Mohammad Ruhan Ahmed - 12ahmedmoh

HaNDSWORTH GRAMMAR SCHOOL   | Centre No.: 20107 | Centre No.: 20107

Nea – Discord bot

NEA – Discord bot

Contents

[Analysis 3](#_Toc6415826)

[Introduction 3](#_Toc6415827)

[Background 3](#_Toc6415828)

[Description of a current system: 3](#_Toc6415829)

[Identification of end user 7](#_Toc6415830)

[Research outline: 7](#_Toc6415831)

[What do you like about using Discord bots? 7](#_Toc6415832)

[What sort of functions would you like a multi-functional bot to have? 7](#_Toc6415833)

[Would you like a bot that “talks” to you in chat? For example, one that replies to your messages? 7](#_Toc6415834)

[What are your current issues with bots? 8](#_Toc6415835)

[Objectives list: 8](#_Toc6415836)

[Core Requirements: 8](#_Toc6415837)

[Miscellaneous features: 9](#_Toc6415838)

[Limitations: 9](#_Toc6415839)

[Critical path table 9](#_Toc6415840)

[Design 10](#_Toc6415841)

[Introduction 10](#_Toc6415842)

[File Structure 10](#_Toc6415843)

[r\_bot.py: 11](#_Toc6415844)

[events.py: 11](#_Toc6415845)

[owner.py: 11](#_Toc6415846)

[mod.py: 11](#_Toc6415847)

[text.py: 11](#_Toc6415848)

[level.py: 11](#_Toc6415849)

[voice.py: 11](#_Toc6415850)

[token.txt: 11](#_Toc6415851)

[prefixes.json: 11](#_Toc6415852)

[State Machine Diagram: Events.py 11](#_Toc6415853)

[Cog Pseudocode: 13](#_Toc6415854)

[Class Diagram 14](#_Toc6415855)

[Data Structures 14](#_Toc6415856)

[Lists: 14](#_Toc6415857)

[Database design (and Data Flow Diagram and queries) 14](#_Toc6415858)

[User\_id 15](#_Toc6415859)

[Guild\_id 15](#_Toc6415860)

[Level 15](#_Toc6415861)

[Xp 15](#_Toc6415862)

[Human Command Interface 15](#_Toc6415863)

[Hardware selection 15](#_Toc6415864)

[Technical solution 16](#_Toc6415865)

[r\_bot.py 16](#_Toc6415866)

[events.py 18](#_Toc6415867)

[text.py 20](#_Toc6415868)

[mod.py 22](#_Toc6415869)

[owner.py 24](#_Toc6415870)

[level.py 25](#_Toc6415871)

[voice.py 27](#_Toc6415872)

[Testing 30](#_Toc6415873)

[Read and respond to commands by users, using a compound command prefix, inside a single discord guild. 30](#_Toc6415874)

[PM (private message) users a list of its commands. 31](#_Toc6415875)

[Join VoIP music channels with other users and stream music via YouTube/Spotify/Soundcloud. 31](#_Toc6415876)

[Enable specific user/role commands to admin users through the bot by means of temporary kicks, server bans, channel mute and changing nicknames. 31](#_Toc6415877)

[Enable a method of reloading bot command scripts during its runtime 31](#_Toc6415878)

[Evaluation 31](#_Toc6415879)

[Sources 32](#_Toc6415880)

# Analysis

## Introduction

Discord is a free VoIP application designed for video gaming communities, specializing in text, image, video and audio communication [0]. Running on multiple OS’ (such as Windows, macOS, Android, iOS, Linux and most modern-day web browsers), it has gained a popular and reputable customer base of over 130 million users. The software is built upon eleven data centres to enable low latency among clients, proving to be the most popular choice of VoIP communication in the modern era.

The app has many problems with hostile and unpleasant behaviour of malicious users. For example, guilds (servers) of communities can be “raided” by other communities which include flooding of controversial topics or unwarranted media (e.g. race, politics and pornography.) Whilst discord has implemented a system to protect guilds from “raiding” and spamming, guild owners can feel they can be given tools to protect their own guilds. I think it would hugely benefit everyone inside a community by introducing a Discord bot with multiple utility properties, hence taking upon multiple roles.

## Background

A bot is a type of user account, dedicated to automating within the Discord App. [1] A Discord guild owners can invite a bot to their guild and simulate human interaction – often by playing games such as tic-tac-toe/blackjack, comment text/images/videos or be capable of managing users, such as a guild admin user.

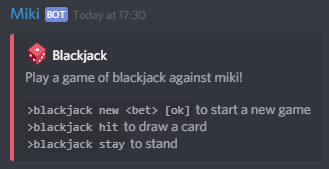
This is what I like about bots, because they make my guild more fun and appealing to my friends, whilst also providing management and protection benefits. Some bots specialise in a certain role (e.g. a music bot to join VoIP rooms or relay information from online video games) – yet some people have no knowledge or experience in using or managing a Discord bot.

Figure 1: Promptly after sending the message ">blackjack", the bot sends an embed with the possible arguments you can send. Any other arguments will result in an error being raised – in this case, the exception handler simply resends this helpful message.

I will create a single discord bot with multi-purpose properties of existing ones, bringing its own personality to communities. To do this, I will need to create an app (using the Discord developer portal).

How bots work is that (with the permissions) they listen for user messages in a given guild’s text channels (sort of a chatroom). When a user starts their message with the bot’s command prefix and types a command following that, the bot reads the message and checks to see if it has that command as a function. The bot then runs the function – with any other strings in the message used as arguments – and sends a message to the same channel.

Figure 2: After creating a game by typing the message ">blackjack new 10 ok", I get put into a blackjack game against the bot and I use the command “>bj hit” – an alias for “>blackjack hit” – until I lose.

## 



## Description of a current system:

To find current bots, Discord have created a website where the community can advertise their freemium apps onto the site discordbots.org [2]. All the following bots require certain permissions to join a guild and function as intended. They also require user-inputted commands with a prefix, to specify that it is a command and not a normal message typed to other users. Every bot must have an internet connection to connect to the discord server, as well as the end users it will service. However, since the following bots aren’t officially supported software, there is always the risk that they can experience downtime or be maliciously used by their developer (i.e. be used a Trojan virus)

### Pokécord

This app is centred on emulating the video game Pokémon and is one of the most popular bot users have unanimously voted, in over 300’000 guilds [3]. It uses embedded links and attached files to allow pseudo-Pokémon (effectively a chat-based game) roam inside a discord guild for users to catch, battle and train them.

Looking at the commands list [4], the bot can make users interact with each other as well as the bot itself (e.g. duelling/trading other users or its own virtual store where users can buy items from virtual cash gained from the bot), which can bring communities closer. The rewarding levelling system and the fact it’s based upon a beloved video game franchise is its strength and another is that the bot functions across multiple guilds. This means that, for example, a Pokémon caught on one guild can be used in another guild, belonging to a different owner.

However, there is no current way to pause its programming; the bot runs constantly and can become quite annoying to some users who don’t want guilds to be flooded. Also, the small list of commands the bot offers disadvantages the owner – who may want to take more control of the bot for a better suited purpose (e.g. limit/increase the no. of Pokémon a user can gain per day or cash gained daily)

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| ✔ Enables users to interact with other users | ✘ Has limited commands |
| ✔ Levelling system allows users to strive for a certain high level for their Pokémon | ✘ Can annoy users who don’t want to interact, due to its R.N.G behaviour |
| ✔ Daily rewards of cash incentivise users to log in to a specific guild | ✘ Can cause guild flooding of commands |
| ✔ Fun to play as an alternative to Pokémon |  |

### D:\NEA\pancake - permissions.pngPancake

This app is a multi-functional bot, just as I intend to make my program into. Invited in over 190’000+ guilds [5], it has many guild permissions which include taking upon the roles of an admin user, music player and fun bot [6]. The command prefix is ‘p!’ which incidentally is the same as the Pokécord bot – but the command syntax is different (i.e. a whitespace is needed and commands themselves are different).

Pancake can privately message users a varied list of its commands using the ‘p! help’ command. It has a shop function similarly to Pokécord, but it has more functionality to what images it can send and there are more games to play with this bot. Users can use Pancake’s image commands to transform their profile picture into something else or find images related to the command (e.g. ‘p! cat’ will find images of cats).

