**Beat Buddy**

**Your personalized**

**Music player**

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

**Bachelor of Technology**

In

**Computer Science and Engineering**

**School of Engineering and Sciences**

Submitted by

**Abdulla Mohammad (AP21110011283)**

**Vamsi Sasidhar Manepalli (AP21110011287)**

**Suresh Babu Yelisetty (AP21110011289)**

**A picture containing text

Description automatically generated**

Under the Guidance of

**Dr. Sanjay Kumar**

**SRM University–AP**

**Neerukonda, Mangalagiri, Guntur**

**Andhra Pradesh – 522 240**

**Apr,2024**

# 

# Certificate

Date: 30-Apr-24

This is to certify that the work present in this Project entitled “**Beat Buddy-Your Personalized Music Player**” has been carried out by **Abdulla Mohammed, Suresh Babu Yelisetty, and Vamsi Sasidhar Manepalli** under my supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in **School of Engineering and Sciences**.

**Supervisor**

**Dr. Sanjay Kumar**

Assistant Professor

Computer Science

**Acknowledgement**

I want to express my heartfelt thanks to all those who contributed a vital role in the successful completion of this research internship. I extend my sincere thanks to Dr. Sanjay Kumar for his consistent assistance and sustainable support throughout the project. The lessons we attended under his guidance were not only motivating but also inspiring. His guidance and support were the main factors for our overall success of this research internship.

The success of this work is due to the advice and encouragement provided by Dr. Sanjay Kumar and my colleagues who have contributed to, and continue to contribute to, this initiative. Their continued support has been critical to our success, and we are thankful for their ongoing assistance.

I would like to take this opportunity to express our gratitude to every one of our group members and their consistent support made this remarkable project. We would not have met our goals without their combined efforts.

**Table of Contents**

**S. No Section Page No**

1 certificate I

2 Acknowledgement II

3 Table of Contents III

4 Table of Diagrams IV

5 Abstract V

6 Introduction 1

7 SRS Documentation 2

8 Technical Frameworks Used 21

9 Software Specification Diagrams 22

10 Functional and Path Coverage Testing 28

11 Description 34

12 Snap Shots Of Our Project 37

13 Conclusion 44

14 Limitations 45

15 References 45

16 Future Work 46

# 

# Table of Diagrams

1. Data Flow Diagrams (Level-0, Level-1 and Level-2)
2. Use Case Diagram (UML)
3. Entity Relationship Diagram
4. Class Diagram
5. Activity Diagram
6. Collaborative Diagram
7. State Diagram
8. Sequence Diagram

**Abstract**

Beat Buddy is a comprehensive music management and playback application designed to enhance users' musical experiences by enabling song recognition, downloading, playlist creation, and seamless playback. With advanced audio recognition technology, users can identify songs quickly and easily by recording a portion of the song or inputting its title. The application offers a built-in downloader that supports various formats and quality levels for offline playback. A robust music player supports essential controls like play, pause, skip, repeat, and shuffle, allowing users to create and manage playlists effortlessly. Beat Buddy provides user account management for personalization and stores preferences securely. Non-functional requirements emphasize performance, reliability, security, usability, and compatibility across major operating systems and browsers. Beat Buddy aims to revolutionize music management, offering an intuitive, reliable, and secure application for music enthusiasts

**Introduction**

Beat Buddy is a state-of-the-art music management and playback application designed to deliver an immersive and user-centric experience for music enthusiasts. This innovative application incorporates a suite of advanced features that cater to the diverse needs of modern users, from song recognition to personalised playlist creation and seamless music streaming. At the heart of Beat Buddy is its audio recognition technology, allowing users to identify songs effortlessly by recording a snippet of audio or entering the song title. Once recognized, users receive comprehensive song details including the title, artist, and album information, along with options to download the song or add it to playlists.

The built-in song downloader supports various file formats such as MP3, FLAC, and AAC, as well as different quality settings to accommodate users' preferences and device capabilities. This functionality enables users to enjoy their favourite music offline with ease. The robust music player within Beat Buddy offers essential playback controls like play, pause, skip, repeat, and shuffle, allowing users to enjoy a tailored listening experience. Users can create and manage playlists directly within the player, personalising their musical journeys.

Beat Buddy prioritizes user account management, enabling users to create individual accounts for a customised experience. User preferences, playlists, and other settings are stored securely in the system, ensuring a consistent and personalized experience across devices. Non-functional requirements such as performance, reliability, security, usability, and compatibility with major operating systems and web browsers ensure that Beat Buddy delivers an exceptional and seamless experience for its users. The application strives to revolutionize the way users manage, discover, and enjoy their music collections, providing a reliable and intuitive platform for music lovers worldwide.

**SRS Documentation**

**Table of Contents**

**S. No Section Page No**

1 Introduction 1

1.1 Purpose 3

1.2 Scope 3

1.3 Definitions, Acronyms, and Abbreviations 3

1.4 References 4

1.5 Overview 4

2 The Overall Description 4

2.1 Product Perspective 4

2.2 Product Functions 5

2.3 User Characteristics 6

2.4 Constraints 6

2.5 Assumptions and Dependencies 6

3 External interface Requirements 7

3.1 User Interface Requirements 7

3.2 Hardware Interface Requirements 8

3.3 Software Interface Requirements 8

3.4 Communications Interface Requirements 9

4 System Features 10

4.1 Use Case Diagram 11

4.2 Data Flow Diagram 12

4.3 ER Diagram 13

4.4 Class Diagram 14

4.5 Activity Diagram 15

4.6 State Diagram 17

4.7 Sequence Diagram 18

4.8 Collaborative Diagram 19

5 Other Non-Functional Requirements 20

5.1 Performance Requirements 20

5.2 Software System Attributes 20

5.3 Business Rules 20

**1. Introduction**

Beat Buddy is a music management and playback application designed to provide users with a seamless experience in recognizing songs, downloading them, creating playlists, and enjoying their favorite music. This Software Requirements Specification (SRS) document outlines the functional and non-functional requirements of the Beat Buddy application.

* 1. **Purpose**

Beat buddy aims to provide seamless music experience by providing a well-designed UI to interact and download and save music of your choice and create playlists to make your experience more personalized

* playing songs
* Downloading songs
* Recognition of songs

* 1. **Scope**

Beat Buddy transforms the landscape of music consumption with its cutting-edge platform built on advanced recognition technology and user-centric design principles. Offering a personalized music experience, Beat Buddy prioritizes seamless interactions and intuitive functionalities. Key features include personalized playlists, effortless song downloads, and accurate song recognition, fostering a deeper connection between users and their favorite tunes. With a focus on minimal latency and robust encryption protocols, Beat Buddy ensures secure and efficient music exploration, setting a new standard for digital audio platforms.

* 1. **Definitions, Acronyms, and Abbreviations.**

|  |  |
| --- | --- |
| **API** | Application Programming Interface |
| **NODE.JS** | JavaScript runtime environment |
| **REACT** | JavaScript Library for building user Interfaces |
| **RAPID API** | Platform for finding and using Api*s* |
| **SRS** | Software Requirement Specifications |
| **UI** | User Interface |
| **YOUTUBE API** | Api for accessing YouTube data |

* 1. **References**
* Node.js Documentation: <https://nodejs.org/en>
* React Documentation: <https://legacy.reactjs.org/docs/getting-started.html>
* Rapid API Documentation:<https://rapidapi.com/>
* YouTube API Documentation: <https://developers.google.com/youtube/v3>

**1.5. Overview**

This SRS document provides a structured framework for understanding Beat Buddy's functionalities and requirements. It includes sections on:

* Product perspective and overall description
* Product functions
* User characteristics
* Constraints
* Assumptions and dependencies
* External interface requirements
* System features (including use case diagram)
* Non-functional requirements (performance, software attributes, business rules)
* Other requirements
* Appendix (Glossary)

1. **The Overall Description**
   1. **Product Perspective**

Beat Buddy aims to provide a user-friendly and personalized music experience through:

* Song recognition: Effortlessly identify songs playing in the surrounding environment.
* Song download: Download desired music from Spotify links, playlists, or by searching by song title.
* Music player: Control music playback, including play/pause, rewind/fast forward, and volume adjustment.
  1. **Product Functions**

The major functions that **Beat Buddy** performs are described as follows: -

* **Song Recognition Integration:**

**Functionality**: Integrates with a third-party song recognition API (e.g., Shazam,) to identify songs played through the user's device microphone.

**Details**: Captures audio input from the microphone, transmits it to the chosen API, and receives the identified song information.

**Usage**: Users can activate song recognition to identify songs playing in their surroundings.

* **Song Download Integration:**

**Functionality**: Integrates with the YouTube API to download songs based on user input.

**Details**: Accepts Spotify song links, playlist links, or individual song names as input.

**Usage**: Users can download songs they discover through various means.

* **Music Player:**

**Functionality**: Offers standard music player controls for playback management.

**Details**: Includes functionalities like play/pause, stop, rewind/fast forward, volume control, and seeking within a song.

**Usage**: Users can control music playback from within the Beat Buddy application

* **User Authentication and Management:**

**Functionality:** User authentication and management involve verifying the identity of users and managing their access to the communication platform and its features.

**Details:** User authentication functionalities include user registration, login, logout, password management, and session management. User management functionalities include user profiles, permissions, roles, and account settings.

**Usage in Project:** In “Beat Buddy” user authentication and management functionality is implemented to ensure secure access to the communication platform, manage user accounts and permissions, and provide personalized experiences for users.

**2.3 User Characteristics**

Beat Buddy caters to a broad user base, including:

* Music enthusiasts who seek a convenient way to discover and manage their music collection.
* Casual listeners who desire a simple tool for playing and organizing their favorite songs.
* Individuals who frequently come across new music they want to identify and download.

* 1. **Constraints**

The major constraints that the project has are as follows: -

* Third-party API limitations: Functionality depends on the capabilities and constraints of the chosen song recognition and download APIs.
* Music copyright: Downloaded songs must comply with copyright laws and user agreements.
* Internet connection: Requires a stable internet connection for song recognition, download, and streaming functionalities.
* Device compatibility: Should function across various devices and operating systems (consideration for web vs. mobile application).
  1. **Assumptions and Dependencies**

Assumptions and dependencies for the "Beat Buddy" project can vary based on the specific requirements, technology stack, and development approach. However, here are some common assumptions and dependencies that may apply:

* **Assumptions:**
  + **User familiarity:** Users have basic computer literacy and are familiar with using web browsers and downloading file**s.**
  + **Internet connection:** Users have a stable internet connection for song recognition, downloading, and streaming functionalities**.**
  + **Third-party API limitations:** Functionality depends on the capabilities and limitations of the chosen song recognition and download APIs (e.g., accuracy of recognition, download speed, quota restrictions**).**
  + **Music copyright:** Downloaded songs must comply with copyright laws and user agreements. Users are responsible for ensuring they have the rights to download and listen to the music.
  + **Device compatibility:** The application should function across various devices with web browsers, considering responsiveness and user experience on different screen sizes.
* **Dependencies:**
  + **Third-party APIs:** 
    - Song recognition API (e.g., Shazam) for identifying songs.
    - YouTube API for downloading songs based on user input.
  + **Development tools and libraries:** 
    - Node.js runtime environment for server-side functionality.
    - React library for building the user interface.
    - Additional libraries or frameworks as needed (e.g., for state management, routing, user authentication).
  + **External services:**
    - Hosting platform for the application (e.g., cloud provider).
    - Database (optional) if user account or persistent data are required.

These assumptions and dependencies provide a foundational understanding of the project's requirements and the external factors that may impact its development and functionality.

1. **External Interface Requirements**

**3.1 User Interface Requirements**

The Beat Buddy user interface (UI) should be:

1. **Intuitive and user-friendly:** Easy to navigate with clear labels and visual cues for different features and functionalities.
2. **Visually appealing:** Employ modern design principles and be aesthetically pleasing to users.
3. **Responsive:** Adapt to different screen sizes and devices (desktop, mobile, tablets) for optimal user experience.
4. **Accessible:** Consider accessibility best practices to cater to users with diverse needs.

* **UI components:**

**Search bar:** Allow users to search for songs using keywords, song titles, or artist names.

**Song recognition:** Provide a button or option to activate song recognition using the device microphone.

**Song download:** Include dedicated functionalities for:

* Downloading songs from Spotify links (including individual songs, playlists, or artist profiles).
* Downloading songs based on song titles (search results may include YouTube links or other sources).

