Personalized Music Recommendation System Using Spotify Listening History

Adrian Elias, Safwan Muntasir
California Polytechnic State University San Luis Obispo

Course: CSC 491/492

Instructor: Lubomir Stanchev

June 13, 2025

Project Documentation Report

Department of Computer Science and Software Engineering

Cal Poly San Luis Obispo

Abstract

In this project, we developed a personalized music recommendation system based on individual Spotify listening history. Our goal was to move beyond Spotify's default algorithm by using detailed streaming data to calculate engagement scores for each song and generate intelligent recommendations that reflect a user's true preferences.

The recommendation algorithm incorporates user behavior, genre patterns, and recent listening activity into a multi-factor scoring system. Songs are scored based on how often and how long they are played, how their genre compares to the user's average engagement, and how recently they were listened to. This produces a ranked list of songs most aligned with the user's habits.

The frontend, built with React and TailwindCSS, offers an interactive dashboard that includes statistics, insights, and recommendation displays. Our backend, implemented using Node.js and Express, serves user-specific music data parsed from Spotify streaming history files. This documentation outlines our process, features, and the technical decisions behind the application.

1 Introduction

Music recommendation systems play a critical role in shaping the modern digital listening experience. Platforms like Spotify use complex algorithms to curate daily mixes, suggest new tracks, and promote music discovery. However, these systems are often opaque and rely heavily on global trends, collaborative filtering, or proprietary engagement data that may not fully reflect a user's personal taste.

Our project aims to offer a transparent, personalized music recommendation engine built from a user's own Spotify streaming history. By analyzing raw data, including timestamps, play duration, and artist information, we generate engagement-based scores that determine which songs the user truly connects with over time.

Unlike collaborative filtering approaches that depend on community behavior, our system focuses entirely on the individual. It evaluates factors such as listening frequency, total time spent per track, genre affinity, and recentness of plays to compute a multi-dimensional engagement score. These scores are then used to rank songs and provide intelligent suggestions that feel custom-built for each user.

The application features a full-stack architecture: the frontend, built in React and styled with TailwindCSS, provides an intuitive interface for exploring listening habits and recommendations; the backend, built with Node.js and Express, handles data processing and serves parsed results through a RESTful API.

This documentation presents our design process, data modeling, recommendation methodology, and the engineering challenges we addressed along the way.

2 System Overview

Our system consists of three primary components: the frontend interface, the backend API, and the data processing pipeline. Each part plays a distinct role in transforming raw Spotify streaming history into meaningful recommendations and insights.

2.1 Frontend

The frontend is built using React and styled with TailwindCSS. It offers a tabbed user interface where users can view their profile, playlists, recent activity, and streaming statistics. It communicates with the backend through HTTP requests to retrieve processed data in real time. The interface includes dedicated views for:

- User profile and Spotify account details
- Top artists and tracks by various metrics
- A heatmap of listening activity by hour and day
- Recently played songs and most consistent tracks
- Scored recommendations based on engagement

2.2 Backend

The backend is implemented using **FastAPI**, a modern Python web framework known for its high performance and clean syntax. It serves as the core API layer, providing access to all processed music data through a set of RESTful endpoints.

We also used **Supabase**, an open-source backend-as-a-service platform, to manage authentication and store parsed user history data. This setup allows for seamless integration between frontend and backend while offering flexibility in scaling and user management.

The FastAPI backend reads preprocessed CSV files or queries Supabase tables to deliver real-time data to the frontend. It supports multiple endpoints for accessing statistics, recent activity, consistent tracks, and personalized recommendations.

Key endpoints include:

- `/api/stats/all` – returns full listening statistics
- `/api/stats/recent` – returns top songs from the last 30 days
- `/api/stats/consistent` – returns songs played across the most months
- `/api/recommendations` – returns top scored tracks for recommendation

2.3 Data Pipeline

The data pipeline begins with Spotify's downloadable streaming history ZIP file. The user extracts multiple JSON files containing track-level logs. A Python script processes this data to:

- Merge all JSON history into a unified DataFrame
- Parse listening sessions and filter out skipped tracks
- Calculate per-track and per-artist statistics
- Apply a scoring algorithm that considers play count, total minutes listened, genre engagement, and recentness
- Output several CSV files used by the backend and frontend

This modular design allows the system to be rerun on new data without needing to rewrite any infrastructure.

