**Rapport PFA :**

**Chapitre 1 :**

1. **Introduction**
2. **Cahier de charge**
3. **Diagramme UML**

Diagramme de cas d’utilisation

Diagramme de classe

**Chapitre 2 :**

**Réalisation du projet**

**Chapitre 1 :**

**Analyse et conception**

* Introduction
* Cahier de charge
* Diagrammes Utilisés

1. **Introduction**

Ce projet vise à développer une application de gestion des employés qui inclut un système de reconnaissance faciale pour le suivi des heures de travail et des fonctionnalités pour la gestion des données des employés. L'application permet aux gestionnaires de suivre les performances, de générer des rapports, et de gérer les tâches et les horaires de travail. L'utilisation du langage Python et des algorithmes de reconnaissance faciale sont au cœur de cette solution. Le projet met également l'accent sur la sécurité des données et la confidentialité des employés.

1. **Cahier de charge**

**1. Gestion des employés**

* **Description**: Cette partie du système concerne la gestion des informations des employés, y compris leurs coordonnées, les informations personnelles, et les horaires de travail.
* **Fonctionnalités**:
  + Les gestionnaires peuvent ajouter, modifier, supprimer et consulter les employés.
  + Des fonctionnalités de recherche et de filtrage permettent aux gestionnaires de trouver rapidement des employés selon des critères tels que le département, le poste, et les performances.
* **Défis techniques**:
  + La sécurité des données est un enjeu majeur, car des informations personnelles sont stockées.
  + Le système doit être facile à utiliser et suffisamment flexible pour s'adapter aux besoins des différents départements.

**2. Système de reconnaissance faciale**

* **Description**: L'application intègre un système de reconnaissance faciale pour enregistrer les heures d'arrivée et de départ des employés.
* **Fonctionnalités**:
  + Les employés utilisent la reconnaissance faciale pour pointer leur arrivée et leur départ.
  + Les données collectées sont utilisées pour calculer automatiquement les heures de travail.
* **Défis techniques**:
  + Le choix de l'algorithme de reconnaissance faciale est crucial pour garantir précision et sécurité.
  + Des préoccupations en matière de confidentialité des données doivent être traitées pour protéger les identités des employés.
  + La tolérance aux erreurs (par exemple, les échecs de reconnaissance) et la capacité à gérer des visages nouveaux ou différents (comme les employés portant des lunettes ou des masques) sont des points importants.

**3. Suivi de la production**

* **Description**: Cette partie du système suit les tâches effectuées par chaque employé et permet de générer des rapports sur la production.
* **Fonctionnalités**:
  + L'application enregistre les tâches réalisées par les employés et peut utiliser les données de reconnaissance faciale pour les heures de travail.
  + Les gestionnaires peuvent générer des rapports pour évaluer les performances individuelles et collectives.
* **Défis techniques**:
  + L'intégration avec le système de reconnaissance faciale pour le suivi des heures de travail.
  + La génération de rapports nécessite des outils d'analyse fiables et des capacités de visualisation des données.
  + La précision des données enregistrées est essentielle pour des évaluations justes des performances des employés.

**Technologies utilisées**

* Le langage de programmation Python est utilisé pour développer l'application. Il offre une grande flexibilité et un large éventail de bibliothèques utiles pour la gestion des données et la reconnaissance faciale.
* Les algorithmes de reconnaissance faciale peuvent provenir de bibliothèques Python comme OpenCV ou d'autres outils spécialisés.
* La sécurité des données et la protection de la confidentialité sont des aspects critiques à prendre en compte dans toutes les étapes du développement.

