Soundit

PatelNSattanapalleRChenJ

CS 5200 - Database Management Systems Project Report

Group Members: Raghu Ram Sattanapalle, Nicholas Patel, Jinpeng Chen

1. **README Section for Project Setup**

Introduction

"Soundit" is a music streaming application designed to deliver an enriching and interactive experience to music enthusiasts. This application allows users to explore a vast collection of music, create and manage playlists, follow their favorite artists, and enjoy a personalized music experience based on their preferences and listening history.

Purpose

The purpose of this README is to provide comprehensive instructions for setting up and running the Soundit application on any compatible system. It includes details on software requirements, installation steps, and how to initiate the application.

Software Requirements

- **MySQL Workbench 8.0:** A visual database design tool that integrates SQL development, administration, database design, creation, and maintenance. It's essential for setting up and managing the Soundit database. Download MySQL Workbench
- **Python:** As the primary programming language for backend development, Python offers the flexibility and libraries necessary for building robust applications like Soundit. Ensure that Python 3.x is installed. Download Python
- **Flask**: A lightweight WSGI (web server gateway interface) web application framework in Python, Flask is used to create the backend server for Soundit. It's known for its simplicity and fine-grained control. Install Flask via Python's package manager (pip).
- **Node.js and npm**: Required for running the JavaScript frontend. Download Node.js
- **Spyder IDE**: An open-source integrated development environment for Python, Spyder is used for developing the Soundit backend. It offers advanced editing, interactive testing, debugging, and introspection features. Download Spyder

Software Requirements

- **Python: 3.10.8 (Check your version using python --version in your Conda environment)**
- **Flask: 2.25 (Use python -m flask --version in your Conda environment to check)**
- **Node.js: v14.18.0**
- **npm: 9.4.0**
- **MySQL Workbench: 8.0**

Library Dependencies

Along with Python and Flask, Soundit utilizes several other libraries for its operation. These can be installed using Python's package manager, `pip`. The following command will install all the required libraries:

pip install Flask pymysql Werkzeug

These libraries are used for various functionalities within Soundit, including server setup, database connectivity, and password hashing.

Database Configuration

Update the database connection settings in the app_main.py file to match your local MySQL setup. This includes specifying the database name, user, password, and host details. Ensure these details correspond to the credentials used in MySQL Workbench.

Installation Instructions

1. **MySQL Database Setup**:
    - Install MySQL Workbench and open it.
    - Import the "Soundit-dump-2" MySQL dump file to create the Soundit_test_2 schema.
2. **Backend Setup (Python/Flask)**:
    - Install Python and use pip to install Flask and other necessary libraries.
    - Download the Soundit code and locate app_main.py.
3. **Frontend Setup (JavaScript/Node.js)**:
    - Check availability by typing node -v in the terminal. If it's not installed or you need to update, download it from the official Node.js website.
    - Navigate to the URL below and download Node.js for your operating system Node.js. Download the recommended version and install it on your local computer.nc
    - After downloading and running the installer with the default settings, restart your computer. Verify the installation by running node -v in the command line again.
    - Download the Soundit frontend code
    - Run npm install npx serve in the frontend project directory to install dependencies. You can navigate to the frontend project directory via your shell using Set-Location -Path "D:\DS Northeastern\CS 5200\Soundit-2\Soundit\frontend". Please replace the directory to your specific location for the frontend.

Running the Application

1. **Start the Backend Server**:
    - In the terminal, navigate to the Soundit backend directory.
    - Run python app_main.py to start the Flask server.
2. **Launch the Frontend Application**:
    - Open a new terminal window and navigate to the frontend project directory. (Set-Location -Path "D:\DS Northeastern\CS 5200\Soundit-2\Soundit\frontend". Please replace the directory to your specific location for the frontend.)
    - Run the command to start the frontend server ("npx serve .").
    - The frontend should now be accessible in a web browser. Please paste the local host line in your browser ("http://localhost:3000/")
3. **Accessing Soundit**:
    - Once both servers are running, access Soundit through the frontend interface.
    - Interact with the application to explore music, manage playlists, and enjoy personalized recommendations.

2. Technical Specifications

Architecture Overview

The Soundit application is structured as a client-server model where the client is a web page interacting with the Flask-based server. The server handles all backend logic, including database interaction, request processing, and response generation.

We decided that it would be best to emulate Spotify's web player by using a web app as our frontend. We decided to use Vue.js as our framework, and serve the files using npx so that the setup would be simpler.

For the "Soundify" project, MySQL is an ideal choice due to its scalability, adept at managing vast and varied data typical of music platforms. Its structured data management allows for intricate relationships, like linking users to playlists and artists to tracks. MySQL's seamless integration with diverse languages complements the group's technology stack of Javascript and Python. Additionally, tools like MySQL Workbench streamline tasks, while its robust transactional support ensures reliability for user subscription processes. Coupled with vast community resources, MySQL equips the team with advanced database design endeavors.

