Logo

Description automatically generatedEvaluation

# Completion of User Requirements

At the start of the project, I came up with a series of user requirements, which I have worked to try and achieve during the development of my software. Overall, I think I have managed to achieve all the goals I set myself in the analysis. Below, I have provided evidence for each of the completed user requirements, evaluating how well I managed to complete each requirement.

## User Requirement 1

The first user requirement, to play music through Discord from a variety of sources, was the most important requirement, as it encompasses the core functionality that is needed for a music bot to work.

Text

Description automatically generated

The final set of possible inputs is shown by the help command for play above. It includes a wide variety of different inputs, supporting both SoundCloud and Spotify, as well as HTTP streams, useful for those that might want to listen to the radio, for example. I have managed to get all these inputs working reliably, as shown in tests 2.0.0 through to 2.0.9 from the testing phase. This means I have managed to achieve user requirement 1.1.

However, one input that I had planned to include, but haven’t managed to include on my final project is the ability to plain text search for Spotify playlists if you have a linked Spotify account. This proved difficult to integrate due to needing to differentiate between whether the text inputted is to search for a song on SoundCloud or referencing one of their Spotify playlists. As a possible extension activity, I could implement this feature in one of two ways, either by keeping a list of their Spotify playlists to check the input again, which would need to stay constantly updated, or to use a separate command so that I could search the text against their playlists using the Spotify Web API.

I have also managed to successfully complete user requirements 1.2 & 1.3. User requirement 1.2.1 is completed by the skip command, which was tested by tests 2.4 & 2.4.1. This works reliably and allows users to move through their queue of songs and stay in control of the exact song their listening to. User requirement 1.2.2 is completed by the pause command, which was tested by tests 2.3 & 2.3.1, and allows users to pause the music and continue the song exactly where they left off. User requirement 1.2.3 is also completed by the implementation of the seek command, which was tested in tests 2.8, 2.8.1, 2.8.2, 2.8.3, and 2.8.4. This command gave the user an easy way to get complete control of the position of the playback of the song and proved to be both efficient and error free during testing.

User requirement 1.3 is fulfilled by both the automatic now playing embed system, as well as the now playing command. The now playing command was tested in tests 2.9 & 2.9.1, and despite a slight issue identified in test 2.9, which was a result of an accidently hard coded value and could be easily fixed, it worked as expected. The automatic now playing system can be seen working in tests 2.0 through to 2.0.9, when the play command is being tested. These two systems make it very easy for users to identify the currently playing song from anywhere in the server, regardless of whether the user is one of the ones listening to the song or not. For user requirements 1.2 and 1.3, I’m happy with the support and range of command available, and it isn’t an area I think that needs any extra work.

## User Requirement 2

User requirement 2 has been fulfilled by the info command, which can be seen in tests 3.0, 3.0.1 and 3.0.2. It has both basic and advanced information across two embed pages, with the advanced information fulfilling user requirement 2.2. User requirement 2.1 requires that basic information is always provided, which is true as basic song information can also be found on the now playing system for audio playing, however, the advanced information is exclusive to the info system as it only needs to be shown when requested, so the completion of this user requirement overall has been successful.

## User Requirement 3

User requirement 3 involved the implementation of a listening history storage system for each user. This was implemented using a queue, which I believe suits the use case perfectly, and proved to be reliable in testing.

The system implemented in the final version of the Discord bot records history for all users in the voice channel with the bot, ignoring tracks if they don’t have any track information associated with them, for example, HTTP streams. This is something that could be changed as a possible extension activity going forward, allowing users to see exactly what they had listened to.

The history command is an important part of this system, fulfilling user requirement 3.2. It allows users to see their listening history, which is necessary as user’s should be able to see exactly what data is stored. This system was tested in tests 3.2 & 3.2.1 and worked as expected in both tested scenarios. The final part of the user requirement references the recommendations system, an important focus of the project, as this is something unique and special to the project.

The recommendations shown during tests 3.3, 3.3.1 & 3.3.2 were unique and proved the system was able to work using both listening history and Spotify playlists from linked accounts. Overall, I am very pleased with how this system turned out, with the overlap of recommended very infrequent and only present when a user requests recommendations repeatedly without changing the data being fed into the algorithm, which is something I can’t rectify, as the same data will produce a very similar set of tracks from the Spotify Web API, which is out of my control. I successfully tried to limit this by getting 50 tracks and randomly selecting 10 to show to the user, but there is still a potential 20% chance at least 1 track will overlap. Overall, I am very happy with the system, and think the final implementation was as efficient as possible given the limitations present.

## User Requirement 4

A screenshot of a computer

Description automatically generated with medium confidenceThis is a small, yet important user requirement, as it is important to have implemented to apply with the data protection act. This involves only storing the necessary data and making sure it is kept up to date. It also involves allowing user’s control over their data, meaning they should be able to easily see and delete their own data at their request. Their stored data can be seen using a variety of commands, such as the profile and history commands, which show all stored personal data on a given user.