**Recommandations finales**

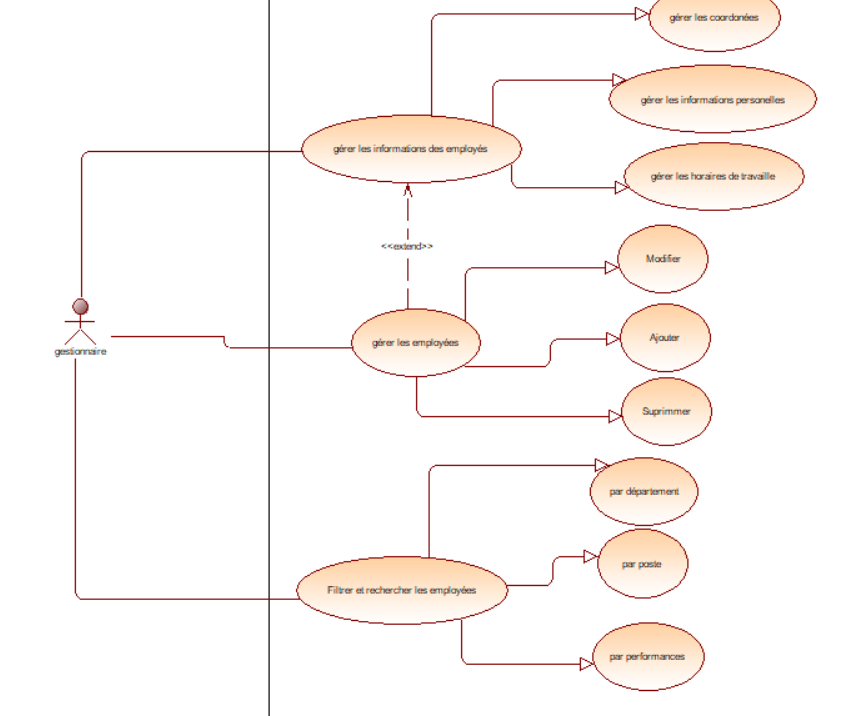
Pour réussir ce projet, il est important de:

* Assurer la sécurité des données et la confidentialité des employés.
* Mettre en place des tests rigoureux pour garantir la précision du système de reconnaissance faciale.
* Fournir des formations aux gestionnaires et aux employés pour l'utilisation efficace de l'application.
* Avoir un système de sauvegarde et de récupération des données pour éviter toute perte de données importantes.
* Rester flexible et ouvert aux ajustements pour répondre aux besoins changeants des utilisateurs finaux.

1. **Diagramme UML**

**1)- Diagramme de cas d’utilisation**

Un diagramme de cas d'utilisation capture le comportement d'un système, d'un sous-système, d'une classe ou d'un composant tel qu'un utilisateur extérieur le voit. Il scinde la fonctionnalité du système en unités cohérentes, les cas d'utilisation, ayant un sens pour les acteurs. Les cas d'utilisation permettent d'exprimer le besoin des utilisateurs d'un système, ils ont donc une vision orientée utilisateur de ce besoin au contraire d'une vision informatique.

Figure1 :

Une image contenant texte, diagramme, capture d’écran, cercle

Description générée automatiquement

Figure 1 :

Pour modéliser ce scénario en utilisant un diagramme de cas d'utilisation, il faut identifier les acteurs principaux, les cas d'utilisation qu'ils réalisent, et les relations entre eux. Voici une généralisation du travail réalisé en utilisant un diagramme de cas d'utilisation :

### Acteurs

* **Gestionnaire** : Cet acteur représente les gestionnaires ou les superviseurs qui gèrent les employés et surveillent les opérations de production.
* **Employé** : Cet acteur représente les employés qui interagissent avec le système, par exemple pour pointer leurs heures de travail.