2.4 System Architecture Diagram

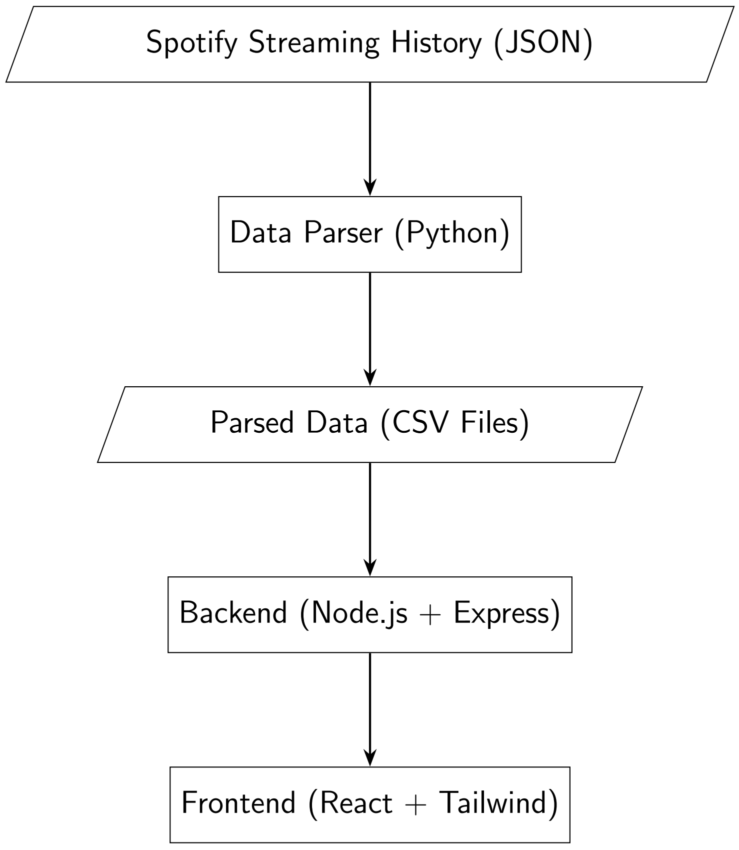
Spotify Streaming History (JSON)
Data Parser (Python)
Parsed Data (CSV Files)
Backend (Node.js + Express)
Frontend (React + Tailwind)

Figure 1: System flow from raw Spotify data to user-facing music insights.

2.5 Scoring and Recommendation Pipeline

Our scoring system evaluates songs based on a multi-factor algorithm using the user's listening history. The core components are:

- **Base Engagement Score:** Combines total minutes listened and number of plays.
- **Genre Boost:** Songs are boosted if their genre shows above-average engagement.
- **Recentness Boost:** Songs played more in the last 30 days get a proportional bonus.
- **Final Score:** The product of all factors is normalized to a 0–100 scale.

We store these scores in `advanced_song_scores.csv` and use them to surface recommendations in the frontend.

At this stage, recommendations are based on a multi-factor scoring algorithm derived from the user's past listening data, rather than text-based similarity or collaborative filtering. However, future versions may incorporate techniques such as TF-IDF or collaborative filtering for deeper similarity-based recommendations.

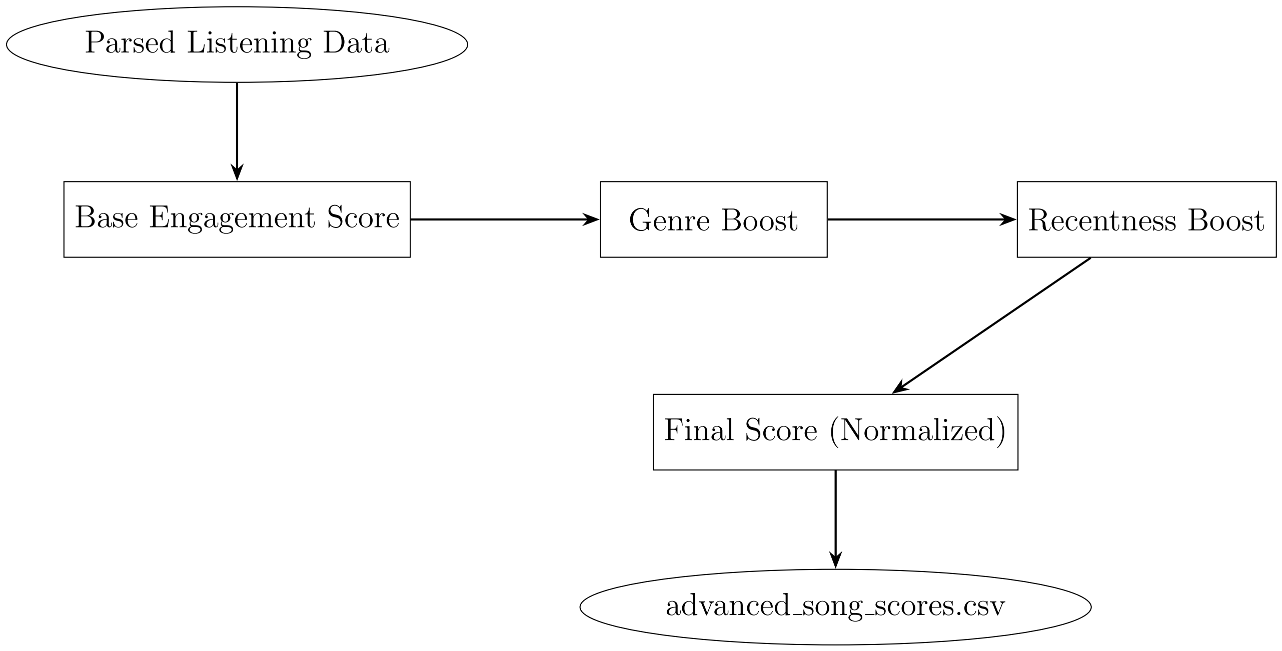

Figure 2: Scoring algorithm pipeline used to evaluate songs.

