Music Mind

Date: November 9, 2025

SUPERVISOR:

Ms. Beenish Urooj

GROUP MEMBERS:

Syed Hamza Mukhtar	2023682
Riyan Khan Durrani	2023611
Arsalan Khalil	2023130
Adan Aamir	2023057
Momina Zahid	2023335

List of Contents

List of Tables

List of Figures

PART-I

Project Requirement Specifications

Music Mind Date: 9/11/2025

Revision History:

Revision History	***Date***	***Comments***
1.00	2025-10-10	Initial Draft (Part 1)
2.00	2025-10-30	Major edits
3.00	2025-11-08	Final Draft (Part 2)

Document Approval:
The following document has been accepted and approved by the following:

Signature	***Date***	***Name***
QASIM	2025-11-09	Mr. Qasim Riaz

1. INTRODUCTION

PURPOSE

This document defines the Software Requirements Specification (SRS) Part 1 — for MusicMind, a web-based recommendation application that suggests song names to users based on personality traits collected through a questionnaire. The SRS Part 1 captures the system goals, user groups, primary functions, external interfaces, functional requirements, and nonfunctional requirements required for an initial, reviewable specification suitable for the course assignment.

PRODUCT SCOPE

MusicMind is a lightweight music suggestion system that:

- Collects personality trait data via a questionnaire.
- Uses the questionnaire responses to generate personalized song suggestions.
- Shows suggestions and top songs on the user dashboard once the questionnaire has been filled at least once.
- Provides an admin interface to add, update, delete songs and view user statistics.
- Integrates with Firebase for authentication/document storage and uses PostgreSQL as the main DB (hosted locally).

Primary goals:

- Provide accurate, personality-based song suggestions.
- Offer a simple admin management console for the song catalog.
- Keep the system easy to deploy and maintain using Flask + PostgreSQL + Firebase.

Table 1 Terms used in this part and their descriptions

Name	Description
MusicMind	The web-based music recommendation application being developed.
User	Any authenticated listener using MusicMind to get song suggestions.
Admin	Privileged user who manages the song catalog and user statistics.
Dashboard	The main interface shown after login; contains links to key features.
Questionnaire	A form that collects personality traits for generating recommendations.
Recommendation	Suggested song names based on the user's questionnaire responses.
Firebase	Cloud platform used for authentication and document storage.
PostgreSQL	Local relational database used to store user and song data.
Graveyard	Admin-only page where deleted songs are listed.
Flask	Python-based micro web framework used for backend logic.

2. OVERVIEW

THE OVERALL DESCRIPTION

MusicMind is a Flask-based web application. Users log in (Firebase authentication), fill out a personality questionnaire, and receive song suggestions. Admins manage the song catalog with CRUD operations and view system statistics. The app uses a local PostgreSQL server for persistent storage and Firebase (admin SDK) for authentication and optional cloud document storage.

PRODUCT PERSPECTIVE

MusicMind is a standalone recommendation web app but can be considered a prototype for integration with larger music services. It is not intended to stream music or provide licensed song content — only song *names* and metadata (artist, genre, short description). It integrates with:

- Local PostgreSQL (primary DB)
- Firebase (auth + document store)
- Flask server (backend)
- HTML/CSS/JS frontend rendered with Jinja2 templates

PRODUCT FUNCTIONS

- User registration/login and session handling (via Firebase + Flask-Login).
- Personality questionnaire collection and storage.
- Recommendation engine that maps personality traits → suggested song names.
- Dashboard view showing questionnaire, recommendations, and top songs.
- User profile page showing stored personality traits and past suggestions.
- Admin dashboard: add song (“Drop the Beat”), update/delete song (“SoundTracker”), view deleted songs (“Graveyard”), view user count (“Spy on Fans”).
- Firebase document creation and display for teacher-requested feature (admin-only document editor).

USER CHARACTERISTICS

- **End User (Listener):** Typical web user, basic computer literacy. Uses app to get song recommendations.
- **Admin:** More technical; manages song catalog and views user stats. Needs CRUD competence.
- **Assessor / Instructor:** Evaluator of the project; accesses the site as an admin or user to verify features.
  All users are expected to know basic web navigation and have internet access for Firebase authentication.

CONSTRAINTS

- System requires Python 3.x compatible environment and packages listed in requirements.txt.
- Requires a running PostgreSQL server (local) accessible to the application.
- Requires Firebase project credentials for authentication and Firestore operations.
- Hosted on local server (for development); not currently deployed to a public cloud in Part 1.
- Storage size, memory, and CPU depend on the host machine used for local server — heavy datasets may slow response.

ASSUMPTIONS AND DEPENDENCIES

- Users will have an internet connection for Firebase login (but core DB is local).
- The PostgreSQL server is installed and correctly configured before running the app.
- Environment variables (DB URL, Firebase credentials) are stored in .env and loaded using python-dotenv.
- The recommendation logic is deterministic based on questionnaire mapping (no complex ML model required for Part 1).
- The project uses the Python packages and versions listed in requirements.txt (Flask, SQLAlchemy, firebase-admin, etc.).

3. STATE OF THE ART

- LITERATURE REVIEW
- EXISTING SYSTEMS

Literature Review

Music recommendation systems have evolved significantly over the past two decades, with most modern platforms using either *collaborative filtering*, *content-based filtering*, or hybrid approaches. However, almost all established systems rely heavily on user listening history, which MusicMind intentionally avoids. Instead, MusicMind uses a personality questionnaire to generate lightweight, rule-based suggestions. This makes it fundamentally different from large commercial systems.

- **Collaborative Filtering (CF):**
  Platforms like Spotify, Apple Music, and YouTube Music primarily use collaborative filtering, where users with similar listening habits receive similar recommendations. CF is highly accurate at scale but requires huge datasets and continuous tracking of listening patterns. Since MusicMind does not collect listening logs, this method is not applicable for the scope of this project.
- **Content-Based Filtering:**
  Systems like Pandora's "Music Genome Project" use manually crafted song attributes (genre, rhythm, instruments) to match users with songs that share similar

characteristics. This approach demands large labeled datasets and detailed audio analysis, which are beyond the educational scope of MusicMind.

- **Hybrid Systems:**
  Most modern music apps combine CF + content-based filtering, sometimes enhanced by ML embeddings, deep learning, or contextual modeling (time of day, recent activity). These systems require scalable servers, user tracking, and large-scale training pipelines. MusicMind remains intentionally simpler and avoids these complexities.
- **Mood-Based Recommendation Systems:**
  Certain applications provide mood-driven playlists using emotion tags or audio mood detection. These rely on either explicit inputs ("I feel happy/sad") or ML-based mood classification from audio features. MusicMind differs by not using emotional states; instead, it gathers stable personality traits through a one-time questionnaire.
- **Personality-Based Recommendation Research:**
  Previous academic studies (e.g., Big Five personality traits influencing music preference) show that personality can be a reliable indicator of music taste patterns. However, most commercial platforms do not implement personality-based matching because it lacks large labeled datasets and requires manual mapping. MusicMind adopts a simplified educational implementation of personality-based recommendations without using machine learning.

Gap Identified:
Unlike large commercial systems, none of the mainstream platforms offer personality-based recommendations through questionnaires. Most depend on listening history, which new users often lack. MusicMind fills this gap by offering:

- A lightweight, no-history-required recommendation method.
- A simple rule-based mapping instead of complex ML models.
- A standalone demo system suitable for academic evaluation.

Thus, MusicMind contributes a simplified, questionnaire-driven model suitable for small-scale environments, educational use cases, and scenarios where user listening history is unavailable.

Existing Systems

- **Spotify:** Recommendation mainly uses listening history and collaborative filtering.
- **Pandora:** Uses the Music Genome Project (content-based) to recommend tracks.
- **Mood-based apps:** Use explicit mood input or audio analysis to suggest songs.

Compatibility Table

Table 2 Compatibility Table

System	Recommendation Basis	Admin Panel	Personality Questionnaire	Local DB	Cloud Auth	Notes
Spotify [1], [2]	Listening history + collaborative filtering	Internally Available	No	No	OAuth	Large-scale, streaming-focused
Pandora [3], [4]	Content-based (genome)	Internally Available	No	No	OAuth	Proprietary music analysis
Mood apps [5], [6]	Mood input / audio mood	Partial	Partial	No	Varies	Focus on mood, not personality
MusicMind [7]	Personality-traits questionnaire --> name suggestions	Yes	Yes	Yes (PostgreSQL)	Firebase Auth	Lightweight, DB-backed, admin CRUD

4. USER/SYSTEM REQUIREMENTS

As per report parameters, only External Interface requirements are stated.

External Interface Requirements

User Interfaces

- **Login Page:** Firebase sign-in integration; shows appropriate link to user or admin dashboard after auth.
- **Dashboard (User):** Shows questionnaire link/status, recommendations, and top songs (visible only after questionnaire is filled at least once).
- **Questionnaire Page:** Multi-field form (personality trait questions). Uses Flask-WTF for validation.
- **Profile Page:** Displays stored personality traits and history of recommendations.
- **Dashboard (Admin):** Buttons/links for “Drop the Beat” (add song), “SoundTracker” (update/delete), “Graveyard” (deleted songs), and “Spy on Fans” (user count).
- **Firebase Document UI (Admin):** Simple editor to create Firestore documents that display as a table in the admin view.

