Appendix - Syncify

Syed Ali Haider* Jasmeen Kaur†
Email: sh6070@nyu.edu* jk7297@nyu.edu†

I. Evolution of Project Focus and Challenges in Data Acquisition

Initially, our project aimed to develop a recommendation system for the financial market, specifically to identify stocks to invest in. However, after extensive research, we realized that accurately predicting stock trends requires in-depth financial expertise and significant time investment—resources we lacked. The challenge is compounded by the fact that even Wall Street experts and major financial institutions, with all their resources, struggle to predict stock movements reliably, often analyzing trends retrospectively to understand the influencing factors, and peers from our class who created the Financial Advisor Chatbot, reported these same challenges and could not come up with predictive policies to forecast stock positions, which highlights the complexity and domain knowledge needed in this field.

Given these obstacles, we shifted our focus to creating a recommendation system for legal cases. Unfortunately, we encountered significant difficulties in obtaining meaningful data. The available legal data was either hashed or lacked crucial elements such as abstracts, and some datasets contained only partial abstracts with insufficient case details. This lack of comprehensive and structured legal data hindered our progress.

Consequently, we decided to explore the field of music recommendation systems. This domain provided accessible and well-structured data, allowing us to successfully implement a recommendation system. Our project now leverages song features and lyrics to deliver personalized music recommendations, as detailed in our main report. In this entire process, we have consistently aimed to explore the idea of recommendation systems across various fields, adhering to our core objective. Despite changing the application domain, our fundamental project goal of developing an effective recommendation system has remained unchanged.

II. Data Model for Syncify

The music recommendation system will store Users and their Ratings for the recommended songs. It also has potential extensions to include Liked Songs and User Playlists.

- Users can have multiple Ratings (via references).
- Each Rating is associated with a specific song index and a user (via references).
- Potentially, Songs can be part of multiple Playlists and each Playlist can belong to a User and contain multiple Songs (via references).

III. Entity Relationships for Syncify

- User to Ratings: One to Many; One user can have multiple ratings.
- Ratings to User: Many to One; Each rating is associated with one user.
- Ratings to Song Index: Many to One; Each rating corresponds to a specific song

Potential Extension:

- User to Playlists: One to Many; One user can have multiple playlists.
- Playlists to Songs: Many to Many; A playlist can have multiple songs, and a song can be part of multiple playlists.
- Songs to Playlists: Many to Many; A song can be part of multiple playlists, and a playlist can have multiple songs.

A. An Example User

```
{
  "username": "musicfan123",
  "email": "musicfan123@example.com",
  "password": "hashed_password",
  "ratings": [1, 2, 3]
# an array of references to Rating IDs
}
```

B. An Example Rating Document

```
{
  "user_id": 1,  # a reference to a User ID
  "rating": 5,
  "song_index": 123,
  "date": "2024-05-17T12:34:56Z"
}
```

C. An Example Playlist Document

```
{
    "playlist_id": 1,
    "user_id": 1,  # a reference to a User ID
    "songs": [1, 2, 3],
# an array of references to Song IDs
    "added_on": "2024-05-17T12:34:56Z"
}
```

D. An Example Song Document

```
{
    "song_id": 1,
    "name": "Imagine",
    "artist": "John Lennon",
    "spotify_id": "spotify:track:12345"
}
```

IV. Use Cases for Syncify

1) As a non-registered user, I can register a new account with the site so that I can personalize my experience.
2) As a registered user, I can log in to the site to access my saved playlist and recommendations.
3) As a user, I can rate songs to provide feedback and improve my personalized recommendations.
4) As a user, I can view a list of song recommendations based on my ratings and preferences.
5) As a user, I can add recommended songs to my personal playlist for easy access.
6) As a user, I can search for specific songs or artists to receive tailored recommendations.
7) As a user, I can log in to my Spotify account to integrate my Spotify playlists with the recommendation system.
8) As a user, I can add recommended songs directly to my Spotify playlists for seamless music management.

V. Site Map

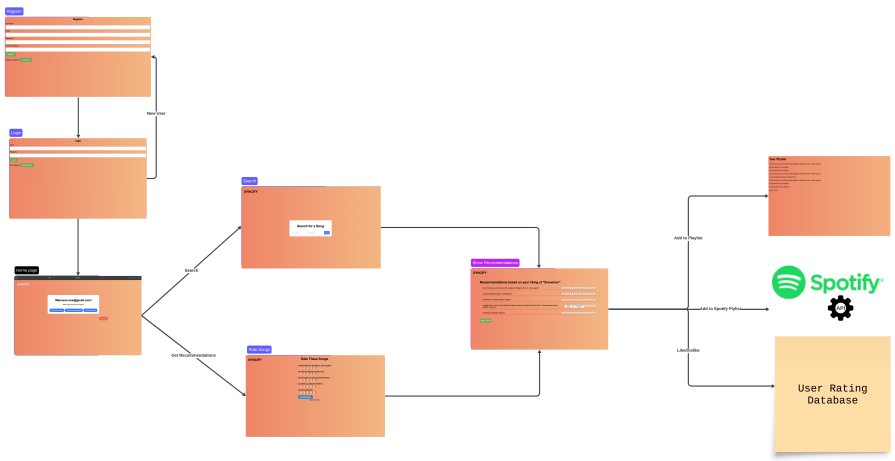

Fig. 1.

VI. Additional Challenges

A. SQL Database

One of the main challenges we faced was trying to store information in the database, particularly adding the ratings information in the SQLAlchemy database. Despite having the tables created with all the required columns which was verified using the SQLAlchemy's inspect library, the compiler showed an error indicating that the data table was not available. This issue took significant time to troubleshoot, as gathering user ratings is a key feature for evaluating our recommendation algorithm. Due to the extensive time required to resolve this problem, we decided to use data frames and text files to store user ratings. This approach works for now since we don't have a large number of users. However, we will continue working on fixing the database issue to ensure it is ready for a larger audience in the future.

B. Spotify API

When we first started working with the Spotify API, we initially fetched songs directly from Spotify's Web API to provide users with a broader range of options for recommendations beyond our current database. However, we realized that continuously fetching and processing information from Spotify in real-time will be highly inefficient and costly. Therefore, we decided to rely on our internal database for now, with plans to implement real-time Spotify data integration in the future.

Another significant challenge was using the API to add recommended songs to the user's playlist. We encountered persistent issues with mismatched redirect URIs and access tokens. After intensive troubleshooting, we discovered that the error was resolved simply by replacing 'localhost' in the redirect URL with the IP address '127.0.0.1', which also represents localhost. This small change successfully resolved the mismatched redirect URI and access token issues.