

**A  
Project Report  
on**

**MUSIC RECOMMENDATION SYSTEM**

**BTech-IT, Sem VI**

Prepared By:  
Pankti Patel (IT103)  
Vaibhavi Patel (IT108)

Guided By:  
Prof. Sunil K. Vithlani  
Department of Information Technology



**Department of Information Technology  
Faculty of technology,  
Dharmsinh Desai University  
College road, Nadiad- 387001**

**April, 2024**



## CANDIDATE'S DECLARATION

We declare that 6<sup>th</sup> semester report entitled “Music Recommendation System” is our own work conducted under the supervision of the guide Prof. Sunil K. Vithlani.

We further declare that to the best of our knowledge the report for B.Tech. VI semester does not contain part of the work which has been submitted either in this or any other university without proper citation.

Candidate's Signature  
Candidate's Name: Pankti Patel  
Student ID: 21ITUOS021

Candidate's Signature  
Candidate's Name: Vaibhavi Patel  
Student ID: 21ITUBS072

**DHARMSINH DESAI UNIVERSITY**  
**NADIAD-387001,**  
**GUJARAT**



**CERTIFICATE**

This is to certify that the project carried out in the subject of Project-1, entitled “Music Recommendation System” and recorded in this report is a bonafide report of work of

1) Pankti Patel	Roll No. IT103	ID No:21ITUOS021
2) Vaibhavi Patel	Roll No. IT108	ID No: 21ITUBS072

of the Department of Information Technology, semester VI. They were involved in Project work during the academic year 2023 – 2024.

Prof. Sunil K. Vithalani(Project Guide)  
Department of Information  
Technology,  
Faculty of Technology,  
Dharmsinh Desai University,  
Nadiad  
Date:

Prof. (Dr.) Vipul K. Dabhi,  
Head, Department of  
Information Technology,  
Faculty of Technology,  
Dharmsinh Desai University, Nadiad  
Date:



## ACKNOWLEDGEMENT

On the very outset of this report, we would like to extend our sincere & heartfelt obligations towards all the personages who have directly or indirectly helped in this endeavor. Without their active guidance, help, cooperation & encouragement, we wouldn't have made the headway in the project.

We are ineffably indebted to our instructor **Prof. Sunil K. Vithlani** for guidance, encouragement and conscientiousness to accomplish this project. We are extremely thankful for his support.

We would also like to express our gratitude to our Head, **Dr. Vipul Dabhi** along with Dharmsinh Desai University for giving us this opportunity.

Any omission in this brief acknowledgement does not mean lack of gratitude.

**Thank You,**

**Pankti Patel  
Vaibhavi Patel**



## TABLE OF CONTENTS

ABSTRACT .....	i
LIST OF FIGURES .....	ii
LIST OF TABLE .....	iii
1 INTRODUCTION .....	1
1.1 INTRODUCTION .....	1
1.2 PURPOSE .....	1
1.3 SCOPE .....	1
1.4 OBJECTIVE .....	1
1.5 TECHNOLOGY AND LITERATURE REVIEW .....	2
2 PROJECT MANAGEMENT .....	4
2.1 FEASIBILITY STUDY .....	4
2.1.1 Technical Feasibility .....	4
2.1.2 Time Schedule Feasibility .....	4
2.1.3 Operational Feasibility .....	4
2.1.4 Implementation Feasibility .....	5
2.2 PROJECT PLANNING .....	6
2.2.1 Project Development Approach and Justification .....	6
2.2.2 Project Plan .....	7
2.2.3 Milestones and Deliverables .....	7
2.2.4 Roles And Responsibilities .....	8
2.2.5 Group Dependencies .....	8
2.3 PROJECT SCHEDULING .....	9
3 SYSTEM REQUIREMENTS .....	10
3.1 STUDY OF CURRENT SYSTEM .....	10
3.2 PROBLEMS AND WEAKNESS OF CURRENT SYSTEM .....	10
3.3 USER CHARACTERISTICS .....	11
3.4 HARDWARE AND SOFTWARE REQUIREMENTS .....	11
3.5 CONSTRAINTS .....	12
3.5.1 Regulatory Policies .....	12
3.5.2 Hardware Limitations .....	12
3.5.3 Interfaces to Other Application .....	13

3.5.4 Parallel Operation .....	13
3.5.5 Higher Order Language Requirements .....	14
3.5.6 Reliability Requirements .....	14
3.5.7 Criticality of the Application .....	14
3.5.8 Safety and Security Consideration .....	14
3.6 ASSUMPTIONS AND DEPENDENCIES .....	15
4 SYSTEM ANALYSIS .....	16
4.1 REQUIREMENT OF NEW SYSTEM .....	16
4.1.1 User Requirements .....	16
4.1.2 System Requirements .....	16
4.1.3 Non-Functional Requirements .....	17
4.2 FEATURES OF NEW SYSTEM .....	18
4.3 NAVIGATION CHART .....	19
4.4 SYSTEM ACTIVITY .....	19
4.5 DATA MODELING .....	21
4.5.1 Data Dictionary .....	21
4.5.2 ER Diagram .....	21
5 SYSTEM DESIGN .....	22
5.1 SYSTEM ARCHITECTURE DESIGN .....	22
5.1.1 Class Diagram .....	22
5.1.2 Sequence Diagram .....	23
5.1.3 Component Diagram .....	23
5.1.4 Deployment Diagram .....	24
5.2 USECASE DIAGRAM .....	25
5.3 ACTIVITY DIAGRAM .....	26
5.4 DATA FLOW DIAGRAM .....	27
5.4.1 Level-0 (Context level) .....	27
5.4.2 Level-1 (Context level) .....	27
5.4.3 Level-2 (Context level) .....	28
5.5 STATECHART DIAGRAM .....	29
6 IMPLEMENTATION PLANNING .....	30
6.1 IMPLEMENTATION ENVIRONMENT .....	30
6.2 PROGRAM/MODULES SPECIFICATION .....	30
6.3 CODING STANDARDS .....	31
7 TESTING .....	34
7.1 TESTING PLAN .....	34

7.2 TESTING STRATEGY .....	34
7.3 TESTING METHODS .....	35
7.4 TEST CASES .....	36
8 USER MANUAL .....	37
9 LIMITATIONS AND FUTURE ENHANCEMENT .....	40
9.1 LIMITATIONS .....	40
9.2 FUTURE ENHANCEMENTS .....	40
10 CONCLUSION AND DISCUSSION .....	41
10.1 CONCLUSION .....	41
10.2 DISCUSSION .....	41
10.2.1 Self-Analysis of Project Viabilities .....	41
10.2.2 Problems Encountered and Possible Solutions .....	41
10.2.3 Summary of Project Work .....	42
REFERENCES: .....	43



## **ABSTRACT**

In today's world, many businesses enhance online user experience using various recommendation systems, which have a growing innovation & research interest. We develop recommendation system based on user's current appearance. To recommend the songs we consider the features like previous listening history, preferences, popularity & artists. It is pleasant if user get the nearest content of their choice. We develop a recommendation System using supervised machine learning models and analyze the results. We used Spotify API data for our experiment.



## LIST OF FIGURES

Figure 2.1	IMPLEMENTATION FEASIBILITY .....	5
Figure 2.2	Agile Model .....	6
Figure 2.3	PROJECT SCHEDULING CHART .....	9
Figure 4.1	NAVIGATION CHART .....	19
Figure 4.2	ER Diagram .....	21
Figure 5.1	CLASS DIAGRAM .....	22
Figure 5.2	SEQUENCE DIAGRAM .....	23
Figure 5.3	COMPONENT DIAGRAM .....	23
Figure 5.4	DEPLOYMENT DIAGRAM .....	24
Figure 5.5	USECASE DIAGRAM .....	25
Figure 5.6	ACTIVITY DIAGRAM .....	26
Figure 5.7	DFD LEVEL-0 .....	27
Figure 5.8	DFD LEVEL-1 .....	27
Figure 5.9	DFD LEVEL-2 .....	28
Figure 5.10	STATECHART DIAGRAM .....	29
Figure 6.1	App.js .....	31
Figure 6.2	userController.js .....	32
Figure 6.3	userAuthenticate.js .....	32
Figure 6.4	App.py .....	32
Figure 6.5	Model.py .....	33
Figure 8.1	HOME PAGE .....	37
Figure 8.2	SIGN UP PAGE .....	37
Figure 8.3	LOGIN PAGE .....	38
Figure 8.4	LOGIN FAIL .....	38
Figure 8.5	LOGIN SUCCESSFUL .....	39
Figure 8.6	RECOMMENDATION PAGE .....	39



## **LIST OF TABLE**

Table 2.1	PROJECT PLAN .....	7
Table 2.2	ROLES AND RESPONSIBILITIES .....	8
Table 2.3	GROUP DEPENDENCIES .....	8
Table 4.1	DATA DICTIONARY .....	21
Table 7.1	TEST CASES .....	36

# **1 INTRODUCTION**

## **1.1 INTRODUCTION**

A Recommendation systems, which first appeared in the 1990s, have advanced significantly in recent years, particularly with the introduction of Machine Learning and networks. On the one hand, the expanding use of today's digital world, which is characterized by a wealth of data, has enabled us to collect massive user databases. On the other hand, when computing power increased, it became possible to handle these data, particularly using Machine Learning, when human skills were no longer capable of conducting a thorough examination of such a large amount of data. Unlike search engines, which get queries with specific information about what the user wants, a recommendation system does not receive a direct request from the user, but instead must provide them fresh options based on their past behaviors.

## **1.2 PURPOSE**

The purpose of this document is to define the requirements for the development of a Music Recommendation System. The goal of this project was to learn about machine learning and its fundamental concepts, as well as numerous data mining approaches and algorithms. Another goal was to become familiar with a variety of machine learning algorithms and how to use them. By using music recommender system, the music provider can predict and then offer the appropriate songs to their users based on the characteristics of the music that has been heard previously.

## **1.3 SCOPE**

The Music Recommendation System will provide personalized music recommendations based on user preferences, history, and other relevant factors. Current system mainly focuses on doing the recommendation on content based features that is based on its acoustic parameters. But in future hybrid model can be created that uses both content based and collaborative(user history, feedback, liking etc.) for recommendations.

## **1.4 OBJECTIVE**

The objective is to recommend the user the types of songs that he would like by comparing his taste i.e., his playlist of music with the songs available in the dataset. By using music recommender system, the music provider can predict and then offer the appropriate songs to their users based on the characteristics of the music that has been heard previously.

## 1.5 TECHNOLOGY AND LITERATURE REVIEW

Technology:

Frontend: React JS

Backend: Python, Flask[1], Spotify API[2], Mongo DB[3]

Literature Review:

We were awestruck with Spotify's recommendation engine. We always wondered how Spotify manages to recommend that perfect song, playlist or even that 'daily mix'. We now have more technology than ever before to ensure that if you're the smallest, strangest musician in the world, doing something that only 20 people in the world will dig, we can now find those 20 people and connect the dots between the artist and listeners. This has been the motivation for this project to use various machine learning techniques and to develop a music recommendation engine similar to that of Spotify, which takes music listening experience to another level.

Recommender systems help consumers deal with the problem of information overload by providing them with individualized, unique content and service suggestions. Various methods for developing recommendation systems have recently been created, including collaborative filtering, content-based filtering, and hybrid filtering. The collaborative filtering approach is the most developed and widely used. Collaborative filtering suggests things by locating other users who have similar tastes to the current user and using their recommendations. Collaborative recommender systems have been used in a variety of settings. problems – new items and new users. The first problem refers to the new items that are meant to be recommended, but the information that is associated with them.

The common characteristics in these systems are constant when using users' preferences compared with users' context (location, mood, weather, etc.). For instance, in the library when people are sitting there maybe they need quiet and melodious music to listen according to the environment where they are in. Last.fm, All music, Spotify, Pandora and Shazam are commercial music recommendation systems which are considered to be excellent systems by focusing on the music already played in order to help the users to find more music. Users are able to connect to a web-based music streaming service to access the recommendations. All the tracks that are played on this stream are recommended.

It is Based on songs or artists which users either upload from your iTunes playlists or add as favorites on the site where users start managing their library of music with tags and keep tracking of the music the friends who listening to and getting multiple recommendations per song played. Additionally, this app filters recommendations by decade, genre, and popularity, as well as builds fabulous playlists (Song et al., 2012[4]).t has been found that CF generally

gives better recommendations than CB. However, this is only true if there is usage data available, such as the ratings given to previous tracks. If this is not the case, then it will not prove accurate results and, consequently, suffer from the Cold-Start problem, which includes two categories of problems – new items and new users. The first problem refers to the new items that are meant to be recommended, but the information that is associated with them.

Meanwhile, many researchers have used social media (Twitter & Facebook) to identify user's mood (tension, depression, anger, vigor, fatigue, confusion) and also identify user's personality (openness, conscientiousness, extraversion, agreeableness, neuroticism) where these are very important factors which influence on user's music taste (Wang et al., 2014[5]; Roberts et al., 2012[6]; Pandarachalil et al. 2015[7]; Ross et al., 2009[8]; Bachrach et al., 2012[9]; Back et al., 2010[10]) and also contextual features (location & event) can lead to different emotional effects due to objective features of the situation or subjective perceptions of the listeners (Scherer et al., 2001[11]).

