TITLE

EmoGroove: Crafting Sentimental Soundscapes from Social Streams

PROBLEM STATEMENT:

Develop a sentiment analysis model, combining textual analysis of social media message and image analysis of webcam snapshots, to categorize sentiments as neutral, negative or positive. Additionally, curate a song playlist that dynamically reflects the emotional tone of the analysed content, blending genres to resonate with the sentiments expressed.

AIM:

Our task is to develop a sentiment analysis multi-model that combines textual analysis of social media messages and image analysis of webcam snapshots. This model is intended to categorize sentiments expressed in these messages and images as neutral, negative, or positive.

Additionally, the objective is to curate a dynamic song playlist that reflects the emotional tone of the analysed content. The playlist should blend genres to resonate with the sentiments expressed in the messages and webcam snapshots.

In summary, the problem involves creating a multifaceted system that analyses both text and images to determine sentiments, and then uses this information to generate a music playlist that aligns with the emotional tone of the content.

TEXTUAL SENTIMENT ANALYSIS EXPLAINATION:

1. Dataset:

Source: "/content/emotion\_dataset\_raw.csv"

Structure: The dataset consists of two columns:

Text column (containing social media messages)

Emotion column (containing labels: happy, sad, angry, etc.)

1. Exploratory Data Analysis (EDA):

Value counts: Examined the distribution of emotions in the dataset. Visualization: Created a count plot to visualize the distribution of emotions.

1. Data Cleaning:

Removed user handles: Ensured analysis focuses on text content, not social media conventions (e.g.: @username, #sports etc.)

Removed stop words: Eliminated common words that often carry less meaning (e.g., "the", "a", "and").

1. Feature and Label Separation:

Features: Used the cleaned text column as input features for the model. Labels: Used the emotion column as target labels for the model to predict.

1. Data Splitting:

Train-test split: Divided the dataset into training (70%) and testing (30%) sets to train and evaluate the model's performance.

1. **Model Building:**

Pipeline: Created a pipeline to streamline the process of feature extraction (using Count Vectorizer) and model training (using Logistic Regression).

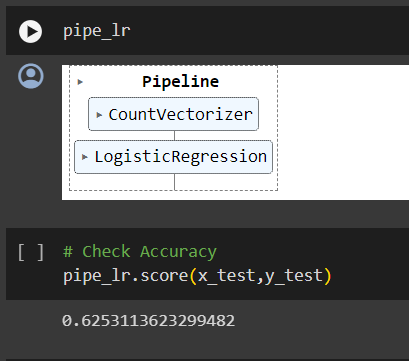
Model fitting: Trained the pipeline on the training data.

**Why we chose Logistic regression for training the model:**

* Interpretability: Logistic Regression provides interpretable results. The coefficients of the model can be analysed to understand the impact of each feature on the prediction.
* Efficiency: Logistic Regression is computationally efficient and works well on relatively large datasets.
* Less Prone to Overfitting: Logistic Regression is less prone to overfitting compared to more complex models, making it suitable when you have limited training data.
* Accuracy: Compared to other models like Decision Tree, Random Forest, Logistic Regression gave more accuracy so we chose this model.

1. Model Evaluation:

Accuracy: Assessed the model's accuracy on the testing set.



1. Prediction:

New example: Demonstrated how to use the trained model to predict the emotion of a new text example.

Prediction probability: Obtained probability scores for each emotion class for the new example.

1. Model Saving:

Serialization: Saved the trained pipeline as a .pkl file for future use in Flask API.

Key Points:

* The code focuses on text-based sentiment analysis.
* It employs a Logistic Regression model with Count Vectorizer for feature extraction.
* The model's accuracy on the testing set provides a measure of its performance.
* The saved pipeline can be used to predict emotions for new text inputs.

IMAGE SENTIMENT ANALYSIS EXPLAINATION:

1. Dataset:

Name: Facial Expression Recognition dataset (FER2013)

Source: Kaggle

Contents: 35,887 grayscale images of faces

Each image: 48x48 pixels

Labels for 7 emotions:

0=Angry,

1=Disgust,

2=Fear,

3=Happy,

4=Sad,

5=Surprise,

6=Neutral

1. Model Architecture:

i)Data Preparation:

* Load the dataset from CSV file
* Convert labels to one-hot encoded vectors
* Preprocess images:
* Convert pixel strings to numerical arrays
* Normalize pixel values to 0-1 range
* Reshape images to (48, 48, 1) format

ii)Model Structure:

Sequential: A linear stack of layers, used for straightforward model building.

