures above are showing the run function being used to perform many actions across the hardware and Firebase. It constantly draws on the OLED screen with 'Emanate Radio' and changes depending on the data that it receives. It manipulates this data and writes to Firebase with the data inside the dictionary that it creates.

2.3.14 Firebase - Realtime Database

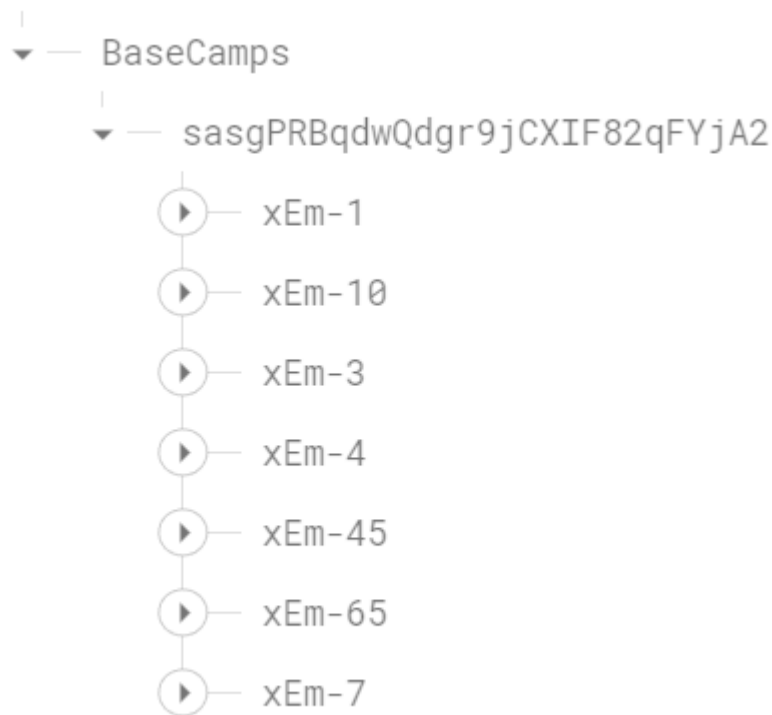


Figure 2.3.14.1: Firebase - Realtime Database Overview

Figure 2.3.14.1 is the key structure for the database. In my design I have the second cell the “admin user id” inside the “BaseCamps” cell. The idea behind the project is that an admin user id will be assigned to more “base camps” as the project grows, but for now there is only one base camp.

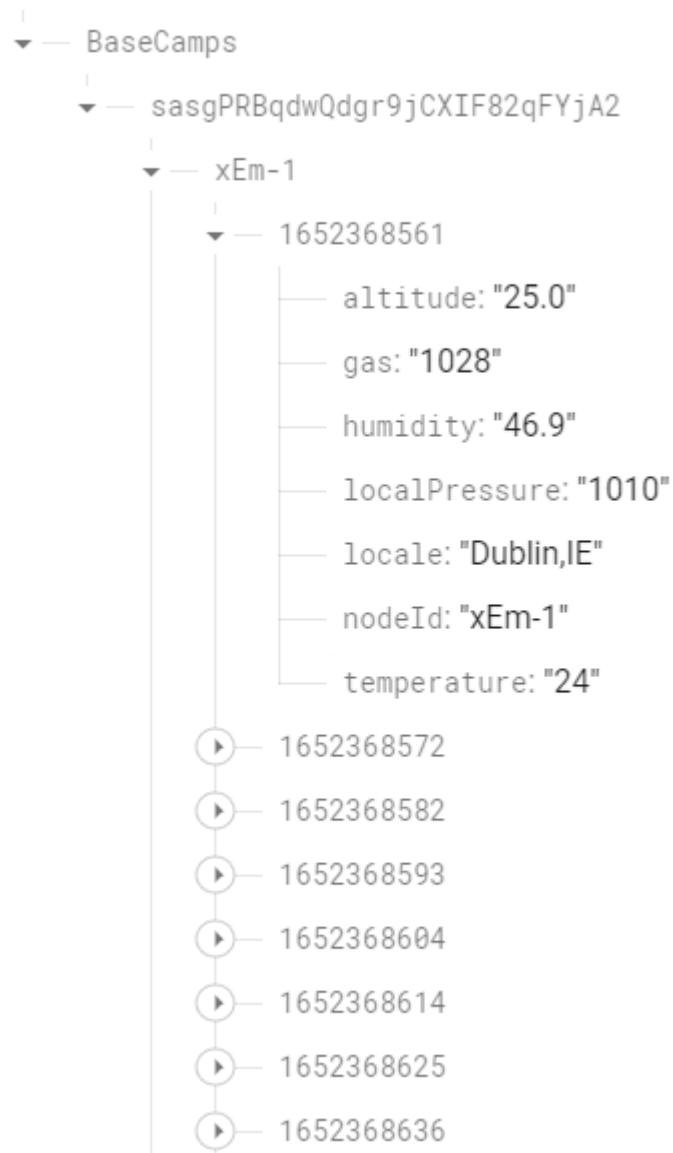


Figure 2.3.14.2: Firebase - Realtime Database Record

Figure 2.3.14.2 Shows inside of the cells is the timestamp of the event and multiple metrics called one “Record”. This is made clearer inside the application.

The next section of the project is the Android application. I will explain the key parts of the activities, but I will address the layout of the application inside the GUI section of this report.

2.3.15 Android Application - Emanate.kt (Application Class)

```
1 package com.example.emanate
2
3 import ...
4
5
6
7 // This is the application class which is run when the application is created
8 class Emanate : Application() {
9     override fun onCreate() {
10         super.onCreate()
11
12         myWorkManager()
13     }
14
15     /*
16      * When the app is created this is the function used to display a notification to the user every
17      * 15 minutes
18      */
19     private fun myWorkManager(){
20         val constraints = Constraints.Builder()
21             .setRequiresCharging(false)
22             .setRequiredNetworkType(NetworkType.NOT_REQUIRED)
23             .setRequiresCharging(false)
24             .setRequiresBatteryNotLow(false)
25             .build()
26
27         val myRequest = PeriodicWorkRequest.Builder(
28             MyWorker::class.java,
29             repeatInterval: 15,
30             TimeUnit.MINUTES
31         ).setConstraints(constraints)
32             .build()
33
34         /*
35          * The minimum interval is 15 minutes. Even though it may be limiting in an emergency this
36          * can be seen as a good aspect, as it prevents the app from otherwise draining the battery
37          */
38
39         WorkManager.getInstance(context: this)
40             .enqueueUniquePeriodicWork(
41                 uniqueWorkName: "15_minute_notification",
42                 ExistingPeriodicWorkPolicy.KEEP,
43                 myRequest
44             )
45     }
46 }
```

Figure 2.3.15.1:Android Application - Emanate.kt (Application Class)

Figure 2.3.15.1 is the application file. It is run when the application is created on the device. This includes a function which uses a WorkManager to notify the user when they have a concerning metric every 15 minutes. It is limiting considering this is an emergency application, but this is a limiting implemented by Android.

2.3.16 Android Application - SignInActivity.kt

```
1 package com.example.emanate
2
3 import ...
4
5 class SignInActivity : AppCompatActivity() {
6
7     companion object {
8         private const val TAG = "SignInActivity"
9     }
10
11     private lateinit var binding: ActivitySignInBinding
12     private lateinit var firebaseAuth: FirebaseAuth
13     private var startTime: Long = -1
14
15     override fun onCreate(savedInstanceState: Bundle?) {
16         AppCompatActivity.setDefaultNightMode(AppCompatActivity.MODE_NIGHT_NO)
17         super.onCreate(savedInstanceState)
18
19         startTime = System.currentTimeMillis()
20
21         /*
22          * When the phone opens with the splash screen, it depends on the first frame of the root
23          * activity for the splash to end. For Emanate this is near instant. To bring more
24          * attention to the splash screen, I added 3 seconds on top of the first frame to load.
25          * This may be seen to worsen the performance but is better for demonstration purposes.
26          *
27          * NOTE THAT SPLASH SCREENS IN THIS IMPLEMENTATION WILL ONLY WORK ON ANDROID 12 (API 31)
28          * BECAUSE FORMER VERSIONS NEEDED USER IMPLEMENTATION WITHOUT GOOGLE'S OWN FUNCTIONS
29          */
30         val content: View = findViewById(android.R.id.content)
31         content.viewTreeObserver.addOnPreDrawListener(
32             object : ViewTreeObserver.OnPreDrawListener {
33                 override fun onPreDraw(): Boolean {
34                     val isReady = System.currentTimeMillis() - startTime > 1000
35                     // Check if the screen is ready
36                     return if (isReady) {
37                         // The content is ready, start drawing.
38                         content.viewTreeObserver.removeOnPreDrawListener(this)
39                         true
40                     } else {
41                         // The content is not ready, wait.
42                         false
43                     }
44                 }
45             }
46         )
47     }
48 }
```

Figure 2.3.16.1: Android Application - SignInActivity.kt

```
57 binding = ActivitySignInBinding.inflate(layoutInflater)
58 setContentView(binding.root)
59
60 firebaseAuth = FirebaseAuth.getInstance()
61
62 binding.notRegistered.setOnClickListener { //TextView
63     val intent = Intent(packageContext, SignUpActivity::class.java)
64     finish()
65     startActivity(intent)
66 }
67
68 binding.signInButton.setOnClickListener { //TextView
69     val email = binding.emailTyped.text.toString()
70     val password = binding.passwordTyped.text.toString()
71
72     if (email.isNotEmpty() && password.isNotEmpty()) {
73         firebaseAuth.signInWithEmailAndPassword(email, password)
74             .addOnCompleteListener { //Task<AuthResult>
75                 if (it.isSuccessful) {
76                     val intent = Intent(packageContext, MainActivity::class.java)
77                     intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)
78                     startActivity(intent)
79                     finish()
80                 } else {
81                     Toast.makeText(
82                         context,
83                         it.exception.toString(),
84                         Toast.LENGTH_SHORT
85                     ).show()
86                 }
87             }
88     }
89 } else {
90     Toast.makeText(
91         context,
92         "Please enter a valid email and password",
93         Toast.LENGTH_SHORT
94     ).show()
95 }
```

Figure 2.3.16.2: Android Application - SignInActivity.kt extended part 1

```

91         context: this,
92         text: "Fields must not be empty",
93         Toast.LENGTH_SHORT
94     )
95     .show()
96 }
97 }
98 }
99
100 /*
101  This function performs an authentication check for when the user is at the sign in page.
102  Since the root of the application is the sign in activity, it is useful for when the user
103  is still signed in. They will be directed to the main activity if they are which avoids the
104  unnecessary sign in again.
105  */
106 override fun onStart() {
107     super.onStart()
108
109     if(firebaseAuth.currentUser != null){
110         val intent = Intent( packageContext: this, MainActivity::class.java)
111         intent.addFlags( flags: Intent.FLAG_ACTIVITY_CLEAR_TOP or Intent.FLAG_ACTIVITY_CLEAR_TASK)
112         startActivity(intent)
113         finish()
114         Log.d(TAG, msg: "Redirected to main activity because user is authenticated")
115     }
116 }
117 }

```

Figure 2.3.16.3: *Android Application - SignInActivity.kt extended part 2*

The figures above show the activity performing a few different actions around the application. It is the beginning of the application and has a splash screen if the device is Android 12. This is the only implementation of a splash screen because Google has changed how they present them on Android

The activity has a check for sign in with email and password. This authentication is handled by Firebase.

At the end of the activity there is a `onStart` function. This checks every time if the user is signed in when they open the application. If they are, they do not have to sign in again. Otherwise, they will need to sign in.

2.3.17 Android Application - SignUpActivity.kt

```
1 package com.example.emanate
2
3 import ...
4
5
6
7
8
9
10
11 class SignUpActivity : AppCompatActivity() {
12
13     private lateinit var binding: ActivitySignUpBinding
14     private lateinit var firebaseAuth: FirebaseAuth
15
16     override fun onCreate(savedInstanceState: Bundle?) {
17         AppCompatActivity.setDefaultNightMode(AppCompatActivity.MODE_NIGHT_NO)
18         super.onCreate(savedInstanceState)
19
20         binding = ActivitySignUpBinding.inflate(layoutInflater)
21         setContentView(binding.root)
22
23         firebaseAuth = FirebaseAuth.getInstance()
24
25         // With this listener the user can click if they already have an account
26         binding.alreadyRegistered.setOnClickListener { it: View!
27             val intent = Intent( packageContext: this, SignInActivity::class.java)
28             finish()
29             startActivity(intent)
30         }
31
32         // This will perform the action inside the onClickListener when the submit button is pressed
33         binding.signUpButton.setOnClickListener { it: View!
34             val email = binding.emailCreateTyped.text.toString()
35             val password = binding.passwordCreateTyped.text.toString()
36             val confirmPassword = binding.passwordConfirmationTyped.text.toString()
37
38             /*
39              All of the conditions inside this statement is for verifying the credentials entered
40              with the users on Firebase
41             */
42             if (email.isNotEmpty() && password.isNotEmpty() && confirmPassword.isNotEmpty()) {
43                 if (password == confirmPassword) {
44                     firebaseAuth.createUserWithEmailAndPassword(email, password)
45                         .addOnCompleteListener { it: Task<AuthResult!>
46                             if (it.isSuccessful) {
47                                 val intent = Intent( packageContext: this, SignInActivity::class.java)
48                                 startActivity(intent)
49                             } else {
50                                 Toast.makeText(
51                                     context: this
```

Figure 2.3.17.1: Android Application - SignUpActivity.kt

```

52         it.exception.toString(),
53         Toast.LENGTH_SHORT
54     )
55     .show()
56     }
57 }
58 } else {
59     Toast.makeText(
60         context: this,
61         text: "Passwords do not match",
62         Toast.LENGTH_SHORT
63     )
64     .show()
65 }
66 } else {
67     Toast.makeText(
68         context: this,
69         text: "Fields must not be empty",
70         Toast.LENGTH_SHORT
71     )
72     .show()
73 }
74 }
75 }
76 }

```

Figure 2.3.17.2: *Android Application - SignUpActivity.kt extended*

Both activities above show that the sign-up activity is for if the user does not have an account and wants to create one. It is very similar to the SignInActivity but with two passwords to enter. They will have their account saved in Firebase and will be able to use it when they go back to the sign in activity

2.3.18 Android Application - MainActivity.kt

```
1 package com.example.emanate
2
3 import ...
4
19
20 class MainActivity : AppCompatActivity() {
21
22     companion object {
23         private const val TAG = "MainActivity"
24     }
25
26     // Here binding is being declared. This makes the code cleaner, removing the need for
27     // findViewById(), reducing boilerplate code. It is also safer by providing null and type safety
28     private lateinit var binding: ActivityMainBinding
29     private val gson = Gson()
30     private lateinit var firebaseAuth: FirebaseAuth
31     private lateinit var menu: Menu
32
33     override fun onCreate(savedInstanceState: Bundle?) {
34         AppCompatActivity.setDefaultNightMode(AppCompatActivity.MODE_NIGHT_NO)
35         super.onCreate(savedInstanceState)
36         setContentView(R.layout.activity_main)
37
38         actionBar?.setHomeButtonEnabled(true)
39         actionBar?.setDisplayHomeAsUpEnabled(true)
40
41         binding = ActivityMainBinding.inflate(layoutInflater)
42         setContentView(binding.root)
43
44         setSupportActionBar(binding.toolbar)
45
46         firebaseAuth = FirebaseAuth.getInstance()
47
48         binding.submitButton.setOnClickListener { it: View!
49             val nodeId: String = binding.editTextInsertId.text.toString()
50             if (nodeId.isNotEmpty()) {
51
52                 readNodeId(nodeId)
53
54                 hideKeyboard(view: currentFocus ?: View(context: this))
55
56             } else {
57                 Toast.makeText(
58                     context: this, text: "Please enter an id in the format xEm-*number(s)*",
59                     Toast.LENGTH_SHORT
60                 ).show()
61             }
62         }
```

Figure 2.3.18.1: Android Application - MainActivity.kt


```

62     }
63 }
64
65 override fun onOptionsItemSelected(item: MenuItem): Boolean {
66     when (item.itemId) {
67         R.id.account_button -> {
68             FirebaseAuth.signOut()
69
70             val intent1 = Intent(applicationContext, SignInActivity::class.java)
71             startActivity(intent1)
72             finish()
73         }
74         R.id.notifications_button -> {
75             val intent1 = Intent(applicationContext, NotificationsActivity::class.java)
76             startActivity(intent1)
77         }
78     }
79     return true
80 }
81
82 // This function is to hide the keyboard for when the user submits a metric to search
83 private fun hideKeyboard(view: View) {
84     val inputMethodManager = getSystemService(Activity.INPUT_METHOD_SERVICE)
85     as InputMethodManager
86     inputMethodManager.hideSoftInputFromWindow(view.windowToken, 0)
87 }
88
89 override fun onCreateOptionsMenu(menu: Menu): Boolean {
90     menuInflater.inflate(R.menu.action_bar, menu)
91     this.menu = menu
92     return true
93 }
94
95 private fun readNodeId(nodeId: String) {
96     val database = FirebaseDatabase
97     val databaseReference = database.reference.child("BaseCamps").child("sasP8BqdeQdgr9jCXIF82qFyA2")
98     //If the node id entered is available in the database, this success listener will retrieve
99     // the data snapshot
100
101     databaseReference.child(nodeId).get().addOnSuccessListener { it: DataSnapshot!
102
103         if (it.exists()) {
104             val value = it.children.last().value as Map<*, *>
105             val json = gson.toJson(value)
106             val record = gson.fromJson(json, Record::class.java)

```

Figure 2.3.18.2: Android Application - MainActivity.kt part 1

```

107         Log.d(TAG, record.toString())
108         Log.d(TAG, it.value.toString())
109
110         Toast.makeText(context: this, text: "Node found", Toast.LENGTH_SHORT).show()
111
112         val altitudeWithUnits = record.altitude + "m"
113         val gasWithUnits = record.gas + "ohms"
114         val humidityWithUnits = record.humidity + "%"
115         val localPressureWithUnits = record.localPressure + "hPa"
116         val temperatureWithUnits = record.temperature + "°C"
117
118         binding.editTextInsertId.text.clear()
119         binding.textViewNodeId.text = record.dbNodeId
120         binding.textViewAltitude.text = altitudeWithUnits
121         binding.textViewGas.text = gasWithUnits
122         binding.textViewHumidity.text = humidityWithUnits
123         binding.textViewLocalPressure.text = localPressureWithUnits
124         binding.textViewLocale.text = record.locale
125         binding.textViewTemperature.text = temperatureWithUnits
126     } else {
127         Toast.makeText(context: this, text: "Node not found in expedition", Toast.LENGTH_SHORT)
128             .show()
129     }
130 }.addOnFailureListener { it: Exception
131     Toast.makeText(context: this, text: "Error! Try again?", Toast.LENGTH_SHORT).show()
132 }
133 }
134 }

```

Figure 2.3.18.3: Android Application - MainActivity.kt part 2

All of the figures above show the main activity of the whole application. Much of the onCreate function is set up for UI elements, but also has an onClickListener which is used to take in input from the user when they are looking for a node.

As we move further down the Kotlin file there are some aesthetic implementations like hiding the keyboard on search and introducing an action bar to the top of the application.

When the user searches for a node, a function is called which looks at the Firebase database, searches through the database for values, puts them inside a map, and puts the map against the Record.kt class. This data class uses Gson to manipulate the map with variables inside the data class. From this, the MainActivity assigns the record values to parts of the UI, with error handling if not achievable.

