```python
import shutil
import tkinter
import tkinter as tk
from tkinter import messagebox

import customtkinter
import cv2
import torch
import torchvision.transforms as transforms
from PIL import Image
from customtkinter import *
from facenet_pytorch import MTCNN, InceptionResnetV1


def recognize_face(input_features, database_features):
    recognized_person = ""
    min_distance = float('inf')
    for person, person_features in database_features.items():
        for feature in person_features:
            distance = ((input_features - feature) ** 2).sum()
            if distance < min_distance:
                min_distance = distance
                recognized_person = person
    return recognized_person if min_distance < 1.0 else None


def display_recognition_result(frame, faces_cascade, recognized_person):
    for (x, y, w, h) in faces_cascade:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(frame, f"{recognized_person}", (x, y - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)


def display_unknown_result(frame, faces_cascade):
    for (x, y, w, h) in faces_cascade:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(frame, "Necunoscut", (x, y - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)


class FaceRecognitionApp:
    def __init__(self, root, mtcnn, model, database_features, face_cascade, database_path):
        self.root = root
        self.root.title("Aplicatie pentru recunoastere faciala")
        self.height = root.winfo_screenheight()
        self.width = root.winfo_screenwidth()
        self.root.geometry("%dx%d" % (0.6 * self.width, 0.7 * self.height))
        self.mtcnn = mtcnn
        self.model = model
        self.database_features = database_features
        self.face_cascade = face_cascade
        self.database_path = database_path
        self.cap = cv2.VideoCapture(0)
        self.snapshot_index = 1
        if not self.cap.isOpened():
            messagebox.showerror("Error", "Nu s-a putut deschide camera. Verifica i conexiunea  i resursele h
ardware.")
            self.root.destroy()
            return
        self.frame_3 = CTkFrame(master=root)
        self.frame_3.grid(row=1, column=1)
        CTkLabel(master=self.frame_3, text="Alege o optiune din stanga", font=("Arial Bold", 20), justify="left
").pack(
            expand=True, pady=(15, 15))

        self.frame_1 = CTkFrame(master=root, fg_color="#CD8C67")
        self.frame_1.grid(row=0, column=0, rowspan=2, sticky="nsew", padx=15, pady=15)

        self.label1 = CTkLabel(master=self.frame_1, text="Alege:", font=("Arial Bold", 20), justify="left")
        self.label1.place(relx=0.5, rely=0.1, anchor=CENTER)

        self.radio_var = tkinter.StringVar(value="default")
        self.detP = customtkinter.CTkRadioButton(master=self.frame_1, variable=self.radio_var,
                                                 command=self.radiobutton_event,
                                                 text="Detectie Persoana", value="Detectie Persoana",
                                                 fg_color="#000000")
        self.addP = customtkinter.CTkRadioButton(master=self.frame_1, variable=self.radio_var,
                                                 command=self.radiobutton_event,
                                                 text="Adaugare Persoana", value="Adaugare Persoana",
                                                 fg_color="#000000")
        self.delP = customtkinter.CTkRadioButton(master=self.frame_1, variable=self.radio_var,
                                                 command=self.radiobutton_event,
                                                 text="Stergere Persoana", value="Stergere Persoana",
                                                 fg_color="#000000")

        self.detP.place(relx=0.48, rely=0.3, anchor=CENTER)
        self.addP.place(relx=0.5, rely=0.4, anchor=CENTER)
        self.delP.place(relx=0.48, rely=0.5, anchor=CENTER)
        self.quit_button = customtkinter.CTkButton(master=self.frame_1, text="Inchide App", command=self.
