Music Recommendation System API: Design and Implementation Document

Chen Jin
Brown University

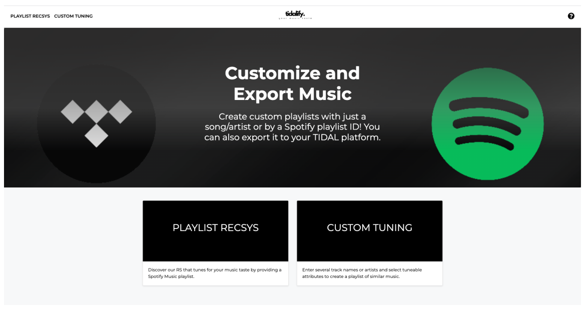
Yingjia
Brown University

Introduction

In the contemporary digital music landscape, the seamless integration of user preferences across diverse streaming platforms remains a significant challenge. Tidalify, developed as a cross-platform music recommender engine, is designed to address this crucial need in the digital music space. It focuses on enabling users to effortlessly transfer their music preferences and playlists between popular services like Spotify and TIDAL, a feature that is not extensively explored under current music recommendation systems.

The core problem Tidalify aims to solve is the fragmentation of user data. The primary goal for our Web API project is to develop endpoints that give users more flexibility and automation in getting recommendations and optionally into their preferred music streaming platforms. We have added APIs for single and batch music recommendations, developed a feature to create a new playlist on Spotify based on recommendations, and ensured usability and reliability through comprehensive unit tests.

Here's a demo video to Tidalify and its web API functionalities: https://youtu.be/VgJObauCoyw



Create custom playlists with just a song/artist or by a Spotify playlist ID! You can also export it to your TIDAL platform.

Discover our RS that tunes for your music taste by providing a Spotify Music playlist.

Enter several track names or artists and select tuneable attributes to create a playlist of similar music.

Figure 1: Home page of Tidalify

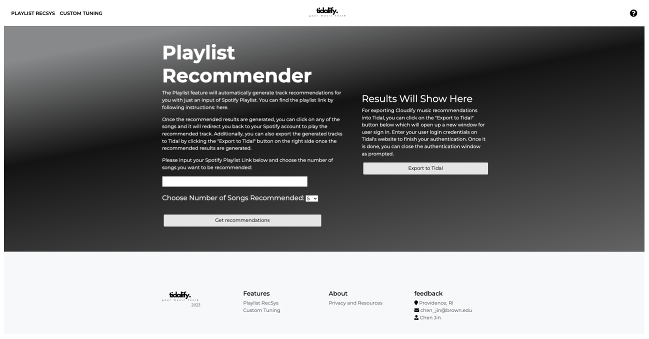
About Tidalify

Tidalify is a Flask project that offers two user-centric features, providing music recommendations based on the Spotify music platform.

Feature 1: Recommender System based on Content-Based Filtering Algorithm

This project utilizes the dataset from the 2018 RecSys Challenge, which comprises 1 million playlists. The implementation of the Content-Based Filtering algorithm is informed by the following workflow framework: 1) Part I: Extracting song data from Spotify's API using Python, 2) Part II: Exploratory Data Analysis (EDA) and Clustering, 3) Part III: Building a Song Recommendation System with Spotify, 4) Part IV: Deploying a Spotify Recommendation Model with Flask

The prediction algorithm facilitates the fetching of recommended songs within the Spotify ecosystem, enabling users to export these recommendations to their TIDAL music accounts. This integration is achieved using TIDAL's OAuth2 protocol for user authorization. For more details, see the tidalapi documentation.



The Playlist feature will automatically generate track recommendations for you with just an input of Spotify Playlist. You can find the playlist link by following instructions: here.

Once the recommended results are generated, you can click on any of the songs and it will redirect you back to your Spotify account to play the recommended track. Additionally, you can also export the generated tracks to Tidal by clicking the "Export to Tidal" button on the right side once the recommended results are generated.

Please input your Spotify Playlist Link below and choose the number of songs you want to be recommended:

Choose Number of Songs Recommended: 5

For exporting Cloudify music recommendations into Tidal, you can click on the "Export to Tidal" button below which will open up a new window for user sign in. Enter your user login credentials on Tidal's website to finish your authentication. Once it is done, you can close the authentication window as prompted.

