

Team 5  
Final Project  
Documentation

## **Table of Contents**

Project Proposal	2
High Level Requirements	3
Technical Specifications	4
Detailed Design	6
Testing	10
Deployment	12

**Project Proposal**

The idea is to create a website where users can input a movie name, and it will generate songs that match its vibe. Users can also input a song, and the site will recommend a movie they might enjoy.

## High Level Requirements

### *Sign Up / Log In*

We are creating a web interface where users can sign up with a username and password, and we will store that information in our users database. They can then log in with that information, and we will validate that with existing users to make sure that they are a valid existing user.

### *Connect Music Account*

If the user is new, we will call an external authentication API to connect their account on our web platform with a music streaming service (Last.fm) If they are a returning user, they should be able to see the account they previously connected to in a past session and also modify it (disconnect that account and/or add a new one).

### *Movie → Song*

Users will be able to input a media source (books, movie, etc.) into the portal. The program will then process the known information about the source and create a call to the Last.fm API to find related music in order to create a playlist that can enhance the experience.

### *Song → Movie*

Alternatively, users can choose to upload a song they enjoy. They will then be able to receive recommendations on a media source (books, movie, etc.) that they might enjoy based off of the input.

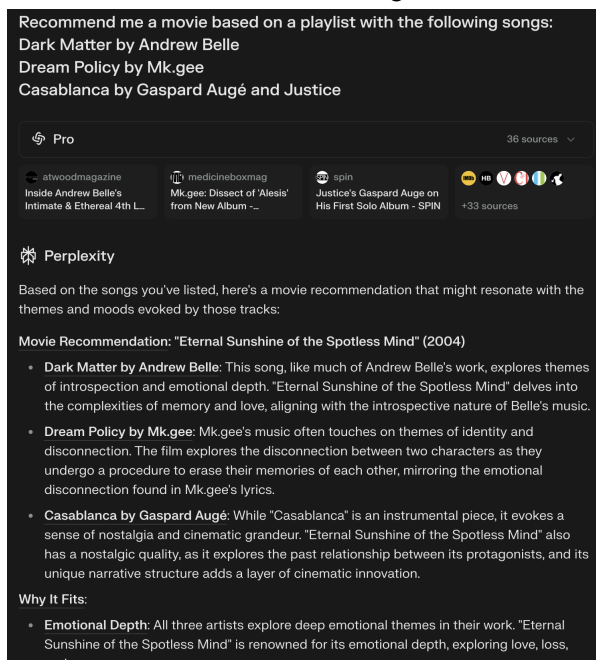
## Technical Specifications

### Web Interface (4 hours)

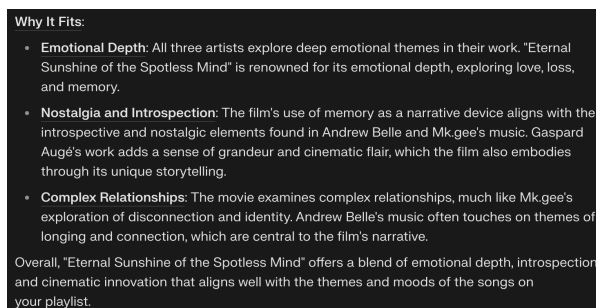
- Login page where users enter a username and password, press submit for verification
  - ◆ On the first time logging in/signing up they would need to connect their Last.fm account
- Once verified, redirected to a page that lets them select if they are entering a song or movie title
  - ◆ If **“song”** is selected, the user is prompted to enter a song name and artist name.
  - ◆ If **“movie”** is selected, the user is prompted to enter the title into the web application.

### Playlist In, Media Source Out (8 hours)

- The Last.fm API is used to get the tags associated with the song so that the genres can be fed into Gemini in order to get movies that fit the vibe.



