Huddle recommendation system implementation plan

1. Requirements and Implementation ideas:

Base on the task 3 description, the functional requirements are listed as below:

Personalized Recommendations:

- Build a Recommendation Engine to suggest artists using user interaction data (likes, listens, skips) and collaborative filtering.
- Display personalized suggestions in a "For You" section prominently on the homepage.

Discover Section:

- Create a dedicated Discover page or tab featuring curated content such as trending, similar, and random artists.
- Add a “Surprise Me” button for serendipitous exploration of new profiles.

Social Proof Features:

- Introduce a "Trending Among Friends" section that shows artists popular with the user’s network.
- Include “Users Like You Also Listened To” recommendations to foster relatability and curiosity.

Implementation Ideas for the recommendation system:

- Use machine learning models to create vector embeddings that represent artists and users in a high-dimensional space. Then, we store embeddings about song features like tempo, key, pitch, energy,... in a vector database like Pinecone and combine embeddings with explicit factors (e.g., genres, description, lyrics, popularity) and behavioral factors (e.g., likes, shares) to get the most effective recommendation for the user.
- All that data must work seamlessly with the user data and their network data (e.g., friends, family, workplace, school to create the best interaction performance for users.
- It may be resource-intensive for smaller applications. Consider simpler collaborative filtering techniques initially. We can try using simple data like Artist

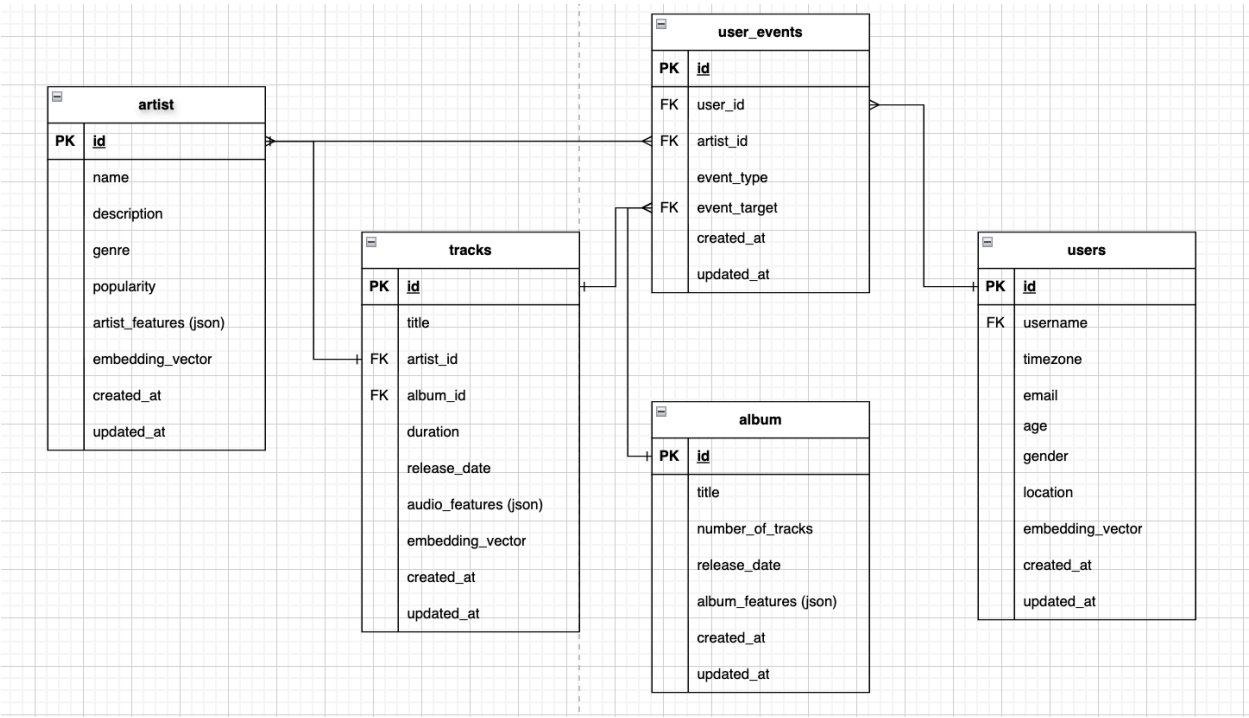
Popularity, Artist Similarities, and Users Like You Also Listened To tables to leverage social proof and explore similar artists

2. Database Design:

To implement a robust recommendation engine, we'll need to store and efficiently query data related to:

- Users: User IDs, demographics, preferences, listening history, etc.
- Artists: Artist IDs, genres, similar artists, popularity, etc.
- Tracks: Track IDs, artists, album, genres, audio features, etc.
- User-Item Interactions: User-artist interactions (likes, plays, skips), user-track interactions, timestamps, etc.

