Logo

Description automatically generatedAnalysis

# Introduction

For my project, I have produced the idea of developing a bot for the social media site, Discord. Discord is a widely used social network, with apps for desktop and mobile, and a user base of over 240 million as of December 2020. Discord consists of servers, in which up to 800,000 users (about half the population of Hawaii) can join and hang out with people from around the world. These are free to create and some of the biggest are owned by big companies and game developers as a community for their user base. A good example of this is Valorant, a game whose server has already reached the cap of 800,000, as of January 2021.

For these servers to run smoothly, they enlist the help of some of Discord’s many user-developed bots. These bots serve different purposes based on that chosen by the author of the bot. Some bots handle moderation, others entertainment, some exclusively music, a few utilities based and some multipurpose, in slightly less detail than those that focus on a single category. Some of the most popular bots currently occupy over 10,000,000 servers!

# The Theory Behind My Concept

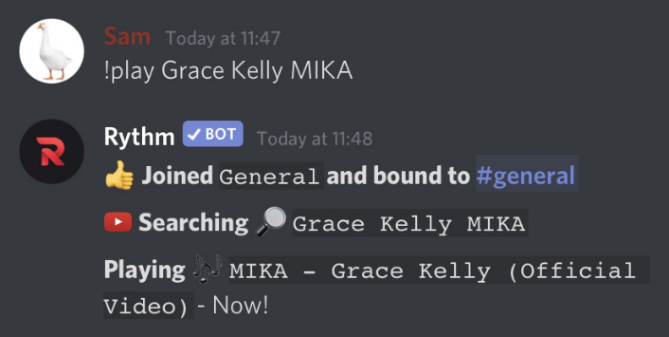
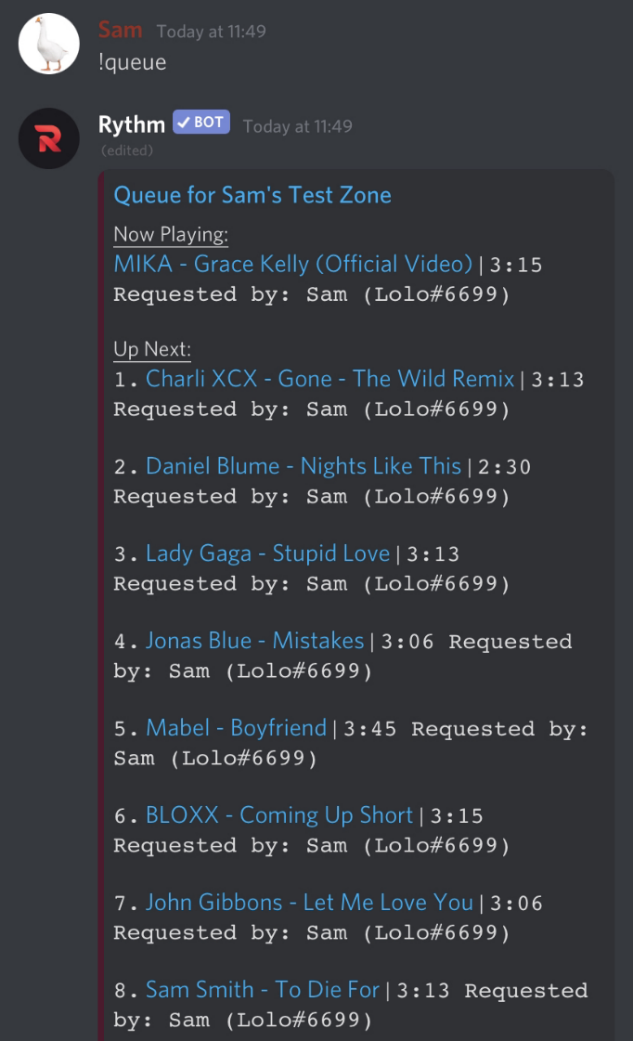
The category I have decided to focus my attention on is music. I have made a few discord bots before, 1 public and then others small personal projects for myself or a group of friends. However, about 2 years ago, when I first decided to try and make a Discord bot, I wanted to make a music bot. However, I did not have enough knowledge of python, or Discord to make this a reality. Since then, I have homed in on my programming skills and learnt how to make a Discord bot to a higher level, leading me to want to recreate my initial project 2 years ago and finally make it work.

