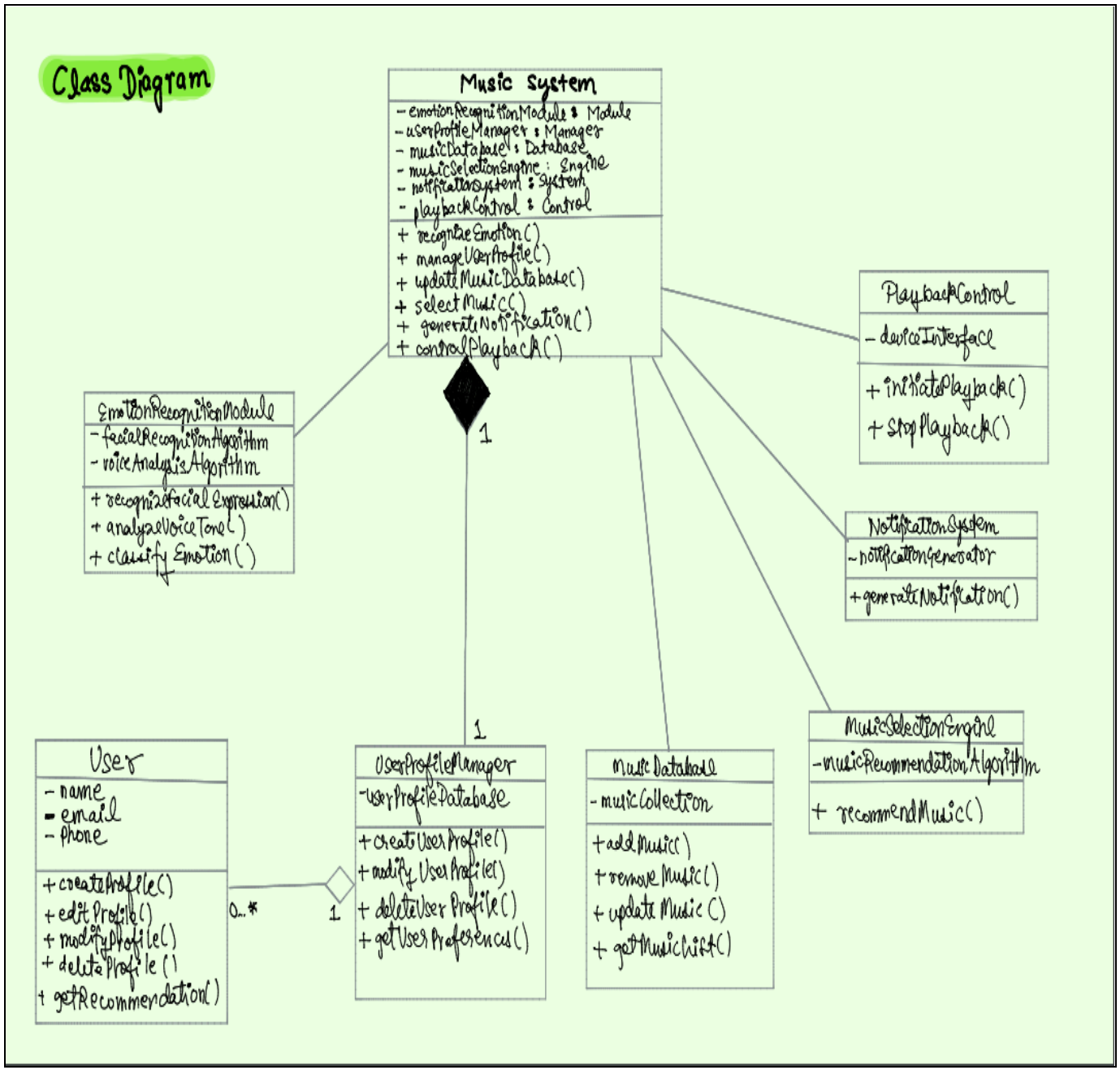
Music Recommendation System Using Facial Emotions - FaceBeats