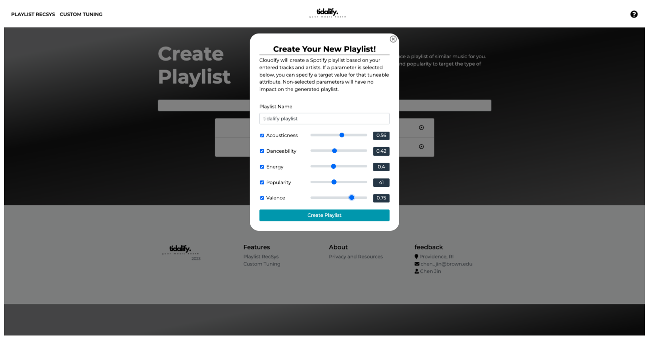
tidalify. 2023

Providence, RI
chen_jin@brown.edu
Chen Jin

Figure 2: Playlist Recommender System in Tidalify

Feature 2: ‘/Recommend‘ API & User Tuning

For the "Custom Tuning" feature, users have the ability to alter some of the audio features available through the Spotify API for music recommendation. They can input their favorite songs or artists along with ‘`audio_features`‘ to generate a playlist recommended to them straight back in their Spotify playlist.



Create Your New Playlist!

Cloudify will create a Spotify playlist based on your entered tracks and artists. If a parameter is selected below, you can specify a target value for that tuneable attribute. Non-selected parameters will have no impact on the generated playlist.

Playlist Name: tidalify playlist

✓ Acousticness	0.56
✓ Danceability	0.42
✓ Energy	0.4
✓ Popularity	41
✓ Valence	0.75

Figure 3: Custom Tuning feature in Tidalify

Information Page

For more information regarding the functionalities offered in this project, please refer to the ‘information‘ page. The application needs to access features within the Spotify SDK, which requires certain user scopes. You can also revoke ‘permissions‘ from the link provided here.

How to use

To clone the repository: ’git clone https://github.com/brown-cs1680-f23/final-web-api.git’ To install the dependencies: ’pip install -r requirements.txt’

You will also need to set up your own environment variables for Spotify and Tidal. Variables like ‘`CLIENT_ID`‘, ‘`CLIENT_SECRET`‘ are used for API calls to these two music platforms. Please acquire necessary credentials from your Spotify for developers account. You will also need to set up the callback route within Spotify Developer Dashboard for rerouting. Similarly for TIDAL.

Implementation

Data Source Tidalify utilizes a comprehensive dataset originating from the ACM 2018 RecSys Challenge, encompassing one million Spotify playlists. A subset of this dataset, incorporating 30,000 songs, forms the basis of our item-centred content-based filtering Music Recommendation System (MRS). Each playlist file data looks like this:

```
"playlists": [
{
    "name": "Road Trippin’",
    "collaborative": "false",
    "pid": 404000,
    "modified_at": 1500076800,
    "num_tracks": 78,
    "num_albums": 65,
    "num_followers": 1,
    "tracks": [
    {
        "pos": 0,
        "artist_name": "Mach & Daddy",
        "track_uri": "spotify:track:1IlJpEiMt4nu
        Lr3cWFQUxa",
        "artist_uri": "spotify:artist:5Q1vijNjTV",
        "track_name": "La Botella",
        "album_uri": "spotify:album:0yno13qSZ1kSJ",
        "duration_ms": 179213,
        "album_name": "Desde Abajo"
    },
```

Each of the files give around 1000 playlists of various sizes and genres. The data retrieved playlist entry included unique track identifiers, metadata (such as artist, album, and genre information), and the order of tracks within the playlist.The dataset showcased how playlists were structured, providing insights into how songs were curated and grouped together by users. This aspect allowed for analyzing playlist diversity, common song sequences, and user preferences regarding track combinations.

Dual-feature approach Tidalify uses a dual-feature approach to music recommendation, each underpinned by distinct recommendation algorithms. This section delves into the algorithmic nuances of these features, alongside an exploration of the platform’s data exportation capabilities which facilitate automated music taste transfer across platforms. The entirety of Tidalify’s architecture can be found in Figure 1.

The content-based filtering algorithm, as detailed previously, activates when a user inputs a Spotify playlist URL via the frontend interface. The backend system then makes an API call to Spotify and parses this URL to extract `audio_features` associated with the tracks in the user-provided playlist. Post extraction, Tidalify summarizes and normalizes the user’s playlist into a singular vector. This vector forms the basis for conducting cosine similarity analyses using `sklearn`. The system evaluates this vector within our song database, pinpointing tracks with the closest similarity scores. The top matches are then sent back to the frontend, allowing users to export these selections to various music platforms.