**Music player:** Provide standard music player controls for:

* Play/pause
* Stop
* Rewind/fast forward
* Volume adjustment
* Displaying song information (title, artist, album)

**3.2 Hardware Interface Requirements**

There are various hardware components with which the machine is required to interact. Various hardware interface requirements that need to be fulfilled for successful functioning of the software are as follows: -

* **Microphone:** Required for song recognition functionality.
* **Speakers/headphones:** Required for audio playback.
* **Internet connection:** Stable internet connection for song recognition, download, and streaming functionalities.
* **Device compatibility:** The application should function across various devices with web browsers, considering responsiveness and user experience on different screen sizes.

**3.3 Software Interface Requirements**

In order to perform various different functions, this software needs to interact with various other software’s. So, there are certain software interface requirements that need to be fulfilled which are listed as follows: -

* **Song recognition API:** Integration with a third-party song recognition API (e.g., Shazam) to identify songs played through the user's microphone.
* **YouTube API:** Integration with the YouTube API to download songs based on user input or search results.
* **Web browser:** The application should function within a standard web browser environment.

**3.4 Communication Interface Requirements**

The machine needs to communicate with the main branch for each session for various functions such as login verification, account access etc. so the following are the various communication interface requirements that are needed to be fulfilled in order to run the software successfully: -

* **API communication:** Secure communication with the chosen song recognition and download APIs to exchange data (e.g., song identification, download requests).
* **Network communication:** Communication with the internet for song recognition, download, and streaming functionalities.

These external interface requirements provide a framework for designing and developing the Beat Buddy user interface and its interactions with hardware, software, and external APIs.

1. **System Features**

* **Song Recognition:**

**Functionality:**

* + Users can activate song recognition through a dedicated button or option.
  + The application captures audio input from the device microphone.
  + It transmits the audio data to the chosen song recognition API.
  + The API identifies the song and returns information like title, artist, album.
  + The application displays the identified song information to the user.

**Use cases:**

* + Users can identify songs playing in their surroundings and add them to playlists.
  + Users can discover new music by recognizing songs they hear in various contexts.
* **Song Download:**

**Functionality:**

* + Users can provide input for downloading songs through:
  + Spotify song link (individual song, playlist, artist profile).
  + Song title search (results may include YouTube links or other sources).
  + The application processes the user input and interacts with the YouTube API.
  + Based on the input, the application initiates the download process.
  + Downloaded songs are stored locally on the user's device.

**Use cases:**

* + Users can download songs they discover through various means and add them to their playlists.
  + Users can create personalized music collections with their favorite songs.
* **Music Player:**

**Functionality:**

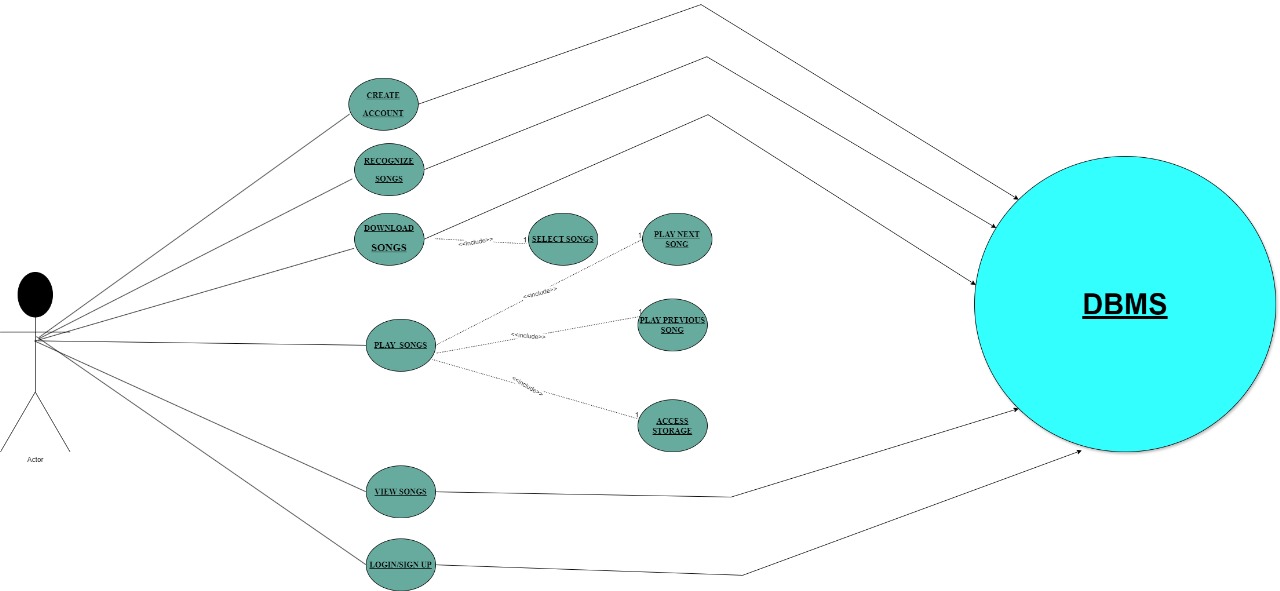
* + Provides standard music player controls for playback management, including:
  + Play/pause
  + Stop
  + Rewind/fast forward
  + Volume adjustment
  + Seeking within a song
  + Displays song information (title, artist, album) during playback.

**Use cases:**

* + Users can control music playback within the Beat Buddy application.
  + Users can enjoy their downloaded music collections with convenient playback functionalities.
* **Additional Features:**
  + User login and account management (for playlist persistence and personalization across devices).
  + Song recommendations based on user preferences and listening history.
  + Integration with music streaming services for broader music access (if rights and partnerships are obtained).
  + Offline playback functionality for downloaded songs.

These system features provide a detailed breakdown of Beat Buddy's functionalities and how they cater to user needs in the context of music discovery, download, organization, and playback.

**4.1 Use Case Diagram**



**Actor:** User

**Use Cases:**

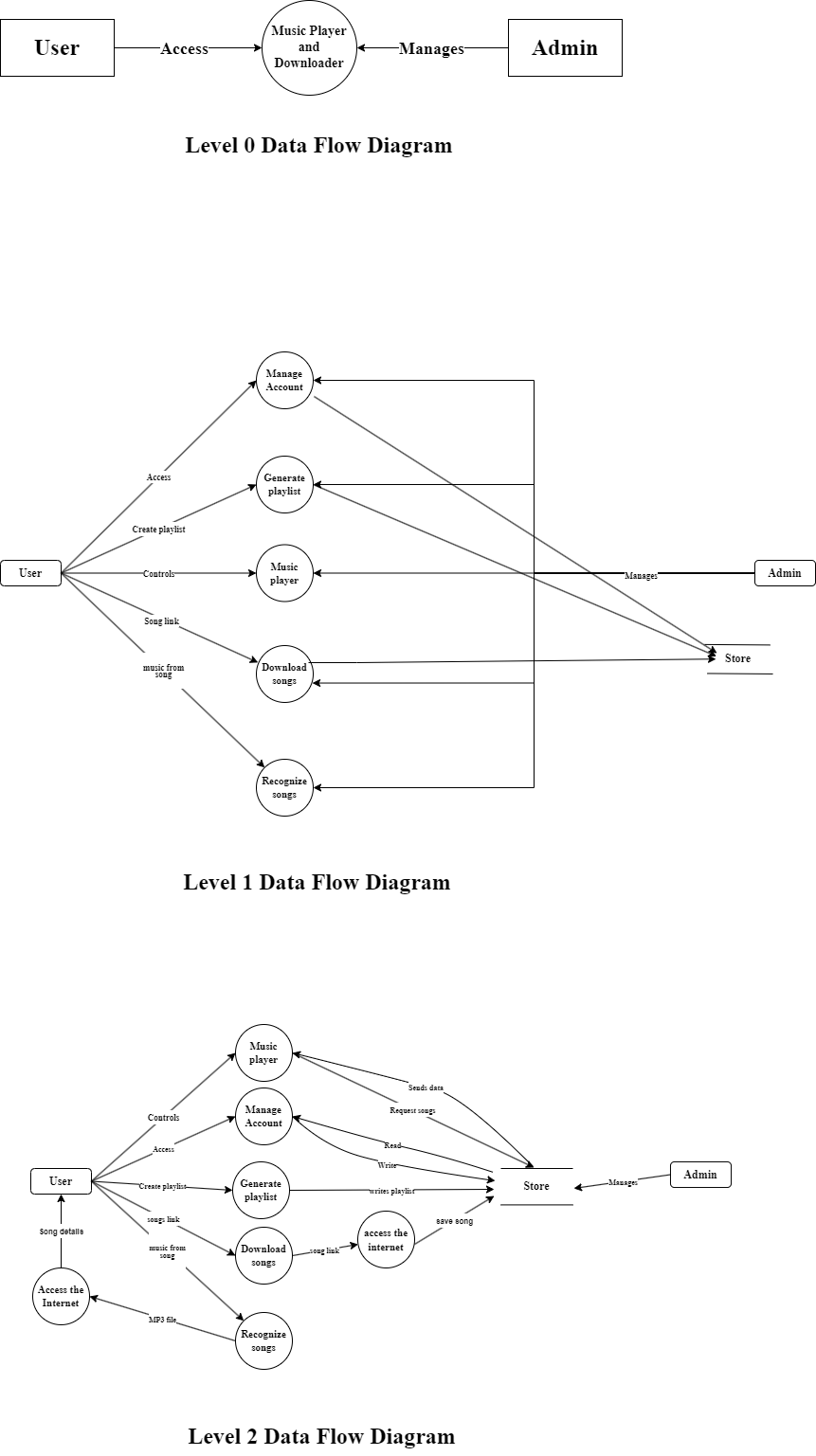
* Recognize Songs
* Play Songs
* View Songs
* View Playlists
* Create Account
* Login
* Download Songs
* Search for Songs

The diagram also shows the following relationships between these elements:

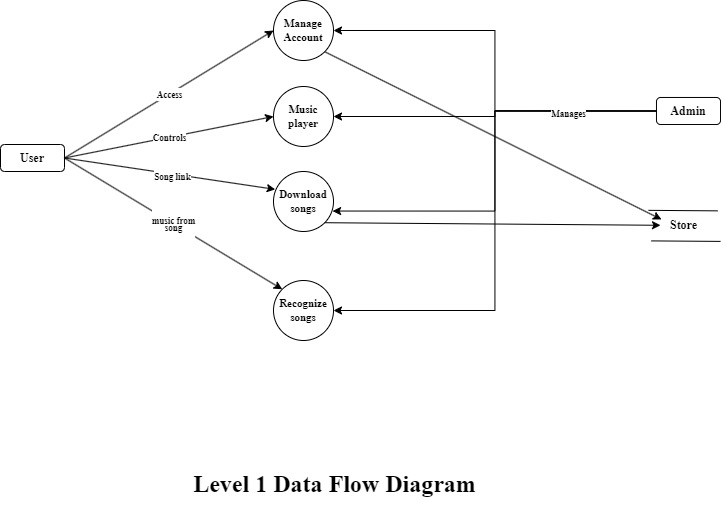
* The User can **Recognize Songs, Play Songs, View Songs, Create Account, Login, Download Songs, and Search for Songs.**
* The **Recognize Songs** use case extends to the Play Songs use case. This means that after recognizing a song, the user can choose to play it.
* The **View Songs and Play Songs** use cases both extend to the **View Playlists** use case. This means that users can view songs and play songs from their playlists.
* The **Download Songs** use case extends to the **Create Playlists** and View Playlists use cases. This means that downloaded songs can be added to playlists and viewed in playlists.
* The **Search** for Songs use case extends to the **Download Songs** use case. This means that users can search for songs and then download them.

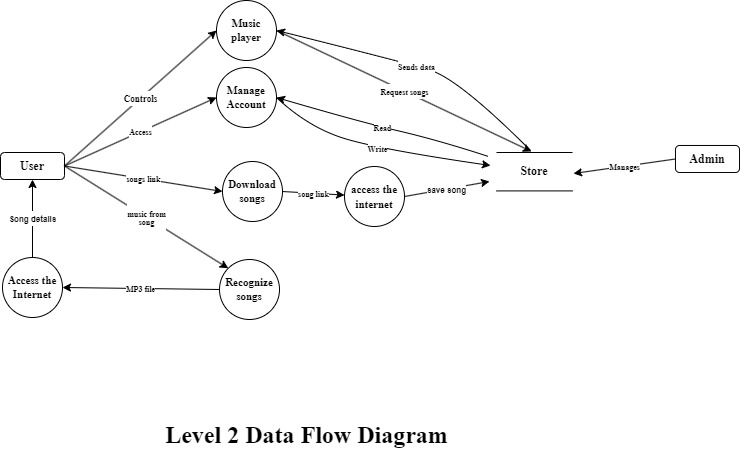
Overall, the use case diagram provides a good overview of the different functionalities that the music player application offers to users and how these functionalities are interrelated.