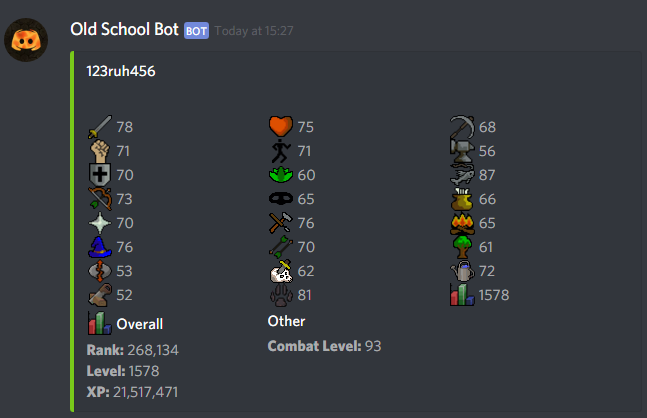
The biggest strength to this bot is that it can play music whilst playing games and moderating other user’s messages – which cannot be said about other bots. Pancake can stream music from popular media such as YouTube and Spotify, which is very useful to users who don’t want to open any additional programs and use up more RAM. However, it can be flooded with commands easily and the moderation commands seem light in quantity; there could be more moderator functions, but they are easy to remember and set up. Also, if there are many users using the bot simultaneously, there can be longer delays than normal.

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| ✔ Has multi-functional commands, which will decrease number of bots in a guild | ✘ Can take long delays to output media if the guild is busy |
| ✔ Able to join music channels and stream music from YouTube/Spotify | ✘ Can cause guild flooding of commands |
| ✔ Daily cash rewards incentivise users to log in to a specific guild | ✘ To use all of the bots functions, the owner needs to give the bot ownership permissions essentially – which can (unlikely) be abused |
| ✔ Moderates users without need of an admin user |  |
| ✔ Plays a variety of user-interactive games |  |

### Old School Bot

This application is a specific utility/fun/information bot to players who play the videogame Old School RuneScape. As a non-affiliated, open source application [7], its commands are very specific to statistics players may want – which cannot be found in game, but rather from websites and databases online. These include the level stats of accounts, information on certain features of the game and simulations of player activities (e.g. the bot can simulate the probability of expensive loot from killing bosses with the command ‘+kill [quantity] [boss]’).

One of its defining features is that it can stream popular Twitch streamers (playing the game at the time) on its status – promoting them to a larger audience and increasing viewer counts. To the player, it can show one of their accounts level stats with the command ‘+stats [username]’ as well as many more features. It also has a command cooldown system implemented, which prevents users from spamming the same commands within a few seconds.

However, it isn’t quite clear on how to use the commands properly as users aren’t told how to use them properly (e.g. for the ‘+kill’ command, I had to find out myself using trial and error) and it’s not very useful to Discord users who don’t play the game at all – only 5 commands out of its 78 commands are non-osrs related.

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| ✔ Enables users to search stats about their accounts/game features | ✘ Has little to no help on using the commands |
| ✔ Simulates game-specific features not found in game | ✘ Not very helpful to users who don’t play OSRS |
| ✔ Supports Twitch streamers by hosting their stream on their Discord status | ✘ Can take a long time to retrieve information from crawling websites |
| ✔ Enables command spam protection |  |

## Identification of end user

Primarily, my target demographic is everyone on a guild – due to the bot’s convergence nature – with my end goal to create an interactive app for all to enjoy. Hence, my standard end user *Faizan #4522* and my admin end user *Brandon #8004* will aid me with feedback into what my discord bot can do – throughout the development process. (Discord uses tags and names combined as a compound key, which is what I’ll use to reference my end users). By creating this bot, it will entertain and service both my end users simultaneously. For example, this could include streaming music or playing games, to managing roles and muting/banning malicious users. The bot’s end goal is to make the guild fun and appeal to non-members ∴ incentivising them to join and increase the guild’s population.

## Research outline:

[Everything from SMART objective list must be derived from here]

To gain more insight to what my end users want, I interviewed them as well as other users inside my own Discord guild. The variety of users gave insight to what certain features people enjoy to what is convenient to them.

### What do you like about using Discord bots?

*FAIZAN #4522*: You can play music with other users without the use of external applications (such as Spotify) in one guild whilst playing games, with the same bot, in another guild.

*BRANDON #8004*: Discord bots are particularly helpful for background music – as well as being able to setup a playlist – with other’s input on the music too.

*OWNER USER:* The apps really help when moderating an entire guild when you’re offline. E.g. I can keep users spam less text/images/media within a certain time frame without me doing anything and if one malicious user seems to ignore the warning, they can be assigned a temporary mute role – which is more efficient than me doing it manually.

### What sort of functions would you like a multi-functional bot to have?

*FAIZAN #4522*: You need music, playing links from YouTube videos and they need to offer easy commands to remember, plus a guide (manual) to assist us how to use the bot. Also, a priority-queue would be nice to use – to combat certain higher-ranked users having privileges over other users.

*BRANDON #8004*: Any good bot will be able to have functions with an easy-to-remember command function. Ideally, a bot will be able to send a private message detailing the functionalities and course of action if the bot doesn’t work as expected.

*OWNER USER*:I want the features of a music, fun and a moderator bot; all-in-one so I don’t need multiple bots for only one/two functions. It would help if it can specifically use role functions in my guild (e.g. @’ing my admins) and if the bot could make moderation easier – e.g. banning/kicking/muting etc.

### Would you like a bot that “talks” to you in chat? For example, one that replies to your messages?

*FAIZAN #4522*: I don’t mind it, but it needs to not be gimmicky in way (i.e. not basic things such as replying to “hi” or “hello”).

*BRANDON #8004*: Not too keen on it e.g. if it replies to certain short phrases, it can become quite annoying and it doesn’t add to the experience.

*OWNER USER*:An A.I. bot would be fun to chat with and it will ultimately benefit me, adding to the fun of my guild. My current users can have more fun with an interactive bot and invite their friends over from different guilds to mine. However, I don’t want users to abuse the bot and make the bot do malicious actions (e.g. if the bot could echo user messages, I don’t want it to respond with offensive language)

### What are your current issues with bots?

*FAIZAN #4522*: The commands are too complicated and there is no definitive guide to use them. When people join a guild, they shouldn’t be expected to have programming knowledge to execute basic tasks.

*BRANDON #8004*: They can be a bit clunky on when they want to work and on the odd occasion, when people want to use their functions within a certain time frame, they either fail to execute accurately – if not, at all. They also can be frustrating to use if two or more bots have the same command keyword, so a unique keyword should be used.

*OWNER USER*: The current bots available are quite buggy with how their commands work, which may be due to the guild’s connection (not the app’s fault) the bot is hosted to – or that the cogs that the bot uses is incorrectly loaded. It would be nice to reload the cogs to my own guild, so that I don’t need to restart the entire bot again

## Objectives list:

### Core Requirements:

[What the system MUST have to function]

Specific, Measurable, Achievable, Realistic and Time bound

1. Read and respond to commands by users, using a compound command prefix, inside multiple discord guilds. Without a prefix, the app cannot take in any command inputs at all – and a compound prefix will help reduce the user accidentally using another bot. Multiple discord guilds expands the bot’s scope in which it can operate.
2. PM (private message) users a list of its commands. It will be very useful to new users who haven’t used the bot at all, or to refresh commands that have been updated
3. Join VoIP music channels with other users and stream music via YouTube/Spotify/Soundcloud. This will be one of the biggest strengths of the bot, as it is a simple concept
4. Enable specific user/role commands to admin users through the bot by means of temporary kicks, guild bans, channel mute and changing nicknames. It will help current admins and the owner to use the bot as a shortcut to manually moderating users, which is time consuming
5. Enable a method of reloading bot command scripts during runtime. This will mean the bot won’t need to be restarted each time errors occur relating to the commands
6. Implement a levelling system. This incentivises people to log on to guilds each day, which will benefit the owner and the general popularity of the guild.

### Miscellaneous features:

[Minor features that could be added if there is time; features that aren’t an absolute necessity]

1. Enable functions like Old School Bot (OSRS related functions). This will be easy to implement, as the current app is open source ∴ it will only need slight modification
2. Respond to user’s messages with an A.I. function. The function itself can be as simple as responding to common phrases from a database of responses, to open-sourced, pre-made programs found online
3. Have a Priority-queue system/threading system. This is to make sure that if many users use the bot simultaneously, it will handle their inputs efficiently as possible.