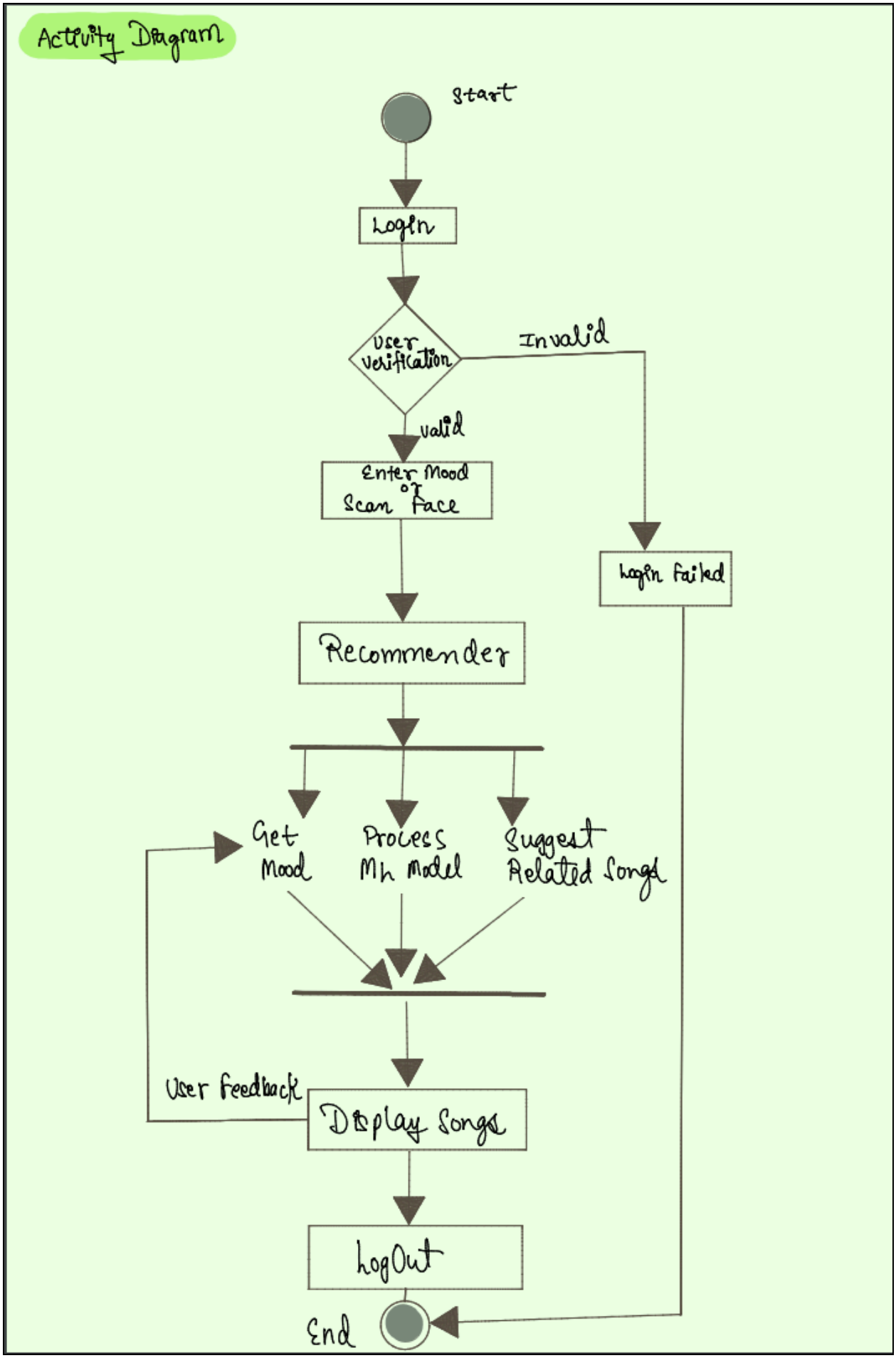
Group Members :

- **Aditya Sahani (B22CS003)**
- **Jaiswal Aditya Ranjit (B22CS025)**

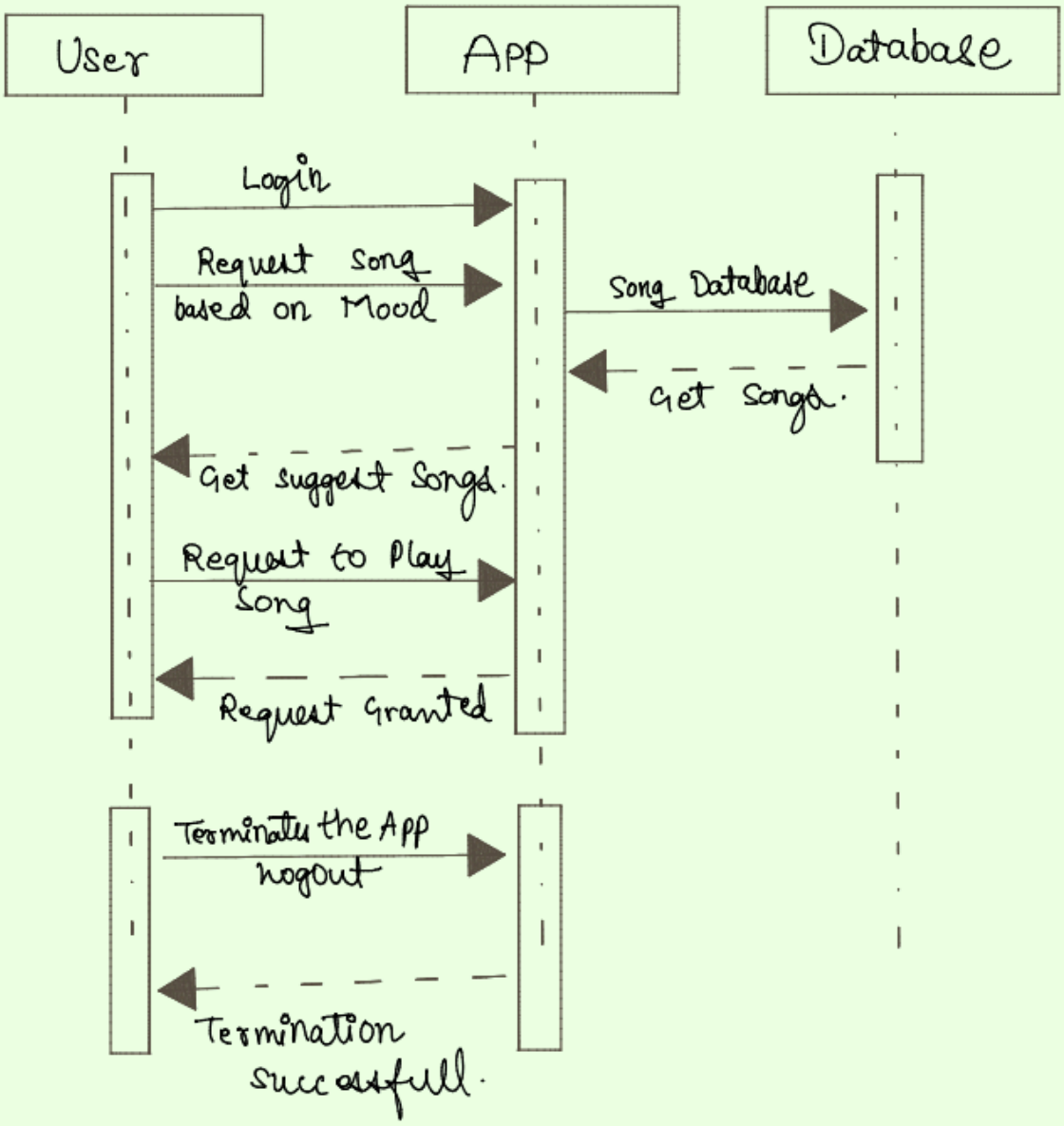
UML Diagrams :