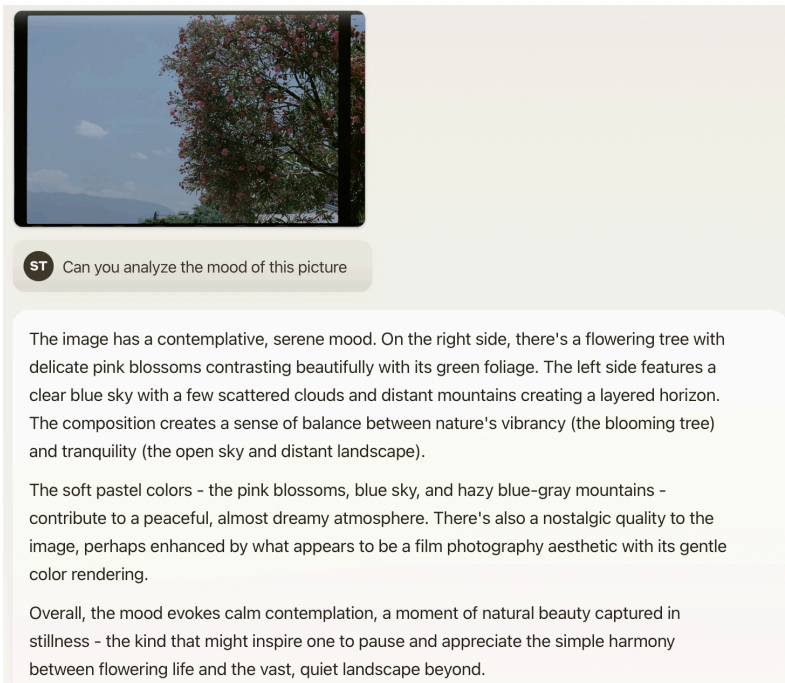
→



- Display the result with a summary related to why a certain piece of media was chosen (details based on the track's audio features, online recommendation content, and more).
- Utilize multithreading to process multiple queries at the same time

### *Media Source In, Playlist Out (8 hours)*

- Use Gemini AI to analyze the sentiment of the inputted movie title and recommend the user songs based on their chosen movie.

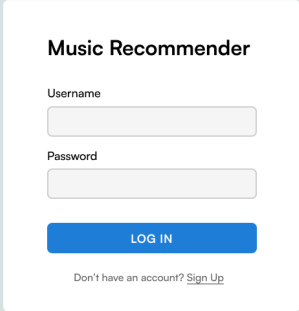


- Using Last.fm, the program will find songs listed under the tags that match the description of the movie in order to find a good match for the request.
- The program can use multi-threading to handle multiple requests for new playlists from the user at the same time.

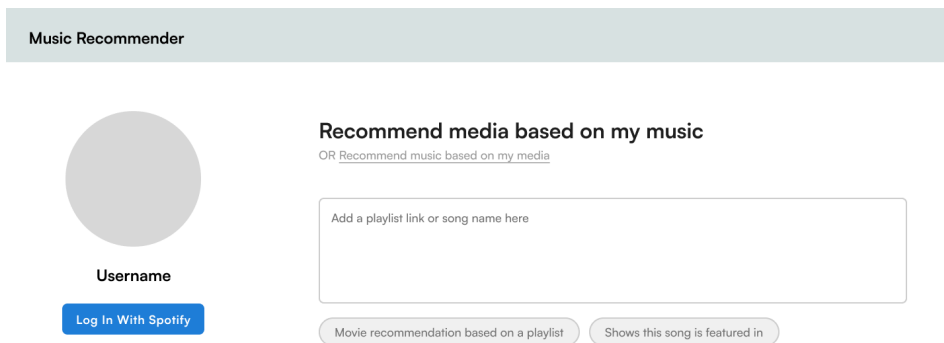
### *Database (2 hours)*

- User table for login. Consists of username, password, userID, firstName, lastName, and associated Last.fm account.
- Queries table that will store past user inputs and the program's outputs.

