Wild Card Bot

Design Document

Team 25:

Konstantin Liehr
Arun Thirumavalavan
Nikolas Damalas
Kyle Arrowood
Michael Gu
Matthew Wang

Index

Purpose

The system we are designing is a bot for the popular communication platform Discord. There are many discord bots readily available online, but it can be difficult to find the right mix of bots to serve the needs of your server without cluttering it with extra features or varying syntax.

We have all ran into the issue of needing various bots on our discord servers but only to find they have conflicting command usage or just are not compatible. For each new feature a server owner wants, they need to scour the internet for a bot, give it permission, and hope it doesn't collide with the currently installed bots.

Current multipurpose bots such as nugget bot, MEE6, and YAGPDB.xyz all contain powerful server administration tools along with fun features such as roles, server stats, and fun additions. Only MEE6 allows some basic user defined functions, so while these bots are "multipurpose" they are not "all-in-one". You still need to find other bots and install them to do functions such as playing music, making polls, or integrating other APIs.

The Wild Card Bot will be an all-in-one discord bot with the ability to allow server owners to customize their bot via the modular function system. This system will allow server owners to easily add specific functions or features to the bot that can be written by anyone, allowing the server owners to avoid code entirely if they prefer. We will allow users to import functions either by placing the file in the functions folder of the bot or by simply using the discord file sharing feature and a command. The ability to fully customize what the bot can and cannot do allows users the freedom to tailor the bot to the needs of their server, similar to how a wild card in card games can be anything it needs to be.

In addition, the Wild Card Bot will contain basic server administration functions. This includes modifying permissions and roles, creating and moderating channels, and the ability to remove users from the server via simple commands. Server owners will also have the ability to change which users have the ability to access certain commands and features in the bot to ensure the more powerful server tools are only accessible by trusted individuals. Additionally, if server owners require more administration tools they'll be able to add others later via the modular feature system.

We will also be designing a number of additional features for the Wild Card Bot to show off the capabilities and potential for the modular function system. These additional features include: a League of Legends game helper via Riot API, a music player, a google search tool, basic stock information, decision making tools, a coding reference documentation tool, a poll maker, and clock integration.

Functional Requirements

1. Modular Function System:
    a. As a Server Administrator, I want to add functions to the bot using Discord's file sharing feature.
    b. As a Server Administrator, I want to remove functions from the bot with a command.
    c. As a Server Administrator, I want to add and remove functions to the bot via a function folder.
    d. As a developer, I want to dynamically load functions and features to the bot from separate files.
    e. As a developer, I want functions to be able to use libraries not found in the main bot.
    f. As a developer, I want to know what format to code commands in.
    g. As a developer, I want to know what languages I can code commands in.
    h. As a Server Administrator, I want to know if the function I am adding uses the same command as a currently implemented command.
    i. As a Server Administrator, I want to specify and change which roles can use a command.
    j. As a Server Administrator, I want to know where I can download additional modules.
2. Administration Services:
    a. As a Server Administrator, I want the bot to automatically delete messages containing banned words
    b. As a Server Administrator, I want the bot to warn users if they try to use a banned word.
    c. As a Server Administrator, I want the bot to warn users if they try to use a banned word.
    d. As a Server Administrator, I want the bot to kick/ban users with a command.
    e. As a Server Administrator, I want the bot to kick/ban users with a command.
    f. As a Server Administrator, I want to assign and change user roles with a command.
    g. As a Server Administrator, I want the bot to have the ability to remove messages with links in certain channels.
    h. As a Server Administrator, I want to assign and change user roles with a command.
3. Basic Bot Usage: As a user,
    a. I want to know what commands are available to use with the bot.
    b. I want to know what a command does and ways to invoke it.

4. Music Function: As a user,
    a. I want to play songs from youtube by providing a youtube link.
    b. I want to queue multiple songs to be played.
    c. I want to add and remove songs from the queue.
    d. I want to view the queue.
    e. I want to clear the queue.
    f. I want to skip a song.
    g. I want to have a private channel where I can listen to music without the interference of other users.
5. Decision Maker: As a user,
    a. I want to have the bot randomly pick between a set of options to help me make decisions.
    b. I want to have the bot to randomly generate a number between a range of values.
6. Poll Maker: As a user,
    a. I want to have the bot generate a poll that members of the server can vote on.
    b. I want the bot to announce the result where the poll was created with voting statistics.
7. Google Search: As a user,
    a. I want to get a google search result in the server with a command.
    b. I want there to be an option to jump to the search in my browser from an extra command.
    c. I want there to be an option for the bot to give me a short answer rather than extra unneeded information.
8. Basic Financial Info: As a user,
    a. I want to see basic stock information when I enter a ticker in chat
    b. want to see expanded stock information.
    c. I want to get a link with more info about the stock.
9. Voice Interaction: (If time allows) As a user,
    a. I want to speak to the bot telling it to do different commands.
    b. I want the bot to talk back to me. (Optional feature)
10. Game statistics display (Riot and Steam (If time allows)): As a user,
    a. I want to display my overall game statistics for a game of choice.
    b. I want to actively know what games my friends play.
    c. I want to be able to see when me or my friends have sales on their wishlist.
    d. I want to know what games me and my friends have in common.
    e. I want the bot to share my game statistics with others on the server.
11. Riot API Integration (League of Legends)

    a. I want to display information for the game I'm currently in.
    b. I want to get information on a specific champion.
    c. I want to get information on a specific account.
    d. I want to see my recent match history.
    e. I want to see my champion mastery score.
    f. I want to see information about an in-game item.
    g. I want to see what pro players build on my champion.
    h. I want to see what jungle pathing is the best.
    i. I want to see the matchup rates of my laners to know which lanes to focus and help.
    j. I want to see my teams individual win rates to know who can carry and which players to enable.
    k. I want to see tips for laning on how to manage minions in all forms: slow pushing, hard pushing, freeze and what to do in each situation.
    l. I want to see if there are any ongoing in-game events or special modes.
12. Code Manual Documents (If time allows: Java, Python, C): As a user,
    a. I want to display code documentation.
    b. I want to be able to easily access the code documentation.
13. Clock Integration: As a user,
    a. I want to be able to run a stopwatch on the server.
    b. I want to be able to set a timer for a specific length of time on the server.
    c. I want the timer to have an alarm when it is done.
    d. I want to type a command to give a timezone in a certain city.

