Elysium: A New Generation Music Sharing Application

`Final Proposal for ECE49595O`

Garrett Phillips — Benjamin Boardley — Jack Gardner
Purdue University, Elmore Family School of Electrical and Computer Engineering
{gwphilli@purdue.edu — bboardle@purdue.edu — gardne97@purdue.edu}

January 2024

1 Analysis of System

The goal of Elysium is to create a music hub for users regardless of their streaming service. It aims to connect users through cross-compatible music sharing and friend connections. The system includes an NLP-based music generation tool that populates playlists based on user input and automatic song and playlist uploads across streaming platforms.

The NLP-based music generation tool leverages ChatGPT APIs for natural language interaction in playlist creation. Users can input preferences, moods, or specific criteria, and the system uses ChatGPT's language understanding capabilities for music recommendations.

Elysium consists of a backend (Django) and frontend (React Native) connected by a web framework for user-friendly interaction. The cloud hosting platform will house servers, user data, and training data.

2 System Overview

Elysium addresses challenges in cross-platform music sharing and playlist generation. The frontend provides a user-friendly interface, while the backend manages communication with external music services, database interactions, and playlist algorithms. AWS RDS ensures reliable data storage, EC2 instances facilitate server hosting and scaling, a load balancer optimizes performance, and an S3 bucket manages code and large data.

3 System Requirements

3.1 Baseline Requirements:

1. Users can create, sign in, and sign out of their accounts with an email and password.
2. Users can connect their preferred streaming service with the application for communication and library access.
3. Users can create content-based posts referencing songs, playlists, or music groupings.
4. Users can add posted music content to their streaming service account.
5. Users can search for other users, follow/subscribe, and view playlists and posts of other users.

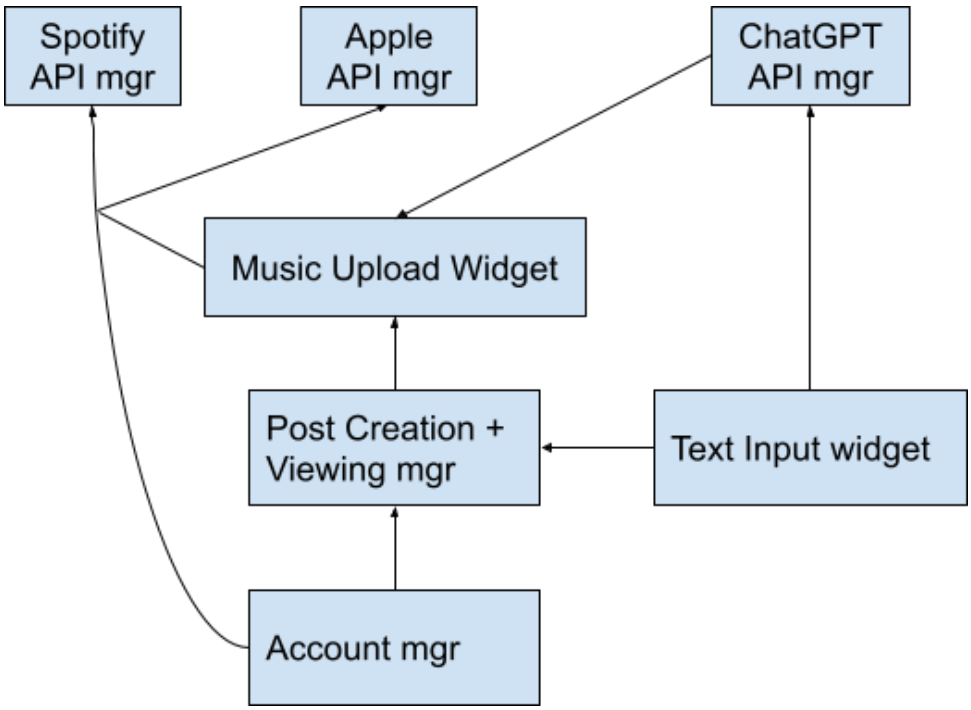
3.2 Non-Baseline Requirements:

6. Users can transfer portions of their music libraries between streaming services.
7. Users can request playlist generation based on natural language input.
8. Users can customize the app and their account via settings.
9. Users can join/create music communities with public/private options and admin privileges.

