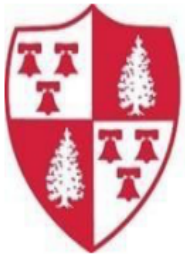

MONTCLAIR STATE UNIVERSITY

EMOTION DETECTION AND MUSIC PLAYER SYSTEM

COURSE

CSIT 515-02 Software Engineering & Reliability- Spring 2024

SUBMISSION DATE

05-01-2024

Prepared For:

Dr. Hubert Johnson BS, MS, Ed.M., EdD

Prepared by: TEAM CREW

Ajay Moru

Chakka Satya Vaishnavi

Nimisha Twinkle, Bonigala

Paramasiva Nandhisai Manikanta Vemavarapu

Team Member Details:

Name	Email id	Phone Number
Ajay Moru	Morua1@montclair.edu	+1 2016794519
Chakka Satya Vaishnavi	chakkav1@montclair.edu	+1 2486881702
Nimisha Twinkle, Bonigala	bonigalan1@montclair.edu	+1 5512603359
Paramasiva Nandhisai Manikanta Vemavarapu	vemavarapup1@montclair.edu	+1 2012842493

Team Meeting Details:

We are meeting virtually.

Tuesday: 4.30 pm -5.30 pm
Thursday: 4.30 pm -5.30 pm
Friday: 4.30 pm -5.30 pm

ACKNOWLEDGEMENT

We are pleased to present our project, the “Emotion based Music Player system,” and extend our heartfelt gratitude to all those who supported us along the way.

We express our deepest gratitude towards Prof. Hubert Johnson who gave valuable and timely advice during the different phases in our project. We would also like to thank him for the resources and facilities he provided, which greatly facilitated our work. We thank him for support, patience, and faith in our capabilities and for giving us flexibility in working and reporting schedules.

ABSTRACT

This project aims to develop an innovative Emotion-Based Music Player System (EMP) designed to enhance the user experience by dynamically adapting music based on the user's emotions. The system's architecture is meticulously designed, featuring a modular structure that utilizes Flask for the web interface and emotion recognition algorithms for analyzing user emotions.

The EMP comprises modules such as emotion detection and playlist generation. It leverages cutting-edge technologies to accurately analyze user emotions solely through facial expressions. The system's secure login and logout functionalities ensure a safe and private platform for users to interact with, enhancing the overall user experience.

Additionally, the EMP includes a contact feature allowing users to send emails directly from the system to admin. It also features a dedicated feature page that showcases all the functionalities available in the existing version and provides insights into upcoming features in the next version.

By focusing on facial expressions as the primary input modality for emotion detection, the EMP aims to create a personalized and engaging music listening experience that resonates with users' emotions.

TABLE OF CONTENTS

1. Introduction

Music is one of the widely accepted cultures and languages which can be accepted by any type of people. Music does fill the silence and can hide the noise. It is an excellent way for a person to relieve stress. Choice of music of a person is dependent on the person's mood. Mood of a person can be inferred from the person's emotion and emotions can be inferred from the person's facial expressions. According to the survey, two-thirds of people's communication is carried through non-verbal communication and facial expression constitutes the largest component in this percentage.

This project will embark on a groundbreaking journey to develop an "Emotion Based Music Player" that will accurately detect the emotion of a user and play the music accordingly. Nowadays, emotion detection is considered intrinsic in many applications such as retail, education, medical diagnosis, video gaming, security, criminal etc.

1.1 Project Overview

The Emotion-Based Music Player(EMP) system will be an innovative and robust application designed to detect emotions and play an appropriate song from a list of songs organized to fit particular moods. First, the user will reflect their emotion through facial expression. After that, the device will detect the condition of the facial expression, analyze it, and interpret the emotion. After determining the user's emotion, the music player will play songs that can suit the detected emotional state.

1.2 Objective

- The main objective of this project will be to automatically play user-preferred music based on emotion identification.
- This model will aim to accurately detect basic emotions, namely happy, angry, neutral, sad, surprised.
- The system will aim to raise user engagement and satisfaction levels using music recommendation systems.
- The system will be user-friendly and simplistic.

2. Requirements Specification Document

2.1 Introduction

This document will outline the requirements for the EMP system, a software solution designed to play music based on the detected emotion of the person. EMP will be particularly helpful for users who have difficulty in choosing the right music to match their mood by offering a more efficient, accurate, and secure solution.

2.2 Product Scope

The EMP system will address the need for a personalized, immersive, and emotionally resonant music listening experience. It will aim to provide user-friendly music tailored to the emotions experienced by the listener. Traditional music players like APPLE MUSIC, SPOTIFY, and WYNK MUSIC often lack the ability to adapt to users' emotions and preferences, requiring manual song selection and resulting in a less engaging and fulfilling experience. EMP will overcome these challenges by employing innovative emotion recognition technology, thereby curating playlists tailored to the user's current emotional state. The domain-specific background will involve understanding the limitations of traditional music players in catering to users' emotional needs and the advantages offered by personalized, emotion-based music recommendations. The implementation of EMP will lead to significant benefits, including a deeper emotional connection with the music, personalized playlists that resonate with the user's mood, efficient feature computation, improved accuracy, and a more engaging and fulfilling music listening experience.

2.3 Product Features

- Registration: Users will be able to create a new account by providing necessary details such as username, email, phone number and password. Upon successful registration, they will receive a confirmation message.
- Login: Registered users can securely log in to their accounts using their username/email and password. The system will authenticate their credentials and grant access to their personalized profile.
- Human Face Detection: Once the user logs in, the system will be capable of identifying and locating faces within a given image using advanced facial recognition technology.
- Emotion Detection: Upon successful face detection, the system will proceed to analyze facial expressions such as eye movement, smiles, and frowns to accurately determine the emotional state of the user. This analysis will be powered by machine learning models that are specifically trained for real-time emotion detection, ensuring precise and responsive results.
- Music Player: Once the user’s emotional state is detected, the system will automatically select music that matches their mood and play an appropriate song.
- Contact Page Feature: Users will have the option to use the contact page to send feedback, report issues, or seek support regarding the system's functionalities.

- Features Page: A dedicated page will showcase the features of both current and upcoming versions in a clear and informative manner, helping users understand the system's capabilities.
- Team Info Page: The system will include a dedicated team information page where users can access details about the development team behind the project.
- Logout: Users will securely log out of their accounts, ensuring the clearance of authentication tokens to prevent unauthorized access.

2.4 Product Constraints

- When a disgusting feeling is identified, our proposed system will find it difficult to select appropriate music to play.
- The system will play songs based on users' moods, including playing depressing music for users feeling sad. However, it may face challenges when users in a sad mood want happy songs to uplift their spirits.
- The system will find it difficult to detect and analyze facial expressions in adverse conditions like bad lighting or poor webcam resolution.
- Users with disabilities, such as visual impairments, hearing impairments, might face challenges when accessing or using the system.

2.5 Functionalities of System

- Face Detection: The system will identify and locate faces within a given image using facial recognition technology, and it will employ a Haar Cascade classifier for face detection, complemented by image preprocessing techniques to enhance accuracy.
- Emotion Detection: The system will employ machine learning models, specifically trained with convolutional neural networks (CNNs), to analyze facial expressions such as eye movement, smiles, and frowns. This approach will aim to determine the emotional state of a user in real-time through accurate emotion detection.
- Mood Classification: The system will categorize the detected emotion into predefined mood categories like happy, sad, angry, neutral.
- Music Selection: Once the user’s emotional state is detected, the system will automatically select music from the list that matches their mood and play an appropriate song. It may include controls such as play, pause, shuffle, next and stop, as well as volume and playback speed controls.
- User Interface: The system will provide a user-friendly interface for users to initiate the face detection process, view detected mood, control music playback, and incorporate functionalities such as login, logout, and registration, contacting admin, team information, and system features.
- Database: The system will utilize SQLAlchemy to connect to a SQLite database specifically designed for storing user login, registration, and logout data, ensuring efficient data management.
- Testing and Optimization: Ensuring the accuracy, reliability, and optimal performance of the system will be achieved through testing and continuous optimization in the future.

2.6 User Characteristics

Early Adopters (18-35):	● Age: Young adults, tech-savvy and open to exploring new technologies. ● Education: Diverse, ranging from high school graduates to professionals. ● Experience: Actively listens to music, uses streaming services, open to music discovery based on mood and emotion. ● Technical Expertise: Moderate to high, comfortable with smartphones, apps, and potentially wearable tech. ● Motivations: Early adopters may be curious about the novelty, exploring emotional well-being through music, or seeking personalized recommendations.
MusicEnthusiasts(20-45):	● Age: Focused on music lovers, not just early adopters. ● Education: Diverse, with an emphasis on passion for music over specific education levels. ● Experience: Deeply connected to music, actively seeking new artists and genres, understanding music-emotion connection. ● Technical Expertise: Moderate to high, comfortable with music apps and exploring new technology related to music. ● Motivations: Discover new music based on their emotions, find personalized playlists for different moods and activities, enhance their overall music experience.
Wellness Seekers (25-55):	● Age: Focus on individuals seeking emotional well-being through music. ● Education: Diverse, with an interest in self-help and mindfulness practices. ● Experience: Open to using technology for managing emotions, may already use meditation apps or biofeedback tools. ● Technical Expertise: Moderate, comfortable with smartphones and open to new tools that support well-being. ● Motivations: Use music for stress management, mood regulation, relaxation, or even guided meditations, improve emotional balance.

Table 3.1: User Characteristics

2.7 User Needs and System Benefits

The primary goal of the EMP system will be to satisfy the emotional needs of users through music.

- It will provide a platform for music lovers to explore, discover, and enjoy personalized music experiences tailored to their emotions.
- It will bridge the gap between growing technologies and music techniques.
- It will provide very good entertainment for the users.

2.8 Functional Requirements

This section will detail the functions of the system from the perspective of the user, outlining the expected inputs, system responses, and the types of outputs the user will encounter.

2.8.1 Inputs Expected

- When registering for a new account, users will be required to provide inputs such as name, email address, phone number, password, and confirmation of password. These inputs will be used to create and manage user accounts within the system.
- During the login process, users will be expected to input their email addresses and passwords to access their accounts securely.
- The system will require images containing facial expressions from users for real-time emotion analysis. These images will be used to detect and classify emotions based on facial features.
- Users will be able to provide feedback or ask queries through the contact page. The system will expect inputs such as name, email addresses, and messages provided by users for feedback purposes.
- The system will also expect playback controls for music, such as clicking on play, next, pause, shuffle, and stop buttons. Users will be able to select music from a list of songs for playback.

2.8.2 Actions Taken

- Emotion Detection:
    - The system will analyze the emotional input provided by the user.
    - If the input is clear, the system detects emotion and proceeds to select music accordingly.
    - The system will accept a range of emotions such as happy, angry, neutral, sad, surprised.
    - It will utilize machine learning models trained on diverse emotional datasets for accuracy.
- Music Selection:
    - Based on the recognized emotion and user preferences, the system will play music.
- Error Handling:
    - If emotional input is unrecognized, the system will prompt the user for clarification, display an error message, and stop playing songs.
    - The system will provide helpful error messages to guide the user.

- The system will implement security measures to protect user data and privacy.
- If the user is not in front of the camera, then the system will throw a message “Oops! Unable to detect mood.”
- If the mp3 files are not found in the directory, then also the system will throw a message to the user.

2.8.3 Outputs

- Upon successful login, the system will redirect the user to the home page.
- User interface elements will display the current emotional state.
- A recommended playlist will be provided based on the detected emotions.
- Options for users to play, pause, stop or shuffle the playlist will be available.
- Users will see an email sent to admin when they submit feedback or queries

2.9 Non-Functional Requirements

2.9.1 Usability:

- An intuitive and user-friendly interface will be developed, making it easy for users to understand and navigate.
- Integration will include clear and user-friendly feedback mechanisms for emotions and music selections.
- Comprehensive user documentation will be provided.

2.9.2 Reliability:

- Will ensure a reliable music playback without interruptions or glitches.

2.9.3 Security:

- Strong encryption will be implemented for user data and communication channels.
- Compliance with relevant data privacy regulations will be ensured.
- Protection against unauthorized access or vulnerabilities will be maintained with robust security protocols and regular updates.

2.10 Physical Environment and Interfaces

The system will be designed to be used on different devices such as laptops, and desktop computers. Users will be able to access the system through a web application ensuring compatibility with mentioned platforms.

2.11 Feasibility

By considering the flexibility and feasibility of the project, it will be implemented as an independent system that will be based on the user's emotion.

- Technical Feasibility:
    - In terms of the technical scope, the system will involve the development of emotion detection models, including data collection, algorithm selection, model architecture, training pipeline, hardware requirements, performance metrics, integration with system components, and maintenance strategies.
- Legal Feasibility:
    - Team will be committed to developing a product that will comply with all legal requirements, respect user privacy, and meet ethical standards.
    - Team will ensure that the system is intuitive, user-friendly, and offers value to users to gain adoption. Team will consider conducting user studies and gathering feedback to refine the system.
- Features will be added as time permits:
    - Will include more emotions like fear and disgust.
    - Will integrate with spotify for music streaming.
    - Will integrate with youtube for video content.

2.12 Assumptions and Dependencies

- Stable Internet Connection: The system will assume a stable and reliable internet connection for seamless web-based operations.
- Availability of Web Technologies: Developers will assume the availability and compatibility of web technologies, including browsers and frameworks like Flask, to deliver a user-friendly web interface.
- High-Resolution Web webcam: The system will rely on the assumption that users will have access to a high-resolution web webcam (720p or higher) for capturing clear facial images, essential for accurate face detection.
- Availability of External Libraries: The system will rely on the availability and proper functioning of external libraries, such as OpenCV and face recognition libraries, for facial detection and recognition tasks.
- Audio quality and latency: The system will aim to maintain high audio quality when personalizing music in real-time.
- Security and data encryption: User data, will be securely stored and transmitted with robust encryption to prevent breaches and ensure privacy.

2.13 Maintenance and Support

Maintenance and Support: Maintaining and supporting an EMP system will require attention to several key areas:

- Technical Maintenance:
    - Software updates:
        - Frequent updates will be pushed to ensure the system stays up-to-date with the latest bug fixes, security patches, and operating system compatibility updates.
    - Hardware maintenance
        - The system will maintain the hardware used for video capture, processing, and playback according to manufacturer recommendations.
- User Support:
    - User documentation:
        - The system will provide clear and comprehensive user documentation explaining its features, limitations, and usage instructions.
        - The documentation will be kept updated with any changes or new features.
    - User feedback mechanism:
        - A mechanism will be implemented for users to provide feedback on the system's performance, suggest improvements, and report bugs.
        - User feedback will be addressed promptly and transparently.
- Data and Model Maintenance:
    - Emotion recognition model:
        - The system will regularly update the emotion recognition model with new data to improve accuracy and adapt to changes in facial expressions or trends.
        - Performance of the model will be monitored, and it will be retrained if its accuracy drops significantly.
    - Music library:
        - The system will keep the music library updated with new releases and ensure proper tagging and organization for effective mood-based selection.
        - The mood associations of each song will be reviewed periodically and adjusted based on user feedback or performance analysis.
- Privacy and security:
    - User privacy will be ensured by implementing appropriate data security measures and obtaining informed consent for data collection and usage.
    - Adherence to relevant data privacy regulations will be ensured.
- Ethical considerations:
    - The technology will be used responsibly and transparently, with mindfulness of the ethical implications of emotion detection technology, including potential biases and limitations.

2.14 Cost

Development Team:	● Project Manager: $8,000 - $15,000 ● Software Developers (2-3): $30,000 - $60,000 ● Quality Assurance Engineers (1-2): $15,000 - $30,000
Technology and Tools:	● Licensing fees for third-party libraries and tools: $5,000 - $10,000 ● Development environment and server costs: $8,000 - $15,000
Hardware:	● High-resolution web cameras for testing and deployment: $2,000 - $5,00
Integration with external APIs or Music Streaming Services	● Cost : $5000 to $10,000
Training and Documentation:	● User training materials: $3,000 - $6,000 ● Documentation development: $5,000 - $10,000
Testing and Quality Assurance:	● Testing tools and resources: $5,000 - $10,000
Legal and Compliance:	● Legal consultation fees for ensuring compliance: $5,000 - $10,000
Miscellaneous:	● Contingency for unforeseen expenses: $5,000 - $10,000
Post-Deployment Support	● Ongoing maintenance and support costs: $10,000 - $20,000 (per year)
Total Estimated Cost Range:	● $96,000 - $201,000

Table 3.2: Cost

3. Design and Implementation Document

3.1 Introduction

The EMP System Design Document will serve as a comprehensive guide, outlining the components of the project designed to elevate the music listening experience. It will be meticulously crafted in line with industry software engineering standards, detailing the design necessary to build the EMP System.

3.1.1 Purpose

This document outlines the design of the EMP System. The system aims to enhance the user's music experience by adapting to their emotional state. By utilizing advanced emotion recognition technologies, the system will customize music playlists to match and enhance the user's emotions.

