Team 2

Clarin Riya Mathias (2348315)
Haripriya MC (2348323)
Muttha Kavitha (2348340)
Sneha SM (2348353)
Sukruth S (2348362)

ABSTRACT

In today's fast-paced world, emotional well-being is often neglected despite its critical importance. Stress, anxiety, and other mental health issues are on the rise, yet many people lack the necessary support to manage their emotions effectively. Digital platforms have revolutionized communication, but they often fail to consider the user's emotional state. Our project aims to address this gap by developing an Emotion-Responsive Chatbot that utilizes advanced technologies to detect emotions from facial expressions and engage users in personalized, supportive conversations. The purpose of this project is to create an intelligent chatbot that can detect and interpret a user's emotions from an image and interact with them accordingly. By recognizing emotions such as happiness, sadness, anger, and surprise, the chatbot will provide responses designed to enhance the user's mood and offer emotional support. It enhances users' well-being through emotion detection and personalized interaction. reasons. An Emotion-Responsive Chatbot can provide an immediate, non-judgmental platform for users to express their emotions and receive support, potentially mitigating feelings of isolation and distress. In addition to individual benefits, the chatbot can have a broader societal impact by promoting emotional intelligence and mental health awareness. It can serve as a tool for educational institutions, workplaces, and healthcare providers to support emotional well-being and encourage open discussions about mental health.

Table of Contents

1. INTRODUCTION

1.1 PROBLEM DESCRIPTION

Current chatbots across the world are not able to effectively recognize and manage the user's emotional state. This limits the potential for meaningful interaction on themes as serious as mental health support and customer service. The Emotion-Driven Chatbot project intends to fill this gap by incorporating emotion detection from facial expressions and Natural Language Processing to ensure that users go through personalized and empathetic interactions.

1.2 EXISTING SYSTEM

Emotion-driven chatbots build on this, allowing recognition, response, and even simulation of emotions during an interaction with a user. In such a case, emotion-driven chatbots would apply NLP to actively detect the feeling-sometimes with the use of machine learning to detect and react to the user's emotional state. Below are some existing emotion-driven chatbots and their respective shortcomings.

1. Replika: It is a particular AI chatbot that is developed as a companion for emotionally deep conversations. It uses both NLP and deep learning in developing human-like responses that create emotional bonds with people. Indeed, Replika has those moments when, by incorrectly detecting the user's emotions and responses, it results in responses out of context or insensitivity. While it can mimic simple emotions, it lacks the complexity to understand subtly nuanced emotional states. Since Replika stores conversations in order to improve responses, it does make it somewhat vulnerable to issues of data privacy and security.

2. Woebot: It is an AI chatbot that would try to help through conversations and CBT-like techniques. It will try to understand the emotional state of the user and advise or intervene appropriately. No more than a general mental health support chatbot, Woebot neither states nor claims it can serve in grave emotional crisis situations or for complex mental health conditions. While helpful, the capacity for empathy or understanding that could come from a human

therapist is beyond the capabilities of Woebot. Sometimes the responses may sound a little repetitive or even scripted, which can make the emotional rapport not as strong.

3. Emotibot: It is an AI platform integrated with emotional intelligence within interactions provided by its chatbot. Among the many areas in which it is applied are customer service, healthcare, and other apps where emotionally cognizant responses are required. With regard to the emotional shades of different languages and cultures, Emotibot may become less effective. It could result in overfitting to particular emotional patterns and the drawing of wrong inferences with respect to the user's emotional condition. Emotion-driven interactions are computationally intensive, and real-time processing can be challenging, leading to delays or inaccuracies.

4. Kuki: Also known as Mitsuku, is a well-known AI chatbot for her conversational capabilities, taking several wins for the Loebner Prize Turing Test. It recognizes the feelings and emotions of text-based conversations and responds in the same vein. Sometimes, though, Kuki responds inconsistently to such emotions; it immediately breaks the illusion of understanding those very emotions. While it engages in small talk and detects basic emotions, it fails to keep an emotional context through longer conversations. Most responses Kuki provides are generalized to reflect emotions without the subtlety of real human emotion.

5. Cleverbot: It is an AI chatbot that learns through conversations and tries to offer responses based on how one should feel. It's been in development for many, many years and has an enormous amount of user interactions. Its responses regarding the emotional state of one are quite unpredictable, but it sometimes might be out of step with the user's feelings. This chatbot does not have real capabilities for understanding and simulating complicated emotional states; it relies on the usage of pre-learned phrases. Like Replika, Cleverbot stores conversations, which raises potential privacy concerns.

General Limitations of Emotion-Driven Chatbots:

1. Contextual Understanding: Most of these emotion-driven chatbots fail to understand context beyond immediate emotional cues and thereby can get out of step with the user's broader emotional state.

2. Emotion Detection Bias: These chatbots may show bias while detecting emotions if the training dataset is biased or restricted to specific demographics.

3. Emotional Intelligence: While chatbots are able to simulate emotional responses, they cannot actually possess emotional intelligence or empathy, and this seriously limits their effectiveness in more emotionally taxing applications.

4. Privacy and Security: Most of the emotionally driven chatbots record user information to analyze further for algorithm improvements; this becomes very concerning in data privacy and ethics in the usage of sensitive emotional information. The limitation furthers the difficulty in developing chatbots that can actually understand and respond to human emotions meaningfully.

1.3 Project Scope

1. Development of an emotion-detection algorithm that analyzes facial expressions from uploaded images.
2. Integration of natural language processing (NLP) techniques to interpret and respond to user input based on detected emotions.
3. Creation of a user-friendly web interface for interaction and image uploading.
4. Testing and validation of the chatbot’s ability to detect emotions accurately and respond appropriately.

2. SYSTEM ANALYSIS

2.1 FUNCTIONAL SPECIFICATION

1. **Emotion Detection:** The system shall correctly detect emotions such as happiness, sadness, anger, disgust, fear and pain from pictures that users upload or even pictures taken through a webcam. Using this information, a machine learning or deep learning model will be applied specifically to recognise emotions to yield accurate and trustworthy results. It will use an ML/DL model for emotion recognition.
2. **Chatbot Response:** Incorporating a pre-trained language model to understand and generate text similar to human-like language/slang. The system shall generate responses based on detected emotions, ensuring improvement in user experience. With the use of pre-trained language models like Groq, chatbots may produce replies that are appropriate and empathetic in context.
3. **User Interface:** A user-friendly, web-based interface will be provided that will allow users to either upload images or use a webcam to capture their facial expressions. In this way, this interface will enable live interaction with the chatbot and thus will be seamless and engaging. Integration at the backend will be done using Flask, which will be set up and used to manage the integration of the emotion detection model with the chatbot logic. Design endpoints that handle HTTP requests on emotion analysis and then create appropriate chatbot responses based on the emotions detected.
4. **Feedback Mechanism:** A feedback system shall also be added to obtain user input on how accurate the detected emotions are and how relevant the chatbot responses are. This feedback shall form a basis for continuously improving the models and enhancing the overall user experience.

2.2 BLOCK DIAGRAM

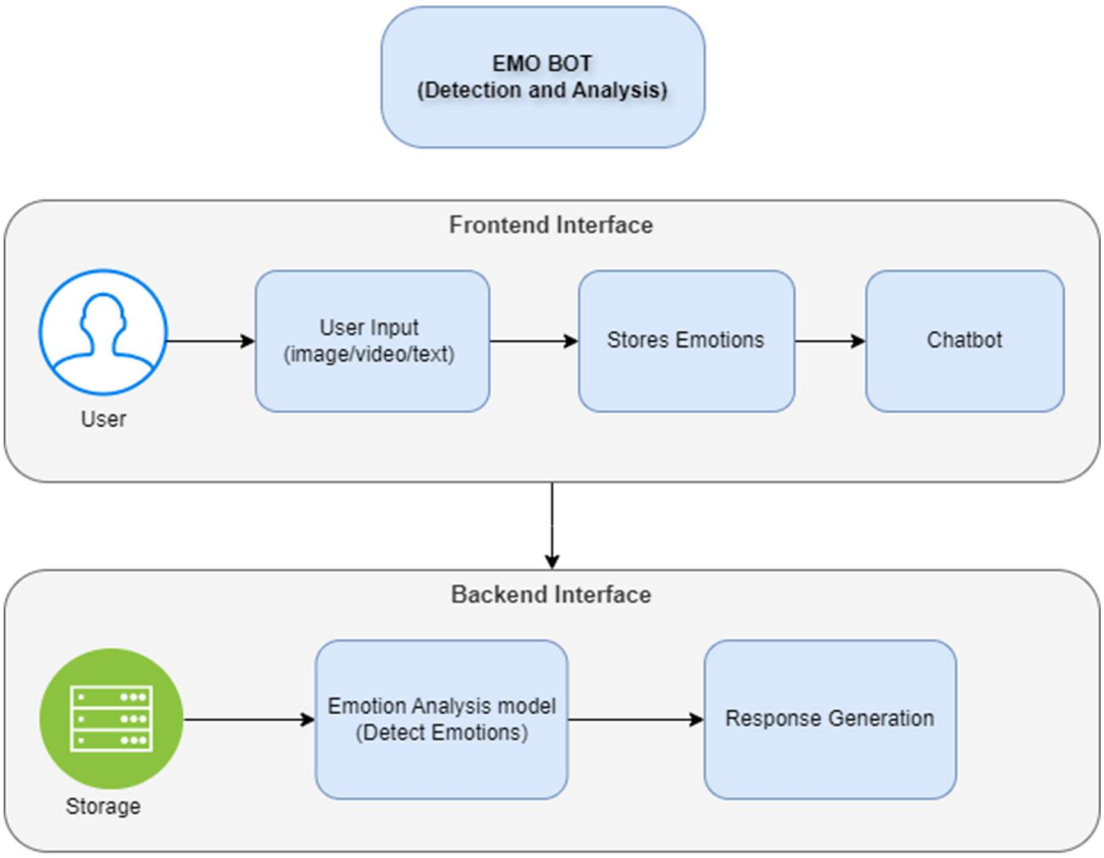


FIG 2.1

2.3 SYSTEM REQUIREMENTS

2.3.1 SOFTWARE REQUIREMENTS

1. Frontend:

   a. HTML, CSS, JavaScript
2. Backend:
   a. Python with Flask and MongoDB
3. Emotion Detection:
   a. TensorFlow or PyTorch
   b. OpenCV for image processing
4. NLP:
   a. Groq or similar language models

2.3.2 HARDWARE REQUIREMENTS

1. Server:
   a. Processor: Intel i5 or equivalent
   b. RAM: 8GB minimum
   c. Storage: 100GB minimum
2. User Devices:
   a. Webcam-enabled device for image capture
   b. Internet connection for web access

2.3.3 CONSTRAINTS

1. The system must comply with data privacy regulations for handling and storing user data.
2. The emotion detection model should be optimized to minimize latency and maximize accuracy.

2.3.4 Tools Survey

1. Development Tools:
   a. Visual Studio Code for code editing
   b. Git for version control
   c. Docker for containerisation and deployment

2. Testing Tools:
    a. Postman for API testing
    b. Selenium for frontend testing

3. SYSTEM DESIGN

3.1 SYSTEM ARCHITECTURE

The system architecture is designed to integrate the following: a web-based user interface, emotion detection module, and the backend of the chatbot. The architecture includes:

1. **Frontend Interface:** Built on HTML, CSS, and JavaScript; this interface enables any user to upload an image or click an image directly with the help of their webcam to capture his/her facial expression.
2. **Back-end Application:** This will be developed using Python, employing Flask for application development. It basically acts as an intermediary between the front-end interface and the modules for detecting human emotion and chatbots.
3. **Emotion Detection Module:** It would include designing machine learning/deep learning models using libraries like TensorFlow or PyTorch for image feature analysis to identify emotions. For image processing, OpenCV would be used.
4. **Chatbot Backend:** Utilizing pre-trained language models like Groq, this module will generate appropriate contextual responses to the detected emotion with empathy.
5. Feedback System: Include a mechanism for feedback that will help tune emotion detection and chat responses based on user input.

3.2 Module Design

1. **Emotion Detection Module:** Given an image file uploaded by the user or obtained through a webcam, this module is responsible for processing these images to detect emotions using a trained ML/DL model. The Chatbot Response Module is generating the responses based on the detected emotions using NLP to ensure truly empathetic responses.

2. **User Interface Module:** Used for user interaction with the system, the inclusion of images captured or uploaded from the computer and real-time images taken with a webcam leads to a perfect user experience.

3. **Feedback Module:** This is used to gather user feedback on the accuracy of the picked emotion and also relevance of the chatbot's responses. Feedback helps in building a system through continuous improvements.

1.3 DATABASE DESIGN

3.3.1 TABLE STRUCTURE

The tables in the database will store details of the user, detected emotions, chatbot responses, and feedback:

1. ***User Table:*** It stores the details of the user along with his interaction history.
2. ***Response Table:*** Store the conversation history conducted by the user and chatbot along with the relevant detected emotions in the process.
3. ***Feedback Table:*** To capture user ratings and feedback for every interaction to evaluate chatbot performance and user satisfaction.

