IUT CLERMONT AUVERGNE
Aurillac - Clermont-Ferrand - Le Puy-en-Velay
Montluçon - Moulins - Vichy

Département
Informatique

Soutenance finale SAE

KeepSafe

GARDEZ UN OEIL OUVERT

Tuteur : Delobel François

Année 2022-2023 / Groupe 03

Enzo JOLYS, Othmane BENJELLOUN, Rami KHEDAIR,
Bruno DA COSTA CUNHA et Nathan VERDIE

Remerciements

Tout d'abord, nous tenons à exprimer notre plus sincère gratitude envers M. Delobel, notre tuteur, qui nous a guidé, aidé et enseigné de nombreuses choses au cours de cette SAE. Les mots ne suffisent pas pour exprimer notre reconnaissance envers lui. Il nous a appris comment un bon programmeur réfléchit, analyse et résout les problèmes. Nous lui sommes redevables pour tous ses enseignements précieux.

Nous voulons également remercier chaleureusement notre tutrice, Mme. Chatti, pour sa disponibilité constante à nous expliquer nos erreurs et nous montrer les bonnes pratiques de gestion de projet.

Enfin, nous ne pouvons pas oublier l'effort déployé par Mme. Mottet, notre secrétaire, qui nous a soutenus et aidé à imprimer nos rapports. Nous lui devons notre gratitude pour cette aide apportée pendant le projet

Sommaire

I. Introduction

Notre projet KeepSafe, il consiste en une caméra connectée. Il permettra aux utilisateurs de surveiller efficacement un bâtiment ou d'autres zones à distance. On avait pour notre projet, une durée d'environ 130 heures.

Nous avons choisi ce sujet, car il y a un besoin croissant de surveillance à distance, spécialement pour les personnes qui ont des propriétés éloignées et qui ne peuvent pas y être présentes en permanence. La surveillance à distance permet de détecter les intrusions ou des anomalies dans un environnement. Ce besoin nécessite d'utiliser une caméra connectée, avec éventuellement des fonctionnalités avancées telles que la vision nocturne, la reconnaissance de mouvements et identifier les anomalies.

Notre problématique principale est donc comment mettre en place une surveillance à distance efficace d'un bâtiment ou d'autres zones en utilisant les fonctionnalités de la Raspberry Pi, tout en garantissant la confidentialité et la sécurité des données ?

Nous avons par la suite élargi notre problématique pour couvrir aussi la photographie animalière, mais aussi la détection de fuite de chaleur afin de proposer un objet connecté versatile, doté de multiples capteurs, qui puisse couvrir de multiples besoins.

Pour réaliser ce projet, nous avons établi des objectifs globaux : nous avions besoin de développer une technique pour capturer des images normales et nocturnes tout en utilisant le moins d'énergie possible. Il était important de mettre en place différents moyens de communication pour permettre à l'utilisateur de visualiser les images de n'importe où et dans différentes situations.

Afin de répondre à ces objectifs, vous retrouverez ci-dessous le plan de notre développement :

- 1 : Partie physique
- 2 : Réseau
- 3 : Logiciel
- 4 : Application Web : Back-end
- 5 : Application Web : Front-end

A. Cas d'utilisation

Notre projet comporte de nombreux cas d'utilisations. Grâce à KeepSafe, vous pourrez laisser votre caméra chez vous pour surveiller votre maison ou bien, vous pourrez aussi effectuer de la photographie animalière en laissant la caméra en pleine forêt. Notre caméra sera aussi équipée d'une caméra thermique afin de pouvoir aussi l'utiliser comme détecteur de fuite de chaleur.

Afin de répondre au mieux à ces cas d'utilisations, notre projet sera équipé de plusieurs parties :

- L'utilisateur peut se connecter à notre système en utilisant un navigateur web, en entrant des informations d'identification sur notre site web hébergé sur notre serveur. Une fois connecté, l'utilisateur a accès à tous les services proposés par notre système.
- L'utilisateur peut changer le mode de fonctionnement de sa Raspberry Pi à distance, via un accès à notre site web hébergé sur notre serveur. Il existe 3 modes différents disponibles :
    - Une fois qu'un mouvement est détecté, le système prend une photo de l'objet détecté et l'identifie. Une fois l'objet identifié, le système envoie une notification à l'utilisateur pour informer l'utilisateur de la détection de mouvement et de l'identification de l'objet. Cela permet à l'utilisateur de surveiller à distance son

système et de réagir rapidement en cas de détection d'un événement inhabituel.