### Limitations:

[The limits of designing the system in the current environment (i.e. school computers)]

* Coding the bot and testing requires the local computer to have various modules installed, which can be difficult to install within a school computer network, as well as having a connection to the Discord A.P.I.’s (my school’s I.P. content blocker currently disallows this packet switching). For me, this would mean the bot has to be entirely coded at home – limiting the time I have to program
* The bot can’t join multiple music channels at any given time, which is a limitation on all Discord user accounts, and there can only be one version of the bot running at a time
* The bot can’t be online 24/7. To implement such a feature would require paying a third-party host, which is beyond the scope of my requirements
* If the bot gets hacked by a malicious user, it may need to be shut down completely to be safe and adhere to the Discord rules and regulations (if said malicious user abuses the bot) [8]
* If there is an A.I. function implemented, it would be expected to not respond to everything users can ask – it is not supposed to be as complex as a machine learning bot

## Critical path table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Section | Activity | Start date | Duration  In weeks | Dependant on |
| A | Analysis | 25/10/18 | 2 | - |
| B | Design | 09/10/18 | 2 | A |
| C | Technical Solution | 06/11/18 | 11 | A, B |
| D | Testing | 26/02/19 | 2 | C |
| E | Evaluation | 05/03/19 | 1 | C, D |

# Design

## Introduction

The core processes and commands of the bot are built upon existing class decorators from the Discord module. It will be structured to the design of Lucas Kamara’s tutorial [9], yet due to the differences between the module’s versions – with my application being programmed with the v1.0.0 rewrite – the syntax will have to be modified almost entirely (with the help of the rewrite documentation [10]) The main processes are to run asynchronous coroutines (the same commands the user will use in Discord) and events while the program is running, via cogs (python scripts not in main.py).

## File Structure

I will separate out each of the Discord commands into their own separate cogs, to organise their commands and functions distinctly to each other (e.g. a normal user command won’t be loaded from). All python scripts will adhere to the PEP8 style, using the Pylint module [11]

### r\_bot.py:

The parent program that initiates the bot client and all the cogs from the ‘Cogs’ directory. The bot’s background processes are also stored here.

### events.py:

All the events listened by the bot client (connection to the Discord WebSocket and REST API [12] – the bot’s online presence) are stored here.

### owner.py:

Commands that the bot’s/guild’s owner (me) will use to test out features relating to the bot itself (e.g. reloading cogs or changing the command prefix).

### mod.py:

Commands guild moderators/admins will use to manage ordinary users (e.g. ban/mute/kick commands).

### text.py:

Commands that any standard user can use (e.g. ping, echo).

### level.py:

All the commands and algorithms related to calculating & storing user data to a database (using PostgreSQL).

### voice.py:

All the commands and queues that relate to the bot’s voice commands (e.g. join, leave, play, pause, resume). The music playlist queue will be instantiated in this script, so users can add & play a continuous stream of music.

### token.txt:

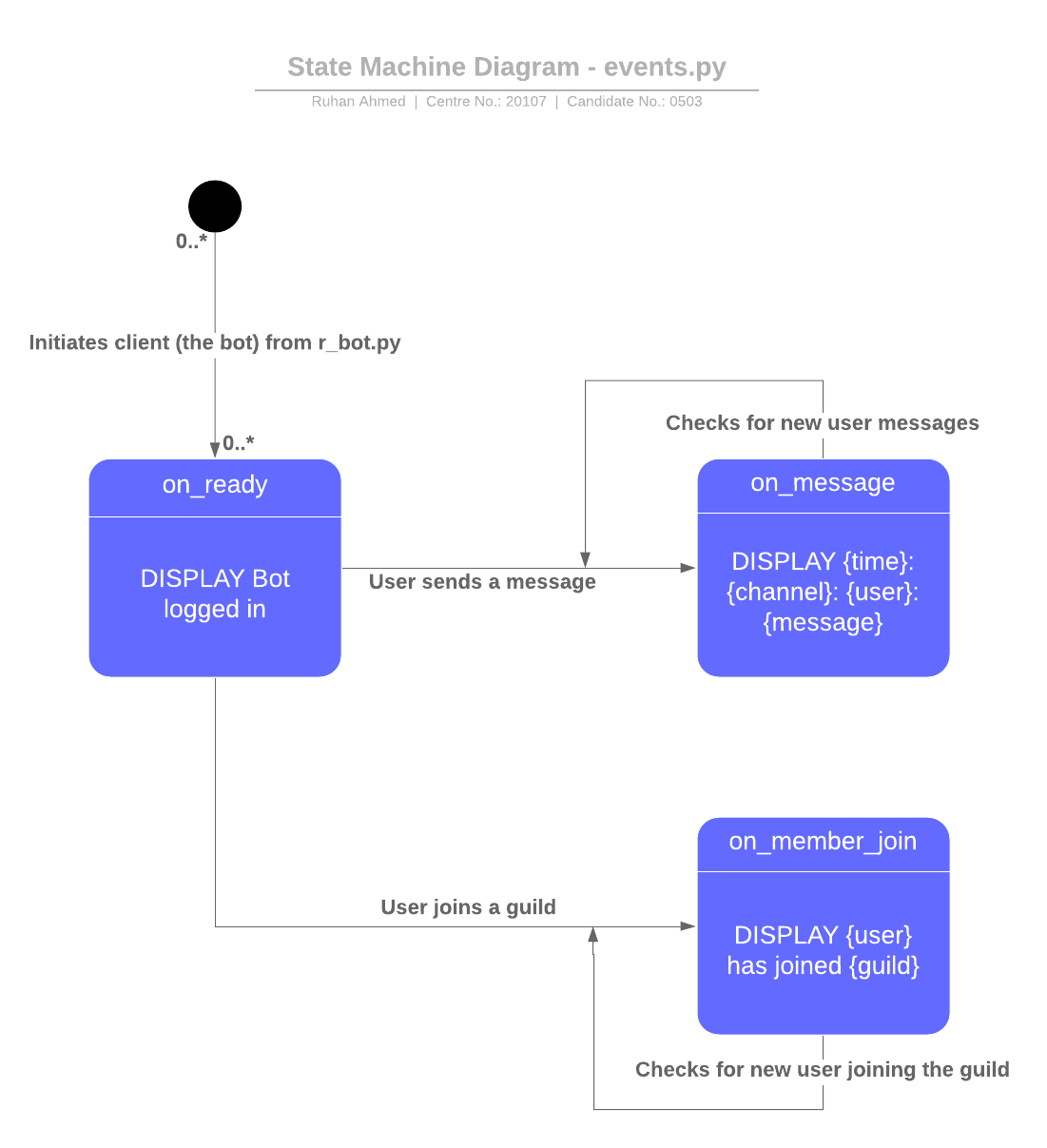
This will be the bot’s unique token, which acts like the bot’s password [13]. Users who access this essentially have full control over the bot, which can be dangerous in the hands of unauthorised users with malicious intent – e.g. they could mute/kick/ban all members in a guild or create a script to ping everyone in a guild continually. Due to how important the token is, I decided it will be best suited for the token to be stored in a text file.

### prefixes.json:

This .json file will store the bot’s prefixes used within multiple guilds. This has to be global, as the bot can be in many guilds simultaneously.

## State Machine Diagram: Events.py

To illustrate how each cog works generally, I decided a S.M.D. was suitable. The **events.py** file is entirely event-driven, which are first class elements of a S.M.D. [14], and the other cogs follow the same algorithmic logic. To simplify, a basic bot will have 2 events: *on\_ready* & *on\_message*, which each run every time the bot is ready to receive input and the bot listens to a message respectively. I want to add a few more events to the bot (e.g. *on\_member\_join*, *on\_reaction\_add* – where the “reaction” is just adding a Discord emoji to a previously-typed message – etc.)