UI notes:

- Use responsive HTML/CSS (works on desktop; mobile friendly optional).

- Use Jinja2 templates rendered by Flask for server-side pages.

Hardware Interfaces

No special hardware needed. Runs on standard desktop/server hardware supporting Python and PostgreSQL.

Software Interfaces

- **Backend:** Flask (3.0.2).
- **ORM / DB:** SQLAlchemy + Flask-SQLAlchemy, psycopg2-binary for PostgreSQL connection.
- **Migrations:** Flask-Migrate + Alembic.
- **Auth:** Firebase Admin SDK (firebase-admin) + Flask-Login for session management.
- **Forms:** Flask-WTF + email_validator.
- **Templates:** Jinja2.
- **Env management:** python-dotenv.

Communication Interfaces

- **HTTP/HTTPS:** Standard web requests between client and Flask server (port 5000 default in dev).
- **DB Connection:** Postgres protocol via psycopg2 (configured by DB URL).
- **Firebase API:** HTTPS calls to Firebase using firebase-admin and service account credentials.

5. Functional Requirements

Requirement IDs use the prefix FR- (functional requirement).

- FR-01: User Authentication (login/logout using Firebase).
- FR-02: Questionnaire: present and collect personality trait responses.
- FR-03: Store questionnaire responses in PostgreSQL.
- FR-04: Generate song recommendations based on questionnaire results.
- FR-05: Show recommendations and top songs on user dashboard (top songs visible if questionnaire completed).
- FR-06: User Profile page shows saved personality traits.
- FR-07: Admin Add Song (Drop the Beat).
- FR-08: Admin Update/Delete Song (SoundTracker) modify or mark songs deleted.
- FR-09: Graveyard — list and retain deleted songs (soft delete).
- FR-10: Spy on Fans — show number of registered users and basic stats.
- FR-11: Firebase Document Editor (admin) — create documents that render in table format.

Functional Requirements with Traceability information

Table 3 FR 01

Requirement ID		**Requirement Type**				**Use Case #**	
FR-01		Functional				UC-01	
Status	New	**Agreed-to**	**–**	**Baselined**	**–**	**Rejected**	**–**
Parent Requirement #		**–**					
Description		The system shall authenticate users and admins using Firebase and redirect them to the appropriate dashboard.					
Rationale		Ensures secure login and separation of user roles.					
Source		Project Specification					
Acceptance/Fit Criteria		On valid credentials, the system logs in and redirects the user; invalid credentials show an error.					
Dependencies		Firebase configuration, internet connection, Flask-Login session management.					
Priority		**Essential**					
Change History		**Created 2025-10-12 by Riyan Khan Durrani**					

Table 4 FR 02

Requirement ID		**Requirement Type**				**Use Case #**	
FR-02		Functional				UC-02	
Status	New	**Agreed-to**	**–**	**Baselined**	**–**	**Rejected**	**–**
Parent Requirement #		**FR-01**					
Description		The system shall present a personality questionnaire form to authenticated users and store their responses.					
Rationale		Collects personality traits needed for music recommendations.					
Source		Project Specification					
Acceptance/Fit Criteria		Logged-in users can fill and submit the questionnaire; data saves correctly in PostgreSQL.					
Dependencies		Requires login (FR-01) and database connection.					
Priority		**Essential**					
Change History		**Created 2025-10-12 by Riyan Khan Durrani.**					

Table 5 FR 03

Requirement ID		**Requirement Type**				**Use Case #**	
FR-03		Functional				UC-02	
Status	New	**Agreed-to**	**–**	**Baselined**	**–**	**Rejected**	**–**

Parent Requirement #	**FR-02**
Description	The system shall store each user's questionnaire responses with user ID and timestamp in PostgreSQL.
Rationale	Needed for generating personalized suggestions and showing traits in profile.
Source	Project Specification
Acceptance/Fit Criteria	On submission, a database record is created with valid fields and linked to the user.
Dependencies	PostgreSQL connection, SQLAlchemy setup, FR-02 completion.
Priority	**Essential**
Change History	**Created 2025-10-12 by Riyan Khan Durrani.**

Table 6 FR 04

Requirement ID		**Requirement Type**				**Use Case #**	
FR-04		Functional				UC-03	
Status	New	**Agreed-to**	–	**Baselined**	–	**Rejected**	–
Parent Requirement #		**FR-03**					
Description		The system shall generate and display a ranked list of song recommendations based on questionnaire results.					
Rationale		Core function — provides users with personalized song suggestions.					
Source		Project Specification					
Acceptance/Fit Criteria		Recommendations appear on dashboard within 2 s when questionnaire data exists.					
Dependencies		FR-03 data, songs available in DB, recommendation logic implemented.					
Priority		**Essential**					
Change History		**Created 2025-10-13 by Riyan Khan Durrani.**					

Table 7 FR 05

Requirement ID	**Requirement Type**	**Use Case #**					
FR-05	Functional	UC-03					
Status	New	**Agreed-to**	–	**Baselined**	–	**Rejected**	–
Parent Requirement #	**FR-04**						
Description	The system shall display generated song recommendations and top songs on the user dashboard once the questionnaire is completed.						
Rationale	Allows users to quickly see their personalized suggestions and top songs, enhancing engagement.						

Source	Project Specification
Acceptance/Fit Criteria	Dashboard shows recommendations and top songs within 2 seconds of generation; only visible after completing questionnaire.
Dependencies	FR-04 (recommendation generation), FR-02 (questionnaire completed), frontend rendering.
Priority	Essential
Change History	**Created 2025-10-14 by Riyan Khan Durrani.**

Table 8 FR 06

Requirement ID		**Requirement Type**				**Use Case #**	
FR-06		Functional				UC-04	
Status	New	**Agreed-to**	–	**Baselined**	–	**Rejected**	–
Parent Requirement #		**FR-03**					
Description		The system shall allow users to view their stored personality traits on their profile page.					
Rationale		Lets users review their data and understand how recommendations are generated.					
Source		Project Specification					
Acceptance/Fit Criteria		Logged-in users can open Profile and see accurate trait data within 2 seconds.					
Dependencies		FR-03 (questionnaire data) and FR-01 (login).					
Priority		**Essential**					
Change History		**Created 2025-10-15 by Riyan Khan Durrani.**					

Table 9 FR 07

Requirement ID		**Requirement Type**				**Use Case #**	
FR-07		Functional				UC-05	
Status	New	**Agreed-to**	–	**Baselined**	–	**Rejected**	–
Parent Requirement #		–					
Description		The admin shall add new songs (title, artist, genre, tags, description) using the “Drop the Beat” option.					
Rationale		Keeps the song catalog updated for user recommendations.					
Source		Project Specification					
Acceptance/Fit Criteria		Added song appears instantly in the admin catalog table after saving.					
Dependencies		Admin authentication (FR-01), PostgreSQL connection.					
Priority		**Essential**					

Change History	**Created 2025-10-16 by Syed Hamza Mukhtar Bukhari.**

Table 10 FR 08

Requirement ID		**Requirement Type**					**Use Case #**
FR-08		Functional					UC-06
Status	New	**Agreed-to**	–	**Baselined**	–	**Rejected**	–
Parent Requirement #		**FR-07**					
Description		The admin shall edit or delete existing songs using the “SoundTracker” module.					
Rationale		Allows correction or removal of outdated song data.					
Source		Project Specification					
Acceptance/Fit Criteria		Updates reflect instantly; deletions move the song to the Graveyard table.					
Dependencies		FR-06, admin privileges, PostgreSQL setup.					
Priority		**Essential**					
Change History		**Created 2025-10-17 by Syed Hamza Mukhtar Bukhari.**					

Table 11 FR 09

Requirement ID		**Requirement Type**					**Use Case #**
FR-09		Functional					UC-07
Status	New	**Agreed-to**	–	**Baselined**	–	**Rejected**	–
Parent Requirement #		**FR-07**					
Description		The system shall store deleted songs in a “Graveyard” view for admin reference and possible restoration.					
Rationale		Prevents accidental permanent loss of data and supports recovery.					
Source		Project Specification					
Acceptance/Fit Criteria		Deleted songs appear in Graveyard with metadata and can be restored by admin.					
Dependencies		FR-07 (delete function) and PostgreSQL connection.					
Priority		**Conditional**					
Change History		**Created 2025-10-20 by Syed Hamza Mukhtar Bukhari.**					

Table 12 FR 10

Requirement ID		**Requirement Type**					**Use Case #**
FR-10		Functional					UC-09
Status	New	**Agreed-to**	–	**Baselined**	–	**Rejected**	–

Parent Requirement #	–
Description	The system shall provide an admin view (“Spy on Fans”) that displays the total number of registered users and basic statistics (questionnaire completion, active users, top genres).
Rationale	Gives admins insights into user engagement and system usage patterns.
Source	Project Specification
Acceptance/Fit Criteria	Admin can access a dashboard showing user count, completion percentages, and simple statistics; data updates in real-time or on page refresh.
Dependencies	FR-01 (admin login), FR-02 (questionnaire), PostgreSQL connection.
Priority	Essential
Change History	**Created 2025-10-22 by Riyan Khan Durrani.**

