ATTENDANCE MANAGEMENT SYSTEM USING FACE-RECOGNITION

Project report submitted

for the industrial internship under the guidance of Atul Rawat Sir, at CDAC

Center in North-East (CINE), Silchar.

By

L D M Satya Sai Teja

Table of Contents:

Introduction:

The Face Recognition App is a Python-based application that provides a user-friendly solution for face registration, recognition, and deletion through a graphical user interface (GUI). It employs the Face Recognition library and OpenCV for accurate face detection and recognition.

In an era where face recognition technology is gaining prominence in security, attendance tracking, and identity verification, the Face Recognition App offers users an accessible means to leverage this technology. The app's methodology encompasses three key functionalities: registration, recognition, and deletion of faces, all designed to be straightforward and efficient.

Methodology:

The methodology of the Face Recognition App is centred around three key functions: Registration, Recognition, and Deletion. These functions are designed to provide a seamless and user-friendly experience while ensuring accuracy and data integrity.

Registration:

In the registration process, users initiate the procedure by selecting "Register a new user" within the GUI. They are prompted to input their essential information, including their name, contact number, and employee ID. The application captures the user's face in real-time, encodes it to create a unique facial signature, and securely stores this data in both a JSON file and a CSV file. To maintain data accuracy and prevent redundancy, the system employs a similarity check, ensuring that multiple registrations of the same face are avoided.

Recognition:

The recognition feature allows users to identify individuals in real-time. By clicking "Recognize a user," the system captures the face using the device's camera, encodes it, and subsequently compares it against the database of registered faces. If a match is found, the app records the user's attendance, providing a digital record of their presence, and displays the relevant user

information, making it highly suitable for applications such as attendance tracking and secure access control.

Deletion:

To offer users control over their data, the application includes a deletion feature. Users can select "Delete a user," and the system captures the face to be deleted, encodes it, and performs a match with the registered faces. Importantly, before initiating the deletion process, the system requests confirmation from the user to avoid accidental data loss.

This well-structured methodology ensures that the Face Recognition App meets user requirements while upholding data integrity and security.

Tech Toolkit:

- **Python (v3.11.5):** The core programming language that underpins the entire application.
- **OpenCV (v4.8.0):** Utilized for real-time video capture and image processing, crucial for face detection and preprocessing.
- **Face Recognition Library (v1.2.3):** Responsible for accurate face detection and recognition, a key component in user registration and recognition.
- **JSON (v2.0.9):** Enables secure storage of user data, ensuring organized and easily retrievable information.
- **CSV (v1.0):** Records and manages attendance data efficiently, creating structured attendance records.
- **Tkinter (v1.3.3):** Facilitates the creation of an intuitive graphical user interface, enhancing user accessibility.
- **PIL (Python Imaging Library) (v9.5.0)**: Handles image conversion and manipulation, ensuring proper image display and management.
- **OS:** Used for essential file system operations, including file creation, management, and data storage.
- **Datetime:** Manages date and time operations, crucial for timestamp recording and attendance management.

Architecture:

Frontend (GUI):

The graphical user interface (GUI) is a critical component of the Face Recognition App, and it is constructed using the Tkinter library. The GUI is thoughtfully organized into three primary frames, each serving a distinct purpose:

Register Frame: This frame is dedicated to user registration. Users can input their personal information, including their name, contact number, and employee ID. The frame also includes an entry field for capturing the user's face. Upon clicking the "Capture Face" button, the application initiates face detection and encoding using OpenCV, followed by data storage in the JSON file and CSV file.

Recognize Frame: Designed for face recognition, this frame allows users to identify individuals in real-time. The frame includes a live video feed for capturing faces. Upon clicking the "Recognize a user" button, the system captures the face in real-time, encodes it, and matches it against registered faces. If a match is found, attendance is recorded in the CSV file, and user information is displayed.

Delete Frame: Users can manage their registered faces from this frame. By clicking the "Delete a user" button, the system captures the face to be deleted, encodes it, and checks for a match with registered faces. Crucially, user confirmation is requested before proceeding with the deletion, ensuring data security and control.

Data Processing Component:

The backend of the application is responsible for handling various essential functions:

Face Detection and Encoding: The OpenCV library is employed for real-time face detection and encoding, while the Face Recognition library ensures precise face recognition. These functionalities are the core of the application's user registration and recognition features.

User Data Storage: User data is securely stored in a JSON file named "faces.json." This file maintains a record of registered users, including their names, contact numbers, employee IDs, and face encodings.

Attendance Recording: Attendance records are stored in a CSV file named "Attendance.csv." The application records attendance data with timestamps, capturing both the entry and exit times of users.

Duplicate Registration Prevention: To maintain data integrity, the application employs a face encoding comparison to prevent the registration of duplicate faces. This mechanism ensures that only one unique representation of each user is stored.

Deletion of Registered Faces: The backend facilitates the deletion of registered faces. It captures the face to be deleted, encodes it, and verifies it against registered faces. Before initiating the deletion process, user confirmation is sought to avoid accidental data removal.