Music bots are the second most popular type of bot on Discord. As of January 2021, the largest public music bot, called Rhythm, was currently in over 18,000,000 servers, with the second most popular, Groovy, in over 13,000,000 servers. They allowed you and a group of friends to hop into a voice channel within a server and listen to music together. You simply run a command, such as !play <Song>, and the bot joins the voice channel with you and starts streaming a song through the bot user into the channel. This makes for a fantastic way to listen to music at the same time as your friends, and can be used for events in big servers, such as karaoke, or guess the song, as well as for personal use in one of Discords many hangout servers. However, those Music Bot’s have since been shut down by YouTube for violating their terms of service. This is due to the use of streaming audio from YouTube videos without the video, as well as monetizing their services.

For my music bot, I am going to be taking onboard some of the features found on these mainstream bots, however I will be using Soundcloud as a source of music due to their terms of service allowing creating of custom audio players as long as the artist is credited, and a Soundcloud logo is visible within the player. I will also be attempting to add a few features that I do not currently see that I think would fit nicely, after discussion with my end user about these features.

Basic Commands for Music Bots:

* **!play** **->** *Allows a user to play a song in their current voice channel.*
* **!queue ->** *Displays the current song queue for the user's voice channel.*
* **!skip ->** *Skips the currently playing song.*
* **!nowplaying ->** *Shows information about the currently playing command.*
* **!stop ->** *Stops audio playback & disconnects the bot from the voice channel.*

These are necessary for the function of a well-built music bot, and features that would be expected by any end user who invites your music bot to their server. Below is an example of some of these features in action on Rhythm, Discord’s most popular music bot.

Most Discord music bots also support multiple platforms to play music from. The most common of these being YouTube, Soundcloud, Spotify or direct HTML stream links. Some bots also support Apple Music, Deezer and MP3 files uploaded to Discord; as well as Twitch & Mixer streams, however, this is less popular. Music bots typically support the playing of tracks and playlists, and this is a feature also expected by the average end user. To keep it simple & to comply with companies’ terms of service, I have decided to use only Soundcloud, Spotify & direct HTML stream links. Text can also be entered to search for a song on Soundcloud.

One of the most important things to users is the stability of the music bot they are going to use. It is an expectation for their bot of choice to have a long uptime, with minimal un-planned downtime that might impact their listening experience. Ease of use is also another big thing to consider. A bot needs to be self-explanatory, and easy to understand without a tutorial. The help command must also be detailed enough to explain how to use those tricker commands! These are two things that I will make sure I consider throughout the whole length of the project.

The main problem I have with all these bots is their lack of use of third-party API’s. An API, or Application Programming Interface, is a set of functions and procedures that allow for applications to access data and features of other applications or services. A lot of modern-day services have an API to allow external developers to use their data in third party applications and services. The most common type of API is a REST API, also known as RESTful API, which is best of the REST architectural system, also known as representational state transfer, first created by computer scientist Roy Fielding. When a client requests, it returns a representation of the state of the resource being requested for. This is often in JSON form but could also be in a range of other forms including html or just plain text.

The most important API’s which I believe are missing are some of the mainstream music API’s, such as the Spotify API. I feel like the wealth of information available through these services is something that should be integrated to give those with an interest in music a better experience when using the bot. This sort of advanced information would not only provide to those with an already wide experience and passion for music, but also those who are curious to learn that little bit more about their favorite songs.

# What I Want to Achieve in my Project

My project hopes to be an expansion of the current range of Discord music bots, taking onboard the basics of what makes them successful and adding some new creative expansion to this in order to give my end user a unique experience.