Database Schema Enhancements:



Users:

- user_id (PK)
- username
- timezone
- email

- age
- gender
- location
- embedding_vector
- created_at
- updated_at

Artists:

- artist_id (PK)
- name
- description
- genre
- popularity
- artist_features (json)
- embedding_vector (array or JSONB)
- created_at
- updated_at

Tracks:

- track_id (PK)
- title
- artist_id (FK)
- album_id (FK)
- duration
- release_date
- audio_features (JSON)
- embedding_vector (array or JSONB)
- created_at
- updated_at

User_events:

- id (PK)
- user_id (FK)
- artist_id
- item_id (FK: can be artist_id or track_id)
- event_type (like, play, skip, etc.)
- event_target (FK)
- created_at

- updated_at

Additional Considerations:

Performance: Optimize database queries, indexing, and caching to ensure efficient retrieval of data for recommendation generation.

3. Frontend Components:

Track Recommendation Panel

- Purpose: Display personalized artist suggestions.
- Key Features:
  - Cards/Grid View: Showcase track with images, genres, and short descriptions matching with user current embeddings
  - Action Buttons: Enable interactions like "Follow," "Listen Now," or "Share."
  - Dynamic Updates: Reflect real-time changes based on user preferences and trends.

Interactive Search Bar

- Purpose: Allow users to search for artists or explore recommendations interactively.
- Key Features:
  - Autocomplete Suggestions: Display top recommendations as the user types.
  - Context-Aware Queries: Enable natural language input like, "Artists similar to [Artist X]."
  - Query History: Show previous searches for quick access.

Trending Artists Section

- Purpose: Highlight popular or emerging artists based on trends.
- Key Features:
  - Dynamic Updates: Fetch trending data in real-time.
  - Filters: Allow users to filter by genre, region, or time (e.g., weekly/monthly trends).
  - Compact Cards: Show minimal details to quickly scan through trending artists.

Visualization Dashboard

- Purpose: Show user interaction history and recommendation patterns.
- Key Features:
  - Charts and Graphs: Visualize listening habits, favorite genres, and artist popularity.
  - Customizable Views: Let users toggle between daily, weekly, or monthly data.
  - Integration with Recommendations: Provide links to discover related artists.

Feedback System

- Purpose: Gather user feedback to improve recommendations.
- Key Features:
  - Feedback Form: Prompt users to rate recommendations or provide input.
  - Inline Feedback: Add thumbs-up/thumbs-down buttons to each recommendation.
  - Learning Integration: Update the RAG system based on feedback to refine suggestions.

4. Backend Development:

Database Setup:

- Choose a database (like PostgreSQL or MongoDB).
- Design a schema for users, artists, tracks, and interactions.

Data Collection:

- Gather data on users, artists, tracks, and their interactions.
- Clean and preprocess the data.

Feature Engineering:

- Extract relevant features like genres, moods, and audio features.
- Convert categorical features into numerical representations.

Vector Database Integration:

- Use a vector database (like Pinecone or Weaviate) to store embeddings.
- Create embeddings for artists and tracks.

Recommendation Engine:

- Choose a recommendation algorithm (collaborative filtering, content-based, or hybrid).
- Train the model on the prepared data.

API Development:

- Create an API to handle user interactions and recommendation requests.
- Use a framework like Flask, Django REST Framework..

Deployment:

- Deploy the backend to a server or cloud platform.
- Monitor performance and update the model as needed.

Continuous Improvement:

- Collect user feedback to refine the recommendations.
- Experiment with different algorithms and features.

5. Potential Problems

Data Quality and Quantity:

- Challenge: Insufficient or low-quality data can hinder the accuracy of recommendations.
- Solution:
    - Data Cleaning: Implement robust data cleaning techniques to handle missing values, outliers, and inconsistencies.
    - Data Augmentation: Use techniques like data augmentation to generate synthetic data.
    - Feature Engineering: Create meaningful features from raw data to improve model performance.

Cold Start Problem:

- Challenge: Recommending for new users with limited interaction history.

- Solution:
    - Content-Based Filtering: Recommend items based on their content similarity.
    - Popularity-Based Recommendations: Recommend popular items to new users.
    - Hybrid Approach: Combine content-based and collaborative filtering techniques.

Real-time Updates:

- Challenge: Keeping the recommendation system up-to-date with real-time user interactions.
- Solution:
    - Incremental Learning: Update the model incrementally with new data.
    - Real-time Inference: Use efficient inference techniques to generate recommendations in real-time.
    - Caching: Cache frequently accessed data to reduce latency.