Input Shape: (48, 48, 1), indicating grayscale images of 48x48 pixels.

Layers:

Convolutional Block 1:

Conv2D(64, (5, 5), activation='relu', padding='same')

Conv2D(64,(5, 5), padding='same')BatchNormalization()

Activation('relu') MaxPooling2D(pool\_size=(2, 2))

Convolutional Block 2:

Conv2D(128, (5, 5), activation='relu', padding='same')

Conv2D(128, (5, 5), padding='same')BatchNormalization()

Activation('relu') MaxPooling2D(pool\_size=(2, 2))

Convolutional Block 3:

Conv2D(256, (3, 3), activation='relu', padding='same')

Conv2D(256, (3, 3), activation='relu', padding='same')

BatchNormalization()MaxPooling2D(pool\_size=(2, 2))

Flatten: Flattens the 2D feature maps into a 1D vector for further processing.

Dense: A fully connected layer with 6 output neurons, using softmax activation for multi-class classification.

1. Compilation:

Loss: categorical\_crossentropy, suitable for multi-class classification.

Optimizer: adam, a widely used optimizer with adaptive learning rates.

Metric: accuracy, to measure the model's correctness in classifying examples.

1. Training:

Number of epochs: 22

Batch size: 64

1. Model Saving:

Model saved in both .h5 and .keras format for further use in Flask API.

Key Points:

* Data balancing: The code limits the number of images for each emotion to 4000, attempting to balance the dataset.
* Image resizing: Test images are resized to 48x48 before prediction. Emotion prediction: The model outputs a probability for each emotion, and the emotion with the highest probability is selected.
* The model uses convolutional layers to extract features from images, which is effective for image-based tasks like facial expression recognition.
* Batch normalization is used to improve model stability and convergence.
* Max pooling layers are used to reduce dimensionality and make the model more efficient.

**Why we chose CNN for emotion recognition:**

Convolutional Neural Networks (CNNs) are commonly used for image-related tasks, and they have been particularly successful in Facial Expression Recognition (FER).

* **Spatial Hierarchy in Facial Features:**

Facial expressions involve complex spatial patterns and hierarchies of features. CNNs are designed to automatically learn hierarchical features from images, capturing both low-level details (such as edges and textures) and high-level patterns (such as facial expressions).

* **Local Receptive Fields:**

CNNs use local receptive fields in their convolutional layers, allowing them to focus on small, overlapping regions of the input image. This is advantageous for capturing local patterns in facial expressions, where the arrangement of facial features can vary across different emotions.

* **Translation Invariance:**

CNNs are inherently translation-invariant due to weight sharing in convolutional layers. This property is beneficial for facial expression recognition as emotions can be expressed in different positions on the face.

* **Parameter Sharing and Efficiency:**

CNNs use parameter sharing through the use of convolutional kernels, which significantly reduces the number of parameters compared to fully connected networks. This makes CNNs more efficient and scalable, especially for tasks like facial expression recognition.

* **Pre-trained Models and Transfer Learning:**

CNNs trained on large datasets, such as ImageNet, can be used as pre-trained models for transfer learning on the FER2013 dataset. This allows the model to leverage knowledge learned from a diverse set of images and adapt it to the specific task of facial expression recognition, even when the dataset is relatively small.

* **Pooling Layers:**

Pooling layers in CNNs help reduce spatial dimensions while retaining important features. This aids in creating a more abstract representation of facial expressions, making it easier for the network to learn discriminative features.

Multi-Model Implementation:

* Before integrating the individual models into a unified prediction, the predictions from Happy, Sad, Fear, Neutral, Angry & Surprise are mapped into 3 categories i.e. Negative, Positive and Neutral.
* Having obtained independent predictions from both the image and text sentiment analysis models, we proceeded to combine them using an averaging strategy. In this approach, we calculated the average confidence scores from both models.
* The resulting combined predictions were then used to make a final sentiment prediction for the multimodal model.
* This averaging technique aims to achieve a balanced contribution from both modalities, providing a unified sentiment analysis outcome.