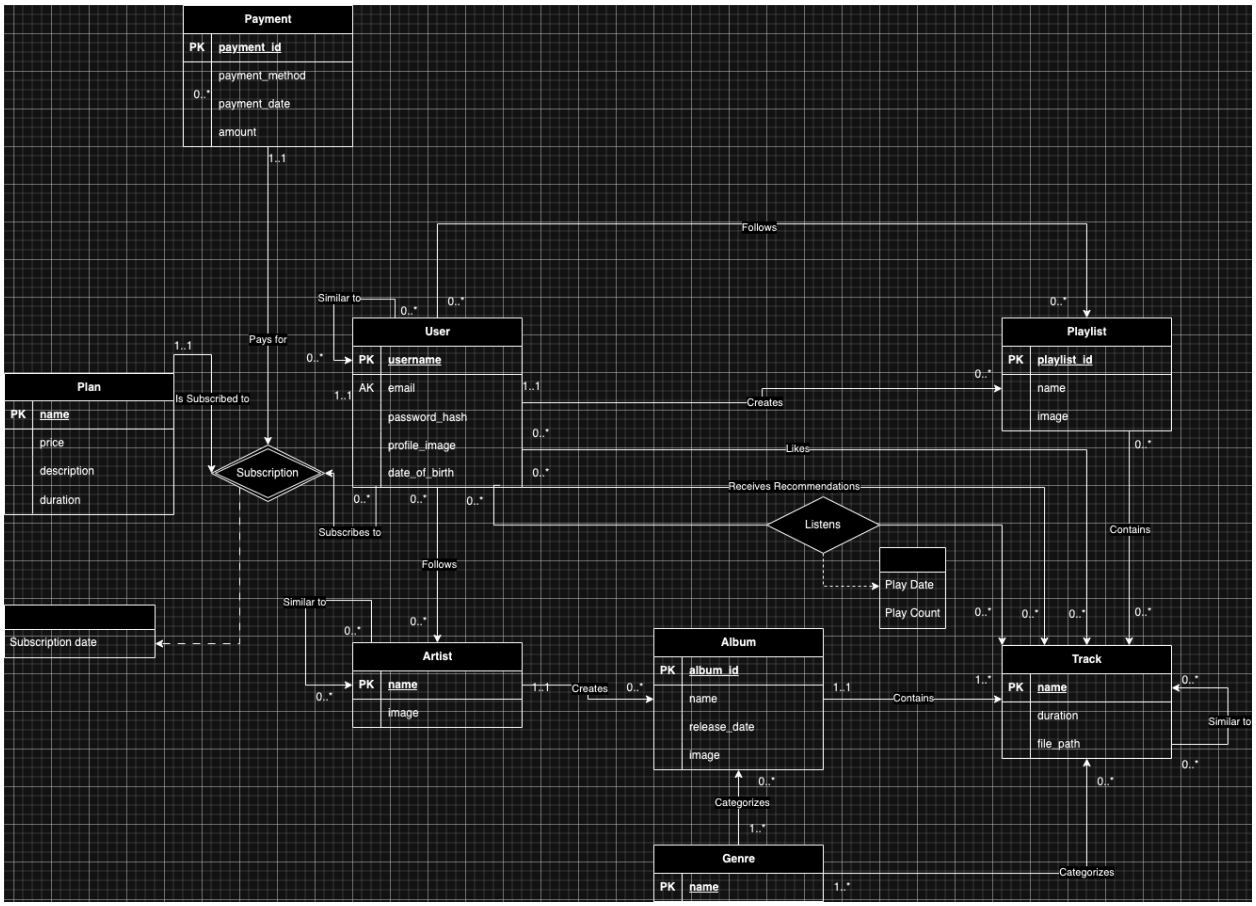
When first accessing the web page, you will be faced with a menu that has the option to sign up or login. There are some users provided in the database dump, but creating a new user is also simple. After logging in as a user, you may search for songs, albums, artists, or playlists using the search button, manage your current playlists by pressing the playlists button, manage your current plans using the plans button, or manage your account by clicking on your username.