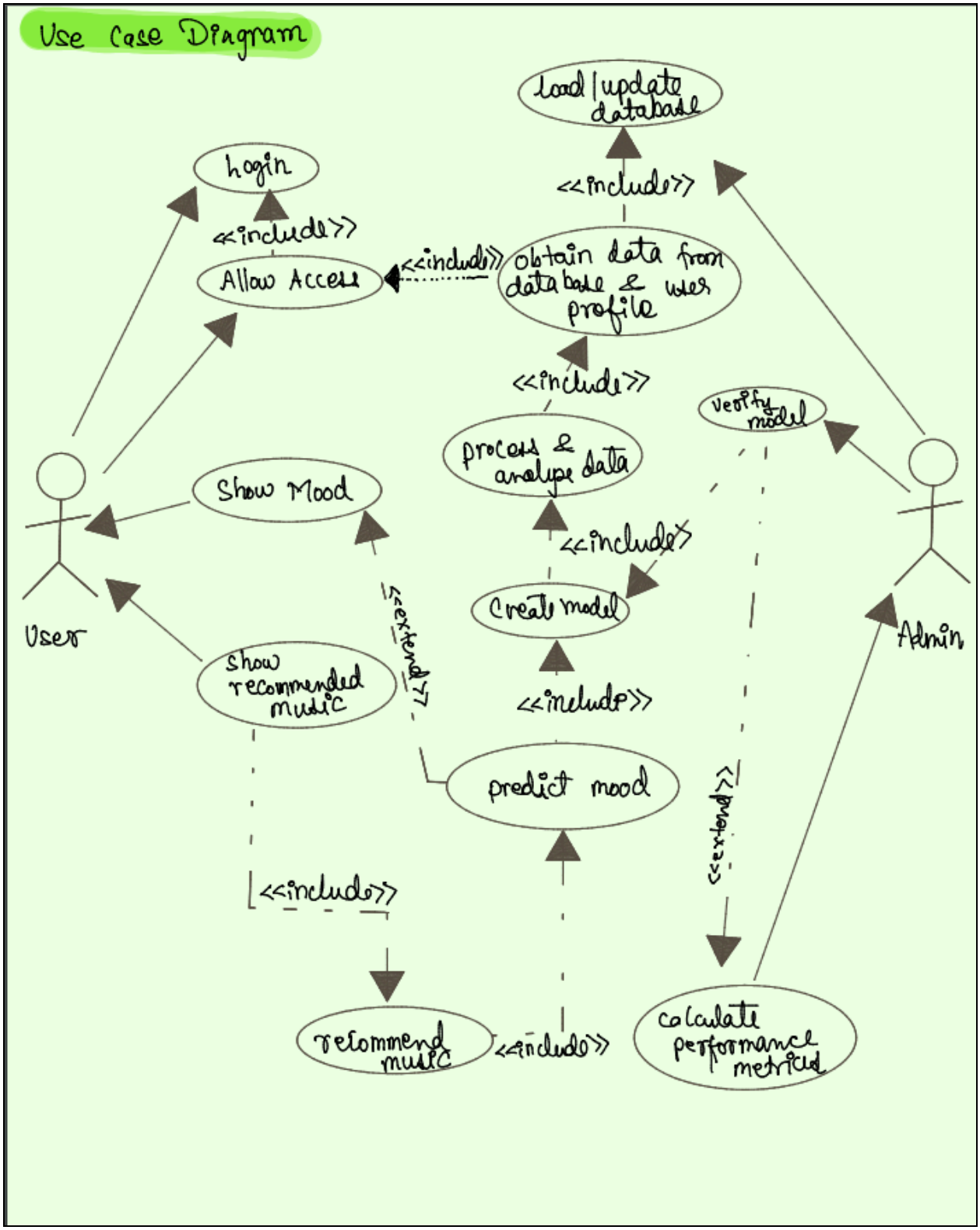
Unified Modeling Language (UML) diagrams are a standardized visual representation of software systems and processes. They provide a common language for software engineers, designers, and stakeholders to communicate and understand the structure, behavior, and interactions of a system. UML diagrams are widely used throughout the software development lifecycle, from initial analysis and design to implementation and maintenance.