Source Code:

```
import cv2
import face_recognition
import json
import tkinter as tk
from tkinter import messagebox
from PIL import Image, ImageTk
import os
import csv
import datetime

# Paths to data files
user_data_path = "faces.json"
attendance_file_path = "Attendance.csv"

# Load existing user data if available, or initialize an empty dictionary
if os.path.exists(user_data_path):
    with open(user_data_path, "r") as file:
        try:
            user_data = json.load(file)
        except json.JSONDecodeError:
            user_data = {}
else:
```

```
    user_data = {}

# Columns for the attendance CSV file
columns = ["Name", "Contact Number", "Employ ID"]

# Create the attendance CSV file if it doesn't exist
if not os.path.exists(attendance_file_path):
    with open(attendance_file_path, "w", newline="") as file:
        writer = csv.writer(file)
        writer.writerow(columns)

# Create the main Tkinter window
root = tk.Tk()
root.title("Face Recognition App")
root.geometry("800x600")

# Frames for different sections of the GUI
register_frame = tk.Frame(root)
recognize_frame = tk.Frame(root)
delete_frame = tk.Frame(root)

# Variable to keep track of the current displayed frame
current_frame = None

# Video capture from the default camera (0)
video_capture = cv2.VideoCapture(0)

# Function to show a frame in the GUI
def show_frame(frame):
    global current_frame
    if current_frame:
        current_frame.pack_forget()
    if frame.winfo_ismapped() == 0:
        frame.pack()
    current_frame = frame

# Function to display a message, hide entry fields, and a button
def display_message_and_hide_button1(message, name_entry, contact_entry,
employ_id_entry, capture_button):
    messagebox.showinfo("Info", message)
    name_entry.pack_forget()
    contact_entry.pack_forget()
    employ_id_entry.pack_forget()
    capture_button.pack_forget()

# Variable to prevent multiple registration attempts
x = False
```

```
# Function to register a user through the GUI
def register_user_gui():
    global x
    if x:
        return
    x = True
    show_frame(register_frame)

    # Entry fields for name, contact, and employee ID
    name_entry = tk.Entry(register_frame, width=40)
    name_entry.insert(0, "Enter your name")
    name_entry.pack()

    contact_entry = tk.Entry(register_frame, width=40)
    contact_entry.insert(0, "Enter your contact number")
    contact_entry.pack()

    employ_id_entry = tk.Entry(register_frame, width=40)
    employ_id_entry.insert(0, "Enter your employ ID")
    employ_id_entry.pack()

    # Functions to handle placeholder text
    def on_name_entry_click(event):
        if name_entry.get() == "Enter your name":
            name_entry.delete(0, tk.END)

    def on_contact_entry_click(event):
        if contact_entry.get() == "Enter your contact number":
            contact_entry.delete(0, tk.END)

    def on_employ_id_entry_click(event):
        if employ_id_entry.get() == "Enter your employ ID":
            employ_id_entry.delete(0, tk.END)

    name_entry.bind("<Button-1>", on_name_entry_click)
    contact_entry.bind("<Button-1>", on_contact_entry_click)
    employ_id_entry.bind("<Button-1>", on_employ_id_entry_click)

    # Function to capture the user's face
    def capture_face():
        global x
        ret, frame = video_capture.read()
        face_locations = face_recognition.face_locations(frame)

        if not face_locations:
            messagebox.showerror("Error", "No face detected. Please ensure your
face is well-lit and visible.")
            x = False
```

```
            return
        elif len(face_locations) > 1:
            messagebox.showinfo("Info", "Multiple faces detected. Please ensure
only one face is in the frame.")
            return

        face_encoding = face_recognition.face_encodings(frame, face_locations)[0]

        # Check for similar faces to prevent duplicate registrations
        for existing_user, user_info in user_data.items():
            existing_encoding = user_info.get("encoding")
            if existing_encoding is not None:
                distance = face_recognition.face_distance([existing_encoding],
face_encoding)[0]
                if distance < 0.5:
                    display_message_and_hide_button1(f"Similar face already
registered as {existing_user}. Skipping registration.", name_entry, contact_entry,
employ_id_entry, capture_button)
                    x = False
                    return

        # Get user data
        name = name_entry.get()
        contact = contact_entry.get()
        employ_id = employ_id_entry.get()

        # Store user data in the JSON file
        user_data[name] = {
            "contact": contact,
            "employ_id": employ_id,
            "encoding": face_encoding.tolist()
        }

        with open(user_data_path, "w") as file:
            json.dump(user_data, file, indent=4)

        # Store user data in the CSV file
        with open('employ_details.csv', mode='a', newline='') as employ_file:
            employ_writer = csv.writer(employ_file)
            if employ_file.tell() == 0:
                employ_writer.writerow(["Name", "Contact Number", "Employ ID"])
            employ_writer.writerow([name, contact, employ_id])

        display_message_and_hide_button1(f"User {name} registered successfully!",
name_entry, contact_entry, employ_id_entry, capture_button)
        x = False
```

```
    capture_button = tk.Button(register_frame, text="Capture Face",
command=capture_face, width=20, height=2)
    capture_button.pack()

# Function to recognize a user through the GUI
def recognize_face():
    ret, frame = video_capture.read()
    face_locations = face_recognition.face_locations(frame)
    if not face_locations:
        messagebox.showinfo("Info", "No face detected.")
        return
    elif len(face_locations) > 1:
        messagebox.showinfo("Info", "Multiple faces detected. Please ensure only
one face is in the frame.")
        return

    face_encoding = face_recognition.face_encodings(frame, face_locations)[0]

    recognized_user = None
    min_distance = 0.5

    # Find the recognized user with the smallest distance
    for name, user_info in user_data.items():
        registered_encoding = user_info["encoding"]
        distance = face_recognition.face_distance([registered_encoding],
face_encoding)[0]

        if distance < min_distance:
            min_distance = distance
            recognized_user = name

    if recognized_user:
        # Get user contact and employee ID
        contact_info = user_data[recognized_user]['contact']
        employ_id_info = user_data[recognized_user]['employ_id']
        current_date = datetime.datetime.now().strftime("%Y-%m-%d")

        # Read and update the attendance CSV file
        with open(attendance_file_path, "r") as file:
            reader = csv.DictReader(file)
            columns = reader.fieldnames
            rows = list(reader)

        date_in_column = f"{current_date} (IN)"
        date_out_column = f"{current_date} (OUT)"

        if date_in_column not in columns:
            columns.extend([date_in_column, date_out_column])
```

```
        found_user = False

        for row in rows:
            if row["Name"] == recognized_user:
                found_user = True
                if not row.get(date_in_column):
                    row[date_in_column] =
datetime.datetime.now().strftime("%H:%M:%S")
                else:
                    row[date_out_column] =
datetime.datetime.now().strftime("%H:M:S")

        if not found_user:
            new_row = {col: "" for col in columns}
            new_row["Name"] = recognized_user
            new_row["Contact Number"] = contact_info
            new_row["Employ ID"] = employ_id_info
            new_row[date_in_column] = datetime.datetime.now().strftime("%H:M:S")
            rows.append(new_row)

        with open(attendance_file_path, "w", newline="") as file:
            writer = csv.DictWriter(file, fieldnames=columns)
            writer.writeheader()
            writer.writerows(rows)

        # Display recognized user information
        messagebox.showinfo("Info", f"Face matched with registered user:
{recognized_user}\nContact: {contact_info}\nEmploy ID: {employ_id_info}")
    else:
        messagebox.showinfo("Info", "User not recognized")

# Function to save user data to the JSON file
def save_user_data(name, user_data):
    with open(user_data_path, "w") as file:
        json.dump(user_data, file, indent=4)

# Function to delete a registered face
def delete_face():
    ret, frame = video_capture.read()
    face_locations = face_recognition.face_locations(frame)
    if not face_locations:
        messagebox.showinfo("Info", "No face detected.")
        return

    face_encoding = face_recognition.face_encodings(frame, face_locations)[0]

    recognized_user = None
```

```
    for name, user_info in user_data.items():
        registered_encoding = user_info["encoding"]
        distance = face_recognition.face_distance([registered_encoding],
face_encoding)[0]

        if distance < 0.5:
            recognized_user = name
            break

    if recognized_user:
        # Ask for confirmation before deleting user data
        confirmation = messagebox.askquestion("Confirm Deletion", f"Shall I delete
the User {recognized_user}'s data?")
        if confirmation == "yes":
            del user_data[recognized_user]
            save_user_data(recognized_user, user_data)
            messagebox.showinfo("Info", f"User {recognized_user}'s data has been
deleted.")
        else:
            messagebox.showinfo("Info", "Deletion canceled.")
        return
    messagebox.showinfo("Info", "User not recognized")

# Label to display video feed from the camera
video_label = tk.Label(root, width=400, height=300)
video_label.pack(side="left", padx=10, pady=10)

# Function to update the video feed
def update_video_frame():
    ret, frame = video_capture.read()
    if ret:
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        frame = Image.fromarray(frame)
        frame = ImageTk.PhotoImage(image=frame)
        video_label.img = frame
        video_label.config(image=frame)
        video_label.after(10, update_video_frame)
    else:
        video_label.after(10, update_video_frame)

update_video_frame()

# Frame to hold buttons
button_frame = tk.Frame(root)
button_frame.pack(side="left", padx=10, pady=10)

# Buttons for various actions
```

```
register_button = tk.Button(root, text="Register a new user",
command=register_user_gui, width=20, height=2)
register_button.pack(pady=10)

recognize_button = tk.Button(root, text="Recognize a user",
command=recognize_face, width=20, height=2)
recognize_button.pack(pady=10)

delete_button = tk.Button(root, text="Delete a user", command=delete_face,
width=20, height=2)
delete_button.pack(pady=10)

exit_button = tk.Button(root, text="Exit", command=root.quit, width=20, height=2)
exit_button.pack(pady=10)

# Start the main loop
root.mainloop()
```