### Cas d'utilisation

Voici les cas d'utilisation principaux que votre application pourrait inclure :

1. **Gestion des employés**
   * **Ajouter un employé** : Le gestionnaire peut ajouter de nouveaux employés avec des détails comme les coordonnées, les informations personnelles, etc.
   * **Modifier les informations des employés** : Le gestionnaire peut modifier les informations des employés existants.
   * **Supprimer un employé** : Le gestionnaire peut supprimer des employés du système.
   * **Consulter les employés** : Le gestionnaire peut visualiser les informations des employés.
   * **Rechercher des employés** : Le gestionnaire peut rechercher et filtrer les employés par divers critères.
2. **Reconnaissance faciale**
   * **Pointer à l'arrivée** : L'employé utilise le système de reconnaissance faciale pour pointer à son arrivée.
   * **Pointer au départ** : L'employé utilise le système de reconnaissance faciale pour pointer à son départ.
3. **Suivi de la production**
   * **Enregistrer les tâches** : Le système enregistre les tâches effectuées par chaque employé, potentiellement généré par le système de reconnaissance faciale.
   * **Générer des rapports de production** : Les gestionnaires peuvent générer des rapports pour évaluer les performances des employés.

**2)- Diagramme de classe**

Le diagramme de classes est un schéma utilisé en génie logiciel pour présenter les classes et les interfaces des systèmes ainsi que les différentes relations entre celles-ci. Ce diagramme fait partie de la partie statique d'UML car il fait abstraction des aspects temporels et dynamiques.

Figure 2 :

Une image contenant texte, capture d’écran, nombre, thermomètre

Description générée automatiquement

**Conclusion**

Nous avons présenté dans ce chapitre les différents diagrammes élaborés dans la conception du projet et qui nous ont aidés à avoir une vue globale sur tout le projet, ce qui nous facilite l’étape suivante, la réalisation.

**Chapitre 2 :**

**Réalisation du projet :**

Details sur la réalisation du projet en utilisant Python :

import tkinter as tk

from tkinter import messagebox

import sqlite3

import cv2

from PIL import Image, ImageTk

import io

import face\_recognition

import json

### **Tkinter**

* **Description**: Tkinter est la bibliothèque standard de Python pour créer des interfaces graphiques (GUI). Elle permet de créer des fenêtres, des boutons, des menus, des zones de texte, et autres éléments interactifs.
* **Usage dans le projet**: Tkinter est utilisé pour créer l'interface utilisateur de l'application, permettant aux gestionnaires et aux employés d'interagir avec le système.

**Messagebox (de Tkinter)**

* **Description**: Messagebox est un sous-module de Tkinter qui fournit des boîtes de dialogue pour afficher des messages ou demander des confirmations à l'utilisateur.
* **Usage dans le projet**: Messagebox est utilisé pour afficher des messages d'information, des avertissements ou des demandes de confirmation dans l'application.

**SQLite3**

* **Description**: SQLite3 est une bibliothèque de gestion de bases de données relationnelles légère et sans serveur. Elle utilise des fichiers locaux pour stocker les données.
* **Usage dans le projet**: SQLite3 est utilisé pour créer et gérer la base de données qui stocke les informations des employés, les horaires de travail, les rapports, etc.

**OpenCV**

* **Description**: OpenCV est une bibliothèque open source pour la vision par ordinateur. Elle fournit des outils pour le traitement d'images et de vidéos.
* **Usage dans le projet**: OpenCV est utilisé pour travailler avec les images capturées par les caméras et pour traiter des images ou des vidéos, par exemple, dans le contexte de la reconnaissance faciale.

### **Face Recognition**

* **Description**: Face Recognition est une bibliothèque Python construite sur d'autres bibliothèques comme dlib et OpenCV. Elle fournit des outils pour la reconnaissance faciale, y compris la détection et la comparaison des visages.
* **Usage dans le projet**: Face Recognition est utilisé pour la détection et la reconnaissance des visages des employés, permettant l'utilisation de la reconnaissance faciale pour le pointage et le suivi des horaires.

### **JSON**

* **Description**: JSON (JavaScript Object Notation) est un format de données léger utilisé pour échanger des données entre systèmes ou composants logiciels.
* **Usage dans le projet**: JSON peut être utilisé pour stocker ou échanger des données dans un format lisible par l'homme, facilitant la communication entre différents composants de l'application.

