

Individual Project – Using bots to expand the functionality of Discord

Table of Contents

| | |
|---|----|
| Introduction | 4 |
| Background | 5 |
| Why Discord? | 5 |
| Discord.py/PyPI | 6 |
| Default Python Packages | 6 |
| Other bots that try to solve a similar problem..... | 6 |
| 2020. <i>Statbot</i> | 6 |
| Mee6.xyz. 2020 | 6 |
| Code used by others | 6 |
| Summary | 7 |
| Design Implementation | 8 |
| Overview | 8 |
| Features Implemented | 8 |
| !setup (Initial Setup Command) | 9 |
| Code Breakdown..... | 11 |
| Implementation Issues..... | 12 |
| Evaluation..... | 12 |
| Registration System | 13 |
| Code breakdown..... | 15 |
| Implementation Issues..... | 15 |
| Evaluation..... | 15 |
| Contribution System | 17 |
| !addcontribution..... | 17 |
| !votecont | 17 |
| !mycontribution..... | 18 |
| !groupcontribution | 19 |
| Code Breakdown..... | 20 |
| Implementation Issues..... | 21 |
| Evaluation..... | 21 |
| Profanity Filtering | 21 |
| Implementation Issues..... | 22 |

| | |
|---|----|
| Evaluation..... | 22 |
| Secondary Features | 22 |
| Code Breakdown | 23 |
| !del | 23 |
| Error Handling | 24 |
| How the design changed..... | 24 |
| <i>Code Refactor</i> | 24 |
| Test Cases | 25 |
| !setup Testing..... | 25 |
| !addcontribution | 25 |
| !votecont..... | 26 |
| !mycontribution | 26 |
| !groupcontribution | 26 |
| Contribution System..... | 27 |
| Overall Evaluation | 28 |
| Discord API Banning and Limiting..... | 28 |
| Discord.py Rewrite | 28 |
| Future Possibilities | 29 |
| Conclusion..... | 30 |
| Personal Reflection on performance..... | 30 |
| Ethics..... | 30 |
| How to Setup and run the bot | 31 |
| References | 32 |
| Appendix..... | 33 |
| Appendix A..... | 33 |
| Appendix B | 34 |
| Appendix C | 35 |
| Appendix D..... | 36 |
| Appendix E | 37 |
| Appendix F | 38 |
| Appendix G..... | 39 |
| Appendix H..... | 40 |
| Appendix I | 41 |
| Appendix J | 42 |
| Appendix K | 43 |

| | |
|--|----|
| Figure 1: Screenshot showing Discord's Layout | 4 |
| Figure 2: Newly Created Discord Server | 9 |
| Figure 3: Discord Layout after !setup 4 (Admin Perspective) | 10 |
| Figure 4: Discord Layout after !setup 4 (Student Perspective) | 10 |
| Figure 5: Code segment which creates channels. | 11 |
| Figure 6: Shows how to edit permissions | 12 |
| Figure 7: Welcome DM by Bot | 13 |
| Figure 8: Student Registering with System | 13 |
| Figure 9: Registered_Students.csv | 14 |
| Figure 10: Code for Registration System | 15 |
| Figure 11: Using !addcontribution | 17 |
| Figure 12: Using !votecont command | 17 |
| Figure 13: Shows !mycontribution command | 18 |
| Figure 14: Embed showing !groupcontribution | 19 |
| Figure 15: On reaction event code | 20 |
| Figure 16: Group Contribution Code | 20 |
| Figure 17: Profanity Filter | 22 |
| Figure 18: Code for !del | 23 |
| Figure 19: Breakdown of permissions | 33 |
| Figure 20: Student Contribution Record Naming | 34 |
| Figure 21: Structure of Contribution Records | 34 |
| Figure 22: My contribution Code | 35 |
| Figure 23: Test for !setup 4 | 36 |
| Figure 24: Test for !setup | 37 |
| Figure 25: Test for !setup !!"£ | 38 |
| Figure 26: Test for !addcontribution @Student 1 Refactored Code -10 | 39 |
| Figure 27: !addcontribution @Student 1 Refactored Code -10 without Supervisor Perm | 40 |
| Figure 28: Test for !votecont @Student 1 -- Added Python script v2 – 14 | 41 |
| Figure 29: Test for !mycontribution | 42 |
| Figure 30: Test for !groupcontribution | 43 |

Author: Kieran Parnell 1613910
Supervisor: Frank C Langbein

Before the report, I just wanted to point out that I have a visual impairment. Most of the time the screen is 300-400% scaled, therefore formatting of a large document like this can be an issue. I apologise in advance if the formatting is slightly off.

Introduction

The main aim of the project was to create a system that would help with group monitoring- where student contributions were recorded, and reports could be automatically produced. Some other aims were to introduce some profanity filtering, automated setup of the system and an automated permission system along with a login system to complement this. My aim from the initial plan was to create this system by taking advantage of Discord (2020) and their API (Discord Developer Portal. (2020)).

For this report, I will assume that the reader has some understanding of what Discord is and a working knowledge of Python, Programming Technologies and a general level of knowledge for creating systems.

Discord provides a way for users to collaborate and communicate, like Skype (Skype.com. 2020.) or Microsoft Teams (Microsoft.com. 2020.), but offers far greater functionality. Discord uses a server system, where anyone can create a server and customise it, these servers can currently support 250,000 people per server. These servers have two different channel types, Text Channels and Voice Channels.

Here is the general layout of a Discord Server:

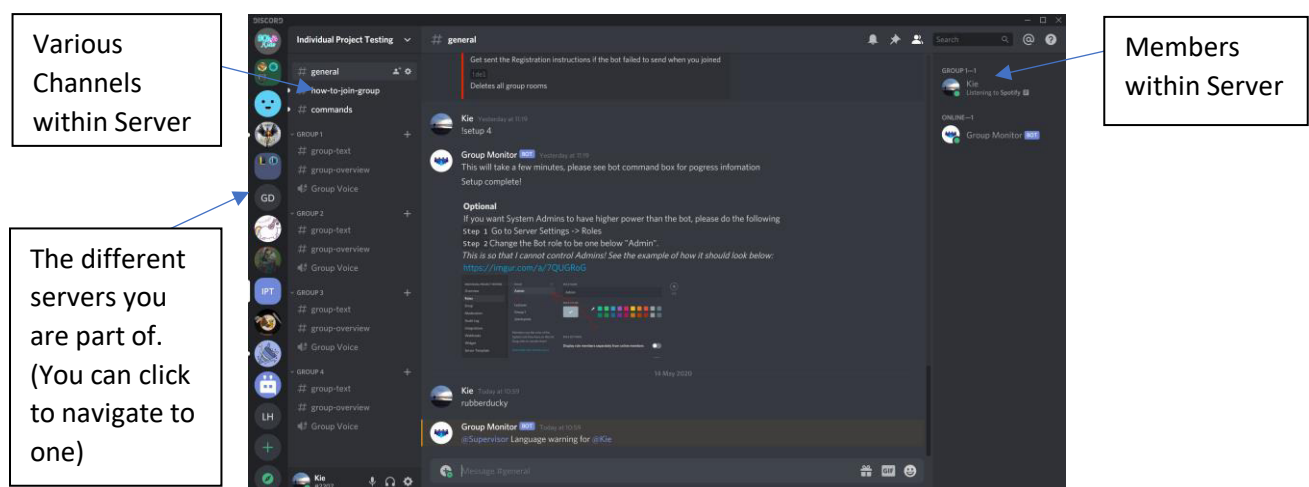


Figure 1: Screenshot showing Discord's Layout

Furthermore, Discord also allows for Bots, which provides extra functionality and allows users to take advantage of the API provided by Discord. As such, we can code these Bots to monitor student participation, to read messages to check for profanity and to create a login system via the bot, where until you have logged in you won't get permissions for that Discord server.

Intended Features

- Implement Monitoring of Group work.
- Simple natural language processing.

- Registration system for someone to join the Discord server, ensuring they are a verified student.
- Automated permission system
- Logging of contribution system actions.

This project had many different features that needed to be designed and implemented, as such I decided I would work and create them in an agile approach. Where, for each feature, I would focus solely on that single feature, design it, implement it, test it, and then move onto the next desired feature.

For this project, there are a significant amount of assumptions made. Firstly, because we are using an API it is assumed that the API would not have some major rewrite or updates that would damage/break the system that I've created- moreover, this would also assume if this project were to be deployed some level of maintenance would need to occur to ensure that the code was at least at the minimum supported level for the current API. Secondly, we are assuming that the Discord Service and API itself is stable enough to have a system like this run- if the API were to break, or the service go offline then this system would not function at any capacity. Moreover, the Discord API deals with personal data, as such we are assuming that the API is secure enough to protect users information, however there is no real way of me confirming this as they do not present the source code for the API or have any documentation on how the backend of Discord works.

To summarise, the outcome of this project is to have a system which will assist group monitoring, by recording participation levels, ensuring profanity is not used and have an elegant solution for student registration and automatically control their permissions within the Discord server. The aim was to achieve this by using the Discord API and Python.

Background

As mentioned previously, I will be using Python to manipulate the API for Discord. Here I will go through the background and some of the Technologies I have used/explored for this project. I'd also like to point out that before this project I had no experience with the Discord API.

Why Discord?

At first, I explored services like Skype, which would offer the communication side of what my project aim needed but provided none of the features I was looking for, Group monitoring, profanity filtering and a registration system for students.

Microsoft Teams was also another option, it too allows for bots and has an API, however, after further research it seemed that the API for Microsoft Teams was more based around Conversational Bots or bots that would automate tasks like graph creation. In comparison to Discord's API, the Microsoft Teams API was arguably more restrictive and would allow for less functionality. I found that Microsoft Teams was intended to be a Chat-based workspace that included the Microsoft Software Suite, it targets more towards collaboration and project development. Microsoft Teams is very similar to Slack in terms of what their target demographic is.

Discord has established itself as an all-in-one solution for collaboration no matter the area, from University work, to gaming. Discord provides a vast array of features that accommodate for all, it allows one-to-one or group voice calls, screen sharing/video calls for up to 50 people at once, text and voice channels, a very open API that would hugely extend functionality, most importantly, Discord allowed for bots to control permissions for servers.

Author: Kieran Parnell 1613910
Supervisor: Frank C Langbein

Discord.py/PyPI

PyPI (2020.) or Discord.py is an API wrapper for Discord, that aims to make it easier to communicate with the Discord API. It uses async and await and is updated on a regular basis. Using PyPI makes it quick and easy to start communicating to the API using Python, it allowed me to focus on the implementation of system and learn the API structure, rather than having to deal with the way in which the API and Python would communicate to each other.

Discord.py also includes a module for logging, this logging module records errors reported by the API. The module would provide information regarding API limiting/bans and some errors that would occur when interfacing with the API.

Default Python Packages

For the project I used a vast array of packages that are built into python by default, from datetime packages to .CSV handling packages.

Other bots that try to solve a similar problem

From research, there are Discord bots out there that have similar features to the bot I have designed. However, all these bots are written on the old Discord.py Branch (I will go into the Discord.py Rewrite later). This meant that a lot of the code for these bots would not assist me in the understanding of the API. Furthermore, from my research it seemed that although there were other bots that would provide one feature that was in my desired features for a bot, I could not find a bot that had all the desired features that my bot would have.

2020. Statbot.

2020. Statbot is an activity tracking bot for Discord, using the old Discord.py branch it provides statistics on users, showing how many messages they have contributed, time in voice rooms and a leader board showing who has the most activity. Although for my project I would not be simply be counting the messages and time spent in voice rooms and instead using a voting system to decide who contributed what, it gave me an idea of what API requests could be accepted, even though it was written on the old Discord.py branch.

Mee6.xyz. 2020

Mee6.xyz. (2020) is a Multifunction bot, providing many features, most notable for my project though was the profanity filtering. Again, although it was written on the old Discord.py branch it gave me some idea of how to approach profanity filtering when dealing with Discord. Looking at large-scale bots like Mee6 may hold some ideas on how to deal with upscaling a bot to a large user base.

Code used by others

The only code that I am using that is created by others are the Python packages mentioned previously. The bot I created, including all the various functions is original code created by me. It was quite difficult to find any code by others as I was using the new Discord.py rewrite, where all other bots were using the old Discord.py branch. The only time I used code by others was to get an understanding of how the API worked, I would take the old branch of code and then spend the time converting it to the new branch, this gave me an idea of the structure, but I decided not to use this code in my bot as doing the conversion myself did make the code very messy and hard to understand from an outsiders perspective.

In the coming years more development using the latest API rewrite may occur and as a result more code using the rewrite available online, but right now, it seems that I am one of a few people

Author: Kieran Parnell 1613910
Supervisor: Frank C Langbein

developing bots on the Discord.py rewrite and furthermore, the only one who has created a Discord.py bot to serve the purpose of a contribution system for students.

Summary

To summarise, Discord was chosen because of its feature rich environment and how it has a very open API which allows for many possibilities. When looking at what others have done to solve a similar problem, one thing became clear, that although there were solutions for activity tracking, or profanity filtering, there was not one bot that did them all. Moreover, from all the research I did I, I did not find a bot that had a student registration system like I intended.

Design Implementation

For this project, there are several features that have been implemented, as such, I will break down each major feature and explain the important parts of the code for each.

