Emoscan :

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

<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <meta name="csrfmiddlewaretoken" content="{{ csrf_token }}">
 <title>Emotion Analysis</title>
 <style>
  /* General Styles */
  :root {
   --dark-bg: #121212;
   --light-bg: #1E1E1E;
   --primary: #00FF00;
   --secondary: #FF0000;
   --text-dark: #FFFFFF;
   --text-light: #CCCCCC;
   --shadow: 0 4px 15px rgba(0, 255, 0, 0.2);
  }

  body {
   margin: 0;
   font-family: Arial, sans-serif;
   background-color: var(--dark-bg);
   color: var(--text-dark);
   justify-items: center;
   min-height: 100vh;
   line-height: 1.6;
  }
```

```
.dashboard {
  min-height: 100vh;
  padding: 20px;
  width: 1396px;
  margin-top: 52px;
}

.alert-card {
  background-color: #FF0000;
  padding: 15px;
  border-radius: 5px;
  margin-top: 10px;
  margin-bottom: 20px;
  color: #FFFFFF;
}

.content {
  padding: 20px;
}

/* Real-Time Emotion Detection */
.video-feed {
  background-color: var(--light-bg);
  padding: 20px;
  border-radius: 10px;
  margin-bottom: 30px;
  box-shadow: var(--shadow);
}

.video-container {
  display: flex;
```

```
  align-items: center;
  gap: 20px;
}

.video-container video {
  width: 60%;
  border-radius: 10px;
}

.emotion-labels {
  flex: 1;
}

.emotion-labels h3 {
  margin-top: 0;
  color: var(--primary);
}

.emotion-labels p {
  margin: 10px 0;
  color: var(--text-light);
}

/* Historical Data */
.historical-data {
  background-color: var(--light-bg);
  padding: 20px;
  border-radius: 10px;
  margin-bottom: 30px;
  box-shadow: var(--shadow);
}
```

```
.historical-data h2 {
  margin-top: 0;
  color: var(--primary);
}

table {
  width: 100%;
  border-collapse: collapse;
}

table th,
table td {
  padding: 10px;
  text-align: left;
  border-bottom: 1px solid var(--dark-bg);
}

table th {
  background-color: var(--primary);
  color: var(--dark-bg);
}

table tr:hover {
  background: rgba(0, 255, 0, 0.1);
}

/* Export Options */
.export-options {
  background-color: var(--light-bg);
  padding: 20px;
```

```
  border-radius: 10px;
  box-shadow: var(--shadow);
}

.export-options h2 {
  margin-top: 0;
  color: var(--primary);
}

.export-buttons {
  display: flex;
  gap: 10px;
}

.export-buttons button {
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  background-color: var(--primary);
  color: var(--dark-bg);
  cursor: pointer;
  transition: all 0.3s ease;
}

.export-buttons button:hover {
  background-color: rgba(0, 255, 0, 0.8);
}

/* Footer */
footer {
  background: var(--light-bg);
```

```
    padding: 1rem;
    text-align: center;
    margin-top: auto;
    width: 98%;
    box-shadow: var(--shadow);
   }

   footer p {
    color: var(--text-light);
    font-size: 0.9rem;
    margin: 0.5rem 0;
   }
  </style>
</head>

<body>
  {% include 'mentalhealthbar.html' %}
  <div class="dashboard">
   <div class="content">
    <h1>Emotion Analysis</h1>
    <audio id="alert-sound" src=""></audio>
    <div id="alert-notification"
     style="display: none; position: fixed; top: 0; left: 0; width: 100%; background-color: red; color:
white; text-align: center; padding: 10px; z-index: 1000;">
     ALERT: Continuous emotion detected for more than 1 minute!
    </div>
    <div class="video-feed">
     <h2>Real-Time Emotion Detection</h2>
     <div class="video-container">
      <img style="width:55%; height:100%;" src="{% url 'video_feed2' %}" id="video-feed"
alt="Video Feed">
      <canvas id="video-canvas" style="position: absolute; width:55%; height:100%;"></canvas>
```

```
    </div>
   </div>

   <div class="historical-data">
    <h2>Historical Data (Averaged over 15 seconds)</h2>
    <table>
     <thead>
      <tr>
       <th>Date & Time</th>
       <th>Emotion</th>
       <th>Confidence</th>
      </tr>
     </thead>
     <tbody id="historical-data-body">
      <!-- Rows will be dynamically inserted here -->
     </tbody>
    </table>
   </div>
  </div>
</div>

<footer>
  <p>Developed By: Manas Premchand Chaudhari (IT 2024-25)</p>
  <p>&copy; 2024 EmoScan - Face Emotion Recognition. All rights reserved.</p>
</footer>



<script>
  // Function to get CSRF token
  function getCSRFToken() {
    const csrfToken = document.querySelector('meta[name="csrfmiddlewaretoken"]');
```

```
  if (csrfToken) {
    return csrfToken.content;
  } else {
    console.error("CSRF token not found!");
    return null;
  }
}

