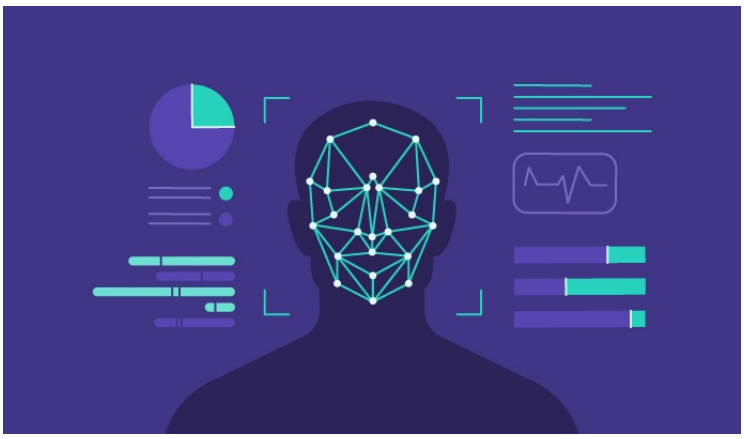
ENSIAS

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat

Projet de Fin d'Année

Face Recognition and Security System



Réalisé par :

Yahya BELGHITI-ALAOUI
Oussama ETTAOUIL

Encadré par :

M. Abdellatif ELFAKER

Année académique 2022/2023

Face Recognition and Security System

Table des matières

Table des figures

Remerciements :

Nous tenons tout d'abord à exprimer nos profonds remerciements à notre professeur, M. Abdellatif EL FAKER, de nous avoir encadré tout au long de la réalisation de ce projet. Nous tenons aussi à remercier tous ceux qui nous ont aidé, de près ou de loin, à réaliser ce travail.

Nos remerciements vont également aux membres du jury qui ont bien accepté d'évaluer notre travail, ainsi qu'a tout le corps professoral et administratif de l'école qui a toujours veillé à ce que notre formation soit dispensée dans de bonnes conditions.

Résumé :

Ce rapport représente notre projet de fin d'année qui a pour but d'approfondir et de mettre en pratique nos connaissances en développement informatique. Ce projet nous a également permis de découvrir le monde de la vision par ordinateur (*computer vision*) et de la reconnaissance faciale. Durant sa réalisation, nous avons dû, en outre, apprendre à programmer des interfaces graphiques pour faciliter à l'utilisateur la manipulation du système. En plus, nous avons réalisé un système client-serveur pour permettre la transmission de données entre deux ordinateurs.

Pour se faire, il a fallu commencer, en premier lieu, par comprendre la problématique posée, les besoins et les objectifs à atteindre. En deuxième lieu, nous avons étudié l'architecture du système et ses différentes composantes. Ensuite, nous avons étudié le fonctionnement d'un algorithme de reconnaissance faciale.

Par la suite, nous avons abordé la réalisation de l'application. Pendant cette phase, nous avons commencé par programmer l'algorithme de reconnaissance faciale. Quand ce dernier était complètement opérationnel, nous nous sommes focalisé sur la réalisation des interfaces graphiques. Enfin, nous avons écrit le code permettant d'assurer une connexion client-serveur entre deux ordinateurs. Après la fin de chacune des étapes de développement, nous avons réalisé des tests afin de nous assurer du bon fonctionnement des différentes fonctionnalités implémentées et, également, pour garantir le bon déroulement du projet.

Ce projet a été fait en langage de programmation Python. Nous avons, de plus, utilisé plusieurs frameworks de ce langage de programmation. Des fichiers Pickle ont fait office de base de données pour cette application.

Abstract :

This report represents our end-of-year project. The purpose of this work was to deepen and put into practice our knowledge of software development. This project also allowed us to discover the world of computer vision and face recognition. During the making of this project, we also had to learn to program graphical interfaces to make it easier for the user to manipulate the system. In addition, we made a client-server system to allow the transmission of data between two computers.

To do this, the first thing we had to do was to understand the problem, the needs and the objectives to be attained. Secondly, we studied the architecture of the system and its different components. Then, we studied the functioning of a face recognition algorithm.

Afterward, we got into the realization of the app. During this phase, we started by programming the face recognition algorithm. When it was fully operational, we focused on creating the graphical user interfaces. Finally, we wrote the code to provide a client-server connection between two computers. After the end of each of the development stages, we did tests to ensure the proper functioning of the different implemented functionalities and, also, to guarantee the smooth running of the project.

This project was made in Python in addition to some of its frameworks. Moreover, Pickle files served as databases for this app.

Introduction Générale

De nos jours, nous nous trouvons confrontés, de plus en plus, à des problèmes de sécurité. Les systèmes de sécurité «traditionnels», tels que les serrures, peuvent facilement être contournés. Pour résoudre ce problème, il existe plusieurs technologies qu'il est possible d'utiliser, telles que la reconnaissance faciale. En effet, cette technologie suscite plus que jamais l'intérêt des entreprises et des particuliers, car elle permet d'identifier ou d'authentifier des personnes à partir d'images. Ainsi, la reconnaissance faciale est facile à utiliser. Elle est, également, relativement sécurisée.

La reconnaissance faciale, dans les systèmes de sécurité, est utilisée dans des systèmes divers et à usages différents. À titre d'exemple, elle peut être utilisée pour gérer l'accés à un bâtiment. Les personnes souhaitant entrer dans ledit bâtiment doivent se mettre devant une caméra sur la porte afin que l'algorithme de reconnaissance faciale puisse identifier leur visage et enclencher le déverrouillage automatique de la porte. Il est possible d'ajouter d'autres couches de sécurité, par exemple un mot de passe, si cela s'avère nécessaire.

Le système étudié, tout au long de ce rapport, est un logiciel de contrôle d'accès à un immeuble à l'aide de la reconnaissance faciale.

Ce rapport est structuré en trois chapitres. Dans le premier chapitre, nous présentons, tout d'abord, l'histoire de la reconnaissance faciale et la relation entre la reconnaissance faciale et la sécurité. Puis, nous définissons la problématique et nous faisons une présentation du projet et de ses objectifs. Le deuxième chapitre présente, en premier lieu, le concept de machine learning et ses différents algorithmes, suivis de leur application dans la reconnaissance faciale. Ensuite, il y a une analyse fonctionnelle du projet qui comporte les besoins techniques, les exigences fonctionnelles et non fonctionnelles et les acteurs du système. Nous y présentons, aussi, le diagramme des cas d'utilisation, l'architecture générale du système et les maquettes de l'application réalisée. Enfin, le troisième chapitre est consacré à la réalisation. Il présente l'architecture technique de l'application, énumère les outils utilisés et met en lumière les différentes fonctionnalités implémentées et leur fonctionnement.

Chapitre 1

Contexte général du projet

1.1 Introduction

Dans ce premier chapitre du rapport, nous allons mettre en relief le contexte général du projet, en mettant en avant le lien entre la reconnaissance faciale et la sécurité. Ce chapitre présente également le projet, dans son ensemble. Lequel projet vise à développer un logiciel de reconnaissance faciale pour contrôler l'accès à un bâtiment.

1.2 Histoire de la technologie de reconnaissance faciale

La reconnaissance faciale automatisée a été introduite, pour la première fois, pendant les années 1960, par les chercheurs Woody Bledsoe, Helen Chan Wolf et Charles Bisson. Leur premier projet exigeait de l'utilisateur de pointer sur les coordonnées des caractéristiques du visage tels que les centres des pupilles, les coins intérieurs et extérieurs des yeux et les lèvres supérieure et inférieure. Ces coordonnées sont utilisées pour calculer 20 distances individuelles, comme la largeur de la bouche et des yeux. Par la suite, un ordinateur compare automatiquement les distances pour chaque photographie, calcule la différence entre les distances et renvoie les enregistrements les plus proches comme une correspondance possible.

En 1970, l'informaticien Japonais, Takeo Kanade, a réalisé un système de correspondance des visages qui situait leurs caractéristiques et calculait le rapport de distance entre les traits du visage de façon automatique.

En 1993, la Defense Advanced Research Projects Agency (DARPA) et le Army Research Laboratory (ARL) ont créé le programme de technologie de reconnaissance faciale FERET (Facial Recognition Technology) pour développer davantage cette technologie. Ce programme avait pour objectif de concevoir des algorithmes de reconnaissance faciale automatiques pour les services de renseignements et de sécurité, et pour aider les forces de l'ordre à exercer leurs fonctions. Les algorithmes de reconnaissance des visages testés dans les laboratoires de recherche, au cours du programme FERET, ont été, par la suite, utilisés pour des usages multiples et dans des contextes différents.

Depuis l'avénement de la reconnaissance faciale et jusqu'au début des années 1990, des portraits d'individus étaient utilisés pour la détetion des visages. À partir du début des années 1990, plusieurs algorithmes, permettant de détecter et de localiser les visages dans les images contenant d'autres objets ont commencé à voir le jour. Parmi les plus notables, on retrouve la méthode de Viola et Jones pour la détection d'objets et les algorithmes d'identification des visages sur de longues distances (Human identification at a distance).[8]

1.3 La reconnaissance faciale et la sécurité

La reconnaissance faciale est une technologie qui permet d'identifier un individu à partir de son visage. Cette technique se base sur les traits faciaux de la personne afin de la reconnaître. La reconnaissance faciale est très utilisée en sécurité. Par exemple, pour contrôler les entrées et sorties aux zones à accès restreint, il est possible d'utiliser la reconnaissance des visages. De plus, la reconnaissance faciale peut aussi être utilisée pour la surveillance des lieux publics tels que les parcs, les aéroports et les gares. Cela permet, entres autres, de contribuer à la prévention de la criminalité en identifiant les individus impliqués dans des activités illégales. En outre, la reconnaissance faciale dans les les lieux publics peut aider à retrouver les personnes portées disparues.

1.4 Problématique et présentation du projet

1.4.1 Problématique

Comme nous venons de le voir dans la section précédente, la reconnaissance faciale et la sécurité sont intimement liées. En effet, la reconnaissance des visages peut être utilisée comme outil de sécurité dans des domaines différents. Alors comment est-il possible d'utiliser la reconnaissance faciale de manière optimale et efficace dans les applications de sécurité, tout en minimisant les risques de fausses identifications et les tentatives de contournement ou de piratage ? Quels sont les différents algorithmes de reconnaissance faciale qui existent ? Finalement, comment peut-on utiliser la reconnaissance faciale pour gérer l'accès à un immeuble ?

1.4.2 Présentation du projet

Ce projet consiste à réaliser un logiciel de reconnaissance faciale pour contrôler l'accès à un immeuble. Les personnes pouvant entrer, sans autorisation du propriétaire, dans le local ou bâtiment, que nous voulons sécuriser, doivent être reconnues par un système de reconnaissance des visages. Puis, elles devrons taper un mot de passe. Si le mot de passe est correct, la porte du bâtiment se déverrouillera automatiquement. Sinon, la porte reste verrouillée et une alarme est déclenchée. Si la personne est non reconnue par le logiciel de reconnaissance faciale, le propriétaire est informé de la présence d'un inconnu à l'extérieur du bâtiment.

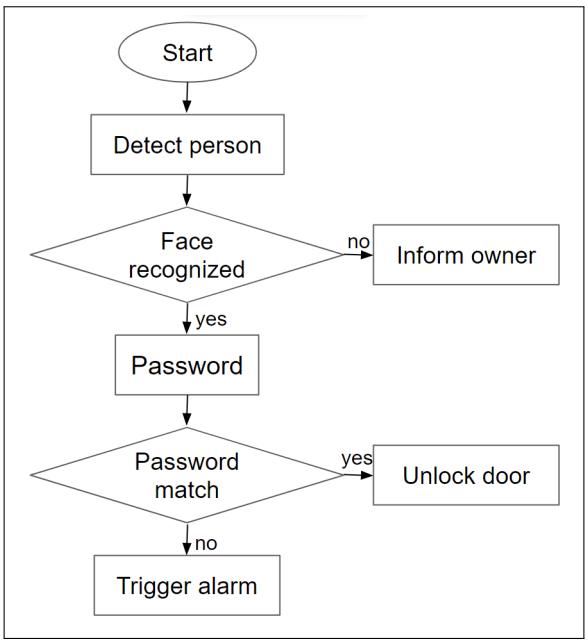


FIGURE 1.1 – Simulation d'une porte de sécurité avec caméra

1.5 Objectifs du projet

Ce projet à pour objectifs de développer un algorithme de reconnaissance faciale par Machine Learning capable d'identifier les personnes autorisées à entrer dans un bâtiment et de mettre en place un système de vérification du mot de passe pour renforcer la sécurité de l'accès. Un autre objectif du projet est d'implémenter un système d'alerte qui informe le propriétaire de la présence d'une personne non reconnue à l'extérieur du bâtiment, et qui déclenche une alarme lorsqu'un utilisateur se trompe de mot de passe. Le dernier objectif du projet est de réaliser des interfaces graphiques pour faciliter l'utilisation du logiciel.

1.6 Conclusion

Ce chapitre a permis de mettre en évidence le contexte général du projet en mettant, tout d'abord, l'accent sur l'importance de la reconnaissance faciale dans le domaine de la sécurité. Nous avons vu que la reconnaissance faciale peut être utilisée pour contrôler l'accès à un immeuble et contribuer à la prévention des activités illégales. Cependant, pour assurer son bon fonctionnement, il est essentiel de résoudre les problématiques liées à la fiabilité et à la prévention des erreurs d'identification, ainsi qu'à la protection contre les tentatives de contournement ou de piratage. Nous avons, également, exposé les objectifs de ce projet qui sont de développer un algorithme de reconnaissance faciale afin de renforcer la sécurité d'un immeuble, d'intégrer une deuxième couche de sécurité (mot de passe), d'automatiser l'accès, de détecter les intrusions, et de faciliter la manipulation du système.

Chapitre 2

Analyse et conception

2.1 Introduction

Dans ce deuxième chapitre, nous commencerons, en premier lieu, par expliquer le concept de machine learning ainsi que ses différents algorithmes. Ensuite, nous présenterons la théorie derrière la reconnaissance faciale. Après cela, nous exposerons l'analyse fonctionnelle du système, suivie du diagramme de cas d'utilisation. Puis, nous expliquerons l'architecture du système et nous montrerons les maquettes de l'application.

2.2 État de l'art

2.2.1 Machine learning et ses applications

Machine learning (apprentissage automatique) est le processus qui permet à une machine d'apprendre, de manière autonome, à partir des données qu'elle reçoit, sans l'intervention des humains.
Il existe différents types d'algorithmes d'apprentissage automatique qui sont utilisés pour des tâches d'apprentissages supervisé et non supervisé. Voici quelques exemples :

2.2.1.1 Les algorithmes d'apprentissage supervisé

Définition : Les algorithmes d'apprentissage supervisé sont des méthodes utilisées en intelligence artificielle. Ces algorithmes servent à prédire des étiquettes ou des valeurs de sortie à partir de données d'apprentissage étiquetées.

