Criterion B: Design

Record of tasks

Task Number	Planned action	Planned outcome	Estimated Time taken	Completion Date Target	Criterion
1	Think about program	To find a possible topic for the program	1 day	14th January	A
2	Find the client	Identify suitable client	1 day	25th January	A
3	First interview with client	Observe the problems and needs of my client	30 min	27th January	A
4	Think of the success criteria	Understand the exact requirements for the program	1 day	28th January	A
5	Email the client	Agree on the success criteria thought of before	1 day	1st February	A
6	Start thinking of the algorithms of the program	Know the algorithms that should be the main function of the system	1 week	10th February	B
7	Think about data stored in database	Realising all the information needed for the database	30 min	12th February	B
8	Finish the database normalisation of the tables	Have at least 3 tables to third normal form	1 day	13th February	B
9	Write data dictionaries	Have the data dictionaries done in the design overview	2 hours	14th February	B
10	Do the high level architecture planning	Know the different software interacting with the program	1 hour	15th February	B

11	GUI planning	Plan and understand how the program will look	1 hour	15th February	B
12	Do Gantt chart for the project	Have the time deadline for every criterion of the project	30 min	16th February	B
13	Figure out the error handling design	Have a way the system won't crash with a wrong input	1 hour	17th February	B
14	Error handling pseudocode	Complete the pseudocode for the error handling design	30 min	18th February	B
15	Data flow diagram level 0	Figure out the basic software interacting in the system with the data types	30 min	19th February	B
16	Data flow diagram level 1	Figure out the detailed components of each software interacting in the system with specific data types	1 hour	19th February	B
17	Product functionality explanation	Have the functionality explained in the design overview	30 min	22nd February	B
18	Come up with an algorithm to add *members* to database	Get the overall idea of what the algorithm does	1 hour	22nd February	B
19	Do a flowchart and pseudocode for the *members* algorithm	Have the documents for the algorithm written in the design overview	2 hours	23rd February	B
20	Come up with an algorithm to give roles to users with an *emoji id*	Get the overall idea of what the algorithm does	1 hour	23rd February	B
21	Do a Flowchart and pseudocode for the *emoji id* algorithm	Have the documents for the algorithm written in the design overview	2 hours	26th February	B

22	Do a structure diagram for *members* algorithm	Have the structure diagram completed in the design overview	2 hours	3rd March	B
23	*Warning* algorithm pseudocode	Find the pseudocode for the algorithm	30 min	4th March	B
24	*Warning* algorithm flowchart	Write a flowchart for the pseudocode	1 hour	4th March	B
25	Spider diagram for the commands	Show all the commands in the design overview with a spider diagram	1 hour	6th March	B
26	Structure diagram for *emoji id* algorithm	Have the structure diagram written in the design overview	1 hour	6th March	B
27	Structure diagram for *warnings* algorithm	Have the structure diagram written in the design overview	1 hour	8th March	B
28	Entity relationship diagram	Have the entity relationship diagram written in the design overview	1 hour	9th March	B
29	Extensibility of the program	Have the extensibility of the system written in the design overview	2 hours	9th March	B
30	Planning of test of the program	Finish the test plan to test the system once the code is finished	2 hours	10th March	B
31	Make a new guild to test the program	Create a new guild to test the system throughout the development stage	30 min	11th March	C
32	Look at the libraries I will use during my code	Learn and understand the libraries needed for the program	3 hours	13th March	C
33	Start building basic event *on_message*	Have the base of the code done to expand in easily in the future	1 week	20th March	C

34	Testing the *on_message* event and debugging	Test every input works for this event and fix any problems	3 hours	21st March	C
35	Start making simple commands *ban* and *unban*	Make a few commands and make them compatible with the events. Only the administrators of the guild can use these commands	1 week	28th March	C
36	Second interview to talk about the function and commands already done and what he would like	Know exactly what other commands, events, and functionalities the client wants	1 hours	29th March	A
37	Built the command *kick*	Finish doing the command that works. Only the administrators of the guild can use these commands	3 hours	30th March	C
38	Make the *on_member_join* event	Finish doing the event that sends a message when a user enters the guild	4 hours	31st March	C
39	Built the personalised *on_error* command for some commands	Finish doing the commands that works. It sends an error message and doesn't crash the program	1 week	7th April	C
40	Learned how to use SQLite3 database	Know how the basics of SQLite3 works	4 days	12th April	C
41	Watched a video to further inform myself about the databases	Have a refined knowledge on how SQLite3 works and how to implement it	5 hours	14th April	C
42	Make an *add* command to add	Finish the command with no data	2 hours	16th April	C

	notes and warning reason to text files	validation. Only the administrators of the guild can use these commands or compatibility errors between the program and the text files			
43	Add a *members* command to display every member in guild	Finish the command with no errors when displaying the members and doesn't display the bots in the guild	4 hours	18th April	C
44	Created the tables for the database	Have the tables planned before in a database	1 hour	20th April	C
45	Upgraded the *on_message* event to suit the database	Add the part of code in the event that inputs the data to the database efficiently	2 days	25th April	C
46	Made an automatic algorithm to add member data to database through the *members* command and the *on_member_join* event	Have an algorithm that automatically adds members without the necessity of human interference and doesn't record duplicated data from the same member. The algorithm can be used through the command to input all the members in the guild the first time the program is implemented in the guild.	3 days	25th April	C
47	Make the warning text file contain banned words and reasons through	the warnings text file retrieves member messages and check for inappropriate	1 hour	26th April	C

	the *on_message* event	comments and gives a warning if the message has a banned word			
48	Add event *on_user_update* to check if a member changes username	The event activates when members change the name and it changes the name from the database	2 hours	26th April	C
49	Used the event *on_message* to add coins and experience to a members for further use	When a member sends a certain amount of messages they get coins and experience that are stored in the database	2 hour	26th April	C
50	Copy the same algorithm as *on_message* for the event *on_voice_state_update* to give rewards for joining voice calls	It gives coins and experience automatically for a certain amount of time in the voice call	1 hours	26th April	C
51	Emailed the client to see which roles/ranks he wanted me to add	Figure out exactly what ranks and their respective roles to put in the program and guild	1 day	26th April	C
52	Create a 2D array called ranks with the experience needed and name of rank	Put the ranks into the program	30 min	27th April	C
53	Create the roles in the guild	Put the roles into the guild	30 min	27th April	C
54	Made the commands *rank* and *ranks* to show the role of each	Have two commands finished one to look at all the members and the other to state a specific member to	3 hours	27th April	C

	member in the guild	check. Only the administrators of the guild can use these commands fully and other members can only check their own rank. Displays members from highest to lowest rank.			
55	Make commands *warning*, and *warnings* to display the number of warnings of users	Have two commands finished one to look at all the members and the other to state a specific member to check. Only the administrators of the guild can use these commands fully and other members can only check their own rank. Displays members from highest to lowest number of warnings.	2 hours	28th April	C
56	Test the commands *rank, ranks, warning, and warnings* by inputting correct and incorrect inputs	Ensure the four commands work efficiently and correctly without crashing the program			C
57	Emailed the client if the data should be deleted when a member exists the guild	Find out if the client wants to delete the banned member data from the database	1 day	29th April	A
58	Made the event *on_member_ban* to delete records of the member	When a member is banned the record from the database gets deleted	1 hour	29th April	C

59	Make an algorithm in the *on_member_join* event to add roles	The event automatically assigns the lower role/rank to members when they join the guild	1 day	29th April	C
60	Add an algorithm that detects when the member reached a new rank	The algorithm changes the role and rank automatically when reaching the experience needed and deletes the previous role and rank from the member	30 min	30th April	C
61	Download images for emojis	Have the emojis in the guild and each emoji has a name that links to a role	30 min	30th April	C
62	Make the event *on_raw_reaction_add* that adds roles to members	When a member adds a reaction, it checks if it is in the correct message and then it adds the corresponding role	3 hours	30th April	C
63	Make the event *on_raw_reaction_remove* that deletes roles to members	When a member removes a reaction from the correct message the role is deleted	1 hour	30th April	C
64	Make members only have one role of each game	By using the database, the program stores the id of the emojis that have been added for every member and if the member adds a second reaction in the same message (each message is for a certain game) the event deletes the second reaction and	2 hours	30th April	C