3.1.2 Scope

This system will focus on the design of the emotion detection algorithm, integration with hardware components like webcam, and the overall system architecture.

3.2 System Architecture

3.2.1 Overview of the System

The EMP System will be architected to provide a highly personalized and interactive music experience by harnessing the power of emotion recognition technology. It will be developed as a web-based application using Flask, powered by Python, to offer a user-friendly interface for listeners. The integration of advanced audio-processing and artificial intelligence algorithms will allow for real-time emotion detection, which will lead to the automatic playing of music.

Central to its functionality, the system will incorporate the following modules:

- Emotion Detection Engine: This will utilize sophisticated algorithms such as Convolutional Neural Networks (CNNs) and Haar cascade classifiers to analyze user input, including facial expressions, and determine their current emotional state.

- Dynamic Playlist Generator: This will integrate with the VLC media player, creating personalized music playlists that align with the detected emotions. This integration facilitates an immersive auditory experience that evolves with the user's mood, enhancing the overall music listening experience.

- User Interface: The system will provide a user-friendly interface for users to initiate the face detection process, view detected mood, control music playback, and incorporate functionalities such as login, logout, and registration, contacting admin, team information, and system features.

This structure will be meticulously designed to streamline the entire music listening process, from the initial emotion detection to the final delivery of a custom-tailored playlist, ensuring a responsive and engaging user experience.

3.2.2 Directory Structure

The system's directory structure is organized as follows:

- dataset_prepare.py: A utility script that will be designed to process and prepare the dataset used for training the emotion recognition model.
- Data: Within the Data folder, there will be two subfolders: the Test folder and the Train folder. These will be used for training the model, with the Train folder containing data for model learning and the Test folder for validating the model's accuracy.
- Songs: This folder will act as the musical library of the system, holding a- collection of songs to be played according to the detected emotions.
- model_train.py: A script that will contain the machine learning training routines to create models capable of understanding and predicting emotions.
- emotions.py : This will include the logic for emotion detection, likely interacting with the machine learning model for real-time emotion recognition and will also be an entry point for flask application.
- haarcascade_frontalface_default.xml: This XML file will be utilized by OpenCV's for detecting frontal faces in both images and video streams.
- model.h5 : This will be the saved machine learning model file, typically in HDF5 format, containing the architecture and weights of the trained neural network.
- musicplayer.py: The main script that will be responsible for controlling the music playback functionality of the system.
- Static : This directory will hold static content like CSS files, JavaScript, and images for the web interface.
- Templates : Contains the HTML templates which will define the structure and layout of the web application's user interface.

These files and directories will form the essential components required to build and run an EMP system that can detect a user's emotions and play music that complements their current state.

3.2.3 Technologies Used

The EMP System will utilize a variety of Python libraries, each contributing to its functionality and performance:

1. Flask: This library will serve as the backbone of our web application. Flask will be used for server setup, handling requests, and rendering pages.
2. OpenCV (cv2): An open-source computer vision and machine learning software library that will be used for image processing and face detection functionalities.
3. SQLAlchemy: It will indeed be a lightweight and efficient ORM (Object-Relational Mapping) library for Python that will provide a flexible and powerful database solution for storing and managing user data.
4. Python-VLC: For the music playback functionality, we will integrate python-vlc to leverage the robust media handling capabilities of the VLC media player.
5. Flask-Mail: It provides a simple interface for sending email messages from your Flask application.
6. Bcrypt: A popular library for secure password hashing, Bcrypt ensures that user passwords are stored securely in the database.
7. NumPy: We will use NumPy extensively for its powerful numerical computation capabilities, data manipulation, array operations, mathematical functions, and data processing tasks.
8. TensorFlow: These libraries will be employed to build and train deep learning models that can accurately predict the user's emotional state from different inputs.
9. Tkinter: A standard GUI (Graphical User Interface) library for Python, Tkinter will be used to build a user-friendly interface with interactive interfaces using buttons, labels, text boxes, menus, and other GUI elements.

For the front-end development of the web application, the following technologies will be utilized:

1. HTML (Hypertext Markup Language): This standard markup language will be used for creating web pages. HTML will design the structure of web pages in the system.
2. CSS (Cascading Style Sheets): This style sheet language will be used for describing the presentation of documents written in HTML. CSS will style and layout web pages, enhancing the user interface.

3.2.4 Imports

- The Emotion detection module will import functionalities from Flask for web application development, including routing, request handling, and template rendering.
- It will also import machine learning libraries such as TensorFlow and OpenCV for facial emotion analysis and classification.
- It will also import libraries for email handling, such as Flask-Mail, to manage user feedback and communication.
- It will also import the NumPy library for data manipulation and processing tasks.
- It will import bcrypt for password hashing and verification, enhancing security measures for user authentication and credential protection.

- It will also import SQLAlchemy for ORM (Object-Relational Mapping) functionalities, enabling efficient interaction with the database for user account management and data storage.
- The Music Player Module will import libraries or modules for audio playback control, such as VLC, to manage music playback functionalities.

3.2.5 Exports

Recognize_emotion(face_coordinates): This function will take the identified face coordinates as input and will determine the corresponding emotion expressed on the detected face. The output of this function will be essential for the Music Player module, as it will rely on recognizing emotions to enhance the user experience.

Manage_playlist(emotion): This function will organize a playlist based on the recognized emotion. It will ensure that the songs in the playlist align with the identified emotion and will play a song based on the detected emotion.

3.2.6 Input/Output

- Registration:
  - Input: Users will provide inputs such as name, email address, phone number, password, and password confirmation.
  - Output: Successful registration will result in the creation of a user account within the system.
- Login:
  - Input: Users will input their email address and password for account access.
  - Output: Successful login will allow users to access their accounts securely.
- Emotion Detection:
  - Input: User's face will be captured by a webcam in real time.
  - Output: The system will output detected and classified emotions based on facial features from input images.
- Feedback and Queries:
  - Input: Users will input their name, email address, and messages for feedback or queries through the contact page. After submitting the details an email will be sent to admin
  - Output: The admin will process feedback and query resolution.
- Music Playback:
  - Input: Users will interact with playback controls including play, next, pause, shuffle, and stop buttons.
  - Output: Once the emotion is detected, a music will be played aligning with emotion. Users will receive output in the form of music playback based on their selections and controls.

3.2.7 Modules

EMP system will employ a multi-tier architecture consisting of the following modules:

1. User Authentication: As the name suggests, this module will mainly deal with login and logout functions. It will also take care of creating an account in a very secure manner.
2. Data Collection Module: This module will be responsible for collecting the necessary data to train the machine learning algorithms. It will include collecting data on facial expressions, music preferences.
3. Emotion Detection Module: This module will analyze the user's facial expressions to determine their emotional state. It will use computer vision techniques such as face detection and emotion recognition algorithms to determine the user's mood.
4. Music Selection Module: This module will select music that corresponds to the user's emotional state.
5. User Interface Module: This module will mainly deal with user interfaces throughout the application. The goals of our UI will be to make it user-friendly, and really simplistic.
6. Machine Learning Model Training Module: This module will train the machine learning algorithms used in the Emotion Detection Module to learn patterns and relationships within the data. It will utilize large datasets of facial expressions to improve the accuracy of emotion recognition within the system.

These modules will work together to create an Emotion Based Music Player System that provides a personalized music experience for the user based on their emotional state.

3.2.8 Pre-conditions

- System Initialization:
    - The system will be installed and properly configured on the user's device by themselves, following the user guide outlined in this document.
    - Required hardware components must be well functional and connected.
- Data Availability:
    - The system will have access to a dataset of music tracks.
    - Emotion detection algorithms and models will be trained and available for use.
- User Interaction:
    - The user will interact with the system and grant necessary permissions for accessing the microphone and other resources to the system itself.

3.2.9 Post-conditions

- Emotion Detection:
    - The system should successfully detect the emotions of the user based on the input data, which typically includes facial expressions captured through a webcam.
    - Emotions will be classified into predefined categories, including happy, sad, angry, neutral, surprised.
    - The system will provide a confidence score or level of certainty associated with each detected emotion.
- User Feedback:
    - Users will offer feedback on the accuracy of the emotion detection.

- Music Playback:
    - The music player will select and play music tracks based on the detected emotions.
    - The music playback will be seamless and without interruptions, providing a satisfying user experience.
- User Satisfaction:
    - The system will provide a user-friendly interface for users to initiate the face detection process, view detected mood, control music playback, and incorporate functionalities such as login, logout, and registration, contacting admin, team information, and system features.

3.2.10 Hardware and Software Requirements

- Hardware Requirements

The EMP system will be designed to be efficient and versatile, capable of running on a variety of hardware setups. However, to ensure optimal performance, the following hardware specifications are recommended:

1. Processor	Intel Core i5 or equivalent, with at least 2 GHz processing speed. This will ensure efficient handling of image processing and facial recognition computations.
2. RAM	A minimum of 4GB RAM is recommended. Higher memory will enhance the system's ability to handle simultaneous requests and larger datasets.
3. Web webcam	A high-resolution web webcam (720p or higher) will be essential for capturing clear images, which will be crucial for accurate face detection and recognition.
4. Hard Disk	At least 10GB of free disk space will be required for the operating system, application files, and database storage.
5. Network Interface	A stable internet connection for accessing the web-based interface.
6. Display	A monitor capable of displaying at least 1024x768 resolution will be recommended for a clear view of the application's graphical user interface
7. Microphone/ Speaker	An audio output devices will be required for playing music based on detected emotions

Table 3.1: Hardware Requirements

- Software Requirements

The software requirements are as follows:

1. Operating System	The system will be platform-independent and can run on different operating systems including Windows, macOS, and Linux.
2. Python	Python 3.6 or newer will be required, as the application and its dependencies are written in Python.
3. Flask	The Flask framework will be used for the web application, which requires Python. It is lightweight and suitable for a variety of web applications.
4. OpenCV	For image processing and face detection features. OpenCV (Open-Source Computer Vision Library) will be an open-source computer vision and machine learning software library.
5. SQLAlchemy	It will indeed be a lightweight and efficient ORM (Object-Relational Mapping) library for Python that will provide a flexible and powerful database solution for storing and managing user data.
6. Web Browser	Any modern web browser (like Chrome, Firefox, Safari) is required to access the web interface of the application.

Table 3.2: Software Requirements

3.3 System Design

The design phase in software development will be crucial, focusing on creating the system's technical blueprint to ensure functionality and user interaction. It will involve defining requirements and conceptualizing the software's architecture, including web page interconnections, and breaking down software components into modules and data structures. This section will cover multiple diagrams to describe the EMP system in detail.

3.3.1 UML Diagrams

3.3.1.1 Use Case Diagram

A use case diagram in UML will showcase the system's functions, illustrating the interactions between actors (users or systems) and their objectives through use cases and dependencies. Key elements include the User class, responsible for registration and login, and the Emotion class, which handles emotion detection triggered by the Start Detection process. The system integrates webcam access and facial analysis to determine the user's mood. Music playback is managed by classes like "Control Music" which enable functions like play, stop, pause, shuffle, and playing the next track.

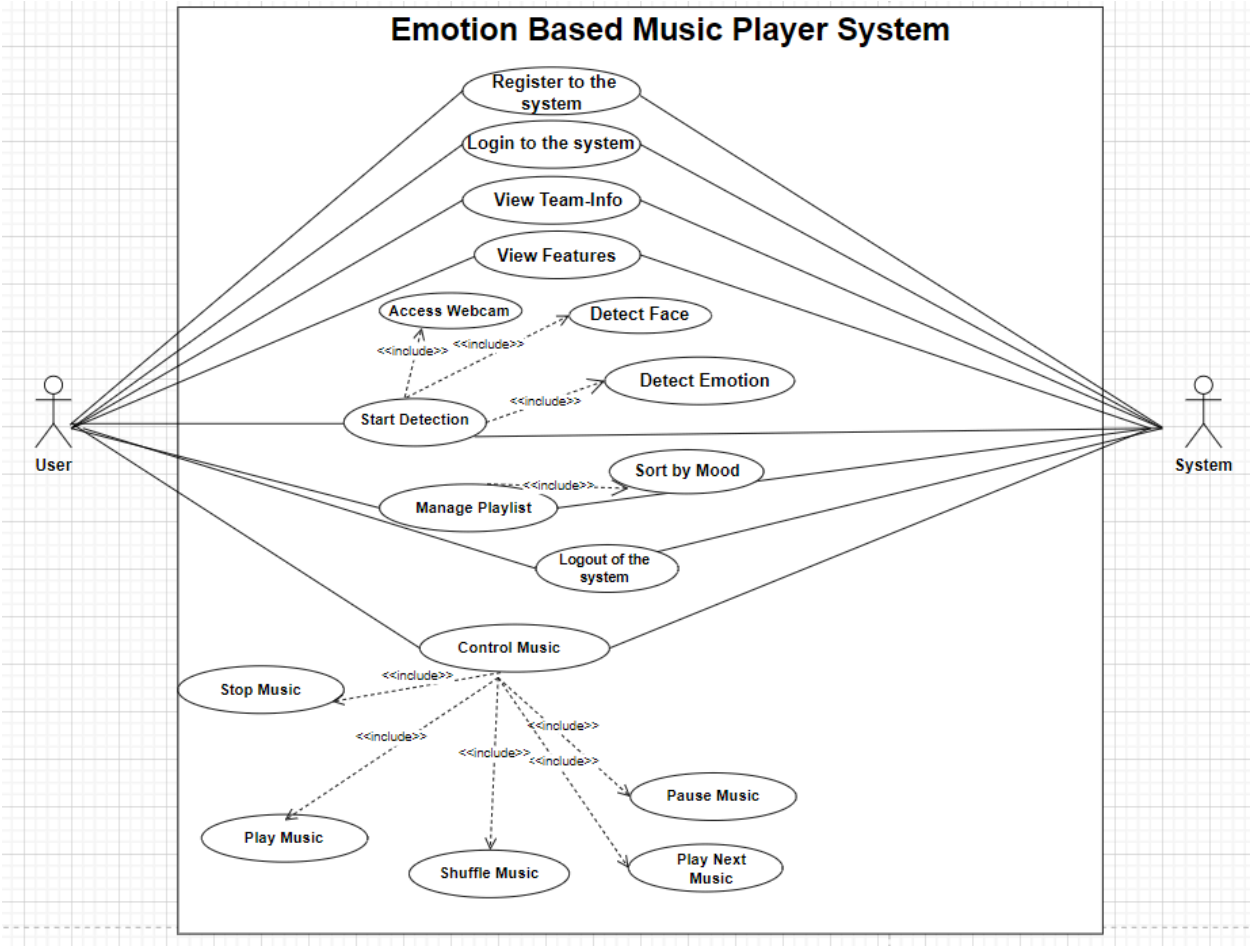

Fig 3.3: Use Case diagram of EMP System

3.3.1.2 Activity Diagram

This Fig 3.4 outlines the process of user interaction with an emotion-based music player. Initially, the system will determine whether the user is new or existing. New users will need to register before proceeding. Existing users will be presented with the option to click a start detection button. If clicked, the system will detect their emotion and select music that aligns with their current mood. Otherwise, the system will check if the user wishes to view team information; if so, this information will be displayed. Next, the system will determine if the user wants to see features. A positive response will lead to a display of current and upcoming features. If none of these actions are selected, the system will direct the user to an option for sending feedback or queries. Finally, the process will conclude with the user logging out of the system.