To give the users the ability to delete their data, I created the delete command, a simple command designed to let them delete all / a selected part of their stored user data. The only table that can’t be manually deleted by users is the cache table, as this doesn’t contain any user data. The command can be seen working in tests 1.4, 1.4.1 & 1.4.2. Although there were some issues in 1.4.2, these were easily fixed, and overall, was successful in giving the users complete control over their data.

## User Requirement 5

Creating an easy-to-use system required a lot of attention to detail throughout the project, making sure to keep it as simple as possible without sacrificing any functionality. To help increase the approachability of my system, I created a help command which can be invoked by anyone at any time, and has a detailed breakdown of all categories of commands, and the commands themselves, giving user’s a good idea of how to use the bot. Each command has a detailed embed with a simple example of how to run the command, and a brief description of each of the parameters than can be passed in, as well as an overall description of the command’s purpose. It was also important to only show commands and categories appropriate to the user in the help command, as per user requirement 5.1.2.

The help command can be seen working in tests 5.0 through to 5.2.2. Although it was slightly unfinished causing some unexpected behaviour during initial testing, the issues where able to be quickly ironed out and the command preformed flawlessly second time round. It has also been designed to tie in nicely with the error messages, with errors giving the exact command for the help embed for the command the user is having issues with, allowing them to easily be able to check out the proper way to run the command, and aiding the user to improve their overall user experience.

User requirements 5.2 and 5.3 set important design principles to follow throughout designing my user interface, which includes both the plain text and embeds sent to the user. It is something I considered during all design throughout my project, and I feel like all data is presented in a clear and easy to interpret manner, with subheadings to make it clear what data is being presented where necessary. All data is also relevantly paired.

## User Requirement 6

Carrying on from making the bot easy to use, having a robust and reliable bot is just as important. Throughout I made sure to program the bot in a way that all errors would be caught and returned to the user. This is important, as it allows me to give to user instructions on how to rectify the issue they are having, improving the overall experience, as per user requirement 6.2. You can see examples of these errors on tests where I have user erroneous data. In tests 5.1.1 & 5.2.1, the bot doesn’t error, leading to the bot failing to respond to the user. This would leave the user confused, which I want to avoid at all costs. Luckily, the bot’s global error system proved to be reliable throughout testing, which shows the bots robustness.

To maximise the uptime of the bot, I implemented a reload command, which is only available to be run by admins, whose DiscordID must be specified in the code, although it can be ran from anywhere. If someone who doesn’t have permissions for a certain command tries to run it, it will return a command not found type error, and doesn’t appear in the help command, like any other command that doesn’t exist, to prevent people from getting confused. The command itself clears up after itself 10 seconds after the reload is finished but should ideally by run in a private text channel. The reload works with any script besides web.py, and overall helps to implement minor changes to code in the event of needing a fix an urgent bug without interrupting the code execution, providing a reliable service.

## User Requirement 7

To implement a secure integration between the music bot and Spotify, I used flask to create a simple, 2-page webpage to allow users to link their Spotify. Using the database, I created a system to force users to first request the link from Discord, so I know exactly what account to link the Spotify to, and to make sure that user has requested to have their account linked. The database also keeps track of when the link request is made in Discord, allowing me to implement a 10-minute timeout on authentication, keeping the process as secure as possible. The web server then generates a random 10-character state, of which there are never more than 1 of the same state, which is passed through Spotify and returned to the web server, with the authentication request being denied if the state doesn’t match. This is another security message taken during the link process to keep it as secure as possible. User data is then encrypted before being stored in the database, to ensure it is kept secure even after the linking process is finished.

The account and the data associated with it can easily be deleted by re-running the link command, which will give an option to unlink the accounts, or by using the delete command. Any errors that occur during the link process are handled using one of three error templates, one for timeout, which occurs when someone tries visiting the URL more than 10 minutes after linking their account. The failure template will be shown when the user has declined to authenticate the bot when redirected back from Spotify. The error template is then shown whenever an unexpected error occurs. Each template has basic instructions for the user about what to do next to re-attempt linking their account, like how error messages work on the bot in Discord. This system proved to work reliably during tests 4.0 through to test 4.3.

# End User’s Evaluation

At the start of the project, I spoke to Brad, an experienced server owner I had met previously through Discord. Now the project is complete, I reached out to Brad again to ask him a few more questions about his view on how the project has gone, and what could be achieved going forward. Below is the transcript from our conversation:

**Sam:** *Hello again, my project is now finished so I’d like to ask you some more questions. First off, looking at the features available on the current bot, what would you say has gone well during this project. Are there any features or commands that stand out to you?*

**Brad:** *The features of the bot are really good from what I wanted from the project really. Something that really stands out to me is the account features. I've been using the '!recommend' command quite frequently when I've used the bot, and it's really helped me to find new music and explore new genres which I haven't really ventured into before. The '!history' command has also been really useful for me as I often like listening to the same music, but can sometimes forget the names of the songs. All the commands are really nicely laid out too, so all positives here!*

**Sam:** *On the other hand, is there any features that have been implemented that you would like to see alterations made to or are there any overall improvements that could be made to the bot?*