**Fonction d’ajout d’un employé :**

* def ajouter\_employe():
* nom = nom\_entry.get()
* prenom = prenom\_entry.get()
* departement = departement\_var.get()
* date\_debut = date\_debut\_entry.get()
* date\_fin = date\_fin\_entry.get()
* poste = poste\_entry.get()
* cap = cv2.VideoCapture(0)
* # Capturer une seule image
* ret, frame = cap.read()
* # Fermer la webcam
* cap.release()
* if ret:
* # Conversion de l'image en format compatible Tkinter
* image= cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)  # Convertir les couleurs
* photo = Image.fromarray(image)  # Créer une image PIL
* photo\_tk = ImageTk.PhotoImage(photo)  # Convertir en format Tkinter
* # Enregistrer l'image en mémoire
* image\_bytes = io.BytesIO()
* photo.save(image\_bytes, format='PNG')  # Sauvegarder l'image au format binaire
* image\_bytes.seek(0)
* face\_encodings = face\_recognition.face\_encodings(image)
* if len(face\_encodings) > 0:
* face\_encoding = face\_encodings[0]
* encoding\_str = json.dumps(list(face\_encoding))
* cursor.execute('''INSERT INTO employes (nom, prenom, departement, date\_debut, date\_fin, poste , face, photo)
* VALUES (?, ?, ?, ?, ?, ?, ?, ?)''', (nom, prenom, departement, date\_debut, date\_fin, poste,encoding\_str,image\_bytes.getvalue()))
* conn.commit()
* messagebox.showinfo("Succès", "Employé ajouté avec succès !")
* afficher\_employes\_listbox()

La fonction **ajouter\_employe()** permet d'ajouter un nouvel employé à une base de données, avec des détails tels que son nom, prénom, département, date de début, date de fin et poste, tout en utilisant la reconnaissance faciale pour capturer et stocker son visage et sa photo. Voici son fonctionnement:

1. **Récupération des Données**: Elle récupère les détails de l'employé à partir de champs d'entrée Tkinter (comme **nom\_entry** et **prenom\_entry**).
2. **Capture d'Image**: Elle utilise **cv2.VideoCapture(0)** pour activer la webcam et capture une seule image (**frame**).
3. **Fermeture de la Webcam**: La webcam est fermée avec **cap.release()** après avoir capturé l'image.
4. **Traitement de l'Image**: L'image est convertie en format RGB et en une image PIL, puis en format compatible avec Tkinter (avec **ImageTk.PhotoImage**).
5. **Encodage du Visage**: La fonction utilise **face\_recognition.face\_encodings(image)** pour obtenir des encodages faciaux à partir de l'image capturée.
6. **Insertion dans la Base de Données**: Si un encodage facial est trouvé, il est converti en chaîne JSON avec **json.dumps** et inséré dans la table des employés avec les autres détails, y compris la photo.
7. **Confirmation et Affichage**: Un message de succès est affiché avec **messagebox.showinfo**, et la liste des employés est mise à jour avec **afficher\_employes\_listbox()**.

Fonction d’affichage des employés dans une listbox :

def afficher\_employes\_listbox():

    # Effacer le contenu précédent de la liste

    employes\_listbox.delete(0, tk.END)

    # Récupération de tous les employés

    cursor.execute("SELECT id,nom,prenom,departement ,date\_debut,date\_fin,poste FROM employes")

    employes = cursor.fetchall()

    # Affichage des employés dans la liste

    for employe in employes:

        id, nom, prenom, departement, date\_debut, date\_fin, poste = employe

        employe\_info = f"{id} | {nom} | {prenom} | {departement} | {date\_debut} | {date\_fin} | {poste}"

        employes\_listbox.insert(tk.END, employe\_info)