My music bot will function in a similar way when playing music to the mainstream bots. It will have a play, skip, and queue command, as well as a volume command, a command usually reserved for most music bots’ premium plan. These will be designed slightly differently, giving the user more information about the content they are listening to. I also do not plan to include a premium plan into my bot.

One unique feature that I’d like to add is the ability to get recommended songs based on the user’s listening history. This would show a selection of songs tailored to the user who ran the command, using Spotify’s recommendations algorithm with my generated figures. I will also be able to display the genre of a song, using a custom genre prediction algorithm within the bot. This is a feature I couldn’t find available on any music bot’s during my research at the start of 2021, and feels like a unique angle, allowing me to give a more music orientated approach to listening to music through discord, something which is a big aim within this project.

The final feature I plan to add is the ability to link your Spotify account with your Discord account. This data could then be used when giving the user recommendations, if specified by the user to be used instead of their listening history, however, this wouldn’t be default. This is also something not found on any current mainstream music bot, however, is something I believe would improve the overall experience, as users would no longer have to go onto Spotify and find the link to their desired track or playlist and could just type in the name of the playlist they want to play into Discord itself.

# Meeting With my End User

For my meeting with my end user, I wanted to find someone with good knowledge of Discord, who had both used plenty of Discord Bots, including music bots, as well as someone who had owned some severs and knew what would be required for a server owner to want to use one bot over another bot. My end user is Brad, an experienced server owner, who owns a couple of large severs, 500+ members, and has also set up and managed plenty of other servers for friends. He also has plenty of interest in music, spending a lot of his time enjoying a wide variety of songs, and sharing his interest in music with others. This made him a perfect example of my target audience. Below is a transcript of our conversation:

**Sam:** *For my project, I’ve decided to make a Discord music bot, which I hope will fill some of the gaps in the market. These include song recommendations, song history and advanced information about tracks. Do you think these are features that you would make use of?*

**Brad:** *Most definitely. I think the song recommendations is a great idea, as I always struggle to find new music to listen to, and it's something that a lot of the bots already existing lack, and something that I can only do via streaming services such as Spotify, so it would be great to have this accessible through a Discord bot. Personally, I don't take a great interest in the finer details of a song, but rather just basic information such as genre, release dates and song length, but I can defiantly see the appeal of the more advanced information for each song. The song history sounds like a great feature too, as especially if someone else plays a song I enjoy, I can easily go back and see the information about that song.*

**Sam:** *Of course, you have an interest in music. Based on your interests and experience with other music bots, are there any features you would particularly like to see on this bot?*

**Brad:** *I'd recommend perhaps including a command to research more into a song by entering the name and the artist, and it gives you all the info you need. With the information, I'd like to know like song length, publish date, if the song is explicit, as well as genres, and even possibly recommendations for other songs/artists.*

**Sam:** *Okay, next question. Do you use Spotify as a streaming service, and if so, would you find it useful to have a proper integration between Spotify & the music bot?*

**Brad:** *Yes, I do use Spotify as a streaming service daily as a matter of fact, so it would* *definitely be useful to integrate Spotify with the music bot as I feel like this would save myself a lot of hassle trying to import my music preferences onto the bot, and it would also allow for me to share my playlists with my friends with ease, so I think it would be a great feature!*

**Sam:** *Thanks for the answers so far. Finally, I just wanted to get a gauge of your expectations when using a bot on Discord. Things like ease of use, etc. Are any of these more important than others?*

**Brad:** *When using a Discord bot, I really value the presentation. I like to ensure that it's easy to use, without explanation to server users to make mine and moderators' lives a bit easier. With a music bot, I expect the audio to be clear and of a good quality where possible to make the experience all the better. Additionally, I expect basic features such as play, pause, skip, and even previous songs. It's also essential that there are commands that allow you to see the song queue, as well as the ability to remove songs from the queue. Just the basic stuff really is essential, plus the other features I mentioned earlier would help form a great bot!*

# User Requirements

For my project, I have set myself a variety of goals which I believe are achievable. For each of these goals, I have also specified the commands needed to achieve this goal and the inputs that could be made by the user when running the commands. These are listed below:

**1) Play music through discord from a variety of sources.**

1.1) Support multiple inputs, including Soundcloud & Spotify.

