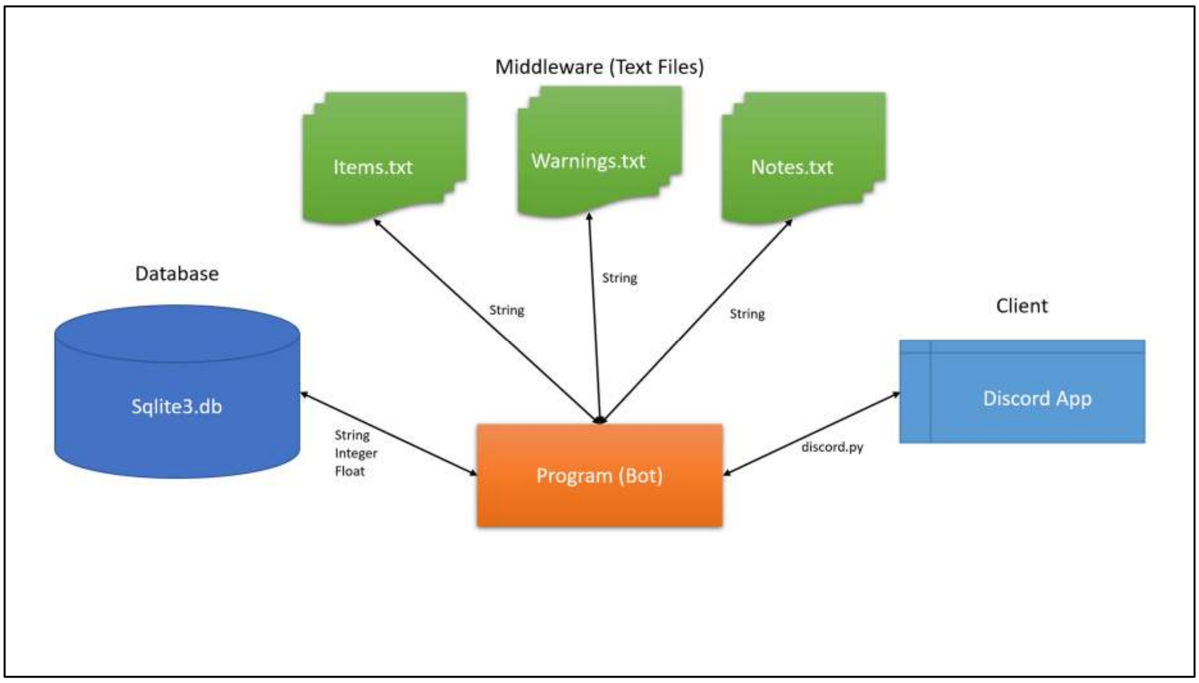
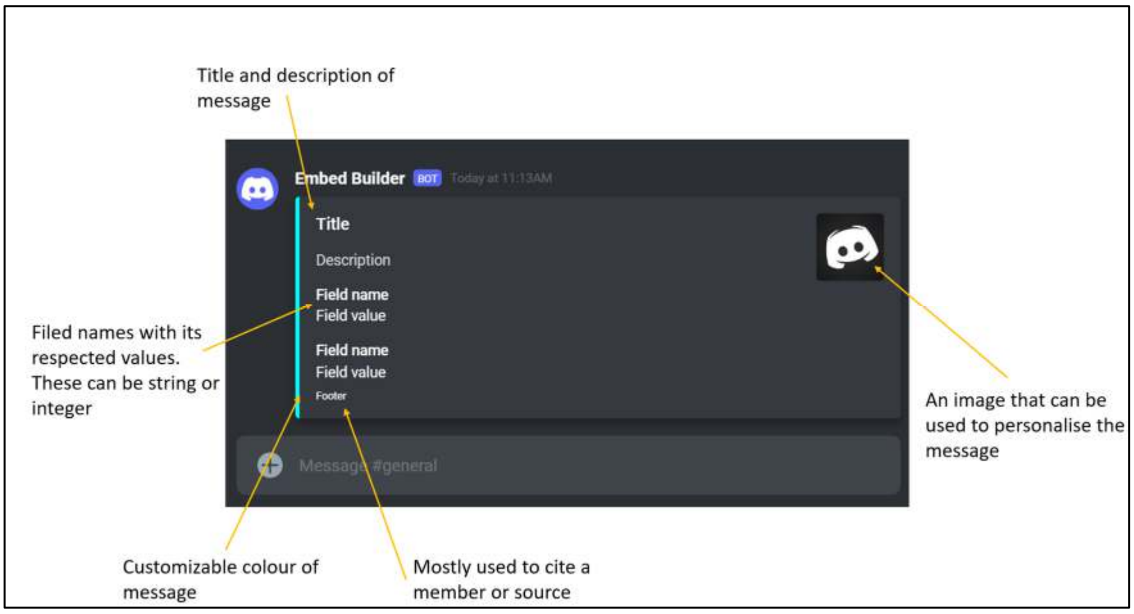
# Design Overview

## Higher level architecture:



## Graphic User Interface (GUI):

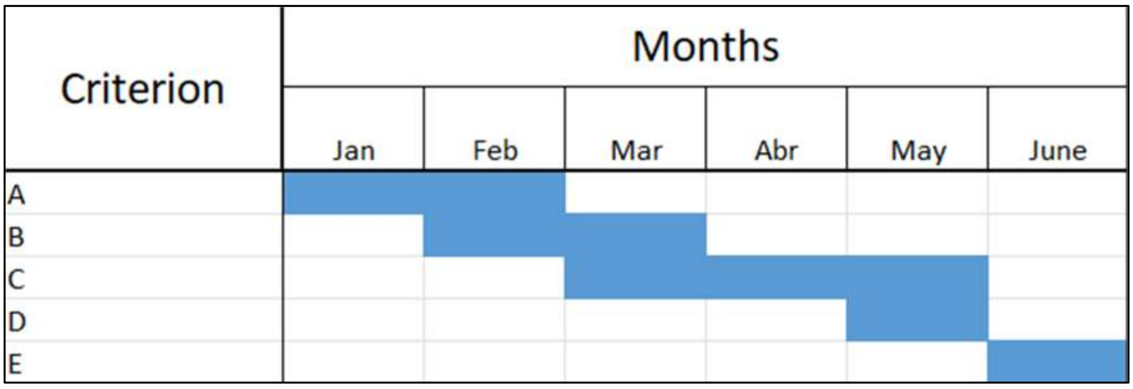
This is a template Embed message from discord. This could be used for more important messages such as the help message. I will format this version to personalise it for different uses. It will be sent to members in the guild when they request information or help.