Software Requirements

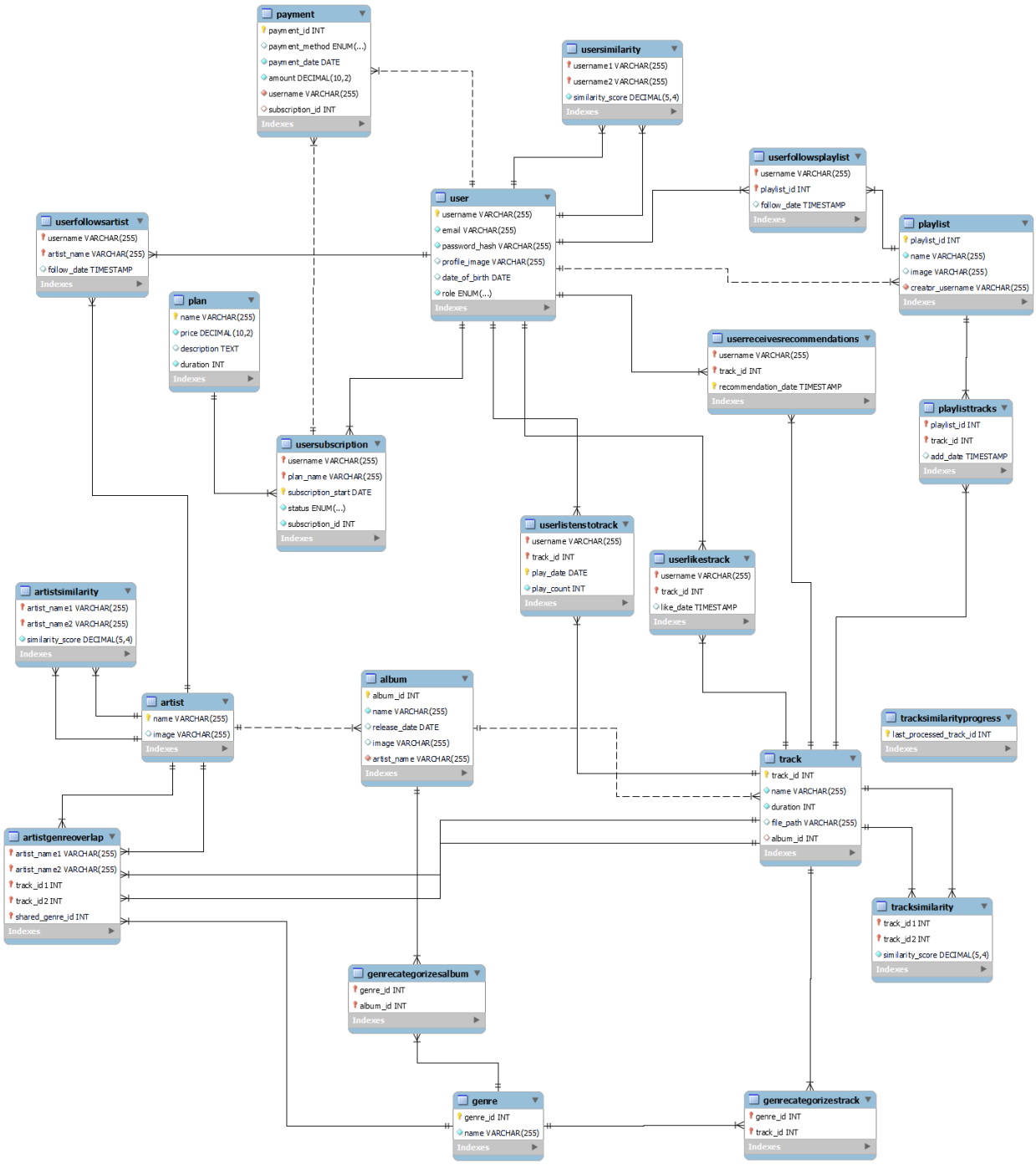
- **Python: 3.10.8 (Check your version using python --version in your Conda environment)**
- **Flask: 2.25 (Use python -m flask --version in your Conda environment to check)**
- **Node.js: v14.18.0**
- **npm: 9.4.0**
- **MySQL Workbench: 8.0**

