

DEVELOPING A MUSIC STREAMING WEBSITE WITH EXPLICIT CONTENT  
FILTERING AND SONG RECOGNITION

A THESIS SUBMITTED TO  
THE FACULTY OF ARCHITECTURE AND ENGINEERING  
OF  
EPOKA UNIVERSITY

BY

ENGJËLL ABAZAJ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR BACHELOR DEGREE  
IN SOFTWARE ENGINEERING

MAY, 2025

**I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.**

Name, Last name: Engjëll Abazaj

Signature:

# **ABSTRACT**

## **DEVELOPING A MUSIC STREAMING WEBSITE WITH EXPLICIT CONTENT FILTERING AND SONG RECOGNITION**

Abazaj, Engjëll

B.Sc., Department of Computer Engineering

Supervisor: M.Sc. Ari Gjerazi

This thesis presents a music streaming web application developed to demonstrate the implementation of explicit content filtering and song recognition capabilities. Aimed at enhancing user safety and control in digital audio consumption, the platform integrates audio analysis and censorship features within a fully functional streaming environment.

The application is built using a modern technology stack: Next.js for both frontend and backend development, Supabase for authentication and data storage, and Python with Flask for server-side audio processing. The core functionality leverages OpenAI's Whisper model to transcribe uploaded audio and identify explicit content using the NeutrinoAPI (NeutrinoAPI, n.d.). Detected explicit segments are censored by muting the corresponding timestamps in the audio, with optimized handling of overlapping mute intervals to ensure playback fluidity.

An additional feature includes song recognition, enabled via the AudD API (AudD, n.d.). Users can record audio snippets through their browser, which are then processed to retrieve metadata about the song. The application supports user authentication (via credentials or GitHub), personal playlists, song uploads, playback controls, and genre-based search.

The primary objective of the project was the development of a robust, automated audio censorship pipeline. A key technical challenge, aligning transcription data with precise audio segments, was resolved through iterative refinement of the timestamp mapping process. Although no formal user studies were conducted, functional testing validated the effectiveness of all features.

This work demonstrates the potential of integrating AI-powered transcription and content moderation tools into streaming platforms. It provides a proof-of-concept for scalable, privacy-aware music applications that prioritize both functionality and user experience.

**Keywords:** Music Streaming; Content Filtering; Speech Recognition; Web Development; Audio Processing

# **ABSTRAKT**

## **ZHVILLIMI I NJË FAQEJE WEB PËR TRANSMETIM MUZIKE ME FILTRIM PËR PËRMBAJTJE EKSPLICITE DHE NJOHJE KËNGËSH**

Abazaj, Engjëll

B.Sc., Department of Computer Engineering

Supervisor: M.Sc. Ari Gjerazi

Kjo temë paraqet një aplikacion web për transmetim muzike, i zhvilluar për të demonstruar zbatimin e filtrimit të përmbajtjes eksplicite dhe aftësive të njohjes së këngëve. Me synimin për të rritur sigurinë dhe kontrollin e përdoruesit në konsumin dixhital të audios, platforma integron analiza audio dhe veçori të censurimit brenda një mjedisi funksional të transmetimit.

Aplikacioni është ndërtuar duke përdorur një teknologji moderne: Next.js për zhvillimin e ndërfaqes dhe pjesës së pasme, Supabase për autentifikim dhe ruajtjen e të dhënave, dhe Python me Flask për përpunimin e audios në anën e serverit. Funksionaliteti kryesor përdor modelin Whisper të OpenAI për të transkriptuar audion e ngarkuar dhe për të identifikuar përmbajtje eksplicite duke përdorur NeutrinoAPI. Segmentet e identifikuara si eksplicite censurohen duke heshtur pjesët përkatëse të

audios, me një algoritëm të optimizuar për të menaxhuar mbivendosjet e intervaleve të heshtjes dhe për të siguruar rrjedhshmëri gjatë dëgjimit.

Një veçori shtesë përfshin njohjen e këngëve, e mundësuar përmes AudD API. Përdoruesit mund të regjistrojnë fragmente audio përmes shfletuesit të tyre, të cilat përpunohen për të marrë metainformacione rreth këngës. Aplikacioni mbështet autentifikimin e përdoruesve (me kredenciale ose përmes GitHub), menaxhimin e listave personale të dëgjimit, ngarkimin e këngëve, kontrollet e riprodhimit dhe kërkimin sipas zhanrit.

Qëllimi parësor i projektit ishte zhvillimi i një zinxhiri të automatizuar dhe të qëndrueshëm për censurimin e audios. Një sfidë teknike kryesore, përputhja e të dhënave të transkriptimit me segmentet e sakta audio, u zgjidh përmes përmirësimeve të përsëritura në procesin e hartimit të intervaleve kohore.

Ky punim demonstroi potencialin e integritet të mjeteve të transkriptimit dhe moderimit të përmbajtjes të bazuara në inteligjencën artificiale në platformat e transmetimit muzikor. Ai ofron një provë konceptuale për aplikacione muzikore të shkallëzueshme dhe të vetëdijshme për privatësinë, që japin përparësi funksionalitetit dhe përvojës së përdoruesit.

**Fjalë kyçe:** Transmetim Muzike; Filtrim Përmbajtjeje; Njohje e të Folurit; Zhvillim Web; Përpunim Audio

*Dedicated to my family, friends, and mentor for their unwavering support and guidance, and to my personal determination that made this thesis possible.*

## **ACKNOWLEDGEMENTS**

I would like to express my sincere gratitude to everyone who supported me throughout the course of this thesis and my academic journey.

First and foremost, I am deeply thankful to my thesis advisor, M.Sc. Ari Gjerazi, for his guidance, mentorship, and constructive feedback throughout the development of this work. His expertise and support played a crucial role in the successful completion of this project.

I also extend my heartfelt appreciation to Professors Arban Uka, Sabrina Begaj, Igli Draçi, Florenc Skuka, and Bekir Karlik for their contributions to my academic growth. Their teaching and support have significantly shaped my understanding and passion for the field.

I am grateful to my friends for their continuous encouragement and moral support. Their presence and motivation were essential during the more challenging moments of this journey.

Above all, I would like to thank my parents, Hikmet Abazaj and Lindita Abazaj, for their unwavering love, patience, and belief in me. Their constant support has been the foundation of all my achievements, and I dedicate this milestone to them.



# TABLE OF CONTENTS

ABSTRACT .....	iii
ABSTRAKT .....	<b>Error! Bookmark not defined.</b>
ACKNOWLEDGEMENTS .....	viii
LIST OF TABLES .....	xii
LIST OF FIGURES .....	xiii
LIST OF APPENDICES .....	xiii
LIST OF ABBREVIATIONS .....	xiv
CHAPTER 1 .....	1
INTRODUCTION .....	1
1.1. Background and Motivation .....	<b>Error! Bookmark not defined.</b>
1.2. Existing Solutions and Their Limitations .....	2
1.3. Educational and Development Context .....	<b>Error! Bookmark not defined.</b>
1.4. Aim and Novelty of the Study .....	3
1.5. Practical Contribution .....	4
CHAPTER 2 .....	5
MATERIALS AND METHODS .....	5
2.1. Materials .....	5
2.1.1. Software and Libraries .....	5
2.1.2. APIs and Cloud Services .....	6
2.1.3. Hardware and Execution Environment .....	7
2.2. Methods .....	8
2.2.1. User Workflow and Interaction Design .....	9
2.2.2. Audio Censorship via Whisper and NeutrinoAPI .....	10
2.2.3. Song Recognition via Browser Microphone .....	12
2.2.4. Testing and Validation .....	14

2.2.5. Deployment and Execution Context .....	14
CHAPTER 3 .....	15
RESULTS AND DISCUSSION .....	15
3.1. Platform Functionality Evaluation .....	15
3.1.1. Audio Upload, Playback, and Interface Usability .....	15
3.1.2. Playlist Features and Song Organization .....	16
3.1.3. Audio Filtering (Censorship) Feature Evaluation .....	17
3.1.4. Song Recognition Accuracy and Robustness.....	17
3.1.5. Error Management and Edge Cases .....	18
3.2. Audio Filtering and Recognition Results Evaluation .....	19
3.2.1. Performance of Audio Censorship Pipeline.....	19
3.2.2. Fidelity of Audio Output.....	20
3.2.3. Song Recognition Insights .....	20
3.3. Discussion .....	21
3.3.1. Alignment with Objectives .....	21
3.3.2. System Limitations .....	22
3.3.3. Proposed Future Enhancements.....	22
CHAPTER 4 .....	24
CONCLUSION .....	24
4.1. Summary of Contributions.....	24
4.2. Future Improvements .....	25
REFERENCES.....	27
APPENDIX A .....	28
ACTIVITY AND SEQUENCE DIAGRAMS .....	28
APPENDIX B .....	34
APPLICATION INTERFACE AND FUNCTIONALITY DEMONSTRATION ....	34

## LIST OF TABLES

### TABLES

Table 1. Performance Comparison of Whisper Model on CPU vs GPU.....	19
Table 2. Accuracy of Explicit Content Detection Using Whisper and NeutrinoAPI....	20
Table 3. Summary of Song Recognition Tests Using AudD API.....	21

## LIST OF FIGURES

### FIGURES

Figure 2.1. Supabase Database Schema (ERD).....	8
Figure 2.2. System Architecture Diagram.....	10
Figure 2.3. Data Flow Diagram for Audio Filtering Pipeline.....	12
Figure A.1. Activity Diagram – Audio Filtering Workflow.....	28
Figure A.2. Sequence Diagram – Audio Filtering Workflow.....	29
Figure A.3. Activity Diagram – Song Recognition.....	30
Figure A.4. Sequence Diagram – Song Recognition.....	31
Figure A.5. Activity Diagram – Playlist Management.....	32
Figure A.6. Sequence Diagram – Audio Filtering Workflow.....	33
Figure B.1. Homepage Interface Showing Audio Player and Song Display.....	34
Figure B.2. Song Upload Modal with Metadata Inputs.....	35
Figure B.3. Playlist Creation Modal.....	35
Figure B.4. Liked Songs Playlist Display.....	36
Figure B.5. Special Features Page Showing Filtering and Recognition Tools.....	36
Figure B.6. Filtered Song Download.....	37
Figure B.7. Song Metadata Retrieved Using AudD API.....	37

## LIST OF APPENDICES

Appendix A: Activity and Sequence Diagrams.....	<b>Error! Bookmark not defined.</b>
Appendix B: Application Interface and Functionality Demonstration.....	34

## **LIST OF ABBREVIATIONS**

API	Application Programming Interface
CSS	Cascading Style Sheets
CUDA	Compute Unified Device Architecture
ERD	Entity-Relationship Diagram
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MP3	MPEG Audio Layer III
UI	User Interface
WAV	Waveform Audio File Format

# CHAPTER 1

## INTRODUCTION

### 1.1. Background and Motivation

Music streaming platforms have transformed how individuals and businesses consume audio content. Services like Spotify are widely used not only for personal entertainment but also in commercial environments such as restaurants, barbershops, gyms, and retail stores. Despite their versatility, a common shortcoming in these platforms is the absence of integrated tools for managing explicit content.