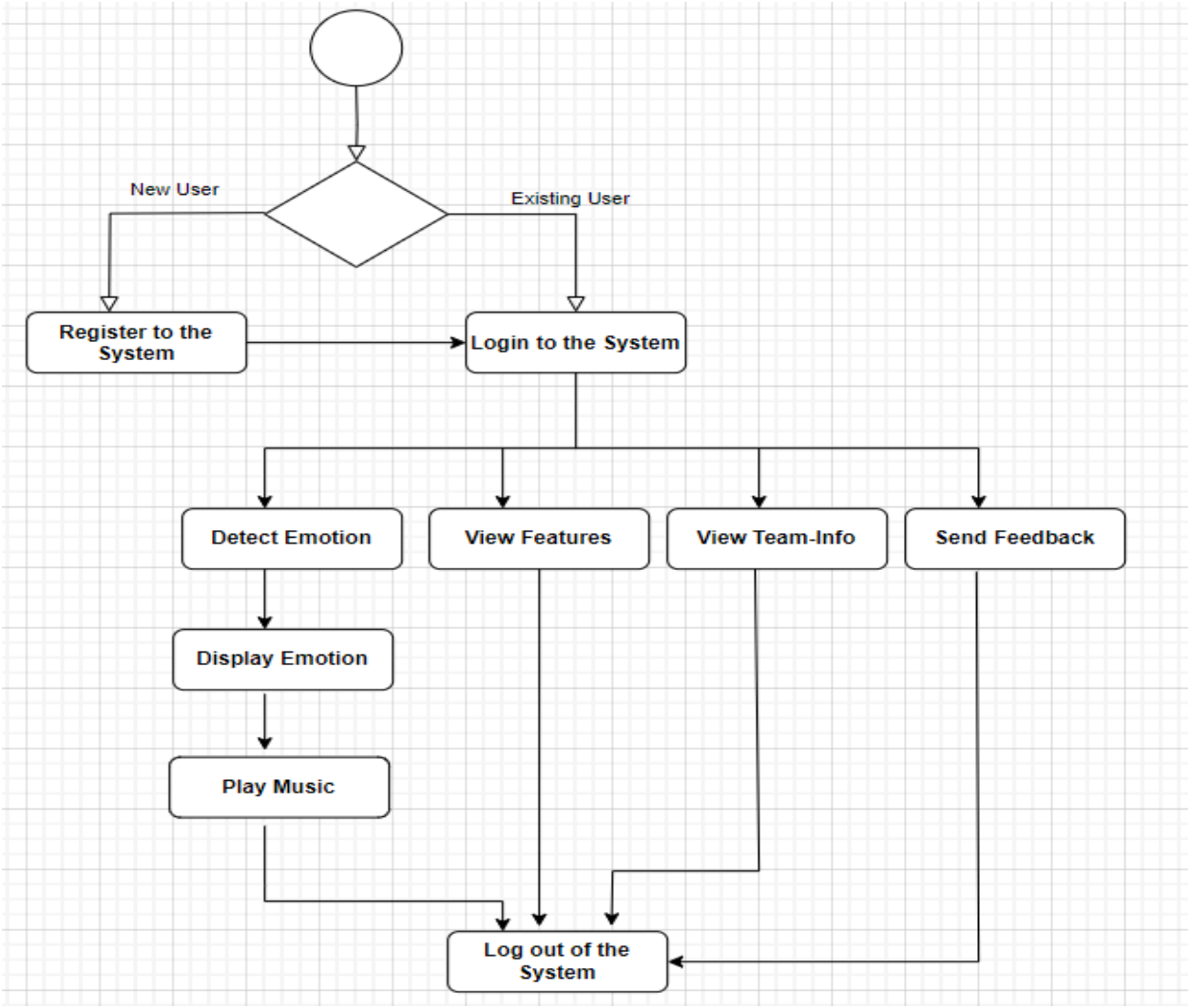

Fig 3.4: Activity diagram of EMP System

3.3.1.3 Class Diagram

A class diagram for an EMP system outlines the structure of the system, focusing on key components and their interactions. The system will likely include classes like User (login information), Emotions (handling the emotion detection process), and Music Player (managing the music library and playback). This diagram highlights relationships between these classes, such as a 1:5 relationship between User and Emotions. This diagram will serve as a blueprint for understanding how the system's components are designed to work together.

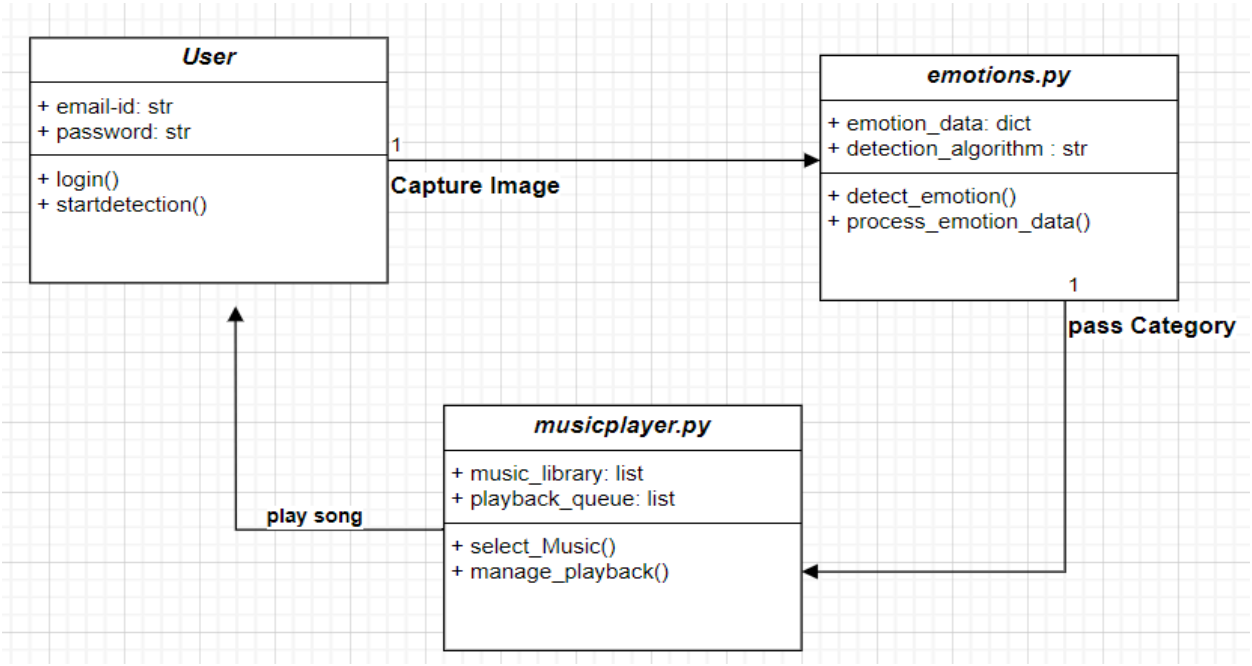

Fig 3.4 : Class diagram of EMP System

3.3.1.4 Sequence Diagram

A sequence diagram in UML depicts the order of interactions and processes in the system, showing how and when messages will be exchanged. In Fig: 3.5, the user will initiate the process by activating the detection function. The application will then turn on the webcam, which will capture video of the user's face. This video will be streamed to a machine learning model, which will analyze the video frames to detect the user's emotions. Once the emotion is determined, the model will select music that aligns with that emotion, and the music will be played through a music player.

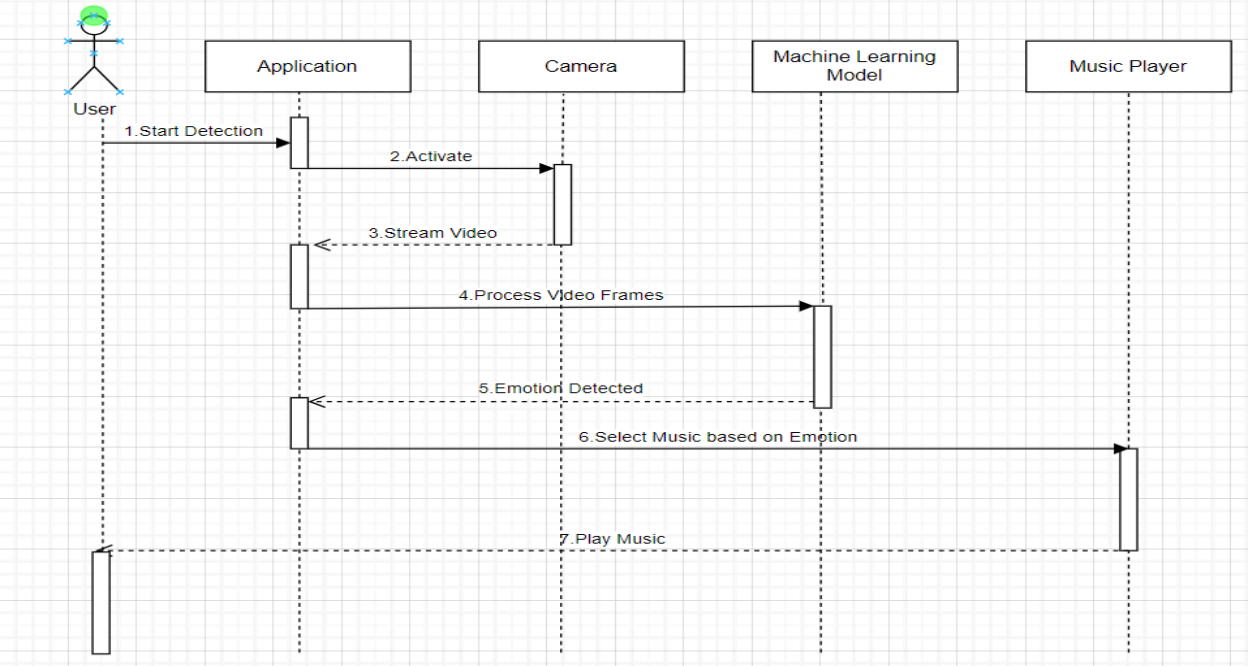

Fig 3.5 : Sequence diagram of EMP System

3.4 Implementation

3.4.1 Website Workflow for EMP System

- Starting the Application - Initialization:
    - The Flask application will be launched by executing emotions.py.
- Accessing the Web Interface - Indexpage (index.html):
    - Users will reach the indexpage typically at localhost on a web browser.
    - The homepage URL will be http://127.0.0.1:5000/.
- Register to the Website - RegisterPage (register.html):
    - Users can sign up to access the website by providing their details.
    - Once the user clicks on the submit button in the register page , it will be redirected to the home page.
    - For users who are already registered, they can navigate to the login page by clicking on the "Log in here" button displayed at the bottom of the page.
- Login to the Website - LoginPage (login.html):
    - Users can log in using their registered email ID and password.
    - Upon clicking the Login button, users will be redirected to the Home page.
    - If a user doesn't have login credentials, they can click on the "Sign up for Emotify" button to register for the website.
- HomePage (home.html):
    - Once the user is redirected to the home page, they will find a "Start Emotion Detection" button.
    - Clicking this button opens the webcam, detects emotions of the users, displays the final emotion on the web page and automatically plays songs.
- Emotion Analysis Process - Analyzing Emotion (emotions.py):
    - Users will trigger Start Detection to start the emotion detection process.
    - The system will utilize the webcam to capture real-time emotional expressions.
    - The captured data will be processed to determine the current emotional state.
- Music Playback - Playing Music (musicplayer.py):
    - Based on the detected emotion, the system automatically selects and plays a suitable song.
    - Music tracks are fetched from the songs folder, matching the emotion.
- Contacting admin - Contact Page (contact.html):
    - If users wish to contact the admin for queries or provide feedback about the website, they will be able to utilize this page to send a direct message.
    - Upon clicking the “Send Message” button, the message will be delivered to the admin via email.
- Features the Website - Features Page (features.html):
    - This page will display the features of both the current and upcoming versions in a clear and informative manner.
- Team Info of the Website - TeamInfo Page (teaminfo.html):
    - This page will showcase all the team members of the website along with their details such as name, email, and phone numbers.
    - Users will be able to use these details to contact the team members if needed.
- Closing the Application - System Shutdown:
    - The application will be closed by clicking on the Logout button.

3.4.2 Code

- Index.html - This webpage acts as the start page for the Emotify website, providing users with navigation options to log in or register, along with a welcoming message introducing them to the platform.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Emotify - Web Player: Emotion-Driven Music Player</title>
  <link rel="icon" type="image/png" href="{{ url_for('static', filename='img2.png') }}">
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
  <link
href="https://fonts.googleapis.com/css2?family=Merienda:wght@400;700&family=Poppins:w
ght@400;500;600&display=swap" rel="stylesheet">
  <style>
    .content {
      text-align: center;
    }
    .content h1{
      font-family: Merienda, sans-serif;
      font-size: 5em;
      margin-bottom: 0.5em;
    }
    .slide-left {
      animation: slideLeft 1s ease-in-out;
    }
    @keyframes slideLeft {
      0% {
        transform: translateX(-100%);
        opacity: 0;
      }
```

```
        100% {
            transform: translateX(0);
            opacity: 1;
        }
    }
  </style>
</head>
<body>
  <div class="header">
    <nav>
      <div class="logo">
        <img src="{{ url_for('static', filename='img2.png') }}" alt="">
        <span>Emotify</span>
      </div>
      <ul>
        <li><a href="/login" class="btn btn-primary mt-3">LOGIN</a></li>
        <li><a href="/register" class="btn btn-primary mt-3">REGISTER</a></li>
      </ul>
    </nav>
    <div class="content">
      <h1 class="slide-left">Welcome to Emotify, where music and emotions harmonize....!</h1>
    </div>
  </br>
 </div>
</body>
</html>
```

- Login.html – This webpage acts as a login page for Emotify, featuring a login form for user authentication and an option to sign up for new accounts.

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
  <title>Login - Emotify</title>
  <link rel="icon" type="image/png" href="{{ url_for('static', filename='img2.png') }}">
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/css/bootstrap.min.css">
  <script src="https://cdn.jsdelivr.net/npm/jquery@3.6.4/dist/jquery.slim.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"></script>
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/js/bootstrap.bundle.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
  <style>
    body {
        width: 100%;
        height: 100vh;
        background-image: url('/static/img3.png');
        background-position: center;
        background-size: cover;
        overflow-x: hidden;
        }
    nav{
        width: 100%;
        display: flex;
        justify-content: space-between;
        padding: 18px 8%;
        transition: background 1s;
    }
    nav .logo{
        display: flex;
        align-items: center;
        font-size: 26px;
        font-weight: 600;
        color: #FFF;
```

```
    }
    nav .logo img{
        width: 40px;
        margin-right: 10px;
    }
  </style>
</head>
<body>

<div class="container">
    <nav>
        <div class="logo">
            <img src="{{ url_for('static', filename='img2.png') }}" alt="">
            <span>Emotify</span>
        </div>
    </nav>
    <div class="row justify-content-center">
        <div class="col-sm-6 mt-4">
            <div class="card">
                <div class="card-body">
                    <h2 class="text-center mb-4">Log in to Emotify</h2>
                    {% if error %}
                    <div class="alert alert-danger" role="alert">
                        {{ error }}
                    </div>
                    {% endif %}
                    <form action="/login" method="POST">
                        <div class="form-group">
                            <label for="email">Email Id:</label>
                            <input type="email" class="form-control" id="email"
placeholder="Username" name="email" style="width: 100%; text-align: center;" required>
                        </div>
                        <div class="form-group">
```

```
                    <label for="pwd">Password:</label>
                    <input type="password" class="form-control" id="pwd"
placeholder="Password" name="password" style="width: 100%; text-align: center;" required>
                </div>
                <button type="submit" class="btn btn-dark btn-block">Login</button>
            </form>
        </div>
    </div>
    <div class="text-center mt-3">
        <p>Don't have an account? <a href="/register" class="btn btn-primary">Sign up for
Emotify</a></p>
    </div>
  </div>
 </div>
</div>
</body>
</html>
```

- register.html - This webpage acts as a sign-up page for Emotify, allowing users to register with their full name, email, contact number, and password. It includes a form for user input and options to navigate to the login page if they already have an account.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Sign up - Emotify</title>
  <link rel="icon" type="image/png" href="{{ url_for('static', filename='img2.png') }}">
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/css/bootstrap.min.css">
  <script src="https://cdn.jsdelivr.net/npm/jquery@3.6.4/dist/jquery.slim.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"></script>
  <script
```

```
src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/js/bootstrap.bundle.min.js"></script>
  <style>
    body {
        width: 100%;
        height: 100vh;
        background-image: url('/static/img3.png');
        background-position: center;
        background-size: cover;
        overflow-x: hidden;
        }
    nav{
        width: 100%;
        display: flex;
        justify-content: space-between;
        padding: 18px 8%;
        transition: background 1s;
    }
    nav .logo{
        display: flex;
        align-items: center;
        font-size: 26px;
        font-weight: 600;
        color: #FFF;
    }
    nav .logo img{
        width: 40px;
        margin-right: 10px;
    }
  </style>
</head>
<body>

<div class="container">
```

```
  <nav>
    <div class="logo">
      <img src="{{ url_for('static', filename='img2.png') }}" alt="">
      <span>Emotify</span>
    </div>
  </nav>
  <div class="row justify-content-center">
    <div class="col-sm-6 mt-4">
      <div class="card">
        <div class="card-body">
          <h2 class="text-center mb-4">Sign up to start listening</h2>
          {% if error %}
          <div class="alert alert-danger" role="alert">
            {{ error }}
          </div>
          {% endif %}
          <form action="/register" method="POST">
            <div class="form-group">
              <label for="name">Fullname:</label>
              <input type="text" class="form-control" id="name" placeholder="Name"
name="name" required>
            </div>
            <div class="form-group">
              <label for="email">Email Id:</label>
              <input type="email" class="form-control" id="email" placeholder="Email
Id" name="email" required>
            </div>
            <div class="form-group">
              <label for="phone">Contact No:</label>
              <input type="tel" class="form-control" id="phone" placeholder="Contact
no" name="phone" required>
            </div>
            <div class="form-group">
```