2. Provide the Technical Specifications for the project.

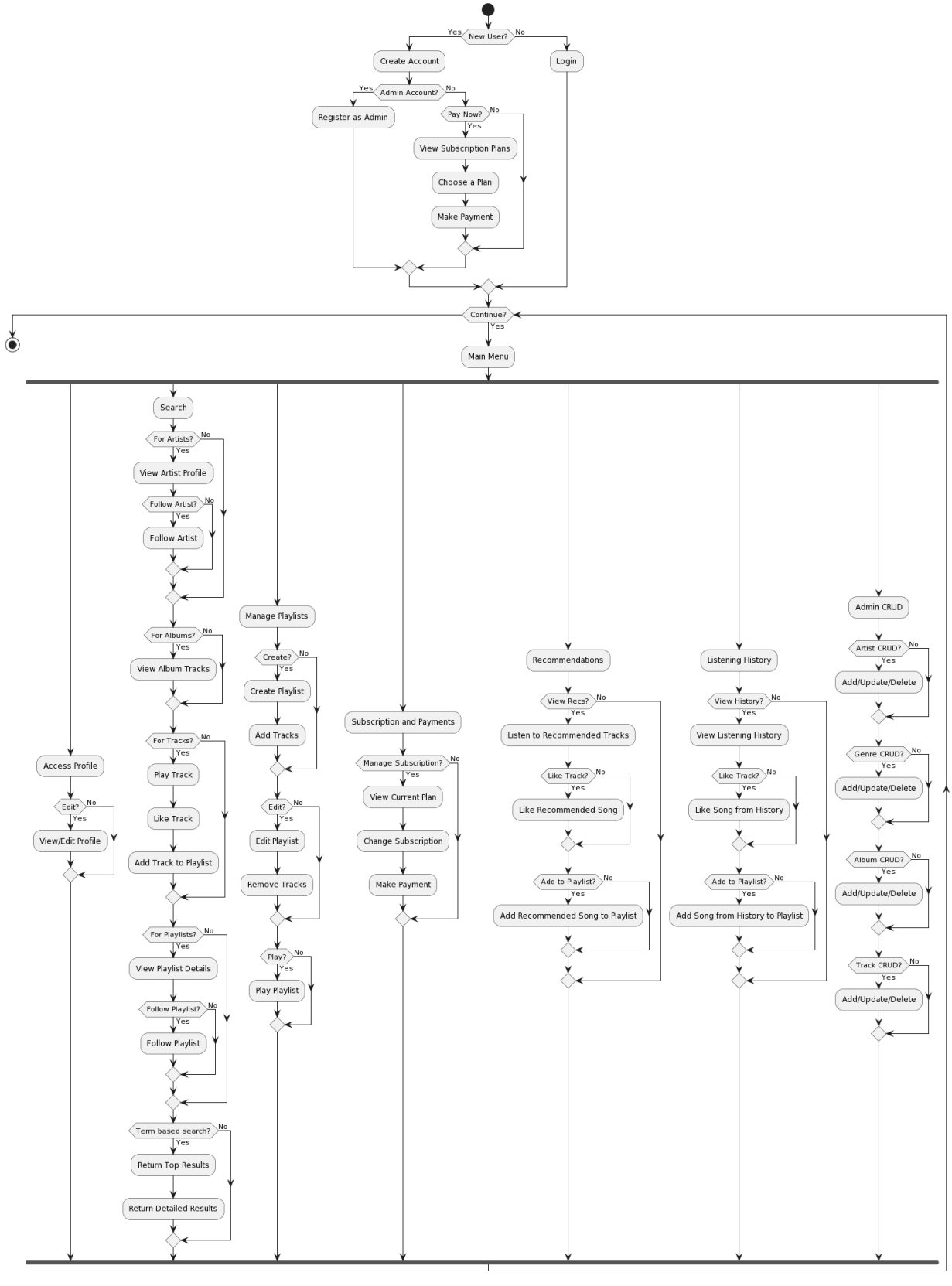
3. Provide the current conceptual design as a UML for the project.



4. Provide a logical design for the submitted database schema (Reverse Engineer your final schema in the MySQL workbench). If you are submitting a Mongo database, please provide a description of your collections.



5. Provide the final user flow of the system. List the commands or methods the user performs to interact with the system.


New User?
Yes
No
Create Account
Login
Admin Account?
Yes
No
Register as Admin
Pay Now?
No
Yes
View Subscription Plans
Choose a Plan
Make Payment
Continue?
Yes
Main Menu
Search
For Artists?
No
Yes
View Artist Profile
Follow Artist?
No
Yes
Follow Artist
For Albums?
No
Yes
View Album Tracks
For Tracks?
No
Yes
Play Track
Like Track
Add Track to Playlist
For Playlists?
No
Yes
View Playlist Details
Follow Playlist?
No
Yes
Follow Playlist
Term based search?
No
Yes
Return Top Results
Return Detailed Results
Access Profile
Edit?
No
Yes
View/Edit Profile
Manage Playlists
Create?
No
Yes
Create Playlist
Add Tracks
Edit?
No
Yes
Edit Playlist
Remove Tracks
Play?
No
Yes
Play Playlist
Subscription and Payments
Manage Subscription?
No
Yes
View Current Plan
Change Subscription
Make Payment
Recommendations
View Recs?
No
Yes
Listen to Recommended Tracks
Like Track?
No
Yes
Like Recommended Song
Add to Playlist?
No
Yes
Add Recommended Song to Playlist
Listening History
View History?
No
Yes
View Listening History
Like Track?
No
Yes
Like Song from History
Add to Playlist?
No
Yes
Add Song from History to Playlist
Admin CRUD
Artist CRUD?
No
Yes
Add/Update/Delete
Genre CRUD?
No
Yes
Add/Update/Delete
Album CRUD?
No
Yes
Add/Update/Delete
Track CRUD?
No
Yes
Add/Update/Delete