Currently, platforms rely on metadata from rights-holders to label explicit tracks, allowing users to hide such content via built-in settings (Spotify, n.d.). However, this approach does not modify or censor the tracks themselves and often requires users to manually identify and select clean versions of songs. This process can be inefficient and unreliable, especially in professional environments where appropriate content is essential for maintaining a respectful atmosphere.

The motivation for this project stems from a personal and practical interest in music consumption. Observing the lack of integrated filtering solutions in mainstream services, the goal was to build a platform that simplifies content moderation. Rather than expecting users to rely on multiple tools or search for edited tracks manually, the application introduces a built-in feature allowing users to upload and clean their own

music with minimal effort. This provides both individuals and small businesses with a more streamlined way to manage audio content.

## **1.2. Existing Solutions and Their Limitations**

Mainstream streaming platforms, including Spotify and Apple Music, offer basic content control mechanisms, typically through the use of "explicit" labels and filter toggles (Soundplate, n.d.). These labels are dependent on publisher-supplied metadata, which is sometimes incomplete or inconsistent. Users must often navigate between different versions of the same track, explicit and clean, without clear indicators or assurances that the selected version is appropriate.

Song recognition technology, such as that provided by Shazam, has advanced significantly and allows users to identify music quickly and accurately through short recordings. Shazam's system works by generating an audio fingerprint and matching it against a large database (Wang, 2003). While such tools are highly effective for identification purposes, they do not address content moderation or enable content manipulation in any way.

Academic research has explored the use of machine learning models, such as convolutional neural networks, for tagging and classifying music content (Choi, Fazekas, & Sandler, 2016). However, these approaches are generally implemented in isolated applications or research environments and have not been widely adopted in commercial platforms for explicit content filtering.

In summary, while content identification and classification are well-supported through existing solutions, an integrated system that both recognizes and filters music remains largely absent from the mainstream market.



### **1.3. Educational and Development Context**

This project was conducted as part of a Graduation Project in Bachelor in Software Engineering. The topic, design, and development process were independently selected and supervised under academic guidance. While developed within an academic context, the concept was also driven by a genuine interest in addressing a real-world usability gap in the domain of music streaming.

The project aims to provide a demonstration of how content filtering functionality can be made accessible through a web interface without requiring technical expertise from users. It also serves as a practical exploration of how artificial intelligence and web technologies can be applied to everyday digital services in a privacy-conscious and user-friendly way.

### **1.4. Aim and Novelty of the Study**

The central aim of this study is to design a music streaming web application that incorporates user-driven explicit content filtering and song recognition capabilities. The key novelty lies in the integration of these features into a single platform. Unlike existing systems that require multiple tools or rely on external applications, this solution allows users to upload a song, clean it using built-in functionality, and download a sanitized version—all from one interface.

Although the filtering process is initiated manually by the user, its seamless integration into the platform provides a level of convenience and usability that is lacking in current offerings. This feature is especially beneficial for small businesses and other settings where content appropriateness is a priority.

## **1.5. Practical Contribution**

This work provides a proof-of-concept for a scalable and user-oriented music application that incorporates features not currently available in mainstream platforms. It demonstrates how content recognition and moderation can be combined into a cohesive service that enhances control over audio consumption.

By focusing on accessibility and practicality, the project contributes to ongoing discussions about how artificial intelligence and digital tools can be applied to improve user experiences in everyday technologies. The outcome supports the idea that responsible content handling should be an integral part of music delivery platforms, especially in contexts where content sensitivity matters.

## CHAPTER 2

### MATERIALS AND METHODS

#### 2.1. Materials

The development of this music streaming platform with explicit content filtering and song recognition required a combination of frontend frameworks, backend services, APIs, and machine learning tools. This section details all major components and resources used in the implementation and evaluation of the system.

##### 2.1.1. Software and Libraries

The platform was developed as a full-stack web application using *Next.js v15.3.1* (Vercel Inc., USA), a modern framework that enables both client-side and server-side rendering of web pages. Although the development was carried out exclusively within the Next.js ecosystem, it inherently depends on *React v19.0.0* (Meta Platforms Inc., USA) for building component-based user interfaces, and *Node.js* (OpenJS Foundation, USA) for backend execution and server-side logic.

The frontend interface was styled using *Tailwind CSS v4.1.4* (Tailwind Labs Inc., USA), a utility-first CSS framework that enabled rapid development of responsive and aesthetically consistent UI elements. Component-level interactions and animations were

implemented using libraries such as *Radix UI*, *Framer Motion*, and *Lucide React*. Form controls, icons, and feedback elements were further enhanced using *React Icons*, *React Hot Toast*, and *React Hook Form*.

On the backend, user authentication, file storage, and database management were facilitated using *Supabase Core v2.22.4* and related helper packages for Next.js and React (Supabase Inc., USA). Supabase was selected primarily for its real-time PostgreSQL-based database, integrated file storage (for audio and image files), and seamless compatibility with the Next.js environment.

For AI-driven audio processing, a separate *Python* microservice was implemented using *Flask v3.1.0* (Pallets Projects, USA). The microservice used *OpenAI Whisper* “tiny” model, an advanced speech recognition model, for transcribing audio files into text. To accelerate inference, *PyTorch v2.6.0+cu118* and *TorchAudio v2.6.0+cu118* (Meta AI, USA) were used alongside CUDA-enabled GPU hardware. Audio processing tasks such as file conversion, segmentation, and silence insertion were handled using *PyDub v0.25.1* and *SoundFile v0.13.1*.

Additional JavaScript libraries used in the frontend include *Zustand v5.0.3* for state management, *React Query v5.75.5* for asynchronous data fetching and caching, and *Query String v9.1.1* for URL parameter manipulation. Type safety and development productivity were enhanced by using *TypeScript v5* (Microsoft Corp., USA) throughout the application.

### **2.1.2. APIs and Cloud Services**

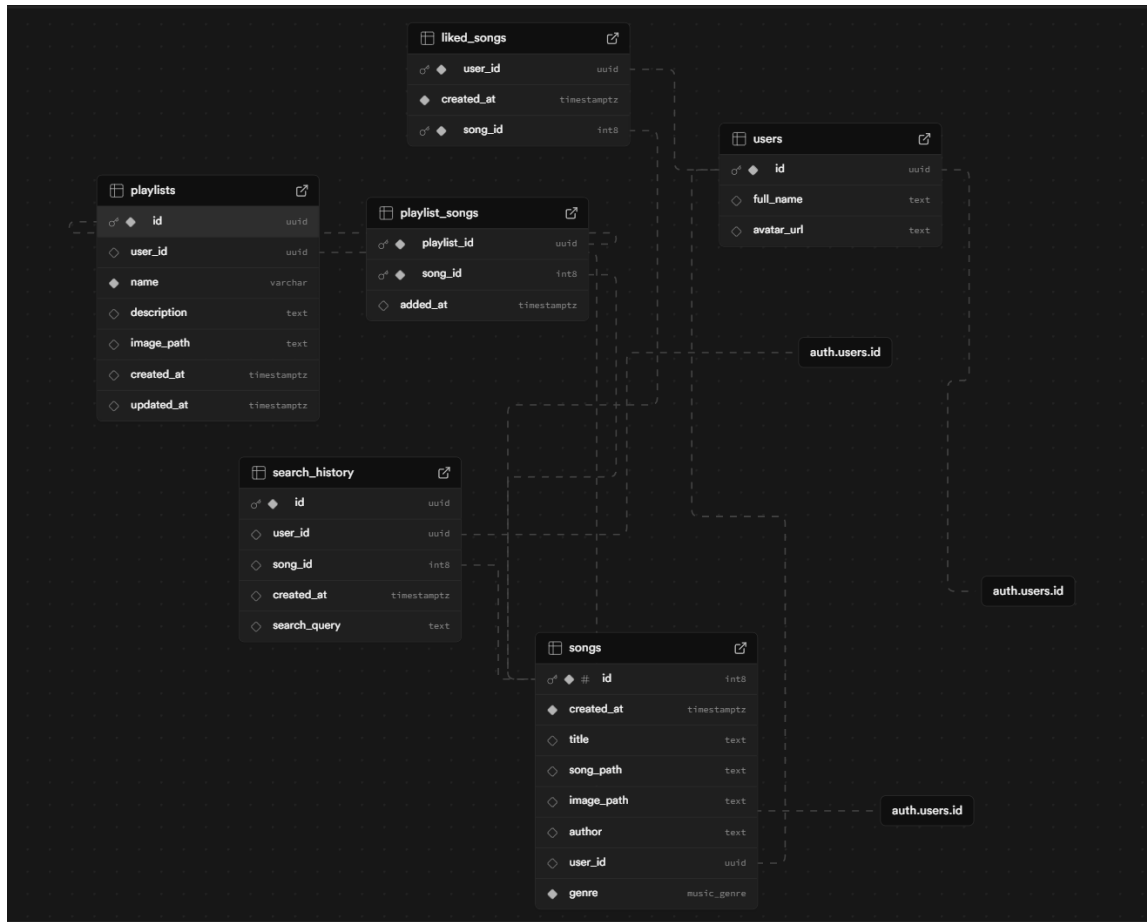
Two external APIs were integrated into the platform to provide intelligent functionality:

- NeutrinoAPI (RapidAPI, USA): Utilized for detecting offensive language in transcribed text. Specifically, the Bad Word Filter endpoint was used to identify and censor inappropriate words found in song lyrics.
- AudD API (AudD Music Recognition, USA): Used to identify song metadata based on short recorded audio samples. The API returns information such as song title, artist, and album after analyzing the acoustic fingerprint.