3.3.2 DATA FLOW DIAGRAM

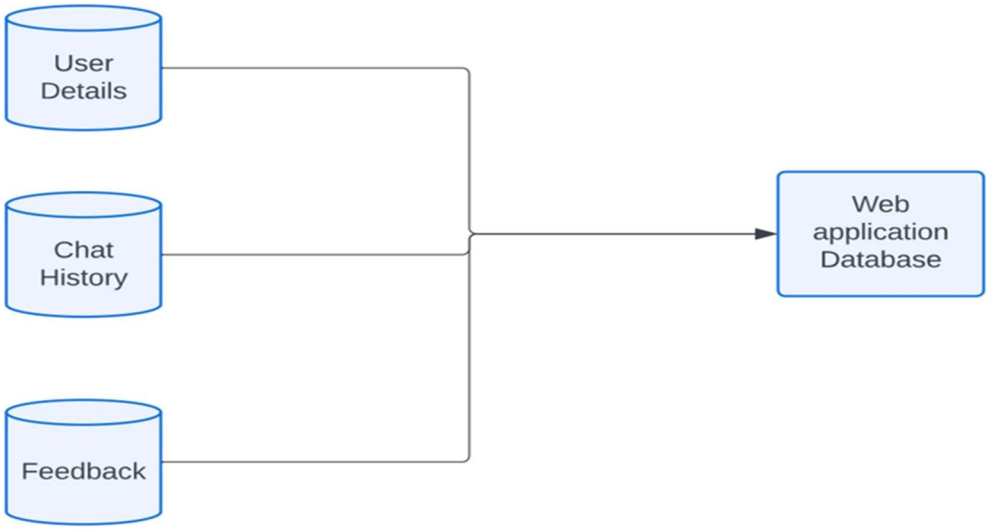
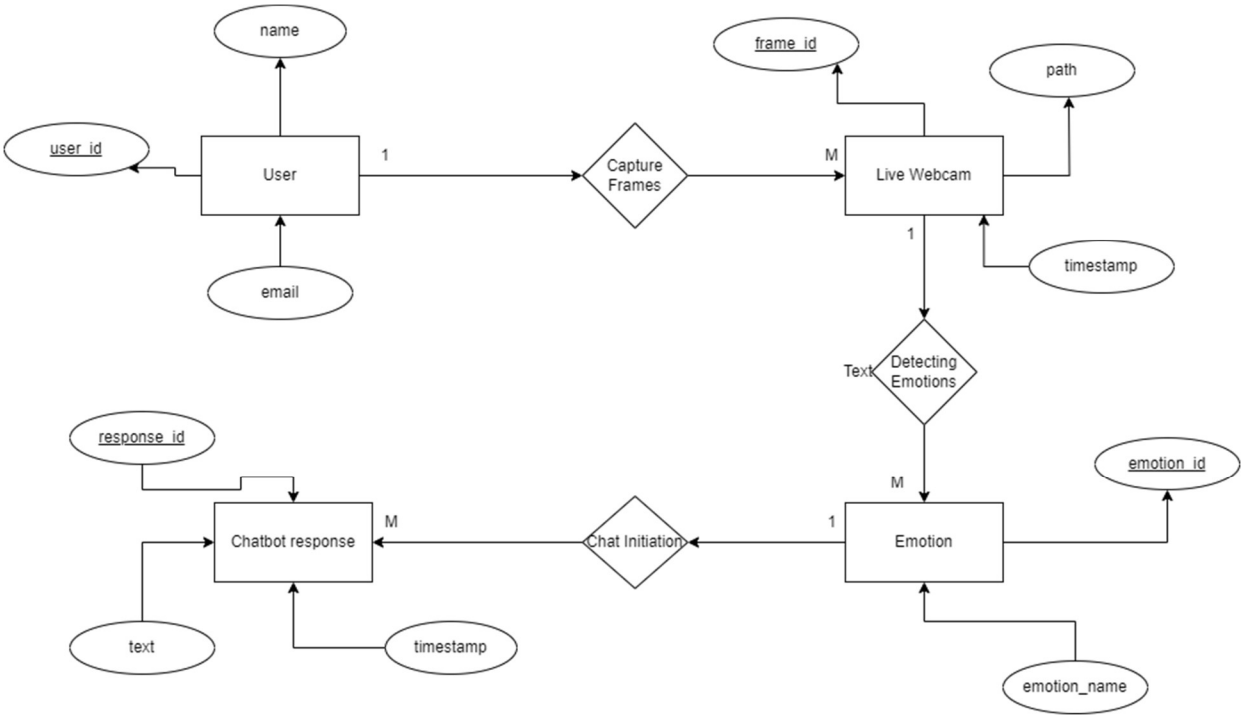


FIG 3.1

3.3.3 ER DIAGRAM



3.4 INTERFACE DESIGN

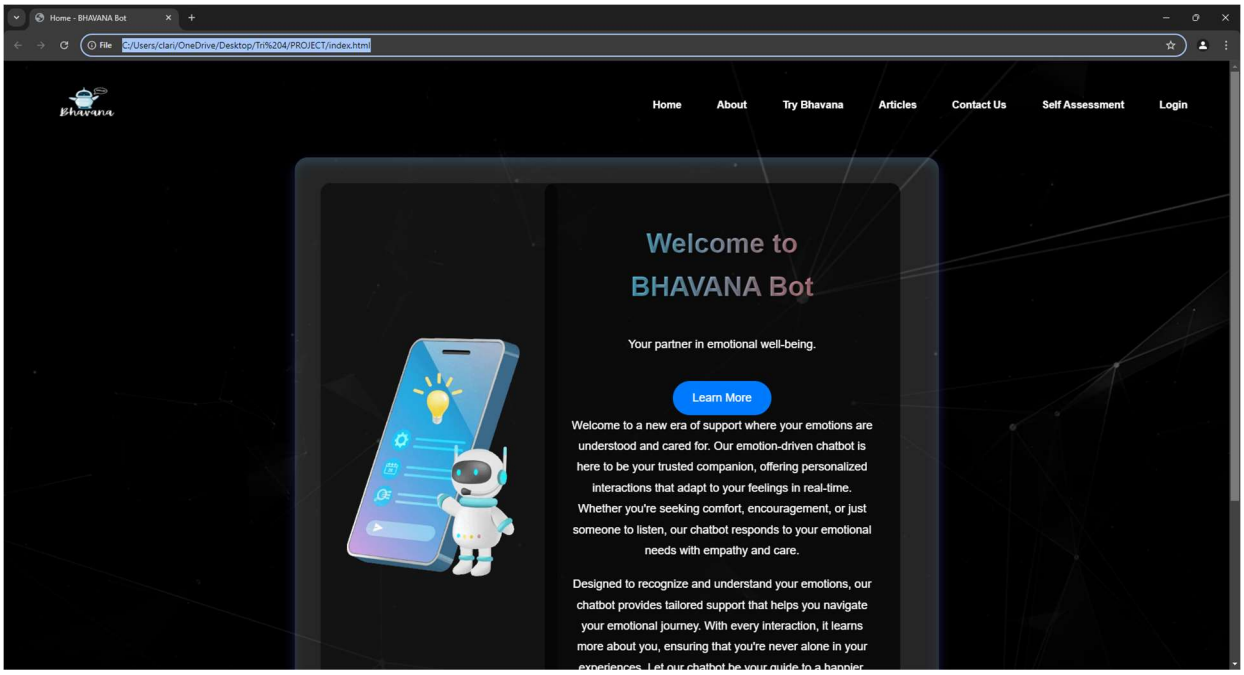
3.4.1 USER INTERFACE SCREEN DESIGN

The user interface (UI) for the website features a clean and intuitive design aimed at enhancing user experience. The design elements include:

- **Logo Container:** Positioned prominently at the top-left, displaying the logo of BHAVANA Bot, which helps in brand recognition and navigation.
- **Navbar:** A horizontal navigation bar located at the top of the page, providing easy access to different sections of the website. The navbar includes:

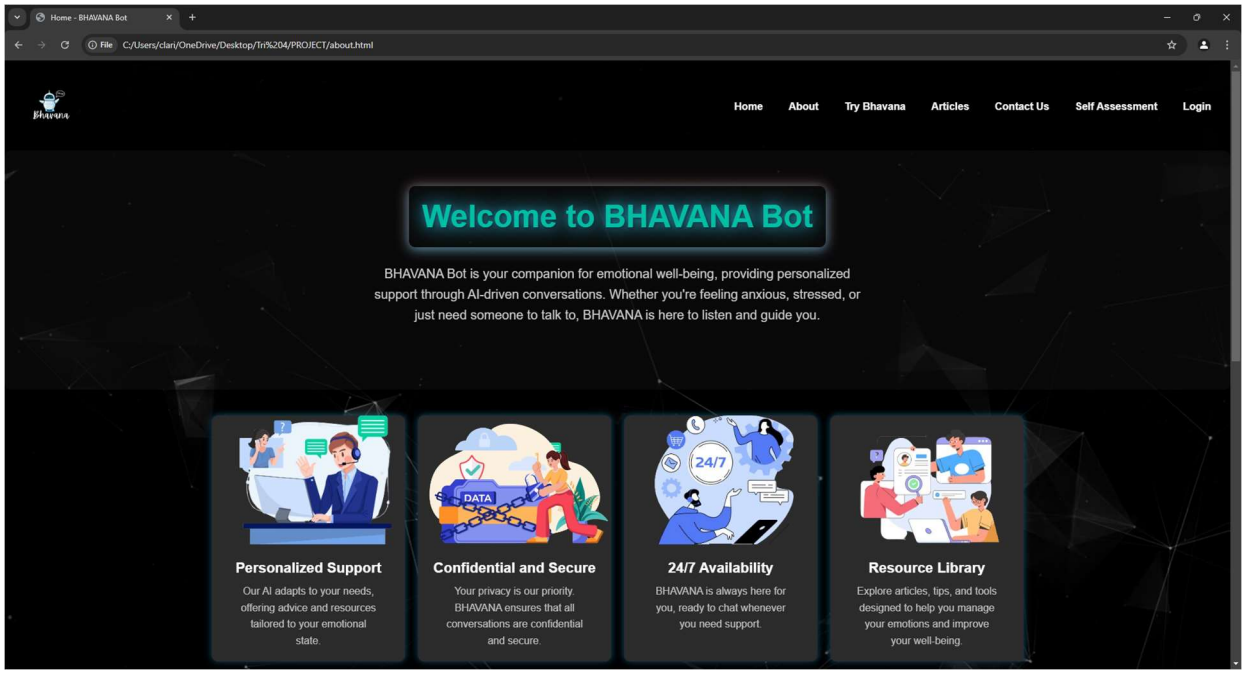
  ★.....

  Home : Link to the home page

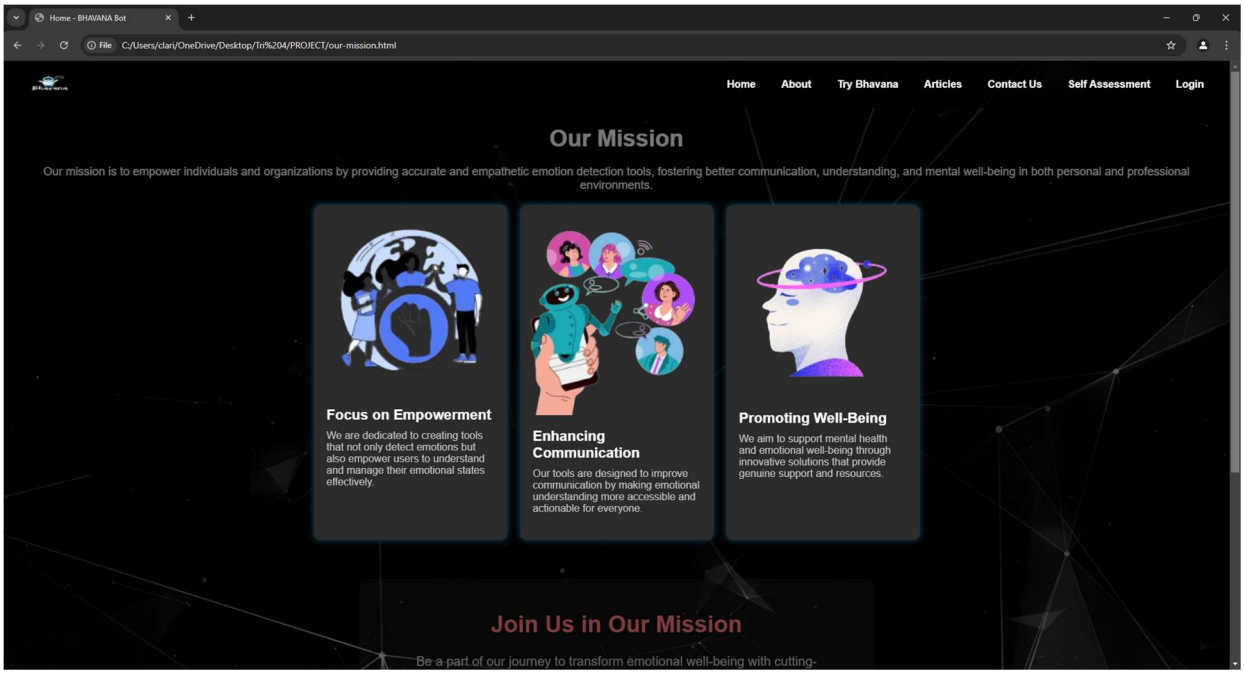


★.....

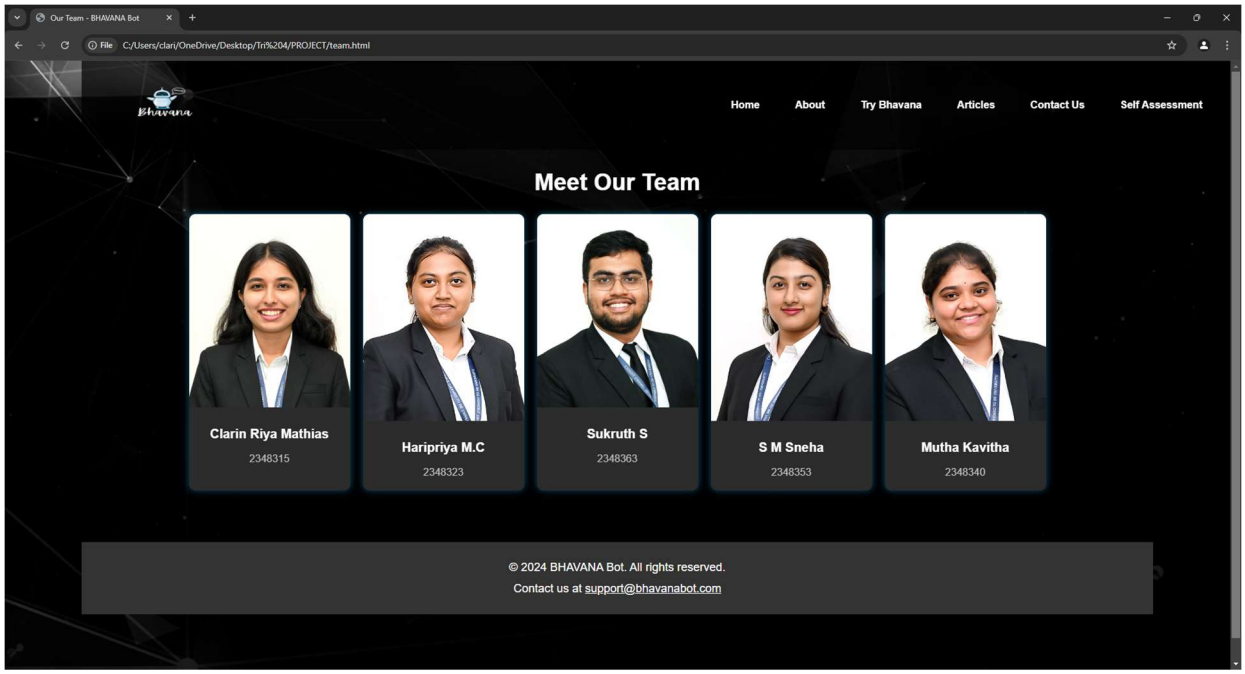
About : A dropdown menu with links to "Our Mission," "Our Team," and "Feedback."