Overview

The implemented system includes a registration system that will allow the user to log into the server and then be given the respective roles within the server. It has a contribution system, where student contributions are recorded and are presented in neat reports, showing information on hours spent per contribution, total hours, and the date/time the contribution was made. Moreover, contributions can be made either by students, where student must vote on a contribution within their group in order to get it approved, or Supervisors can manually push through a contribution for a student without the need of a vote. Because Discord doesn't have this contribution system by default, Discord does not have a log of contributions, therefore I've implemented a simple logging system which records what supervisor pushed through what contribution, this is more of a proof of concept and is not fully implemented. Finally, a profanity filter was added, where the bot takes in a .txt file containing blacklisted words, if any of these words are used within the server a Supervisor is notified of a possible infraction being made within the server.

For all the below implemented commands and features you can control the bot through the public text rooms, unless advised otherwise. I opted to use a command structure as using a system that used Natural Language processing is very difficult and most implementation of Natural Language Processing just do not work well enough for a system like this.

Features Implemented

For each feature/command I will provide an explanation of what it does, then if that feature has any important code snippets. Following this I will talk about issues I had implementing this feature and then finally the evaluation of that feature. I will have a completely different section for testing as I believe this will be the easiest way to explain the functionality of the entire system. I have moved from the original structure because the nature of Discord is very visual and difficult for new users, therefore I have had to provide many screenshots showing how the system works.

How this section is structured:

- Feature/System
 - o Description of Feature/System.
 - o Snippet of any important/ noteworthy code.
 - o Evaluation

**I repeat this structure for each Feature/System. Testing is a different section of the report.*

Author: Kieran Parnell 1613910
Supervisor: Frank C Langbein

!setup (Initial Setup Command)

In Discord, when you create a Server, you must set up the rooms manually, this including all the desired permissions, the role hierarchy, and the permissions for each group. This can take several hours to configure and understand, as such I've created a setup feature that allows the bot to do all this for you, removing the issue of human error and the need to understand the full permission system of Discord.

Here is what a newly created Discord Server looks like:

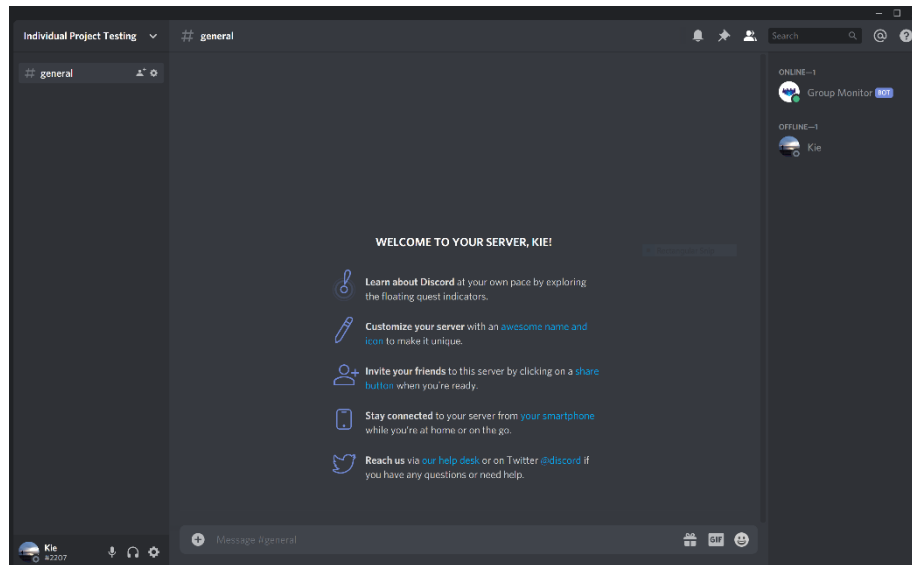


Figure 2: Newly Created Discord Server

As you can see in *Figure 2* there is only one text room "general" and there are no voice rooms. This is where normally you would have to spend the time creating and setting the permissions for each room and permission groups for users. By using my bot, you can enter "!setup 4". "!setup" is the command, the "4" is the amount of groups you wish there to be. If you would like there to be 12 groups, you would use "!setup 12". This deals with creating all the groups, managing the permission and creates the permission groups for you. It makes the setup process very quick compared to someone who is brand new to Discord and has no understanding of how to configure Discord. Below is *Figure 3* that shows what Discord looks like after using "!setup 4".

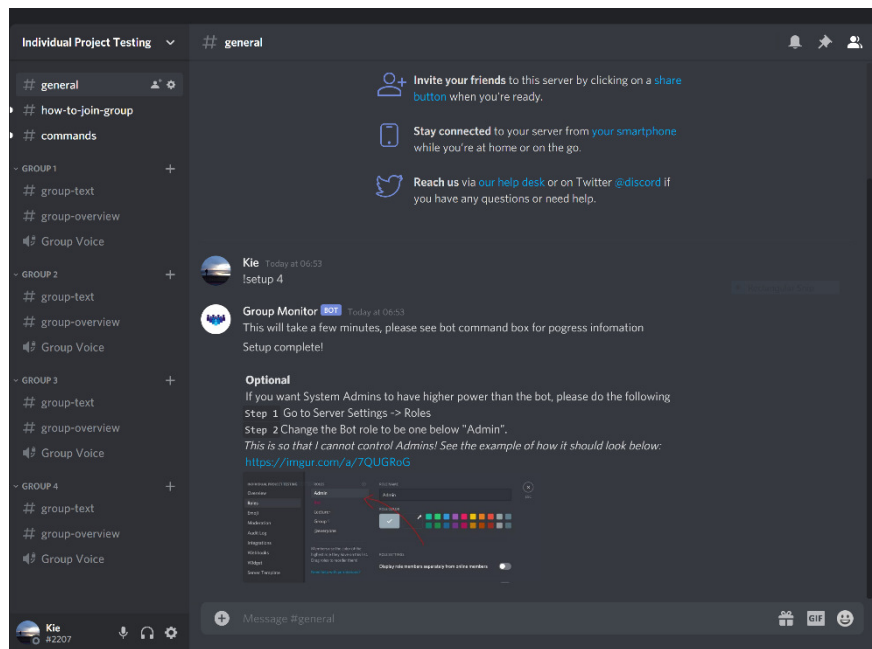


Figure 3: Discord Layout after !setup 4 (Admin Perspective)

On the left-hand side of *Figure 3* you can see all the rooms that have been created. By using the command, the bot creates all the Desired group areas, dealt with the permissions, and also provided some extra details on how to customise the server. Moreover, it created two unique rooms “How-to-Join-group” which provides information on how to register with the bot. The second unique room is “commands”, which provides a full list of commands that the bot will respond to.

Figure 3 is taken from the Administrator view, which allows you to see the entirety of the system. From the student’s perspective, they would only be able to see their respective group area. This is achieved with a complex permission system that I implemented into the bot; I will go over this in greater detail later.

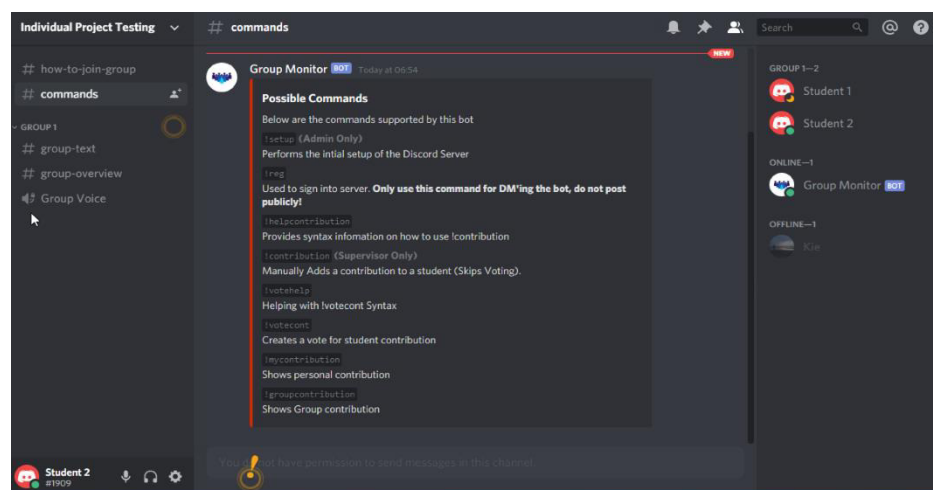


Figure 4: Discord Layout after !setup 4 (Student Perspective)

Figure 4 Shows the view from the students perspective, as you can see they are only able to see their own group area, they do not have access to view or type in the Group 2, 3 or 4 area as seen in *Figure 3*.

Code Breakdown

```
249 #Creation of Group/s and thier respective perms
250 print("Creating Groups")
251 count = 0
252 while count != int(m_in):
253     count+=1
254     name = "Group "+str(count)
255     await message.guild.create_category(name)
256     progresscounter+=1
257     print("Progress: ",progresscounter,"/",progressbar)
258     category = discord.utils.get(message.guild.categories, name=name)
259     await message.guild.create_text_channel("Group Text",category = category)
260     progresscounter+=1
261     print("Progress: ",progresscounter,"/",progressbar)
262     await message.guild.create_text_channel("Group Overview",category = category)
263     progresscounter+=1
264     print("System sleeping for 10 seconds. (Prevents Discord API ban or Limiting)")
265     time.sleep(10)
266     print("Progress: ",progresscounter,"/",progressbar)
267     await message.guild.create_voice_channel("Group Voice",category = category)
268     progresscounter+=1
269     print("Progress: ",progresscounter,"/",progressbar)
270     await message.guild.create_role(name=name)#Creates roles
271     print("System sleeping for 10 seconds. (Prevents Discord API ban or Limiting)")
272     time.sleep(10)
273
274
275
```

Figure 5: Code segment which creates channels.

Figure 5 is a segment of code that creates the text channels, voice channels and the Category for each of the groups. This is just how the channels are created, this is not the code that modifies the permissions. The While loop will loop until the Counter is equal to the requested number of groups. I am using this segment of code to show the API requests, each “await” seen in Figure 5 is an API request.

On line 257 in Figure 5 is code that deals with the progress bar, this is just shown in the command prompt where the bot is running to show the Administrator that the bot is still doing work. This is important as on the last line in Figure 5 is time.sleep(10), this was implemented as the bot was sending too many API requests and resulting in an API ban. As such, implementing a progress bar for the Administrator to see was necessary, it shows when the system is sleeping to avoid the API banning.

Implementation Issues

There were a few problems with implementing this !setup command, firstly is how !setup requires a huge amount of API requests, which meant adding `time.sleep(10)` throughout the setup code to prevent bans. For the !setup command it takes $(13+9*\text{Amount of Desired groups})$ API requests, therefore if I wanted to create 15 groups, it would be $13 + 9 * 15 = 148$ API requests. These 148 requests would be made within seconds without the `time.sleep()` and result in an instant API ban.

A secondary issue was how the API was written; it was counter-intuitive when dealing with editing permissions.

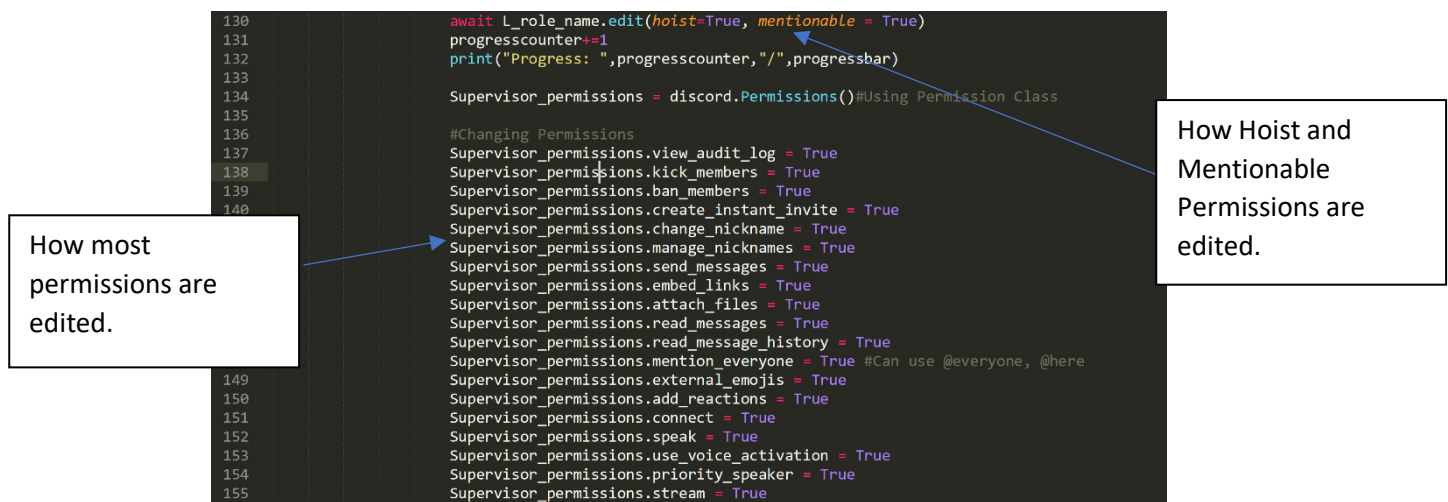


Figure 6: Shows how to edit permissions

In Figure 6 permissions are edited, the issue is that most permissions are edited in this format:

`student_permissions.embed_links = True`

However, permissions Mentionable (Allows others to @Mention that role) and Hoist (Displays the role separately from others) are edited in a different way:

`await L_role_name.edit(hoist=True, mentionable = True)`

This caused a lot of confusion during implementation and took many hours for me to understand why the API was not responding when I was trying to edit Hoist and Mention Permissions using the other format. In my opinion, these permission for the API were created by different people and rather than ensure consistency they decided to implement them in this way.