**4.2 Data Flow Diagram (DFD)**

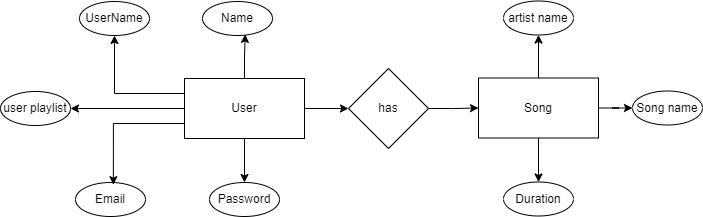
**Level 0 - DFD:**

**Level 1 – DFD:**

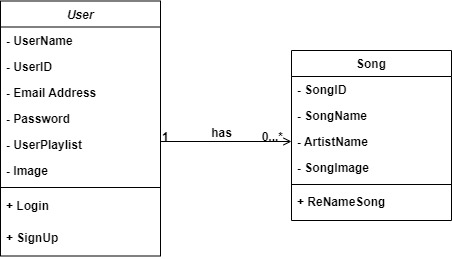


**Level 2-DFD**

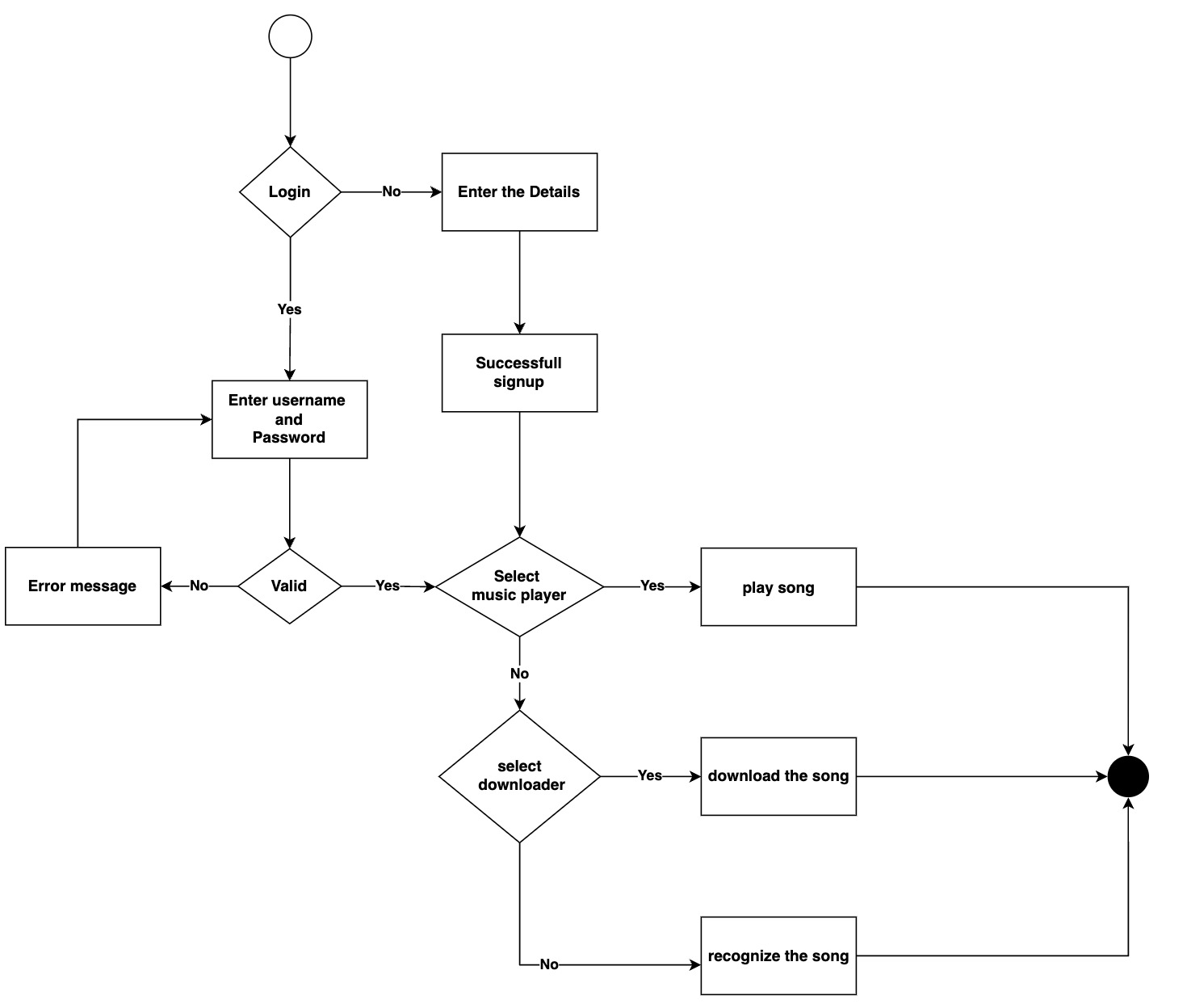
**4.3 Entity Relationship Diagram (ER Diagram):**



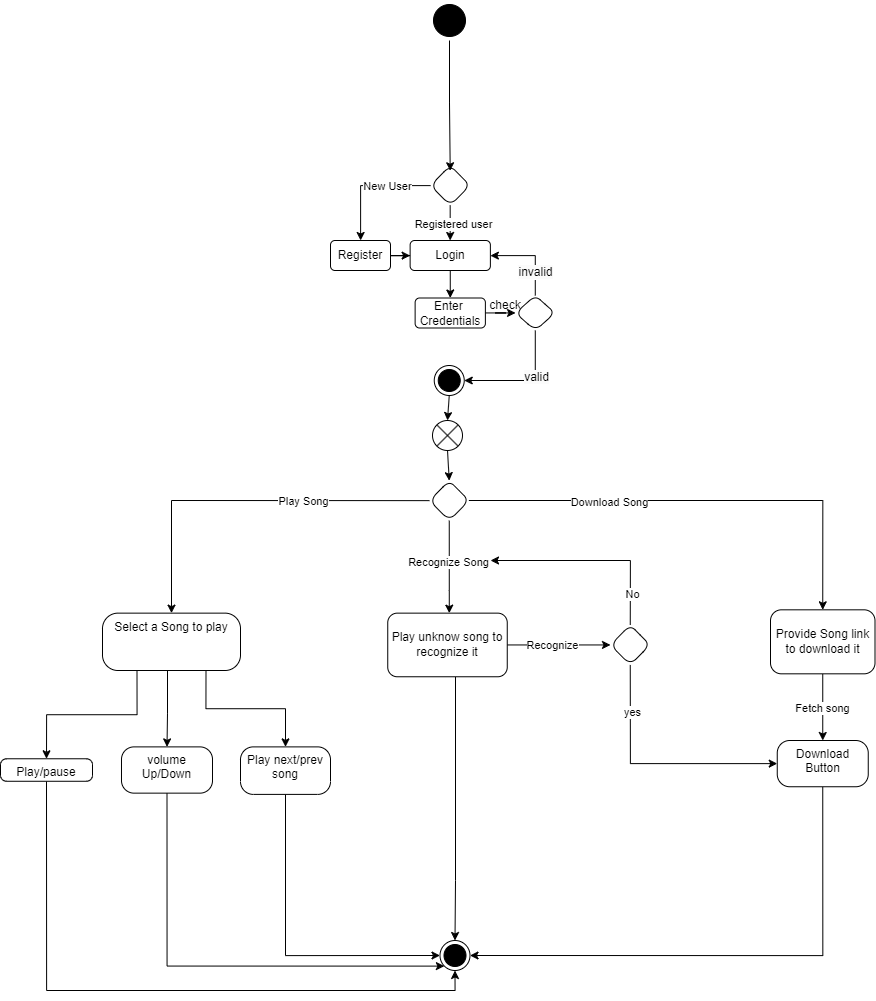
**4.4 Class Diagram:**



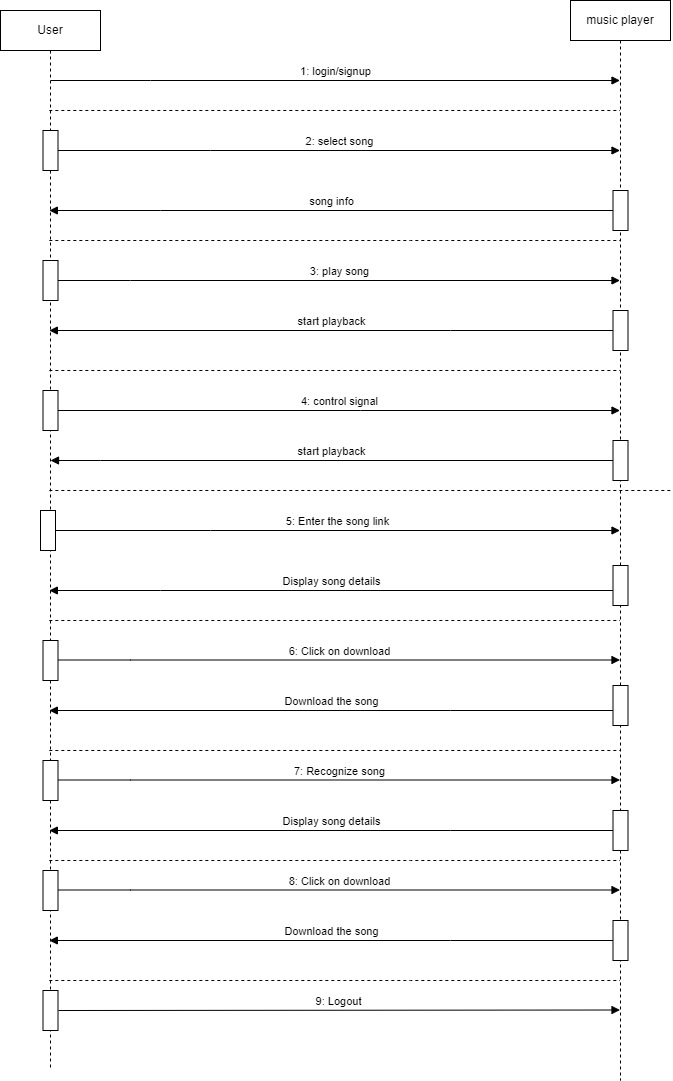
**4.5 Activity Diagram:**

****

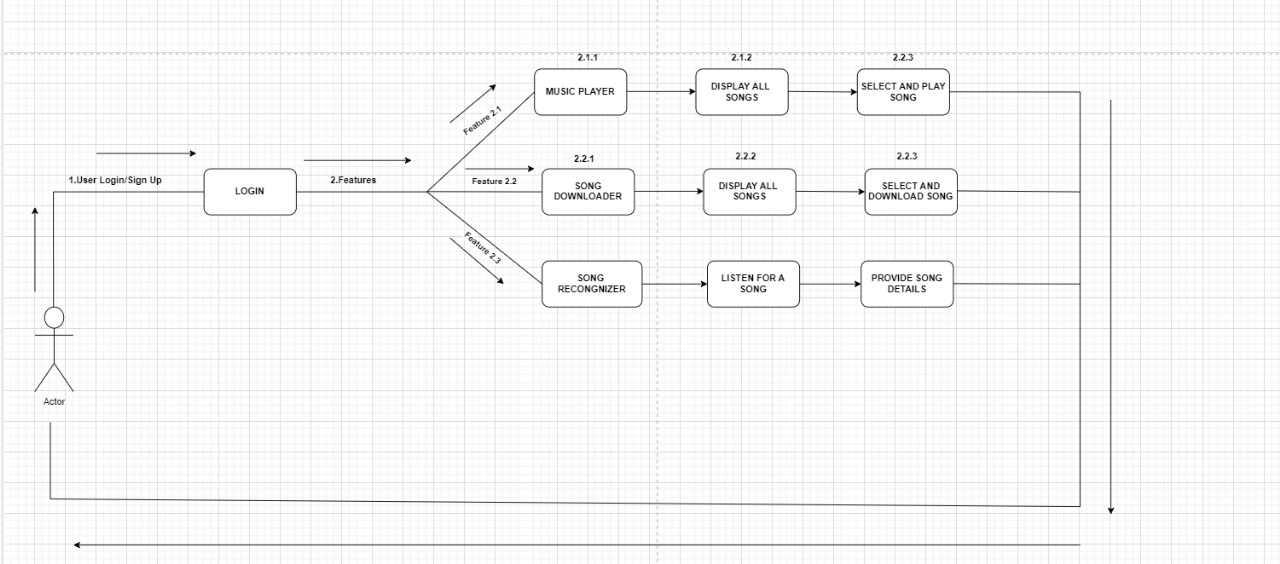
**4.6 State Diagram:**

****

**4.7 Sequence Diagram:**

****

**4.8 Collaborative Diagram:**

****

1. **Other Non-Functional Requirements**

**5.1 Performance Requirements**

**5.1.1 Song recognition:**

* + **Response time:** Identify songs within 5-10 seconds after activating the recognition feature.
  + **Accuracy:** Achieve at least 80% accuracy in identifying songs from various genres and audio quality.