User account functionality (http://localhost:3000) :

Here is a detailed explanation of the commands or methods users can use to interact with our system:

1. **Login:**
    - Users click on the "Login" button to open the login modal.
    - They enter their username and password into the fields provided.
    - Submitting the form logs them into the system and fetches their role and profile information.
2. **Sign Up:**
    - New users can sign up by clicking the "Sign Up" button.
    - They must provide an email, username, password, and date of birth.
    - Once they submit the form, their account is created, and they are automatically logged in.
3. **Search:**
    - Users can perform searches by clicking the "Search" button to open the search modal.
    - They enter search terms into the input field and submit to retrieve results across different categories like albums, artists, tracks, playlists, and users.
4. **View and Follow Artists/Albums/Playlists:**
    - Users can view artists, albums, and playlists from the search results.
    - They can follow artists or playlists by clicking the corresponding "Follow" button next to each item.
5. **Listen and Like Tracks:**
    - Tracks can be played by clicking the "Listen" button next to them.
    - Users can like tracks by clicking the "Add to Liked Songs" button.
6. **Create and Manage Playlists:**
    - Users can create a new playlist by clicking the "Create a new playlist" button and entering the required information.
    - They can view, add songs to, and delete their playlists using buttons in the "Playlists" section.
7. **Subscription Plans:**
    - Users can view available subscription plans by clicking "View Plans".
    - They can subscribe by selecting a plan and providing a payment method.
8. **Profile Management:**
    - Users can manage their account by updating their email, username, profile image, and password.
    - They can view their listening history and manage their subscription.
9. **Logout:**
    - Logged-in users can log out of the system by clicking the "Sign Out" button.

Each of these interactions provides a direct and straightforward method for users to engage with the various functionalities of the Soundit system. The user flow is designed to be intuitive, with modal dialogs and instant feedback on actions such as errors or confirmations. This ensures a seamless and user-friendly experience.

The login details for a user account that we have generated and used for the project is:

Username: rrs44

Password: password

Admin account functionality (http://localhost:3000/admin_portal):

The second user interface focuses on administrative functionalities, providing a suite of tools for managing various aspects of the system. Here's a breakdown of the user flow for this interface:

1. **Admin Login:**
    - Admins log in using their username, password, and a registration key (DBMS).
    - Upon successful login, the admin is granted access to the system's administrative functions.
2. **Register New Admin:**
    - A new admin can be added by filling out a form with username, email, password, date of birth, and a registration key.
    - Submitting the form registers the new admin.
3. **Manage Artists, Genres, Albums, and Tracks:**
    - Admins can add, update, or delete artists, genres, albums, and tracks using respective modal dialogs.
    - Each action requires specific details such as name, image, release date, etc., depending on the item being managed.
4. **Create New Artist:**
    - Admins can create a new artist by providing the artist's name and image.
5. **Update Artist:**
    - Existing artist information can be updated by submitting the artist's new details.
6. **Delete Artist:**
    - Admins can delete an artist from the system.
7. **Manage Genres:**
    - Genres can be created, updated (including renaming), or deleted.
8. **Manage Albums:**
    - Admins can add new albums, update existing ones, or delete them. Album details such as ID, name, release date, image, and associated artist name are required.
9. **Manage Tracks:**

    - Tracks can be added, updated, or deleted. Necessary details include track ID, name, duration, file path, and album ID.
10. **Error and Information Handling:**
    - The system provides feedback through error messages and information strings, informing admins of the success or failure of their actions.
11. **Logout:**
    - Admins can log out of the system, which restricts access to administrative functionalities until they log in again.

This administrative interface is designed to give admins full control over the content and structure of the system, allowing them to maintain and update the database effectively. The user flow is straightforward, with modal dialogs guiding the admin through each process and providing instant feedback on their actions.

The login details for the administrative account that we have created is:

Username: admin

Password: password

Registration Key: DBMS

Lessons Learned

1. **Technical Expertise Gained**

MySQL Workbench Mastery: This project advanced our proficiency in MySQL Workbench. We learned to manage and manipulate database structures effectively, honing our SQL programming skills.

Advanced UML Skills: Developing detailed UML diagrams was pivotal for conceptualizing the database architecture. This skill was crucial for planning and structuring the database efficiently before commencing the coding phase.

Flask and PyMySQL Proficiency: Utilizing Flask and PyMySQL, we gained valuable insights into Python's capabilities in web development and database interaction. This experience was instrumental in creating secure, efficient backends that interact seamlessly with databases.

Security Implementation with Werkzeug: We focused on implementing robust security features, like password hashing using Werkzeug, underscoring the importance of protecting user data in web applications.