Explanation:

```
import cv2
import face_recognition
import json
import tkinter as tk
from tkinter import messagebox
from PIL import Image, ImageTk
import os
import csv
import datetime
```

- Importing necessary Libraries.

```
# Paths to data files
user_data_path = "faces.json"
attendance_file_path = "Attendance.csv"

# Load existing user data if available, or initialize an empty dictionary
if os.path.exists(user_data_path):
    with open(user_data_path, "r") as file:
        try:
```

```
            user_data = json.load(file)
        except json.JSONDecodeError:
            user_data = {}
else:
    user_data = {}

# Columns for the attendance CSV file
columns = ["Name", "Contact Number", "Employ ID"]

# Create the attendance CSV file if it doesn't exist
if not os.path.exists(attendance_file_path):
    with open(attendance_file_path, "w", newline="") as file:
        writer = csv.writer(file)
        writer.writerow(columns)
```

- The above lines of code there are two files user_data_path and attendance_file_path in which the faces json file is used for the storing of the user data which are Name, Contact Number, Employ ID and the Face Encoding.The attendance_file_path Attendance csv file if used for the noting of the employ entry and the exit timings in a day.The above lines are for creating the respective files of they doesn't exist and writing in them.

```
# Create the main Tkinter window
root = tk.Tk()
root.title("Face Recognition App")
root.geometry("800x600")
```

- The above lines are for creating the Tkinter window with the Title as Face Recognition App and with a specific window size.

```
# Frames for different sections of the GUI
register_frame = tk.Frame(root)
recognize_frame = tk.Frame(root)
delete_frame = tk.Frame(root)
```

- These above lines are for creating the separate frames in which when we click on the respective buttons there will be frame displayed on the GUI.

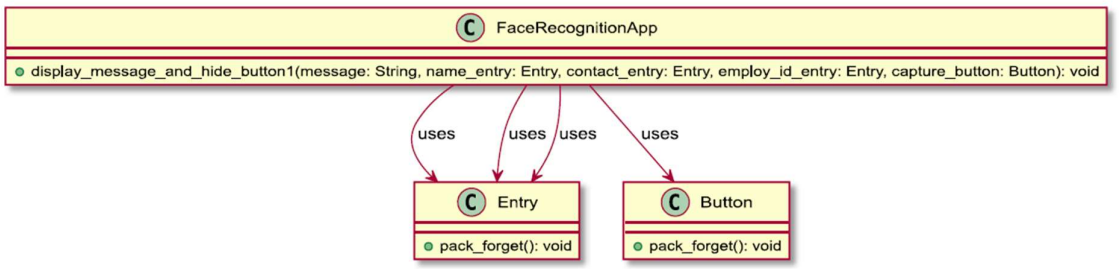
```
# Variable to keep track of the current displayed frame
current_frame = None

# Video capture from the default camera (0)
video_capture = cv2.VideoCapture(0)
```

```
# Function to show a frame in the GUI
def show_frame(frame):
    global current_frame
    if current_frame:
        current_frame.pack_forget()
    if frame.winfo_ismapped() == 0:
        frame.pack()
    current_frame = frame
```

These above lines are for the visibility of the live video frame on the GUI, such that the user himself can see in that frame and it will be updated time to time.

```
# Function to display a message, hide entry fields, and a button
def display_message_and_hide_button1(message, name_entry, contact_entry,
employ_id_entry, capture_button):
    messagebox.showinfo("Info", message)
    name_entry.pack_forget()
    contact_entry.pack_forget()
    employ_id_entry.pack_forget()
    capture_button.pack_forget()
```

- The above function is for the displaying the message box about the respective details in the functions register_user_gui, recognize_face and delete_face such that then the respective entry boxes and capture button will be disappeared.