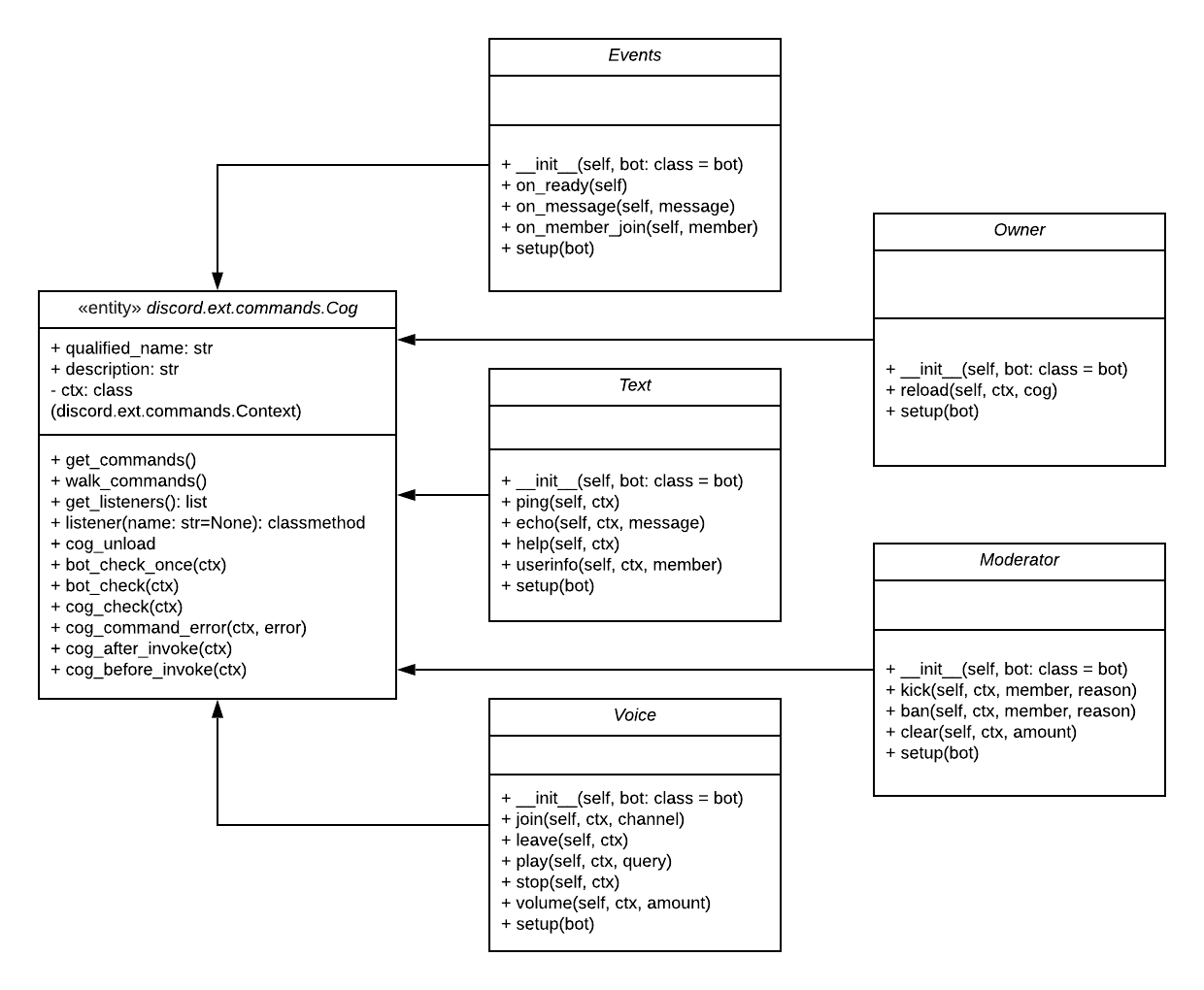
## Cog Pseudocode:

This is the general format on how each cog will be instantiated

1. **##### PSEUDOCODE #####**
3. **IMPORT** discord    # Imports necessary modules for the bot to function
5. **CLASS** **<cog>**(commands.Cog)         # Created cog inherits from base class “commands.Cog”
6. **PROCEDURE** initialiser (self, bot)
7. **BEGIN** **PROCEDURE**
8. self.bot = bot
9. **END** **PROCEDURE**
11. **DECORATOR** commands.Cog.listener   # Decorator marks functions as a listener
12. ASYNC **COROUTINE** **<event 1>**(self, \*)
13. **BEGIN** **COROUTINE**
14. **<event 1 expressions>**
15. ...
16. AWAIT **<last expression for event 1>**       # All coroutines will return 'awaitables' in the last line of their expressions e.g. AWAIT SEND <message> TO DISPLAY
17. **END** **COROUTINE**
18. ...  # Events created for different procedures e.g. “on\_ready”, “on\_message”
20. **DECORATOR** commands.Command(aliases) # Decorator transforms coroutines into commands
21. ASYNC **COROUTINE** **<command 1>**(self, \*)
22. **BEGIN** **COROUTINE**
23. **<command 1 expressions>**
24. ...
25. AWAIT **<last expression for command 1>**
26. **END** **COROUTINE**
27. ...
29. **PROCEDURE** setup(bot)  # Entry point to the main file, that will load all the cogs
30. **BEGIN** **PROCEDURE**
31. bot.add\_cog(**<cog>**(bot))
32. **END** **PROCEDURE**

## Class Diagram

Each cog is a subclass of “commands.Cog” – with every command marked with a “command.command()” decorator and every listener is marked with a “command.Cog.listener()” decorator [15]. To initialise each class as a cog, a setup function is required to “load” them:



## Data Structures

### Lists:

For small items (of the same data type) grouped together, I will use a simple list to store them e.g. the bot’s statuses; It doesn’t require the complexities of a database. E.g. (using the bot statuses example), the bot can have the following strings: “Testing…”; “I am a bot!”; “Playing with the gpu…” saved in a statuses array – and a cycle module can be imported, so the statuses cycle through the array in regular intervals.

### Database design (and Data Flow Diagram and queries)

To implement the levelling system, I will store the following data into a guild-based database using PostgreSQL. This application allows the bot to update the database over an internet connection, which means it isn’t reliant on my own computer alone. E.g. If I ran the bot on another computer, if the PostgreSQL guild is up – the database will update without my own computer being turned on.

The way levels will function is, for every message a user types in (all while the bot is running), the bot increments a whole number to the xp field – until the record is incremented to a certain number (calculated by an algorithm). This will then increment the level field by 1. E.g. If level 1 started at 0 xp, and the algorithm stated that every 10 xp points equates to a level, with 1 message typed equal to 1 xp – level 2 will be at 10 points (10 messages typed), level 3 at 20 points (20 messages typed) and so on. If a user types their first message, the bot will create a new record for them and update it accordingly.

|  |  |  |  |
| --- | --- | --- | --- |
| User\_id | Guild\_id | Level | Xp |
| *BigInteger = [not Null]* | *BigInteger = [not Null]* | Integer = 1 | Integer = 0 |

#### User\_id & Guild\_id:

The “user\_id” will the primary key, even though the database isn’t a relational type, because it is unique to each client (including the bot) [17]. The “guild\_id” will contain the corresponding guild a user is in. This would imply that a user can be in different guilds but have the same level/xp stored. Both fields will be given *[Not Null]* values – since they will always contain a value – and stored as a big integer, since the id’s (or snowflakes) are stored as 64-bit integers.

#### Level & Xp:

These fields will store the user’s level & experience as integers. The default values are 1 & 0 respectively, since this would imply the user’s first message sent in any given guild (with the bot running in).

## Human Command Interface

The client will only access the bot via commands in the Discord app – implying there is no need for creating a new graphical user interface for the client, as it is already implemented. It would be visually pleasing for the bot’s developer, but it would be a nuisance to manage an unnecessity, along with the program’s core functions; a command line interface is sufficient enough and is straight to the point. Even then, the interface would only be used to print discord messages and debug errors; the bot can be made solely independent of any visual output once finished.

## Hardware selection

Since the finished program won’t be reliant on a g.u.i., this opens the possibilities of running the program on devices with or without an optical output (e.g. a raspberry pi or smartphone) – but having to connect to the discord API’s, hence the internet. This would involve compiling the program into an executable file, and distributing it out on devices with different processor architectures. I would like to test the program on a Lenovo h420 desktop computer (with an Intel i3-2100, 64 bit processor) [18] and a Samsung Galaxy S8 – with an Octa-core (2.3GHz Quad + 1.7GHz Quad), 64 bit processor [19]. To make sure the bot will run, each device will need to have the necessary python modules imported (e.g. discord.py).