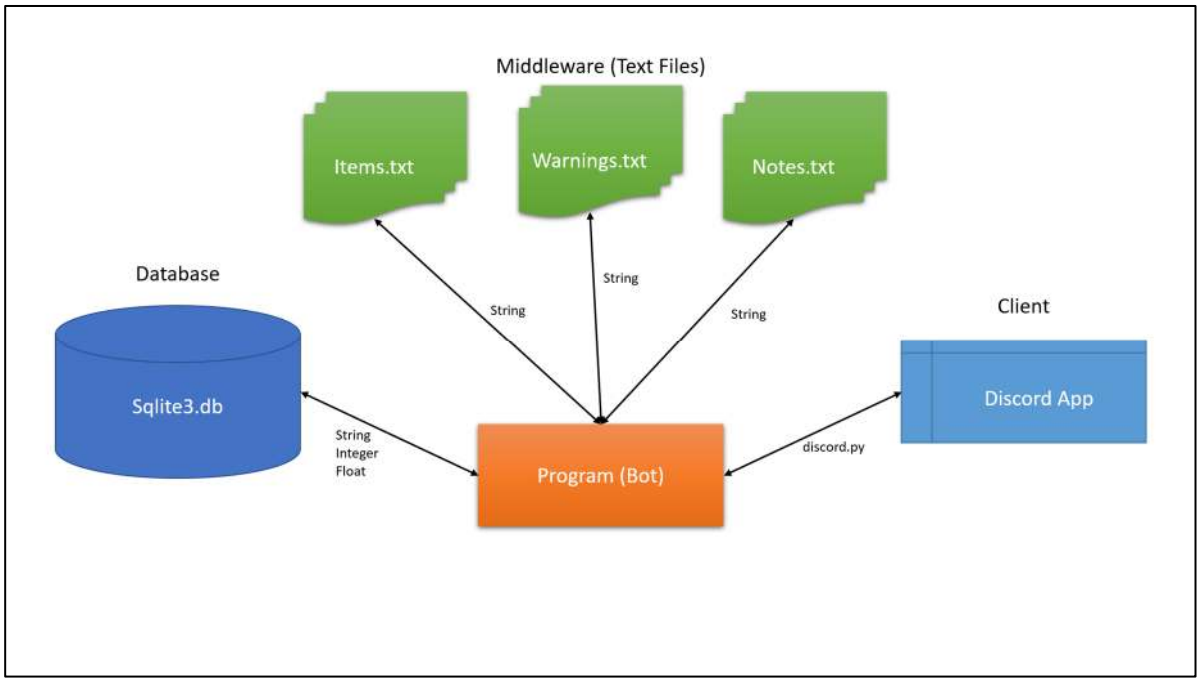
		sends an error message			
65	Made the *on_command_error* event to manage error from users	When an error occurs for a bad input from users the event activates and sends the error to the guild	1 hour	1st May	C
66	Create a *shop* command to display items that members can buy	When a member activates the command an embed message is sent in the guild with all the items	2 hours	2nd May	C
67	Make the *buy* and *gift* commands to buy items from shop with coins	The commands send a message when the item is bought for who. It takes away the money spent from the member and the new amount is saved in the database. If the member doesn't have enough money the commands send an error message	3 hours	2nd May	C
68	Making a *coins* command to display to users the number of coins they have	When a member activates the command, it displays the number of coins he has	30 min	3rd May	C
69	Made a *notes* command to display the notes in the text file Notes.txt	The command displays a list of all the notes in the file. Only administrators can use this command	30 min	3rd May	C
70	Made a *help* command to display every command and how to use it	It displays a list of the commands and an explanation of how to use them. It checks if the member who used the command is an	1 hour	3rd May	C

		administrator. If the member is administrator, it shows all commands, if not it displays only the available ones			
71	Added comments to my code	Have all the code with detailed comments so another person understands the code and is able to extend it.	1 hour	4th May	C
72	Test every event and command again in the testing guild made before	Have a list of all the events and commands that work correctly and the ones that give unexpected outputs	2 hours	10th May	C
73	Fix the events and commands by testing and changing the program	Finish all the changes so that the program works perfectly	4 hours	13th May	C
74	Install the program to the guild and prepare for use	Have the program ready to use by everyone	2 hours	15th May	C
75	Training: Talked to my client and some of the admins in the guild about the program	The people that are going to use my program the most know how it works	1 hour	15th May	C
76	Plan the video of the program	Have a list of ideas and a structure of the video	1 hour	19th May	D
77	Record the video of the uses of the program	Finish recording how the program interacts in the guild	2 hours	20th May	D
78	Watched my client using the program, Interview about how the program	Know if there are any weaknesses with my program and how I	1 hour	5th June	E

	went with the client	met the success criteria			
79	Fix the minor issues of the program	Have solved the issues stated by the client	1 hour	6th June	C
80	Evaluate the met success criteria	Write the strengths and weaknesses	2 hours	10th June	E

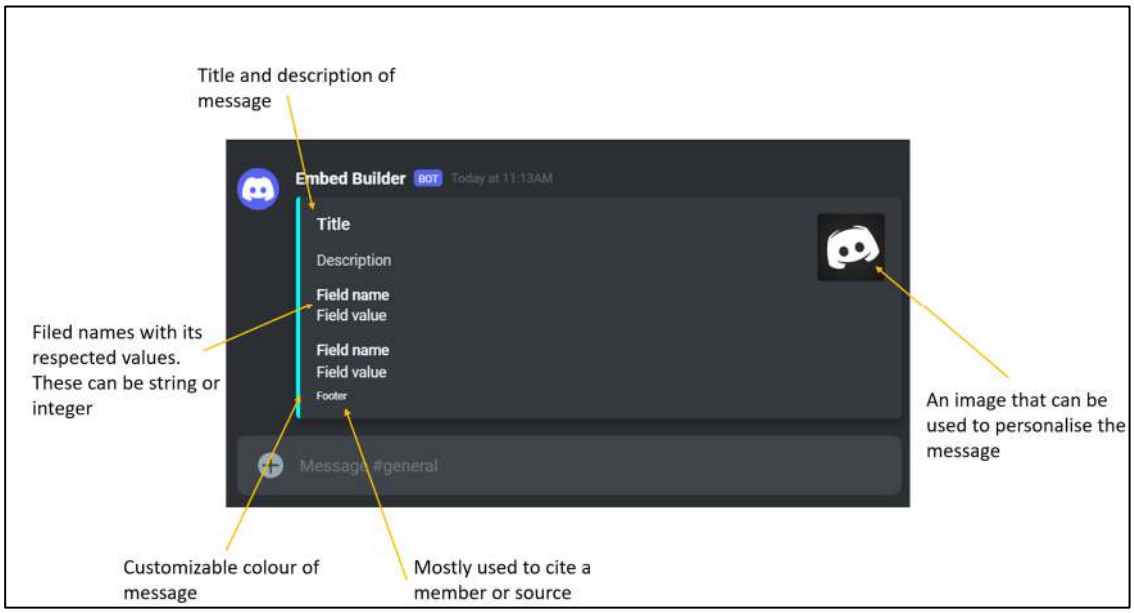
Design Overview

Higher level architecture:



Graphic User Interface (GUI):

This is a template Embed message from discord. This could be used for more important messages such as the help message. I will format this version to personalise it for different uses. It will be sent to members in the guild when they request information or help.



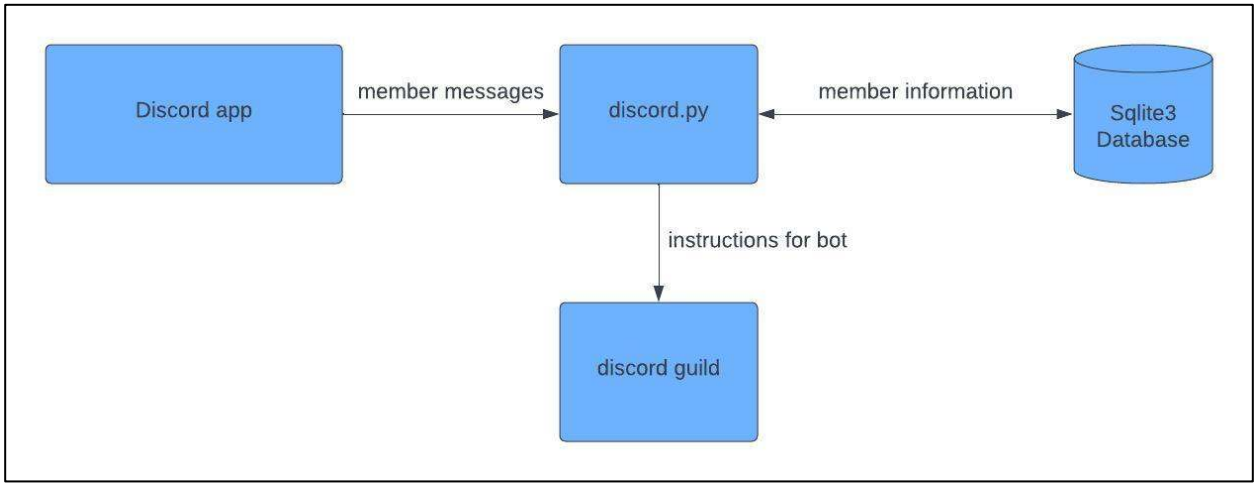
Gantt chart:

This is my initial plan for every criterion of the project during the six months.