```
                    <label for="pwd">Password:</label>
                    <input type="password" class="form-control" id="pwd"
placeholder="Password" name="password" required>
                </div>
                <div class="form-group">
                    <label for="confirmpwd">Confirm Password:</label>
                    <input type="password" class="form-control" id="confirmpwd"
placeholder="Confirm Password" name="confirmpwd" required>
                </div>
                <button type="submit" class="btn btn-dark btn-block">Register</button>
            </form>
        </div>
    </div>
    <div class="text-center mt-3">
        <p>Already have an account? <a href="/login" class="btn btn-primary">Log in
here</a></p>
    </div>
  </div>
 </div>
</div>
</body>
</html>
```

- home.html - This acts as a homepage for the website and contains a button to start emotion detection using the webcam.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Music Recommendation System - Emotify</title>
  <link rel="icon" type="image/png" href="{{ url_for('static', filename='img2.png') }}">
```

```
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
    <style>
        .content2 h2{
            font-family: Merienda, sans-serif;
            font-size: 2.5em;
            margin-bottom: 0.5em;
        }
    </style>
</head>
<body>
    <div class="header">
        <nav>
            <div class="logo">
                <img src="{{ url_for('static', filename='img2.png') }}" alt="">
                <span>Emotify</span>
            </div>
            <ul>
                <li><a href="/home">HOME</a></li>
                <li><a href="/features">FEATURES</a></li>
                <li><a href="/contact">CONTACT</a></li>
                <li><a href="/team_info">TEAM-INFO</a></li>
                <li><a href="/logout" class="btn btn-primary mt-3">Logout</a></li>
            </ul>
        </nav>
        <div class="content2">
            <h2 class="custom-font">Experience Emotify: Where Music Meets Mood!</h2>
            <h2 class="custom-font">Click Start Detection below to engage with the webcam and
discover the magic of synchronized music and emotions!</h2>
                <form action="/start_detection" method="post">
                    <div style="text-align: center;">
                        <button id="startDetectionBtn" type="submit" class="start-button">Start
Emotion Detection</button>
                    </div>
```

```
        </form>
    </div>
  </br>
  </div>
 </body>
</html>
```

- Features.html - This acts as a web page for showcasing features of the Emotify application

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Features - Emotify</title>
  <link rel="icon" type="image/png" href="{{ url_for('static', filename='img2.png')
}}">
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/css/bootstrap.min.css">
  <script
src="https://cdn.jsdelivr.net/npm/jquery@3.6.4/dist/jquery.slim.min.js"></script>
  <script
src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"></script>
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/js/bootstrap.bundle.min.js"></sc
ript>
  <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
  <style>
   body {
    width: 100%;
    height: 100vh;
    background-image: url('/static/img3.png');
```

```
  background-position: center;
  background-size: cover;
  overflow-x: hidden;
     }
.card {
  max-width: 800px;
  width: 100%;
}
nav{
     width: 100%;
     display: flex;
     justify-content: space-between;
     padding: 18px 8%;
     transition: background 1s;
  }
  nav .logo{
     display: flex;
     align-items: center;
     font-size: 26px;
     font-weight: 600;
     color: #FFF;
  }
  nav .logo img{
     width: 40px;
     margin-right: 10px;
  }
  nav ul li{
    display: inline-block;
    list-style: none;
    margin: 10px 20px;
  }
  nav ul li a{
    text-decoration: none;
```

```
        color: #FFF;
        font-weight: 800;
      }
      nav:hover{
        background: #fff;
      }
      nav:hover .logo{
        color: #333;
      }
      nav:hover ul li a{
        color: #333;
      }
    </style>
</head>

  <body>
    <div class="header">
      <nav>
        <div class="logo">
          <img src="{{ url_for('static', filename='img2.png') }}" alt="">
          <span>Emotify</span>
        </div>
        <ul>
            <li><a href="/home">HOME</a></li>
            <li><a href="/features">FEATURES</a></li>
            <li><a href="/contact">CONTACT</a></li>
            <li><a href="/team_info">TEAM-INFO</a></li>
            <li><a href="/logout" class="btn btn-primary mt-3">Logout</a></li>
        </ul>
      </nav>
    </div>

<div class="container">
```

```
<div class="row justify-content-center">
  <div class="card">
    <div class="card-body">
      <h2 class="card-title text-center">Application Features</h2>
      <div id="application">
        <div class="version">
          <h3>Version 1</h3>
          <ul>
            <li>Emotion Detection for emotions like Happy, Neutral, Surprised</li>
            <li>Music Playback through VLC Media Player</li>
            <li>Contact Us</li>
            <li>About Us</li>
            <li>Team Info</li>
          </ul>
        </div>
        <hr> <!-- Horizontal line to separate versions -->
        <div class="version">
          <h3>Version 1.1</h3>
          <ul>
            <li>Enhanced Emotion Detection for emotions like Disgust, Fearful</li>
            <li>Integration with Spotify for music streaming</li>
            <li>YouTube Integration for video content</li>
            <li>User Dashboard for personalized experience</li>
          </ul>
        </div>
      </div>
    </div>
  </div>
</div>
 </body>
</html>
```

- contact.html - This is a contact page where users can send feedback or queries.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Contact Us - Emotify</title>
    <link rel="icon" type="image/png" href="{{ url_for('static', filename='img2.png')
}}">
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/css/bootstrap.min.css">
    <script
src="https://cdn.jsdelivr.net/npm/jquery@3.6.4/dist/jquery.slim.min.js"></script>
    <script
src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"></script>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/js/bootstrap.bundle.min.js"></sc
ript>
    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
    <style>
        body {
                width: 100%;
                height: 100vh;
                background-image: url('/static/img3.png');
                background-position: center;
                background-size: cover;
                overflow-x: hidden;
            }
        nav{
            width: 100%;
            display: flex;
            justify-content: space-between;
```

```
    padding: 18px 8%;
    transition: background 1s;
}
nav .logo{
    display: flex;
    align-items: center;
    font-size: 26px;
    font-weight: 600;
    color: #FFF;
}
nav .logo img{
    width: 40px;
    margin-right: 10px;
}
nav ul li{
    display: inline-block;
    list-style: none;
    margin: 10px 20px;
}
nav ul li a{
    text-decoration: none;
    color: #FFF;
    font-weight: 800;
}
nav:hover{
    background: #fff;
}
nav:hover .logo{
    color: #333;
}
nav:hover ul li a{
    color: #333;
}
```

```
</style>
</head>
<body>
  <div class="header">
    <nav>
      <div class="logo">
        <img src="{{ url_for('static', filename='img2.png') }}" alt="">
        <span>Emotify</span>
      </div>
      <ul>
        <li><a href="/home">HOME</a></li>
        <li><a href="/features">FEATURES</a></li>
        <li><a href="/contact">CONTACT</a></li>
        <li><a href="/team_info">TEAM-INFO</a></li>
        <li><a href="/logout" class="btn btn-primary mt-3">Logout</a></li>
      </ul>
    </nav>
  </div>

  <div class="container">
    <div class="row justify-content-center">
      <div class="col-sm-6 mt-4">
        <div class="card">
          <div class="card-body">
            <h1 class="card-title">Contact Us</h1>
            {% if error %}
            <div class="alert alert-danger" role="alert">
              {{ error }}
            </div>
            {% endif %}
            <form action="/submit_contact_form" method="POST">
              <div class="form-group">
                <label for="name">Name:</label>
```

```
                    <input type="text" id="name" name="name" placeholder="Name"
class="form-control" required>
                </div>
                <div class="form-group">
                    <label for="email">Email:</label>
                    <input type="email" id="email" name="email"
placeholder="Email" class="form-control" required>
                </div>
                <div class="form-group">
                    <label for="message">Your Message or Feedback</label>
                    <textarea id="message" name="message" placeholder="Message"
class="form-control" rows="4" required></textarea>
                </div>
                <button type="submit" class="btn btn-primary">Send
Message</button>
            </form>
            {% with messages = get_flashed_messages() %}
            {% if messages %}
            <div class="flash-messages">
                {% for message in messages %}
                <div class="alert">{{ message }}</div>
                {% endfor %}
            </div>
            {% endif %}
            {% endwith %}
        </div>
    </div>
</div>
</div>
<script>
    window.onload = function() {
        setTimeout(function() {
```

```
            var messages = document.querySelector('.flash-messages');
            if (messages) {
                messages.style.display = 'none';
            }
        }, 5000);  // 5000 milliseconds = 5 seconds
    };
  </script>
</body></html>
```

- team-info.html - This is a team-info page where users can see team members names, email addresses, and contact numbers along with their respective images.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Team Info - Emotify</title>
  <link rel="icon" type="image/png" href="{{ url_for('static', filename='img2.png')
}}">
  <link rel="stylesheet"
href="https://use.fontawesome.com/releases/v5.8.1/css/all.css">
  <link rel="stylesheet" href="/static/team_info.css">
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
  <style>
    .content1 h1{
      font-family: Merienda, sans-serif;
      font-size: 2.5em;
      margin-bottom: 0.5em;
    }
  </style>
</head>
<body>
<div class="header">
    <nav>
```

```
    <div class="logo">
      <img src="{{ url_for('static', filename='img2.png') }}" alt="">
      <span>Emotify</span>
    </div>
    <ul>
      <li><a href="/home">HOME</a></li>
      <li><a href="/features">FEATURES</a></li>
      <li><a href="/contact">CONTACT</a></li>
      <li><a href="/team_info">TEAM-INFO</a></li>
      <li><a href="/logout" class="btn btn-primary mt-3">Logout</a></li>
    </ul>
  </nav>
  <div class="content1">
    <h1>Emotify - Team</h1>
    <div class="our_team">
      <div class="team_member">
      <div class="member_img">
        <img src="/static/woman.png" alt="our_team">
      </div>
      <h3>Vaishnavi</h3>
      <span><h4>chakkav1@montclair.edu</h4></span>
      <span>+1 2486881702</span>
      </div>
      <div class="team_member">
        <div class="member_img">
        <img src="/static/woman.png" alt="our_team">
        </div>
        <h3>Nimisha</h3>
        <span><h4>bonigalan1@montclair.edu</h4></span>
        <span>+1 5512603359</span>
      </div>
      <div class="team_member">
        <div class="member_img">
```

```
                <img src="/static/man1.png" alt="our_team">
                </div>
                <h3>Ajay</h3>
                <span><h4>Morua1@montclair.edu</h4></span>
                <span>+1 2016794519</span>
            </div>
            <div class="team_member">
                <div class="member_img">
                <img src="/static/man1.png" alt="our_team">
                </div>
                <h3>Manikanta</h3>
                <span><h4>vemavarapup1@montclair.edu</h4></span>
                <span>+1 2012842493</span>
            </div>
        </div>
    </div>
</body>
</html>
```

- Display.html - This page displays the final emotion of the user

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Music Recommendation System - Emotify</title>
    <link rel="icon" type="image/png" href="{{ url_for('static', filename='img2.png')
}}">
    <link rel="stylesheet" href="/static/button_style.css">
    <link rel="stylesheet" href="/static/style.css">
</head>
```

```
<body>
  <div class="header">
    <nav>
      <div class="logo">
        <img src="{{ url_for('static', filename='img2.png') }}" alt="">
        <span>Emotify</span>
      </div>
      <ul>
        <li><a href="/home">HOME</a></li>
        <li><a href="/features">FEATURES</a></li>
        <li><a href="/contact">CONTACT</a></li>
        <li><a href="/team_info">TEAM-INFO</a></li>
        <li><a href="/logout" class="btn btn-primary mt-3">Logout</a></li>
      </ul>
    </nav>
    <div class="content">
        <div class="output" >
          {% if final_output %}
            <h2 style="font-family: Bio-Rhyme; color: white; margin: 40px; font:
40px; font-size: 70px">You look {{ final_output }}</h2>
            <h2 style="font-family: Bio-Rhyme; color: white; margin: 40px; font:
40px; font-size: 70px">Embrace the moment - Music will play shortly</h2>
          {% endif %}
        </div>
    </div>
  </div>
<script>
    document.addEventListener('DOMContentLoaded', function() {
      var delay = 2000;
      var final_output = "{{ final_output }}";

      setTimeout(function() {
        if (final_output.trim() === "") {
```

```
                var outputDiv = document.querySelector('.output');
                outputDiv.innerHTML = "<h2 style='font-family:Bio-Rhyme; color:
white; margin:40px; font:40px; font-size:70px'>Oops! Unable to detect mood.</h2>";
            } else {
                // If final_output is not empty, redirect to /music
                window.location.href = '/music?final_output=' +
encodeURIComponent(final_output);
            }
        }, delay);
    });
  </script>
</body>
</html>
```

- dataset_prepare.py - This python class prepares the data set required for training an emotion detection model.

```
import numpy as np
import pandas as pd
from PIL import Image
from tqdm import tqdm
import os

# convert string to integer
def atoi(s):
    n = 0
    for i in s:
        n = n*10 + ord(i) - ord("0")
    return n

# making folders
outer_names = ['test','train']
inner_names = ['angry', 'disgusted', 'fearful', 'happy', 'sad', 'surprised', 'neutral']
os.makedirs('data', exist_ok=True)
```

```
for outer_name in outer_names:
    os.makedirs(os.path.join('data',outer_name), exist_ok=True)
    for inner_name in inner_names:
        os.makedirs(os.path.join('data',outer_name,inner_name), exist_ok=True)

# to keep count of each category
angry = 0
disgusted = 0
fearful = 0
happy = 0
sad = 0
surprised = 0
neutral = 0
angry_test = 0
disgusted_test = 0
fearful_test = 0
happy_test = 0
sad_test = 0
surprised_test = 0
neutral_test = 0

df = pd.read_csv('./fer2013.csv')
mat = np.zeros((48,48),dtype=np.uint8)
print("Saving images...")

# read the csv file line by line
for i in tqdm(range(len(df))):
    txt = df['pixels'][i]
    words = txt.split()

    # the image size is 48x48
    for j in range(2304):
        xind = j // 48
```

```
    yind = j % 48
    mat[xind][yind] = atoi(words[j])

img = Image.fromarray(mat)

# train
if i < 28709:
    if df['emotion'][i] == 0:
        img.save('data/train/angry/im'+str(angry)+'.png')
        angry += 1
    elif df['emotion'][i] == 1:
        img.save('data/train/disgusted/im'+str(disgusted)+'.png')
        disgusted += 1
    elif df['emotion'][i] == 2:
        img.save('data/train/fearful/im'+str(fearful)+'.png')
        fearful += 1
    elif df['emotion'][i] == 3:
        img.save('data/train/happy/im'+str(happy)+'.png')
        happy += 1
    elif df['emotion'][i] == 4:
        img.save('data/train/sad/im'+str(sad)+'.png')
        sad += 1
    elif df['emotion'][i] == 5:
        img.save('data/train/surprised/im'+str(surprised)+'.png')
        surprised += 1
    elif df['emotion'][i] == 6:
        img.save('data/train/neutral/im'+str(neutral)+'.png')
        neutral += 1

# test
else:
    if df['emotion'][i] == 0:
        img.save('data/test/angry/im'+str(angry_test)+'.png')
```

```
            angry_test += 1
        elif df['emotion'][i] == 1:
            img.save('data/test/disgusted/im'+str(disgusted_test)+'.png')
            disgusted_test += 1
        elif df['emotion'][i] == 2:
            img.save('data/test/fearful/im'+str(fearful_test)+'.png')
            fearful_test += 1
        elif df['emotion'][i] == 3:
            img.save('data/test/happy/im'+str(happy_test)+'.png')
            happy_test += 1
        elif df['emotion'][i] == 4:
            img.save('data/test/sad/im'+str(sad_test)+'.png')
            sad_test += 1
        elif df['emotion'][i] == 5:
            img.save('data/test/surprised/im'+str(surprised_test)+'.png')
            surprised_test += 1
        elif df['emotion'][i] == 6:
            img.save('data/test/neutral/im'+str(neutral_test)+'.png')
            neutral_test += 1

print("Done!")
```

- mode_train.py - This python script is responsible for training a machine learning or deep learning model. Its main purpose is to take a dataset, preprocess it, define a model architecture, train the model on the training data, and evaluate its performance on a separate validation or test dataset

```
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
def plot_model_history(model_history):
    """
    Plot Accuracy and Loss curves given the model_history
    """
    fig, axs = plt.subplots(1, 2, figsize=(15, 5))
    # summarize history for accuracy
    axs[0].plot(range(1, len(model_history.history['accuracy']) + 1),
model_history.history['accuracy'])
    axs[0].plot(range(1, len(model_history.history['val_accuracy']) + 1),
model_history.history['val_accuracy'])
    axs[0].set_title('Model Accuracy')
    axs[0].set_ylabel('Accuracy')
    axs[0].set_xlabel('Epoch')
    axs[0].set_xticks(np.arange(1, len(model_history.history['accuracy']) + 1),
len(model_history.history['accuracy']) / 10)
    axs[0].legend(['train', 'val'], loc='best')
    # summarize history for loss
    axs[1].plot(range(1, len(model_history.history['loss']) + 1),
model_history.history['loss'])
    axs[1].plot(range(1, len(model_history.history['val_loss']) + 1),
model_history.history['val_loss'])
    axs[1].set_title('Model Loss')
    axs[1].set_ylabel('Loss')
    axs[1].set_xlabel('Epoch')
    axs[1].set_xticks(np.arange(1, len(model_history.history['loss']) + 1),
len(model_history.history['loss']) / 10)
    axs[1].legend(['train', 'val'], loc='best')
    fig.savefig('plot.png')
    plt.show()

# Define data generators
train_dir = 'data/train'
val_dir = 'data/test'
```

```
num_train = 28709
num_val = 7178
batch_size = 64
num_epoch = 80

train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(48, 48),
    batch_size=batch_size,
    color_mode="grayscale",
    class_mode='categorical')

validation_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(48, 48),
    batch_size=batch_size,
    color_mode="grayscale",
    class_mode='categorical')

# Create the model
model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(48, 48, 1)),
    Conv2D(64, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),
    Conv2D(128, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(128, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),
```

```
    Flatten(),
    Dense(1024, activation='relu'),
    Dropout(0.5),
    Dense(6, activation='softmax')
])

model.compile(loss='categorical_crossentropy',
optimizer=Adam(learning_rate=0.0001, decay=1e-6), metrics=['accuracy'])

model_info = model.fit(
    train_generator,
    steps_per_epoch=num_train // batch_size,
    epochs=num_epoch,
    validation_data=validation_generator,
    validation_steps=num_val // batch_size)

model.save('model_complete1.h5')
print(model.summary())
```