**5.1.2 Song download:**

* + **Download time:** Download songs within a reasonable timeframe based on song size and internet connection speed. Ideally, strive for an average download speed of 1-2 Mbps.
  + **Download success rate:** Achieve a success rate of at least 95% for song downloads, with mechanisms to handle and notify users of any failed downloads.

**5.1.3 Music playback:**

* + **Smooth playback:** Ensure smooth and uninterrupted audio playback without buffering or lag.
  + **Seek functionality:** Allow users to seek within songs with minimal delay.

**5.2 Software System Attributes**

* **Reliability:** The application should be reliable and function consistently without unexpected crashes or errors.
* **Availability:** The application should be available and accessible to users with minimal downtime.
* **Maintainability:** The code should be well-structured, documented, and easy to maintain and update.
* **Scalability:** The application should be scalable to accommodate an increasing number of users and data without performance degradation.
* **Security:** The application should implement security measures to protect user data and prevent unauthorized access. This includes secure data storage, user authentication, and encryption of sensitive information.

**5.3 Business Rules**

* **Copyright compliance:** Downloaded songs must comply with copyright laws and user agreements. Users are responsible for ensuring they have the rights to download and listen to the music.
* **Fair use and limitations:** Implement mechanisms to prevent abuse and excessive downloads, adhering to any limitations or fair use policies of the used APIs or services.
* **User data privacy:** Users' data, such as search history and downloaded songs, should be treated with respect and privacy. The application should adhere to relevant data privacy regulations and user agreements.
* **Content moderation (optional):** If the application allows user-generated content or playlists, consider implementing content moderation policies to ensure respectful and appropriate content is shared.

These additional requirements provide a comprehensive overview of the performance expectations, software qualities, and business rules that guide the development and functionality of Beat Buddy.

**Technologies Used:**

**Front-end Technologies:**

HTML, CSS, React, Tailwind, Vite, Formik, Yup

**Back-end Technologies:**

Node Js, Express, MongoDB, Mongoose, JWT, Rapid API.

**Database Description**:

We used MongoDB as our database and Mongoose as our ORM. Our database has three entities namely User, Playlist and Song.