Table 13 FR 11

Requirement ID		**Requirement Type**				**Use Case #**	
FR-11		Functional				UC-08	
Status	New	**Agreed-to**	**–**	**Baselined**	**–**	**Rejected**	**–**
Parent Requirement #		–					
Description		The system shall allow admins to create, edit, and display documents in a table format using Firebase Firestore.					
Rationale		Provides admins with a flexible tool to store and present structured information without altering the main database.					
Source		Project Specification					
Acceptance/Fit Criteria		Admin can create a document, edit its content, and display it in a table on the admin dashboard; changes save to Firestore in real-time.					
Dependencies		FR-01 (admin login), Firebase Firestore setup, frontend table rendering.					
Priority		Essential					
Change History		**Created 2025-10-27 by Riyan Khan Durrani.**					

Table 14 FR 12

Requirement ID		**Requirement Type**				**Use Case #**	
FR-12		Functional				UC-10	
Status	New	**Agreed-to**	**–**	**Baselined**	**–**	**Rejected**	**–**
Parent Requirement #		**–**					

Description	The system shall allow both Users and Admin to securely log out, immediately terminating the active session and redirecting the user to the homepage or login screen.
Rationale	Ensures account security, prevents unauthorized access, and supports proper session management.
Source	Project Specification
Acceptance/Fit Criteria	Upon clicking “Logout,” the session is cleared, access to all authenticated pages is blocked, and the user is redirected successfully.
Dependencies	Authentication module (Firebase/PostgreSQL session handling)
Priority	Essential
Change History	**Created 2025-10-29 by Riyan Khan Durrani.**

6. Nonfunctional Requirements & Software System Attributes

Performance Requirements

- **Response Time:** Typical page requests (dashboard, profile, questionnaire) should render with no more than **1 second** on local dev machines (single-user). Recommendation generation should complete with no more than 1 **second** for typical DB sizes used in the project.
- **Throughput:** System shall handle at least 50 simultaneous users for basic operations during the demo without critical slowdowns (a reasonable local dev expectation).
- **Scalability:** Designed so the DB or Firebase services can be migrated to cloud-hosted servers if required.

Reliability & Availability

- The system should handle expected database connectivity interruptions gracefully (show user-friendly errors).
- Data written to PostgreSQL must be persisted reliably; admin deletions use soft-delete to prevent accidental data loss (Graveyard feature).

Security

- Use Firebase Authentication for user identity management.
- Sensitive credentials stored in environment variables and never committed to source control.
- Use server-side input validation (Flask-WTF) and parameterized DB queries via SQLAlchemy to prevent injection.

Usability

- Simple, minimal UI. Dashboard should be intuitive for both users and admins.
- Questionnaire should not exceed a reasonable number of questions (e.g., 10-15) and should validate required fields.

Maintainability

- Codebase uses Flask blueprints and modular structure.
- Database migrations handled by Flask-Migrate/Alembic to track schema changes.

Portability

- Application should run on any machine with Python 3.x, PostgreSQL installed, and Firebase credentials available. Use .env to configure environment specifics.

7. Project Design/Architecture

- Project design and detailed architecture diagrams (4+1 view model, ER diagrams, class diagrams, deployment view) will be developed in SRS Part 2.
- Foreshadowing design part:
  - **Backend:** Flask app with blueprints.
  - **ORM:** SQLAlchemy models for User, QuestionnaireResponse, Song, Recommendation, DeletedSong.
  - **Auth:** Firebase + Flask-Login.
  - **Database:** PostgreSQL (local during development).
  - **Cloud features(Optional):** Firestore documents (admin editor).

8. Work Breakdown (WB)

The following table shows work breakdown of first part of this document.

Table 15 Work Breakdown

Group Members	Reg No.	Contribution
Syed Hamza Mukhtar Bukhari	2023682	Sections 1 & 2, FR Drafting, WB
Arsalan Khalil	2023130	Section 3 (State of the Art)
Riyan Khan Durrani	2023611	Section 4 (Interfaces) and FR traceability
Adan Amir	2023057	Section 6 (Nonfunctional) & Proofreading
Momina Zahid	2023335	Section 5 (Functional Requirements) & Tables

9. Appendix A — Environment & Setup Notes

- **Python packages:**
  Flask==3.0.2, Flask-SQLAlchemy==3.1.1, Flask-Login==0.6.3, Flask-Migrate==4.0.5, psycopg2-binary==2.9.9, SQLAlchemy==2.0.28, Werkzeug==3.0.1, Jinja2==3.1.3, click==8.1.7, itsdangerous==2.1.2, alembic==1.13.1, python-dotenv==1.0.1, Flask-WTF==1.2.2, email_validator==2.2.0, firebase-admin==6.4.0.
- **DB:**
  PostgreSQL (local). Provide DB connection string via DATABASE_URL in .env.
- **Firebase:**
  Service account JSON and project config loaded from environment.

10. References

Following are the references which were used in literature review.

[1] Spotify AB, “How Spotify recommendations work.” Spotify Newsroom. Accessed: Nov. 27, 2025. [Online]. Available: https://newsroom.spotify.com

[2] Spotify AB, “Spotify for Developers – Authentication Guide.” Accessed: Nov. 27, 2025. [Online]. Available: https://developer.spotify.com/documentation/general/guides/authorization

[3] T. Westergren, “The Music Genome Project: Inside Pandora’s recommendation engine.” Pandora Media. Accessed: Nov. 27, 2025. [Online]. Available: https://www.pandora.com

[4] T. Vanderbilt, “How Pandora’s Music Genome Project works,” *Wired Magazine.* Accessed: Nov. 27, 2025. [Online]. Available: https://www.wired.com

[5] K. McFerran, “Mood-based music recommendation apps: A review of current approaches,” *Journal of Music & Emotion Research,* vol. 14, no. 2, pp. 55–70, 2020.

[6] R. A. Calvo and S. D’Mello, *“Affect detection and mood-adaptive systems,” IEEE Trans. Affect. Comput.*, vol. 1, no. 1, pp. 18–37, 2010.

[7] S. H. M. Bukhari, R. K. Durrani, A. Khalil, *MusicMind: A personality-driven music recommendation system using questionnaire-based profiling*. GIKI, 2025 (unpublished project documentation).

PART-II

Project Design and Architecture

Music Mind Date: 9/11/2025

11. INTRODUCTION

PURPOSE

This document (SRS Part 2) presents the design for **MusicMind**. It shows architecture, modules, data flow, class structure, sequence and activity diagrams, and UI mockups to demonstrate how the system will be implemented and how design maps to requirements from Part 1.

PRODUCT SCOPE

MusicMind suggests song names to users based on personality questionnaire input. This design focuses on the web app backend (Flask), local PostgreSQL DB, Firebase integration (auth + optional Firestore), and the frontend (HTML/CSS/Javascript via Jinja2).

Table 16 Terms used in this part and their descriptions

Name	Description
MusicMind	The name of the software application that recommends songs based on user personality traits.
User	A person who logs in to MusicMind to fill the questionnaire and receive song recommendations.
Admin	A person with privileges to manage songs, view users, and perform administrative tasks in MusicMind.
Dashboard	The main landing page after login, displaying key features for User or Admin.
Questionnaire	A form where the User provides personality traits to help generate song recommendations.
Recommendations	The list of songs suggested to a User based on their questionnaire responses.
Drop the Beat	Admin feature to insert a new song into the system.
SoundTracker	Admin feature to update or delete existing songs.
Graveyard	Admin feature that stores deleted songs for potential recovery.
Spy on Fans	Admin feature to view the total number of users.
Firebase Integration	Cloud service used for authentication and Firestore document storage.
PostgreSQL	Local database used to store Users, Questionnaire responses, Recommendations, and Songs.
Flask	Python web framework used to build the backend of MusicMind.

12. OVERVIEW

This part includes an architecture diagram, justification for the chosen architecture, module identification (with responsibilities), 4+1 views (Use Case, Logical, Development, Process, Physical), UI mockups, and the Work Breakdown Diagram for Part 2 tasks.

THE OVERALL DESCRIPTION

This section summarizes the runtime environment and main components.

- **Runtime environment:** Python + Flask server running on a development machine (port 5000). PostgreSQL runs locally. Firebase used for auth/Firestore (requires internet).
- **Main components:** Client browser (UI), Flask server (controllers & views), DB service (SQLAlchemy models), Firebase Service (auth/docs), Recommendation engine (rulebased mapper), Admin console.
- **Data flow (high level):** Browser → Flask routes → services (AuthService / DBService / Recommender) → PostgreSQL / Firebase → Flask → Browser.

PRODUCT PERSPECTIVE

- **Type:** Client-server web application, single-server for Part 2 (development).
- **Layering:** Presentation (templates), Application (Flask controllers), Business logic (Recommendation logic), Data (Postgres via SQLAlchemy), External service (Firebase).

13. WORK BREAKDOWN STRUCTURE

Following is the work breakdown of our group members for the Design Phase.