Evaluation

In the initial plan, there was no mention of having a command like "!setup" this was an oversight, at the planning stage; I neglected how complex Discord was in terms of its permissions and structure, as such when I realised this during implementation I decided to implement this "!setup" command, the idea being that it takes control out of the Administrators hands, reducing the opportunity for human error. Moreover, through development it became clear that the initial setup of the Discord server would be essential to how the rest of the system would work.

The "!setup" command does have some issues, like how intensive it is on the API, it by far makes the most API requests out of all the created commands and did cause me to get API banned on various occasions. As mentioned previously, I implemented `time.sleep()` throughout the code, to slow down the rate that API requests made. From a time standpoint, this isn't ideal. If `time.sleep()` wasn't used the initial setup it would take a matter of seconds, but then resulted in an API ban.

Author: Kieran Parnell 1613910
Supervisor: Frank C Langbein

Whereas now, the system will take minutes to finish the setup, this varies depending upon the number of groups you wish to make. If you wish to create 20 groups, then the system would take approximately 20 minutes to finish the initial setup.

I believe implementing “!setup” was essential to ensuring the smooth running of the bot for other commands, although it does take time to run, it would still take much longer for someone to manually setup the server themselves and understand how Discord permissions and structure works.

Registration System

As mentioned previously in the !setup section, I showed how groups were only able to interact with their own group and are unable to see other groups. In order to achieve this, I needed to implement a registration system, where people would login and then be given their respective group role within the Discord server.

When a Student joins the server, they are sent this DM by the bot:

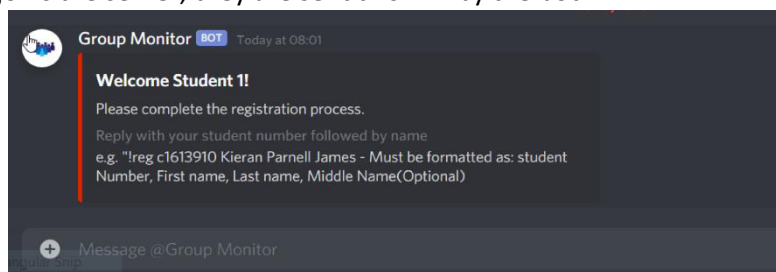


Figure 7: Welcome DM by Bot

Figure 7 shows the DM that a student would receive when they join the server, it provides information on how to register. In this example, the students name is “Student 1”, therefore to login in they would use !reg StudentNumber Student 1. This would then log them into the system, see below for an example:

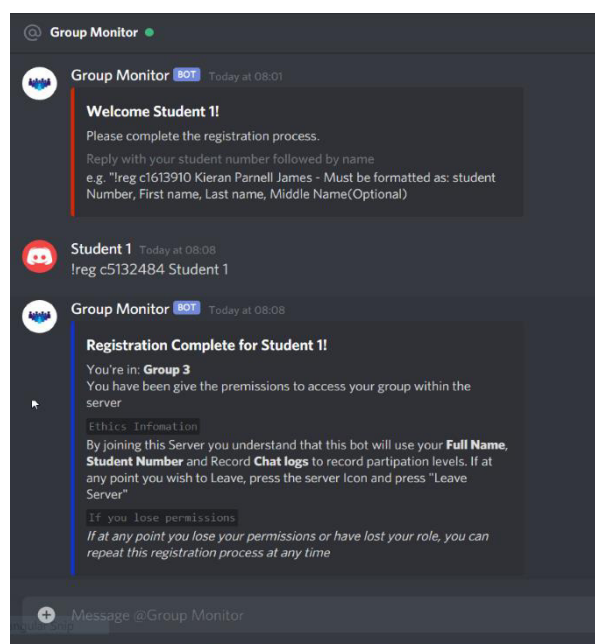


Figure 8: Student Registering with System

In Figure 8, Student 1 has registered and the bot has therefore given the respective role to the

Author: Kieran Parnell 1613910
Supervisor: Frank C Langbein

student within the server, in this case they have been given the “Group 3” role. The registration system works by comparing what the bot is messaged and what is stored within the .CSV, in this case it is the Student name and their student ID. I am aware that there is no password, but for simplicity I kept it as the student number. You could simply switch the student number for a password if desired in the .CSV. The .CSV is named “Registered_Students.csv”.

| A | B | C | D | E |
|----------------|------------|-----------|-------------|----------|
| Student Number | First Name | Last Name | Middle Name | Group ID |
| c1613910 | Kieran | Parnell | James | !1 |
| c16123123 | Joe | Jones | | !2 |
| c16332 | Adam | Jenkings | Jones | !1 |
| c1615677 | David | Parnell | | !1 |
| c1231233 | Poppy | Brown | | !le |
| c5132484 | Student | | 1 | !3 |

Figure 9: Registered_Students.csv

Figure 9 is how the .CSV is structured. You can see that all the system does is compare the message content sent by the student with the contents of the .CSV. The bot knows what role to give the student by looking at the “Group ID” within the .CSV, “!3” would indicate the bot to give the “Group 3” permission.

Supervisors can also log into the system, they log in the exact same way as students do (As shown above) but instead of the bot giving a group role, they are given “Supervisor”. The bot knows to do this by looking at the .CSV, instead of “!3” it contains “!le”, which is used to indicate that this person is part of the supervisor role. Within Discord, the supervisor is given higher permissions over students. Please see **Appendix A** for a breakdown of permissions.

Although Supervisors roles and Student group roles can be given through this registration system, I decided not to make it possible for someone to login in as Administrator. Instead, the Administrator that initially created the server is the only person who can give others Admin. This was because of Security concerns, as anyone with Admin would be able to completely delete the server if they wished.

Additionally, this is all done in a direct message conversation with a bot for security. The permissions are set up in such a way that a Student is unable to type in the Server until they have registered with the bot, thus preventing the student from accidentally trying to log in through a public text room within the server.

Author: Kieran Parnell 1613910
Supervisor: Frank C Langbein

Code breakdown

```
414 #Prevents registration in public rooms
415 if not isinstance(message.channel, discord.DMChannel):
416     await message.channel.send("You can only use this command via DM. Please use the command \!senddm")
417 else:
418     #Compare with .CSV Input
419     hit = 0
420     with open('Registered_Students.csv', 'rb') as csvfile:
421         count = 0
422         x = []
423         for each in csvfile:
424             x.append(each.decode("utf-8"))#Read in CSV and load to List
425         for each in x:
426             #Lots of formatting for comparing CSV to User input
427             each = each.replace(" ", "")
428             each = ''.join(each.split()) #Remove Double whitespace
429             each = each.lower()
430             groupid = each.split(" ")
431             groupid = groupid[1]#Holds groupID for later
432             groupid = groupid.replace("!", "")
433             each = re.sub("!", "", each) #ends before !
434             each = each.rstrip()#remove whitespace
435             target_server_id = 673446119233028107 #ID of the server STATI
436             #If UserInput Matches a CSV row.
437
438             if m in each:
439                 hit = 1
440                 #Role Assignment
441                 try:
442                     if groupid == "le":
443                         server = client.get_guild(target_server_id)
444                         role = discord.utils.get(server.roles, name="Supervisor")
445                         member = server.get_member(message.author.id)#finds member in server
446                         await member.add_roles(role)
447
448                 #Send reply using Embed
449                 await message.author.create_dm()
450                 t1 = ("Registration Complete for " + str(message.author.name) + "!")
451                 embed = discord.Embed(title=t1, description = "You have been given Supervisor Permissions", color=1127128)
```

Figure 10: Code for Registration System

In Figure 10 line 427 – 436 deals with formatting of the user input to get it ready to compare to the .CSV file. Then line 442-446 will give a role depending upon what is found in the ID field, in this code segment it deals with giving roles to supervisors.

Implementation Issues

There were no major implementation issues when creating this system, the biggest concern was security. Like mentioned before, I have made it so that it is not possible for users to log in via a public room, however, what is uncertain is how secure the Discord API is. Maybe dealing with student ID is not a great idea and instead I should implement a system where students are given an encrypted student ID and then once the bot receives that ID it has the key to see the actual student ID, this avoiding the concern of the student ID being leaked from the API.

Evaluation

In the initial plan I mentioned that I would like a registration system like the one implemented, but again I failed to appreciate the complexity and importance that this was done correctly in my initial plan. The registration system easily took the most time to implement, I had to fully understand how Discord worked, pre-plan who had what permission and ensure there were minimal bugs. The registration system assigns roles in the Discord server, as such, these roles would control what a user could do and what they could see within the server. Moreover, this registration system is used for both Students and Supervisors, so ensuring that the correct roles were given were essential.

Admittedly, this registration system is functional but it is not pretty, a solution to this would be to use some sort of web dashboard that a user is given in order to register, rather than doing it through Bot direct messages. From a code perspective, the registration system could become computational complex if the user base were to grow. For example, if there were 100,000 students, the registration system would take time to respond and give the correct role. This is because of the way I implemented the system, it works by searching through all of the people in the registered_student.csv, if there were 100,000 entries in this .CSV then it would take some time to

Author: Kieran Parnell 1613910

Supervisor: Frank C Langbein

find the user. If this bot were to be deployed to a scale this large using something other than a .CSV would be necessary.

Contribution System

I have implemented a contribution system, where students can add their own contributions that will then be recorded by the bot and presented in reports. Supervisors can push a contribution through for a student if they deem it necessary, but if a student wants to add a contribution then a vote is held by the bot. For testing reasons, the vote is set to four people or higher must agree with the contribution, but this could easily be changed to a percentage-based agreement rather than four votes.

!addcontribution

"!addcontribution" is what Supervisors use to manually add a contribution by a Student, it does not require a vote and will be added immediately. If a Student were to try and use this command the bot will inform them that only Supervisors can use this command.

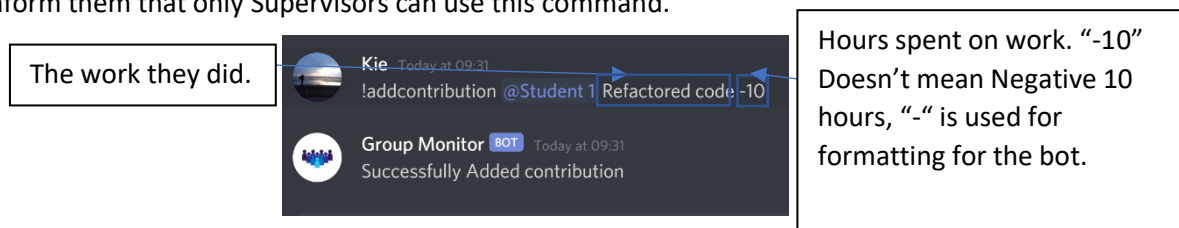


Figure 11: Using *!addcontribution*

In Figure 11 you can see how the command is used, it must follow the structure of:

"!addcontribution @MentionStudent WorkDone -HoursSpentOnWork"

Assuming the syntax is correct, and the bot replies with "Successfully Added Contribution" then this contribution will be added. Contributions are stored in separate .CSV files for each student, the name of the .CSV is the students unique Discord ID that is linked to their Discord account, all these files are stored in the folder "contribution_records"(See **Appendix B** for an example).

!votecont

"!votecont" is what students will use if they wish to add a contribution, It must be done via this voting command- this vote can either be for themselves or another student.

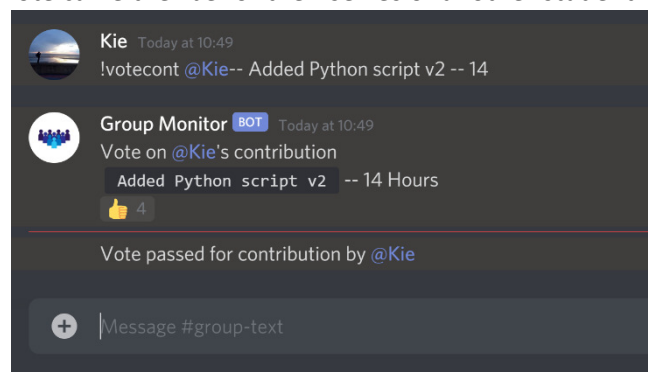


Figure 12: Using *!votecont* command

Figure 12 shows how the voting works for students. Firstly, the student uses the command to create a vote. The vote is then created by the bot. The voting is done by using a yellow thumbs up button, as seen above. Once this has reached at least 4 votes, the vote is considered passed and the contribution is then added to the respective .CSV. If the vote is not passed, then the contribution is not added.

Author: Kieran Parnell 1613910
Supervisor: Frank C Langbein

!mycontribution

This will show the students personal contributions so far.

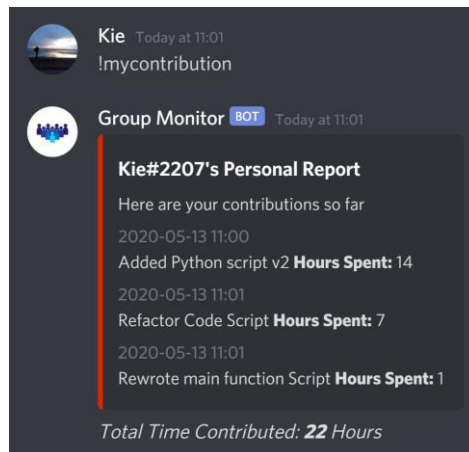


Figure 13: Shows !mycontribution command

Figure 13 shows the personal report for a student, this personal report will only show the contributions for the person who typed the command. As you can see in the above figure, the embed shows the date that each contribution was made, the actual contribution made, and the time spent on that contribution. After the embed, the total amount of time contributed over all contributions is displayed.