Non Functional Requirements

1. Architecture:
    a. I want to develop the program in python with the Discord API.
2. Performance:
    a. I want the bot to respond within a second of requesting something from it.
    b. I want the bot to be available 24 hours of the day.
3. Usability:
    a. I want the bot to be easy to download, set up, and customize for my server.
    b. I want to be able to set up the bot easily with no previous experience with discord bots.
    c. I want to never have to interact with code while running my bot.
    d. I want to be able to know which commands I can use.
    e. I want to use commands easily.
4. Security:
    a. I only want trusted individuals to be able to upload functions to the bot.

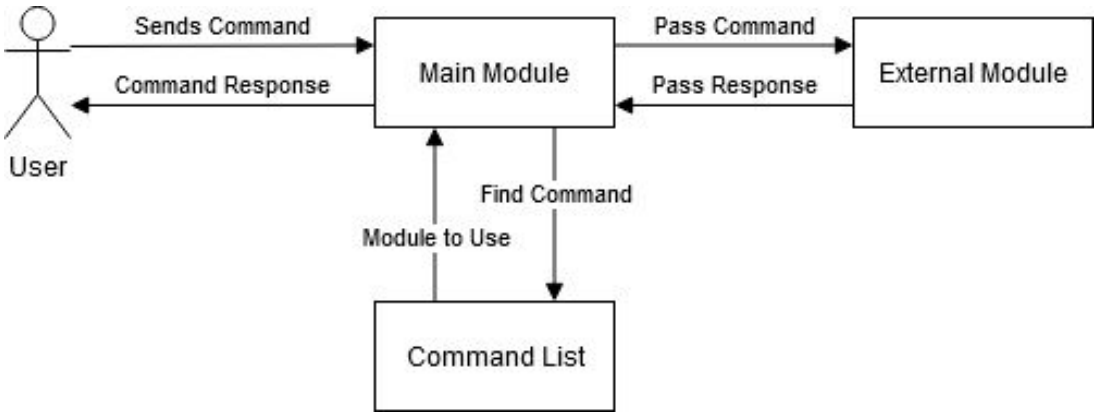
Design Outline

This project is a program that will be running within a discord server, it will be able to satisfy users' requests quickly and efficiently. It uses a Transaction-processing architectural pattern. In this case, the transactions are commands that the users are giving to the bot. The bot then has to process commands and send them to specific modules based on what the command requires. There are two main components to our system.

1. Bot
    a. The Bot will be running 24 hours a day inside a server.
    b. The Bot will react when a user is given a command.
    c. The Bot will send requests to a module when performing a function.
    d. When receiving data from modules the bot will display it to users.
2. Module
    a. The module will actually perform the functions.
    b. The module will return data to the bot to display back to the user.

High Level Overview of the system

Wild Card Bot will use a modular function system. This requires us to have a design that supports external modules where we do not know beforehand the number or contents of the module. Therefore, we plan to implement the following system which will pass commands designed for an external module to the module itself. There will be a main module that contains the infrastructure of the Discord bot and has server administration commands required for all discord servers. The main module will access the command list which will include the command name and also the module it belongs in. If the command belongs to an external module, the execution is passed to the external module which will generate a response that the main module can use to satisfy the user's request.