1.2) Give user complete control over the playback of music

*1.2.1) User should be able to skip songs.*

*1.2.2) Songs should be able to be paused & un-paused.*

*1.2.3) Where possible, songs should be able to be skipped forwards & backwards.*

1.3) The song currently playing should be easily identifiable, or easy to find out when requested.

**2) Provide users with as much information about songs as is reasonable** **to.**

2.1) Always provide at least basic information about songs.

2.2) Provide advanced song information when requested.

**3) Store the last 20 songs listened to by a user through the bot.**

3.1) Record listening history for all songs, regardless of whether the user requested to play it or not.

3.2) Return complete listening history to user where requested.

3.3) Users should be able to get track recommendations based on their listening history.

*3.3.1) Users with a linked Spotify should be able to use this as well.*

**4) Give users complete control over their data.**

4.1) All stored data should be deleted at the user's request.

4.2) Provide as much flexibility as possible to what data can be deleted by the user.

**5) Provide a simple and easy to navigate user interface.**

5.1) Create a multi-tiered help command with different levels of detail, right down to individual commands.

*5.1.1) All help commands should follow the same structure to make them easy to understand.*

*5.1.2) Commands and categories not relevant to the user should not be shown to avoid unnecessary confusion.*

5.2) All data should be presented in a clear manner and paired with similar, relevant data points.

5.3) All titles, sub-headings, and images should be relevant to the data they’re paired with.

**6) Create a robust and reliable bot.**

6.1) All errors should be caught and returned to the user.

6.2) Each error should have a unique error message with appropriate steps for the user to take to resolve the issue.

6.3) Uptime of the bot should be maximized.

*6.3.1) Every command should be able to be reloaded should minor changes need to be made.*

*6.3.2) Anything that may need to be altered periodically should be able to be done without interrupting the operation of the bot.*

**7) Support a secure integration between the music bot and Spotify.**

7.1) Create a web server to allow for a seamless account linking process.

*7.1.1) The web server should handle errors by displaying appropriate error pages.*

*7.1.2) The process should be easy to follow from start to finish.*

7.2) Accounts should be able to be unlinked and any stored data deleted at any point.

# Technical Requirements

To achieve my desired goals, there are certain technical elements that I need to include to ensure the smooth delivery of my user requirements. Below is a brief overview of some of these elements, what they will be made up of, as well as what user requirements they are necessary for:

**1) Create a class to interface between the discord bot and the Spotify Web API. Necessary for user requirements 1 & 2.**

1.1) Create a function for each API call.

1.2) Handle errors with good feedback when they occur to the bot admin.

*1.2.1) Refresh token automatically once it has expired.*

*1.2.2) Back off for the necessary amount of time when a rate limit has been encountered.*

1.3) Return all data in a similar universal format.

**2) Create a relational database using MySQL to store information about the end user. Necessary for user requirements 1, 2, 3 4 & 7.**

2.1) Data is fetched from python using properly formatted and efficient SQL statements.

2.2) Create multiple linked tables each storing different data that works together well.

**3) Create an efficient audio player using Lavalink. Necessary for user requirements 1 & 2.**

3.1) Bot should automatically join & leave the voice channel.

3.2) Keep information about currently playing & stored tracks for external use by commands.

3.3) Audio players should be able to be controlled from within multiple different commands.

**4) Interface with 3rd Party APIs to gather additional data about songs. Necessary for user requirements 1, 2 & 3.**

4.1) Create a class with different functions to get data from each accessed method within the API.

4.2) Handle any errors or possible issues automatically within a contained class.

*4.2.1) Bot tokens should both be automatically generated and refreshed when necessary.*

*4.2.2) Errors should be returned when encountered to where function was called from initially.*

*4.2.3) Rate limits should be automatically waited until the rate limit has passed.*

**5) Handle errors using a global error catcher system. Necessary for user requirement 6.**

5.1) Error catcher should always be running.