quit_app)
        self.quit_button.place(relx=0.5, rely=0.6, anchor=CENTER)

        self.frame_2 = CTkFrame(master=root, fg_color="#606190")
        self.frame_2.grid(row=0, column=1, padx=15, pady=15)
        self.video_label = customtkinter.CTkLabel(master=self.frame_2, text="")
        self.video_label.pack()
        self.warning_label = customtkinter.CTkLabel(master=self.frame_3,
                                                    text="Nu s-a putut detecta fata in imagine", fg_color="red",
                                                    font=("Helvetica", 14))
        self.warning_label.pack_forget()
        self.recognition_enabled = False
        self.user_name_entry = CTkEntry(master=self.frame_3, placeholder_text="Nume_Prenume", width=
400)
        self.user_name_entry.pack_forget()
        self.add_user_mode = False
        self.user_message_label = customtkinter.CTkLabel(master=self.frame_3,
                                                         text="Introdu numele utilizatorului, zâmbeste  i fă- i câte "
                                                              "poze dore ti. La final apasă pe \"Finalizare\"",
                                                         font=("Helvetica", 12))
        self.user_message_label.pack_forget()
        self.take_photo_button = customtkinter.CTkButton(master=self.frame_3, text="Fa poza",
                                                         command=self.take_photo,
                                                         bg_color="blue", fg_color="white",
                                                         font=("Helvetica", 12, "bold"))
        self.take_photo_button.pack_forget()
        self.delete_user_button = customtkinter.CTkButton(master=self.frame_3, text="Sterge Utilizator",
                                                          command=self.confirm_delete_user,
                                                          bg_color="red", fg_color="white",
                                                          font=("Helvetica", 12, "bold"))
        self.delete_user_button.pack_forget()
        self.status_label = customtkinter.CTkLabel(master=self.frame_3, text="", font=("Helvetica", 12))
        self.status_label.pack_forget()
        self.finalize_button = customtkinter.CTkButton(master=self.frame_3, text="Finalizare",
                                                       command=self.finalize_add_user,
                                                       bg_color="green", fg_color="white",
                                                       font=("Helvetica", 12, "bold"))
        self.finalize_button.pack_forget()

        self.update_video()

    def detect_faces(self, frame):
        return self.mtcnn(frame)

    def detect_faces_cascade(self, gray_frame):
        return self.face_cascade.detectMultiScale(gray_frame, scaleFactor=1.3, minNeighbors=5)

    def update_video(self):
        ret, frame = self.cap.read()
        if ret:
            faces_cascade = self.detect_faces_cascade(cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY))

            for (x, y, w, h) in faces_cascade:
                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

            if self.recognition_enabled:
                faces = self.detect_faces(frame)
                if faces is not None and len(faces) > 0:
                    img_resized = transforms.Resize((480, 640))(Image.fromarray(frame))
                    input_features = extract_face_features(img_resized, self.mtcnn, self.model, set_device())

                    if input_features is not None:
                        recognized_person = recognize_face(input_features, self.database_features)
                        if recognized_person is not None:
                            display_recognition_result(frame, faces_cascade, recognized_person)
                            self.hide_warning()
                        else:
                            display_unknown_result(frame, faces_cascade)

                else:
                    display_unknown_result(frame, faces_cascade)
                    self.show_warning()
            else:
                self.hide_warning()

            self.display_frame(frame)
    self.root.after(10, self.update_video)

def display_frame(self, frame):
    rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    pil_image = Image.fromarray(rgb_image)
    tk_image = CTkImage(pil_image, size=(0.4 * self.width, 0.4 * self.height))
    self.video_label.img = tk_image
    self.video_label.configure(image=tk_image)
    self.video_label.update_idletasks()

def detect_and_recognize(self):
    self.recognition_enabled = not self.recognition_enabled

def show_warning(self):
    self.warning_label.pack()

def hide_warning(self):
    self.warning_label.pack_forget()

def quit_app(self):
    self.cap.release()
    self.root.destroy()

def clear_frame(self):
    for widgets in self.frame_3.winfo_children():
        widgets.destroy()

def add_user(self):
    user_name = self.user_name_entry.get()
    if user_name:
        print(f"Adaugare utilizator: {user_name}")
    else:
        messagebox.showwarning("Avertisment", "Introduceti un nume de utilizator valid.")

def finalize_add_user(self):
    user_name = self.user_name_entry.get()

    if not user_name:
        self.status_label.configure(text="Introduceti un nume de utilizator valid.")
        return

    if self.snapshot_index <= 1:
        self.status_label.configure(text="Face i cel pu in o poză înainte de a finaliza.")
        return

    user_folder_path = os.path.join('Images', user_name)
    os.makedirs(user_folder_path, exist_ok=True)

    for i in range(1, self.snapshot_index):
        old_file_name = f"{user_name}_{i:04d}.jpg"
        new_file_path = os.path.join(user_folder_path, old_file_name)
        shutil.move(old_file_name, new_file_path)

    self.snapshot_index = 1
```

```
        success_message = f"Utilizatorul '{user_name}' a fost adăugat cu succes!"
        self.user_message_label.configure(text=success_message)
        self.root.after(2500, self.update_database_and_recognize)
        self.user_message_label.configure(text="Baza de date a fost actualizata cu succes")
        self.root.after(2500, self.refresh)

    def take_photo(self):
        user_name = self.user_name_entry.get()
        if not user_name:
            self.status_label.configure(text="Introduceti un nume de utilizator valid.")
            return
        ret, frame = self.cap.read()
        if not ret:
            messagebox.showerror("Eroare", "Nu s-a putut ob ine frame-ul de la cameră.")
            return
        file_name = f"{user_name}_{self.snapshot_index:04d}.jpg"
        file_path = os.path.join(os.getcwd(), file_name)
        cv2.imwrite(file_path, cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
        self.snapshot_index += 1
        self.user_message_label.pack()
        self.status_label.configure(text="")
        self.user_message_label.configure(text=f"Ai făcut {self.snapshot_index - 1} poze. Fă- i câte poze dor
e ti.")