Building the recommendation system:

1. User Interaction in React App

* Webcam Capture:

The user accesses the React app and navigates to the webcam capture component.

A button triggers the capture of an image from the user's webcam.

* Sending Image to Backend:

The captured image is converted to a base64-encoded string.

Using an API service, the React app sends the image data to the Flask backend API.

2. Backend Processing

* Image Handling:

The Flask backend receives the base64-encoded image data.

Optionally, image processing tools like OpenCV may be used for additional preprocessing.

* Emotion Analysis:

A pre-trained machine learning model analyzes the image to detect the user's emotion.

The detected emotion is sent back to the React app.

3. Displaying Emotion Analysis Result in React App

* Emotion Analysis Result Component:

The React app updates the UI to display the detected emotion.

User feedback based on the emotion analysis result is provided.

4. Spotify API Integration

* Authentication with Spotify API:

The React app authenticates with the Spotify API using client credentials obtained during the application registration process.

An access token is acquired for subsequent API requests.

* Requesting Music Playlists:

Using the received emotion, the React app triggers requests to the Spotify API's endpoint.

The access token is included in the request for authorization.

* Fetching and Displaying Recommendations:

The React app receives music playlists from the Spotify API.

The recommendations are displayed to the user for exploration. Clicking on the recommendations redirects you the dedicated playlist on Spotify.

5. User Exploration and Interaction

* User Interaction:

The user explores the displayed music recommendations.