2.3.19 Android Application - NotificationsActivity.kt

```
1 package com.example.emanate
2
3 import ...
4
20
21 class NotificationsActivity : AppCompatActivity() {
22
23     companion object {
24         private const val TAG = "NotificationsActivity"
25     }
26
27     // Here binding is being declared. This makes the code cleaner, removing the need for
28     // findViewById(), reducing boilerplate code. It is also safer by providing null and type safety
29     private lateinit var binding: ActivityNotificationsBinding
30     private lateinit var firebaseAuth: FirebaseAuth
31     private val gson = Gson()
32     private lateinit var adapter: ListAdapter
33     private var altitudeSicknessFlag = false
34
35
36     override fun onCreate(savedInstanceState: Bundle?) {
37         AppCompatActivity.setDefaultNightMode(AppCompatActivity.MODE_NIGHT_NO)
38         super.onCreate(savedInstanceState)
39         setContentView(R.layout.activity_notifications)
40
41         actionBar?.setHomeButtonEnabled(true)
42         actionBar?.setDisplayHomeAsUpEnabled(true)
43
44         binding = ActivityNotificationsBinding.inflate(layoutInflater)
45         setContentView(binding.root)
46
47         firebaseAuth = FirebaseAuth.getInstance()
48
49         isAuthenticated()
50
51         adapter = ListAdapter(context = this)
52
53         binding.nodesListView.adapter = adapter
54
55         concerningMetrics()
56
57     }
```

Figure 2.3.19.1: Android Application - NotificationsActivity.kt

```

59 private fun concerningMetrics() {
60     val database = Firebase.database
61     val databaseReference = database.reference.child( pathString: "BaseCamps").child(
62         pathString: "sasgPR8gdwQdgr9jCXIF82qFYjA2"
63     )
64
65     val valueEventListener = object : ValueEventListener {
66
67         // The function will be run if data changes in the database, so it will be used a lot
68         override fun onDataChange(dataSnapshot: DataSnapshot) {
69
70             for (nodeSnapshot in dataSnapshot.children) {
71                 val name = nodeSnapshot.key
72                 Log.d(TAG, name.toString())
73
74                 /*
75                  This variable takes in the 'event' of each of the nodes throughout the
76                  database. The event is the recorded time with a record
77                 */
78                 val eventList = nodeSnapshot.children.toList()
79
80                 /*
81                  Transformation from list of type DataSnapshot to Record for the metrics
82                  inside each record
83                 */
84                 val recordList: List<Record> = eventList
85                     .subList(eventList.size - 2, eventList.size)
86                     .map { it: DataSnapshot!
87                         val value = it.value as Map<*, *>
88                         val json = gson.toJson(value)
89                         val record = gson.fromJson(json, Record::class.java)
90                         Log.d(TAG, record.toString())
91                         val twoNodesRecords = it.value.toString()
92                         val timeNode = it.key.toString()
93                         Log.d(TAG, record.toString())
94                         Log.d(TAG, twoNodesRecords)
95                         Log.d(TAG, timeNode)
96
97                     }
98                 return@map record
99             }
100         }
101     }
102 }

```

Figure 2.3.19.2: Android Application - NotificationsActivity.kt extended part 1

```

198 // Creating instance of TimeConverter to access getDateTime
199 val converter = TimeConverter()
200
201 /*
202  * Using eventList this variable gets the last event of the nodes. It then
203  * selects the key from the event(timestamp), passing it to the getDateTime
204  * function. The null operator will check if the value is null as we move
205  * through the declaration. If it fails somehow, the elvis operator will throw
206  * an error
207  */
208 val timestampConvertedToTime = eventList.lastOrNull()?.key?.let(converter::getDateTime)
209   ?: error("Timestamp was null")
210
211 // Destructuring for the bottom two records
212 val (past, current) = recordList[0] to recordList[1]
213
214 // This StringBuilder will be added to the adapter for the notifications
215 val stringBuilder = StringBuilder()
216
217 /*
218  * For all of the notifications on concerning metrics, they are passed into a
219  * dataBuilder to be referenced in the public companion object. Since value event
220  * listener will only take in an object and does not provide a return value.
221  */
222
223 if (current.altitude.toDouble() - past.altitude.toDouble() <= -10) {
224     stringBuilder.append(
225         "The altitude dropped over 30 metres " +
226         "suddenly.\n"
227     )
228 }
229
230 /*
231  * The additional flag in this check will be so when the hiker goes above 2500
232  * metres they won't be notified if they have passed that altitude again
233  */
234 if (current.altitude.toDouble() >= 2500) {
235     if (!altitudeSicknessFlag) {
236         stringBuilder.append("Risk of altitude sickness.\n")
237     }
238     altitudeSicknessFlag = true

```

Figure 2.3.19.3: Android Application - NotificationsActivity.kt extended part 2

```

131      /*
132       * The additional flag in this check will be so when the hiker goes above 2500
133       * metres they won't be notified if they have passed that altitude again
134       */
135      if (current.altitude.toDouble() >= 2500) {
136          if (!altitudeSicknessFlag) {
137              stringBuilder.append("Risk of altitude sickness.\n")
138          }
139          altitudeSicknessFlag = true
140      }
141
142      if (current.altitude.toDouble() <= 2500 && altitudeSicknessFlag) {
143          altitudeSicknessFlag = false
144      }
145
146      if (current.temperature.toInt() - past.temperature.toInt() >= 3 &&
147          current.humidity.toDouble() - past.humidity.toDouble() <= -3.0
148      ) {
149          stringBuilder.append("Suspected fire in local area.\n")
150      }
151
152      Log.d(TAG, "msg: " + "Printing timeList = " + eventList[0].key.toString())
153
154
155      Log.d("tag: " + "Jumble", "msg: " + "$timestampConvertedToTime ${current.dbNodeId} ${stringBuilder.toString()}")
156
157      /*
158       * This will check if any concerning metrics were added to the stringBuilder.
159       * If not, it won't be added to the notifications.
160       */
161
162      if (stringBuilder.isNotEmpty()) {
163          //Setting the current node id
164          adapter.nodeAlerts = adapter.nodeAlerts.plus(current.dbNodeId)
165          adapter.timeOfNotificationTrigger.add(timestampConvertedToTime)
166          WorkerMetrics.concerningNodes.add(current.dbNodeId)
167          adapter.description.add(stringBuilder.toString())
168          adapter.notifyDataSetChanged()
169      } else {
170          WorkerMetrics.concerningNodes.remove(current.dbNodeId)
171      }

```

Figure 2.3.19.4: Android Application - NotificationsActivity.kt extended part 3

```

170      }
171  }
172  }
173
174      override fun onCancelled(databaseError: DatabaseError) {
175          Log.d(TAG, databaseError.message)
176      }
177  }
178      databaseReference.addListenerForSingleValueEvent(valueEventListener)
179  }
180
181      // Additional security check if the user gets past the sign in screen
182      private fun isAuthenticated(){
183          if(firebaseAuth.currentUser == null){
184              val intent = Intent(packageContext: this, SignInActivity::class.java)
185              intent.addFlags(flags: Intent.FLAG_ACTIVITY_CLEAR_TOP or Intent.FLAG_ACTIVITY_CLEAR_TASK)
186              startActivity(intent)
187              Log.d(TAG, "msg: " + "Redirected to sign in activity because user is not authenticated")
188          }
189      }
190  }

```

Figure 2.3.19.5: Android Application - NotificationsActivity.kt extended part 3

The figures above show that the Notifications Activity performs the most amount of work for application. It takes in the node data after some configurations are done for Firebase, with some added security to check if the user is still authenticated. This is unlikely to happen, but further prevents a risk.

The file works with a “DataSnapshot”, which is used to break the cells further down in the database. It uses the Record data class in the same way that the MainActivity does with the database, then brings the node id and converted timestamp into the ListAdapter.kt where it is projected onto the UI. This is also done with a StringBuilder but for the description of currently concerning node metrics. The concerningNodes mutable list variable in the WorkerMetrics is brought in to add the concerning metric, which is then passed into MyWorker.kt for the notification information.

2.3.20 Android Application - ListAdapter.kt

```
1 package com.example.emanate
2
3 import ...
4
5 class ListAdapter(
6     private val context: Context,
7     var nodeAlerts: List<String> = ArrayList(),
8     val description: MutableList<String> = ArrayList(),
9     val timeOfNotificationTrigger: MutableList<String> = ArrayList(),
10 ) : BaseAdapter() {
11
12     // Returns total number of items in the list
13     override fun getCount(): Int {
14         return description.size
15     }
16
17     // Returns list item at the position it's in
18     override fun getItem(position: Int): Any {
19         return position
20     }
21
22     // Returns the item id
23     override fun getItemId(position: Int): Long {
24         return position.toLong()
25     }
26
27     override fun getView(position: Int, convertView: View?, parent: ViewGroup?): View {
28         var view: View? = convertView
29         val viewHolder: ViewHolder
30
31         // Inflating the layout for each list row
32         if (view == null) {
33             viewHolder = ViewHolder()
34             val inflater = context.getSystemService(Context.LAYOUT_INFLATER_SERVICE)
35                 as LayoutInflater
36             view = inflater.inflate(R.layout.list_item, parent, attachToRoot = false)
37
38             //Inserting the current item into the textView of each row
39             viewHolder.nodeAlerts = view.findViewById(R.id.node_alert)
40             viewHolder.description = view.findViewById(R.id.description)
41             viewHolder.timeOfNotificationTrigger = view
42                 .findViewById(R.id.time_of_notification_trigger)
43         }
44         return view
45     }
46 }
```

Figure 2.3.20.1: Android Application - ListAdapter.kt


```

46         viewHolder.timeOfNotificationTrigger = view
47         .findViewById(
48             R.id.time_of_notification_trigger
49         )
50         view!!.tag = viewHolder
51     } else {
52         viewHolder = view.tag as ViewHolder
53     }
54
55     // Getting current item to be displayed
56     viewHolder.nodeAlerts?.text = nodeAlerts[position]
57     viewHolder.description?.text = description[position]
58     viewHolder.timeOfNotificationTrigger?.text = timeOfNotificationTrigger[position]
59
60     // Returns view for the current row
61     return view
62 }
63
64 //ViewHolder class will hold the views for the list items
65 private inner class ViewHolder {
66     var nodeAlerts: TextView? = null
67     var description: TextView? = null
68     var timeOfNotificationTrigger: TextView? = null
69 }
70 }

```

Figure 2.3.20.2: Android Application - ListAdapter.kt extended

Both of the figures above show the file for adding list items to the list_item.xml file. The metrics that were discussed in the NotifcatonsActivity.kt file are handled here.

2.3.21 Android Application - WorkerMetrics.kt

```

1     package com.example.emanate
2
3     object WorkerMetrics {
4         val concerningNodes: MutableSet<String> = mutableSetOf()
5     }

```

Figure 2.3.21.1: Android Application - WorkerMetrics.kt

Figure 2.3.21.1 is an object that holds the concerningNodes variable. Isolating it away from activities allows for it to be accessed across the application easier while making use of object orientated programming.

2.3.22 Record.kt

```
1 package com.example.emanate
2
3 import com.google.gson.annotations.SerializedName
4
5 data class Record(
6     @SerializedName(value = "nodeId") val dbNodeId: String,
7     val altitude: String,
8     val gas: String,
9     val humidity: String,
10    val localPressure: String,
11    val locale: String,
12    val temperature: String,
13 )
```

Figure 2.3.22.1: *Record.kt*

Figure 2.3.22.1 is a file using Gson to read the data when it passed into the record class. It is used around a few activities and allows for a more efficient strategy of implementing the metrics as one “record”.

2.3.23 Android Application - MyWorker.kt

```
1 package com.example.emanate
2
3 import ...
4
15
16 class MyWorker(context: Context, workerParameters: WorkerParameters):
17 Worker(context, workerParameters){
18
19     companion object{
20         const val CHANNEL_ID = "emanate_node_warning"
21         const val NOTIFICATION_ID = 201
22         private const val TAG = "MyWorkerActivity"
23     }
24
25     override fun doWork(): Result {
26         Log.d(TAG, "msg: \"doWork: Success function called\"")
27
28         if(WorkerMetrics.concerningNodes.isEmpty()){
29             return Result.success()
30         }
31
32         showNotification()
33
34         return Result.success()
35     }
36
37     /*
38     Inside this function is the setup for the notifications across devices. Although importance
39     of the notifications are determined as high, this may not always be the case as other
40     processes in the system can overwrite the priority
41     */
42     private fun showNotification(){
43
44         val intent = Intent(applicationContext,
45             NotificationsActivity::class.java).apply { this.intent
46             flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
47         }
48
49         //A pending intent is for allowing the application to execute a predefined piece of code
50         val pendingIntent = PendingIntent.getActivity(
51             applicationContext, requestCode: 0, intent, PendingIntent.FLAG_IMMUTABLE
52         )
```

Figure 2.3.23.1: Android Application - MyWorker.kt

```

53
54     val notification = NotificationCompat.Builder(
55         applicationContext,
56         CHANNEL_ID
57     )
58     .setSmallIcon(R.drawable.emanate_logo)
59     .setContentTitle("Warning(s) for ${WorkerMetrics.concerningNodes.size} node(s)")
60     .setContentText("Check notifications in app")
61     .setPriority(NotificationCompat.PRIORITY_HIGH)
62     .setAutoCancel(true)
63     .setContentIntent(pendingIntent)
64
65     // Notification channels are needed for Android Oreo and higher
66     if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O){
67
68         val channelName = "Emanate Nodes"
69         val channelDescription = "Channel is for monitoring node metrics on expeditions"
70         val channelImportance = NotificationManager.IMPORTANCE_HIGH
71
72         val channel = NotificationChannel(CHANNEL_ID, channelName, channelImportance).apply{
73             description = channelDescription
74         }
75
76         val notificationManager = applicationContext.getSystemService(
77             Context.NOTIFICATION_SERVICE
78         ) as NotificationManager
79
80         notificationManager.createNotificationChannel(channel)
81     }
82
83     with(NotificationManagerCompat.from(applicationContext)){
84         notify(NOTIFICATION_ID, notification.build(), )
85     }
86 }
87 }

```

Figure 2.3.23.1: Android Application - MyWorker.kt extended

Both figures above show a class that is mainly for customising the notifications which pop up on a user's device. It is called in the Emanate application class which is used onCreation of the application. There is another if statement which checks the user's Android version to implement notification channels which are required in Android 8 and higher.

2.3.24 Android Application - TimeConverter.kt

```

1     package com.example.emanate
2
3     import ...
4
5
6     class TimeConverter {
7         // Conversion from timestamp to datetime
8         fun getDateTime(s: String): String? {
9             return try {
10                 val sdf = SimpleDateFormat( pattern: "H:mm:s - dd/MM/yyyy", Locale.UK)
11                 val netDate = Date( date: s.toLong() * 1000)
12                 sdf.format(netDate)
13             } catch (e: Exception) {
14                 "Unreadable time"
15             }
16         }
17     }

```

Figure 2.3.24.1: Android Application - TimeConverter.kt

Figure 2.3.24.1 is a simple Kotlin class which only has the `getDateTime` method. It separates the code to make the method be readily accessed anywhere, but also is the only method that is testable in the program.

2.4. Graphical User Interface (GUI)

Since hardware is a key component of the project, I will consider it as part of the GUI as well as the Android application.

2.4.1 RFM69 Radio Bonnet and Raspberry Pi



Figure 2.4.1.1: RFM69 Radio Bonnet and Raspberry Pi

Figure 2.4.1.1 is the radio bonnet that is attached to the main Raspberry Pi. It has an improvised antenna from a jumper cable that I cut and soldered to size for a frequency of 433MHz. It has a small RFM69HCW radio attached to it along with an OLED screen and three programmable buttons.

2.4.2 Node components (BME688 Sensor, RFM69HCW Breakout, LEDs, and resistors)

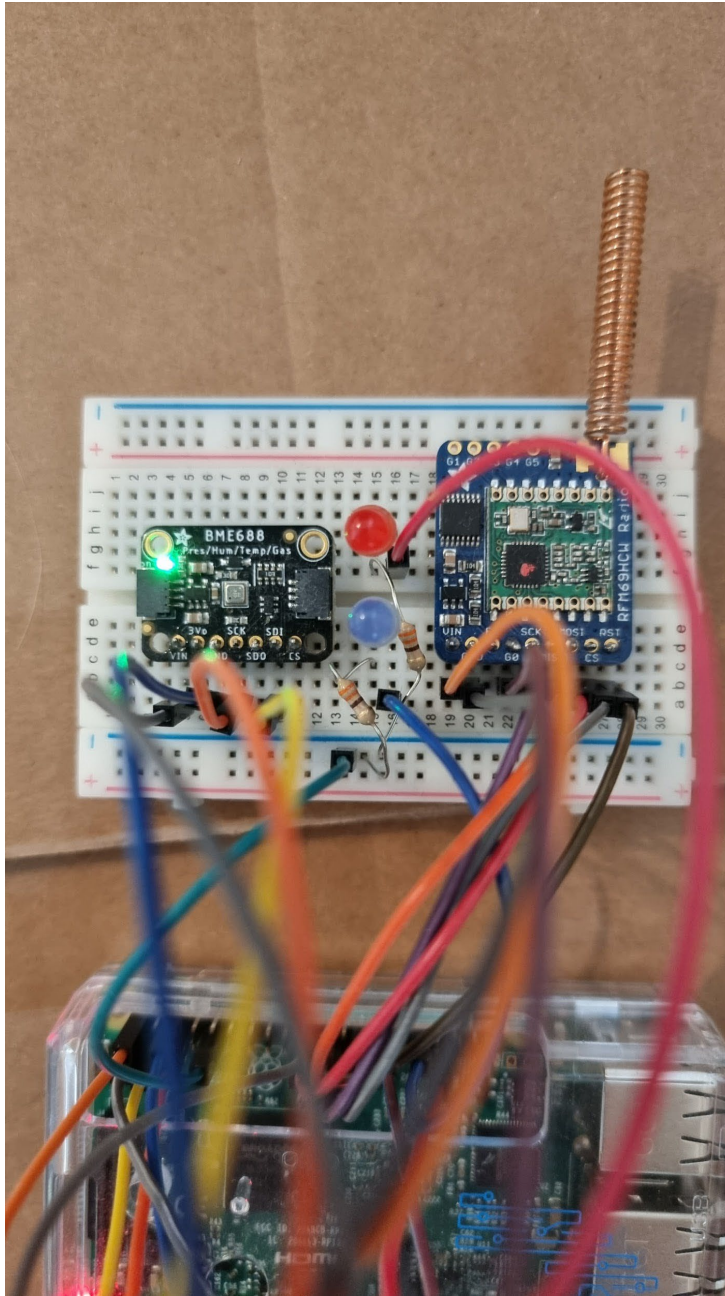


Figure 2.4.2.1: Node components (BME688 Sensor, RFM69HCW Breakout, LEDs, and resistors)

Figure 2.4.2.1 shows both components that are wired to a breadboard. They have pin headers soldered to the pin connectors to make the breadboard prototyping possible. There is a coil antenna soldered to the breakout radio. The coil measurements are tuned for a 433MHz frequency.

Additionally, the breadboard also has LEDs and resistors which are used to signal to the user if data has been sent or received.

2.4.3 Raspberry Pi



Figure 2.4.3.1: Raspberry Pi

Figure 2.4.3.1 is a Raspberry Pi that has been used for testing the functionality of the components. Apart from the wiring, the other Raspberry Pi operates in the same way. The pins are connected to the GPIO pins to work with the breakout and BME688 sensor. For programming on the microcomputer, it has been connected to VNC to allow interaction with the software wirelessly through the network.

2.4.4 Blue LED sending data

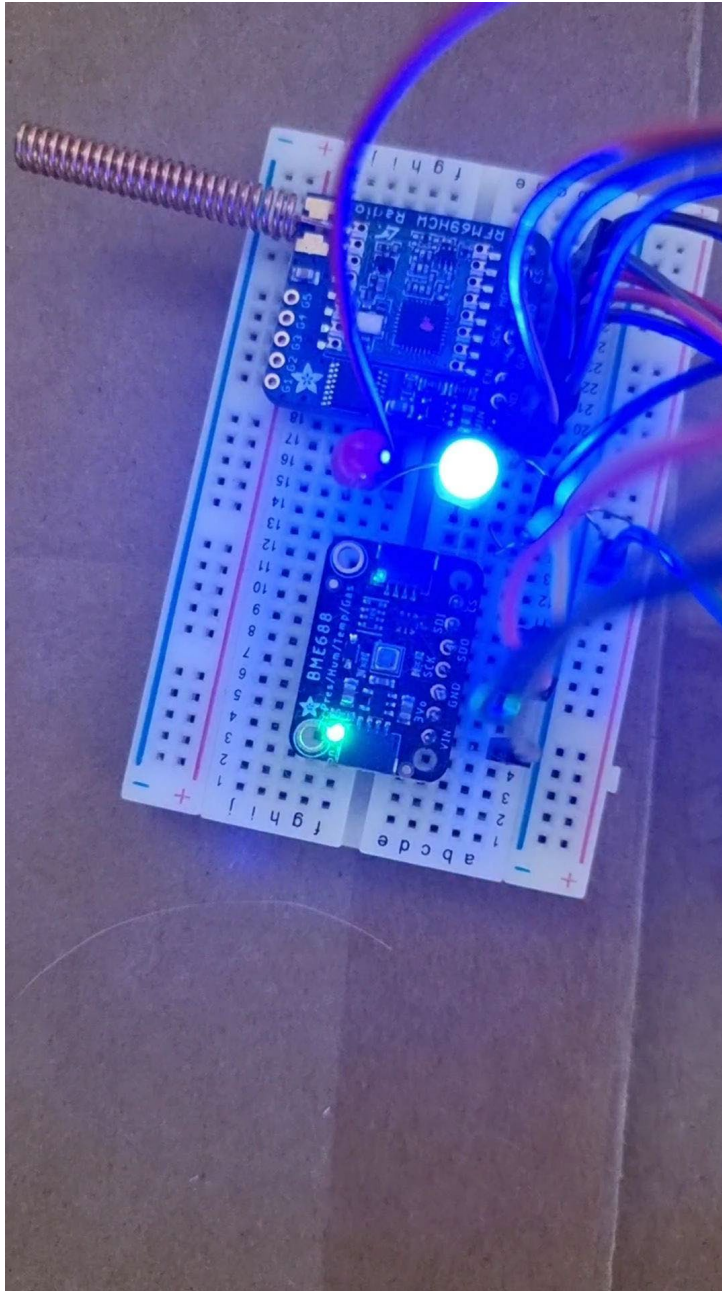


Figure 2.4.4.1: Blue LED sending data

Figure 2.4.4.1 shows the moment a packet is sent over the 433MHz frequency from the packet radio.