The hybrid MRS plus custom tuning on `audio_feature` is used for the second feature. Tidalify capitalizes on Spotify’s advanced Hybrid Recommendation System (RS), which synergizes multiple algorithmic approaches for optimal music

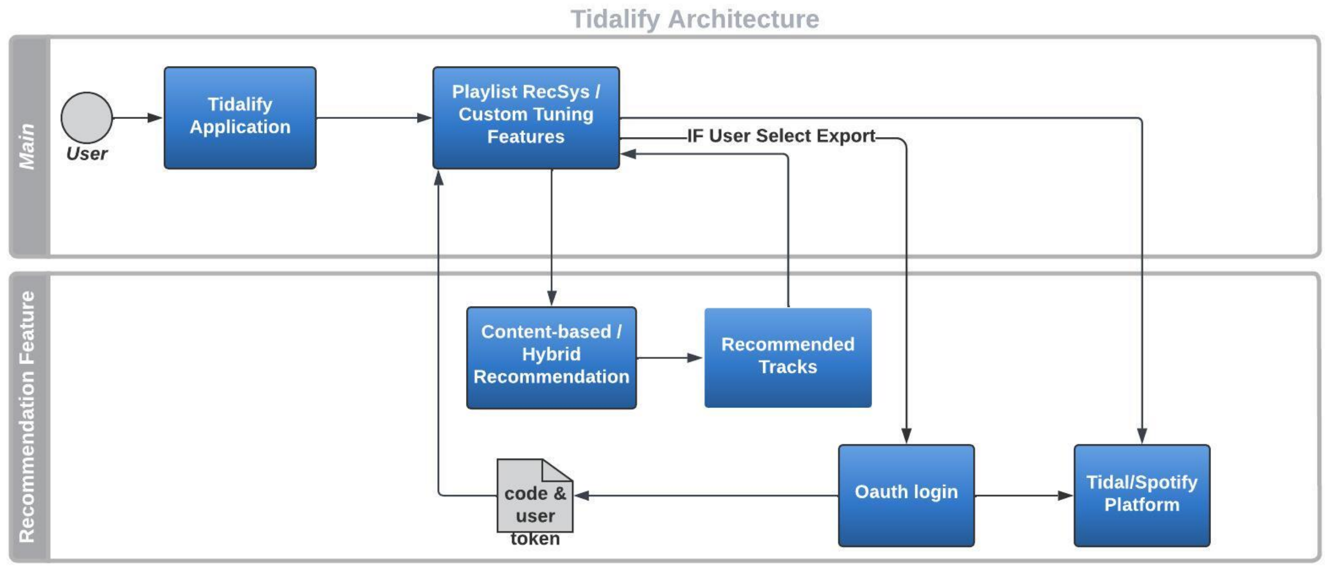


Figure 4: Prototype Architecture

recommendation. A key aspect of Tidalify's second feature is its integration with Spotify's Hybrid RS through the developer API, coupled with a unique custom tuning capability. This custom tuning allows users to manually adjust up to five out of twelve sonic characteristics identified in `audio_features`, tailoring the music recommendations to their specific preferences.

User Authentication: Users log into their Spotify account via OAuth2 authentication. The user employs the Tidalify application interface, termed the UserAgent, to request a login to Spotify, which then directs them to Spotify's OAuth2 login interface. Upon entering their credentials and consenting to Tidalify's access, Spotify authenticates the user and returns an authorization code to Tidalify's web server. Tidalify's server exchanges this code for an access token from Spotify's authorization server. With this token, Tidalify gains the ability to fetch data from Spotify's resource server, allowing Tidalify to curate personalized playlists and music recommendations for the user.

Track and Artist Selection: Users can input song or artist names, with an autocomplete feature aiding in the selection process.

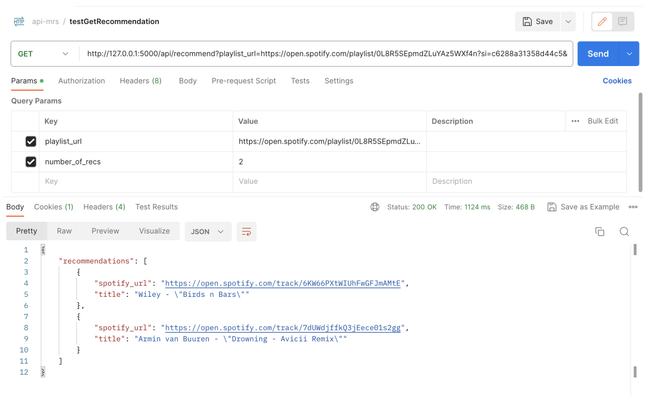
Export to Spotify: A modal prompts users to name their new playlist and customize `audio_features`. Based on the user's login token, Tidalify then creates this personalized playlist in the user's Spotify account.

