Software Design Document (SDD)

AI Music Recommender System

Snapshots 1–4

Version 4.0

Project Group 5

David Nazaryan, Eric Castellon, Gustavo Trejo, Rojina Zalzar

December 7, 2025

Contents

Version Description

Version	Description	Date
1.0	Initial SDD for Snapshot 1 of the AI Music Recommender System. Describes base architecture, core components, and database design.	December 7, 2025
2.0	SDD updated for Snapshot 2. Extends architecture and database design for mood-based recommendations.	December 7, 2025
3.0	SDD updated for Snapshot 3. Adds social graph (follow relationships) and collaborative recommendation logic.	December 7, 2025
4.0	Final SDD for Snapshot 4. Adds Quick Mix flow, performance refinements, and design notes for future work.	December 7, 2025

1 Snapshot 1 – Core Design

1.1 Introduction

1.1.1 Purpose of the Document

This Software Design Document describes the architecture and internal design of the AI Music Recommender System for Snapshot 1, mapping the SRS requirements to implementable components that generate personalized song recommendations based on user interactions and preferences.[file:2]

1.1.2 Scope

Snapshot 1 includes a client application, backend services, a simple recommendation engine, and a relational database for user, catalog, and interaction data that support authentication, playback, interaction logging, and baseline recommendations.[file:2]

1.1.3 Intended Audience

The audience includes the course instructor and teaching assistants, Project Group 5 members responsible for implementation, and any future developers or maintainers who need to understand how the system is structured.[file:2]

1.2 System Architecture

1.2.1 Workflow of the System

1. The user logs into the client application.
2. The user plays songs and provides feedback such as likes or skips.
3. The client sends interaction events to the backend via REST APIs.
4. The backend stores interactions and invokes the recommendation engine.
5. The recommendation engine scores candidate songs and returns top recommendations.
6. The backend returns the recommended list to the client, which displays it to the user.

[file:2]

1.2.2 Components

Client-Side

- User interface for login and registration.
- Home screen displaying recommended songs and playlists.
- Search screen for finding songs by title, artist, or genre.

- Now-playing view with playback controls and feedback buttons such as like and skip.

[file:2]

Server-Side

- Authentication module for user login and session management.
- REST API layer for handling requests from the client.
- Recommendation service that prepares features, invokes algorithms, and formats recommendation results.

[file:2]

Data Storage

- Database for user accounts, songs, artists, and interaction logs.
- Optional tables for playlists and pre-computed recommendation data.

[file:2]

1.3 Database Design

1.3.1 Conceptual Schema

A simple conceptual schema for Snapshot 1 includes:

- User(userId, name, email, passwordHash, createdAt).
- Track(trackId, title, artistId, album, genre, duration).
- Artist(artistId, name).
- Interaction(interactionId, userId, trackId, actionType, timestamp).
- Playlist (optional)(playlistId, userId, name).

[file:2]

1.3.2 Relationships

- One User to many Interactions.
- One Track to many Interactions.
- One Artist to many Tracks.
- One User to zero or many Playlists.

[file:2]

1.4 User Interface Design

1.4.1 How to Use the System

1. Users open the application and either register for a new account or log in with existing credentials.
2. After logging in, users are taken to the home screen where they can see recommended songs and start playback.
3. Users can like or skip songs and use the search screen to find specific tracks, artists, or genres, and their actions influence future recommendations.

[file:2]

1.5 API Endpoints (Logical)

Example logical endpoints for Snapshot 1:

- POST /auth/register
- POST /auth/login
- GET /songs/recommended
- GET /songs/search
- POST /interactions

[file:2]

2 Snapshot 2 – Mood-Based Design Updates

2.1 Introduction

2.1.1 Purpose

This snapshot updates the Snapshot 1 design to incorporate mood-based recommendations, including new UI elements, backend logic, and database structures that handle mood metadata and mood preferences.[web:8][web:12]

2.1.2 Scope

Snapshot 2 modifies the recommendation workflow to accept a mood parameter, adds storage for mood preferences, and updates the client to support mood selection while reusing the core architecture.[web:8]

2.2 Architecture Changes

2.2.1 Updated Workflow

1. The user logs in and views the home screen.
2. The user selects a mood (e.g., Happy, Chill, Focus).
3. The client calls the recommendation API with the user identifier and selected mood.
4. The backend queries song features or tags associated with that mood and the user's interaction history.
5. The recommendation engine ranks candidate songs using a mood-aware scoring function and returns a playlist.
6. The user plays songs and provides like or skip feedback, which is stored with the active mood label.

[web:8]

2.3 New and Extended Components

2.3.1 Client Application

- Adds a mood selector UI element on the home screen.
- Displays a "Mood Playlist" section populated by mood-aware recommendations.

[web:8]

2.3.2 Backend Services

- Extends the recommendation API to accept an optional mood parameter.
- Adds a mood preference handler that records the user's selected mood.

[web:8]

2.3.3 Recommendation Engine

- Incorporates mood metadata (e.g., mood labels, energy, valence) into the ranking algorithm.
- Applies filters and boosts for songs whose mood attributes align with the selected mood.