2.4.5 Red LED receiving data

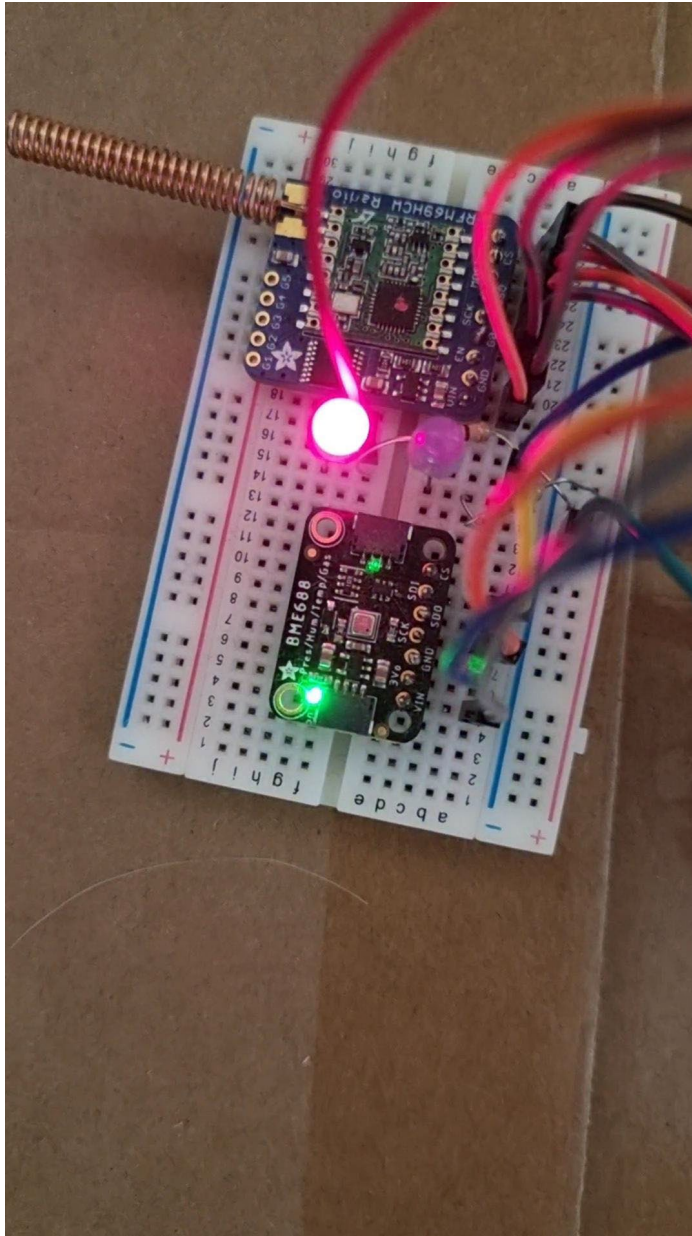


Figure 2.4.5.1: Red LED receiving data

Figure 2.4.5.1 shows confirmation from the master radio that the packet sent by the RFM69HCW breakout was received.

The connectors for the jumper cables are as follows:

Raspberry Pi	RFM69HCW
3V	VIN
Ground	Ground
SCK	SCK

MOSI	MOSI
MISO	MISO
D5	CS
D6	RST

Raspberry Pi	BME688
3V	VIN
Ground	Ground
SCL	SCK
SDA	SDI

Raspberry Pi	Blue LED
GPIO Pin 12	Blue LED foot
Ground	Ground, Resistor

Raspberry Pi	Red LED
GPIO Pin 13	Red LED foot
Ground	Ground, Resistor

The other part of the GUI is the Android application. I will be running the application on my smartphone as it has Android 12 which allows splash screens. It will demonstrate the full functionality of the application.

2.4.6 Splash screen

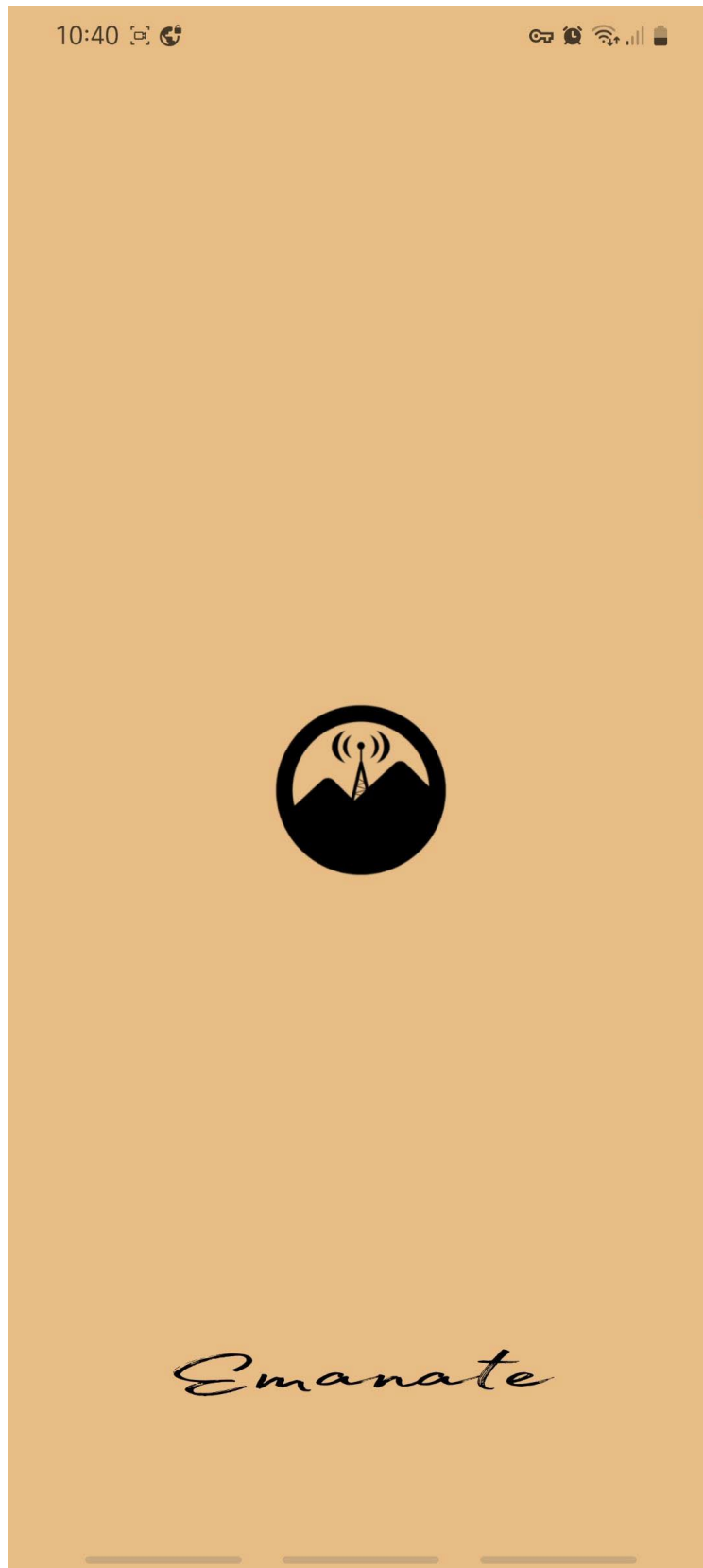
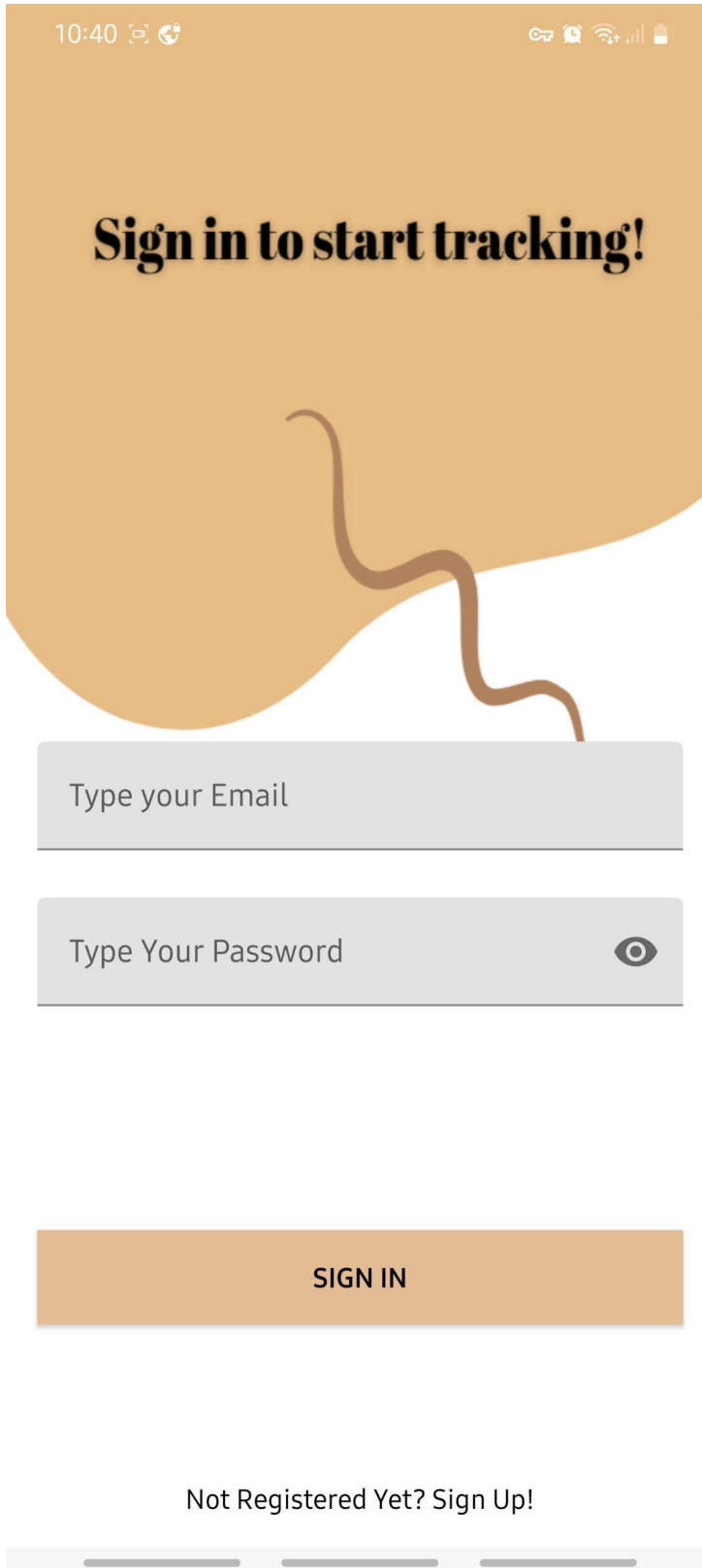


Figure 2.4.6.1: Splash screen

Figure 2.4.6.1 is the splash screen of Emanate. It has the logo and branding text at the bottom of the screen. It shows for around two seconds then brings the user to the sign in screen, it is only visible on devices with Android 12 or higher.

2.4.7 Sign in

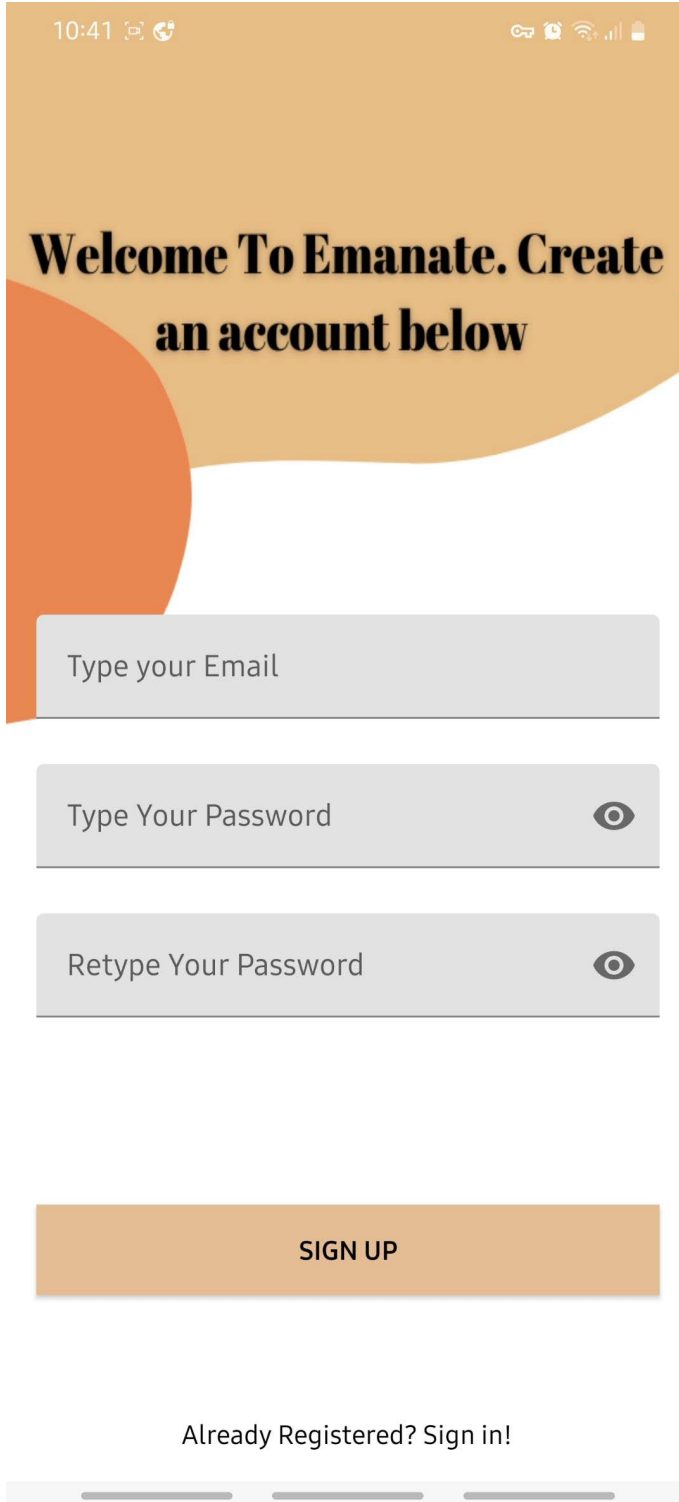


The image shows a mobile application sign-in screen. At the top, the status bar displays the time 10:40 and various icons. The main background is a solid orange color. In the center, the text "Sign in to start tracking!" is written in a bold, black, serif font. Below this text is a large, light brown, wavy line that resembles a stylized path or a drop. Underneath the line are two input fields: the first is labeled "Type your Email" and the second is labeled "Type Your Password" with an eye icon to its right. Below these fields is a large orange button with the text "SIGN IN" in black, uppercase letters. At the bottom of the screen, there is a link that says "Not Registered Yet? Sign Up!". The very bottom of the screen shows a white bar with three horizontal lines, likely representing the home indicator on an iPhone.

Figure 2.4.7.1: Sign in

Figure 2.4.7.1 is where the user can sign in. Their credentials will read from Firebase to see if they are a user. If they are not, they can access the sign-up screen to create an account.

2.4.8 Sign up



The image shows a mobile app sign-up screen. At the top, the status bar displays the time 10:41 and various icons. The main heading reads "Welcome To Emanate. Create an account below" in bold black text. Below this, there are three input fields: "Type your Email", "Type Your Password" (with an eye icon for toggling visibility), and "Retype Your Password" (also with an eye icon). A large orange button labeled "SIGN UP" is positioned below the password fields. At the bottom, there is a link that says "Already Registered? Sign in!". The screen has a light orange background with a large orange circle on the left side.

Figure 2.4.8.1: Sign up

Figure 2.4.8.1 is the sign-up screen. A user can create an account where it will be saved on Firebase. Once it is created, they can sign in with their new credentials.

2.4.9 Main screen - On open

10:41

Emanate SIGN OUT

Display Node Metrics

Here you can insert a node id of any of the nodes on expeditions. Any notifications where a hiker may be in trouble can be accessed through the bell icon above. This is where you will be directed to when the notifications are accessed. For security purposes, node ids entered below must be exactly correct!

xEm-*Numbers*

GET METRICS

Metrics

Node Id	
	Humidity:
	Locale:
	Altitude:
	Gas:
	Pressure:
	Temperature:

Figure 2.4.9.1: Main screen - On open

Figure 2.4.9.1 is the screen where the user can access most of the other activities. It is the screen that opens when the user is signed in. The bottom of the metrics is empty now because the user did not enter a node id but will fill when entered.

2.4.10 Main screen - Enter node id

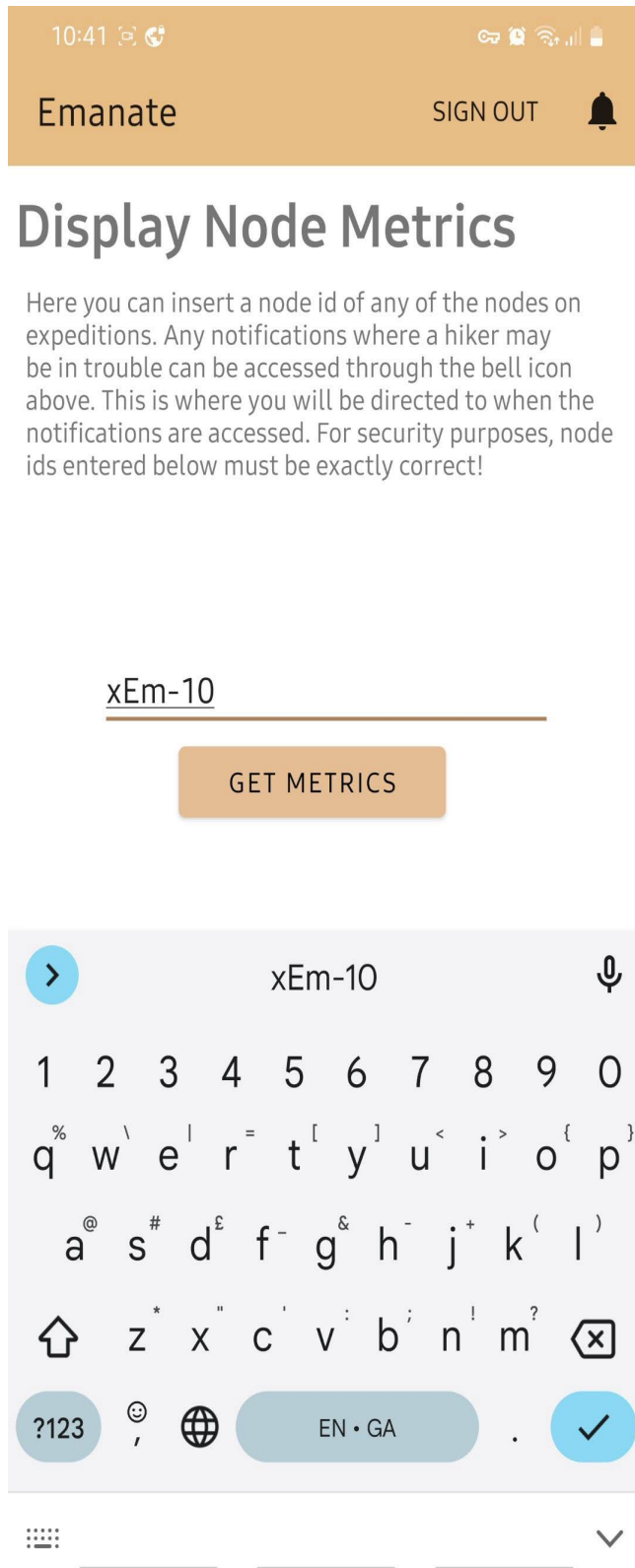


Figure 2.4.10.1: Main screen - Enter node id

Figure 2.4.10.1 shows an example node id that is being entered in the main screen. A user should know the node ids of those on expeditions. The metrics are personal and real time

which may let the user know metrics like heart rate and exact location with GPS if those sensors are attached.

2.4.11 Main screen - Toast message

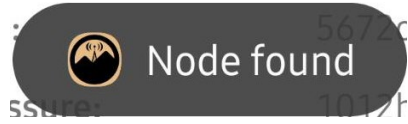


Figure 2.4.11.1: Main screen - Toast message

Figure 2.4.11.1 Shows a Toast message that pops up depending on what the user has entered. Node found means that the entered id is correct, but may also say node id not found, or give an error, prompting them to try again.