In addition, Supabase Storage was used as a cloud-hosted storage solution for uploaded audio files and song images. The Supabase instance operated as the backend for the platform, enabling remote access and persistent storage of user-generated content even though the application itself was not publicly deployed.

### **2.1.3. Hardware and Execution Environment**

Development and testing were conducted on an Asus Zephyrus G14 laptop equipped with an NVIDIA GeForce RTX 4060 GPU with 8GB VRAM supporting CUDA 12.8, which significantly accelerated the Whisper model's performance during transcription. The local development environment was based on Windows PowerShell (Microsoft Corporation, USA), using browser-based testing tools such as Google Chrome for frontend validation. The application was not deployed to a live server but was demonstrated locally for evaluation and review.



**Figure 2.1:** Supabase Database Schema (ERD)

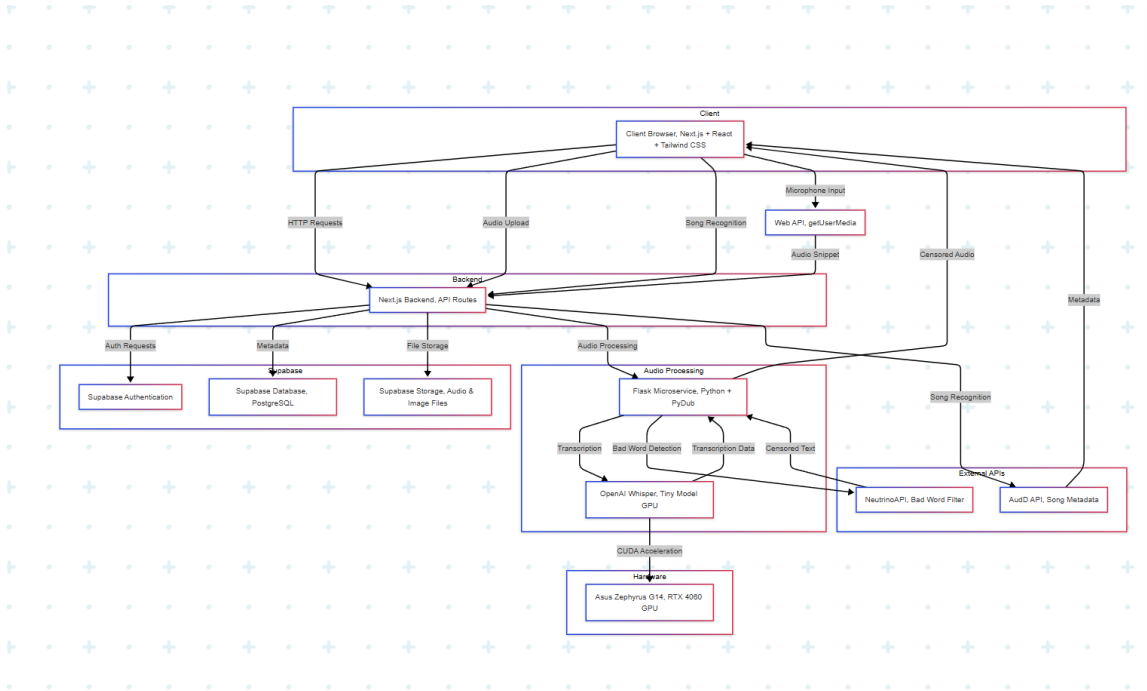
## 2.2. Methods

This section details the methodologies used to implement, test, and demonstrate the functionality of the web application. The system comprises several integrated components supporting user interaction, audio censorship, playlist management, and song recognition.

### **2.2.1. User Workflow and Interaction Design**

The platform is structured around an intuitive user interface with sidebar navigation. Users can interact with the system through various features, including uploading songs, creating playlists, managing audio playback, and accessing additional audio processing tools.

- **Uploading Songs:** Users click the "Upload Song" icon in the sidebar, which opens a modal form. The form collects the song title, artist name, and genre from a dropdown menu. Users may optionally upload a cover image; otherwise, a default image is assigned. Upon submission, the song file and metadata are saved to Supabase Storage and the database, respectively. Uploaded songs become immediately available on the homepage and searchable via the platform's search interface.
- **Playlist Management:** A separate modal allows users to create named playlists. Songs can be added directly from the playback interface or from search results. Each playlist supports standard queue-based playback and shuffle mode. Songs can be removed from playlists and deleted entirely through the search interface only.
- **Liked Songs:** A heart icon appears next to each song in the UI, allowing users to add tracks to a "Liked Songs" list, which is handled as a special dynamic playlist.
- **Profile Tools:** The user profile page provides two additional special features: audio filtering and song recognition. These tools are accessed via buttons and run independently of the main song database.



**Figure 2.2:** System Architecture Diagram

### 2.2.2. Audio Censorship via Whisper and NeutrinoAPI

One of the core innovations of the platform is its integrated audio censorship functionality, designed to mute explicit language in user-uploaded songs. This feature is implemented using a dedicated Python-based microservice built with Flask v3.1.0, which interacts with machine learning models and third-party APIs to deliver processed, sanitized audio content back to the user.

The censorship pipeline begins when a user accesses the profile page and selects the “Filter Explicit Content” option. This triggers a modal allowing the user to upload an audio file (typically in MP3 format) to the censorship service. The frontend sends the audio file via an HTTP POST request to the Flask server's /upload endpoint.



Upon receiving the file, the server first ensures it is valid, sanitizes the filename to prevent injection or path traversal attacks, and stores it temporarily in the uploads directory. Using PyDub, the audio is converted to WAV format, as this is the required input type for the transcription model.

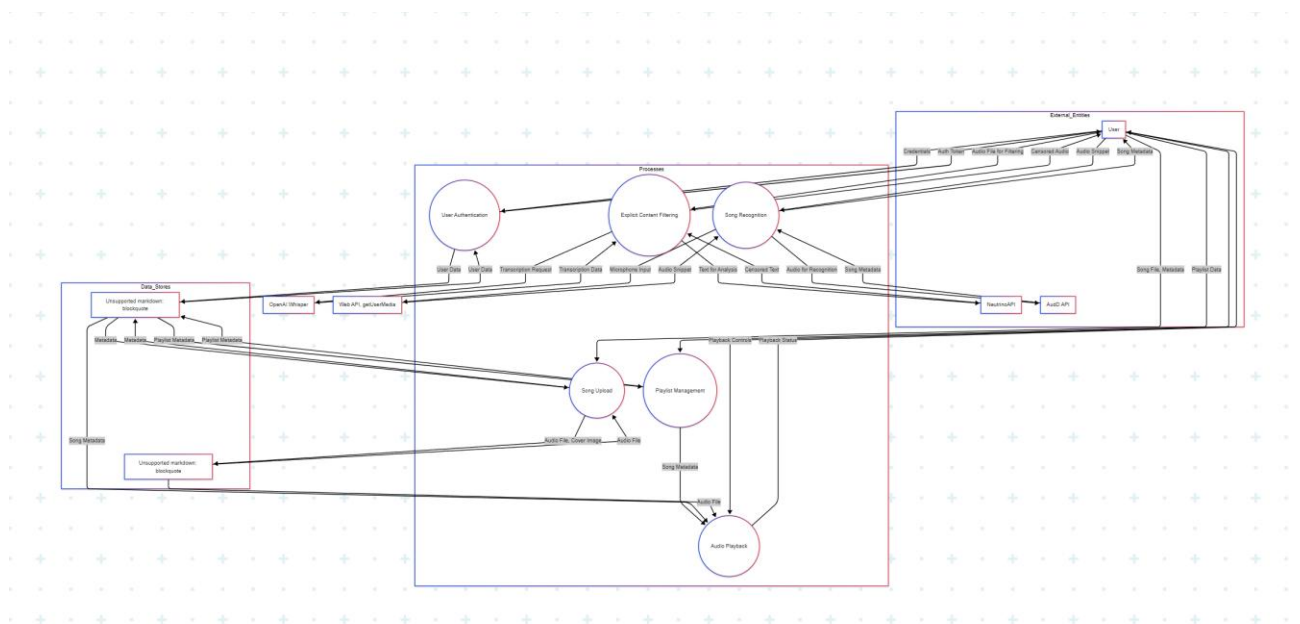
The converted audio is then transcribed using OpenAI's Whisper model, which is run on CUDA-enabled GPUs for accelerated processing (Radford, et al., 2022). Whisper's support for word-level timestamps is crucial here, as it enables precise localization of words within the audio stream. The output of this step is a JSON-like structure that includes the full text transcript and a list of each spoken word, along with its corresponding start and end time in the audio file.

This transcription is submitted to the NeutrinoAPI's Bad Word Filter endpoint. The API analyzes the text and returns a censored version where offensive words are replaced with asterisks (e.g., "f\*\*\*"). The original and censored versions are compared programmatically to identify which words have been masked. The result is a deduplicated list of explicit words detected in the transcription.