2.2.1.1.1 Régression linéaire : Une technique d'analyse de données utilisé pour modéliser une relation linéaire entre une variable cible continue et des variables prédictives. Elle prédit la valeur de données inconnues en utilisant une autre valeur de données apparentée et connue.

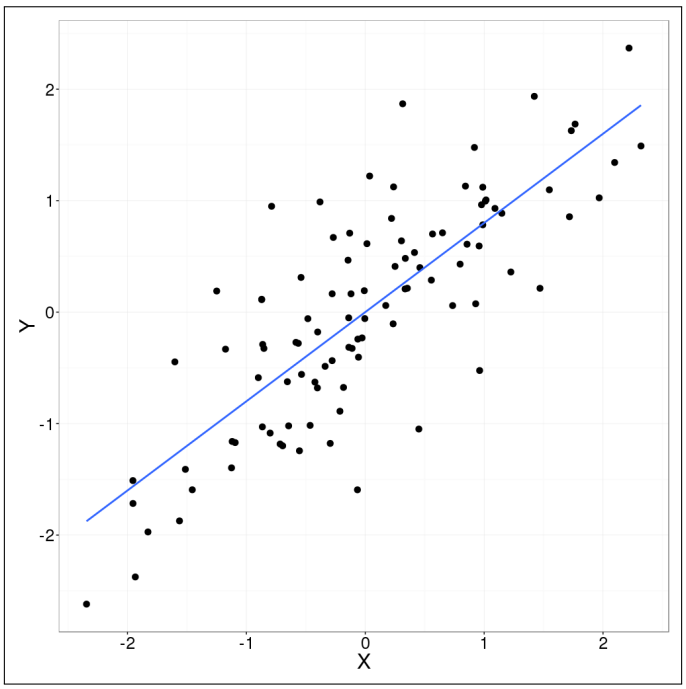


FIGURE 2.1 – Droite de régression linéaire

2.2.1.1.2 Régression logistique : La régression logistique fait partie des modèles mathématiques très souvent utilisés en Machine Learning. Elle est similaire à la régression linéaire, sauf qu'au lieu d'un résultat graphique, la variable cible est binaire, la valeur étant soit 1, soit 0.

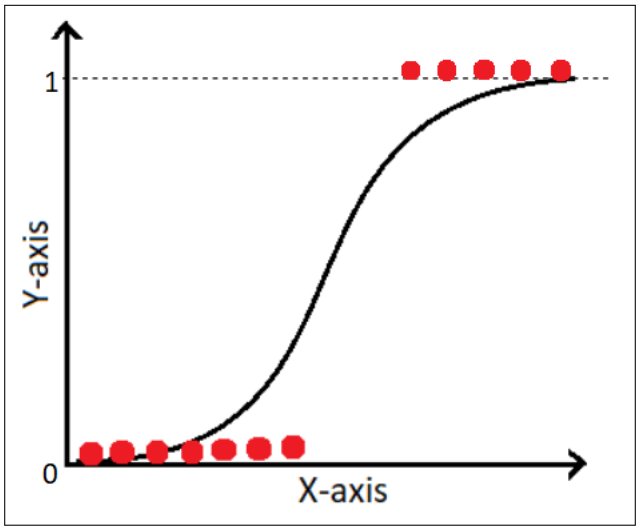


FIGURE 2.2 – Courbe de régression logistique

2.2.1.1.3 Arbre de décision : Un arbre de décision est une méthode de machine learning utilisée en classification et en régression. Cet algorithme utilise une structure arborescente pour prendre des décisions basées sur certaines caractéristiques et prédire les étiquettes de classe ou les valeurs de sortie.

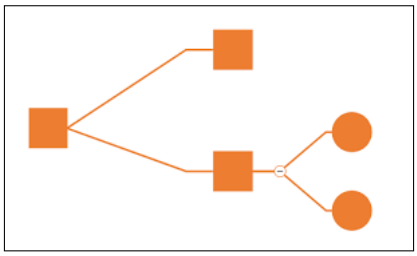


FIGURE 2.3 – Exemple d'arbre de décision

2.2.1.1.4 Support Vector Machine (SVM) : Les *support vector machines*, ou machines à vecteurs de support, sont des algorithmes d'apprentissage supervisé utilisés pour la classification et la régression.[7]

2.2.1.2 Les algorithmes d'apprentissage non supervisé

Définition : Les algorithmes d'apprentissage non supervisé sont des techniques qui permettent de découvrir des structures ou des modèles cachés dans des données non étiquetées, sans connaître les étiquettes ou les valeurs de sortie associées.

2.2.1.2.1 K-means : *K-means* est un algorithme de *clustering* (regroupement) non supervisé qui partitionne un ensemble de données en un certain nombre de *clusters* (groupes) en minimisant la variance intra-cluster et maximisant la variance inter-cluster, en assignant itérativement les points de données aux clusters et en recalculant le centre de chaque cluster.

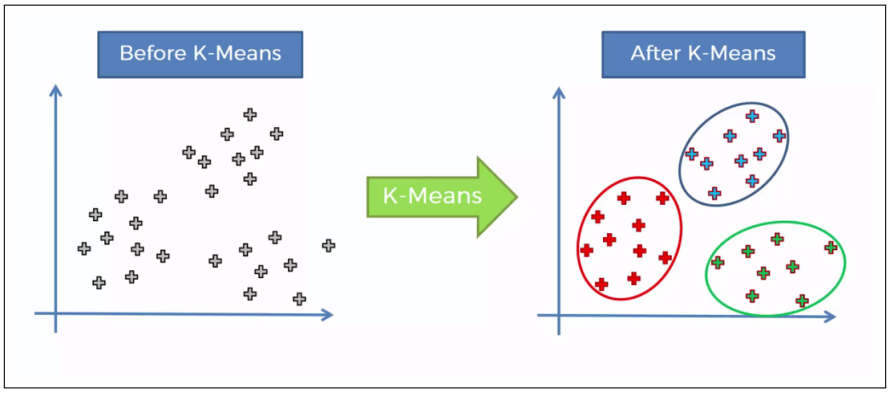


FIGURE 2.4 – K-means

2.2.1.2.2 Analyse en composantes principales (ACP) :

L'Analyse en Composantes Principales (ACP) est une technique d'analyse statistique utilisée en apprentissage non supervisé pour réduire la dimensionnalité d'un ensemble de données tout en préservant au mieux l'information contenue dans les variables d'origine. Elle permet de transformer les variables initiales en un nouvel espace de variables appelées composantes principales qui sont des combinaisons linéaires des variables d'origine. L'objectif de l'ACP est de réduire la complexité des données en identifiant les axes principaux de variation et en projetant les données sur ces axes, facilitant ainsi l'interprétation et l'analyse des données.

2.2.1.3 Réseaux de neurones :

Les réseaux de neurones sont des modèles d'apprentissage automatique inspirés du fonctionnement du cerveau humain. Ils sont constitués de couches de neurones artificiels interconnectés, où chaque neurone effectue des calculs sur les données d'entrée et passe les résultats aux neurones de la couche suivante. Grâce à des processus d'apprentissage itératifs, les réseaux de neurones peuvent ajuster les poids des connexions entre les neurones pour améliorer leurs performances de prédiction ou de classification.

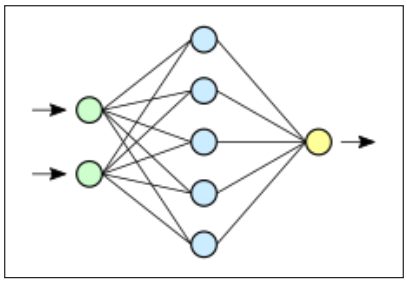


FIGURE 2.5 – Exemple de réseaux de neurones

2.2.1.4 Convolutional Neural Network (CNN)

Un *convolutional neural network*, ou réseau de neurones convolutifs, est un réseau de neurones dont l'architecture est inspirée du cortex visuel des animaux. Ils sont utilisés dans le domaine de la vision par ordinateur.[10]

2.2.2 La reconnaissance faciale : Aspects théoriques

2.2.2.1 Détection du visage

La première étape consiste à localiser et à extraire les régions du visage dans une image ou une vidéo. On utilise pour la détection faciale des algorithmes tels que le détecteur de Viola-Jones ou les réseaux de neurones convolutifs (CNN). Ces algorithmes analysent les caractéristiques visuelles pour détecter les visages présents dans une image.

2.2.2.2 Estimation des Landmarks

Les Landmarks sont des points clés du visage, tels que les coins des yeux, les coins de la bouche, le nez, etc. Leur estimation se fait une fois le visage détecté pour capturer les caractéristiques spécifiques du visage. L'estimation des Landmarks peut être réalisée par les modèles de forme ou les réseaux de neurones.

2.2.2.3 Extraction des caractéristiques

Une fois que les landmarks sont estimés, des caractéristiques sont extraites à partir de ces points pour représenter le visage de manière précise. Ces caractéristiques peuvent être la forme du visage, la distance entre les yeux ou la largeur du nez. Elles sont utilisées pour créer une empreinte faciale unique pour chaque individu.

2.2.2.4 Entraînement du Data Set

Les empreintes faciales sont stockées dans une base de données, qui est utilisée pour entraîner le modèle de reconnaissance faciale. Ce modèle utilise des algorithmes d'apprentissage profond (deep Learning) pour comparer l'empreinte faciale stockée avec l'image en direct.

2.2.2.5 Comparaison des visages

Une fois qu'un modèle est entraîné, il peut être utilisé pour comparer les visages. Lorsqu'une nouvelle image est capturée, le logiciel de reconnaissance faciale compare les caractéristiques de cette image avec celles stockées dans la base de données. Si une correspondance est trouvée, l'identité de la personne est confirmée. Pour effectuer cette comparaison, différentes méthodes peuvent être utilisées, par exemple les réseaux de neurones.

2.3 Analyse fonctionnelle

2.3.1 Besoins techniques

Pour réaliser ce système, nous avons, tout d'abord, besoin d'une caméra, d'une base de donnée contenant l'ensemble des visages des individus reconnus par le propriétaire et d'un algorithme de reconnaissance faciale qui compare le visage détecté avec les visages présents dans la base de donnée. De plus, il faut un programme qui demande à l'utilisateur de saisir son mot de passe, si son visage est reconnu, et qui informe le propriétaire de la présence d'un inconnu à l'extérieur dans le cas où il n'est pas reconnu.

2.3.2 Exigences fonctionnelles

Pour garantir le fonctionnement du système, il faut, en premier lieu, que le logiciel soit capable de détecter les visages des personnes qui se présentent devant la caméra de l'immeuble, et de les comparer avec ceux présents dans la base de données, afin de différencier les personnes autorisées à entrer dans le bâtiment avec celles non autorisées. Le système doit demander à l'utilisateur, après l'avoir reconnu, d'entrer le mot de passe. Le système doit être capable de vérifier la validité du mot de passe. Si ce dernier est correct, la porte doit se

déverrouiller automatiquement. Sinon, le système doit maintenir la porte fermée et déclencher une alarme. Si le visage détecté n'est pas reconnu comme étant autorisé, l'application doit informer le propriétaire de la présence d'un inconnu à l'extérieur du bâtiment. Par ailleurs, la base de données du logiciel doit permettre l'ajout de nouveaux utilisateurs si cela s'avère nécessaire. Finalement, il est capital que les deux couches de sécurité dont dispose le système, à savoir, la reconnaissance faciale et le mot de passe, soient complètement opérationnelles.

2.3.3 Exigences non fonctionnelles

Pour que le système fontionne bien, il faut, en premier lieu, qu'il puisse détecter les visages avec une grande précision. Par ailleurs, les changements d'éclairage et d'angles de vue ne doivent pas dégrader les performances de l'algorithme de reconnaissance des visages. Ensuite, il faut que le programme de reconnaissance faciale soit assez bien optimisé pour limiter l'utilisation de la mémoire et des ressources du processeur afin de réduire son temps d'exécution et sa consommation d'électricité. En quatrième lieu, le système doit être sécurisé afin d'éviter toute tentative de contournement ou de piratage. Les données des utilisateurs doivent, également, être protégées de manière adéquate et ne doivent pas être divulguées à des tiers sans autorisations des personnes concernées. De plus, le système doit être capable de gérer un nombre de visages relativement élevé et, éventuellement, croissant. En outre, le système doit être compatible avec plusieurs types de caméras et de matériels informatiques afin de garantir sa portabilité. Sans oublier la maintenance qui doit être facile grâce à des mécanismes de mise à jour et de correction d'erreurs, afin de minimiser les interruptions. Enfin, il doit être simple d'utilisation. Pour cela, il doit disposer d'une interface graphique qui permette à l'utilisateur de manipuler le système sans trop de difficultés.
Ces exigences sont essentielles pour garantir la précision, l'efficacité, la sécurité, la longévité et la simplicité d'utilisation du système.

2.3.4 Les acteurs du système

On distingue trois principaux acteurs :

✓ ***Le visiteur*** **:**

- Pour pouvoir déverrouiller la porte, le visiteur devra manipuler un écran avec une interface graphique. Lorsque le visage est reconnu, l'utilisateur devra cliquer sur un bouton afin de pouvoir saisir son mot de passe. Si le mot de passe est correct, la porte du bâtiment se déverrouillera automatiquement. Sinon, la porte restera verrouillée et une alarme sera déclenchée.

✓ ***Le propriétaire*** :

- Le propriétaire de la maison (ou concierge de l'immeuble) aura accès sur son ordinateur à une application qui envoit une notification à chaque fois qu'une personne inconnue se présente devant la porte ou que quelqu'un se trompe de mot de passe. Le propriétaire aura également accès en temps réel aux images filmées par la caméra.

✓ ***L'administrateur*** :

- L'administrateur du système aura accès à une base de données dans laquelle il pourra ajouter des visages.

2.3.5 Schéma de fonctionnement du système de sécurité

2.3.5.1 Acquisition des visages

Pour obtenir les visages, il suffit de prendre un cliché du visage à détecter ou de filmer une vidéo dans laquelle se trouve ledit visage. (Voir annexe 2.2.3)

2.3.5.2 Détection des Visages

Pour détecter l'emplacement du visage, il existe deux méthodes :

- La première méthode est la classification (des visages) avec un SVM. Pour cet algorithme, la classification se base sur la méthode histogramme de gradient orienté (HOG) qui est utilisée comme représentation des caractéristiques. Pour utiliser cette technique, il existe la classe `dlib.fhog_object_detector` de la bibliothèque dlib.