Author: Kieran Parnell 1613910
Supervisor: Frank C Langbein

!groupcontribution

Unlike “!mycontribution”, “!groupcontribution” will post two embeds or three in the situation that someone in the group has not contributed at all. The first embed will detail all the contributions made to the group including the hours for each contribution and who it was contributed by. The second embed displays the total contribution time for each student. Finally, the third embed displays any students that have not contributed for that group, in the case all students have contributed, embed three will not be posted.



Figure 14: Embed showing !groupcontribution

Author: Kieran Parnell 1613910
Supervisor: Frank C Langbein

Code Breakdown

“!votecont” from a code standpoint only creates an embed message, sends it and adds a thumbs up reaction. However, the real technicality starts when someone presses the thumbs up on the embed:

```
661 #--- Deals with counting votes ---#
662 @client.event
663 async def on_raw_reaction_add(payload):
664     #Ensures Reaction is on bot message
665     if payload.emoji.name == "👍":
666         channel = client.get_channel(payload.channel_id)
667         message_for_count = await channel.fetch_message(payload.message_id)
668         reaction = get(message_for_count.reactions, emoji=payload.emoji.name)
669
670         #If reaction is 4 or greater then add as official contribution
671         if reaction.count > 3:
```

Figure 15: On reaction event code

Due to API limitations I had to come up with a method that was not going to cause too many request to be made, originally I was going to code it so that every few seconds it would retrieve the message and see how many reaction were on it. Instead, I managed to create a system which used an “@client.event” where if a new reaction was added, it would retrieve the message with the newly added reaction, check it is the correct reaction and check if the reaction count was greater than three.

```
591 #Loops prints all contributions for people in the same group as message author
592 if "Group" in str(message.author.roles):
593     #for all members in same server as message author
594     for member in message.guild.members:
595         individualCounter = 0
596         #loop through all members and thier roles in server
597         for each_role in member.roles:
598             #if the message author and found user are in same group
599             if each_role == GroupHolder:
600                 my_id = member.id #holds USeR Discord ID
601                 #Loads users files to find any contributions
602                 filename = "contribution_records\\"+str(my_id)+".csv" #Stores location of
603                 try:
604                     with open(filename, 'r') as myfinder:
605                         for items in myfinder:
606                             items = items.rstrip()#Removes blank newlines
607                             if items == "":#used to do nothing in loop if there is no char
608                                 continue
609                             else:
610                                 items = items.split(",")#sperates note and the date/time
611                                 embed.add_field(name=member.name+" "+items[1], value=items
612                                 individualCounter += int(items[2])
```

Figure 16: Group Contribution Code

For Figure 16 there is quite a bit going on, I will break it down loop-by-loop:

Loop on Line 594 – This loops through all people within the same server as the person who used the “!groupcontribution” command.

Loop on Line 597- Then for each member within that server, loop through the roles that they have associated with them.

If Line 599 – If the found person is in the same group as the person who used “!groupcontribution”. (This ensures that only people from the same group are presented in the report).

Line 603 -> 612 It will Try to open the contribution files for each student within that group, if an error is returned with not being able to find the file, then this indicates that the user has no contributions yet. “if items == “”:” on line 607 was used to deal with how each entry in the .CSV has a

Author: Kieran Parnell 1613910
Supervisor: Frank C Langbein

blanks line before it, this happens because of how Python deals with appending to .CSV's. "if items == "":" meant that that loop would do nothing and therefore not add an empty entry to the report.

Line 610->612 Adds the student contribution information to the embed and counts uses a counter to count up the number of hours for the total contributed hours.

The code for "!mycontribution" is simpler as you only have to find the information for the message author, as such the file for the author is loaded and added to the embed. See **Appendix C** for this code snippet.

Implementation Issues

Although there were no major issues with implementing the contribution system it did require a deeper understanding of the Discord API, therefore in order to create this contribution system I had to spend some weeks learning and testing the API for what it could and could not do. The only noteworthy issue with implementation was how the Discord API searches for users, it seems to do this linearly, where for large user bases this could become an issue.

The first implementation of a contribution system included having the bot record hours spent in voices rooms and the total numbers of messages sent by users, but again this caused too many API request to be made. I also made the decision that hours spent in voice rooms and message sent would provide any valuable information to the supervisor and that the contributions made through the voting system help much more value.

Evaluation

Like other functions, in the initial plan I made it clear that I intended to create a contribution system like this, but I again failed to appreciate its complexity. The system works from a functional perspective, but if this system where to be upscaled there would be issues. Right now, the contribution system works by searching the entire server for other users that have the same role, then printing their contributions in a neat report. The issue here is how it needs to search through everyone in the server, a easy solution to this would be locally storing which users are in which group in the Discord server and then reading the file to get their User IDs, rather than searching the entire server. The contribution system does require more error checking, I'll go over this issue later.

To summarise, the contribution system here can accept contribution votes by students or have a contribution pushed through by a supervisor, then these contributions can be seen in neat reports that the bot produces as embeds. You can see two different reports, either just your own personal contribution to the group or your entire groups combination.

Profanity Filtering

I implemented a simple feature, where for each message sent within the Discord server, the bot would grab the message content and check if any word in the message content contained any banned words. These banned words are defined in the word_blacklist.txt, which is just a simple .txt. If the message did contain a banned word, a Supervisor is alerted.

Author: Kieran Parnell 1613910
Supervisor: Frank C Langbein

For demonstration, I have put the word “rubberducky” in the word_blacklist, this is the bots response:

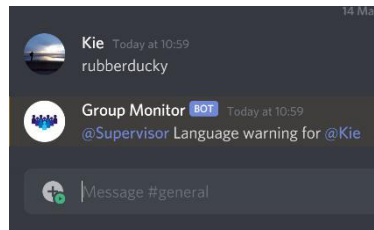


Figure 17: Profanity Filter

As you can see in *Figure 17* the bot “@Supervisor” by doing this the Supervisor gets a direct notification. I decided to make it that the bot did not delete the original message and instead leave that decision to the Supervisor.

Implementation Issues

The profanity filtering that the system currently uses is very simple and easy to implement, I did however have issues with previous attempts at dealing with profanity. Originally I wanted some natural language processing, but after some time spent with natural language processing it became clear that it was a very complicated area and non-one had perfected it yet. Moreover, I opted for this more simplistic system because I was concerned about the project becoming about natural language processing rather than a contribution monitoring system.

Evaluation

There is not too much to talk about with this profanity filter, it does not do much other than flag a Supervisor. During implementation, I tried some natural language processing tools, but found that all have their own problems, mostly that none can understand context correctly. Therefore I opted to only Notify the Supervisor and not delete the message because in my opinion having a profanity filtering system that is overly aggressive is much worse than having a passive profanity filtering system.

Secondary Features

There are some other features that I have implemented that I would consider outside the scope of the original initial plan, but I would like to bring some attention to. Firstly, I implemented a “!help” that would produce a list of possible commands that the bot would respond to. Secondly, I implemented two commands “!HelpAddContribution” and “!HelpVoteCont” that provide more detailed syntax information on how the command should be used/structured.

Within Discord, users can have their privacy settings set to “Not Receive messages from people within a server”, this means that the bot would not be able to message them upon joining. In the “how-to-join-group” it provides information and points out that people will need to disable this feature to use the bot. After they’ve disabled this privacy feature, they can then use !senddm, that will open a direct message conversation between the user and the bot.

When creating and testing the bot, there were many instances where I would need to delete all the text/voice rooms and the roles because they were not setup correctly. At first, I did this manually, but quickly realised that it was taking far too much time and as such I implemented two commands to help with this process “!del” and “!delrole”. “!del” deletes all text and voice rooms within the server, where “!delrole” deletes all the roles within the server. Again, I had to be very careful using these commands as they require a large amount of API requests to complete their

actions. To help prevent API bans when using these, I used `time.sleep()` to reduce the rate at which API requests were made.

All these secondary features can be used by anyone, including the functions that delete all rooms and delete all roles. I am aware that this could be exploited by a student, but if the system were to be deployed these would either be removed from the code or only be usable by an Administrator. I only kept these features in the system to help show what tools I used to help develop the system. Moreover, the “!addcontribution” command can only be used by a supervisor, so this shows that it is easy and possible to restrict commands to only be used by users who have the specified roles.

Code Breakdown

!del

```
327 #--- !del (testing Only) ---#
328 if message.content.lower().startswith('!del'):
329     try:
330         #Deletion
331         hold = []
332         count = 1
333
334         #Deletes help text area
335         help_text = discord.utils.get(client.get_all_channels(), name='how-to-join-group')
336         await discord.TextChannel.delete(help_text)
337         print("System sleeping for 10 seconds. (Prevents Discord API ban or Limiting)")
338         time.sleep(10)
339         #Loops and deletes all Group areas
340         print("Deleting All Groups")
341         for guild in client.guilds:
342             for channel in guild.categories:
343                 hold.append(channel)
344                 await discord.TextChannel.delete(channel)
345                 group_cha_del = discord.utils.get(client.get_all_channels(), name='group-text')
346                 group_ov_del = discord.utils.get(client.get_all_channels(), name='group-overview')
347                 group_voice_del = discord.utils.get(client.get_all_channels(), name='Group Voice')
348
349                 await discord.TextChannel.delete(group_cha_del)
350                 print("System sleeping for 10 seconds. (Prevents Discord API ban or Limiting)")
351                 time.sleep(10)
```

Figure 18: Code for !del

The code is simple, it loops through all of the channels for each group and deletes them, each delete is an API request, therefore if there are 80 rooms, that will be 80 API requests. This is why the use of `time.sleep()` is so important when dealing with high API request functions.

Line 335 of Figure 18:

`“help_text = discord.utils.get(client.get_all_channels(), name='how-to-join-group')”`

Here, we are getting the unique ID for the room “how-to-join-group”. In the Discord API, you cannot delete a room just by using the name, you must first request all channels, find all the servers that match the name you want to find, then using that name you can get the unique ID for that channel. This is because in Discord you can have channels with the same name, therefore you can only delete channels using their unique ID.

Error Handling

All the commands explained have some level of error handling to ensure that an error does not completely crash the bot, however there is a lot of improvements that could be made. Due to time constraints, I was unable to add a good level of error handling, for instance, if someone uses the incorrect format for adding a contribution this contribution may still be added, but then when the bot tries to produce a report it will get an error, this being because the structure of the contribution was not maintained (This will become clear in the tests).

If a bad request is made to the Discord API it will not respond, it will not provide an error in most cases. This makes it very difficult to add a good level of error handling, in the best case the Discord API will respond with "Bad Request" but provide no further details. I would assume that this is because Discord doesn't want their full backend being known by everyone as people may start to exploit bugs that are found.

How the design changed

The final solution has the functionality of all the desired features as mentioned previously, but the design changed multiple times throughout development. At first, I tried developing the Bot by taking code created by others on the Old API branch and convert that code to the new branch, however, by doing this I quickly realised how hard it became to read the code and how inefficient it became therefore I scrapped this approach and opted to learn the new Discord API branch and write all of my own code.

The original design of the system allowed for the API requests to be made instantly and as often as they were needed, however, due to API banning and limiting being a reality I made the system sleep through functions to slow down API requesting rates. Furthermore, the original Design used Threading, it allowed the bot to reply to multiple people at once, yet, again this proved problematic with making too many API request, as such I removed Threading from the final design.

The design for "!votecont" also changed, as it currently stands students vote by clicking on the Thumbs Up reaction, whereas in a previous design student had to use "!vote yes" to indicate to the bot that they were putting forward a yes vote. I was not stratified by students having to use "!vote yes" therefore I came up with the bot posting an embed which it adds a reaction to, then student click that reaction if they agree with the contribution- this was a much more elegant solution.

Code Refactor

The process of creating this bot took months, therefore my understanding of the API grew, the code for the first feature I implemented was very messy as I was inexperienced with the API compared to the last feature I added which was clean and short. In the end, I decided to refactor the entirety of my code, I kept the functionality the same, but rewrote the code, removed unnecessary variables, reducing the API request by making two requests in a single request and I also made the structure of the code easier to read. Moreover, I commented the code to the best of my ability so that someone would be able to understand it in the future.

Even though I have refactored the code once, it needs to be done again. Although the code is a lot better structurally compared to before, there is still opportunity to improve it further. Currently, almost all of the code is within the same Main.py file, if I had time on my side I would take each function and put them into their own .py file, allowing for the code to be more easily understood.

Test Cases

I did not have the time to create a full testing system or fully test every possible input for my bot, however here are what I deemed as some of the most essential tests for my bot that showed its functionality and tests that would cause the bot to produce errors due to bugs within the code that I was unable to fix in time.