The Data Exportation Framework The data exportation framework is designed such that recommendation contents from one platform is parsed into a standardized `Song` instance by our middleware which is then normalized for consistency in searching. Using the similar oauth2 workflow, users will then get authentication into their TIDAL platform and post-authentication, system will automatically imports recommendations into user's TIDAL account `SEARCH` based on artist name and song name in the `Song` instance, `CREATE` playlist, and `ADD` tracks. The universal data structure for exportation is, however, limited based on platforms. Some music streaming platforms does not support International Standard Recording Code (ISRC) and to bypass the searching issue, we used artists and track names for searching instead.

Extended API Design/Implementation

1. **Single Recommendation API (`/api/recommend`)**: This endpoint enables users to obtain music recommendations based on a single playlist URL. It takes a `playlist_url` and an optional `number_of_recs` parameter, ensuring the number of recommendations does not exceed 40. Upon receiving valid parameters, the API utilizes the `recommend_from_playlist` function, employing a content-based filtering algorithm to suggest songs tailored to the user's musical preferences, returning the results in a JSON object.

2. **Batch Recommendation API (`/api/batch_recommend`)**: Designed to process multiple playlist URLs simultaneously, this endpoint returns a set of recommendations for each playlist. It accepts a JSON payload containing `playlist_urls` and an optional `number_of_recs`.



GET http://127.0.0.1:5000/api/recommend?playlist_url=https://open.spotify.com/playlist/0L8R5SEpmdZLuYAz5WXf4n?si=c6288a31358d44c5&

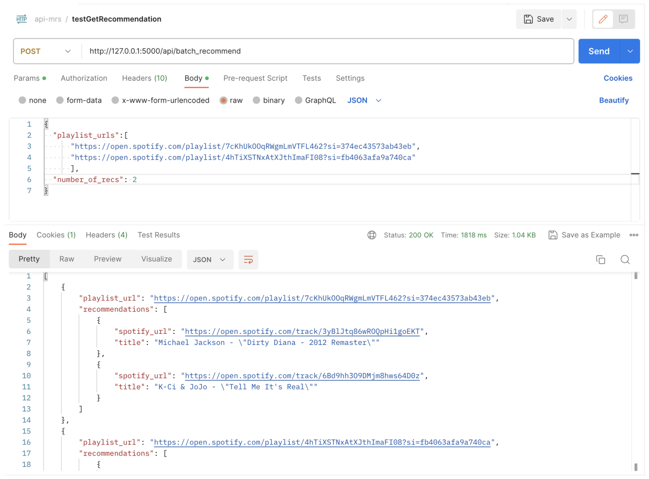
Query Params

	Key	Value	Description
✓	playlist_url	https://open.spotify.com/playlist/0L8R5SEpmdZLu...	
✓	number_of_recs	2	
	Key	Value	Description

Status: 200 OK Time: 1124 ms Size: 468 B

```
{
    "recommendations": [
        {
            "spotify_url": "https://open.spotify.com/track/6KW66PXtWIUhFwGFJmAMtE",
            "title": "Wiley - \"Birds n Bars\""
        },
        {
            "spotify_url": "https://open.spotify.com/track/7dUWdjffkQ3jEece01s2gg",
            "title": "Armin van Buuren - \"Drowning - Avicii Remix\""
        }
    ]
}
```

Figure 5: Postman test for Single Recommendation API



POST http://127.0.0.1:5000/api/batch_recommend

```
{
  "playlist_urls":[
      "https://open.spotify.com/playlist/7cKhUkOOqRWgmLmVTFL462?si=374ec43573ab43eb",
      "https://open.spotify.com/playlist/4hTiXSTNxAtXJthImaFI08?si=fb4063afa9a740ca"
      ],
  "number_of_recs": 2
}
```