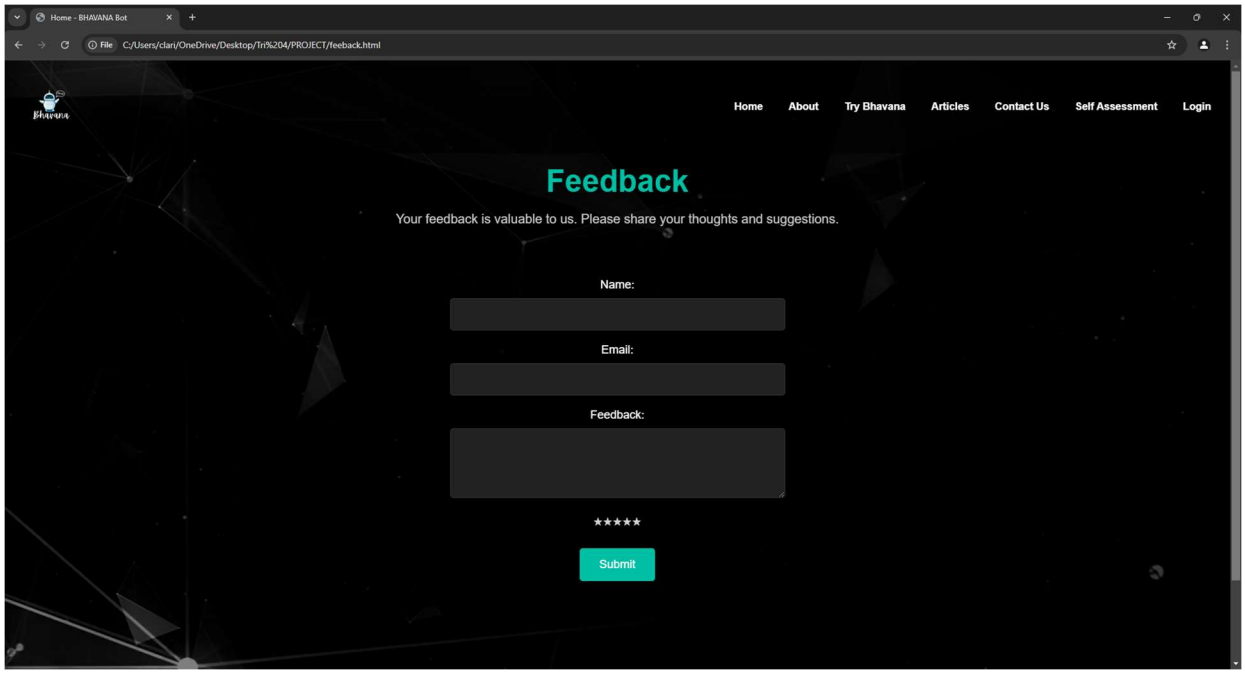
➔ Our Mission



➔ Our Team

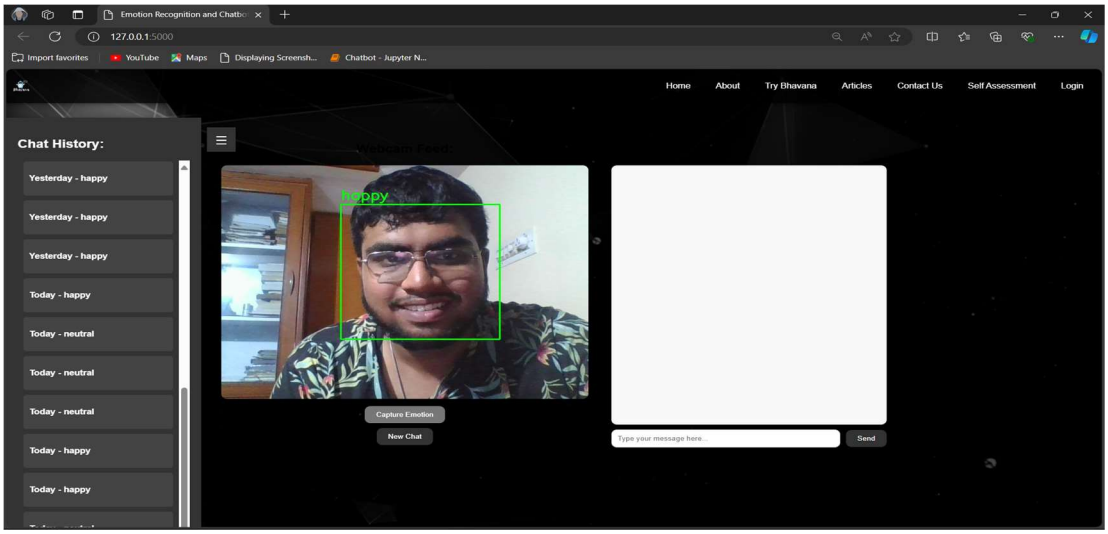


➔ Feedback



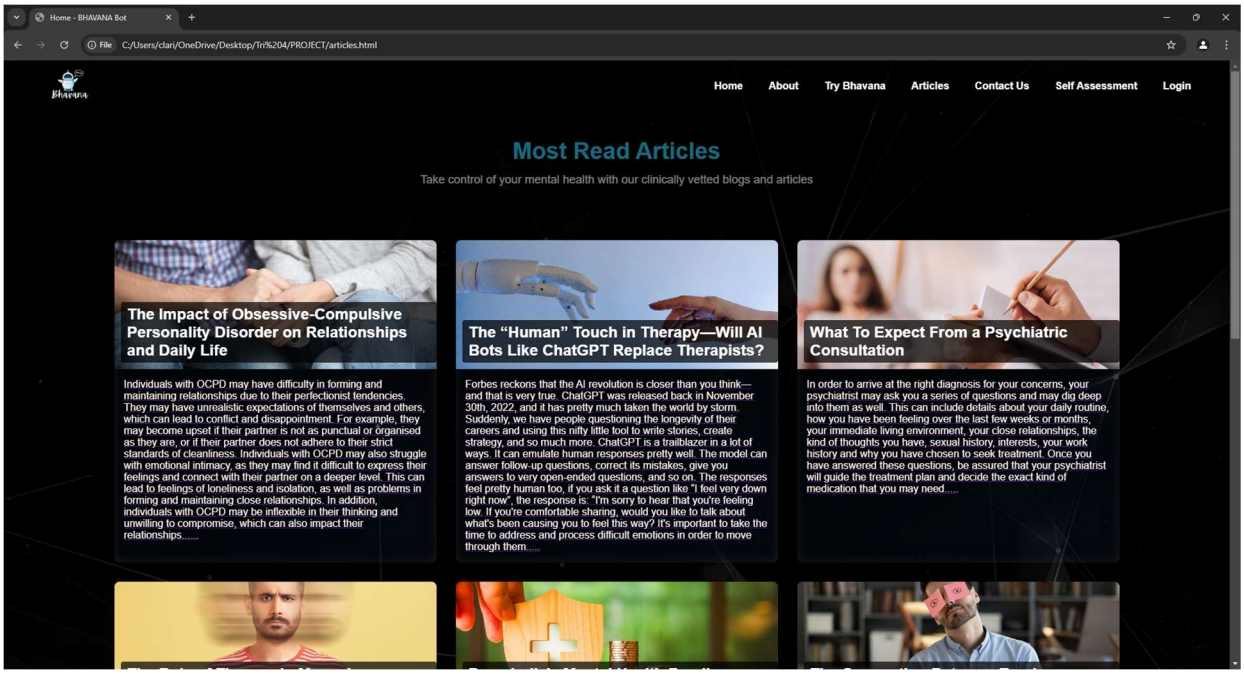
★.....

Try Bhavana: Directs users to the page where they can interact with Bhavana.



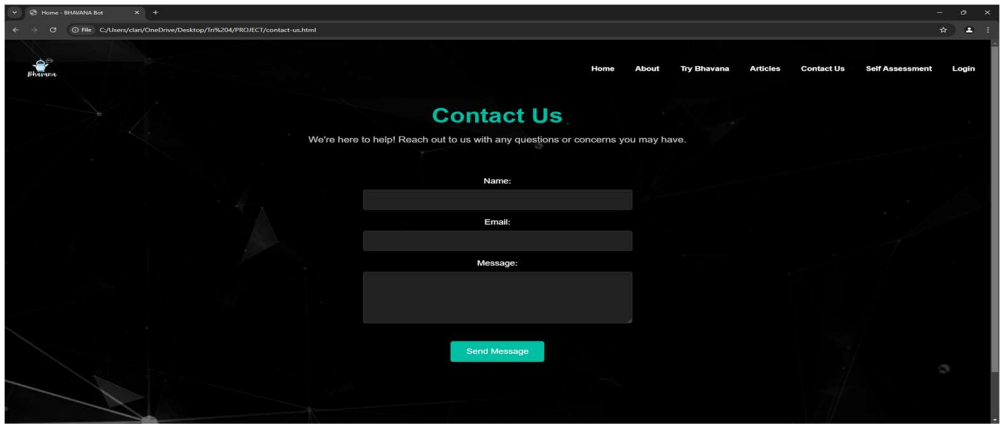
★.....

Articles: Link to the articles section for further reading.



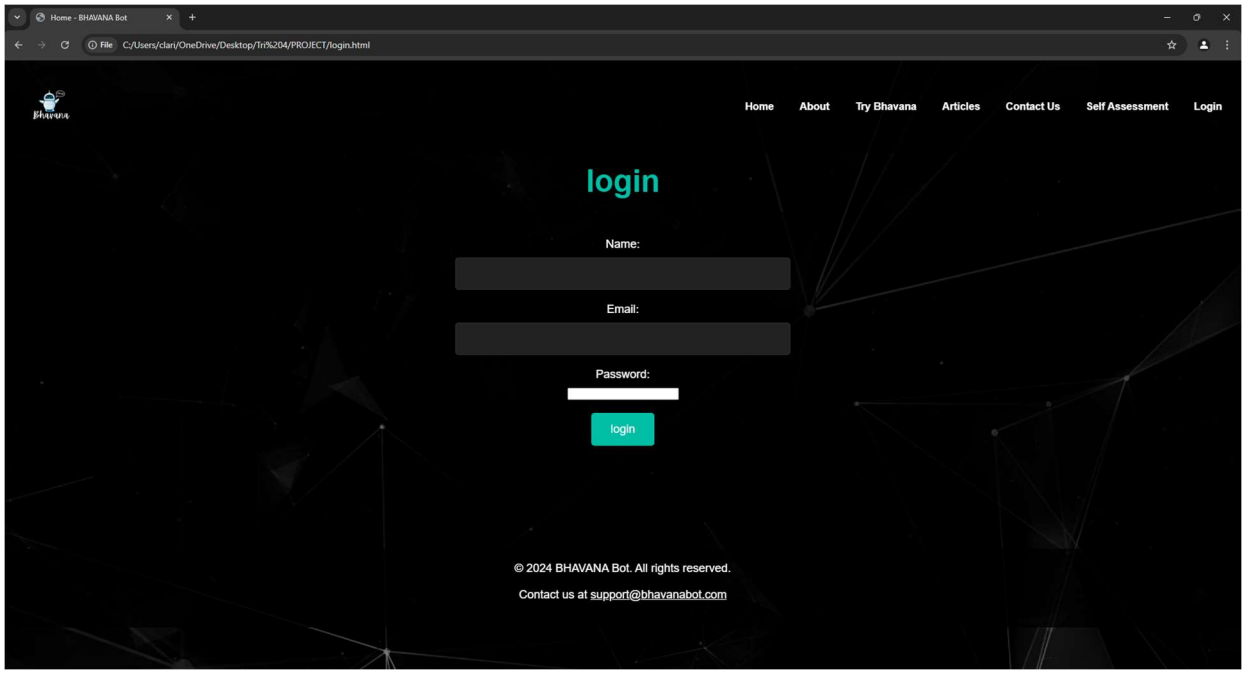
★.....

Contact Us: Provides a way to get in touch with the team.



★.....

Login: Option for users to log into their accounts.