!setup Testing

| Test.No | Action | Input | Exected Result | Actual Output | Test Result | Test Comments | Screensho t in Appendix |
|---------|---|-------------|----------------------|--|-------------|---|-------------------------|
| 1 | Using command correctly | !setup 4 | Creates rooms/ perms | Created rooms and perms | Pass | As expected, but progress bar in CMD prompt broken. | Appendxi D |
| 2 | Using Command with no value | !setup | Bot Does nothing. | Bot does nothing. Gets error display on command prompt | Pass | Bot does nothing in discord, error displayed in cmd prompt, but operation carries on. | Appendix E |
| 3 | Using command with incorrect characters | !setup !!”£ | Bot does nothing | Bot does nothing. Gets error display on command prompt | Pass | Bot does nothing in discord, error displayed in cmd prompt, but operation carries on. | Appendix F |

!addcontribution

| Test.No | Action | Input | Exected Result | Actual Output | Test Result | Test Comments | Screensho t in Appendix |
|---------|-----------------------------|---|--|---------------|-------------|---------------|-------------------------|
| 1 | Using command correctly | !addcontribution @Student 1#2774 Refactored Code - 10 | Contribution gets added to .csv and confirmation posted. | As expected | Pass | N/A | Appendix H |
| 2 | Using Command with no value | !addcontribution | Nothing happens. Bot error displayed | As expected | Pass | N/A | Not supplied |
| 3 | Using command | !addcontribution | Nothing happens. | As expected | Pass | N/A | Not supplied |

| | | | | | | | |
|---|---|---|--|-------------|------|-----|------------|
| | d with incorrect characters | 1323£"! "£~"£ | Bot error displayed | | | | |
| 4 | Use as expected without Supervisor Role | !addcontribution @Student 1#2774 Refactored Code - 10 | Bot will state that only supervisors can use this command. | As Expected | Pass | N/A | Appendix H |

!votecont

| Test.No | Action | Input | Exected Result | Actual Output | Test Result | Test Comments | Screenshot in Appendix |
|---------|---|--|--------------------------------------|---------------|-------------|---------------|------------------------|
| 1 | Using command correctly | !votecont @Student 1 -- Added Python script v2 -- 14 | Bot creates vote | As expected | Pass | N/A | Appendix I |
| 2 | Using Command with no value | !votecont | Nothing happens. Bot error displayed | As expected | Pass | N/A | Not supplied |
| 3 | Using command with incorrect characters | !votecont 213sadam | Nothing happens. Bot error displayed | As expected | Pass | N/A | Not supplied |

!mycontribution

| Test.No | Action | Input | Exected Result | Actual Output | Test Result | Test Comments | Screenshot in Appendix |
|---------|-------------------------|-----------------|---|---------------|-------------|---------------|------------------------|
| 1 | Using command correctly | !mycontribution | Bot displays embed with personal contribution | As expected, | Pass | N/A | Appendix J |

!groupcontribution

| Test.No | Action | Input | Exected Result | Actual Output | Test Result | Test Comments | Screenshot in Appendix |
|---------|--------|-------|----------------|---------------|-------------|---------------|------------------------|
|---------|--------|-------|----------------|---------------|-------------|---------------|------------------------|

Author: Kieran Parnell 1613910
Supervisor: Frank C Langbein

| | | | | | | | |
|---|-------------------------|--------------------|--|--------------|------|-----|------------|
| 1 | Using command correctly | !groupcontribution | Bot displays embed with group contribution | As expected, | Pass | N/A | Appendix K |
|---|-------------------------|--------------------|--|--------------|------|-----|------------|

Contribution System

| Test.No | Action | Input | Exected Result | Actual Output | Test Result | Test Comments | Screensho t in Appendix |
|---------|---|---------------------------|---|---------------|-------------|---------------|-------------------------|
| 1 | Using command correctly | !reg c1231233 Poppy Brown | Bot gives roles and confirms as message | As expected | Pass | N/A | Not supplied |
| 2 | Using Command with no value | !reg | Bot replies with "Invalid Input" | As expected | Pass | N/A | Not supplied |
| 3 | Using command with incorrect characters | !reg 2431ad121 2esd | Bot replies with "Invalid Input" | As expected | Pass | N/A | Not supplied |

Overall Evaluation

I would argue that the bot works great compared to the initial design, in fact it has some extra functionality that was not defined in the initial plan was added. However, there are also some small features that were not implemented but were in the initial plan. The question voting system is one of those features. I decided to cut the question voting system due to time constraints, I instead spent this time on other more essential parts of the system, like the “!setup” command that was not previously part of the initial plan . Furthermore, in the initial plan I wanted to implement some level of natural language processing for profanity filtering, but in the end decided against this, as in all test I did with natural language processing the system would either be too aggressive with removing messages or not do anything at all. Natural Language processing is still in its early stages and putting it into a system like this and it added unnecessary complexity for something that was not required for the bot to function correctly.

Finally, I wanted to add a full logging system, that would log absolutely everything done within the Discord server, I only implemented this partially. When developing, I became aware that Discord already logs actions, therefore implanting this again through the bot would make no sense, instead I implemented a logging system that would log the bots actions, something that the Discord logging system did not do. Moreover, a logging system that would log absolutely everything done within a sever may be arguably useless, it would be very cluttered and hard to read.

Discord API Banning and Limiting.

In my initial plan and at the start of development I was not aware that API banning or limiting was a reality, but I quickly learnt what it was. The problem with Discord’s API Bans/Limits is that they lack any documentation on what is considered “Too Much” or an “Acceptable amount” of API requests. After being banned a few times, I decided to contact Discord, they informed me that the amount of acceptable API request can vary depending upon the Gateway your Bot is using, but as a general rule 1,000 request an hour was an acceptable amount- this was not true. After many more bans and talking to others on forums it turns out that there is no consistency on how many API requests you can make. I implemented a wrapper around my code that counted the number of requests made, by doing this I learnt how many API requests were getting me banned. Some days 600 API requests would get me banned; other days 1,400 API request would get me banned. This made it very difficult to program a bot and not knowing how many API requests would get me banned, in the end I opted for the solution to slow the API request down so much that they would be well below 400 request an hour. This in turn caused the issue of “!setup” not being instant and instead take a large amount of time, scaling with the amount of groups you wish to make. Moreover, this is the reason that threading was removed for the final design.

Discord.py Rewrite

From August 23rd 2015 to April 8th 2019 there was only one branch for the Discord.py API, meaning that there is four years of Documentation for this branch, however April 8th 2019 marked the date that the new Discord.py rewrite was launched bring more functionality. For my project, I opted to use the new rewrite branch as it offered more functionality, the negative side to this is how there was only 9 months from when the rewrite was launched to when I started my project, meaning in that time no one online had really used the rewrite branch yet.

Outside of the Discord.py Official documentation, there is limited material out there using the rewrite version of Discord.py, as such the learning process for how to use this new rewrite branch was very difficult. Even the official documentation of the PyPI. 2020. *Discord.Py.* was very

vague, it does not clearly show what the API request do or what they will output. Moreover, I tried to find Discord bots created by others, but for all the bots I found the source code for they were all using the old Discord.py branch before the rewrite making them useless to help my understanding. This affected the start of my project, I had to spend the first two months of the project teaching myself this new Discord.py rewrite with limited examples online on how to use it, although in the end I managed to fully complete the project, it did take valuable time away that could have been spent adding error handling and fixing bugs within the current system.

Another Discord.py rewrite may happen again in the future and although Discord does support these old Discord.py branches for a few years after the new rewrite it would mean that this system is likely to be out of date within the next 5-10 years, in that time this bot may no longer have any support and therefore no longer work. Additionally, If Discord were to ever end as a service, then this code would no longer hold any value.

Future Possibilities

My understanding of Discord, the API how a registration system might work and how a contribution system might work has given my ideas and other features that could have been implemented but were not part of my original plan.

There is a lot of room for improvement on this system, I would call this current system an early proof of concept of how this system would work and be implemented. Things like the registration system are not very user friendly, instead of using a direct message system with the bot an online portal should be used to register instead.

At its current state, the bot is unable to scale to a large user base due to API limitations, however I am aware that it is possible to have a huge user base. For example, Octave (bot.to. 2020. *Octave - Bot.To.*) is a music bot that is currently used in over 250,000 servers and they somehow avoid Discord API bans. With more time and research it would be possible to create my bot in such a way that API banning is prevented and the `time.sleep()` within the code could be removed along with reintroducing threading. From what I have found, there does not seem to be an option for paying Discord for better API request rates, but one solution that I did find is the idea of sharding (*Dsharpplus.emzi0767.com. 2020*) , where you split the bot of multiple gateways depending upon how many servers the bot is in. However, documentation on how to implement something like sharding for Discord bots is very limited.

The Contribution system needs work too, right now you can view and add contributions but there is no way to delete or edit the contributions made without an administrator manually editing the .CSV files. Moreover, while developing I realised that contribution isn't just in the work or hours you do for the group, but also the ideas and participation levels you have at meetings, therefore in the future it would be nice if this contribution system could also record how many meetings people turn up to and record how much time they spend in voice rooms or how many messages they send to record participation. Although arguable, this could be exploited quite easily.

The bot is running through command prompt on windows, currently the bot does not reboot or start on windows start-up. If this were to be deployed a system Administrator would have to implement something that would restart the bot on request and after a specified amount of time. Moreover, although this should work on Linux and Mac, some further testing would need to be done to ensure that the bot does not run into issues and these operating systems.

Conclusion

To conclude, I am satisfied that the system I have developed provides a good proof of concept and demonstrates a basic system for dealing with group contribution levels. Compared to the initial plan, I completed most of what I stated I would, the features I did not complete were replaced with more important parts of the system that I had not thought about during the initial plan. The system records contributions, a registration feature, and some simple profanity filtering.

Some features are better implemented than others, the registration system is very robust and from my testing has no bugs, whereas the contribution system has many bugs, when you provide a command call that is not in the intended structure. What I have produced so far has achieved what I was aiming for, a system that shows proof of concept.

The only major concerns I have about this system is how it would not be able to scale to a large user base without some big changes to how the system works and how the code is structured. In order to overcome these issues, I would have to learn and implement Technologies related to bot sharding, where the bot is split over multiple gateways. Despite this, if you ignore the bugs, the system would be sustainable for a small to medium size user base.

Personal Reflection on performance

This project has taught me a lot, especially how in the initial plan I did overestimate how much I would be able to get done, although I completed almost everything mentioned in the initial plan, it caused me to spend too many sleepless nights on this project. Additionally, if I were to do this project again, I would be sure to do much more research. At the initial plan stage, I should have already found out about the recent Discord.py Rewrite, then I could of adjusted my expectations and intended features accordingly.

I also failed to appreciate the complexity of what I stated in my initial plan, for instance, one of my planned features was simply "Create a registration system", at the time I had no idea what that entailed and just assumed it would be easy. The registration system was complex as it was not just the registration that had to be correct, but the structure and the permissions for the server too., if !setup did not exist then neither could the registration system. For future endeavours I will spend more time thinking about features I purpose and just how complex some of these features may be to implement.

From this project, I can say it has greatly improved my literacy with Python and I would feel confident creating any bot for Discord- I now have a full understanding of the Discord.py API. For future projects I will take more care in planning, in the initial plan I didn't decide how the project was going to be designed in terms of the methodology, a waterfall approach, iterative and so on.

Ethics

At the start of the project I did not fully understand how ethics would affect my project, my designed changed due to ethics. At first, I did not understand the importance of ensuring data protection and ensuring that I was not storing any personal information on someone that could be damaging. After correspondence with the Ethics department I was able to increase my understanding of how my design could be reshaped and fit the ethic guidelines. One of the ways I did this was when someone joined the server, they get a direct message from the bot, within that message it contains ethics information. I would argue that dealing with ethics did take away time I could have spent on creating a more robust and feature rich system, but it was necessary.

How to Setup and run the bot

Here I will give a guide on how to get the bot running on your own server. I have added this at the end of the report as I did not find it necessary in the main body of the report. For this setup I will assume you are using **Windows**.

What you will need:

- Install Python 3.8.2 or higher: <https://www.python.org/downloads/>
- Install Discord.py 1.3.2. This can be installed by using "pip install discord.py" in command prompt. *You must install python before doing this.*
- The code folder that was submitted with this report.

Step 0: Within the Discord application, create your own server. This can be done by pressing the Green "+" within Discord.

Step 1: Go and use the developer portal to start an application for your bot:
<https://discordapp.com/developers/applications>

Step 2: After creation, you will need to invite the bot to your server. You can find this link in the developer portal.

Step 3: In the code folder submitted with this report, open Main.py and at the very bottom of the code you'll see: `client.run("")`, between the "" you must put your Client key, again this can be found at the Discord Developer Portal.

Step 4: Double Click Main.py to run it. If you want to see errors, call the file through command prompt.

Step 5: If everything was done correctly, the bot will now be in your Discord server and be online.

Step 6: In the Discord server type "!help", if the bot is working it will respond to this command.