Music lyrics are also considered to be one of emotional presentation because they include some kinds of implicit thinking, thus we can fully understand emotions and their associated thinking in each song (Nunes and Jannach, 2017[12]; Tintarev and Masthoff, 2008[13]). Cano et al. (2017)[14] mentioned that there is a strong relation between the user mood and listening to the music. The people may want to listen to music which has the same mood of them when they are in specific mood and in contrast the people want to listen to different kind of music which encourage them to enhance their mood and this thing depend on the psychological studies and therefore, the author produced a contextual mood-based music recommender system which is able to regulate the driver's mood and also try to put the driver in a positive mood when driving because listening to the music while driving has always been one of the most favorite activities carried out by people. Finally, similarly, active learning approaches suffer from various limitations.

## **2 PROJECT MANAGEMENT**

### **2.1 FEASIBILITY STUDY**

#### **2.1.1 Technical Feasibility**

The music recommendation system also faces some problems, such as data storage confusion, low computational efficiency, cold start and data sparsity caused by large data scale. The recommendation system is analyzed and studied, and a hybrid recommendation algorithm based on collaborative filtering is designed.

#### **2.1.2 Time Schedule Feasibility**

Developing a music recommendation system can be a time-consuming process, especially if the system needs to be highly customized or integrated with other institutional systems. It is essential to establish a clear project timeline with specific milestones and deadlines to ensure that the development process stays on track.

The time required for developing a music recommendation system can also depend on the development approach adopted. Agile project management approaches, for example, allow for more flexibility and can adapt to changes in project requirements or user feedback, but may take longer to complete. A more traditional project management approach with a strict timeline can help ensure timely completion, but may not allow for as much flexibility.

Another important factor in the time schedule feasibility of a music recommendation system is the availability of resources for development, including personnel, technology, and funding. Adequate resources must be allocated to ensure that the development process runs smoothly and efficiently.

In general, the time schedule feasibility of a music recommendation system can be optimized by following a well-planned development process, adopting an appropriate project management approach, allocating adequate resources, and continuously monitoring progress against the project timeline.

#### **2.1.3 Operational Feasibility**

From an operational perspective, a music recommendation system can provide significant benefits for users. It can streamline the process of managing recommended playlists, facilitate search tracks and playlists.

However, the success of a music recommendation system will depend on how well it aligns with the existing operations and uses of the users. It is essential to ensure that the system is user-friendly, easy to use, and requires minimal additional resources to manage.

Additionally, the system should be scalable and adaptable to meet the changing needs of the institution over time. It should also be designed to integrate seamlessly with other related recommendation systems, such as movie recommendation systems and podcast recommendation systems, to ensure a cohesive user experience.

The system should also have robust technical support available to assist with any issues that may arise.

#### 2.1.4 Implementation Feasibility

The recommender system is done by calculating cosine similarity of extraction features (equation 1) from one music to another music. The extraction features are in vector form; thus, it is possible to calculate their distance. First, we chose one music for each genre as the basis for the recommender system. Next the prediction of the basis music genre is calculated based on neural networks. The feature vectors that produce before the classification layer are used as a basis for recommendations. After the basis music features are obtained, cosine similarity calculations are performed on other music features.

Implementing a music recommendation system requires careful planning and coordination with various stakeholders, including users and potentially external vendors or consultants. It is important to ensure that all stakeholders are involved in the implementation process and have a clear understanding of the system's purpose and benefits.

Another critical aspect of implementation feasibility is the availability of technical infrastructure and resources. The system must be able to operate effectively and efficiently within the existing IT infrastructure of the institution, including networks, servers, and databases.

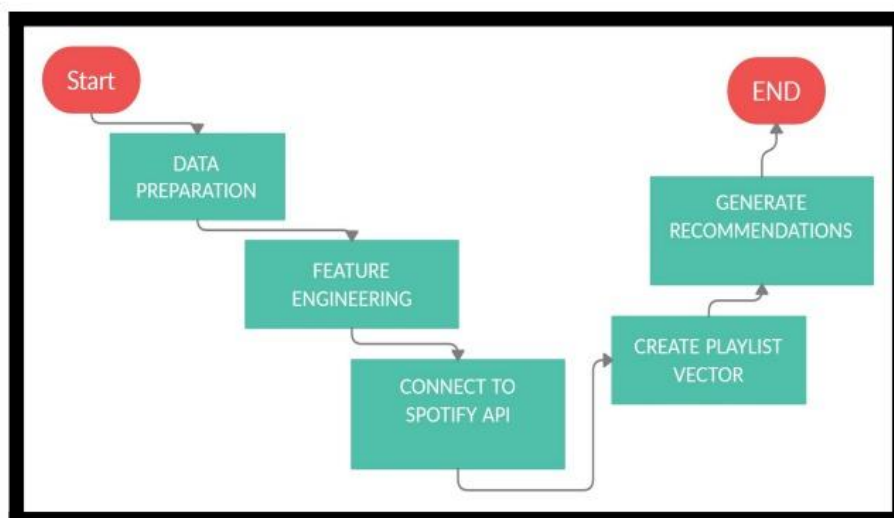


Figure 2.1 IMPLEMENTATION FEASIBILITY

## 2.2 PROJECT PLANNING

### 2.2.1 Project Development Approach and Justification

In the case of a music recommendation system, the Agile model can enable the development team to work closely with users to ensure that the system meets their needs and requirements. The development team can work on delivering features or modules that are most critical or high priority in each sprint. For example, the first sprint could focus on basic functionality such as user registration, login. Subsequent sprints could focus on more advanced features such as recommend songs, research playlists and tracks, and reporting and analytics.

The Agile model can be an effective development approach for a music recommendation system due to the system's complexity and potential for frequent changes in requirements. The Agile model involves breaking down the project into smaller iterations or sprints, with each sprint delivering a working prototype or feature set. This approach allows the development team to gather feedback from stakeholders and make adjustments to the system as needed.

Another advantage of the Agile model is that it promotes frequent communication and collaboration between the development team and stakeholders. This approach can enable the development team to quickly respond to changes in requirements and gather feedback from stakeholders throughout the development process. It can also help to ensure that the system is user- friendly and meets the needs of users.

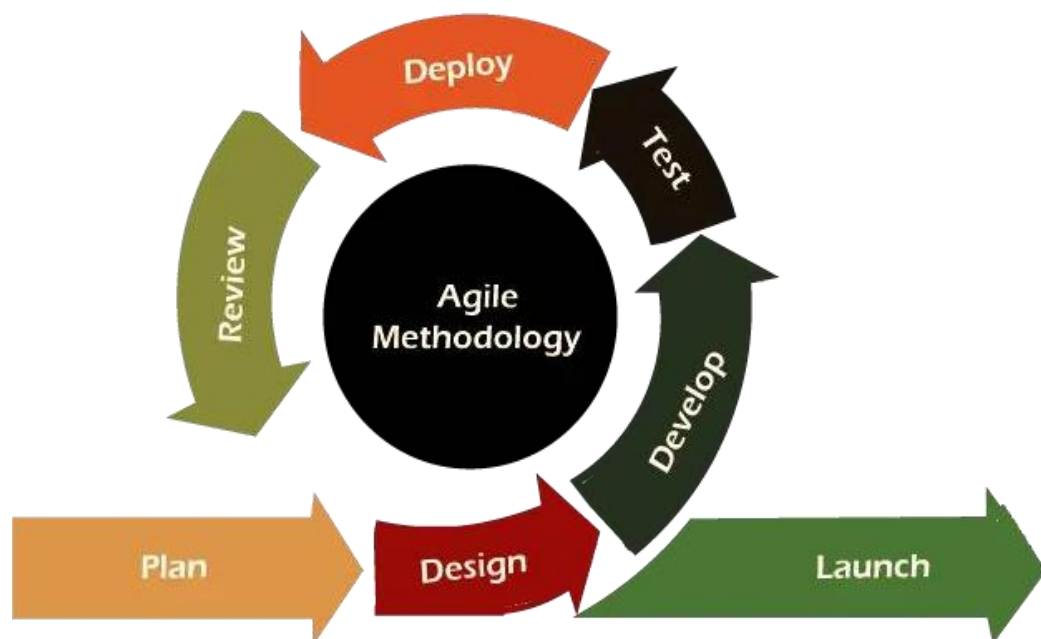


Figure 2.2 Agile Model



### 2.2.2 Project Plan

Table 2.1 PROJECT PLAN

Sr. No.	Documents	Timeline
1	Project Definition, Submission and approval	December
2	Project analysis and design	January
3	Development	February
4	Testing	March
5	Documentation	March-April
6	Deployment	April

### 2.2.3 Milestones and Deliverables

#### Phase 1: Planning and Requirements (1-2 Weeks)

Milestone: Project plan finalized.

Deliverables: Document outlining target users, recommendation goals, data sources, and evaluation metrics. Chosen development approach (collaborative filtering, content-based, or hybrid).

#### Phase 2: Data Collection and Preprocessing (2-3 Weeks)

Milestone: Data collection and cleaning complete.

Deliverables: Secured data source (API, local files) Cleaned and preprocessed music data (formatted for analysis)

#### Phase 3: Model Development and Training (3-4 Weeks)

Milestone: Recommendation model trained and evaluated.

Deliverables: Trained recommendation model Evaluation report with performance metrics (accuracy, precision, etc.)

#### Phase 4: System Design and Implementation (2-3 Weeks)

Milestone: Functional music recommendation system prototype.

Deliverables: Developed user interface for recommendations Integrated recommendation model with the UI

#### Phase 5: Deployment and Testing (1-2 Weeks)

Milestone: System deployed on a platform (web app, mobile app).

Deliverables: Functional music recommendation system deployed on a chosen platform user testing report with feedback on usability and recommendations.

Phase 6: Maintenance and Improvement (Ongoing)

Milestones: Continuous monitoring and improvement of the system.

Deliverables: Ongoing updates to the recommendation model based on user data and feedback

Bug fixes and performance improvements to the system.

## 2.2.4 Roles And Responsibilities

Table 2.2 ROLES AND RESPONSIBILITIES

Name	Analysis	Research	Design	Coding	Testing	Documentation	Maintenance
Pankti	√	√	√	√	√	√	√
Vaibhavi	√		√	√	√	√	

## 2.2.5 Group Dependencies

Table 2.3 GROUP DEPENDENCIES

Name	Dependencies
Data Collection	Spotify API
Data Processing	Pandas, Numpy, Scikit-Learn
Machine Learning	TensorFlow, PyTorch, Scikit-Learn
Database	MongoDB
Web Server	Flask
Frontend	React JS
Authentication	OAuth
Logging	ElasticSearch

## 2.3 PROJECT SCHEDULING

### 2.3.1 Project Scheduling Chart



Figure 2.3 PROJECT SCHEDULING CHART

## **3 SYSTEM REQUIREMENTS**

### **3.1 STUDY OF CURRENT SYSTEM**

The current system of music recommendation varies across the world. Some people may use paper-based systems, such as music reviews and magazines, while others may use digital systems, such as music charts and top hits, concerts.

Paper-based systems can be cumbersome and time-consuming to manage, as Radio Stations, Music Charts, and Top Hits are Focused on general popularity, not individual taste. This might expose listeners to music they don't enjoy.

Current music recommendation systems have made significant strides compared to traditional systems, primarily due to advances in machine learning and data analytics. While traditional systems relied on simplistic algorithms like collaborative or content-based filtering, current systems employ sophisticated models such as deep learning and neural networks. These models analyze vast amounts of data, including user listening history, contextual information, and even physiological data from wearable devices, enabling hyper-personalized recommendations. Unlike traditional systems, which often provided generic suggestions based on aggregated data, current systems focus on individual preferences, leveraging real-time feedback, conversational interfaces, and implicit feedback mechanisms to engage users actively. Additionally, they employ advanced content analysis techniques, analyzing audio features, lyrics, and user-generated content to enhance recommendation accuracy. Moreover, context awareness is a key feature of current systems, adapting recommendations dynamically based on factors like user location, time of day, and social context. They also prioritize serendipity and diversity, introducing novel recommendations while ensuring relevance, thus offering a more engaging and enriching music discovery experience for users.

In summary, Music recommendation systems have come a long way. Today's systems use clever algorithms to analyze huge amounts of data about users and music to suggest songs you'll love. This is a big leap from traditional methods where recommendations came from DJs or record store staff, limiting choices and personalization. While there are concerns about data privacy and recommending new music, current systems offer a much richer way to explore and enjoy music tailored to your unique taste.

### **3.2 PROBLEMS AND WEAKNESS OF CURRENT SYSTEM**

Current existing music recommendation systems have made significant strides in providing personalized recommendations to users. However, they still face several problems and weaknesses, including:

**Cold Start Problem:** This is a major hurdle. When a new user signs up or a new song is added, the system lacks data on user preferences or the song's characteristics. This makes it difficult to generate relevant recommendations.

**Limited Context:** Most systems focus on a user's listening history or song attributes. They might not consider factors like the user's mood, activity (workout vs. relaxation), or time of day.

**Echo Chamber Effect:** Recommendations can become repetitive, suggesting similar artists or genres the user already enjoys. This limits exploration and discovery of new music.

**Data Privacy Concerns:** These systems rely on user data, raising privacy concerns. Finding the right balance between personalization and user data protection is important.

**Data Sparsity:** Even with vast music libraries, user listening data can be sparse. This means there might not be enough information to make accurate recommendations for users with unique tastes.

**Overspecialization:** The system might become too good at recommending similar music, hindering the discovery of genres or artists outside a user's usual preferences.

### 3.3 USER CHARACTERISTICS

Each part of the user interface intends to be as user friendly as possible. The fonts and buttons used will be intended to be very fast and easy to load on web pages. The pages will be kept light in space so that it won't take a long time for the page to load.