- Accéder à la vidéo en direct capturée par la caméra depuis notre site web hébergé sur notre serveur. De ce fait, l'utilisateur peut visualiser en temps réel ce qui se passe à l'endroit où la caméra est installée, ce qui lui donne une vision en direct de ce qui se passe dans son environnement.
- Changer le mode de capture vidéo entre le mode thermique et le mode normal pour la vidéo en direct.

- L'utilisateur peut accéder à une bibliothèque ou une *"library"* qui contient toutes les photos prises par la caméra. Il peut visualiser les images enregistrées, les trier par date ou par catégories d'objets, et les télécharger sur son ordinateur ou les partager avec d'autres personnes.
- Une application qui permet d'améliorer la qualité des images capturées par la caméra thermique en les fusionnant avec des images capturées par une caméra classique. Cela permet d'obtenir des images plus détaillées et plus précises en combinant les informations fournies par les deux types de caméra.
- L'utilisateur a aussi la possibilité d'accéder à la Raspberry PI sans passer par le site web, soit en local, soit à distance en utilisant un outil appelé *SSH (Secure Shell)*.

Pour plus d'information référez vous au schéma qui décrit la structure de notre projet dans l'annexe.

B. Objectifs généraux

- Mettre en place une surveillance à distance d'un bâtiment ou d'autres zones en utilisant les fonctionnalités de la Raspberry Pi.
- Fournir des fonctionnalités avancées telles que la vision nocturne, la reconnaissance de mouvement et l'IA pour détecter les anomalies.
- Garantir la confidentialité et la sécurité des données.

II. Développement

A. La présentation du sujet

1. Les objectifs :

Notre objectif était avant tout de proposer une alternative aux solutions de surveillance actuelle, en effet notre produit permet une mobilité difficilement trouvable chez la concurrence.

2. Gestion de projet :

Pour la gestion de notre projet, vous pouvez vous référer à notre rapport de gestion que nous avons rendu.

B. Partie physique

1. Matériel

Nous devions donc avoir :

- Une carte de contrôle doté d'un microprocesseur assez puissante pour manipuler des images et des flux vidéo.
- Une caméra pour pouvoir filmer et voir en temps réel.
- Une caméra thermique permettant de voir la nuit.
- Un détecteur de mouvement nous avertissant de divers mouvements détectés lors de notre absence.

Il nous a fallu déterminer quelles cartes électroniques nous allions utiliser, nous avions déjà deux cartes potentielles à notre disposition, une carte *Arduino,* mais sur celle-ci, il aurait été assez compliqué de traiter la partie live par exemple. Nous avons donc opté pour la Raspberry PI 4 avec les modules suivants : caméra, caméra thermique et détecteur de mouvement.

2. Le bon système :

Une fois notre carte électronique et ses modules choisis, nous devions choisir un bon système d'exploitation et ceci n'a pas été simple, il fallait que tous les *framework* et les *drivers* dont nous avions besoin pour utiliser nos modules soit disponibles dessus. Nous avons dans un premier temps installer *Bookworm,* une version de *debian (distribution Linux)* en phase de test. Cependant, elle ne possédait pas les drivers dont nous avions besoin. Nous avons donc décidé de regarder ce que préconisent les constructeurs de la Raspberry pi. Ces derniers ont développé un système d'exploitation basé sur GNU Linux nommé anciennement Raspbian mais actuellement Raspberry Pi OS. Nous avons donc utilisé celui-ci pour notre pi.

3. Branchement et soudure :

Dans l'attente d'un *kit d'extension GPIO* et d'un câble Raspberry Pi B+ femelle, femelle (voir image) nous avons dû souder des câbles Arduino à une *extension du GPIO* pour tester les 3 modules sur notre platine d'essai (cette solution est temporaire en attendant la réception du kit). Pour brancher la caméra il suffit de connecter la *naps* sur le port Caméra CSI afin de s'en servir. Également une soudure a été réalisée sur la caméra thermique pour fixer 5 pins. Des câbles permettent de relier la PI4 à la caméra thermique via la platine d'essai pour que le 3V, le SDA, le SCL et le GND soient reliés. Il ne reste donc plus qu'à brancher le détecteur de mouvement pour cela il suffit de connecter le VCC (câble rouge) au 5v de la carte, le OUT (câble orange) au GPIO 4 de la carte et pour finir le GND (câble noir) au GND de la carte.