3.4.2 CLASS DIAGRAM

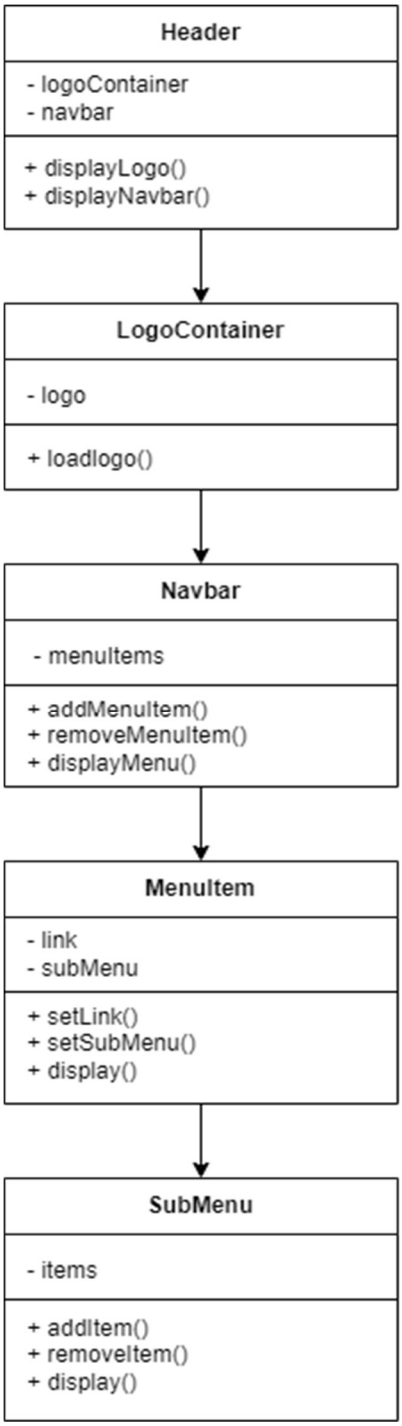

Header
- logoContainer
- navbar
+ displayLogo()
+ displayNavbar()
LogoContainer
- logo
+ loadlogo()
Navbar
- menuItems
+ addMenuItem()
+ removeMenuItem()
+ displayMenu()
MenuItem
- link
- subMenu
+ setLink()
+ setSubMenu()
+ display()
SubMenu
- items
+ addItem()
+ removeItem()
+ display()

FIG 3.3

4. IMPLEMENTATION

4.1 CODING STANDARDS

The programming language chosen for the implementation is Python. The coding standards for Python are provided by a document called PEP8. It is Python Enhancement Proposal 8. It is a document that provides coding conventions for the Python code. Some details from PEP8 are described as follows

1. Indentation: The guideline suggests using 4 spaces per indentation level.

2. Tabs or Spaces: Tabs should be used solely to remain consistent with code that is already indented with tabs. Python disallows mixing tabs and spaces for indentation.

3. Maximum Line Length: Limit all lines to a maximum of 79 characters.

4. Imports: Multiple Imports should usually be on separate lines. Also, imports are always put at the top of the file. Absolute imports are recommended, as they are usually more readable and tend to be better performed if the import system is incorrectly configured

5. Whitespace in Expressions and Statements: Avoid unnecessary whitespace.

6. Comments: Comments should be complete sentences and The first word should be capitalized, unless it is an identifier that begins with a lowercase letter. Block comments generally consist of one or more paragraphs built out of complete sentences, each ending in a period.

To create the interactive website, HTML and CSS is used.The conventions for HTML and CSS are:

1. Declaration of document type in HTML: It is essential to mention the type of document that the browser should display. This allows the browser to know what type of document it should render and thus use its resources.

2. Use of lowercase element names in HTML: However, HTML allows the use of lowercase or uppercase element names; we recommend you to opt for lowercase names as it looks clean and is easier to write.

3. Close all elements in HTML: HTML does not impose on closing all the elements, but it is better to do so.

4. Use Lowercase Attribute Names in HTML: Like elements, there is no clear distinction provided by HTML in using upper or lower case names for attributes. However, it is good practice to use lowercase letters for attribute names.

5. HTML Indentation and Blank lines: Using blank lines and indentation unnecessarily in an HTML document is not recommended. It is a better approach to add blank lines to separate blocks of code, and indentation should be limited to two spaces rather than the tab key.

4.2 SCREENSHOTS

4.2.1 EMOTION DETECTION MODEL INTEGRATED TO CHATBOT

App.py :

```
from flask import Flask, render_template, Response, request, jsonify
import cv2
from fer import FER
import google.generativeai as genai
import os
from datetime import datetime, timedelta
from pymongo import MongoClient
from bson.objectid import ObjectId

# Configure the API key for Gemini
genai.configure(api_key=os.environ["GEMINI_API_KEY"])

# Initialize the generative model
model = genai.GenerativeModel('gemini-1.5-flash')

# Initialize Flask app
app = Flask(__name__)

# Initialize FER detector
detector = FER()
```

```
camera = None  # Initialize the camera variable

# MongoDB setup
mongo_uri = 'mongodb://localhost:27017'
client = MongoClient(mongo_uri)
db = client['chat_history']
collection = db['History']

def start_camera():
    global camera
    if camera is None or not camera.isOpened():
        camera = cv2.VideoCapture(0)

def stop_camera():
    global camera
    if camera is not None and camera.isOpened():
        camera.release()

def generate_frames():
    global camera
    start_camera()  # Ensure the camera is started
    while True:
        success, frame = camera.read()
        if not success:
            break
        else:
            # Detect emotions in the frame
            emotion_data = detector.detect_emotions(frame)
            if emotion_data:
                for face in emotion_data:
                    (x, y, w, h) = face['box']
                    emotion_dict = face['emotions']
                    emotion, _ = max(emotion_dict.items(), key=lambda item: item[1])
                    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
                    cv2.putText(frame, f'{emotion}', (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

            ret, buffer = cv2.imencode('.jpg', frame)
            frame = buffer.tobytes()
```

```
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

@app.route('/')
def index():
    start_camera()  # Start the camera when visiting the index page
    return render_template('index.html')

@app.route('/video_feed')
def video_feed():
    return Response(generate_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/capture_emotion', methods=['POST'])
def capture_emotion():
    global camera
    success, frame = camera.read()
    if success:
        emotion_data = detector.detect_emotions(frame)
        if emotion_data:
            face = emotion_data[0]
            emotion_dict = face['emotions']
            detected_emotion, _ = max(emotion_dict.items(), key=lambda item: item[1])
            return jsonify({"captured_emotion": detected_emotion})
    return jsonify({"captured_emotion": "neutral"})

@app.route('/chatbot_response', methods=['POST'])
def chatbot_response():
    data = request.json
    user_name = data.get('user_name', 'User')
    detected_emotion = data.get('detected_emotion', 'neutral')
    user_input = data.get('user_input', '').lower()
    chat_id = data.get('chat_id')

    if user_input == '':
        prompt = f"Generate a friendly and super excited message from a chatbot named Bhavana. The user is
named {user_name} and is feeling {detected_emotion}. Please engage the user warmly."
    else:
        prompt = (f"Generate an enthusiastic and excited response from a chatbot named Bhavana based on the
user's emotion, which is {detected_emotion}. "
```

```
            f"The user has said: '{user_input}'. The response should be lively, conversational, and tailored to the user's current emotional state.")

    response = model.generate_content(prompt)
    response_text = response.text.strip()

    # Update chat history in MongoDB
    timestamp = datetime.now()

    if chat_id:
        collection.update_one(
            {'_id': ObjectId(chat_id)},
            {'$push': {'chat_history': {
                'timestamp': timestamp,
                'user_input': user_input,
                'response': response_text
            }}}
        )
    else:
        chat_entry = {
            'timestamp': timestamp,
            'user_name': user_name,
            'detected_emotion': detected_emotion,
            'chat_history': [
                {
                    'timestamp': timestamp,
                    'user_input': user_input,
                    'response': response_text
                }
            ]
        }
        result = collection.insert_one(chat_entry)
        chat_id = str(result.inserted_id)

    return jsonify({"response": response_text, "chat_id": chat_id})

@app.route('/new_chat', methods=['POST'])
def new_chat():
    start_camera()  # Reactivate the camera for the new chat
```

```
    return jsonify({"status": "new_chat_started"})

@app.route('/chat_history', methods=['GET'])
def chat_history():
    now = datetime.now()
    yesterday = now - timedelta(days=1)
    last_week = now - timedelta(days=7)

    history = collection.aggregate([
        {"$unwind": "$chat_history"},
        {"$group": {
            "_id": {
                "chat_id": "$_id",
                "emotion": "$detected_emotion",
                "time_class": {
                    "$cond": [
                        {"$eq": [{"$dateToString": {"format": "%Y-%m-%d", "date": "$timestamp"}},
now.strftime("%Y-%m-%d")]},
                        "Today",
                        {
                            "$cond": [
                                {"$eq": [{"$dateToString": {"format": "%Y-%m-%d", "date": "$timestamp"}},
yesterday.strftime("%Y-%m-%d")]},
                                "Yesterday",
                                {"$cond": [{"$gte": ["$timestamp", last_week]}, "Previous 7 Days", "Older"]}
                            ]
                        }
                    ]
                }
            },
            "first_message": {"$first": "$chat_history.user_input"},
            "chat_count": {"$sum": 1}
        }},
        {"$sort": {"_id.time_class": -1, "_id.chat_id": -1}}  # Sort by timeline descending
    ])
    history_list = list(history)
    for h in history_list:
        h['_id']['chat_id'] = str(h['_id']['chat_id'])
    return jsonify(history_list)
```

```
@app.route('/load_chat/<chat_id>', methods=['GET'])
def load_chat(chat_id):
    chat = collection.find_one({"_id": ObjectId(chat_id)})
    if not chat:
        return jsonify({"status": "error", "message": "Chat not found"}), 404

    return jsonify({
        "_id": {"$oid": str(chat['_id'])},
        "chat_history": chat.get("chat_history", [])
    })


if __name__ == "__main__":

    app.run(debug=True)
```