2.4.12 Main screen - Metrics

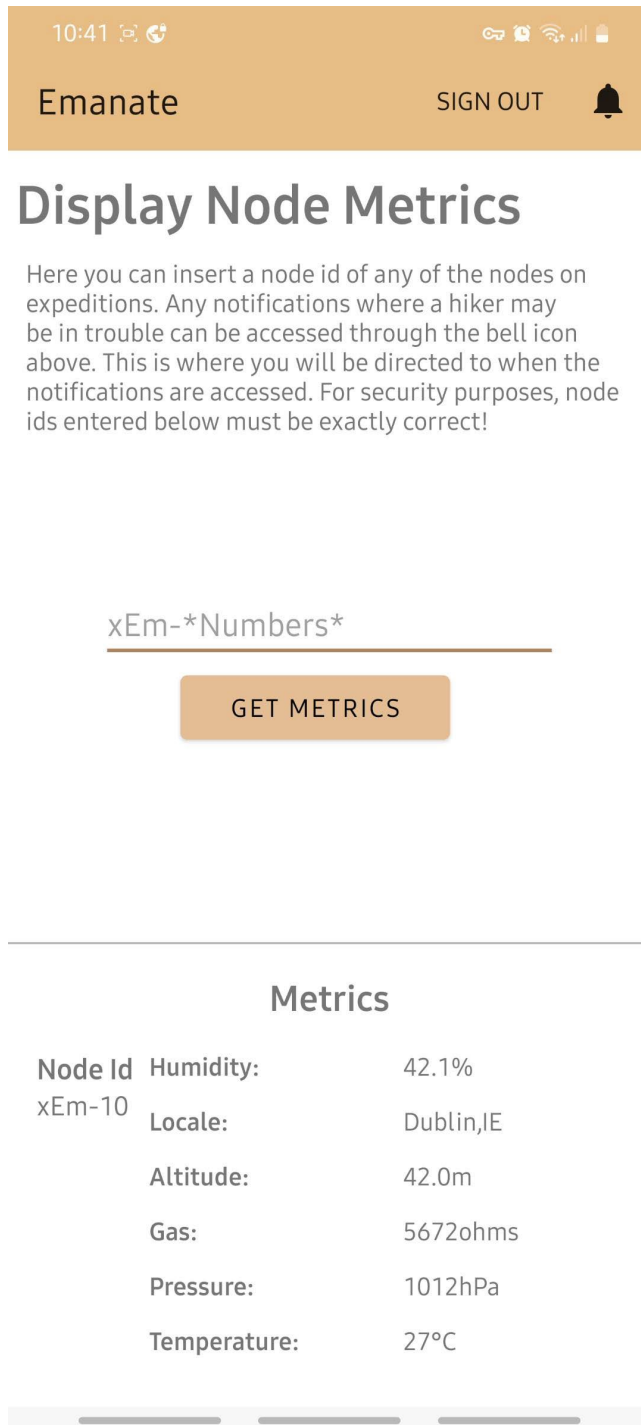


Figure 2.4.12.1: Main screen - Metrics

Figure 2.4.12.1 Below are the metrics for the node id entered. It is the most recent metrics of the user that can be examined for hiker safety.

2.4.13 Notifications screen

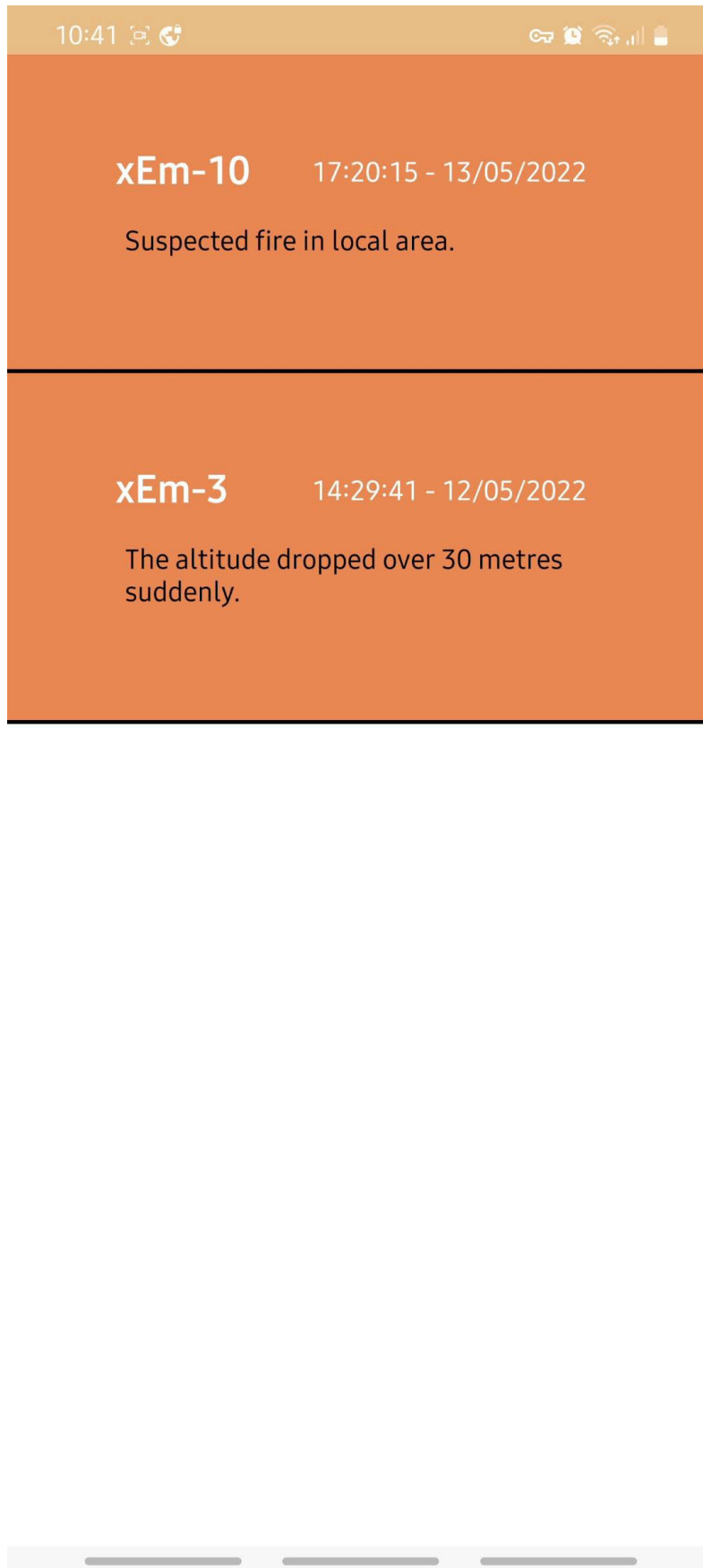


Figure 2.4.13.1: Notifications screen

Figure 2.4.13.1 is the notifications screen with two warnings for nodes on expeditions. The metrics are compared between the past and current events constantly in the code. This is the output if any of the metrics are deemed concerning with a time and date assigned to when the warning happened. New warnings would be able to be implemented with more hardware, like heart rate and wind sensors.

2.4.14 Notification - Notification bar

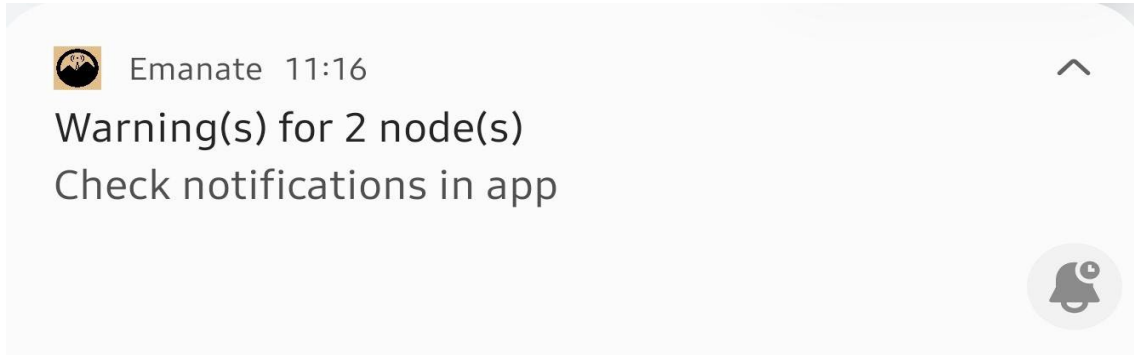


Figure 2.4.14.1: Notification - Notification bar

Figure 2.4.14.1 is the notification that pops up in the notification bar every 15 minutes. Tapping on it will bring the user straight to the notifications.

2.4.15 Application Icon



Figure 2.4.15.1: Application Icon

Figure 2.4.15.1 shows the application also has an icon for the user to identify with. The orange '1' tells the user how many notifications they have in the application. To stop the notifications a user can disable them, or force close the application.

2.5. Testing

For my testing I will be working with the physical components, the two Raspberry Pis, the Firebase database, and the Android application. I expect the Android tests to be somewhat

limiting as functions inside the Android application access the database, making replicating those functions overly complicated.

2.5.1 LED Test

For my first test I will be testing one of the LEDs on the node. Since the LED is purely visual and is not reliant on the rest of the system, I expect the outcome to only affect the visual aesthetics. The red LED has a similar purpose so this test will apply to that component as well.

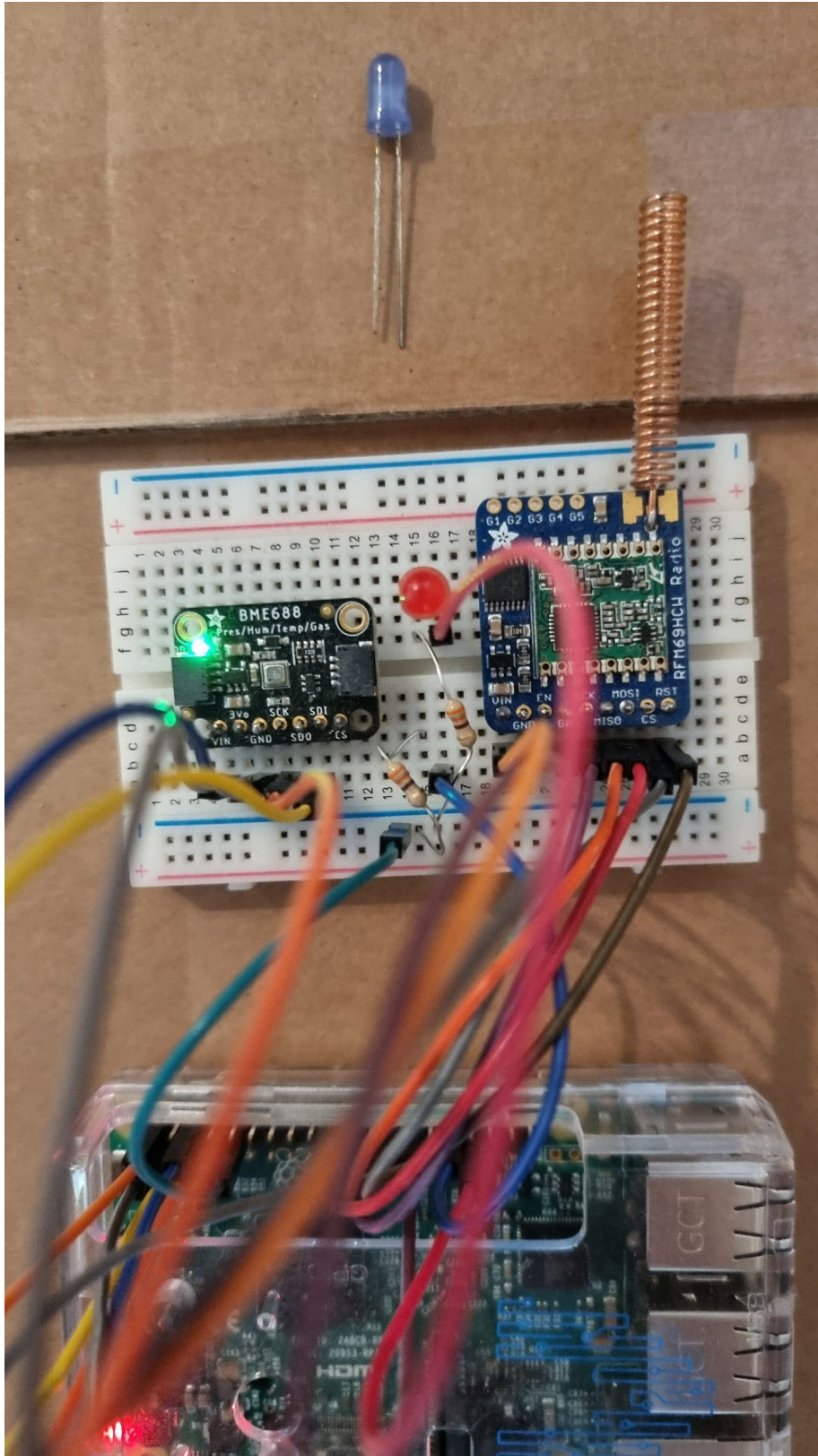


Figure 2.5.1.1: LED Test Hardware

Figure 2.5.1.1 shows that the blue LED which signals a packet has been sent is taken out of the breadboard.

```
rfm69_breakout.py *  
9 # Defining GPIO pins  
10 spi = busio.SPI(board.SCK, MOSI = board.MOSI, MISO = board.MISO)  
11 cs = digitalio.DigitalInOut(board.CE1)  
12 reset = digitalio.DigitalInOut(board.D25)  
13  
14 rfm69 = adafruit_rfm69.RFM69(spi, cs, reset, 433.0)  
15  
16 # Will continuously send and receive packets  
17 while True:  
18  
19     # Defining environmental metrics from BME688 sensor  
20     temp_values = '{}'.format(bme688.sensor.temperature)  
21     gas_values = '{}'.format(bme688.sensor.gas)  
22     humidity_values = '{}'.format(bme688.sensor.humidity)  
23     pressure_values = '{}'.format(bme688.sensor.pressure)  
24     humidity_values = '{}'.format(bme688.sensor.humidity)  
25     node_id = 'xEm-3'  
26  
27     # Bytes to be sent in the packet to the master radio  
28     rfm69br_data = bytes(f'"{:.0f}"'.format(float(temp_values)), f'"{:.0f}"'.format(float(pressure_values)), f'"{:.0f}"'.format(float(humidity_values)), f'"{:.0f}"'.format(float(gas_values)), f'"{:.0f}"'.format(float(humidity_values)), f'"{:.0f}"'.format(float(node_id)))  
29  
30     # Sending the packet with a series of blue LED calls signaling that the packet is sent outside of the code  
31     rfm69.send(rfm69br_data)  
32     GPIO.setup(12, GPIO.OUT)  
33     print("Packet sent")  
34     GPIO.output(12, GPIO.HIGH)  
35     sleep(0.2)  
36     GPIO.output(12, GPIO.LOW)  
37  
38     # Ensuring that master radio recieved the packet from the node. Brings bidirectional data transfer to project  
39     packet = rfm69.receive(timeout = 3.0)
```

```
Shell  
>>> %Run rfm69_breakout.py  
Packet sent  
No packet recieved!  
Packet sent  
No packet recieved!  
Packet sent  
No packet recieved!
```

Python 3.9.2

Figure 2.5.1.2: LED Test Software

Figure 2.5.1.2 shows the program left unaffected software wise. No packet was received because the master radio was not being used. Test has the outcome of what was expected.

2.5.2 Unwire BME688 Sensor

Now I will be unwiring the BME688 sensor. The BME688 sensor code is expected to break because the code will not be able to detect the hardware. Also, the breakout code will likely break due to it needing access to the BME688 sensor code.

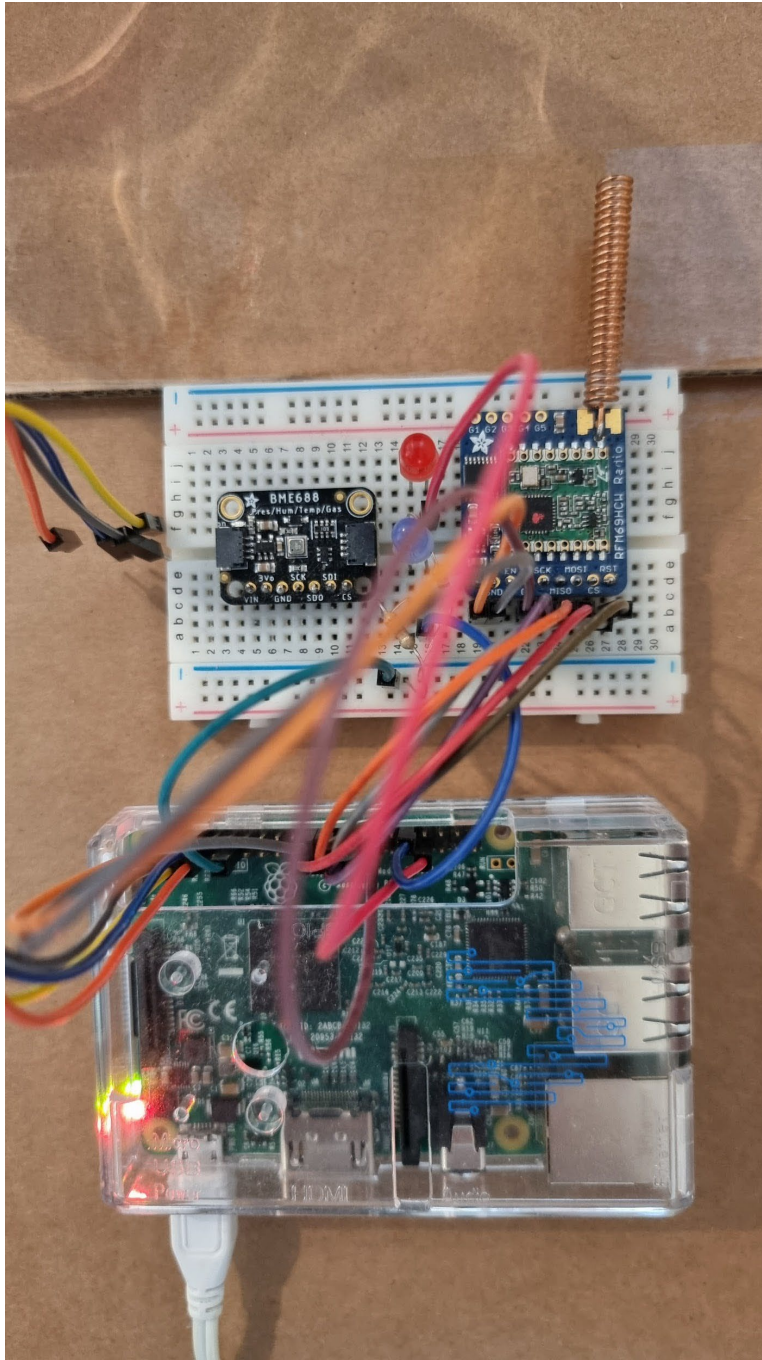


Figure 2.5.2.1: Unwire BME688 Sensor Hardware

Figure 2.5.2.1 shows that the wiring for the BME688 sensor has been taken out of the breadboard.


```

Python 3.9.2 (/usr/bin/python3)
>>> %Run rfm69_breakout.py
Traceback (most recent call last):
  File "/home/pi/.local/lib/python3.9/site-packages/adafruit_bus_device/i2c_device.py", line 154, in _probe_for_device
    self.i2c.writeto(self.device_address, b"")
  File "/usr/local/lib/python3.9/dist-packages/busio.py", line 166, in writeto
    return self.i2c.writeto(address, buffer, stop=stop)
  File "/usr/local/lib/python3.9/dist-packages/adafruit_blinka/microcontroller/generic_linux/i2c.py", line 49, in writet
0
    self._i2c_bus.write_bytes(address, buffer[start:end])
  File "/usr/local/lib/python3.9/dist-packages/Adafruit_PureIO/smbus.py", line 314, in write_bytes
    self._device.write(buf)
OSError: [Errno 121] Remote I/O error

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/home/pi/.local/lib/python3.9/site-packages/adafruit_bus_device/i2c_device.py", line 160, in _probe_for_device
    self.i2c.readfrom_into(self.device_address, result)
  File "/usr/local/lib/python3.9/dist-packages/busio.py", line 156, in readfrom_into
    return self.i2c.readfrom_into(address, buffer, stop=stop)
  File "/usr/local/lib/python3.9/dist-packages/adafruit_blinka/microcontroller/generic_linux/i2c.py", line 56, in readfr
om_into
    readin = self._i2c_bus.read_bytes(address, end - start)
  File "/usr/local/lib/python3.9/dist-packages/Adafruit_PureIO/smbus.py", line 181, in read_bytes
    return self._device.read(number)
OSError: [Errno 121] Remote I/O error

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/home/pi/rfm69_breakout.py", line 7, in <module>
    import bme688
  File "/home/pi/bme688.py", line 6, in <module>
    sensor = adafruit_bme680.Adafruit_BME680_I2C(i2c)
  File "/usr/local/lib/python3.9/dist-packages/adafruit_bme680.py", line 446, in _init
    self.i2c = i2c_device.I2CDevice(i2c, address)

```

Figure 2.5.2.2: Unwire BME688 Sensor Software part 1

```

  File "/usr/local/lib/python3.9/dist-packages/adafruit_bme680.py", line 446, in _init
    self.i2c = i2c_device.I2CDevice(i2c, address)
  File "/home/pi/.local/lib/python3.9/site-packages/adafruit_bus_device/i2c_device.py", line 50, in _init
    self._probe_for_device()
  File "/home/pi/.local/lib/python3.9/site-packages/adafruit_bus_device/i2c_device.py", line 163, in _probe_for_device
    raise ValueError("No I2C device at address: 0x%x" % self.device_address)
ValueError: No I2C device at address: 0x77
>>>

```

Figure 2.5.2.3: Unwire BME688 Sensor Software part 2

Both figures above show that as expected, an error was thrown from the rfm69_breakout.py file when it was run. All the errors explain that the connectors for the i2c are not functioning. Test had expected result.