Class Diagram
Music System
- emotionRecognitionModule : Module
- userProfileManager : Manager
- musicDatabase : Database
- musicSelectionEngine : Engine
- notificationSystem : System
- playbackControl : Control
+ recognizeEmotion()
+ manageUserProfile()
+ updateMusicDatabase()
+ selectMusic()
+ generateNotification()
+ controlPlayback()
PlaybackControl
- deviceInterface
+ initiatePlayback()
+ stopPlayback()
EmotionRecognitionModule
- facialRecognitionAlgorithm
- voiceAnalysisAlgorithm
+ recognizeFacialExpression()
+ analyzeVoiceTone()
+ classifyEmotion()
NotificationSystem
- notificationGenerator
+ generateNotification()
MusicSelectionEngine
- musicRecommendationAlgorithm
+ recommendMusic()
User
- name
- email
- phone
+ createProfile()
+ editProfile()
+ modifyProfile()
+ deleteProfile()
+ getRecommendation()
UserProfileManager
- userProfileDatabase
+ createUserProfile()
+ modifyUserProfile()
+ deleteUserProfile()
+ getUserPreferences()
MusicDatabase
- musicCollection
+ addMusic()
+ removeMusic()
+ updateMusic()
+ getMusicList()
1
1
0..*
1

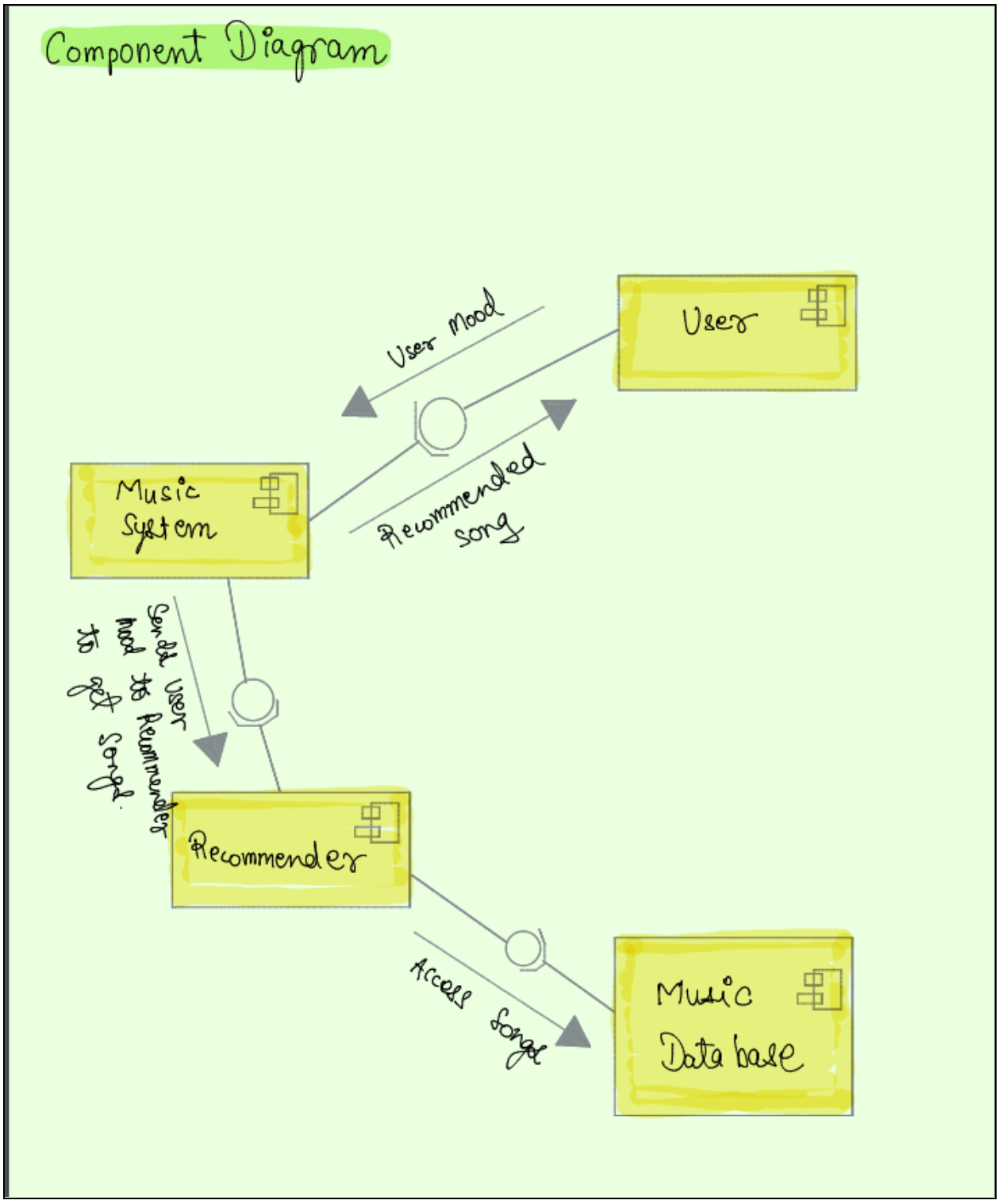

Activity Diagram
start
Login
User verification
Invalid
valid
Enter Mood or Scan Face
Login failed
Recommender
Get Mood
Process Ml model
Suggest Related Songs
User feedback
Display Songs
LogOut
End