You can go here for further help: <https://discordpy.readthedocs.io/en/latest/discord.html>

References

1. bot.to. 2020. Octave - Bot.To. [online] Available at: <<https://bot.to/bot/octave/>> [Accessed 14 May 2020].
2. Discord. 2020. Discord — A New Way To Chat With Friends & Communities. [online] Available at: <<https://discord.com/>> [Accessed 11 May 2020].
3. Discord Developer Portal. (2020). Discord Developer Portal — API Docs for Bots and Developers. [online] Available at: <https://discordapp.com/developers/docs/intro> [Accessed 1 Feb. 2020].
4. Discordpy.readthedocs.io. 2020. Creating A Bot Account — Discord.Py 1.4.0A Documentation. [online] Available at: <<https://discordpy.readthedocs.io/en/latest/discord.html>> [Accessed 14 May 2020].
5. Dsharpplus.emzi0767.com. 2020. Sharding - For When Your Bot Gets Huge. [online] Available at: <<https://dsharpplus.emzi0767.com/articles/sharding.html>> [Accessed 14 May 2020].
6. Mee6.xyz. 2020. MEE6 - The Discord Bot. [online] Available at: <<https://mee6.xyz/>> [Accessed 11 May 2020].
7. Microsoft.com. 2020. Chat, Meetings, Calling, Collaboration | Microsoft Teams. [online] Available at: <https://www.microsoft.com/en-gb/microsoft-365/microsoft-teams/group-chat-software?ef_id=Cj0KCQjw-_j1BRDkARIsAJcfmTGGi81CvCHhuMBt3jfz8R7_WFxCpfxQYAP49LZzDylk2-YoFdB7dQaAvqyEALw_wcB:G:s&OCID=AID2000956_SEM_Cj0KCQjw-_j1BRDkARIsAJcfmTGGi81CvCHhuMBt3jfz8R7_WFxCpfxQYAP49LZzDylk2-YoFdB7dQaAvqyEALw_wcB:G:s> [Accessed 15 May 2020].
8. PyPI. 2020. Discord.Py. [online] Available at: <<https://pypi.org/project/discord.py/>> [Accessed 11 May 2020].
9. Skype.com. 2020. Skype | Communication Tool For Free Calls And Chat. [online] Available at: <<https://www.skype.com/en/>> [Accessed 15 May 2020].
10. 2020. Statbot. [online] Available at: <<https://top.gg/bot/491769129318088714>> [Accessed 11 May 2020].

Appendix

Appendix A

| | A | B | C | D | E | F | G |
|----|----------------------------------|---------|----------|---------------|---|--------------------------|------------------|
| 1 | Global Permissions | | | | | | |
| 2 | | Student | Lecturer | Administrator | | | |
| 3 | Display Role Separately | | | | | | |
| 4 | Allow @everyone to @ role | | | | | | |
| 5 | Full Administrator | | | | | Enabled | |
| 6 | View Audit Logs | | | | | Takes Global Permission | |
| 7 | Manage Server | | | | | Disabled | |
| 8 | Manage Roles | | | | | | |
| 9 | Manage Channels | | | | | | |
| 10 | Kick Members | | | | | | |
| 11 | Ban Members | | | | | | |
| 12 | Create Invite | | | | | | |
| 13 | Change Nickname | | | | | | |
| 14 | Manage Nicknames | | | | | Perms Hierarchy | |
| 15 | Manage Emojis | | | | | Individual Room | Overrides |
| 16 | Manage Webhooks | | | | | Global | Default |
| 17 | Read Text and See voice Channels | * | | | | | |
| 18 | Send Message | * | | | | *Can still see own group | |
| 19 | Send TTS | | | | | | |
| 20 | Manage Messages | | | | | | |
| 21 | Embed Links | | | | | | |
| 22 | Attach Files | | | | | | |
| 23 | Read Message History | * | | | | | |
| 24 | Everyone Mentions | | | | | | |
| 25 | External Emojis | | | | | | |
| 26 | Add Reactions | | | | | | |
| 27 | Connect | | | | | | |
| 28 | Speak | | | | | | |
| 29 | Mute Members | | | | | | |
| 30 | Deafen | | | | | | |
| 31 | Move Members | | | | | | |
| 32 | Use Voice Activity | | | | | | |
| 33 | Priority Speaker | | | | | | |
| 34 | GoLive (Screenshare). | | | | | | |
| 35 | | | | | | | |
| 36 | Category Permissions | | | | | | |
| 37 | Display Role Separately | | | | | | |
| 38 | Allow @everyone to @ role | | | | | | |
| 39 | Full Administrator | | | | | | |
| 40 | View Audit Logs | | | | | | |
| 41 | Manage Server | | | | | | |
| 42 | Manage Roles | | | | | | |
| 43 | Manage Channels | | | | | | |
| 44 | Kick Members | | | | | | |
| 45 | Ban Members | | | | | | |
| 46 | Create Invite | | | | | | |
| 47 | Change Nickname | | | | | | |
| 48 | Manage Nicknames | | | | | | |
| 49 | Manage Emojis | | | | | | |
| 50 | Manage Webhooks | | | | | | |
| 51 | Read Text and See voice Channels | | | | | | |
| 52 | Send Message | | | | | | |
| 53 | Send TTS | | | | | | |
| 54 | Manage Messages | | | | | | |
| 55 | Embed Links | | | | | | |
| 56 | Attach Files | | | | | | |
| 57 | Read Message History | | | | | | |
| 58 | Everyone Mentions | | | | | | |
| 59 | External Emojis | | | | | | |
| 60 | Add Reactions | | | | | | |
| 61 | Connect | | | | | | |
| 62 | Speak | | | | | | |
| 63 | Mute Members | | | | | | |
| 64 | Deafen | | | | | | |
| 65 | Move Members | | | | | | |
| 66 | Use Voice Activity | | | | | | |
| 67 | Priority Speaker | | | | | | |
| 68 | GoLive (Screenshare). | | | | | | |

Figure 19: Breakdown of permissions

Appendix B

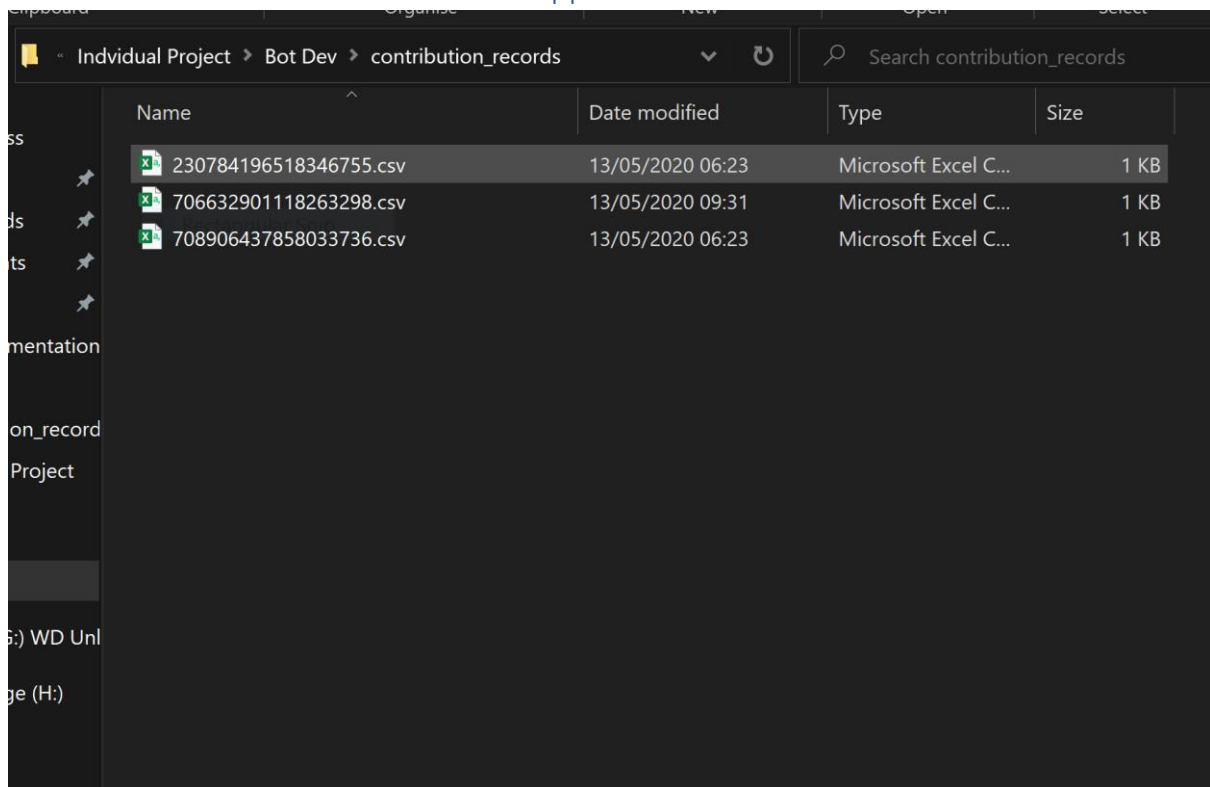


Figure 20: Student Contribution Record Naming

| | A | B | C |
|---|-----------------------------------|------------------|----|
| 1 | Created Python script for project | 13/05/2020 09:31 | 4 |
| 2 | | | |
| 3 | Refactored code | 13/05/2020 09:31 | 10 |
| 4 | | | |
| 5 | | | |
| 6 | | | |

Figure 21: Structure of Contribution Records

Appendix C

```
--- Viewing Personal Contribution ---#
if message.content.lower() == ('mycontribution'):
    hourscounter = 0
    try:
        my_id = message.author.id #Holds users Discord ID
        filename = "contribution_records\\"+str(my_id)+".csv" #Stores location of folder to access
        embed = discord.Embed(title=str(message.author)+'s Personal Report',description = "Here are your contributions so far",color=0xd32905)
        with open(filename, 'r') as myfinder:
            for items in myfinder:
                items = items.rstrip() #Removes blank newlines
                if items == "": #used to do nothing in loop if there is no characters
                    continue
                else:
                    items = items.split(",") #seperates note and the date/time
                    #Send Personal Report
                    embed.add_field(name=items[1], value=items[0]+" **Hours Spent:** "+items[2], inline=False)
                    hourscounter+=int(items[2])
            await message.channel.send(embed=embed)
        await message.channel.send("**Total Time Contributed: **"+str(hourscounter)+"** Hours**")
    except FileNotFoundError:
        await message.channel.send("**You have no contributions yet!**")
```

Figure 22: My contribution Code

Appendix D

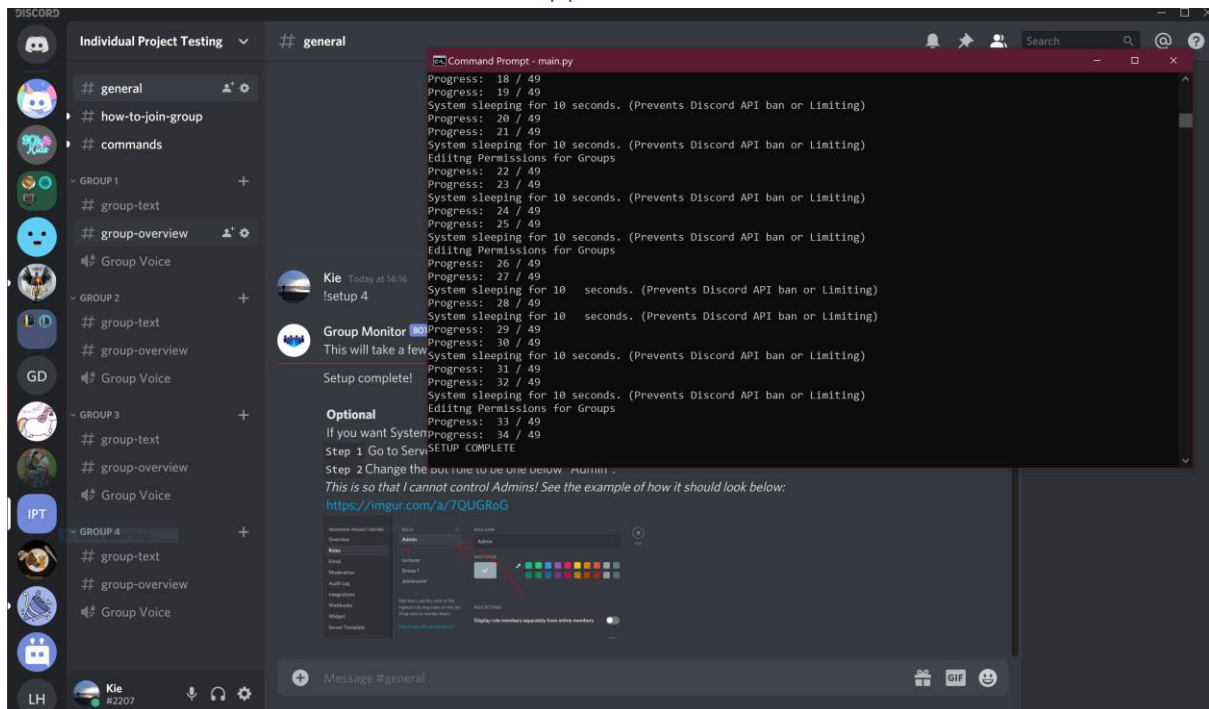
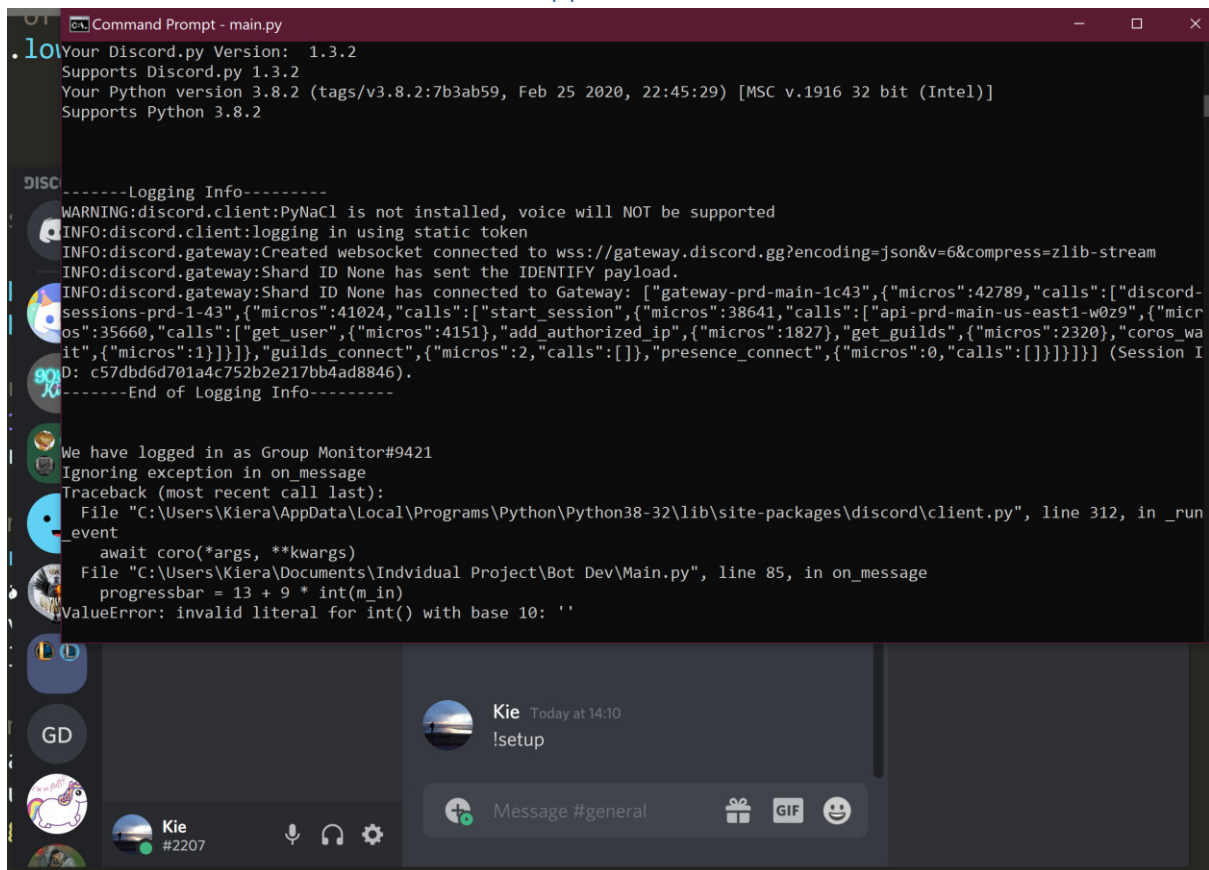