Title and description of message
Embed Builder
BOT
Today at 11:13AM
Title
Description
Field name
Field value
Field name
Field value
Footer
Message #general
Filed names with its respected values. These can be string or integer
An image that can be used to personalise the message
Customizable colour of message
Mostly used to cite a member or source

## Gantt chart:

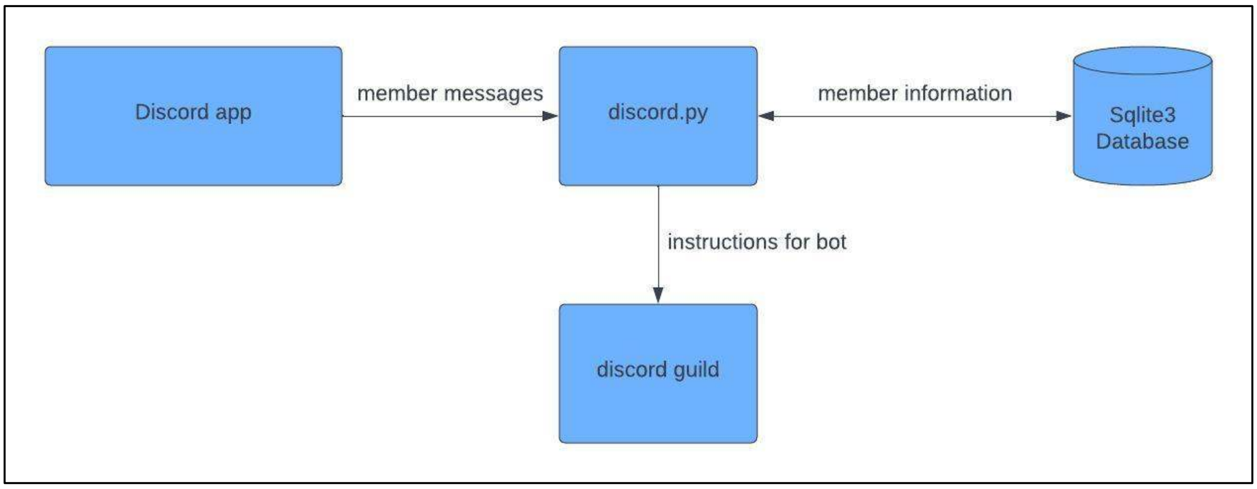
This is my initial plan for every criterion of the project during the six months.



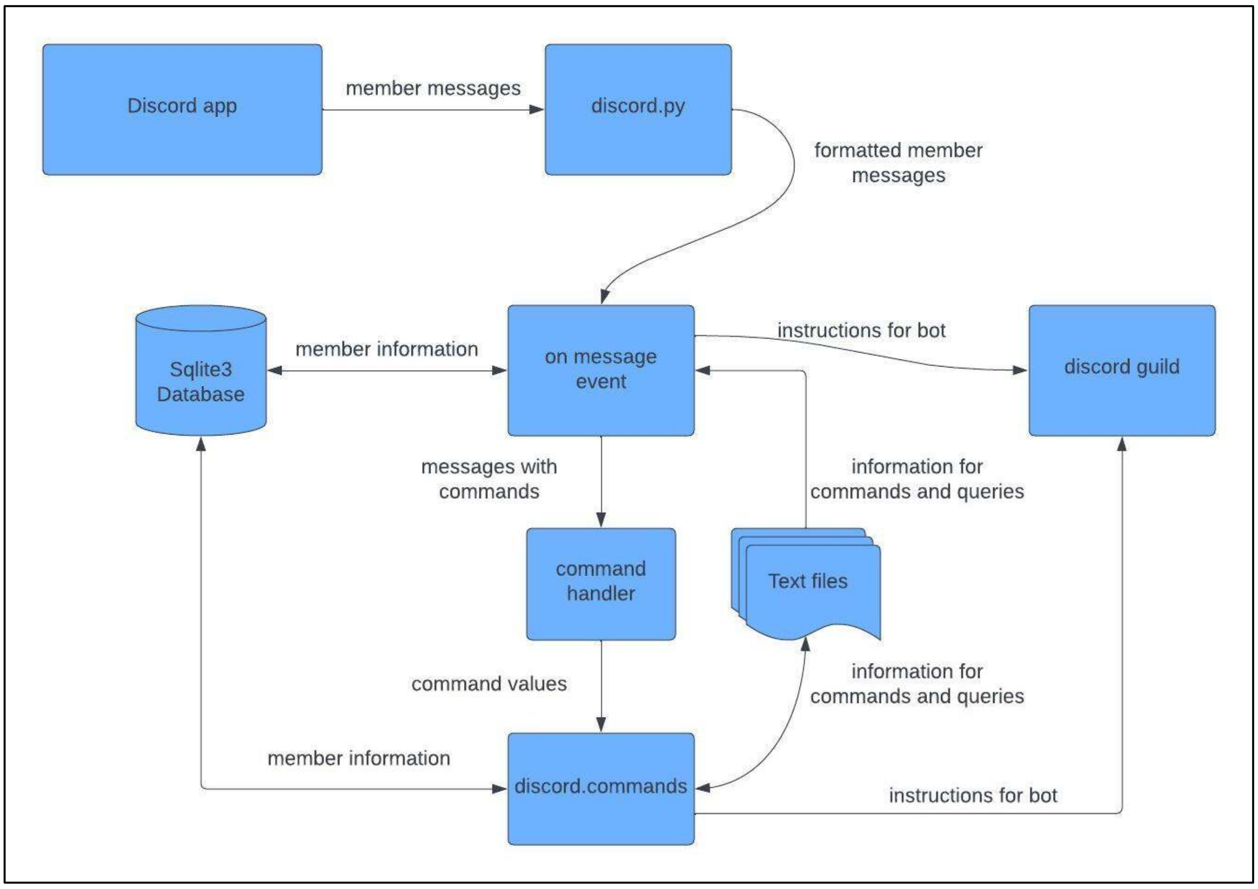
| Criterion | Months | | | | | |
|---|---|---|---|---|---|---|
| | Jan | Feb | Mar | Abr | May | June |
| A | 🟦 | 🟦 | | | | |
| B | | 🟦 | 🟦 | | | |
| C | | | 🟦 | 🟦 | 🟦 | |
| D | | | | | 🟦 | |
| E | | | | | | 🟦 |

## Data Flow Diagrams:

Level 0:



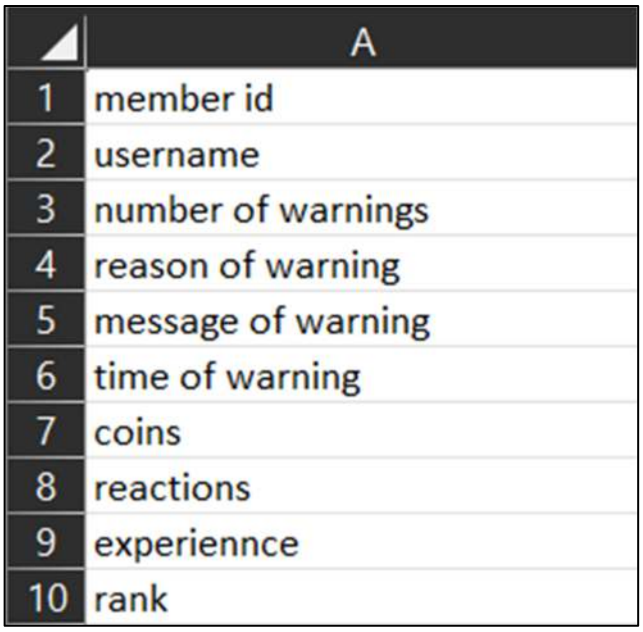
Level 1:



## 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

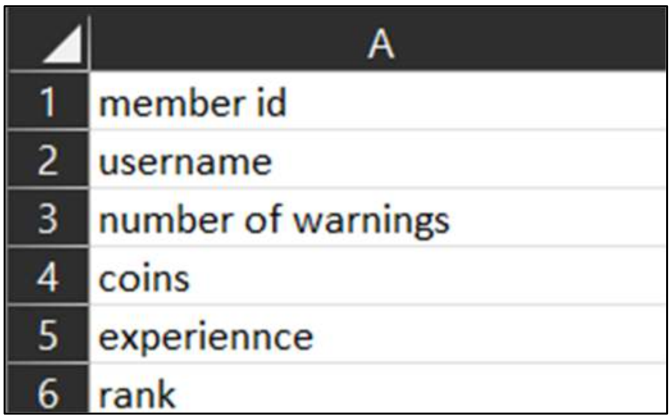


| | A |
|---|---|
| 1 | member id |
| 2 | username |
| 3 | number of warnings |
| 4 | reason of warning |
| 5 | message of warning |
| 6 | time of warning |
| 7 | coins |
| 8 | reactions |
| 9 | experiennce |
| 10 | rank |

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.