1. User entity consist of the attributes

1. Name - string

2. Username - string

3. Email - string

4. Password - string

2. Song entity consist of the attributes

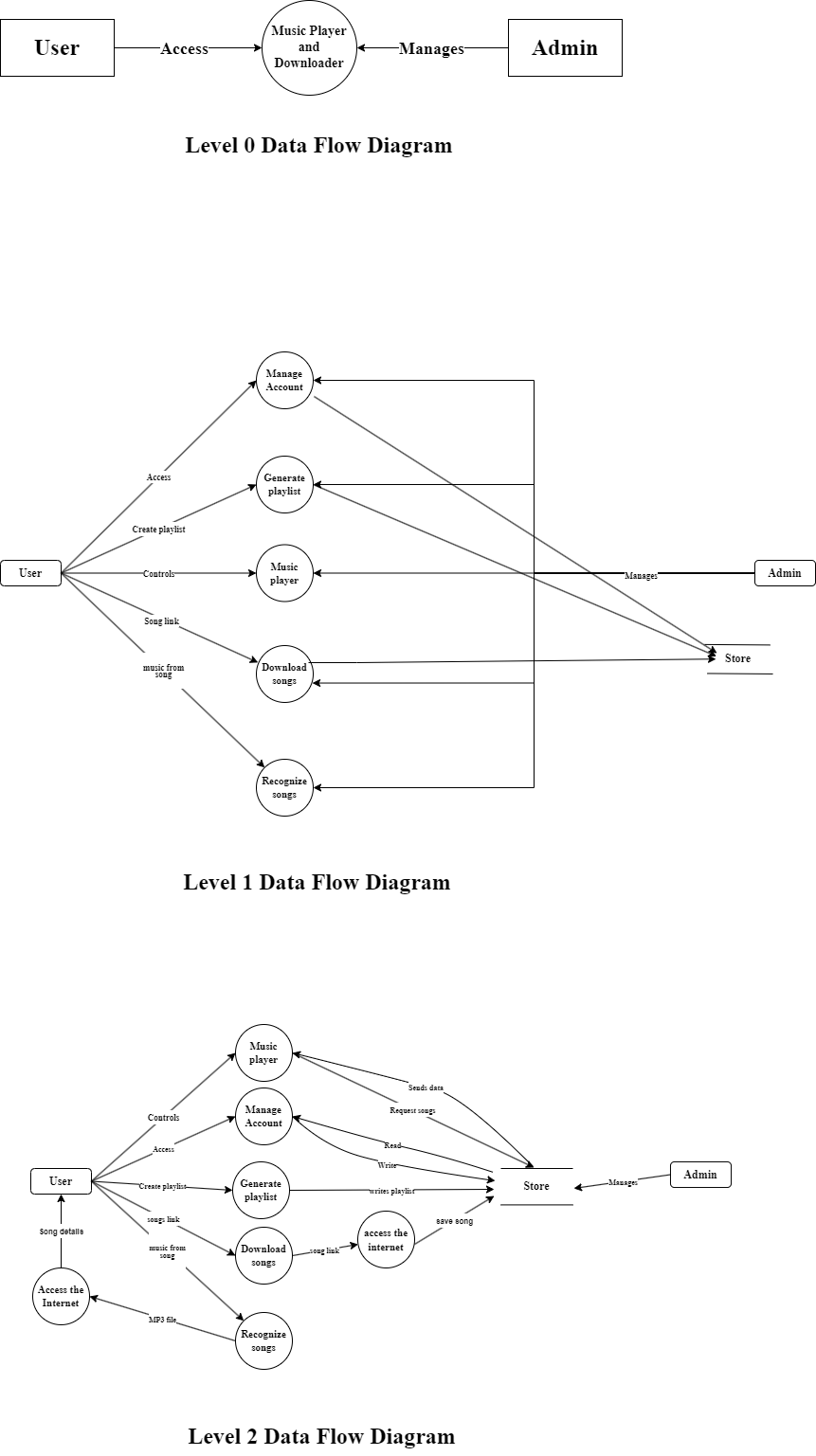
1. Name - string

2. Artist Name - String

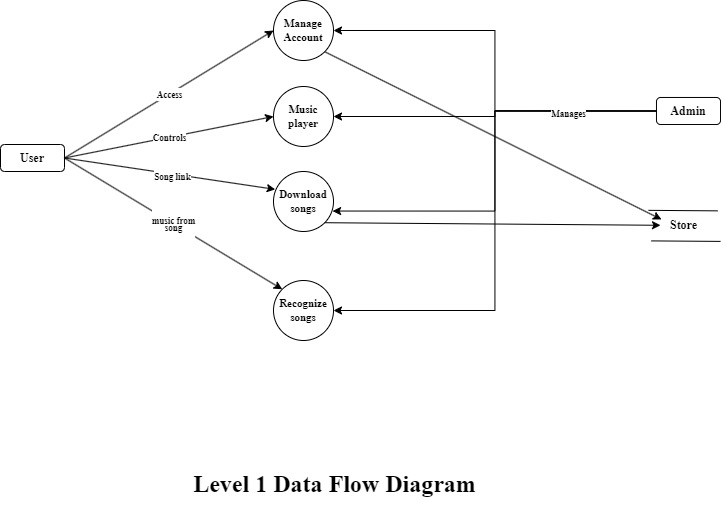
3. Duration – String

**Software Specification Diagrams**

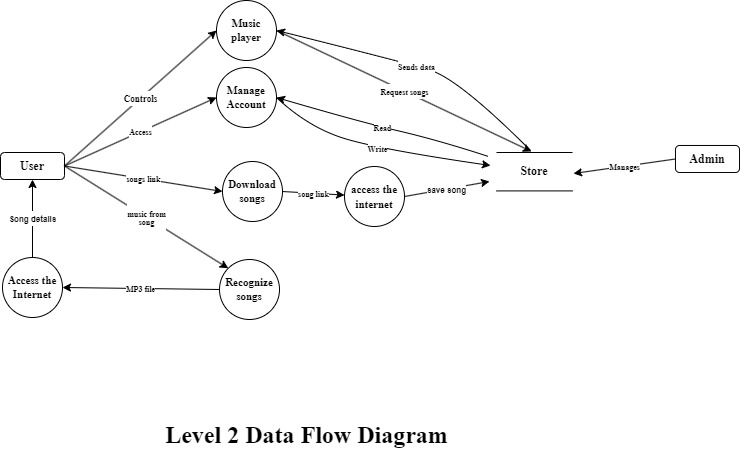
**DFD (Data Flow Diagrams):**

**Level 0 - DFD**

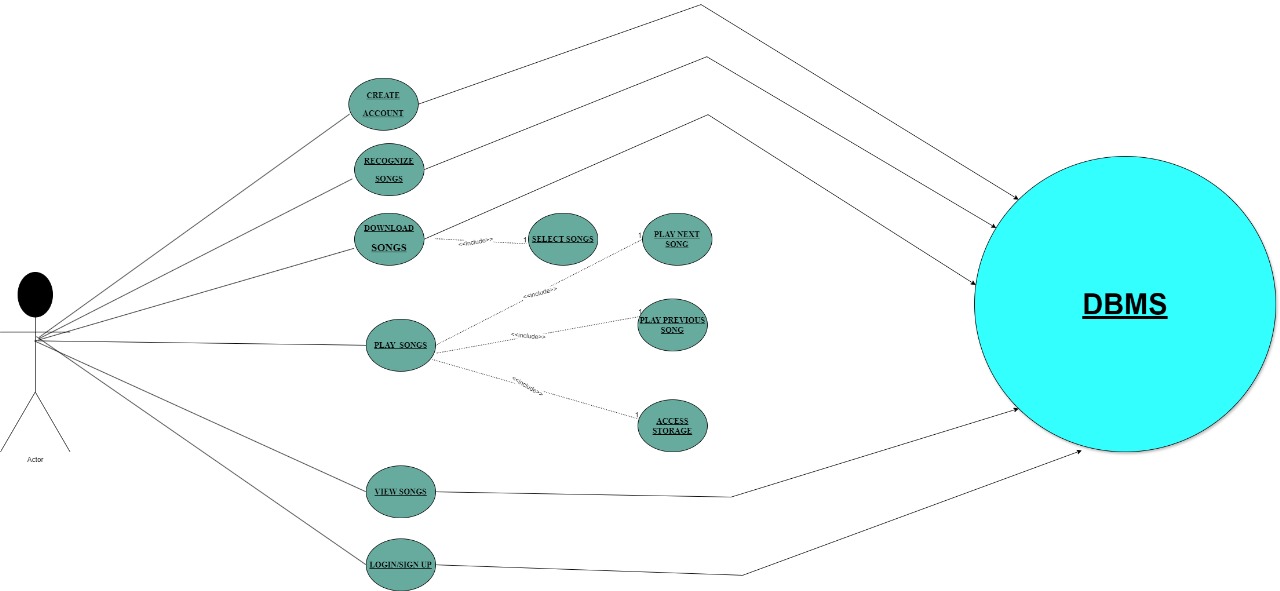
**Level 1- DFD**



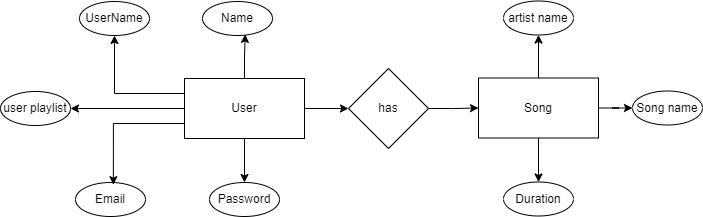
**Level 2-DFD**



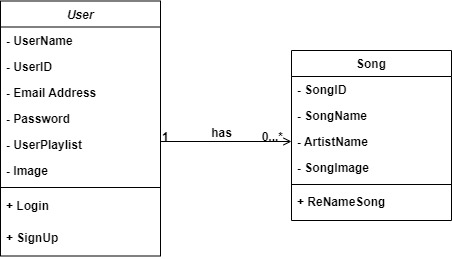
**Use Case Diagram (UML):**

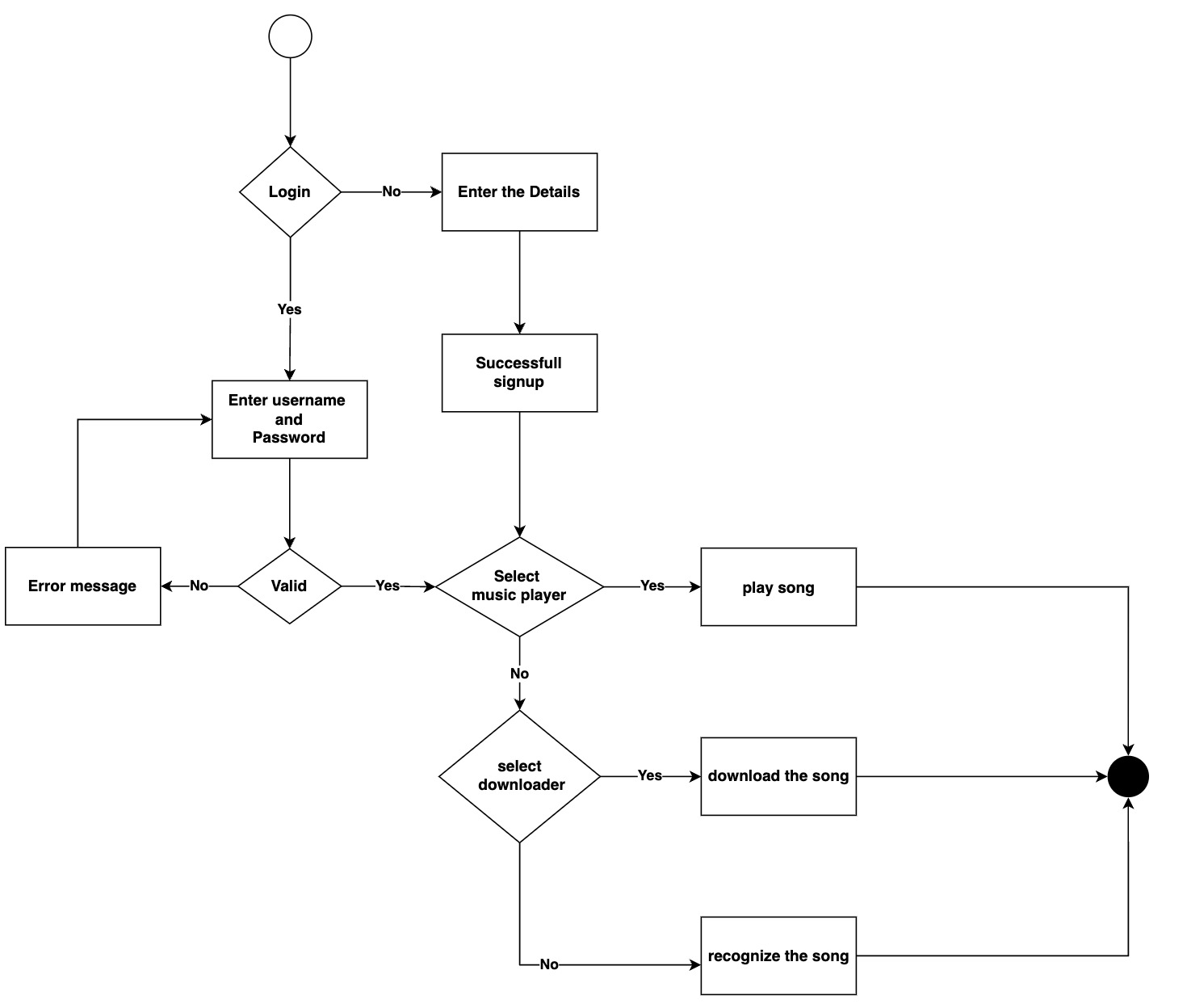


**Entity – Relationship Diagram (ER Diagram):**

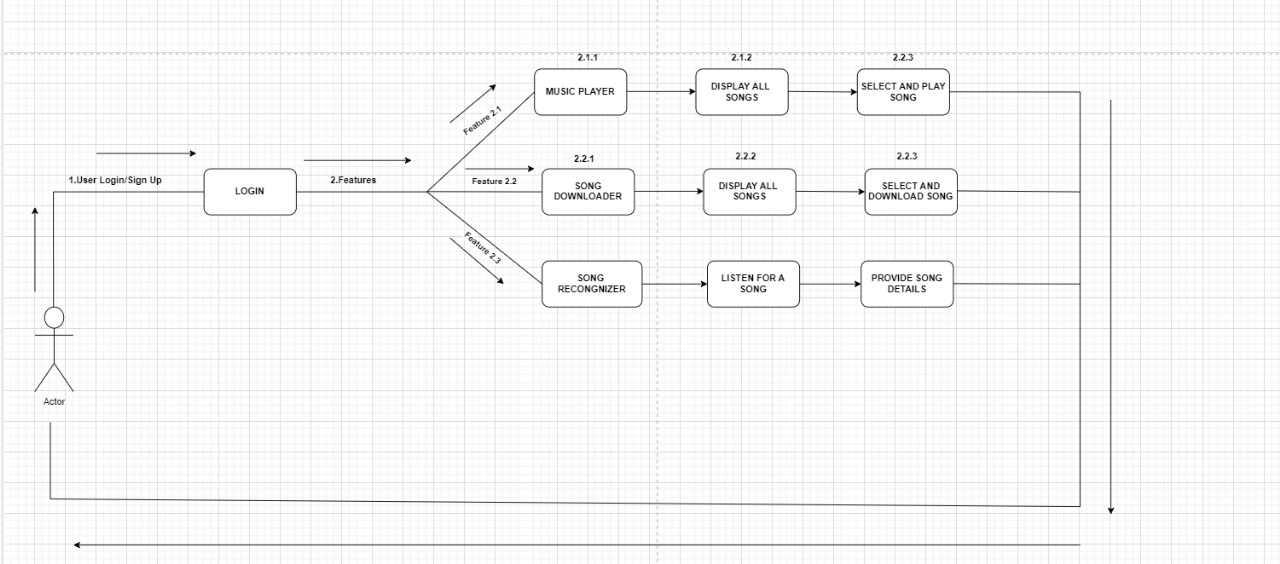


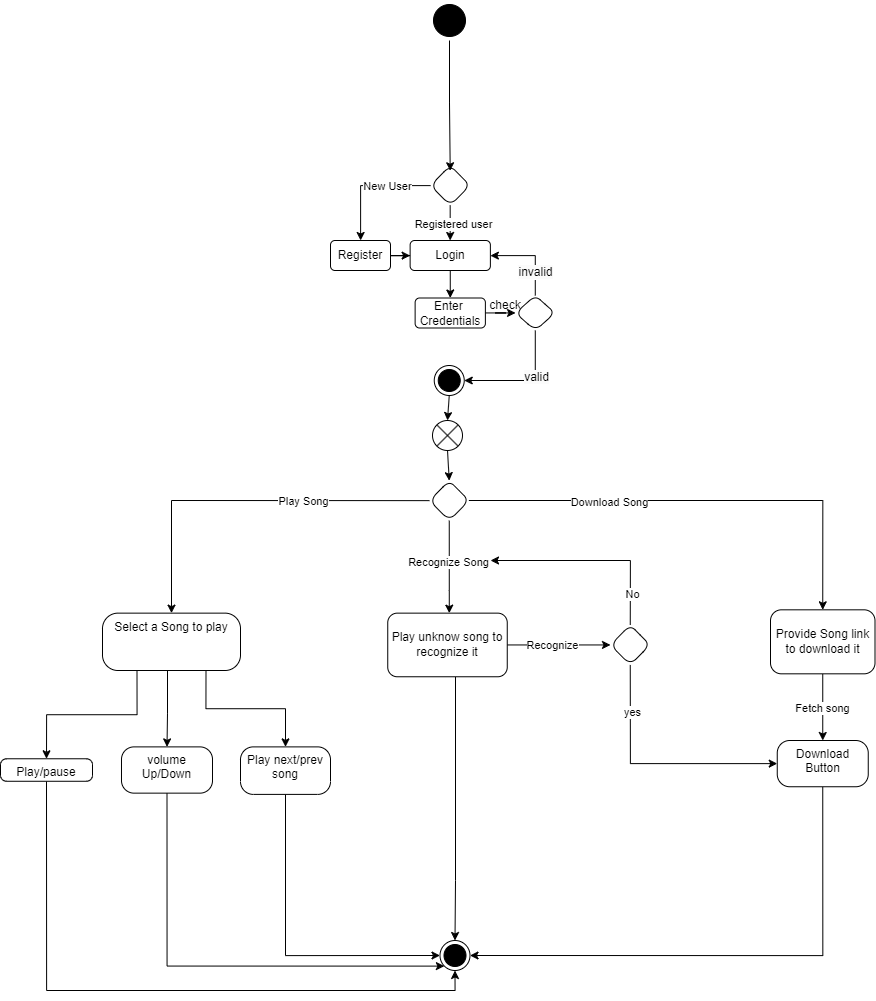
**Class Diagram:**



**Activity Diagram:**

**Collaboration Diagram:**

****

**State Diagram:**

**Functional and Path Coverage Testing**

**Functional Testing for Login Module:**

**Test Case 1: Valid Login**

**Description**: User enters a valid email address and password associated with an existing account.

**Expected Result:** The system successfully validates the credentials and grants access to the application.

**Variations:**

* Test with different valid email/password combinations.

**Test Case 2: Empty Email or Password**

**Description:** User enters an empty email address or password.

**Expected Result:** The system displays an error message indicating missing credentials.

**Variations:**

* Test with an empty email and some password.
* Test with some email and empty password.

**Test Case 3: Invalid Email and Password**

**Description:** User enters an email address or password that is not associated with an existing account.

**Expected Result:** The system displays an error message indicating invalid credentials.

**Variations:**

* Test with different combinations of invalid emails and passwords.
* Verify the message differentiates between invalid credentials and empty fields (if applicable).

**Functional Testing for Song Recognition Module:**

**Test Case 1: Successful Song Recognition**

**Description:** User activates the song recognizer, plays a song through the device microphone.

**Expected Result:** The module successfully captures the audio, recognizes the song accurately, and displays song information (title, artist, album).

**Variations:**

* Test with different music genres and audio qualities.
* Verify song information accuracy for recognized songs.

**Test Case 2: Song Recognition Failure - Insufficient Audio Clarity**

**Description:** User activates the song recognizer and plays a song with background noise or poor audio quality.

**Expected Result:** The module fails to recognize the song and informs the user about the issue.

**Variations:**

* Simulate different background noise scenarios.
* Test with varying audio qualities (low volume, muffled sound).

**Test Case 3: Song Recognition Failure - Information Not Found**

**Description**: User activates the song recognizer and plays a song not present in the recognition database.

**Expected Result:** The module fails to recognize the song and informs the user that the song information is not available.

**Variations:**

* Test with songs from different languages or genres not typically in the database.

**Test Case 4: Download Song**

**Description:** User successfully recognizes a song and chooses to download it (if the functionality exists).

**Expected Result:** The module downloads the song successfully and stores it within the application.

**Variations:**

* Test downloading capabilities with different internet connection speeds.
* Verify downloaded songs are playable within the application.