Each bad word is then matched against the transcribed word list to determine its exact audio timestamps. To ensure that muting is both effective and natural-sounding, a buffer of 200 milliseconds is added before and after each explicit segment. Adjacent mute intervals are merged if they are within 100 milliseconds of each other to prevent stuttering or excessive silence in the playback.

The censored version of the audio is reconstructed using PyDub by stitching together unmuted audio segments and replacing detected explicit regions with silence generated by `AudioSegment.silent()`. The final audio track preserves the structure and rhythm of the original song while removing or obscuring offensive language.

After the audio reconstruction is complete, the cleaned file is exported to the outputs directory in MP3 format with a “cleaned\_” prefix. This file is immediately returned to the user as a downloadable asset via the Flask response. The system does not retain the original or cleaned audio after the operation concludes; all temporary files are deleted to ensure privacy and efficient resource usage.



### 2.2.3. Song Recognition via Browser Microphone

platform's utility beyond streaming by integrating real-time acoustic fingerprinting functionality similar to what is available in commercial applications like Shazam.

The feature is accessible via the user's profile page, where a dedicated button labeled "Music Recognition" initiates the recognition process. When clicked, the application requests permission to access the user's microphone. This request is handled using standard Web APIs (`navigator.mediaDevices.getUserMedia`), ensuring compatibility across major browsers and adherence to privacy standards.

Once microphone access is granted, the system records a short audio sample, typically ranging from 5 to 15 seconds. This is long enough to capture a meaningful portion of a song while keeping the data payload manageable for API communication. The recorded sample is then encoded and sent to the backend, which in turn forwards the request to the AudD API, a commercial audio recognition service known for its high accuracy and large music database.

The AudD API performs audio fingerprinting, a technique that involves extracting unique acoustic features from the submitted sample and comparing them to a reference database (Cano, Battle, Kalker, & Haitsma, 2005). This matching process is robust against background noise and distortions, enabling accurate identification even in suboptimal recording conditions.

The API returns a structured response containing various metadata fields, including the song title, artist name, album name, release date, label and a link to the song on streaming platforms. This data is parsed by the backend and sent back to the frontend, where it is displayed to the user in a modal or dedicated section.

Importantly, the recognized metadata is used for display purposes only. The recorded audio sample is neither stored in the application's database nor linked to the user's account. This design choice supports user privacy and minimizes storage requirements, aligning with best practices for data handling in web applications.

The integration of browser-based microphone capture with API-driven recognition enhances the platform's value as a general-purpose music tool. It allows users to interactively engage with their audio environment, whether identifying a song playing in the background or enriching their understanding of unknown music. The feature is particularly useful for casual listeners who want quick, accessible song identification without switching to another application.

#### **2.2.4. Testing and Validation**

The system was tested manually using a local development environment. All primary features, such as song upload, playback, playlist management, audio filtering, and recognition, were executed multiple times to verify correctness.

Testing was performed without predefined unit or integration test cases, following an ad hoc methodology. Functional behavior and UI responsiveness were the main focus areas. Issues identified during manual testing were resolved iteratively through code modifications and retesting.

#### **2.2.5. Deployment and Execution Context**

The application was not deployed to a live server. All components, frontend, backend, and audio processing microservices, were executed locally on the development machine (Asus Zephyrus G14 laptop). Supabase, as the only remote service, handled cloud-based data and file storage. The application was demonstrated live in front of the thesis supervisor to confirm its functionality and feature completeness.

## **CHAPTER 3**

### **RESULTS AND DISCUSSION**

#### **3.1. Platform Functionality Evaluation**

To validate the completeness and effectiveness of the music streaming platform, a full range of features was tested manually in a controlled local environment. The testing process focused on evaluating the correctness of functionalities, reliability of integrations, user experience, and performance under typical usage scenarios. The results confirm that the platform operates as intended and fulfills its primary design goals of content moderation, media playback, and music recognition within a single cohesive system.

##### **3.1.1. Audio Upload, Playback, and Interface Usability**

Users begin by interacting with the interface through a clearly labeled upload icon. Upon clicking, a modal dialog prompts the user to input metadata such as the song title, artist, and genre. An optional field allows uploading a custom cover image, although a default image is assigned if left empty. This design ensures that songs are never uploaded with missing visuals, improving the aesthetics and consistency of the user interface.

Once uploaded, songs are stored in Supabase Storage and referenced in the platform's database via their filenames. The uploaded files immediately appear on the homepage

and are indexed in the search interface, allowing users to discover and interact with them directly. Playback of uploaded songs was smooth and consistent, confirming the reliability of the media player and its event-driven queue management system. Songs could be played, paused, skipped, and resumed with zero latency, and switching between tracks did not produce any glitches or interruptions.

The queue management system was thoroughly tested in both standard and shuffle modes. When the shuffle toggle is activated, the system dynamically reorganizes the song order without affecting the original playlist sequence. This ensures that the user experience remains predictable while providing the flexibility of randomized playback.

### **3.1.2. Playlist Features and Song Organization**

Playlist creation and manipulation were another critical part of the functionality evaluation. The platform allows users to create multiple playlists using an intuitive modal interface similar to the song upload process. Songs can be added to playlists from two contexts: while being played or from search results. This dual-point entry system increases flexibility and enhances the overall usability of the interface.

Each playlist supports playback functionality that mirrors the main queue system, including continuous play, shuffle mode, and real-time updates to the player interface. Users can remove songs from playlists through the search interface, which helps maintain playlist integrity and provides full user control over content.

The “Liked Songs” playlist operates as a dynamic, user-generated list. Songs can be liked using a heart icon located on playback and search interfaces. Once liked, songs automatically appear in the user’s favorites without additional steps, simplifying the process of curating preferred content.

### **3.1.3. Audio Filtering (Censorship) Feature Evaluation**

The platform's content moderation feature allows users to submit an audio file for automated censorship. This functionality was thoroughly tested with songs containing various levels of explicit language. The system leverages OpenAI Whisper (tiny model) to transcribe the audio and the NeutrinoAPI to identify inappropriate language.

Although the tiny Whisper model is optimized for performance and speed rather than maximum transcription accuracy, it performed well for the intended purpose. In all test cases, the model was able to capture enough of the lyrical content to identify offensive words. The most critical part of this feature is timestamp precision, which was handled effectively using Whisper's word-level segmentation and refined through a custom timestamp buffer mechanism. A  $\pm 200\text{ms}$  buffer was applied before and after each explicit word to ensure full coverage, while mute intervals within 100ms of each other were merged to minimize playback fragmentation.

After processing, the filtered audio files were downloadable through the UI. Playback tests confirmed that the cleaned versions retained their original quality, with only the muted sections rendered silent. The insertion of silence was done in a way that did not interrupt the flow or tempo of the song, ensuring a natural listening experience. This demonstrated the pipeline's effectiveness in balancing censorship precision with audio quality preservation.

### **3.1.4. Song Recognition Accuracy and Robustness**

The song recognition feature was evaluated using both mainstream international hits and lesser-known regional tracks. The process began with the user clicking a microphone activation button, which prompted the browser to request access to the device's

microphone. Upon approval, a short audio sample was recorded and sent to the backend, where it was forwarded to the AudD API.

Out of 15 test cases, 14 were identified accurately. Recognized songs included globally known tracks like “Starboy” by The Weeknd and “Illusion” by Dua Lipa, as well as Albanian songs like “Fajet” by Azet and Dhurata Dora. Only one case resulted in a mismatch, potentially due to environmental noise or a partial clip.

The returned metadata, comprising song name, artist, album, and streaming links, was displayed clearly in the frontend. The feature does not store the recorded audio, reinforcing user privacy. Given the high recognition rate and fast response time, the system is considered highly reliable for song identification.

### **3.1.5. Error Management and Edge Cases**

During testing, a few implementation and platform-specific issues were encountered:

- Transcription delay vs. accuracy: Larger Whisper models produced more accurate results but introduced significant delays. The tiny model offered a good compromise for real-time user workflows.
- AudD API activation: Initially, the API did not respond correctly after subscription purchase, likely due to latency in activation on the provider’s side. This resolved automatically within hours.
- Supabase inactivity suspension: On the free tier, the Supabase database can be temporarily disabled after prolonged inactivity. While this affected some test sessions, it would not be an issue on a paid plan in production.



Despite these challenges, none of the issues critically affected core functionality or user experience.

## 3.2. Audio Filtering and Recognition Results Evaluation

### 3.2.1. Performance of Audio Censorship Pipeline

While no formal timing benchmarks were recorded, the processing duration was observed to be approximately equal to or slightly less than the duration of the audio file. This responsiveness was crucial for user satisfaction, ensuring that users could receive their cleaned audio files without prolonged delays.

GPU acceleration was a key factor in achieving acceptable performance. Tests comparing GPU and CPU execution showed that while the CPU occasionally provided marginally better transcription accuracy, the processing time was significantly slower, often 3–5 times longer. GPU execution was preferred for real-time interaction.

Device	Avg. Time (60s Song)	Notes
CPU	~90 seconds	Higher accuracy but slow
GPU	~35 seconds	Fast with acceptable accuracy

**Table 1:** Performance Comparison of Whisper Model on CPU vs GPU

### 3.2.2. Fidelity of Audio Output

The censorship mechanism was designed to ensure that the insertion of silent segments did not degrade the quality or continuity of the music. Post-processing evaluation of cleaned songs revealed that transitions into and out of muted sections were seamless.

There was no evidence of audio glitches, pitch shifts, or distortion introduced by the processing steps.

Early development iterations had issues with mute placement, often leading to mismatches between detected bad words and their actual occurrence in the audio. These issues were resolved by refining the timestamp mapping algorithm and optimizing the mute buffer logic. Subsequent tests confirmed accurate alignment and high user-perceived quality.