# Technical solution

* A system that meets almost all the requirements of a solution/an investigation (ignoring any requirements that go beyond the demands of A-level)
* A system that achieves many of the requirements but not all. The marks at the top end of the band are for systems that include some of the most important requirements
* A system that tackles some aspects of the problem or investigation

1. Complex data model in database (e.g. several interlinked tables): Cross-table parameterised SQL; Aggregate SQL functions; User/CASE-generated DDL script
2. Hash tables, lists, stacks, queues, graphs, trees or structures of equivalent standard: Graph/Tree Traversal
3. Files(s) organised for direct access: List operations; Linked list maintenance; Stack/Queue Operations; Hashing
4. Complex scientific/ mathematical/ robotics/ control/ business model: Advanced matrix operations; Recursive algorithms; Complex user-defined algorithms (e.g. optimisation, minimisation, scheduling,  
   pattern matching) or equivalent difficulty; Merge sort or similarly efficient sort
5. Complex user-defined use of object-orientated programming (OOP) model, e.g. classes, inheritance, composition, polymorphism, interfaces: Dynamic generation of objects based on complex user-defined use of OOP model
6. Complex client-server model: Server-side scripting using request and response objects and server-side extensions for a complex client-server model; Calling parameterised Web service APIs and parsing JSON/XML to service a complex client-server model

[Talk how Voice.py is copied from <website> and it is standard – no marks for above points, but marks for completeness] [Comment where group A things achieved]

I was following Lucas’s python tutorial [9] in designing the code, but I had to modify almost every line of code - so that it was compatible for the rewritten discord module version I was using [21]

## r\_bot.py

1. '''''
2. r\_bot is the main python script that initiliases the bot client
3. and runs all the cogs together asynchronously
4. '''
6. # bot ID:519932173969522698
7. # perm integer:8
8. # https://discordapp.com/oauth2/authorize?bot\_id=519932173969522698&scope=bot&permissions=8
9. # Use CMD instead of IDLE, E.g. echo command won't work properly with idle
11. **from** itertools **import** cycle
12. **import** json
13. **import** os
14. **import** asyncio
15. **import** asyncpg
16. **import** discord
17. **from** discord.ext **import** commands
19. # ------------------------- Constants -------------------------
21. FILE\_PATH = os.path.dirname(\_\_file\_\_)
22. # Sets current directory to R-bot's directory
23. TOKEN\_FILE = os.path.join(FILE\_PATH + "\\data\\token.txt")
24. TOKEN = open(TOKEN\_FILE, 'r').read()
25. # Reads the token found in the text file

28. **def** get\_prefix(rbot, message):
29. '''''Changes prefix depending on what guild the user typed the command'''
30. **if** **not** message.guild:
31. **return** commands.when\_mentioned\_or('r-')(rbot, message)
33. file\_path = os.path.dirname(\_\_file\_\_)
34. prefix\_file = os.path.join(file\_path + "\\data\\prefixes.json")
35. with open(prefix\_file, 'r') as \_f:
36. prefixes = json.load(\_f)
38. **if** str(message.guild.id) **not** **in** prefixes:
39. # If the current guild doesn't have a custom prefix
40. **return** commands.when\_mentioned\_or('r-')(rbot, message)
42. prefix = prefixes[str(message.guild.id)]
43. **return** commands.when\_mentioned\_or(prefix)(rbot, message)
45. DESCRIPTION = "A bot made for helping out human users"
46. # Bot's description in Discord
47. BOT = commands.Bot(command\_prefix=get\_prefix, description=DESCRIPTION)
48. # Initiates bot with keyword 'r!'
49. BOT.remove\_command('help')
50. # Removes default help command so I can make my own
52. # ------------------------- Background tasks -------------------------
54. async **def** change\_status():
55. '''''aynchronous function without threading'''
56. await BOT.wait\_until\_ready()    # Will not run until ready() is True
57. status = ['Eating cake', 'Debugging errors', 'Testing']
58. msgs = cycle(status)    # Cycles through the list of pre-given messages
60. **while** **not** BOT.is\_closed():
61. current\_status = next(msgs)
62. await BOT.change\_presence(activity=discord.Game(current\_status))
63. # shows what 'game' the bot is playing
64. await asyncio.sleep(5)
65. # Cycles this specific function every 3 seconds
66. # time.sleep(3) pauses entire program for 3 seconds
68. async **def** create\_db\_pool():
69. '''''Connects to the database using asyncpg'''
70. BOT.pg\_con = await asyncpg.create\_pool(
71. database='levelDB',
72. user='postgres',
73. password='password')
75. # ------------------------- Main loop -------------------------
77. **for** cog **in** os.listdir('.\\cogs'):
78. **if** cog.endswith('.py'):
79. # Looks for the python cogs in the current directory
80. **try**:
81. cog = f"cogs.{cog.replace('.py', '')}"
82. BOT.load\_extension(cog)
83. **print**(f'{cog} loaded')
84. # Loads each file as cog.{filename} and prints them
85. **except** Exception as \_e:
86. **print**(f'{cog} cannot be loaded')
87. **raise** \_e
89. BOT.loop.run\_until\_complete(create\_db\_pool())
90. BOT.loop.create\_task(change\_status())   # Starts the status' cycle
91. BOT.run(TOKEN)  # Runs the bot using it's unique token

## events.py

1. '''''
2. events.py is the cog that encapsulates the listeners a bot uses
3. in the Discord guilds (servers) it has been invited to (running in).
4. The events are: on\_ready, on\_message, on\_message\_join, on\_reaction\_add
5. on\_reaction\_remove, on\_command\_error
6. '''
8. **import** time
9. **import** discord
10. **from** discord.ext **import** commands
12. **class** Events(commands.Cog):
13. '''''Encapsulates all event listeners in the Events class'''
15. **def** \_\_init\_\_(self, bot):
16. self.bot = bot
18. @commands.Cog.listener()
19. async **def** on\_ready(self):
20. '''''
21. Event is called every time bot connects / reconnects
22. Due to how the listener works, this must be named "on\_ready"
23. like reserved methods such as "\_\_init\_\_"
24. '''
26. **print**(
27. f"""
28. {time.strftime('%X')}: {self.bot.user} Logged in
29. Discord Version: {discord.\_\_version\_\_}\n
30. """
31. )
32. # Prints when the bot connected to the guild, in the shell
34. @commands.Cog.listener()
35. async **def** on\_message(self, message):
36. '''''Event is called on every message recieved by the bot'''
38. **if** message.author == self.bot.user:
39. # Makes sure the bot doesn't reply to itself
40. **return**
42. **print**(
43. f"{time.strftime('%X')}: {message.channel}: {message.author}: {message.content}"
44. )
45. # Prints each message members post in the guild
47. **if** message.content.startswith('hi there'):
48. await message.channel.send(f'Hi {message.author.name}! :smiley:')
49. **elif** message.content.startswith('good bot'):
50. await message.channel.send(f'Thank you! :smile:')
51. # Bot finds string message without a prefix and responds
53. @commands.Cog.listener()
54. async **def** on\_member\_join(self, member):
55. '''''Event is called when a member joins the guild'''
57. role = discord.utils.get(member.guild.roles, name='Newcomer')
58. await member.add\_roles(role)
59. # Autoroles the member to the newcomer role
61. **print**(f"{member.author} has joined the guild")
63. @commands.Cog.listener()
64. async **def** on\_reaction\_add(self, reaction, user):
65. '''''Event is called when a reaction is added to any message'''
67. await reaction.message.channel.send(
68. f"""
69. {time.strftime('%X')}:
70. {user.name} has added {reaction.emoji} to the message:
71. `{reaction.message.content}`
72. in the {reaction.message.channel} channel
73. """
74. )
76. @commands.Cog.listener()
77. async **def** on\_reaction\_remove(self, reaction, user):
78. '''''Event is called when a reaction is removed from any message'''
80. await reaction.message.channel.send(
81. f"""
82. {time.strftime('%X')}:
83. {user.name} has removed {reaction.emoji} from the message:
84. `{reaction.message.content}`
85. in the {reaction.message.channel} channel
86. """
87. )
89. @commands.Cog.listener()
90. async **def** on\_command\_error(self, ctx, error):
91. '''''Listens for command errors and prints them'''
93. **if** isinstance(error, commands.MissingPermissions):
94. await ctx.send("You don't have permission to do that!")
95. **if** isinstance(error, commands.BotMissingPermissions):
96. await ctx.send("I don't have permission to do that!")
97. **if** isinstance(error, commands.CommandNotFound):
98. await ctx.send("Err0r 404: Command not found!")
100. **raise** error
102. **def** setup(bot):
103. '''''Entry point to the "r\_bot.py" file'''
104. bot.add\_cog(Events(bot))
105. # Registers the "Event.py" cog to the bot