The starting page will be the home page of the system. All the features available are displayed. The software provides good graphical interface for the front end of the management system so that new users can make use of the system with ease.

### 3.4 HARDWARE AND SOFTWARE REQUIREMENTS

Hardware Requirements:

- A modern computer with at least 4GB of RAM and a dual-core processor for running the system smoothly.
- A dedicated server with a minimum of 8GB of RAM and a quad-core processor if the system is hosted on-premise.
- Sufficient storage space for the database and any associated files, depending on the size of the faculty and the amount of data to be stored.

- A reliable backup system to prevent data loss in case of hardware failure or other issues.
- A network infrastructure with sufficient bandwidth to support multiple users accessing the system simultaneously.

### Software Requirements:

- **Operating System:** The system is compatible with a modern operating system Windows.
- **Web Server:** The system is built on a web server platform such as Apache to enable users to access the system through a web browser.
- **Database Management System:** The system is using a robust database management system MySQL, to store and manage faculty data.
- **Programming Languages:** The system is developed using programming languages such as Java and JavaScript.
- **Frameworks and Libraries:** The system is using modern frameworks and libraries such as SpringBoot and ReactJs to speed up development and ensure code quality.

## 3.5 CONSTRAINTS

### 3.5.1 Regulatory Policies

Music recommendation systems are under scrutiny for potential drawbacks. Regulations might focus on giving users more control. This could involve understanding how recommendations work (transparency) and having a say in how their listening data is used (data privacy). Additionally, rules could aim to prevent bias towards popular music, promoting discovery of new artists and genres.

### 3.5.2 Hardware Limitations

While music recommendation systems focus on complex algorithms and data analysis, hardware limitations can also come into play. The main concern is data can address some hardware limitations, but it doesn't eliminate them entirely. The API itself restricts the number of data calls you can make, potentially limiting the comprehensiveness of your training data. Additionally, while the API delivers the data, your system will likely handle the heavy lifting of analysis and recommendation generation locally. Here, your hardware's processing power and memory become crucial. Even with Spotify's data, large datasets can strain these resources, impacting the system's performance and recommendation quality. Local data storage for processed information or user profiles can also be a concern, especially with a growing user base or complex recommendation models. In short,

the Spotify API is a valuable tool, but project's hardware will still play a significant role in the system's overall effectiveness.

### 3.5.3 Interfaces to Other Application

Leveraging APIs, or Application Programming Interfaces, our system can connect with external applications like Spotify. This opens a treasure trove of music data, including user listening habits, artist details, and vast music libraries. These rich data streams fuel your recommendation engine, allowing it to build a deeper understanding of music and user preferences. The benefits are numerous: deeper data leads to more personalized recommendations, catering to individual tastes. APIs also unlock exciting possibilities for integration, like displaying lyrics alongside songs. However, even with APIs as partners, hardware limitations persist. API calls often have limits, potentially restricting the data available for training. Ultimately, your system's processing power and memory remain crucial, as it performs the heavy lifting of analysis and recommendation generation. Even the vast datasets retrieved via APIs can strain these resources, impacting performance and recommendation quality. In essence, APIs are powerful collaborators, but a strong hardware foundation is still vital for a smoothly operating and effective music recommendation system.

### 3.5.4 Parallel Operation

To address the computational challenges of your music recommendation system, consider utilizing parallel processing. Imagine a team of processors working together, simultaneously analyzing data and generating recommendations. This can dramatically speed things up, especially when dealing with mountains of music data from Spotify or other APIs.

**Swifter Training:** By distributing training tasks across multiple processors, parallel processing drastically reduces the time it takes to train your recommendation algorithms on vast datasets of music and user profiles.

**Real-Time Recommendations:** With parallel processing, your system can churn through complex calculations and data analysis much faster. This allows for generating personalized recommendations on-the-fly, as users interact with the system.

**Enhanced Scalability:** As your user base expands, so will the data demands on your system. Parallel processing empowers you to effortlessly scale your hardware by adding more processors. This ensures smooth operation and efficient recommendation generation even with a large and ever-growing user population.

However, implementing parallel processing effectively requires careful planning. Hardware and software compatibility are crucial. Not all hardware is equally suited for parallel tasks, and your software needs to be specifically designed to effectively leverage multiple processors.

### **3.5.5 Higher Order Language Requirements**

While the core functionality of your music recommendation system can be built with traditional programming languages, higher-order languages (HOLs) offer significant advantages for complex tasks like recommendation algorithms and data analysis. Unlike traditional languages that focus on step-by-step instructions, HOLs provide a more abstract and concise way to express complex operations.

Developing a music recommendation system with the Spotify API demands meticulous documentation across technical and user-facing aspects. Technical documentation should elucidate system architecture, algorithmic methodologies, data handling procedures, and API integration. User guides are essential for navigating account setup, playlist creation, and recommendation retrieval. Transparent explanations of recommendation algorithms, data preprocessing, and privacy measures bolster trust and compliance. Defining performance metrics, evaluation methods, integration protocols, future directions, ethical considerations, and user experience design enriches understanding and fosters stakeholder communication. This comprehensive documentation framework supports system refinement and enhancement endeavors.

### **3.5.6 Reliability Requirements**

Music recommendation system's reliability hinges on two key aspects: data integrity and consistent performance. The system must ensure the data used for recommendations (user preferences, music attributes) is accurate and free from corruption. Furthermore, it should deliver recommendations consistently, even with high user traffic or large datasets, to maintain a smooth user experience.

Lastly, conducting thorough testing, including stress testing and fault tolerance assessments, is imperative to validate the system's reliability under diverse scenarios and ensure it meets performance expectations.

### **3.5.7 Criticality of the Application**

Music recommendation systems, unlike air traffic control systems or financial transaction platforms, wouldn't be classified as critical applications. However, their importance lies in their impact on user experience and engagement. These systems act as curators, helping users discover new music, tailor their listening experiences to specific moods, and ultimately fostering a deeper connection with music. The effectiveness of these systems relies heavily on several factors: robust hardware to handle complex calculations, efficient algorithms to identify user preferences and music patterns, and reliable data to ensure the accuracy and timeliness of recommendations. In essence, while not critical for basic functioning, music recommendation systems play a vital role in enhancing user satisfaction and enjoyment within the music listening experience.

### **3.5.8 Safety and Security Consideration**

Music recommendation systems, while focused on user enjoyment, also need to consider safety and security. Here's a breakdown of the key concerns:



**Data Privacy:** The system likely collects user data on listening habits and preferences. Safeguarding this data is crucial, requiring robust security measures to prevent unauthorized access and ensure user privacy.

**Algorithmic Bias:** Recommendation algorithms can perpetuate biases present in the training data. This could lead to unfair or discriminatory recommendations, disadvantaging certain genres or artists. Mitigating bias is essential for promoting diversity and inclusivity within the system.

**Content Filtering:** Depending on the target audience, the system might need to implement content filtering mechanisms. This could involve filtering out explicit lyrics or genres unsuitable for certain users.

By addressing these safety and security considerations, music recommendation systems can provide a safe, enjoyable, and inclusive experience for all users.

### 3.6 ASSUMPTIONS AND DEPENDENCIES

Assumptions:

- **Availability of Data:** We assume access to a rich dataset of music information (genres, artists, attributes) and user data (listening habits, preferences). This data could come from internal user interaction or external APIs like Spotify.
- **User Cooperation:** The system relies on users providing at least some data on their searching preferences to generate recommendations.
- **Computational Resources:** We assume access to a computer system with adequate processing power, memory, and storage to handle data analysis, algorithm training, and recommendation generation.

Dependencies:

- **Functionality of APIs:** The system depends on the continued functionality and reliability of any external APIs used for data acquisition (e.g., Spotify).
- **Choice of Algorithms:** The effectiveness of the system hinges on the selection and implementation of appropriate recommendation algorithms. These algorithms will require training on the available data.
- **Hardware Performance:** The system's performance relies on the processing power and memory of the underlying hardware infrastructure.
- By carefully considering these assumptions and dependencies, we ensure a more robust foundation for developing our music recommendation system.

## 4 SYSTEM ANALYSIS

### 4.1 REQUIREMENT OF NEW SYSTEM

#### 4.1.1 User Requirements

The user requirements for our music recommendation system define what the system should do from the user's perspective. Here's a breakdown of the key functionalities:

**Discover New Music:** Users should be able to easily discover new songs and artists based on their preferences and listening habits. The system should suggest music they might enjoy but haven't heard before.

**Personalized Recommendations:** Users expect recommendations tailored to their individual tastes. This could involve filtering by genre, mood, artist preference, or activity (workouts, relaxation, etc.).

**User Control:** Users should have some control over the recommendation process. This might include the ability to provide feedback on past recommendations (like/dislike), adjust recommendation settings (preferred genres, artists), or even create custom playlists based on suggestions.

**Seamless Integration:** The music recommendation system should integrate smoothly within the user's existing music experience. This could involve suggesting songs within a music streaming service or providing recommendations based on a user's currently playing song.

**Intuitive Interface:** The system should have a user-friendly and intuitive interface that allows users to easily explore recommendations, understand why specific songs are suggested, and interact with the system without difficulty.

#### 4.1.2 System Requirements

##### R4.1 User Functions

###### R4.1.1 Login

Intro: In login user logged into system.

Input: In input there is email-id and password.

Output: If id and password is correct so redirect to home page.

Process: In process check id and password is correct or not and if not correct so give error.

###### R4.1.2 Manage Search Details

Intro: In this functionality user can Search different types of songs.

Input: Add song details or name.

Output: Display recommended song lists.  
Process: Run model and check if song available

#### R4.1.3 List of Playlists

Intro: In this functionality user can check new playlist.  
Input: User have permission to go through playlist.  
Output: Change details in recommendations.  
Process: Change details in recommendations.

#### R4.1.4 List of Songs

Intro: In this functionality user can check new songs and old songs.  
Input: User have permission to go through song list.  
Output: Change details in recommendations.  
Process: Change details in recommendations.

### R4.2 Server Function

#### R4.2.1 Authentication

Intro: In login the server-side authentication will be generated.  
Input: In input there is id and password.  
Output: If id and password is correct so redirect to home page.  
Process: In process check id and password is correct or not and if not correct so give error.

#### R4.2.2 ML Model

Intro: In this functionality server will update information or edit information using API.  
Input: Add information or edit information.  
Output: Display profile details.  
Process: Adding information and updating to database.

### 4.1.3 Non-Functional Requirements

**Security:** The system needs to log client's information of registration such as IP address and time for security purpose. Password should be encrypted and stored in the database.

**Maintainability:** The application is to be designed so that it is easily maintained. Also, it should allow incorporating new requirements in any module of system.

**Portability:** The application will be easily portable on any window-based system.

**Usability:** The system shall allow the users to access the system from the Internet using SpringBoot or its derivative technologies. The system uses a web browser as an interface. Since all users are familiar with the general usage of browsers, no specific training is required. The system is user friendly and self-explanatory

**Reliability:** The system has to be very reliable due to the importance of data and the

damages incorrect or incomplete data can do.

**Availability:** The system is available 100% for the user and is used 24 hours a day and 365 days a year. The system shall be operational 24 hours a day and 7 days a week.

**Mean Time between Failures (MTBF):** The system will be developed in such a way that it may fail once in a year.

**Mean Time to Repair (MTTR):** Even if the system fails, the system will be recovered back up within an hour or less. The accuracy of the system is limited by the accuracy of the speed at which the employees of the library and users of the library use the system.

**Access Reliability:** The system shall provide 100% access reliability.

**Validation Criteria:** A Software Testing Plan (STP) will be created to define a set of qualification methods and to specify for each requirement in this document a method of ensuring that the requirement is satisfied.

## 4.2 FEATURES OF NEW SYSTEM

### Recommendation Algorithms:

**Collaborative Filtering:** Recommend music based on similarities between users' preferences and behaviors.

**Content-Based Filtering:** Recommend music based on the attributes of songs that users have interacted with positively in the past.

**Hybrid Approaches:** Combine collaborative and content-based filtering techniques for improved recommendation accuracy.

### Cross-Platform Integration:

**Mobile Apps:** Offer mobile applications for iOS and Android platforms for on-the-go access to music recommendations.

**Web Player:** Provide a web-based player accessible from desktop browsers for seamless listening across devices.

**Smart Speaker Integration:** Integrate with voice-controlled smart speakers (e.g., Amazon Echo, Google Home) for hands-free music playback.

### Data Privacy and Security:

**Privacy Controls:** Offer users granular control over their data, including options to opt-out of data collection or delete personal information.

**Secure Authentication:** Implement robust authentication mechanisms to protect user accounts and prevent unauthorized access.

