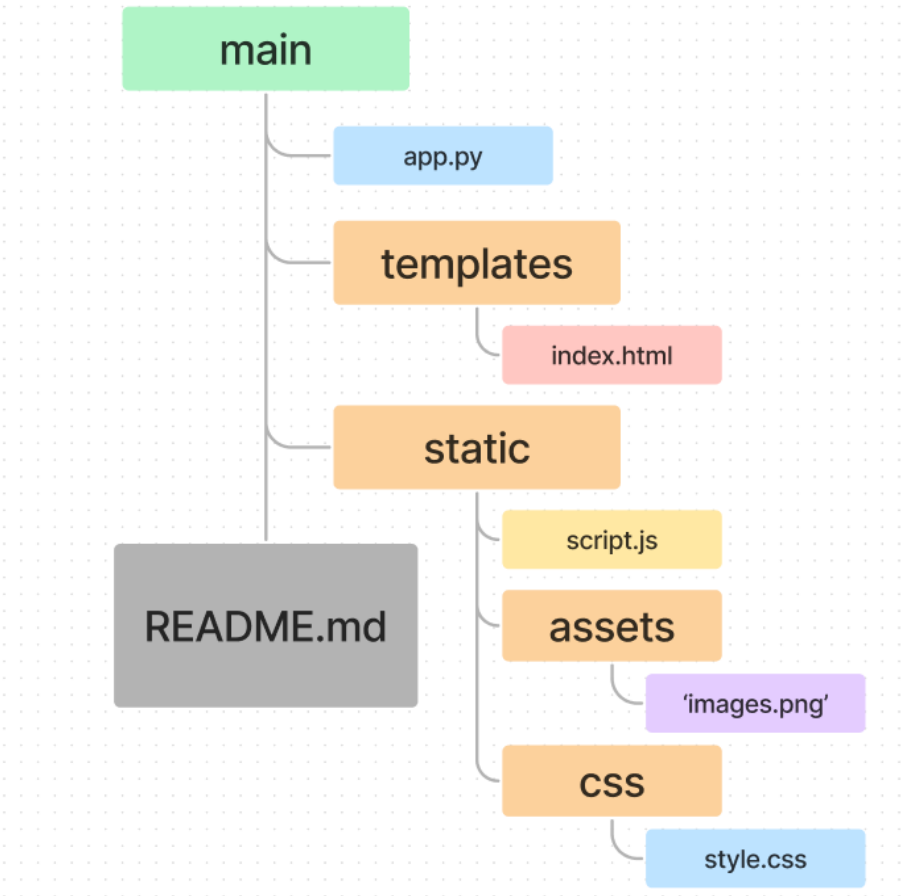
Introduction

The purpose of this file is to document several areas of the development process behind Notes4You, a personal software side project designed to help to strengthen my skills in building, from scratch, an application based on an idea I once had.

Description of Project

At its base, this project is a tool that allows users to connect their Spotify libraries to my Flask application. The retrieved song data is then used to dynamically query YouTube videos, to connect users to the top-rated MIDI piano tutorials for their songs.

Folder Structure



Quick-Access Bookmarks:

Description of Tree

Settled in the main branch, *app.py* is the Flask micro-web-framework that manages the core logic of implementing the Spotify API, along with handling OAuth and project endpoints, in accordance with the Spotify Developer Tools. The frontend visuals are provided by vanilla *index.html* and *style.css,* placed in Flask readable folders named *templates* and *static.* Moreover, *script.js* is used to handle query requests using the YouTube Data API v3.

Technical Details

The following section will cover another layer of the project, exploring the two most important components of the functionality: **app.py, index.html**, and **script.js**.

Flask Application

I decided to use Flask as the web framework due to its rapid development speed. Within a few minutes of starting, I was able to form crucial endpoints to navigate the different pages of my app.

Endpoints:

- Provided through @app.route('/<identifiers>) function, an endpoint serves as a location for pages of your application to live
- **Notes4You**:
    - "/": Acts as a rough landing page for users to immediately begin the OAuth procedure with SpotifyAPI.
    - "/redirect": Acts as a return page, using Spotify Development Tools redirect_uri to encompass several combinations of localhost and 127.0.0.1 addresses
    - "/getTrack": Interacts with SpotifyAPI for track retrieval, music playback using track_uri. Also interacts with the HTML frontend, listening to POST requests to control the in-house music control panel.