Figure 23: Test for !setup 4

Appendix E



The screenshot shows a Discord bot's command prompt window titled "Command Prompt - main.py". The window displays the following text:

```
.!o Your Discord.py Version: 1.3.2
Supports Discord.py 1.3.2
Your Python version 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (Intel)]
Supports Python 3.8.2

DISC -----Logging Info-----
WARNING:discord.client:PyNaCl is not installed, voice will NOT be supported
INFO:discord.client:logging in using static token
INFO:discord.gateway:Created websocket connected to wss://gateway.discord.gg?encoding=json&v=6&compress=zlib-stream
INFO:discord.gateway:Shard ID None has sent the IDENTIFY payload.
INFO:discord.gateway:Shard ID None has connected to Gateway: ["gateway-prd-main-1c43",{"micros":42789,"calls":["discord-
sessions-prd-1-43",{"micros":41024,"calls":["start_session",{"micros":38641,"calls":["api-prd-main-us-east1-w0z9",{"micr
os":35660,"calls":["get_user",{"micros":4151,"add_authorized_ip",{"micros":1827,"get_guilds",{"micros":2320,"coros_wa
it",{"micros":1}}]}], "guilds_connect",{"micros":2,"calls":[]}, "presence_connect",{"micros":0,"calls":[]}}]}] (Session I
D: c57dbd6d701a4c752b2e217bb4ad8846).
-----End of Logging Info-----

We have logged in as Group Monitor#9421
Ignoring exception in on_message
Traceback (most recent call last):
  File "C:\Users\Kiera\AppData\Local\Programs\Python\Python38-32\lib\site-packages\discord\client.py", line 312, in _run
_event
    await coro(*args, **kwargs)
  File "C:\Users\Kiera\Documents\Individual Project\Bot Dev\Main.py", line 85, in on_message
    progressbar = 13 + 9 * int(m_in)
ValueError: invalid literal for int() with base 10: ''
```

The bottom part of the image shows the Discord chat interface. A message from "Kie" (Today at 14:10) says "!setup". The chat window shows the message history and the input field with the text "Message #general". The user's name "Kie" and ID "#2207" are visible in the bottom left corner.

Figure 24: Test for !setup

Appendix F

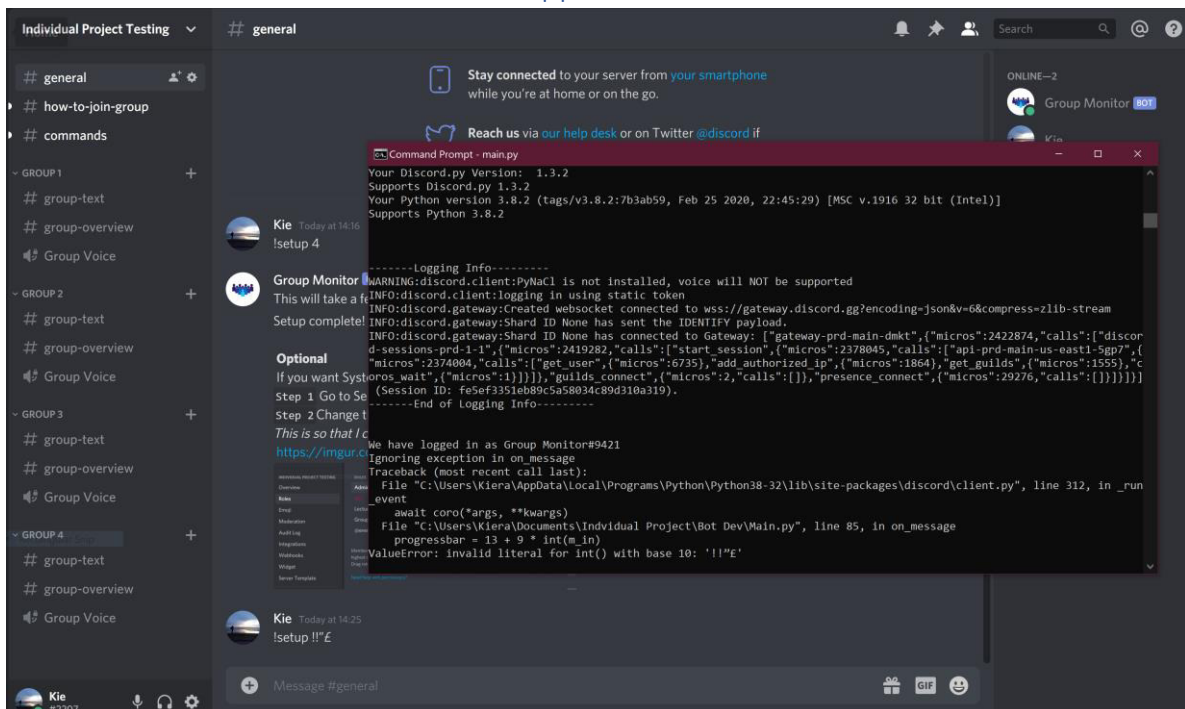


Figure 25: Test for !setup !!"£

Appendix G

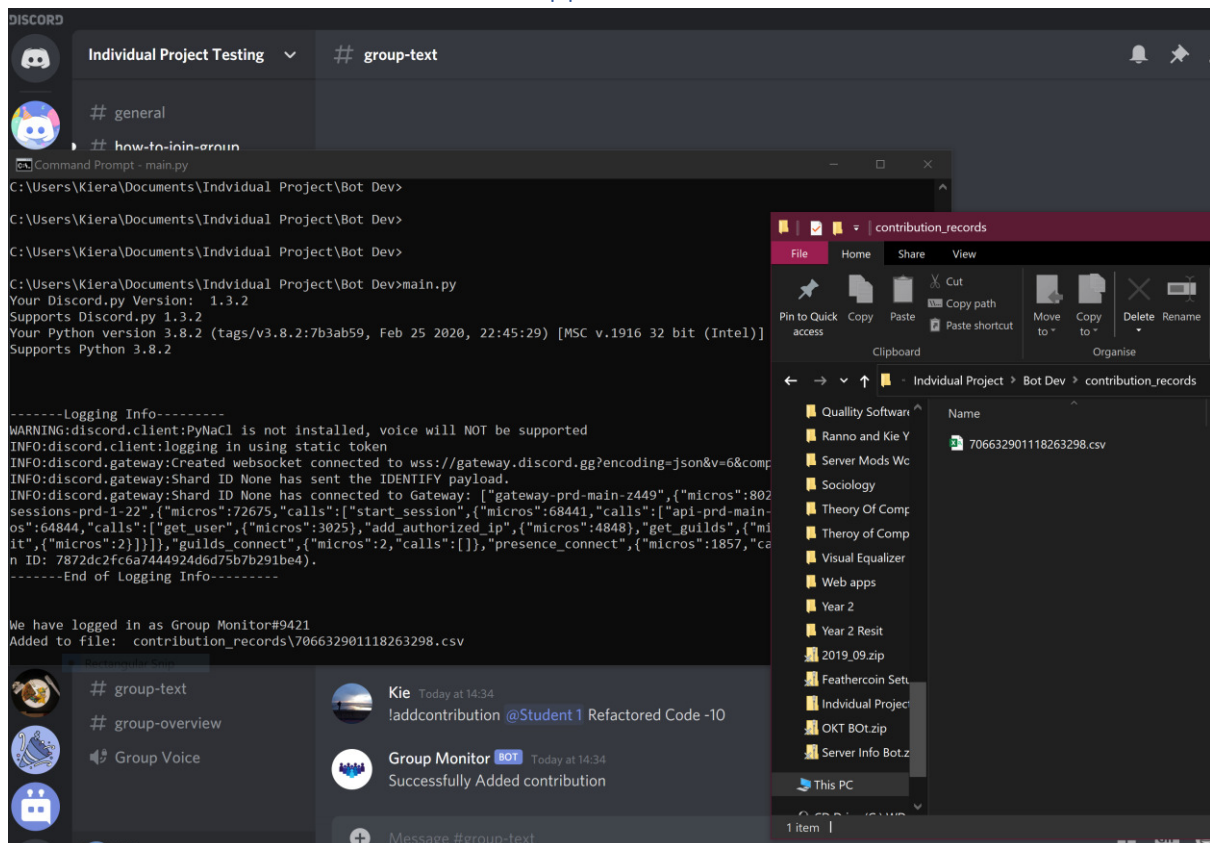
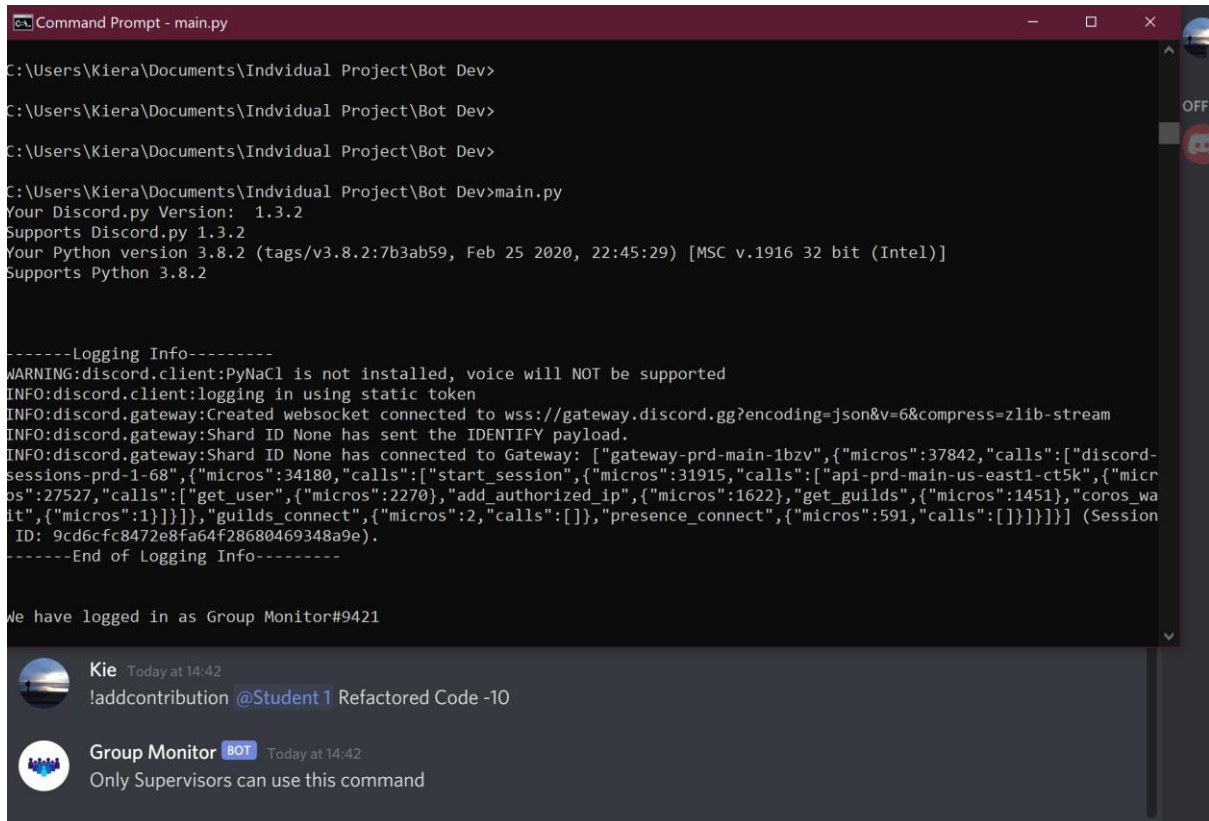