Test #	Song Title	Explicit Words Detected	Missed	Muting Accuracy
1	DENIAL IS A RIVER – Doechii snippet	4	0	Accurate
2	Took Her To The O – King Von snippet	4	0	Accurate
3	I Feel It Coming – The Weeknd snippet	2	0	Accurate

**Table 2:** Accuracy of Explicit Content Detection Using Whisper and NeutrinoAPI

### 3.2.3. Song Recognition Insights

The song recognition feature maintained high accuracy under normal environmental conditions. While most songs were recognized within 3–4 seconds of submission, results varied slightly depending on the clarity of the recording and background noise.

AudD’s metadata response was comprehensive, and its support for multiple languages allowed accurate recognition of both English and Albanian music. The platform currently does not save metadata or audio recordings, reinforcing user control and privacy.

Test #	Song Title	Language	Result	Accuracy
1	Starboy – The Weeknd	English	Correct	✓
2	Illusion – Dua Lipa	English	Correct	✓
3	Fajet – Azet & Dhurata Dora	Albanian	Correct	✓
4	Unknown Speaking Audio	English	Incorrect	✗

**Table 3:** Summary of Song Recognition Tests Using AudD API

### 3.3. Discussion

The results of development and testing confirm that the platform achieved its intended purpose: delivering a user-friendly, privacy-aware, and functionally rich music streaming application with integrated content moderation and song recognition.

#### 3.3.1. Alignment with Objectives

The key project goals were met:

- Explicit content detection and filtering worked as intended across diverse audio samples.
- Song recognition was robust, accurate, and language-inclusive.
- Audio and playlist management features provided a seamless user experience.
- Performance was satisfactory, particularly with GPU-assisted processing.

The platform successfully unifies multiple tools into a single service, reducing the need for users to switch between apps for censorship, playback, and song discovery.

### **3.3.2. System Limitations**

Some limitations remain:

- Real-time filtering: Currently, users must upload a song for processing. Real-time muting during streaming playback was outside the scope of this implementation but is a key target for future development.
- Language scope: Bad word detection is currently limited to English. Expanding this to support multilingual profanity detection would improve inclusivity and global usability.
- No formal user study: While manual tests confirmed correctness, a larger user evaluation would provide insight into usability and satisfaction.

### **3.3.3. Proposed Future Enhancements**

The platform offers significant room for extension:

- Real-time censorship: Implementing real-time muting for streamed audio playback.
- Integration with streaming APIs: Connecting to services like Spotify for live content filtering.
- AI-powered music creation: Using tools like Suno AI to generate music from text prompts.
- Audio editing suite: Adding tools for noise reduction, lyric extraction, vocal/instrument separation, and waveform cropping.

- Multilingual support: Adding support for more languages in the transcription and filtering pipeline.

These additions would transform the platform into a comprehensive music toolkit, appealing to casual users, content creators, and businesses alike.

## **CHAPTER 4**

### **CONCLUSION**

#### **4.1. Summary of Contributions**

This thesis, titled *Developing a Music Streaming Website with Explicit Content Filtering and Song Recognition*, presents the design, implementation, and evaluation of a full-stack music streaming platform that uniquely integrates automated audio censorship and song identification features. The primary motivation behind the project was to address the lack of built-in content moderation options in existing commercial streaming services, particularly for use in public or commercial settings where the presence of explicit content can be problematic.

The developed system allows users to upload audio tracks, which are then processed to detect and mute offensive language through a backend transcription and filtering pipeline. Additionally, users can identify unknown songs by recording ambient audio, which is analyzed using acoustic fingerprinting technology. These two core features—content filtering and song recognition—are presented within a unified web interface that also supports user authentication, playlist creation, audio playback, and song organization.

From a technical perspective, the project combines modern frontend technologies (Next.js) with AI-powered audio processing models (OpenAI Whisper) and third-party APIs (NeutrinoAPI, AudD). The backend logic is implemented in Python using Flask, and cloud-based file and database storage is managed via Supabase. Manual testing

confirmed that the censorship feature accurately detected and muted explicit content without compromising audio quality, while the recognition feature demonstrated a high success rate across both international and regional tracks.

The platform was demonstrated in a supervised academic setting and received positive feedback from the thesis advisor and other evaluators. The system fulfills its intended goals, offering a proof-of-concept for a novel kind of music streaming service that prioritizes content control, utility, and simplicity within a self-contained application.

## **4.2. Future Improvements**

Although the prototype achieved its primary objectives, several enhancements are planned to expand the system’s functionality and increase its appeal for broader use cases.

The most immediate priority is to implement real-time content filtering during song playback. Unlike the current pre-processing approach, this enhancement would allow users to stream and listen to songs while the system dynamically detects and censors offensive language on the fly. This feature would significantly improve usability in business and public environments by eliminating the need for manual upload and download cycles.

Another key area of future development involves the integration of AI-powered music generation capabilities. By enabling users to create songs based on text prompts using models like Suno AI, the platform could appeal to content creators and hobbyists, positioning itself not just as a player or filter but as a generative music tool. This would extend the application’s reach beyond general listeners and into the domains of music production and entertainment.

In addition, the system may be adapted to support multilingual content filtering, enhancing its accessibility for global users. The current implementation is restricted to English-language detection, but language model adaptation and multilingual profanity databases could broaden its utility in diverse cultural and linguistic settings.

From a deployment perspective, the project is intended to evolve into a public-facing web application. Further development will focus on refining the user interface, improving system performance, and ensuring production-level stability. Once mature, the application will be hosted on a commercial platform to provide unrestricted access to individual users, small businesses, educators, and organizations that require curated audio experiences.

Finally, future iterations may introduce additional sound editing features such as lyric extraction, vocal and instrumental separation, noise reduction, and audio cropping. Together, these enhancements could transform the platform into a comprehensive digital audio hub for streaming, editing, generating, and managing music content.

In summary, the project has successfully laid the foundation for a flexible and user-centric audio platform, with demonstrated innovation in content filtering and recognition. Its design and implementation highlight the feasibility of integrating advanced audio technologies into consumer-facing applications, and its future development promises to deliver expanded functionality with meaningful real-world applications.

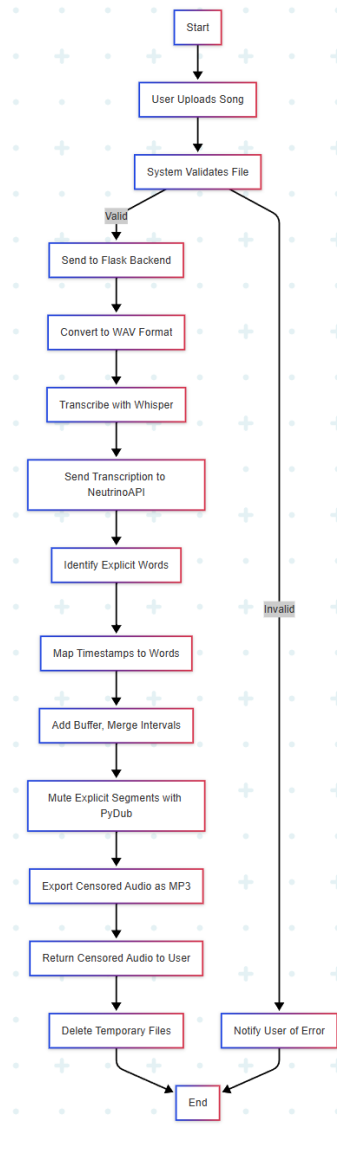


## REFERENCES

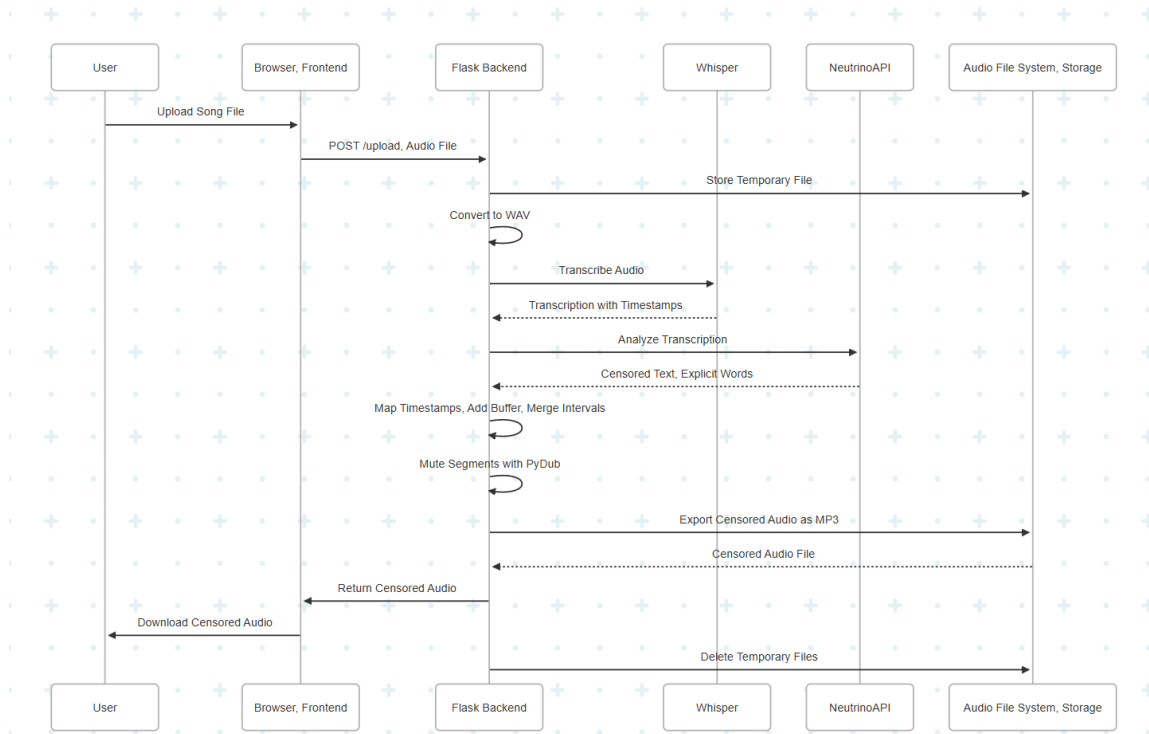
- AudD. (n.d.). *AudD Music Recognition API Docs*. Retrieved from AudD: <https://docs.audd.io/>
- Cano, P., Batlle, E., Kalker, T., & Haitzma, J. (2005). A Review of Audio Fingerprinting. *Journal of VLSI signal processing systems for signal, image and video technology*, 271-284.
- Choi, K., Fazekas, G., & Sandler, M. (2016). Automatic tagging using deep convolutional neural networks. *arXiv*.
- NeutrinoAPI. (n.d.). *Bad Word Filter API - Neutrino API Docs*. Retrieved from NeutrinoAPI: <https://www.neutrinoapi.com/api/bad-word-filter/>
- Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., & Sutskever, I. (2022). Robust Speech Recognition via Large-Scale Weak Supervision. *arXiv*.
- Soundplate. (n.d.). *Explicit Content Meaning : What Does 'Explicit Content' Mean on Spotify, Apple Music etc.* Retrieved from Soundplate: <https://soundplate.com/explicit-content-meaning/>
- Spotify. (n.d.). *Explicit content filter*. Retrieved from Spotify: <https://support.spotify.com/us/article/explicit-content/>
- Wang, A. L.-C. (2003). An Industrial-Strength Audio Search Algorithm. *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)*. Baltimore, Maryland, USA.