2.5.3 RFM69HCW Unwire

For the next test I will be plugging out the RFM69HCW board. This sends packets across to the master radio and will likely only crash the code for the breakout when it is run. Since the master radio is an RFM69HCW on a bonnet instead of a breakout, this test will cover the same result if I were to test on that component as well. I expect that the code will throw a wiring error when it is run.

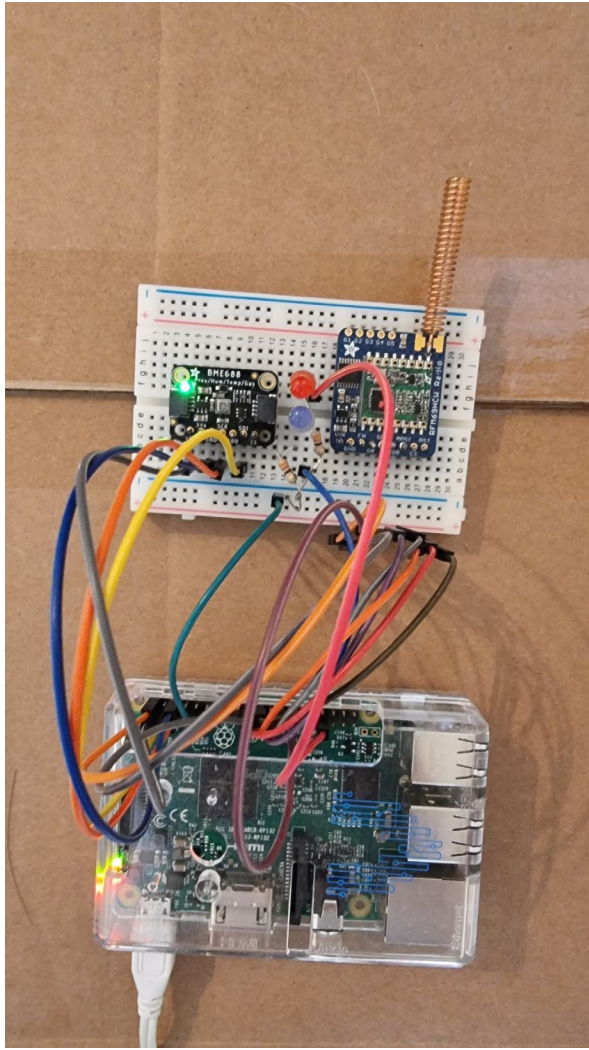


Figure 2.5.3.1: RFM69HCW Unwire Hardware

Figure 2.5.3.1 Shows RFM69HCW unwired from the Raspberry Pi

```

rfm69_breakout.py *
1  import RPi.GPIO as GPIO
2  from time import sleep
3  import board
4  import busio
5  import digitalio
6  import adafruit_rfm69
7  import bme688
8
9  # Defining GPIO pins
10 spi = busio.SPI(board.SCK, MOSI = board.MOSI, MISO = board.MISO)
11 cs = digitalio.DigitalInOut(board.CE1)
12 reset = digitalio.DigitalInOut(board.D25)
13
14 rfm69 = adafruit_rfm69.RFM69(spi, cs, reset, 433.0)
15
16 # Will continuously send and receive packets
17 while True:
18
19     # Defining environmental metrics from BME688 sensor
20     temp_values = '{}'.format(bme688.sensor.temperature)
21     gas_values = '{}'.format(bme688.sensor.gas)
22     humidity_values = '{}'.format(bme688.sensor.humidity)
23     pressure_values = '{}'.format(bme688.sensor.pressure)
24     humidity_values = '{}'.format(bme688.sensor.humidity)
25     node_id = 'xEm-3'
26
27     # Bytes to be sent in the packet to the master radio
28     rfm69br_data = bytes(f'{{:.0f}}'.format(float(temp_values)),'{{:.0f}}'.format(float(pressure_values)),'{{:.0f}}'.format(float(humidity_values)),'{{:.0f}}'.format(float(gas_values)),'{{:.0f}}'.format(float(humidity_values)),'{{:.0f}}'.format(float(node_id)))
29
30     # Sending the packet with a series of blue LED calls signaling that the packet is sent outside of the code
31     rfm69.send(rfm69br_data)
32     # Sleep for 10 seconds
33     sleep(10)
34
Shell
>>> %Run rfm69_breakout.py
Traceback (most recent call last):
  File "/home/pi/rfm69_breakout.py", line 14, in <module>
    rfm69 = adafruit_rfm69.RFM69(spi, cs, reset, 433.0)
  File "/usr/local/lib/python3.9/dist-packages/adafruit_rfm69.py", line 301, in __init__
    raise RuntimeError(
RuntimeError: Failed to find RFM69 with expected version, check wiring!
>>>

```

Figure 2.5.3.2: RFM69HCW Unwire Software

Figure 2.5.3.2 Shows that as expected, the program gives a wiring error for the code. The expected result is the same as the actual result.

2.5.4 Internet Robustness

I will now test how the system reacts to losing internet connection while it is running. I expect that the Raspberry Pi with the master radio will run into an error while it is trying to write to the database, and the node Raspberry Pi will keep running as usual.

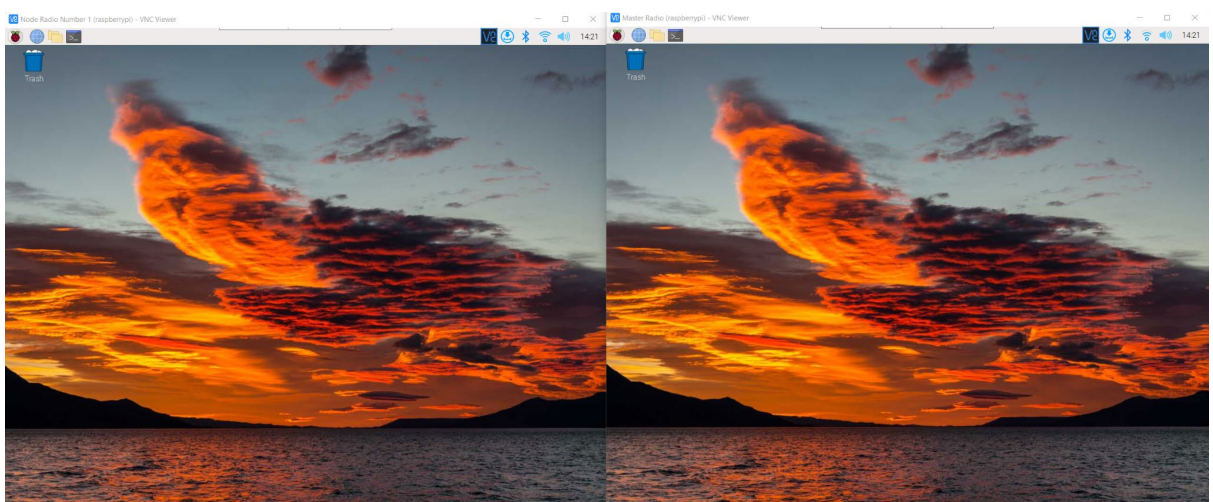


Figure 2.5.4.1: Internet Robustness Startup Screen


```

    resp = conn.urlopen(
File "/usr/local/lib/python3.9/dist-packages/requests/packages/urllib3/connectionpool.py", line 639, in urlopen
    retries = retries.increment(method, url, error=e, _pool=self,
File "/usr/local/lib/python3.9/dist-packages/requests/packages/urllib3/util/retry.py", line 287, in increment
    raise MaxRetryError(_pool, url, error or ResponseError(cause))
requests.packages.urllib3.exceptions.MaxRetryError: HTTPSConnectionPool(host='api.openweathermap.org', port=443
): Max retries exceeded with url: /data/2.5/weather?q=Dublin,IE&appid=4b1d1de8d743ff0d538d643cf0cbc850 (Caused
by NewConnectionError('<requests.packages.urllib3.connection.VerifiedHTTPSConnection object at 0x748352c8>: Fai
led to establish a new connection: [Errno -3] Temporary failure in name resolution'))

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
File "/home/pi/rfm69_bonnet.py", line 101, in <module>
    response = requests.get("https://api.openweathermap.org/data/2.5/weather?q=" + locale + "&appid=4b1d1de8d74
3ff0d538d643cf0cbc850")
File "/usr/local/lib/python3.9/dist-packages/requests/api.py", line 70, in get
    return request('get', url, params=params, **kwargs)
File "/usr/local/lib/python3.9/dist-packages/requests/api.py", line 56, in request
    return session.request(method=method, url=url, **kwargs)
File "/usr/local/lib/python3.9/dist-packages/requests/sessions.py", line 475, in request
    resp = self.send(prepare, **send_kwargs)
File "/usr/local/lib/python3.9/dist-packages/requests/sessions.py", line 596, in send
    r = adapter.send(request, **kwargs)
File "/usr/local/lib/python3.9/dist-packages/requests/adapters.py", line 487, in send
    raise ConnectionError(e, request=request)
requests.exceptions.ConnectionError: HTTPSConnectionPool(host='api.openweathermap.org', port=443): Max retries
exceeded with url: /data/2.5/weather?q=Dublin,IE&appid=4b1d1de8d743ff0d538d643cf0cbc850 (Caused by NewConnectio
nError('<requests.packages.urllib3.connection.VerifiedHTTPSConnection object at 0x748352c8>: Failed to establis
h a new connection: [Errno -3] Temporary failure in name resolution'))

>>>

```

Figure 2.5.4.4: Internet Robustness Pis Reconnected part 2

Both figures above demonstrate that after around 10 minutes or so I reconnected the Raspberry Pi to the internet. The Raspberry Pi with the master radio ran into errors as it was unable to write to the database. There is little I can do to fix this without rewriting most of the code across the project. It would be hard to account for the time lost without internet if the packets sent had built up on the master radio. The testing's expected result was the same as the actual result.

2.5.5 Node Distancing

For the last testing of the Raspberry Pi, I will be pushing the distance of the node and the master radio. To achieve this, I will attach a portable power bank (10,000 mAh) to the node and put the node in a box in my bag to carry for extra protection from rain. My phone will be monitoring the database as I walk away, once I stop seeing the database have new written data, I will stop walking and record the distance, trying this in multiple trails from the master radio.

I expect for the Raspberry Pi to work for up to 1.5km and 200 metres at the very least.



Figure 2.5.5.1: Node Distancing Powerbank Connected

Figure 2.5.5.1 Show a power bank that is plugged into the node. Note that the power bank is at 69 percent.



Figure 2.5.5.2: Node Distancing Node in Box

Figure 2.5.5.2 Shows the node being put into the box. It will be put into my bag.

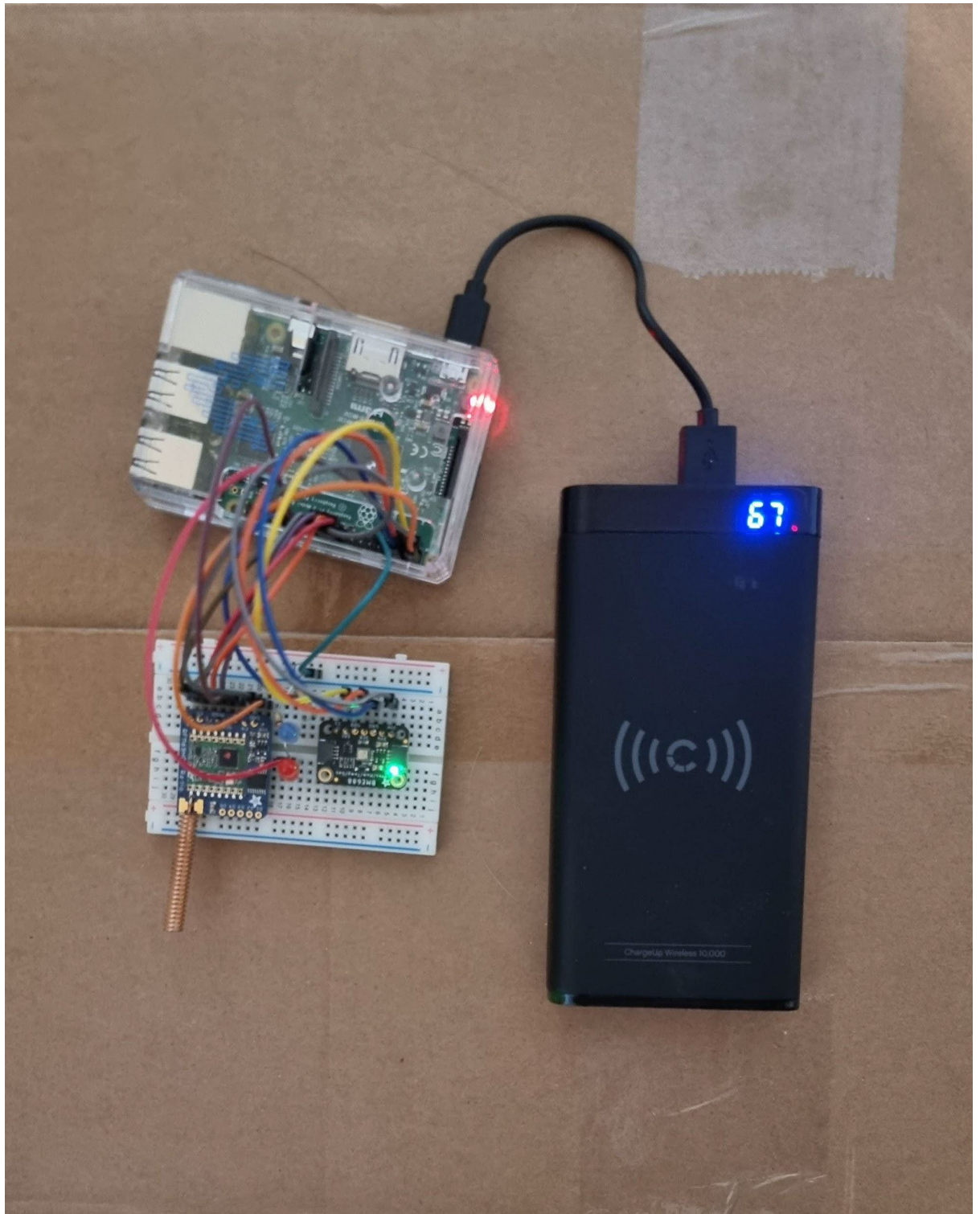


Figure 2.5.5.3: Node Distancing Node After Testing

Figure 2.5.5.3 shows a picture of the node after testing for 20 minutes, different trails from the master radio were done. The longest journey was 75 metres with a battery loss of 2 percent. The range is disappointing but there are many factors to consider:

- The master node was inside a house with many concrete walls
- Area was not well ventilated with not windows open and a front door being closed

- The antennas are small, and one is a jumper cable, affecting its range
- The radio was in a box within a bag which will affect how the radio disperses its radio waves
- Estate buildings were blocking the line of sight

The test was not expected to perform this poorly. However, Adafruit had said that this packet radio was the weakest of the two on offer, with the LoRa (Long range) radio being significantly longer in distance at maximum of over 10km.

2.5.6 bytes_data test

Moving on to the Python testing I will be testing the code as best as I can with unittest. The two files will be rfm69_and_bme688.py and the rfm_bonnet.py.

Only one test was feasible in rfm69_and_bme688.py as the rest of the functions rely on accessing the hardware.

```

rfm69_and_bme688.py test_rfm69_and_bme688.py %
1 import unittest
2 import RPi.GPIO as GPIO
3 from rfm69_and_bme688 import bytes_data
4
5 class TestRfm69AndBme688(unittest.TestCase):
6
7     def test_bytes_data(self):
8
9         # Given
10        temp_values = "24"
11        pressure_values = "1018"
12        gas_values = "516"
13        humidity_values = "50"
14        node_id = "xEm-47"
15        expected_data = b'24,1018,516,50.0,xEm-47'
16
17        # When
18        result = bytes_data(temp_values, pressure_values, gas_values, humidity_values, node_id)
19
20        # Then
21        self.assertEqual(result, expected_data)
22
23        # The rest of the functions rely on hardware to give realistic test results
24
25
26 if __name__ == '__main__':
27     unittest.main()

```

```

Shell
>>> %run test_rfm69_and_bme688.py
.
-----
Ran 1 test in 0.001s

OK

Python 3.9.2 (/usr/bin/python3)
>>>

```

Figure 2.5.6.1: bytes_data test

Figure 2.5.6.1 Shows that the test passed and would have failed if any of the values given in the variables were not in the right formatting.

2.5.7 pressure_to_altitude and test_weather_api_data

The second Python test was on the rfm_bonnet.py file. Two functions were able to be tested but the rest of the files were not feasible because they relied on sensitive data, network requests, and Firebase access.


```

rfm69_bonnet.py test_rfm69_bonnet.py *
1 import unittest
2 from rfm69_bonnet import pressure_to_altitude
3 from rfm69_bonnet import weather_api_data
4
5 # Creating a class with a case of unit tests
6 class TestRfm69Bonnet(unittest.TestCase):
7
8     def test_pressure_to_altitude(self):
9
10         # Given
11         local_pressure = "1028"
12         sea_level_pressure = "1032"
13         expected_altitude = "33.0"
14
15         # When
16         result = pressure_to_altitude(local_pressure, sea_level_pressure)
17
18         # Then
19         self.assertEqual(result, expected_altitude)
20

```

Figure 2.5.7.1: *pressure_to_altitude and test_weather_api_data*

```

21 def test_weather_api_data(self):
22
23     # Given
24     locale = "Ontario,CA"
25     open_weather_map_api_key = "adsa883n1ks2dk"
26     isTest = True
27     expected_sea_level_pressure = '1020'
28
29     # When
30     result = weather_api_data(locale, open_weather_map_api_key, isTest)
31
32     # Then
33     self.assertEqual(result, expected_sea_level_pressure)
34
35     # token_refresh, and run functions are not testable due to exposing sensitive data
36
37 if __name__ == '__main__':
38     unittest.main()

```

```

>>> %Run test_rfm69_bonnet.py
..
-----
Ran 2 tests in 0.001s
OK

```

Figure 2.5.7.2: *pressure_to_altitude and test_weather_api_data extended*

The figures above show that the tests passed. The pressure to altitude function would have failed if either local pressure or sea level pressure had been given values in the wrong formatting. The other function was altered to work with a check if it was a test or not. This is not the best way to test the function as the data is false but does give an idea as to how it may function when it is properly being used. I would have failed in the real function if the API key was wrong, and the locale was not given.

2.5.8 Database Rules

For Firebase there are rules for the database that I set up. These rules prevent the user from writing to the database when they are not allowed to. They must be an admin with the id in the inner cell of the basecamps and must be logged in to read the database.



Figure 2.5.8.1: Database Rules in Firebase

Figure 2.5.8.1 Shows the rules of the database.