Figure 1 - Naps

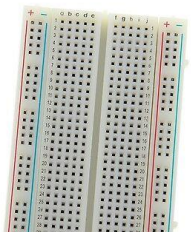


Figure 2 - Platine

Figure 3 - Raspberry

4. Configuration :

Une fois tous les modules installés et branchés à la Raspberry PI nous devions configurer chaque module pour son utilisation.

5. Caméra :

Figure 4 - Caméra

Pour configurer la caméra de la PI il a fallu faire beaucoup de recherche avant de trouver une documentation correspondante à notre matériel, il suffit d'installer le paquet disponible sur ce git : https://github.com/ArduCAM/Arducam-Pivariety-V4L2-Driver/releases/download/install_script/install_pivariety_pkgs.sh

Nous obtiendrons un fichier nommé *« install_pivariety_pkgs.sh -p imx519_kernel_driver_low_speed ».* Il faut donner les droits d'exécution, l'exécuter et redémarrer la pi4. Pour vérifier que tout marche bien, il suffit de lancer cette commande dans un terminal : *dmesg | grep imx519*. Il permettra de bien visualiser la présence des drivers nécessaires à la caméra. Pour détecter la caméra, il suffira d'aller dans le fichier *« /dev/video0 »* et voir s'il n'est pas vide. Une fois arrivé ici, tout est bien configuré. Il ne reste que quelques détails. Par la suite, nous avons installé un autofocus.

En donnant les droits et en exécutant les fichiers adéquats nous nous retrouvons avec une caméra fonctionnelle avec un autofocus intégré.

6. Caméra thermique :

Figure 5 - Caméra Thermique

La partie caméra thermique a été la plus complexe pour nous, trouver de la documentation et les bons drivers a été un réel défi. Nous sommons aller sur ce gitlab : Git pimoroni afin d'installer son contenu, après avoir modifié des lignes dans le fichier /boot/config.txt nous avons dû installer plusieurs paquets de lib. A l'aide de ces paquets nous avons pu voir le bon fonctionnement de la caméra thermique.

7. Détecteur de mouvement :

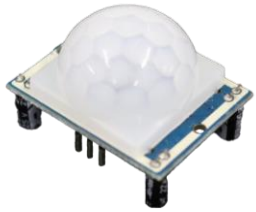


Figure 6 - Détecteur

Le détecteur de mouvement a été le plus simple à configurer, car après les branchements, il a suffi de créer un fichier python donnant le port sur lequel était branché le capteur et à l'intérieur d'une boucle infinie et dans le cadre de nos tests nous avons affiché quand il détecte et ne détecte pas un mouvement.

Après avoir travaillé sur les différents éléments, nous avons dû les monter sur une plaque pour garantir la stabilité et fixer les positions des images. Cela était nécessaire pour assurer la qualité de l'image finale. Voici le résultat obtenu.

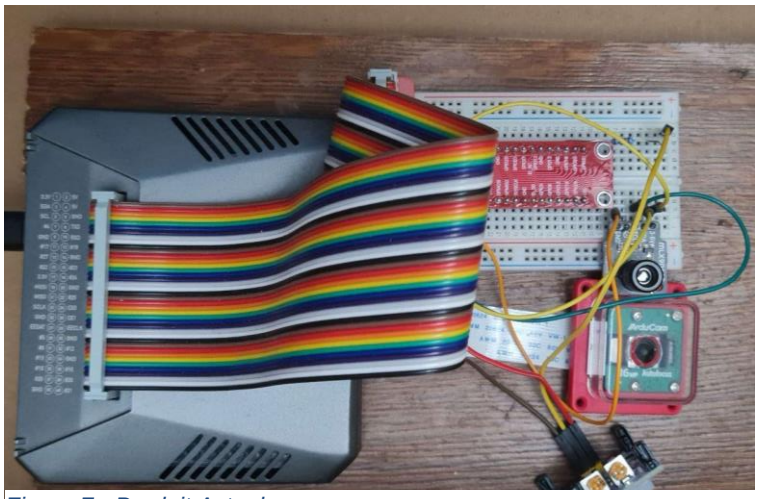


Figure 7 - Produit Actuel

C. Réseau

La partie réseau de notre projet a été conçue pour répondre à deux cas d'utilisation principaux : se connecter à la Raspberry PI dans un réseau privé et utiliser la Raspberry PI comme dispositif de surveillance animalière.