La fonction **afficher\_employes\_listbox()** met à jour une ListBox Tkinter avec les employés extraits d'une base de données SQLite. Voici une version simplifiée :

1. **Effacement de la Liste**:
   * Efface la ListBox **employes\_listbox** avec **delete(0, tk.END)**, pour la rafraîchir.
2. **Récupération des Employés**:
   * Exécute une requête SQL pour obtenir tous les employés avec **cursor.execute("SELECT id, nom, prenom, departement, date\_debut, date\_fin, poste FROM employes")**.
   * Stocke les résultats avec **employes = cursor.fetchall()**.
3. **Affichage dans la ListBox**:
   * Pour chaque employé récupéré, extrait ses détails (**id, nom, prenom, etc.**) et les ajoute à la ListBox avec un format spécifique. Un exemple d'ajout pourrait être **employes\_listbox.insert(tk.END, f"{nom} {prenom} - {poste}")**.

Fonction des reconnaissances facial :

def detecter\_visages(image, cascade):

    # Convertir l'image en niveaux de gris

    gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

    name="test"

    frame\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

    face\_encodings = face\_recognition.face\_encodings(frame\_rgb )

    if len(face\_encodings) > 0:

            face\_encoding = face\_encodings[0]

            encoding\_str = json.dumps(list(face\_encoding))

            # Rechercher des correspondances dans la base de données

            cursor.execute("SELECT face, nom FROM employes")

            rows = cursor.fetchall()

# Convertir les encodages en listes de nombres flottants

            database\_encodings = [json.loads(row[0]) for row in rows]

            database\_names = [row[1] for row in rows]

            matches = face\_recognition.compare\_faces(database\_encodings, face\_encoding, tolerance=0.6)

            if True in matches:

              match\_index = matches.index(True)

              name = database\_names[match\_index]

            else:

              name = "Inconnu"

    # Détecter les visages dans l'image

    name = str(name)

    visages = cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    # Dessiner des rectangles autour des visages détectés

    for (x, y, w, h) in visages:

        # Dessiner un rectangle autour du visage

        cv2.rectangle(image, (x, y), (x+w, y+h), (255, 0, 0), 2)

        # Générer un encodage facial

        # Ajouter une étiquette avec le nom attribué au visage (remplacer 'NomVisage' par le nom réel)

        cv2.putText(image, name, (x, y-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.9, (255, 0, 0), 2)

    return image

def demarrer\_reconnaissance\_faciale():

    cascade\_path = cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml'

    cascade = cv2.CascadeClassifier(cascade\_path)

    # Démarrer la capture vidéo à partir de la webcam

    cap = cv2.VideoCapture(0)

    # Vérifier si la capture vidéo est ouverte

    if not cap.isOpened():

        print("Erreur: la webcam ne peut pas être ouverte.")

        return

    while True:

        # Lire l'image de la webcam

        ret, frame = cap.read()

        # Vérifier si la lecture de l'image a réussi

        if not ret:

            print("Erreur: Impossible de lire l'image de la webcam.")

            break

        # Détecter les visages dans l'image et afficher le nom attribué à chaque visage

        frame= cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

        frame = detecter\_visages(frame, cascade)

        # Afficher l'image traitée

        cv2.imshow('Reconnaissance faciale', frame)

        key = cv2.waitKey(30)

        if key == 27:

            break

    cap.release()

    cv2.destroyAllWindows()

Ces deux fonctions, **detecter\_visages** et **demarrer\_reconnaissance\_faciale**, sont conçues pour implémenter un système de reconnaissance faciale utilisant OpenCV et la bibliothèque de reconnaissance faciale. Voici une explication de chacune de ces fonctions :