## text.py

1. '''
2. text.py is the cog that encapsulates all the commands a standard user
3. of a given Discord guild (server) would use.
4. The commands are: ping, echo, help, display\_embed, userinfo
5. '''
7. **import** discord
8. **from** discord.ext **import** commands
10. **class** Text(commands.Cog):
11. '''''Encapsulates all text commands in the Text class'''
13. **def** \_\_init\_\_(self, bot):
14. self.bot = bot
16. @commands.command(aliases=['p'])
17. # Aliases are shortcuts to the commands e.g. "r-ping" ≡ "r-p"
18. # Youtube tutorial in v0.16.0 shows problematic error
19. # Add this correction to Design part of NEA (context)
20. async **def** ping(self, ctx):
21. '''''Sends "pong" and the latency between the client and bot'''
22. # Commands docstrings are in the "r-help" list (default on all bots)
24. await ctx.send(f'Pong! The time latency was: {self.bot.latency}')
25. # E.g. bot.say() ==> ctx.send(), say this in design
27. @commands.command(aliases=['e'])
28. async **def** echo(self, ctx, \*, message: commands.clean\_content):
29. '''''
30. Echoes messages without commands triggering
31. i.e. "r-echo @here" will print out the string "@here"
32. but won't trigger the Discord command "@here"
33. '''
35. await ctx.send(message)
37. @commands.command(aliases=['h'])
38. async **def** help(self, ctx):
39. '''''Sends an embedded PM to the user of a list of commands'''
41. embed = discord.Embed(
42. title='Help commands:',
43. colour=ctx.author.color,
44. timestamp=ctx.message.created\_at
45. )
47. embed.set\_author(name='Help', icon\_url=ctx.author.avatar\_url)
49. **for** cmds **in** self.bot.commands: # Creates a field for each command
50. embed.add\_field(name=cmds, value=f'{cmds.aliases}: {cmds.help}', inline=False)
51. await ctx.author.send(embed=embed)
53. @commands.command(aliases=['de'])
54. async **def** display\_embed(self, ctx):
55. '''''Displays a default embedded media on discord'''
57. embed = discord.Embed(
58. title='Google',
59. description='Description',
60. colour=discord.Colour.blue(),
61. url='https://www.google.co.uk',
62. )
64. embed.set\_footer(text='footer')
65. embed.set\_image(url='https://discordpy.readthedocs.io/en/rewrite/\_images/snake.png')
66. embed.set\_thumbnail(url='https://www.python.org/static/img/python-logo.png')
67. embed.set\_author(name='Author name', icon\_url='https://bit.ly/2wGaL05')
68. embed.add\_field(name='Field name', value='Field value', inline=False)
70. await ctx.send(embed=embed) # Bot posts the embed
72. @commands.command(aliases=['ui'])
73. async **def** userinfo(self, ctx, member: discord.Member = None):
74. '''''Displays the user's account information'''
76. member = ctx.author **if** **not** member **else** member
77. # Makes member argument the user if it isn't given
78. roles = [role **for** role **in** member.roles]
79. # Makes a list of member roles a user has
81. user = await self.bot.pg\_con.fetchrow(
82. """
83. SELECT \*
84. FROM users
85. WHERE user\_id = $1 AND guild\_id = $2
86. """,
87. member.id, member.guild.id
88. )
89. # Get's the user's record
91. **if** **not** user: # Creates a new user if user is returned false
92. user = await self.bot.pg\_con.fetchrow(
93. """
94. INSERT INTO users (user\_id, guild\_id)
95. VALUES ($1, $2)
96. RETURNING \*
97. """,
98. member.id, member.guild.id
99. )
101. await self.bot.pg\_con.execute(
102. """
103. UPDATE users
104. SET xp = $1
105. WHERE user\_id = $2 AND guild\_id = $3
106. """,
107. user['xp'], member.id, member.guild.id
108. )
110. embed = discord.Embed(
111. colour=member.colour,
112. timestamp=ctx.message.created\_at)
114. embed.set\_author(name=f'User Info - {member}')
115. embed.set\_thumbnail(url=member.avatar\_url)
116. embed.set\_footer(text=f'Requested by {ctx.author}', icon\_url=ctx.author.avatar\_url)
118. embed.add\_field(name='ID:', value=member.id)
119. embed.add\_field(name='Guild name:',
120. value=member.display\_name)
121. embed.add\_field(name='Created at:',
122. value=member.created\_at.strftime('%a, %#d %B %Y, %I:%M %p GMT'))
123. embed.add\_field(name='Joined at:',
124. value=member.joined\_at.strftime('%a, %#d %B %Y, %I:%M %p GMT'))
125. embed.add\_field(name=f'No. of Roles: {len(roles)}',
126. value=' '.join([role.mention **for** role **in** roles]))
127. embed.add\_field(name='Top role:',
128. value=member.top\_role.mention)
129. embed.add\_field(name='Level',
130. value=user['level'])
131. embed.add\_field(name='Experience',
132. value=user['xp'])
133. embed.add\_field(name='Bot?',
134. value='Human.' **if** **not** member.bot **else** 'B33p b00p, True')
136. await ctx.send(embed=embed)
138. **def** setup(bot):
139. '''''Entry point to the "r\_bot.py" file'''
140. bot.add\_cog(Text(bot))

## mod.py

1. '''''
2. mod.py is the cog that encapsulates all the commands a Moderator user
3. of a given Discord guild (server) would use (to moderate other players).
4. The current commands are: kick, ban, clear
5. '''
7. **import** discord
8. **from** discord.ext **import** commands
10. **class** Mod(commands.Cog):
11. '''''Encapsulates all moderation commands in the Mod class'''
12. **def** \_\_init\_\_(self, bot):
13. self.bot = bot
15. @commands.command(aliases=['k'])
16. @commands.has\_permissions(kick\_members=True)
17. #  The user must have permission to kick the other member
18. async **def** kick(self, ctx, member: discord.Member, \*, reason=None):
19. '''''Kicks a member'''
21. ##  if member.author.id == ctx.author.id: - Need to test this
22. ##      raise Exception
24. await member.kick(reason='No reason' **if** **not** reason **else** reason)
25. await ctx.send(
26. f'{member.mention} was kicked by {ctx.author.mention} for: `[{reason}]`'
27. )

30. @commands.command(aliases=['b'])
31. @commands.has\_permissions(ban\_members=True)
32. # The user must have permission to kick the other member
33. async **def** ban(self, ctx, member: discord.Member, \*, reason=None):
34. '''''Bans a member'''
36. reason = 'No reason' **if** **not** reason **else** reason
37. await member.ban(reason=reason)
38. await ctx.send(
39. f'{member.mention} was banned by {ctx.author.mention} for: `[{reason}]`'
40. )
42. @commands.command(aliases=['c'])
43. @commands.has\_permissions(manage\_messages=True)
44. async **def** clear(self, ctx, amount: int):
45. '''''Clears the past messages'''
47. await ctx.channel.purge(limit=amount + 1, check=**lambda** msg: **not** msg.pinned)
48. await ctx.send(f'{amount} messages were deleted')
50. @kick.error
51. @ban.error
52. @clear.error
53. async **def** \_error(self, ctx, error):
54. '''''Runs when the prefix error is raised'''
55. # Underscore implies this is not a command to be used
57. **if** isinstance(error, commands.MissingRequiredArgument):
58. await ctx.send('Missing a required argument')
59. **elif** isinstance(error, commands.BadArgument):
60. await ctx.send('Give an appropriate argument')
61. **else**:
62. await ctx.send("You can't do that")
64. **raise** error
66. **def** setup(bot):
67. '''''Entry point to the "r\_bot.py" file'''
68. bot.add\_cog(Mod(bot))