- La deuxième méthode repose sur l'utilisation d'un réseau de neurones convolutifs. Pour utiliser cette technique, il existe la classe `dlib.cnn_face_detection_model_v1` de la bibliothèque dlib.

(Voir annexe 1.8, 1.9, 1.10 et 1.11)

2.3.5.3 Méthode d'extraction des caractéristiques

Lorsque l'emplacement du visage est détecté, il faut en extraire les caractéristiques. À chaque caractéristique, on associe une ou plusieurs valeurs appelées encodages faciaux. Ces valeurs sont regroupées dans une matrice. Cette matrice est stockée et utilisée, plus tard, pour reconnaître le visage. (Voir annexe 1.12, 1.13 et 1.14)

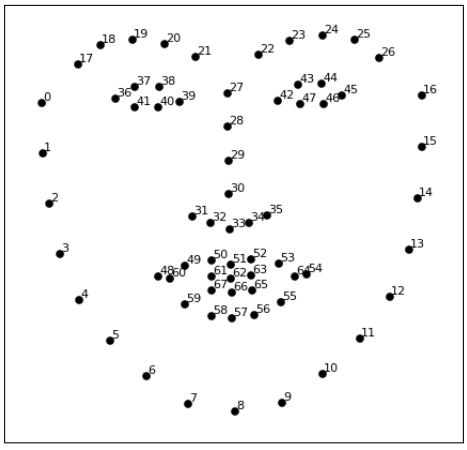


FIGURE 2.6 – Caractéristiques utilisées pour créer les encodages faciaux [11]

2.3.5.4 Choix du classificateur :

Le SVM avec la méthode HOG donne d'assez bons résultats pour des problèmes de reconnaissance faciale relativement simples, avec une consommation minimale des ressources. D'un autre côté, si on dispose d'un assez grand nombre de données et si on n'a pas de contraintes de ressources, il est préférable d'utiliser un réseau de neurones convolutifs, car ce dernier donne de meilleurs résultats.

2.3.5.5 Reconnaissance des visages :

Pour reconnaître le visage d'un individu, il faut extraire ses encodages faciaux et les comparer avec les encodages faciaux présents dans la base de données. Pour ce faire, on calcule la distance entre le visage détecté et ceux dans la base de données. Cette distance correspond à la norme de la matrice des encodages faciaux du visage détecté moins la matrice des encodages faciaux du visage dans la base de données. En python, cette distance est calculée avec la fonction `numpy.linalg.norm` de la bibliothèque numpy. On choisit, de la base de données, les visages qui donnent une distance inférieure à une valeur de tolérance (par défaut égale à 0.6). Parmi les visages sélectionnés, on prend celui qui donne la plus petite distance. Ce visage est (fort probablement) celui de la personne devant la caméra. Si aucun visage, de la base de données, ne donne une distance inférieure à la valeur de tolérance, alors on conclue que le visage est inconnu. (Voir annexe 1.6 et 1.15)

2.3.6 Protocole client-serveur

Le propriétaire reçoit en temps réel les images filmées par la caméra ainsi que des notifications, comme le montre la figure ci-dessous. Le visiteur joue le rôle du serveur et le propriétaire joue le rôle du client.

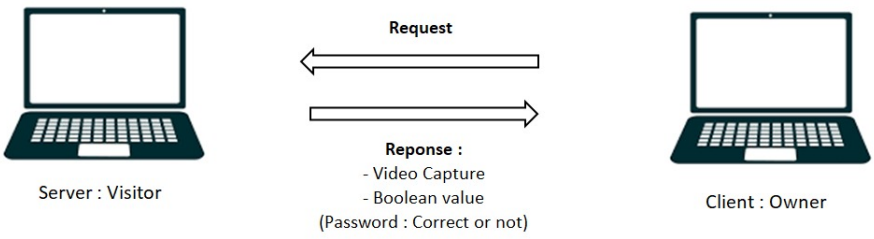


FIGURE 2.7 – Protocole client-serveur

2.4 Diagramme de cas d'utilisation

La figure suivante est le diagramme de cas d'utilisation, il représente les différents cas d'utilisation du système par les visiteurs, le propriétaire et l'administrateur.

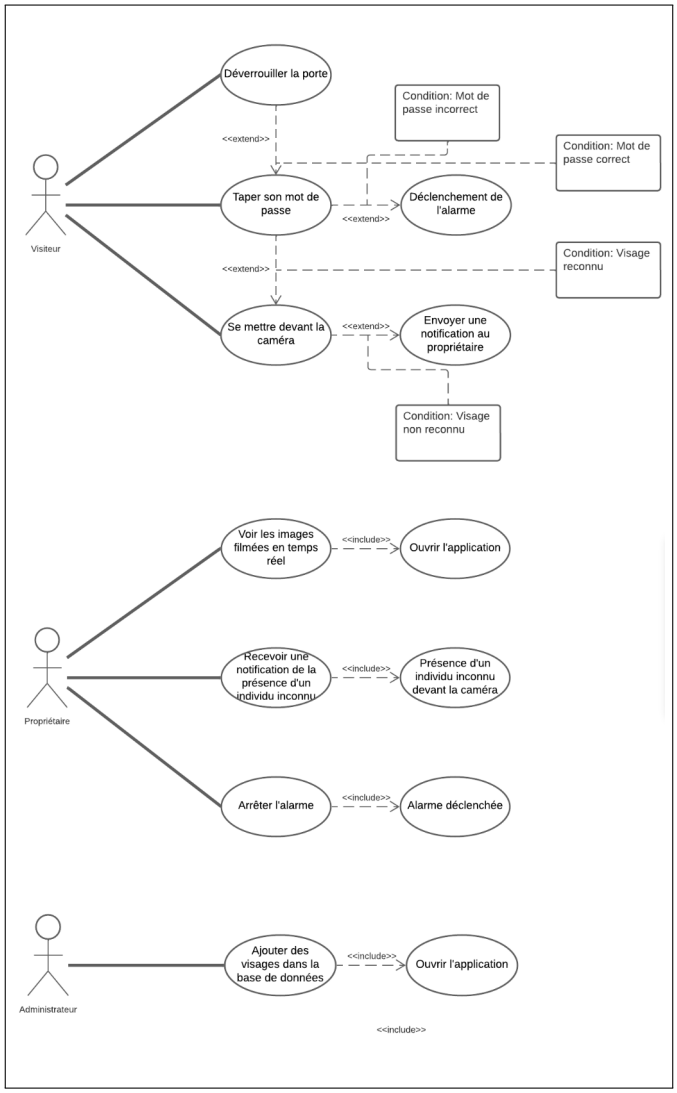


FIGURE 2.8 – Diagramme de cas d'utilisation

Le visiteur peut se mettre devant la caméra. Si son visage n'est pas reconnu par l'algorithme de reconnaissance faciale, une notification est envoyée au propriétaire. Sinon, le visiteur pourra entrer le mot de passe pour déverrouiller la porte. Si le mot de passe tapé est incorrect, la porte ne s'ouvre pas et une alarme se déclenche.

Le propriétaire peut regarder, en temps réel, les images filmées par la caméra. Si un inconnu (non reconnu par l'algorithme de reconnaissance faciale) se présente devant la porte, le propriétaire reçoit une notification. Le propriétaire peut, également, arrêter l'alarme qui se déclenche lorsque quelqu'un se trompe de mot de passe.

L'administrateur peut ajouter des visages dans la base de donnéees.

2.5 Architecture du système

La figure suivante montre l'architecture générale du système :

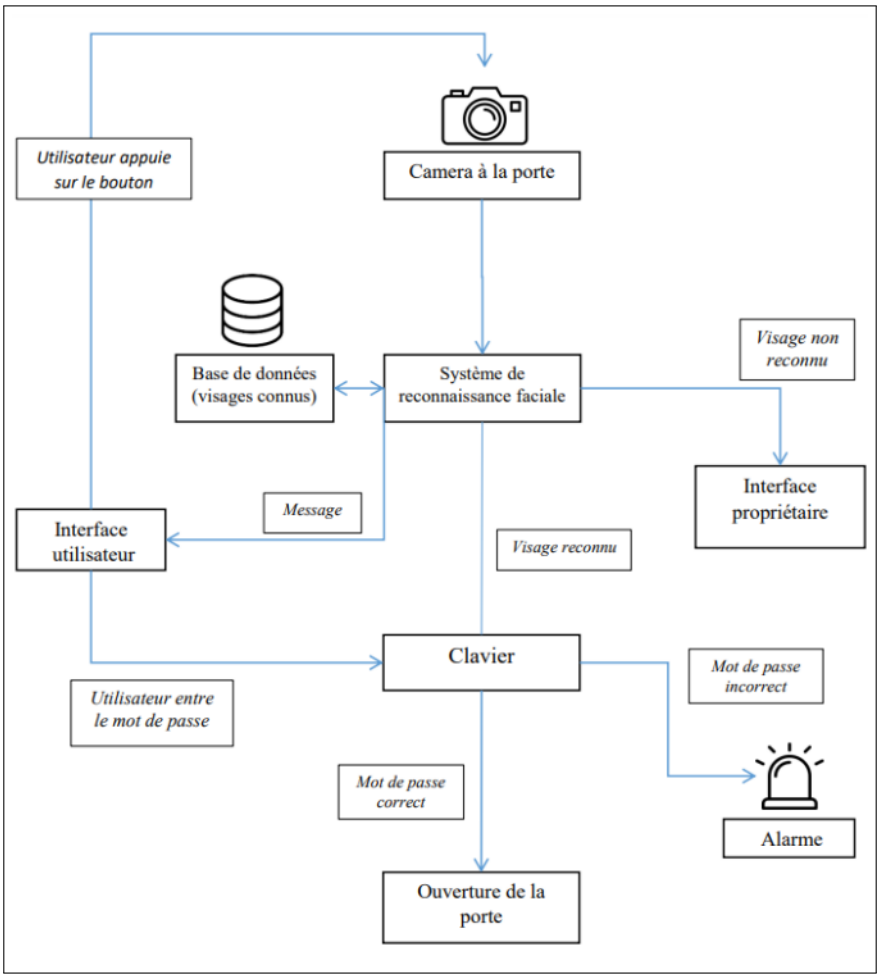


FIGURE 2.9 – Diagramme d'architecture du système

Ce diagramme montre comment les différents éléments du système sont connectés entre eux. La caméra capture les images des personnes devant la porte lorsqu'il appuie sur un bouton. Ces captures seront par la suite analysées par le système de reconnaissance faciale. Si le visage n'est pas reconnu, un avertissement est envoyé au propriétaire. Sinon, il entre un code qui a pour effet de déclencher soit l'ouverture de la porte, soit une alarme.

2.6 Les maquettes de l'application

2.6.1 Interface utilisateur

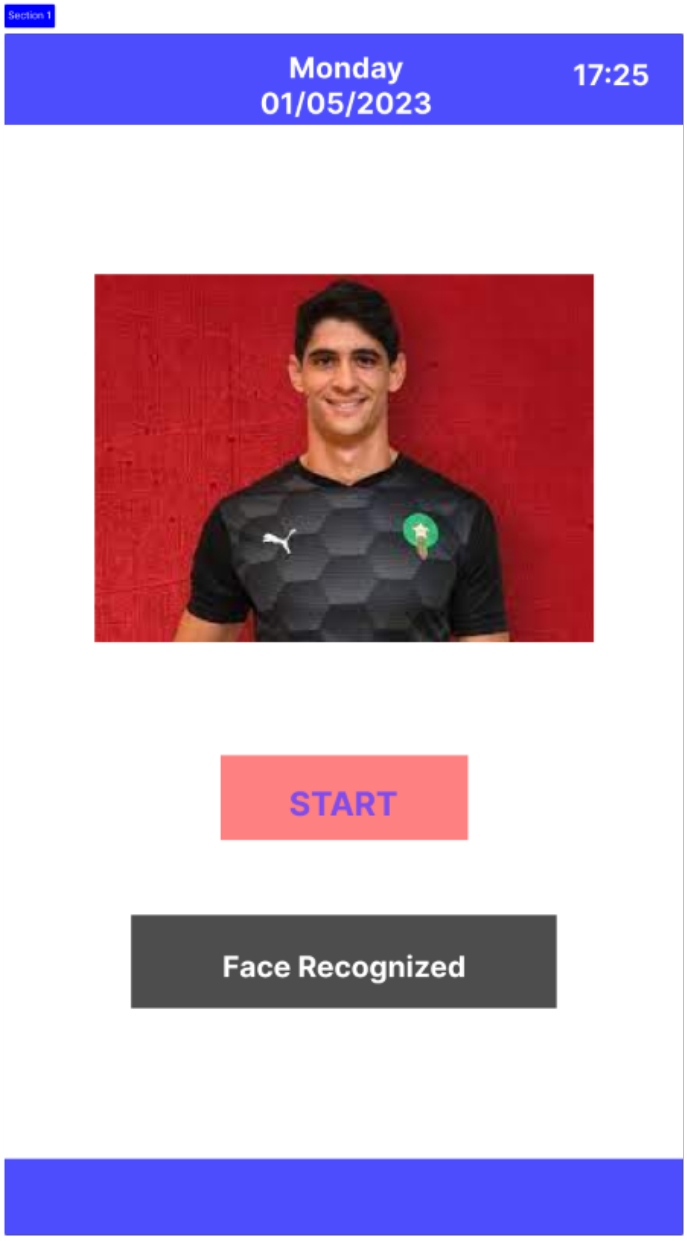


FIGURE 2.10 – Interface utilisateur : Visage reconnu

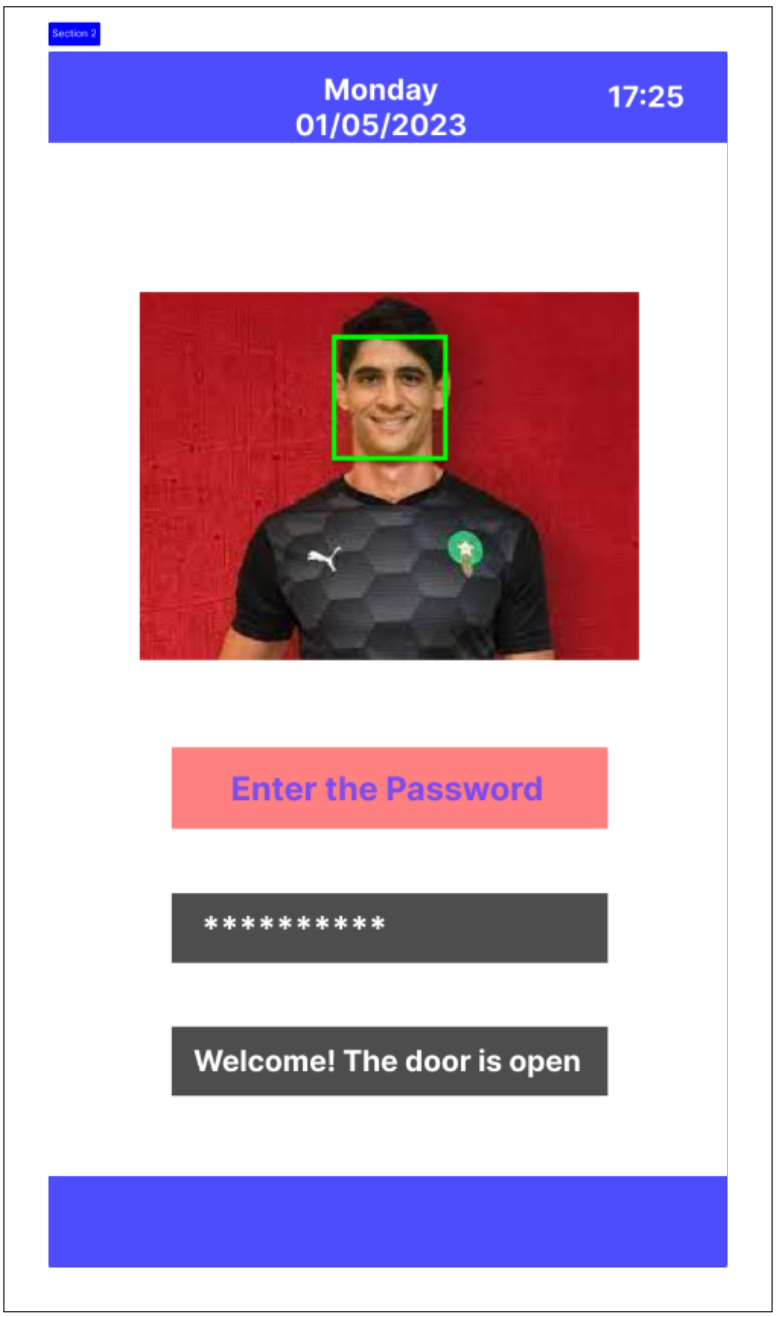


FIGURE 2.11 – Interface utilisateur : Mot de passe correct, la porte se déverrouille