### detecter\_visages

* **Description**: Cette fonction prend une image et un cascade Haar comme paramètres et détecte les visages dans l'image. Elle cherche également des correspondances dans une base de données pour identifier les visages.
* **Conversion en Niveaux de Gris**:
  + L'image est convertie en niveaux de gris avec **cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)**. Cette conversion facilite la détection de visages avec un cascade Haar.
* **Reconnaissance Faciale**:
  + L'image est également convertie en RGB pour la reconnaissance faciale avec **cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)**.
  + Les encodages faciaux sont générés avec **face\_recognition.face\_encodings(frame\_rgb)**.
  + Si des encodages sont trouvés, la fonction compare ces encodages avec ceux de la base de données, récupérés avec **cursor.execute("SELECT face, nom FROM employes")**. Les correspondances sont vérifiées avec **face\_recognition.compare\_faces**, avec un seuil de tolérance de 0.6.
* **Détection de Visages**:
  + Les visages sont détectés dans l'image avec **cascade.detectMultiScale**, qui utilise des paramètres tels que **scaleFactor**, **minNeighbors**, et **minSize**.
  + La fonction dessine des rectangles autour des visages détectés avec **cv2.rectangle(image, (x, y), (x+w, y+h), (255, 0, 0), 2)**.
  + Elle ajoute également le nom identifié (ou "Inconnu") au-dessus des visages avec **cv2.putText**.
* **Retourne l'Image Traitée**:
  + La fonction retourne l'image traitée, avec les rectangles et les noms indiqués.

### demarrer\_reconnaissance\_faciale

* **Description**: Cette fonction démarre la capture vidéo à partir de la webcam, détecte les visages dans le flux vidéo, et les affiche avec le nom attribué, si disponible.
* **Chargement du Cascade Haar**:
  + Le chemin du cascade Haar utilisé pour la détection de visages est obtenu avec **cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml'**.
  + Le cascade est chargé avec **cv2.CascadeClassifier(cascade\_path)**.
* **Démarrage de la Webcam**:
  + La capture vidéo démarre avec **cv2.VideoCapture(0)**.
  + La fonction vérifie si la webcam s'est ouverte correctement. Si ce n'est pas le cas, un message d'erreur est affiché et la fonction s'arrête.
* **Boucle de Capture Vidéo**:
  + Tant que la boucle tourne, l'image est capturée avec **cap.read()**.
  + Si la capture échoue, un message d'erreur est affiché et la boucle s'arrête.
  + L'image est convertie en RGB et passée à la fonction **detecter\_visages**.
* **Affichage de l'Image**:
  + L'image traitée est affichée dans une fenêtre avec **cv2.imshow('Reconnaissance faciale', frame)**.
  + La boucle s'arrête si la touche "Échap" (27) est pressée.
* **Fermeture de la Webcam et des Fenêtres**:
  + La webcam est relâchée avec **cap.release()**, et toutes les fenêtres OpenCV sont fermées avec **cv2.destroyAllWindows()**.

cursor.execute('''CREATE TABLE IF NOT EXISTS horaires (

                    id INTEGER PRIMARY KEY,

                    employe\_id INTEGER,

                    temps\_arrivee TEXT,

                    temps\_depart TEXT,

                    heures\_travail REAL, -- Ajout de la nouvelle colonne pour stocker les heures de travail

                    FOREIGN KEY (employe\_id) REFERENCES employes(id))''')

conn.commit()

Le code crée une table appelée "horaires" dans une base de données, avec plusieurs colonnes:

* **id** est la clé primaire, utilisée pour identifier chaque enregistrement de manière unique.
* **employe\_id** est une référence à un employé, connectée à une autre table ("employes").
* **temps\_arrivee** et **temps\_depart** stockent les heures d'arrivée et de départ d'un employé.
* **heures\_travail** enregistre le temps total travaillé.