Figure 26: Test for !addcontribution @Student1 Refactored Code -10

Author: Kieran Parnell 1613910
Supervisor: Frank C Langbein

Appendix H



The screenshot shows a Discord chat window with a dark theme. At the top, a command prompt window titled "Command Prompt - main.py" is open, displaying the execution of a Python script. The script outputs version information for Discord.py and Python, followed by a detailed logging session. The logging session includes a warning about PyNaCl, information about gateway connections, and a successful login for a bot named "Group Monitor". Below the command prompt, the Discord chat shows a message from a user named "Kie" using the command `!addcontribution @Student 1 Refactored Code -10`. A response from the "Group Monitor" bot follows, stating "Only Supervisors can use this command".

```
Command Prompt - main.py
C:\Users\Kiera\Documents\Individual Project\Bot Dev>
C:\Users\Kiera\Documents\Individual Project\Bot Dev>
C:\Users\Kiera\Documents\Individual Project\Bot Dev>
C:\Users\Kiera\Documents\Individual Project\Bot Dev>main.py
Your Discord.py Version: 1.3.2
Supports Discord.py 1.3.2
Your Python version 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (Intel)]
Supports Python 3.8.2

-----Logging Info-----
WARNING:discord.client:PyNaCl is not installed, voice will NOT be supported
INFO:discord.client:logging in using static token
INFO:discord.gateway:Created websocket connected to wss://gateway.discord.gg?encoding=json&v=6&compress=zlib-stream
INFO:discord.gateway:Shard ID None has sent the IDENTIFY payload.
INFO:discord.gateway:Shard ID None has connected to Gateway: [{"gateway-prd-main-1b2v",{"micros":37842,"calls":["discord-sessions-prd-1-68",{"micros":34180,"calls":["start_session",{"micros":31915,"calls":["api-prd-main-us-east1-ct5k",{"micros":27527,"calls":["get_user",{"micros":2270},{"add_authorized_ip",{"micros":1622},{"get_guilds",{"micros":1451},{"coros_wa",{"micros":1}],}],{"guilds_connect",{"micros":2,"calls":[]},{"presence_connect",{"micros":591,"calls":[]}]}}]}]} (Session ID: 9cd6cfc8472e8fa64f28680469348a9e).
-----End of Logging Info-----

We have logged in as Group Monitor#9421

Kie Today at 14:42
!addcontribution @Student 1 Refactored Code -10

Group Monitor BOT Today at 14:42
Only Supervisors can use this command
```

Figure 27: `!addcontribution @Student 1 Refactored Code -10` without Supervisor Perm

Appendix I

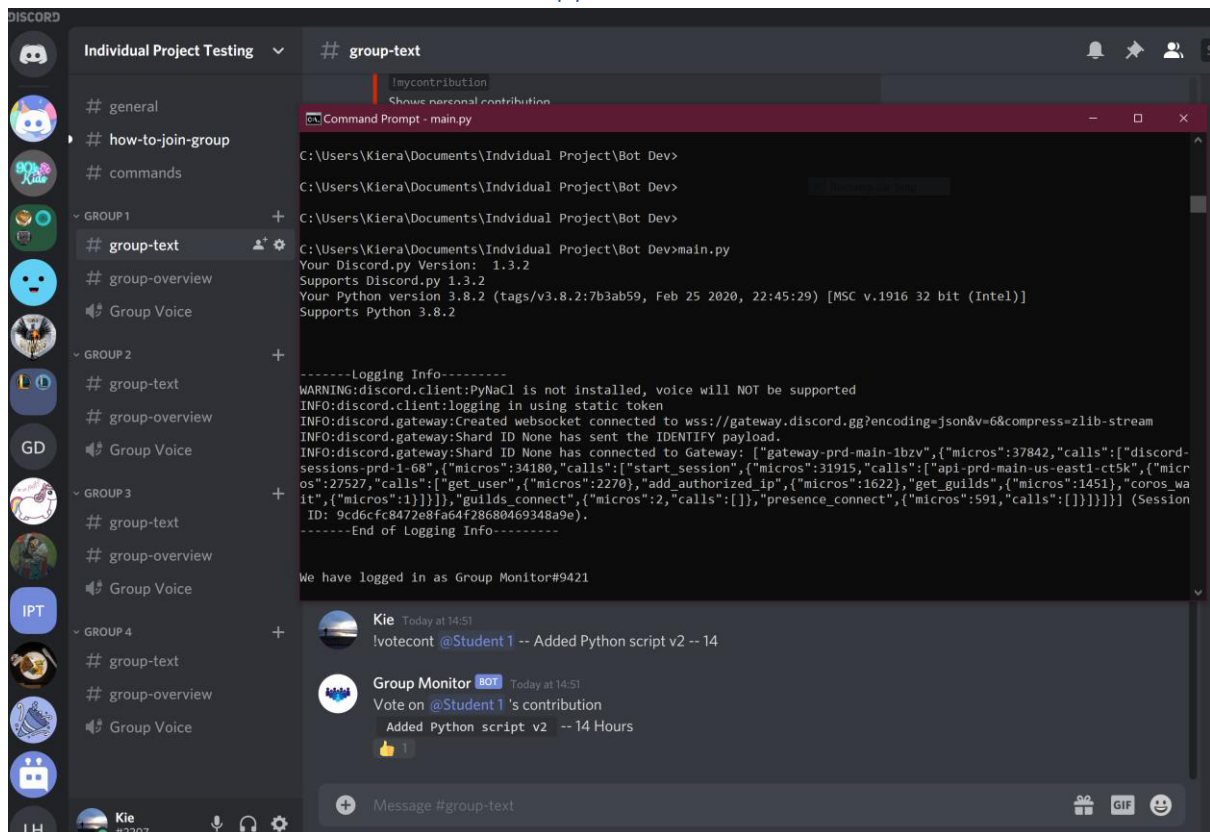


Figure 28: Test for Ivotecont @Student 1 -- Added Python script v2 – 14

Appendix J

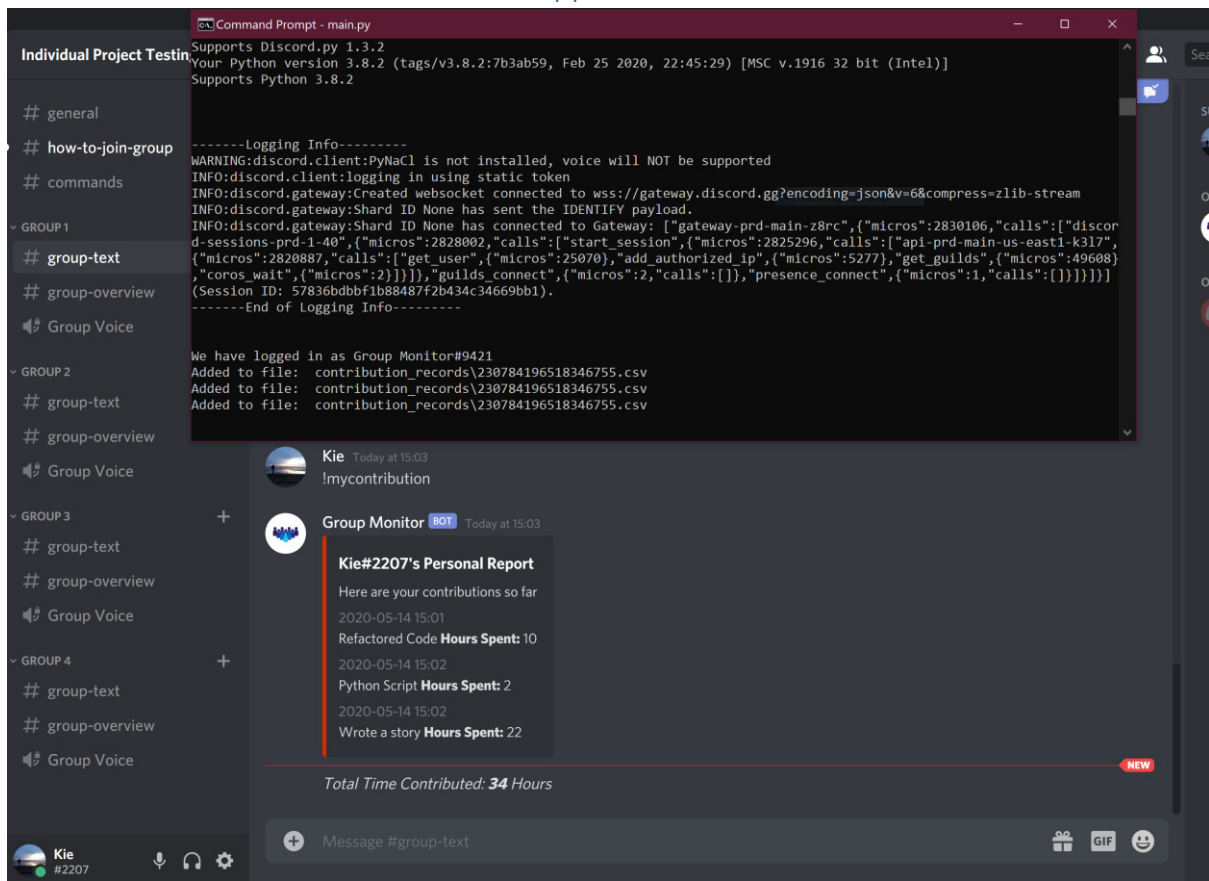


Figure 29: Test for !mycontribution

Author: Kieran Parnell 1613910
Supervisor: Frank C Langbein

Appendix K

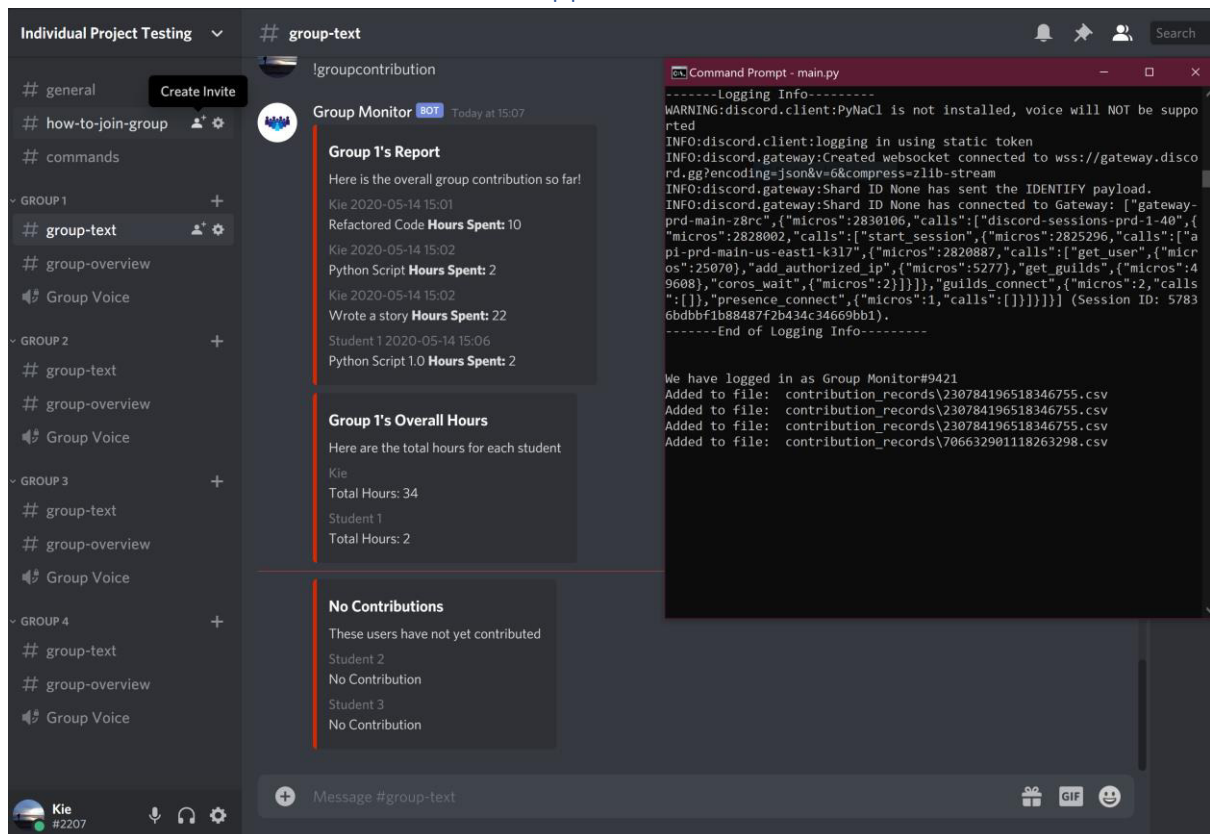


Figure 30: Test for !groupcontribution