

Group Report

Curated by: Team 17

Kaveh Nejad(1905933),

Borislav Aleksandrov(1906490),

Bradley Todd(2005704)

and Allegra Santoro(1911599)

*CM2110 Internet of Things*

*Deliverable 2*

*Robert Gordon University - School of Computing*

*April 2021*

# Index

[**Index**](#_2pmov2l2u88p) **1**

[**Github repository**](#_w650db67ykj1) **1**

[**Group presentation**](#_jltg2isq0gny) **1**

[**Discord server**](#_e514gstrpn8n) **1**

[**Introduction**](#_nigpuoyzvh9i) **2**

[**Set up**](#_myaarfbm9l0g) **2**

[Libraries](#_ns4tj9uaicq) 2

[Software needed](#_qkazr1bclndm) 3

[Running the file](#_poyl8aflsqwg) 3

[Running Tests](#_pslftirybzr9) 3

[**Services**](#_lmmijmmaq8lc) **4**

[Discord Server](#_nsxoz2tt98li) 4

[MQTT](#_9j93iakr08sl) 4

[**Implementation Review**](#_9evx7qvxyzvb) **5**

[Design and changes](#_np5ajhc60qkm) 5

[Additional features](#_gftod72z4ybl) 5

[Features that were not implemented](#_xbt9iwsk3c86) 6

[**Conclusion**](#_4c20l4f2h8qk) **7**

# 

# Github repository

<https://github.com/WorkAtRGU/cm2110-2021-coursework-team-17-smartie-connect>

# 

# 

# Group presentation

<https://liverguac.sharepoint.com/:p:/t/Team17/EbdBSecVUKhMvCwT6R5QopIBZOC5YQEJkdK25qYnN0RmZw?e=qC3fF4>

# Discord server

<https://discord.gg/a28yCHPU>

# 

# Introduction

Smartie Connect allows you to have full control over your home from wherever you are by implementing various security and house-monitoring features at a significantly more affordable price. The product includes a variety of devices and capabilities, including smart lights, an automated heating system, entrance monitoring, CO2 and smoke sensors, smart timer and alarms. Our smart managing system connects every aspect of the user’s environment, allowing a remote and more accessible monitoring of your home.

Smartie connects utilises scripts that mimic the values sensors would retrieve in a real world environment. These values are then accessed by their respective nodes, which all then managed in the main node. MQTT is used in order to allow nodes to communicate with each other and with the discord server. The latter allows us to receive input from the user and connect them to the smart environment.

# Set up

In order to run and test the Smartie Connect prototype, as with any software, a series of initialisation steps will be required. These will be described to the user through an instruction manual that will come with our product.

## Libraries

As expected, many libraries were imported during the development of the digital twin. However, these will not need to be reinstalled beforehand on a new device as they are located in a python virtual environment. All of the libraries are shown in Diagram 1.



Diagram 1 - list of all the required libraries

## Software needed

In order to enter the server, the user will require a Discord account, which will be requested once the join link is pressed. Discord can be used both from laptop and mobile. From a computer it can be accessed either through a web browser or by downloading the desktop application. From mobile, an app is available on most app stores.

Furthermore, in order to run the code itself you will require two terminals. One terminal will be used to simulate the main raspberry pi node. The second terminal is used to run the Discord bot.

To run the commands, Python should also be installed. There are two ways to install Python on Windows. The easiest way would be to log in to the Windows Store and download the Python idle. This installs Python and adds the Python PATH to environmental variables. The other way to install Python is to download the installer from the Python website and run it. To verify the installation, type python3 into the terminal.

## Running the file

Once the project has been downloaded into a local folder, the user should enter the Window command line and enter the folder directory by using ‘cd [folder’s absolute path]’ command. To run the main file with a GUI, use the command:

‘& [absolute path to the project folder]/.venv/Scripts/python.exe [absolute path to the project folder]/code/helper\_functions/tkinter\_gui.py’ on Windows.

To run the file without a GUI, type instead:

‘&[absolute path to the project folder]/.venv/Scripts/python.exe [absolute path to the project folder]/code/nodes/main.py’

.