Le code utilise **FOREIGN KEY** pour établir une relation entre les tables "horaires" et "employes", assurant que **employe\_id** doit correspondre à un identifiant dans la table "employes". Le code assure également que la table n'est créée que si elle n'existe pas déjà.

def pointer\_arriver():

    try:

        # Ouvrir la caméra

        cap = cv2.VideoCapture(0)

        ret, frame = cap.read()

        cap.release()

        if not ret:

            messagebox.showerror("Erreur", "Impossible de capturer l'image de la webcam.")

            return

        # Conversion pour face\_recognition

        frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

        face\_encodings = face\_recognition.face\_encodings(frame\_rgb)

        if not face\_encodings:

            messagebox.showerror("Erreur", "Aucun visage détecté.")

            return

        # Récupérer les encodages de la base de données

        face\_encoding = face\_encodings[0]

        cursor.execute("SELECT id, face FROM employes")

        rows = cursor.fetchall()

        # Comparer avec ceux de la base de données

        for row in rows:

            employe\_id = row[0]

            employe\_face = json.loads(row[1])

            match = face\_recognition.compare\_faces([employe\_face], face\_encoding, tolerance=0.6)

            if match[0]:

                heure\_arrivee = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

                cursor.execute("INSERT INTO horaires (employe\_id, temps\_arrivee) VALUES (?, ?)",

                               (employe\_id, heure\_arrivee))

                conn.commit()  # Engager la transaction

                messagebox.showinfo("Succès", f"L'employé {employe\_id} a pointé à l'heure {heure\_arrivee}.")

                calculer\_heures\_travail()  # Calculer les heures de travail après chaque pointage d'arrivée

                return

        messagebox.showerror("Erreur", "Aucun employé correspondant trouvé.")

    except sqlite3.OperationalError as e:

        messagebox.showerror("Erreur de base de données", f"Erreur opérationnelle : {str(e)}")

    except Exception as ex:

        messagebox.showerror("Erreur inattendue", f"Une erreur inattendue s'est produite : {str(ex)}")

def calculer\_heures\_travail():

    try:

        cursor.execute("SELECT id, temps\_arrivee, temps\_depart FROM horaires WHERE temps\_arrivee IS NOT NULL AND temps\_depart IS NOT NULL")

        rows = cursor.fetchall()

        for row in rows:

            employe\_id, temps\_arrivee, temps\_depart = row

            temps\_arrivee = datetime.datetime.strptime(temps\_arrivee, "%Y-%m-%d %H:%M:%S")

            temps\_depart = datetime.datetime.strptime(temps\_depart, "%Y-%m-%d %H:%M:%S")

            difference = temps\_depart - temps\_arrivee

            heures\_travail = difference.total\_seconds()  # Convertir en heures

            # Mise à jour de la base de données avec les heures de travail calculées

            cursor.execute("UPDATE horaires SET heures\_travail = ? WHERE id = ?", (heures\_travail, employe\_id))

            conn.commit()

    except sqlite3.OperationalError as e:

        messagebox.showerror("Erreur de base de données", f"Erreur opérationnelle : {str(e)}")

    except Exception as ex:

        messagebox.showerror("Erreur inattendue", f"Une erreur inattendue s'est produite : {str(ex)}")

# Fonction pour pointer le départ

def pointer\_depart():

    try:

        # Ouvrir la caméra

        cap = cv2.VideoCapture(0)

        ret, frame = cap.read()

        cap.release()

        if not ret:

            messagebox.showerror("Erreur", "Impossible de capturer l'image de la webcam.")

            return

        # Conversion pour face\_recognition

        frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

        face\_encodings = face\_recognition.face\_encodings(frame\_rgb)

        if not face\_encodings:

            messagebox.showerror("Erreur", "Aucun visage détecté.")

            return

        # Récupérer les encodages de la base de données

        face\_encoding = face\_encodings[0]

        cursor.execute("SELECT id, face FROM employes")

        rows = cursor.fetchall()