- emotion.py - In this class, the emotion detection logic captures video from the webcam, detects faces using a Haar cascade classifier, predicts emotions for each detected face using a pre-trained convolutional neural network (CNN) model, and displays the detected emotions on the video feed in real-time.

```
from flask import Flask, render_template, request,flash, redirect,
url_for, session
from flask_mail import Mail, Message as FlaskMessage
import numpy as np
import cv2
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
import os
import vlc
import time
```

```
from pathlib import Path
from random import randint
from subprocess import call
from tkinter import *
import statistics as st
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.exc import SQLAlchemyError
import bcrypt
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '4'

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '4'

app = Flask(__name__, template_folder='templates',
static_folder='static')
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
db = SQLAlchemy(app)
app.config['MAIL_SERVER'] = 'smtp.gmail.com'
app.config['MAIL_PORT'] = 587
app.config['MAIL_USERNAME'] = 'vyshnavi.kantheti97@gmail.com'
app.config['MAIL_PASSWORD'] = 'kpaz zbbt ajcv ekbj'
app.config['MAIL_USE_TLS'] = True
app.config['MAIL_USE_SSL'] = False
mail = Mail(app)

app.secret_key = '12345678'
text = ""

def music_player(emotion_str):
    from musicplayer import MusicPlayer
    root = Tk()
    print('\nPlaying ' + emotion_str + ' songs')
    MusicPlayer(root,emotion_str)
    root.mainloop()

@app.route('/music')
def music():
    text = request.args.get('final_output')
    music_player(text)
    return redirect('/home')
```

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=(48,48,1)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(6, activation='softmax'))
model.load_weights('model_complete1.h5')

emotion_dict = {0: "Angry", 1: "Disgusted", 2: "Happy", 3:
"Neutral", 4: "Sad", 5: "Surprised"}

def display_info(frame, timer_text, emotion_text):
    print(emotion_text)
    if not emotion_text or emotion_text not in
emotion_dict.values():
        error_text = "Emotion not recognized. Please clarify."
        cv2.putText(frame, error_text, (10, 60),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv2.LINE_AA)
    cv2.putText(frame, timer_text, (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
    return frame

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(100), unique=True)
    password = db.Column(db.String(100))
    phone = db.Column(db.String(20))
```

```
    def __init__(self,email,password,name,phone):
        self.name = name
        self.email = email
        self.password = bcrypt.hashpw(password.encode('utf-8'),
bcrypt.gensalt()).decode('utf-8')
        self.phone = phone

    def check_password(self,password):
        return
bcrypt.checkpw(password.encode('utf-8'),self.password.encode('utf-
8'))

with app.app_context():
    db.create_all()

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/home')
def index1():
    return render_template('home.html')

@app.route('/team_info')
def team_info():
    return render_template('team_info.html')

@app.route('/login',methods=['GET','POST'])
def login():
    if request.method == 'POST':
     try:
        email = request.form['email']
        password = request.form['password']
        user = User.query.filter_by(email=email).first()
        if user and user.check_password(password):
            session['email'] = user.email
            return redirect('/home')
        elif user is None:
            error_msg = 'Email ID is not registered. Please use a
```

```
registered email ID'
        elif not is_valid_password(password):
            error_msg = 'Password should be at least 8 characters
long and include at least one uppercase letter, one lowercase
letter, one digit, and one special character.'
        else:
            error_msg = 'Incorrect username or password'
        return render_template('login.html',error=error_msg)
     except SQLAlchemyError as e:
            return render_template('login.html',
error_message='Database error: ' + str(e))
     except Exception as e:
            return render_template('login.html', error_message='An
unexpected error occurred: ' + str(e))
    return render_template('login.html',
error=session.pop('error', None))

@app.route('/register',methods=['GET','POST'])
def register():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        phone = request.form['phone']
        password = request.form['password']
        confirmpwd = request.form['confirmpwd']
        if User.query.filter_by(email=email).first():
            return render_template('register.html', error='This
email id is already registered. Please log in or use a different
email id to register.')
        elif User.query.filter_by(phone=phone).first():
            return render_template('register.html', error='This
phone number is already registered. Please use a different phone
number to register.')
        elif not phone.isdigit():
           return render_template('register.html', error='Mobile
numbers must be numeric.')
        elif len(phone) != 10:
           return render_template('register.html', error='Kindly
enter a 10 digit mobile number.')
```

```
        elif not is_valid_password(password):
           return render_template('register.html',error='Password
should be at least 8 characters long and include at least one
uppercase letter, one lowercase letter, one digit, and one special
character.')
        elif password != confirmpwd:
            return render_template('register.html',error='Password
and confirm password are not matching.')

        new_user =
User(name=name,email=email,phone=phone,password=password)
        db.session.add(new_user)
        db.session.commit()
        return redirect('/home')
    return render_template('register.html')

def is_valid_password(password):
    if len(password) < 8:
        return False
    if not any(char.isupper() for char in password):
        return False
    if not any(char.islower() for char in password):
        return False
    if not any(char.isdigit() for char in password):
        return False
    if not any(char in '!@#$%^&*()_+-=[]{}|;:,.<>?/' for char in
password):
        return False

    return True

@app.route('/contact')
def contact():
    return render_template('contact.html')

@app.route('/submit_contact_form', methods=['GET', 'POST'])
def submit_contact_form():
    if request.method == 'POST':
        msg = FlaskMessage("Query",
```

```
sender='vyshnavi.kantheti97@gmail.com',

recipients=['vyshnavi.kantheti97@gmail.com'])
        msg.body = request.form['message']
        email = request.form['email']

        # Check if the email exists in the User table
        user = User.query.filter_by(email=email).first()

        if user:
            # Email exists, we can proceed with sending the email
            mail.send(msg)
            flash('Message sent successfully!')
            return redirect(url_for('contact'))
        else:
            # Email does not exist in the User table
            return render_template('contact.html', error='Email
not registered. Please enter a valid email.')

    # No message on GET request, only shown after POST
    return render_template('contact.html')

@app.route('/logout')
def logout():
    session.pop('email',None)
    return redirect('/')

@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/features')
def application():
    return render_template('features.html')

@app.route('/start_detection', methods = ['GET','POST'])
def start_detection():
    global text, final_emotion
    cap = cv2.VideoCapture(0)
```

```
    now = time.time()
    future = now + 10  # Detect emotions for 10 seconds
    emotion_file = open(str(Path.cwd()) + r"\emotion.txt", "w")
    webcam_on = True

    while webcam_on:
        ret, frame = cap.read()
        if not ret:
            break
        facecasc =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = facecasc.detectMultiScale(gray,scaleFactor=1.3,
minNeighbors=5)

        for (x, y, w, h) in faces:
            cv2.rectangle(frame, (x, y-50), (x+w, y+h+10), (255,
0, 0), 2)
            roi_gray = gray[y:y + h, x:x + w]
            cropped_img =
np.expand_dims(np.expand_dims(cv2.resize(roi_gray, (48, 48)), -1),
0)
            prediction = model.predict(cropped_img)
            maxindex = int(np.argmax(prediction))
            cv2.putText(frame, emotion_dict[maxindex], (x+20,
y-60), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2,
cv2.LINE_AA)
            text = emotion_dict[maxindex]
            emotion_file.write(emotion_dict[maxindex]+"\n")
            emotion_file.flush()

        remaining_time = max(0, int(future - time.time()))
        timer_text = f"Time left: {remaining_time} sec"
        frame = display_info(frame, timer_text, text)

        cv2.imshow('Video',
cv2.resize(frame,(1600,960),interpolation = cv2.INTER_CUBIC))
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
```

```
        if time.time() >= future:  ##after 10 second music will
play
            cv2.destroyAllWindows()
            webcam_on = False
            future = time.time() + 10

    cap.release()
    emotion_file.close()
    cv2.destroyAllWindows()
    print(text)
    return render_template("buttons.html", final_output=text)
```

- musicplayer.py - The MusicPlayer class creates a simple Tkinter-based music player with controls for playback and a playlist display. The system integrates the VLC media player for handling audio playback and provides basic user interaction for controlling music playback.

```
from tkinter import *
import os
import sys
import vlc
from pathlib import Path
import random
from tkinter import messagebox

Instance = vlc.Instance()
player = Instance.media_player_new()

class MusicPlayer(object):
    def __init__(self,root,emotionStr):
        self.root = root
        self.root.title("Music Player")
        self.root.geometry("1000x200+200+200")
        self.track = StringVar()
        self.status = StringVar()

        trackframe = LabelFrame(self.root,text="Song
Track",font=("times new
```

```
roman",15,"bold"),bg="grey",fg="white",bd=5,relief=GROOVE)
        trackframe.place(x=0,y=0,width=620,height=100)

        songtrack =
Label(trackframe,textvariable=self.track,width=20,font=("times new
roman",24,"bold"),bg="grey",fg="gold").grid(row=0,column=0,padx=10
,pady=5)

        trackstatus =
Label(trackframe,textvariable=self.status,font=("times new
roman",18,"bold"),bg="grey",fg="gold").grid(row=0,column=1,padx=5,
pady=5)

        buttonframe = LabelFrame(self.root,text="Control
Panel",font=("times new
roman",15,"bold"),bg="grey",fg="white",bd=5,relief=GROOVE)
        buttonframe.place(x=0,y=100,width=620,height=100)

        playbtn =
Button(buttonframe,text="PLAY",command=self.playsong,width=6,heigh
t=1,font=("times new
roman",16,"bold"),fg="navyblue",bg="gold").grid(row=0,column=0,pad
x=10,pady=5)

        playbtn =
Button(buttonframe,text="PAUSE",command=self.pausesong,width=8,hei
ght=1,font=("times new
roman",16,"bold"),fg="navyblue",bg="gold").grid(row=0,column=1,pad
x=10,pady=5)

        playbtn =
Button(buttonframe,text="SHUFFLE",command=self.shufflesong,width=1
0,height=1,font=("times new
roman",16,"bold"),fg="navyblue",bg="gold").grid(row=0,column=2,pad
x=10,pady=5)

        playbtn =
Button(buttonframe,text="STOP",command=self.stopsong,width=6,heigh
t=1,font=("times new
```

```
roman",16,"bold"),fg="navyblue",bg="gold").grid(row=0,column=3,pad
x=10,pady=5)

        playbtn =
Button(buttonframe,text="NEXT",command=self.nextsong,width=6,heigh
t=1,font=("times new
roman",16,"bold"),fg="navyblue",bg="gold").grid(row=0,column=4,pad
x=10,pady=5)

        songsframe = LabelFrame(self.root,text="Song
Playlist",font=("times new
roman",15,"bold"),bg="grey",fg="white",bd=5,relief=GROOVE)
        songsframe.place(x=600,y=0,width=400,height=200)

        scrol_y = Scrollbar(songsframe,orient=VERTICAL)

        self.playlist =
Listbox(songsframe,yscrollcommand=scrol_y.set,selectbackground="go
ld",selectmode=SINGLE,font=("times new
roman",12,"bold"),bg="silver",fg="navyblue",bd=5,relief=GROOVE)

        scrol_y.pack(side=RIGHT,fill=Y)
        scrol_y.config(command=self.playlist.yview)
        self.playlist.pack(fill=BOTH)


os.chdir(str(Path(__file__).parent.absolute())+"\songs\\"+emotionS
tr+"\\")

        songtracks = os.listdir()
        songtracks = [track for track in songtracks if
track.endswith('.mp3')]

        self.songtracks = songtracks

        for track in songtracks:
            self.playlist.insert(END,track)

        if not songtracks:
```

```
            messagebox.showerror("Error", "No MP3 files found in
the specified directory.")
            self.root.destroy()
        elif(player.is_playing() == 0):
            ranSong = random.choice(self.songtracks)
            self.pos = self.songtracks.index(ranSong)
            self.track.set(ranSong)
            self.status.set("-Playing "+emotionStr)
            Media = Instance.media_new(ranSong)
            player.set_media(Media)
            player.play()

    def playsong(self):
        self.track.set(self.playlist.get(ACTIVE))
        self.status.set("-Playing")
        Media = Instance.media_new(self.playlist.get(ACTIVE))
        player.set_media(Media)
        player.play()

    def stopsong(self):
        self.status.set("-Stopped")
        player.stop()
        self.root.destroy()
        os.chdir(str(Path(__file__).parent.absolute()))

    def pausesong(self):
        self.status.set("-Paused")
        player.pause()

    def nextsong(self):
        i=0
        while i< len(self.songtracks):
            if i == self.pos:
                i = i+1
                if i >= len(self.songtracks):
                    i= 0
                nsong = self.songtracks[i]
                self.pos = i
            i = i + 1
```

```
        player.stop()
        self.track.set(nsong)
        Media = Instance.media_new(nsong)
        player.set_media(Media)
        player.play()