## owner.py

1. '''''
2. owner.py is the cog that encapsulates all the commands the owner
3. of a given Discord guild (server) would use
4. The commands are: prefix, reload
5. '''
7. **import** json
8. **import** os
9. **from** discord.ext **import** commands
11. async **def** is\_guild\_owner(ctx):
12. '''''Checks if the user is the guild owner'''
13. **return** ctx.author.id == ctx.guild.owner.id
15. **class** Owner(commands.Cog):
16. '''''Encapsulates all moderation commands in the Owner class'''
17. **def** \_\_init\_\_(self, bot):
18. self.bot = bot
20. @commands.command(aliases=['pref'], hidden=True)
21. @commands.check(is\_guild\_owner)
22. async **def** prefix(self, ctx, \*, pre):
23. '''''Changes the bot's prefix within a guild'''
25. file\_path = os.path.dirname(\_\_file\_\_)
26. prefix\_file = os.path.join(file\_path + "\\..\\data\\prefixes.json")
27. with open(prefix\_file, 'r') as \_f:
28. prefixes = json.load(\_f)
30. pre = prefixes[(ctx.guild.id)]  # Checks the prefix assigned to the guild
31. msg = await ctx.send(f'Guild prefix is `{pre}`')
32. await msg.pin() # Pins the message to the channel
34. with open(prefix\_file, 'w') as \_f:
35. json.dump(prefixes, \_f, indent=4)
37. @prefix.error
38. async **def** \_prefix\_error(self, ctx, error):
39. # Underscore implies this is not a command to be used
40. '''''Runs when the prefix error is raised'''
42. **if** isinstance(error, commands.MissingRequiredArgument):
43. await ctx.send('Specify a prefix')
44. **if** isinstance(error, commands.BadArgument):
45. await ctx.send('Set an appropriate prefix')
47. @commands.command(aliases=['r'])  # Hides from the help function
48. @commands.has\_permissions(administrator=True)
49. async **def** reload(self, ctx, cog): # Reloads a cog within discord
50. '''''Reloads the cog within discord'''
52. **try**:
53. self.bot.unload\_extension(f'cogs.{cog}') # Unloads and reloads the cog
54. self.bot.load\_extension(f'cogs.{cog}')
55. await ctx.send(f'```{cog} was reloaded```')
56. **except** Exception as \_e:
57. await ctx.send(f'```{cog} cannot be loaded```')
58. **raise** \_e
60. **def** setup(bot):
61. '''''Entry point to the "r\_bot.py" file'''
62. bot.add\_cog(Owner(bot))

## level.py

1. '''''
2. level.py is the cog that encapsulates all the algorithms the bot uses
3. to maintain a database of users info (user id, guild id, level and experience).
4. The events are: on\_message
5. The commands are: level
6. '''
8. **import** discord
9. **from** discord.ext **import** commands
11. **class** Level(commands.Cog):
12. '''''Encapsulates all algorithms & commands in the Levels class'''
13. **def** \_\_init\_\_(self, bot):
14. self.bot = bot
16. async **def** lvl\_up(self, user):
17. '''''The algorithm that updates the user's level'''
18. cur\_xp = user['xp']
19. cur\_lvl = user['level']
21. **if** cur\_xp >= round((4 \* (cur\_lvl \*\* 3)) / 5):
22. # The algorithm to compute the next current xp
23. await self.bot.pg\_con.execute(
24. """
25. UPDATE users
26. SET level = $1
27. WHERE user\_id = $2 AND guild\_id = $3
28. """,
29. cur\_lvl + 1, user['user\_id'], user['guild\_id'])
30. **return** True
31. **return** False
33. @commands.Cog.listener()
34. async **def** on\_message(self, ctx):
35. '''''Event is called on every message recieved by the bot and levels up the user'''
36. **if** ctx.author == self.bot.user: # Doesn't level up the bot itself
37. **return**
39. user = await self.bot.pg\_con.fetchrow(
40. """
41. SELECT \*
42. FROM users
43. WHERE user\_id = $1 AND guild\_id = $2
44. """,
45. ctx.author.id, ctx.guild.id)
46. # Dollar sign used as placeholder for variable names not defined in SQL
48. **if** **not** user: # Creates a new user if user is returned false
49. user = await self.bot.pg\_con.fetchrow(
50. """
51. INSERT INTO users (user\_id, guild\_id)
52. VALUES ($1, $2)
53. RETURNING \*
54. """,
55. ctx.author.id, ctx.guild.id)
57. await self.bot.pg\_con.execute(
58. """
59. UPDATE users
60. SET xp = $1
61. WHERE user\_id = $2 AND guild\_id = $3
62. """,
63. user['xp'] + 1, ctx.author.id, ctx.guild.id)
64. # Finds the user and gives them +1 xp
66. **if** await self.lvl\_up(user): # Sends a mention to the user that they have levelled up
67. await ctx.channel.send(f"{ctx.author.mention} is now level {user['level'] + 1}")
69. @commands.command(aliases=['lvl'])
70. async **def** level(self, ctx, member: discord.Member = None):
71. '''''Shows the level of members in the current guild'''
72. **if** member == self.bot.user:
73. **return**
75. member = ctx.author **if** **not** member **else** member
77. user = await self.bot.pg\_con.fetchrow(
78. """
79. SELECT \*
80. FROM users
81. WHERE user\_id = $1 AND guild\_id = $2
82. """,
83. member.id, member.guild.id)
85. **if** **not** user:
86. user = await self.bot.pg\_con.fetchrow(
87. """
88. INSERT INTO users (user\_id, guild\_id)
89. VALUES ($1, $2)
90. RETURNING \*
91. """,
92. member.id, member.guild.id)
93. # Creates a new user if user is returned false
95. await self.bot.pg\_con.execute(
96. """
97. UPDATE users
98. SET xp = $1
99. WHERE user\_id = $2 AND guild\_id = $3
100. """,
101. user['xp'], member.id, member.guild.id)
103. embed = discord.Embed(
104. color=member.color,
105. timestamp=ctx.message.created\_at)
107. embed.set\_author(name=f"Level - {member}", icon\_url=member.avatar\_url)
108. embed.add\_field(name='Level', value=user['level'])
109. embed.add\_field(name='Experience', value=user['xp'])
111. await ctx.send(embed=embed)
112. **print**(user['user\_id'])
114. **def** setup(bot):
115. '''''Entry point to the "r\_bot.py" file'''
116. bot.add\_cog(Level(bot))

## voice.py

I found that creating a voice cog was too hard for me to work with, so I modified this *basic\_voice.py* [20]so that it became compatible with my bot (i.e. using a cog structure, since the initial script just ran on it’s own, without any other files).

1. **import** asyncio
2. **import** discord
3. **from** discord.ext **import** commands
4. **import** youtube\_dl
6. # ------------------------- Voice channel -------------------------
8. # Suppress noise about console usage from errors
9. youtube\_dl.utils.bug\_reports\_message = **lambda**: ''