### 4.3 NAVIGATION CHART

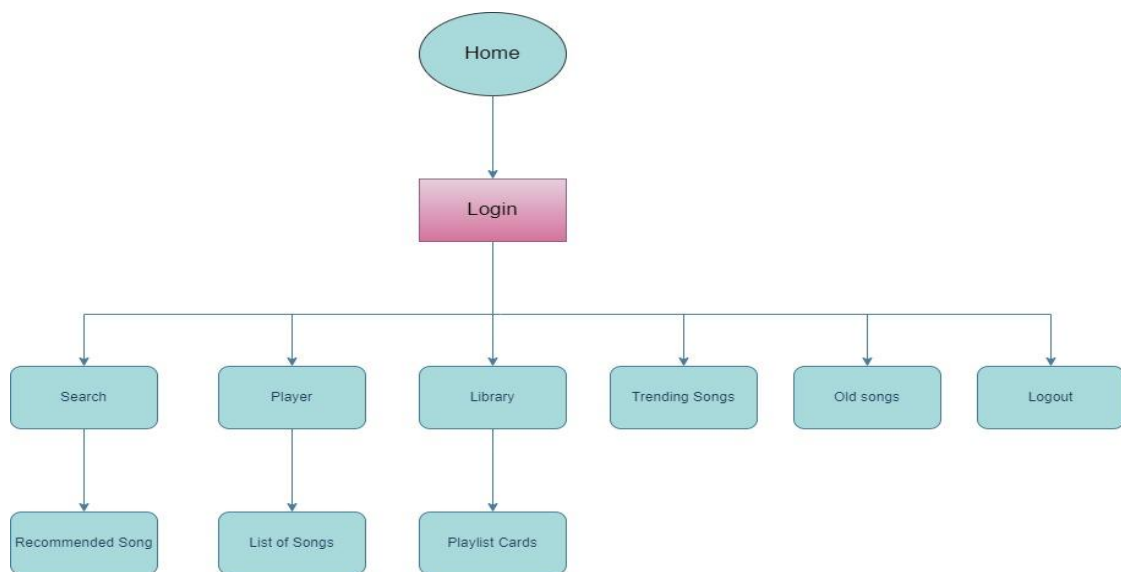


Figure 4.1 NAVIGATION CHART

### 4.4 SYSTEM ACTIVITY

The system activity of a music recommendation system encompasses the various processes and interactions that occur within the system to provide personalized music recommendations to users. Here's an overview of the system activity in a typical music recommendation system:

#### Data Collection:

**User Interaction Logging:** The system collects data on user interactions with the platform, including song plays, likes, skips, searches, and user profiles.

**Music Metadata Acquisition:** Retrieves metadata about songs, albums, artists, genres, and other relevant information from external sources such as music databases or APIs (e.g., Spotify, Last.fm).

**Social Media Integration:** Optionally, the system may integrate with social media platforms to gather additional user preferences or music-related activities.

#### Data Processing:

**Feature Extraction:** Extracts relevant features from the collected data, such as user preferences, song characteristics (e.g., genre, tempo, mood), and contextual information (e.g., time of day, location).

**Data Cleaning and Transformation:** Preprocesses and cleans the collected data to handle missing values, outliers, and inconsistencies.

**Data Integration:** Integrates various data sources and formats to create a unified dataset for analysis.

#### Machine Learning Model Training:

**Model Selection:** Chooses appropriate machine learning algorithms or models for recommendation generation, such as collaborative filtering, content-based filtering, or hybrid approaches.

**Training Data Preparation:** Prepares training data by splitting datasets into training, validation, and testing sets.

**Model Training:** Trains machine learning models using historical user interactions and music features to learn patterns and preferences.

**Model Evaluation:** Evaluates the trained models' performance using metrics like accuracy, precision, recall, and ranking metrics.

### **Recommendation Generation:**

**Personalized Recommendation Generation:** Generates personalized music recommendations for users based on their historical interactions, preferences, and contextual information.

**Diversification and Serendipity:** Optionally, incorporates techniques to diversify recommendations and introduce serendipity to expose users to new and relevant music outside their usual preferences.

**Real-Time Recommendations:** Provides real-time or near-real-time recommendations based on user interactions and dynamically changing preferences.

### **Feedback and Adaptation:**

**Feedback Collection:** Collects user feedback on recommended songs (e.g., thumbs up/down, ratings, explicit feedback).

**Model Adaptation:** Incorporates user feedback to update and improve recommendation models through techniques like collaborative filtering updates, retraining, or online learning.

### **Delivery and Presentation:**

**Presentation Layer:** Renders and delivers recommended music content to users through various interfaces, such as web applications, mobile apps, or smart devices.

**User Interface Personalization:** Personalizes the user interface to enhance user experience, showcasing recommendations prominently and providing intuitive navigation options.

### **Monitoring and Maintenance:**

**System Monitoring:** Monitors system performance, availability, and resource usage to ensure smooth operation.

**Error Handling and Logging:** Handles errors gracefully and logs relevant information for troubleshooting and debugging.

**Regular Maintenance:** Conducts regular maintenance tasks such as database cleanup, model retraining, and software updates to ensure system reliability and performance.

By understanding and analyzing these system activities, developers and stakeholders can optimize and enhance the performance of the music recommendation system, providing users with high-quality, personalized music recommendations.