Status: 200 OK Time: 1818 ms Size: 1.04 KB

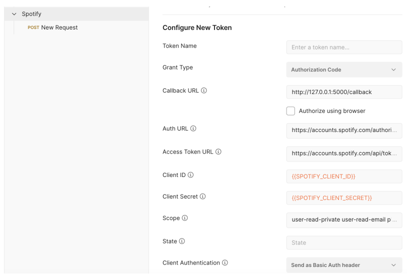
```
[
    {
        "playlist_url": "https://open.spotify.com/playlist/7cKhUkOOqRWgmLmVTFL462?si=374ec43573ab43eb",
        "recommendations": [
            {
                "spotify_url": "https://open.spotify.com/track/3yBlJtq86wROQpHi1goEKT",
                "title": "Michael Jackson - \"Dirty Diana - 2012 Remaster\""
            },
            {
                "spotify_url": "https://open.spotify.com/track/6Bd9hh309DMjm8hws64D0z",
                "title": "K-Ci & JoJo - \"Tell Me It's Real\""
            }
        ]
    },
    {
        "playlist_url": "https://open.spotify.com/playlist/4hTiXSTNxAtXJthImaFI08?si=fb4063afa9a740ca",
        "recommendations": [
            {
```

Figure 6: Postman test for Batch Recommendation API

3. **Playlist Creation API (`/api/playlist/create`)**: This API interacts with Spotify's API to create a new playlist based on the recommendations. It requires a valid Spotify token for authentication and authorization. The workflow is as follows:

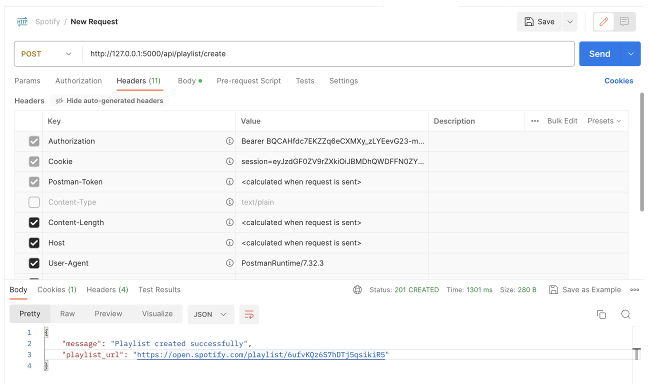
    - In Postman, clients initially call the Spotify OAuth2 API to obtain a user token, setting up an environment with `client_key` and `client_secrets` for Spotify authentication.
    - With the `Authorization` token in the request header, users can make API calls to `/api/playlist/create`, which in turn makes calls to the user's Spotify account and exports the recommended songs to the user's profile.

4. **Playlist Creation Based on Songs and Attributes API (`/api/playlist/create_from_songs`)**: This API interacts with Spotify's API to create a new playlist



Token Name	Enter a token name...
Grant Type	Authorization Code
Callback URL	http://127.0.0.1:5000/callback
	☐ Authorize using browser
Auth URL	https://accounts.spotify.com/authori...
Access Token URL	https://accounts.spotify.com/api/tok...
Client ID	{{SPOTIFY_CLIENT_ID}}
Client Secret	{{SPOTIFY_CLIENT_SECRET}}
Scope	user-read-private user-read-email p...
State	State
Client Authentication	Send as Basic Auth header

Figure 7: Setting up Postman for Playlist Creation API



POST http://127.0.0.1:5000/api/playlist/create

	Key	Value	Description
✓	Authorization	Bearer BQCAHfdc7EKZZq6eCXMXy_zLYEevG23-m...	
✓	Cookie	session=eyJzdGF0ZV9rZXkiOiJBMDhQWDFFN0ZY...	
✓	Postman-Token	<calculated when request is sent>	
☐	Content-Type	text/plain	
✓	Content-Length	<calculated when request is sent>	
✓	Host	<calculated when request is sent>	
✓	User-Agent	PostmanRuntime/7.32.3	