The register_user_gui() function:

```
def register_user_gui():
    global x
    if x:
        return
    x = True
    show_frame(register_frame)
```

- The above lines are foe displaying the register_frame such that if it already exists on the gui then it won't again create the frame.

```
name_entry = tk.Entry(register_frame, width=40)
name_entry.insert(0, "Enter your name")
name_entry.pack()

contact_entry = tk.Entry(register_frame, width=40)
contact_entry.insert(0, "Enter your contact number")
contact_entry.pack()

employ_id_entry = tk.Entry(register_frame, width=40)
employ_id_entry.insert(0, "Enter your employ ID")
employ_id_entry.pack()
```

- These lines are for the taking the input which are the Employ Name, Contact Number and Employ ID such that these details can be stored in the faces.json file and in the employ_details.csv file.

```
# Functions to handle placeholder text
def on_name_entry_click(event):
    if name_entry.get() == "Enter your name":
        name_entry.delete(0, tk.END)

def on_contact_entry_click(event):
    if contact_entry.get() == "Enter your contact number":
        contact_entry.delete(0, tk.END)

def on_employ_id_entry_click(event):
    if employ_id_entry.get() == "Enter your employ ID":
        employ_id_entry.delete(0, tk.END)

name_entry.bind("<Button-1>", on_name_entry_click)
contact_entry.bind("<Button-1>", on_contact_entry_click)
employ_id_entry.bind("<Button-1>", on_employ_id_entry_click)
```

- The above lines are for the specifying to the entry box and the employ details and it also performs the handle placeholder text which is previously the entry field is specified and on clicking the text of info will be disappeared.

```
    # Function to capture the user's face
    def capture_face():
        global x
        ret, frame = video_capture.read()
        face_locations = face_recognition.face_locations(frame)

        if not face_locations:
            messagebox.showerror("Error", "No face detected. Please ensure
your face is well-lit and visible.")
            x = False
            return
        elif len(face_locations) > 1:
            messagebox.showinfo("Info", "Multiple faces detected. Please
ensure only one face is in the frame.")
            return

        face_encoding = face_recognition.face_encodings(frame,
face_locations)[0]

        # Check for similar faces to prevent duplicate registrations
        for existing_user, user_info in user_data.items():
            existing_encoding = user_info.get("encoding")
            if existing_encoding is not None:
                distance = face_recognition.face_distance([existing_encoding],
face_encoding)[0]
                if distance < 0.5:
                    display_message_and_hide_button1(f"Similar face already
registered as {existing_user}. Skipping registration.", name_entry,
contact_entry, employ_id_entry, capture_button)
                    x = False
                    return

        # Get user data
        name = name_entry.get()
        contact = contact_entry.get()
        employ_id = employ_id_entry.get()

        # Store user data in the JSON file
        user_data[name] = {
            "contact": contact,
            "employ_id": employ_id,
            "encoding": face_encoding.tolist()
        }

        with open(user_data_path, "w") as file:
            json.dump(user_data, file, indent=4)

        # Store user data in the CSV file
```

```
        with open('employ_details.csv', mode='a', newline='') as employ_file:
            employ_writer = csv.writer(employ_file)
            if employ_file.tell() == 0:
                employ_writer.writerow(["Name", "Contact Number", "Employ
ID"])
            employ_writer.writerow([name, contact, employ_id])

        display_message_and_hide_button1(f"User {name} registered
successfully!", name_entry, contact_entry, employ_id_entry, capture_button)
        x = False

    capture_button = tk.Button(register_frame, text="Capture Face",
command=capture_face, width=20, height=2)
    capture_button.pack()
```

- The capture_face function captures a frame from a video source (assumed to be initialized elsewhere as video_capture), detects faces in the frame, and performs various actions related to face recognition and user data storage.

```
        if not face_locations:
            messagebox.showerror("Error", "No face detected. Please ensure
your face is well-lit and visible.")
            x = False
            return
```

- If no face is detected in the frame, it displays an error message using the messagebox.showerror function and sets a global variable x to False.

```
        elif len(face_locations) > 1:
            messagebox.showinfo("Info", "Multiple faces detected. Please
ensure only one face is in the frame.")
            return
```

- If more than one face is detected, it displays an informational message using messagebox.showinfo.

```
face_encoding = face_recognition.face_encodings(frame, face_locations)[0]
```

- It extracts the facial encoding of the detected face using face_recognition.face_encodings.

```
        for existing_user, user_info in user_data.items():
            existing_encoding = user_info.get("encoding")
            if existing_encoding is not None:
                distance = face_recognition.face_distance([existing_encoding],
face_encoding)[0]
```

```
                if distance < 0.5:
                    display_message_and_hide_button1(f"Similar face already
registered as {existing_user}. Skipping registration.", name_entry,
contact_entry, employ_id_entry, capture_button)
                    x = False
                    return
```

- It checks for similar faces by comparing the newly captured face's encoding with previously registered faces. If a similar face is found, it displays a message and skips the registration.

```
        name = name_entry.get()
        contact = contact_entry.get()
        employ_id = employ_id_entry.get()
```