Sequence Diagram

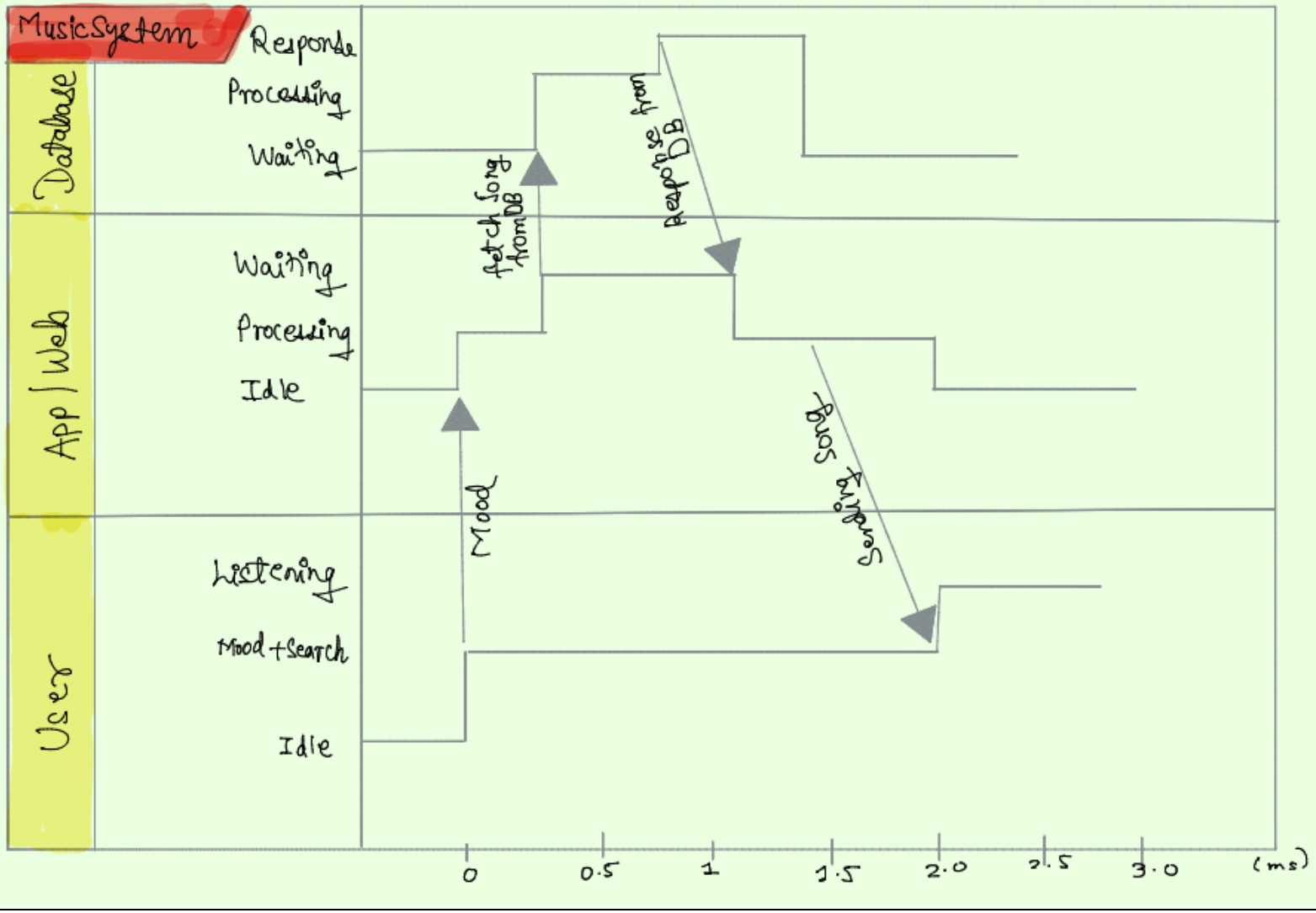



Use Case Diagram
load/update database
login
<<include>>
<<include>>
Allow Access
<<include>>
obtain data from database & user profile
<<include>>
verify model
process & analyse data
Show Mood
<<include>>
Create model
User
Admin
<<extend>>
Show recommended music
<<include>>
predict mood
<<include>>
<<extend>>
<<include>>
recommend music
<<include>>
calculate performance metrics

Component Diagram



Timing Diagram



Software Requirement Specification:

Introduction:

Purpose

The purpose of this project is to create an innovative system that enhances user experience by employing emotion recognition technology to identify the emotional state of a person. Based on the recognized emotions,by taking emotion as user input the system will automatically initiate the playback of the individual's favorite music, thereby creating a personalized and emotionally resonant music experience.

Scope

The scope of the project encompasses the development of a comprehensive system consisting of both hardware and software components. It involves the integration of emotion recognition algorithms, a user profile management system, a dynamic music database, and a seamless control mechanism to trigger music playback

devices.

System Description:

System Overview

The system will be a sophisticated amalgamation of cutting-edge technology. Emotion recognition algorithms will be utilized to analyze various facets such as emotion of a person through text inputs , facial recognition or other biometric parameters as textual input .These algorithms will be responsible for determining the user's emotional state, which will serve as the foundation for selecting appropriate music.

Functional Requirements

Emotion Recognition

- The system shall employ state-of-the-art emotion recognition algorithms capable of real-time analysis.
- Recognition methods may include emotional inputs or a combination of biometric indicators to ensure accuracy.