// Global variable to store detected faces
let faces = [];

async function analyzeEmotion() {
  try {
    const patientId = getPatientIdFromURL();
    if (!patientId) {
      throw new Error("Patient ID is missing");
    }

    const response = await fetch('/health/detect_emotion2/', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'X-CSRFToken': getCSRFToken()
      },
      body: JSON.stringify({
        patient_id: patientId
      })
    });

    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
```

```
}

const data = await response.json();

// Debugging: Log the response data
console.log("Response data:", data);

// Process the data
if (data.alert) {
  // Play the alert sound
  const alertSound = document.getElementById('alert-sound');
  alertSound.play();

  // Show an alert message
  alert(data.message);
}



if (data.error) {
  throw new Error(data.error);
}

// Update the global faces array
faces = data.faces;

if (faces.length === 0) {
  return; // No faces detected, do nothing
}
// Read the response body once and store it in a variable
```

```
    // Update the UI with the detected emotions
    if (data.faces && data.faces.length > 0) {
      updateHistoricalDataTable(data.faces);
      drawBoundingBoxes(data.faces);
    }
    console.log("Emotion analysis updated successfully");
  } catch (error) {
    console.error('Error:', error);
    alert('Failed to analyze emotion: ' + error.message);
  }
}
function checkForAlert(faces) {
  faces.forEach(face => {
    if (face.emotion === 'anger' || face.emotion === 'sadness') {
      alertSound.play();
      alert(`ALERT: Continuous ${face.emotion} detected for more than 1 minute!`);
    }
  });
}
function checkForAlert(faces) {
  faces.forEach(face => {
    if (face.emotion === 'anger' || face.emotion === 'sadness') {
      alertSound.play();
      alert(`ALERT: Continuous ${face.emotion} detected for more than 1 minute!`);
    }
  });
}

function exportData(format) {
  const patientId = getPatientIdFromURL();
```

```
  fetch('/health/export_data/', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'X-CSRFToken': getCSRFToken()
    },
    body: JSON.stringify({ patient_id: patientId, format: format })
  }).then(response => {
    if (response.ok) {
      return response.blob();
    } else {
      throw new Error('Export failed');
    }
  }).then(blob => {
    const url = window.URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = `emotion_data_${patientId}.${format}`;
    a.click();
  }).catch(error => {
    console.error('Error:', error);
    alert('Export failed: ' + error.message);
  });
}

function drawBoundingBoxes(faces) {
  const videoFeed = document.getElementById('video-feed');
  const canvas = document.getElementById('video-canvas');
  const ctx = canvas.getContext('2d');

  // Set canvas dimensions to match the video feed
```

```
  canvas.width = videoFeed.videoWidth || videoFeed.width;
  canvas.height = videoFeed.videoHeight || videoFeed.height;

  // Clear the canvas
  ctx.clearRect(0, 0, canvas.width, canvas.height);

  // Draw bounding boxes for each detected face
  faces.forEach(face => {
    const { x, y, width, height } = face.bbox;

    // Scale coordinates if necessary
    const scaleX = canvas.width / videoFeed.videoWidth;
    const scaleY = canvas.height / videoFeed.videoHeight;

    const scaledX = x * scaleX;
    const scaledY = y * scaleY;
    const scaledWidth = width * scaleX;
    const scaledHeight = height * scaleY;

    // Draw the bounding box
    ctx.strokeStyle = '#00FF00'; // Green color for the bounding box
    ctx.lineWidth = 2;
    ctx.strokeRect(scaledX, scaledY, scaledWidth, scaledHeight);
  });
}
// Function to update historical data table
function updateHistoricalDataTable(faces) {
  const historicalDataBody = document.getElementById('historical-data-body');
  if (!historicalDataBody) {
    console.error("Historical data body not found!");
    return;
```

```
  }

  // Add new rows for each detected face
  faces.forEach(face => {
    const { emotion, confidence } = face;
    const row = document.createElement('tr');
    row.innerHTML = `
      <td>${new Date().toLocaleString()}</td>
      <td>${emotion}</td>
      <td>${confidence.toFixed(2)}%</td>
    `;
    // Insert the new row at the beginning of the table body
    historicalDataBody.insertBefore(row, historicalDataBody.firstChild);
  });
}

function calculateAverageEmotionData(faces) {
  const emotionCounts = {};
  const emotionConfidences = {};

  faces.forEach(face => {
    const { emotion, confidence } = face;
    if (!emotionCounts[emotion]) {
      emotionCounts[emotion] = 0;
      emotionConfidences[emotion] = 0;
    }
    emotionCounts[emotion]++;
    emotionConfidences[emotion] += confidence;
  });

  const averageEmotions = Object.keys(emotionCounts).map(emotion => ({
```

```
    emotion,
    averageConfidence: (emotionConfidences[emotion] / emotionCounts[emotion]).toFixed(2)
  }));

  return averageEmotions;
}

// Function to start periodic emotion analysis
function startPeriodicAnalysis(interval = 3000) {
  setInterval(analyzeEmotion, interval);
}