2.6.2 Interface propriétaire

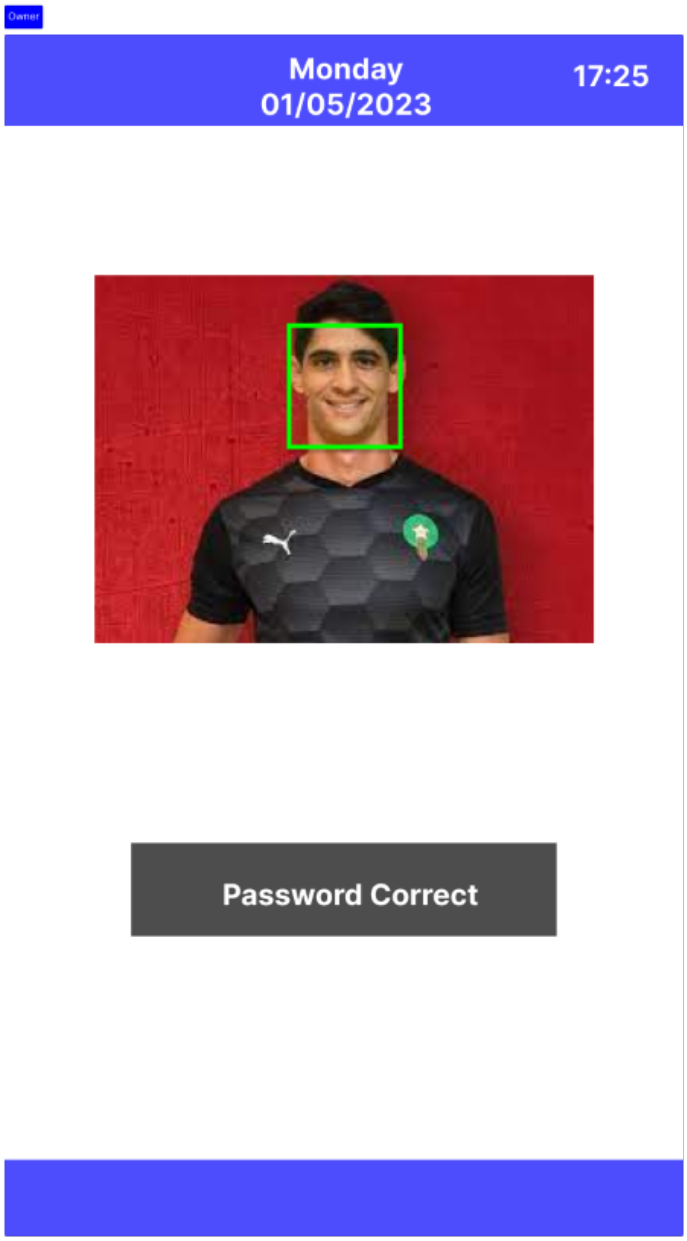


FIGURE 2.12 – Interface propriétaire

2.6.3 Interface administrateur

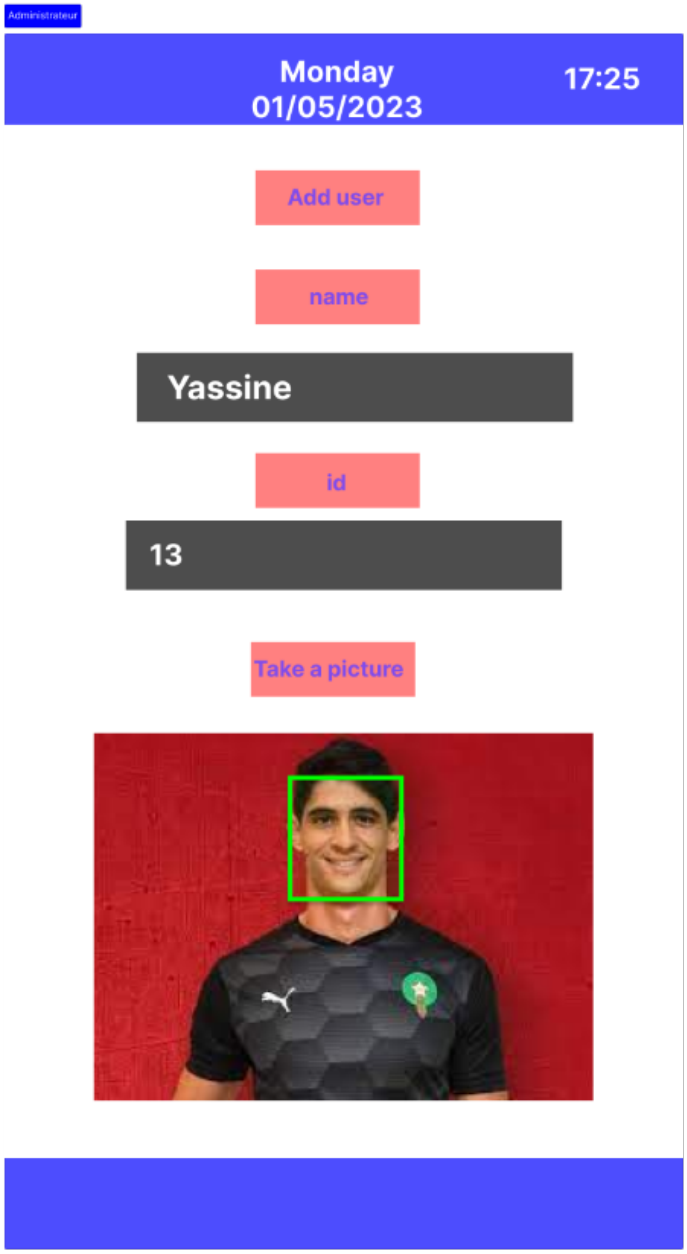


FIGURE 2.13 – Interface administrateur

2.7 Conclusion

Dans ce chapitre, nous avons présenté, au début, un aperçu du fonctionnement des différents algorithmes d'apprentissage automatique ainsi que l'application de ce dernier dans la reconnaissance faciale. Plus précisément, le machine learning est utilisé pour localiser le visage et extraire ses régions. Puis, nous avons vu l'analyse fonctionnelle du système, qui comporte les besoins techniques, les exigences fonctionnelles et non fonctionnelles, les acteurs du système, le schéma de fonctionnement de l'algorithme de reconnaissance faciale et le protocole client-serveur. Par la suite, nous avons vu le diagramme de cas d'utilisation, l'architecture du système et, finalement, les maquettes de l'application.

Chapitre 3

Réalisation

3.1 Introduction

Dans ce chapitre, nous allons montrer l'architecture technique de notre application. Ensuite, nous présenterons les différentes bibliothèques utilisées pendant sa réalisation. Nous expliquerons, également, comment ont été réalisé l'algorithme de reconnaissance faciale, les différentes interfaces graphiques et le protocole client-serveur.

3.2 Architecture technique de l'application

La figure ci-dessous montre l'architecture technique de l'application :

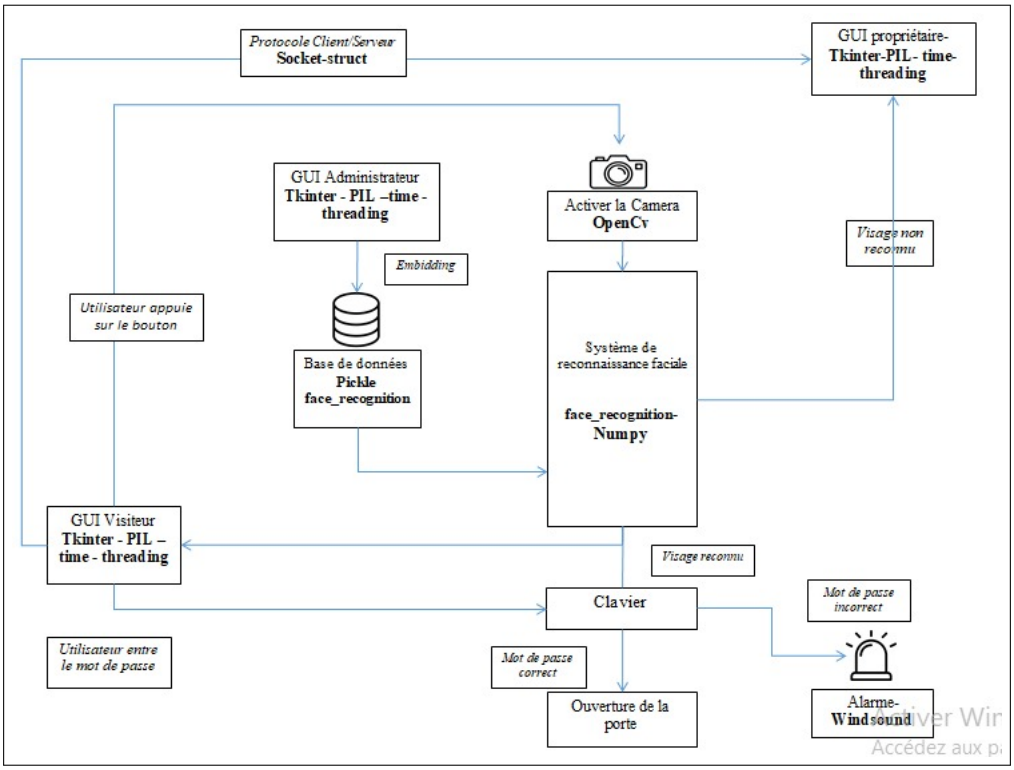


FIGURE 3.1 – l'architecture technique de l'application

L'application comprend différentes interfaces graphiques pour l'administrateur, le propriétaire et le visiteur.

Plusieurs outils sont utilisés pour réaliser ces interfaces. La bibliothèque Tkinter pour créer les fenêtres, les boutons et les frames. La bibliothèque Time est utilisée pour afficher l'heure et la date. La bibliothèque PIL est employée pour afficher la vidéo de la webcam dans la fenêtre, tandis que la bibliothèque threading est utilisée pour mettre en œuvre le multithreading.

Pour stocker les visages, la bibliothèque Pickle est utilisée, en combinaison avec face_recognition. L'implémentation de l'algorithme de reconnaissance faciale fait appel aux bibliothèques numpy et face_recognition. Enfin, en cas de mot de passe incorrect, la bibliothèque Winsound est utilisée pour déclencher l'alarme.

Pour lier la partie visiteur et propriétaire on établit le protocole Client-Serveur Grace à la bibliothèque Socket

3.3 Outils utilisés

3.3.1 Numpy

Numpy est une bibliothèque qui permet de faire du calcul numérique sous Python. Numpy dispose de plusieurs fonctions mathématiques et permet de manipuler des tableaux multidimensionnels et des matrices. Nous allons utiliser la fonction `argmin` de cette bibliothèque qui renvoie un tableau contenant les indices des valeurs minimales le long d'un axe.

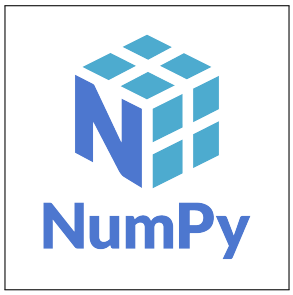


FIGURE 3.2 – Numpy

3.3.2 Time

Time est une bibliothèque pour Python qui permet de représenter le temps sous plusieurs formes. Cette bibliothèque fournit également des fonction pour la manipulation du temps. Nous utiliserons cette bibliothèque pour afficher le jour et l'heure dans les différentes interfaces graphiques.

3.3.3 OpenCV

OpenCV est une bibliothèque de vision par ordinateur. Nous l'utiliserons pour activer la webcam et extraire l'image filmée.[1]

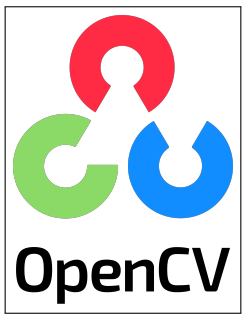


FIGURE 3.3 – OpenCV

3.3.4 Dlib et face-recognition

Dlib est une bibliothèque d'apprentissage automatique (machine learning) pour la reconnaissance des formes. Elle peut être utilisée pour la détection des visages, la détection des formes, la reconnaissance faciale et la segmentation d'images.[1]

Face_recognition est une bibliothèque Python qui fournit une interface facile à utiliser pour la détection et la reconnaissance faciales. Elle fournit des fonctionnalités pour la détection de visages dans des images et des vidéos, ainsi que pour la reconnaissance de visages. Cette bibliothèque utilise des techniques d'aprentissage profond (deep learning) de la bibliothèque dlib. Nous allons utiliser cette bibliothèque pour détecter les visages, extraire les encodages faciaux et identifier les personnes qui se présentent devant la caméra.[1]

3.3.5 Pickle

Pickle est une bibliothèque qui permet de transformer les données sous Python en un flux d'octets (*byte stream*). Le but étant la transmission de ces données dans un réseau ou leur stockage dans un fichier. Cette transformation s'appelle le *pickling*. On utilisera Pickle pour stocker les noms, ids et encodages faciaux dans des fichiers.

