Team: Dewbed

Members: William, Bettina, Emily, and Devika

Project: 2C - Final Architecture

Final Architecture:

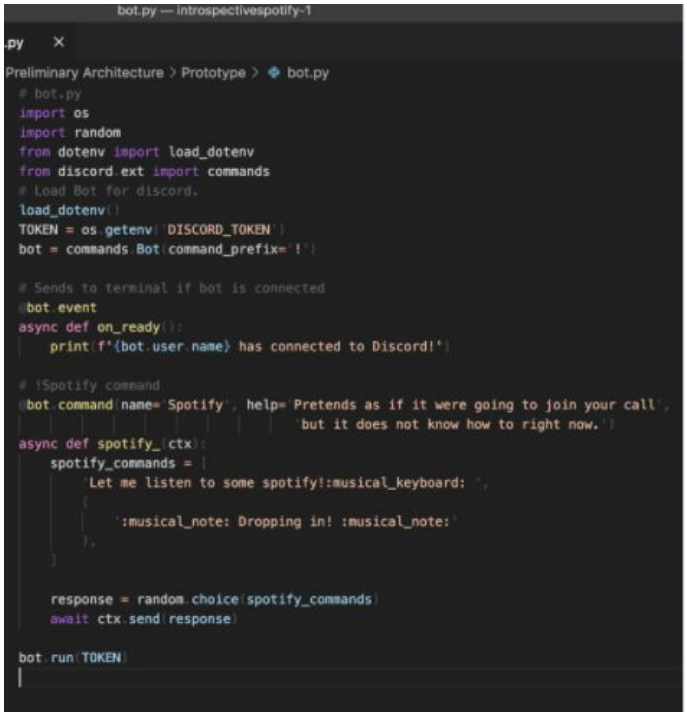
https://docs.google.com/document/d/1wc58_iXdFFnJTfAqEHvy1596dyz1_m5cbDPMeigivDE/edit?usp=sharing

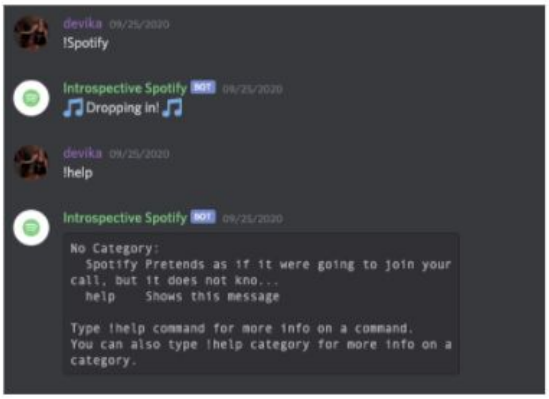
Primary Author: William

Document Table of Contents

I. How introspective Spotify works:

Text in discord chat to interact with the bot. We conducted a prototype which interacts with dummy messages such as !Spotify to get the bot to respond. Using Python, the bot has specific bot commands that are default triggered with a prefix, “!” , and a command, such as “!Spotify”, but a user can change the prefix to ensure that there is no confusion with other Discord bots. and a !help guide. Similarly, the bot will also send a message to any new users and can react to specific messages (currently edited out due to spam). Music plays on your own Spotify account. Currently, it is possible to play music on Spotify through Discord through listening parties in a channel, and listening along on individual Spotify. To prevent spamming, there will be a minute cooldown on specific actions. If a specific user is trolling the bot, as in, reacting or spamming commands, a Server Admin may call a command to assign a “muted” role to the troll user which disables them from using our bot.