Although these variables and terminologies may seem foreign, they will be shortly brought in full context by the other sections in **Technical Details**.

HTML Frontend

The main visual components in Notes4You are encompassed by a class="player", which is composed of sections for text content, interactive sliders, and control buttons. Further enhancing the functionality of this file, is the dynamic templating, achieved through Jinja2.

Dynamic Templating:

- Recall the **Folder Structure** contains a templates directory. Flask uses the Jinja2 templating engine to allow modification of HTML pages located in templates.

Jinja2:

- Allows for dynamically rendered templates, through the use of the following syntax.
- E.g. I can loop through instances of relevant details from an array
  Python: track_name = "Happy", artist _name = "Pharell Williams"
  HTML: {{ track_name }} and {{ artist_name }}
- With each instance, the HTML file is written with appropriate details

JavaScript Video Retrieval

The objective of this component is to interface with the HTML frontend and YouTube API. It features concepts like DOM management, event handler based on user input, API requests, and template literals

DOM Management through 'DOMContentLoaded' EventListener:

- Checks to ensure entire HTML page is loaded before executing retrieval functions

Event Handler:

- Uses a reference to HTML called 'search,' listening to a button users may click if they are not satisfied with the auto-queried video.
- Leads user to "https://www.youtube.com/search?q=" + trackName + " " + artistName + " midipianotutorial" through loadYouTubeVideo() function.

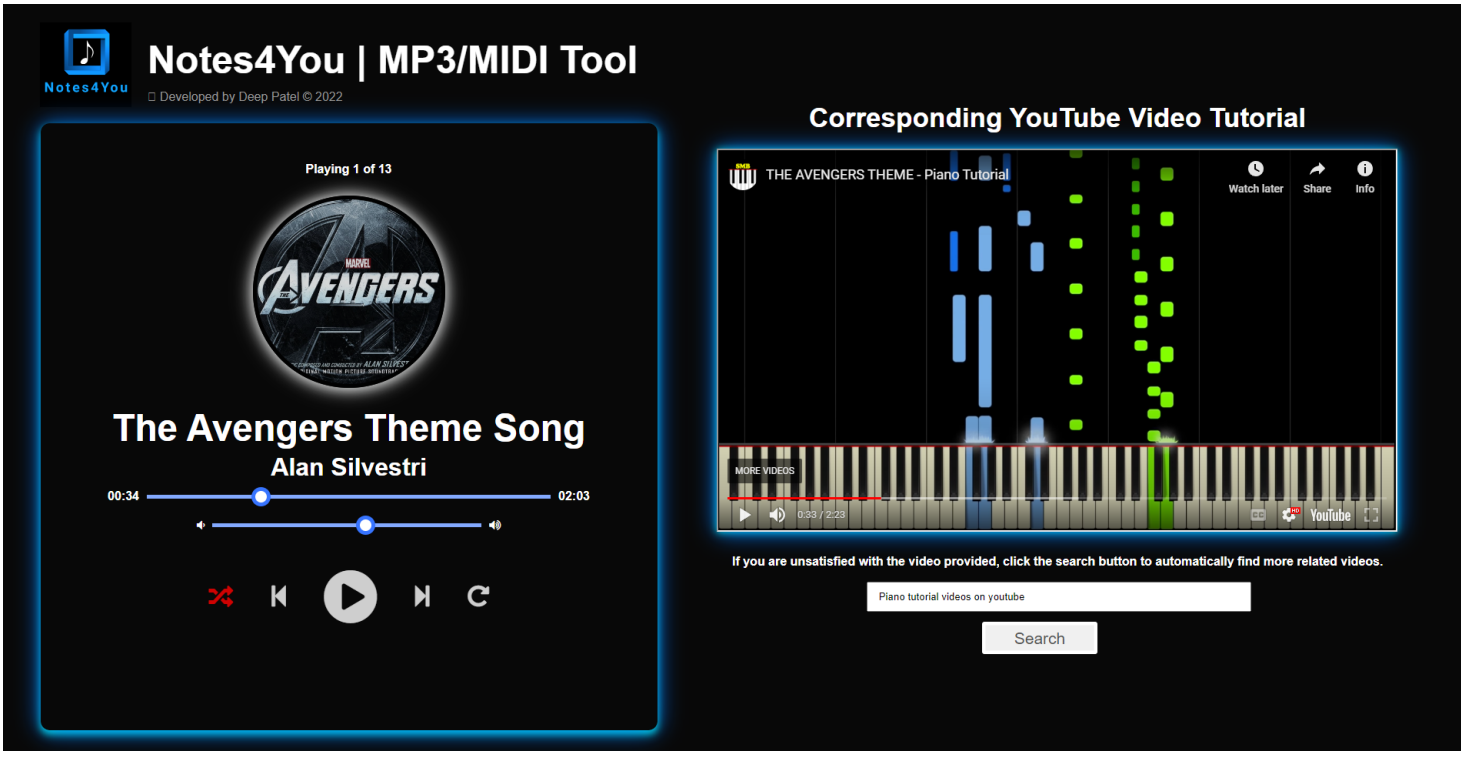
API Request:

- The searchYouTubeVideo function has arguments with current Jinja2 loaded textContent values.
- API URL Breakdown:
  - Base URL: "https://www.googleapis.com/youtube/v3/search"
  - API Key: Authentication and authorizations access for API, delivered through Google Cloud Platform's YouTube Data API v3
  - Type: Self-explanatory, needs to search for 'video' mp4 content
  - Search Query: `q=${query}` built using JavaScript template literals

Template Literals:

- Enclosed by backticks (` {placeholder} `) for dynamic features, including the embedding of variable strings into urls.

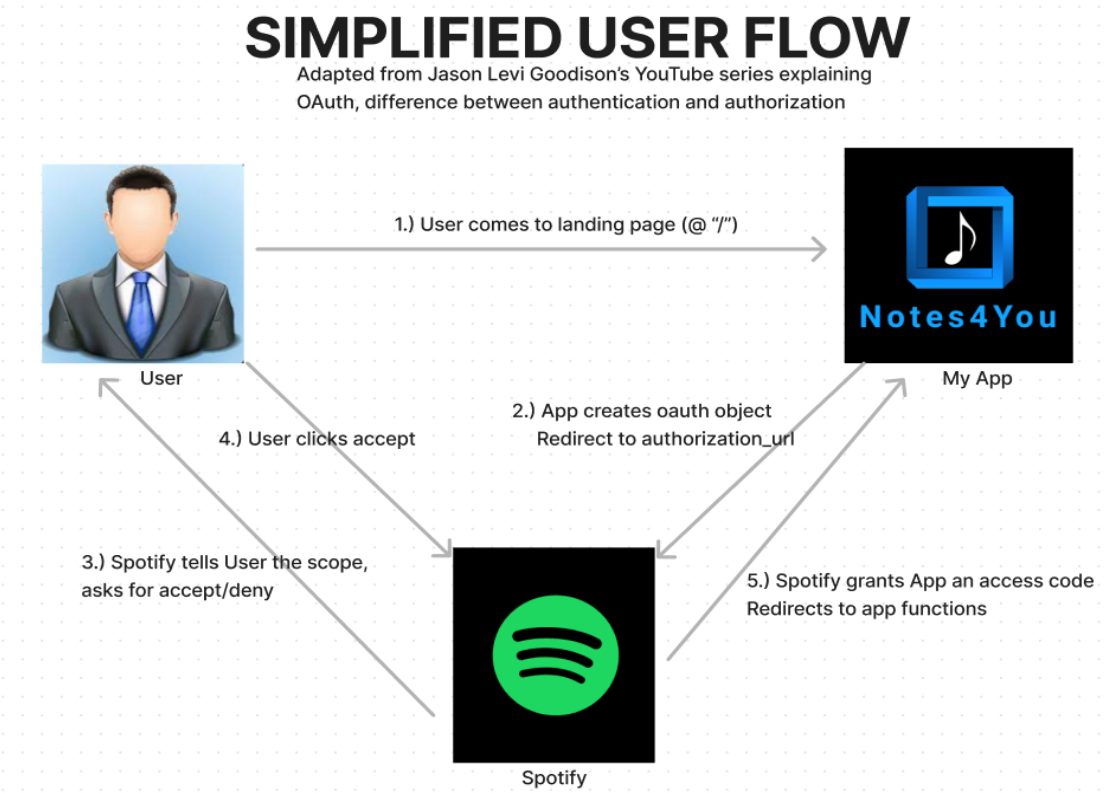
By launching the above components, users will see a GUI as shown below.