4 Open Source Libraries

- **Spotipy - Open Source Python Library:**
  - Usage: Acts as a wrapper for the Spotify API, enabling connection with the website.
  - Significance: Facilitates backend interaction with user streaming service accounts.
- **Apple Music Python - Open Source Python Library:**
  - Usage: Wrapper for the Apple Music API, enabling connection with the website.
  - Significance: Facilitates backend interaction with user streaming service accounts.
- **Openai - Open Source Python Library:**
  - Usage: Acts as a wrapper for the Chat GPT API.
  - Significance: Enables communication with OpenAI's language model for text-string playlist generation.
- **Django Python - Open Source Web Framework:**
  - Usage: Builds robust and scalable web applications using Python.
  - Significance: Provides a flexible framework for web development, ensuring efficient creation of dynamic websites.
- **React Native - Open Source Front-End Framework:**
  - Usage: Provides an interface for users to interact with the application's backend API.
  - Significance: Supports both mobile and web development, making the code compatible for web, iOS, and Android.

5 Content Diagram



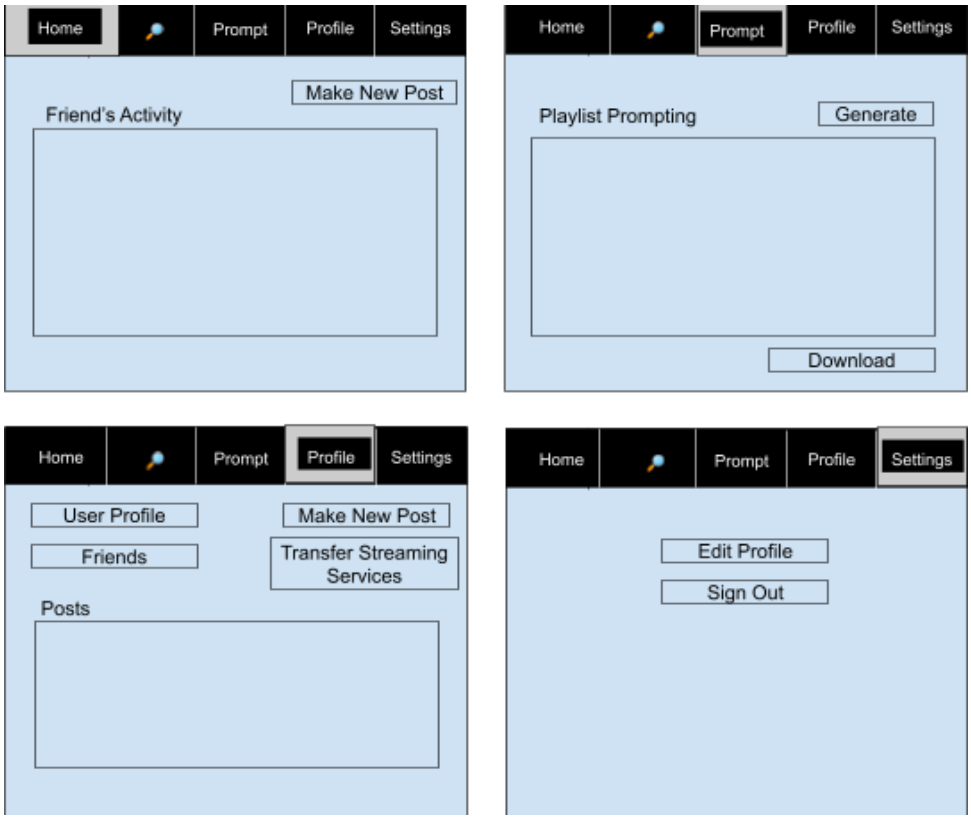
6 Unique Features

Elysium emphasizes cross-platform music sharing, providing a neutral space for music sharing, and includes NLP playlist generation. This distinguishes the app, addressing limitations in existing platforms and enhancing user experience.

7 Risk Mitigation

If unable to connect with other music streaming services, focus on leveraging the Spotify API. If beta testers are scarce, evaluate the project internally to ensure it adheres to the minimum viable product.