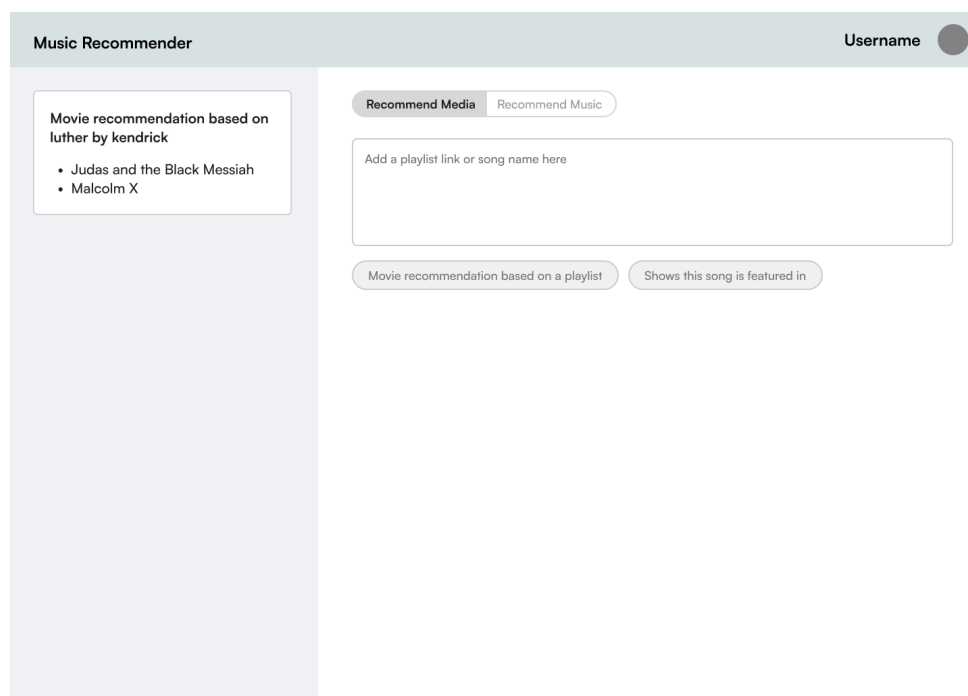
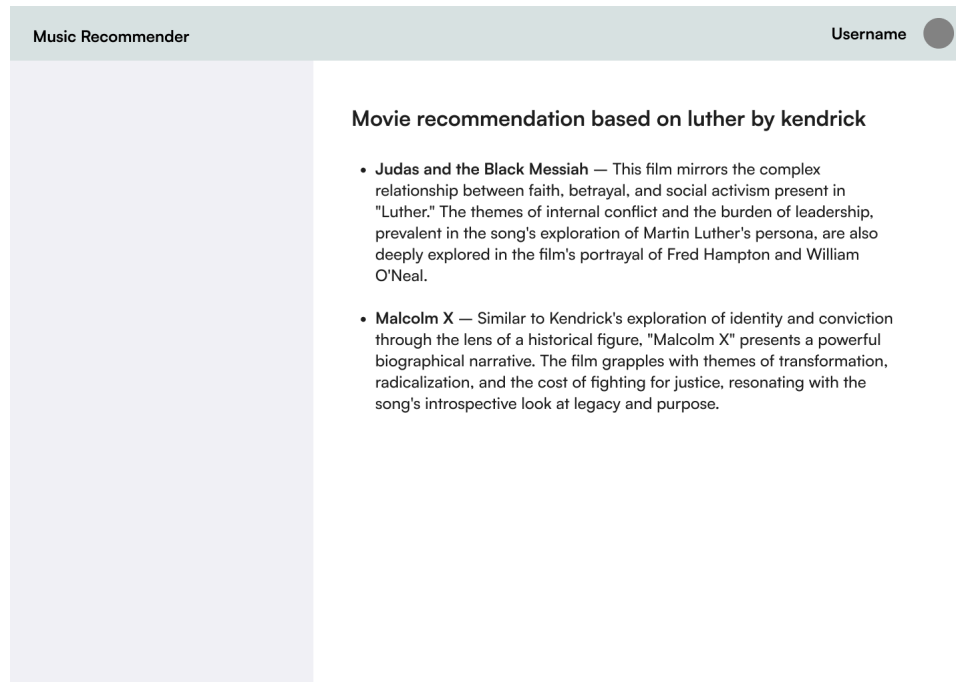
**Detailed Design**  
*GUI / Design*  
[Figma File Link](#)