User Profile Management

- Users shall have the ability to create, modify, and delete profiles.
- Profile updates should encompass preferences such as favorite genres, artists, and specific songs.

Music Database

- The system shall maintain a dynamic and expansive music database, regularly updated to include a diverse range of genres, artists, and moods.
- The database should be easily expandable to accommodate new music selections.

Music Playback Control

- The system shall seamlessly interface with various music playback devices, ranging from smart speakers to headphones.
- Music playback initiation shall be prompt, creating a

fluid and immersive experience for the user.

Notification System

- Users shall receive notifications detailing the activated music, providing information such as title, artist, and a brief description of why it was selected.
- Notifications may be customizable to suit user preferences.

Non-Functional Requirements

Performance:

- The system shall strive for minimal latency in emotion recognition and music playback initiation, ensuring a responsive and engaging user experience.
- Response times for user profile updates and database queries shall be optimized for efficiency.

Security:

- Robust security measures shall be implemented to safeguard emotion recognition data and user profiles, ensuring privacy and data integrity.
- Access to sensitive information shall be restricted to authorized users.

Usability:

- The user interface shall be designed with an emphasis on intuitiveness and simplicity.

- The system should be adaptable to various environments and cater to a diverse user demographic.

System Constraints

Hardware:

- The system may necessitate compatible devices equipped with high-quality cameras or microphones for optimal emotion recognition.

Connectivity:

- The system may have a dependency on a stable internet connection for real-time updates to the music database and remote management functionalities.

Tech Stack:

Frontend : ReactJS, Tailwind

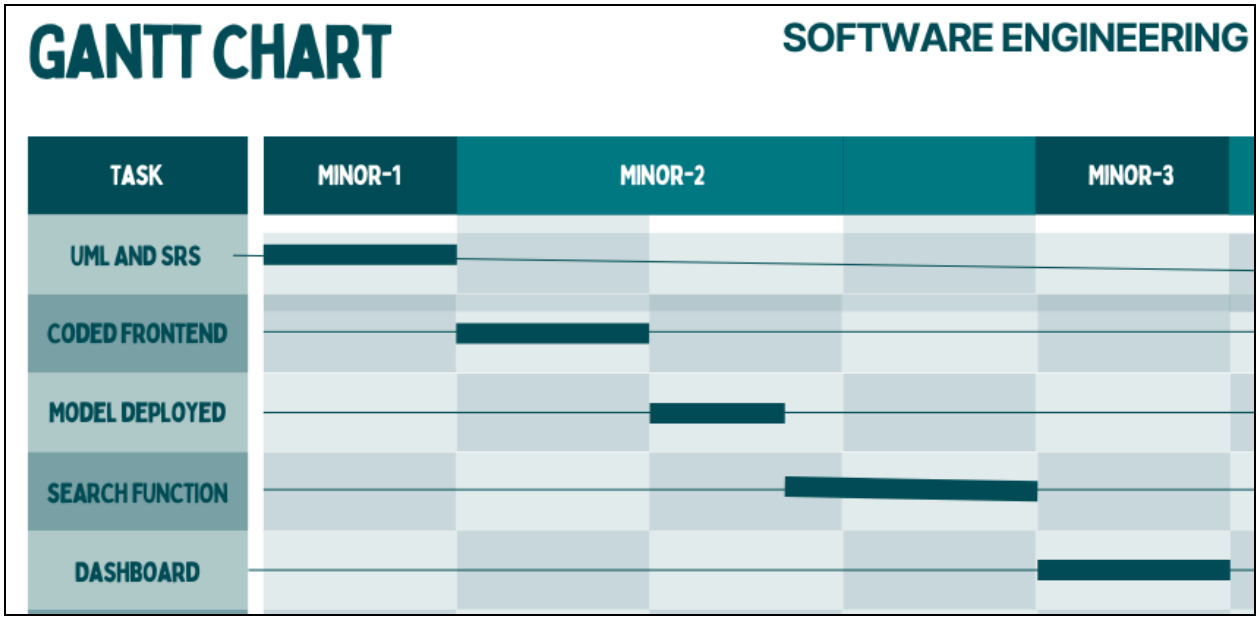
Backend : NodeJS, ExpressJS, Flask

Database : MongoDB

Libraries : Tensorflow, Pytorch

Features:

1.) Dashboard/Main
2.) Recommendation
3.) Search
4.) Profile



Software Development LifeCycle used for the Project:

Software Development Lifecycle (SDLC) Used: Waterfall

Description:
Each phase must be completed before moving on to the next, making it suitable for projects with well-defined requirements and limited iterations.

How Our Project Followed Its Steps:

1. Requirements Phase:

Waterfall Approach: As students, we started by understanding the project requirements and documenting them thoroughly.
Project Implementation: We gathered project requirements by discussing the scope and objectives of the music recommendation system based on emotions. This included identifying features such as user authentication, emotion analysis, recommendation, and profile management. We documented these requirements in a simple project proposal or specification document.

2. Design Phase:

Waterfall Approach: Once the requirements were clear, we proceeded to design the system's architecture and user interface.
Project Implementation: We created mockups or wireframes to visualize the user interface components, such as the Navbar, Homepage, ProfilePage, and LoginPage. Additionally, we outlined the backend services and database structure needed to support the application.