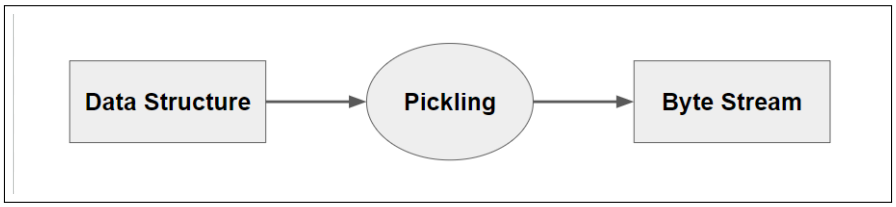


FIGURE 3.4 – Pickling

3.3.6 Tkinter et PIL

Tkinter est une bibliothèque qui permet de créer des interfaces graphiques sous Python.[6]

FIGURE 3.5 – Tkinter

PIL (Python Imaging Library) est une bibliothèque qui permet d'ouvrir et de manipuler des formats d'images différents. Nous allons utiliser les fontions `ImageTk.PhotoImage` et `Image.fromarray` de PIL afin de placer la webcam dans l'interface graphique.[3]

3.3.7 Winsound

Winsound est une bibliothèque pour la gestion du son sous Python. Nous l'utiliserons pour déclencher une alarme si le visiteur se trompe de mot de passe.

3.3.8 Threading

Threading est une bibliothèque qui permet la gestion des threads sous Python. Nous l'utiliserons pour gérer les processus de notre programme.[2]

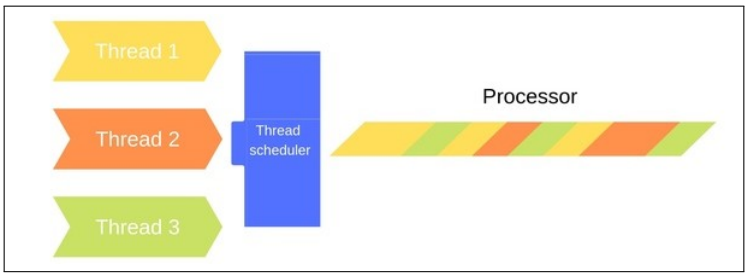


FIGURE 3.6 – Gestion des threads

3.3.9 Socket

Socket est une bibliothèque qui permet d'établir des communications réseau entre des ordinateurs. Nous l'utiliserons pour établir une connexion entre l'appareil qui détecte le visage du visiteur et celui du propriétaire.[4][5]

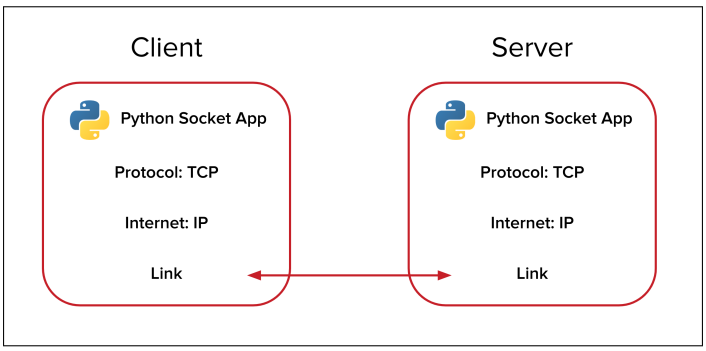


FIGURE 3.7 – Gestion de la connexion client-serveur

3.4 Mise en oeuvre

3.4.1 Algorithme de reconnaissance faciale

3.4.1.1 Webcam

Pour utiliser la webcam, il faut instancier un objet de la classe `cv2.VideoCapture`.[1]

3.4.1.2 Ajout et stockage des données d'un nouvel utilisateur

Lorsque l'adminitrateur veut ajouter un nouveau visage, il doit, en premier lieu, entrer le nom et l'id de la personne en question. Ensuite, il devra prendre 5 photos du visage à stocker afin de créer les encodages faciaux. Les nom et ids des utilisateurs seront stockés dans un fichier, appelé `ref_name.pkl`, et les encodages faciaux seront stockés dans un fichier, appelé `ref_embed.pkl`.[1] (Voir annexe 2)

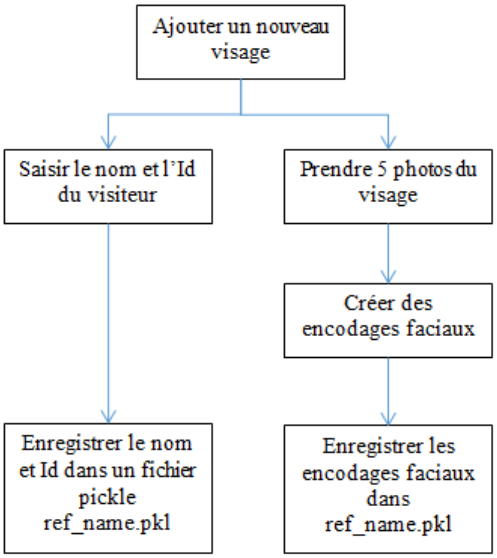


FIGURE 3.8 – Organigramme : Ajout et stockage des données d'un nouvel utilisateur

3.4.1.3 Création et stockage des encodages faciaux (face encodings)

Pour pouvoir reconnaître des visages, il faut transformer les images en des données numériques, appelées encodages faciaux, et les stocker dans un fichier à l'aide de la bibliothèque pickle. Pour réaliser l'encodage facial, on utilise les fonctions `face_locations(img, number_of_times_to_upsample=1, model="hog")` et `face_encodings(face_image, known_face_locations=None, num_jitters=1, model="small")`. La première fonction implémente un algorithme d'apprentissage profond pour la détection des visages. La deuxième fonction renvoie l'encodage facial (face encoding) du visage en paramètre.[1]

Algorithm 1 Reconnaître les visages

```
0: function ReconnaîtreVisages(img)
0:    img : image en entrée
0:    Charger l'image img
0:    Utiliser l'algorithme de détection de visages pour trouver les positions des visages dans l'image
0:    positions_visages ← face_recognition.face_locations(img, number_of_times_to_upsample=1, model="hog")
0:    Encodages_faciaux ← []
0:    for each position_visage in positions_visages do
0:       Visage ← extraire_visage(img, position_visage)
0:       Encodage_facial ← face_recognition.face_encodings(Visage, known_face_locations=None, num_jitters=1, model="small")
0:       Encodages_faciaux.append(Encodage_facial)
0:    Enregistrer les encodages faciaux dans un fichier à l'aide de la bibliothèque pickle
0:    return Encodages_faciaux
=0
```

3.4.1.4 Identification des visages

On détecte les visages dans la vidéo en continu. Tout d'abord, la vidéo est lue et une image est extraite à chaque itération de la boucle. Ensuite, les emplacements des visages sont détectés dans l'image. Les encodages faciaux sont également extraits à partir de ces emplacements de visages. Ils seront par la suite comparés aux encodages faciaux connus pour identifier les personnes présentes.[1] (Voir annexe 3)

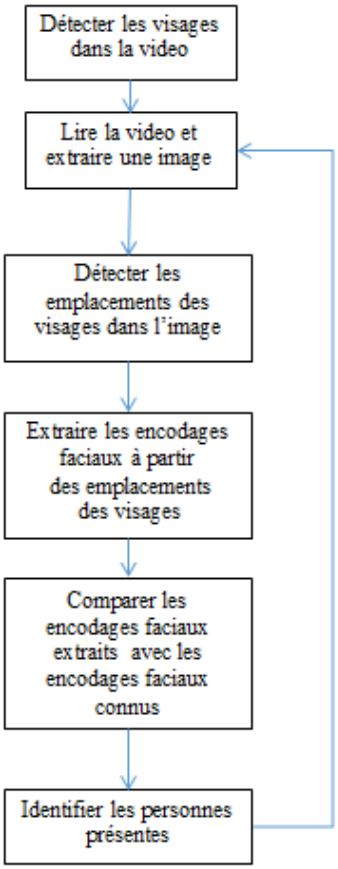


FIGURE 3.9 – Organigramme : Identification des visages

3.4.2 Protocole client-serveur

Nous voulons transmettre, en temps réel, les images filmées par la webcam ainsi que des notifications sur l'ordinateur du propriétaire. Pour sela nous devons établir une connexion client-serveur entre l'ordinateur du propriétaire et l'ordinateur du visiteur.[4][5] (Voir annexe 3.4 et 4)

3.5 Présentation des interfaces

3.5.1 Interface utilisateur

Il s'agit d'une interface graphique avec un bouton "Next" et une caméra. Lorsque le visiteur appuie sur ledit bouton, l'application lui demande de taper le mot de passe, si son visage est reconnu. Si le visage n'est pas reconnu, un message s'affiche sur l'écran. (Voir annexe 4)

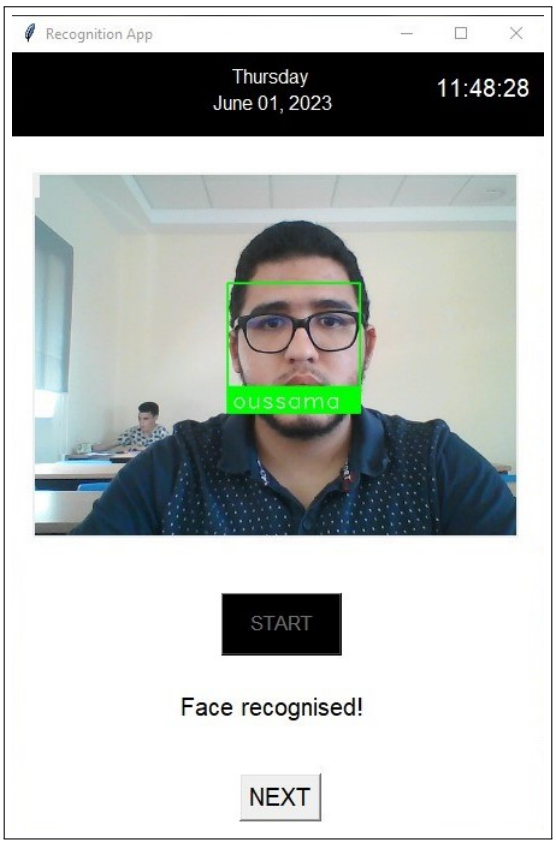


FIGURE 3.10 – Interface utilisateur : Visage reconnu

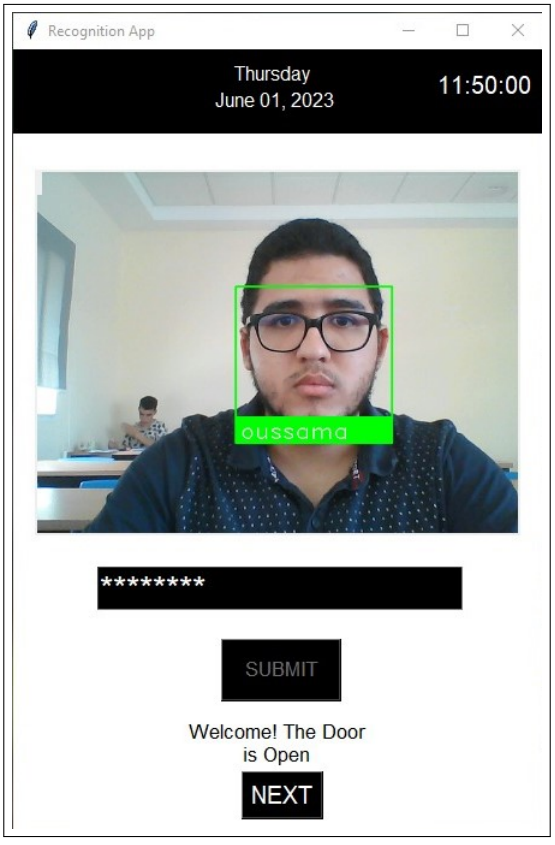


FIGURE 3.11 – Interface utilisateur : Mot de passe correct, la porte se déverrouille

3.5.2 Interface propriétaire

L'interface propriétaire permet à ce dernier de voir en temps réel les images filmées par la caméra et affiche des messages sur l'écran si le visiteur est reconnu par le système et si le visiteur se trompe de mot de passe. (Voir annexe 5)

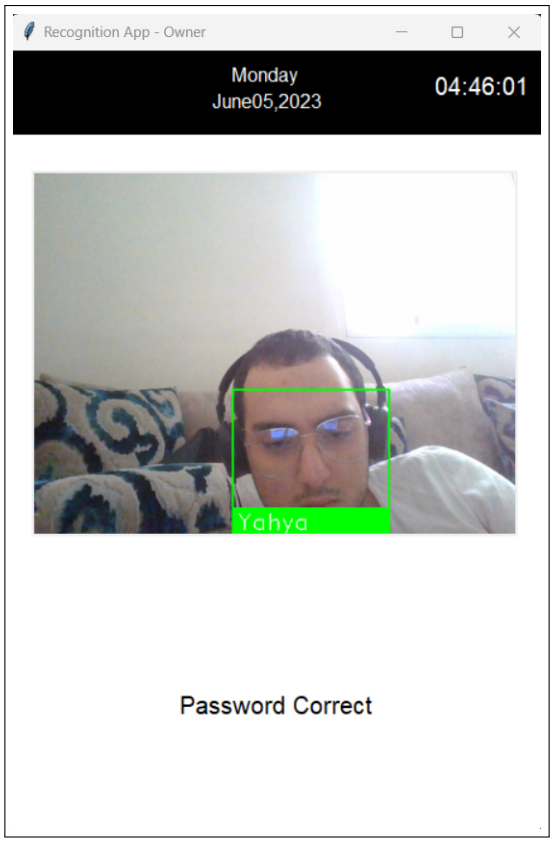


FIGURE 3.12 – Interface propriétaire

3.5.3 Interface administrateur

Lorsque l'application réservée à l'administrateur est ouverte, une fenêtre avec un bouton "*Add user*" apparaît sur l'écran. Lorsque l'utilisateur appuie sur ce bouton, l'application demande à l'administrateur d'entrer le nom et l'id du nouveau visage à enregistrer. De plus, un bouton "*Continue*" apparaît sur l'écran sur lequel l'utilisateur doit cliquer après avoir entré le nom l'id. Ensuite, la webcam et un bouton "*Take picture*" apparaissent sur la fenêtre. Pour prendre cinq photos, l'utilisateur doit appuyer sur ledit bouton cinq fois. (Voir annexe 2)