1. Accéder la Raspberry PI dans un réseau privé

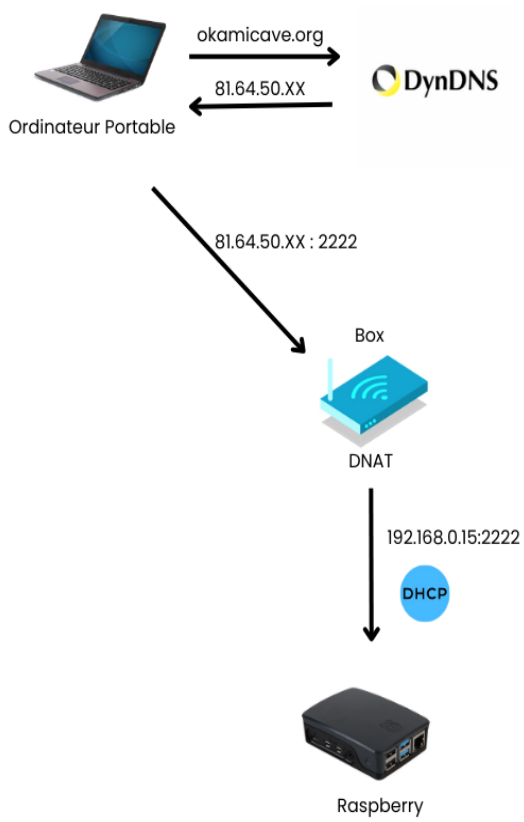


Figure 8 - Schéma du fonctionnement du réseau

Afin de permettre une connexion à notre Raspberry PI dans un réseau privé, nous avons dû mettre en place plusieurs configurations. Tout d'abord, nous avons configuré un service *"Dynamic Host Configuration Protocol" (DHCP)* pour attribuer automatiquement une adresse IP statique à notre Raspberry. Ensuite, pour pouvoir accéder à la Raspberry depuis un réseau extérieur, nous avons utilisé une technique de *"Network Address Translation" (NAT)* appelée *"Destination NAT" (DNAT)* pour rediriger les connexions entrantes vers l'adresse IP de notre Raspberry. Pour que cette technique fonctionne, nous avons configuré un service de *"Secure Shell" (SSH)*, notamment en changeant les ports *SSH* par défaut pour qu'on puisse se connecter depuis un réseau extérieur en fonction du port utilisé.

De plus, grâce aux configurations précédemment établies, nous pouvons maintenant effectuer des connexions *SSH* depuis des machines connectées au même réseau que la Raspberry de manière fiable et

sécurisée, en utilisant simplement l'adresse IP privée de la Raspberry PI et le port configuré pour *SSH*.

En outre, pour gérer les problèmes liés à des changements fréquents d'adresse IP publique de l'emplacement de la Raspberry PI, nous avons utilisé un service de nom de domaine dynamique (*DynDNS*).

Ce service maintient un nom d'hôte statique (okamicave.hopto.org) pour la Raspberry PI, de sorte que même si l'adresse IP publique change, il est toujours possible de se connecter à distance à la Raspberry en utilisant ce nom de domaine statique. Cela permet une connexion à distance efficace et sans difficulté à la Raspberry PI, même en cas de changements fréquents d'adresse IP publique.

2. Transformer la PI en point d'accès

Figure 9 - Image d''illustration du point d'accès

Afin de permettre la Raspberry PI d'être un dispositif de surveillance animalière, il était important de réfléchir à une solution qui permette à la PI de fonctionner en mode hors-ligne, étant donné qu'il n'est pas toujours possible d'accéder à internet dans une forêt par exemple. Pour résoudre ce problème, nous avons choisi de transformer la Raspberry PI en point d'accès en mettant en place les services *HostAP* et *DNSMasq*.