**Path Coverage Testing**

**Path Coverage Testing for Login Module:**

**Pseudo code for Login Module:**

input email and password

if (! email ||! password) {

print error

}

else {

if((email||password) is not valid){

print invalid user

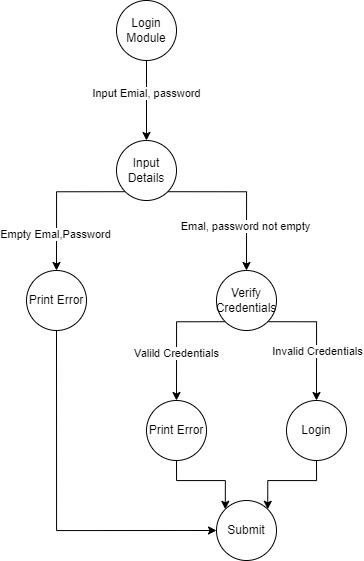
}

else {

print login successful

}

}



Based on the provided paths and considering a cyclomatic complexity (CC) of 3, here's the complete path coverage testing for the login module:

**Paths:**

Empty Email or Password: User enters an empty email or password (but not both).

Invalid Email and Password: User enters an invalid email and password (neither are saved).

Valid Email and Password (Existing User): User enters a valid email and password that are saved in the system.

Test Cases:

**Test Case 1: Empty Email**

Input: Email = "", Password = "valid\_password"

Expected Output: "Error" message

**Test Case 2: Empty Password**

Input: Email = "[email address removed]", Password = ""

Expected Output: "Error" message

**Test Case 3: Invalid Email and Password**

Input: Email = "invalid\_email", Password = "invalid\_password"

Expected Output: "Invalid details" message (assuming the message differentiates between invalid credentials and empty fields)

**Test Case 4: Valid Email and Password (Existing User)**

Input: Email = "[email address removed]", Password = "correct\_password" (assuming these credentials are saved)

Expected Output: "Successfully logged in" message

**Path Testing for Song Recognizer Module:**

Given a song recognizer module with a cyclomatic complexity (CC) of 2, there are two primary execution paths to consider. Pseudo code for Login Module:

**Pseudo code for Song Recognizer Module:**

input song

Recognize song

if(song data){

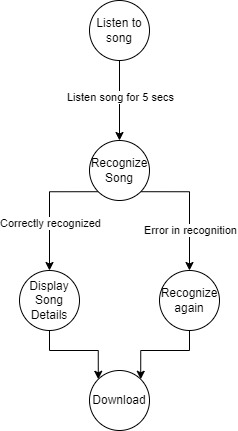
display song details and download song

}

else{

print no song found

}



**Paths:**

1. **Song Recognized Successfully (Success Path):**

* User activates the song recognizer.
* The module successfully captures audio from the device microphone.
* The module processes the audio and identifies the song accurately.
* The user is presented with the recognized song information (title, artist, album).
* The user chooses to download the recognized song (leads to download functionality testing).

1. **Song Recognition Failure (Error Path):**

* User activates the song recognizer.
* The module captures audio from the microphone (potential failure point not tested in this path).
* The module fails to recognize the song due to:
* Insufficient audio clarity (e.g., background noise)
* Song information not present in the recognition database
* Module encountering technical errors
* The module informs the user about the recognition failure.
* The user has the option to:
* Retry recognition (loops back to the beginning of path 2)
* Cancel the song recognition process

**Description**

1. **User Sign-up**
   1. The user opens the Beat Buddy application and selects the “Sign Up” option.
   2. The sign-up form requires the user to enter the following information:

* Full Name
* Username
* Email address (for verification and potential password reset)
* Password (with enforced complexity requirements)
  1. The user clicks the “Sign Up” button.
  2. Upon successful verification, Beat Buddy creates a new user account in the system database, storing the user’s credentials securely (hashed and salted passwords).
  3. The user is redirected to the login page or a welcome message, indicating successful account creation.

**2. User Login**

2.1. The user opens the Beat Buddy application and selects the “Login” option.  
2.2. The login form requires the user to enter

* Email
* Password

2.3. The user clicks the “Login” button.

2.4. Beat Buddy performs the following action:

* Retrieves the user’s account information from the database based on the provided username.
* Compares the entered password with the stored for the user account.

2.5. If the username and password matches:

The user is directed to the Beat Buddy’s main interface with access to personalized features. 2.6. If the username or password is incorrect, Beat Buddy displays an error message prompting the user to retry with the correct credentials.

**3.Music Playlist and Management**

3.1.The User can directly access a list of songs which are downloaded in the backend that is Database

3.2. We allow users to Search for a particular song and provides functionalities such as Playing and Pausing the song.

3.3. We allow users to move to next song by using Next Play Button and previous song by Previous Play Button

3.4. We also allow users to maintain the frequency levels of the song (Volume increase and decrease)

**4. Song Download**

4.1.The user expresses their desire to download a song. Beat Buddy offers the follows the following method:

4.2**. Downloading a song using Spotify Playlist**

4.2.1. **Link Input:** The user enters a valid Spotify link targeting a specific playlist in the provided search bar

4.2.2. **Link Parsing:** Beat Buddy parses the link to extract relevant information about the songs to be downloaded.

4.2.3. **API Interaction:** Considering copyright limitations and fair use policies, Beat Buddy interacts with the YouTube API to search for downloadable versions of the songs based on the Spotify information.

4.2.4. **Download Process:** Based on search results and availability, the download process is initiated for the corresponding songs.

4.3. Beat Buddy displays download progress to the user, along with any errors encountered during the download process (e.g., network issues, file availability).

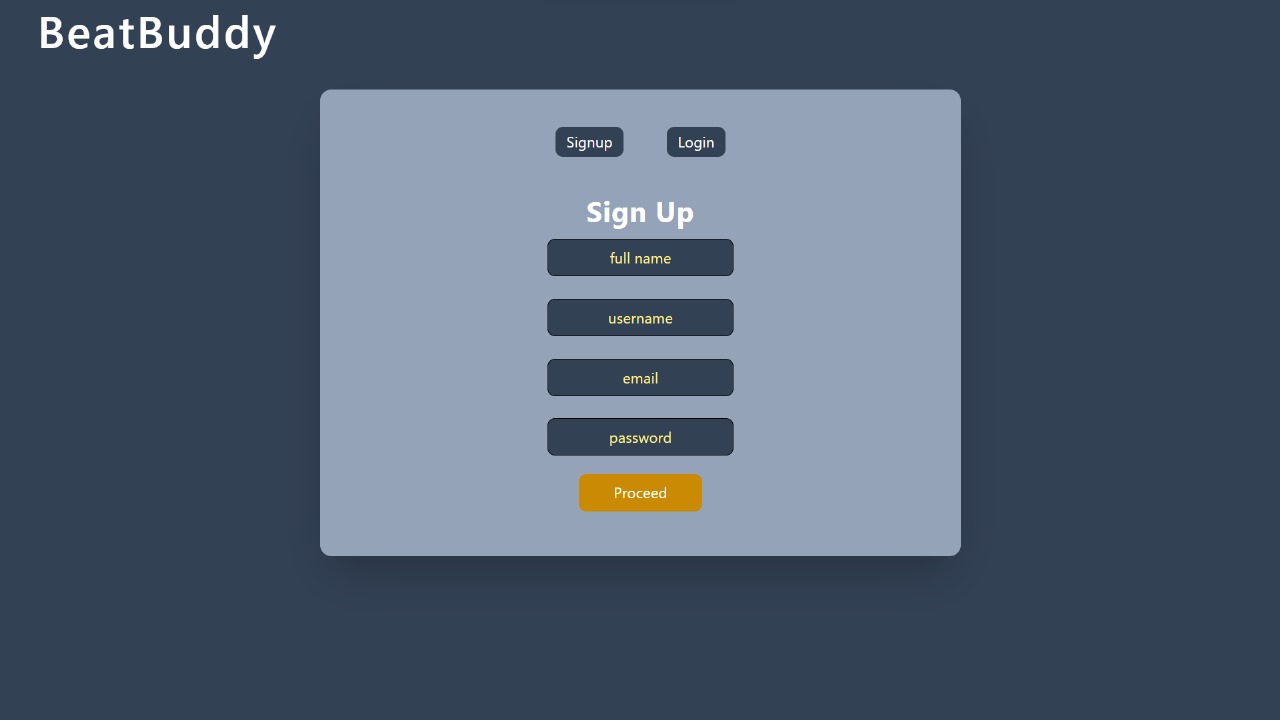
4.4. Upon successful download, the song is stored locally on the user’s device in a designated folder and prompt is displayed on the screen indicates that the Song has been downloaded.

**5. Song Recognition**

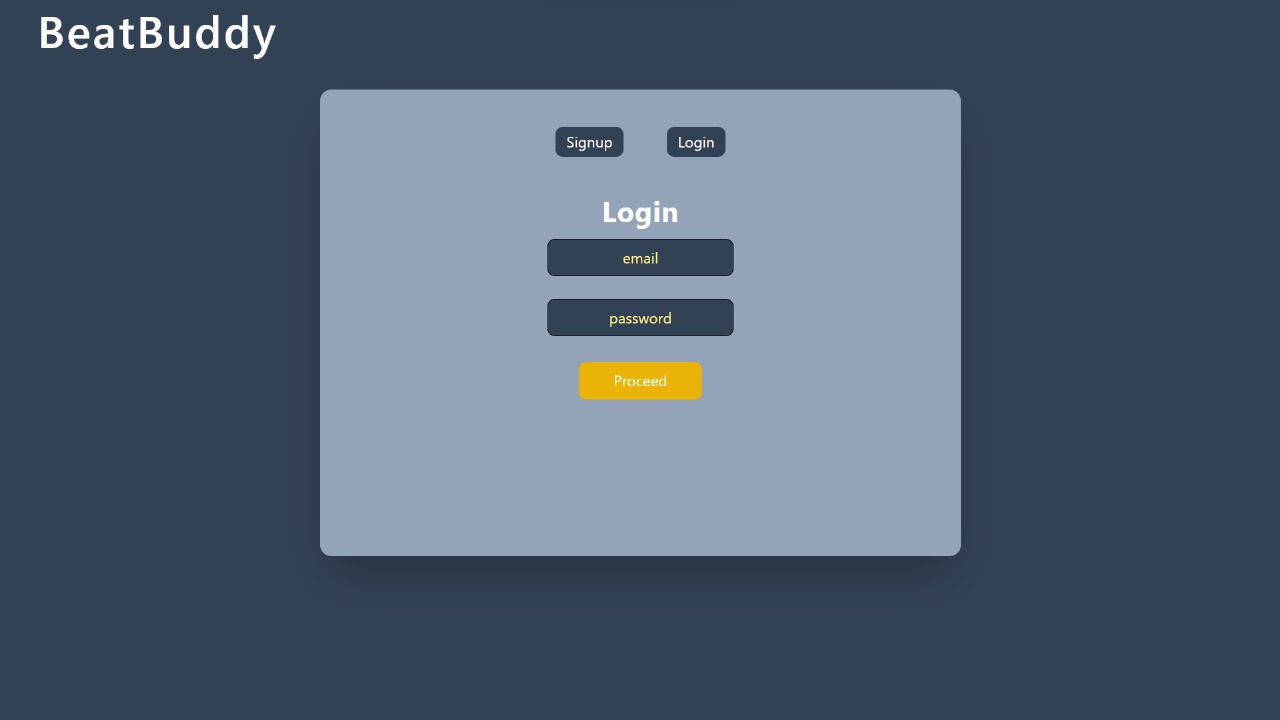
1. **User Activation:** The user initiates song recognition by pressing a dedicated button or using a voice command.
2. **Audio Capture:** The Beat Buddy application starts recording audio input from the device’s microphone.
3. **API Communication:** The captured audio data is transmitted securely to the chosen third-party song recognition API (e.g., Shazam).
4. **Song Identification:** The API processes the audio data using advanced algorithms to identify the song.
5. **Response Retrieval:** If successful, the API returns a response containing information about the identified song, including:
6. Title
7. Artist
8. Album (and potentially other details like genre, year of release)
9. Download Button (we can download the following song if necessary)
10. **Information Display:** Beat Buddy receives the response from the API and displays the identified song information to the user in a clear and user-friendly manner