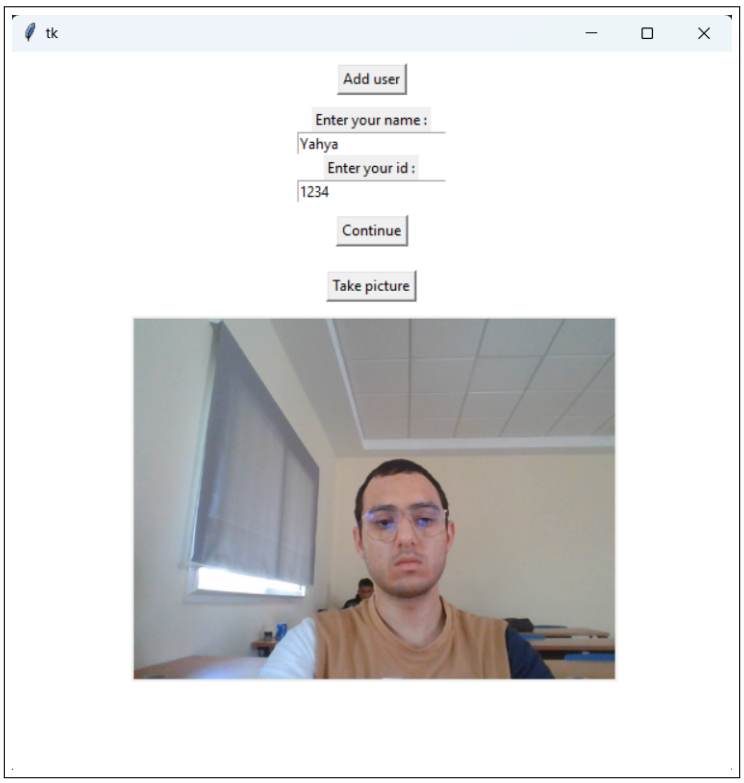


FIGURE 3.13 – Interface administrateur

3.6 Conclusion

Dans ce troisième et dernier chapitre, nous avons exposé l'architecture technique qui montre le fonctionnement général de l'application. Nous avons, également, décrit les différents outils utilisés, tels que Numpy, OpenCV, et face_recognition. Ensuite, nous avons expliqué les différentes étapes que nous avons suivies pour réaliser l'algorithme de reconnaissance faciale. Finalement, nous avons mis en lumière le fontionnement et la réalisation des interfaces graphiques et du protocole client-serveur.

Conclusion générale et perspectives

En guise de conclusion, nous avons réalisé un système de contrôle d'accès à un bâtiment à l'aide de la reconnaissance faciale. Les personnes enregistrées dans la base de données du propriétaire sont reconnues par l'algorithme de reconnaîssance faciale. Il peuvent, ainsi, déverrouiller la porte de bâtiment en tapant leur mot de passe. Si le mot de passe est incorrect, la porte ne se déverrouille pas et une alarme se déclenche. Si une personne n'est pas reconnue par l'algorithme de reconnaîssance faciale, le propriétaire est mis au courant de la présence d'un inconnu à proximité de la porte grâce au protocole client-serveur.

Dans ce rapport, nous avons commencé par une présentation du projet et de ses besoins, suivis d'une analyse de l'architecture générale du système et de ses différentes composantes. Ensuite, nous avons présenté l'architecture technique de l'application et les outils utilisés lors de la réalisation du projet. Enfin, nous avons expliqué le fonctionnement des différentes fonctionnalités de l'application.

De nos jours, les algorithmes de détection de visages deviennent de plus en plus précis avec des taux d'erreurs de plus en plus faibles. Ce développement ouvre la voie à la reconnaissance faciale en tant que technologie de prédilection pour renforcer la sécurité dans différents domaines. Ainsi, on pourrait, dans les années avenir, assister à une démocratisation de la reconnaissance faciale dans des usages diverses et variés tels que le contrôle d'accès aux quartiers résidentiels, aux unités industriels, aux administrations ...

Finalement, la question qui se pose est celle du degré d'efficacité de ce système de sécurité. Le système que nous avons réalisé dispose de deux couches de sécurité. La première couche est la reconnaissance faciale, et la deuxième couche est le mot de passe. La porte ne se déverrouille que si le visage est reconnu et le mot de passe est correct. Alors, les interrogations suivantes se posent : Cette sécurité est-elle suffisante ? Si oui, est-il possible de supprimer la deuxième couche de sécurité, c'est-à-dire, est-il possible de se contenter de la reconnaissance faciale ? Sinon, quels sont les protocoles de sécurité qu'il faudrait ajouter pour rendre le système plus sécurisé ?

Annexe

Dans cette partie, nous présentons les principaux fichiers Python développés pour notre projet de système de reconnaissance faciale. Ces fichiers sont essentiels pour la mise en œuvre du système, couvrant à la fois l'interface graphique pour les visiteurs et les propriétaires, le stockage des visages dans les fichiers Pickle, ainsi que l'implémentation de l'algorithme de reconnaissance faciale et du protocole Client/Serveur.

1 Fonctions de la bibliothèque face_recognition [9]

1.1 Importation des bibliothèque

```
import PIL.Image
import dlib
import numpy as np
from PIL import ImageFile
```

1.2 Configuration de la reconnaissance faciale

```
try:
    import face_recognition_models
except Exception:
    print("Please install `face_recognition_models` with this command before using
        `face_recognition`:\n")
    print("pip install git+https://github.com/ageitgey/face_recognition_models")
    quit()

ImageFile.LOAD_TRUNCATED_IMAGES = True

face_detector = dlib.get_frontal_face_detector()

predictor_68_point_model = face_recognition_models.pose_predictor_model_location()
pose_predictor_68_point = dlib.shape_predictor(predictor_68_point_model)

predictor_5_point_model =
    face_recognition_models.pose_predictor_five_point_model_location()
pose_predictor_5_point = dlib.shape_predictor(predictor_5_point_model)

cnn_face_detection_model = face_recognition_models.cnn_face_detector_model_location()
cnn_face_detector = dlib.cnn_face_detection_model_v1(cnn_face_detection_model)

face_recognition_model = face_recognition_models.face_recognition_model_location()
face_encoder = dlib.face_recognition_model_v1(face_recognition_model)
```

1.3 Conversion d'un objet rect de Dlib en tuple CSS

```
def _rect_to_css(rect):
    """
    Convert a dlib 'rect' object to a plain tuple in (top, right, bottom, left) order
    :param rect: a dlib 'rect' object
    :return: a plain tuple representation of the rect in (top, right, bottom, left) order
    """
    return rect.top(), rect.right(), rect.bottom(), rect.left()
```

1.4 Conversion d'une tuple CSS en objet rect de Dlib

```
def _css_to_rect(css):
    """
    Convert a tuple in (top, right, bottom, left) order to a dlib `rect` object
    :param css:  plain tuple representation of the rect in (top, right, bottom, left)
        order
    :return: a dlib `rect` object
    """
    return dlib.rectangle(css[3], css[0], css[1], css[2])
```

1.5 Réduction d'un tuple CSS aux limites de l'image

```
def _trim_css_to_bounds(css, image_shape):
    """
    Make sure a tuple in (top, right, bottom, left) order is within the bounds of the
        image.
    :param css:  plain tuple representation of the rect in (top, right, bottom, left)
        order
    :param image_shape: numpy shape of the image array
    :return: a trimmed plain tuple representation of the rect in (top, right, bottom,
        left) order
    """
    return max(css[0], 0), min(css[1], image_shape[1]), min(css[2], image_shape[0]),
        max(css[3], 0)
```

1.6 Calcul de la distance entre les encodages faciaux

```
def face_distance(face_encodings, face_to_compare):
    """
    Given a list of face encodings, compare them to a known face encoding and get a
        euclidean distance
    for each comparison face. The distance tells you how similar the faces are.
    :param face_encodings: List of face encodings to compare
    :param face_to_compare: A face encoding to compare against
    :return: A numpy ndarray with the distance for each face in the same order as the
        'faces' array
    """
    if len(face_encodings) == 0:
        return np.empty((0))

    return np.linalg.norm(face_encodings - face_to_compare, axis=1)
```

1.7 Chargement d'un fichier image en tableau numpy

```
def load_image_file(file, mode='RGB'):
    """
    Loads an image file (.jpg, .png, etc) into a numpy array
    :param file: image file name or file object to load
    :param mode: format to convert the image to. Only 'RGB' (8-bit RGB, 3 channels) and
        'L' (black and white) are supported.
    :return: image contents as numpy array
    """
    im = PIL.Image.open(file)
    if mode:
        im = im.convert(mode)
    return np.array(im)
```

1.8 Localisation brute des visages dans une image

```
def _raw_face_locations(img, number_of_times_to_upsample=1, model="hog"):
    """
    Returns an array of bounding boxes of human faces in a image
    :param img: An image (as a numpy array)
    :param number_of_times_to_upsample: How many times to upsample the image looking for
        faces. Higher numbers find smaller faces.
    :param model: Which face detection model to use. "hog" is less accurate but faster
        on CPUs. "cnn" is a more accurate
                  deep-learning model which is GPU/CUDA accelerated (if available). The
                      default is "hog".
    :return: A list of dlib 'rect' objects of found face locations
    """
    if model == "cnn":
        return cnn_face_detector(img, number_of_times_to_upsample)
```

```
    else:
        return face_detector(img, number_of_times_to_upsample)
```

1.9 Localisation des visages dans une image

```
def face_locations(img, number_of_times_to_upsample=1, model="hog"):
    """
    Returns an array of bounding boxes of human faces in a image
    :param img: An image (as a numpy array)
    :param number_of_times_to_upsample: How many times to upsample the image looking for
        faces. Higher numbers find smaller faces.
    :param model: Which face detection model to use. "hog" is less accurate but faster
        on CPUs. "cnn" is a more accurate
                  deep-learning model which is GPU/CUDA accelerated (if available). The
                      default is "hog".
    :return: A list of tuples of found face locations in css (top, right, bottom, left)
        order
    """
    if model == "cnn":
        return [_trim_css_to_bounds(_rect_to_css(face.rect), img.shape) for face in
            _raw_face_locations(img, number_of_times_to_upsample, "cnn")]
    else:
        return [_trim_css_to_bounds(_rect_to_css(face), img.shape) for face in
            _raw_face_locations(img, number_of_times_to_upsample, model)]
```

1.10 Localisation brute des visages en mode batch dans une liste d'images

```
def _raw_face_locations_batched(images, number_of_times_to_upsample=1, batch_size=128):
    """
    Returns an 2d array of dlib rects of human faces in a image using the cnn face
        detector
    :param images: A list of images (each as a numpy array)
    :param number_of_times_to_upsample: How many times to upsample the image looking for
        faces. Higher numbers find smaller faces.
    :return: A list of dlib 'rect' objects of found face locations
    """
    return cnn_face_detector(images, number_of_times_to_upsample, batch_size=batch_size)
```

1.11 Localisation en mode batch des visages dans une liste d'images

```
def batch_face_locations(images, number_of_times_to_upsample=1, batch_size=128):
    """
    Returns an 2d array of bounding boxes of human faces in a image using the cnn face
        detector
    If you are using a GPU, this can give you much faster results since the GPU
    can process batches of images at once. If you aren't using a GPU, you don't need
        this function.
    :param images: A list of images (each as a numpy array)
    :param number_of_times_to_upsample: How many times to upsample the image looking for
        faces. Higher numbers find smaller faces.
    :param batch_size: How many images to include in each GPU processing batch.
    :return: A list of tuples of found face locations in css (top, right, bottom, left)
        order
    """
    def convert_cnn_detections_to_css(detections):
        return [_trim_css_to_bounds(_rect_to_css(face.rect), images[0].shape) for face
            in detections]

    raw_detections_batched = _raw_face_locations_batched(images,
        number_of_times_to_upsample, batch_size)

    return list(map(convert_cnn_detections_to_css, raw_detections_batched))
```

1.12 Extraction brute des repères faciaux à partir d'une image

```
def _raw_face_landmarks(face_image, face_locations=None, model="large"):
    if face_locations is None:
        face_locations = _raw_face_locations(face_image)
    else:
        face_locations = [_css_to_rect(face_location) for face_location in
            face_locations]

    pose_predictor = pose_predictor_68_point

    if model == "small":
        pose_predictor = pose_predictor_5_point

    return [pose_predictor(face_image, face_location) for face_location in
        face_locations]
```

1.13 Extraction des repères faciaux à partir d'une image

```
def face_landmarks(face_image, face_locations=None, model="large"):
    """
    Given an image, returns a dict of face feature locations (eyes, nose, etc) for each
        face in the image
    :param face_image: image to search
    :param face_locations: Optionally provide a list of face locations to check.
    :param model: Optional - which model to use. "large" (default) or "small" which only
        returns 5 points but is faster.
    :return: A list of dicts of face feature locations (eyes, nose, etc)
    """
    landmarks = _raw_face_landmarks(face_image, face_locations, model)
    landmarks_as_tuples = [[(p.x, p.y) for p in landmark.parts()] for landmark in
        landmarks]

    # For a definition of each point index, see
        https://cdn-images-1.medium.com/max/1600/1*AbEg31EgkbXSQehuNJBlWg.png
    if model == 'large':
        return [{
            "chin": points[0:17],
            "left_eyebrow": points[17:22],
            "right_eyebrow": points[22:27],
            "nose_bridge": points[27:31],
            "nose_tip": points[31:36],
            "left_eye": points[36:42],
            "right_eye": points[42:48],
            "top_lip": points[48:55] + [points[64]] + [points[63]] + [points[62]] +
                [points[61]] + [points[60]],
            "bottom_lip": points[54:60] + [points[48]] + [points[60]] + [points[67]] +
                [points[66]] + [points[65]] + [points[64]]
        } for points in landmarks_as_tuples]
    elif model == 'small':
        return [{
            "nose_tip": [points[4]],
            "left_eye": points[2:4],
            "right_eye": points[0:2],
        } for points in landmarks_as_tuples]
    else:
        raise ValueError("Invalid landmarks model type. Supported models are ['small',
            'large'].")
```

1.14 Encodage des visages dans une image

```
def face_encodings(face_image, known_face_locations=None, num_jitters=1, model="small"):
    """
    Given an image, return the 128-dimension face encoding for each face in the image.
    :param face_image: The image that contains one or more faces
```

```
    :param known_face_locations: Optional - the bounding boxes of each face if you
        already know them.
    :param num_jitters: How many times to re-sample the face when calculating encoding.
        Higher is more accurate, but slower (i.e. 100 is 100x slower)
    :param model: Optional - which model to use. "large" or "small" (default) which only
        returns 5 points but is faster.
    :return: A list of 128-dimensional face encodings (one for each face in the image)
    """
    raw_landmarks = _raw_face_landmarks(face_image, known_face_locations, model)
    return [np.array(face_encoder.compute_face_descriptor(face_image, raw_landmark_set,
        num_jitters)) for raw_landmark_set in raw_landmarks]
```