3. Implementation (Coding) Phase:

Waterfall Approach: With the designs finalized, we began coding the project according to the specifications.
Project Implementation: We developed the frontend components using React.js, focusing on creating basic functionality for navigation, user authentication, and profile management. For the backend, we used Flask and Express to set up simple API endpoints to handle user requests.

4. Testing Phase:

Waterfall Approach: After coding, we tested the system to ensure it met the requirements and worked as expected.
Project Implementation: We conducted testing after completing each major component. This involved manually testing the user interface for usability and functionality and ensuring that the backend services responded correctly to requests. We fixed any issues discovered during testing before proceeding to the next phase.

Software Architecture pattern used for the Project:

1. Client-Server Architecture:

Description : Our application follows a client-server architecture, where the React.js frontend serves as the client and communicates with the Flask and Express backend servers.

Implementation : React.js handles the presentation layer and user interactions in the browser, while Flask and Express handle the business logic and data processing on the server-side.

2. Microservices Architecture:

Description : The backend services are designed as microservices, with each service responsible for a specific task or functionality. For example, Flask handles machine learning model inference, while Express manages user authentication and profile management.
Implementation : Flask and Express provide lightweight, independent services that can be developed, deployed, and scaled independently. This architecture promotes modularity, flexibility, and scalability in your application.
React components such as Navbar, Homepage, ProfilePage, and LoginPage encapsulate specific UI elements and functionality. Each component can be developed, tested, and deployed independently, promoting modularity and code reusability.

3. Model-View-Controller (MVC) Pattern:

Description : The backend services follow the MVC pattern, where the model represents the data (e.g., user profiles, song recommendations), the view handles the presentation logic (e.g., HTML rendering), and the controller manages the application flow and business logic.
Implementation: Flask and Express route incoming requests to the appropriate controller, which interacts with the model to retrieve or manipulate data. The controllers then render the appropriate views (e.g., JSON responses for API endpoints) to the client.

4. RESTful API Design:

Description : Your backend services expose RESTful APIs, providing a standard and scalable way for the frontend to interact with the server.

Implementation: Flask and Express define API endpoints that follow REST principles, using HTTP methods (GET, POST, PUT, DELETE) to perform CRUD operations on resources such as user accounts, recommendations, and playlists. These APIs are stateless and platform-independent, facilitating interoperability and integration with the React.js frontend.

5. Single Page Application (SPA) Architecture:

Description: Your frontend application is built as a single-page application, where the entire UI is loaded once and subsequent interactions are handled dynamically without full page reloads.
Implementation: React.js enables the creation of a responsive and interactive UI that updates dynamically based on user actions and server responses. Routing is managed internally within the React application using libraries like React Router, allowing for smooth navigation between different views (e.g., dashboard, recommendation, profile).

Testing:

Unit Testing:

- Test individual components or modules of the system in isolation to verify that they function correctly.
- For example, we wrote unit tests for the facial emotion recognition module, user authentication module, and search functionality to ensure they perform as expected.

Integration Testing:

- Test the interactions and integration between different components/modules of the system.
- For instance, tested how the facial emotion recognition module integrates with the recommendation engine, and how user authentication interacts with the dashboard functionality.

System Testing:

- Test the system as a whole to verify that all components work together as intended and meet the specified requirements.
- Tested scenarios such as user authentication followed by successful music recommendations based on facial emotion, or searching for songs by emotion.

Performance Testing:

- Evaluate the system's responsiveness and scalability under different load conditions.
- Test scenarios such as multiple users accessing the system simultaneously, and measure response times for various operations.
- Used tools like Apache JMeter or Gatling for load testing.

Usability Testing:

- Assessed the system's ease of use, user interface design, and overall user experience.
- Involve representative users to perform usability testing and gather feedback on the system's usability.
- Test scenarios such as navigating the dashboard, using the search feature, and interpreting music recommendations based on facial emotion.

Security Testing:

- Identify and mitigate security vulnerabilities to protect sensitive user data and ensure system integrity.
- Test scenarios such as ensuring secure user authentication, preventing unauthorized access to user accounts, and protecting against potential security threats.

Problems faced till now:

- Integration Challenges : Integrating frontend components with backend services and ensuring smooth communication between them presented challenges, especially when there were discrepancies in data formats or API endpoints.
- Complexity of Emotion Analysis : Implementing emotion analysis functionality using machine learning models was challenging, particularly in terms of model accuracy and real-time performance.
- Resource Limitations : As students, we had limited access to resources such as computing power or datasets, which hindered the development and testing process.

- Technical Debt : Rushing to meet deadlines or lack of experience led to the accumulation of technical debt, such as poorly written code or insufficient documentation.
- User Experience Issues : Designing an intuitive and user-friendly interface while ensuring it effectively communicates.

Problems that we will be facing in future:

- Scalability Issues: If our project gains popularity among students or users, it may struggle to handle increased traffic or data volume. Scaling the system to accommodate growth while maintaining performance could be challenging, especially with limited resources and expertise.
- Solution: We can mitigate scalability issues by designing our system with scalability in mind from the beginning. This could involve using cloud-based services that offer scalable infrastructure, optimizing database queries and server-side code for efficiency, and periodically reviewing and upgrading our hosting plan as needed.