Self Assessment Test Coding

```
import fitz  # PyMuPDF for PDF reading

def get_questionnaire_from_pdf(pdf_path):
    questions = {
        'stress': [
            "Been upset because of something that happened unexpectedly?\na. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often",
            "In the LAST MONTH, how often have you: Felt that you were unable to control important things in your life?\na. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often",
            "In the LAST MONTH, how often have you: Felt nervous and 'stressed'?\na. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often",
            "In the LAST MONTH, how often have you: Felt confident about your ability to handle your personal problems?\na. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often",
            "In the LAST MONTH, how often have you: Felt that things were going your way?\na. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often",
            "In the LAST MONTH, how often have you: Found that you could NOT cope with all the things you had to do?\na. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often",
            "In the LAST MONTH, how often have you: Been able to control irritations in your life?\na. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often",
            "In the LAST MONTH, how often have you: Felt that you were on top of things?\na. Never\nb. Almost
```

```
Never\nc. Sometimes\nd. Fairly Often\ne. Very Often",
        "In the LAST MONTH, how often have you: Been angered because of things that happened that were out of your control?\na. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often",
        "In the LAST MONTH, how often have you: Felt difficulties were piling up so high that you could not overcome them?\na. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often"
    ],
    'anxiety': [
        "I find it very hard to unwind, relax or sit still\na. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often",
        "I have had stomach problems, such as feeling sick or stomach cramps\na. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often",
        "I have been irritable and easily become annoyed\na. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often",
        "I have experienced shortness of breath\na. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often",
        "I have felt dizzy and unsteady at times\na. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often",
        "I have had difficulties with sleep (including waking early, finding it hard to go to sleep)\na. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often",
        "I have felt panicked and overwhelmed by things in my life\na. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often",
        "I have felt nervous and on edge\na. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often",
        "I have had trembling hands\na. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often",
        "I seem to be constantly worrying about things\na. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often"
    ],
    'depression': [
        "I feel overwhelmingly sad at times\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree",
        "When I think of the future I feel hopeless\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree",
        "I feel like a complete failure\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree",
        "I get a lot of satisfaction/joy from doing things\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree",
        "I feel guilty about something most of the time\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree",
        "I feel like I am being punished\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely
```

```
disagree",
        "I feel disappointed (even disgusted) with myself\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree",
        "The bad things in my life aren't all my fault\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree",
        "I am often on the brink of tears or cry\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree",
        "I feel irritated and annoyed by things in my life\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree",
        "I am very interested in other people's lives and like to listen to them\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree",
        "I find it easy to make decisions, big and small\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree",
        "I think I am unattractive or ugly\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree",
        "I find it really hard to do anything, especially work\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree",
        "My sleep patterns have been really disrupted\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree",
        "I am so tired I don't have the energy to do anything\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree",
        "My appetite has changed a lot\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree",
        "I have lost a lot of weight\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree",
        "I am very concerned, even preoccupied, with my physical health\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree",
        "I am not interested in sexual relations at all\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree",
        "I have thought about ending my life\na. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree"
    ]
  }

  return questions


def calculate_score(answers, test_type):
  # Define scoring values for answers
```

```
    score_values = {'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4}
    total_score = sum(score_values.get(answer, 0) for answer in answers)

    # Define score ranges and recommendations for each test type
    recommendations = {
        'stress': [
            ("Low to Mild Stress", 0, 7),
            ("Mild Stress", 8, 15),
            ("Moderate Stress", 16, 23),
            ("Severe Stress", 24, 31, True),
            ("Extremely Severe Stress", 32, 40, True)
        ],
        'anxiety': [
            ("Unlikely", 0, 10),
            ("May be suffering", 11, 26),
            ("Likely", 27, 40, True)
        ],
        'depression': [
            ("Little / none", 0, 10),
            ("Mild", 11, 16),
            ("Moderate", 17, 20),
            ("High", 21, 30),
            ("High to severe", 31, 63, True)
        ]
    }

    for level, min_score, max_score, *recommend in recommendations[test_type]:
        if min_score <= total_score <= max_score:
            print(f"Chatbot: Your score is {total_score}, which falls under '{level}'.")
            if recommend and recommend[0]:
                recommend_doctor(test_type)
            else:
                print("Chatbot: You are safe, but if you feel concerned, consider speaking to a professional.")
            break

def recommend_doctor(test_type):
    doctor_recommendations = {
        'stress': [
            ("Dr Parul Mathur", "912071171501", "https://maps.app.goo.gl/eKrNHPcJfe3LK1pc6", "5+ years"),
```

```
            ("Dr Shruti Nair", "912071171501", "https://maps.app.goo.gl/eKrNHPcJfe3LK1pc6", "7+ years"),
            ("Dr Soumya Parameshwaran", "912071171501", "https://maps.app.goo.gl/LV9tdkYxxWTsFa4H9", "12+
years"),
            ("Dr Sharadhi C", "912071171501", "https://maps.app.goo.gl/LV9tdkYxxWTsFa4H9", "6+ years")
        ],
        'anxiety': [
            ("Dr Ahmed Reshamvala", "912071171501", "https://maps.app.goo.gl/ZYqTPhM9eyso9g2d7", "1+
years"),
            ("Dr Dhruva Ithal", "912071171501", "https://maps.app.goo.gl/eKrNHPcJfe3LK1pc6", "11+ years"),
            ("Dr Anand Jayaraman", "918071966822", "https://g.co/kgs/CmsYz8R", "27 years"),
            ("Dr Harsha GT", "919513610115", "https://maps.app.goo.gl/UZcvr8Adb9hyU6HT6", "11 years")
        ],
        'depression': [
            ("Ms Usha Sathish", "911141193943", "https://maps.app.goo.gl/A9e8uiPSPpTA16Tz9", "1+ years"),
            ("Dr Venkatesh Babu G M", "918071910148", "https://maps.app.goo.gl/EB36Wz5uZsm8FWMXA", "18
years"),
            ("Ms Anunka Mondal", "918045685300", "https://maps.app.goo.gl/EB36Wz5uZsm8FWMXA", "5
years"),
            ("Dr Ravi Prakash", "08071909298", "https://maps.app.goo.gl/QgeccTmRxFzzikFp9", "20 years")
        ]
    }

    doctors = doctor_recommendations.get(test_type, [])
    if doctors:
        print("Chatbot: Based on your score, I recommend consulting the following doctors:")
        for doctor in doctors:
            name, phone, location, experience = doctor
            print(f"- {name} (Phone: {phone}, Experience: {experience})")
            print(f"  Location: {location}")
    else:
        print("Chatbot: No specific recommendations available for this test type.")

def chatbot_conversation():
    print("Chatbot: Hey, do you want to take a test? (yes/no)")
    user_input = input("You: ").strip().lower()

    if user_input == "yes":
        print("Chatbot: Great! You can choose from the following tests: stress, anxiety, or depression. Which one
would you like to take?")
```

```
    test_type = input("You: ").strip().lower()

    # Define options globally within the function to be reused across different tests
    options = {
        "a": 0,  # Never
        "b": 1,  # Almost Never
        "c": 2,  # Sometimes
        "d": 3,  # Fairly Often
        "e": 4   # Very Often
    }

    depression_options = {
        "a": 3,  # Definitely agree
        "b": 2,  # Slightly agree
        "c": 1,  # Slightly disagree
        "d": 0   # Definitely disagree
    }

    if test_type == "stress":
        print("Chatbot: Let's start the stress test!")
        questions = [
            "Been upset because of something that happened unexpectedly?",
            "Felt that you were unable to control important things in your life?",
            "Felt nervous and 'stressed'?",
            "Felt confident about your ability to handle your personal problems?",
            "Felt that things were going your way?",
            "Found that you could NOT cope with all the things you had to do?",
            "Been able to control irritations in your life?",
            "Felt that you were on top of things?",
            "Been angered because of things that happened that were out of your control?",
            "Felt difficulties were piling up so high that you could not overcome them?"
        ]
        score = 0

        for question in questions:
            while True:
                print(f"Chatbot: {question}")
                print("a. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often")
                answer = input("You: ").strip().lower()
```

```
            if answer in options:
                score += options[answer]
                break
            else:
                print("Chatbot: Please provide a valid option: a, b, c, d, or e.")

    # Scoring for Stress
    if 0 <= score <= 7:
        result = "Low to Mild Stress"
    elif 8 <= score <= 15:
        result = "Mild Stress"
    elif 16 <= score <= 23:
        result = "Moderate Stress"
    elif 24 <= score <= 31:
        result = "Severe Stress"
    elif 32 <= score <= 40:
        result = "Extremely Severe Stress"

    print(f"Chatbot: Your stress level is: {result}")
    print(f"Chatbot: Your score is {score}.")

    # Display detailed scoring information
    print("\nThis is the score for stress:")
    print("Low to Mild Stress:\nScore Range: 0 to 7")
    print("Mild Stress:\nScore Range: 8 to 15")
    print("Moderate Stress:\nScore Range: 16 to 23")
    print("Severe Stress:\nScore Range: 24 to 31")
    print("Extremely Severe Stress:\nScore Range: 32 to 40")

    if result in ["Severe Stress", "Extremely Severe Stress"]:
        print("Chatbot: Based on your score, it's recommended to consult a doctor.")
        recommend_doctor('stress')
    else:
        print("Chatbot: Your stress level is manageable. Remember to take care of yourself!")

elif test_type == "anxiety":
    print("Chatbot: Let's start the anxiety test!")
    questions = [
```

```
    "I find it very hard to unwind, relax or sit still",
    "I have had stomach problems, such as feeling sick or stomach cramps",
    "I have been irritable and easily become annoyed",
    "I have experienced shortness of breath",
    "I have felt dizzy and unsteady at times",
    "I have had difficulties with sleep (including waking early, finding it hard to go to sleep)",
    "I have felt panicked and overwhelmed by things in my life",
    "I have felt nervous and on edge",
    "I have had trembling hands",
    "I seem to be constantly worrying about things"
]
score = 0

for question in questions:
    while True:
        print(f"Chatbot: {question}")
        print("a. Never\nb. Almost Never\nc. Sometimes\nd. Fairly Often\ne. Very Often")
        answer = input("You: ").strip().lower()

        if answer in options:
            score += options[answer]
            break
        else:
            print("Chatbot: Please provide a valid option: a, b, c, d, or e.")

# Scoring for Anxiety
if 0 <= score <= 10:
    result = "Unlikely"
elif 11 <= score <= 26:
    result = "May be suffering"
elif 27 <= score <= 40:
    result = "Likely"

print(f"Chatbot: Your anxiety level is: {result}")
print(f"Chatbot: Your score is {score}.")

# Display detailed scoring information
print("\nThis is the score for anxiety:")
print("Unlikely:\nScore Range: 0 to 10")
```

```
        print("May be suffering:\nScore Range: 11 to 26")
        print("Likely:\nScore Range: 27 to 40")

        if result == "Likely":
            print("Chatbot: Based on your score, it's recommended to consult a doctor.")
            recommend_doctor('anxiety')
        else:
            print("Chatbot: Your anxiety level is manageable. Consider speaking to a professional if needed.")

    elif test_type == "depression":
        print("Chatbot: Let's start the depression test!")
        questions = [
            "I feel overwhelmingly sad at times",
            "When I think of the future I feel hopeless",
            "I feel like a complete failure",
            "I get a lot of satisfaction / joy from doing things",
            "I feel guilty about something most of the time",
            "I feel like I am being punished",
            "I feel disappointed (even disgusted) with myself",
            "The bad things in my life aren't all my fault",
            "I am often on the brink of tears or cry",
            "I feel irritated and annoyed by things in my life",
            "I am very interested in other people's lives and like to listen to them",
            "I find it easy to make decisions, big and small",
            "I think I am unattractive or ugly",
            "I find it really hard to do anything, especially work",
            "My sleep patterns have been really disrupted",
            "I am so tired I don't have the energy to do anything",
            "My appetite has changed a lot",
            "I have lost a lot of weight",
            "I am very concerned, even preoccupied, with my physical health",
            "I am not interested in sexual relations at all",
            "I have thought about ending my life"
        ]
        score = 0

        for question in questions:
            while True:
                print(f"Chatbot: {question}")
```

```
            print("a. Definitely agree\nb. Slightly agree\nc. Slightly disagree\nd. Definitely disagree")
            answer = input("You: ").strip().lower()

            if answer in depression_options:
                score += depression_options[answer]
                break
            else:
                print("Chatbot: Please provide a valid option: a, b, c, or d.")

    # Scoring for Depression
    if 0 <= score <= 10:
        result = "Little / none"
    elif 11 <= score <= 16:
        result = "Mild"
    elif 17 <= score <= 20:
        result = "Moderate"
    elif 21 <= score <= 30:
        result = "High"
    elif 31 <= score <= 63:
        result = "High to severe"

    print(f"Chatbot: Your depression level is: {result}")
    print(f"Chatbot: Your score is {score}.")

    # Display detailed scoring information
    print("\nThis is the score for depression:")
    print("Little / none:\nScore Range: 0 to 10")
    print("Mild:\nScore Range: 11 to 16")
    print("Moderate:\nScore Range: 17 to 20")
    print("High:\nScore Range: 21 to 30")
    print("High to severe:\nScore Range: 31 to 63")

    if result == "High to severe":
        print("Chatbot: Based on your score, it's recommended to consult a doctor.")
        recommend_doctor('depression')
    else:
        print("Chatbot: Your depression level is manageable. Consider reaching out for support if needed.")

else:
```

```
            print("Chatbot: That's not a valid test type. Please choose either stress, anxiety, or depression.")

    else:
        print("Chatbot: Alright, feel free to reach out if you change your mind or if you need any other assistance. Goodbye!")

# To run the conversation function
chatbot_conversation()
```

4.2.2 Frontend Coding

Index.html

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Emotion Recognition and Chatbot</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>

<body>
    <!-- Navbar -->
    <header>
        <div class="logo-container">
            <img src="{{ url_for('static', filename='images/logo.png') }}" alt="BHAVANA Bot" class="logo">
        </div>
        <nav class="navbar">
            <ul>
                <li><a href="index.html">Home</a></li>
                <li>
                    <a href="about.html">About</a>
                    <ul>
                        <li><a href="our-mission.html">Our Mission</a></li>
                        <li><a href="team.html">Our Team</a></li>
                        <li><a href="feedback.html">Feedback</a></li>
                    </ul>
```

```
            </li>
            <li><a href="bhavana.html">Try Bhavana</a></li>
            <li><a href="articles.html">Articles</a></li>
            <li><a href="contact-us.html">Contact Us</a></li>
            <li><a href="sa.html">Self Assessment</a></li>
            <li><a href="login.html">Login</a></li>
        </ul>
    </nav>
</header>

<div class="container">
    <!-- Background Video -->
    <video autoplay muted loop id="background-video">
        <source src="{{ url_for('static', filename='images/Dark Background Web.mp4') }}" type="video/mp4">
        Your browser does not support HTML5 video.
    </video>

    <!-- Sidebar -->
    <div class="slidebar" id="sidebar">
        <button id="toggle-sidebar" onclick="toggleSidebar()">☰</button>
        <h2>Chat History:</h2>
        <div id="history-sidebar">
            <!-- Chat history list will be inserted here -->
        </div>
    </div>

    <!-- Webcam Feed -->
    <div class="webcam-feed">
        <h2>Webcam Feed:</h2>
        <img id="webcam" src="{{ url_for('video_feed') }}" width="640" height="480">
        <button id="capture-button" onclick="captureEmotion()" disabled>Capture Emotion</button>
        <button id="new-chat-button" onclick="startNewChat()">New Chat</button>
    </div>

    <!-- Chat Box -->
    <div class="chat-box">
        <h2>Chat with Bhavana:</h2>
        <div id="chat-response">
            <!-- Initial message will be inserted here after emotion is captured -->
```

```
        </div>
        <form id="chat-form">
            <input type="text" id="user_input" placeholder="Type your message here...">
            <button type="submit">Send</button>
        </form>
    </div>
</div>
<script>
    const userName = 'Sukruth';
    let detectedEmotion = 'neutral';  // This will be updated after capturing the emotion
    let chatId = null;  // This will be updated after the first chat interaction
    let capturingEmotion = false;  // Flag to prevent multiple captures

    // Capture emotion and start the chat
    function captureEmotion() {
        if (capturingEmotion) return;
        capturingEmotion = true;

        document.getElementById('capture-button').disabled = true;

        fetch('/capture_emotion', {
            method: 'POST',
        })
        .then(response => response.json())
        .then(data => {
            detectedEmotion = data.captured_emotion;
            alert(`Emotion detected: ${detectedEmotion}`); // Send an alert with the detected emotion
            freezeCamera();  // Freeze the camera feed
            startChatWithEmotion();  // Start chat with the captured emotion
        });
    }

    // Start the initial chat with detected emotion
    function startChatWithEmotion() {
        const userInput = '';  // Initial chat does not have user input
        freezeCamera();  // Freeze the camera feed

        fetch('/chatbot_response', {
            method: 'POST',
```

```
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify({
                user_name: userName,
                detected_emotion: detectedEmotion,
                user_input: userInput,
                chat_id: chatId
            }),
        })
        .then(response => response.json())
        .then(data => {
            chatId = data.chat_id;  // Update the chat ID
            displayMessage(data.response, 'bhavana');  // Display Bhavana's response
        });
    }

    // Display a message in the chatbox
    function displayMessage(message, sender = 'user') {
        const chatResponse = document.getElementById('chat-response');
        const messageElement = document.createElement('div');
        messageElement.className = sender === 'bhavana' ? 'bhavana-message' : 'user-message';

        if (sender === 'bhavana') {
            messageElement.innerHTML = `<img src="{{ url_for('static', filename='images/logo.png') }}"
class="bhavana-logo"> <p>${message}</p>`;
        } else {
            messageElement.innerHTML = `<p>${message}</p>`;
        }

        chatResponse.appendChild(messageElement);
        chatResponse.scrollTop = chatResponse.scrollHeight;  // Auto-scroll to the bottom
    }

    // Send user message and get chatbot response
    document.getElementById('chat-form').addEventListener('submit', function(event) {
        event.preventDefault();
        const userInput = document.getElementById('user_input').value.trim();
        if (userInput === '') return;
```

```
    displayMessage(userInput, 'user');  // Display user's message
    document.getElementById('user_input').value = '';  // Clear the input field

    // Send the user's message to the server
    fetch('/chatbot_response', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({
            user_name: userName,
            detected_emotion: detectedEmotion,
            user_input: userInput,
            chat_id: chatId
        }),
    })
    .then(response => response.json())
    .then(data => {
        chatId = data.chat_id;  // Update the chat ID
        displayMessage(data.response, 'bhavana');  // Display Bhavana's response
    });
});