- Architectural design (Hamza)
- Module identification & classes (Arsalan)
- Use case diagrams and activity diagrams (Riyan)
- Sequence diagrams and process flow (Momina)
- UI mockups & wireframes (Adan)
- Document editing & formatting (All members)

14. Design

ARCHITECTURAL DESIGN

Architecture Type: *MVC (Model-View-Controller) within a Client-Server setup.*

The overall architecture of **MusicMind** follows a **client–server model** based on the **MVC (Model–View–Controller)** pattern implemented using **Flask**.

The web application interacts with two data sources.

- **PostgreSQL** for core user and song data.
- **Firebase** for authentication and additional cloud-based storage.

ARCHITECTURE OVERVIEW

- **Client (Front-End):**
  Developed using HTML, CSS, and JavaScript, the client provides all user interfaces for login/signup, questionnaire, recommendations, and admin operations.
- **Server (Back-End):**
  Implemented in **Python (Flask)**, acting as the controller that manages user requests, communicates with databases, and returns dynamic content using **Jinja2** templates.
- **Databases:**
  - **PostgreSQL (Local Server):**
    - Stores structured data such as user profiles, questionnaire results, and song details. o **Firebase (Cloud Service):**
    - Handles authentication (login/signup validation) and stores admin-level collections or temporary data through Firestore.
    - This connection uses the **Firebase Admin SDK**, integrated through API credentials in the Flask backend.
- **Flow Summary:**
  User accesses the homepage with three options — *Login*, *Signup*, and *Firebase Integration*.
  - Login/signup operations are verified via Firebase authentication.
  - After authentication, users are redirected to either:
    - **User Dashboard** – questionnaire, recommendations, profile, and logout.
    - **Admin Dashboard** – drop the beat, soundtrack management, graveyard, and spy on fans.
  - Firebase integration also allows the creation of new collections (documents) visible in Firestore but not linked to PostgreSQL.
  - Flask routes handle all HTTP requests, manage logic, and connect to the PostgreSQL database when needed.

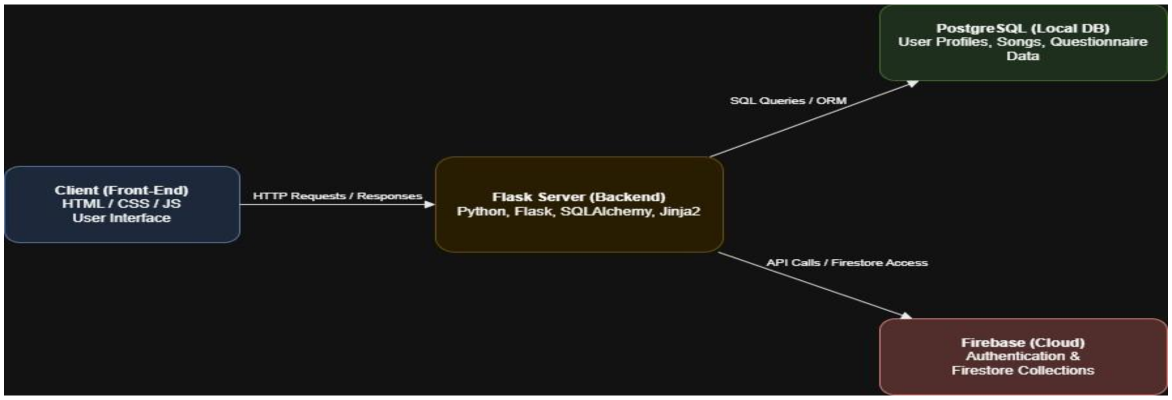


Figure 1: Architectural Design(MVC)

Why we chose MVC/Client Server Architecture

Justification:

- **Separation of concerns:** MVC isolates UI from application logic making development and testing simpler (Flask supports this pattern).
- **Simplicity for demo:** Single Flask server is lightweight and easy to deploy locally for demos and grading.
- **Modularity:** Services (DB, Firebase, Recommender) can be replaced or scaled later (e.g., microservices) without redesign.
- **Tooling fit:** Flask + Jinja2 naturally implements Controller + View; SQLAlchemy forms the Model layer.

MODULE IDENTIFICATION

List of major modules and short **responsibility** notes:

- **AuthController / AuthModule**
  Login/logout, role routing (user → user dashboard, admin → admin dashboard), session management via Flask-Login and Firebase auth.
- **UserController**
  Serve user dashboard, profile page, questionnaire page; handle questionnaire submission.
- **AdminController**
  CRUD for songs (Drop the Beat / SoundTracker), Graveyard management, user stats (Spy on Fans), Firestore editor.
- **Recommender**
  Map personality traits to song suggestions (rule-based mapping or lookup), returning ranked list.
- **DBService / Models**
  SQLAlchemy models (User, Song, QuestionnaireResponse, Recommendation, DeletedSong), DB migrations.
- **FirebaseService**

Firebase Admin SDK interactions: verify tokens (if used), Firestore document CRUD (admin feature).

- **Templates / UI Module**
  Jinja2 templates for rendering pages, static files (CSS/JS).
- **Utilities**
  Env loader, logging, helpers.

Module Interaction Summary:
Controllers call Services → Services access Models → Controllers render templates or return JSON.

15. 4+1 ARCHITECTURE VIEW MODEL

(This section is intentionally skipped as per the assignment requirements.)

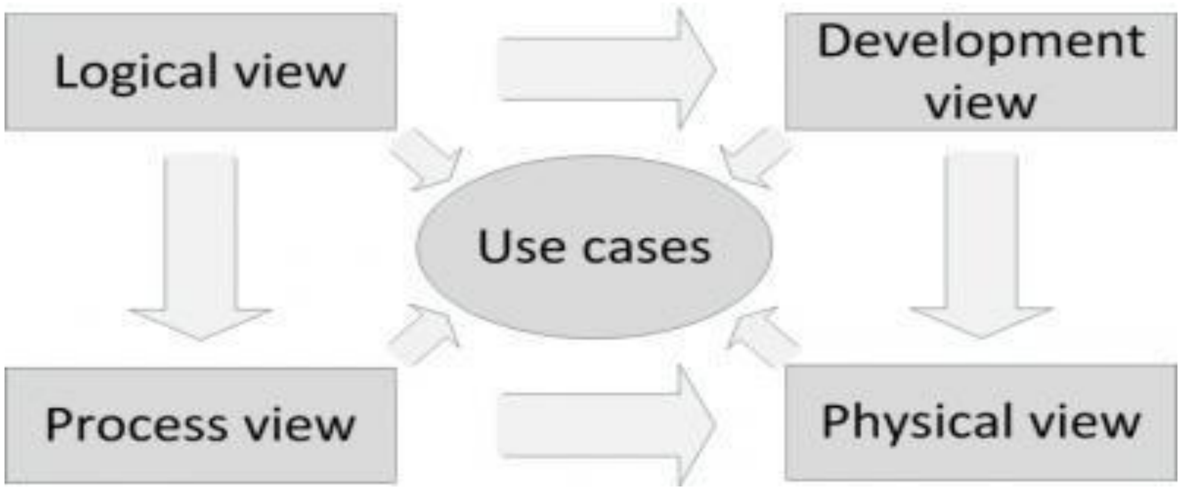


Figure 2: 4+1 Architectural View

USE CASE VIEW

This is a list of use-cases that represent major functionality of the final system:
Primary Actors:
User, Admin
Major Use Cases:

- UC-01: Login (User / Admin)
- UC-02: Fill Questionnaire
- UC-03: View Recommendations
- UC-04: View Profile
- UC-05: Admin Add Song (Drop the Beat)
- UC-06: Admin Update/Delete Song (SoundTracker)
- UC-07: View Graveyard
- UC-08: Firebase Document Editor
- UC-09: View User Statistics (Spy on Fans)
- UC-10: Logout

Use Case Diagram:

- Actors: User, Admin
- Draw associations from each actor to corresponding use cases listed above.