- It retrieves user data such as name, contact, and employee ID.

```
        # Store user data in the JSON file
        user_data[name] = {
            "contact": contact,
            "employ_id": employ_id,
            "encoding": face_encoding.tolist()
        }

        with open(user_data_path, "w") as file:
            json.dump(user_data, file, indent=4)

        # Store user data in the CSV file
        with open('employ_details.csv', mode='a', newline='') as employ_file:
            employ_writer = csv.writer(employ_file)
            if employ_file.tell() == 0:
                employ_writer.writerow(["Name", "Contact Number", "Employ
ID"])
            employ_writer.writerow([name, contact, employ_id])
```

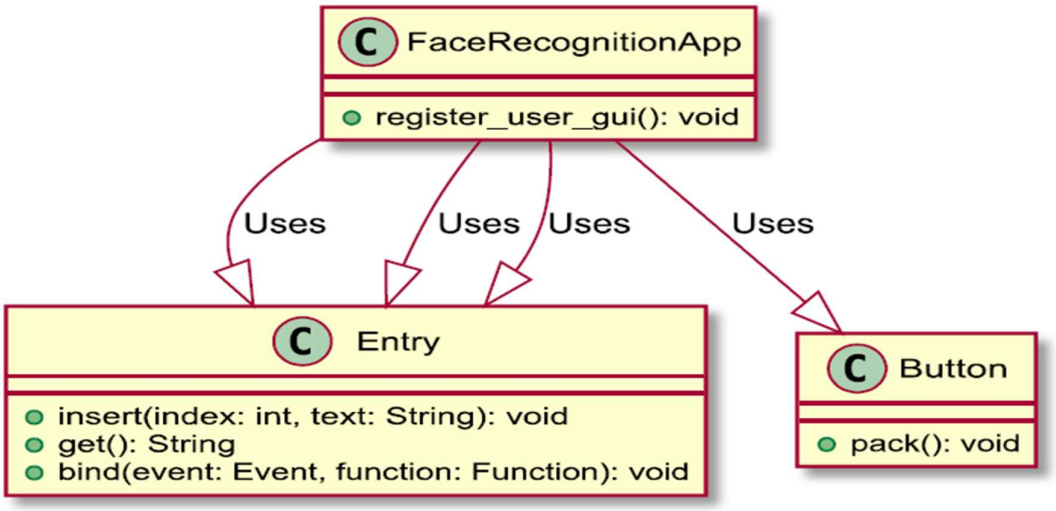
- It stores user data in a JSON file and a CSV file.

```
        display_message_and_hide_button1(f"User {name} registered
successfully!", name_entry, contact_entry, employ_id_entry, capture_button)
        x = False
```

- It displays a success message and hides a button (assumed to be defined elsewhere as display_message_and_hide_button1) to indicate a successful registration.

```
    capture_button = tk.Button(register_frame, text="Capture Face",
command=capture_face, width=20, height=2)
    capture_button.pack()
```

- Finally, it creates a Tkinter button labeled "Capture Face" associated with the capture_face function.



The recognize_face() function:

- This below function mainly does two categories which are recognizing the person in the frame and adding the time of IN or OUT in the attendance csv file of the respective employ or user. Such that by this function the attendance will be notified.

```
# Function to recognize a user through the GUI
def recognize_face():
    ret, frame = video_capture.read()
    face_locations = face_recognition.face_locations(frame)
    if not face_locations:
        messagebox.showinfo("Info", "No face detected.")
        return
    elif len(face_locations) > 1:
        messagebox.showinfo("Info", "Multiple faces detected. Please ensure
only one face is in the frame.")
        return

    face_encoding = face_recognition.face_encodings(frame, face_locations)[0]
```

```
    recognized_user = None
    min_distance = 0.5

    # Find the recognized user with the smallest distance
    for name, user_info in user_data.items():
        registered_encoding = user_info["encoding"]
        distance = face_recognition.face_distance([registered_encoding],
face_encoding)[0]

        if distance < min_distance:
            min_distance = distance
            recognized_user = name

    if recognized_user:
        # Get user contact and employee ID
        contact_info = user_data[recognized_user]['contact']
        employ_id_info = user_data[recognized_user]['employ_id']
        current_date = datetime.datetime.now().strftime("%Y-%m-%d")

        # Read and update the attendance CSV file
        with open(attendance_file_path, "r") as file:
            reader = csv.DictReader(file)
            columns = reader.fieldnames
            rows = list(reader)

        date_in_column = f"{current_date} (IN)"
        date_out_column = f"{current_date} (OUT)"

        if date_in_column not in columns:
            columns.extend([date_in_column, date_out_column])

        found_user = False

        for row in rows:
            if row["Name"] == recognized_user:
                found_user = True
                if not row.get(date_in_column):
                    row[date_in_column] =
datetime.datetime.now().strftime("%H:%M:%S")
                else:
                    row[date_out_column] =
datetime.datetime.now().strftime("%H:M:S")

        if not found_user:
            new_row = {col: "" for col in columns}
            new_row["Name"] = recognized_user
            new_row["Contact Number"] = contact_info
            new_row["Employ ID"] = employ_id_info
```

```
            new_row[date_in_column] =
datetime.datetime.now().strftime("%H:M:S")
            rows.append(new_row)

        with open(attendance_file_path, "w", newline="") as file:
            writer = csv.DictWriter(file, fieldnames=columns)
            writer.writeheader()
            writer.writerows(rows)

        # Display recognized user information
        messagebox.showinfo("Info", f"Face matched with registered user:
{recognized_user}\nContact: {contact_info}\nEmploy ID: {employ_id_info}")
    else:
        messagebox.showinfo("Info", "User not recognized")
```

```
    ret, frame = video_capture.read()
    face_locations = face_recognition.face_locations(frame)
```

- Face Detection: It captures a frame from a video source (assumed to be initialized elsewhere as video_capture) and detects face locations using the face_recognition.face_locations function.

```
    if not face_locations:
        messagebox.showinfo("Info", "No face detected.")
        return
    elif len(face_locations) > 1:
        messagebox.showinfo("Info", "Multiple faces detected. Please ensure
only one face is in the frame.")
        return
```