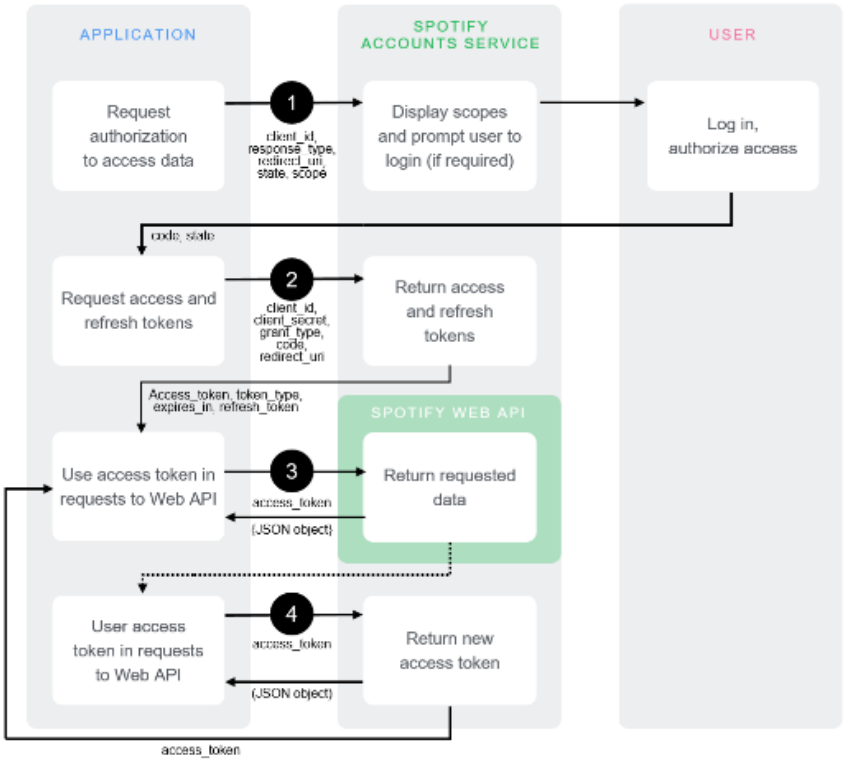


OAuth2

A standard user authorization protocol is **OAuth2** which is accessed through the SpotiPy Python library for the Spotify Web API. Seen below are screenshots presenting the simplified and comprehensive versions of the process.

*Please see full reference at: https://developer.spotify.com/documentation/web-api/tutorials/code-flow



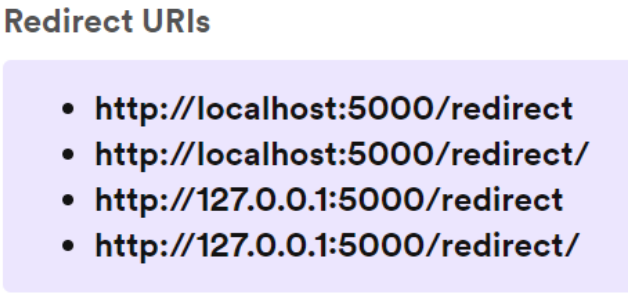


Continuing with **OAuth**, there are 4 important parameters required to create the object when the create_spotify_oauth() function is called in any endpoint.

```
#Method Usage: Implement Spotify OAuthentication procedure
def create_spotify_oauth():
    return SpotifyOAuth(
        client_id = "placeholder",
        client_secret = "placeholder",
        redirect_uri = url_for("redirectPage", _external = True),
        scope = "user-library-read, user-modify-playback-state, user-read-playback-state")
```

- **client_id:** references the application instance hosted by Spotify on the Developer Dashboard.
- **client_secret:** a highly sensitive key that protects authentication information used by Spotify to verify the client.
- **redirect_uri:** list of locations that users will be redirected to after authorization
- **scope:** permissions that app requires users to agree to, for full functionality

For example, these are a list of redirect_uri addresses which I have included on the Spotify Notes4You Developer Dashboard to ensure users can reach my /redirect and /getTrack endpoints.



These details allow me to send various API requests to get track data, information about user devices, and their inputs for track controls.

Spotify Web API

This section contains a quick rundown of the crucial API features used here for obtaining and using the required Spotify data in this project.