## 4.5 DATA MODELING

### 4.5.1 Data Dictionary

Table 4.1 DATA DICTIONARY

Name	Email	Password	OTP

### 4.5.2 ER Diagram

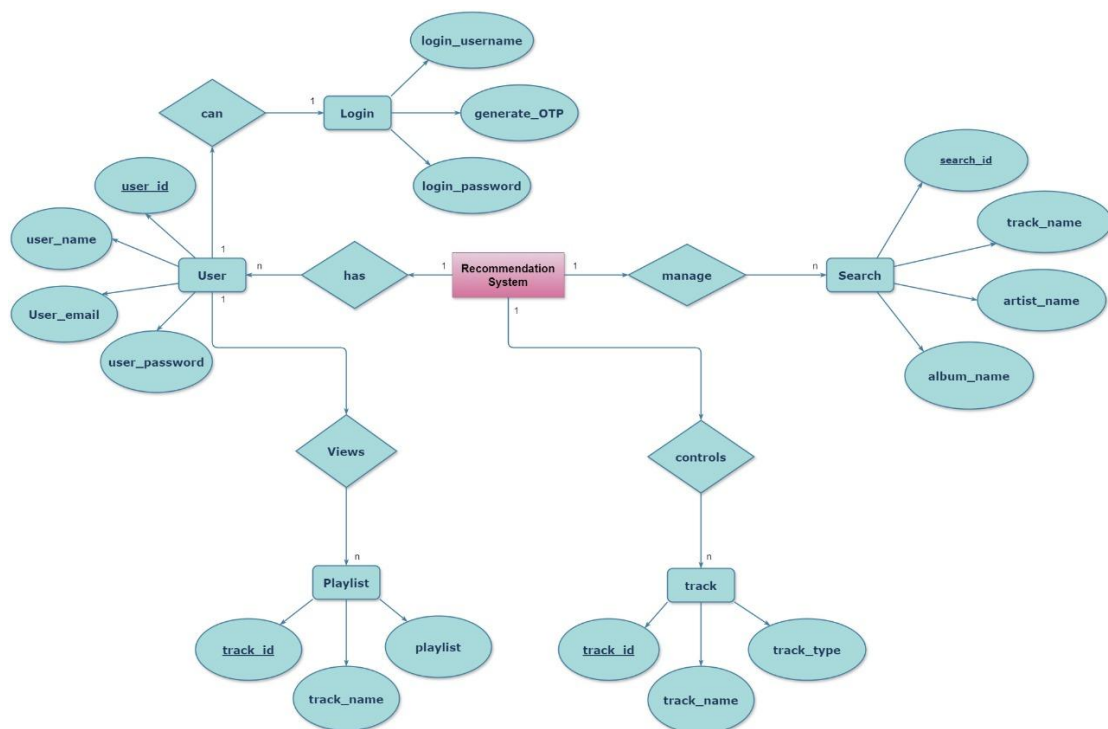


Figure 4.2 ER Diagram

## 5 SYSTEM DESIGN

### 5.1 SYSTEM ARCHITECTURE DESIGN

#### 5.1.1 Class Diagram

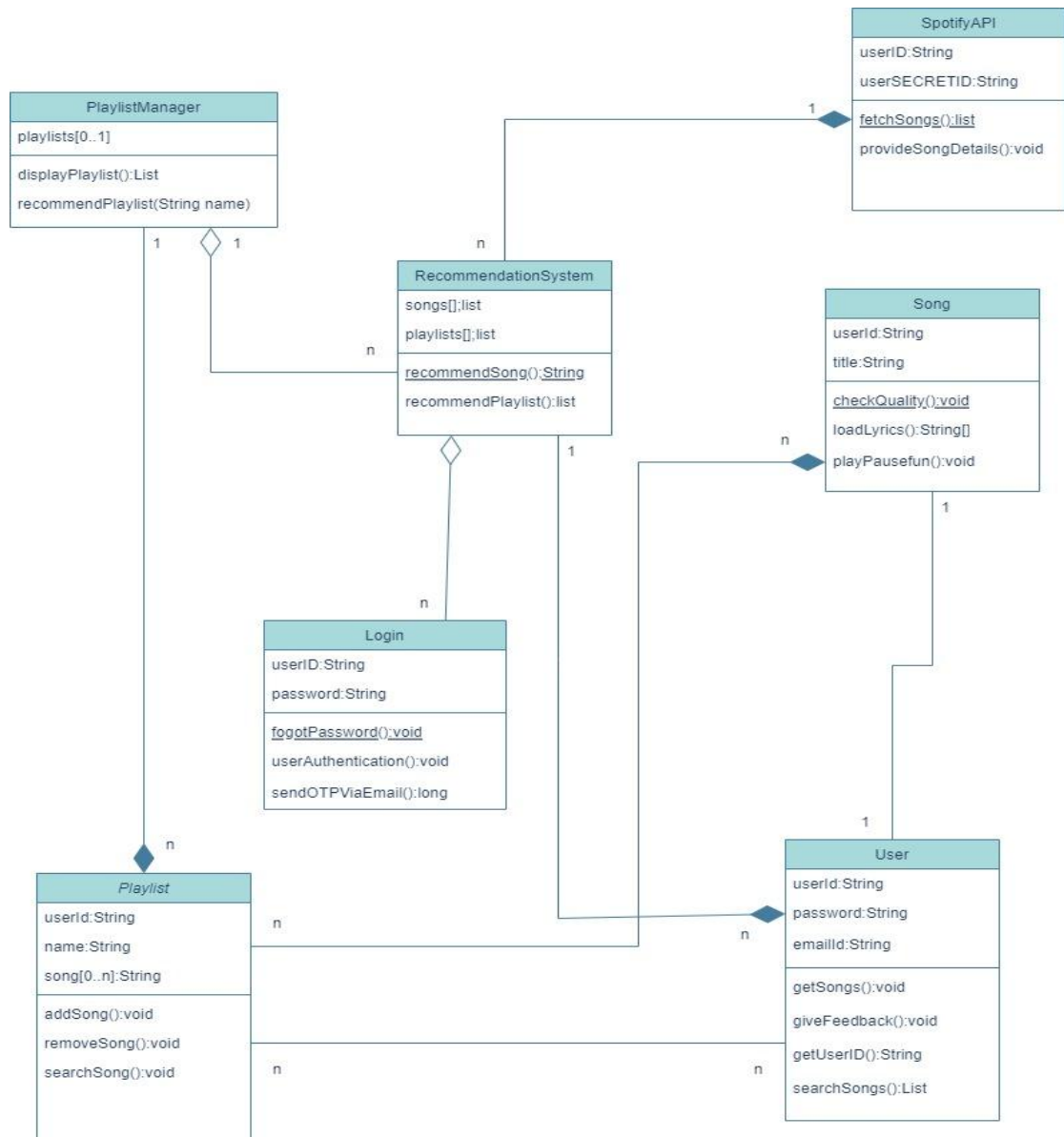


Figure 5.1 CLASS DIAGRAM



### 5.1.2 Sequence Diagram

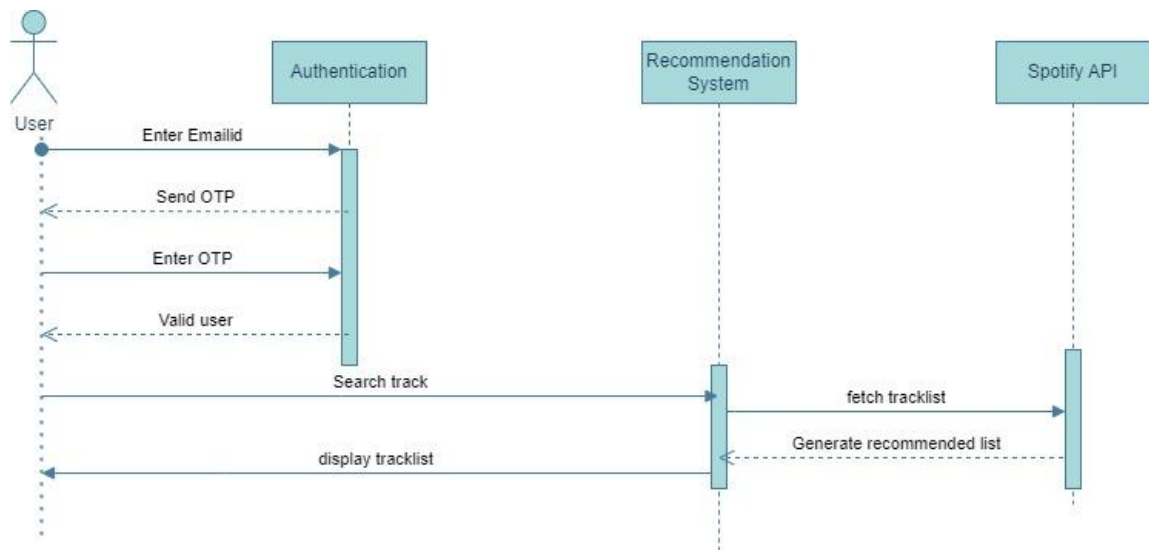


Figure 5.2 SEQUENCE DIAGRAM

### 5.1.3 Component Diagram

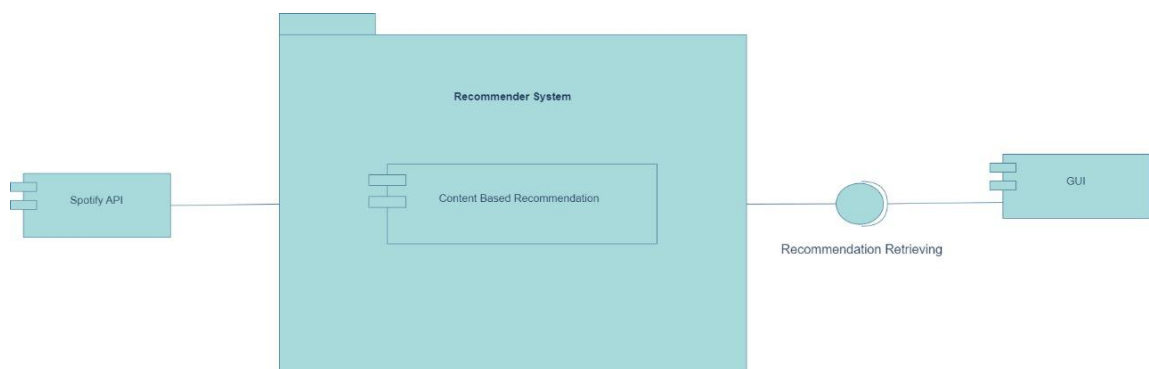


Figure 5.3 COMPONENT DIAGRAM

### 5.1.4 Deployment Diagram

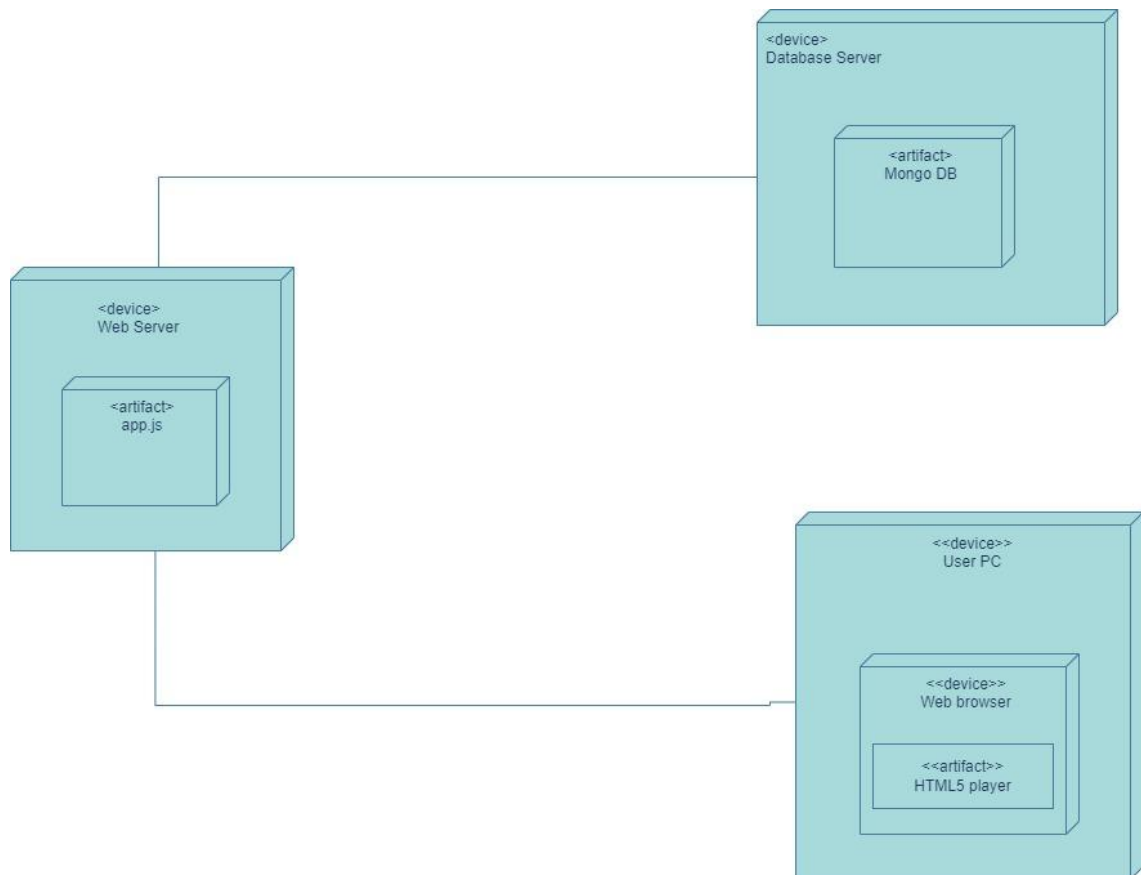


Figure 5.4 DEPLOYMENT DIAGRAM

## 5.2 USECASE DIAGRAM

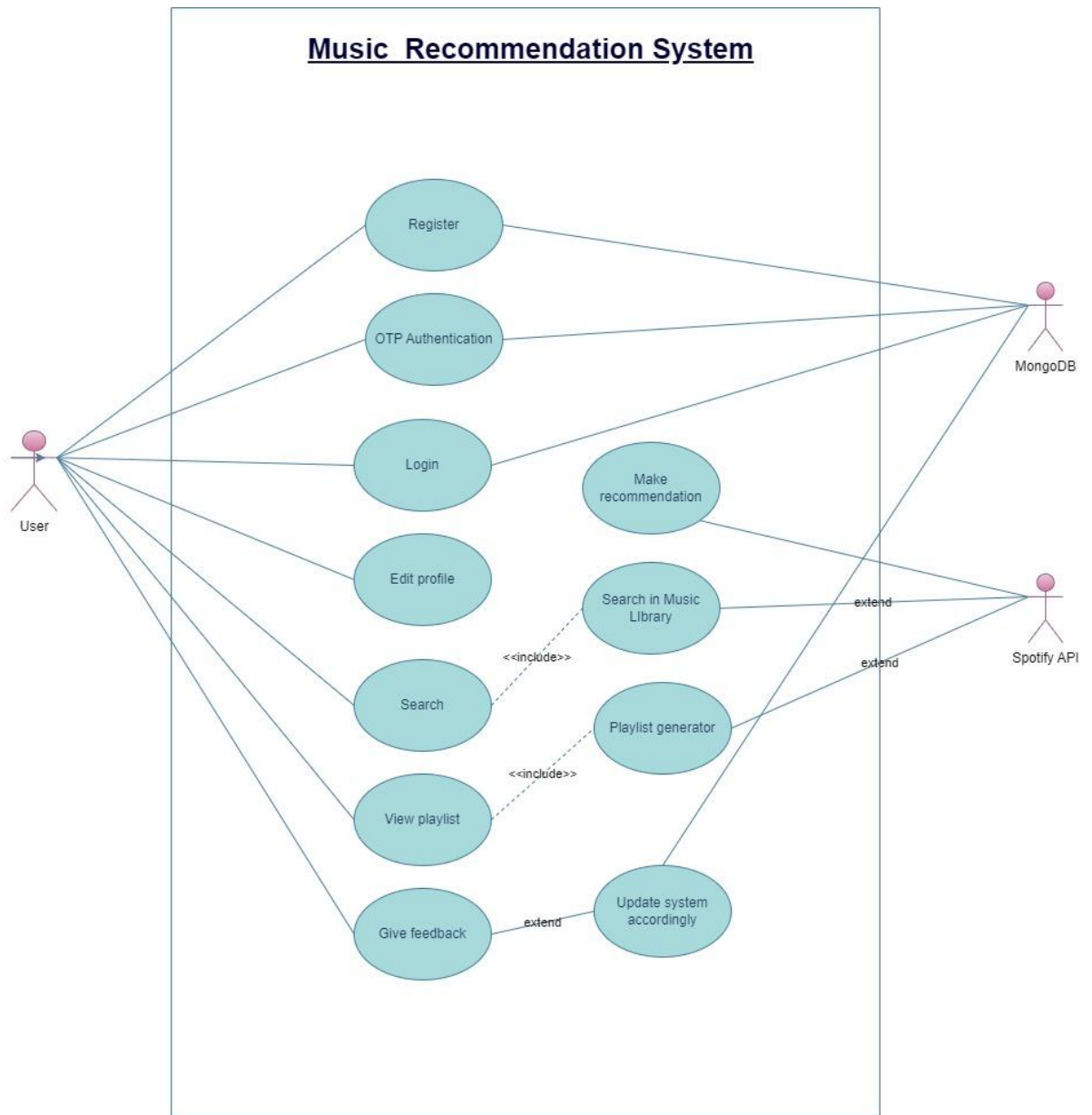


Figure 5.5 USECASE DIAGRAM

### 5.3 ACTIVITY DIAGRAM

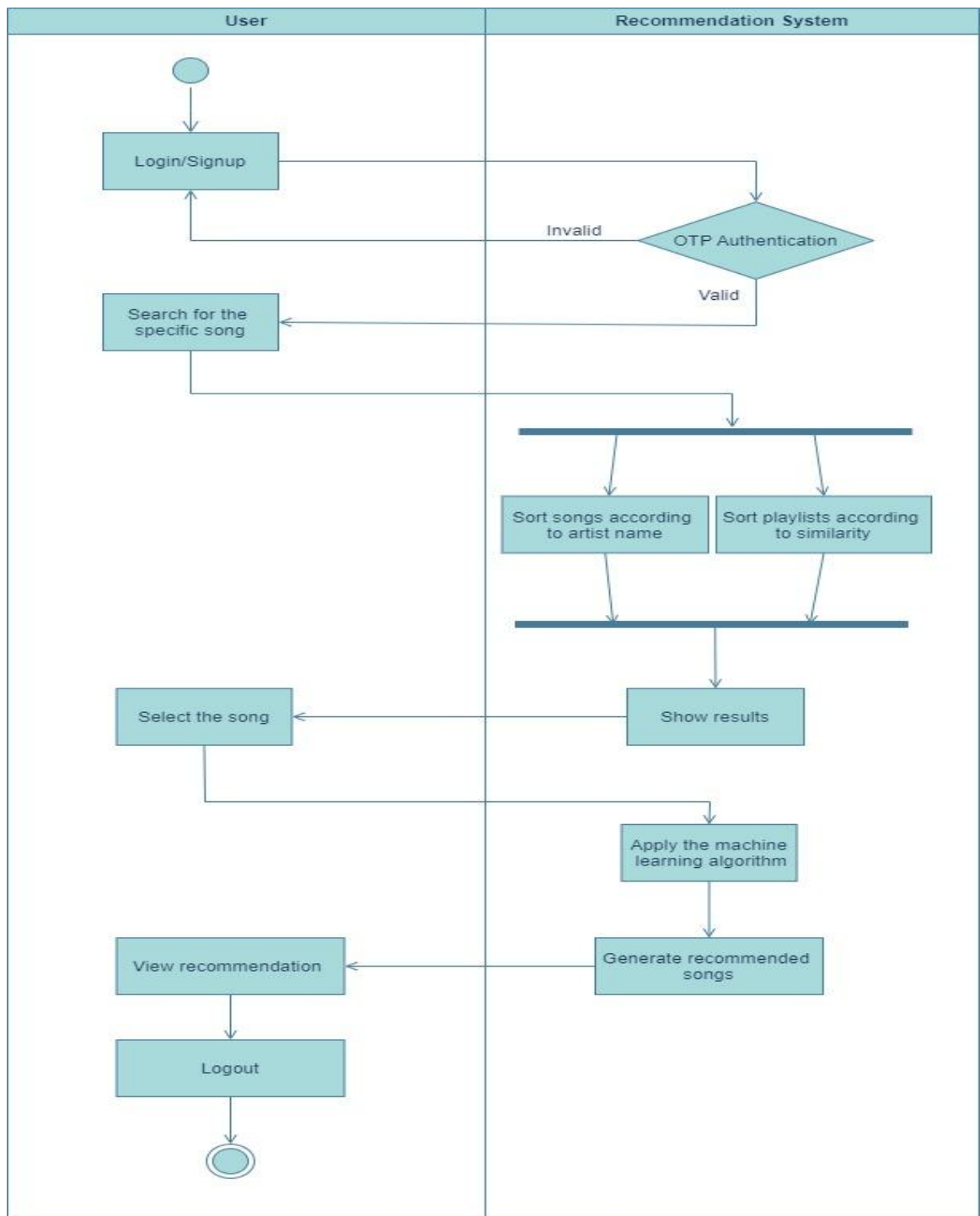


Figure 5.6 ACTIVITY DIAGRAM

## 5.4 DATA FLOW DIAGRAM

### 5.4.1 Level-0 (Context level)

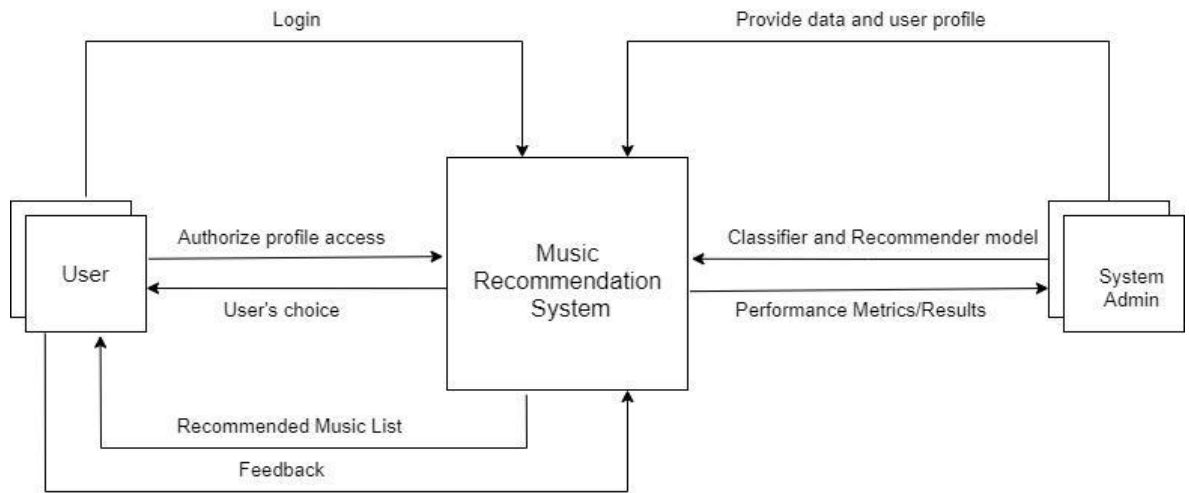


Figure 5.7 DFD LEVEL-0

### 5.4.2 Level-1 (Context level)

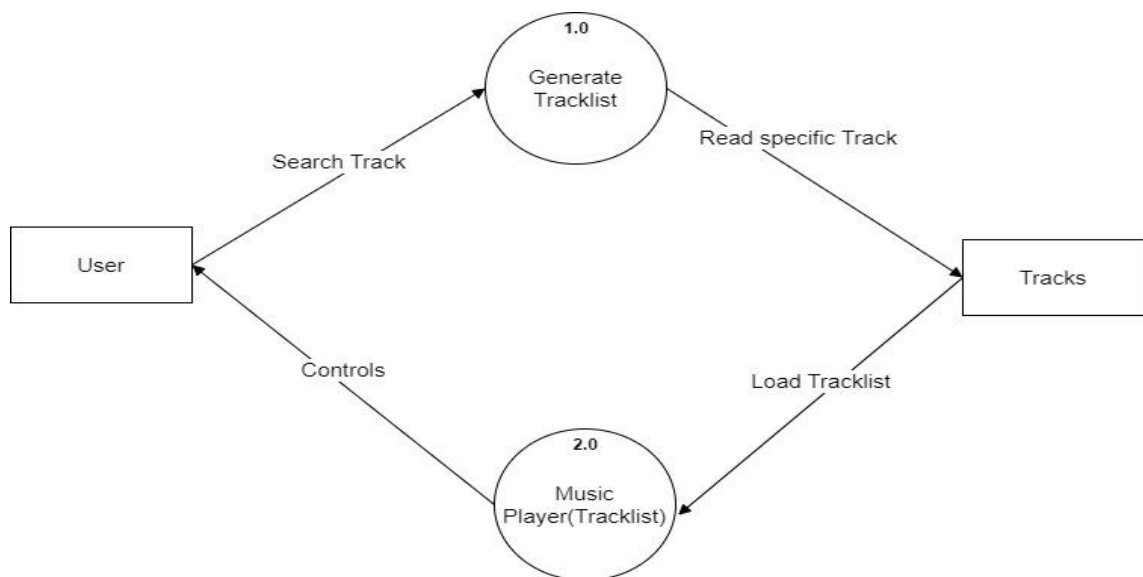


Figure 5.8 DFD LEVEL-1

### 5.4.3 Level-2 (Context level)

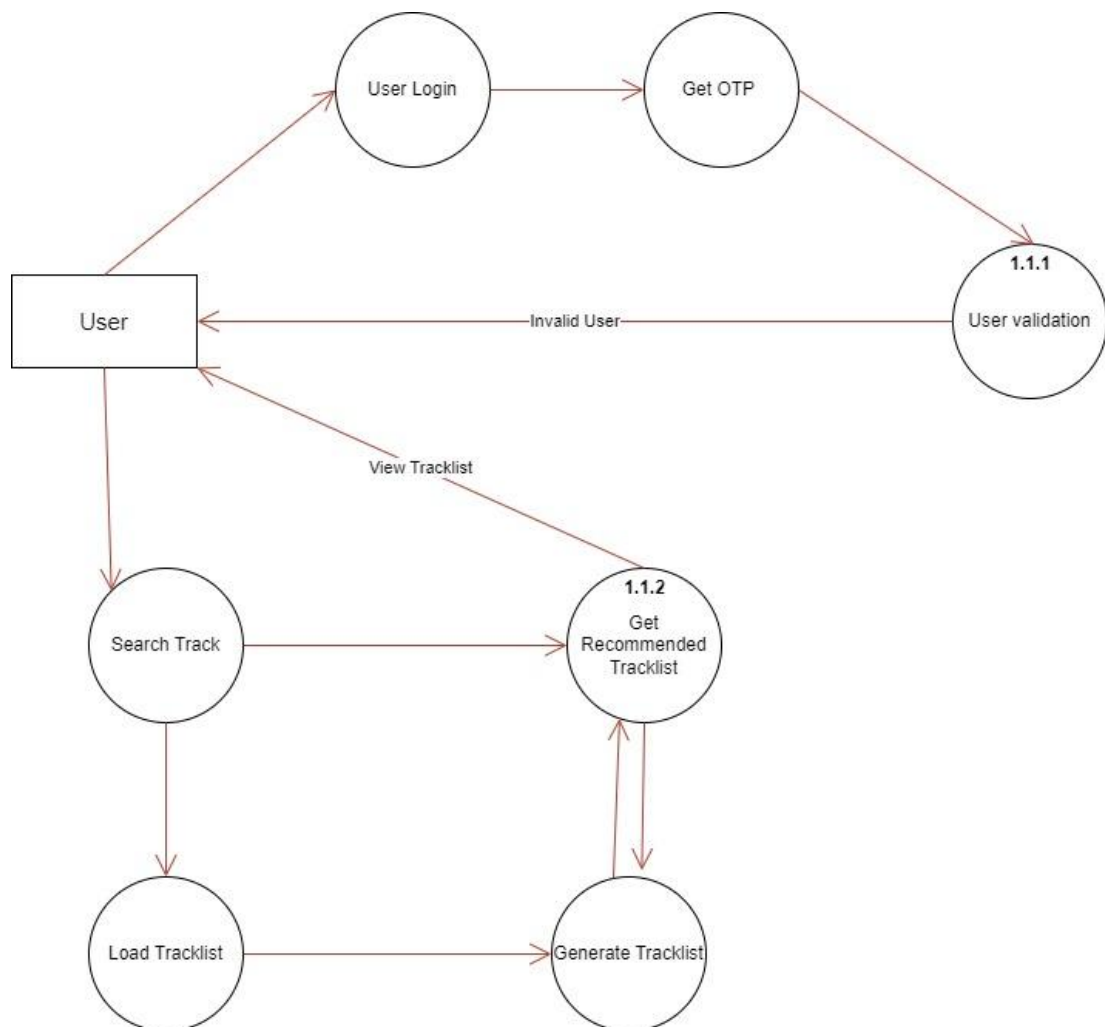


Figure 5.9 DFD LEVEL-2

## 5.5 STATECHART DIAGRAM

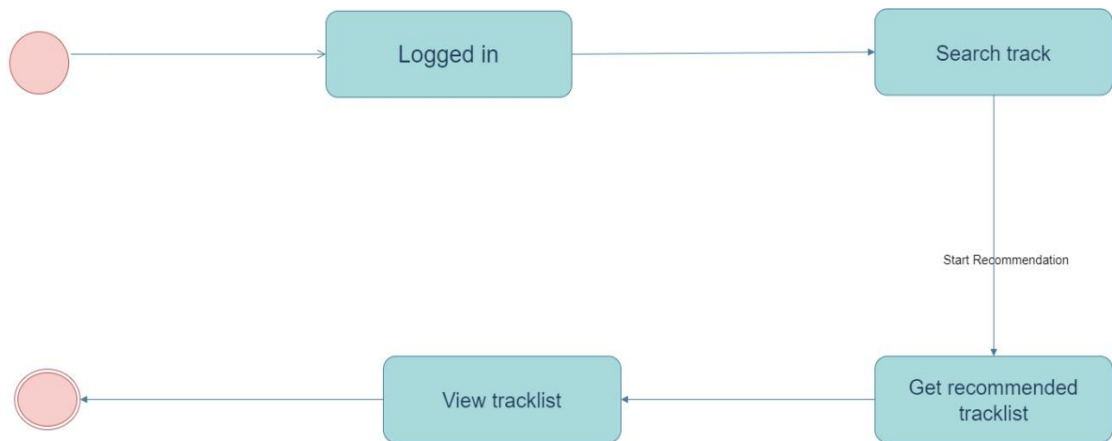


Figure 5.10 STATECHART DIAGRAM

## **6 IMPLEMENTATION PLANNING**

### **6.1 IMPLEMENTATION ENVIRONMENT**

During the complete implementation, we've worked on Visual Studio Code which is a free source for GUI Editor. Features included support for debugging syntax highlighting intelligent code completion and code refactoring. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality. It also provides extension to work in a team with a multi user experience. Even compiling and deploying of the application was done easily with the help of VS Code.