- No Face Detected Handling: If no face is detected in the frame, it displays an informational message using messagebox.showinfo and exits the function.Multiple Faces Detected Handling: If more than one face is detected, it displays an informational message to ensure that only one face is in the frame and then exits.

```
    face_encoding = face_recognition.face_encodings(frame, face_locations)[0]
```

- Face Encoding: It extracts the facial encoding of the detected face using face_recognition.face_encodings.

```
    recognized_user = None
    min_distance = 0.5

    # Find the recognized user with the smallest distance
    for name, user_info in user_data.items():
        registered_encoding = user_info["encoding"]
        distance = face_recognition.face_distance([registered_encoding],
face_encoding)[0]

        if distance < min_distance:
            min_distance = distance
            recognized_user = name
```

- Recognized User Initialization: It initializes variables to keep track of the recognized user and the minimum distance (set to 0.5, but you can adjust this threshold). It is first initialized as “none” and after detecting the face by minimum distance the “none” is updated.

```
    if recognized_user:
        # Get user contact and employee ID
        contact_info = user_data[recognized_user]['contact']
        employ_id_info = user_data[recognized_user]['employ_id']
        current_date = datetime.datetime.now().strftime("%Y-%m-%d")

        # Read and update the attendance CSV file
        with open(attendance_file_path, "r") as file:
            reader = csv.DictReader(file)
            columns = reader.fieldnames
            rows = list(reader)

        date_in_column = f"{current_date} (IN)"
        date_out_column = f"{current_date} (OUT)"

        if date_in_column not in columns:
            columns.extend([date_in_column, date_out_column])

        found_user = False

        for row in rows:
            if row["Name"] == recognized_user:
                found_user = True
                if not row.get(date_in_column):
                    row[date_in_column] =
datetime.datetime.now().strftime("%H:%M:%S")
                else:
                    row[date_out_column] =
datetime.datetime.now().strftime("%H:M:S")
```

```
        if not found_user:
            new_row = {col: "" for col in columns}
            new_row["Name"] = recognized_user
            new_row["Contact Number"] = contact_info
            new_row["Employ ID"] = employ_id_info
            new_row[date_in_column] =
datetime.datetime.now().strftime("%H:M:S")
            rows.append(new_row)

        with open(attendance_file_path, "w", newline="") as file:
            writer = csv.DictWriter(file, fieldnames=columns)
            writer.writeheader()
            writer.writerows(rows)
```

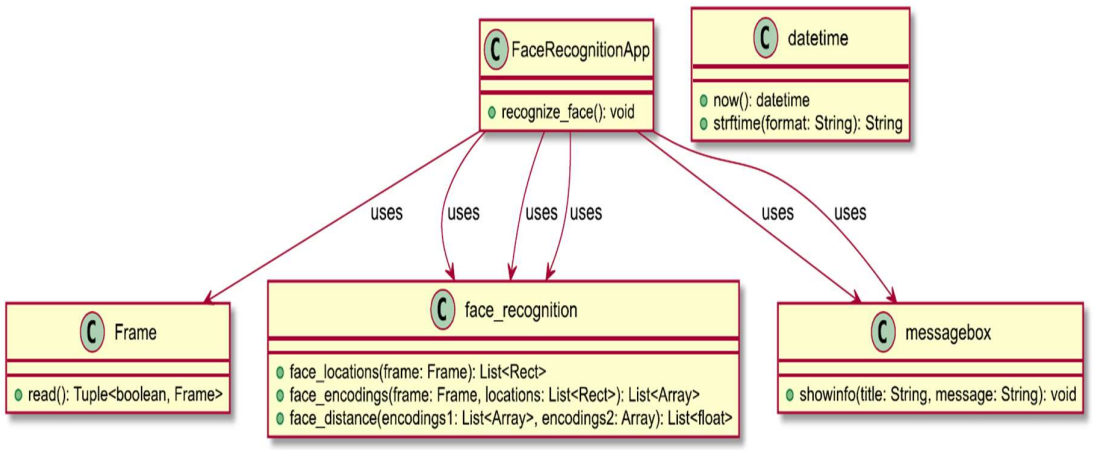
- Updating Attendance Records: If a recognized user is found, it updates the attendance records. It reads the existing CSV file (assumed to be defined by attendance_file_path), appends the current date and time to the "IN" column if the user is entering, and appends it to the "OUT" column if the user is leaving. If the user is not found in the CSV file, a new row is created with the user's information and the "IN" time.

```
        # Display recognized user information
        messagebox.showinfo("Info", f"Face matched with registered user:
{recognized_user}\nContact: {contact_info}\nEmploy ID: {employ_id_info}")
```

- Display User Information: If a user is recognized, it displays an informational message using messagebox.showinfo, showing the recognized user's name, contact information, and employee ID.

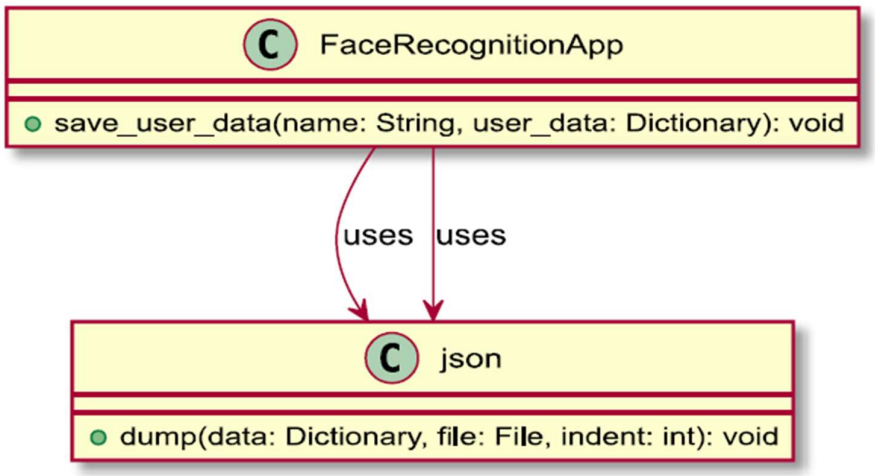
```
    else:
        messagebox.showinfo("Info", "User not recognized")
```

- User Not Recognized: If no recognized user is found, it displays an informational message indicating that the user is not recognized.




```
# Function to save user data to the JSON file
def save_user_data(name, user_data):
    with open(user_data_path, "w") as file:
        json.dump(user_data, file, indent=4)
```

- The save_user_data function is called within the delete_face function. When a user's face is recognized and the user's data needs to be deleted, the delete_face function removes the user's data from the user_data dictionary (which likely contains information about registered users). After the removal, it calls the save_user_data function to update the JSON file with the modified user_data dictionary.