Pour mettre en place ces services, nous avons installé les paquets *"hostapd"* et *"dnsmasq"* sur notre Raspberry PI. Le service *DNSMasq* intègre également un service *DHCP* qui permet d'attribuer des adresses IP aux machines qui se connectent via Wi-Fi à la Raspberry PI en mode point d'accès.

Il a fallu effectuer de nombreuses recherches pour configurer le service *HostAP* de manière à ce qu'il fonctionne correctement.
Enfin, nous avons également mis en place une sécurité pour les connexions à notre point d'accès en utilisant un mot de passe *WPA/WPA2*, un protocole de sécurité utilisé pour chiffrer les données transmises sur un réseau Wi-Fi.

3. Communication entre le serveur et la PI

La communication entre le serveur et la Raspberry Pi est un élément fondamental pour notre projet, car elle permet à la Raspberry Pi de recevoir des instructions sur la prise de photo et de transmettre cette photo au serveur afin qu'elle soit visible par l'utilisateur via l'interface web. Pour gérer cette communication, nous avons choisi d'utiliser le protocole de *"Secure copy" (SCP)* qui utilise le service *SSH* pour transférer les fichiers.

Lorsque l'utilisateur clique sur le bouton "prendre une photo" sur notre site web, une requête *POST* est envoyée au serveur avec une action spécifique pour la prise de photo. Le serveur traite cette requête et exécute la commande *SCP* pour envoyer un fichier à la Raspberry Pi qui contient les instructions pour la prise de photo. La Raspberry Pi reçoit ce fichier et utilise les informations qu'il contient pour prendre la photo, puis envoie la photo capturée à nouveau au serveur via une requête *HTTP POST*.

Pour assurer une communication efficace et sécurisée entre le serveur et la Raspberry Pi, il est important de configurer correctement le service SSH. Cela implique de configurer une clé d'authentification pour éviter les piratages, de configurer les ports appropriés pour éviter les conflits avec d'autres services et de vérifier que le firewall ne bloque pas les communications. En utilisant *SCP* et *HTTP POST*, nous sommes en mesure de transférer des fichiers de manière sécurisée et fiable entre le serveur et la Raspberry Pi, tout en permettant une communication facile pour envoyer et recevoir des informations.