## APPENDIX A

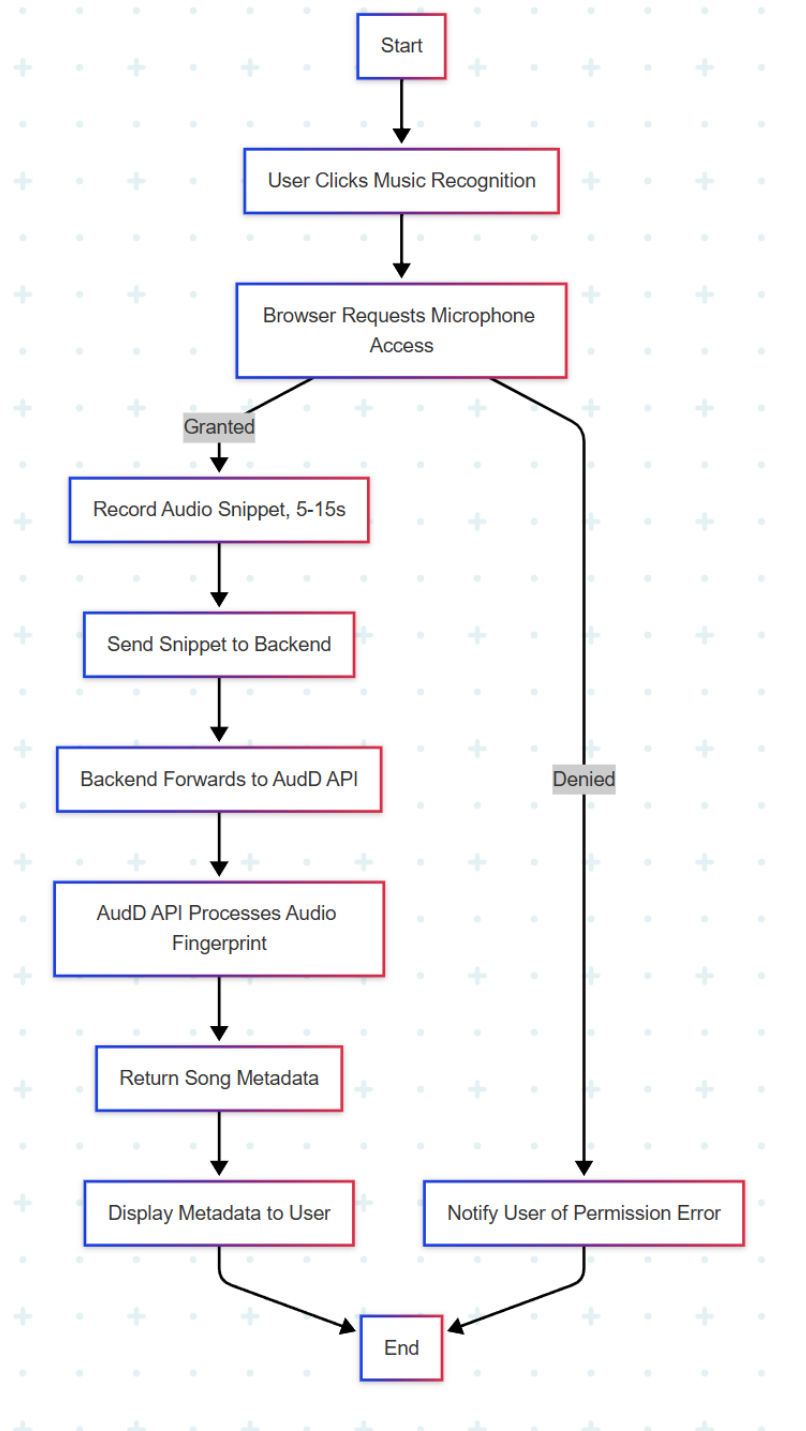
### ACTIVITY AND SEQUENCE DIAGRAMS



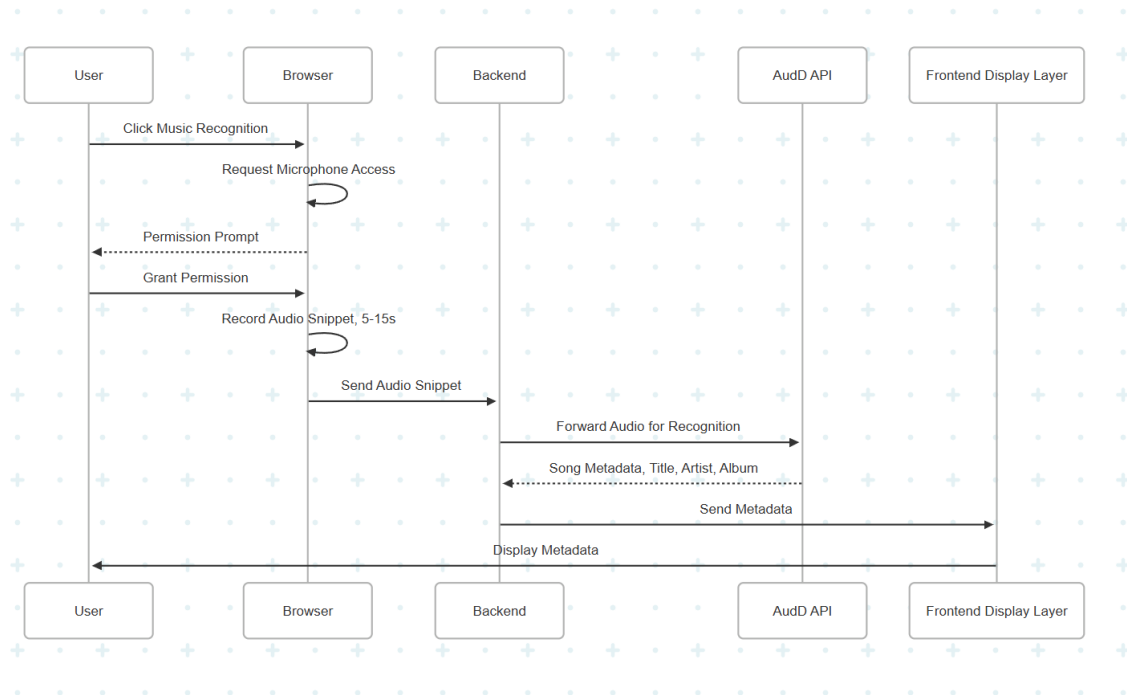
**Figure A.1:** Activity Diagram – Audio Filtering Workflow



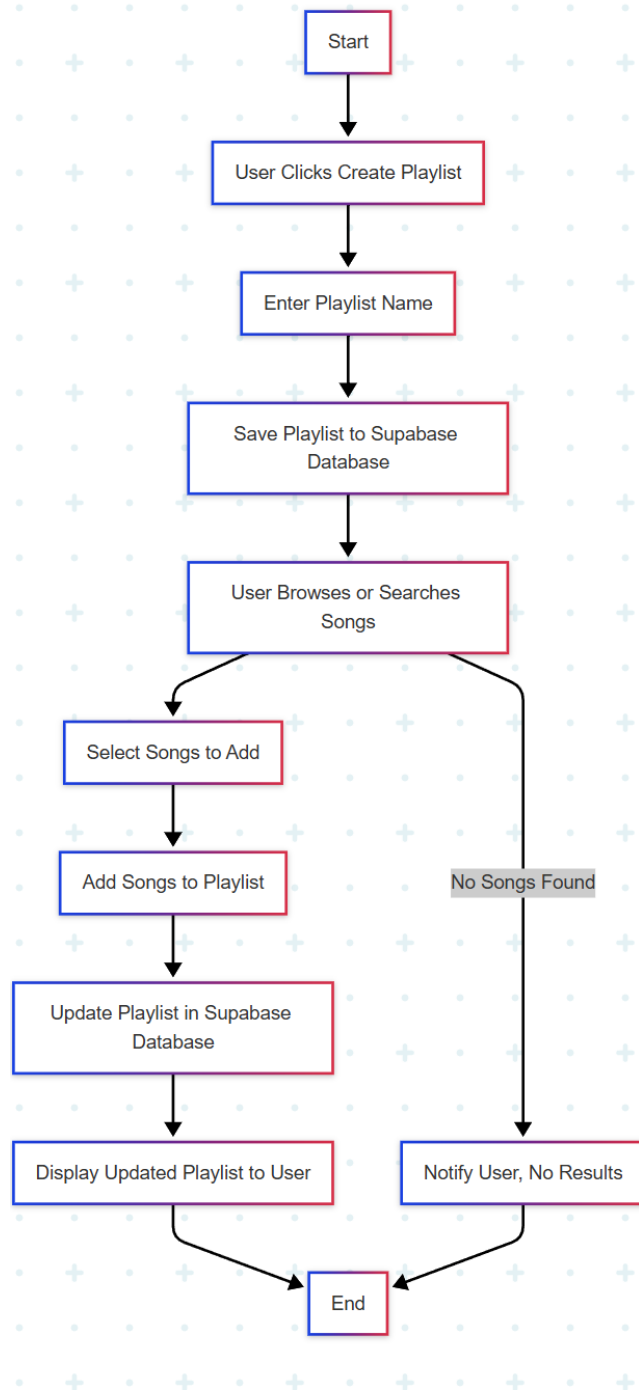
**Figure A.2:** Sequence Diagram – Audio Filtering Workflow