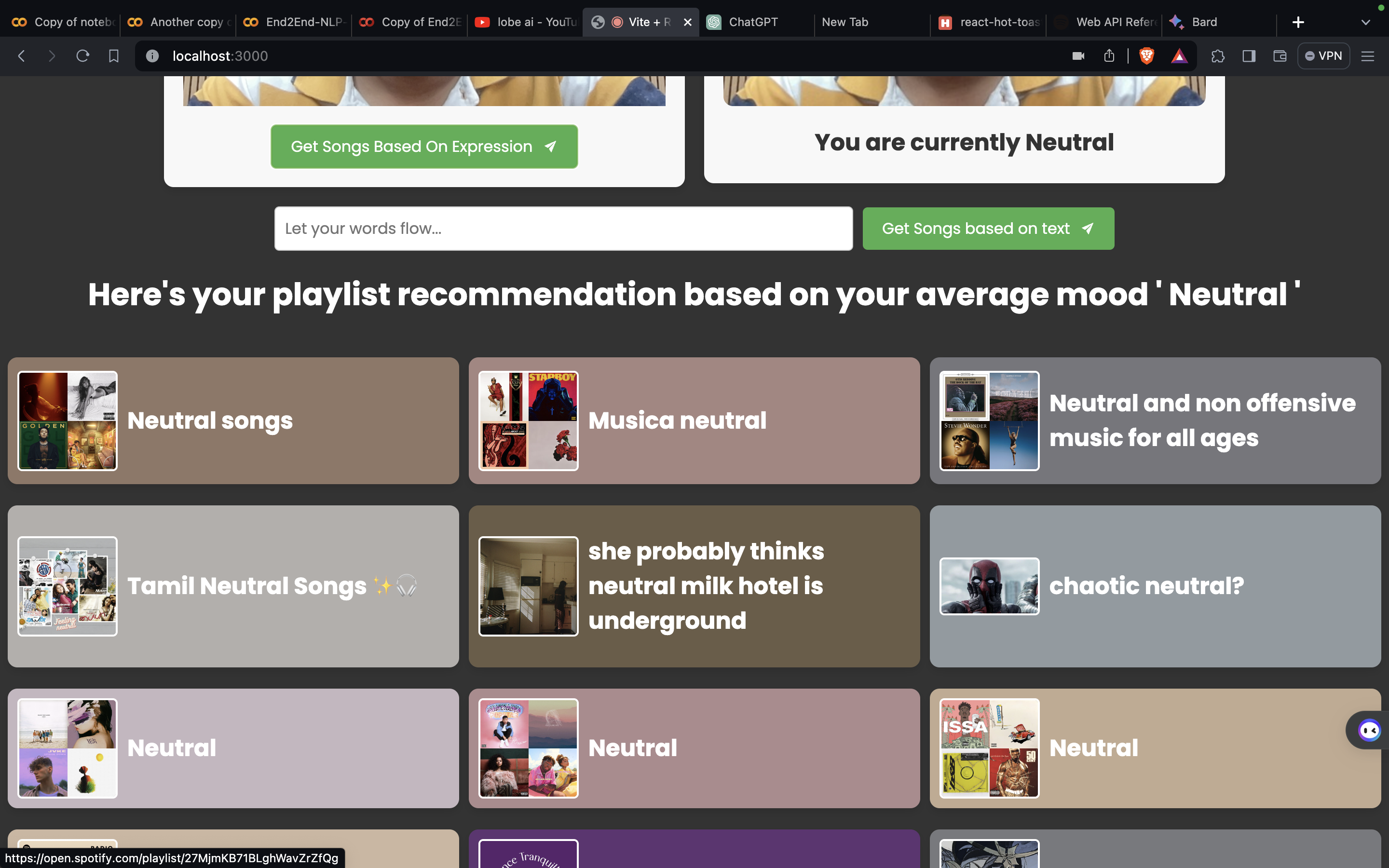
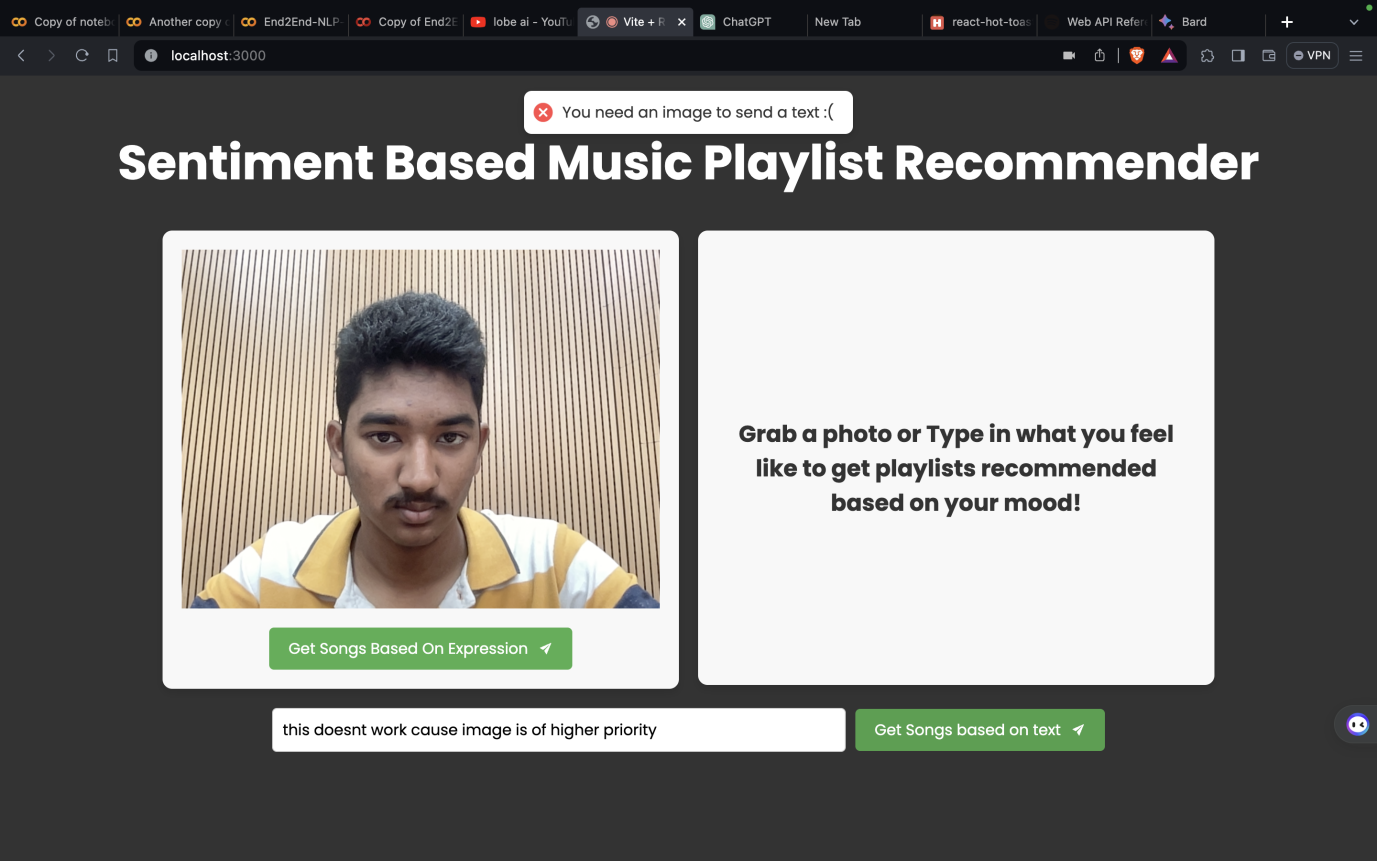
Additional actions, such as listening to previews or saving tracks, can be implemented based on the app's features.