    def shufflesong(self):
        self.status.set("-Shuffle Play")
        song2 =random.choice(self.songtracks)
        self
```

4. Testing Document

4.1 Introduction

This document will detail the procedures for testing the EMP System. The tests will aim to verify that the system operates properly, accurately recognizes emotions, and delivers a smooth music listening experience.

4.2 Purpose

The testing document for the EMP System will establish a structured approach to evaluate its performance, accuracy, functionality, reliability, and assess Integration. It will serve as a manual for evaluating the system's performance and functionality, aiming to provide users with a top-notch experience and achieve the system's intended objectives.

4.3 Objective and Success Criteria

A comprehensive test plan will involve defining clear objectives and success criteria to ensure the system's functionality, accuracy, and reliability.

- System Functionality:
  - Objective: The EMP System will ensure accurate detection and processing of emotional data for personalized music recommendations.
  - Success Criteria:
    - It will effectively process a wide range of emotions to suggest music that aligns with the user's mood or feeling.

- Accuracy:
    - Objective: The EMP System will evaluate its accuracy in detecting and matching emotions with appropriate music recommendations.
    - Success Criteria:
        - The system must achieve a predefined accuracy rate in emotion detection, such as 90% or higher.
        - False positives (suggesting inappropriate music) and false negatives (failing to suggest suitable music) will be minimized to acceptable levels.
- Performance:
    - Objective: The EMP System will evaluate its performance in terms of speed and response time.
    - Success Criteria:
        - The system will process emotion detection and music recommendations within a specified time frame.
- Security:
    - Objective: The EMP System will ensure its security and resistance to unauthorized access.
    - Success Criteria:
        - The system will have robust measures in place to prevent unauthorized access attempts, such as implementing strong authentication mechanisms and encryption protocols.
- Usability:
    - Objective: The EMP System will assess its user-friendliness for both administrators and users.
    - Success Criteria:
        - The system will feature an intuitive user interface for easy configuration and management.
        - Users will find the music selection process straightforward and enjoyable, enhancing their overall experience.
- Reliability:
    - Objective: The EMP System will ensure its reliability and availability when needed.
    - Success Criteria:
        - The system's ability to recover from failures or interruptions will be evaluated and optimized for seamless performance.

4.4 Testing Methodologies

In developing the EMP System, a comprehensive testing strategy will be essential to ensure its reliability and accuracy. The testing phase will be designed to rigorously evaluate every component of the application, from the backend logic handling emotion detection to the front-end user interface interactions.
The test plan will be structured to cover different aspects of the system:

4.4.1 Unit Testing:

In this testing, each unit will be tested separately or individually to determine its performance when checked alone. Any faults found in each unit will be corrected in this phase.

4.4.2 Integration testing:

This testing will involve combining two or more modules and testing their performance. When these modules are integrated, the interactions between them are identified. If the modules do not perform well during this phase, they will be tested individually again, modified, and then integrated once more.

4.4.3 System testing:

This testing will be the last level of testing, where all modules in the system will be assembled and tested for any errors and ambiguities. If the system performs well, it will be further processed, and the system will be approved. Otherwise, the testing phase will be repeated until the system performs correctly.

4.5 Schedules

- Test Planning Phase:
    - Team will define objectives, scope, and success criteria - 1 week.
    - Team will assign themselves roles - 1 week.
- Test Design Phase:
    - Team will develop test scenarios and test cases - 2 weeks.
    - Team will review and finalize test design with the team - 1 week.
- Test Execution Phase:
    - Team will execute unit testing - 1 week.
    - Team will execute integration tests - 1 week.
    - Team will execute system testing- 1 week.
- Defect Resolution and Re-Testing:
    - Team will address and resolve defects - Done.
    - Team will retest fixed issues - As needed.
- Documentation and Reporting:
    - Team will document test results and findings - Done.
    - Team will generate test summary reports - 3 Days.
- Training and Deployment:
    - Team will conduct training sessions for administrators and end-users - 2 weeks.
    - Team will coordinate the deployment of the EMP System - 1 week.

- Monitoring and Maintenance:
    - Team will implement monitoring tools and processes - Done.
    - Team will address any post-deployment issues - As needed.

4.6 Test Report

4.6.1 Table: Modules with sample input and output

Module Name	Input	Output
Registration	The Users will register themselves with their email id and password.	The users will get their login credentials so that they can login afterwards with those credentials
Login	The user will login with the login credentials.	The system will bring them to the home screen.
Features	The user can view the current and upcoming version features of the EMP system	The current and upcoming version features of the system are displayed on the screen
Contact	The user can contact with admin through this page	The query or feedback will be sent to the admin through an email
Team-info	The user can view the contact details of all the team members of the website	Name and Contact details are displayed on the screen
Logout	The user will request to logout by clicking on the logout button.	The system will logout.

4.6.2 Table: Modules with objective and testing criteria

Test Number	Test Objective	Completion criteria	Date	Test priority	Team Member Testing
1.	Test the functionality of the Login page.	Allowed the user to input their login information and verified it with the database.	04/23/2024	High	Vaishnavi
2.	Test the connection between the database and the Login Page.	After a successful login, the user is redirected to the home page and cannot go back to the login page without logging out.	04/23/2024	Average	Vaishnavi, Nimisha
3.	Test if after login each user is confined within a session.	Ensure that the user is logged in to switch between pages, and redirect the user to the login page upon logging out or being inactive for too long.	04/23/2024	Average	Vaishnavi, Nimisha
4.	Test the connection between “Start Detection” button and the webcam	After clicking on “Start Detection” button, the webcam is activated	04/23/2024	Average	Vaishnavi
5.	Test the functionality of webcam	Emotions are detected	04/23/2024	High	Vaishnavi, Ajay
6.	Test the functionality of “Music Player”	Play songs aligning with the user's emotions.	04/23/2024	High	Vaishnavi, Ajay
7.	Test the accuracy of Emotion Detection	The user’s face is detected with 95% accuracy	04/23/2024	Average	Vaishnavi, Manikanta

8.	Test the usability of the system	The system is easy to use for any user	04/23/2024	Average	Vaishnavi, Manikanta
9.	Test logging out	Ensure the user can logout, and not go back into the system without signing back in.	04/23/2024	High	Vaishnavi, Manikanta

4.6.3 Table: Module Testing

4.6.3.1 Table: User Registration

Serial No.	Condition to be tested	Test Data	Expected output	Remarks
1.	If name is not entered	Full Name	Please fill out this field.	SUCCESSFUL
2.	If the mobile number is not entered.	Mobile Number	Please fill out this field.	SUCCESSFUL
3.	If the mobile number is not equal to 10 digits.	Mobile Number	Kindly enter a 10 digit mobile number.	SUCCESSFUL
4.	If the mobile number contains other than numeric values.	Mobile Number	Mobile numbers must be numeric.	SUCCESSFUL
5.	If the mobile number is already registered	Mobile Number	This phone number is already registered. Please use a different phone number to register.	SUCCESSFUL
6.	If Email ID is not entered.	Email ID	Please fill out this field.	SUCCESSFUL
7.	If Email ID is invalid.	Email ID	Please include an '@' in the email address. Email ID is missing an '@'	SUCCESSFUL

8.	If the email id is already registered	Email ID	This email id is already registered. Please log in or use a different email id to register.	SUCCESSFUL
9.	If the password is not entered.	Password	Please fill out this field.	SUCCESSFUL
10.	If the password is invalid	Password	Passwords should be at least 8 characters long and include at least one uppercase letter, one lowercase letter, one digit, and one special character.	SUCCESSFUL
11.	If “confirm password” is not entered.	Confirm Password	Please fill out this field.	SUCCESSFUL
12.	If password and confirm password does not match.	Password and Confirm Password	Password and confirm password are not matching.	SUCCESSFUL
13.	If all fields are entered.	All mandatory fields.	The system will redirect them to the home Page.	SUCCESSFUL

4.6.3.2 Table: User Login

Serial No.	Condition to be tested	Test Data	Expected output	Remarks
1.	If Email ID is not entered	Email ID	Please fill out this field	SUCCESSFUL
2.	If the password is not entered.	Password	Please fill out this field	SUCCESSFUL
3.	If Email ID is invalid.	Email ID	Please include an '@' in the email address. Email ID is missing an '@'.	SUCCESSFUL
4.	If Email ID is not registered	Email ID	Email ID is not registered. Please use a registered email ID.	SUCCESSFUL
5.	If the password is invalid	Password	Passwords should be at least 8 characters long and include at least one uppercase letter, one lowercase letter, one digit, and one special character.	SUCCESSFUL
6.	If Email ID and passwords are not valid.	Email ID and Password	Incorrect Username or password.	SUCCESSFUL
7.	If Email Id is valid and password is not valid	Email ID and Password	Incorrect Username or password.	SUCCESSFUL
8.	If Email Id is not valid and password is valid	Email ID and Password	Incorrect Username or password.	SUCCESSFUL
9.	If Email Id and password are valid	Email ID and Password	The system will bring them to the home Page.	SUCCESSFUL

10.	When a user clicks on the “Sign up for Emotify” button for registration.	User	The system will redirect them to the Registration Page.	SUCCESSFUL

4.6.3.3 Table: Emotion Recognition Module

Serial No.	**Condition to be tested**	**Test Data**	**Expected output**	**Remarks**
1.	If there is a user in front of the webcam.	User	It needs to display the final emotion on the screen.	SUCCESSFUL
2	It needs to show a timer while detecting emotion.	User	It shows a timer for 5 sec while detecting emotion.	SUCCESSFUL
3	If the user is not in front of the webcam	Empty	Oops! Unable to detect mood.	SUCCESSFUL

4.6.3.4 Table: Contact Page

Serial No.	**Condition to be tested**	**Test Data**	**Expected output**	**Remarks**
1.	If the Email Id is empty	Email Id	Please fill out this field	SUCCESSFUL
2.	If the Name is empty	Name	Please fill out this field	SUCCESSFUL
3.	If the Message is empty	Message	Please fill out this field	SUCCESSFUL
4.	If the Email Id is invalid	Email Id	Email not registered. Please enter a valid email.	SUCCESSFUL
5.	If the Email Id is valid	Email Id	It will display a message “Message sent successfully!” on the contact page and an email will be sent to the admin.	SUCCESSFUL

4.6.3.5 Table: Music Player Module

Serial No.	Condition to be tested	Test Data	Expected output	Remarks
1.	Test user interaction with the music player controls (e.g., play, pause,next, stop).	User performs actions like playing, pausing, stopping and skipping to the next track.	The music player accurately responds to user commands, ensuring that the playback state is maintained as intended.	SUCCESSFUL
2.	Test shuffle playback mode	User activates shuffle mode during music playback	Proper functioning of shuffle mode, with music tracks played in random order	SUCCESSFUL
3	If the mp3 files are not found in the directory	Empty folder	No MP3 files found in the specified directory	SUCCESSFUL

4.6.3.6 Table: Modules

Team Member Names	Modules	Status
Vaishnavi	Registration	☑
Vaishnavi, Nimisha	Login	☑
Vaishnavi, Ajay	Emotion Detection	☑
Vaishnavi, Ajay	Music Player	☑
Vaishnavi, Manikanta	Viewing Features	☑
Vaishnavi, Nimisha	Viewing Team-Info	☑
Vaishnavi, Manikanta	Contact admin	☑
Vaishnavi	Logout	☑

4.6.3.7 Table: Testing Modules

Module Name	Black Box Testing	White Box Testing	Integration Testing	Regression Testing
Register	☑	☑	☑	☑
Login	☑	☑	☑	☑
Logout	☑	☑	☑	☑
Contact admin	☑	☑	☑	☑
Emotion Detection	☑	☑	☑	☑
Music Player	☑	☑	☑	☑
Viewing Features	☑	☐	☑	☐
Viewing Team-Info	☑	☐	☑	☐
Logout	☑	☑	☑	☑

4.6.4. Detailed Test cases

Test Number 1	Register
Test Objective:	To test the register module and make sure it works properly
Test Input:	
Valid input case	The user registers with their email-id and password
Invalid input case	Enter a registered email-id
Test procedures:	The input is compared with the information in the database
Test controls:	The test is run multiple times to cover every test case.
Test Output:	
Valid input case	The user is redirected to the home page
Invalid input case	A message will be displayed to the user that explains the email-id entered is already registered.

Table 4.1: Register Test Case

Test Number 2	Login
Test Objective:	To test the login module and make sure it works properly
Test Input:	
Valid input case	Enter a username and password that match a username and password from the database
Invalid input case	Enter a username that is in the database but with an incorrect password
Test procedures:	The input is compared with the information in the database
Test controls:	The test is run multiple times to cover every test case.
Test Expected Output:	
Valid input case	The user is redirected to the home page
Invalid input case	The user has a message popup that explains that the credentials entered are invalid.

Table 4.2: Login Test Case

Test Number 3	User session
Test Objective:	Test the functionality of each logged in user creates a session that allows for their information to be processed until logging off
Test Input:	
Input case 1	A successful login
Test procedures:	A session is created
Test controls:	For each test we create a different session and check if the old session ended properly
Test Output:	
Input case 1	The user is redirected to the home page and cannot go back to the login page without logging out, and the user can only view their information and is unable to create another session until logged out.

Table 4.3: User Session Test Case

Test Number 4	Contact
Test Objective:	To test the contact module and make sure it works properly
Test Input:	
Valid input case	Enter an email-id that match with the database
Invalid input case	Enter an incorrect email-id that is not in the database.
Test procedures:	The input is compared with the information in the database
Test controls:	The test is run multiple times to cover every test case.
Test Output:	
Valid input case	The query or feedback is sent to the admin through an email
Invalid input case	A message will be displayed to the user that explains the email-id entered is invalid.

Table 4.4: Contact Test Case

5. User Manual

5.1 Introduction

The EMP System utilizes advanced emotion recognition algorithms and machine learning techniques to analyze and synchronize music with user emotions. This manual will guide users with the outline and its key functionalities, so that users can start using it right away.

5.2 Hardware Requirements

To use the EMP System , the user will need a computer with a webcam and an active internet connection. An audio output device like speakers or microphone are required for playing music based on detected emotions.

5.3 Software Requirements

- The user will need to have a web browser such as Chrome, Firefox, Safari and Python 3.6 or higher installed on their computer.
- Python libraries such as NumPy, OpenCV (cv2), TensorFlow, Flask, Flask-Mail, SQLAlchemy, bcrypt, and Python-VLC must be installed.

5.4 Installation and Set Up

Python can be downloaded for free from https://www.python.org/downloads/. Once the user clicks on the download option, an installer will be downloaded into the system. After the installation is completed, the user will be able to verify if Python is successfully installed by running the command `python3 --version` in the command prompt or terminal. The installed Python version will then be displayed.

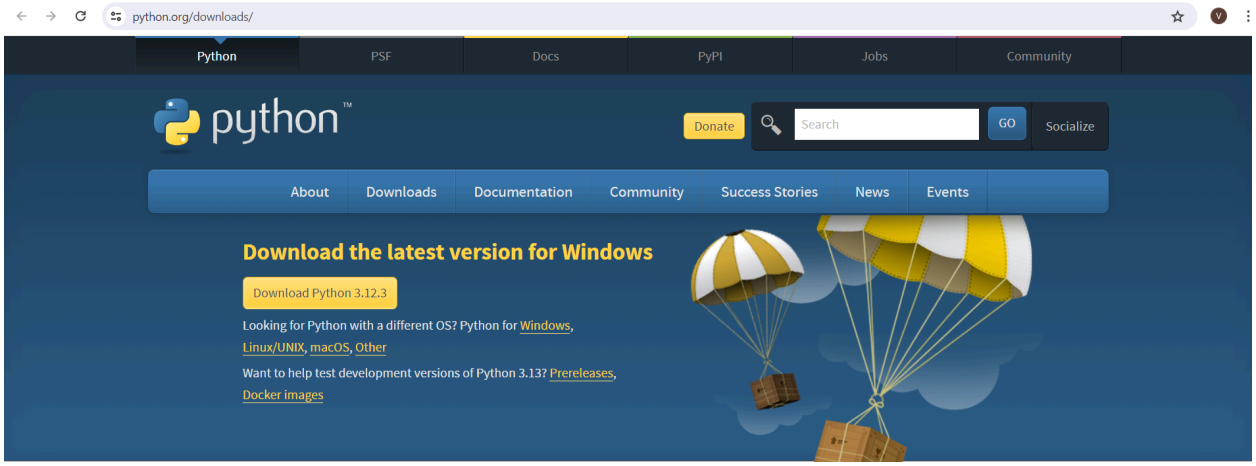

Fig 5.1: Python Installation

Since the system is not hosted on any external servers, users can deploy it locally. To accomplish this, users will need to utilize the VISUAL STUDIO CODE IDE. The IDE can be conveniently downloaded from the following URL https://code.visualstudio.com/download. Once Visual Studio Code is successfully downloaded, the user needs to clone the repo and open it with the IDE to get started.

- Important libraries:

  All the below-mentioned libraries should be installed by running the installation commands

  - Installation Commands:
    - pip3 install flask
    - pip3 install opencv-python
    - pip3 install numpy
    - pip3 install python-vlc
    - pip3 install Flask-SQLAlchemy
    - pip3 install Flask-Mail
    - pip3 install tensorflow
    - pip3 bcrypt
    - pip3 install tk
  - After all the libraries are successfully installed, the user needs to run the emotions.py class to deploy the system locally. The following output “Running on http://127.0.0.1:5000” indicates that the system is running on that port.

```
 * Serving Flask app 'emotions'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Fig 5.2 : VS Code Terminal Image for the port address

5.5 Website Pages Overview

- Open your preferred web browser and navigate to the system's URL http://127.0.0.1:5000 to access the index page below
- Ensure your computer's webcam is functioning properly for optimal performance.
- Ensure that your computer's audio system is enabled and working correctly.

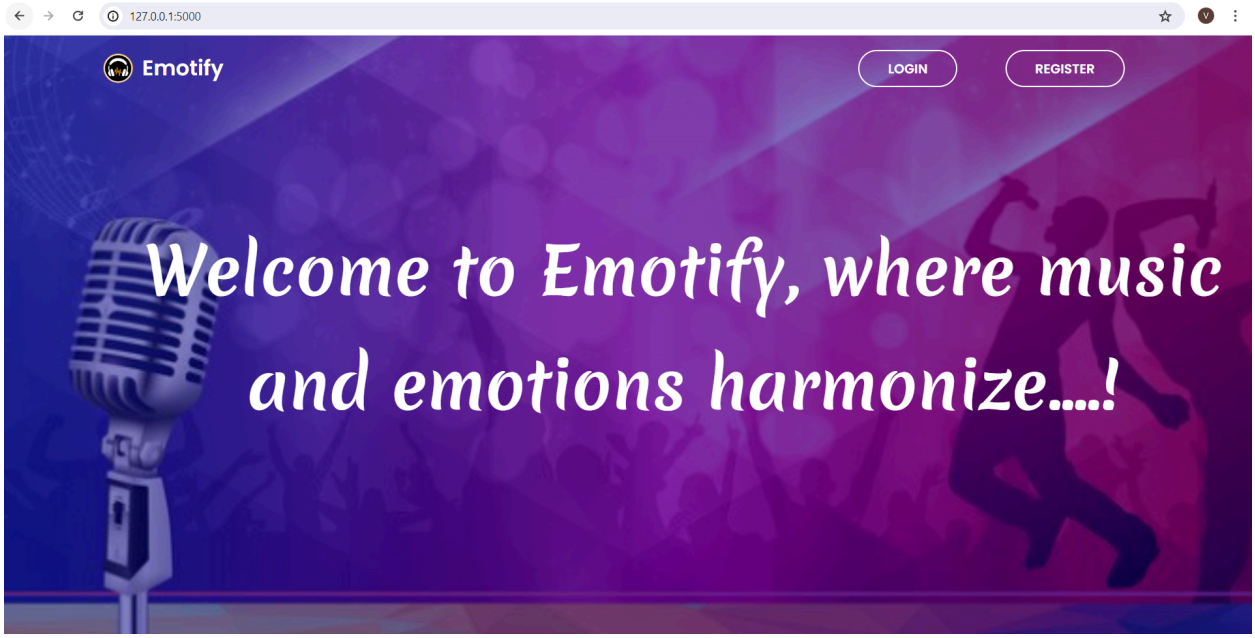

Fig 5.3: Index page of EMP System

5.5.1 Register Page

- This is the registration page where users can sign up to access the website by providing their details.
- Upon clicking the "Register" button, users will be redirected to the Home page.
- For users who are already registered, they can navigate to the login page by clicking on the "Log in here" button displayed at the bottom of the page.