Activity / State Diagram

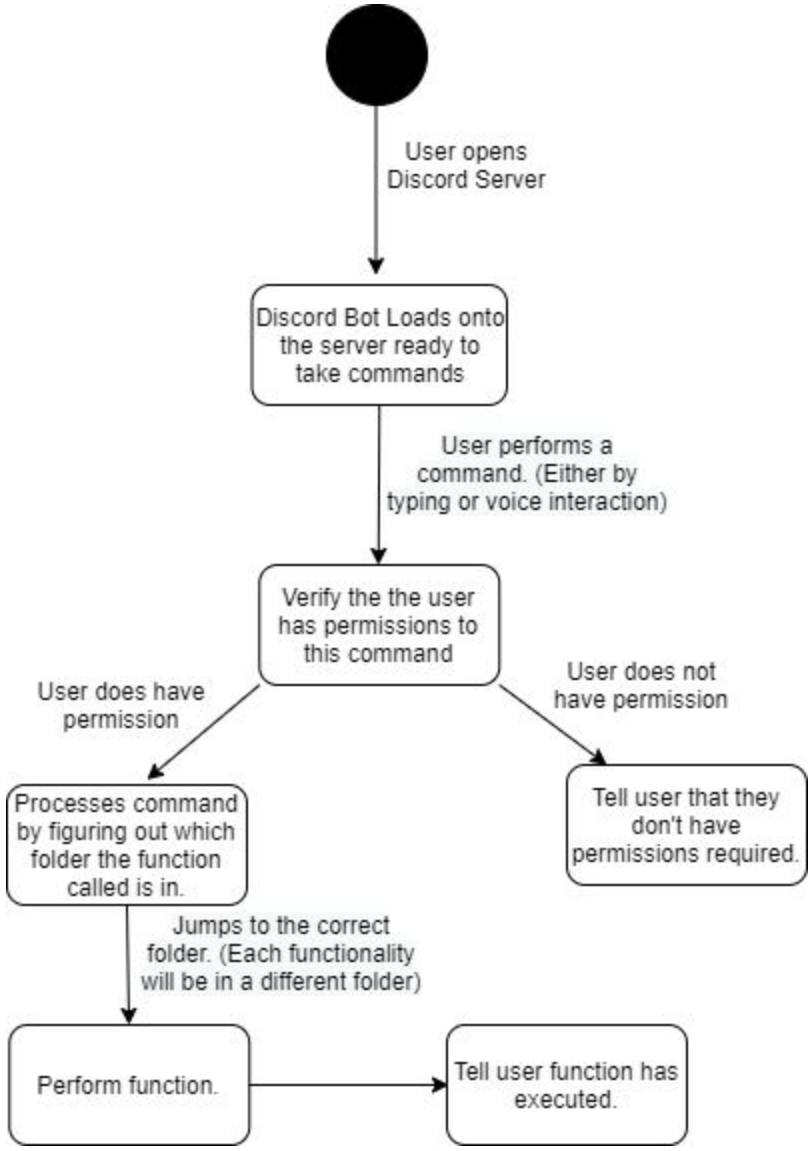
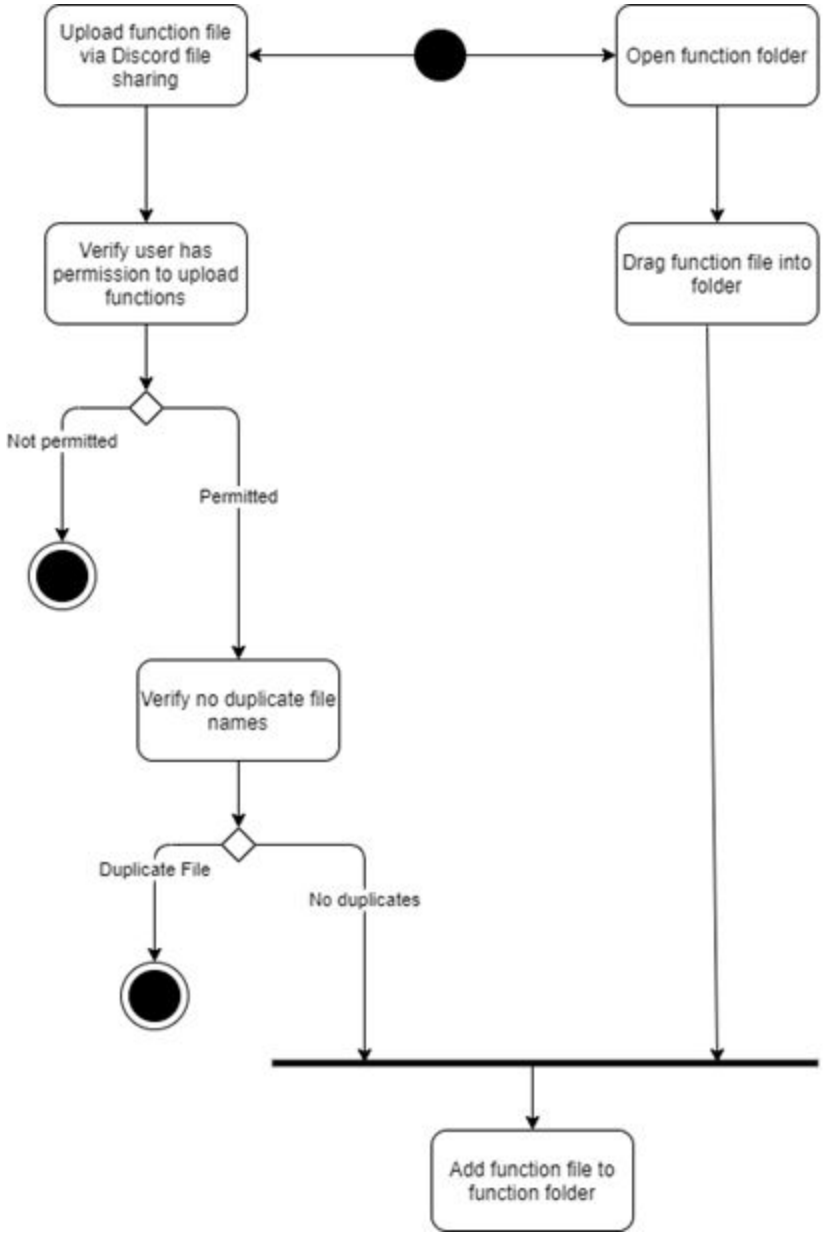
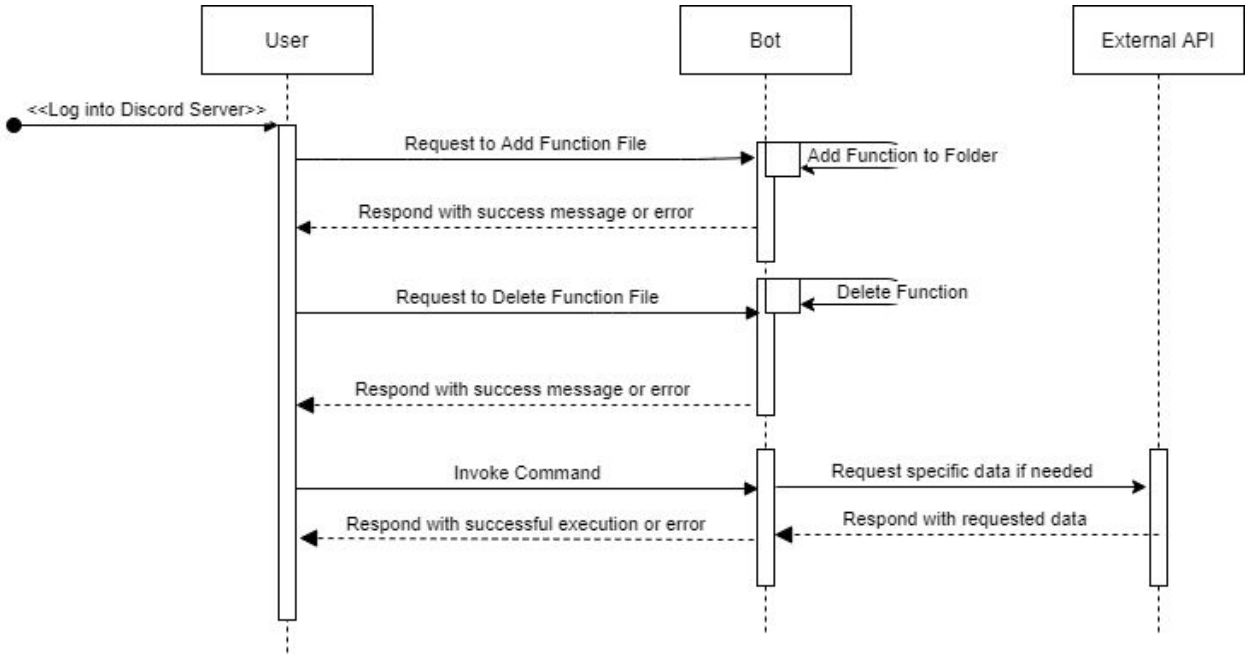


Diagram for adding functions:



Sequence of Events Overview

The sequence shown below displays the generic interactions between the user, the bot, and any external APIs. The sequence is started by a discord user invoking a command in a text channel. When a user sends a message, the bot scans the message for any commands. If a command is found, the bot checks if the user has permission to invoke said command. If the check passes, the bot will execute the command. If the command invokes an API call to an external API, then the bot will send the request after the command is called. Once the request is responded to, the bot will parse the data and present it to the user. If the user wants to add or delete a function, the bot will check the permission level and then if permitted, execute the command.



Design Issues

Functional Issues:

Functional Issue #1:

How can other Discord Developers include their modules to be utilized by Wild Card Bot?