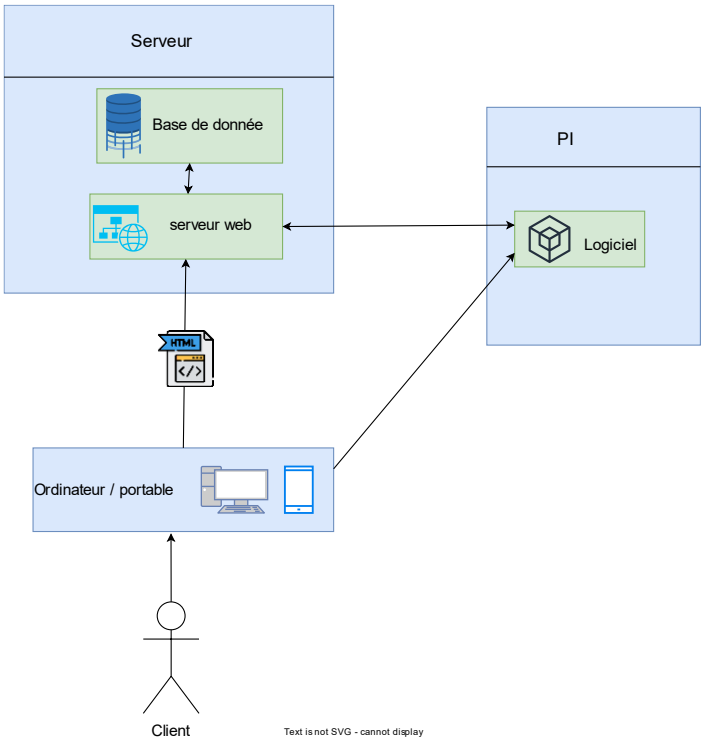


Figure 10 - Schéma structurel du réseau

4. Automatiser les étapes réseaux.

Pour optimiser les étapes de configuration des éléments réseaux vues précédemment, nous avons développé un script automatisé en utilisant le langage shell. Ce script automatise les étapes de configuration nécessaires pour mettre en place un environnement de réseau fonctionnel sur une Raspberry Pi, en simplifiant les tâches fastidieuses et répétitives de la configuration manuelle. Il permet également de garantir une configuration standard sur chaque nouvelle unité, en incluant toutes les étapes de configuration nécessaires pour que l'équipement soit prêt à l'emploi, garantissant ainsi une compatibilité et une fiabilité des connexions réseaux. Ce script inclut les fonctionnalités suivantes :

- Activation du routage pour permettre à la Raspberry Pi de fonctionner en tant que routeur et de connecter les périphériques à Internet.
- Installation et configuration du service *SSH* pour permettre aux utilisateurs de se connecter à distance à la Raspberry Pi.

- Installation et configuration de *DNSMasq* pour gérer les demandes de noms de domaine et les résolutions d'adresses IP.
- Installation et configuration de *HostAP* pour configurer le point d'accès wifi sur la Raspberry Pi.
- Configuration du service DHCP pour attribuer automatiquement des adresses IP aux périphériques connectés à la Raspberry Pi.
- Configuration du fichier "Interfaces" pour gérer les connexions réseau de la Raspberry Pi.
- Configuration pour permettre aux utilisateurs connectés à la Raspberry Pi d'avoir une connexion internet.

D. Logiciel

1. Environnement et Méthode de travail

La partie logicielle de la Raspberry Pi 4 comprend tout ce qu'elle doit effectuer comme traitement (prise de photos, fusion d'images, partage de flux vidéo sur un réseau, etc.) et offre à l'utilisateur diverses fonctionnalités pour répondre à ses besoins. Nous allons examiner cette partie en la décomposant en sous-parties et en montrant l'ordre chronologique d'exécution des tâches. Dans le développement logiciel général de la Raspberry Pi 4, nous avons choisi d'utiliser *Python,* car le *backend* était déjà en *Python*, ce qui nous permet de travailler dans un environnement de développement commun pour l'ensemble du projet.

Il est important de préciser que nous avons commencé le projet avec seulement deux Raspberry Pi, puis nous en avons utilisé un seul par la suite. Il était donc nécessaire de partager les ressources entre les membres de l'équipe. Au début, les Raspberry Pi étaient principalement utilisés pour les tâches liées aux réseaux et au matériel, il fallait donc développer la partie logicielle sur nos ordinateurs personnels.

2. IA identification d'objet

Tout d'abord, nous avons commencé par rechercher *l'IA* qui conviendrait le mieux à nos besoins. L'IA choisie était *yoloV5**.

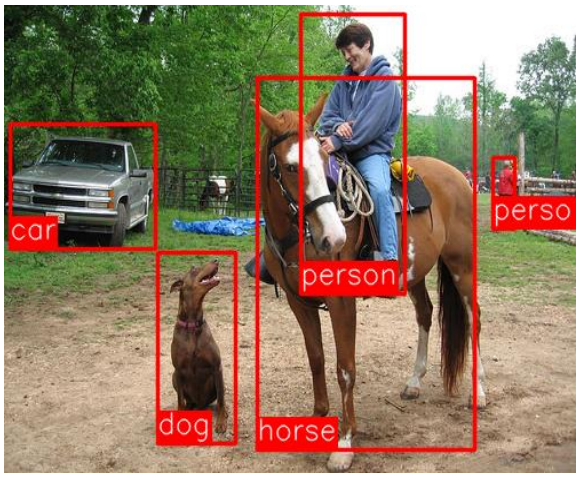


Figure 11 - Identification d'objet Yolov5

Notre travail a consisté à écrire du code *Python* qui permet à partir d'une image de renvoyer tous les types d'objets mobiles identifiés. Il est important de préciser que *l'IA* n'est pas extrêmement précise et qu'elle peut parfois mal identifier les objets, mais cela arrive rarement. Cela est compréhensible, car elle n'a pas été entraînée avec tous les types d'objets.