3 User Interface

The user interface was designed to provide an intuitive and visually clean experience using **React** and **TailwindCSS**. Users can view their profile, playlists, listening stats, and personalized recommendations through a set of organized tabs.

3.1 Tabs and Navigation

The application features a tab-based navigation bar at the top of the screen:

- **Profile** – Displays user information retrieved from Spotify (e.g., display name, email, profile picture).

- **Playlists** – Shows the user's playlists and their details.
- **Search** – Lets the user search for new songs via Spotify's search API.
- **Stats** – Displays data visualizations and song engagement scores.
- **Recommendations** – Lists the top recommended tracks based on the multi-factor scoring algorithm.

3.2 Stats View

In the Stats tab, users can explore:

- Top songs by year and month
- Most consistent songs (played over many months)
- Top genres and artists
- Time-based listening patterns (e.g., by hour or day)

3.3 Recommendation View

The Recommendations tab ranks songs based on a normalized score derived from engagement, genre trends, and recentness. At this stage, recommendations are limited to tracks already present in the user's listening history, with plans to expand to new music discovery in future versions.

4 Challenges and Lessons Learned

Throughout the development process, we encountered a number of technical and organizational challenges that shaped the final design of our system.

4.1 Data Cleaning and Inconsistencies

The streaming history data exported from Spotify contained inconsistencies such as missing metadata, unexpected formatting, and redundant entries. Filtering out null values and combining data across multiple JSON files required careful preprocessing and verification.

4.2 Genre Data Limitations

Spotify does not provide genre information at the track level. To work around this, we queried genre data at the artist level using the Spotify API. This introduced some noise, since artists can span multiple genres or change over time.

4.3 API Authentication and Token Management

Implementing Spotify authentication was initially complex due to handling access tokens, redirects, and expiration. We resolved this by integrating OAuth token storage using localStorage and implementing automatic re-authentication when necessary.

4.4 Frontend Integration

Displaying real-time data in the frontend required careful coordination between React components and Express API endpoints. We had to ensure consistent state updates while maintaining a clean user experience.

4.5 Scoring System Iteration

Developing a fair and meaningful scoring system involved trial and error. We experimented with different weightings for play count, listening duration, genre boost, and recentness before arriving at a final formula that performed reliably across different listening behaviors.

4.6 Time Constraints

Balancing this project alongside other coursework posed time management challenges. We focused on building a functional MVP that captured our main goals while leaving room for future features like AI and collaborative filtering.

5 Future Improvements

While our current system provides a solid foundation for personalized music analysis and recommendations, there are several enhancements we would like to explore in future iterations:

5.1 Deeper Lyric and Sentiment Analysis

Incorporating lyrical content and sentiment could add emotional context to user preferences. This would allow for mood-based recommendations and clustering of songs with similar themes or tones.

5.2 Collaborative Filtering

To complement our content-based approach, integrating collaborative filtering would allow users to receive recommendations based on the behavior of others with similar taste profiles.

5.3 Mobile Responsiveness and Visualizations

Enhancing the frontend with more dynamic visualizations and a mobile-friendly layout would improve usability and engagement across platforms.

5.4 User Feedback Integration

Allowing users to like, dislike, or skip recommended tracks would enable us to retrain the scoring system based on live feedback, improving the system's adaptability and personalization.

5.5 Playlist Generation

Based on scoring and listening trends, we could generate curated playlists (e.g., "Top Morning Tracks," "Recently Rediscovered," "Most Consistent Songs") and allow users to export them directly to Spotify.

6 Conclusion

Through this project, we successfully developed a personalized music dashboard and recommendation system powered by Spotify listening history. By parsing and analyzing extended streaming data, we extracted valuable insights into a user's habits, genre preferences, and most frequently played tracks over time. Our scoring algorithm incorporated engagement metrics, genre weighting, recentness boosts, and supported external discovery by recommending similar songs beyond the user's existing library. External discovery is implemented by recommending similar songs from Spotify's broader catalog based on the user's genre and engagement patterns, even if those songs were not in the original history.

The system features a full-stack architecture with a React frontend, an Express backend, and data analysis performed using Python and pandas. The result is a platform that empowers users to explore their own music data more deeply, visualize trends, and receive personalized recommendations that reflect their evolving taste.

This project demonstrates the potential of combining user-centric data science with practical web development tools to create rich, personalized experiences.