Prototype 1.0

II. Structure

II.A Program Organization

Introspective Spotify is a music sharing application that uses Spotify's API and Discord as an agent of communication. Discord is a free Voice Over Internet Protocol messaging application and will act as our agent for a bot that interacts with Spotify, a free digitized music streaming service. To interact with Introspective Spotify, a user must invite the bot to their server with friends. Once invited, the bot will act as a member in the server and only interact with users through commands. All commands are prefixed with an ! (such as ![command]), due to Discord's requirements for Bot commands. Introspective Spotify will have four main functionalities for our initial release. The largest package of Introspective Spotify will be synchronous listening through Spotify Connect Web API. Users can listen synchronously to a specific playlist and interact with our specific Discord commands: queue, pause, play, shuffle, create a playlist, and rewind. Also, users will be able to call our Music Theory commands to retrieve a brief breakdown of different music theory functionalities from Spotify's Music Analysis API. Similarly, our Music History component provides an user specific analysis over a designated span of time as a Bot reply in the specific channel. Finally, our "Spotify Wrapped" component combines both our Music History and Music Theory to allow users in the same Discord server to either do a server wide analysis of overall music history or a partner analysis. The response will be a bot message @ing either everyone or the two accounts being analyzed. All in all, Introspective Spotify will be an interactive, friendly bot that will help bring music theory, history, and listening to Discord through simple commands. To ensure anti-spam and no trolling, server admins can call a !muteis @user to assign a "muted" role which disables a user from using any commands.

II.B Major Classes

- **Input Class** - This class will read in user inputs, process them, and then call the correct method to respond to the input.
    - Uses the Discord API to read user inputs.
    - Will reply in Discord chat with errors if users type bad inputs.
- **SpotifyAuthorization Class** - This class will contain methods to facilitate the log-in process (See Figure 1 in Getting Start Commands). This class will also contain user data that is needed to alter, such as a user_spotify_ID variable.
    - Interacts with Spotify Web API and OAuth2.0 API to receive tokens and user data
    - Adds tokens and user data to User Login database.
- **SpotifyListen Class**
    - Has access to tokens stored in the User Login database.
    - Uses these tokens to interact with Spotify Web API.
    - Add songs from the current listening party to the Song database.
- **MusicTheory Class**
    - Has access to tokens stored in the User Login database.
    - Uses these tokens to interact with Spotify Web API.
    - Uses the Discord API to reply in chat.

- **MusicHistory Class**
    - Has access to tokens in the User Login database.
    - Uses tokens to interact with MusicTheory Class and Spotify Web API.
    - Uses the Discord API to reply and react.
- **SpotifyWrapped Class**
    - Interacts with the MusicHistory Class. (Does not need to interact with Spotify Web API directly).
    - Use the Discord API to reply in chat.

II.C Data Design

We plan on having 2 databases. We decided to use 2 instead of 1 because the databases serve very different purposes and have different access structures.

1. **User login database**
    a. This database will be a random access data structure which holds user information such as their Spotify/Discord usernames and access tokens. The information stored in this database is very confidential, which is why we chose to separate it from our other database.
2. **Song database**
    a. The song database will be a sequential access structure where we will store information from the current listening parties occurring in the Discord servers. Our program will keep track of the queue of songs made by users and can create playlists out of the saved songs. The information stored in this database will be deleted after the listening party ends.

II.D Interfaces

- **External Interface:** Discord chat to interact with users (chat messages / message reaction)
- **Internal Interface:** See UML diagram below.