5.2) Errors that seem out of place should stop code execution to prevent potential abuse of code.

**6) Maximize uptime using a reloading system for minor code changes. Necessary for user requirement 6.**

6.1) Split commands into cogs to allow categorization of commands and not all commands having to be reloaded at once.

6.2) Allow all cogs and commands to be reloaded individually by developers.

# My Prototype

For my prototype, I wanted to test out some of the most important features of my project. This includes my genre prediction algorithm, as well as how I’ll interface with the Spotify API, as that is my most important web API, and I want to make sure that it is efficient and I’m able to get the most out of the resources available.

To do this, I created some small python files to test individual requests and how I might need to format the data that came back. The most important test file was for the Spotify recommendations API call, as this allowed me to keep testing different values and observe how adjusting them might impact the recommendations returned for the user. This is something I could then use to my advantage to provide the end user with the most accurate suggestions, whilst also keeping the recommendations fresh time after time.

After I knew the recommendations worked, I set to work on my genre prediction algorithm, another defining feature of my project that I needed to get right before I went forwards. To do this, I wrote a script that allowed me to write a list of audio features for each song to a file.

Audio features are float values between 0 & 1 that are assigned by Spotify and represent how much of each feature a song has. These were placed within a dictionary based on their genre which was then all stored in a json file.

To get the songs, I made a request to get every playlist from Spotify under a category which corresponded with the genre I was trying to get. For each of the playlists in this category, I went through each song, and made a request to Last.fm, a music site where users can assign tags to songs. I then checked if the song in question had a tag corresponding to the genre needed in its top tags, and if it did, I then added it to the dictionary. At the end of the script, the data is then dumped back into the json file.

This data is them loaded onto a Decision Tree when the bot is first loaded up. The decision tree is a machine-learning modeling technique used for regression and classification problems. When data is inputted, it makes a hierarchical decision about the outcome based on the tree that has been initially produced. This was implemented using the scikit-learn module, a library built for data science and machine-learning in python.

The second to last prototype I decided to make was a stripped-down version of my bot, using the python shell as an output rather than integrating it with Discord. This allowed me to test the more important backend processes involved with some of my initial commands, in particular ones that use the Spotify Web API, without worrying about design and how the data is displayed to the user. For this prototype, I only choose a selection of commands, as some commands rely directly on Discord to work, and others don’t have a large enough backend to make it worthwhile using this prototype method on them.

My final prototype was a simple discord bot integrating the commands I’d originally tested before in shell output only. This simply involved adding in the Discord wrapper alongside my original code to see how it would hold up. I could then test it inside its final environment to make sure it was going to work the way I wanted it to. This required a bit of Discord design practice as well, giving me an idea going forward about how I might want it eventually to all look like. I also had a go catching some of the most common errors I will encounter throughout my project and how I might return this error to the user.

## Evaluation Of My Prototype (See Appendix for Code)

My prototype successfully showed that what I want to do for my project will work, as it functioned well throughout. My final prototype didn’t encounter any issues, and I felt happy with the design and functionality already shown, even at such a simple level. It’s also shown that there is a lot of room for expansion going forward in the project, and even a bit of restructuring once I start on the project itself.

I’ve come out of the prototype phase with a better understanding of the Spotify API, as well as how to work with and handle errors within the Discord.py module. I’ve also been able to show a decently high level of accuracy with my genre prediction algorithm, with the test data collected during this phase.

However, my recommendations set up currently gives a little bit of repetition, which I would like to try and reduce as much as possible going forward. This is currently around 2-3 repeated songs every 3 batches of 10 recommendations, when using a person’s Spotify playlists. However, the account used to test didn’t change any of the playlists throughout testing, so figure does represent a worst-case scenario where no change is made to the data used between command calls. However, I have been able to show it will work, and will use this on song history later in the project using the same system from my prototype, which might influence the song recommendations given. For me, it was more important that the system worked well, as my plan in the long run was to use constantly changing data in the form of listening history, however, I would like to retain the ability to use Spotify playlists as this may suit some user’s needs.