// Start a new chat and reactivate the camera
function startNewChat() {
    capturingEmotion = false;
    document.getElementById('capture-button').disabled = false;

    fetch('/new_chat', {
        method: 'POST',
    })
    .then(response => response.json())
    .then(() => {
        chatId = null;  // Reset chat ID for a new conversation
        document.getElementById('chat-response').innerHTML = '';  // Clear the chat box
        startCamera();  // Reactivate the camera feed
    });
}
```

```
// Freeze the camera feed by stopping the video stream
function freezeCamera() {
    const webcam = document.getElementById('webcam');
    webcam.src = '';  // Stop the video feed
}

// Reactivate the camera feed
function startCamera() {
    const webcam = document.getElementById('webcam');
    webcam.src = "{{ url_for('video_feed') }}";  // Restart the video feed
}

// Toggle the visibility of the sidebar
function toggleSidebar() {
    const sidebar = document.getElementById('sidebar');
    sidebar.classList.toggle('active');
}

// Load chat history
fetch('/chat_history', {
    method: 'GET',
})
.then(response => response.json())
.then(data => {
    const historySidebar = document.getElementById('history-sidebar');
    historySidebar.innerHTML = '';  // Clear previous history

    data.forEach(chat => {
        const chatItem = document.createElement('div');
        chatItem.className = 'chat-item';
        const title = `${chat._id.time_class} - ${chat._id.emotion}`;  // Use the captured emotion in the title
        chatItem.innerHTML = `<p><strong>${title}</strong></p>`;
        chatItem.addEventListener('click', () => loadChat(chat._id.chat_id));
        historySidebar.appendChild(chatItem);
    });
});

// Load a previous chat by its ID
```

```
        function loadChat(chatId) {
            fetch(`/load_chat/${chatId}`, {
                method: 'GET',
            })
            .then(response => response.json())
            .then(data => {
                const chatResponse = document.getElementById('chat-response');
                chatResponse.innerHTML = '';  // Clear the chat box

                data.chat_history.forEach(chat => {
                    displayMessage(chat.user_input, 'user');
                    displayMessage(chat.response, 'bhavana');
                });

                chatId = data._id.$oid;  // Set the chat ID to the loaded chat
            });
        }
    </script>
</body>
</html>
```

Styles.css

```
/* Global Styles */
body {
  font-family: Arial, sans-serif;
  background-color: #f4f4f4;
  margin: 0;
  padding: 0;
  display: flex;
  flex-direction: column;
  align-items: center;
  overflow-x: hidden;
}

/* Background Video */
#background-video {
  position: fixed;
```

```
  top: 0;
  left: 0;
  min-width: 100%;
  min-height: 100%;
  z-index: -1;
  filter: brightness(60%);
}

/* Navbar Styles */
header {
  width: 100%;
  background-color: rgba(0, 0, 0, 0.8);
  padding: 10px 20px;
  position: fixed;
  top: 0;
  z-index: 1000;
  display: flex;
  align-items: center;
  justify-content: space-between;
  box-shadow: 0px 2px 10px rgba(0, 0, 0, 0.5);
}

.logo-container {
  display: flex;
  align-items: center;
}

.logo {
  height: 50px;
  width: auto;
  margin-right: 20px;
}

.navbar ul {
  list-style: none;
  padding: 0;
  margin: 0;
  display: flex;
  align-items: center;
```

```
}

.navbar ul li {
  position: relative;
  margin-right: 20px;
}

.navbar ul li a {
  text-decoration: none;
  color: #fff;
  padding: 8px 12px;
  transition: background-color 0.3s, color 0.3s;
}

.navbar ul li a:hover {
  background-color: #555;
  border-radius: 5px;
}

.navbar ul li ul {
  position: absolute;
  top: 100%;
  left: 0;
  background-color: rgba(0, 0, 0, 0.9);
  border-radius: 5px;
  display: none;
  padding: 10px;
}

.navbar ul li:hover ul {
  display: block;
}

.navbar ul li ul li {
  margin-right: 0;
  width: 200px;
}

.navbar ul li ul li a {
```

```
  display: block;
  padding: 10px;
  color: #fff;
}

/* Container Styles */
.container {
  display: flex;
  flex-direction: row;
  justify-content: space-between;
  width: 100%;
  max-width: 1200px;
  margin: 100px auto 20px auto; /* Adjusted for navbar height */
  padding: 10px;
  background-color: #fff;
  border-radius: 10px;
  box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
  z-index: 1;
}

/* Sidebar Styles */
.slidebar {
  position: fixed;
  left: 0;
  top: 100px; /* Adjusted for navbar height */
  width: 300px;
  height: calc(100% - 100px);
  background-color: #333;
  color: #fff;
  padding: 20px;
  box-shadow: 2px 0 5px rgba(0, 0, 0, 0.2);
  transform: translateX(-100%);
  transition: transform 0.3s ease;
  z-index: 1000;
}

.slidebar.active {
  transform: translateX(0);
}
```

```
#toggle-sidebar {
  position: absolute;
  top: 20px;
  right: -60px;
  width: 50px;
  height: 50px;
  background-color: #333;
  color: #fff;
  border: none;
  cursor: pointer;
  font-size: 24px;
}

#history-sidebar {
  overflow-y: auto;
  height: calc(100% - 50px);
  padding: 10px;
}

.chat-item {
  background-color: #444;
  padding: 10px;
  border-radius: 5px;
  margin-bottom: 10px;
  cursor: pointer;
}

.chat-item:hover {
  background-color: #555;
}

/* Webcam Feed Styles */
.webcam-feed {
  flex: 1;
  display: flex;
  flex-direction: column;
  align-items: center;
  padding: 20px;
```

```
}

#webcam {
  border-radius: 10px;
  margin-bottom: 10px;
}

/* Chat Box Styles */
.chat-box {
  flex: 1;
  display: flex;
  flex-direction: column;
  padding: 20px;
  max-width: 600px;
}

#chat-response {
  flex: 1;
  overflow-y: auto;
  margin-bottom: 10px;
  background-color: #f9f9f9;
  padding: 10px;
  border-radius: 10px;
  box-shadow: inset 0px 0px 5px rgba(0, 0, 0, 0.1);
}

.bhavana-message,
.user-message {
  margin-bottom: 10px;
  display: flex;
  align-items: flex-start;
}

.bhavana-message p,
.user-message p {
  background-color: #e1f7d5;
  border-radius: 10px;
  padding: 10px;
  margin: 0;
```

```
  max-width: 80%;
}

.user-message p {
  background-color: #c2f0fc;
  margin-left: auto;
}

.bhavana-logo {
  width: 40px;
  height: 40px;
  margin-right: 10px;
}

#chat-form {
  display: flex;
  align-items: center;
}

#user_input {
  flex: 1;
  padding: 10px;
  border-radius: 10px;
  border: 1px solid #ccc;
  margin-right: 10px;
}

#chat-form button {
  padding: 10px 20px;
  background-color: #333;
  color: #fff;
  border: none;
  border-radius: 10px;
  cursor: pointer;
}

#capture-button,
#new-chat-button {
  padding: 10px 20px;
```

```
  background-color: #333;
  color: #fff;
  border: none;
  border-radius: 10px;
  margin: 5px;
  cursor: pointer;
}

#capture-button:disabled {
  background-color: #777;
}

#chat-response {
  display: flex;
  flex-direction: column;
  gap: 10px;
}

.user-message,
.bhavana-message {
  display: flex;
  align-items: flex-start;
}

.user-message p,
.bhavana-message p {
  margin: 0;
  padding: 10px;
  border-radius: 10px;
  max-width: 80%;
}