A login form titled "Music Recommender" centered on a light gray background. The form contains a "Username" label above a text input field, a "Password" label above another text input field, a blue "LOG IN" button, and a link "Don't have an account? [Sign Up](#)" at the bottom.



A user interface for the "Music Recommender" application. It features a header "Music Recommender" on a light gray bar. Below the header, on the left, is a gray circular profile picture placeholder labeled "Username" with a blue "Log In With Spotify" button underneath. To the right, the main heading is "Recommend media based on my music", followed by a subtext "OR Recommend music based on my media". Below this is a large text input field with the placeholder "Add a playlist link or song name here". At the bottom, there are two buttons: "Movie recommendation based on a playlist" and "Shows this song is featured in".



## Database

The project will use MySQL to create and manage the primary database  
The database would support a web application that allows users to:

- Sign-up and log in with username/password
- Connect to music streaming services



- Generate playlists from media sources
- Get media recommendations from playlists
- Store user history

Schema:

- User Table: This table stores user authentication data and their associated music streaming account information. The primary key for this table is user\_id
- MusicAccounts Table: This table stores connections to music streaming service accounts. Primary key is account\_id and foreign key is user\_id
- Query Table: This table stores past user inputs and corresponding program outputs for recommendation tracking. The primary key is query\_id and the foreign key is user\_id
- MediaRecommendations Table: This table stores recommendations generated for users. The primary key is recommendation\_id and the foreign keys are user\_id and query\_id

The database architecture will incorporate transaction management protocols to ensure ACID compliance, particularly important when multiple users are generating recommendations simultaneously. The system will also incorporate soft deletion mechanisms for user content, allowing recovery of accidentally removed playlists or recommendations within a configurable time window before permanent removal from storage.

#### *Software Requirements*

- Front end: HTML/CSS/JavaScript
- Back end: Java, JDBC
- Database: MySQL
- APIs: Last.fm, Gemini

Login:

- authenticate() method will take two strings as parameters, representing the username and password
  - Returns true if both username and password match a pair in the User table
  - If username exists but password doesn't match → prompt user to try again or reset password
  - If username doesn't exist → prompt user to sign up/create account
- On the first time a user logs in, ask them to authorize the connection to their Last.fm account, use API authorization code flow

Song Upload:

- Take in song and artist name
- Send GET request to Last.fm API to retrieve the song and parse the tags
  - Store the song and the tags in the user's Query table
- UploadSong class takes the values into the program and ask it for a movie that matches
  - POST request to Gemini that prompts it to recommend the user a movie
  - Prompt should include:
    - The tags associated with the inputted song

- What the return format should look like (in JSON):
      - Title: string
      - Reason for choosing: string
    - Store the output in the Query table
  - Send information back to the front end to be displayed

#### Movie Upload:

- Take in movie title as search parameter
- Determine related keywords by sending request to Gemini
- GET request to Last.fm API to retrieve songs related to keywords
- UploadMovie class takes the movie name in the program and returns a list of songs
  - Prompt should include:
    - The tags associated with the inputted song
    - What the return format should look like (in JSON):
      - Title: string
      - Reason for choosing: string
  - Store the output in the Query table
- Send information back to the front end to be displayed