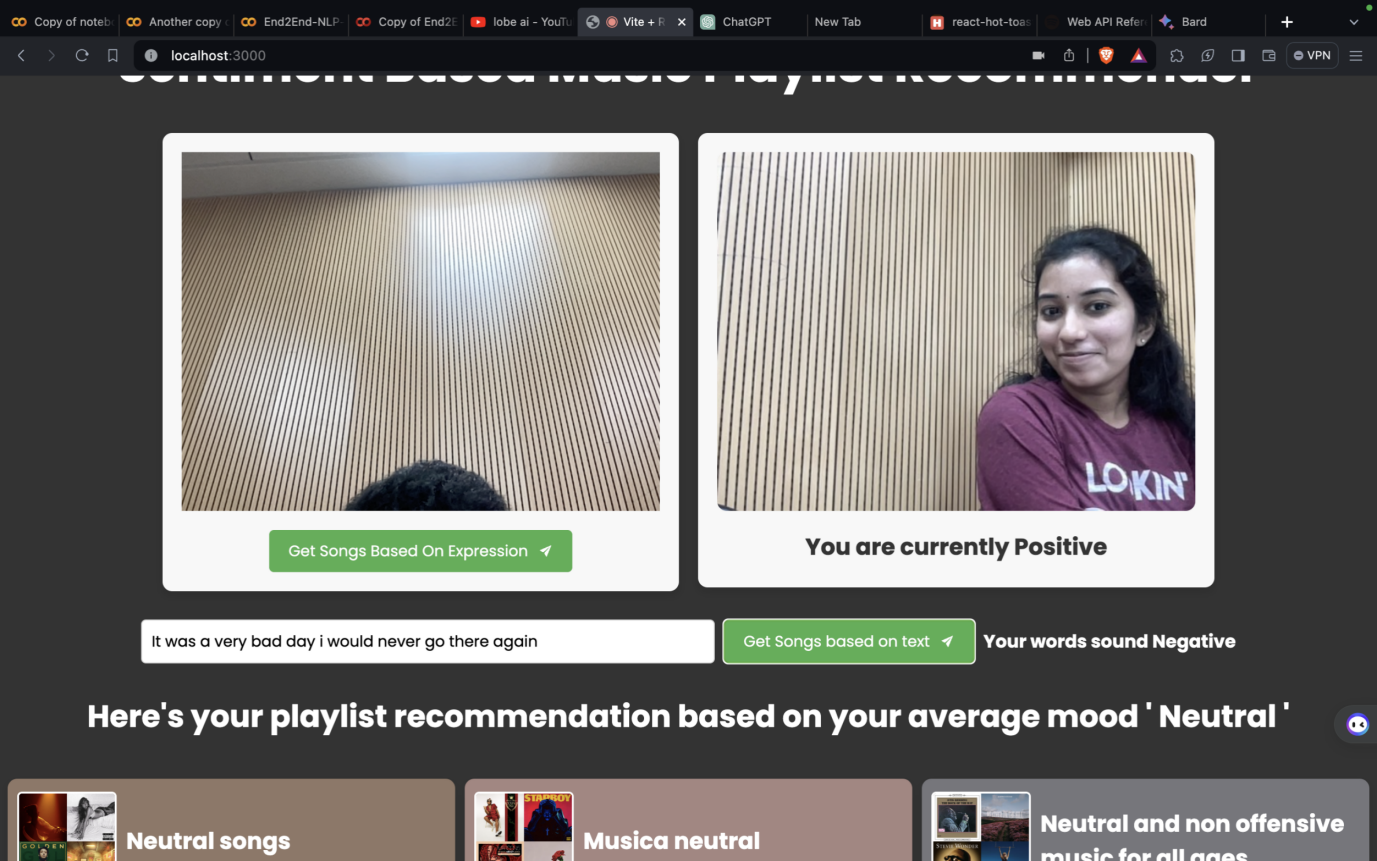
Conclusion

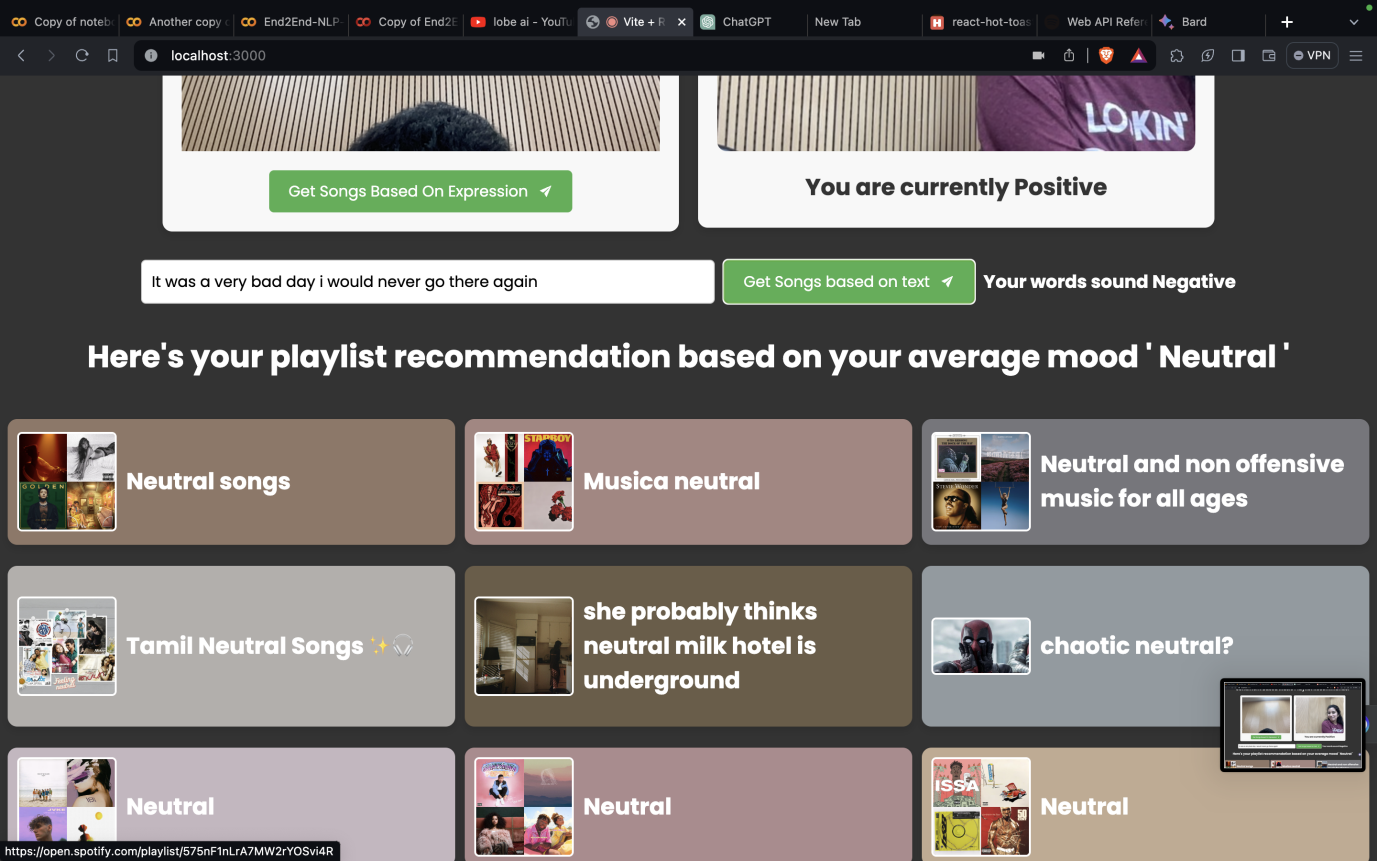
This flow encompasses the complete journey from user interaction to backend processing, emotion analysis, Spotify API integration, and the final display of music recommendations.