Option 1: Create a private Discord text channel where modules are uploaded
Option 2: Drag the module into the function folder stored locally on the Server Admin's device

Justification: We would like to support both options. Upon installation, a private Discord text channel is created to host modules because it is convenient for the developer to simply upload a file to the text channel and have it be utilized by the Discord bot. Alternatively, users can also drag the module into the function folder associated with their Discord server. This function folder will be stored locally on the Server Admin's device to prevent interference from members. Both locations will be synchronized.

Functional Issue #2:

Where do we store the entire list of commands from all modules?

Option 1: Do not store and design a searching algorithm that visits every module and obtains the respective commands
Option 2: Use a text file to keep track of the list of commands and append when a new module is added
Option 3: Have a specialized List object in Python in the main module

Justification: We chose to use a specialized List object in the main module. Since Wild Card bot is expected to run 24/7, the List object will never be destroyed while the bot is running. Therefore, we will only need to reconstruct the List object when restarting the bot or when upon first use, which are rare occurrences. The List object is also extremely easy to update and account for new commands from newly appended modules.

Function Issue #3:

Who is allowed to control what modules run on the WildCard bot?

Option 1: The server administrator
Option 2: The server administrator and anyone with granted permissions/roles
Option 3: Anyone on the server

Justification: We decided to only allow the server administrator to have access to what files run on the Wild Card bot because this way it limits the probability of malicious code being run on the server by other users. In addition we hope that the server administrator will be representative of the users on their server and their wishes.

Function Issue #4:

How will users access the bot?

Option 1: The users will access the bot through voice commands and the bot will reply
Option 2: The users will type commands in the server channels to access the bot
Option 3: The users will private message the bot to access it.

Justification: Our consensus is that users typing commands or keywords in the server channels of Discord to access the bot is the most efficient way of communication. With the voice chat option issues arise in voice recognition (i.e. volume, clarity of speech especially those with accents, background noise) and this would only increase the request process and overall time and with the private messages option it would be not effective if someone wishes to share information with others on the server (i.e. roles, timers, music, game tracking statistics, trading information).

Functional Issue #5:

Who will decide what keywords (commands) will invoke a response from the bot?

Option 1: The server administrator
Option 2: Options are created and voted on, decided by majority vote from the server members on the commands
Option 3: There are global command words that work with the bot on any server

Justification: We decided to use global command/keyword for the bot instead of allowing users or the server admin to decide the words that will access the both as not only is it easier to manage on the developer side of things, it makes it easier for users who are in multiple servers that use bot as they won't have to remember custom keywords

appropriate for each server.

Functional Issue #6:

Who (which users) can use the WildCard bot?

Option 1: The server administrator
Option 2: Everyone and anyone on the server
Option 3: The server administrator and only those with permissions to use the bot.

Justification: Everyone and anyone on the server should be able to use the bot as it is general purpose and is meant for people to be able to look up and create anything from timers to playing music, viewing statistics, using external API, and making their own functions as users. If need be roles could be implemented by the server admin to stop that are misusing the WildCard bot, however hopefully it doesn't come to that.

Non-Functional Issues:

1. What language are we using and why?

Option 1: Python
Option 2: Javascript
Option 3: React

Justification: We are using python the majority of our developers are more comfortable using python and the language is easier to pick up for those of us who don't know it. It is also syntactically easy allowing for faster and more efficient development.

2. How are we getting up to date runes, builds and meta on league?

Option 1: Internet
Option 2: Professional Played matches
Option 3: RIOT API

Justification: We are using the RIOT API since it keeps up to date with all the patches of League of Legends and because it is constantly updated and is a more reliable source than the other two options. Also by using the RIOT API, this section of the WildCard bot will be constantly updated automatically instead of being dependent on players matches or the internet.

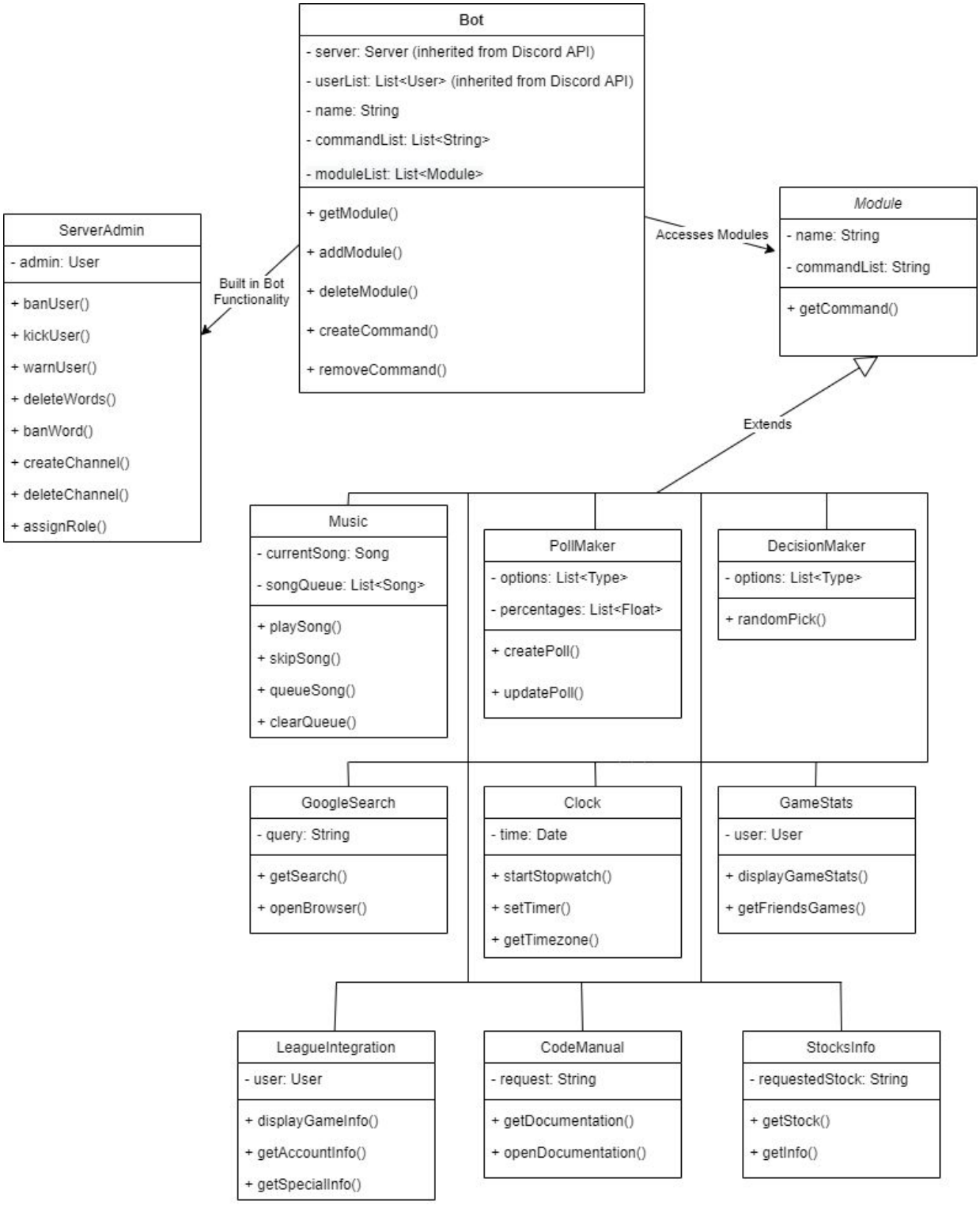
3. What are we going to use as a gamelink to allow users to interact with their friends?