API Development with Flask: Developing RESTful APIs using Flask was a significant part of our project. This experience taught us how to handle complex queries and data manipulation, ensuring efficient communication between the frontend and backend.

Performance Optimization: We learned the importance of selecting the right tool for specific tasks, particularly for data-intensive operations. Moving computationally heavy tasks like similarity scoring from MySQL to Python due to performance issues was a key learning point.

Data Integration from External Sources: Utilizing the Spotify API for data collection and then importing it into our database via Excel files was a unique challenge. It taught us how to effectively integrate and manipulate large datasets from external sources.

Handling Sparse Data and Creative Problem-Solving: Faced with limited data for generating similarity scores and recommendations, we learned to creatively overcome these challenges through data synthesis and simulation.

In an ideal scenario with access to rich datasets, Soundit would employ robust data analysis methods to calculate artists and track similarity scores. Here's an insight into the procedures we would use:

Artist Similarity Calculation (Procedure outlined in MySQL):

1. We start by establishing cursors for iterating through each artist in our database.
2. For every pair of artists, we calculate the genre overlap by counting how many genres they share between all the tracks they have produced.
3. We also determine the total distinct genres associated with both artists to understand the breadth of their music styles.
4. The similarity score is then calculated as a ratio of the genre overlap count to the total genres.
5. This score is stored in the ArtistSimilarity table, creating a comprehensive map of artist relationships based on musical genres.

Track Similarity Calculation (Procedure outline in MySQL):

1. Similarly, cursors iterate through each track in the database.
2. For every pair of tracks, we again calculate the genre overlap.
3. The total number of distinct genres for the pair is calculated.
4. A ratio of genre overlap to the total genres is used to derive the track similarity score.
5. This score is updated in the TrackSimilarity table, providing a basis for recommending similar tracks to users.

User Similarity Calculation (Python):

1. We fetch user interaction data, such as tracks listened to, artists followed, and playlists followed.
2. The data is preprocessed and standardized to ensure uniformity for comparison.

3. We then use cosine similarity, a measure that determines the cosine of the angle between two non-zero vectors of an inner product space, to calculate the similarity between users. This metric helps us understand user preferences and patterns.
4. These similarity scores are updated in the UserSimilarity table, which is then used to generate user-specific recommendations.

In our project, due to data limitations, we used a simulated approach to demonstrate these functionalities. We randomly generated similarity scores to showcase our system's capabilities. However, the procedures described above illustrate our intended approach for a fully functional and data-driven music streaming platform.

In the Soundit project, recommendations are generated using a composite score based on user, track, and artist similarity. Here's a deeper look into the recommendation process as implemented and how it aligns with industry practices:

Implemented Recommendation Procedure:

1. The system initiates a loop through all tracks and their corresponding artists.
2. For each track, it calculates an average similarity score for the user based on their historical interactions, which may include listens, likes, and follows.
3. It then calculates an average similarity score for the track based on its relationship with other tracks.
4. An average artist score is also computed, reflecting the user's affinity towards the artist's work.
5. These scores are then weighted and combined into a composite score, which influences the track's likelihood of being recommended to the user.
6. The top tracks based on the composite score are stored as recommendations, with a limit to prevent an overwhelming number of suggestions.
7. This process is repeated for each user, generating personalized recommendations.

Understanding the Complexity of Streaming Services: The project gave us a glimpse into the complex functionalities that streaming services must manage. It made us appreciate the intricacies involved in such platforms, even though our project only scratched the surface of what these services entail.

Error Handling and Robustness: We made a concerted effort to include comprehensive error handling throughout the system. This approach not only improved the robustness of our application but also enhanced the overall user experience by anticipating and managing potential failures.

These points collectively reflect the range of technical skills and insights we gained throughout the project. The experience was not just about learning new technologies but also about understanding their practical application in a real-world context.

2. **Insights, time management insights, data domain insights, etc.**

Project Management and Time Allocation: One of the key insights gained was the importance of effective project management, particularly in terms of time allocation. Balancing various aspects of the project, such as database design, API development, and frontend considerations, required meticulous planning and execution. This project reinforced the value of setting realistic timelines and adhering to them to ensure a cohesive and functional end product.

Data Domain Complexity: Working with a music streaming service's database illuminated the complexity of the data domain. We learned about the nuances of handling diverse data types, such as user data, music tracks, artist profiles, and playlists. This experience highlighted the need for a well-structured database that can efficiently manage and relate different data entities.

Importance of Scalability and Flexibility: The project underscored the necessity of building scalable and flexible systems. As we worked with Python and MySQL, we realized the importance of designing a system that can adapt to increasing data volumes and evolving user requirements. This insight is particularly relevant in the fast-evolving field of digital streaming services.