For the data base implementation, we've used MONGO DB. MONGO DB is powerful, open-source data base system. For the background tasks implementation, we've used Jupyter Notebook which is a web-based application that acts as an interactive environment for developing code, data analysis, and data visualization.

### **6.2 PROGRAM/MODULES SPECIFICATION**

In our entire system we have 5 modules.

#### **1. User Management Module:**

Responsible for user registration, authentication, and profile management. Includes functionalities for user sign-up, login, password recovery, and account setting.

#### **2. Security Module:**

Implements security measures to protect user data, including encryption of sensitive information, secure authentication mechanisms, and access controls. Ensures compliance with data privacy regulations and industry best practices for information security.

#### **3. Music Retrieval Module:**

Interfaces with the Spotify API to fetch music-related data, including tracks, artists, albums, and genres.

Implements mechanisms for searching and browsing music content based on user queries or preferences.

#### **4. Recommendation Engine Module:**

Employs recommendation algorithms to generate personalized music recommendations for users.

Analyzes user listening history, preferences, and behavior to deliver relevant suggestions.



### 5.Payment Module:

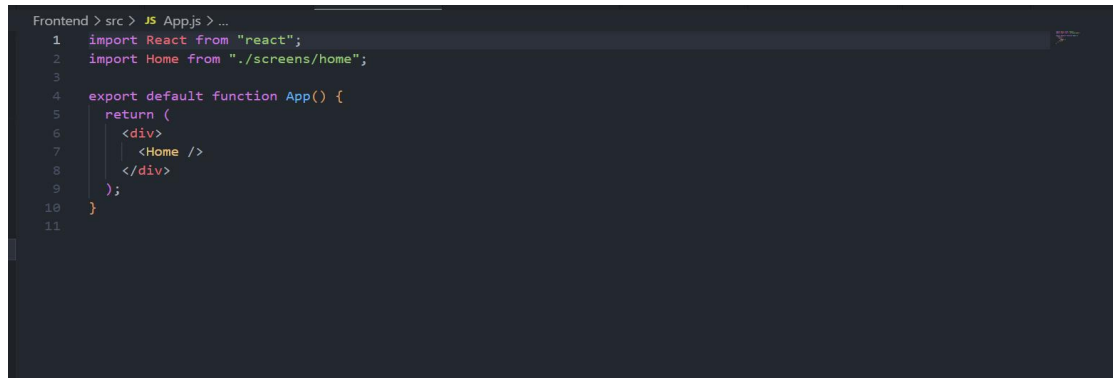
This module will give you enhanced features if you subscribe to the application.