3. Fusion d'image

Une fois que *l'IA* était fonctionnelle, nous avons entamé la fusion des images. Cette tâche était importante, car la caméra thermique achetée ne donnait pas une qualité d'image satisfaisante, souvent floue ou pixelisée (24 x 32 pixels). Nous avons donc eu l'idée de combiner l'image de la caméra normale avec celle de la caméra thermique. Étant donné que la partie matérielle n'était pas encore terminée, nous avons pris des exemples d'images pour trouver un moyen de combiner les deux images.

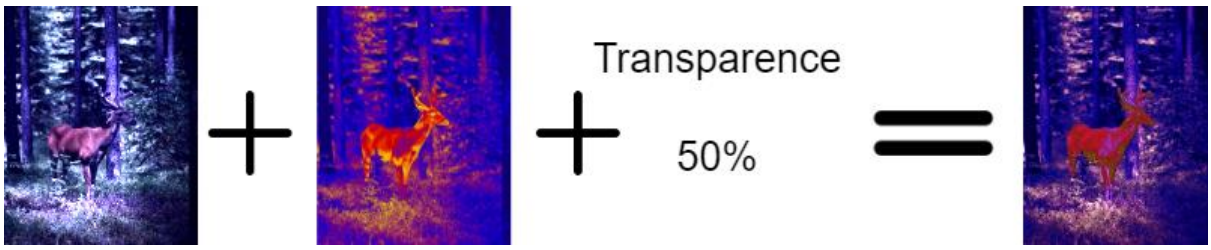


Figure 12 - Illustration de fusion des images

Après un peu de recherche, nous avons trouvé la bibliothèque *OpenCV* qui permet de réaliser divers traitements d'images.

Nous avons donc écrit une fonction en *Python* qui prend en paramètre une image de base et une autre image à superposer, avec un taux de transparence pour cette dernière. Si ce taux est égal à 0, seule l'image de base sera affichée, et s'il vaut 1, seule l'image superposée sera affichée.

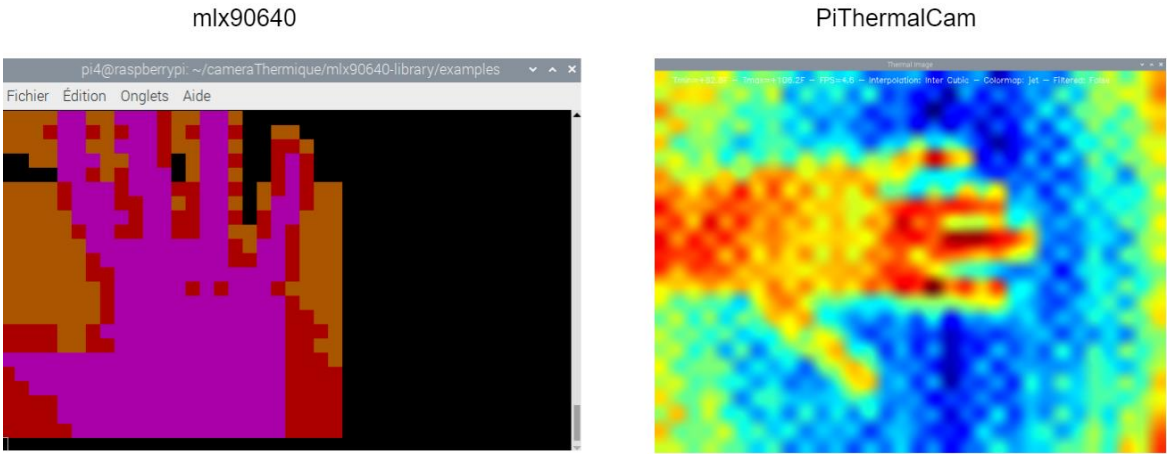


Figure 13 - Comparaison entre mlx90640 / PiThermalCam

4. Caméra Thermique

Pour la caméra thermique, nous avons commencé en utilisant la bibliothèque de base fournie avec l'équipement, qui est MLX90640. Cependant, le résultat de cette bibliothèque était uniquement affiché sur le terminal et ce que nous voulions, c'est avoir une image sauvegardée et pouvoir faire un flux vidéo. Ensuite, nous avons trouvé une bibliothèque PiThermalCam en *Python* qui correspond à deux fonctionnalités différentes : la première permet de prendre une photo, la deuxième permet de partager la vue de la caméra thermique sur un réseau local. Malheureusement, cette bibliothèque ne correspondait pas entièrement à nos besoins, il a donc fallu ouvrir les fichiers Python de cette bibliothèque et les adapter à nos besoins.