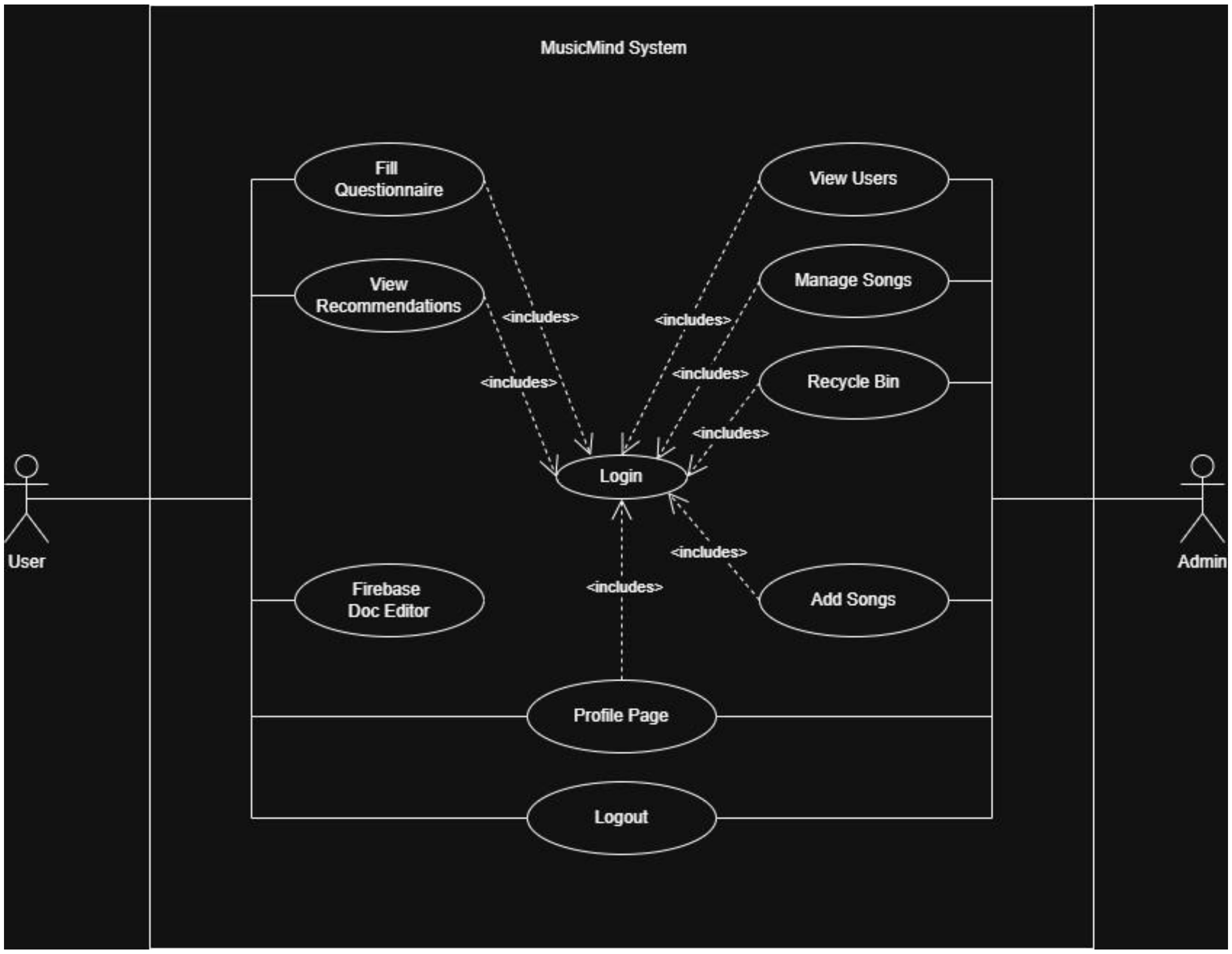


Figure 3 Use Case Diagram

LOGICAL VIEW

The Logical View describes the internal structure of the system, focusing on the key classes, components, and their relationships. It explains how data models such as User, QuestionnaireResponse, Song, and Recommendation interact within the system.

Classes / Key Data Structures:

- **User**
  - **Attributes:** user_id, name, email, role, created_at
  - **Methods:** get_profile(), get_recommendations()
- **QuestionnaireResponse**
  - **Attributes:** response_id, user_id, answers (JSON), completed_at
  - **Methods:** validate(), to_vector()
- **Song**

    - **Attributes:** song_id, name, artist, genre, tags, description, is_deleted, created_at
    - **Methods:** mark_deleted(), restore()
- **Recommendation**
    - **Attributes:** rec_id, user_id, generated_at, song_list(ordered)
    - **Methods:** generate_from_responses()
- **DeletedSong (Graveyard)**
    - **Attributes:** grave_id, song_id, deleted_at, deleted_by
- **FirebaseDocument**
    - **Attributes:** doc_id, title, content, created_by, created_at
- **Services (no state): DBService, FirebaseService**

Activity Diagrams:

This section contains two key activity diagrams that visually represent the step-by-step flow of important users and system operations. Each diagram shows the sequence of actions, decisions, and system responses involved in processes such as user login, questionnaire submission, recommendation generation, and Firebase document creation. These diagrams help clarify how MusicMind behaves in different scenarios by mapping actions from start to finish in a workflow-oriented structure

User Login & Dashboard Flow

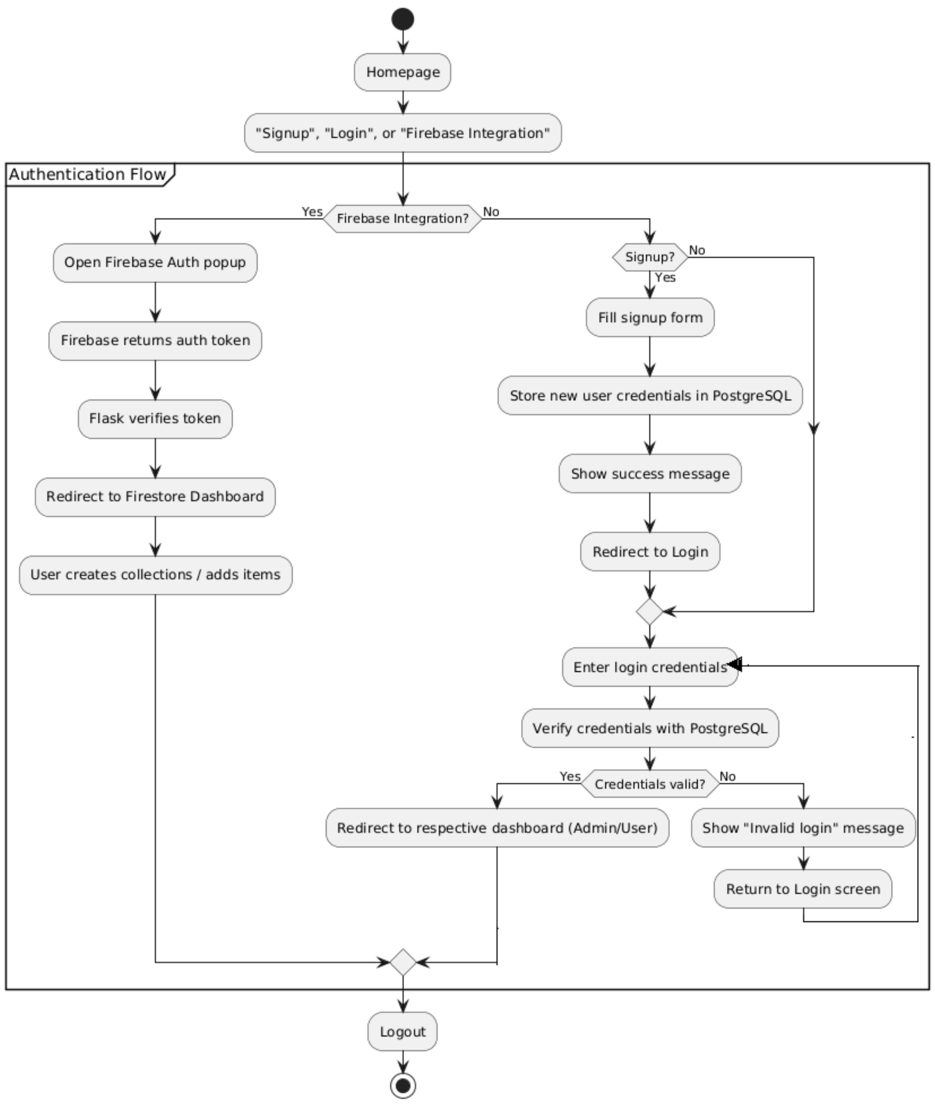
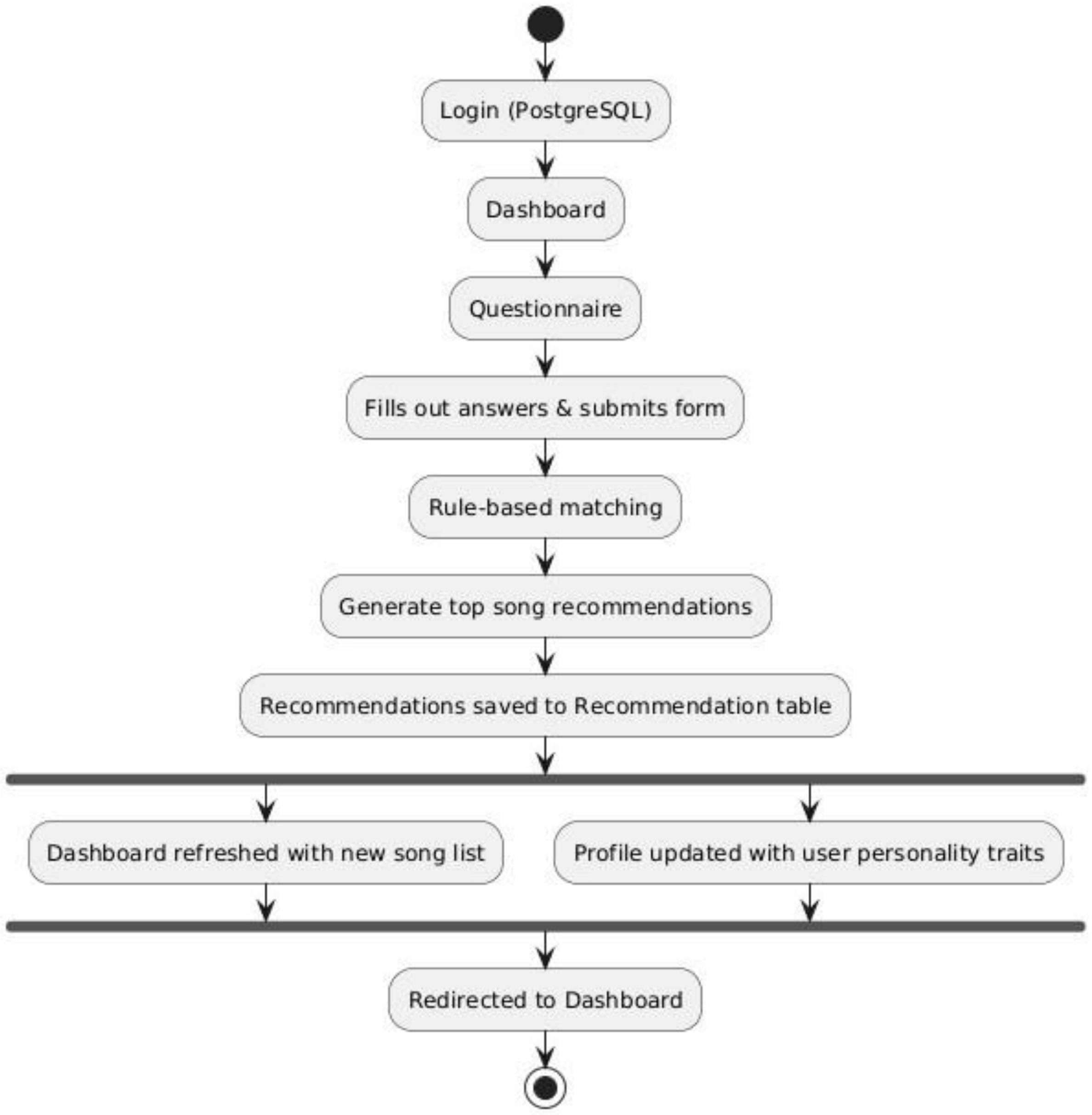