## 6.3 CODING STANDARDS

While writing our code we took the utmost care to follow the basic coding standards while writing a code like,

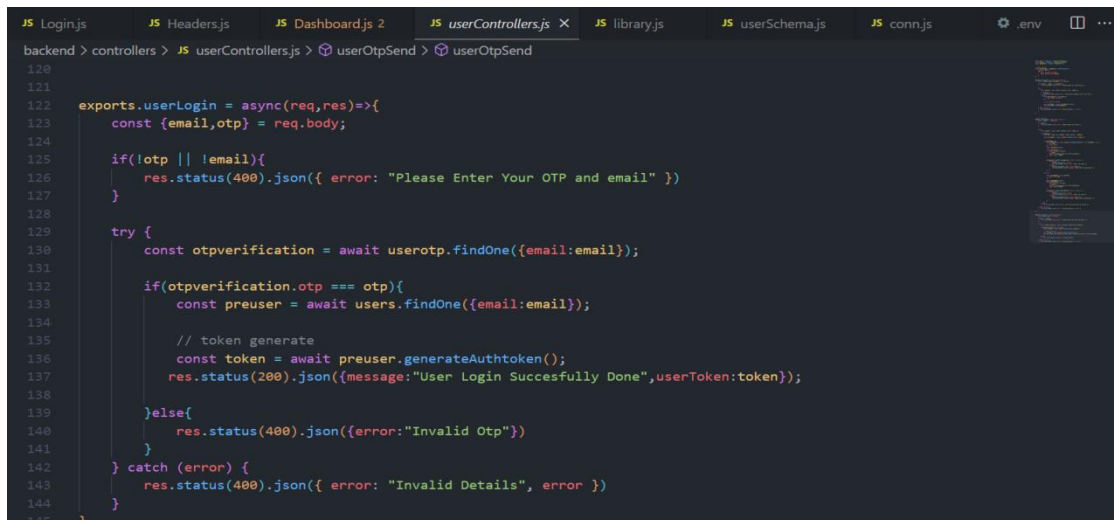
- Doing proper indentation.
- Following proper indentation.
- Proper error handling.
- Adding comments for better understanding.
- Proper Syntax

Here are some of the code snippets written during implementation,



```
Frontend > src > JS App.js > ...  
1  import React from "react";  
2  import Home from "./screens/home";  
3  
4  export default function App() {  
5    return (  
6      <div>  
7        <Home />  
8      </div>  
9    );  
10 }  
11
```

Figure 6.1 App.js



```

120
121
122 exports.userLogin = async(req,res)=>{
123     const {email,otp} = req.body;
124
125     if(!otp || !email){
126         res.status(400).json({ error: "Please Enter Your OTP and email" })
127     }
128
129     try {
130         const otpverification = await userotp.findOne({email:email});
131
132         if(otpverification.otp === otp){
133             const preuser = await users.findOne({email:email});
134
135             // token generate
136             const token = await preuser.generateAuthtoken();
137             res.status(200).json({message:"User Login Succesfully Done",userToken:token});
138
139         }else{
140             res.status(400).json({error:"Invalid Otp"})
141         }
142     } catch (error) {
143         res.status(400).json({ error: "Invalid Details", error })
144     }
145 }

```

Figure 6.2 userController.js

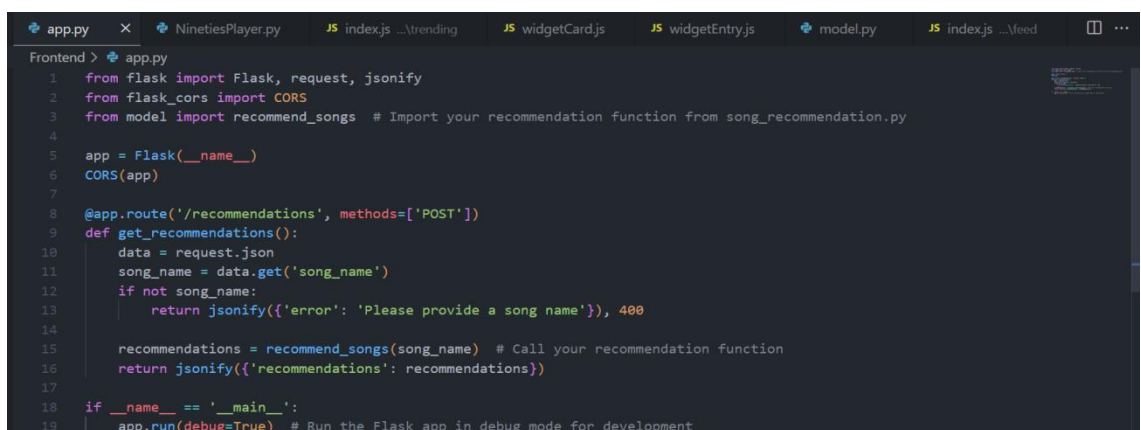


```

16 exports.userregister = async (req, res) => {
17     const { fname, email, password } = req.body;
18
19     if (!fname || !email || !password) {
20         res.status(400).json({ error: "Please Enter All Input Data" })
21     }
22
23     try {
24         const presuer = await users.findOne({ email: email });
25
26         if (presuer) {
27             res.status(400).json({ error: "This User Allready exist in our db" })
28         } else {
29             const userregister = new users({
30                 fname, email, password
31             });
32
33             // here password hasing
34
35             const storeData = await userregister.save();
36             res.status(200).json(storeData);
37         }
38     } catch (error) {
39         res.status(400).json({ error: "Invalid Details", error })
40     }
41 }

```

Figure 6.3 userAuthenticate.js

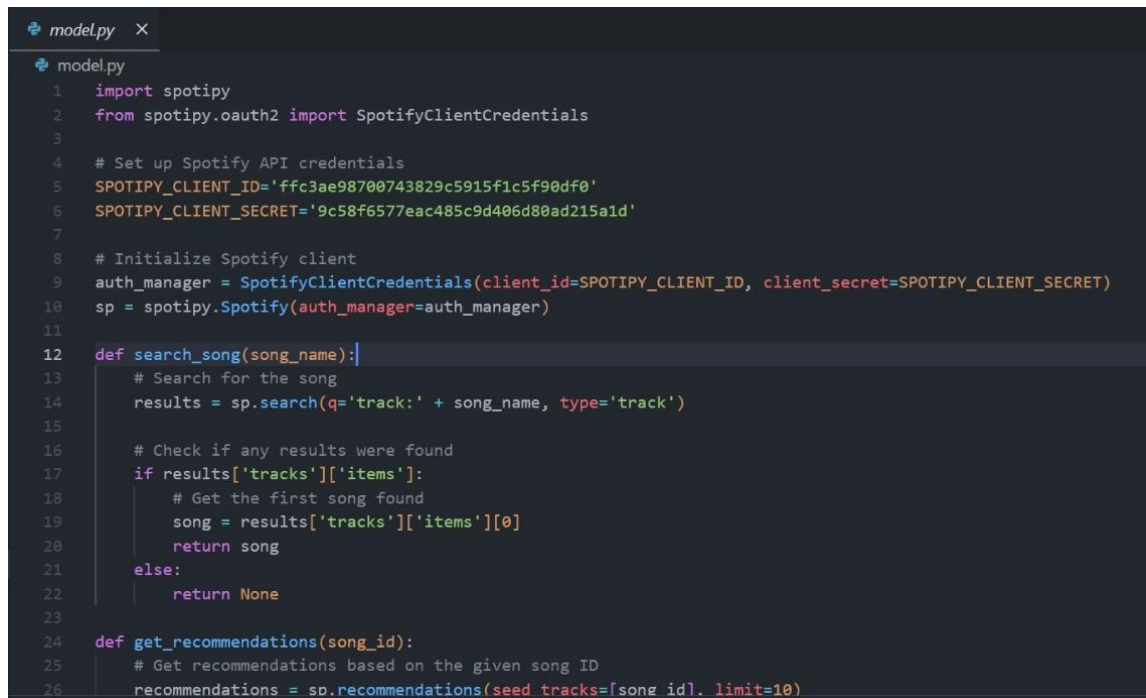


```

1 from flask import Flask, request, jsonify
2 from flask_cors import CORS
3 from model import recommend_songs # Import your recommendation function from song_recommendation.py
4
5 app = Flask(__name__)
6 CORS(app)
7
8 @app.route('/recommendations', methods=['POST'])
9 def get_recommendations():
10     data = request.json
11     song_name = data.get('song_name')
12     if not song_name:
13         return jsonify({'error': 'Please provide a song name'}), 400
14
15     recommendations = recommend_songs(song_name) # Call your recommendation function
16     return jsonify({'recommendations': recommendations})
17
18 if __name__ == '__main__':
19     app.run(debug=True) # Run the Flask app in debug mode for development

```

Figure 6.4 App.py



```
model.py x
model.py
1 import spotipy
2 from spotipy.oauth2 import SpotifyClientCredentials
3
4 # Set up Spotify API credentials
5 SPOTIPY_CLIENT_ID='ffc3ae98700743829c5915f1c5f90df0'
6 SPOTIPY_CLIENT_SECRET='9c58f6577eac485c9d406d80ad215a1d'
7
8 # Initialize Spotify client
9 auth_manager = SpotifyClientCredentials(client_id=SPOTIPY_CLIENT_ID, client_secret=SPOTIPY_CLIENT_SECRET)
10 sp = spotipy.Spotify(auth_manager=auth_manager)
11
12 def search_song(song_name):
13     # Search for the song
14     results = sp.search(q='track:' + song_name, type='track')
15
16     # Check if any results were found
17     if results['tracks']['items']:
18         # Get the first song found
19         song = results['tracks']['items'][0]
20         return song
21     else:
22         return None
23
24 def get_recommendations(song_id):
25     # Get recommendations based on the given song ID
26     recommendations = sp.recommendations(seed_tracks=[song_id], limit=10)
```

*Figure 6.5 Model.py*

## 7 TESTING

### 7.1 TESTING PLAN

To guarantee a high-performing and user-friendly music recommendation system, a well-defined testing plan is essential. This plan can be broken down into four stages:

**Unit Testing:** Here, the focus is on individual components. We meticulously test each module and function in isolation to verify they perform as intended. This helps identify and squash bugs within these building blocks. Imagine testing data retrieval functions to ensure they grab information from APIs or user interactions correctly.

**Integration Testing:** Once individual modules are solid, we assess how they work together as a system. This stage ensures smooth data flow between components and identifies any hiccups in communication. For instance, we'd test how the data acquisition module feeds information into the preprocessing module for seamless compatibility.

**System Testing:** Now, we evaluate the entire system in a simulated real-world environment. This stage assesses the system's ability to generate personalized and accurate recommendations for various users and scenarios. We'd simulate user interactions (if applicable) to see if recommendations are relevant and test performance under load (multiple users) to ensure responsiveness and scalability. User testing sessions can also be conducted to gather feedback on the system's interface, recommendation quality, and overall user experience.

Throughout testing, relevant metrics like precision, recall, recommendation accuracy, and user satisfaction (gathered through surveys) are crucial for quantifying the system's performance. By following this comprehensive testing plan, we can ensure our music recommendation system delivers exceptional value to users, keeping them engaged and excited about discovering new music.

### 7.2 TESTING STRATEGY

The development process repeats this testing sub-process a number of times for the following phases.

**Manual Testing:** Put human testers in the driver's seat. They'll interact with the system, evaluating usability, performance, and overall functionality. Manual testing is ideal for aspects like user interface design and user experience.

**Automated Testing:** Let the machines do the legwork! Automated scripts or tools can execute tests repeatedly and efficiently. This is perfect for unit testing, integration testing, and regression testing (re-running tests after code changes).

**Performance Testing:** Can your system handle the music? Test how it performs under various loads (many users, large datasets). Ensure it remains responsive and generates recommendations efficiently.

**Exploratory Testing:** Think outside the box! This strategy involves open-ended exploration to uncover unexpected behaviors or scenarios not covered in traditional test plans. It can help identify usability issues or edge cases that might be missed otherwise.

**Integration Testing:** Integration Testing tests whether the various programs that make up a system, interface with each other as desired, fit together and whether the interfaces between the programs are correct.

## 7.3 TESTING METHODS

There are two kinds of testing mainly Blackbox and Whitebox testing:

- **Black-Box (User-Centric):** Imagine yourself as the user. Test the system's functionality from their perspective. Does it take the right inputs (user data, songs) and deliver the expected outputs (relevant recommendations)? This ensures a user-friendly experience.
- **White-Box (Code-Centric):** Dive into the code. Test individual modules, functions, and algorithms to verify they operate correctly based on their design. This helps identify bugs within the system's inner workings.

## 7.4 TEST CASES

Table 7.1 TEST CASES

ENTITY	TEST CASE	EXPECTED OUTPUT	ACTUAL OUTPUT	STATUS
Sign Up	Validation	Redirect to Login Page	Redirect to Login Page	Pass
Login	Validation	Redirect to OTP Page	Redirected to OTP Page	Pass
OTP	Verification	OTP sent to user Email	OTP sent to user Email	Pass
Search Songs	Searching	Update to screen and data base	Updated to screen and in data base	Pass
View Song List	Update song	Updating details	Updating details in GUI	Pass

## 8 USER MANUAL

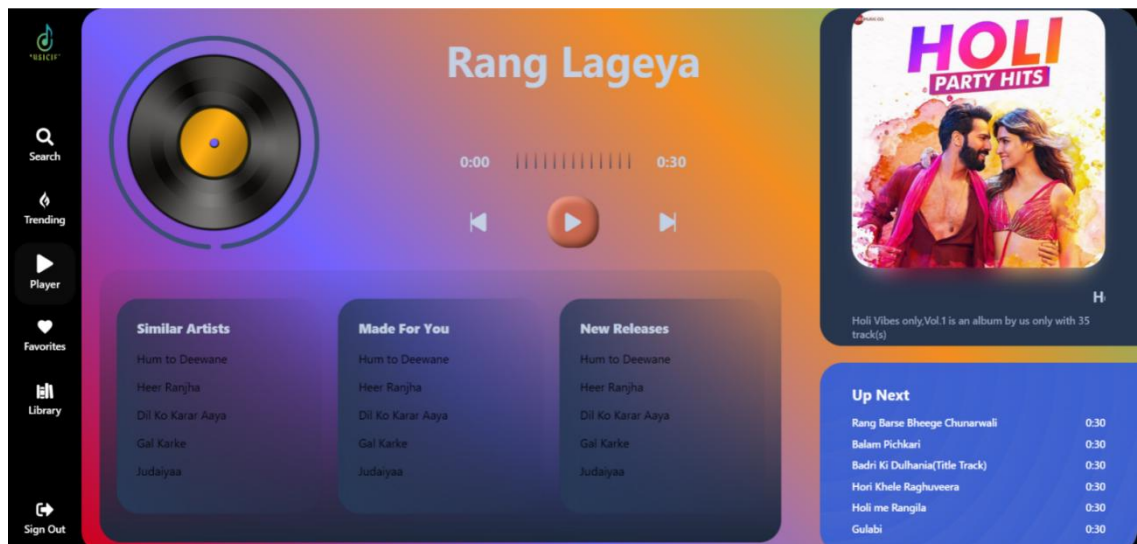
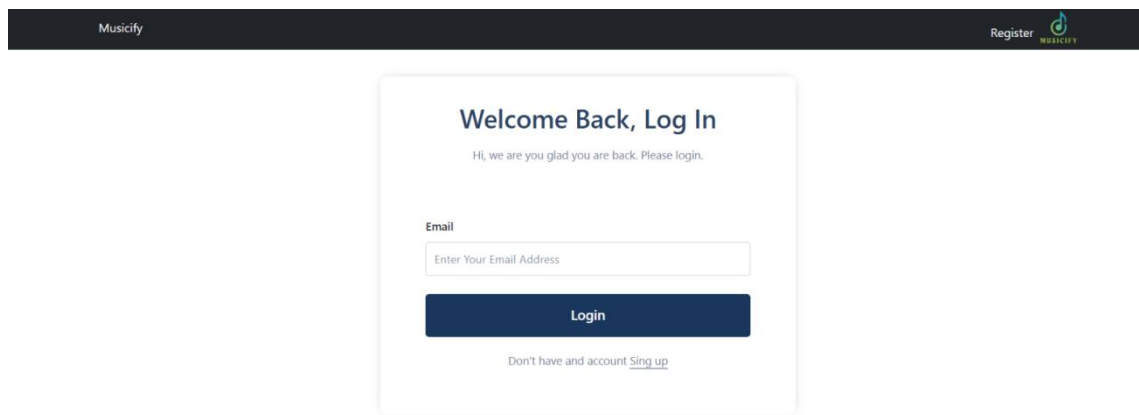


