Software Requirements Specification (SRS)
Project Name: Music Database Management System
Date: 2024-11-12
Version: 1.0
Author: 565-Divyanshi Singh
564-Ananya Shroff
563-Shruti Srinivasan
572-Vaishnavi Waghole

---

1. Introduction

1.1 Purpose

This document specifies the requirements for the *Music Database Management System (MDBMS)*, which will manage data related to users, songs, playlists, albums, concerts, and venues. The MDBMS provides functionalities for adding, updating, deleting, and retrieving music-related data, using a Java Swing interface for user interactions.

1.2 Scope

The MDBMS will support SQL operations on various music-related entities, including users, songs, albums, playlists, and concerts. The system will enable data querying, view creation, triggers, and stored procedures, allowing efficient data handling. Java Swing will be used to develop a user-friendly interface for easier data interaction.

1.3 Definitions, Acronyms, and Abbreviations

- **SQL**: Structured Query Language
- **SRS**: Software Requirements Specification

- **CRUD**: Create, Read, Update, Delete operations
- **MDBMS**: Music Database Management System
- **Java Swing**: A Java toolkit for building graphical user interfaces (GUIs).

1.4 References

- SQL Documentation
- Database Management System Textbooks

1.5 Overview

This SRS provides a comprehensive guide for developers, administrators, and users, detailing the functional requirements, non-functional requirements, design constraints, and SQL queries employed within the MDBMS.

---

2. System Overview

The MDBMS manages extensive data related to music, users, and activities. Key functionalities include:

1. **User Management**: Adding, updating, and retrieving user data.
2. **Song and Playlist Management**: Handling song details, genres, and playlists.
3. **Album and Artist Management**: Storing album and artist information.
4. **Concert and Venue Management**: Organizing concert schedules and venues.
5. **Data Analytics**: Using SQL for data analysis, reporting, and user activity logging.

Java Swing will be used to create a graphical interface for these operations, providing an interactive and visually appealing user experience.

---

3. Functional Requirements

3.1 Database Management

3.1.1 Users Table

- **Description**: Stores user information.
- **Attributes**: `user_id`, `username`, `email`, `created_at`, `password`.
- **Operations**:
  - Insert a new user.
  - Update user information.
  - Delete user details.
  - Retrieve user data.

3.1.2 Songs Table

- **Description**: Stores song information.
- **Attributes**: `song_id`, `title`, `duration`, `album_id`, `artist_id`.
- **Operations**:
  - Insert a new song.
  - Retrieve song details.
  - Update song information.
  - Delete song records.

3.1.3 Albums Table

- **Description**: Stores album information associated with artists.
- **Attributes**: `album_id`, `title`, `release_date`, `artist_id`.

- **Operations**:
  - Insert a new album.
  - Retrieve album details.
  - Update album data.

3.1.4 Playlists Table

- **Description**: Manages playlists and tracks associated songs.
- **Attributes**: `playlist_id`, `name`, `user_id`, `created_at`.
- **Operations**:
  - Create a new playlist.
  - Retrieve playlist details.
  - Add songs to a playlist.
  - Remove songs from a playlist.

3.1.5 Concerts and Venues

- **Description**: Stores concert schedules and venue details.
- **Attributes**: `concert_id`, `concert_date`, `venue_id`, `ticket_price`.
- **Operations**:
  - Insert concert details.
  - Retrieve concert and venue information.
  - Update concert schedules.

---

4. Non-Functional Requirements

- **Usability**: The system should be user-friendly, using Java Swing to provide a GUI for seamless data management.
- **Performance**: Fast and reliable responses to SQL queries, ensuring smooth operation for large datasets.

- **Reliability**: Consistent performance with minimal errors. Data should remain accurate and up-to-date.
- **Security**: User passwords must be securely stored, with sensitive data access restricted.
- **Maintainability**: The database schema should be structured to allow easy updates and scalability.

---

5. Design Constraints

- **Technology Stack**: Java for application logic, Java Swing for GUI, MySQL for data storage.
- **Database Connection**: Uses JDBC to connect Java applications with the MySQL database.
- **Error Handling**: System should provide clear error messages for database connection or query failures.

---

6. Database Queries and Stored Procedures

The following SQL queries and procedures are implemented in MDBMS:

- **Insert User**: Adds a new user to the `users` table.
- **Retrieve Songs**: Displays all song records from the `songs` table.
- **Insert Playlist**: Adds a new playlist record to the `playlists` table.
- **Concert Details**: Retrieves all scheduled concerts and associated venues.

---

7. Appendix

Data Flow Diagrams (DFD)

- **Level 1**: Shows data flow between users and music-related entities (songs, albums, playlists, concerts).
- **Level 2**: Detailed view of CRUD operations on each entity.

ER Diagrams

- Entity-Relationship diagrams provide a visual representation of how the MDBMS entities are interconnected.

Sample SQL Statements

- Sample statements for CRUD operations and stored procedures.

```
--  Users Table


-- Insert User

INSERT INTO Users (username, email, created_at, password)

VALUES ('JohnDoe', 'john@example.com', CURDATE(),
'encrypted_password');


-- Retrieve User by ID

SELECT * FROM Users WHERE user_id = 1;


-- Update User Information
```

```
UPDATE Users
SET email = 'new_email@example.com'
WHERE user_id = 1;

-- Delete User
DELETE FROM Users WHERE user_id = 1;

--  Songs Table

-- Add Song
INSERT INTO Songs (title, duration, album_id, artist_id)
VALUES ('Song Title', '00:03:30', 1, 1);

-- Retrieve All Songs
SELECT * FROM Songs;

-- Update Song Details
UPDATE Songs
SET title = 'New Song Title'
WHERE song_id = 1;
```

```
-- Delete Song
DELETE FROM Songs WHERE song_id = 1;

--  Albums Table

-- Add Album
INSERT INTO Albums (title, release_date, artist_id)
VALUES ('Album Title', '2024-11-12', 1);

-- Retrieve Album by ID
SELECT * FROM Albums WHERE album_id = 1;

-- Update Album Details
UPDATE Albums
SET title = 'New Album Title'
WHERE album_id = 1;

-- Delete Album
DELETE FROM Albums WHERE album_id = 1;
```

```
--  Playlists Table

-- Create Playlist
INSERT INTO Playlists (name, user_id, created_at)
VALUES ('My Playlist', 1, CURDATE());

-- Retrieve Playlist
SELECT * FROM Playlists WHERE playlist_id = 1;

-- Add Song to Playlist
INSERT INTO Playlist_Songs (playlist_id, song_id)
VALUES (1, 1);

-- Remove Song from Playlist
DELETE FROM Playlist_Songs WHERE playlist_id = 1 AND song_id
= 1;

--  Concerts and Venues Table

-- Add Concert
```

```
INSERT INTO Concerts (concert_date, venue_id, ticket_price)
VALUES ('2024-12-25', 1, 50.00);

-- Retrieve Concerts
SELECT * FROM Concerts;

-- Update Concert Details
UPDATE Concerts
SET concert_date = '2024-12-31'
WHERE concert_id = 1;

-- Delete Concert
DELETE FROM Concerts WHERE concert_id = 1;
```

Glossary

- **Database**: An organized collection of structured information.
- **Entity**: A distinct data item in the system, such as a user or song.
- **JDBC**: Java Database Connectivity, a standard for connecting Java applications with databases.

- **Java Swing**: A GUI toolkit for Java applications to create graphical interfaces.