Homepage
"Signup", "Login", or "Firebase Integration"
Authentication Flow
Firebase Integration?
Yes
No
Open Firebase Auth popup
Firebase returns auth token
Flask verifies token
Redirect to Firestore Dashboard
User creates collections / adds items
Signup?
No
Yes
Fill signup form
Store new user credentials in PostgreSQL
Show success message
Redirect to Login
Enter login credentials
Verify credentials with PostgreSQL
Credentials valid?
Yes
No
Redirect to respective dashboard (Admin/User)
Show "Invalid login" message
Return to Login screen
Logout

Figure 4 Activity Diagram - I

Fill Questionnaire → Generate Recommendations



Login (PostgreSQL)
Dashboard
Questionnaire
Fills out answers & submits form
Rule-based matching
Generate top song recommendations
Recommendations saved to Recommendation table
Dashboard refreshed with new song list
Profile updated with user personality traits
Redirected to Dashboard

Figure 5 Activity Diagram - II

DEVELOPMENT VIEW

(This section is intentionally skipped as per the assignment requirements.)

PROCESS VIEW

The Process View describes three dynamic behavior of the system by showing how different components interact during runtime. It explains the flow of control and the sequence of operations that occur when users perform key actions such as logging in, submitting the questionnaire, generating recommendations, or when the admin manages songs. Sequence diagrams and activity diagrams are included to illustrate how data moves between the user interface, the Flask backend, PostgreSQL, and Firebase modules. This view highlights communication paths, decision points, and major processing steps within the system.

User Login:

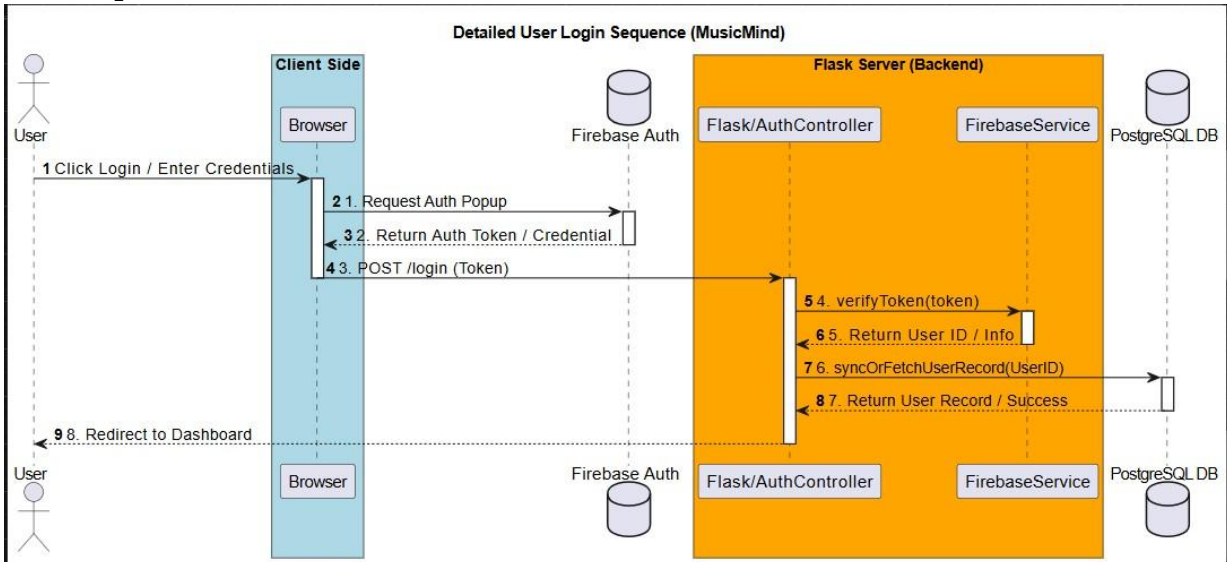

Detailed User Login Sequence (MusicMind)
Client Side
Flask Server (Backend)
User
Browser
Firebase Auth
Flask/AuthController
FirebaseService
PostgreSQL DB
1 Click Login / Enter Credentials
2 1. Request Auth Popup
3 2. Return Auth Token / Credential
4 3. POST /login (Token)
5 4. verifyToken(token)
6 5. Return User ID / Info
7 6. syncOrFetchUserRecord(UserID)
8 7. Return User Record / Success
9 8. Redirect to Dashboard

Figure 6 Sequence Diagram – I

Recommendation Generation:

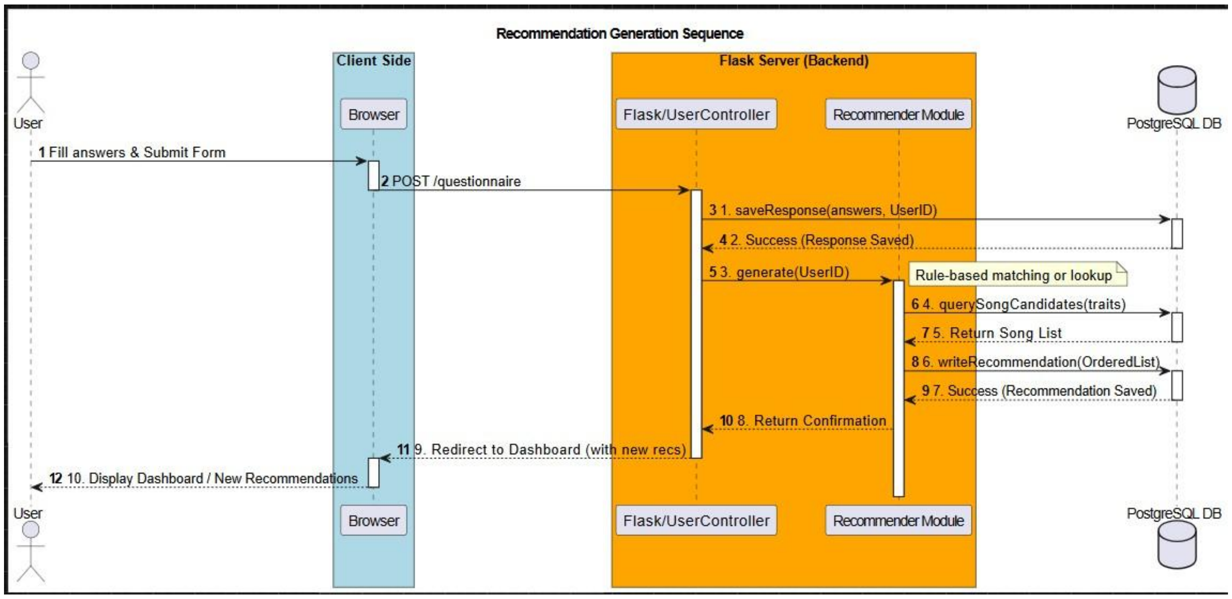

Recommendation Generation Sequence
Client Side
Flask Server (Backend)
User
Browser
Flask/UserController
Recommender Module
PostgreSQL DB
1 Fill answers & Submit Form
2 POST /questionnaire
3 1. saveResponse(answers, UserID)
4 2. Success (Response Saved)
5 3. generate(UserID)
Rule-based matching or lookup
6 4. querySongCandidates(traits)
7 5. Return Song List
8 6. writeRecommendation(OrderedList)
9 7. Success (Recommendation Saved)
10 8. Return Confirmation
11 9. Redirect to Dashboard (with new recs)
12 10. Display Dashboard / New Recommendations