**Brad:** *Most of the features I'd expect of a music bot are present on the Alto bot, so nothing really jumps to mind as something that needs urgent implementation. The only thing I could suggest to improve going forward is to potentially have a feature that allows you change the bots prefix, as when its used in servers with other Discord bots, this could potentially clash with other existing bots. By allowing different prefix's other than the default of ‘!’, this would allow for compatibility with other bots. Otherwise, all the features work well!*

**Sam:** *Going forward, if I carry on working on the project, what direction would be good to pursue, and are there any features themselves you’d like to see added?*

**Brad:** *If you were to continue, I'd recommend just making the bot easy to implement on a variety of servers, such as potentially allowing server admins to restrict who can utilise the bot and who can’t. In addition, I'd also just recommend expanding features based on any feedback you receive, and as stated in the previous question, potentially consider implementing a feature which allows you to change the prefix of the bot. I think that if you continued working on this, you would be able to get it into a much better state for mass use across multiple different servers, as its definitely in a good state to achieve that already.*

**Sam:** *Overall, what’s your view on the outcome of the project. Is it something you would consider using in one of your servers and if so, why?*

**Brad:** *I'm really satisfied with the performance and design of the bot. It has been really easy to use when I've wanted to listen to music, both on my own and with friends. As the owner of multiple different community Discord servers, this bot is definitely something I'd consider implementing due to its wide range of features, but I would want the freedom of restricting commands to certain users before implementing the bot in certain servers. In terms of what I asked for in our initial conversation, I'm more than happy about what you've been able to achieve with the bot and I've had a really good experience with it!*

# How Would I Approach the Project Again?

If I was to approach the project again, I think the biggest benefit would be to come up with a much more detailed plan, and make sure to code with the future in mind. Although I had a fairly detailed plan, there were a lot of times where I needed to go back and change code, I’d already written in order to implement new features. Although this is guaranteed to some extent, as for example, I needed to implement the output for nowplaying when adding in the ability to use audio streams later in the project, I ended up completely re-writing the play command about 2 times throughout the project in order to make it easier to expand the command later on, which used up possible time I could’ve used to work on other features. However, this was partially due to me learning new programming methods to make the program run more efficiently as the project progressed.

Another difference I would consider to my approach to the project is the use of third-party software to aid the development process throughout. Once I’d finished my prototype, I started a Github repository for the project, which proved useful. I also created a Trello board for the development of the bot, which acted as a planning board, however, I failed to utilise this much, and eventually it became out of data and abandoned. If I was to redo the project, I think it’d be really helpful to make use of this service throughout, to keep track of what features need to be implemented, what is partially implemented, and what bug fixes need to be made. This is particularly important, as I found myself working on the project for a period, then taking a break, and coming back again to the project to make more progress. The planning board would help if I hadn’t worked on the code for a few weeks to re-familiarise myself with where I was up to quickly and maximise progress.

# Extension Activities

Continuing with the project, my next goal would be to create a better integration with the server’s themselves, giving control over to server owner, similar to what my end user Brad mentioned during our conversation. This would include a DJ only mode, which can either lock the use of the bot to certain roles or certain people, giving ultimate flexibility to the server owner. I’d also be interested in expanding the bot to become integrated with the stage channel, a relatively new addition to Discord that came in during the development of my project and is starting to become supported by the libraries I use. These work by having presenters and allow anyone to listen to the presenters talk about the topic of the event. I think it’d be cool to have a system where event organisers can play songs into these stage channels, and listeners can request songs, ideal for filling time pre-event or post-event.

I’d also like to add to the music playing capabilities of the bot, adding a pre-programmed set of radio stations that can be played by the user through the bot, without having to find the HTML stream link, which can be hard to find for some stations. I’d also like to add the ability to create playlists on the bot, possibly allowing users to save these playlists to their Spotify account, if they have a linked Spotify. This would allow users to listen to playlists without having to have a Spotify or SoundCloud account with playlists.

Furthermore, I’d like to expand on the recommendations concept I created with the command by allowing users to listen to a set of auto-played tracks generated using their listening history. However, this would be fairly difficult to implement, but could be a stand-out addition to the bot, and could be quite fun to experiment with. I have seen some well-renowned bots implemented a paid auto-play feature, like that found on some streaming services, which will continue the playback of the queue, however, I would like to take that one step further, as many streaming services already offer tailored playlists, and would be amazing to see a similar system in Discord.

Finally, I’d like to go back and work more with Interactions, a new way for users to interact with bots in Discord, using slash commands, with cool features such as buttons, and autocomplete on commands. This is a system Discord plans to shift towards all bot’s using, however, interactions were released after I’d completed most of my prototyping and have only recently started to be supported in the python libraries for the Discord API, however, even now, are still in BETA. This made it un-reasonable to include them in my project, and to support them, it would have required coding the bot significantly differently than how I’d planned, which didn’t make it appealing to work with at the time of doing the project. However, once these are officially released, it’d be nice to go back and experiment with these methods of programming with a Discord bot.