    def refresh_del(self):
        self.clear_frame()
        self.recognition_enabled = False
        CTkLabel(master=self.frame_3, text="Pe cine doresti sa stergi?", font=("Arial Bold", 20),
                 justify="left").pack(expand=True, pady=(30, 15))
        self.user_name_entry = CTkEntry(master=self.frame_3, placeholder_text="Nume_Prenume", width=
400)
        self.user_name_entry.pack(expand=True, pady=15, padx=20)
        self.delete_user_button = customtkinter.CTkButton(master=self.frame_3, text="Sterge Utilizator",
                                                          command=self.confirm_delete_user,
                                                          bg_color="red", fg_color="white",
                                                          font=("Helvetica", 12, "bold"))
        self.delete_user_button.pack()
        self.status_label = customtkinter.CTkLabel(master=self.frame_3, text="", font=("Helvetica", 12))
        self.status_label.pack()

    def confirm_delete_user(self):
        user_name = self.user_name_entry.get()
        if not user_name:
            self.status_label.configure(text="Introdu un nume de utilizator valid.")
            return

        user_folder_path = os.path.join('Images', user_name)
        try:
            shutil.rmtree(user_folder_path)
        except OSError as e:
            self.status_label.configure(text=f"Eroare la stergerea utilizatorului '{user_name}': {e}.")
            return

        success_message = f"Utilizatorul '{user_name}' a fost sters cu succes!"
        self.status_label.configure(text=success_message)
        self.user_name_entry.pack_forget()
        self.user_name_entry.delete(0, tk.END)
        self.user_message_label.pack_forget()
        self.root.after(2500, self.update_database_and_recognize)
        self.status_label.configure(text="Baza de date a fost actualizata cu succes")
        self.root.after(2500, self.refresh_del)

    def refresh(self):
        self.clear_frame()
        CTkLabel(master=self.frame_3, text="Introdu numele persoanei pe care o adaugam", font=("Arial Bo
ld", 20),
                 justify="left").pack(
            expand=True, pady=(30, 15))
        self.user_name_entry = CTkEntry(master=self.frame_3, placeholder_text="Nume_Prenume", width=
400)
        self.user_name_entry.pack(expand=True, pady=15, padx=20)
        self.user_message_label = customtkinter.CTkLabel(master=self.frame_3,
                                                         text="Introdu numele utilizatorului, zâmbeste  i fă- i câte "
                                                              "poze dore ti. La final apasă pe \"Finalizare\"",
                                                         font=("Helvetica", 12))
        self.user_message_label.pack()
        self.take_photo_button = customtkinter.CTkButton(master=self.frame_3, text="Fa poza",
                                                         command=self.take_photo,
                                                         bg_color="blue", fg_color="white",
                                                         font=("Helvetica", 12, "bold"))
        self.take_photo_button.pack(expand=True)
        self.finalize_button = customtkinter.CTkButton(master=self.frame_3, text="Finalizare",
                                                       command=self.finalize_add_user,
                                                       bg_color="green", fg_color="white",
                                                       font=("Helvetica", 12, "bold"))
        self.finalize_button.pack()
        self.status_label = customtkinter.CTkLabel(master=self.frame_3, text="", font=("Helvetica", 12))
        self.status_label.pack()

    def update_database_and_recognize(self):
        self.database = load_database_images(self.database_path)
        self.device = set_device()
        self.mtcnn, self.model = initialize_models(self.device)
        self.database_features = extract_database_features(self.database, self.mtcnn, self.model, self.devic
e)
        return self.database_features

    def radiobutton_event(self):
        if self.radio_var.get() == "default":
            self.clear_frame()
            self.recognition_enabled = False
            self.default_lable = CTkLabel(master=self.frame_3, text="Alege:", font=("Arial Bold", 20), justify="l
eft")
            self.default_lable.pack(expand=True)

        elif self.radio_var.get() == "Detectie Persoana":
            self.clear_frame()
            self.detect_button = customtkinter.CTkButton(master=self.frame_3, text="Detecteaza si recunoast
e",
                                                   command=self.detect_and_recognize)
            self.detect_button.pack()
            self.warning_label = customtkinter.CTkLabel(master=self.frame_3,
                                                  text="Nu s-a putut detecta fata in imagine", fg_color="red",
                                                  font=("Helvetica", 14))