Figure 7 Sequence Diagram - II

Admin Add Song Sequence:

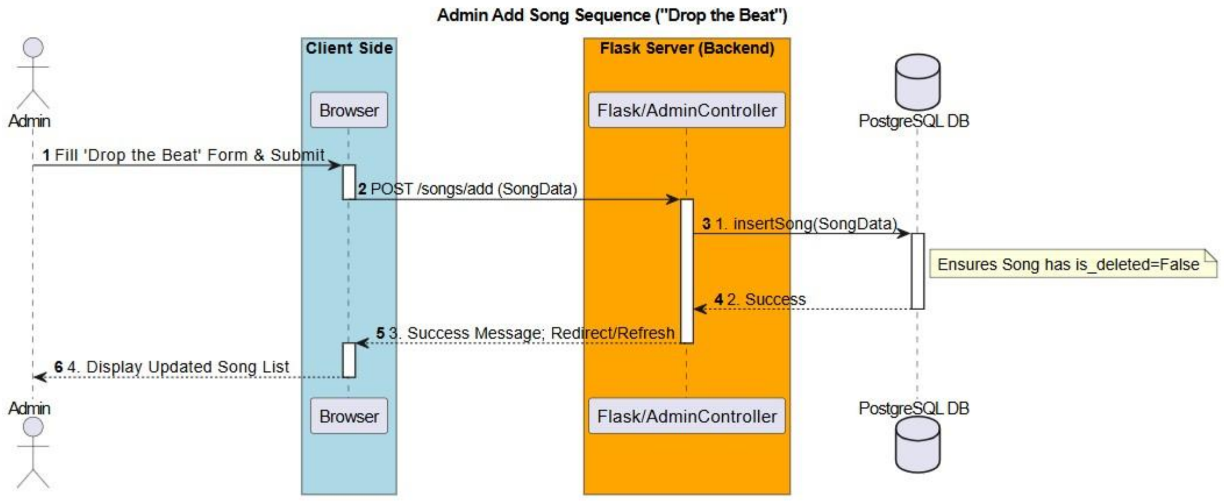

Admin Add Song Sequence ("Drop the Beat")
Client Side
Flask Server (Backend)
Admin
Browser
Flask/AdminController
PostgreSQL DB
1 Fill 'Drop the Beat' Form & Submit
2 POST /songs/add (SongData)
3 1. insertSong(SongData)
Ensures Song has is_deleted=False
4 2. Success
5 3. Success Message; Redirect/Refresh
6 4. Display Updated Song List
Admin
Browser
Flask/AdminController
PostgreSQL DB

Figure 8 Sequence Diagram - III

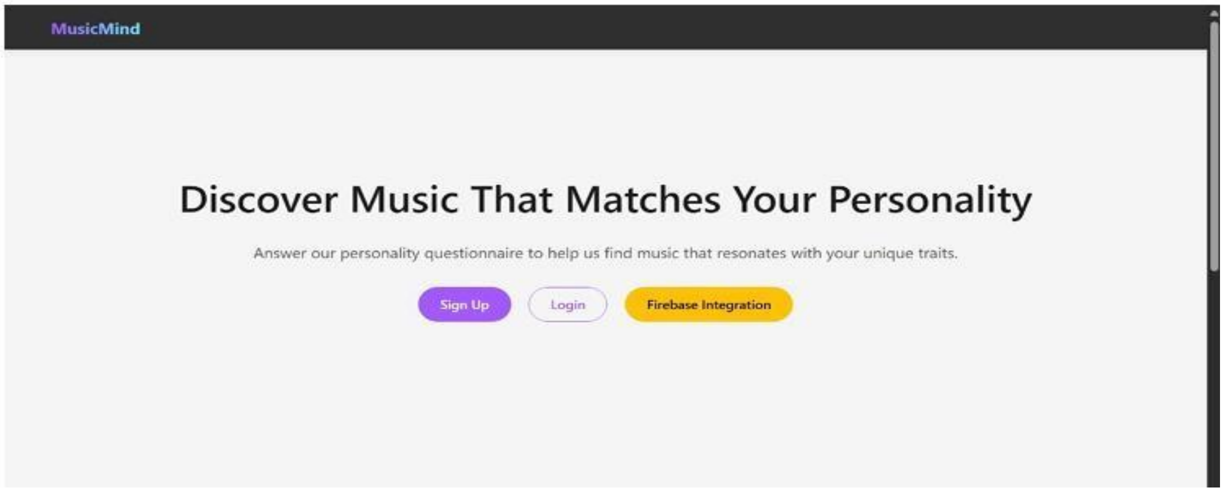
Physical View

(This section is intentionally skipped as per the assignment requirements.)

16. User Interface Design

This section presents the visual layout and navigation structure of the MusicMind system. It includes mockups or wireframes of the primary screens for both User and Admin roles. The UI design focuses on simplicity, clarity, and ease of use.
Major screens such as Login, User Dashboard, Recommendations, Admin Dashboard, and Profile-page interfaces are illustrated to show how users interact with core functionalities. The purpose of this section is to demonstrate how the interface supports the system's functional requirements and provides an intuitive user experience.

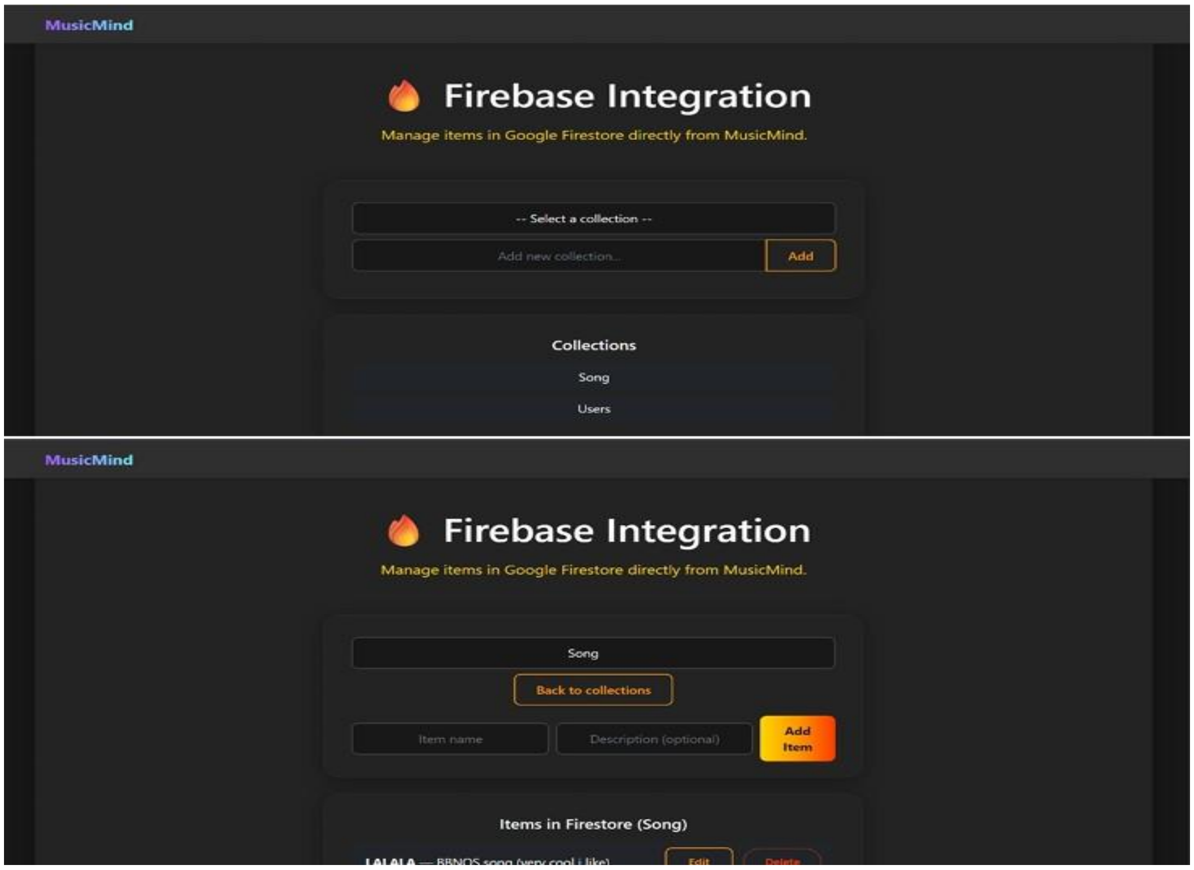
- **Login Page**
  - **Elements:** Email field, Password field (or Firebase Sign-In button), Login button, "Forgot password" link, "Sign up" link.
  - **Behavior:** After login, redirect to user or admin dashboard.



Discover Music That Matches Your Personality

Answer our personality questionnaire to help us find music that resonates with your unique traits.