// Start periodic analysis when the page loads
document.addEventListener('DOMContentLoaded', () => {
  startPeriodicAnalysis(3000);
});

// Function to export data
function exportData(format) {
  const patientId = getPatientIdFromURL();
  if (patientId) {
    alert(`Exporting data for patient ${patientId} as ${format.toUpperCase()}....`);
  } else {
    alert(`No patient ID found.`);
  }
}

// Function to get patient ID from URL
function getPatientIdFromURL() {
  const urlParams = new URLSearchParams(window.location.search);
  return urlParams.get('id');
```

```
    }
  </script>
</body>

</html>
```

Views .py :

```
from pyexpat.errors import messages
from django.http import StreamingHttpResponse, JsonResponse
from django.views.decorators.csrf import csrf_exempt
import cv2
import numpy as np
import base64
from emotion.models import signup
from django.shortcuts import redirect, render
from mentalhealth.views import health
from mentalhealth.models import Patient
from .utils.video_stream import VideoCamera
from .utils.emotion_detector import EmotionDetector

# Global instances (for demo purposes - consider thread-local storage for production)
video_camera = VideoCamera()
emotion_detector = EmotionDetector()

def index(request):
    return render(request, 'index.html')

def login(request):
    if request.method == "POST":
        email = request.POST["email"]
        password = request.POST["password"]
```

```
        # Authenticate user
        user = signup.objects.filter(email=email).first()

        if user and user.password == password:  # In production, use check_password
            request.session["user_id"] = user.id
            request.session["email"] = user.email
            request.session["industry"] = user.industry
            return redirect("dashboard")
        else:
            return render(request, "login.html", {"error": "Invalid email or password."})

    return render(request, "login.html")

def sign_up(request):
    if request.method == 'POST':
        email = request.POST["email"]
        password = request.POST["password"]
        industry = request.POST["industry"]

        # Create new user
        user = signup.objects.create(email=email, password=password, industry=industry)

        # Log the user in by setting session variables
        request.session["user_id"] = user.id
        request.session["email"] = user.email
        request.session["industry"] = user.industry

        return redirect("dashboard")
    return render(request, "signup.html")
```

```
def dashboard(request):
    if "user_id" not in request.session:
        return redirect("login")
    if request.session["industry"]=="personal":
      return render(request, "index.html")
    if request.session["industry"]=="healthcare":
      return redirect("health")


def logout(request):
    request.session.flush()
    return redirect("login")


@csrf_exempt  # Temporary for testing - use proper CSRF protection in production
def video_feed(request):
    def generate():
        while True:
            frame = video_camera.get_frame()
            if frame is None:
                break
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')

    return StreamingHttpResponse(
        generate(),
        content_type='multipart/x-mixed-replace; boundary=frame'
    )


@csrf_exempt
def detect_emotion(request):
    if request.method == 'POST':
        try:
```

```
        # Get frame from camera directly
        frame = video_camera.get_frame()
        if frame is None:
            return JsonResponse({"error": "Failed to capture frame"}, status=400)

        # Convert frame to OpenCV format
        img_array = np.frombuffer(frame, dtype=np.uint8)
        img = cv2.imdecode(img_array, flags=cv2.IMREAD_COLOR)

        # Detect emotion
        processed_frame = emotion_detector.detect_emotion(img)
        emotion, confidence = emotion_detector.get_latest_emotion()

        # Detect all faces and their emotions
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        faces = emotion_detector.face_cascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=5, minSize=(30, 30))

        # Prepare response data
        faces_data = []
        for (x, y, w, h) in faces:
            face_img = gray[y:y+h, x:x+w]
            resized = cv2.resize(face_img, (48, 48))
            normalized = resized / 255.0
            reshaped = np.reshape(normalized, (1, 48, 48, 1))
            result = emotion_detector.model.predict(reshaped)
            label = np.argmax(result, axis=1)[0]
            confidence = np.max(result) * 100

            # Extract face region from the original frame
            face_region = img[y:y+h, x:x+w]
```

```
            # Convert face region to base64-encoded data URL
            _, buffer = cv2.imencode('.jpg', face_region)
            face_data_url = base64.b64encode(buffer).decode('utf-8')

            # Add face data to the response
            faces_data.append({
                "emotion": emotion_detector.labels_dict[label],
                "confidence": confidence,
                "bbox": [int(x), int(y), int(w), int(h)],  # Bounding box coordinates
                "face_img": face_data_url
            })

            # Draw rectangle and emotion text on frame
            cv2.rectangle(processed_frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
            cv2.putText(processed_frame, emotion_detector.labels_dict[label], (x, y-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

        return JsonResponse({
            "faces": faces_data,
            "processed_frame": base64.b64encode(cv2.imencode('.jpg', processed_frame)[1]).decode()
        })
    except Exception as e:
        return JsonResponse({"error": str(e)}, status=500)
  return JsonResponse({"error": "Invalid request"}, status=400)
```