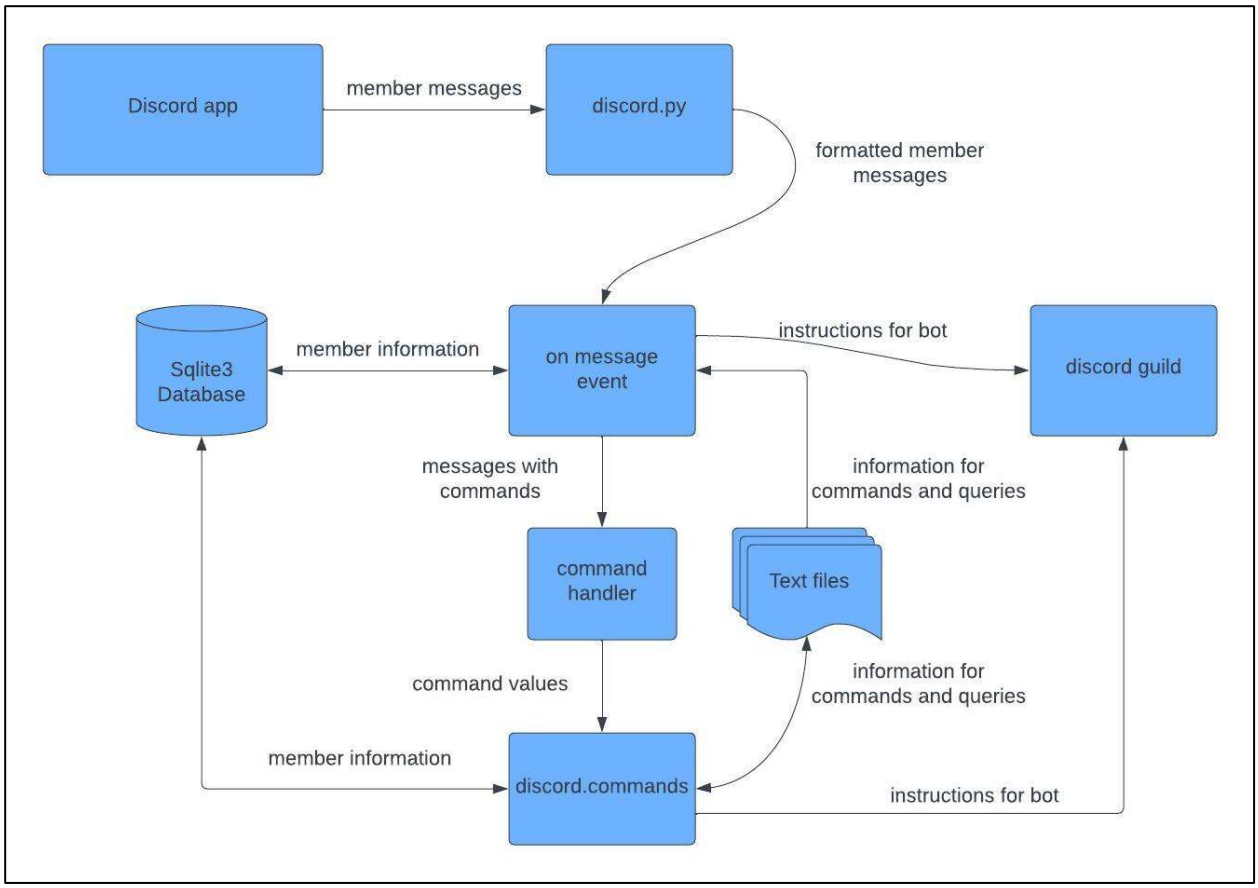
Criterion	Months					
	Jan	Feb	Mar	Abr	May	June
A						
B						
C						
D						
E						

Data Flow Diagrams:

Level 0:



Level 1:



Database Normalization:

The first values of information I wanted to save are listed below:

- Member ID
- Username
- Number of warnings
- Reason of warning
- Message of warning
- Time of warning
- Coins
- Reactions
- Experience
- Rank

	A
1	member id
2	username
3	number of warnings
4	reason of warning
5	message of warning
6	time of warning
7	coins
8	reactions
9	experiennce
10	rank

Since the member ID was the Primary Key (PK) I had to move the reason, message, and time of warning to another table since they can be repeated as a member can have many warnings.

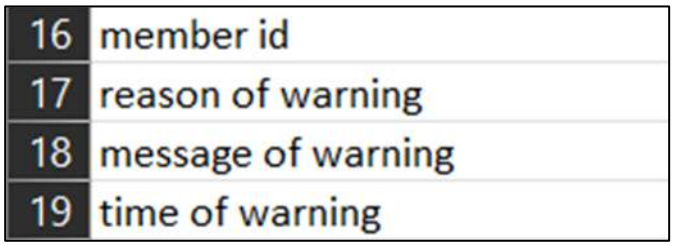
The same idea goes with the reactions. Then I added the member ID as a Foreign Key (FK) in the other tables.

- Member ID
- Username
- Number of warnings
- Coins
- Experience
- Rank

	A
1	member id
2	username
3	number of warnings
4	coins
5	experiennce
6	rank

- Member ID
- Reactions

- Member ID
- Reason of warning
- Message of warning
- Time of warning



Then I realised I wanted to keep the time and date of warning and I can't put different type of data in one value, so I made two columns.

I also had to think of a way to keep all the possible ranks to the database, so I moved the ranks value to another table and made the experience value a FK.

Then I added columns to keep track of the messages and voice calls to add coins and experience to users.

- Member ID
- Username
- Number of warnings
- Coins
- Experience
- Number of messages
- Voice call connection

	A
1	member id
2	username
3	number of warnings
4	coins
5	experiennce
6	number of messages
7	voice call connection

- Experience
- Rank

24	experiennce
25	rank

- Member ID
- Reactions

10	member id
11	reactions

- Member ID
- Reason of warning
- Message of warning
- Date of warning
- Time of warning

16	member id
17	reason of warning
18	message of warning
19	date of warning
20	time of warning

The database will have 4 tables with the PK and Composite Keys (CK) as shown below:

- Members: PK as the member ID
- Ranks: PK as the experience referencing foreign key from members
- Roles: CK member ID referencing foreign key from members and reactions
- Warnings: CK member ID referencing foreign key from members, date, and time

	A		B		C	
1	member id	PK	member id	PK	member id	PK
2	reactions	PK	username		reason of warning	
3			number of warnings		message of warning	
4			coins		date of warning	PK
5	experiennce	PK	experiennce		time of warning	PK
6	rank		number of messages			
7			voice call connection			

Data Dictionary:

Members table:

When a member joins the guild, their data is written to this table. Contains the PK member ID which is the smallest possible integer. The username is a string with the members name.
The rest of the values are integers which are initially set to 0 when the member joins the guild and increase as the member uses the guild

Field	Type	Format	Size	Description	Validation
Member ID	integer	XXXX	4 digits	ID of the user	Is it an integer Is it repeated
Username	string	X…X	Length of name + 5 letters	Username of member	Is it a string Is it repeated
Number of warnings	integer	X	1 digit	Number of warnings the user has	Is it an integer Smaller than 2
Coins	integer	XXX	4 digits	How may coins the member has	Is it an integer
Experience	integer	XX	2 digits	How much experience the member has	Is it an integer Bigger than 200
Number of messages	integer	X	1 digit	Number of messages	Is it an integer Bigger than 10

Voice call connections	float	XXXXXXXXXX.XXXXX	16 digits	Time of connection	Is it a float

Example table:

member_ID	username	num_warn	coins	exp	num_mess	vc_conn
1	Joesef	0	150	36	8	3

Roles table:

This table contains the member ID of the user and the id of the emoji reactions they have.
This is a CK as a member can react many times but can't react two times to the same emoji.
When they react to the reactions it adds the ID of the member and the id of the emoji.
When the member removes the reaction, it deletes that record from the table.

Field	Type	Format	Size	Description	Validation
Member ID	integer	XXXX	4 digits	ID of the user	Is it an integer Is it repeated
Reaction	integer	XXXXXXXXXXXXXXXXXX	19 digits	ID of emoji of reaction	Is it an integer

Example table:

Member_ID	reaction
18	**1100538721310359653**

Warnings table:

It contains the information when a member sends a disrespectful message.
It holds the date and time and member ID which are a CK as a member can't receive two warnings at the same time, but they can have many warnings in the same day, or same time in different days.
It also has the reason of the warning and the message they sent.

Field	Type	Format	Size	Description	Validation
Member ID	integer	XXXX	4 digits	ID of the user	Is it an integer Is it repeated
Reason	string	XXX…XXXX	~10 letters	Reason of the warning	Is it a string
Message	string	XX…XXX	Length of message	Message sent by member	Is it a string
Date	string	XX/XX/XX	8 letters	Date of warning	Is it a string
Time	string	XX:XX:XX	8 letters	Time of warning	Is it a string

Example table:

member_ID	reason	message_sent	date	time
357	insulting	f***	23/05/2023	18:46:19

Ranks table:

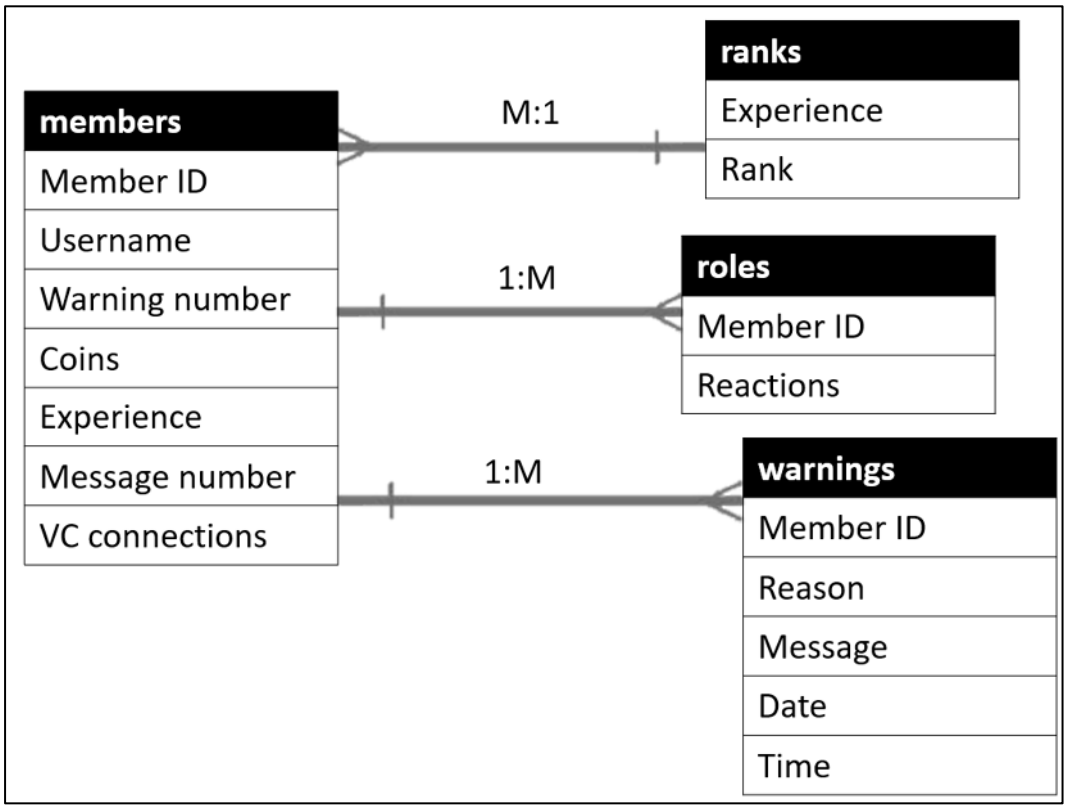
The PK is the experience which allows the members to see their rank from the experience they have as it references the members table.
The experience starts at 0 and the maximum value is 200. Every 10 experience there is a new rank.

Field	Type	Format	Size	Description	Validation
Experience	integer	XXX	3 digits	Experience needed for that rank	Is it an integer
Rank	string	XX…XX	Length of name	Name of rank	Is it a string

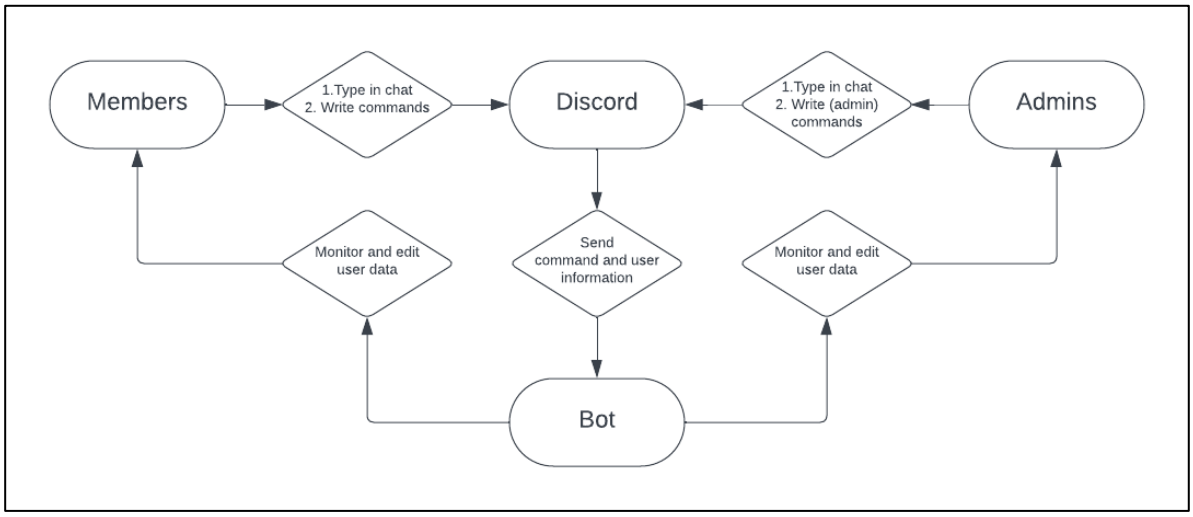
Example table:

experience	rank
10	Novice

Database table interaction:



Entity Relationship Diagram:



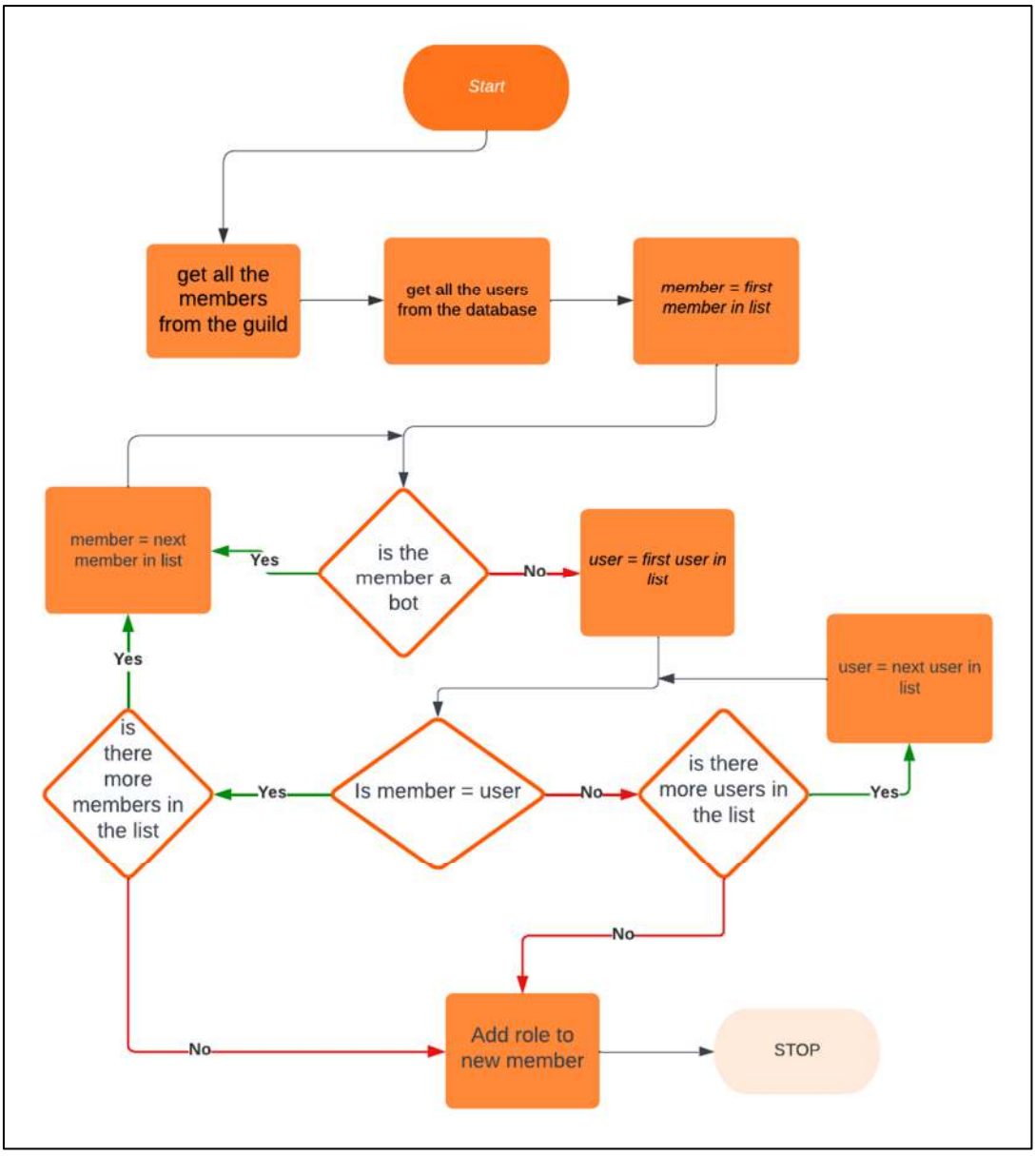
Product functionality:

- The system will be linked to discord through the discord.py library included in my program.
- The data type is specific to the app so it will be changed to string or integer when writing into the database.
- The library includes a send function which converts the strings to the discord data type and sends it to the guild.
- The members will have to put a “!” in front of the message to state it’s a command which the program will detect and process the command.
- Members use the discord app so the user experience will not change from other guilds as it is the same GUI and bots are used similarly.
- The difference between each bot is the use of commands, so a help command can be used to explain every other command.

Member adding algorithm:

When a new member joins the guild, this algorithm is used. If the member is not in the database, it activates the algorithm. The member can be in the database already if it was in the guild before and then abandoned the guild.