        # Comparer avec ceux de la base de données

        for row in rows:

            employe\_id = row[0]

            employe\_face = json.loads(row[1])

            match = face\_recognition.compare\_faces([employe\_face], face\_encoding, tolerance=0.6)

            if match[0]:

                heure\_depart = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

                # Vérifier si l'employé a déjà pointé l'arrivée

                cursor.execute("SELECT id FROM horaires WHERE employe\_id = ? AND temps\_arrivee IS NOT NULL AND temps\_depart IS NULL",

                               (employe\_id,))

                resultat = cursor.fetchone()

                if resultat:

                    # Ajouter le temps de départ

                    cursor.execute("UPDATE horaires SET temps\_depart = ? WHERE id = ?",

                                   (heure\_depart, resultat[0]))

                    conn.commit()  # Engager la transaction

                    messagebox.showinfo("Succès", f"L'employé {employe\_id} a pointé à l'heure {heure\_depart} pour le départ.")

                    calculer\_heures\_travail()  # Calculer les heures de travail après chaque pointage de départ

                    return

                else:

                    messagebox.showerror("Erreur", "L'employé n'a pas pointé l'arrivée aujourd'hui.")

                    return

        messagebox.showerror("Erreur", "Aucun employé correspondant trouvé.")

    except sqlite3.OperationalError as e:

        messagebox.showerror("Erreur de base de données", f"Erreur opérationnelle : {str(e)}")

    except Exception as ex:

        messagebox.showerror("Erreur inattendue", f"Une erreur inattendue s'est produite : {str(ex)}")

Ce code Python montre comment enregistrer les horaires d'arrivée et de départ des employés à l'aide de la reconnaissance faciale. Il contient deux fonctions principales: **pointer\_arriver()** et **pointer\_depart()**, qui enregistrent le pointage des employés à l'arrivée et au départ, respectivement. Voici ce que fait chaque fonction :

* **pointer\_arriver()** :
  + Ouvre la webcam pour capturer une image du visage de l'employé.
  + Si la webcam ne fonctionne pas, un message d'erreur s'affiche.
  + Utilise la reconnaissance faciale pour détecter et encoder le visage dans l'image capturée.
  + Compare cet encodage avec ceux des employés enregistrés dans la base de données.
  + Si une correspondance est trouvée, enregistre l'heure d'arrivée de l'employé dans la base de données.
  + Sinon, affiche un message d'erreur si aucun visage n'est détecté ou si aucun employé correspondant n'est trouvé.
  + Appelle **calculer\_heures\_travail()** pour mettre à jour les heures de travail.
* **pointer\_depart()** :
  + Similaire à **pointer\_arriver()**, mais pour enregistrer l'heure de départ.
  + Vérifie d'abord si l'employé a déjà pointé l'arrivée.
  + Si oui, enregistre l'heure de départ dans la base de données.
  + Sinon, affiche un message d'erreur si l'employé n'a pas pointé l'arrivée.
  + Appelle également **calculer\_heures\_travail()** après avoir enregistré le départ.
* **calculer\_heures\_travail()** :
  + Récupère les enregistrements d'arrivée et de départ pour chaque employé.
  + Calcule le temps de travail total entre l'heure d'arrivée et l'heure de départ.
  + Met à jour ces valeurs dans la base de données.

L'ensemble du code utilise des blocs **try-except** pour gérer les exceptions et afficher des messages d'erreur en cas de problème avec la base de données ou d'autres erreurs inattendues.

**Conclusion Générale**

Notre projet de fin d’année porte sur une application web et mobile intitulé « Resto 707» permet la gestion d’un restaurant.

Les étapes que nous avons suivie pour atteindre nos objectifs consiste à faire une étude préalable qui nous a permis de collecter les informations, après nous avons modélisé toutes les fonctionnalités nécessaire pour réaliser notre application. En se basant sur UML, nous avons modélisé les diagrammes (classe, et cas d’utilisation). Apres on a passé a la phase de codage.

Réalisé par : Ghayt El idrissi Dafali.

Reda Bouimakliouine.