#### *Class Diagram/Inheritance Hierarchy*

1. User Class
  - a. UserID: str
  - b. Username: str
  - c. firstName: str
  - d. lastName: str
  - e. spotifyAccount: str
  - f. queryHistory: list[Query]
2. Query Class
  - a. queryID: str
  - b. userID: str
  - c. Timestamp: datetime
  - d. Result: str
3. PlaylistQuery
  - a. playlistID: str
  - b. trackData: dict[str, float]

## Testing

### *Login / Sign Up*

#### Unit Testing:

- Valid Credentials: Input a correct username & password and get success response
- Invalid Password: Correct username with the wrong password, get auth failure
- Nonexistent User: Username not in database, user not found error
- Empty Fields: Empty username and/or password, validation error
- Duplicate Username: Username already exists, show that username is taken

#### White box testing:

- New User In Database: Check whether an input to the Sign Up form adds a new user row to the Users database
- Authentication(): Verify each branch in the logic (correct user/pw, username exists with incorrect password, non-existent username)
- Validate that on the first login, the flow correctly starts the Last.fm API auth code

### *Last.fm Connection*

- White Box Testing:
  - Authorization URL: Tests if login generates a valid Last.fm authentication response
  - Getting Auth Token: Checks if the authentication token from Last.fm has been received after a user logs in and the user is redirected to the callback URL
  - Persistent Access: Checks if the app still has access to the Last.fm key after logging back in
  - Authentication success case: subsequent function calls work expectedly
- Blackbox
  - Authentication failure case: is the error message helpful, does the program crash
  - Authentication success case: successfully authenticated message is printed, user is asked the next action
  - Simulate network failure during user login, check program doesn't crash

### *Song Input*

- Black Box testing:
  - Input name of a song that exists
    - This should return recommendations based on the song's information.
  - Input song that does not exist
    - This should inform the user that the song cannot be found and thus, no recommendations can be made. Re-prompt the user for input.
  - Submit blank input
    - This should inform the user that no information was inputted and thus, no recommendations can be made. Re-prompt the user for input.
- White Box testing:
  - Check user authentication

- Make call to API, if user is not properly authenticated, return UserAuthentication error
- Test API calls to Last.fm
  - Ensure that calls to the API are properly formatted, print to console as needed
  - Check for empty headers

#### *Gemini API Calls*

- Black Box Testing:
  - Last.fm/Gemini integration: Simulate user actions by entering a Last.fm playlist URL and confirm
    - system sends proper GET request
    - returned data is stored
    - POST request to Gemini returns a media recommendation in expected format
- White Box Testing:
  - Input Validation (empty input): verify that it raises appropriate exceptions when called with empty/missing prompts.
  - Request formation: check that the method constructs the correct POST request using proper API URL
  - Response handling: Make sure that cases of all successful, missing, and error responses are all handled correctly.

#### *Database:*

- White Box testing:
  - Request user's previous movie recommendation history
    - Should be stored in the MediaRecommendations table, which should print out on request.
  - Request user's previous movie recommendation history
    - Should be stored in the MusicRecommendations table, which should print out on request.

## Deployment

Production Environment: Make sure Java, Tomcat, and MySQL are installed and accessible

- Configure the LastFM API credentials:
  - lastfm.api.key=YOUR\_LASTFM\_API\_KEY
  - lastfm.api.secret=YOUR\_LASTFM\_API\_SECRET
- Database configuration
  - jdbc:mysql://127.0.0.1:3306/FinalProject
    - Username: user
    - Password: Lucas20021214=

Run application: start on Tomcat server

Verify application is running: Go to [http://localhost:8080/CSCI201\\_FinalProject/index.html](http://localhost:8080/CSCI201_FinalProject/index.html) (or your configured port, if different)

Regression Test:

- Ensure that linking the Last.fm account stores the connection to the user's account
- Check that logging in a second time saves the user's account information/past queries
- Check that pressing Login brings the user to the correct landing page (where they can upload a song)
- Check that submitting a song/media brings the user to a results page