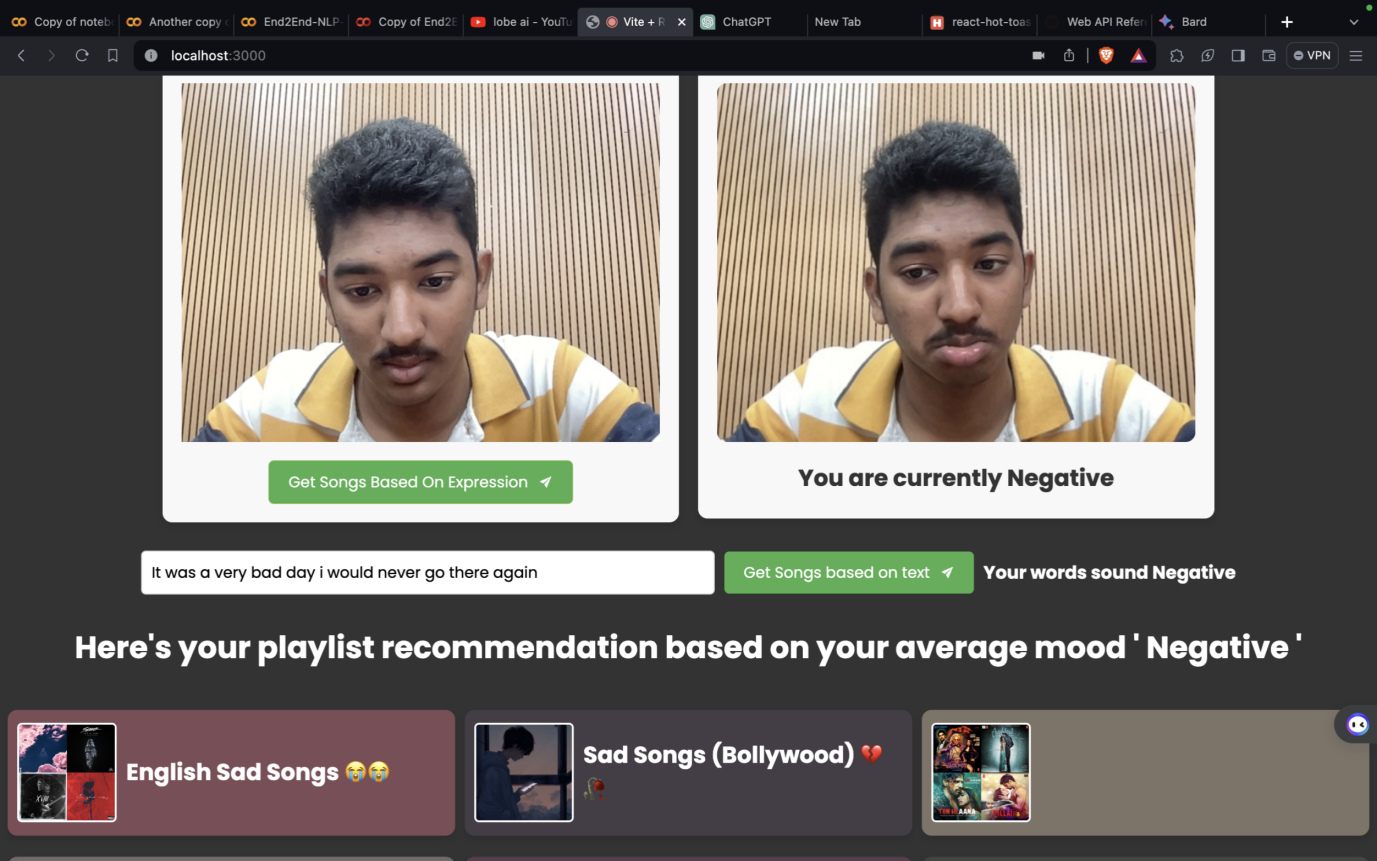
It aims to enhance the user's experience by offering personalized music suggestions based on their detected emotion.