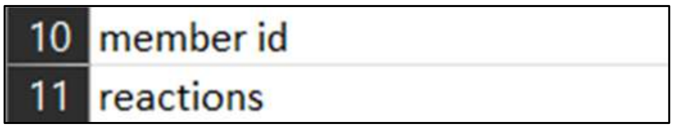
The same idea goes with the reactions. Then I added the member ID as a Foreign Key (FK) in the other tables.

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



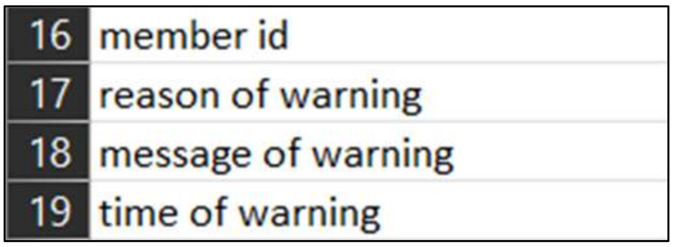
| | A |
|---|---|
| 1 | member id |
| 2 | username |
| 3 | number of warnings |
| 4 | coins |
| 5 | experiennce |
| 6 | rank |

- Member ID
- Reactions



| | |
|---|---|
| 10 | member id |
| 11 | reactions |

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



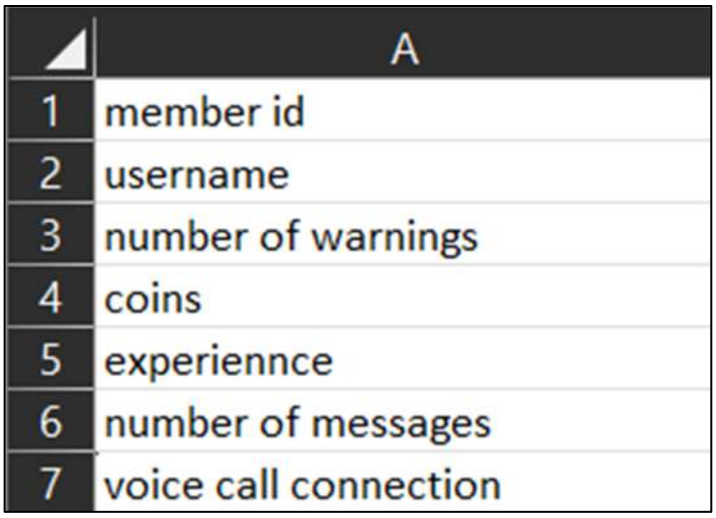
| | |
|---|---|
| 16 | member id |
| 17 | reason of warning |
| 18 | message of warning |
| 19 | time of warning |

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.

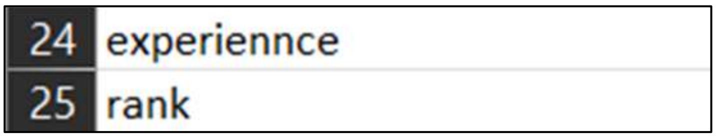
Then I added columns to keep track of the messages and voice calls to add coins and experience to users.

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



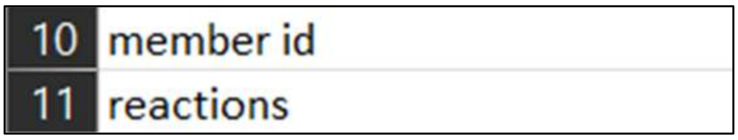
| | A |
|---|---|
| 1 | member id |
| 2 | username |
| 3 | number of warnings |
| 4 | coins |
| 5 | experiennce |
| 6 | number of messages |
| 7 | voice call connection |

- Experience
- Rank



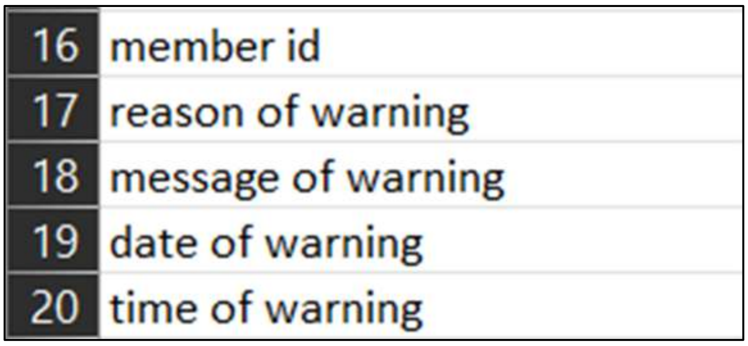
| | |
|---|---|
| 24 | experiennce |
| 25 | rank |

- Member ID
- Reactions



| | |
|---|---|
| 10 | member id |
| 11 | reactions |

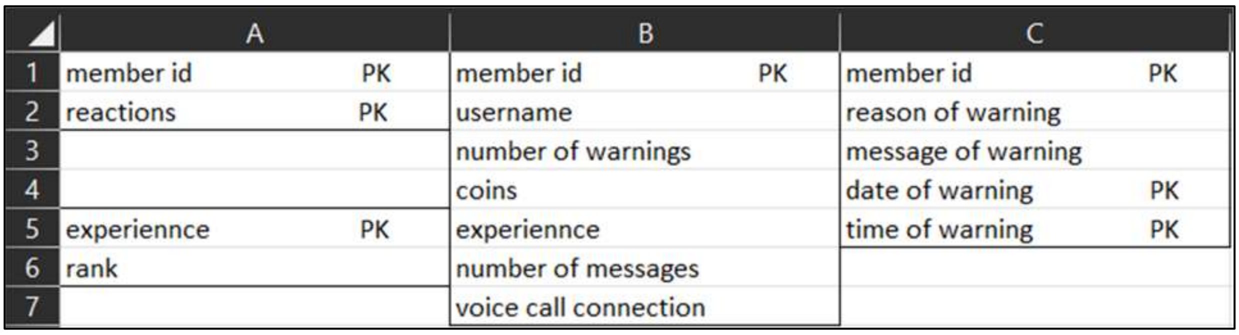
- Member ID
- Reason of warning
- Message of warning
- Date of warning
- Time of warning



| | |
|---|---|
| 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<br>Is it repeated |
| **Username** | string | X…X | Length of name + 5 letters | Username of member | Is it a string<br>Is it repeated |
| **Number of warnings** | integer | X | 1 digit | Number of warnings the user has | Is it an integer<br>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<br>Bigger than 200 |
| **Number of messages** | integer | X | 1 digit | Number of messages | Is it an integer<br>Bigger than 10 |

| | | | | | |
|---|---|---|---|---|---|
| **Voice call connections** | float | XXXXXXXXXX.XXXXXX | 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<br>Is it repeated |
| **Reaction** | integer | XXXXXXXXXXXXXXXXXXX | 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<br>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:

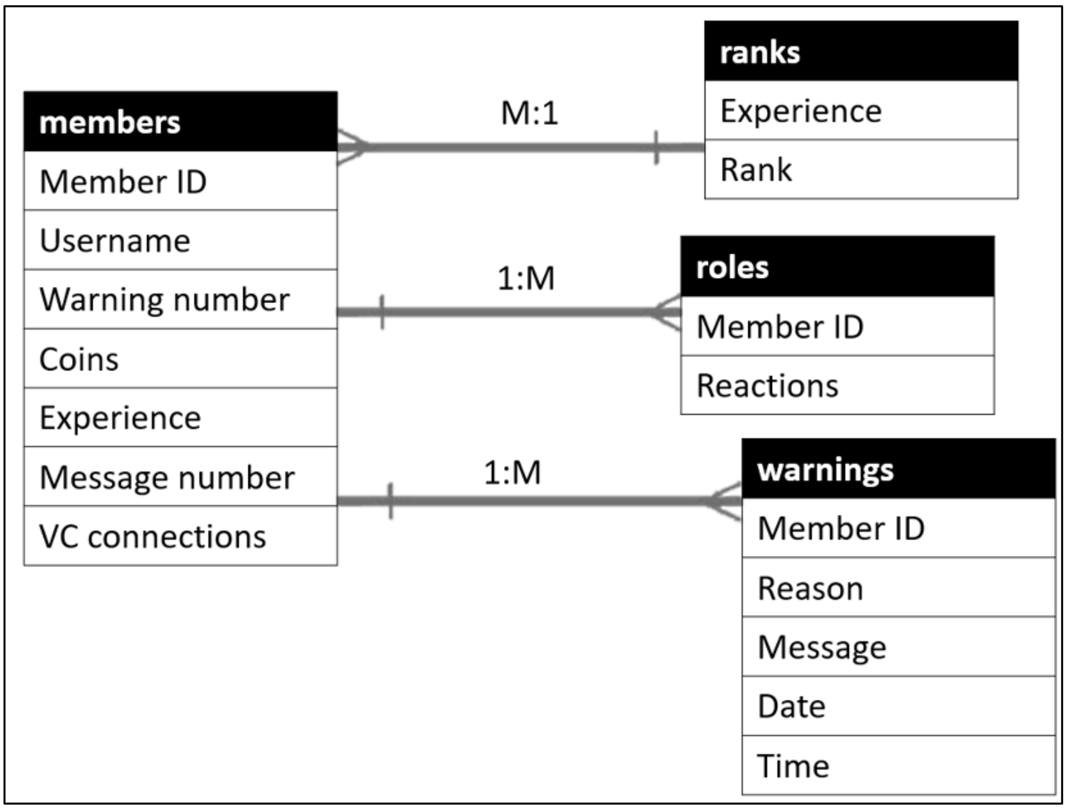
The PK is the experience which allows the members to see their rank from the experience they have as it references the members table.
The experience starts at 0 and the maximum value is 200. Every 10 experience there is a new rank.

| 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 |

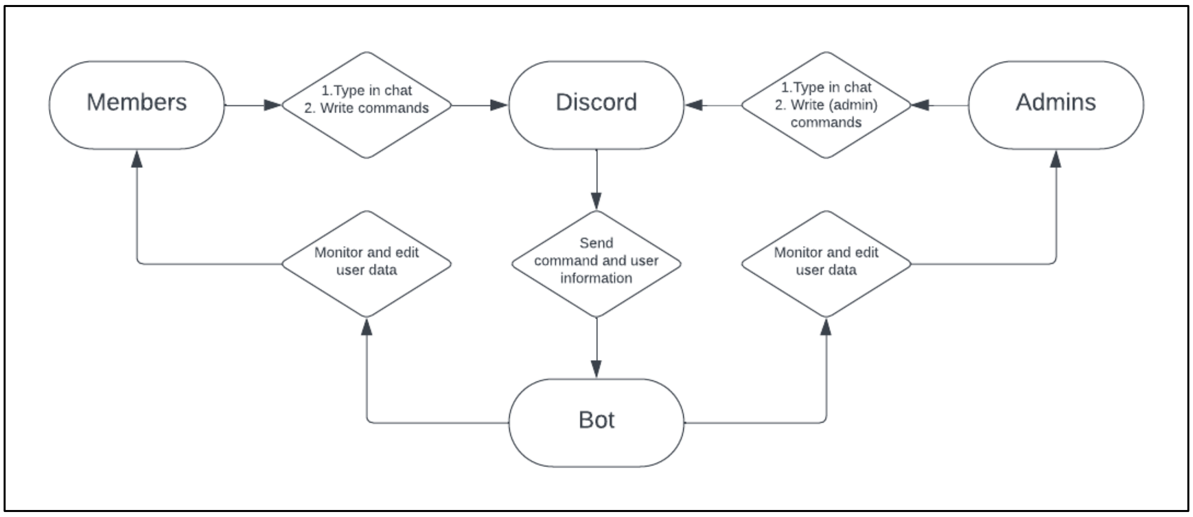
Example table:

| experience | rank |
|---|---|
| **10** | Novice |

Database table interaction:



Entity Relationship Diagram:


Members
1.Type in chat
2. Write commands
Discord
1.Type in chat
2. Write (admin) commands
Admins
Monitor and edit user data
Send command and user information
Monitor and edit user data
Bot

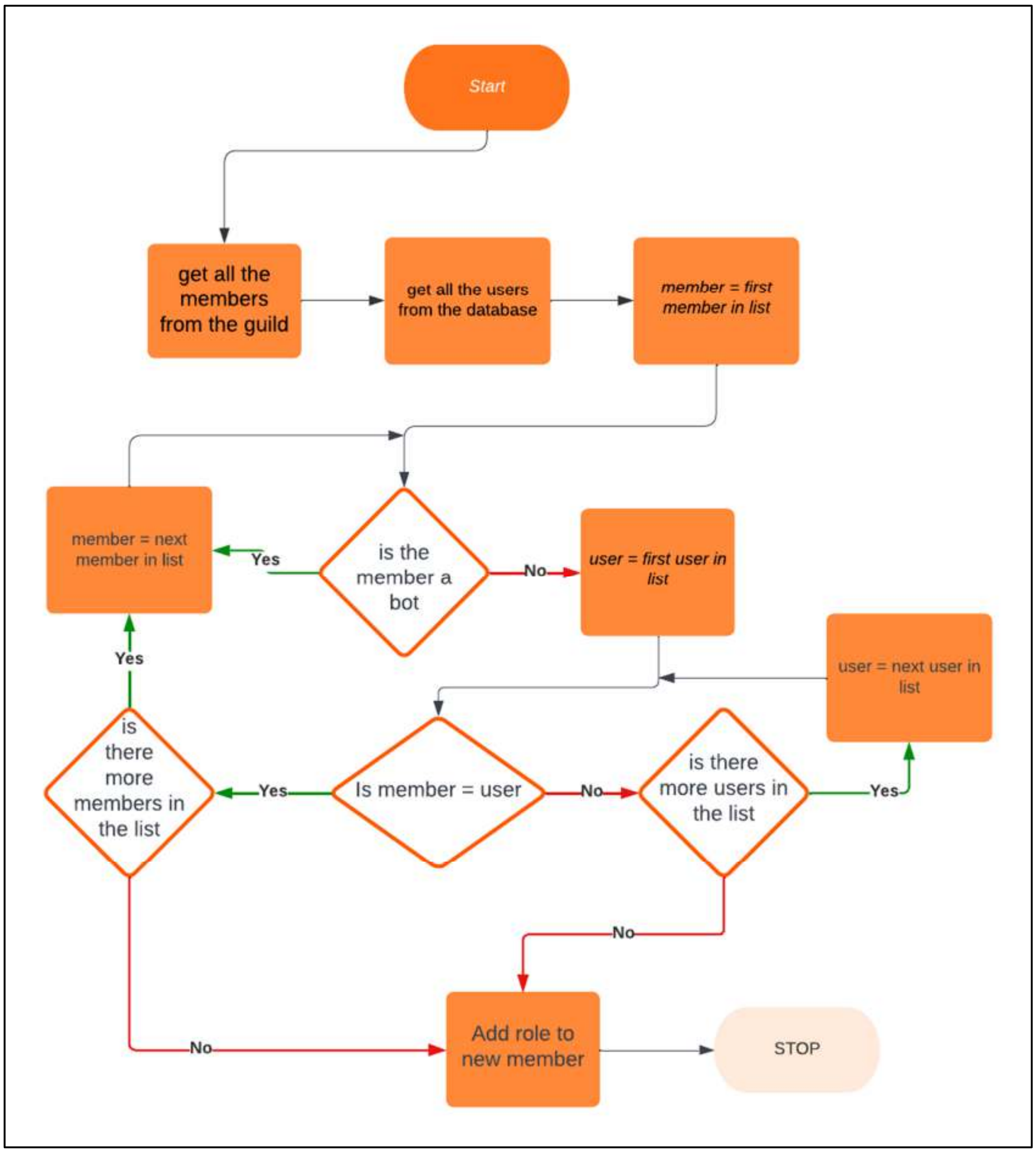
## Product functionality:

- 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.

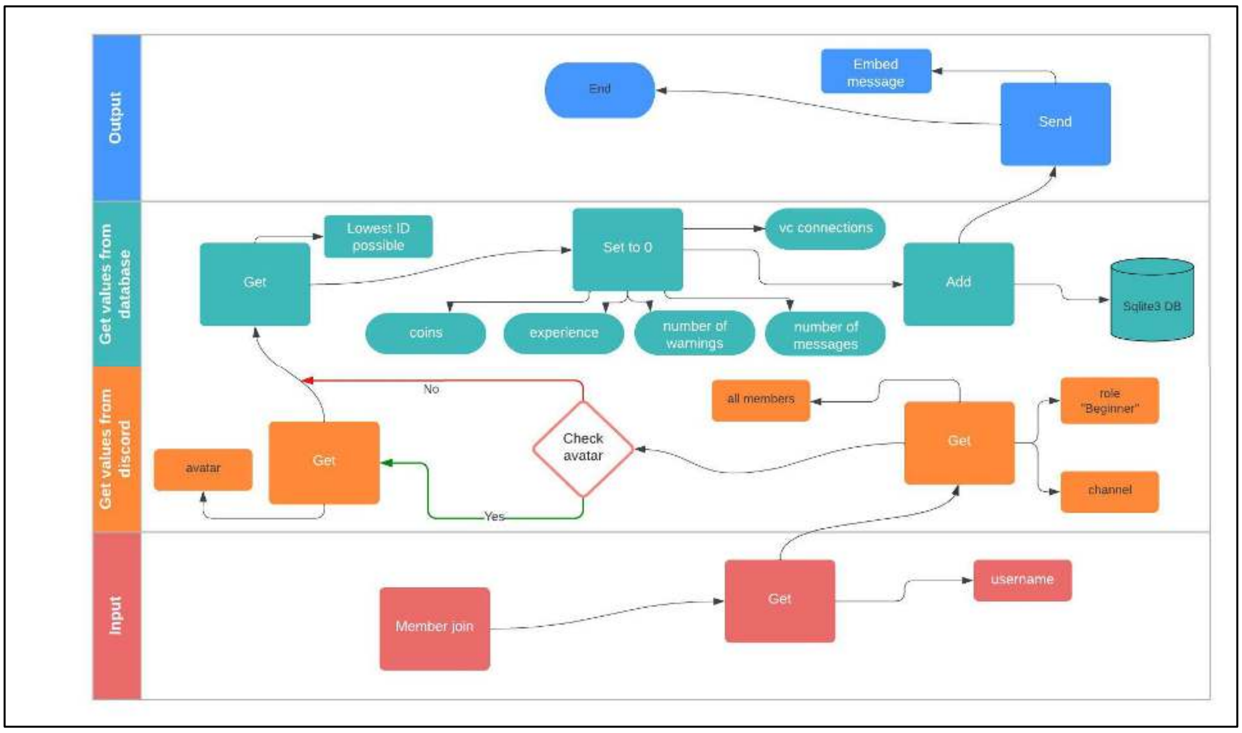
Flowchart:


Start
get all the members from the guild
get all the users from the database
member = first member in list
is the member a bot
Yes
No
member = next member in list
user = first user in list
user = next user in list
is there more members in the list
Yes
Is member = user
Yes
No
is there more users in the list
Yes
No
No
Add role to new member
STOP