The main node is now running and we now need to run the discord file. To do this, type:

‘& [absolute path to the project folder]t/.venv/Scripts/python.exe [absolute path to the project folder]/code/discord/main\_discord.py’.

## Running Tests

To test our code, we run unit tests. Unit testing was used as a way to assert that a function returned the desired output. Therefore, to test our project, we once again need to use the terminal. After getting into the project directory, as previously explained, type the following command:

& [absolute path to the project folder]/.venv/Scripts/python.exe -m unittest discover

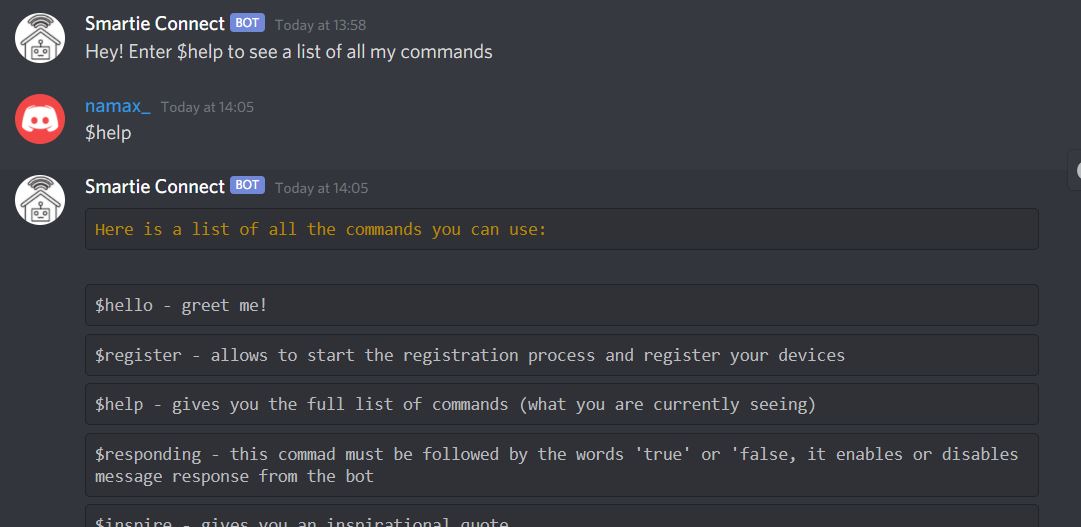
If all the tests were successful, the message “OK” will be returned, along with the number of tests run.

# Services

In order to allow communication external to the individual nodes, we adopted two web services in our design: a Discord server and Message Queuing Telemetry Transport(MQTT).

## Discord Server

As aforementioned, a discord server was implemented to allow communication between the environment and the user. This utilises a bot to reply and send messages to the user through a series of commands and keywords. These will allow the user to register and interact with their devices. As previously mentioned, an instruction manual will be provided with our product, containing the procedure to initialise the user’s devices through the server and the command needed (i.e. “$register”). However, the bot, Smartie, once run will also provide a command that once sent would return the full list of commands together with their appropriate descriptions, as shown in diagram 2.

Diagram 2 - Discord bot initialsation message and $help command

## MQTT

MQTT is a vital part of the smartie connect system. The code incorporates the RGU Computing credentials to log into the MQTT broker. Ideally in the future, all messages sent between the nodes would be sent using a communication protocol for LAN, and therefore, would be safer as no information has to leave the house. However, for this system, MQTT had to be used. All nodes and systems that use MQTT import a MQTT script written and then assign functions to be called when a message is received and also create the class with a list of topics to subscribe to.

Each MQTT message has a topic that starts with “smartie\_connect/” then has more detailed information about the scenario, in order to allow multiple users to create and manage their homes, a user ID would have to be added after each “smartie\_connect/”.

# Implementation Review

## Design and changes

Throughout the software developing process, we have encountered various design flaws that we concluded to be inadequate for our system. Therefore, many of our original plans were either altered or not included in our prototype.