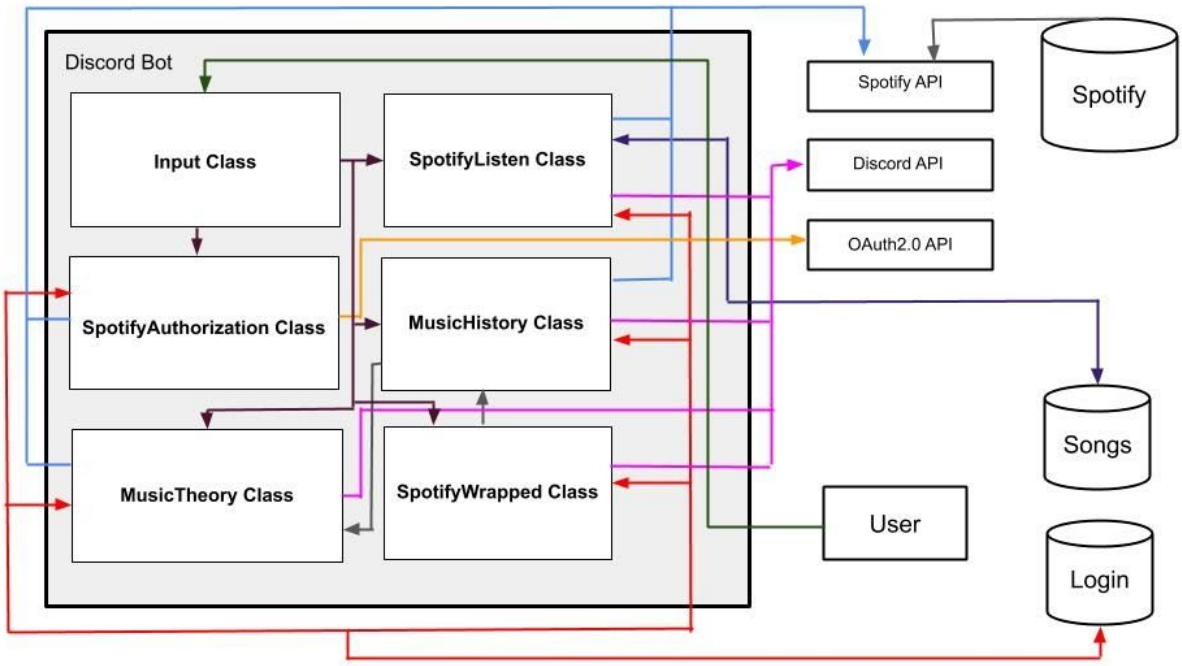


Figure 1: UML Component Diagram

II.E Performance/ Error Processing

- As our Bot acts as a member of a server, it should be up and running and available whenever a user wants to interact with it. At most, some of the computations may take time to compute, but should take no more than a minute. In the case of a timeout, the bot should quit the process and return a message to the requesting user telling them that there was an error and to try again in a few minutes.
- **Error Handling:** Using a try-except block for all bot commands in bot.py, if the try fails, the bot will reply with an error handling message based on the Discord.errors class in the Discord API. Additionally, if invalid input is given to the Spotify API, the API returns an error message to our bot.

II.F Architecture Approach

To start our project, we first used the top-down approach. Because we had a strong understanding of our requirements, the top-down approach was able to give us a strong high level understanding of our modules. Once we got stuck on implementation decisions, we moved to a bottom-up approach. In this process, we researched how the Spotify API reads inputs and how we could give this input to our Discord Bot, By using a mixture of top-down and bottom-up, we were able to draw inspiration from our requirements without being confined to them.

III. Commands:

III.A Getting Started Commands:

- **Overview:** Type these commands in the Discord chat to get started! These commands also aim to increase usability. Only users who are not assigned under the muted role can do these.
- **About the Commands:**
    - **!help:** Sends the discord channel a list of all of the discord bot's commands.
        - If users type **!help [command]**, will provide a detailed description of the specific command
    - **!login:** Sends the user a direct message asking them to log into their Spotify account via Introspective Spotify and to give permission for Introspective Spotify to modify their data. This will be accomplished by interacting with Spotify Web API and OAuth2.0 via HTTP requests to provide the user with an external login screen and permissions acceptance page.
    - **!logout:** Allows the user to log out of Introspective Spotify and revoke permissions to the user's Spotify account.