Flowchart:

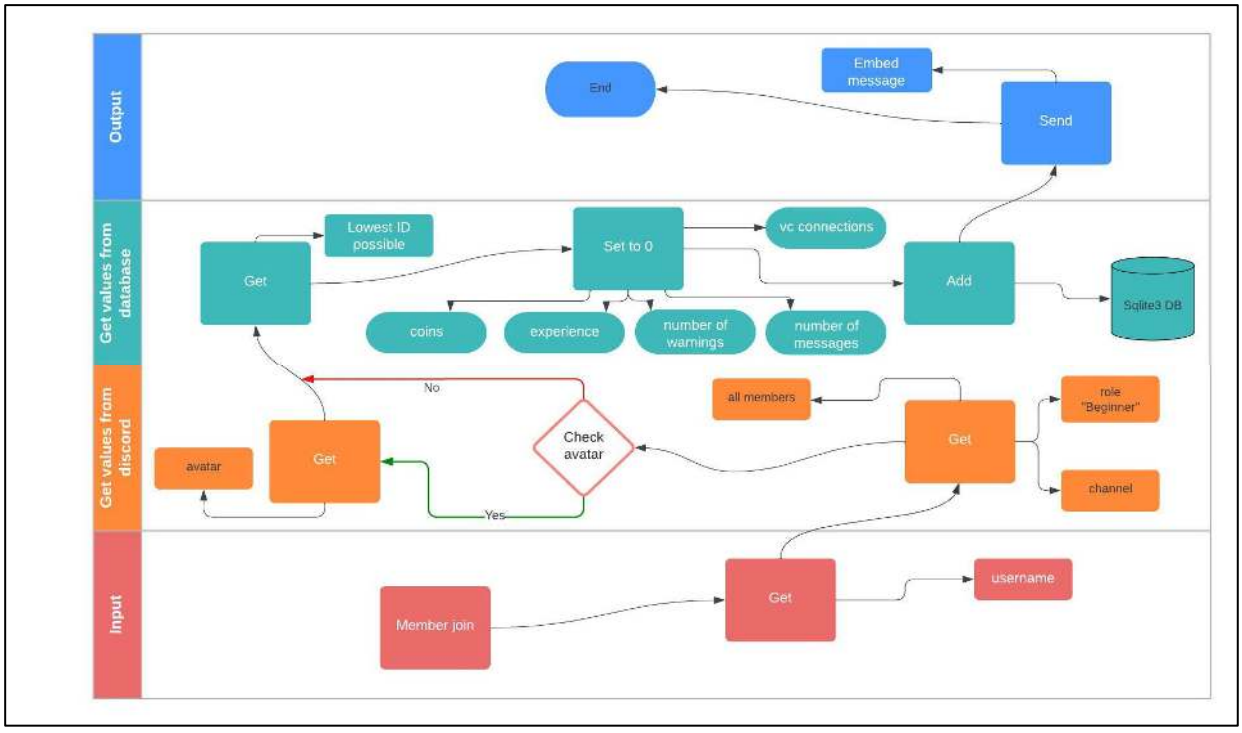


Pseudocode:

```
loop for member = members in guild # loops through every member
      if member is a bot then
            next member in loop
      end if
```

```
        loop for username = members in database # loops through every
username
            if member = username then # checks member is in database
                next username in loop
                repeated = True
            end if
        end for loop
        if repeated then
            next member in loop
        end if
        # sets all the values and adds member to database
        set values to 0
        set member id to highest in database + 1
        set member name to member
        add (values, member name, member id) to database
end for loop
```

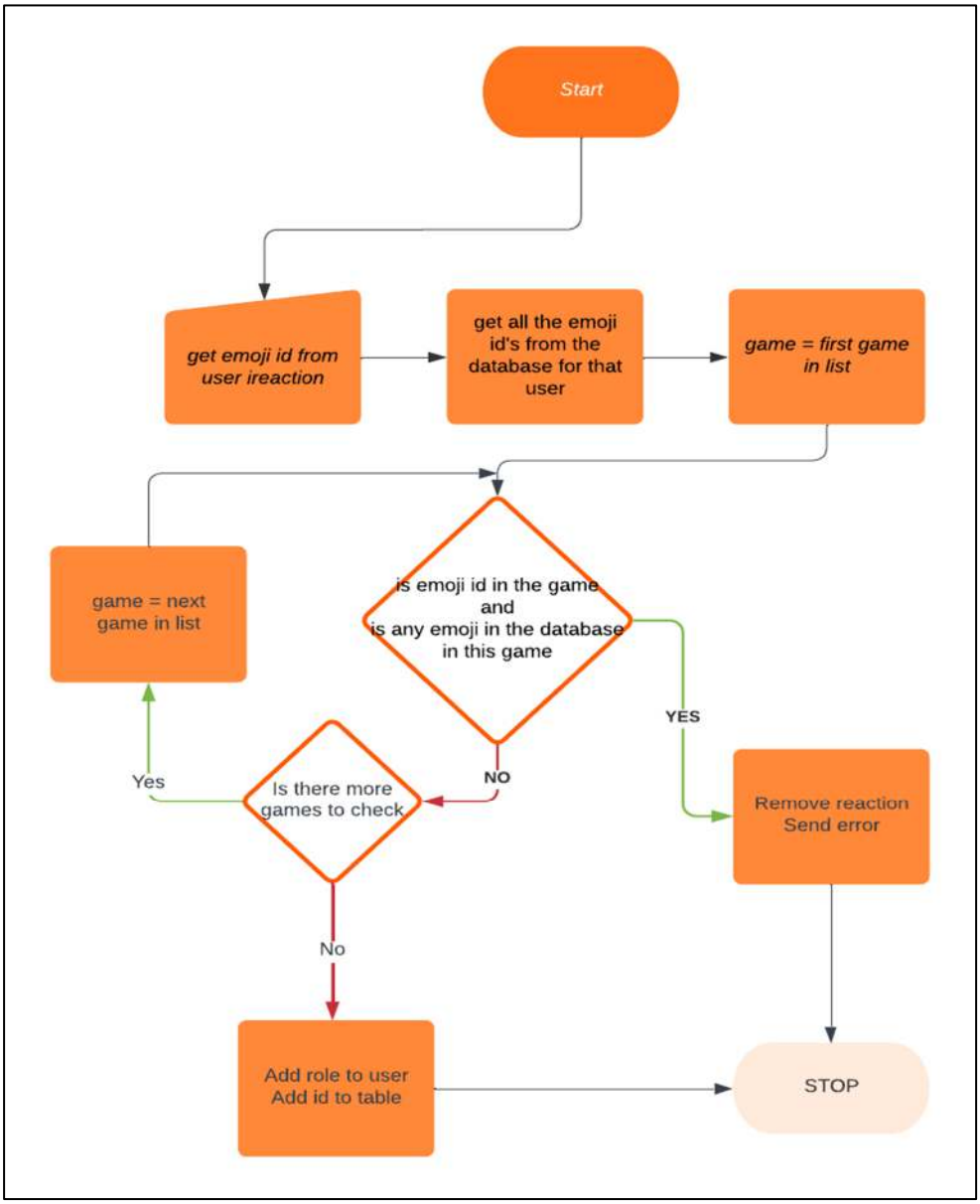
Structure Design:



Emoji id algorithm:

This technique is used when a user reacts with an emoji to the embed message sent in the wanted channel of the guild.

Flowchart:

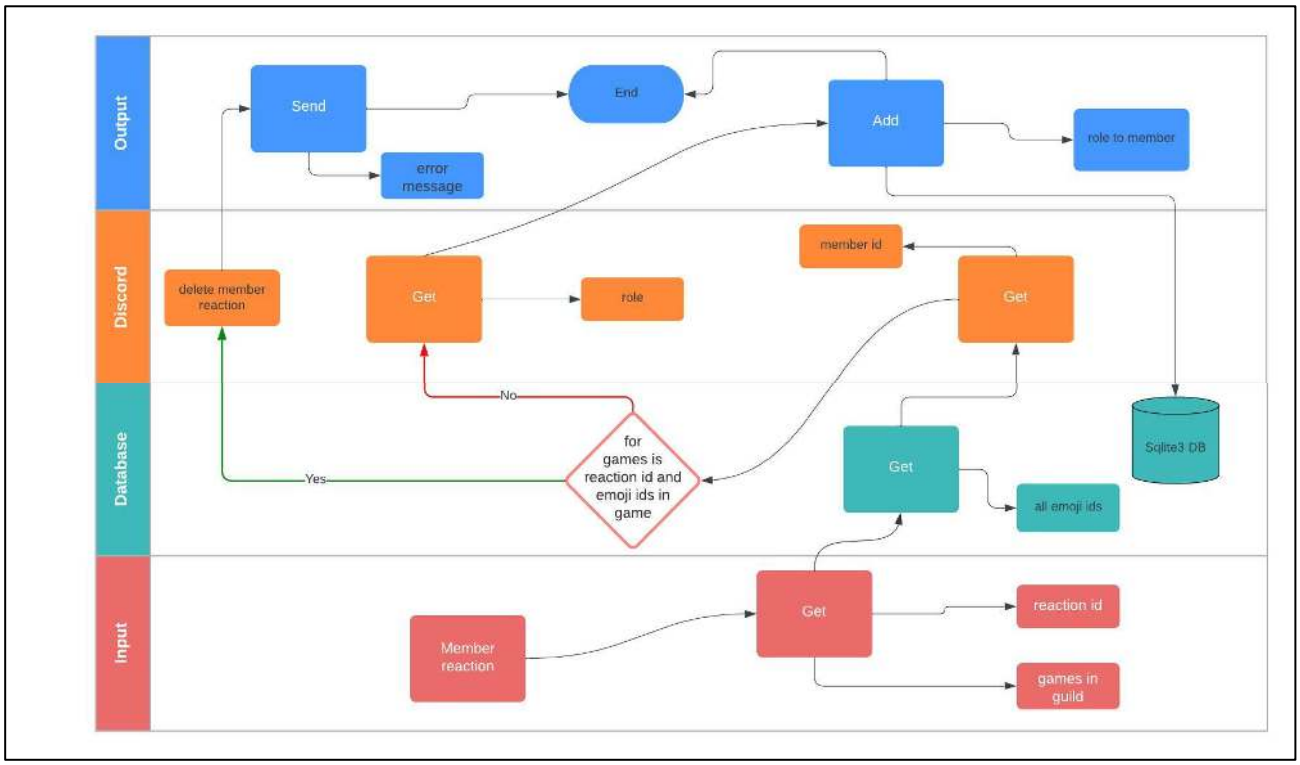


Pseudocode:

```
# gets information from the guild
Emoji id = id of emoji from reaction message
emojis = all id’s in database for user
games = games in guild
loop for x = game in games # loops through every game in the guild
```

```
        loop for i = emoji id in emojis
                if (emoji_id in x) and (i in x) then # checks the member
hasn't reacted in that game before
                        remove reaction of member
                        send error message
                else
                        add role to user
                        add id to user in database
                end if
        end for loop
end for loop
```

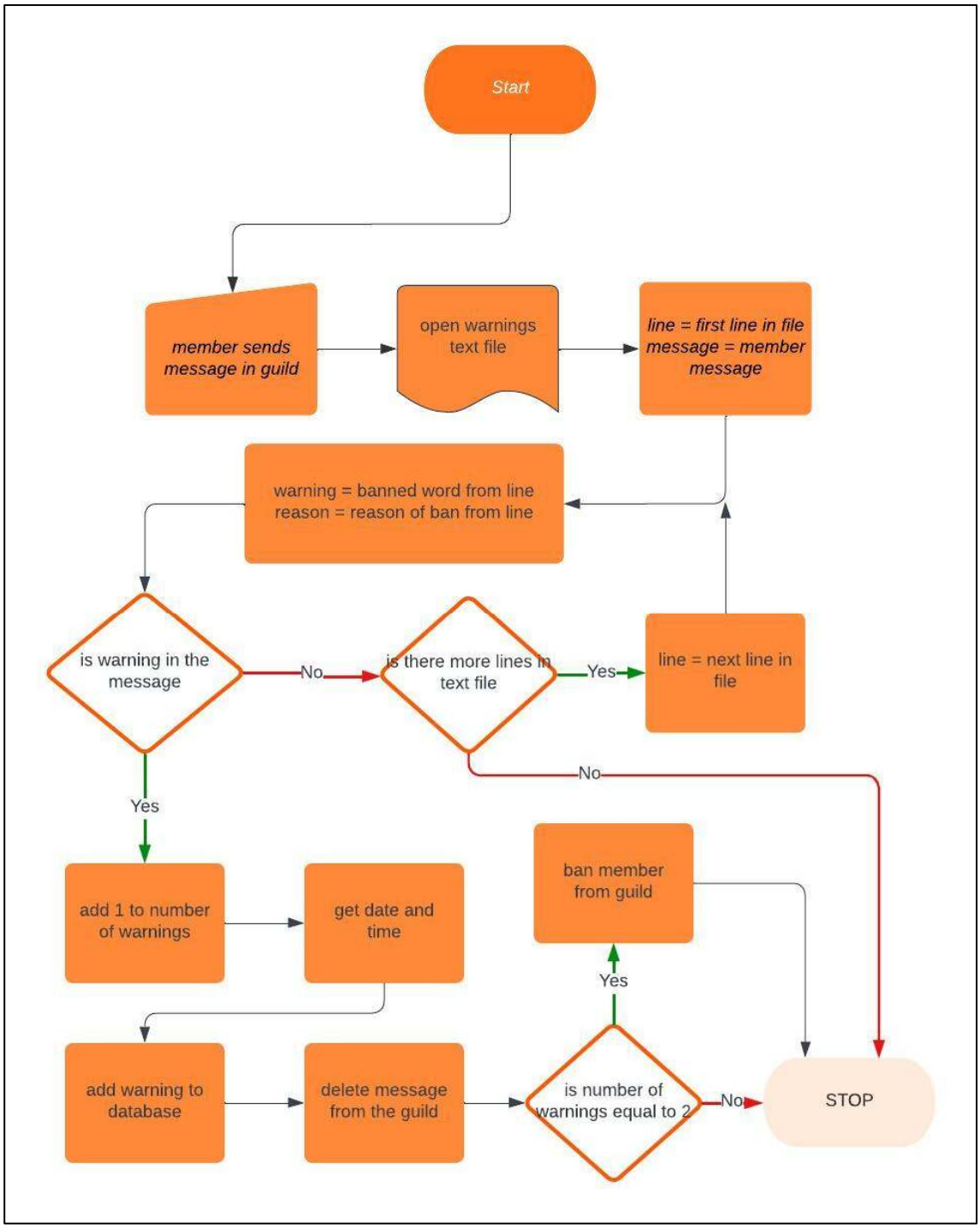
Structure diagram:



Warning algorithm:

This algorithm activates when a member sends a message and gets all the banned words from a text file and check if the message of the user contains one of these words. After, it gets the message sent from the member, the reason of warning from the text file and the date and time to add the warning to the database.

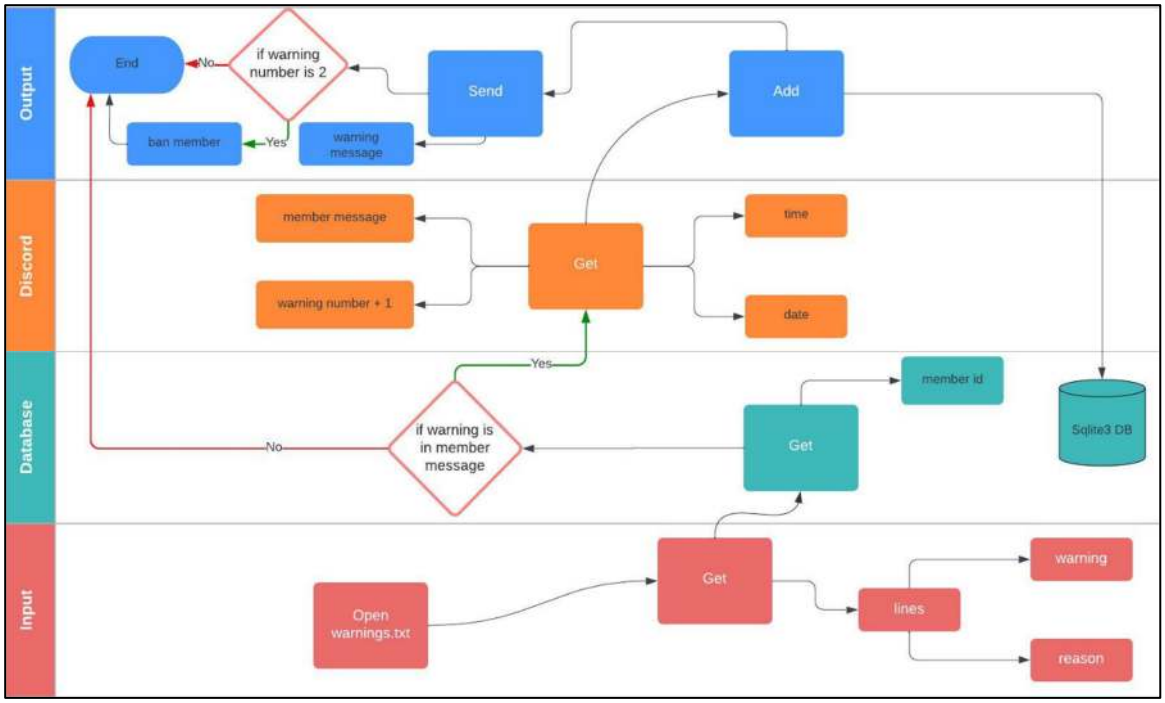
Flowchart:



Pseudocode:

```
open warnings.txt # opens the local text file used for warnings
message = member message
loop for line = lines in warnings # loops through every line in the warnings file
      warning = banned word in line
      reason = reason of ban in line
      if warning in message # checks if the message has the banned word
            warning_number = warning_number in database
            warning_number = warning_number + 1 # adds one to the warning number in database
            date = date now
            time = time now
            add to database # updates the database
            delete message from guild
            if warning number = 2 then
                  ban member
            end if
            stop # stops the process as the message contains a banned word
      end if
end for loop
```

Structure diagram:



Error handling design:

I am planning to use the discord.py library which includes an event called “on command error”. This event activates whenever a user has entered a command in the wrong format or has made a mistake when typing the command.

Pseudocode:

```
on command error # stating the event which activates when an error is found
      error = error in command
      # checks for the type of error and sends a message to the member
      if error is missing permissions
            send “You don’t have the permissions needed”
      elseif error is command is not found
            send “That command doesn’t exist”
      elseif error is member is not found
            send “This member is not in the server”
      end if
```

The member adding algorithm includes a loop to check if members are already in the database. Therefore, the program doesn’t try to write a PK twice in a table.