The delete_face() function:

```
# Function to delete a registered face
def delete_face():
    ret, frame = video_capture.read()
    face_locations = face_recognition.face_locations(frame)
    if not face_locations:
        messagebox.showinfo("Info", "No face detected.")
        return
    elif len(face_locations) > 1:
        messagebox.showinfo("Info", "Multiple faces detected. Please ensure
only one face is in the frame.")
        return


    face_encoding = face_recognition.face_encodings(frame, face_locations)[0]

    recognized_user = None

    for name, user_info in user_data.items():
        registered_encoding = user_info["encoding"]
        distance = face_recognition.face_distance([registered_encoding],
face_encoding)[0]

        if distance < 0.5:
            recognized_user = name
            break

    if recognized_user:
        # Ask for confirmation before deleting user data
        confirmation = messagebox.askquestion("Confirm Deletion", f"Shall I
delete the User {recognized_user}'s data?")
        if confirmation == "yes":
            del user_data[recognized_user]
            save_user_data(recognized_user, user_data)
            messagebox.showinfo("Info", f"User {recognized_user}'s data has
been deleted.")
        else:
            messagebox.showinfo("Info", "Deletion canceled.")
        return
    messagebox.showinfo("Info", "User not recognized")
```

```
    ret, frame = video_capture.read()
    face_locations = face_recognition.face_locations(frame)
```

- Face Detection: It captures a frame from a video source (assumed to be initialized elsewhere as video_capture) and detects face locations using the face_recognition.face_locations function.

```
    if not face_locations:
        messagebox.showinfo("Info", "No face detected.")
        return
    elif len(face_locations) > 1:
        messagebox.showinfo("Info", "Multiple faces detected. Please ensure
only one face is in the frame.")
        return
```

- No Face Detected and Multiple Face Detected Handling: If no face is detected in the frame, it displays an informational message using messagebox.showinfo and exits the function.

```
    face_encoding = face_recognition.face_encodings(frame, face_locations)[0]
```

- Face Encoding: It extracts the facial encoding of the detected face using face_recognition.face_encodings. This encoding is used to compare the detected face with registered faces.

```
    recognized_user = None

    for name, user_info in user_data.items():
        registered_encoding = user_info["encoding"]
        distance = face_recognition.face_distance([registered_encoding],
face_encoding)[0]

        if distance < 0.5:
            recognized_user = name
            break
```

- User Recognition Loop: The function then loops through the user_data dictionary, which presumably contains information about registered users. For each registered user, it calculates the distance between the encoding of the detected face and the encoding of the registered user's face. If the distance is less than 0.5 (you can adjust this threshold), it considers the user as "recognized."

```
        # Ask for confirmation before deleting user data
        confirmation = messagebox.askquestion("Confirm Deletion", f"Shall I
delete the User {recognized_user}'s data?")
```

- Confirmation for Deletion: If a recognized user is found, it asks for user confirmation before proceeding with the deletion. A messagebox is used to prompt the user with a confirmation question.

```
        if confirmation == "yes":
            del user_data[recognized_user]
```

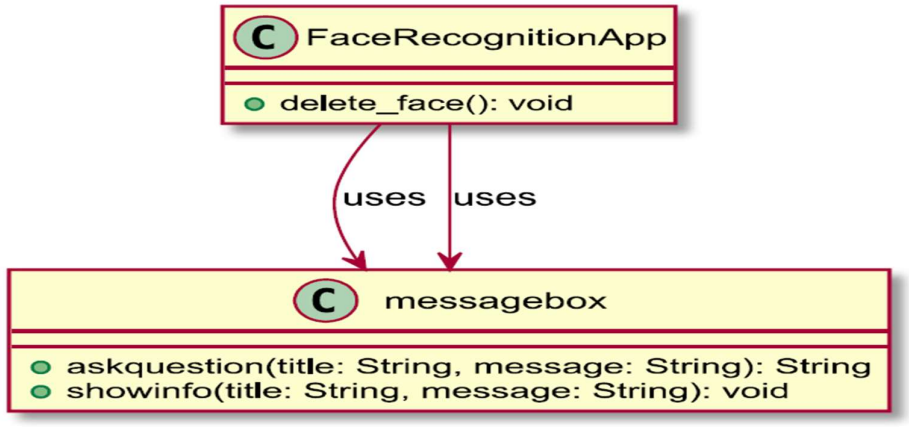
- User Data Deletion: If the user confirms the deletion, it removes the recognized user's data from the user_data dictionary using del user_data[recognized_user].

```
            save_user_data(recognized_user, user_data)
```

- Updating User Data File: After the removal, it calls the save_user_data function (not shown in this code snippet, but likely defined elsewhere) to update the JSON file that stores user data. This ensures that the changes are reflected in the persistent storage.

```
            messagebox.showinfo("Info", f"User {recognized_user}'s data has
been deleted.")
```

- Notification and Exit: Finally, an informational message is displayed to inform the user that the user's data has been deleted. If the user cancels the deletion, another informational message is displayed, and the function exits.



```
# Label to display video feed from the camera
video_label = tk.Label(root, width=400, height=300)
video_label.pack(side="left", padx=10, pady=10)

# Function to update the video feed
def update_video_frame():
    ret, frame = video_capture.read()
    if ret:
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        frame = Image.fromarray(frame)
        frame = ImageTk.PhotoImage(image=frame)
        video_label.img = frame
        video_label.config(image=frame)
        video_label.after(10, update_video_frame)
    else:
        video_label.after(10, update_video_frame)

update_video_frame()
```

```
video_label = tk.Label(root, width=400, height=300)
video_label.pack(side="left", padx=10, pady=10)
```