Option 1: Discord status
Option 2: Steam API
Option 3: Internet

Justification: We will use the STEAM API because steam is the most popular game library in the world at the moment and the API will give us access to information from the users friend list to check their friends hours on each game, achievements in each game as well as what game they have played most recently and whether they are currently in a game.

Design Details

Class Design



Description of Classes and their Interactions

Our class design is driven by the tasks that the bot will need to complete when prompted. It has a one to many relationship in multiple different sections. At the top, there will be one bot to many modules, both one bot and one module to many commands, and one bot to many users.

- **Bot**
    - The Bot class will be the largest and most important class since it will be what the User is directly sending requests to.
    - Each Bot will have a server that it is a part of.
    - Each bot will have a user which is the current user sending requests to the bot.
    - Each bot can be named with a different name depending on what the users want to name it.
    - Each bot will have a command list of all of the possible commands available.
    - Each bot will have a module list of the modules to use when performing functions.
    - Each bot will have the ability to add or delete modules from the module list.
    - Each bot will have the ability to add or delete commands from the command list.
- **Module**
    - The Module is an interface for each of the modules that will be implemented.
    - The individual function modules will inherit from Module.
    - Each module will have a name, which will generally be the name of the function.
    - Each module will have a command list which will be an array of commands. It is important to consider that all commands will be unique among all modules.
    - Each module will be able to access the command requested. This will be used to parse the command and call the corresponding function in the module.
- **ServerAdmin**
    - ServerAdmin is the only set of functions or features that is automatically built into the bot other than the modular functionality.
    - ServerAdmin is its own class, but is used directly by the bot rather than being an editable module like the rest of the features.
    - The ServerAdmin functionality allows users with the required permissions

to quickly perform basic server tasks.
    - The ServerAdmin can warn, ban, and kick users.
    - The ServerAdmin can delete and ban words.
    - The ServerAdmin can create and delete channels.
    - The ServerAdmin can assign roles or permissions to specific users.
- **Music**
    - The Music module implements the music functionality.
    - Features a queue of songs which users can add songs into.
    - Includes functionality to control the songs in the queue such as clearing the queue or skipping over certain songs in the queue.
    - It will create a new voice channel designated for playing music if asked by the user when the user requests to play a song
- **PollMaker**
    - The PollMaker module implements the poll making functionality.
    - The basis of this module is to generate a poll for the bot to display back to the user.
    - It will be given a list of options from the user. If no such options were given, default options will be provided. These options will be stored for reference.
    - After creation of a poll, the module will continuously monitor additional votes and update the numbers internally, which will be stored in an array of floats.
    - Upon reaching a certain time limit given by the user, the poll results are displayed to the user via a notification ping.
- **DecisionMaker**
    - The DecisionMaker module implements the decision making functionality.
    - It will be given from the user either a command argument specifying the specific set of options (‘coin’ for options ‘heads’ and ‘tails’) or a number indicating the number of decisions.
    - If given a number, a random number is generated from 1 to the specified number and displayed back to the user.
    - If given a command argument, it finds the corresponding set of options, picks one of them, and displays it back to the user. All possible options all for command arguments will be stored in this module.
- **GoogleSearch**
    - The GoogleSearch module implements the internet searching capability.
    - It will be given a search phrase by the user, which will be stored for reference.
    - The module will establish a connection to the internet and conduct a Google search using the search phrase from the user.

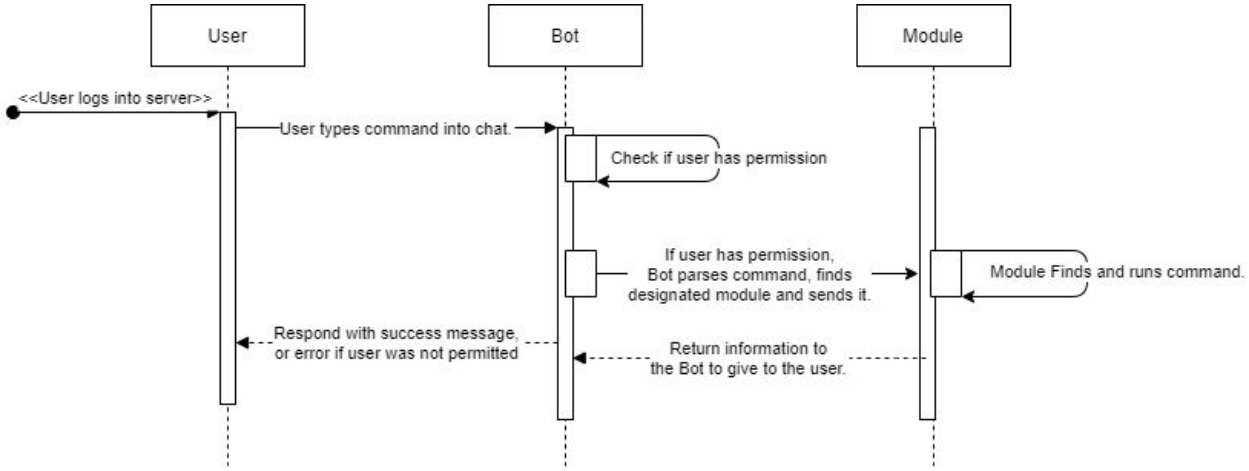
    - The results are then parsed into text and displayed back to the user.
- **Clock**
    - The Clock module implements the stopwatch and timer functionalities.
    - Upon receiving the start stopwatch command from the user, it will start a time count keeping track of how many seconds passed until the user calls stop.
    - For the timer command, the module will receive a time amount in seconds from the user. It will wait for the given amount of seconds and set an alarm off when done.
    - The user can optionally specify for the above two commands to have the time be displayed every 30 seconds or 1 minute.
    - For the timezone command, it will display to the user the time zone of the city whose name is provided by the user as a command argument.
- **GameStats**
    - The GameStats module allows users to display their in game stats.
- **LeagueIntegration**
    - The LeagueIntegration module allows users to check League of Legends statistics of specified users with the help of the Riot API.
    - The LeagueIntegration module also allows users to use the Riot API to help with their in-game decision making and statistics.
    - Users can check in-game statistics if the bot is given a specified League of Legends username and the user is in a League of Legends game. If they are not in a game the brief overall statistics of the user will be listed including match history, champion statistics, win rates, ranking, etc.
    - The LeagueIntegration module will also help the user in-game by analyzing a user's match given a username and providing them with helpful tips specifically applying to the jungle role, the bot will tell the user what path to take, give them advice for lanes, etc.
    - The module also helps notify users about in-game events.
- **CodeManual**
    - The CodeManual module allows other Discord bot developers to easily access the code documentation.
    - As the documentation will be stored in multiple files, it will open the documentation files and allow access to the contents.
    - The developer will specify a part of the documentation they want to view such as a specific module. That part is then parsed and displayed to the developer.
- **StocksInfo**
    - The StocksInfo module implements the stocks functionality using the yahoo finance API.