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
```

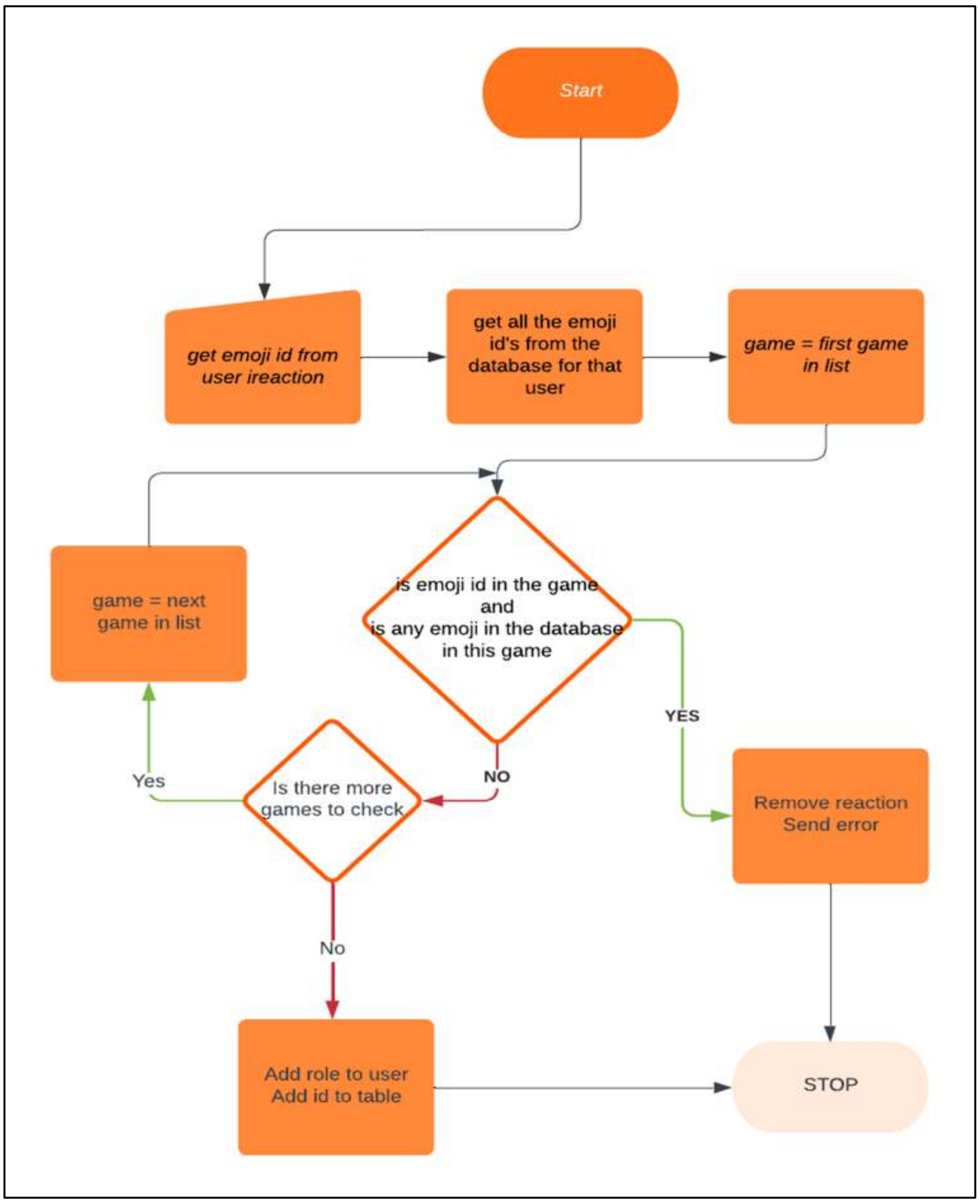
Structure Design:



## 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:

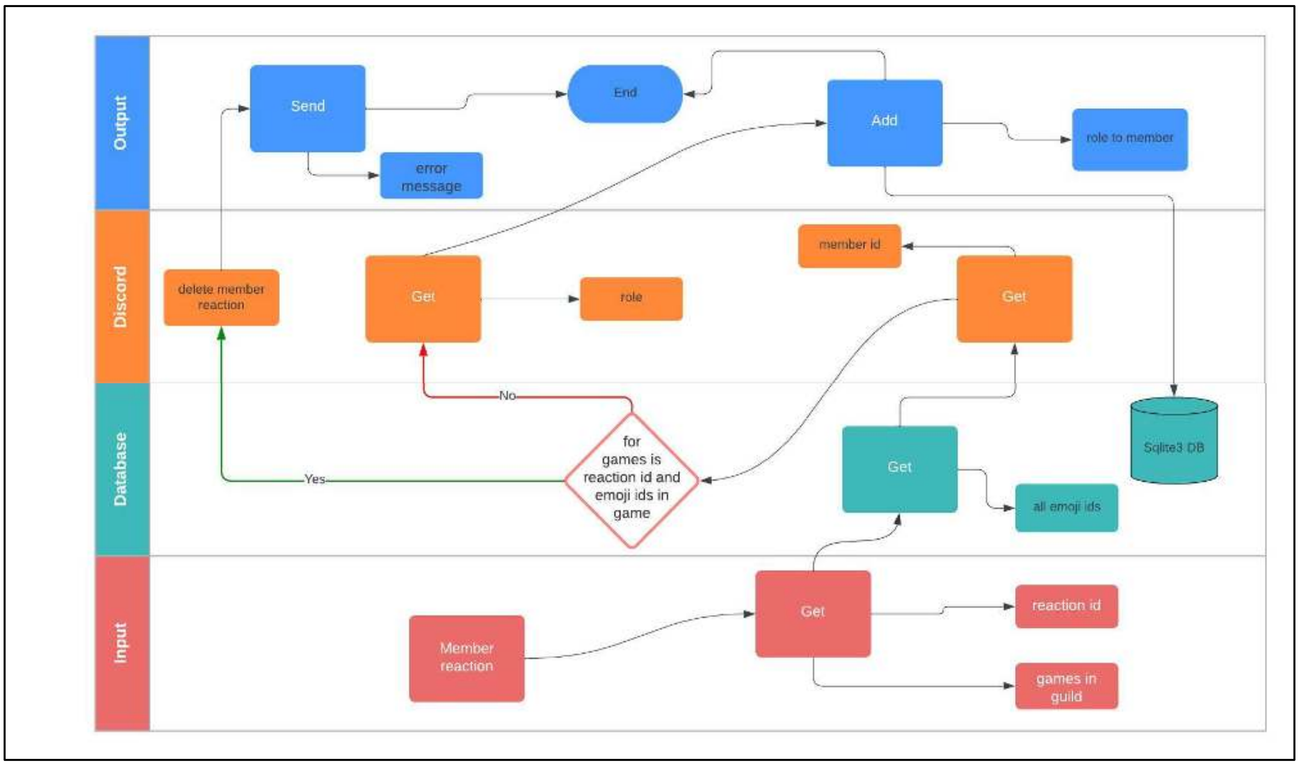

Start
get emoji id from user ireaction
get all the emoji id's from the database for that user
game = first game in list
is emoji id in the game and is any emoji in the database in this game
YES
NO
Remove reaction Send error
Is there more games to check
Yes
No
game = next game in list
Add role to user Add id to table
STOP

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
```

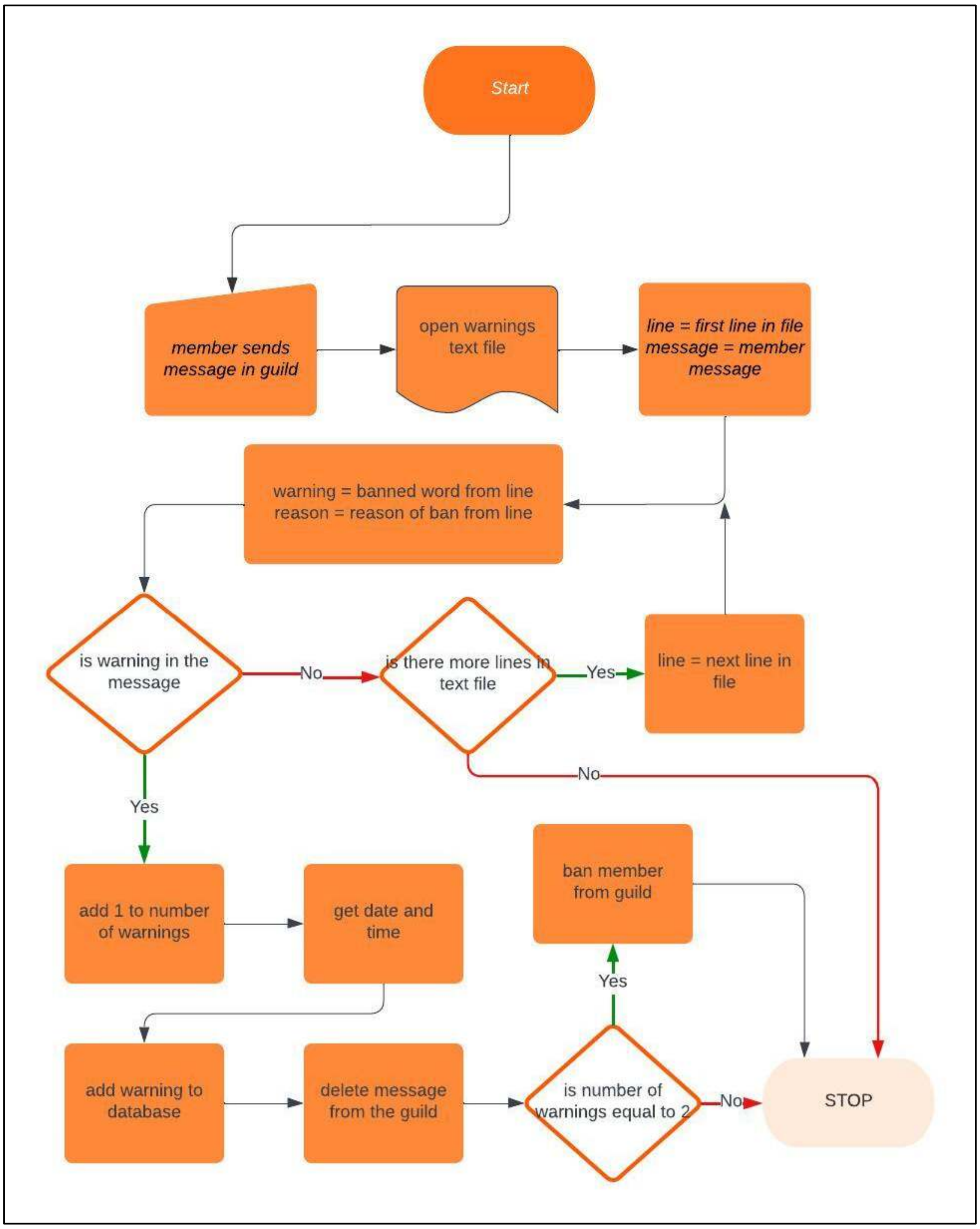
## Structure diagram:



## 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.

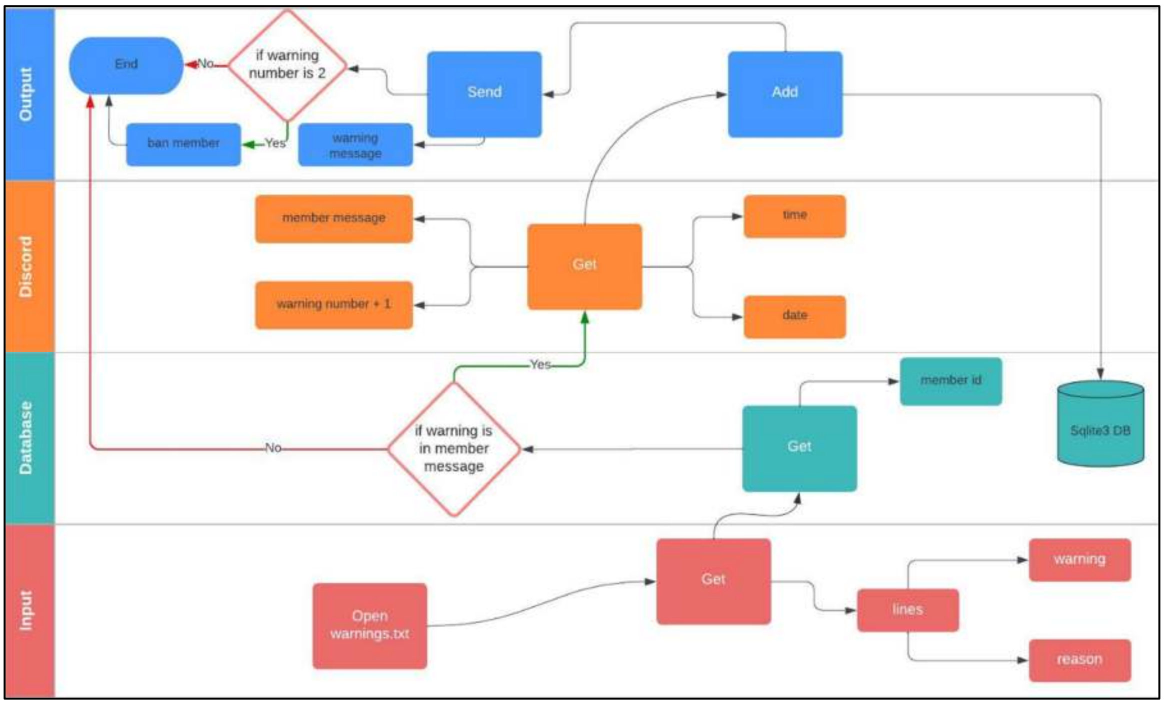
Flowchart:


Start
member sends message in guild
open warnings text file
line = first line in file
message = member message
warning = banned word from line
reason = reason of ban from line
is warning in the message
No
is there more lines in text file
Yes
line = next line in file
No
Yes
add 1 to number of warnings
get date and time
ban member from guild
Yes
add warning to database
delete message from the guild
is number of warnings equal to 2
No
STOP

## 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.

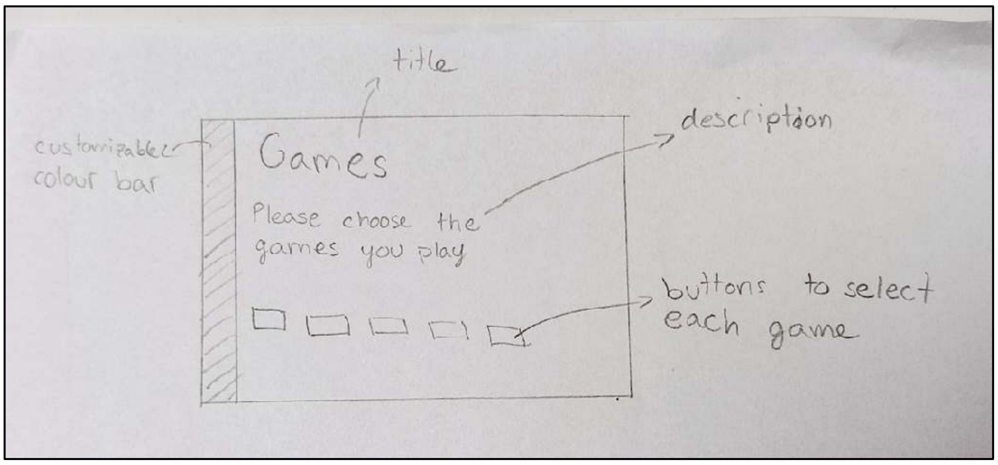
When a member tries to add values to a text file it checks the format and sends an error message if what the member is trying to add doesn't meet the standard format of the file.

## Formatting:

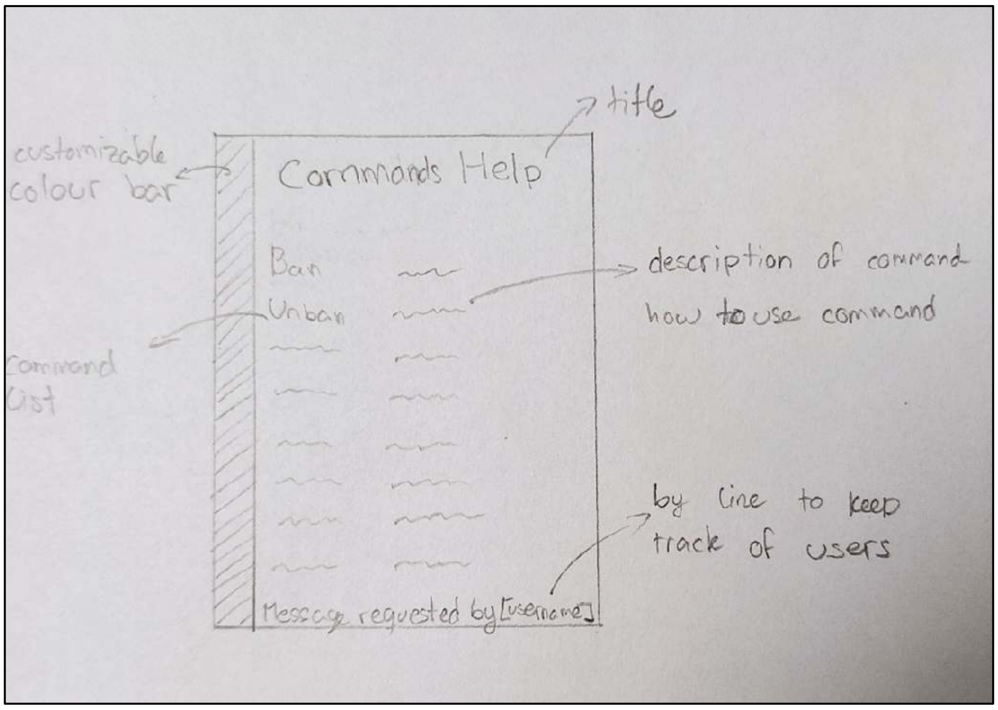
The data sent to the guild chat will have to be formatted. When retrieving the rank or warning data set for the members I will have to format the string, so it makes the message easier to read. This aids the user experience in the guild.