- **Get Available Devices**
  - Request Type: **GET**
  - Function call: sp.devices()
  - Useful Return Parameters
    - id: device id referencing (webPlayer || exectuablePlayer)
    - isActive: activation of device to ensure working order
    - isRestricted: false, as N4Y has buttons to manipulate track state
- **Get Playlist Tracks**
  - Request Type: **GET**
  - Function Call: sp.playlist_tracks(playlist_id, limit = int)
  - Useful Return Parameters
    - track_name: continuously captures String into array
    - artists: continuously captures String into array
    - cover_art_url: obtained from ‘album’ → ‘images’ → [0] → ‘url’
- Start Playback
  - Request Type: **PUT**
  - Function Call: sp.start_playback(trackUri, deviceId)
  - Useful Return Parameters
    - offset: *used to sync playback with YouTube video
- Pause Playback
  - Request Type: **PUT**
  - Function Call: sp.pause_playback(device_id)
  - Useful Return Parameters
    - device_id: fulfills request with current, or with any device_id running on Spotify servers that fulfills isActive == True
- Current Playback
  - Request Type: **GET**
  - Function Call: sp.current_playback()
  - Useful Return Parameters
    - track_state; enables the same HTML form’s POST logic to be used for play and pause functionality

* The version of Notes4You as of October 5, 2023 does not support auto-sync functionality to match Spotify audio to the proper MIDI timings found in YouTube tutorials. Please see **Future Upgrades** for upcoming plans.

YouTube Data API v3

This section contains a quick rundown of the aforementioned JavaScript functionality, from the perspective of Google Cloud Platform provided Youtube Data API v3.

Due to the information access not requiring user YouTube data, an API key from the Google Cloud Console was created to access public data.

Next, as mentioned in the section **JavaScript Video Retrieval,** a function call is made to construct an API url based on variable text content. Notice the dynamic apiKey and query.

```
const apiKey = "placholder";
```

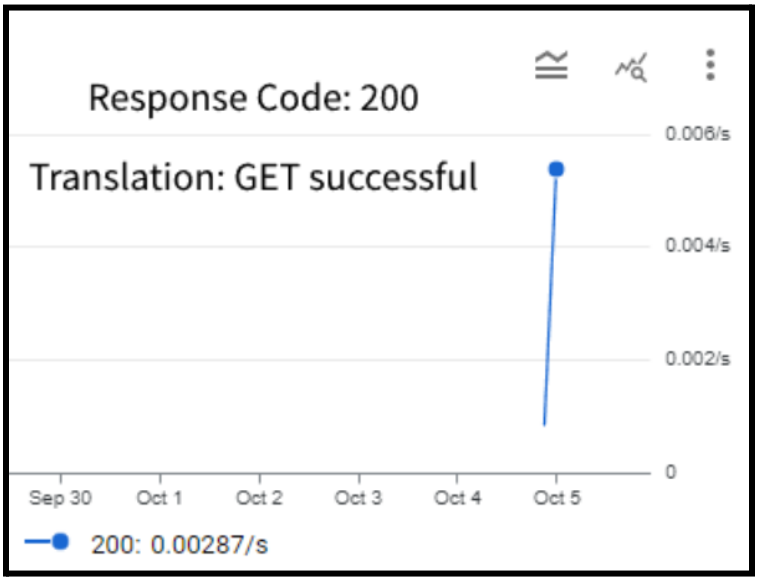
```
const query = `${trackName} ${artistName} piano tutorial`;
```

```
const apiUrl = `https://www.googleapis.com/youtube/v3/search?key=${apiKey}
                    &part=snippet&type=video&maxResults=${maxResults}&q=${query}`;
```

Said url can be forwarded via .fetch, and the response data, composed of videoId can be delivered to the HTML frontend on an iframe.

To validate this functionality, I can visit my dashboard for API/Service Details at https://console.cloud.google.com/apis/api/youtube.googleapis.com/metrics.

A response code of 200 corresponds to a successful GET request for our YouTube video query.