1.15 Comparaison des visages encodés

```
def compare_faces(known_face_encodings, face_encoding_to_check, tolerance=0.6):
    """
    Compare a list of face encodings against a candidate encoding to see if they match.
    :param known_face_encodings: A list of known face encodings
    :param face_encoding_to_check: A single face encoding to compare against the list
    :param tolerance: How much distance between faces to consider it a match. Lower is
        more strict. 0.6 is typical best performance.
    :return: A list of True/False values indicating which known_face_encodings match the
        face encoding to check
    """
    return list(face_distance(known_face_encodings, face_encoding_to_check) <= tolerance)
```

2 Embedding.py

2.1 Importation des bibliothèques

```
import tkinter as tk
import cv2
import face_recognition as fr
import pickle
from PIL import Image, ImageTk
import threading
import time
```

2.2 Les fonctions

2.2.1 fonction pour afficher le bouton "Continue"

```
#Functions
# Function to show continue button
def show_continue():
     continue_button.pack(pady=10)
```

2.2.2 fonction pour afficher des champs de saisie

```
# Function to show input fields
def show_input():
    # Pack name label
    label_name.pack()
    # Pack name entry
    input_name.pack()
    # Pack ref_id label
    label_id.pack()
    # Pack ref_id entry
    input_id.pack()
```

2.2.3 fonction pour obtenir le contenu du saisie

```
# Function to get user input
def get_input():
    global name
    global ref_id
    # Put the name entered by the user in the variable name
    name = input_name.get()
    # Put the id entered by the user in the variable ref_id
    ref_id = input_id.get()
    if name and ref_id: # If the user entered his name and id store them in ref_name.pkl
        ref_dict[ref_id] = name # The id is the key and the name is the value
        f = open("ref_name.pkl", "wb") # Open the pickle file (ref_name.pkl)
        pickle.dump(ref_dict, f) # Send ref_dict as a byte stream to ref_name.pkl
        f.close() # Close the file after finishing
        display_webcam() # When the name and the id are stored in the pickle file,
            display the webcam
```

2.2.4 fonction pour afficher la webcam (show webcam)

```
# Function to show webcam
def show_webcam():
    def face_loc(): # Function to create face encodings
        global button_pressed
        face_locations = fr.face_locations(rgb_small_frame) # Detect face locations
```

```
        if face_locations!=[]: #If  there's a face
            face_encoding = fr.face_encodings(frame)[0] # Store face encodings in
                face_encoding
            if ref_id in embed_dict: # If the user id already exists in embed_dict
                embed_dict[ref_id] += [face_encoding] # Append face_encoding to the face
                    encodings that already exist
            else: # If the user id doesn't exist in embed_dict
                embed_dict[ref_id] = [face_encoding] # Store face encoding in the
                    embed_dict
            # Stop webcam
            webcam.release()
            cv2.waitKey(1)
            cv2.destroyAllWindows()

        button_pressed = True # There's a button that starts thisfunction, this boolean
            variable is used to detect when the button is pressed

    # Button to take a picture and create face locations
    button = tk.Button(root, text="Take picture", command=face_loc) # Create the button
        that starts the function face_loc
    button.pack(pady=10) # Pack the button

    for i in range(5): # Take 5 pictures
        key = cv2.waitKey(1)
        webcam = cv2.VideoCapture(0) # Instantiate a cv2.VideoCapture object (webcam)

        # Create a label to display the video feed
        video_label = tk.Label(root) # Create a label in the GUI
        video_label.place(x=100, y=220)  # Use place method to position the label

        while True:
            button_pressed = False # Initially False because initially the button is not
                pressed
            ret, frame = webcam.read()

            if not ret: break # If it doesn't film, stop the loop

            small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25) # Make a smaller
                image to make the face_encodings function run faster
            rgb_small_frame = cv2.cvtColor(small_frame, cv2.COLOR_BGR2RGB) # Matrice
                with RGB values for face_encodings function

            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB) # Matrice with RGB values
                to insert webcam in the GUI
            # Resize the frame to fit the window
            frame_resized = cv2.resize(frame_rgb, (400, 300)) # Resize the image that
                will be inserted in the GUI
            # Convert the frame to ImageTk format
            img = Image.fromarray(frame_resized) # Create an image memory from an object
                exporting the array interface (For GUI)
            img_tk = ImageTk.PhotoImage(image=img) # A Tkinter-compatible photo image
                (For GUI)

            # Update the image in the Tkinter label
            video_label.config(image=img_tk)
            video_label.image = img_tk

            key = cv2.waitKey(1)

            if button_pressed: break # When the button "Take picture" is pressed stop
                the loop
    # When 5 pictures are taken, show the message " Face Saved " and stop the program
        after 3 seconds
    button.destroy()
    video_label.destroy()
    frame_start_webcam.destroy()
    message_label = tk.Label(root,font=("Arial",15),bg="black",fg="white", text="")
    message_label.place(x=200,y=290)
    message_to_start= tk.Label(root,font=("Arial",25),bg="black",fg="white", text=" Face
        Saved ")
```

```
    message_to_start.place(x=200,y=290)
    time.sleep(3)

    f = open("ref_embed.pkl", "wb")
    pickle.dump(embed_dict, f)
    f.close()
    # Close window
    root.destroy()
    # Close program
    exit()
```

2.2.5 fonction pour afficher des champs de saisie (display webcam)

```
def display_webcam():
    # Function that calls show_webcam()
    def execution_cap():
        show_webcam()

    thread = threading.Thread(target=execution_cap) # Create a thread for the function
        execution_cap
    thread.start() # Start the thread
```

2.2.6 fonction pour executer le code

```
# Function to run code
def run_code():
    show_input() # Call the show_input function
    show_continue() # Call the show_continue function
```

2.3 main

```
#Execution starts here
# Global variables
name = "" # name of the user
ref_id = "" # id of the user

#Create a dictionnary with all names and ids
try:
    f = open("ref_name.pkl", "rb")
    ref_dict = pickle.load(f)
    f.close()
except:
    ref_dict = {}

#Create a dictionnary with all faces and ids
try: # If ref_embed.pkl, store its data in embed_dict
    f = open("ref_embed.pkl", "rb")
    embed_dict = pickle.load(f)
    f.close()
except: # If ref_embed.pkl doesn't exist create an empty dict
    embed_dict = {}

# Create GUI
root = tk.Tk()
root.geometry("600x600") #GUI geometry
root.config(background="white") # GUI background color is white

# Label for name
label_name = tk.Label(root, text="Enter your name :")

# Entry for name
input_name = tk.Entry(root)

# Label for ref_id
```

```
label_id = tk.Label(root, text="Enter your id :")

# Entry for ref_id
input_id = tk.Entry(root)

# Button
button = tk.Button(root, text="Add user", command=run_code)
button.pack(pady=10)

# Button to continue
continue_button = tk.Button(root, text="Continue", command=get_input)

# Frame in the GUI for the webcam
frame_start_webcam = tk.Frame(root, height=300, width=400, bg="white",bd=1,
    relief="flat")
frame_start_webcam.place(x=100,y=200)


# Run GUI
root.mainloop()
```

3 recognition.py

3.1 Importation des bibliothèques

```
import threading
import cv2
import numpy as np
import pickle
import face_recognition as fr
import tkinter as tk
from PIL import ImageTk, Image
import socket, pickle,struct
import winsound
import os
```

3.2 Chargement de données picklées et création de listes de visages

Ce code charge des données picklées à partir des fichiers `"ref_name.pkl" et "ref_embed.pkl"`, et les utilise pour créer des listes d'identifiants et d'encodages de visages.

```
# Open the file "ref_name.pkl" in binary mode for reading
f = open("ref_name.pkl", "rb")
ref_dict = pickle.load(f)           # Load the pickled data from the file and assign it
    to the variable ref_dict
f.close()           # Close the file

# Open the file "ref_embed.pkl" in binary mode for reading
f = open("ref_embed.pkl", "rb")
embed_dict = pickle.load(f)
f.close()

# Create empty lists to store face IDs and face encodings
known_face_ids = []
known_face_encodings = []

# Iterate over the items in embed_dict, which contains reference IDs and corresponding
    embedding lists
for ref_id, embed_list in embed_dict.items():
    # Iterate over the embedding list for each reference ID
    for my_embed in embed_list:
```

```
        # Append the embedding to the known_face_encodings list
        known_face_encodings += [my_embed]
        # Append the reference ID to the known_face_ids list
        known_face_ids += [ref_id]

isRecognized_val = False
```

3.3 Reconnaissance faciale

```
# Returns wether the face is Recognized or not
def prog(window,x,y):
    global isRecognized_val


    face_locations = []
    face_encodings = []
    face_ids = []
    process_this_frame = True

    #start webcam
    cap = cv2.VideoCapture(0)

    # Create a label to display the video feed
    video_label = tk.Label(window)
    video_label.place(x=x, y=y)  # Use place method to position the label

    i=0
    isRec_list = list()

    while True:
        global isRecognized_val
        ret, frame = cap.read()
        if not ret:
                break

        # Resize the frame to fit the window
        small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

        # Convert the frame to RGB format
        rgb_small_frame  = cv2.cvtColor(small_frame, cv2.COLOR_BGR2RGB)

        if process_this_frame:
            face_locations = fr.face_locations(rgb_small_frame)
            face_encodings = fr.face_encodings(rgb_small_frame, face_locations)

            face_ids = []

            for face_encoding in face_encodings:
                matches = fr.compare_faces(known_face_encodings, face_encoding)
                id = "Unknown"

                face_distances = fr.face_distance(known_face_encodings, face_encoding)
                best_match_index = np.argmin(face_distances)


                if matches[best_match_index]:
                    id = known_face_ids[best_match_index]
                    face_ids.append(id)
                    isRecognized_val =True
                    i=i+1
                    isRec_list.append(isRecognized_val)
                    print(isRecognized_val)
                else:
                    i=i+1
                    isRecognized_val= False


        process_this_frame = not process_this_frame
```

```
        for (top_s, right, bottom, left), f_id in zip(face_locations, face_ids):
            top_s *= 4
            right *= 4
            bottom *= 4
            left *= 4

            cv2.rectangle(frame, (left, top_s), (right, bottom), (0, 255, 0), 2)

            cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 255, 0),
                cv2.FILLED)
            font = cv2.FONT_HERSHEY_DUPLEX
            cv2.putText(frame, ref_dict[f_id], (left+6, bottom-6), font, 1.0, (255, 255,
                255), 1)

        font = cv2.FONT_HERSHEY_DUPLEX
        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Resize the frame to fit the window
        frame_resized = cv2.resize(frame_rgb, (400, 300))

        key = cv2.waitKey(1)

        if cv2.waitKey(1) & key == ord('q'): break

        if ((True not in isRec_list) and i==10):
            print(False)
            isRecognized_val= False
            return isRecognized_val
        else:
            if isRecognized_val == True :
                cap.release()
                cv2.destroyAllWindows()
                return isRecognized_val

    cap.release()
    cv2.destroyAllWindows()
    return isRecognized_val
```

3.4 Implementation du protocole Client/Serveur

Serveur :

```
# To send in Real time the webcam feed to the Owner
def Server(window,x,y):
    global video_label

    # Socket Create
    server_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    host_name = socket.gethostname()
    host_ip = socket.gethostbyname(host_name)
    print('HOST IP:', host_ip)
    port = 9999
    socket_address = (host_ip, port)

    # Socket Bind
    server_socket.bind(socket_address)

    # Socket Listen
    server_socket.listen(5)
    print("LISTENING AT:", socket_address)


    face_locations = []
    face_encodings = []
    face_ids = []
    process_this_frame = True
```

```
#start webcam
cap = cv2.VideoCapture(0)

# Create a label to display the video feed
video_label = tk.Label(window)
video_label.place(x=x, y=y)  # Use place method to position the label


# Socket Accept
while True:
    client_socket, addr = server_socket.accept()
    print('GOT CONNECTION FROM:', addr)
    if client_socket:
        cap = cv2.VideoCapture(0)
        if not cap.isOpened():
            print("Failed to open the camera.")
            break

        while cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                    break

            small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)
            rgb_small_frame  = cv2.cvtColor(small_frame, cv2.COLOR_BGR2RGB)

            if process_this_frame:
                face_locations = fr.face_locations(rgb_small_frame)
                face_encodings = fr.face_encodings(rgb_small_frame, face_locations)

                face_ids = []

                for face_encoding in face_encodings:
                    matches = fr.compare_faces(known_face_encodings, face_encoding)
                    id = "Unknown"

                    face_distances = fr.face_distance(known_face_encodings,
                        face_encoding)
                    best_match_index = np.argmin(face_distances)


                    if matches[best_match_index]:
                        id = known_face_ids[best_match_index]
                        face_ids.append(id)
                        isRecognized_val = True

            process_this_frame = not process_this_frame

            for (top_s, right, bottom, left), f_id in zip(face_locations, face_ids):
                top_s *= 4
                right *= 4
                bottom *= 4
                left *= 4

                cv2.rectangle(frame, (left, top_s), (right, bottom), (0, 255, 0), 2)

                cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 255,
                    0), cv2.FILLED)
                font = cv2.FONT_HERSHEY_DUPLEX
                cv2.putText(frame, ref_dict[f_id], (left+6, bottom-6), font, 1.0,
                    (255, 255, 255), 1)

            font = cv2.FONT_HERSHEY_DUPLEX
            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

            # Resize the frame to fit the window
            frame_resized = cv2.resize(frame_rgb, (400, 300))

            # Serialization
            a = pickle.dumps(frame_resized)
```

```
                message = struct.pack("Q", len(a)) + a
                client_socket.sendall(message)

                # Convert the frame to ImageTk format
                img = Image.fromarray(frame_resized)
                img_tk = ImageTk.PhotoImage(image=img)

                # Update the image in the Tkinter label
                video_label.config(image=img_tk)
                video_label.image = img_tk


                if cv2.waitKey(1) & 0xFF == ord('q'): break

            cap.release()
            cv2.destroyAllWindows()
```