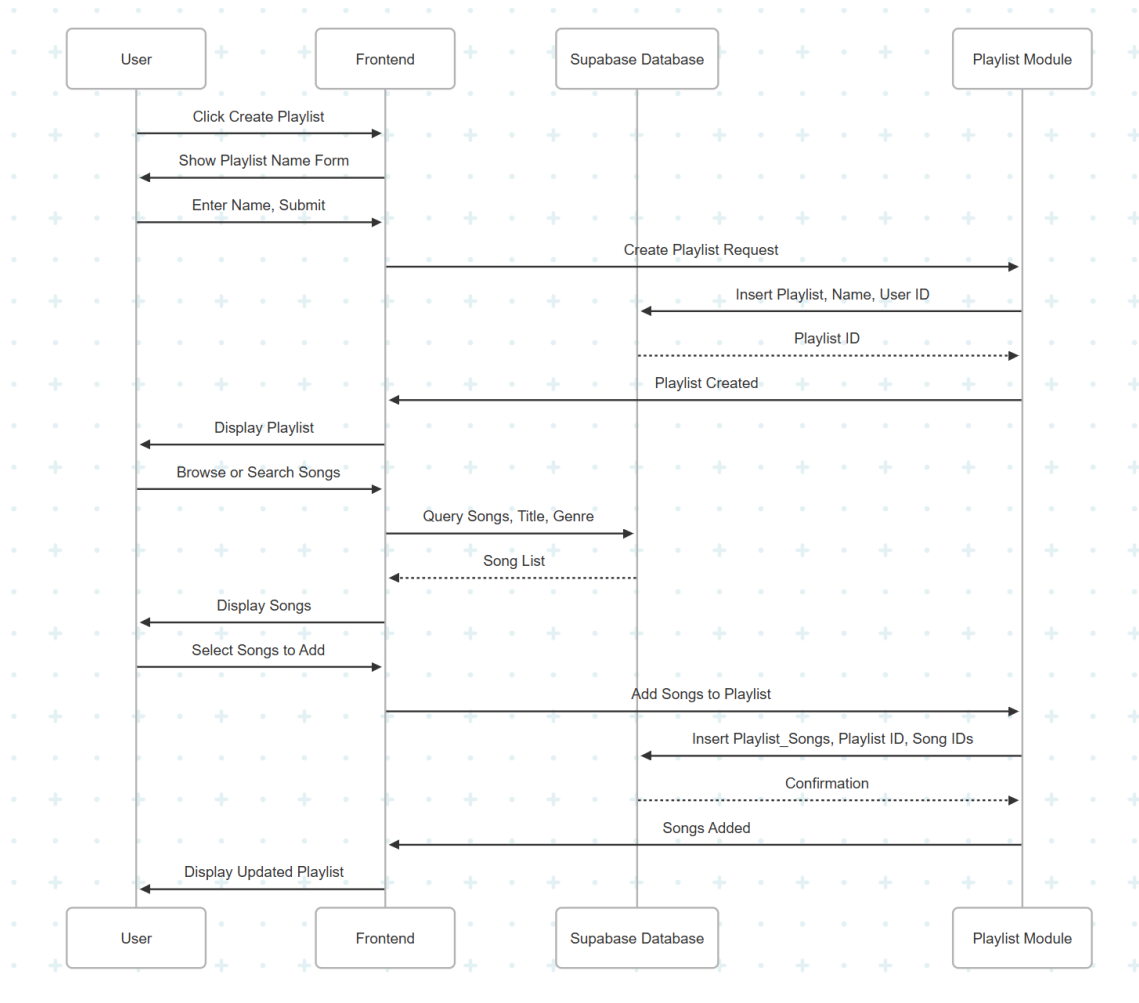
**Figure A.3:** Activity Diagram – Song Recognition



**Figure A.4:** Sequence Diagram – Song Recognition



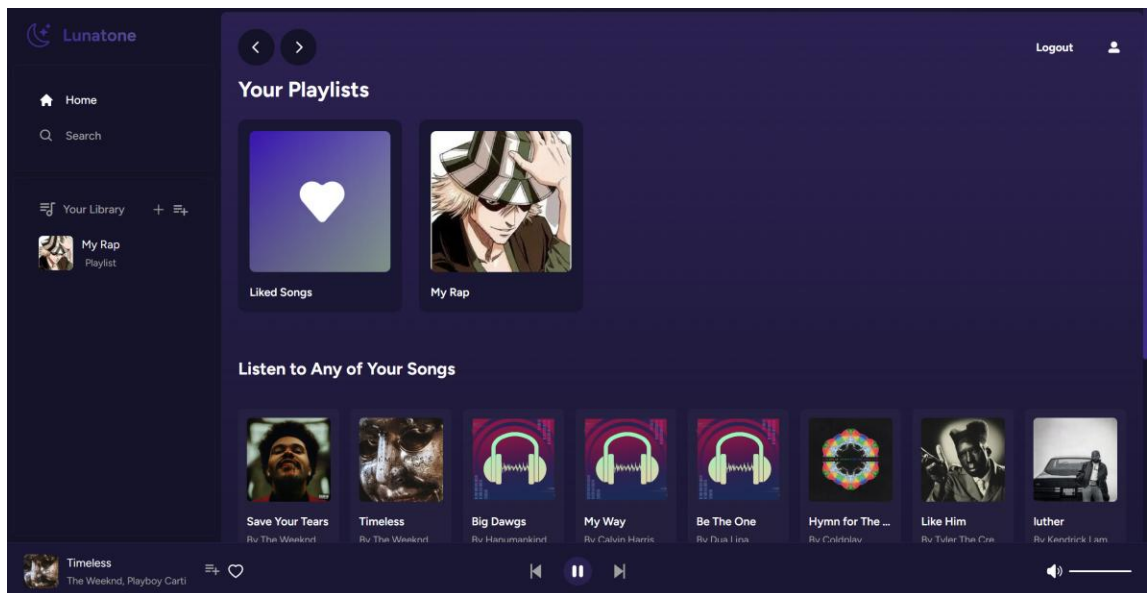
**Figure A.5:** Activity Diagram – Playlist Management



**Figure A.6:** Sequence Diagram – Playlist Operations

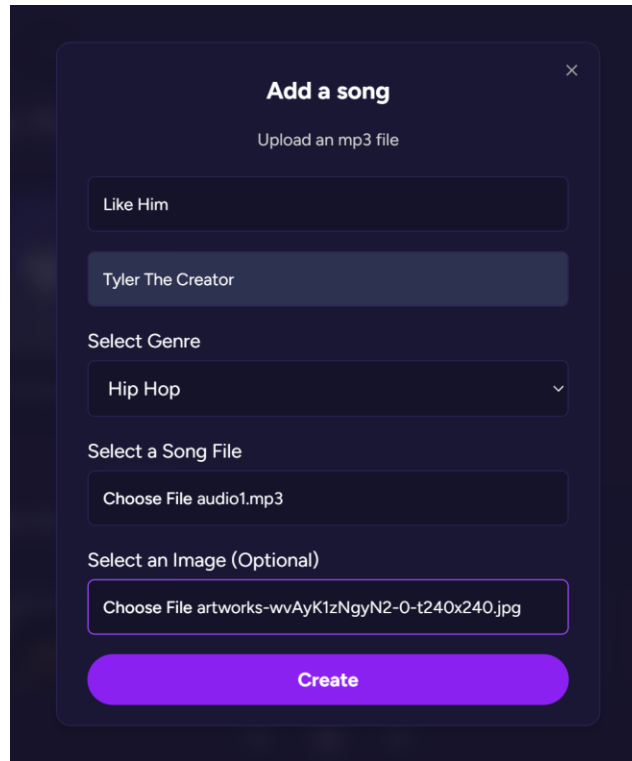
## APPENDIX B

### APPLICATION INTERFACE AND FUNCTIONALITY DEMONSTRATION



*Figure B.1:* Homepage Interface Showing Audio Player and Song Display





The 'Add a song' modal is a dark-themed interface for uploading music. It features a title bar with a close button (X) and a subtitle 'Upload an mp3 file'. The form contains several input fields: a text field for the song title 'Like Him', a text field for the artist 'Tyler The Creator', a dropdown menu for 'Select Genre' with 'Hip Hop' selected, a text field for 'Select a Song File' showing 'Choose File audio1.mp3', and another text field for 'Select an Image (Optional)' showing 'Choose File artworks-wvAyK1zNgyN2-0-t240x240.jpg'. A large, rounded 'Create' button is at the bottom.