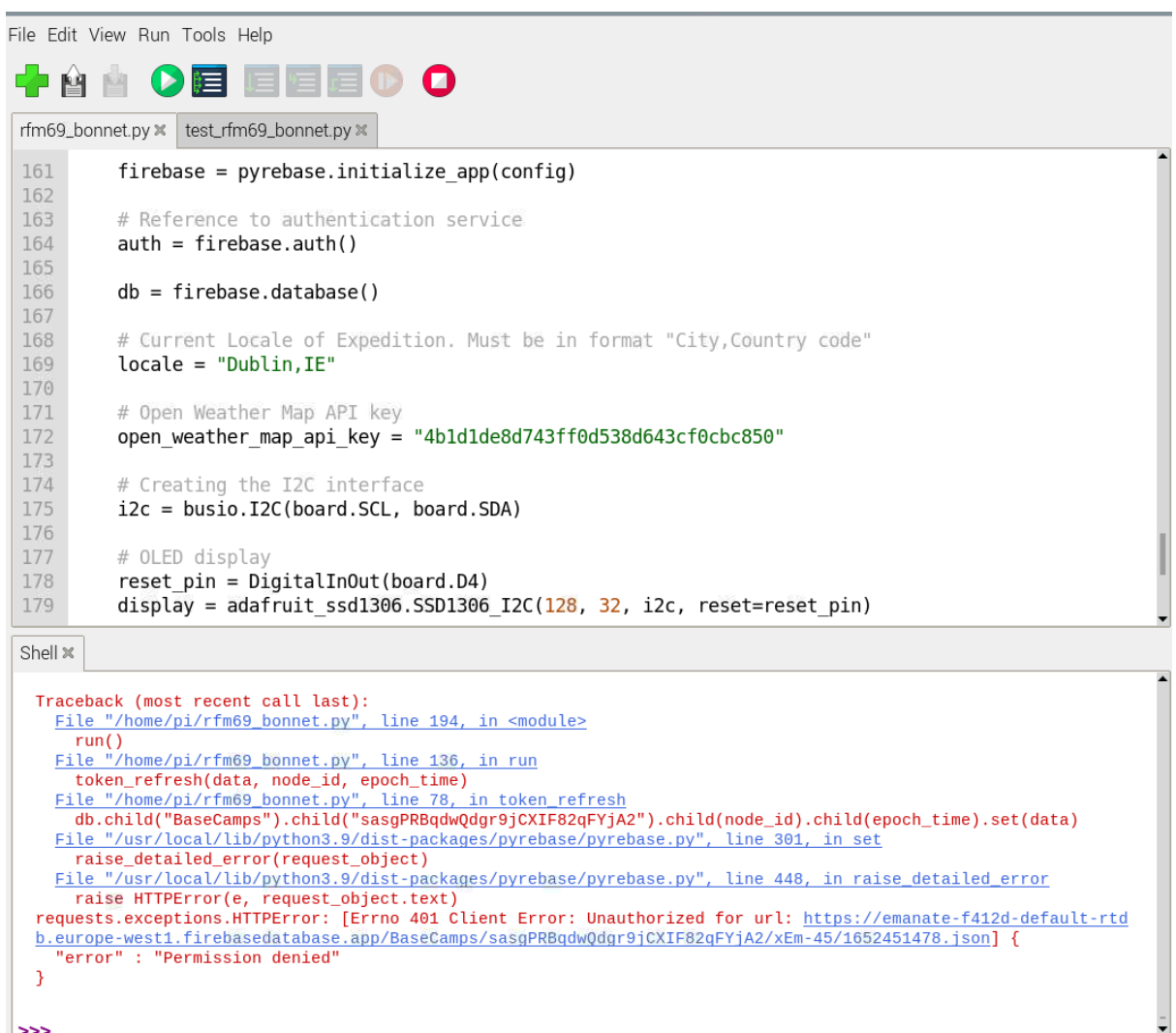
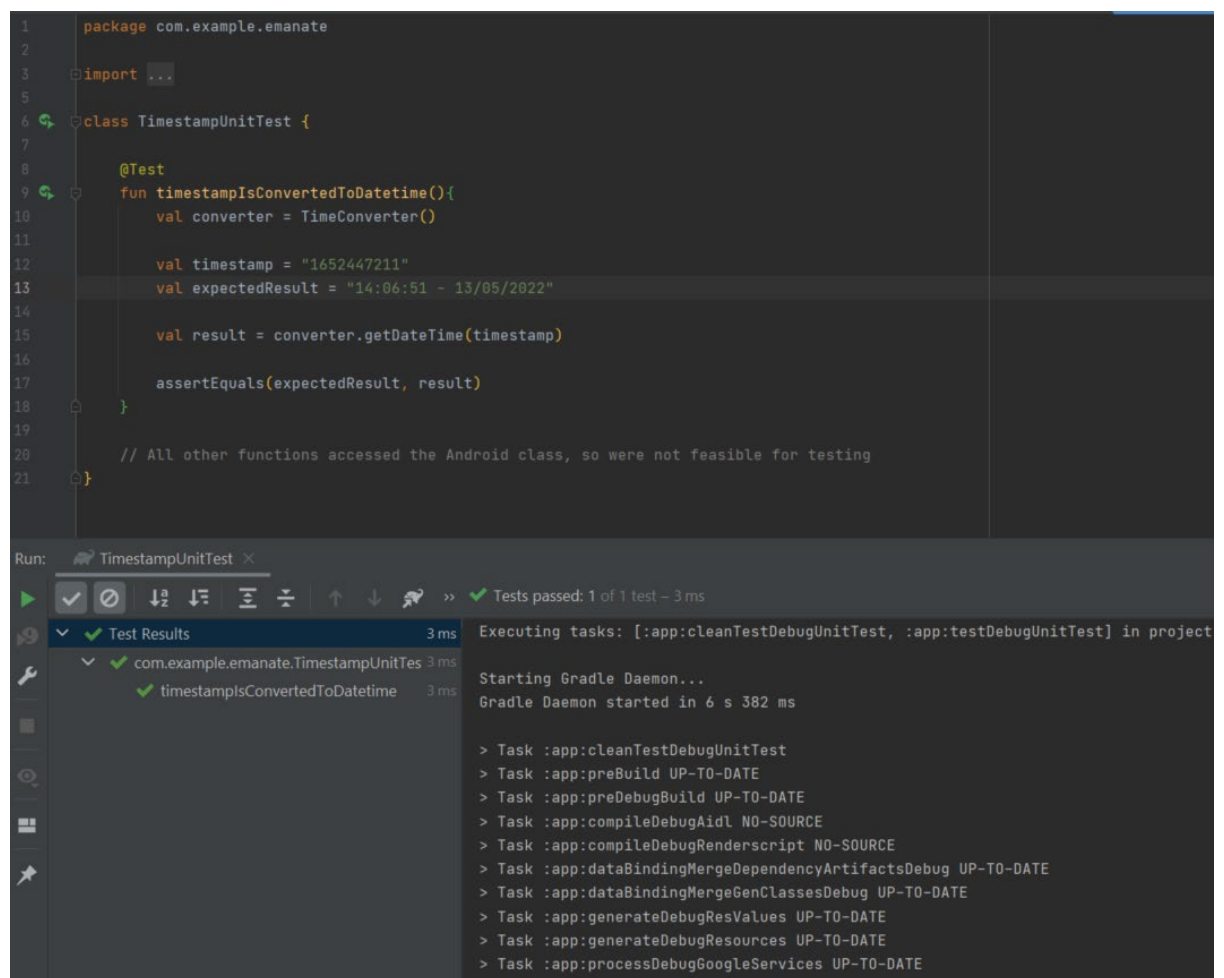


Figure 2.5.8.2: Database Rules Trying to Write Without Admin

Figure 2.5.8.2 is the rfm69_bonnet.py file without the credentials in the master radio, which prevents it from writing to the database. The permission is denied, and we know that it was because of the rules in the database.

2.5.9 Android Timestamp Conversion

Finally, JUnit tests were attempted in the Android application. All the classes and objects inside the application were examined to see if they were fit for testing. Most were not testable because they had to access the Android class, including anything to do with Android, and access to Firebase. This caused tests to fail when they were run so they were removed. For this reason, only one test could be conducted, a conversion function from timestamp to datetime.



```
1 package com.example.emanate
2
3 import ...
4
5
6 class TimestampUnitTest {
7
8     @Test
9     fun timestampIsConvertedToDatetime(){
10         val converter = TimeConverter()
11
12         val timestamp = "1652447211"
13         val expectedResult = "14:06:51 - 13/05/2022"
14
15         val result = converter.getDateTime(timestamp)
16
17         assertEquals(expectedResult, result)
18     }
19
20     // All other functions accessed the Android class, so were not feasible for testing
21 }
```

Run: TimestampUnitTest x

Tests passed: 1 of 1 test - 3 ms

Test Results 3 ms

- com.example.emanate.TimestampUnitTest 3 ms
 - timestampIsConvertedToDatetime 3 ms

Executing tasks: [:app:cleanTestDebugUnitTest, :app:testDebugUnitTest] in project

Starting Gradle Daemon...

Gradle Daemon started in 6 s 382 ms

- > Task :app:cleanTestDebugUnitTest
- > Task :app:preBuild UP-TO-DATE
- > Task :app:preDebugBuild UP-TO-DATE
- > Task :app:compileDebugAidl NO-SOURCE
- > Task :app:compileDebugRenderscript NO-SOURCE
- > Task :app:dataBindingMergeDependencyArtifactsDebug UP-TO-DATE
- > Task :app:dataBindingMergeGenClassesDebug UP-TO-DATE
- > Task :app:generateDebugResValues UP-TO-DATE
- > Task :app:generateDebugResources UP-TO-DATE
- > Task :app:processDebugGoogleServices UP-TO-DATE

Figure 2.5.9.1: Android Timestamp Conversion

Figure 2.5.9.1 shows that the test passed and would have failed if a timestamp that was unreadable was given.

```

1 import android.util.Log
2 import java.text.SimpleDateFormat
3 import java.util.*
4
5 class TimeConverter {
6     // Conversion from timestamp to datetime
7     fun getDateTime(s: String): String? {
8         return try {
9             val sdf = SimpleDateFormat(pattern: "H:mm:s - dd/MM/yyyy", Locale.UK)
10            val netDate = Date(date: s.toLong() * 1000)
11            sdf.format(netDate)
12        } catch (e: Exception) {
13            Log.d(tag: "TimeConverter.kt", msg: "Could not convert to datetime")
14            "Unreadable time"
15        }
16    }
17 }

```

Figure 2.5.9.2: Android Timestamp Conversion Using Android Class

Highlighted in figure 2.5.9.2 I introduced a log which would trigger if the conversion was not able to be done. This should make the code not testable since log is an Android class as seen in the library at the top of the figure.

```

1 package com.example.emanate
2
3 import ...
4
5 class TimestampUnitTest {
6
7     @Test
8     fun timestampIsConvertedToDatetime(){
9         val converter = TimeConverter()
10
11         val timestamp = "16524472f11"
12         val expectedResult = "14:06:51 - 13/05/2022"
13
14         val result = converter.getDateTime(timestamp)
15
16         assertEquals(expectedResult, result)
17     }
18
19     // All other functions accessed the Android class, so were not feasible for testing
20 }

```

Figure 2.5.9.3: Android Timestamp Conversion Throwing Error to Activate Android Class

For the conversion to datetime function I am passing in 16524472f11 as seen in figure 2.5.9.3. This timestamp is not applicable as it has the letter “f” in it and will throw an error from the method.

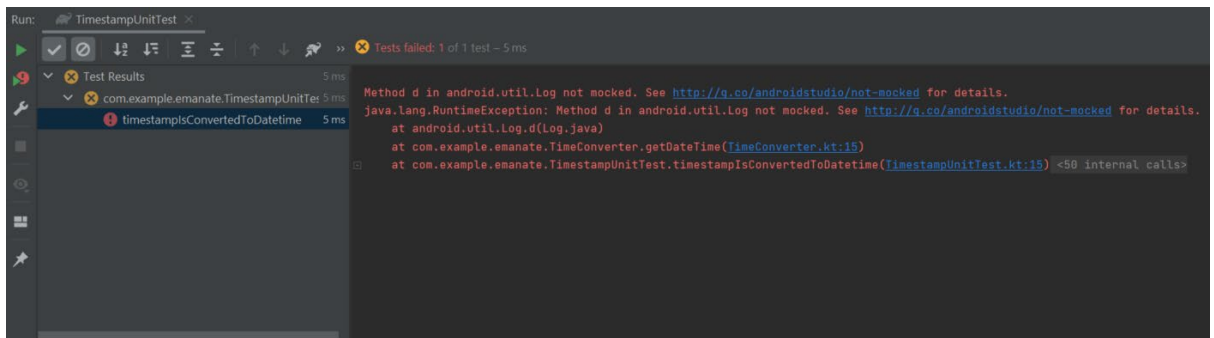


Figure 2.5.9.4: Android Timestamp Conversion Throwing Error Because Android Class Being Used

Figure 2.5.9.4 shows that the error was thrown but could not show the log in test because it is an Android class, thus the test has failed but the function has succeeded. Similarly, this is how Android prevents me from testing all the other functions inside the Android application.

2.6. Evaluation

Emanate was designed to accommodate for hikers or people on similar expeditions who needed a device which acted as a handsfree monitor for emergencies. Originally, the idea involved electronic platforms which were able to be stitched to a wearer and were small enough to be simple to integrate.

As development progressed and higher cost was involved, the idea of small wearable technology became a more impossible task. The electronic platforms, both Gemma and Flora by Adafruit, did not provide enough connectors for the radio and environmental sensor. Heart rate monitors, wind sensors, and GPS attachments also were not calculated for costs, often over €50 each. This was not ideal since money was already spent on the electronic platforms.

Two Raspberry Pis needed to be used instead with one breadboard attaching the hardware together. This still allowed for the nodes to be portable with a portable power bank but needed a much larger setup for the Raspberry Pis to function compared to the Arduino setup.

Other college projects and exams interrupted the projected Gantt chart and timeline after February. This moved most of the work to the start of the year beginning of May, but most objectives were still met. The database functions exactly as needed as well as the Android application providing a real time monitoring of hardware over a radio frequency. Although, it can be said that if other college modules did not interfere with the scope of project more careful planning of the overall architecture and testing could have been achieved.

3.0 Conclusions

Emanate can be used as planned and does satisfy the core needs of the initial planning. Nodes can send packets of data to the master radio, the master radio can write to the secure database on Firebase, and all data can be viewed on the Android application with added notifications in case of emergencies.

The unfortunate downsides to all these functionalities are the node sensors, which are somewhat lacking for hikers with the sensors attached. Radio conditions and technology

also limit distance for the transceivers, and the less-than-ideal portability of the nodes make it difficult to carry over long distances.

The closest competitor to Emanate is GoTenna, which requires the hikers to always have an internet connection. This is not a requirement for Emanate. If the technology was implemented into smaller hardware with more sensors, I believe that this project would be capable of real-world use.

4.0 Further Development or Research

Research was conducted on the metrics that were given by the BME688 sensor from Bosch. The sensor provided some metrics that can be useful for hikers but with deeper analysis another component could have been better. The metrics provided are temperature, gas, pressure, and humidity.

Temperature, pressure, and humidity was used to identify a range of scenarios that a hiker may be in, such as high altitude which puts the hiker at risk of developing altitude sickness above 2,500 metres above sea level (Altitude sickness - NHS, 2020), a sudden drop in altitude which may indicate a fall, and a rise in temperature and fall in humidity which could be a sign of a forest fire in the area.

The gas metric was not applicable for many usages as however, as it only gave a singular resistance value for the overall volatile organic compounds (VOC) in the environment. These gases could be used in a scenario where the hiker is hiking near a volcano and needs to be careful with the air they breathe. Even though it is unhealthy, far too many sources produce VOC gases in everyday life such as indoor cupboards, paint, pesticides, and regular furniture. (Volatile Organic Compounds in Commonly Used Products, 2021). To quote the product page: *"The BME688 takes those sensors to the next step in that it contains a small MOX sensor. The heated metal oxide changes resistance based on the volatile organic compounds (VOC) in the air, so it can be used to detect gasses & alcohols such as Ethanol, Alcohol, and Carbon Monoxide, and perform air quality measurements. Note it will give you one resistance value, with overall VOC content, but it cannot differentiate gasses or alcohols."* (Industries, 2016)

Bosch's sensor has an AI onboard. This can be accessed with the company's library, however there is no real use for it because it provides air quality measurements which are indoor, useless in outdoor expeditions. (Busler, 2022) It is useful to note that when testing the sensor and seeing the gas measurements inside a house, the gas resistance values varied greatly, from 300 to 4000 ohms. This would be increasingly hard to measure when moving. These measurements could also be more of an annoyance if the VOC gases are high in the area but are not a cause for concern in

Two qualified outdoor activity instructors were asked to give their opinion on Emanate in its early stages. They were going to test the system but because of time constraints and distance between me and them, this was not done. However, it was quite useful in the idea behind the project, and influenced how the whole system functioned, identifying parts which would likely have little use to hikers in real world scenarios.

Below is a link to both anonymous data gathering that was received, one recording and one text file.

5.0 References

Bosch. 2018. *How to calculate the altitude from the pressure sensor data?*. [online] Available at: <<https://community.bosch-sensortec.com/t5/Question-and-answers/How-to-calculate-the-altitude-from-the-pressure-sensor-data/qaq-p/5702>> [Accessed 14 May 2022].

Ada, L., 2016. *Adafruit RFM69HCW and RFM9X LoRa Packet Radio Breakouts*. [online] Adafruit Learning System. Available at: <<https://learn.adafruit.com/adafruit-rfm69hwc-and-rfm96-rfm95-rfm98-lora-packet-padio-breakouts>> [Accessed 14 May 2022].

Rembor, K., 2019. *Adafruit Radio Bonnets with OLED Display - RFM69 or RFM9X*. [online] Adafruit Learning System. Available at: <<https://learn.adafruit.com/adafruit-radio-bonnets>> [Accessed 14 May 2022].

Health.ny.gov. 2021. *Volatile Organic Compounds in Commonly Used Products*. [online] Available at: <https://www.health.ny.gov/environmental/air_quality/vocs.htm> [Accessed 14 May 2022].

Industries, A., 2016. *Adafruit BME688 - Temperature, Humidity, Pressure and Gas Sensor*. [online] Adafruit.com. Available at: <<https://www.adafruit.com/product/5046#:~:text=The%20BME688%20takes%20those%20sensor%20s,and%20perform%20air%20quality%20measurements.>>> [Accessed 14 May 2022].

Busler, N., 2022. *GitHub - pi3g/bme68x-python-library: Python 3 Library for BME688 and BME680 (Bosch Sensortec sensors), supports Bosch BSEC*. [online] GitHub. Available at: <<https://github.com/pi3g/bme68x-python-library>> [Accessed 14 May 2022].

nhs.uk. 2020. *Altitude sickness - NHS*. [online] Available at: <<https://www.nhs.uk/conditions/altitude-sickness/#:~:text=Symptoms%20of%20altitude%20sickness%20usually,headache>> [Accessed 14 May 2022].

6.0 Appendices

6.1. Project Proposal

1.0 Objectives

Emanate will create a reliable radio network for hikers in low connectivity conditions. These could be casual hikes or emergencies. The project will attempt to solve present day problems but will be regarded as a prototype. Mobile electronics are difficult to protect outside conditions, so ratings like Ingress Protection will not be covered. Thus, for example, even though sailing is applicable, that environment will not be measured as it is hard to research.

The project should provide a ubiquitous wearing experience for the users of the radio nodes. They should not get in the way of the user when they are performing their outdoor activity. A headquarters away from the nodes should be equipped with a Raspberry Pi with radio technology

(That writes node data to a database), and a user with the mobile phone app to read the data from the wireless nodes.

The purpose of the mobile app is to monitor the safety of the individuals on the hike. Any concerning metrics given out by the sensors will be represented as an alert in the app. The app will have a UI and rely on push notifications for needed attention within the nodes.

The IoT wearables will be modified to suit the monitoring of the hikers. Presently, I think that monitoring altitude, heart rate, temperature, and barometric pressure would benefit hikers the most. These sensors would change if another activity would be considered, for example, running as part of a group or sailing.

2.0 Background

I chose to undertake this project because I wanted to explore technology in an outside setting. I often enjoy exercising outside and liked the idea of applying software to a new environment. I also wanted to improve and expand my software skills in categories I never worked with before. I have very little experience with Android Studio and wanted to find out more about the software. Additionally, I found that working with Raspberry Pis in the past quite rewarding and radio communications is a useful area for outdoor adventures.

The Adafruit Feather 32u4 with RFM69HCW would be perfect for this project, being that it's a small microcomputer that saves a lot of space with a radio already attached. I could integrate this module in the future if the Raspberry Pis are not applicable for the nodes. To adhere to the easy wearing of the nodes I will have to limit my modifications on the nodes, preventing unnecessary modules from getting in the way. I'm aware that a smartwatch would satisfy many of the requirements for wearable technology, but many of them are reliant on a phone as well for communication. This becomes more complex when costs are a factor for everyone in the group.

The Raspberry Pi will have a straightforward function of retrieving the data from the nodes and writing them to a database. This database should be handled by Firebase and be connected to a mobile app written in Kotlin. The UI will include aspects like node IDs for monitoring, alerts for data that is concerning, and the possibility of an interactive map using Google API.

3.0 State of the Art

Emanate is a somewhat unique software project, many of the similar applications have the drawback of relying on internet coverage. Emanate is instead based on radio. Similar physical IOT devices also usually cost a significant amount compared to my estimated software project costs. Several phone apps fulfil the goal of Emanate's hiking communications but aren't practical for emergencies if the phone were to die.

The goTenna Mesh is the most similar application to Emanate. The Mesh are mobile devices that allow peer-to-peer communication between each other. A person carries the device with something like a carabiner and connects it to their phone with Bluetooth. The Mesh provides many ways of

staying in contact like messaging and even a button for relaying information in case of emergencies. The Mesh communicates through radio, although it has many problems with providing a reliable phone connection and line-of-sight (LoS). LoS is the ability to maintain a radio connection with obstacles in the way.

Although quite similar, there are a few other issues. The Mesh is expensive for users with a \$180 price tag, in-app subscriptions, and a lack of on-body sensors. My project should address these issues and give alternatives, the key difference being that Emanate would be using wireless nodes on the person instead of being attached to a strap.

4.0 Technical Approach

The most important part of the project is the electronic platforms. I plan to experiment with the new technology, seeing how I can best implement my code functionality. I'm unsure what approach I will take now but getting the Raspberry Pi set up with the wireless node radio transceivers will be the best way to understand the platforms. It would be good to mention that these platforms would need to be delivered to me from outside of Ireland and could have an impact on planning if there are delays in delivery times.

I think that examining the platforms will help me identify the requirements more accurately. For example, there may be an additional required component that I'm unaware of before working with the electronic platforms. This could slow down the initial plans for the project. As well as that, it's good to note that I'll likely have to tune and consider specifications such as volts, battery mAh, and frequencies associated with the radio transceivers. I will be designating a large timeframe to get this completed early (and to compensate for handling the platforms if I run into complications). This means that any activities afterwards are likely to change. If the components aren't delivered in time I may need to work on the app and database with simulated data in the meantime.