Client :

```
# To Receive in Real time the webcam feed from the Visitor
def Client(window,x,y):

    # Create a label to display the video feed
    video_label = tk.Label(window)
    video_label.place(x=x, y=y)  # Use place method to position the label
    # create socket
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    host_ip = '192.168.56.1'  # replace with the actual server IP address
    port = 9999
    client_socket.connect((host_ip, port))  # a tuple

    data = b""
    payload_size = struct.calcsize("Q")


    # This section is related with the file that was created to check if Visitor.py was
        executed
    # the path of the file want to delete
    file_path = 'py1_executed.marker'

    try:
        # Delete the file
        os.remove(file_path)
        print(f"File '{file_path}' deleted successfully.")
    except OSError as e:
        print(f"Error occurred while deleting file: {e}")


    # Define the address and port to receive the message
    HOST = host_ip
    PORT = 12345

    # Receiving wether the password is correct or not
    def receive_message():
        # Create a socket object
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            # Bind the socket to the address and port
            s.bind((HOST, PORT))
            # Listen for incoming connections
            s.listen(1)
            # Accept a connection from the sender
            conn, addr = s.accept()
            # Receive the message
            data = conn.recv(1024)

        message_received = data.decode()

        message_label = tk.Label(window,font=("Arial",15),bg="white",fg="black", text="")
        message_label.place(x=138,y=530)
```

```
        # "1" :: True and "0" :: False
        if message_received == "1" :
            print("Received message:", message_received)
            message_label.config(text="Password Correct")       # Update the text of
                message_label to "Password Correct"
        elif message_received=="0" :
            print("Received message:", message_received)
            message_label.config(text="Password not Correct")       # Update the text of
                message_label to "Password not Correct"
            #ALARM
            winsound.PlaySound("Alarm_Sound_Effect",winsound.SND_FILENAME)      # Play
                an alarm sound effect


    thread_msg = threading.Thread(target=receive_message)
    thread_msg.start()

    # Receiving the webcam feed
    while True:
        # Receive and reconstruct the frame data

        # Receive and assemble data packets until the length reaches the expected
            payload size
        while len(data) < payload_size:
            packet = client_socket.recv(4 * 1024)  # Receive packets of size 4K (4 *
                1024 bytes)
            if not packet:
                break
            data += packet

        # Extract the packed message size from the received data
        packed_msg_size = data[:payload_size]
        data = data[payload_size:]
        msg_size = struct.unpack("Q", packed_msg_size)[0]

        # Receive and assemble data packets until the length reaches the expected
            message size
        while len(data) < msg_size:
            data += client_socket.recv(4 * 1024)

        # Extract the frame data from the received data and deserialize it
        frame_data = data[:msg_size]
        data = data[msg_size:]
        frame = pickle.loads(frame_data)

        frame_resized = cv2.resize(frame, (400, 300))

        # Convert the frame to ImageTk format
        img = Image.fromarray(frame_resized)
        img_tk = ImageTk.PhotoImage(image=img)

        # Update the image in the Tkinter label
        video_label.config(image=img_tk)
        video_label.image = img_tk
        cv2.waitKey(1)  # Update the video window

        # Check for termination condition
        key = cv2.waitKey(1) & 0xFF
        if key == ord('q'):
            break

    client_socket.close()
```

4 Interface graphique visiteur : Visitor.py

Importation des bibliothèques

```
import tkinter as tk
import threading, time, recognition2, winsound
import socket
```

fonction traitant la section mot de passe :

la fonction ci-dessous retourne une valeur booleen "True" si le mot de passe est correcte "False" sinon.

```
# To send to the Owner whether the password is wrong or not
def send_password_state(message):
    PORT = 12345
    # iP adress of the host (Visitor)
    host_ip='192.168.56.1'
    #   To send a message
    def send_message(message):
        # Create a socket object
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:

            # Connect to the Owner
            s.connect((host_ip, PORT))

            # Send the message
            s.sendall(message.encode())

    thread_msg = threading.Thread(target=send_message,args=(message,))
    thread_msg.start()
```

Fonction password section.

Cette fonction nous permet de traiter la section du mot de passe où on crée un champ pour saisir le code, un boutton "Submit" pour le soumettre et un zone où s'affiche un message.

```
def password_section(_fen):
    def command_test():
        global isPasswordCorrect
        password = entry.get()          # Get the value entered in the 'entry' widget
            and store it in the 'password' variable
        isPasswordCorrect = (password == "password")

        # If the password is correct:
        if  isPasswordCorrect :
                # door unlocks
                # send message to owner
                send_password_state("1")
                # Update the message in the Visitor GUI
                message_label_mdp.config(text=" Welcome! The Door\n is Open")
                submit_button.config(state="disabled")          # Disable the submit
                    button to prevent further input

        # If the password is incorrect:
        else:
                #send message to owner
                send_password_state("0")
                message_label_mdp.config(text=" Wrong Password!!!")

                # Alarm
                winsound.PlaySound("Alarm_Sound_Effect",winsound.SND_FILENAME)

    # Creates an entry widget for user input, with a black background, white text, and
        obscured characters (*).
```

```
    entry=tk.Entry(_fen,font=("Arial",20),bg="black",fg="white",show="*")
    entry.place(x=70,y=430)

    # Creates a button widget labeled "SUBMIT" with specified font, size, and color, and
        associates it with a command called "command_test".
    submit_button= tk.Button(_fen,text="SUBMIT",font=("Arial",13), width=10,
        height=2,bg="black",fg="white",command=command_test)
    submit_button.place(x=173,y=490)

    # Creates a label widget for displaying a message, with specified font, background
        color, text color, and an empty initial text.
    message_label_mdp = tk.Label(_fen,font=("Arial",13),bg="white",fg="black", text="")
    message_label_mdp.place(x=138,y=555)
```

fonction d'affichage de la date et le temps

```
# to display the day date time
def date_time_vis(fen_):
    def update_():
        # Get the current time and format it as HH:MM:SS
        timeString = time.strftime("%I:%M:%S")
        timeLabel.config(text=timeString)  # Update the text of the timeLabel widget
            with the current time

        # Get the current date and format it as Month DD, YYYY
        dateString = time.strftime("%B %d, %Y")
        dateLabel.config(text=dateString)  # Update the text of the dateLabel widget
            with the current date

        # Get the current day of the week
        dayString = time.strftime("%A")
        dayLabel.config(text=dayString)  # Update the text of the dayLabel widget with
            the current day of the week

        fen_.after(1000, update_)  # Schedule the next update after 1000 milliseconds (1
            second)

    # Create a label widget to display the time
    timeLabel = tk.Label(fen_, font=("Arial", 15), bg="black", fg="white")
    timeLabel.place(x=350, y=15)

    # Create a label widget to display the date
    dateLabel = tk.Label(fen_, font=("Arial", 12), bg="black", fg="white")
    dateLabel.place(x=165, y=30)

    # Create a label widget to display the day of the week
    dayLabel = tk.Label(fen_, font=("Arial", 12), bg="black", fg="white")
    dayLabel.place(x=181, y=8)

    update_()  # Call the update function to start displaying the current time, date,
        and day
```

Fonction principale

```
# the main function where we create the main window
def create_window_visitor():
    isRecognized_val = False

    window = tk.Tk()# Create the main window for the GUI

    window.geometry("443x650")  # Set the dimensions of the window
    window.config(background="white")   # Set the background color of the window to white
    window.title("Recognition App - Visitor")     # Set the title of the window to
        "Recognition App - Visitor"
```

```
x=18
y=100

def window_components():
    global startButton

    # Function called when the "START" button is clicked
    def command_start(isRecognized_val):
        print("start!")
        startButton.config(state="disabled")

        print(isRecognized_val)

        b_value = isRecognized_val
        if b_value:
            message_label.config(text="Face recognised!")
            button_next_2.config(state="active")
        else:
            message_label.config(text="Face not recognised!")

    # Function for the first tab of the interface
    def tab1():
        global button_next_2

        # Function called when the "NEXT" button is clicked
        def tab2():
            message_label.destroy()
            startButton.destroy()
            password_section(window)

        #create next button (it executes the tab2 function)
        button_next_2 =
            tk.Button(window,text="NEXT",font=("Arial",15),bg="black",fg="white",
        command=tab2)
        button_next_2.place(x=190,y=600)
        button_next_2.config(state="disabled")

    #create Start button (it executes the command_start function)
    startButton = tk.Button(window,font=("Arial",13),text="START", width=10,
        height=2,bg="black",fg="white",command=lambda
        :command_start(isRecognized_val))
    startButton.place(x=175, y=450)


    message_label = tk.Label(window,font=("Arial",15),bg="white",fg="black", text="")
    message_label.place(x=138,y=530)

    # create a Frame where we display the date, time and day
    frame1 = tk.Frame(window, height=70, width=443, bg="black",bd=1, relief="flat")
    frame1.place(x=0,y=0)

    date_time_vis(window)        # Display the date, time and day in the GUI
    tab1()      # Set up the first tab of the interface

# execution of theface recognition code imported from the recognition2 file
def execution_recognition2():
        nonlocal isRecognized_val
        isRecognized_val = recognition2.prog(window,x,y)
        return isRecognized_val

window.after(0,window_components)
window.after(100,execution_recognition2)

# to execute the Server code imported from recognition2 where the Visitor send the
    webcam feed to the Owner
thread_server=threading.Thread(target=recognition2.Server,args=(window,x,y))
thread_server.start()

window.mainloop()
```

```
    thread_server.join()
```

Main

```
#MAIN

# Create a marker file to indicate that Visitor.py has been executed
with open('py1_executed.marker', 'w') as marker_file:
    marker_file.write('Executed')

create_window_visitor()
```

5 Interface graphique propriétaire : Owner.py

Importation des bibliothèques

```
import tkinter as tk
import threading
import time
from recognition2 import Client
```

Le cadre de l'entête :

```
# Create a frame where we display the time, day and date
def create_Frame_date(fenetre,height_val,width_val):
    frame1 = tk.Frame(fenetre, height=height_val, width=width_val, bg="black",bd=1,
        relief="flat")
    frame1.place(x=0,y=0)
```

Fonction d'affichage de la date :

```
# Same as date_time_vis in Visitor
def date_time_owner(fen_):
    def update_():
        timeString = time.strftime("%I:%M:%S")
        timeLabel.config(text=timeString)

        dateString = time.strftime("%B%d,%Y")
        dateLabel.config(text=dateString)

        dayString = time.strftime("%A")
        dayLabel.config(text=dayString)
        fen_.after(1000,update_)

    fen_.update()

    timeLabel = tk.Label(fen_,font=("Arial",15),bg="black",fg="white")
    timeLabel.place(x=350,y=15)

    dateLabel = tk.Label(fen_,font=("Arial",12),bg="black",fg="white")
    dateLabel.place(x=165,y=30)

    dayLabel = tk.Label(fen_,font=("Arial",12),bg="black",fg="white")
    dayLabel.place(x=181,y=8)

    update_()
```

Fonction principale :

```
def create_window_owner():
    def create_Frame_date(window,70,443):
    pass   #already defined
    # Create a Tkinter window
    window = tk.Tk()
    window.geometry("443x650")      # Set the dimensions of the window
    window.config(background="white")       # Set the background color of the window to
        white
    window.title("Recognition App - Owner")         # Set the title of the window to
        "Recognition App - Owner"
    create_Frame_date(window,70,443)
    date_time_owner(window)
    import os
    import time

    # Checking if the Visitor.py is executed or not
    while not os.path.exists('py1_executed.marker'):
        # Code to execute in py2 until py1.py is executed
        print("Waiting for Visitor.py to be executed...")

        time.sleep(1)  # Delay for 1 second


    # Code to execute after py1.py is executed
    print("Visitor py has been executed!")
    # Add any other code you want to execute after py1.py is executed

    # to execute the client code imported from recognition2 where the Owner receive the
        webcam feed and wether the password is correct or not from the Visitor
    thread_client=threading.Thread(target=Client,args=(window,18,100))
    thread_client.start()
    window.mainloop()

    #thread_client.join()
```

Main :

```
#Main
create_window_owner()
```

Conclusion

L'annexe présente les principaux fichiers Python qui constituent notre système de reconnaissance faciale. Ces fichiers sont essentiels pour assurer le bon fonctionnement du système, offrant une interface utilisateur conviviale, un stockage efficace des données, ainsi qu'une reconnaissance faciale précise et en temps réel.

Bibliographie

[1] 12-05-2023 :
Deep Learning Project – Face Recognition with Python OpenCV : https://projectgurukul.org/deep-learning-project-face-recognition-with-python-opencv/
C'est un projet de reconnaissance faciale en Python utilisant les bibliothèques dlib et face_recognition d'OpenCV, en deux parties : embedding et recognition.

[2] 21-05-2023 :
Multithreading in Python https://www.geeksforgeeks.org/multithreading-python-set-1/
Cet article couvre les bases du multithreading en Python, expliquant le fonctionnement des threads pour atteindre le multitâche, avec des exemples de création et d'exécution de threads.

[3] 22-05-2023 :
Documentation de la bibliothèque Pillow : https://pillow.readthedocs.io/en/stable/reference/Image.html
Le site présente la bibliothèque de traitement d'images Pillow en Python, qui fournit des fonctions pour charger et d'afficher des images dans une interface graphique tkinter et créer de nouvelles images.

[4] 25-05-2023 :
Python Socket Programming – Server, Client Example : https://www.digitalocean.com/community/tutorials/python-socket-programming-server-client
Ce site présente l'architecture du modèle client/serveur et la programmation Socket Sever, Socket Client.

[5] 25-05-2023 :
Programmation Socket pour envoyer et recevoir la video de la webcam : https://pyshine.com/Socket-programming-and-openc/
Ce site fournit un exemple de code en python utilisant des sockets, permettant de transmettre une vidéo en temps réel.

[6] 27-05-2023 :
Construction d'une application GUI en python avec Tkinter : https://realpython.com/python-gui-tkinter/
Ce site web propose un tutoriel pour apprendre la programmation d'interfaces graphiques avec Tkinter en Python.

[7] 07-06-2023 :
Support Vector Machine (SVM) : https://en.wikipedia.org/wiki/Support_vector_machine
Il s'agit de la page wikipedia des SVM.

[8] 07-06-2023 :
Historique de la reconnassance faciale : https://en.wikipedia.org/wiki/Facial_recognition_system#History_of_facial_recognition_technology
Il s'agit de la section qui traite l'histoire de la reconnaissance faciale de la page wikipedia sur la reconnaissance faciale.

[9] 07-06-2023 :
Implémentation de quelques fonctions de la bibiothèque face_recognition : https://github.com/ageitgey/face_recognition/blob/2e2dccea9dd0ce730c8d464d0f67c6eebb40c9d1/face_recognition/api.py#L26
Page github de la bibliothèque face_recognition.

[10] 07-06-2023 :
Réseaux de neurones convolutifs (CNN) : https://fr.wikipedia.org/wiki/R%C3%A9seau_neuronal_convolutif
Page wikipedia sur les réseaux de neurones convolutifs.

[11] 07-06-2023 :
Caractéristiques à extraire : https://cdn-images-1.medium.com/v2/resize:fit:1600/1*AbEg31EgkbXSQehuNJBlWg.png
Cette image montre les caractéristiques du visage à extraire pour créer les encodages faciaux.