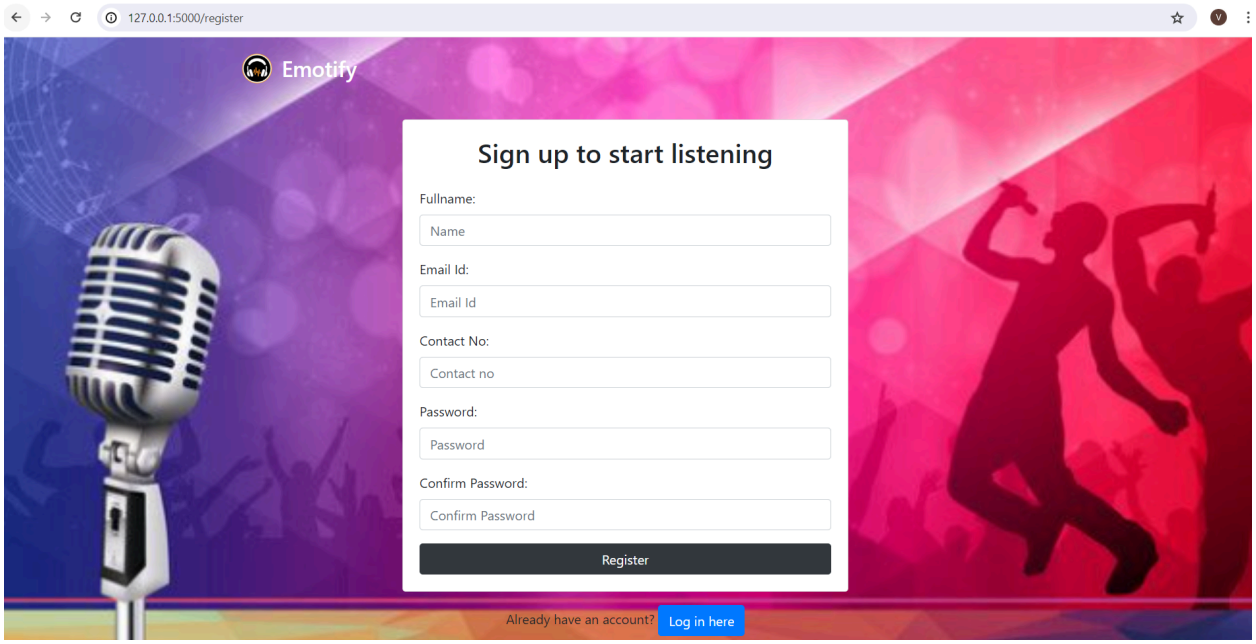

Fig 5.3: Register page of EMP System

5.5.2 Login Page

- This is the Login Page where users can log in using their registered email ID and password.
- Upon clicking the Login button, users will be redirected to the Home page.
- If a user doesn't have login credentials, they can click on the "Sign up for Emotify" button to register for the website.

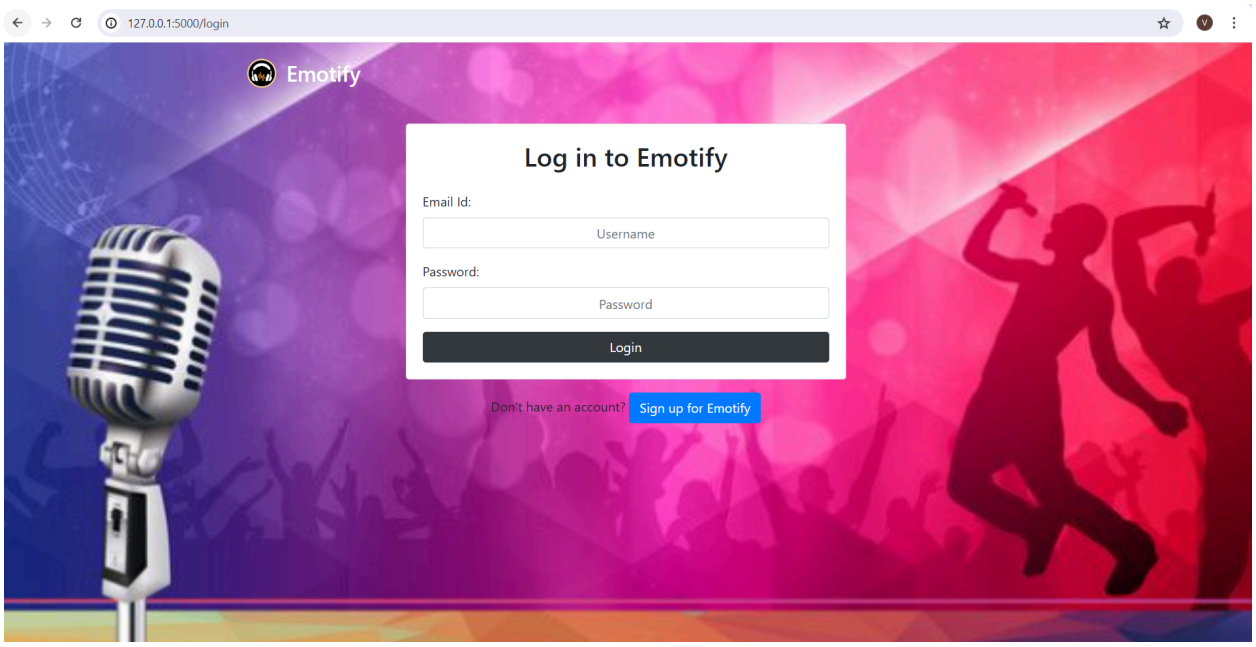

Fig 5.4: Login page of EMP System

5.5.3 Home Page

- Once the user is redirected to the home page as shown in the figure 5.5, they will find a "Start Emotion Detection" button.
- Clicking this button opens the webcam.

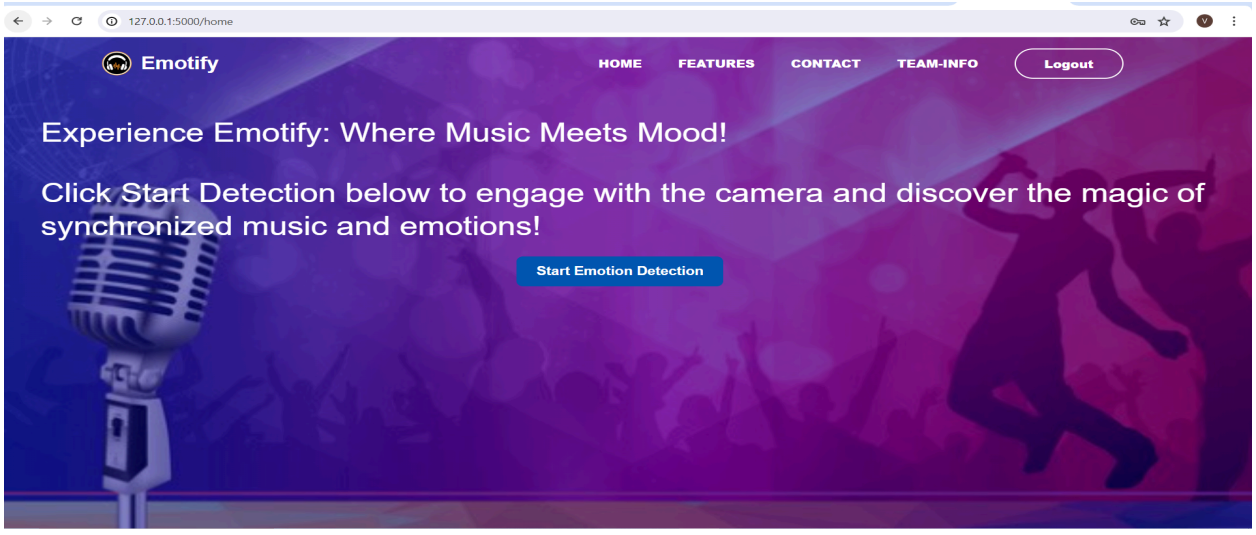

Fig 5.5: Home page of EMP System

- Once the webcam is activated, the system detects different types of emotions such as happy, neutral, sad, angry, and surprised, as shown in the figures from Figure 5.6 to Figure 6.0.