- Video Label: The code initializes a tk.Label widget called video_label that will be used to display the video feed from a camera in a graphical user interface (GUI). The label is given a width of 400 pixels and a height of 300 pixels.

```
    ret, frame = video_capture.read()
```

- Reading Video Frames: Inside the update_video_frame function, it captures a frame from a video source (presumably a camera) using video_capture.read(). The variable ret is used to check if a frame was successfully retrieved.

```
frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
frame = Image.fromarray(frame)
frame = ImageTk.PhotoImage(image=frame)
```

- If a frame is successfully captured (ret is True), it is then converted from the OpenCV format (BGR color space) to RGB format using cv2.cvtColor to ensure it can be displayed correctly.The frame is further converted into an Image object using the Image.fromarray function. This is a necessary step to display the frame in a Tkinter label.The Image is then converted to an ImageTk.PhotoImage object, which is a Tkinter-compatible image format.

```
        video_label.img = frame
        video_label.config(image=frame)
        video_label.after(10, update_video_frame)
    else:
        video_label.after(10, update_video_frame)
```

- Updating the Label: The image attribute of the video_label is updated with the new frame, and the video_label widget is updated to display the latest frame.

```
update_video_frame()
```

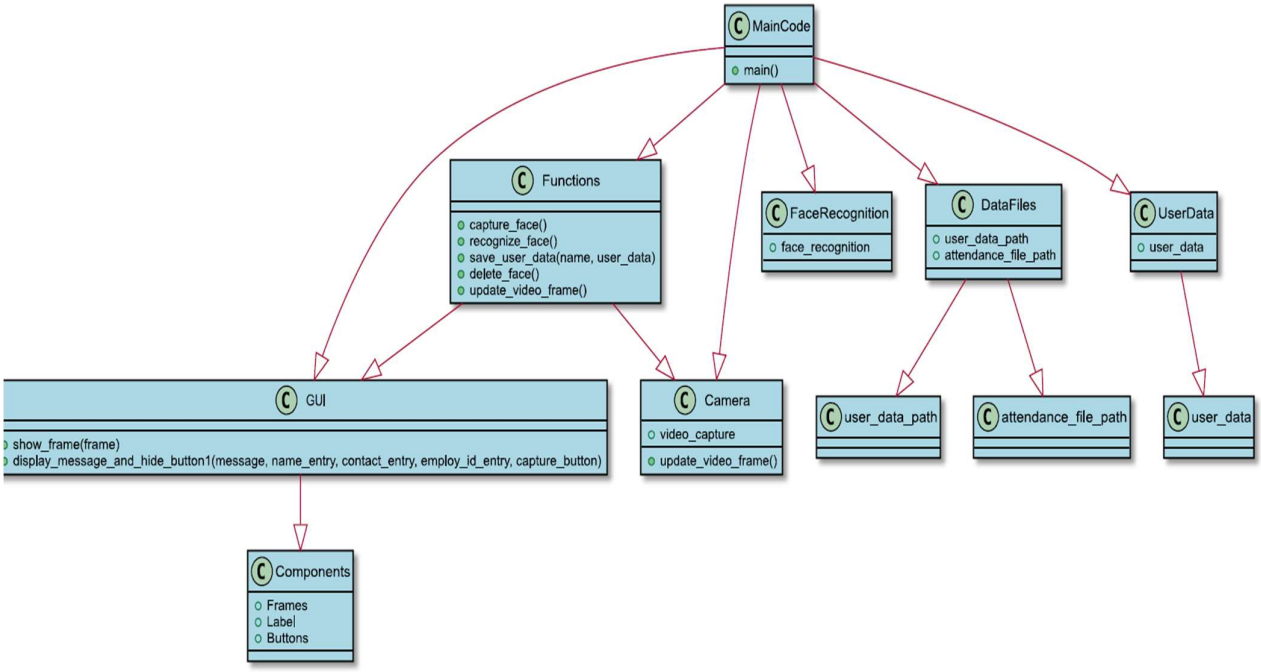
- Continual Update: The update_video_frame function is set to be called repeatedly with a delay of 10 milliseconds using video_label.after(10, update_video_frame). This ensures that the video feed is continuously updated, creating a real-time video display in the GUI.

Buttons in GUI:

```
# Frame to hold buttons
button_frame = tk.Frame(root)
button_frame.pack(side="left", padx=10, pady=10)

# Buttons for various actions
register_button = tk.Button(root, text="Register a new user",
command=register_user_gui, width=20, height=2)
register_button.pack(pady=10)

recognize_button = tk.Button(root, text="Recognize a user",
command=recognize_face, width=20, height=2)
recognize_button.pack(pady=10)

delete_button = tk.Button(root, text="Delete a user", command=delete_face,
width=20, height=2)
delete_button.pack(pady=10)

exit_button = tk.Button(root, text="Exit", command=root.quit, width=20,
height=2)
exit_button.pack(pady=10)
```

- Button Frame: A frame to hold buttons, providing structure to the user interface.
- Register Button: Initiates user registration.
- Recognize Button: Activates user recognition functionality.
- Delete Button: Deletes a registered user's data.
- Exit Button: Closes the GUI application.

UML Diagram:


MainCode
main()
Functions
capture_face()
recognize_face()
save_user_data(name, user_data)
delete_face()
update_video_frame()
FaceRecognition
face_recognition
DataFiles
user_data_path
attendance_file_path
UserData
user_data
GUI
show_frame(frame)
display_message_and_hide_button1(message, name_entry, contact_entry, employ_id_entry, capture_button)
Camera
video_capture
update_video_frame()
user_data_path
attendance_file_path
user_data
Components
Frames
Label
Buttons