.bhavana-message p {
  background-color: #e1f7d5;
}

.user-message p {
  background-color: #c2f0fc;
}
```

```
.bhavana-logo {
  width: 40px;
  height: 40px;
  margin-right: 10px;
}
```

4.2.3 BACKEND IMPLEMENTATIONS SCREENSHOTS

MongoDB Chat History

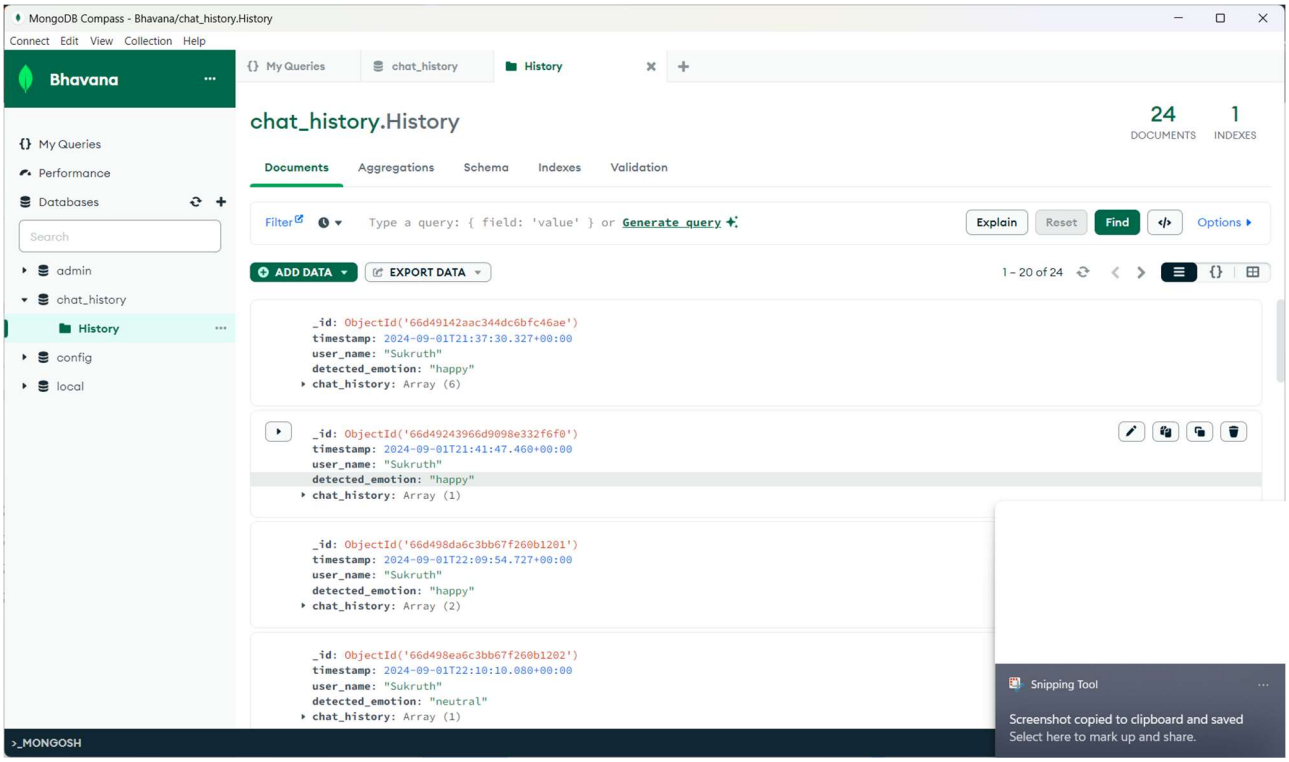


FIG 4.1

Sign-up/Login

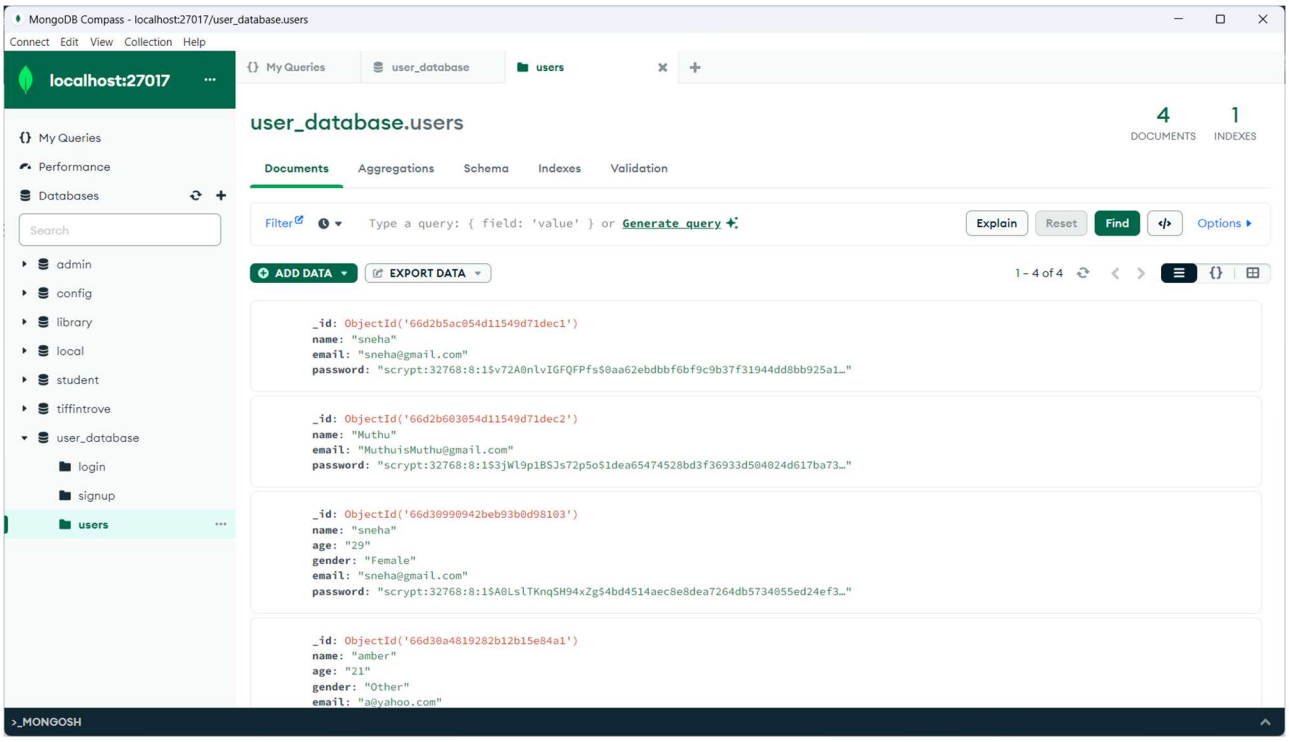


FIG 4.2

Feedback Form

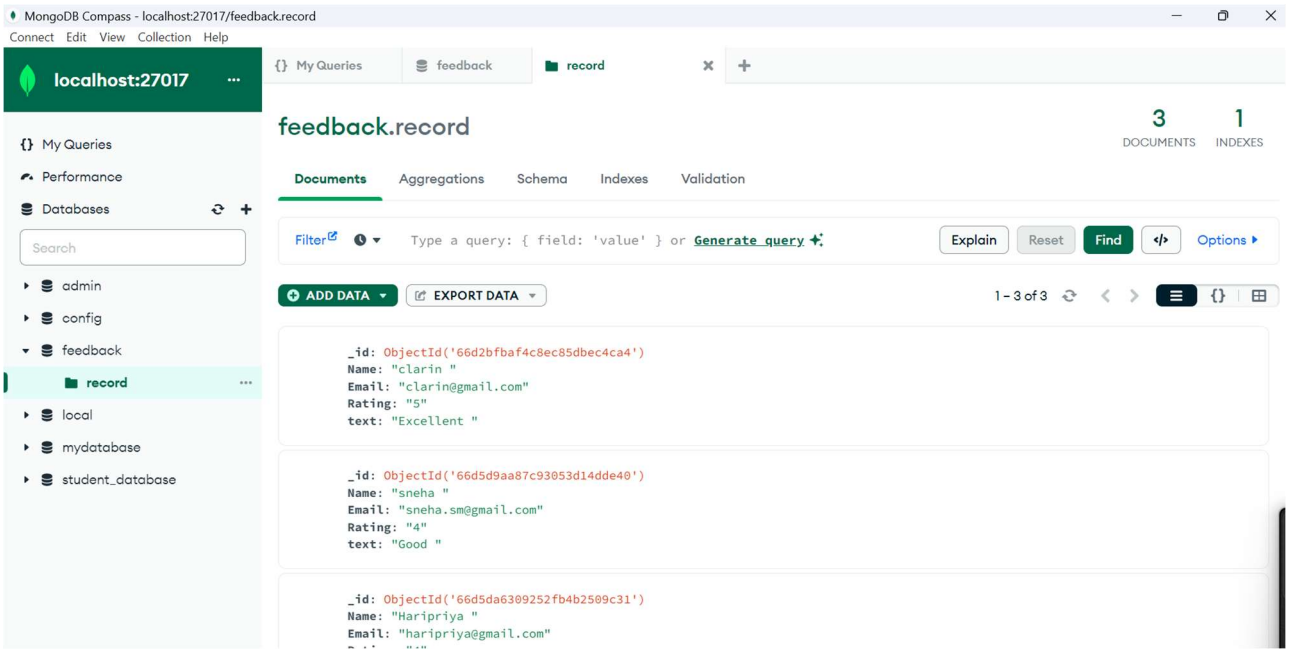


FIG 4.3

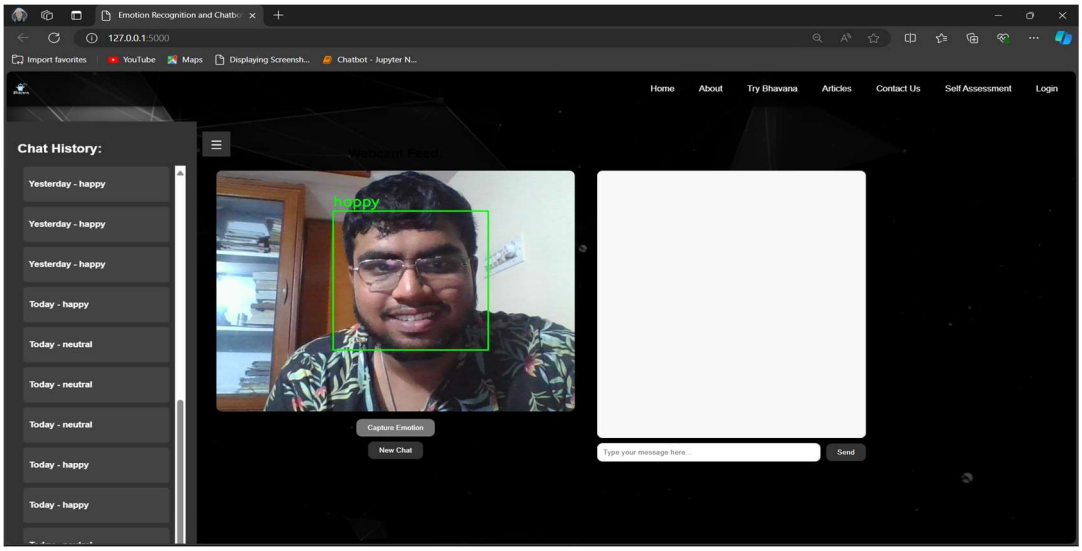
5. TESTING

5.1 TEST CASES

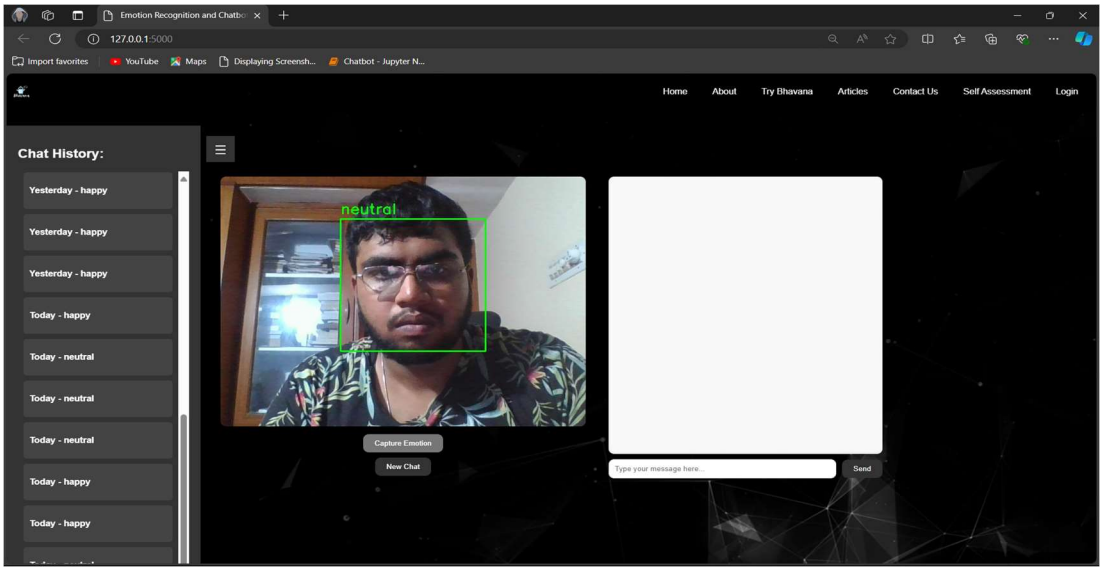
5.1.1 EMOTION DETECTION CHATBOT

1. ..... Verify to capture Correct emotion

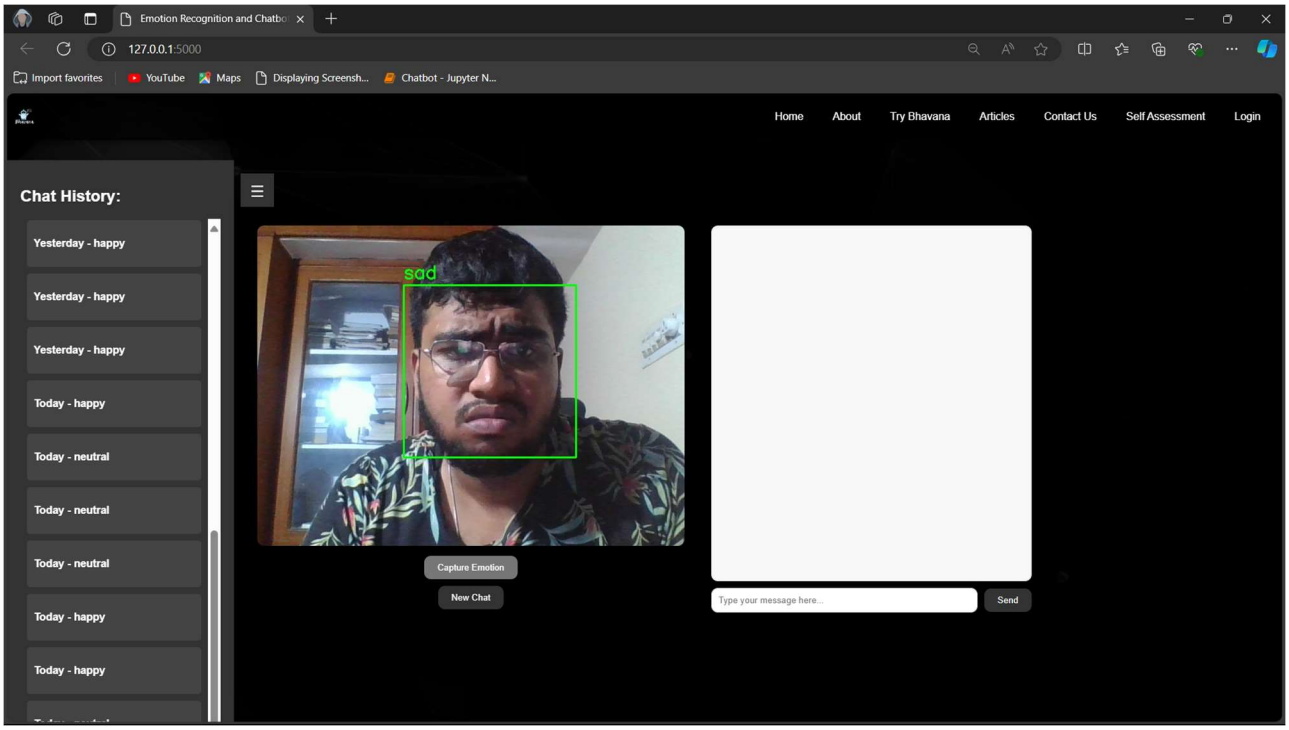
Happy



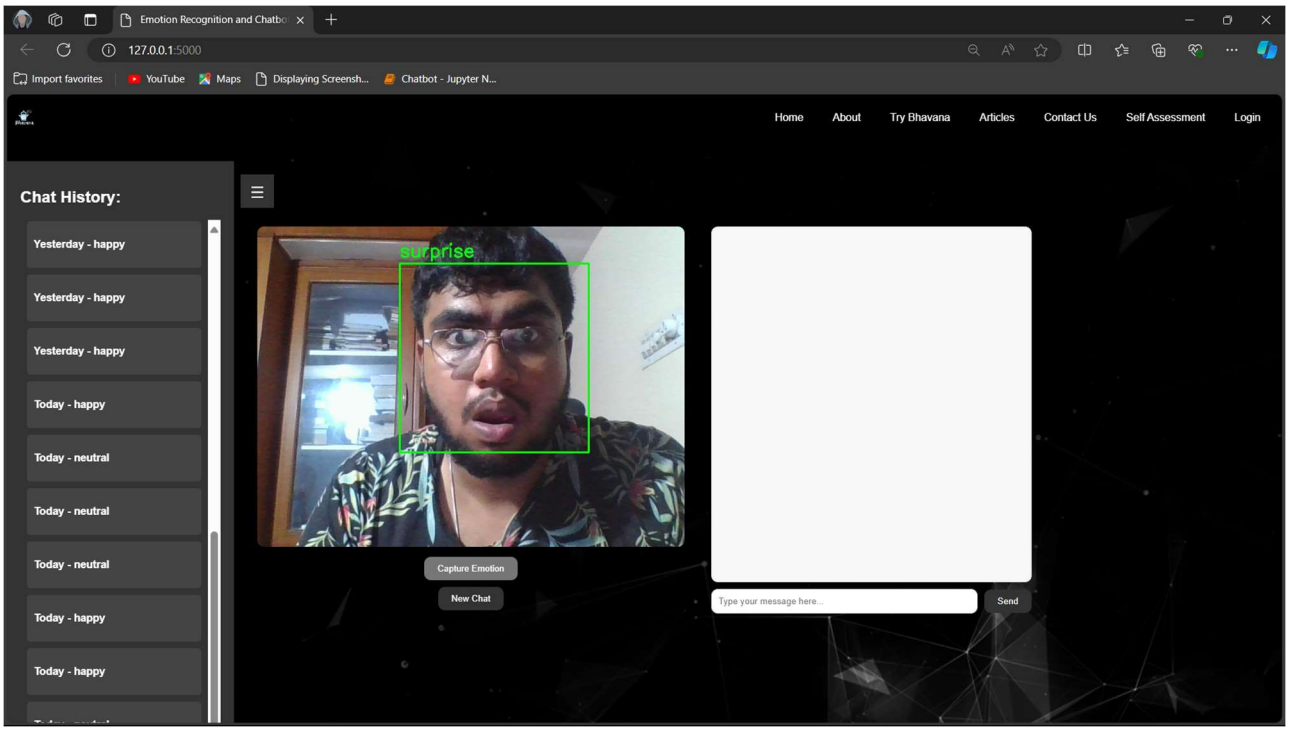
Neutral



Sad



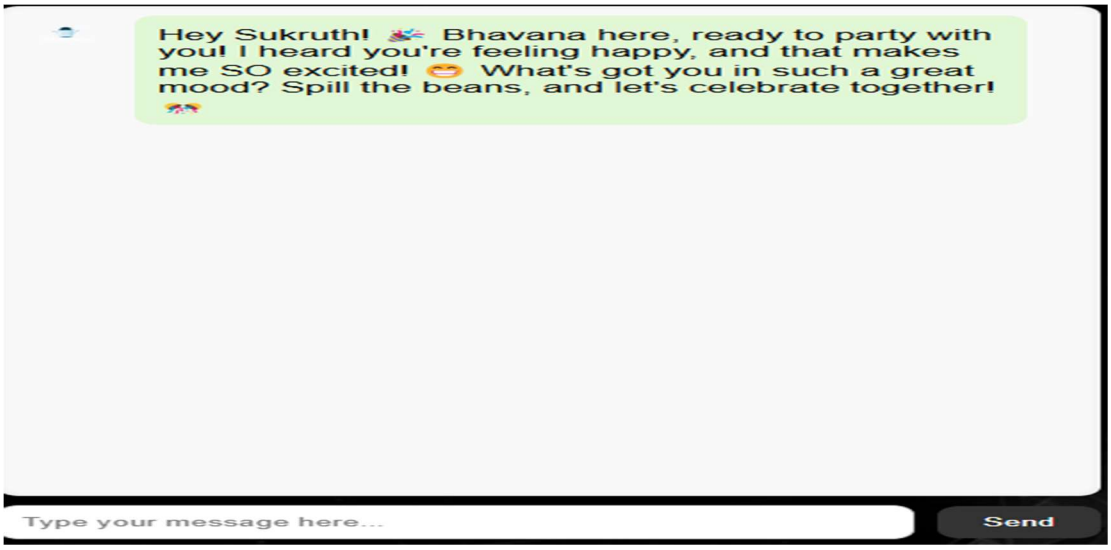
Surprise



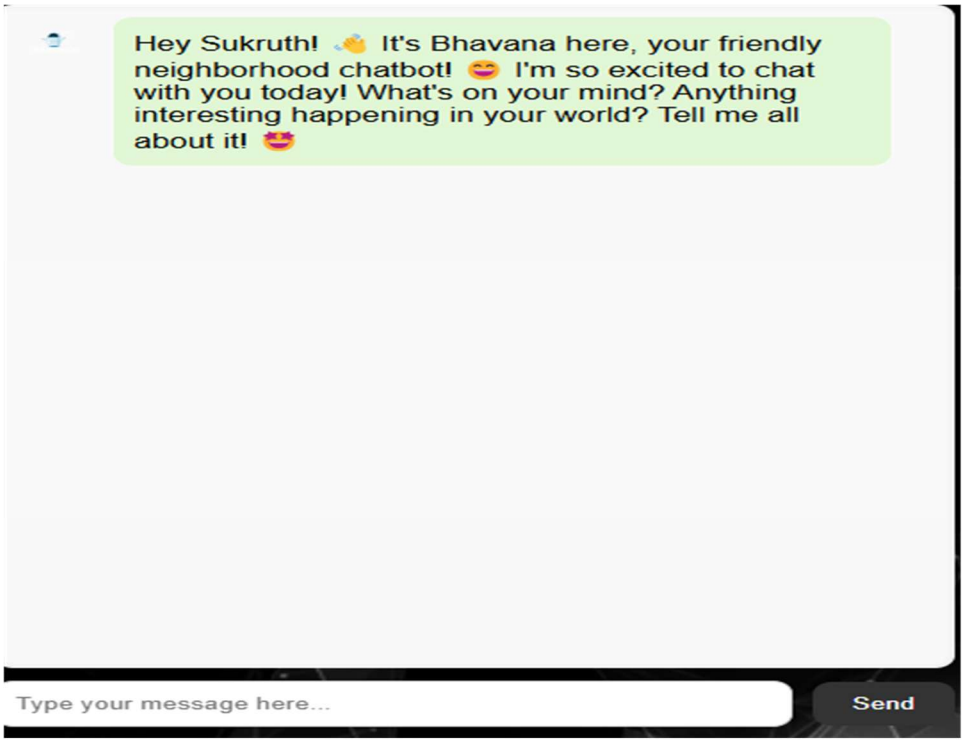
2.....

Chatbot Responses

Happy



Neutral



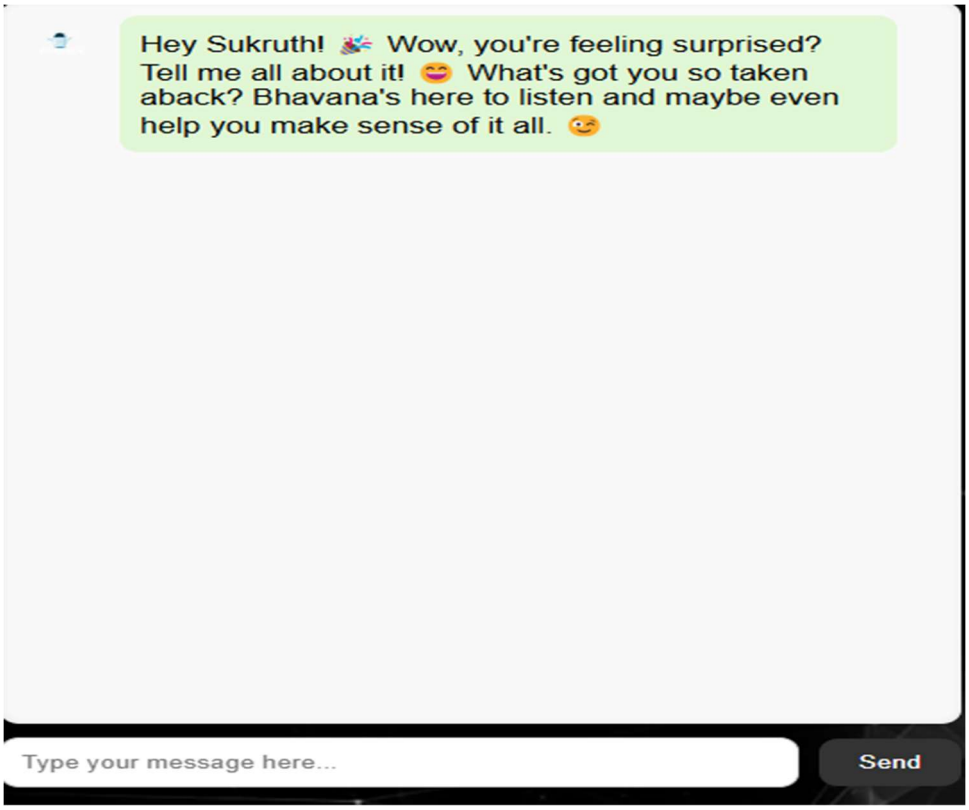
Sad


Hey Sukruth! It's Bhavana here, ready to bring some sunshine into your day! I heard you're feeling a little down, and I just wanted to send you a big virtual hug! Tell you what, let's chat about something fun! What's your favorite movie, or maybe a cool new song you've been listening to? I'm all ears, and I promise to do my best to cheer you up!
Type your message here...
Send

Angry


Hey Sukruth! It's Bhavana here, ready to brighten your day! I hear you're feeling a little...heated? Don't worry, we can work through this together! Tell me what's got you feeling so fired up, and maybe we can find a way to cool things down. You know I'm always here for you, ready with a big virtual hug and a listening ear!
Type your message here...
Send

Surprised


Hey Sukruth! 🎉 Wow, you're feeling surprised? Tell me all about it! 😄 What's got you so taken aback? Bhavana's here to listen and maybe even help you make sense of it all. 😉
Type your message here...
Send

5.2 TEST REPORT

This test case report outlines the testing conducted for the chatbot designed to administer and score stress, anxiety, and depression tests. The key objectives were to verify the chatbot's ability to capture correct emotions, validate emotion recognition, and ensure appropriate chatbot responses.

The first test case, "Verify to Capture Correct Emotion," aimed to ensure that the chatbot accurately identifies the user's emotions based on their inputs. The test was conducted by initiating a conversation with the chatbot, providing various text inputs reflecting different emotional states such as stress, anxiety, or calmness, and observing whether the chatbot captured and recognized the emotions correctly.

The second test case, "Emotion Verification," focused on validating the accuracy of the chatbot's emotion recognition capabilities. In this test, we provided a range of inputs with known emotional contexts and cross-checked the chatbot's identified emotions against expected outcomes to ensure that it reliably recognized and categorized each emotion.

The third test case, "Chatbot Responses," evaluated the appropriateness and relevance of the chatbot's responses based on the detected emotions and test scores. This test involved simulating conversations with various scenarios, where the chatbot needed to provide relevant feedback, guidance, or recommendations for further action (e.g., suggesting consultation with a doctor based on high stress or anxiety levels). The chatbot's responses were assessed for their accuracy, sensitivity, and alignment with the users' emotional states and the objectives of the stress, anxiety, and depression tests.

Overall, the testing process confirmed that the chatbot effectively captures emotions, accurately verifies them, and provides responses that are appropriate and aligned with the test results, thereby meeting the functional requirements of the project.

6.CONCLUSION

6.1 DESIGN AND IMPLEMENTATION ISSUES

The challenges that we have faced while implementing the design and implementation of Bhavana, an emotion-driven chatbot:

Emotion Detection Accuracy: The accuracy in emotion detection could only be achieved by fine-tuning machine learning models; this is very critical when dealing with varied facial expressions from different demographic segments of people. It needed training on a large dataset to reduce biases; even then, subtle emotional variations could not be variably detected without giving incongruous results.

Integration of NLP with Emotion Detection: Emotion detection integrated with the functionality of the Natural Language Processing system was quite complex. The responses coming from the chatbot had not only to be contextually appropriate but also needed to match the detected emotions. In this case, smooth interaction between these modules would be possible if one gave due care to the design of the API endpoint and data flow.

Real-time Processing: Given that emotion detection and response generation needs to be done in real time, this imposed some performance challenges. Being sure that the system can process images, detect emotions, and then generate responses quickly enough to sustain the flow of a natural conversation requires both code-level and hardware-level optimizations.