- Maintenance and Updates: Keeping our project up-to-date with the latest technologies, addressing bugs, and implementing new features may become increasingly complex and time-consuming as students. Limited time and resources may hinder our ability to maintain and update the system effectively.
- Solution: To handle maintenance and updates effectively, we can establish a regular schedule for reviewing and updating our project. This includes allocating time for bug fixes, security patches, and feature enhancements. Additionally, leveraging automated testing and deployment tools can streamline the process and reduce the burden on our limited resources.

Future scope for the project :

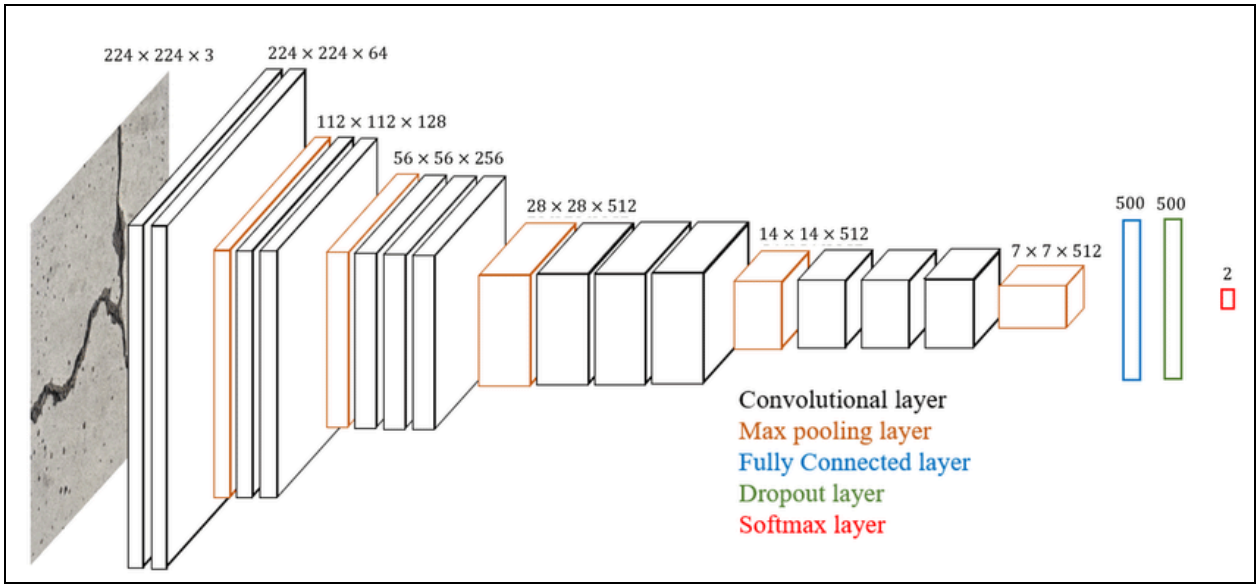
1. Enhanced Recommendation Algorithm:
Implement more sophisticated emotion analysis algorithms to improve the accuracy of song recommendations based on facial expressions. Explore machine learning models to personalize recommendations further based on user preferences, listening history, and feedback.

2. Social Integration:
Allow users to connect with friends and share their favorite songs or playlists.

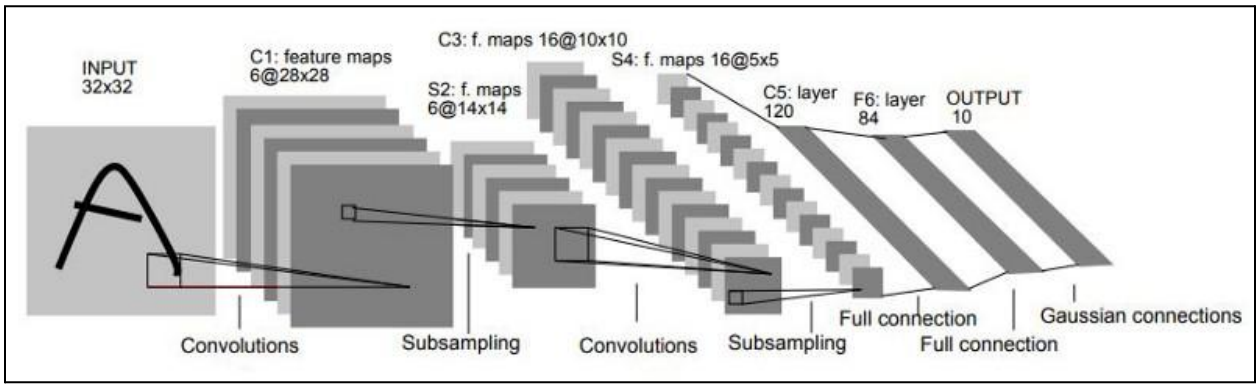
3. Playlist Management:
Enable users to create and manage their playlists based on their mood or activity.

4. Emotion-Based Playlist Generation:

Introduce the ability for users to create playlists based on specific emotions or moods. For example, users could select a "happy" or "calm" playlist and automatically populate it with recommended songs.



VGG-16 ARCHITECTURE



LENET-5 ARCHITECTURE

Contributions for the project :

1. Aditya Sahani (B22CS003) :
    - Created “fetchUser.js” middleware.
    - Created “api’ endpoints for user-authentication.
    - Developed ML model for prediction .
    - Created Recommendation Section and Profile Bar.
2. Aditya Jaiswal (B22CS025) :
    - Created Models for database.
    - Created Server using ExpressJS.
    - Deployed the model using flask.
    - Created Dashboard and Search section.