**Snap Shots of our Project**

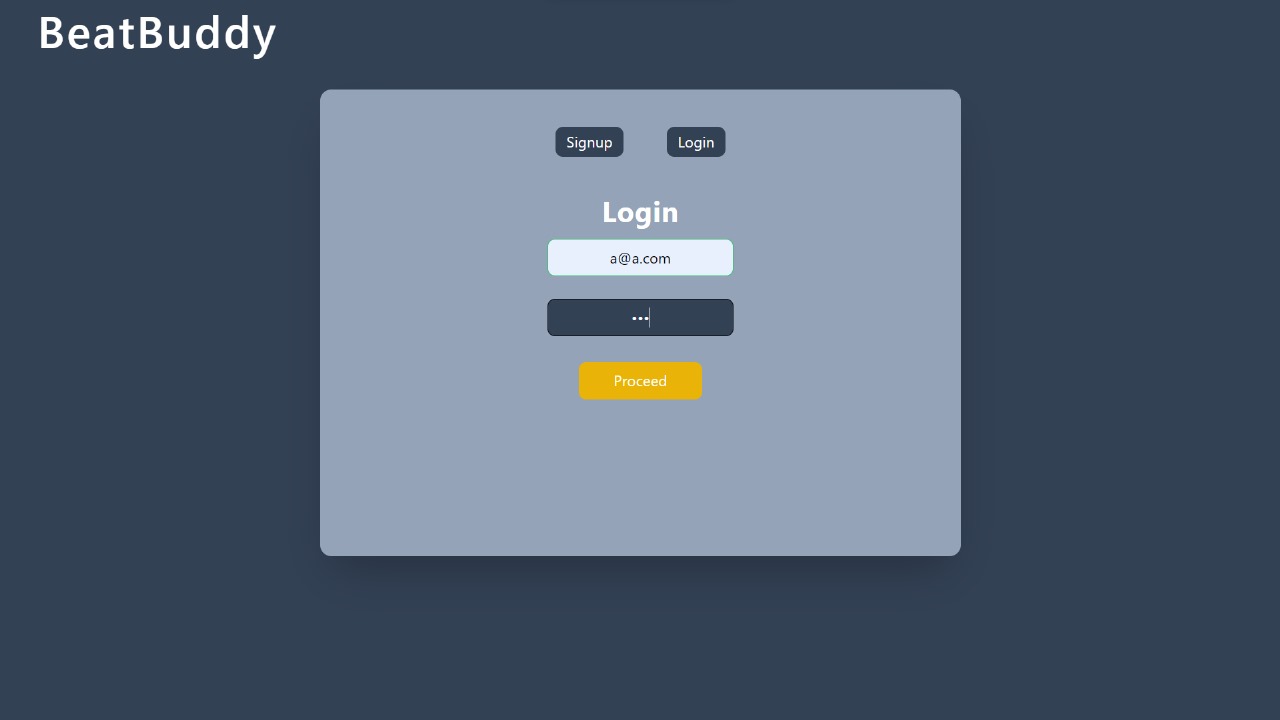
**Sign Up Page**



**Login Page**



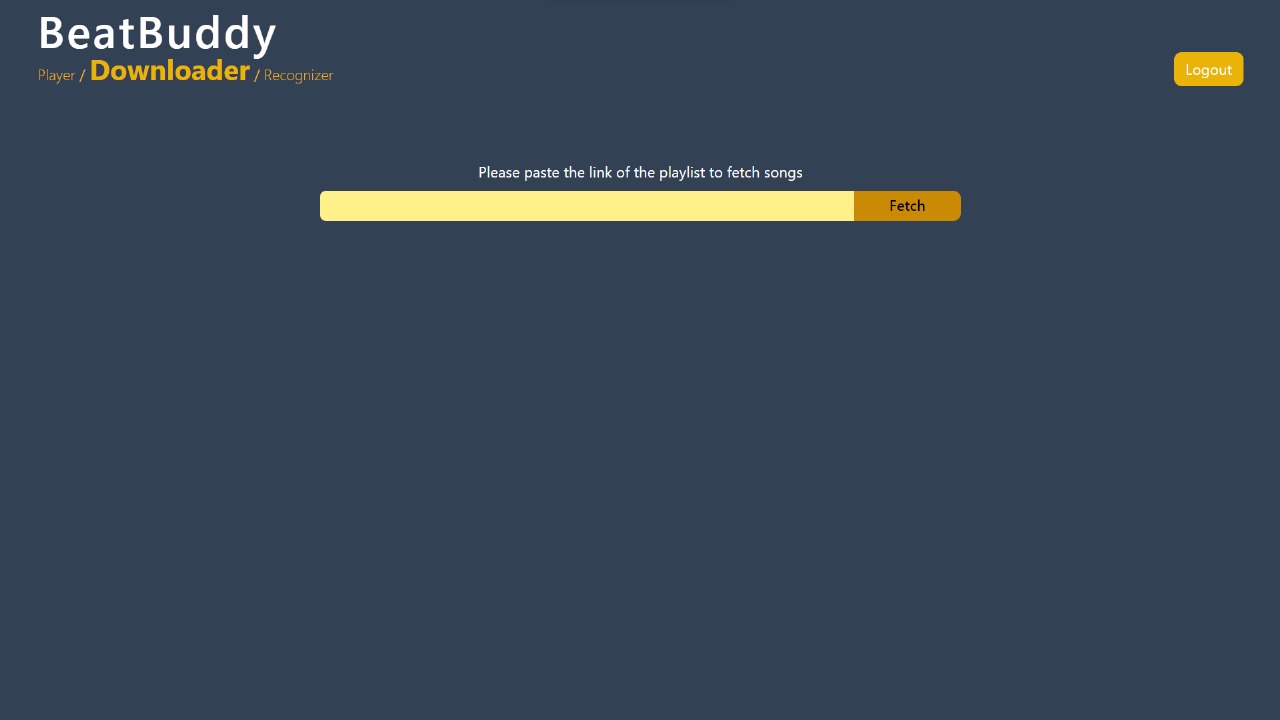
**During Login Process**



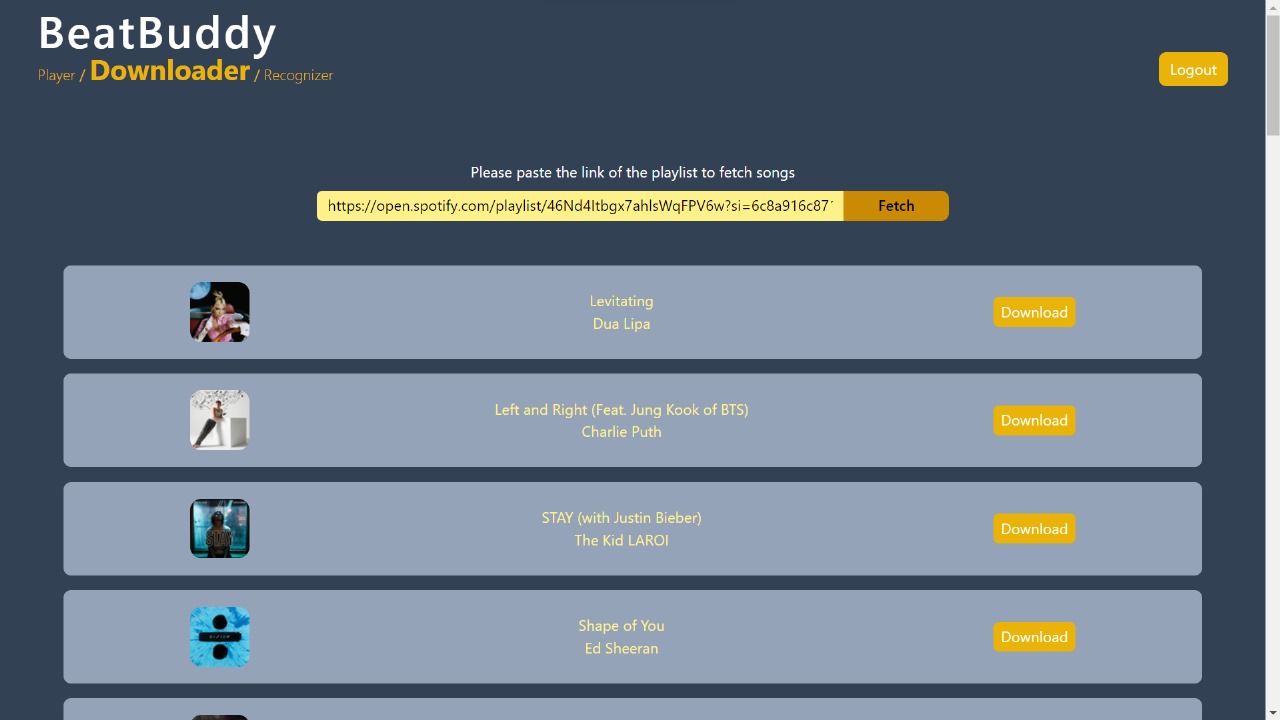
**Music Player**



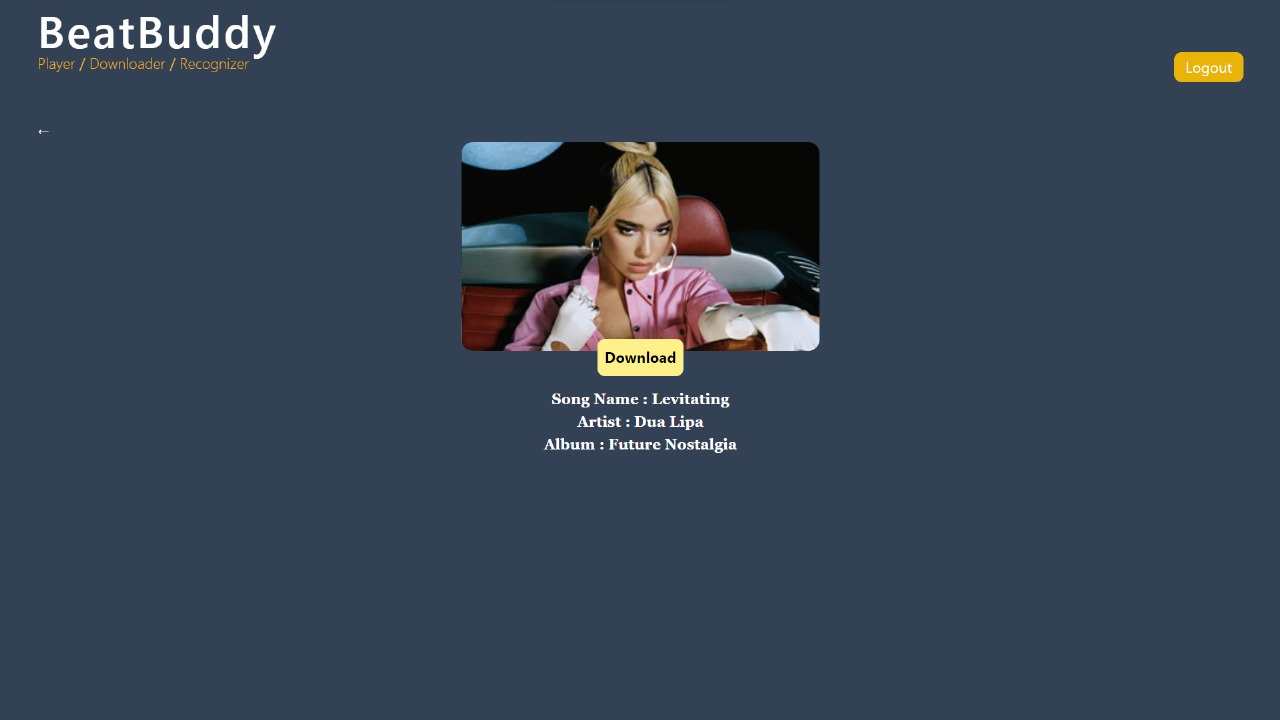
**Song Downloader Interface**



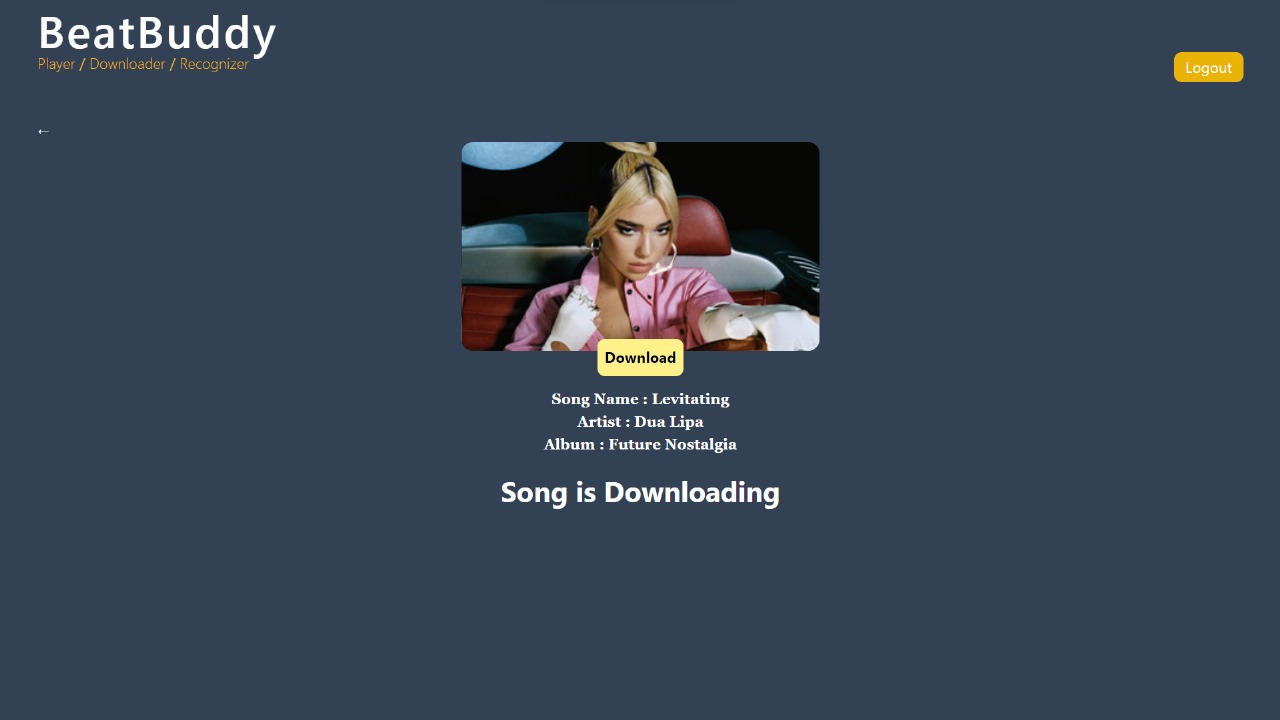
**Display all the songs in Spot**



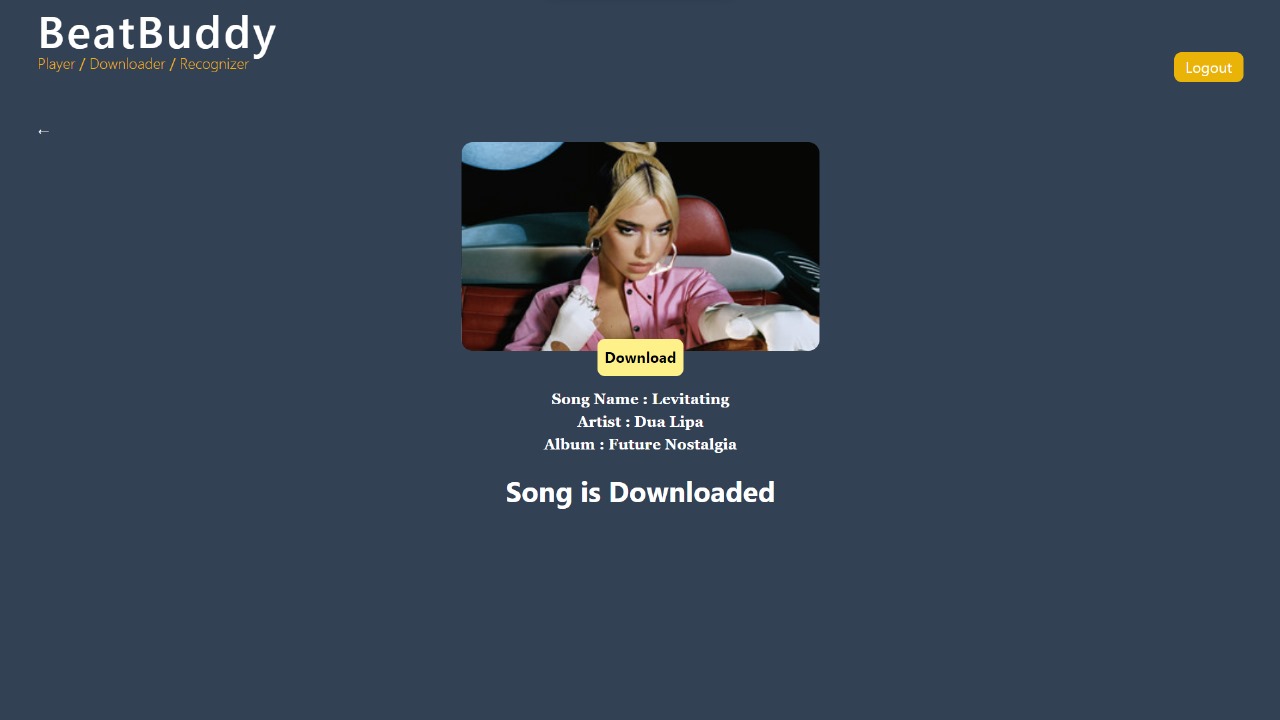
**Song Preview in Download Section**



**Song is Downloading**



**Song is Downloaded in the playlist**



**Downloaded Song will be in our Playlist**



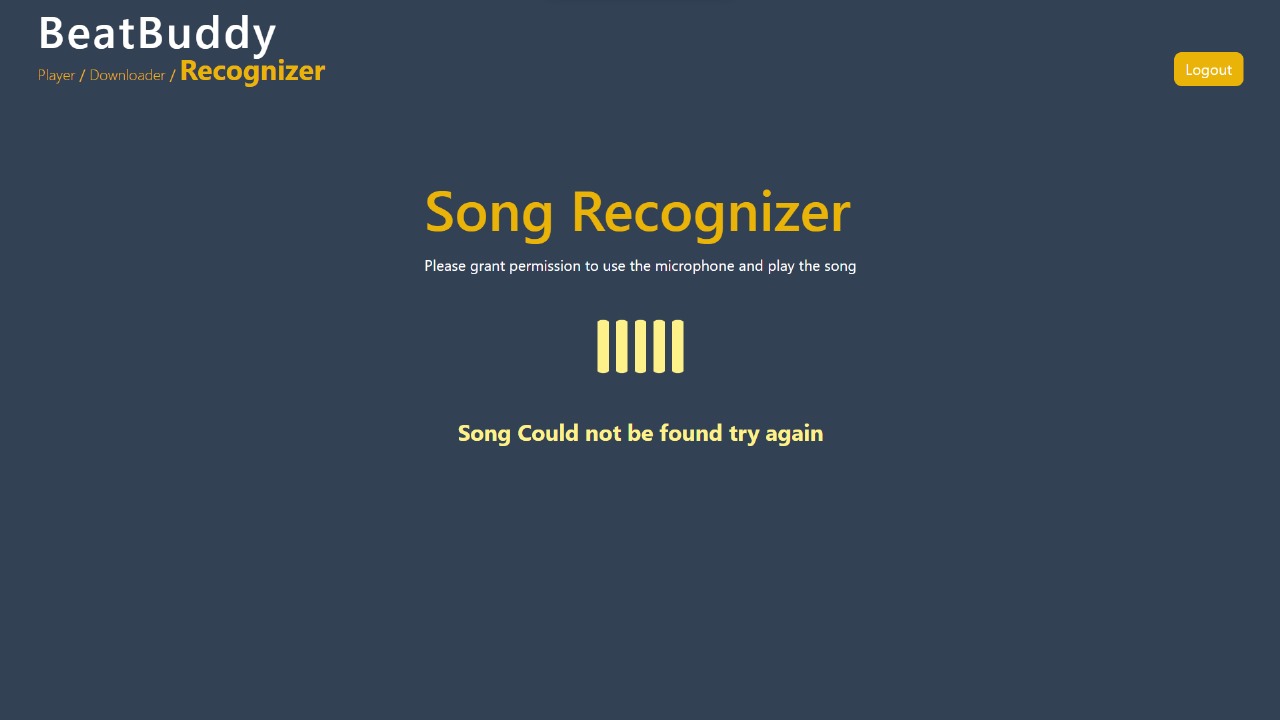
**Song Recognizer Page**



**Song Recognized Successfully**



**ERROR – When Song is not Recognized**



**Log Out Button**



**Conclusion**

Beat Buddy goes beyond simply managing music. Its intuitive design empowers users to become active curators of their musical journeys. Imagine effortlessly identifying a captivating song playing in a cafe, instantly downloading it for offline listening, and seamlessly adding it to a personalized playlist for your next workout. Beat Buddy fosters this level of engagement by removing the friction from music discovery and organization. This newfound freedom allows users to delve deeper into their musical passions, explore new genres with ease, and curate playlists that perfectly suit their mood and activity.

The potential of Beat Buddy extends far beyond its current feature set. Its focus on user-centricity and adaptability positions it as a platform poised for continuous evolution. As music streaming services and technologies change, Beat Buddy can seamlessly integrate to remain at the forefront. Imagine future versions offering personalized music recommendations, intelligent playlist creation based on listening habits, or even integration with smart speakers for voice-controlled music management. By fostering a culture of innovation and user feedback, Beat Buddy has the potential to redefine the way we experience music for years to come.

**Limitations**

Here are some potential limitations to consider for our music application:

**Functionality:**

* **Limited Music Library:** The application might rely on a specific music database or streaming service, potentially restricting the available songs for search or recognition.
* **Offline Functionality:** The core functionalities might depend on an internet connection, limiting usability without a stable network. Background playback might still function with downloaded songs (if implemented).
* **Offline Song Recognition (Optional):** If implemented, the offline recognition feature might have a smaller database compared to online access, potentially reducing accuracy or song availability.
* **Search Accuracy:** Search results might not be perfect, especially for complex queries or songs with variations in titles or artist names.

**Technical Aspects:**

* **Device Compatibility:** The application might be designed for specific operating systems or devices, limiting accessibility for users with different platforms.
* **Performance:** Performance might vary depending on the user's device capabilities and internet connection speed. Background playback could drain battery life faster.