12. ytdl\_format\_options = {
13. 'format': 'bestaudio/best',
14. 'outtmpl': '%(extractor)s-%(id)s-%(title)s.%(ext)s',
15. 'restrictfilenames': True,
16. 'noplaylist': True,
17. 'nocheckcertificate': True,
18. 'ignoreerrors': False,
19. 'logtostderr': False,
20. 'quiet': True,
21. 'no\_warnings': True,
22. 'default\_search': 'auto',
23. 'source\_address': '0.0.0.0'
24. # Makes all connections via IPv4
25. }
27. ffmpeg\_options = {
28. 'options': '-vn'
29. }
31. ytdl = youtube\_dl.YoutubeDL(ytdl\_format\_options)
33. # ------------------------- Voice cog -------------------------
35. **class** YTDLSource(discord.PCMVolumeTransformer):
36. '''''Gets the youtube source from the url'''
38. **def** \_\_init\_\_(self, source, \*, data, volume=0.5):
39. super().\_\_init\_\_(source, volume)
41. self.data = data
43. self.title = data.get('title')
44. self.url = data.get('url')
46. @classmethod
47. async **def** from\_url(cls, url, \*, loop=None, stream=False):
48. loop = loop **or** asyncio.get\_event\_loop()
49. data = await loop.run\_in\_executor(None, **lambda**: ytdl.extract\_info(url, download=**not** stream))
51. **if** 'entries' **in** data:
52. # take first item from a playlist
53. data = data['entries'][0]
55. filename = data['url'] **if** stream **else** ytdl.prepare\_filename(data)
56. **return** cls(discord.FFmpegPCMAudio(filename, \*\*ffmpeg\_options), data=data)
58. **class** Voice(commands.Cog):
59. '''''Encapsulates all the voice commands in the Voice cog'''
61. **def** \_\_init\_\_(self, bot):
62. self.bot = bot
64. @commands.command()
65. async **def** join(self, ctx, \*, channel: discord.VoiceChannel = None):
66. """Joins a voice channel
67. If no channel is given, the bot joins the author's current voice channel"""
69. channel = channel **or** ctx.author.voice.channel
71. **if** ctx.voice\_client **is** **not** None:
72. **return** await ctx.voice\_client.move\_to(destination)
74. await destination.connect()
76. @commands.command()
77. async **def** play(self, ctx, \*, query):
78. """Plays a file from the local filesystem"""
80. source = discord.PCMVolumeTransformer(discord.FFmpegPCMAudio(query))
81. ctx.voice\_client.play(source, after=**lambda** e: **print**('Player error: %s' % e) **if** e **else** None)
83. await ctx.send(f'Now playing: {query}')
85. @commands.command()
86. async **def** yt(self, ctx, \*, url):
87. """Plays from a url (almost anything youtube\_dl supports)"""
89. async with ctx.typing():
90. player = await YTDLSource.from\_url(url, loop=self.bot.loop)
91. ctx.voice\_client.play(player, after=**lambda** e: **print**('Player error: %s' % e) **if** e **else** None)
93. await ctx.send(f'Now playing: {player.title}')
95. @commands.command()
96. async **def** stream(self, ctx, \*, url):
97. """Streams from a url (same as yt, but doesn't predownload)"""
99. async with ctx.typing():
100. player = await YTDLSource.from\_url(url, loop=self.bot.loop, stream=True)
101. ctx.voice\_client.play(player, after=**lambda** e: **print**('Player error: %s' % e) **if** e **else** None)
103. await ctx.send(f'Now playing: {player.title}')
105. @commands.command()
106. async **def** volume(self, ctx, volume: int):
107. """Changes the player's volume, in the range [0, 100]"""
109. **if** ctx.voice\_client **is** None:
110. **return** await ctx.send("Not connected to a voice channel.")
112. ctx.voice\_client.source.volume = volume / 100
113. await ctx.send(f"Changed volume to {volume}%")
115. @commands.command()
116. async **def** pause(self, ctx):
117. """Pauses the player."""
119. **if** ctx.voice\_client:
120. ctx.voice\_client.pause()
122. @commands.command()
123. async **def** resume(self, ctx):
124. """Resumes the player."""
126. **if** ctx.voice\_client:
127. ctx.voice\_client.resume()
129. @commands.command()
130. async **def** stop(self, ctx):
131. """Stops the song playing and disconnects the bot from the channel"""
133. await ctx.voice\_client.disconnect()
135. @play.before\_invoke
136. @yt.before\_invoke
137. @stream.before\_invoke
138. async **def** ensure\_voice(self, ctx):
139. **if** ctx.voice\_client **is** None:
140. **if** ctx.author.voice:
141. await ctx.author.voice.channel.connect()
142. **else**:
143. await ctx.send("You are not connected to a voice channel.")
144. **raise** commands.CommandError("Author not connected to a voice channel.")
145. **elif** ctx.voice\_client.is\_playing():
146. ctx.voice\_client.stop()
148. **def** setup(bot):
149. '''''Entry point to the "r\_bot.py" file'''
150. bot.add\_cog(Voice(bot))

# Testing

* Clear evidence, in the form of carefully selected representative samples, that thorough testing has been carried out. This demonstrates the robustness of the complete or nearly complete solution/thoroughness of investigation and that the requirements of the solution/investigation have been achieved.

[Take screenshots of testing unique functions]

To see if the bot has met its requirements, I have tested the bot against the core objectives:

## Read and respond to commands by users, using a compound command prefix, inside a single discord guild.

I created a simple test guild, which I can freely test out the bot’s commands (without annoying other users). For some user specific testing, I asked my end users, *Faizan#4522* & *Brandon#8004*, to give feedback on testing the bot against their respective core objectives.

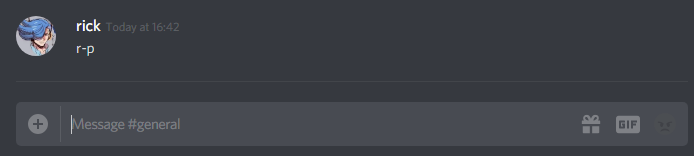


Figure 3: Using the compound prefix “r-” as well as the command “p” (an alias for “ping”), I received no response at all from the bot, similar to my client. I was expecting the bot to respond with the message “Pong!” – and then stating the latency between the command being sent to execution.



Figure 4: This turned out to be an Attribute Error; my bot had no attribute “say”. I used the line “await bot.say(…)” to send the message, but I had found out that this was due to the different versions of Discord I was working on – compared to the video tutorial I was following. I promptly translated & updated all the old syntax accordingly. In this case, the attribute “bot.say” was turned into “ctx.send” where “ctx” is the parameter my bot takes in the message as context.

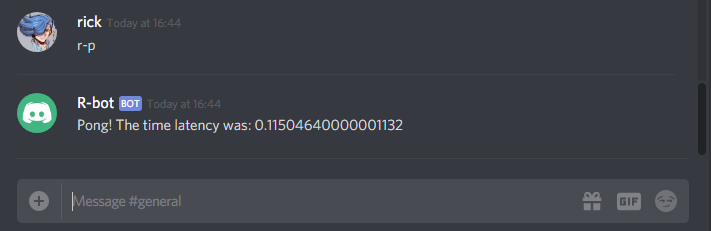


Figure 5: Alas, the bot responded as intended.

## PM (private message) users a list of its commands.

*FAIZAN#4522: The bot does respond to the command “r-h” (in the same way as “r-help”) by privately messaging me a list of commands. This is very helpful, but it is quite hard to see what all of the commands do – because they are clumped together, it would be easier for me to read if they were presented in a more aesthetically pleasing way.*

## Join VoIP music channels with other users and stream music via YouTube/Spotify/Soundcloud.

## Enable specific user/role commands to admin users through the bot by means of temporary kicks, server bans, channel mute and changing nicknames.

## Enable a method of reloading bot command scripts during its runtime

To test out the bot, I had my end users [x & y] give feedback regarding the bot’s processes and whether it meets their expectations & requirements.

# Evaluation

* Full consideration given to how well the outcome meets all its requirements
* How the outcome should be improved if the problem was revisited is discussed and given detailed consideration
* Independent feedback obtained of a realistic and useful nature, evaluated and discussed in a meaningful way

## Objectives met

## Objectives unmet

## User feedback

## Future improvements

# Sources

*Correct as of October 2018:*

[0] - <https://en.wikipedia.org/wiki/Discord_(software)>

[1] - <https://discordapp.com/developers/docs/topics/oauth2#bots>

[2] - <https://discordbots.org/list/top>

[3] - <https://www.pokecord.com/>

[4] - <https://www.pokecord.com/commands/>

[5] - <https://discordbots.org/bot/pancake>

[6] - <https://pancakebot.org/>

[7] - <https://discordbots.org/bot/osrs>

[8] - <https://discordapp.com/developers/docs/legal>

[9] - <https://www.youtube.com/playlist?list=PLW3GfRiBCHOiEkjvQj0uaUB1Q-RckYnj9>

[10] - <https://discordpy.readthedocs.io/en/rewrite/index.html>

[11] - <https://pylint.readthedocs.io/en/stable/intro.html>

[12] - <https://www.toptal.com/chatbot/how-to-make-a-discord-bot>

[13] - <https://discordpy.readthedocs.io/en/rewrite/discord.html>

[14] - <https://www.quora.com/How-do-I-design-a-flowchart-for-an-event-driven-program>

[15] - <https://discordpy.readthedocs.io/en/rewrite/ext/commands/api.html#discord.ext.commands.Cog>

[16] - <https://www.postgresql.org/>

[17] - <https://discordapp.com/developers/docs/reference>

[18] - <https://www.cnet.com/products/lenovo-h420-7752-core-i3-2100-3-1-ghz-monitor-none-series/>

[19] - <https://www.samsung.com/global/galaxy/galaxy-s8/specs/>

[20] - <https://github.com/Rapptz/discord.py/blob/rewrite/examples/basic_voice.py>

[21] - <https://discordpy.readthedocs.io/en/rewrite/migrating.html>