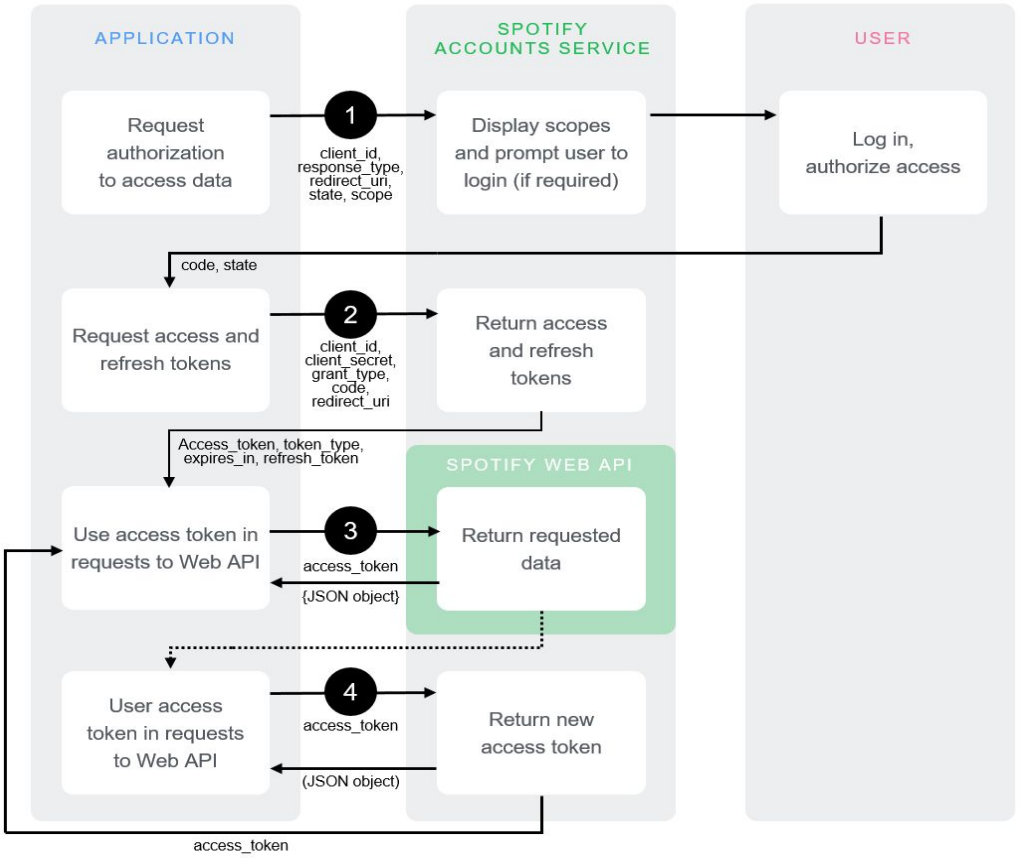


Figure 2: Spotify Authorization Process **Source**: https://developer.spotify.com/documentation/general/guides/authorization-guide/

III.B Music Theory Commands:

- **Overview:** Music Theory Commands provides the user music theory analysis about a song. There will be a 1 minute cooldown between commands to prevent spamming.Only users who are not assigned under the muted role can do these.
- **About the commands:**
    - **How it works:** All music theory commands use the Spotify API to retrieve song IDs (accessed via standard HTTPS requests in UTF-8 format to an API endpoint). Use the song ID and access token (access tokens are granted after user logs-in. See diagram above) to request from "Audio Features" documentation. Introspective Spotify bot will respond in the server chat with the requested music theory information.
        - **[song]:** this parameter has type song and refers to the search query keyword used to search Spotify's API. If no song is provided, this parameter will find the current song.
    - **!musictheory [song]**: Uses "Audio Features" API to get the estimated key, temp, time signature, mode, moode, danceability, acousticness, energy, and instrumentalness of the track.
    - **!key [song]:** Use "Audio Features" API to get the estimated key of the track.
    - **!tempo [song]:** Use "Audio Features" API to get the estimated tempo (beats per minute) of the track.
    - **!timesignature [song]:** Use "Audio Features" API to get the estimated time signature of the track.
    - **!mode [song]:** Use "Audio Features" API to get the estimated mode (major or minor) of the track.
    - **!mood [song]:** Use "Audio Features" API to get the estimated mood (happy, sad, etc.) of the track.
    - **!danceability [song]:** Use "Audio Features" API to get the estimated danceability (high, medium, low) of the track.
    - **!acousticness [song]:** Use "Audio Features" API to get the estimated acousticness (high, medium, low) of the track.
    - **!energy [song]:** Use "Audio Features" API to get the estimated mode energy (high, medium low) of the track
    - **!instrumentalness [song]:** Use "Audio Features" API to get the estimated instrumental (high, medium, low) such as "Oohs" and "Aahs" in a track.
    - **!musictheoryhelp:** This command gives a description of key, tempo, time signature, mode, mood, danceability, acousticness, energy, and instrumentalness for users that do not have familiarity with these terms. We will type these descriptions ourselves and then send a private message to the

requested user explaining what each theory term means. We will ensure it is simple and easy to read while using emojis as well to make it look friendly.