* **Scalability:** The application might need improvements to handle a large user base or a vast music library, potentially impacting search speed or recognition accuracy.

**Security:**

* **Data Security:** User data, such as search history or downloaded songs, might require security measures to prevent unauthorized access or breaches.
* **Song Download Security:** Downloaded songs might require DRM (Digital Rights Management) or other mechanisms to protect copyright if applicable.

**Other Limitations:**

* **User Interface (UI) Complexity:** The inclusion of new features like search bars, playback controls, and recognition details might add complexity to the UI, requiring careful design for user-friendliness.
* **Development and Maintenance:** Ongoing development and maintenance efforts might be required to address bugs, introduce new features, and adapt to changing technologies.

**References**

* Node.js Documentation: <https://nodejs.org/en>
* React Documentation: <https://legacy.reactjs.org/docs/getting-started.html>
* Rapid API Documentation:<https://rapidapi.com/>
* YouTube API Documentation: <https://developers.google.com/youtube/v3>
* <https://vitejs.dev/guide/>
* <https://reactrouter.com/en/main>
* <https://www.npmjs.com/>
* <https://www.mongodb.com/>
* <https://tailwindcss.com/>

**Future Works**

Here are some directions for future work on our music application, incorporating the planned features of a search bar, background playback, and displaying related songs during recognition:

1. **Search Functionality:** Search Implementation: Integrate a search bar that allows users to search for music by title, artist, album, or even lyrics.
2. **Search Filters:** Implement filters to refine search results based on genre, release date, or other relevant criteria.
3. **Search Results:** Display search results clearly, including song title, artist name, album cover, and possibly a short preview snippet.
4. **Search Accuracy:** Ensure the search functionality delivers accurate and relevant results based on user queries.
5. **Background Playback:**

* **Background Playback Integration:** Implement background playback functionality that allows users to listen to music while using other apps on their device.
* **Playback Controls**: Integrate controls for background playback, such as play/pause, skip track, and adjust volume, accessible from the notification bar or a dedicated mini-player.
* **Battery Optimization:** Consider battery consumption when implementing background playback and offer options for users to manage this aspect.
* **User Interface:** Design a user-friendly interface for interacting with background playback controls.

1. **Song Recognition Enhancements:**

* **Related Song Display:** During song recognition, display information about related songs by the same artist, similar genre, or other relevant criteria.
* **Multiple Song Matches:** Handle scenarios where multiple songs might match the audio. Allow users to choose the correct song from a list of potential matches based on displayed information or short previews.