- Users can get basic information (Current price, open and closing prices) on a stock by simply typing “$TICKER” into the discord chat, where TICKER is replaced by the stock ticker.
- Users can get more advanced or specific stock information by specifying one of several commands along with the ticker, such as finding the 52 week lows and highs, trading volume, or more.
- Users can request a link to yahoo finance with a command for the desired ticker if they want more niche information.

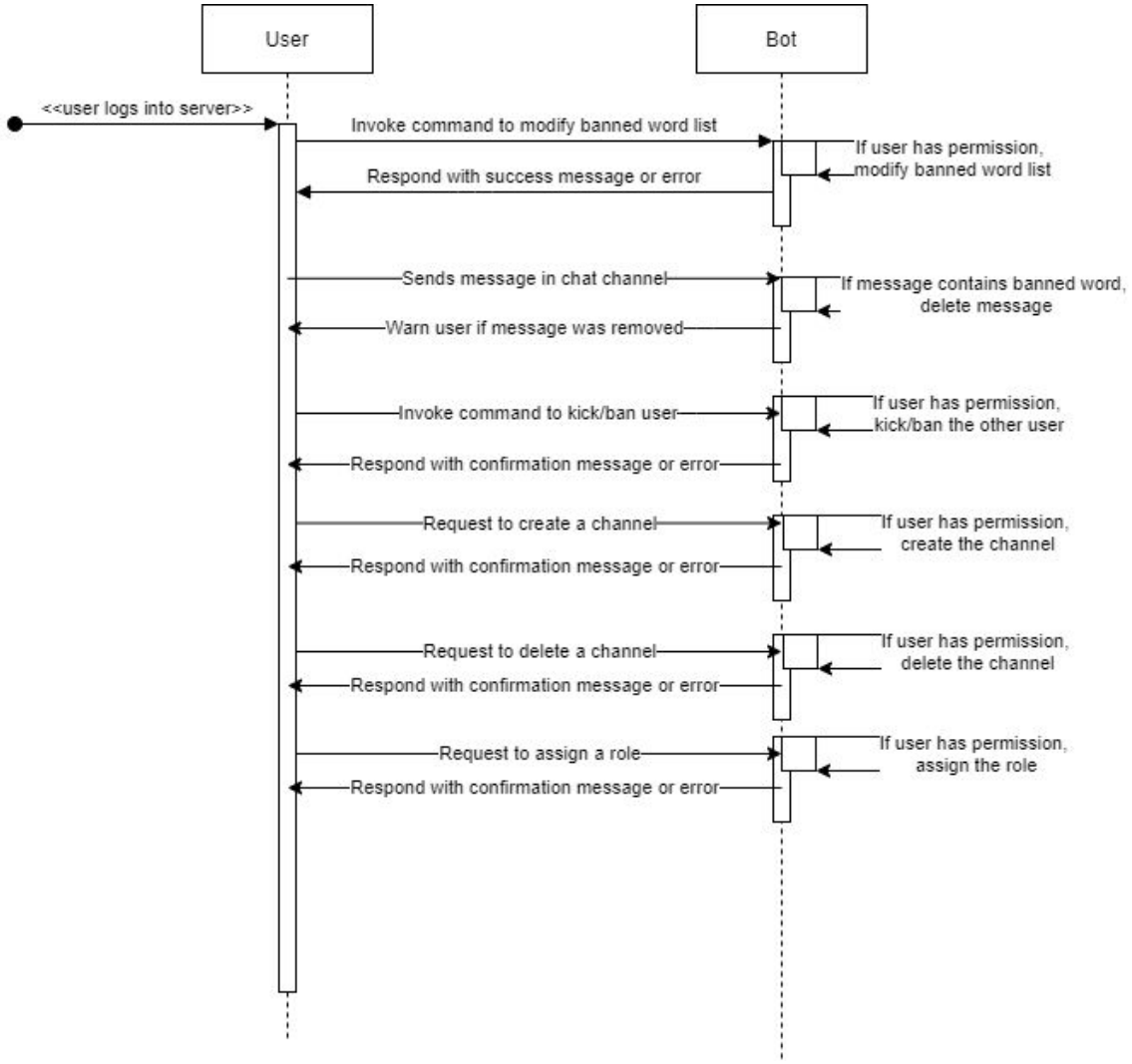
Sequence Diagrams

The following diagrams show the sequence of usage of major features of this bot. This includes adding and removing functions, calling external commands, and using the provided administration tools. This also includes the usage of the external command modules such as the Music Player, League of Legends Tool, Steam Tool, Decision Maker, Stock Tracker, and more.