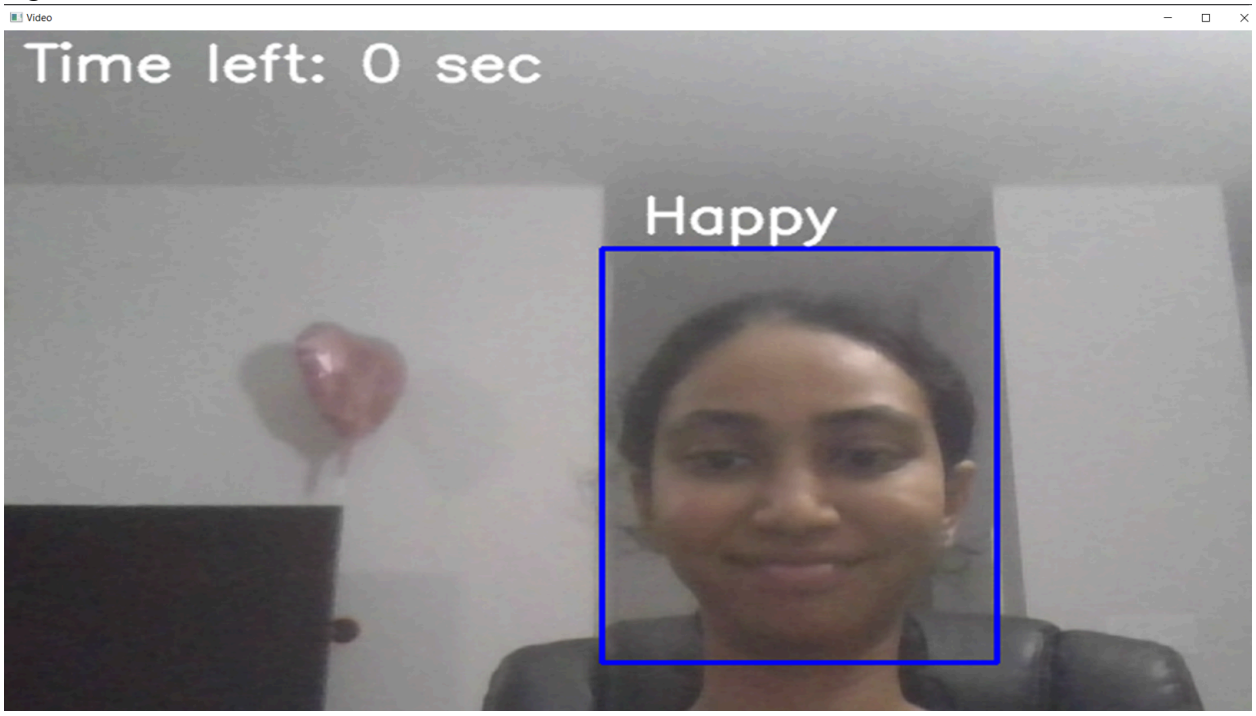

Fig 5.6: Happy Emotion

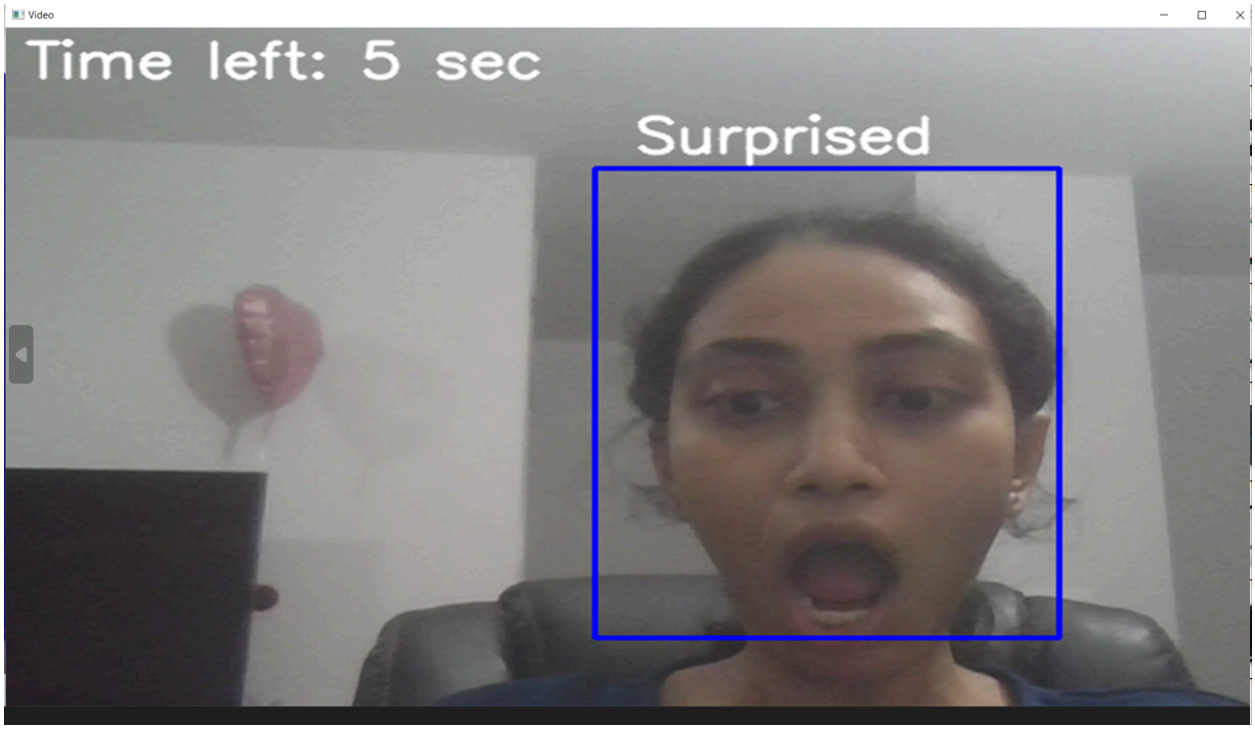

Fig 5.7: Surprised Emotion

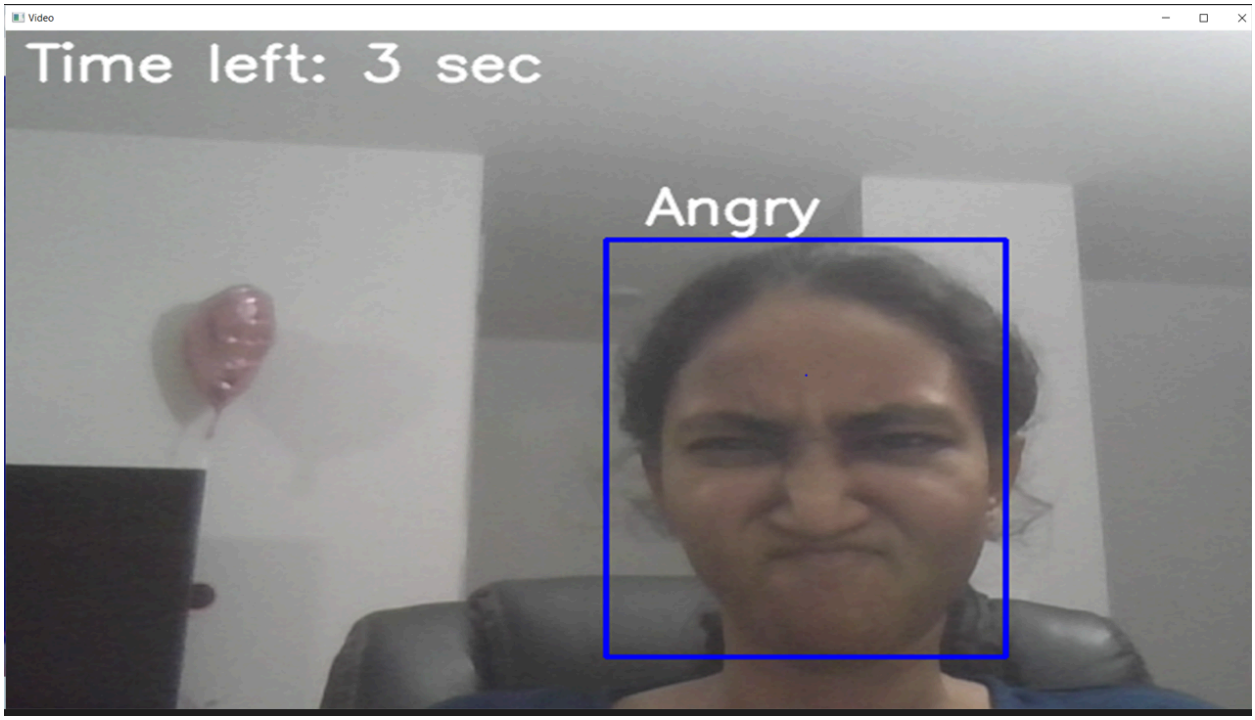

Fig 5.8: Angry Emotion

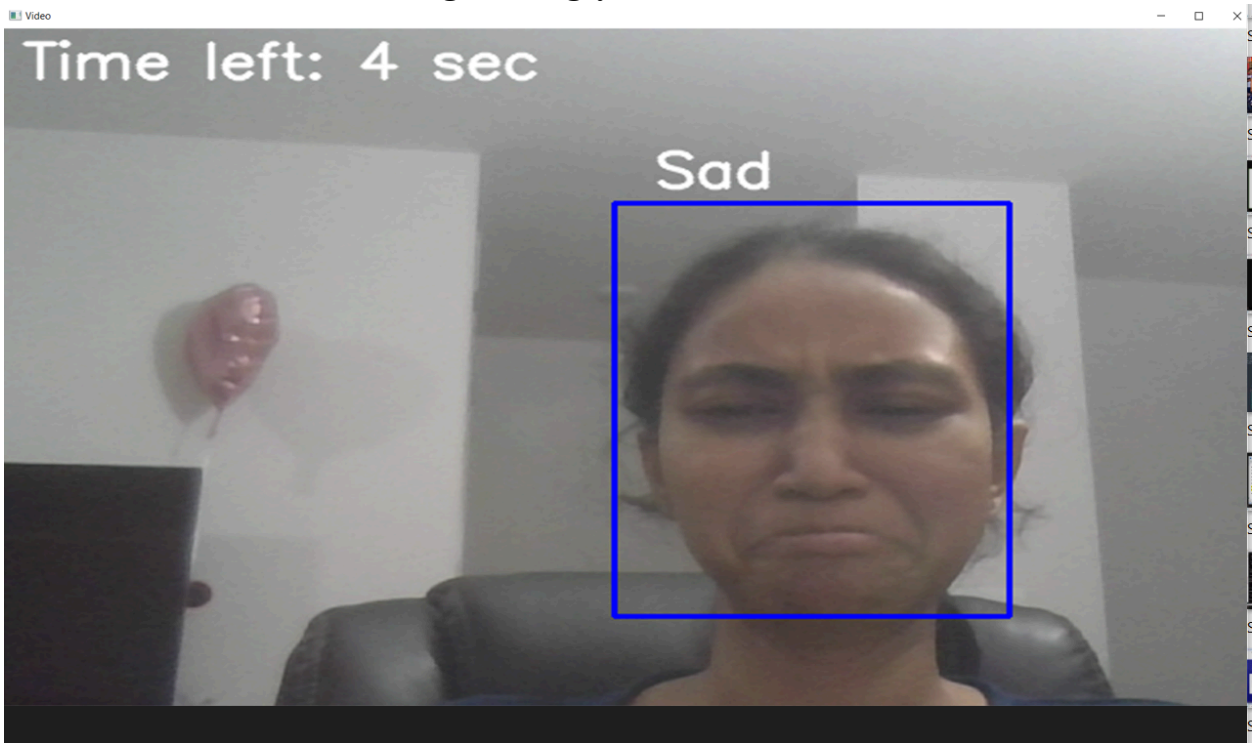

Fig 5.9: Sad Emotion

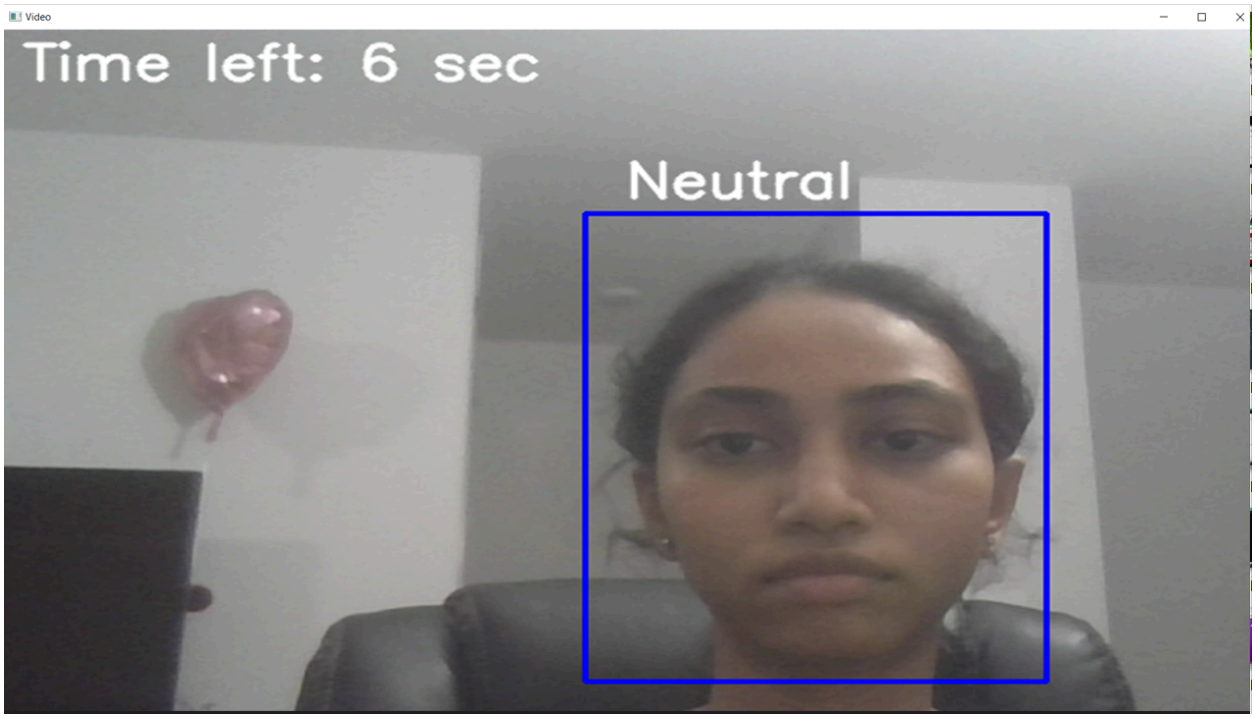

Fig 6.0: Neutral Emotion

5.5.4 Displaying Emotion of the Users

- Once the emotion is detected, the final emotion of the user gets displayed as shown in Figure 6.1.
- After a few seconds, the music player automatically opens and starts playing songs.

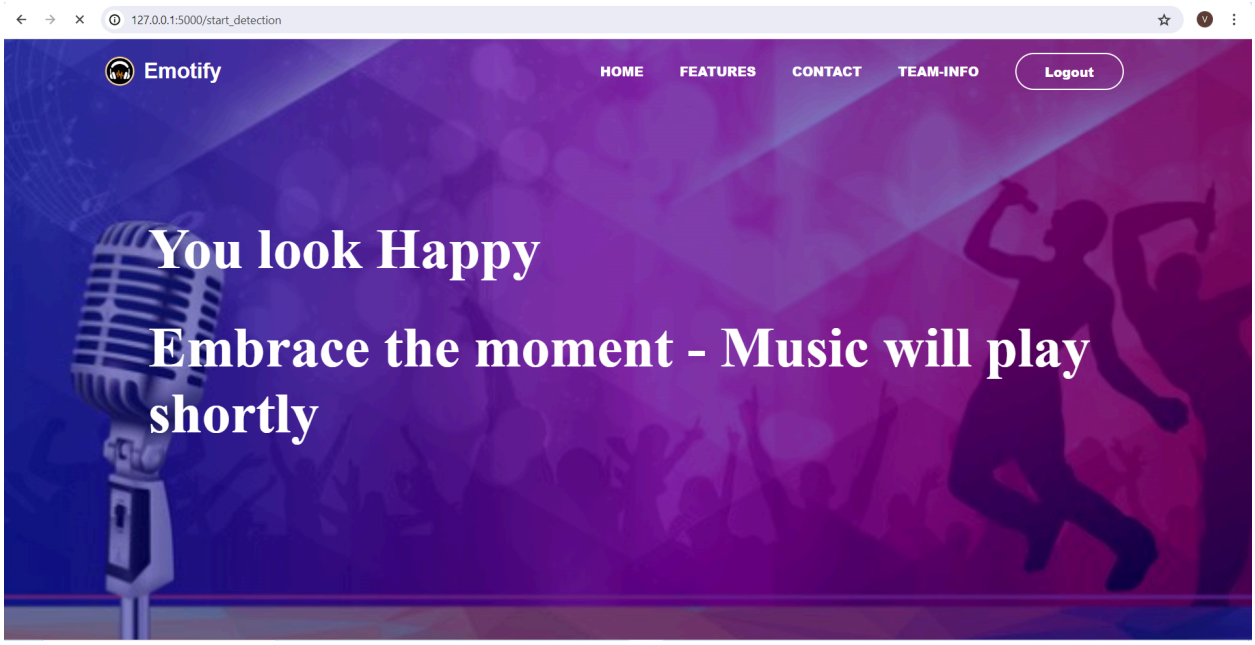

Fig 6.1:Displaying Emotion of the User

5.5.5 Music Player

- The Music Player(VLC-Media Player) includes controls such as play, pause, stop, next, and shuffle in the Control Panel.
- Users can also choose the types of songs they want to listen to from the Song Playlist.
- The Song Track will display the names of the songs being played.
- Clicking the stop button in the music player will halt the music playback.

Fig 6.2: Music Player of EMP System

5.5.6 Team-Info Page

- This page showcases all the team members of the website along with their details such as name, email, and phone numbers.
- Users can use these details to contact the team members if needed.

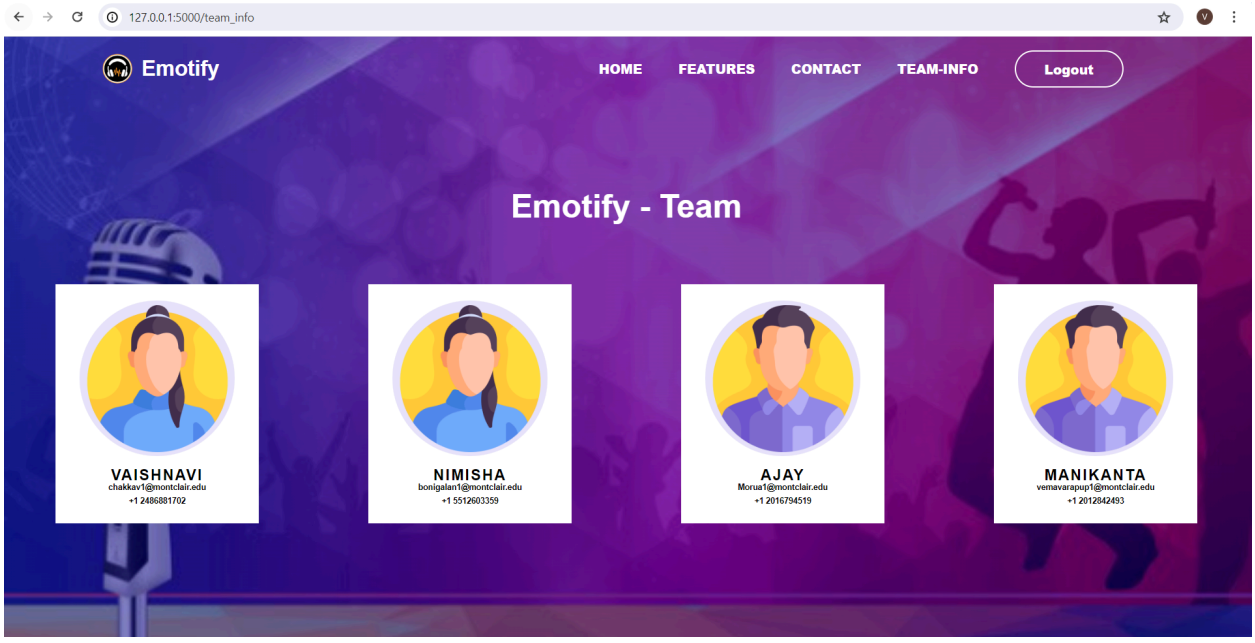

Fig 6.3: Team-Info of EMP System

5.5.7 Contact Us Page

- If users wish to contact the admin for queries or provide feedback about the website, they can utilize this page to send a direct message.
    - Upon clicking the “Send Message” button, the message is delivered to the admin via email.

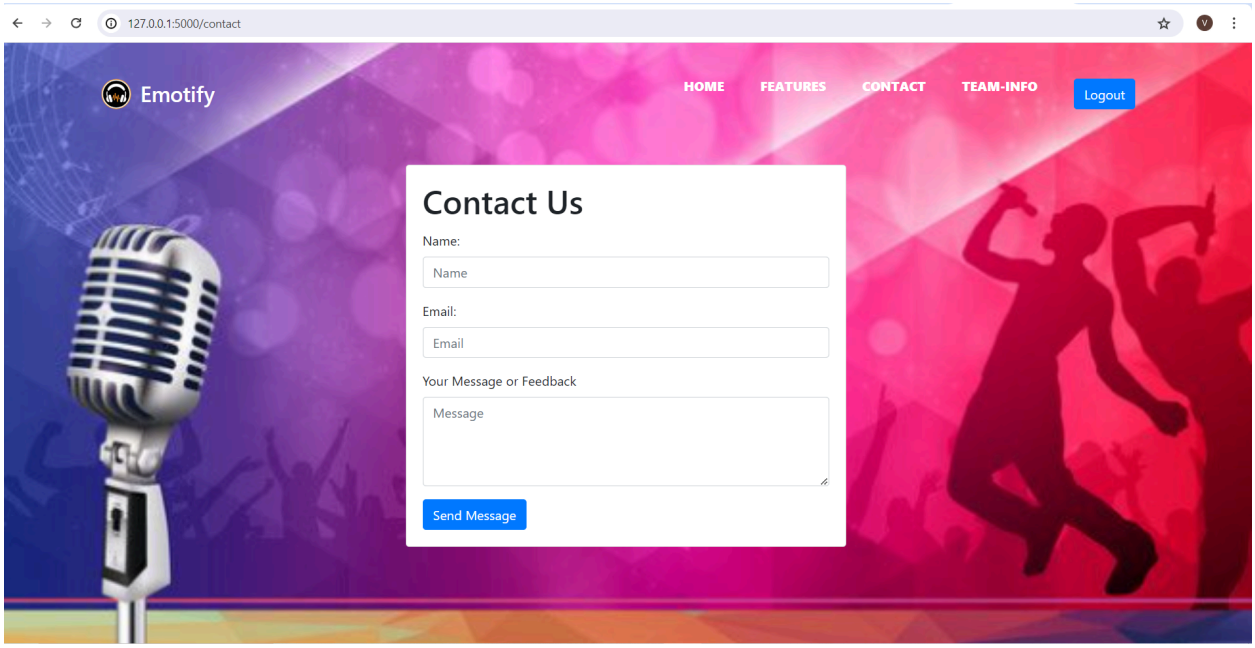

Fig 6.4: Contact page of EMP System

5.5.8 Features Page

- This page displays the features of both the current and upcoming versions in a clear and informative manner.

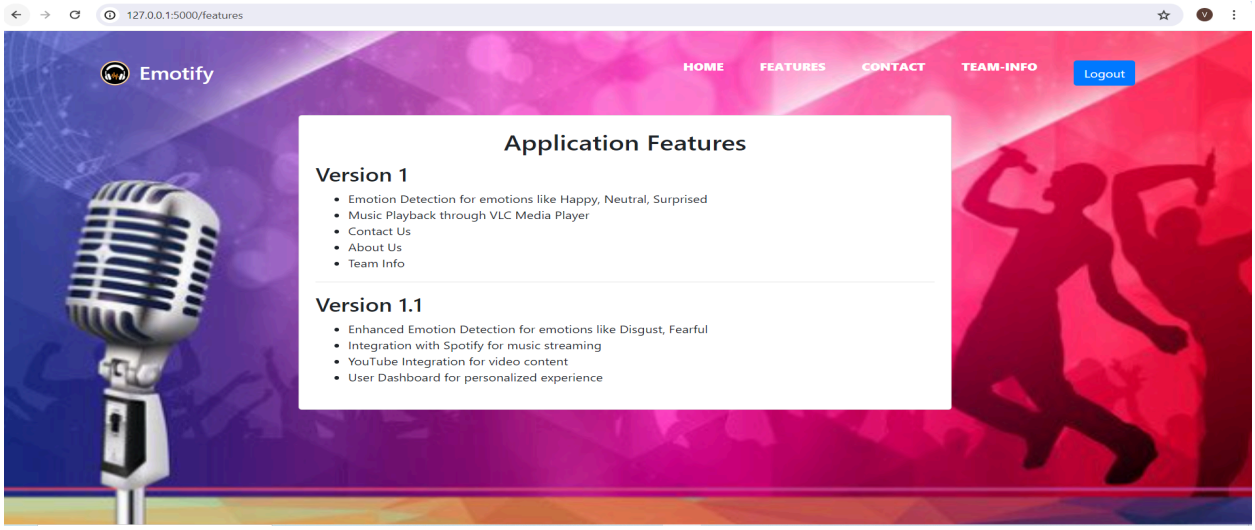

Fig 6.5: Features page of EMP System