**Add a song** ×

Upload an mp3 file

Like Him

Tyler The Creator

Select Genre

Hip Hop

Select a Song File

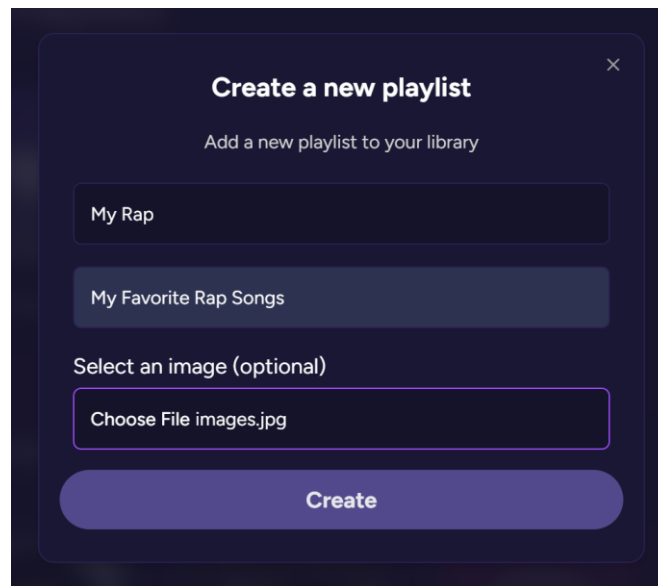
Choose File audio1.mp3

Select an Image (Optional)

Choose File artworks-wvAyK1zNgyN2-0-t240x240.jpg

Create

**Figure B.2:** Song Upload Modal with Metadata Inputs



The 'Create a new playlist' modal is a dark-themed interface for creating a new playlist. It features a title bar with a close button (X) and a subtitle 'Add a new playlist to your library'. The form contains two text fields: 'My Rap' and 'My Favorite Rap Songs'. Below these is a text field for 'Select an image (optional)' showing 'Choose File images.jpg'. A large, rounded 'Create' button is at the bottom.

**Create a new playlist** ×

Add a new playlist to your library

My Rap

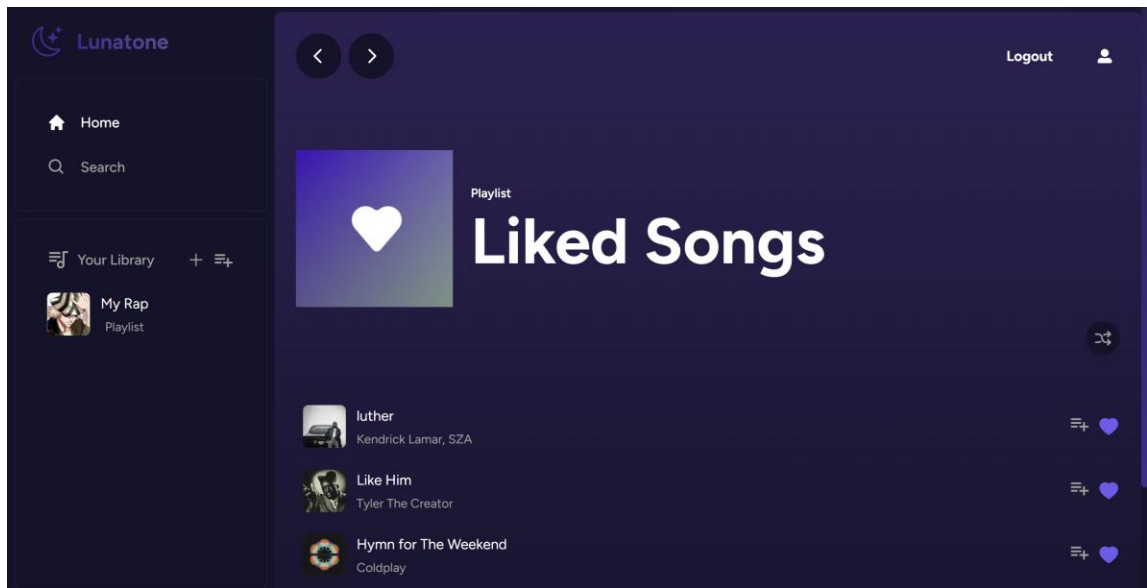
My Favorite Rap Songs

Select an image (optional)

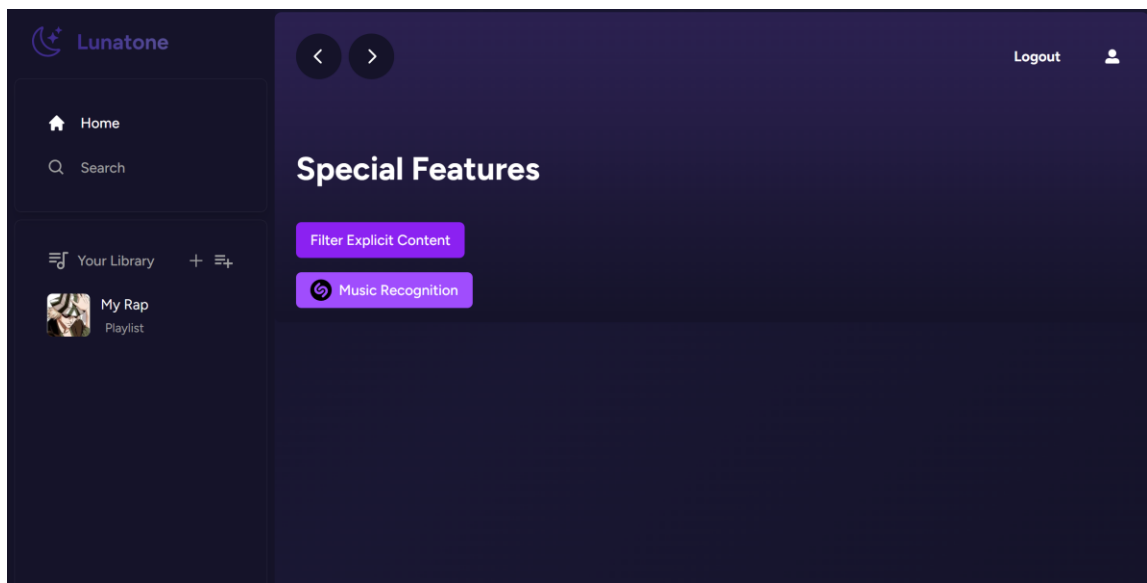
Choose File images.jpg

Create

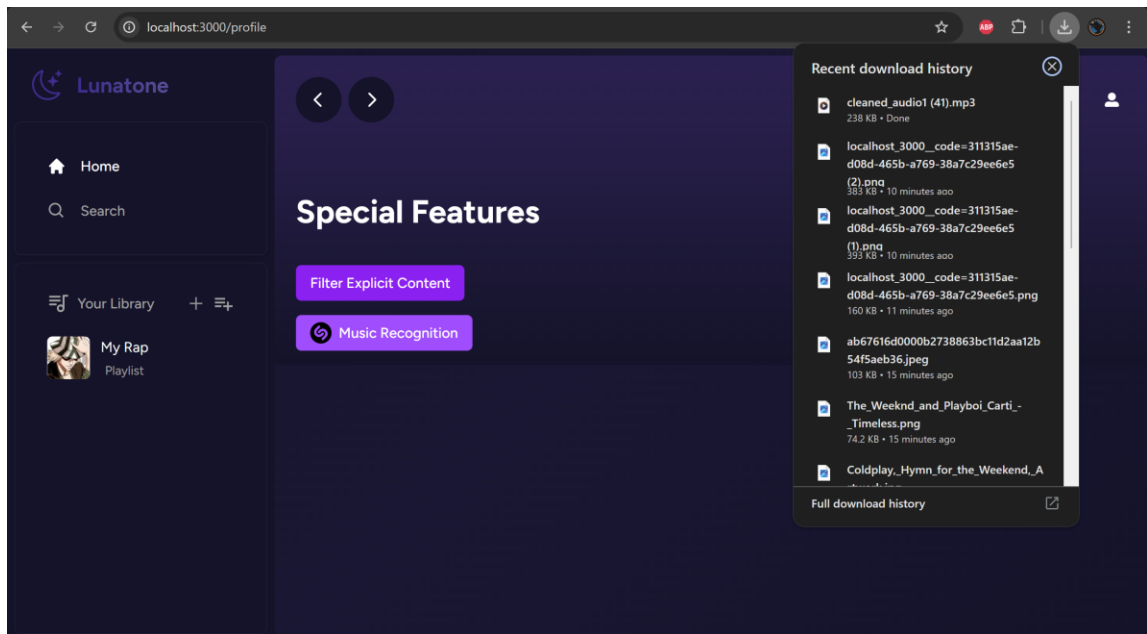
**Figure B.3:** Playlist Creation Modal



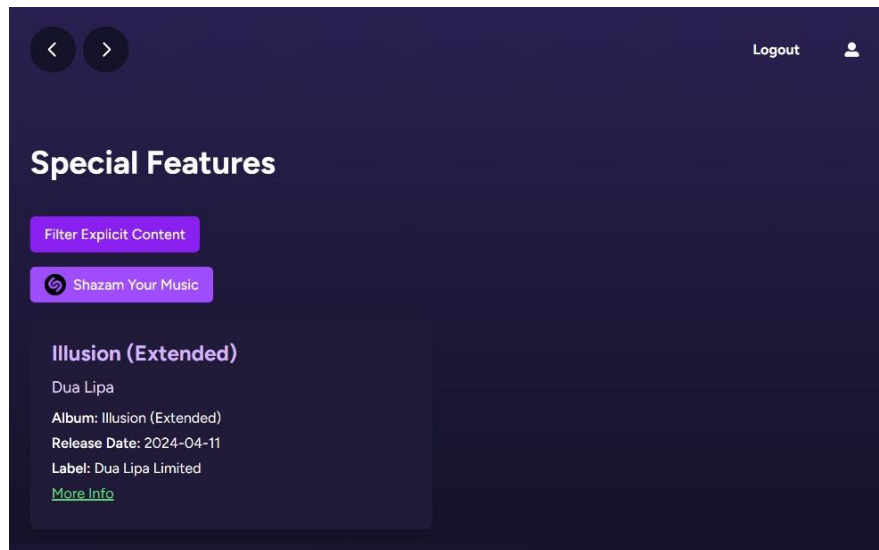
**Figure B.4:** Liked Songs Playlist Display



**Figure B.5:** Speacial Features Page Showing Filtering and Recognition Tools



**Figure B.6:** Filtered Song Download



**Figure B.7:** Song Metadata Retrieved Using AudD API