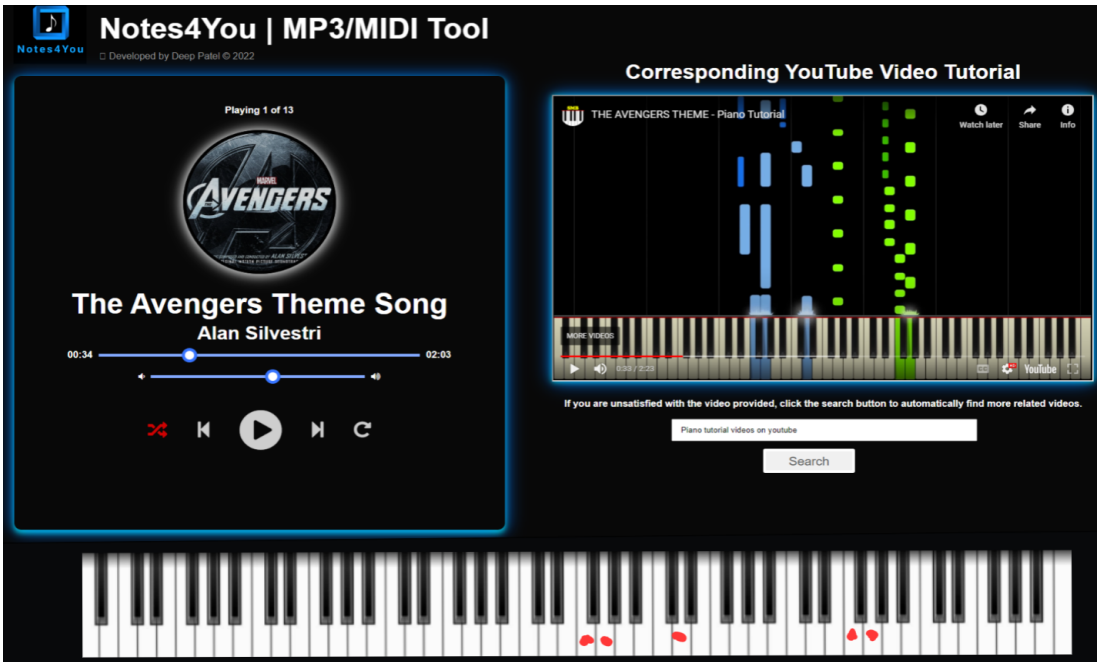
Future Upgrades

The base of this project came from an idea I had to improve my personal time when playing piano. For close to a decade, being a piano player who learned the most complex of songs faster by watching and listening to other musicians rather than reading sheet music, I wanted a system to learn new songs in a rapid fashion.

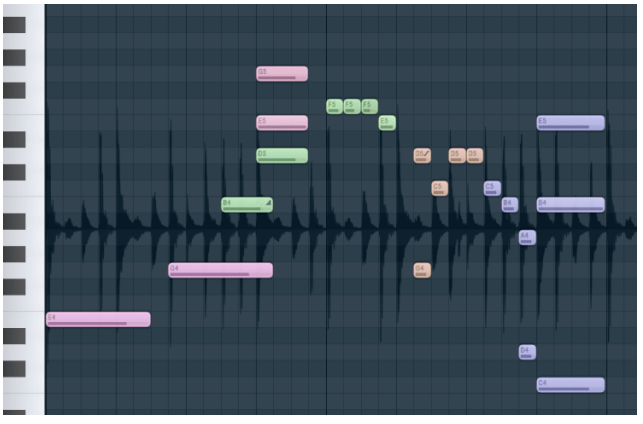
Last year when I began to learn the basics of vanilla HTML, CSS, and JavaScript, I managed to learn the development of a music control panel, able to cycle hardcoded song data (track name, artist name, album art, etc.)

Fast forward to 2023, I wanted to move towards a better automated implementation, which would require no extra work on my behalf, outside the realm of making my Spotify playlists.

Next, through the use of the MIDO Python library, I would like to connect my very own digital piano and use the MIDI information (the colored falling blocks in the YouTube video) to render a entirely digital piano on-screen, which would highlight notes needing to be pressed, listening for input, and automatically managing the video playback to help users visualize the precise note names and timings in real-time.



Coupled with music production softwares like FL Studio, I believe this could be a great learning tool for anybody looking to enhance their knowledge in the structuring of piano compositions, even if modern songs are not being orchestrated like they used to.



Screenshot Library

For a display of various screenshots, and potentially a screen-recording of the software in use, please visit my Github page, linked https://github.com/deep-patel21/Notes4You/tree/main.

Credits

A large portion of my understanding of these systems was kickstarted by Jason Goodison, found at https://www.youtube.com/@JasonGoodison, who made a short video series describing his personal experience using the Spotify API.

Watching his resources, I became interested in implementing the automated playlist fetching for the project I had once started to develop.

For anybody who wants to use the Spotify API, visit
https://developer.spotify.com/documentation/web-api/tutorials/getting-started

For anybody who wants to use the YouTube Data API v3, visit
https://developers.google.com/youtube/v3

For anybody who wants to use the MIDO API, visit
https://mido.readthedocs.io/en/stable/

Thank you for reading :)
- Deep Patel