Integrating External Data Sources: The process of integrating data from the Spotify API into our system provided valuable lessons in dealing with external data sources. It taught us about the challenges of API limitations, data formatting, and the need for effective data import strategies.

User Experience Considerations: Although our focus was primarily on backend development, the project brought to light the significance of frontend considerations and user experience. This includes understanding how backend decisions impact the frontend, ensuring responsiveness, and maintaining a user-friendly interface.

Error Handling and System Robustness: Our extensive work on error handling in both MySQL and Python components reinforced the importance of building robust systems. We learned that anticipating and planning for potential errors not only prevents system failures but also enhances user trust and reliability.

Data Security and Privacy: Implementing security features using Werkzeug was a crucial part of our learning. This experience highlighted the importance of safeguarding user data, adhering to privacy standards, and ensuring the ethical handling of sensitive information.

Collaboration and Communication: The project emphasized the need for teamwork and effective communication. Collaborating on various aspects of the project, from database design

to API integration to integration with the front-end, emphasized the need for clear, consistent communication among team members.

Adapting to Technological Changes: The project was a practical exercise in adapting to new technologies and methodologies. It highlighted the importance of being flexible and open to learning, especially in a field that constantly evolves.

Insights into the Music Streaming Industry: Finally, this project offered us a deeper understanding of the operational complexities of a music streaming service. It provided valuable insights into user behavior, content management, and the technological intricacies that underpin such services.

3. **Realized or contemplated alternative designs/approaches to the project**

Database Management System (DBMS) Alternatives: The choice of MySQL Workbench was pivotal for this project. However, contemplating the use of a NoSQL database like MongoDB could have offered a different approach. NoSQL databases are known for their flexibility in handling unstructured data, scalability, and faster querying for certain types of applications, particularly when dealing with large volumes of data or real-time analytics.

Frontend Development Approach: While a Node.js frontend is underway, exploring alternative frontend frameworks or libraries could offer varied user experiences. For instance, integrating a Vue.js frontend might provide a more intuitive and reactive user interface, enhancing user engagement.

User Authentication and Session Management: The project's focus on backend and database aspects meant limited emphasis on sophisticated user authentication and session management. Future iterations of the project could benefit from implementing OAuth or JWT for more secure and efficient user authentication processes, enhancing the overall security and user experience.

Data Analysis and Processing Techniques: The decision to shift certain calculations from MySQL to Python was a significant one. However, alternative data processing techniques or specialized analytical tools like Apache Spark could have been considered. Spark's ability to handle the large-scale data processing in real-time could have potentially improved the efficiency and accuracy of complex calculations like similarity scoring.

Error Handling and Debugging Approaches: The chosen methods for error handling and debugging were essential for the project's success. An alternative approach could involve more sophisticated error tracking and logging tools like Sentry. These tools can provide deeper insights into errors and system performance, aiding in more efficient debugging and optimization.

Project Scope and Focus: Since this is a DBMS class project, the primary focus was understandably on the backend and database aspects. However, reflecting on the project,

incorporating more frontend and user experience considerations could offer a more holistic understanding of application development. Even though it might not be central to the class's objectives, understanding how backend decisions impact the frontend can be crucial for real-world applications.

By considering these alternative approaches, we can appreciate the broad spectrum of choices available in software development and database management. Each choice comes with its own set of advantages and trade-offs, making it important to align them with the project's goals, resources, and constraints.

Challenges in Data Analysis and Simulation

- **Context and Limitations**: Our project's goal to implement sophisticated similarity scores was hindered by data limitations and API constraints, leading to a reliance on approximations and simulated scores.
- **Workaround with Random Generation**: To demonstrate system functionality, we generated random similarity scores. This approach, while not data-driven, allowed us to showcase the intended feature without the depth and accuracy of actual data analysis.
- **Practical Implications**: The necessity of this workaround underscores the importance of having access to comprehensive, varied datasets for meaningful data-driven features in digital platforms.
- **Reflection on Real-World Implementation**: A true implementation would require extensive data collection, advanced analytical techniques, and regular data updates to reflect user preferences and content dynamics.

Documenting Code Limitations

Simulated Similarity Scores: Due to the challenges in obtaining robust data, the similarity scores in our system were generated randomly. This method was primarily for demonstration purposes and not indicative of real analytical outcomes.

Recommendations for new users: The design is set up such that recommendations and similarities are generated at midnight (12:00 AM) everyday. Hence, a new user who registers with the service will not see any recommended tracks as soon as they login. You can see this happening with the account that we give you for testing (Username: rrs44, Password: password). The similarity score procedure for artists, the python scripts for track and user similarity will have to be run each time a new artist, track, or user is added respectively for it to function as designed. We have created MySQL events that would automatically run at midnight everyday for the artist similarity and recommendation procedures. However, since the track and user similarity are calculated and injected into the database these two scripts would have to be automated via Windows Event Scheduler or cron for a Linux/Mac server system.