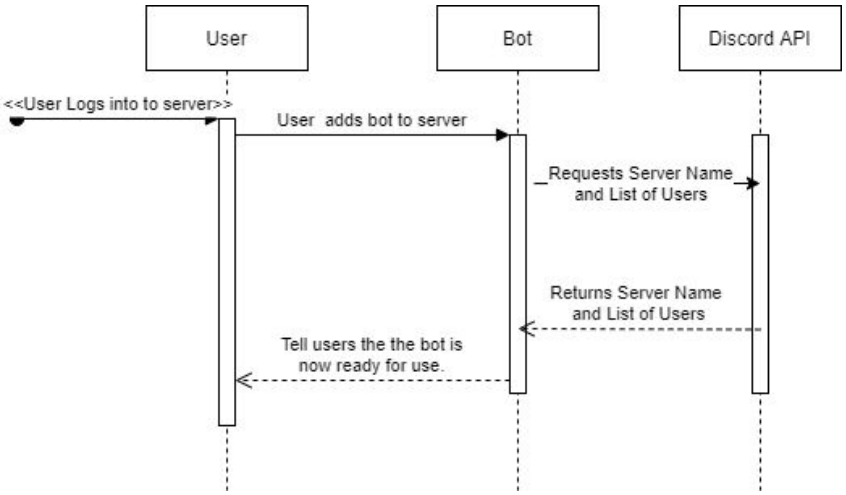
1. Sequence of events when a user uses a command invoking a module



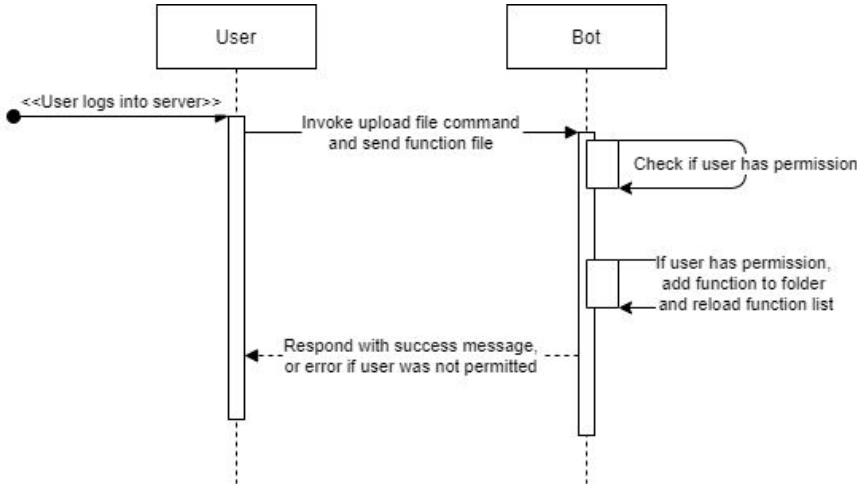
2. Sequence of events for server administration tools



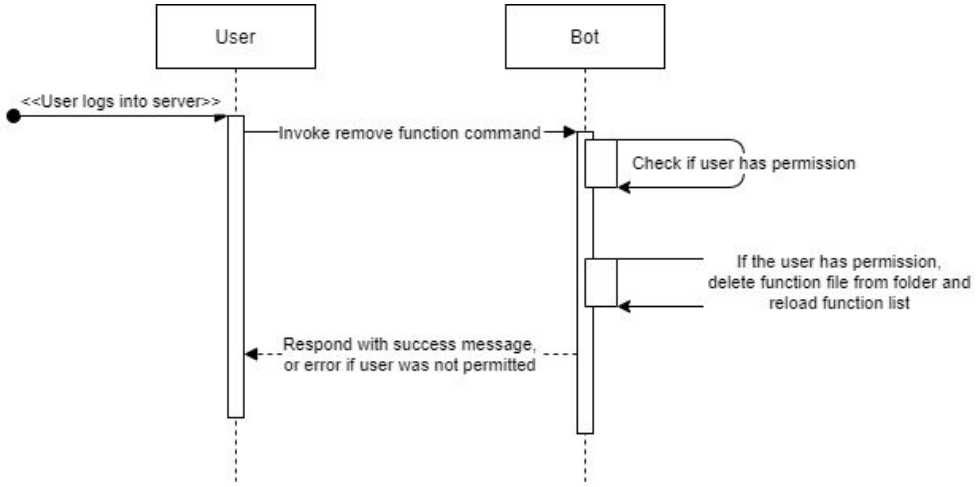
3. Sequence of events when a bot is added to a server



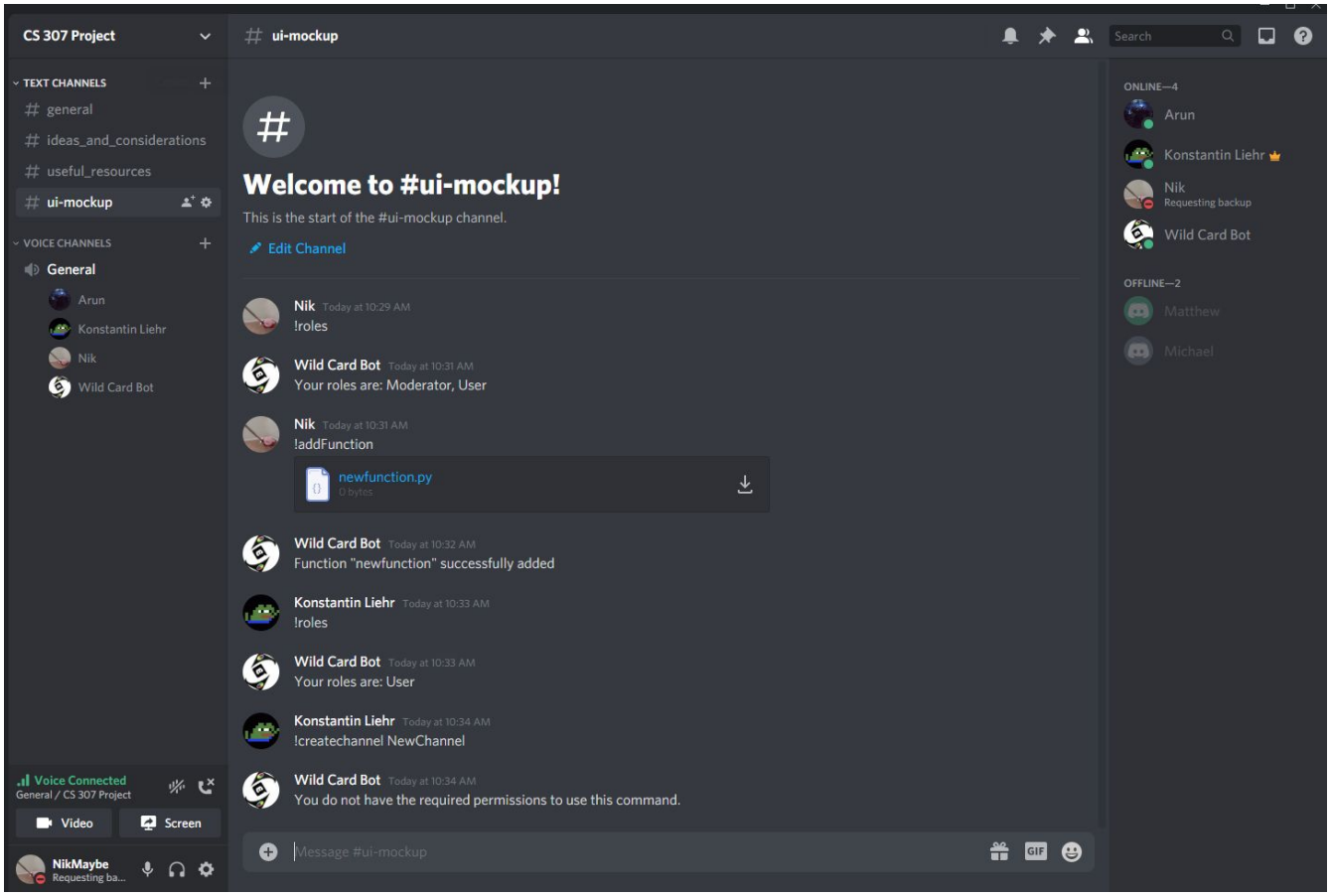
4. Sequence of events when a user adds a function