The database and the mobile app will be the final parts of the project. The database should be the less complicated of the two, but more directly related to the electronic components because it stores and transforms the data for the mobile app. These would likely be attempted after Christmas. The phone application will be more complicated than the database due to the functionality of handling and displaying the platform information for human interaction. I'm not sure how basic the UI/UX design will be, it will depend on how the rest of the project is functioning towards the end of the last semester.

I will design a Trello Board Gantt chart with the Elegantt extension. This will give a great visual representation of the deadlines I set myself for Emanate. I am planning to follow this Gantt chart as close as I can, in respect of other module CAs which may get in the way of the planned workload

5.0 Technical Details

This project has few complicated algorithms as of the present hypotheses. From an overview of Emanate, the project only reads data and transports it to the user. It will be using a variety of languages to do so, such as Python, Java, and Kotlin as of now.

The platforms will probably need some Arduino/Raspberry Pi libraries to function with the Mu Editor. Any more libraries could arise when coding with Python, for example, the SQLite libraries if I were to handle my database from the Raspberry Pi.

Moving to the database this is where some algorithms could be present. Firebase will likely be hosting my database, a platform that supports coding in a range of languages. I'll probably be using Java for this. It's possible that because I'm representing data through the app in Kotlin I'll need to have some algorithm for transforming the data from Java, however, it's not expected to have any since I'll only be querying data within the database.

As for the app, I will have to import some libraries for taking in the database data along with any libraries relevant for my customization of the app. Android Studio handles a lot of the importing libraries for me which happens when designing the activities. Typical libraries of the "android.widget" classes like SearchView, Toast, ListView, ArrayAdapter, etc, will likely be used.

Emanate is working with a few different technologies. This means that unprecedented changes and implementations of languages and libraries are certain to happen. The above text is only an assumption for the possible resources required for Emanate.

6.0 Special Resources Required

I will be needing many components from Adafruit. Here is a list of the components:

- Raspberry Pis/Arduinos
- Adafruit Feather 32u4 with RFM69HCW
- Adafruit LoRa Radio Bonnets
- Adafruit LoRa Radio Transceivers
- Adafruit BME688 - Temperature, Humidity, Pressure and Gas Sensor
- Lithium-Ion Polymer Battery
- Polar Heart Rate Sensor Starter Pack

The Raspberry Pi or Arduino will be the most expensive part in Emanate. It's crucial for the project's functionality.

Either a Raspberry Pi or Adafruit Feather 32u4 with RFM69HCW will be the wireless nodes for the users.

The radio bonnets and transceivers are needed for the node to Raspberry Pi communication.

The Adafruit BME688 is a sensor that provides temperature, humidity, pressure, gas, and altitude in one package. This covers every sensor aspect I'm planning to work with except for heart rate.

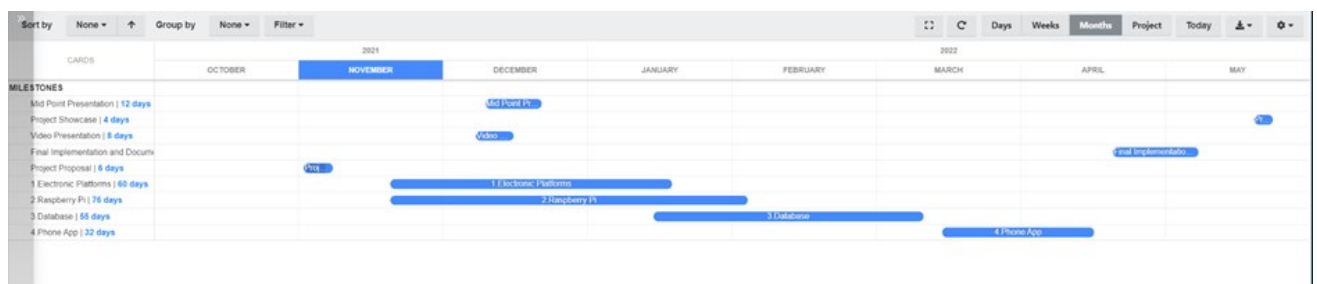
I will need a small lithium-ion battery with a mAh depending on how much power the nodes need.

This heart rate sensor could be implemented for the nodes but is currently quite costly.

7.0 Project Plan

Since I don't own Microsoft Project, and it would be difficult to manage over NCI's virtual desktop, I'm designing the Gantt chart with Elegantt. I have also been using Trello to manage my projects since my first year of NCI.

I chose not to include the Viva Examination in the timeline because no due date has been given yet.



I designed the Gantt chart in a WBS format. Above are my first level project elements, electronics platforms, raspberry pi, database, phone app, and the CAs. The lower levels are inside of those cards. I left a lot of room for the sensors and Raspberry Pi since they could take a larger amount of time to complete.

Below are the lower levels of the elements. Unfortunately, I can't assign due dates without a subscription, but they will lie within the first level element timeline.

The screenshot shows the 'Deliverables' section of the Elegantt project management interface. It features a list of tasks with checkboxes and a progress bar at 0%.

- ☒ **Deliverables** Delete
- 0% Progress bar
- ☐ 1.1 Testing Sensors
- ☐ 1.2 Manipulating the Data for the Raspberry Pi to Read
- ☐ 1.3 Linking Sensors to Raspberry Pi
- ☐ 1.4 Testing Human Comfort with Platforms i.e. Wearable Hats
- Add an item

☒ Deliverables Delete

0%

☐ 2.1 Setting Up the Mu Editor and Connections for Designing Pi with Laptop

☐ 2.2 Getting the Platforms and Radio Tranceivers Connected to the Raspberry Pi

☐ 2.3 Manipulating the Sensor Data for Viewing with Raspberry Pi

☐ 2.4 Designing How Raspberry Pi Will Talk to Database

Add an item

☒ Deliverables Delete

0%

☐ 3.1 Designing the Database with Firebase

☐ 3.2 Designing the Database so it Extracts Data From the Raspberry Pi

☐ 3.3 Working Out How to Implement Data Concisely into the Database

☐ 3.4 Writing the Database Data to the Mobile App

Add an item

8.0 Testing

I hope to test Emanate with real-life hikers if the ethics board approves my application. Outside of that, I will be testing the project platforms by myself. I could have the Raspberry Pi set up in my home and take the platforms outside to measure the realistic radio distance they can achieve. This will also be good for seeing how they will fit on the body. I may need another person to test the app in my home for alerts within the app, but it would also be possible to take the phone with my app when using the sensors for quicker testing.

Coding the platforms will likely be done with Python on the Raspberry Pi. I would say that testing this code will be mainly exploratory testing, where I test features manually since a lot of the components are working on separate systems from each other. However, in a case where more varied testing is needed, I will probably be using Unit testing with a Python test runner. This could help solve some more difficult problems within the scripts.

Emanate makes use of hardware along with software. This means that the hardware could break when designing the project. It also makes it difficult to test problems if hardware specifications act differently without me knowing. To be fully aware of the hardware, I should test and note the voltage output and other readings. In a case where the hardware fails, I would be able to narrow down the root of the issue easier. In past projects, I have had hardware components break. This hugely affected the project structure and caused myself and my team members to change how the project operated.

The database will be handled by Firebase. I don't have any experience with Firebase, but I know that I can use Firebase Test Lab to test any problems within my database. This will be particularly useful because I can see how my database performs with my application. It does this as an instrumentation test against a test matrix on Android.

For my android app, I can use Android Studio's test feature with the "testImplementation" for JUnit testing. I have some familiarity with this from doing some JUnit testing from second-year modules. For Android Studio I can test multiple devices in parallel. This will aid me in finding the issues early before they arise when coding.

Some of the UI in the mobile app could be challenging to work with. At times the XML design elements in Android behave differently across platforms. This could affect how the information retrieved from the database is displayed. The problem that I may have will be that different phones would display elements in separate dimensions. I must note that experimenting with the end-user interface will be needed, meaning that I will have to test if the elements are responsive across platforms.

1.1. Ethics Approval Application (only if required)

National College of Ireland

Ethical Guidelines and Procedures for Research involving Human Participants



SEPTEMBER 2017

1. Introduction

All research involving human participants that is conducted by students or staff at the National College of Ireland should be done so in an ethical manner. The college has therefore developed an Ethics Committee, which acts as a sub-committee of the Research Committee, to ensure that ethical principles pertaining to research involving human participants are upheld and adhered to. All researchers intending to use human participants as part of their projects are thus required to reflect upon any potential ethical issues and submit their research proposals for ethical review before commencing data collection.

This document gives an overview of the core ethical principles guiding research in NCI, while also documenting the procedures required for seeking ethical approval of research involving human participants.

Am I conducting research?

Research is defined as “the attempt to derive generalisable new knowledge by addressing clearly-defined questions with systematic and rigorous methods” (NHS Health Research Authority). Sometimes, we collect data in order to evaluate a service or practice we are engaged in (“service evaluation”). The main difference between research and service evaluation is in the aim: research is trying to create new generalisable knowledge, and service evaluation is trying to evaluate whether a delivered service/practice is working well. One project may have both aims included in it. It can be confusing if a service or intervention is involved, whether or not research is being conducted. If new or competing interventions are being evaluated, then it is likely to be research, whereas if an existing service is being conducted anyway, with an evaluative component, then it is likely to be a service evaluation. Research requires consideration of the below guiding principles, whereas service evaluation does not require approval from an ethics committee.

2. Guiding Principles

In line with other research institutions, there are three core guiding principles governing the ethical conductance of research involving human participants at NCI. These principles stem from the *Belmont Report* (1979) published by the National Commission for the Protection of Human Subjects of Biomedical and Behavioural Research. While it is recognised that these principles may be operationalised differently depending on the specific research discipline, it is recommended that these are consulted as a starting point for any research involving human participants.

2.1 Principle 1: Respect for Persons

This principle entails recognition that participants should be treated as autonomous individuals and hence should never be coerced or swayed into participating in a research project against their will. The participant’s right to withdraw from a research study at any time should be respected, as well as their right to dignity and protection from harm.

Respect for individuals can often be implemented in practice via the process of informed consent, whereby potential participants are made fully aware of the requirements involved in participation. While it is recognised that in certain cases deception (i.e. the withholding of certain information from participants) may take place, this should only occur when it is robustly justified for the validity of the research. In cases where deception is justified, researchers should ensure that any potential risk

resulting from this measure is minimised. Participants should also be fully debriefed on the nature of the research after it has taken place.

The principle of respect also requires researchers to protect individuals from vulnerable groups who may have diminished autonomy (see section 4.2 for more detail as to what constitutes vulnerable groups). Where full informed consent is not possible for such population groups, consent may instead be sought from their guardians. In all cases however clear assent, or willingness to participate, should be demonstrated from participants.

2.2 Principle 2: Beneficence and non-maleficence

This principle specifically focuses on the need to protect the well-being of participants. Any potential risk to participants should be minimised, whether that be risk of physical discomfort or of any psychological, emotional or social distress, while possible benefits should be maximised. Researchers adhering to this principle should thus ensure that any potential benefits derived from carrying out the study (e.g. in terms of knowledge gained) should outweigh potential risks. Even in cases where there is only a slight potential risk of harm, participants should be provided with appropriate support to alleviate this.

2.3 Principle 3: Justice

This principle emphasises the need to employ fairness in the distribution of benefits and risks to participants. The way in which participants are selected to take part in research should relate to the purpose of the study, as opposed to other factors such as availability or manipulability of participants. The exploitation of vulnerable populations should be avoided.

Where applicable, researchers are encouraged to consult guidelines stemming from their own professional bodies (e.g. The Psychological Society of Ireland) in addition to the general guiding principles above when planning their research. Researchers should also be sensitive to those issues which are specific to the population under investigation and the methodology that is employed in the project (e.g. qualitative methodologies involving the recording of data may raise issues relating to participants' right to anonymity, as well as the ethical management and use of data). Detailed consideration should be given to all these issues when planning research and when completing the Ethical Review Application form.

3. Ethics Committee

The NCI Ethics Committee was established by the Academic Council in 2012. Acting as a sub-committee to the Research Committee, its role is to oversee ethical issues arising from all research involving human participants that is conducted by students and staff of the college. The key purpose of this committee is to safeguard against any potential harm to participants, and to ensure that their rights are recognised in line with the guiding principles outlined above.

The Ethics Committee reviews all research proposals posing ethical risk to the participants involved, however the decision as to whether projects pose ethical risk is firstly made via the appropriate Filter

Committee which operates at School level (see organisational structure in Figure 1 below). The Filter Committees may review and approve research proposals which are of low ethical risk, while referring those of high ethical risk to be considered by the Ethics Committee (see categories of ethical risk in section 4.1).

While the Filter Committees are made up of staff members with subject-specific knowledge, membership of the Ethics Committee should comprise of no less than five representatives from both the School of Computing and the School of Business, including representatives from the Research Committee.

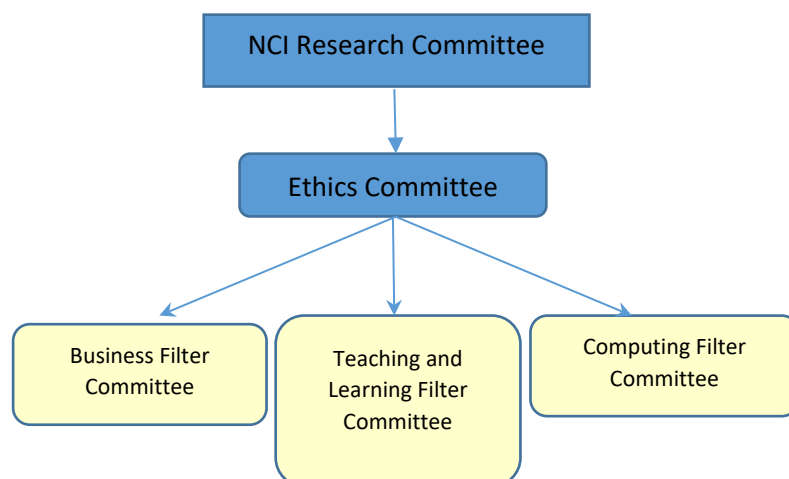


Figure 1: Committee Structures.

4. Review Process

Any staff or student of NCI wishing to conduct a study involving human participants should first submit the Ethical Review Application Form (included at the end of this document), to the relevant School Filter Committee at proposal stage. This initial review will result in a graded categorisation of ethical risk, as outlined below.

4.1 Categorisation of Ethical Risk

Research category A

Research in this category poses little ethical risk to the participants involved. Specifically, it refers to research involving human volunteers, but **excluding** studies involving:

- therapeutic interventions
- new research methodologies
- vulnerable populations (see section 4.2)
- deception of the participants
- any other significant physical, social or psychological risk to participants

Research category B

Research in this category involves human volunteers **including** studies involving:

- therapeutic interventions
- new research methodologies
- vulnerable populations (see section 4.2)
- deception of the participants
- any potentially significant risk to participants

Research Category C

This specifically refers to research involving human volunteers who are service users, patients, staff, records, etc., within the sphere of the HSE or similar setting (but not including clinical trials of investigative medicinal products).

4.2 Vulnerable groups

There are a number of participant populations that may fall under the heading of ‘vulnerable groups’. These groups require consideration of unique ethical challenges regardless of the nature of the project. Research involving such populations should therefore always be reviewed by the Ethics Committee.

Groups that may be classed as vulnerable include, but are not limited to:

- Children (under 18 years of age)
- The older old (aged 85+)
- People with an intellectual or learning disability
- Individuals or groups receiving help through the voluntary sector
- Those in a subordinate position to the researcher (e.g. employees)
- Any other groups who might not understand the research and consent process

Note: in addition to the Ethical Review process, any researchers intending to work directly with children will be required to undergo Garda Vetting in advance of the proposed research.

4.3 Exemption from Full Ethical Review

In certain limited cases, researchers can apply for an exemption from full ethical review. In such cases, the Ethical Review Exemption form should be completed, explicitly detailing why the exemption is sought.

In completing this form, researchers must declare that the research does not involve any of the following:

- Vulnerable groups
- Sensitive topics
- Risk of psychological or mental distress
- Risk of physical stress or discomfort
- Any other risk to participants
- Use of drugs or invasive procedures (e.g. blood sampling)
- Deception or withholding of information from participants

- Conflict of interest issues
- Access to data by individuals or organisations other than the researchers
- Any other ethical dilemmas

4.4 Outcomes of Review Process

Following consideration of research projects submitted for Ethical Review, each Filter Committee will submit a report to the Ethics Committee summarising the applications considered and the decisions made.

For research that is deemed to fall under Research Category A (low ethical risk), a favourable outcome at the relevant Filter Committee will be sufficient to secure ethical approval. Research falling under the other two categories must however be considered by the Ethics Committee before approval may be granted.

On the basis of this review, four key outcomes may arise:

1. Research proposal approved (no recommendations)
2. Research proposal approved pending minor revisions (to be accepted by the Chair and Research Supervisor)
3. Research proposal approved pending major revisions (to be resubmitted and approved by the Ethics Committee)
4. Research proposal rejected (resubmission necessary)

A summary of the processes involved in applying for ethical approval can be seen in Figure 2.

Appeals

Appeals against the Committee's decision may be made within ten working days. In this case, at least three members of the Ethics Committee, none of whom will have reviewed the initial application, may review this along with any additional information submitted by the applicant.

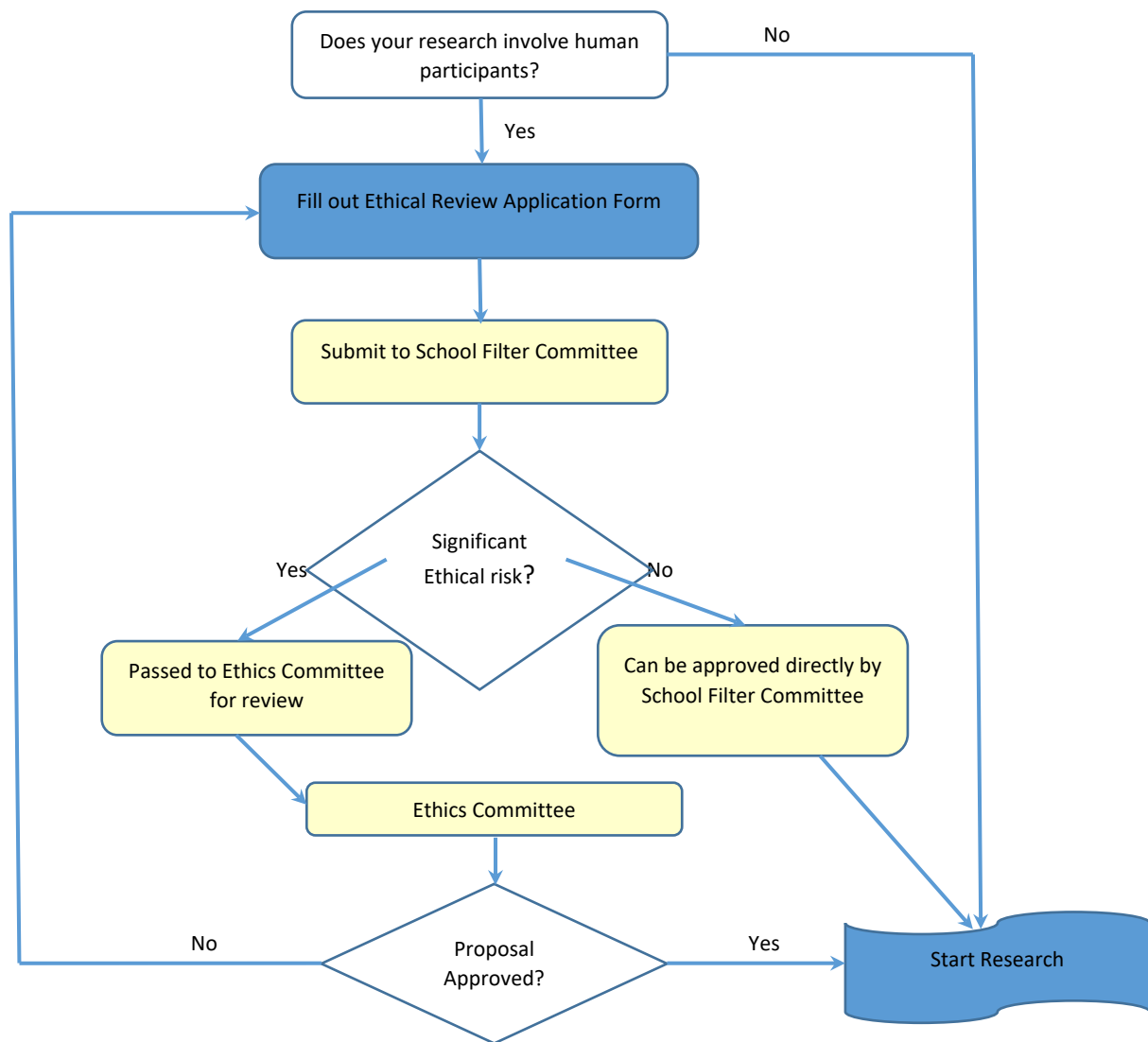


Figure 2: Process chart for seeking Ethical Approval

Ethics Application Checklist

To be submitted alongside the ethics application.

Please complete the below checklist, ticking each item to confirm that it has been addressed.