4. **Document any code not working in this section**

There isn't any code that isn't necessarily working. However, as discussed above we had to create a workaround for the similarity scores and recommendation parts of the project. At the same time the similarity and recommendations for newly added artists, tracks, users, and then an update of the tracks recommended to a user is not automated. So if you add any of these via adminCRUD or if you register a new user and want recommendations, please run the procedures related to these on your end.

Future Work for Soundit

Planned Uses of the Database:

1. **Advanced Analytics and User Insights:**
    - Develop detailed analytics capabilities in the database to track user behaviors, preferences, and engagement metrics. This includes analyzing listening patterns, genre preferences, and user interactions within the app.
2. **Enhanced Content Management:**
    - Expand the database to include more diverse metadata for tracks and artists, allowing for richer content curation and exploration. This could involve tagging songs with mood, instruments used, and lyrical themes.
3. **Dynamic User Recommendation System:**
    - Utilize the database to build a dynamic recommendation system. This system would analyze user data and listening history to suggest personalized playlists and new artists. Part of this future work includes refining our approach to automating the updates of similarity scores. As new users, tracks, and artists are added, the system should dynamically update the scores to reflect these changes, enhancing the accuracy and relevance of the recommendations provided.
4. **Community and Social Interaction Data:**
    - Implement data structures to support community features, such as user comments, ratings, and playlist sharing, to foster a more interactive and social user experience.

Potential Areas for Added Functionality:

1. **AI-Driven Music Discovery Tools:**
    - Introduce AI algorithms for mood-based and activity-based playlists, using machine learning to analyze song attributes and user preferences for more intuitive music discovery.
2. **Mobile Application Development:**

- Develop a mobile application to make the platform more accessible, focusing on user-friendly design and seamless cross-device synchronization.

3. **Smart Device Integration:**
   - Expand the platform's functionality to integrate with smart home devices and IoT ecosystems, such as smart speakers and wearables, for a more versatile music streaming experience.
4. **Enhanced Security and Privacy Measures:**
   - Implement advanced security protocols and privacy measures, including sophisticated encryption and multi-factor authentication, to protect user data.
5. **User Feedback and Community Engagement:**
   - Develop features for users to provide feedback and suggestions directly within the app, using this data to guide future updates and features.
6. **Business Intelligence and Market Analysis:**
   - Leverage the database for business intelligence purposes, analyzing market trends, user demographics, and content performance to inform business strategies.
7. **Real-World Recommendation Engines:** In a commercial setting, recommendation engines are far more complex and data-driven. They often involve machine learning models that can process vast amounts of data in real time, adapting to new user behavior and content. These systems may use collaborative filtering, content-based filtering, or a hybrid approach combining both methodologies to achieve high accuracy and personalization.
   - **Collaborative Filtering:** This technique makes automatic predictions about a user's interests by collecting preferences from many users. It assumes that if users agree on the quality of certain items, they will likely agree on others as well.
   - **Content-Based Filtering:** This method uses item features to recommend additional items similar to what the user likes, based on their previous actions or explicit feedback.
   - **Hybrid Systems:** By combining collaborative and content-based filtering, hybrid systems can provide more reliable recommendations than either approach alone, especially when dealing with sparse datasets.
   - **Future Considerations:** For Soundit, moving towards such sophisticated algorithms would require:
     1. Accumulating more detailed user data for a nuanced understanding of preferences.
     2. Implementing real-time analytics to adapt recommendations to current user activity.
     3. Integrating machine learning to uncover patterns not immediately apparent to rule-based logic.
     4. Consider context-aware recommendations that take into account the time of day, device used, or current user activity.

The potential for incorporating more advanced machine learning models such as deep learning is also significant. These models can capture the complexity of music tastes and preferences in a way that far surpasses the capabilities of simpler, rule-based systems.

By addressing these aspects, Soundit can significantly enhance its service offerings, user engagement, and market position. The planned uses of the database focus on harnessing data for insightful analysis and personalization, while the potential areas for added functionality aim to expand the service's reach, improve user experience, and ensure data security.

Bonus Point Sections

Using the Spotify API to scrape data

The Spotify API has fantastic documentation; However, in order to get the necessary amount of data from the API we had very few tries for testing, since we were limited to a total of 500 queries per month. Fortunately, we were able to find some methods for reducing our number of queries by using some of the more complex queries (such as getting several albums for a given artist at once, rather than querying each album separately). Finally, we decided to save the JSON bodies of the responses so that testing the code to clean the responses would not require additional queries to the Spotify server, further saving our query count. In the end, we had queries to spare and could easily add substantially more data to the database at any time.