One of the design changes we implemented was splitting the speaker control from the safety node. This was deemed beneficial as it allows for speakers to exist independently and to be used more for alarms, timers, and doorbells, without the need to purchase the additional safety features. However, it would be beneficial to implement a combined one, as that would be more secure in an emergency situation as the speakers could still sound without an internet connection. This could be implemented in a future combined node.

Furthermore, we originally planned to store the user information in a JSON file, however, storing them in a regular dictionary resulted to be much more convenient. It made retrieving the data through MQTT much easier and it could still be stored in a file on the company’s end since the user information is sent to our company at the end of the registration process.

Another deviation from the original design was the creation of the nodes. In our original pseudocode, the node objects were created during the registration process. However, we ended up creating them in the main file instead, after adding their information to a dictionary during the setup, as aforementioned.

## Additional features

Besides implementing the core elements required, our project also encompasses four additional features: a Tkinter display, the aforementioned Discord server, an API within the Discord server, and unit tests to test the sensors, activators and nodes.

The Tkinter display will be used to show the user all the information about the nodes in their house, as shown in Diagram 3.

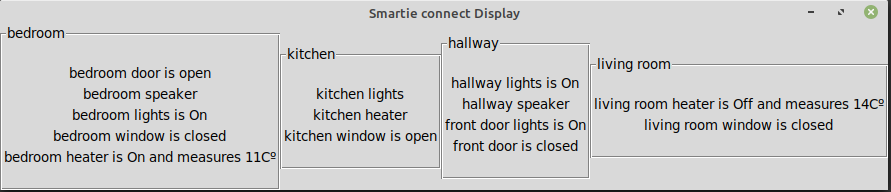


Diagram 3 - Tkinter

Any time that a device has been updated, it will send an MQTT message to the Tkinter display with new information about it, then the display will update accordingly. Developing the Tkinter display was especially difficult as any code altering the Tkinter display needs to be run in the main thread. As smartie connect relies on multi-threading, this resulted to be a challenge. In order to overcome the multi-threading issue, the main thread had to be devoted to updating the display on command. In implementing the main thread feature, a todo list was needed to store all the information that needed to be displayed, and a loop always checking it in the main thread and updating it when necessary. This feature originated as a tool to ease testing, as there are no physical objects in this model. However, once polished, it could become a useful feature in any smart home.

The use of the discord library also does fit into any of the descriptions provided for the core elements. This allowed the use of coroutines to listen for messages sent by the user, along with the abilitiy to reply and send a default message once the file is run. Within the discord, we also implemented a command(“$inspire”) that returns inspirational quotes to the user. This is done throught the use of the zenquote API, which may also count as an additional feature.

Lastly, we also implemented unittests in order to rapidly test that all the functions within the sensors, activators and nodes were working correctly. If these individual tests relied on other classes, in order to not compromise the results, fake recreations of those classes were implemented. For instance, to test the front door node, we created a fake camera object within the test file. This allowed us to confirm that when a doorbell is rang, the function would call the camera class to return a picture from the front door camera.

## Features that were not implemented

Unfortunately, either due to complications or simply lack of time, many features we hoped to implement were not included into the prototype.

One of the challenging features we wanted to implement was the ability for the discord server to be able to send messages to the user as soon as it received the request from the other nodes. This feature was challenging as the send function in the discord library needed to be awaited and the trace of the methods originated in another library (MQTT). Therefore, it was not practical to achieve without the addition of a get\_log feature.

Other features that, instead, weren’t implemented due to lack of time include: the ability to see which devices of a certain type are on or off when calling a command related to them, for example when turning on a heater, and being able to delete, modify and create new devices. If this was a real product, we would also probably develop our own messaging platform to get user input as it would be more professional and would allow us to message the user without needing an input.

# 

# Conclusion

In conclusion, our project contains all of the core elements required on top of various additional features. We challenged ourselves by using libraries and platforms outside of the university curriculum on top of already using a language we were not familiar with. We implemented a working Discord bot to communicate with our user, implemented a Tkinter display to show the device’s activity and implemented unit tests to be able to rapidly test the functionality of our code. Overall, we believe Smartie Connect to be an adequate emulation of a smart home environment and a suitable example of an Internet of Things implementation.

# 