User Interface Design: The design should be user-friendly and integrate all functions for uploading an image and capturing from the webcam in real time. It needed to be intuitive and maintain smooth communication between the frontend and backend systems.

6.2 ADVANTAGES AND LIMITATIONS

Advantages:

Personal Interaction: The fact that a chatbot detects emotions and responds accordingly lets the users feel more personalized and empathetic.

User Engagement: The response of a chatbot keeps the user more engaged, based on his or her emotional state, and thus may have a positive impact on the user's emotional state.

Real-Time Feedback: The real-time processing lets users know instantly about their emotional state and suitable support.

Limitations:

Emotion Detection Bias: Emotion detection shall be biased if the dataset used to train is not really diverse and representative, leading to inaccuracies for a certain part of users.

Low Emotional Intelligence: While the chatbot can be empathetic, it lacks depth in nuanced emotional situations compared to what a human should be able to provide.

Performance Limitations: Real-time processing may introduce latency when the system load is high, thus disrupting the flow of normal conversation.

6.3 FUTURE ENHANCEMENTS

1.Expanded Emotion Detection Capability: In the future, it would aim at recognizing a wider range of emotions and subtle emotional cues through increasing the training dataset and tuning the model.

2.Improved NLP Responses: Besides improving the NLP module so that the responses are more subtle and contextually appropriate, it would involve the addition of more advanced language models that could accurately capture the models' contextual emotional expressions.

3.Performance optimization: The system architecture and code will undergo further optimizations to reduce latency, allowing for speedups of real-time processing to create smoother interactions. Voice recognition integration will add the functionality to take the user's input in the form of voice to the chatbot, thus making the experience more natural and accessible.

4.Mobile Application Development: A fully-fledged mobile application will be developed in order to make Bhavana more accessible. In this way, users will get a chance to interact with the chatbot from any place, thus receiving real-time emotional support and personalized interaction with mobile devices. This will also incorporate other features in the mobile application, such as push notifications that may be used to serve regular emotional check-ins among other means of improving user engagement.

7. Appendix

8. REFERENCES

[1] Languré, A., d., L., Zareei, M. (2024). Evaluating the Effect of Emotion Models on the Generalizability of Text Emotion Detection Systems. IEEE Access, 12 .

[2] Balahur, A., Hermida, J., M., & Montoyo, A. (2012). Building and Exploiting EmotiNet, a Knowledge Base for Emotion Detection Based on the Appraisal Theory Model. IEEE Transactions on Affective Computing, 3 (1).

[3] (2023). Comparison of Complex Emotion Detection Based on the Basic Emotion Model and the Complex Emotion Model. CNML '23: Proceedings of the 2023 International Conference on Communication Network and Machine Learning.

[4] (2023). Transformer transfer learning emotion detection model: synchronizing socially agreed and self-reported emotions in big data. Neural Computing and Applications.

[5] Bharti, Drsantosh & Varadhaganapathy, S & Gupta, Rajeev & Shukla, Prashant & Bouye, Mohamed & Hinga, Simon & Mahmoud, Amena. (2022). Text-Based Emotion Recognition Using Deep Learning Approach. Computational Intelligence and Neuroscience. 2022. 1-8. 10.1155/2022/2645381.

[6] Ramesh, Kiran & Ravishankaran, Surya & Joshi, Abhishek & Chandrasekaran, K.. (2017). A Survey of Design Techniques for Conversational Agents. 336-350. 10.1007/978-981-10-6544-6_31.

[7] Graves, A., Wayne, G., & Danihelka, I. (2014). Neural Turing Machines. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1410.5401

[8] Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1409.3215

[9] Harbola, Aditya. (2021). Design and Implementation of an AI Chatbot for Customer Service.

Mathematical Statistician and Engineering Applications. 70. 1295-1303. 10.17762/msea.v70i2.2321.

[10] Gamage, G., De Silva, D., Mills, N. *et al.* Emotion AWARE: an artificial intelligence framework for adaptable, robust, explainable, and multi-granular emotion analysis. *J Big Data* 11, 93 (2024). https://doi.org/10.1186/s40537-024-00953-2.

[11] EMOTION DETECTION IN VIRTUAL ASSISTANTS AND CHATBOTS. (2023). *International Research Journal of Modernization in Engineering Technology and Science*. https://doi.org/10.56726/irjmets44626.

[12] https://www.morphcast.com/#:~:text=MorphCast%20Emotion%20AI%2C%20with%20130,and%20emotionally%20empowering%20digital%20experiences

[13] Gemini AI. (2024). *Gemini API*. https://ai.google.dev/api?lang=python