Overall, the prototypes were a positive start to my project, showing that I have a clear path ahead with my current plans, and have given me a good idea about how to approach the completion of my user requirements going forward.

### Stage 1 Evaluation

My first stage of prototype explores first drafts of the essential algorithms that will be needed going forward with my project. Some of these proved to be very successful and weren’t changed much as I progressed throughout my prototyping stages, such as the *Get Playlist Test & Get Song Features.* Some parts of the prediction dating gathering process were also successful, such as the way I used Last.fm, to cross check that the genre of the song was correct before I wrote it to the test data set. However, the overall process of using standard deviations and mean of the test data to predict genres proved to be unreliable, as the overlap between genres was too large, and it became very difficult to pinpoint the specific genre of the song using this method. Nonetheless, the process of using the song features I used within the *History Analyzer* test file towards the end, where I used the data returned from Spotify to get mode, key and duration of the song was very useful, and was able to be reused. Overall, it was a very strong start and gave me a good indication of where I needed to go next with my prototype.

### Stage 2 Evaluation

During the second stage I reworked my previous method of predicting genres and came up with a method using a Decision Tree, which involved storing the data I fetched in a .json file and then fitting it to the decision tree when the code first starts.

This method proved to be significantly more accurate, and I also found from here that the more data I could add to the .json file, the better my Genre Predictions would be, due to the wider variety of data available to predict with. To facilitate this, I re-wrote the original Test Data Writing script, using a similar method to before, however, saving it into a dictionary format, rather than writing it to a text file. This file and its functions proved to be very successful and were used to help write all the thousands of songs worth of data I needed to collect to get accurate predictions. This, plus the *AI Tests* file formed the completed genre predictions algorithm to be used in my project due to how well it functioned, and how easy it was to fine tune the predictions later.

The *AI Tests* file also allowed me to test the accuracy of the genre predictions algorithm I had set up, by putting hundreds of songs under a certain category in Spotify through the predictions algorithm to get a genre and check how many of the predicted genres matched the category. Although this originally seemed like a good idea, the accuracy proved to be low since not every song in one category matched the genre of the category it was under. This was something out of my control, and I had to manually compare my predicted genre with another source of data, Apple Music, to see if my predictions were correct. After doing this manually quite a few times though, it seemed to be accurate, so I continued onwards to create the *Main.py* file.

This was the first time all my work so far had been put together in one large file. It aimed to pull together all my tests to create functions that were going to be needed in my final project, all accessible through the shell. This prototype was formed of 2 classes and a main function, as well as 3 working functions: Getting A Song’s Genre; Getting A Playlist’s Genre; & Analyzing Spotify Profiles. These all worked as I wanted them to, so I decided to move onto my stage 3 prototyping stage and take what I have.

### Stage 3 Evaluation

The aim of stage 3 was to take my *Main.py* file from stage 2, and transition from outputting into the shell to outputting into Discord. This is important as I needed to make sure what I wanted to do was possible within the confines of the target platform. I therefore created a new file called *DiscordBot.py*, which was a long, basic-structured Discord bot which used my previous work as a basis, but built on it massively, taking all the different pieces of prototype code, and turning them into one user function. I also made a script called *web.py* to allow me to handle people linking their Spotify accounts. This worked alongside the discord bot and used states to see which Discord account was making the connection. This proved to be very reliable after some initial testing, and overall, the whole bot code gave a very positive outlook on the project, and made it evident that, although I wanted to do some restructuring, the overall concept I was heading towards was completely achievable.

During this stage, I also decided to experiment with cleaning down my test data which I use for generating the recommendations. I did this by iterating through the json file for each genre and seeing if the same two numbers appeared in more than one genre. I then backwards searched the song in Spotify, and manually checked the song’s generated genre against Apple Music to see which is the actual genre according to that platform. I could then delete the data from the wrong genres, which did have a positive effect on the genres, and I started seeing more accurate recommendations which is overall a very positive outcome. After this, I decided I was ready to start my project, and continue with my discord bot prototype code.