Figure 8.1 HOME PAGE

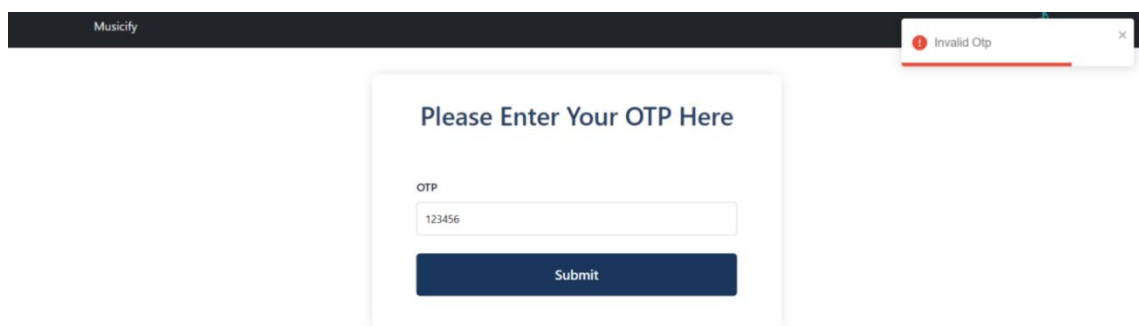
The screenshot shows the Musicify app's sign-up page. At the top, it says 'Sign Up' and includes a welcome message: 'We are glad that you will be using Project Cloud to manage your tasks! We hope that you will get like it.' Below this are three input fields: 'Name' (placeholder: 'Enter Your Name'), 'Email' (placeholder: 'Enter Your Email Address'), and 'Password' (placeholder: 'Enter Your password' with a 'Show' button). A dark blue 'Sign Up' button is at the bottom, followed by a link that says 'Don't have an account'.

Figure 8.2 SIGN UP PAGE



The screenshot shows the Musicify login page. At the top, there is a dark header bar with the word "Musicify" on the left and a "Register" link with a logo on the right. The main content area is a white box with a light gray border. Inside, the heading "Welcome Back, Log In" is centered, followed by the text "Hi, we are glad you are back. Please login." Below this is a form with a label "Email" and a text input field containing the placeholder "Enter Your Email Address". A dark blue "Login" button is positioned below the input field. At the bottom of the form, there is a link that says "Don't have an account [Sing up](#)".

Figure 8.3 LOGIN PAGE



The screenshot shows the Musicify login page after an OTP verification attempt. The main content area is a white box with a light gray border. The heading "Please Enter Your OTP Here" is centered. Below it is a form with a label "OTP" and a text input field containing the value "123456". A dark blue "Submit" button is positioned below the input field. In the top right corner, there is a red error message box that says "Invalid Otp" with a close button (X).

Figure 8.4 LOGIN FAIL



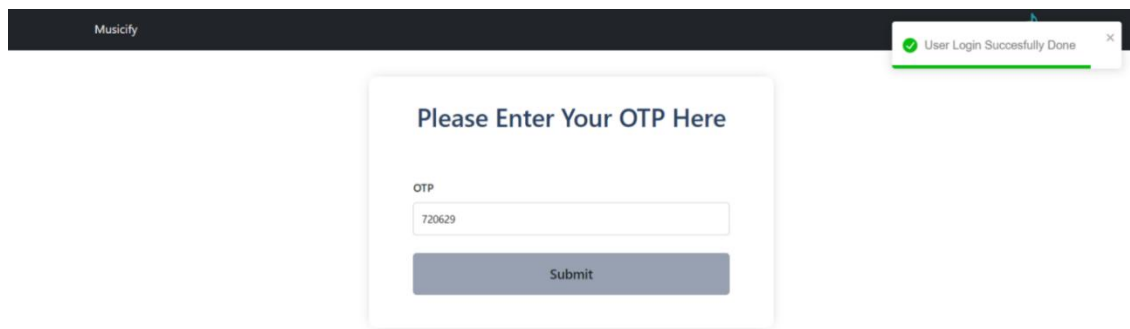


Figure 8.5 LOGIN SUCCESSFUL

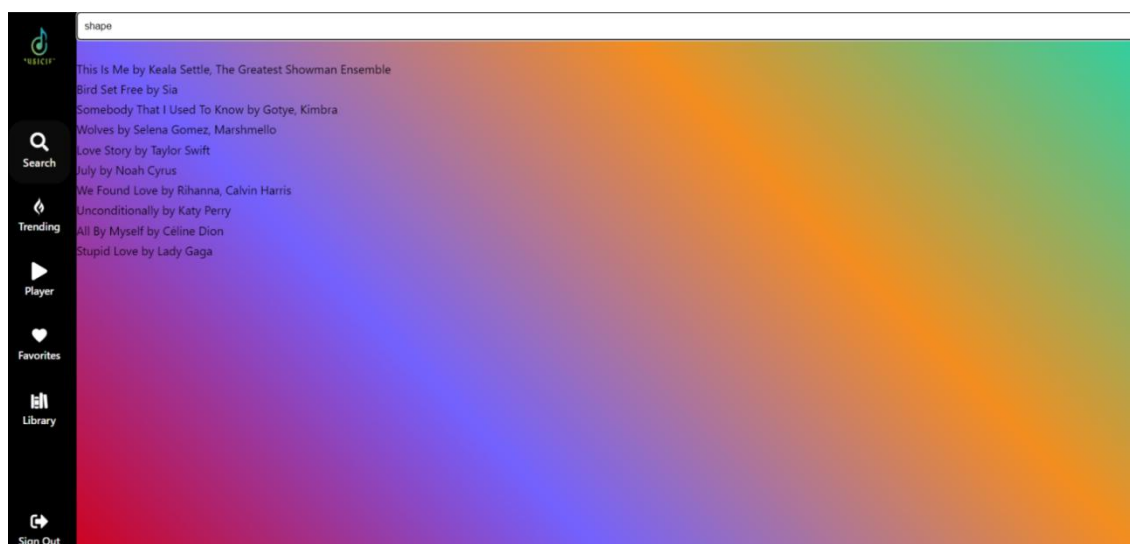


Figure 8.6 RECOMMENDATION PAGE

## **9 LIMITATIONS AND FUTURE ENHANCEMENT**

### **9.1 LIMITATIONS**

- Currently our GUI is not good.
- Our system is not compatible to Run and Load this KNN based classification model.
- Our system can't play the songs.
- Podcasts, story tell are not included yet.

### **9.2 FUTURE ENHANCEMENTS**

- We will improve the efficiency of the website.
- We will keep on the changing on UI-Design to make the application more attractive.
- Add podcasts, story feature
- we will try to add play, pause feature
- Also, we can make research paper on this Project.
- Increase speed of embedding and compressing.
- We can implement notification sending system for new updates and trending song

## **10 CONCLUSION AND DISCUSSION**

### **10.1 CONCLUSION**

- Software is easy to use but it requires user is pre-knowledge of system and refers the user manual.
- We provide ease to get recommendation
- We have tried to keep the UI alluring and easy enough for our users to feel comfortable
- We used API to fetch data so no more complexity to handle vast

### **10.2 DISCUSSION**

#### **10.2.1 Self-Analysis of Project Viabilities**

The viability of a music recommendation system hinges on its ability to accurately understand and anticipate user preferences, thereby enhancing user engagement and satisfaction. By leveraging advanced machine learning algorithms and data analytics, such a system can continuously adapt to evolving user tastes and preferences, offering personalized recommendations tailored to individual preferences and contexts. Furthermore, the system's integration across multiple platforms and devices, coupled with robust privacy controls, ensures widespread accessibility while safeguarding user data. Ultimately, a well-executed music recommendation system holds significant potential for fostering user loyalty, driving user retention, and generating revenue through targeted advertising and premium subscription models.

#### **10.2.2 Problems Encountered and Possible Solutions**

There were so many problems encountered during this project: -

1. Problem to maintain databases and change them often.
2. Need to change some functionality fully which leads to doing the whole work again.

Like, fetch data from datasets – In these algorithms, it was very time consumption method and complex to run on every system

3. Many instances where the errors weren't understandable.
4. Problem to integrate some functionalities.

### **10.2.3 Summary of Project Work**

We have completed our project work using software engineering, system analysis and design approach. We have done our work with planned scheduling pertaining to time constraints and result oriented progress in project development. We have been given an industrial exposure which is very beneficial for our field. This project will be able to recommend the music and can be further modified to music recommendation system. As it is platform independent, can be run in all environments.

## REFERENCES:

- [1] “Welcome to Flask — Flask Documentation (3.0.x).” Accessed: Apr. 03, 2024. [Online]. Available: <https://flask.palletsprojects.com/en/3.0.x/>
- [2] “Home | Spotify for Developers.” Accessed: Apr. 03, 2024. [Online]. Available: <https://developer.spotify.com/>
- [3] “MongoDB Compass | MongoDB.” Accessed: Apr. 03, 2024. [Online]. Available: <https://www.mongodb.com/products/tools/compass>
- [4] “Frontiers | Deep Learning in Music Recommendation Systems.” Accessed: Apr. 03, 2024. [Online]. Available: <https://www.frontiersin.org/articles/10.3389/fams.2019.00044/full?ref=https://githubhelp.com>
- [5] “Music recommendation system based on user’s sentiments extracted from social networks | IEEE Journals & Magazine | IEEE Xplore.” Accessed: Apr. 03, 2024. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/7298296>
- [6] “A content-based literature recommendation system for datasets to improve data reusability – A case study on Gene Expression Omnibus (GEO) datasets - ScienceDirect.” Accessed: Apr. 03, 2024. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1532046420300277>
- [7] A. Katoch and D. [Guided by Gupta, “Music Recommendation System,” 2022, Accessed: Apr. 03, 2024. [Online]. Available: <http://ir.juit.ac.in:8080/jspui/jspui/handle/123456789/8276>
- [8] A. van den Oord, S. Dieleman, and B. Schrauwen, “Deep content-based music recommendation,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2013. Accessed: Apr. 03, 2024. [Online]. Available: <https://proceedings.neurips.cc/paper/2013/hash/b3ba8f1bee1238a2f37603d90b58898d-Abstract.html>
- [9] “Speeding up the Xbox recommender system using a euclidean transformation for inner-product spaces | Proceedings of the 8th ACM Conference on Recommender systems.” Accessed: Apr. 03, 2024. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/2645710.2645741>
- [10] “Music Recommendation Systems: Techniques, Use Cases, and Challenges | SpringerLink.” Accessed: Apr. 03, 2024. [Online]. Available: [https://link.springer.com/chapter/10.1007/978-1-0716-2197-4\\_24](https://link.springer.com/chapter/10.1007/978-1-0716-2197-4_24)
- [11] “Which Emotions Can be Induced by Music? What Are the Underlying Mechanisms? And How Can We Measure Them?: Journal of New Music Research: Vol 33, No 3.” Accessed: Apr. 03, 2024. [Online]. Available: <https://www.tandfonline.com/doi/abs/10.1080/0929821042000317822>
- [12] P. Lops, D. Jannach, C. Musto, T. Bogers, and M. Koolen, “Trends in content-based recommendation,” *User Model. User-Adapt. Interact.*, vol. 29, no. 2, pp. 239–249, Apr. 2019, doi: 10.1007/s11257-019-09231-w.
- [13] F. Ricci, L. Rokach, and B. Shapira, “Recommender Systems: Introduction and Challenges,” in *Recommender Systems Handbook*, F. Ricci, L. Rokach, and B. Shapira, Eds., Boston, MA: Springer US, 2015, pp. 1–34. doi: 10.1007/978-1-4899-7637-6\_1.
- [14] “Topology of music recommendation networks | Chaos: An Interdisciplinary Journal of Nonlinear Science | AIP Publishing.” Accessed: Apr. 03, 2024. [Online]. Available: <https://pubs.aip.org/aip/cha/article->

abstract/16/1/013107/321859/Topology-of-music-recommendation-networks?redirectedFrom=fulltext