In the emoji id algorithm, it checks whether the member already has an emoji for that game. It is impossible to have two ranks in the same game, so the program denies a member of having two roles referring to two ranks in a game.

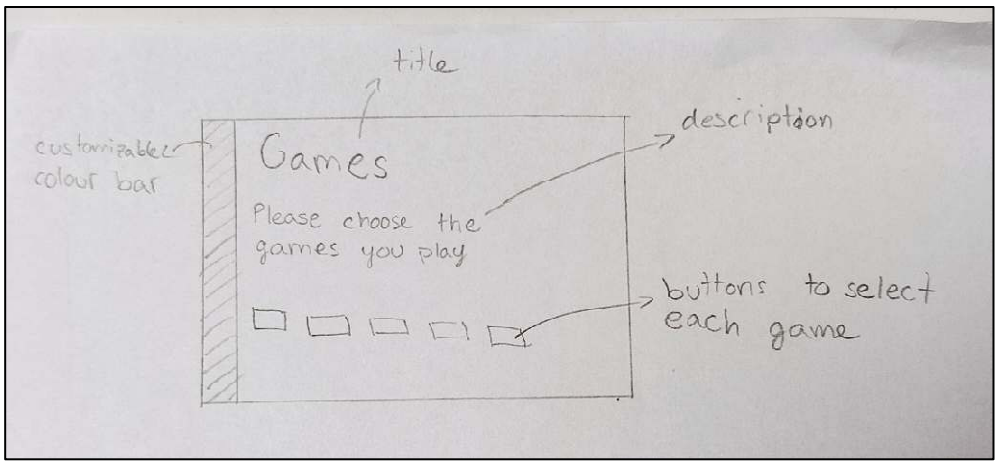
When a member tries to add values to a text file it checks the format and sends an error message if what the member is trying to add doesn’t meet the standard format of the file.

Formatting:

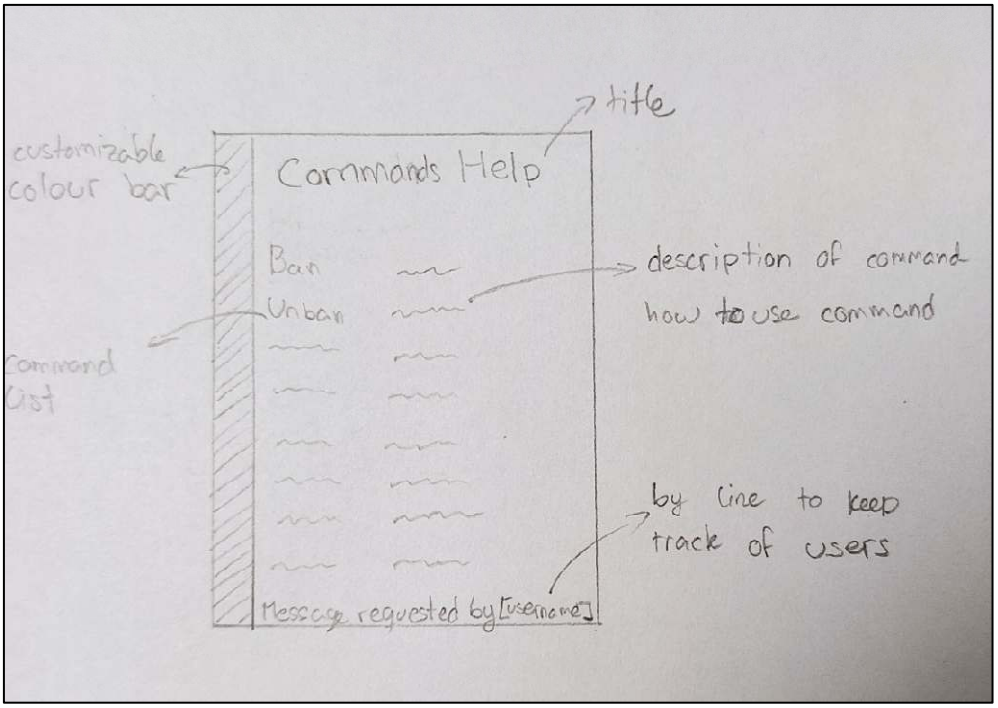
The data sent to the guild chat will have to be formatted. When retrieving the rank or warning data set for the members I will have to format the string, so it makes the message easier to read. This aids the user experience in the guild.

Embed messages will be formatted to my client's preference. Which also help create a better user experience when using the guild. Some of the embed messages are shown in the sketches below.

Games command sketch to display embed message that will allow users to automatically get roles for different games.



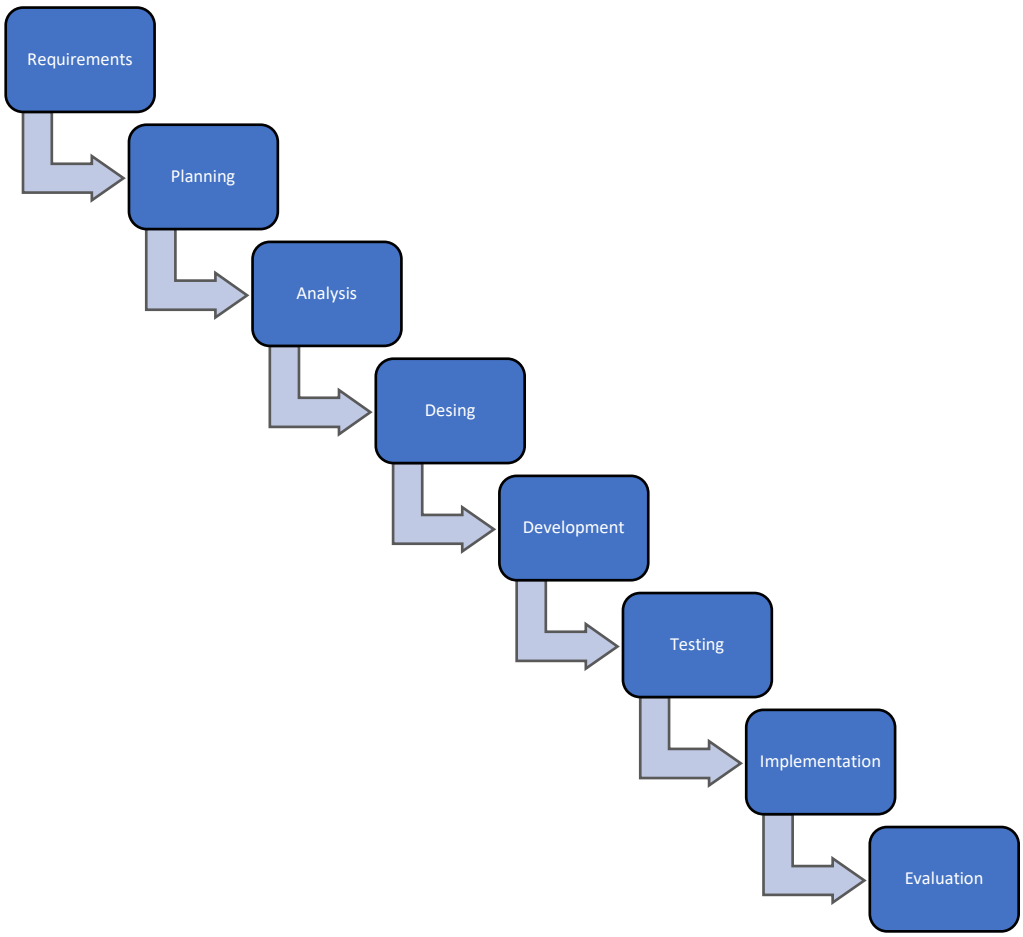
Help command that will display an embed message to give users information about the syntax of every command.



Design Methodology:

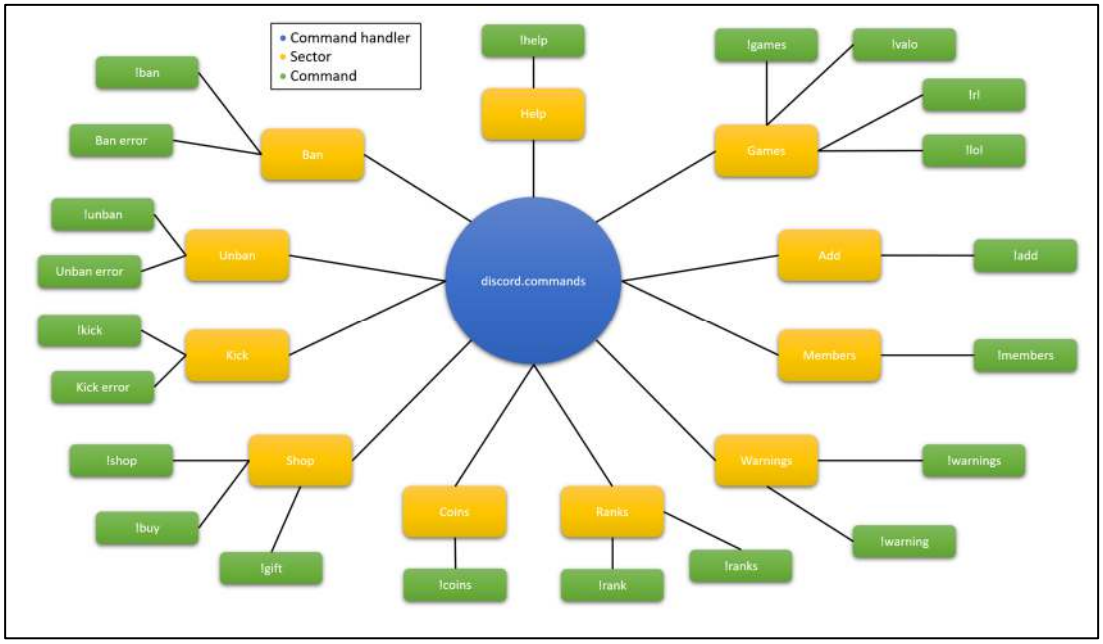
I will use a procedural programming method to approach my project. The program procedure will contain subroutines in a series of steps. Any subroutine will be able to be called at any given point while the program is active.