8 Rough Sketch of System



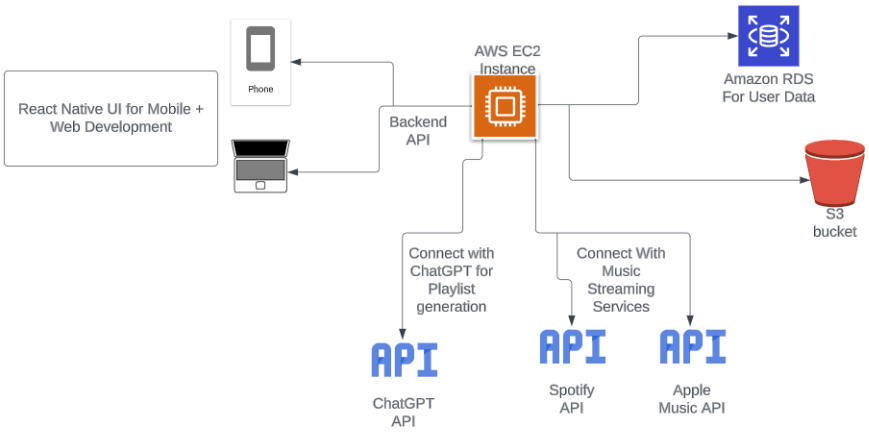
The user will interact with the application through a series of different task specific interfaces.

1. Prompt Interface - This will provide an interface similar to that of chatgpt for user prompt input, but it will provide a button that allows a user to download the entire playlist or individual components of the play list.
2. Home Media Feed:
   (a) Public Feed - Twitter-like interface for users to post songs, playlists, or commentary.
   (b) Make Post - Allows users to post songs, playlists, and commentary.
3. Search - Users can search and view profiles of other users.
4. Playlist Generation - Users can populate playlists from text input based on preferences.
5. Profile - Allows users to see post history, manage posts, playlists, and view profile activity.
6. Settings - Users can modify their profile and sign out.

All music displays will have a download button for users to download music to their platform regardless of the streaming service.

9 Building Blocks Needed

9.1 System Architecture



- Front End Server React Native - Manages the user interface, social media feed, playlist prompting.
- Back End Server Django - Communicates with external services, manages database interactions, implements algorithms.
- AWS RDS - Stores essential data such as user playlists, followings, followers, login credentials.
- AWS EC2 Instances - Hosts both front end and back end servers.
- AWS S3 Bucket - Stores the code base for EC2 instances and large data.

9.2 Considerations

Ensure robust security for user data in AWS RDS and regularly update algorithms for playlist generation. Monitor and adjust AWS infrastructure based on usage patterns for optimal performance and costs.

10 Competitive Analysis

10.1 Prominent Competitors

1. **Cross-Platform Music Sharing**
   (a) **Elysium**
      - Strengths: Focus on bridging the gap between users across various streaming services.
      - Weaknesses: Success depends on integration with multiple platforms.

   (b) **Competitors**
      - Last.fm: Primarily a music tracking service without direct cross-platform sharing.
      - Smule: Emphasis on karaoke without cross-platform music sharing.

   (c) Conclusion: Elysium stands out by addressing cross-platform sharing, a unique focus not found in competitors.

2. **NLP-Based Music Generation**
   (a) **Elysium**
      - Strengths: Unique feature with NLP-based music generation for personalized playlists.
      - Weaknesses: Success depends on the accuracy of the NLP algorithm.

   (b) **Competitors**
      - Last.fm and Smule: Lack features for playlist generation on the application.

(c) Conclusion: Elysium distinguishes itself by incorporating NLP-based music generation.

3. **Usability and Integration**

   (a) **Elysium**

   - Strengths: Cross-platform compatibility, familiar social media interface.
   - Weaknesses: Complexity in integrating with multiple streaming platforms.

   (b) **Competitors**

   - Last.fm: UI focused on new music discovery, lacks a scrolling social media aspect.
   - Smule: Simple UI but lacks social media features and integration with streaming platforms.

   (c) Conclusion: Elysium's choice of Django could contribute to a user-friendly experience.

4. **Cost Considerations**

   (a) **Elysium**

   - Strengths: EC2 instances are relatively inexpensive, scalable as the user base increases.
   - Weaknesses: Rapid expansion may cause AWS expenses to rise significantly.

   (b) **Competitors**

   - Last.fm, Smule, Playlist: Established competitors with larger resources.

   (c) Conclusion: While cost considerations are acknowledged, understanding competitor pricing models will be crucial.

10.2 Competetive Analysis Conclusion

While cost considerations are acknowledged, understanding competitor pricing models will be crucial.