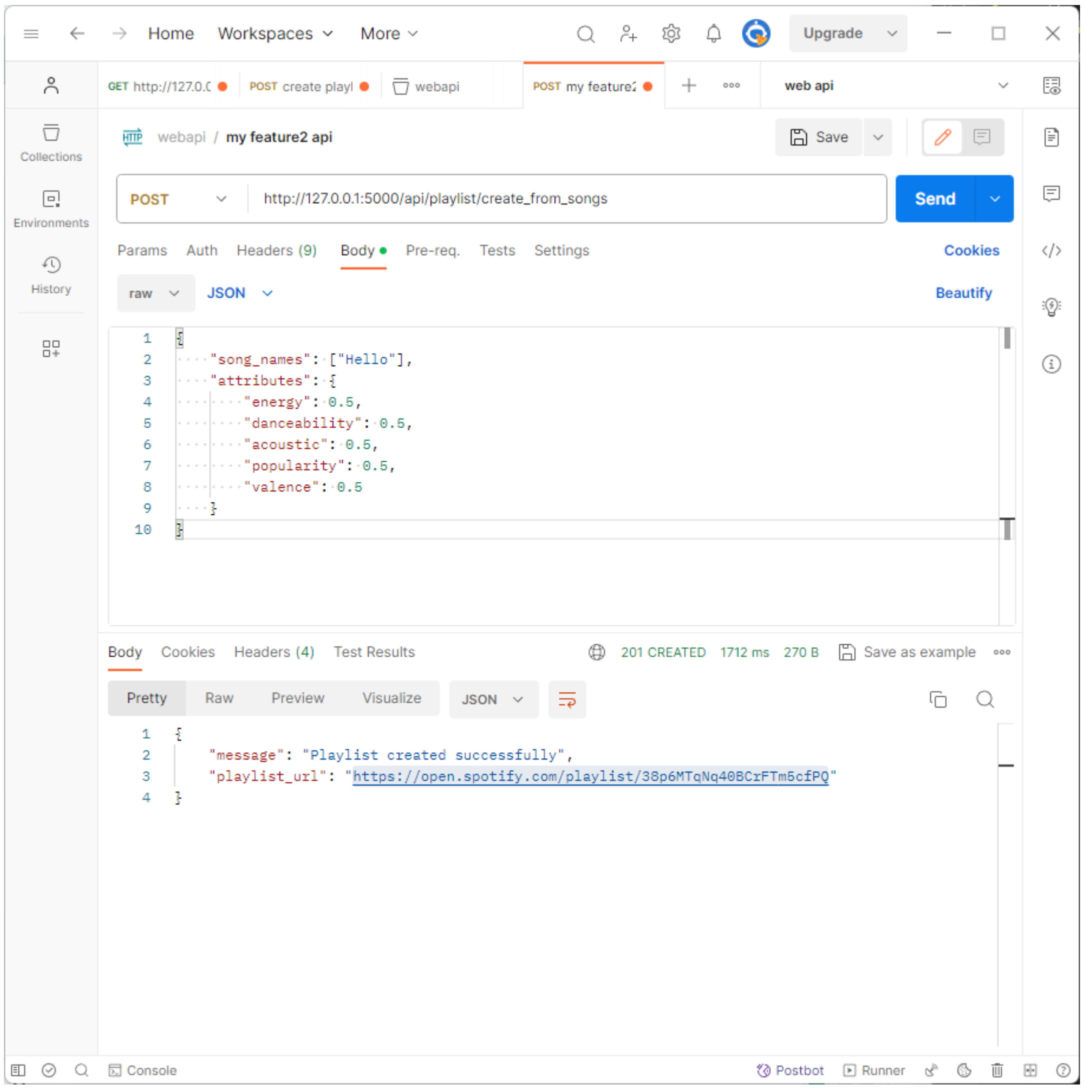
Status: 201 CREATED Time: 1301 ms Size: 280 B

```
{
    "message": "Playlist created successfully",
    "playlist_url": "https://open.spotify.com/playlist/6ufvKQz6S7hDTj5qsikiR5"
}
```

Figure 8: Making API calls to Playlist Creation API

based on user's input song names and tunable attributes. It also requires a valid Spotify token for authentication and authorization. The workflow is as follows:

- Similar to the last API, clients initially call the Spotify OAuth2 API to obtain a user token, setting up an environment with `client_key` and `client_secrets` for Spotify authentication.
- With the `Authorization` token in the request header, users can make API calls to Spotify and search for the best match songs of the input song names.
- Once we have the matched songs' URLs, we will use them together with the user's input attributes to get the recommended songs by making API calls to Spotify.
- Finally, we will create a new song list, add recommended songs to this list, and export this list to user's Spotify account.



POST http://127.0.0.1:5000/api/playlist/create_from_songs

```
{
    "song_names": ["Hello"],
    "attributes": {
        "energy": 0.5,
        "danceability": 0.5,
        "acoustic": 0.5,
        "popularity": 0.5,
        "valence": 0.5
    }
}
```

```
{
    "message": "Playlist created successfully",
    "playlist_url": "https://open.spotify.com/playlist/38p6MTqNq40BCrFTm5cfPQ"
}
```

Figure 9: Playlist Creation Based on Songs and Attributes API