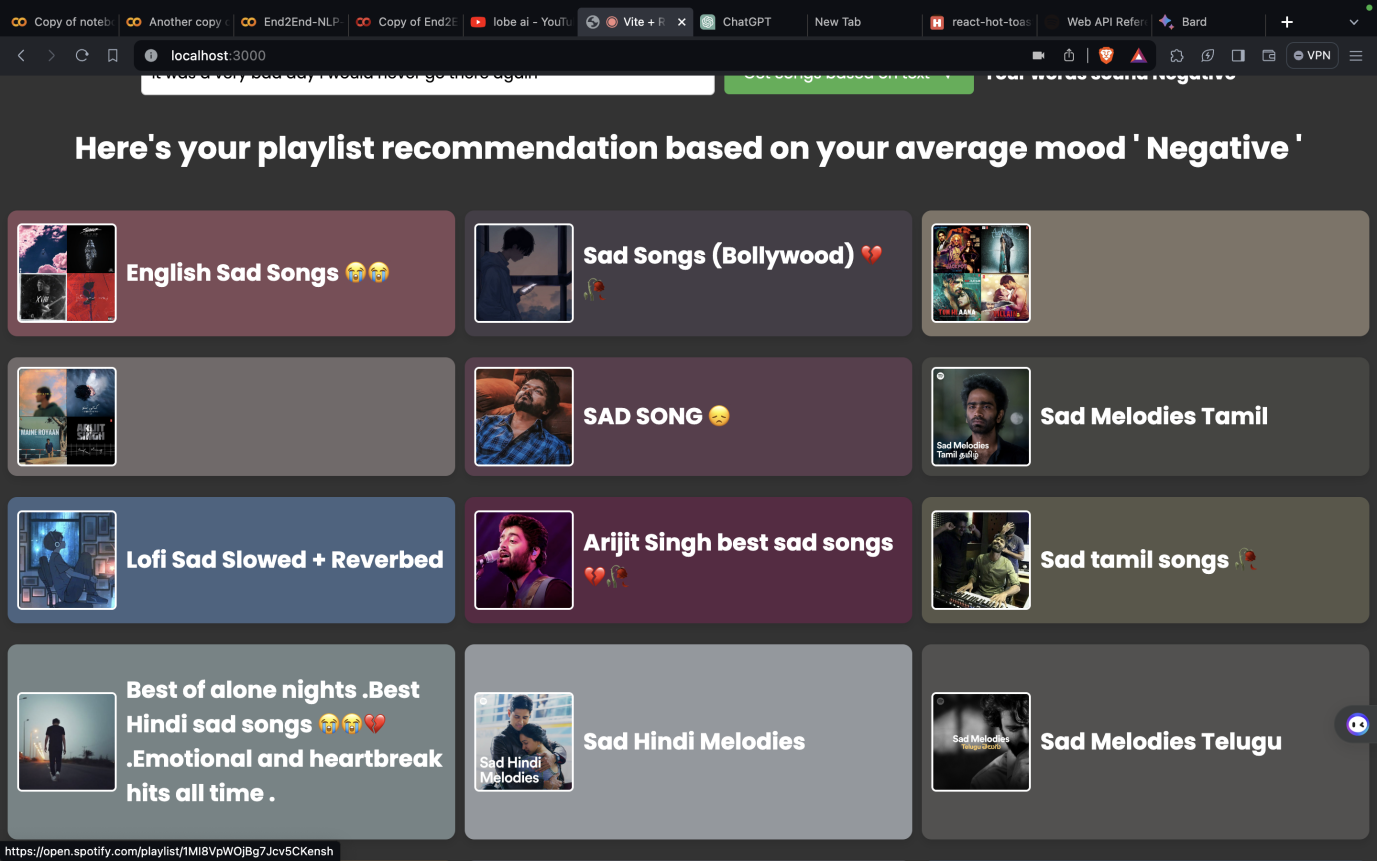
DEMO:











GitHub Repository - https://github.com/bhargavrepaka/Sentiment-Analysis-hackathon