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.



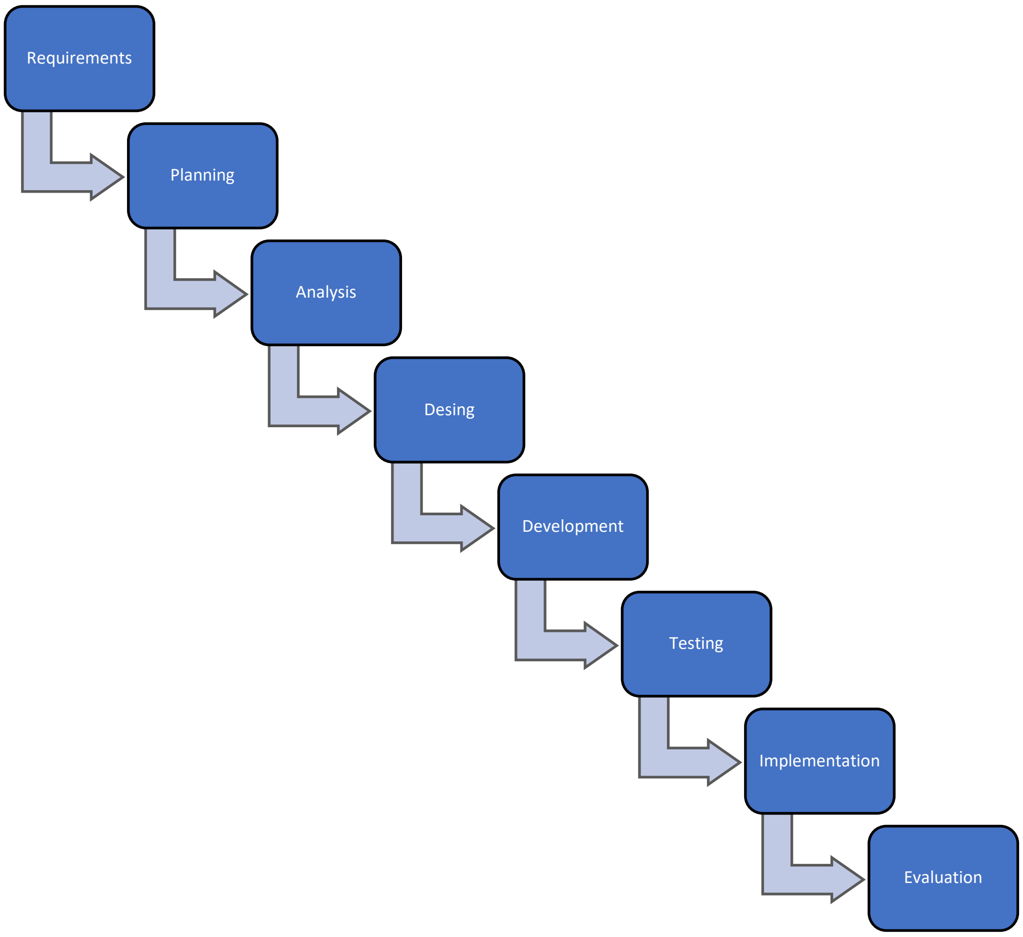
Help command that will display an embed message to give users information about the syntax of every command.


title
customizable colour bar
Commands Help
Ban
Unban
description of command how to use command
Command List
by line to keep track of users
Message requested by [username]

## Design Methodology:

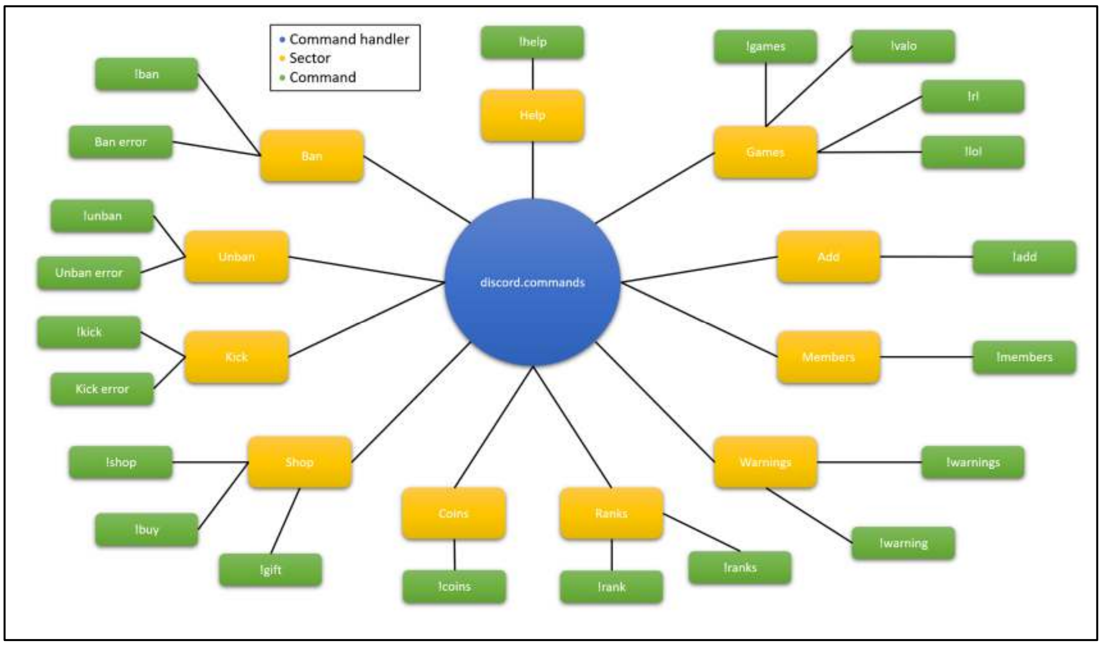
I will use a procedural programming method to approach my project. The program procedure will contain subroutines in a series of steps. Any subroutine will be able to be called at any given point while the program is active.

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]<br><br>!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<br><br>@[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<br>2 !help<br>3 !members | The bot responds to all members rapidly | 1 number between 1 and 10<br>2 list of all commands<br>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]<br><br>!rank [no member]<br><br>![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<br><br>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<br><br>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]<br><br>!rank [no member]<br><br>![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<br><br>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]<br><br>!add [file name] [no text]<br><br>!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<br><br>Input text to add<br><br>Input a correct file name | n/a | |
| **Ban command** | Ban a member from the guild. Ban a member not in the guild | !ban [member]<br><br>!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<br><br>Member not found in guild | n/a | 6 |
| **Unban command** | Unban a banned member and unban a member in the guild | !unban [banned member]<br><br>!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<br><br>Member not banned | n/a | 6 |
| **Kick command** | Kick a member from the guild and a member not in the guild | !kick [member]<br><br>!kick | The member is kicked if it's in the guild if it's not, send an error message | @[member] has been kicked<br><br>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]<br><br>!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<br><br>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<br><br>[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]<br><br>!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<br><br>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)<br><br>!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<br><br>[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)<br><br>!gift [item] [target member not in guild]<br><br>!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]<br><br>[member] you don’t have enough coins<br><br>Member not found<br><br>[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 | |