Figure 9 Login Page



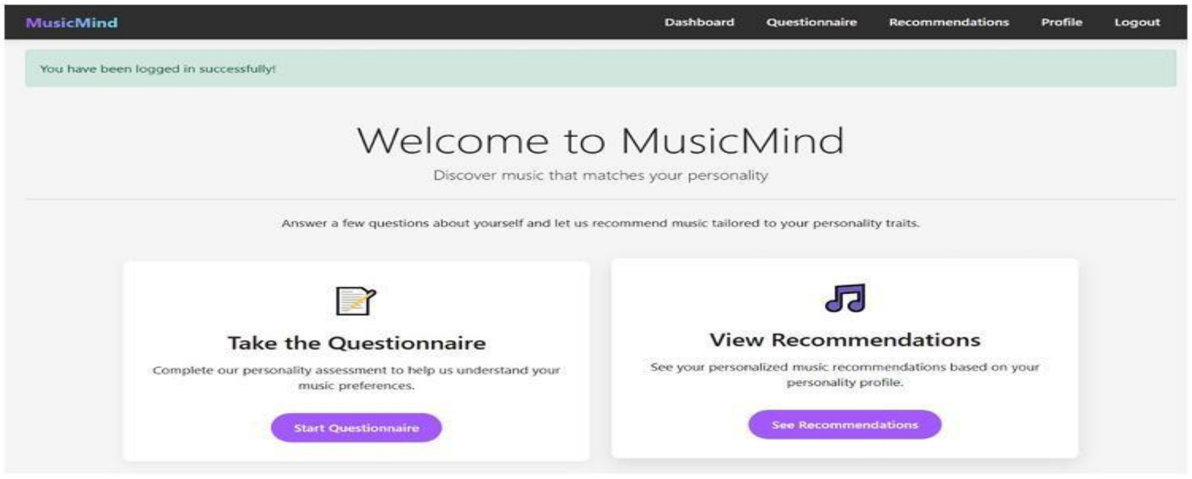
Manage items in Google Firestore directly from MusicMind.

Manage items in Google Firestore directly from MusicMind.

LALALA — BBNOS song (very cool i like)

Figure 10 Firebase Module

- **User Dashboard**
  - **Top:** Welcome message, Logout button, Profile link.
  - **Left pane:** Questionnaire status card (button to Fill Questionnaire or View).
  - **Center pane:** Recommendations (list of song names with artist + short tag).
  - **Right pane:** Top songs (visible once questionnaire completed).
  - **Footer:** Small About or contact message.


You have been logged in successfully!

Welcome to MusicMind

Discover music that matches your personality

Answer a few questions about yourself and let us recommend music tailored to your personality traits.

Take the Questionnaire

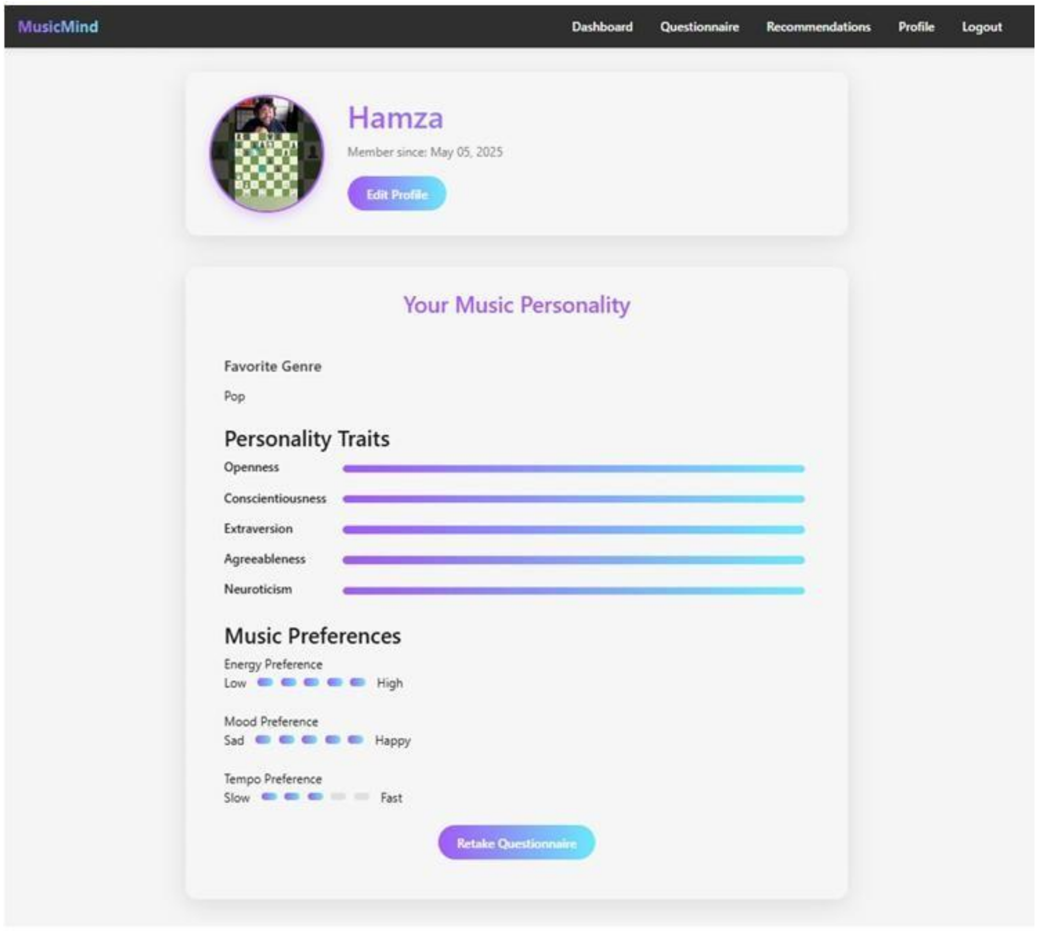
Complete our personality assessment to help us understand your music preferences.

View Recommendations

See your personalized music recommendations based on your personality profile.

Figure 11 User Dashboard

- **Profile Page**
  - Show saved personality traits (readable labels), history of previous recommendations.



Hamza

Member since: May 05, 2025

Your Music Personality

Favorite Genre

Pop

Personality Traits

Openness

Conscientiousness

Extraversion

Agreeableness

Neuroticism

Music Preferences

Energy Preference

Low ● ● ● ● ● High

Mood Preference

Sad ● ● ● ● ● Happy

Tempo Preference

Slow ● ● ● ○ ○ Fast

Figure 12 Profile Page

- **Admin Dashboard**
  - **Buttons / Tiles:** Drop the Beat (Add Song), SoundTracker (Edit/Delete), Graveyard, Spy on Fans, Firebase Documents.
  - **Table view:** song list with edit/delete actions.

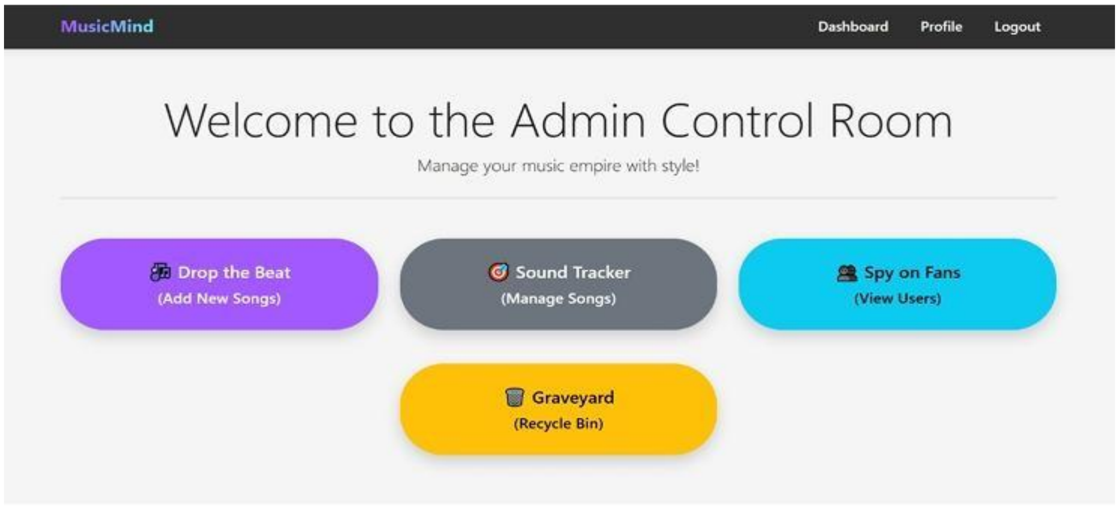

MusicMind
Dashboard
Profile
Logout
Welcome to the Admin Control Room
Manage your music empire with style!
Drop the Beat (Add New Songs)
Sound Tracker (Manage Songs)
Spy on Fans (View Users)
Graveyard (Recycle Bin)

Figure 13 Admin Dashboard

- **Wireframe Design:** We used simple grid layout; kept components consistent across pages. Colors and CSS are minimal for clarity.

17. Contributions

The following table shows the contributions of our group members in designing our application.

Table 17 Contributions

Group Member	Reg No.	Contributions
Syed Hamza Mukhtar Bukhari	2023682	Architectural design, DBService, Recommender design
Arsalan Khalil	2023130	Logical view & class diagrams
Riyan Durrani	2023611	Use case diagrams, activity diagrams
Adan Aamir	2023057	Sequence diagrams, process view
Momina Zahid	2023335	UI mockups, templates & documentation formatting