III.C Music History Commands:

- **Overview:** Music History Commands provides music theory analysis over a span of time that they listened to music on Spotify. The response will be a bot message with the requested information. There will be a 1 minute cooldown between commands to prevent spamming. Only users who are not assigned under the muted role can do these.
- **About the commands:**
    - **!genre [timeframe]**: tells users their most listened to genre of music during a timeframe. To obtain the genre, this command will get the top 10 artists using Spotify Web API's audio feature which returns an object that contains the associated genre of an artist. The command will then analyze the genres given and tell the user what genre is the most common genre in the user's top 10 artists.
    - **!topsongs [time_range] [limit]**: tells user what their most listened to songs are.
        - **Parameter Descriptions:**
            - **[time_range]:** this parameter indicates the time range of the user's history to retrieve data. long_term (several years of data and including all new data as it becomes available), medium_term (last 6 months), short_term (last 4 weeks). Default: medium_term. Taken from Get a User's Top Artists and Tracks.
            - **[limit]:** this parameter refers to the number of top tracks the user wants to get with a minimum of 1 and a maximum of 50. If omitted, will default to the top 5 tracks/artists.
        - **Reaction:** After giving the users their top tracks, the Bot can ask the users if they want a more detailed analysis of their top tracks. Users will react to the bot's analyze message if they want to learn more. Discord Bot API for interpreting reactions. The Spotify id tracks will be taken and be analysed using Spotify Web API's get audio feature for several tracks. The tracks will be analysed individually and given as an object whose key is audio_features and whose value is an array of audio features objects pertaining to the tracks in JSON objects. Based on the array of features given, the analyse command will take the most prevalent key, tempo, time signature and mode of all tracks.
    - **!topartists [time_range] [limit]:** tells user what their most listened to artists are.
        - **Parameter Descriptions:**
            - **[time_range]:** same as !TopSongs's time_range parameter
            - **[limit]:** same as !TopSongs's limit parameter but for artists instead of tracks
- **Music History Commands VS Music Theory Commands:**
    - The Music History Analysis is similar to Music Theory Analysis. The difference between these two commands is the music theory command analyses a certain

song and music history command analyses multiple tracks. There is no !TopArtist reaction analysis because Spotify does not give much artist information besides the genre, which we already have the command for.

III.D Synchronous Listening Commands:

- **Overview:** These commands are entered in the Discord chat to allow users in the same Discord server to sync their Spotify accounts. Music will be played on each user's Premium Spotify account. Using Discord's API for reactions, the bot will react to the user's messages to ensure that it has processed the command. Only users who are not assigned under the muted role can do these.
- **About the commands:**
    - **!join:** If there is no current listening party, starts a listening party session by connecting to Spotify Connect Web API and adds the user to the listening party. If a listening party is already in session, this command simply adds the user listening party. Once users are in the listening party, they will be able to use any of the following commands.
    - **!leave:** Removes the user from the listening party. If the user is the last person in the listening party, we close the listening party by disconnecting from the Connect Web API service.
    - **!play [song name]:** searches for the specified song in Spotify and adds it to the Spotify queue of all users currently in the listening party using the Spotify queue endpoint. Will store the current listen party's played songs in local memory until the listen party is over.
    - **!queue [song name]:** Displays all songs added in the listening party's queue.
    - **!pause:** Pauses the playback for all users in the listening party.
    - **!skip:** Skips current song.
    - **!rewind:** Plays previous song.
    - **!shuffle:** Shuffles the songs on users' Spotify queues. We can not use the Spotify API shuffle endpoint as we have to ensure the same order of songs for all users in the listening party. We will shuffle the locally stored queue and then call the remove/add API endpoints to remove the old list of songs and add the newly ordered list of songs to all of the users' queues.
    - **!createplaylist:** Creates a playlist of all the played and queued songs for the current listen party. The playlist will be created on the user's Spotify account and a link to the playlist will be provided by Introspective Spotify. All of this is done through Introspective Spotify interacting with Spotify Playlist API.

III.E Spotify Wrapped (Music History) with Friends Commands:

- **Overview**: These commands will be entered in the Discord chat to allow users in the same Discord server to either do a server wide analysis of overall music history or a partner analysis. The response will be a bot message @ing either everyone or the two accounts being analyzed. There will be a 1 minute cooldown between commands to prevent spamming. Only users who are not assigned under the muted role can do these.
- **About the commands:**
    - **!serveranalyze [timeframe]** - Creates a Spotify Wrapped (runs music history commands) for the whole server over a certain time period. Will provide average analytics for BPM, genre, and key, and reply with a message containing the information. The specific **[timeframe]** would only be “short, medium, and long” due to Spotify’s limitations. Using Spotify’s API specifically made for top tracks and “Audio Features” documentation, we can easily pool user specific audio features and top tracks and average the numbers out using our own algorithm (the most listened to genre being the average, using averaging to find BPM, and most listened to tracks becoming average) to display back in a simple Bot reply message formatted so it is easy for all users to glance over.
    - **!friendanalyze [username] [timeframe]:** Creates a personalized Spotify Wrapped (runs music history commands) between requesting user and another **[username**] in the server between a **[timeframe]**. Will provide average analytics for BPM, genre, and key between the two users and will reply with a message containing the information. The specific timeframe would only be “short, medium, and long” due to Spotify’s limitations. Using Spotify’s API specifically made for top tracks and “Audio Features” documentation, we can easily pool user specific audio features and top tracks and average the numbers out using our own algorithm (the most listened to genre being the average, using averaging to find BPM, and most listened to tracks becoming average) to display back in a simple Bot reply message formatted so it is easy for all users to glance over.

~~Must Fix~~

~~● Defect: Prefix~~

~~○ The bot needs to allow people to change the prefix for bot commands.~~

~~○ ! is common for many bots, and trying to call one bot will call multiple if they~~

~~have overlapping commands. (eg. !help)~~

● Defect: Remove a song from the queue

○ There is currently no command that allows users to remove a song from the

queue.

● Defect: Trolling

~~○ The bot needs a way to prevent users from trolling, or messing with, the other listeners. (ie. by continuously pausing and playing a track, skipping forward through the whole queue without listening to everyone's songs, etc.)~~

~~○ Having a time constraint on how many commands one can make in a certain amount of time was suggested.~~

○ Having a voting system for skipping songs could be implemented using emoji reactions for release 2.0.

● Issue: Security/ Privacy

○ The bot has to handle confidential information correctly.

○ We do not want users to access other people's song history without their confirmation, because some users may want their privacy.

○ Emoji reactions were suggested as a means of accepting a potential query to use the history comparison feature with friends.

○ Hierarchy

■ Creating one class to listen for commands in the Discord server which can access the database of login information and call the other classes on a need-based basis was suggested.

~~■ This would also help with complexity, making it so the bot doesn't have five classes all listening unnecessarily to the text channels despite not all being needed at once.~~

~~■ This one listening component was also suggested to help with the "trolling" problem, adding the command cooldown functionality.~~

● Defect: No DBMS selected

○ There is a lot of information that the developers want to store, but they have not selected their DBMS.

○ The DBMS needs to communicate with their Discord bot and possibly the Spotify API and needs to be selected carefully.

Should Fix

● Defect: Different queues if using the Spotify app to add songs

○ If a user decides to queue a song through the Spotify app using their personal account, the user will be out of sync with the listening group.

○ The bot should be able to access the user queue from Spotify and sync with everyone else's Spotify queues by updating the listening party queue.

~~● Issue: Capitalization of commands~~

~~○ The bot commands currently all start with a capital letter. (eg. !Spotify)~~

~~○ Users might not want to capitalize their letters, so the bot should allow users to use lower case for their commands. (eg. !spotify)~~

Comments

~~● Issue: The architecture approach is not outlined~~

~~○ In the document, there should be a section where they outline the approach they will be using.~~

~~○ The document should also provide justification for their specific approach, which they mentioned was a mixture of top-down and bottom-up.~~

● Issue: Conflicting Commands

○ The developers are not expecting conflicting commands, except users trying to

use listening party commands where there is no listening party. In this situation,

the bot will return an error into the text channel.

● Question: Searching songs with the same names but different artists

○ How will the bot deal with vague search queries?

○ A suggestion would be to allow users to add Spotify song links, to ensure their

specific song will be played in the queue instead of a song with the same name.

These song links can be found by right-clicking on the song, and navigating to

“Share”; there’s a “Copy Song Link” button.

Based on the architecture report and the issues and defects raised during the meeting, we decided to approve the project with required changes. The main points in the must fix section are issues that deal with usability and security regarding user’s paid subscriptions. The other main issue is the lack of DBMS in their architecture. During the meeting, the group had plans on how to address all of these key issues, and from their responses they seemed reasonably easy to fix. For this reason, we believe that once these necessary changes are made, this project will be ready to be approved.