[web:8][web:12]

2.4 Database Design Updates

2.4.1 New and Modified Tables

- MoodPreference(userId, mood, selectedAt).
- Optional mood-related attributes added to Track, such as moodLabel or energy/valence fields.
- Interaction table extended with an optional mood column to record the active mood at the time of interaction.

[web:8]

2.4.2 Example Relationships

- One User to many MoodPreference records.
- One Track may be associated with one or more mood labels via attributes or auxiliary tables.

[web:8]

2.5 Interface Design

2.5.1 API Changes

- GET /songs/recommended?mood={mood}: returns a mood-aware recommendation list.
- POST /moods/select: stores the user's selected mood for the current session.

[web:8]

2.6 Design Considerations

- Mood-based logic is designed to be modular, allowing additional moods or a more advanced mood detection model in future snapshots.
- Default behavior falls back to Snapshot 1 recommendations when no mood is selected.

[web:8][web:11]

3 Snapshot 3 – Social and Collaborative Design

3.1 Introduction

3.1.1 Purpose

This snapshot extends the design to include social and collaborative recommendations, combining user–user relationships with existing content and mood-based approaches.[web:12]

3.1.2 Scope

Snapshot 3 adds follow and unfollow operations, friends' recommendations, and collaborative algorithms while reusing existing modules from earlier snapshots.[web:12]

3.2 Architecture Changes

3.2.1 Updated Workflow

1. The user follows one or more other users through the client.
2. The client sends follow and unfollow requests to the backend social service.
3. The social service stores follow relationships in the database.
4. When the user opens the "Friends' Picks" section, the client requests collaborative recommendations.
5. The backend retrieves followed users and their liked tracks, optionally incorporating similar-user data, and returns a ranked list.

[web:12]

3.3 New and Extended Components

3.3.1 Social Service

- Manages follow and unfollow operations.
- Provides APIs to fetch a user's follow list and followers.

[web:12]

3.3.2 Collaborative Recommendation Module

- Aggregates liked songs from followed users.
- Optionally computes similarity scores between users based on interaction patterns.
- Combines collaborative scores with existing content and mood scores in a hybrid ranking model.

[web:12]

3.4 Database Design Updates

3.4.1 New Tables

- Follow(userId, followsUserId, createdAt).
- Optional UserSimilarity(userId, otherUserId, similarityScore) for precomputed similarity.

[web:12]

3.4.2 Relationships

- A User can follow many other Users via Follow records.
- UserSimilarity allows fast lookup of similar users for collaborative filtering.

[web:12]

3.5 Interface Design

3.5.1 API Changes

- POST /social/follow
- POST /social/unfollow
- GET /social/following
- GET /songs/friends-picks

[web:12]

3.6 Design Considerations

- Social features are designed as a separate module to isolate privacy and policy concerns.
- The hybrid approach allows combining content-based, mood-based, and collaborative scores for better recommendations.

[web:12]

4 Snapshot 4 – Final Architecture Refinements

4.1 Introduction

4.1.1 Purpose

This snapshot refines the design by adding a Quick Mix feature, performance-related improvements, and architectural notes that support future enhancements.[web:11]

4.1.2 Scope

Snapshot 4 does not introduce a major new subsystem but extends existing modules and clarifies design decisions for maintainability, performance, and future growth.[web:11]

4.2 Architecture Updates

4.2.1 Quick Mix Workflow

1. The user triggers Quick Mix from the main screen.
2. The client calls a Quick Mix endpoint with the user identifier and context (e.g., current mood, recent interactions).
3. The backend collects candidate tracks from content-based, mood-based, and collaborative sources.
4. The recommendation engine combines these sources into a hybrid ranked playlist.
5. The backend returns the playlist, and the client starts playback.

[web:11][web:12]

4.3 Caching and Optimization (Optional)

- Frequently requested recommendation lists (e.g., top mood playlists) may be cached briefly to reduce load.
- Database indexes may be added on key fields (userId, trackId, actionType, mood, follow relationships) to improve query performance.

[web:11]

4.4 Interface Design

4.4.1 API Additions

- GET /songs/quick-mix: returns a hybrid playlist combining multiple recommendation strategies.
- Optional endpoints may expose diagnostic data for debugging recommendation issues for internal use only.

[web:11]

4.5 Design Considerations and Future Work

4.5.1 Design Decisions

- The system uses a hybrid design combining content-based, mood-based, and collaborative filtering to leverage strengths of different recommendation techniques.
- Separation of concerns is maintained through distinct modules for recommendation, mood handling, and social features.

[web:11][web:12]

4.5.2 Future Work Support

- The architecture is designed to accommodate more advanced machine learning models and context-aware features without major structural changes.
- Additional services can be added, such as analytics or an explanation service, using the same API-driven modular approach.

[web:8][web:11]

Glossary

Acronym	**Meaning**
UI	User Interface
API	Application Programming Interface
DB	Database
SRS	Software Requirements Specification
SDD	Software Design Document
ML	Machine Learning

References

- Articles and papers on mood-based and AI-powered music recommendation systems that discuss content filtering, collaborative filtering, and hybrid approaches.[web:8][web:11][web:12]
- General resources on software design documents, test planning, and system architecture used to shape the structure of this SDD.[web:12]