            self.warning_label.pack_forget()

        elif self.radio_var.get() == "Adaugare Persoana":
            self.clear_frame()
            self.recognition_enabled = False
            CTkLabel(master=self.frame_3, text="Introdu numele persoanei pe care o adaugam", font=("Arial
Bold", 20),
                     justify="left").pack(
                expand=True, pady=(30, 15))
            self.user_name_entry = CTkEntry(master=self.frame_3, placeholder_text="Nume_Prenume", widt
h=400)
            self.user_name_entry.pack(expand=True, pady=15, padx=20)
            self.user_message_label = customtkinter.CTkLabel(master=self.frame_3,
                                                      text="Introdu numele utilizatorului, zâmbeste  i fă- i "
                                                           "câte poze dore ti. La final apasă pe \"Finalizare\"",
                                                      font=("Helvetica", 12))
            self.user_message_label.pack()
            self.take_photo_button = customtkinter.CTkButton(master=self.frame_3, text="Fa poza",
                                                      command=self.take_photo,
                                                      bg_color="blue", fg_color="white",
                                                      font=("Helvetica", 12, "bold"))
            self.take_photo_button.pack(expand=True)
            self.finalize_button = customtkinter.CTkButton(master=self.frame_3, text="Finalizare",
                                                    command=self.finalize_add_user,
                                                    bg_color="green", fg_color="white",
                                                    font=("Helvetica", 12, "bold"))
            self.finalize_button.pack()
            self.status_label = customtkinter.CTkLabel(master=self.frame_3, text="", font=("Helvetica", 12))
            self.status_label.pack()


        else:
            self.clear_frame()
            self.recognition_enabled = False
            CTkLabel(master=self.frame_3, text="Pe cine doresti sa stergi?", font=("Arial Bold", 20),
                     justify="left").pack(expand=True, pady=(30, 15))
            self.user_name_entry = CTkEntry(master=self.frame_3, placeholder_text="Nume_Prenume", widt
h=400)
            self.user_name_entry.pack(expand=True, pady=15, padx=20)
            self.delete_user_button = customtkinter.CTkButton(master=self.frame_3, text="Sterge Utilizator",
                                                       command=self.confirm_delete_user,
                                                       bg_color="red", fg_color="white",
                                                       font=("Helvetica", 12, "bold"))
            self.delete_user_button.pack()
            self.status_label = customtkinter.CTkLabel(master=self.frame_3, text="", font=("Helvetica", 12))
            self.status_label.pack()


def set_device():
    return torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
def initialize_models(device):
    mtcnn = MTCNN(keep_all=True, device=device)
    model = InceptionResnetV1(pretrained='vggface2').eval().to(device)
    return mtcnn, model


def load_database_images(database_path):
    database = {}
    for person in os.listdir(database_path):
        person_path = os.path.join(database_path, person)
        if os.path.isdir(person_path):
            database[person] = [os.path.join(person_path, image_name) for image_name in os.listdir(person_
path)]
    return database


def extract_face_features(img, mtcnn, model, device):
    img_cropped = mtcnn(img)
    if img_cropped is None:
        return None
    img_resized = transforms.Resize((160, 160))(img_cropped[0])
    img_resized = torch.as_tensor(img_resized, dtype=torch.float32)
    img_resized = img_resized.unsqueeze(0)
    img_resized = img_resized.to(device)
    features = model(img_resized)
    return features[0].detach().cpu().numpy()


def extract_database_features(database, mtcnn, model, device):
    database_features = {}
    for person, person_images in database.items():
        person_features = []
        for image_path in person_images:
            img = Image.open(image_path).convert('RGB')
            input_features = extract_face_features(img, mtcnn, model, device)
            if input_features is not None:
                person_features.append(input_features)
        database_features[person] = person_features
    return database_features


if __name__ == "__main__":
    device = set_device()
    mtcnn, model = initialize_models(device)
    database_path = 'Images'
    database = load_database_images(database_path)
    database_features = extract_database_features(database, mtcnn, model, device)

    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml'
)

    root = CTk()
    app = FaceRecognitionApp(root, mtcnn, model, database_features, face_cascade, database_path)
```

```
root.mainloop()
```