5. Sequence of events when a user removes a function



UI Mockup



The image above shows how the Wild Card Bot would present itself in a Discord server, along with showing how it would interact with users. The bot itself does not have a user interface, as users interact with the bot through discord. The bot interacts with users though the various text channels in a discord server.

As seen above, a general user does not have the same permission levels as a moderator, and as such has less commands at their disposal.

Below are examples of other commands and responses for interactions with the bot. Note that the responses include syntax that specify pinging the invoker of the command or including the name of the invoker of the command. Here are the syntax considerations:

<@name> - Pings the command caller. If the invoker is the member User, this will be replaced with @User.

<name> - This should be replaced with the member's name. If the invoker is the member User, this will be replaced with User.

Here is Wild Card bot listing the commands for server administration. Note that this is not a complete list of functionalities but provides a general overview of what the bot will display when a command is issued to list all available commands. This list is provided when the person invoking the command is the Server Administrator. If a member uses the same command, certain higher privileged commands will be removed.

Command:
!commands

Response:
Hello there <@name>! As a server admin, you have elevated privileges!

Main Commands:
!role
Lists all roles you have.

!addrole “role” “member”
Assigns “member” the given “role”. If no member is specified, the role will be added to you.

!removerole “role” “member”
Removes “role” from “member”. If no member is specified, the role will be removed from you.

!bannedwords
Lists all banned words on this server.

!addbw “banned word”
Adds the “banned word” to the list of banned words.

!removebw “banned word”
Removes the “banned word” from the list of banned words.

!reqtext “name”
Creates a text channel with the given “name”.

!reqvoice “name”
Creates a voice channel with the given “name”.

!ban “member”
Bans a “member” from the server. Automatically kicks the member from the server.

!unban “member”
Unbans a “member” from the server.

!removelinks “text channel”
Removes all messages with links in the given “text channel”.

Other modules:
Music, Decision Maker, Poll Maker, Clock, League of Legends

To view commands from other modules, type **!commands “module”**.

Here is what the bot would display to the user if they specified a specific external module. Instead of listing the server administration commands like before, it lists all commands specific to that module. We will use the Music module for this particular example.

Command:
!commands Music

Response:
Hello there <@name>!

Note: The Music module uses YouTube links to parse songs. Please provide a YouTube link for the song you want rather than its name. A song queue is also provided for your convenience.

Music Commands:
!joinv “voice channel”
Instructs Wild Card bot to join the “voice channel”. If no “voice channel” is specified, a private voice channel will be created temporarily for you to listen to music.

!playsong “YouTube link”
Plays the song specified by “YouTube link”. If no YouTube link is provided, the bot will play the songs in the song queue.

!viewqueue
*Displays the songs currently in the song queue.**

!addsong “YouTube link”
*Adds the song specified by “YouTube link” to the song queue.**

!skipsong
Skips the current song playing.

!clearqueue
Clears the song queue. Note that this does not stop the current song from playing.

The following are some example commands being used by the Server Administrator. It displays how the Server Administrator is able to alter the roles of members in the Discord server and add words to the banned list.

Command:
!role

Response:
Hello there <@name>! Here are your current roles:

admin
developer

Command:
!addrole developer <name>

Response:
Success! <name> has been successfully assigned the role of **developer**.

<@name>, congratulations on obtaining the role of **developer**!

Command:
!bannedwords

Response:
Here is the current list of banned words:

supercalifragilisticexpialidocious

Command:
!addbw hell

Response:
The word “hell” has been added to the list of banned words.

Refrain from using “hell” @here. It is now a banned word.

Command:
!bannedwords

Response:
Here is the current list of banned words:

supercalifragilisticexpialidocious
hell

The following are examples of what happens when a normal member of a discord server attempts to utilize commands outside of his or her privileges. This is necessary as only the Server Administrator should be able to manipulate roles or ban members.

Command:
!removerole developer <name>

Response:
Permission denied! Only the server admin can use this command!

To list all commands you can use, type **!commands**.

Command:
!ban <name>

Response:
Permission denied! Only the server admin can use this command!

To list all commands you can use, type **!commands**.

Here are examples of commands being used by a server member for an external module. We will use the Music module whose commands were listed earlier.

Command:
!joinv General

Response:
Hello there <@name>! I have just joined the **General** voice channel!

Feel free to ask me to play songs, add songs to the song queue, or skip songs for you. For a full list of commands for the Music module, type **!commands Music**.

Enjoy listening to your favorite tunes!

Command:
!addsong https://www.youtube.com/watch?v=60ItHLz5WEA

Response:
Success! **Alan Walker - Faded** has been added to the song queue.

It is currently in position **1**.

Command:
!playsong

Response:
Playing from the song queue. There are currently **1** song(s) in the queue.
Now playing:

Alan Walker - Faded *3:32*

Command:
!skipsong

Response:
Successfully skipped the song **Alan Walker - Faded**. There are currently no songs in the queue.

Response:

No members were found in the **General** voice channel. I have left the channel!

If you would like me to join or rejoin a voice channel, use **!joinv “voice channel”**.