Additionally, the project is based on the waterfall model. It started in the interview with the client to find the requirements and will end with the implementation and evaluation of the product.



Spider diagram for commands:

This will be all the commands available for the members. The bot will process these commands with the command handler and send an output message to the user telling if the command was processed correctly or incorrectly.



Extensibility:

- The program file will have a structure that allows easy adding new commands and events.
- I will separate the bot events and the commands in two sections in the file.
- Every command and event will be in a separated subroutine. Having many functions will allow to change an individual event or command if needed, and multiple people could change different subroutines at the same time.
- When the discord app is updated, the bot doesn't need to be changed. All the compatibility issues by the discord.py library in my program and the app, are fixed by discord.
- All commands, events, loops, queries, and statements will be separated by a line which clearly indicates it's a new function.
- Every variable will have a proper name to the data it handles and will be lower case, and every two-word variable will be separated by a "_". Keeping the same format will ease the reading of the code to allow better understanding.
- Database will be always connected in the top of the event or command and closed at the end.

- ⊂ Member data will have to be normalised to import to database. So I will create dictionaries to store each member data set and use the SQLite3 functions to store the data in the dictionary to the database.
- ⊂ Normalised Database tables will be created at the start of the code, this will make sure no structure change is necessary in the future
- ⊂ Code will have comments thought to explain every technique and how it works.
- ⊂ The code will stop the loop when the member is found

Test plan:

Action to test	Method of testing	Input data	Expected result	Output	Action taken	Success criteria
The system can be used from all devices	Open the guild through different devices and test the commands	roll	With the same input the output result should be the same for the devices (e.g. data stored in the database and message sent by the bot)	A number between 1 and 10	n/a	1
Help is available for users for how to use the commands.	Use the help command in the guild	!help	Every command useful to the member is sent to the guild	An embed message with all the commands and how to use them	n/a	2
System will always be online, so it doesn't matter the time the user needs to use the system.	Run the bot and wait a few hours	n/a	The bot is still online in the guild available to use, and the code is running	n/a	n/a	3
The users can create public and private conversations with the system in the chat.	Use a "?" prefix before the message	?roll	The bot creates a private conversation and sends the expected answer	Number between 1 and 10 in a private conversation	n/a	4
System gets data from discord and saves it into database and vice versa.	Use different commands that require connection to the database	!members	The bot should easily retrieve data from the guild and database without any errors	List of all the members saved into the database is sent in the guild	n/a	5

The administrators of the guild can easily ban or unban other users without going into the settings of the guild and searching for the specific user which saves time.	Use the ban/unban command with an admin and a non-admin	!ban @[member] !unban [member]	The bot processes the command for the admin member and sends an error message for the non-admin member	@[member] has been banned @[member] has been unbanned	n/a	6
After 2 warnings members should be banned.	Use a test member to write two messages with banned words in the guild	F***	The bot should send a warning to the member for first message and directly ban the member for the second message	@[member]	n/a	7
A member should be given rewards for using the server.	Use a test member to write many messages in the guild	Any message	The bot should add coins and experience to the member for sending many messages and store those in the database	New number of coins and experience stored in database	n/a	8
Fast response time in the chat.	Tell more than one member to send a message or command at the same time	1 Roll 2 !help 3 !members	The bot responds to all members rapidly	1 number between 1 and 10 2 list of all commands 3 list of members	n/a	9
Error handling to stop the program from crashing if users enter silly entries.	Tell members to write commands without knowing the correct input. This gets as many unexpected inputs as possible for the program	!ban [no member] !rank [no member] ![command that doesn't exist]	The bot handles all the incorrect inputs and sends an error message to the guild. The bot continues to work efficiently after these entries.	Member not found Command not found	n/a	10

Finish the system before the 30th of June	Have every success criteria done without errors before the deadline	n/a	n/a	n/a	n/a	11
On member join event	Different members join the guild. Same member joins several times	n/a	The members are automatically added to the database with a unique member id and a message is sent to the guild	Welcome message sent in guild	n/a	
On message event	Send a message with a banned word and another with a word that the bot responds to	Roll F***	If the message contains a banned word, it should add a warning to the user and send a warning message, if the message has a word that the bot responds to it should send the response to the guild. If warning number is 2 then ban the member	Responds to message or deletes the message and sends warning or/and ban message	n/a	7
Reaction add event	Add one reaction to a message, then add a second reaction to the same message	Add reaction	The member is given the role linked to that emoji reaction and if the member already has one it will delete the reaction and send an error message	Add role You can't have two roles for the same game	n/a	
Reaction remove event	Remove the reaction from one of the messages	Remove reaction	The role linked with that emoji is removed from that member	Remove role	n/a	
User update event	A member changes their username	n/a	The member's name is changed in the database	n/a	n/a	
Voice state update event	Members join a voice call and leave at different times	n/a	Different amounts of coins and experience are awarded to the members	The new number is stored in the database	n/a	8

			depending on time spent			
Member ban event	Ban a member from the server	!ban @[member]	Their records are deleted from the database	[member] has been banned	n/a	
Command error event	Input a command incorrectly. Telling many members to input the commands to have the most possible errors.	!ban [no member] !rank [no member] ![command that doesn't exist]	All the errors made by the members are sent in a message, saying the input command is wrong.	Member not found Command not found	n/a	10
Games command	Put the command in the guild	!games	The embed message is sent in the correct channel with the correct format and all reactions are added	Embed message sent with all reactions	n/a	
Add command	Input correct and incorrect formats	!add [file name] [text] !add [file name] [no text] !add [no file name]	The command denies incorrect formats and sends an error message and writes the correct format to the files	Text has been added Input text to add Input a correct file name	n/a	
Ban command	Ban a member from the guild. Ban a member not in the guild	!ban [member] !ban	If the member is in the guild, it is banned and if the member is not in the guild, it sends an error message	@[member] has been banned Member not found in guild	n/a	6
Unban command	Unban a banned member and unban a member in the guild	!unban [banned member] !unban [not banned member]	If the member is not banned it will send an error message, otherwise it will unban the member	@[member] has been unbanned Member not banned	n/a	6
Kick command	Kick a member from the guild and a member not in the guild	!kick [member] !kick	The member is kicked if it's in the guild if it's not, send an error message	@[member] has been kicked Member not found in guild	n/a	

Members command	Use command in guild	!members	The list of every ember in the guild is sent	List of all members	n/a	
Warnings command	Use command in guild by an admin and non-admin member	!warnings	The list of every person's warning number is sent	List of all members with warnings from high to low [member] you don't have permission for this command	n/a	
Warning command	Target a member in the guild and a member not in the guild	!warning [member] !warning [member not in guild]	If the member is not in the guild send an error message if the member is in the guild a message is sent with the details of every warning of that member. If the member is an admin, it can target another member	List of all warnings for member Member not found	n/a	
Ranks command	Use command in guild by an admin and non-admin member	!ranks	The list of every person's rank is sent	List of all members high to low ranks [member] you don't have permission for this command	n/a	
Rank command	Target a member in the guild and a member not in the guild	!rank [member] !rank [member not in guild]	If the member is not in the guild send an error message if the member is in the guild a message is sent with that member's rank. If the member is an admin, it can target another member	Rank for member Member not found	n/a	
Shop command	Use command in guild	!shop	An embed message is sent with all the items and their prices	Embed message with list of items	n/a	
Buy command	Use command with enough	!buy [item] (with enough	An error message is sent if the member doesn't	[member] you have bought [item]	n/a	

	and not enough coins to buy the item	and not enough coins) !buy [item not in list]	have enough coins, if the member has enough coins a confirmation message is sent, and the coins are reduced by the price	[member] you don't have enough coins [item] not found		
Gift command	Use command with enough and not enough coins to buy the item	!gift [item] [target member] (with enough and not enough coins) !gift [item] [target member not in guild] !gift [item not in list] [member]	An error message is sent if the member doesn't have enough coins, if the member has enough coins a confirmation message is sent, and the coins are reduced by the price	[member] you have bought [item] for [target member] [member] you don't have enough coins Member not found [item] not found	n/a	
Coins command	Use command in guild	!coins	A message with the number of coins the author of the message has is sent	[member] you have [coins number] coins	n/a	