1. I agree to obtain informed written consent from all human participants aged over 18 who are involved in this research (or if circulating digitally, I will ensure that informed consent is completed, and will have the participants indicate their informed consent by continuing with their study engagement).	<input type="checkbox"/>
2. I agree to obtain informed written consent from the parents of anyone aged under 18 in this research (or from the schools if appropriate), and informed written assent from those under 18 in this research.	<input type="checkbox"/>
3. I include a letter of agreement from a clinically responsible individual agreeing to (where appropriate) help me recruit/provide clinical support in the event that participants become distressed/host the study data collection.	<input type="checkbox"/>
4. I append a letter of agreement from an external institution or organisation agreeing to host the study.	<input type="checkbox"/>
5. I agree to comply with NCI's Data Retention Policy.	<input type="checkbox"/>
6. I have appended a) information sheet, b) consent form/assent form, c) debriefing sheet.	<input type="checkbox"/>
7. I have provided details of how non-anonymised data will be stored, in a safe and encrypted manner.	<input type="checkbox"/>
8. I have included my contact details and those of my supervisor (where appropriate). I have only included my NCI email address and not included any personal contact information.	<input type="checkbox"/>
9. I have given sufficient details on the proposed study design, methodology, and data collection procedures, to allow a full ethical review, and I understand that my failure to give sufficient detail may result in a resubmission being required.	<input type="checkbox"/>
10. I understand that if I make changes to my study following ethical approval, it is my responsibility to seek an ethics amendment if the change merits ethical consideration.	<input type="checkbox"/>

National College of Ireland

Human Participants Ethical Review Application Form

All parts of the below form must be completed. However in certain cases where sections are not relevant to the proposed study, clearly mark NA in the box provided.

Part A: Title of Project and Contact Information

Name

Adam Downey

Student Number (if applicable)

X18455846

Email

X18455846@student.ncirl.ie

Status:

Undergraduate ☒

Postgraduate ☐

Staff ☐

Supervisor (if applicable)

Frances Sheridan

Title of Research Project

Emanate: Radio communication for hikers

Category into which the proposed research falls (see guidelines)

Research Category A ☒

Research Category B ☐

Research Category C ☐

Have you read the NCI Ethical Guidelines for Research with Human Participants?

Yes ☒

No ☐

Please indicate any other ethical guidelines or codes of conduct you have consulted

NA

Has this research been submitted to any other research ethics committee?

Yes ☒

No ☐

If yes please provide details, and the outcomes of this process, if applicable:

NA

Is this research supported by any form of research funding?

Yes ☐

No ☒

If yes please provide details, and indicate whether any restrictions exist on the freedom of the researcher to publish the results:

NA

Part B: Research Proposal

Briefly outline the following information (not more than 200 words in any section).

Proposed starting date and duration of project

Proposed date: 8/11/21 Duration of Project: 7-8 months

The rationale for the project

To design a custom radio communications software project for final year BSHC

The research aims and objectives

To gather information from outdoor activity instructors. Information may include thoughts on the project, and demonstrations of its physical component usage in an outdoor setting.

There is a possibility that data will be asked of students in NCI for some outdoor activity experience. This would be voluntary and anonymous data being gathered through Outlook.

The research design

This project is planned to have four main platforms:

- Multiple wearable wireless nodes (Flora or Gemma) that have sensors and a radio transceiver attached onto them
- A Raspberry Pi that will have a radio transceiver on it for recording wireless node data
- A database designed with Firebase
- A mobile application

It's likely that only the wireless nodes will be used for the people involved in the real-world research.

The research sample and sample size

Please indicate the sample size and your justification of this sample size. Describe the age range of participants, and whether they belong to medical groups (those currently receiving medical treatment, those not in remission from previous medical treatment, those recruited because of a previous medical condition, healthy controls recruited for a medical study) or clinical groups (those undergoing non-medical treatment such as counselling, psychoanalysis, in treatment centres, rehabilitation centres, or similar, or those with a DSM disorder diagnosis).

Age group: 21-45

Sample will more than likely focus on a select few who attend an outdoor course, have experience in this industry, or similar

If the study involves a MEDICAL or CLINICAL group, the following details are required:

- a) Do you have approval from a hospital/medical/specialist ethics committee?
If YES, please append the letter of approval. Also required is a letter from a clinically responsible authority at the host institution, supporting the study, detailing the support mechanisms in place for individuals who may become distressed as a result of participating in the study, and the potential risk to participants.
If NO, please detail why this approval cannot or has not been sought.
- b) Does the study impact on participant's medical condition, wellbeing, or health?
If YES, please append a letter of approval from a specialist ethics committee.
If NO, please give a detailed explanation about why you do not expect there to be an impact on medical condition, wellbeing, or health.

The nature of any proposed pilot study. Pilot studies are usually required if a) a new intervention is being used, b) a new questionnaire, scale or item is being used, or c) established interventions or questionnaires, scales or items are being used on a new population. If no such study is planned, explain why it is not necessary.

Large gathering of data in a population is not needed because of the research focus on the wearable technology. Will likely only need to test with only two people.

The methods of data analysis. Give details here of the analytic process (e.g. the statistical procedures planned if quantitative, and the approach taken if qualitative. It is not sufficient to name the software to be used).

The wireless nodes will be used to gather data. This data will probably be represented in a graph for characteristics like heart rate, altitude, environment temperature, etc, on the individual.

Study Procedure

Please give as detailed an account as possible of a participant's likely experience in engaging with the study, from point of first learning about the study, to study completion. State how long project participation is likely to take, and whether participants will be offered breaks. Please attach all questionnaires, interview schedules, scales, surveys, and demographic questions, etc. in the Appendix.

A lot of the study will be "passive wearing", where the users will just need to simply wear the technology on their trek while hiking. The research data will likely be limited to a day of activity monitoring

Part C: Ethical Risk

Please identify any ethical issues or risks of harm or distress which may arise during the proposed research, and how you will address this risk. Here you need to consider the potential for physical risk, social risk (i.e. loss of social status, privacy, or reputation), outside of that expected in everyday life, and whether the participant is likely to feel distress as a result of taking part in the study. Debriefing sheets must be included in the appendix if required. These should detail the participant's right to withdraw from the study, the statutory limits upon confidentiality, and the obligations of the researcher in relation to Freedom of Information legislation. Debriefing sheets should also include details of helplines and avenues for receiving support in the event that participants become distressed as a result of their involvement in this study.

I believe that there will be little to no ethical or risks of harm in the research. The only problem that may occur is that individuals may get hurt from attempting the hiking/sailing activity. Although this is not directly related to the electronics I'm planning to gather data with, more so a problem with their unrelated movement.

Do the participants belong to any of the following vulnerable groups?

(Please tick all those involved).

- ☐ Children;
- ☐ The older old (85+)
- ☐ People with an intellectual or learning disability
- ☐ Individuals or groups receiving help through the voluntary sector
- ☐ Those in a subordinate position to the researchers such as employees
- ☐ Other groups who might not understand the research and consent process
- ☐ Other vulnerable groups

How will the research participants in this study be selected, approached and recruited? From where will participants be recruited? If recruiting via an institution or organisation other than NCI please attach a letter of agreement from the host institution agreeing to host the study and circulate recruitment advertisements/email etc.

The research participants will likely be only friends of mine in a Level 6 training course within Kerry. The institution is not involved.

Some data may be gathered from NCI students via Outlook.

What inclusion or exclusion criteria will be used?

Inclusion Data

- In good physical shape
- Willingness to participate in the project research
- Honesty, and data given to best of the participant's ability.

Exclusion Data

-Any person that may be in the category of vulnerable

How will participants be informed of the nature of the study and participation?

Direct contact in person and/or over Outlook before they begin the survey

Does the study involve deception or the withholding of information? If so, provide justification for this decision.

The use of information will purely be used to demonstrate real world usage in research. Any information will not be withheld.

What procedures will be used to document the participants' consent to participate?

Agree to consent in a survey page accessed via Outlook, verbal confirmation over recordings, direct messaging confirmation if needed

Can study participants withdraw at any time without penalty? If so, how will this be communicated to participants?

Yes, they can exit the research whenever. Will have direct contact to physical demonstration participants. People doing survey will be able to access me over Outlook to remove their information.

If vulnerable groups are participating, what special arrangements will be made to deal with issues of informed consent/assent?

NA

Please include copies of any information letters, debriefing sheets, and consent forms with the application.

Part D: Confidentiality and Data Protection

Please indicate the form in which the data will be collected.

☐ Identified



☒ Potentially Identifiable

☐ De-Identified

What arrangements are in place to ensure that the identity of participants is protected?

No names, addresses, or identifiers will be present in the research. Spoken word will be used to gather some information, but will not be present in the study. The survey will likely be mostly multiple choice questions

Will any information about illegal behaviours be collected as part of the research process? If so, detail your consideration of how this information will be treated.

NA

Please indicate any recording devices being used to collect data (e.g. audio/video).

A smartphone will be used to record some data. They will be voice recorded over "Interview" mode.

Please describe the procedures for securing specific permission for the use of these recording devices in advance.

Direct contact over personal messaging

Please indicate the form in which the data will be stored.

☐ Identified

☐ Potentially Identifiable

☒ De-Identified

Who will have responsibility for the data generated by the research?

Myself. All of the data will be given to me to handle outputting graphs and documentation in the final report

Is there a possibility that the data will be archived for secondary data analysis? If so, has this been included in the informed consent process? Also include information on how and where the data will be stored for secondary analytic purposes.

No data will be published elsewhere for public viewing. All of the data being gathered will be localised with the project

If not to be stored for secondary data analysis, will the data be stored for 5 years and then destroyed, in accordance with NCI policy?

☒ Yes

☐ No

Dissemination and Reporting

Please describe how the participants will be informed of dissemination and reporting (e.g. submission for examination, reporting, publications, presentations)?

Outlook for the surveys, direct messaging and verbal communication for the component testing.

If any dissemination entails the use of audio, video and/or photographic records (including direct quotes), please describe how participants will be informed of this in advance.

These records will only be present in the physical demonstration of the project. Participants will be made aware of the audio, video, photo usage within direct messaging and verbal communication

Part E: Signed Declaration

I confirm that I have read the NCI Ethical Guidelines for Research with Human Participants, and agree to abide by them in conducting this research. I also confirm that the information provided on this form is correct (Electronic signature is acceptable).



Signature of Applicant _____

Date 2/11/21

Signature of Supervisor (where appropriate):

Date _____

Any other information the committee should be aware of?

NA

1.2. Reflective Journals

October Reflective Journal

Supervision & Reflection

Student Name	Adam Downey
Student Number	x18455846
Course	BSHCSD4

Month: October

What?

Reflect on what has happened in your project this month?

This was the first month of designing my software project. I was focusing on mainly planning the work before I got started, such as gathering the project components and solidifying my idea by visualizing the layout within class time. Performing the project idea video immensely reinforced my understanding of my project, letting me put my ideas into words for an audience.

I researched available modules that I can work with, I'll likely have to decide on either the Gemma or Flora electronic platform for the wireless node. Will therefore have to use a Raspberry Pi and Mu editor for implementing the code. I learned that using the app written in Kotlin will be using Firebase for my database.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

Researching the components of Adafruit has given me peace of mind. There is an abundance of sensors I can decide on, which leaves a lot of room for settling on a clear project idea. The hardware available is relatively cheap which is also important for putting my project together since it relies on a range of differing hardware.

The compatibility of the Flora and Gemma with the Raspberry Pi is uncertain now. I found that they can be worked on together with the Mu editor, but it was limited information from websites. I'll need to plan this more carefully if I can't work with the Raspberry Pi, the radio transceiver needs to work with the Raspberry Pi and not Arduino.

There is some stress for working with Kotlin, an entirely new language to me. I have been practising within another module to help gain some confidence in it. Thankfully the language is similar to Java, so the initial learning is the most important stage for understanding how to develop the Software Project app.

Now What?

What can you do to address outstanding challenges?

I can start purchasing the technology I'm planning to work with. A safe decision would be to buy the sensor components first as they are universal for the final project outcome. The Raspberry Pi/Arduino needs to be researched more before I can finalise buying the Gemma or Flora.

I need to understand Kotlin some more before I can be comfortable designing the app. I plan to leave this part closer to the end of the project because this displays the database data from the sensors. Still, it's important to get the foundations down early so I can solve any problems easier in the future.

Student Signature**November Reflective Journal****Supervision & Reflection**

Student Name	Adam Downey
Student Number	x18455846
Course	BSHCSD4

Month: November

What?

Reflect on what has happened in your project this month?

This month I completed my Project Proposal. It involved the complete overview of my project, including a Gantt chart for what I have to do for the next few months.

I also got the project components this month. I plan to do a lot more experimentation than I already have. There are a lot of parts that are involved with the project. I have been trying to get the Raspberry Pi to connect the radio module with CircuitPython.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

I spoke with Frances about my Project Proposal. I may need to be more specific in my deliverable due dates. There's a lot to be done in the future, I should get on top of the planning to make workflow better.

These components need to be examined more closely. It's more complicated than I thought. I need to make sure that all the components will work as hypothesized or else it will cause issues in the future.

Now What?

What can you do to address outstanding challenges?

I will need to move forward and work on my presentation deliverables for the 21st. I should give myself time to work on this around the other deadlines in the course.

I will need to get through the initial setting up for the components and Raspberry Pi. I may have to use a breadboard in the final design, but that remains to be seen until I get the technology set up. I should start small and get the components talking with Python first.

Student Signature


December Reflective Journal**Supervision & Reflection**

Student Name	Adam Downey
Student Number	x18455846
Course	BSHCSD4

Month: December

What?

Reflect on what has happened in your project this month?

Coming up to Christmas I had to handle working on the midpoint presentation while also working on other CAs that were due. I incrementally worked on parts of the midpoint presentation until its deadline. This led me to rush the work towards the end of the submission.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

Moving most of the work until the final deadline caused me to have to compensate for the hardware that broke with little time to fix it. My Radio Breakout worked for a few days, then stopped working soon after. Unfortunately, as I was so close to the deadline, I could not find the necessary time to troubleshoot the problem.

I submitted the midpoint presentation and prototype with issues within the system. I will have to resolve these problems to make the rest of Emanate work again.

Now What?

What can you do to address outstanding challenges?

I can investigate the baud rate issue when I have time again. This was not an issue with the hardware at first, so it may be able to be solved with some more research into the subject.

Student Signature


January Reflective Journal**Supervision & Reflection**

Student Name	Adam Downey
Student Number	x18455846
Course	BSHCSD4

Month: January

What?

Reflect on what has happened in your project this month?

This month I completed other module projects and TABAs. Little to no progress was made in the project.

So What?

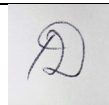
Consider what that meant for your project progress. What were your successes? What challenges still remain?

While doing a project for Multimedia and Mobile Application Development I was exposed to working with Kotlin for an Android app. I worked with linking to Firebase also. Doing so indirectly helped to gain a better understanding of the two platforms.

Now What?

What can you do to address outstanding challenges?

This means that I know how to code an Android application in Kotlin, but also know what is involved to develop a database on Firebase. This will be crucial for the future of Emanate.

Student Signature

February Reflective Journal**Supervision & Reflection**

Student Name	Adam Downey
Student Number	x18455846
Course	BSHCSD4

Month: February**What?**

Reflect on what has happened in your project this month?

Before the midpoint submission, I ran into an issue with the radio breakout. This was a big problem that stopped me from sending information across a radio frequency. I solved this problem this month with proper wiring, introduced an LED to represent sending the data, and moved the files onto my GitHub repo.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

Fixing the radios and the other changes are crucial to completing the project. This is very good for the future of the project as it demonstrates that the core functionality of the project idea is possible.

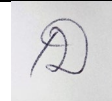
Now I will need to finish the rest of the project. Creating the database on Firebase is next on the timeline, it is challenging but using the Pyrebase wrapper could help get the metrics on the cloud more smoothly.

Now What?

What can you do to address outstanding challenges?

I need to understand the GitHub repo and Pyrebase well so I don't create problems for myself. Once this is inside the project, I will be able to create the Android application. In order to complete the rest of the project with other CAs being in the way, I will need to incrementally find information and implement it into Emanate. Supervisor meetings with Frances will be helpful in keeping me on track.

Student Signature

**March Reflective Journal****Supervision & Reflection**

Student Name	Adam Downey
Student Number	x18455846
Course	BSHCSD4

Month: March

What?

Reflect on what has happened in your project this month?

For this month, I did not complete any work on the project. This was heavily to do with upcoming CAs consuming my time.

So What?

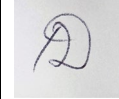
Consider what that meant for your project progress. What were your successes? What challenges still remain?

This has hurt the progress of the project, but I believe that it was necessary when juggling the other CAs. Still, it would have been beneficial to work even a small bit on the project report.

Now What?

What can you do to address outstanding challenges?

I need to manage my time more wisely. CAs are clashing with the plan of the project, but there are definitely ways around the assigned work.

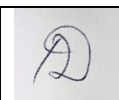
Student Signature	
--------------------------	---

April Reflective Journal

Supervision & Reflection


Student Name	Adam Downey
Student Number	x18455846
Course	BSHCSD4

Month: April

<p>What?</p> <p>Reflect on what has happened in your project this month?</p> <p>The core sections of the project have been developed this month. This includes the node data being processed and sent to Firebase, the Firebase database, and the Android application. Data is read from the database and displayed on the Android application with a node given.</p>	
<p>So What?</p> <p>Consider what that meant for your project progress. What were your successes? What challenges still remain?</p> <p>I have been able to get the node and master radio communicating with each other, bidirectionally, and have brought an API in to make the altitude metrics more accurate. I was able to write all of this to a database, and I connected my Android application to the database.</p> <p>I still need to add some additional aspects to the project. Namely, notification alerts, security features including database authentication, testing the Python and Kotlin code, and adding aesthetic elements like splash screen, colouring, and usability designs. I possibly will be putting some functionality into the buttons on the master radio.</p>	
<p>Now What?</p> <p>What can you do to address outstanding challenges?</p> <p>I need to lay out what might be concerning metrics from the data in the database, investigate the Firebase documentation for authentication with Android, create Unit tests for Kotlin and Python, along with instrumented tests for Kotlin. Making the aesthetic elements is something that I should experiment with to be pleasing to the eye. The button functionality is simple enough to implement, I just need to consider why I would need to use them in an expedition scenario.</p>	
Student Signature	

1.3. Other materials used

This hardware was bought and used in Emanate for the master radio and node. Not all the hardware was implemented but is discussed in the document why they were not feasible.



Adafruit Industr... 23/11/2021
to me ▾

← ⋮

Order Confirmation from Adafruit Industries

Thanks for shopping with us today!

**Orders are currently shipping on a 1-2 day delay.
Thank you for your
patience.**

=====

For more from our YouTube channel subscribe here:
<http://adafru.it/subscribe/>

For a weekly round-up of New nEw NEWS from
Adafruit, subscribe here:
<https://www.adafruit.com/newsletter>

Join Adafruit on Discord!
<http://adafru.it/discord>

=====

Products

2 x Lithium Ion Polymer Battery - 3.7v
150mAh[ID:1317] = \$11.90
1 x Adafruit RFM69HCW Transceiver Radio Bonnet -
433 MHz (RadioFruit)
[ID:4073] = \$19.95
2 x Adafruit RFM69HCW Transceiver Radio Breakout -
433 MHz (RadioFruit)
[ID:3071] = \$19.90
2 x Adafruit GEMMA v2 - Miniature wearable
electronic platform[ID:1222] =
\$19.90
2 x Adafruit BME688 - Temperature, Humidity,
Pressure and Gas Sensor (STEMMA
QT) [ID:5046] = \$39.90
2 x Adafruit Micro Lipo - USB Lilon/LiPoly charger (v1)
[ID:1304] = \$11.90
1 x Adafruit Perma-Proto Half-sized Breadboard PCB -
Single[ID:1609] = \$0.00

Sub-Total: \$123.45
VAT: \$35.06
DHL Express (1 x 0.40lbs) (Express Worldwide):
\$28.97
Sales Tax: \$0.00
Total: \$187.48



Adafruit Industr... 22/12/2021

to me ▾



Order Confirmation from Adafruit Industries

Thanks for shopping with us today!

**Orders are currently shipping on a 1-2 day delay.
Thank you for your
patience.**

Adafruit is closed Friday December 24 through
Sunday December 27.

=====
For more from our YouTube channel subscribe here:
<http://adafru.it/subscribe/>

For a weekly round-up of New nEw NEWS from
Adafruit, subscribe here:
<https://www.adafruit.com/newsletter>

Join Adafruit on Discord!
<http://adafru.it/discord>

=====

Products

2 x Adafruit GEMMA M0 - Miniature wearable
electronic platform[ID:3501] =
\$19.90
4 x Simple Spring Antenna - 433MHz[ID:4394] = \$3.80
1 x Solder Wire - 60/40 Rosin Core - 0.5mm/0.02"
diameter - 50
grams[ID:1886] = \$6.95

Sub-Total: \$30.65
VAT: \$15.50
DHL Express (1 x 0.33lbs) (Express Worldwide):
\$36.75
Sales Tax: \$0.00
Total: \$82.90