5. Merger Les photos

En fin de compte, nous avons obtenu deux images : une image thermique et une image normale. Cependant, les caméras utilisées pour prendre ces images n'ont pas les mêmes caractéristiques (position, angle de vue, résolution).

Il était donc nécessaire de développer un algorithme pour identifier les parties communes des deux images, puis de les utiliser pour effectuer la fusion des images.

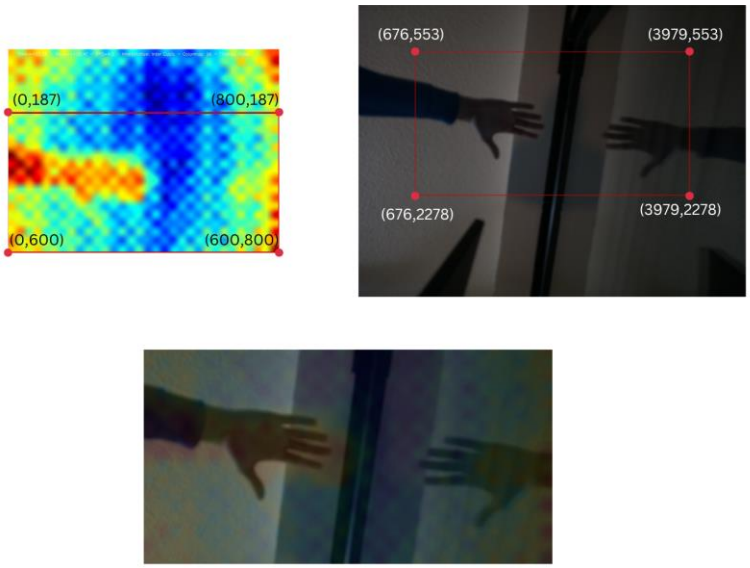


Figure 14- Différence entre les deux caméras et le résultat final

6. Communication avec Serveur

Comme nous l'avons expliqué précédemment dans la section sur les réseaux, la Raspberry Pi reçoit un fichier du serveur qui lui indique quel mode de fonctionnement elle doit utiliser. L'algorithme que nous avons développé est conçu pour détecter les modifications apportées à ce fichier, et en fonction de son contenu, lancer la fonction appropriée. Pour protéger l'accès aux ressources, nous avons mis en place un *sémaphore* de type *Mutex**.

Nous avons aussi utilisé les *sémaphores* pour faire des rendez-vous et synchroniser les actions.

Ceci est un bref résumé de l'algorithme décrit :

```
enChangementFichier(){
  content = lireCOntenueDeFichier("file.txt")
  si unThreadDejaTrainDeFonctionner
     kill(thread)
     semV(jeton)
  si contenueDeFichier == "fonctionnalite1"
     semP(jeton) // il décrémente le nombre de jetons ou il s'endort
     fonctionnalite1()
     semV(jeton) // il incrémente le nombre de jetons ou il réveille celle qui est
endormi
  sinon
     si contenueDeFichier == "fonctionnalite2"
     semP(jeton) // il décrémente le nombre de jetons ou il s'endort
     fonctionnalite2()
     semV(jeton) // il incrémente le nombre de jetons ou il réveille celle qui est
endormi
}

fonctionnalite2(){
  /*traitement*/
}

fonctionnalite1(){
  /*traitement*/
}

jeton = 1
notifierEnChangementFichier("file.txt",enChangementFichier) //appel de fonction
enChangement si le contenu du fichier a changé
```

(L'algorithme utilisait pour la communication avec le serveur)

E. Application Web : Back - end

Le back-end est la partie cachée d'une application web ou d'un système qui s'occupe des tâches de traitement et de gestion des données. Il est composé de plusieurs parties (base de données, l'application, le serveur).

1. Base de données :

Nous avons choisi d'utiliser *Mariadb* comme système de gestion de bases de données pour notre application. En utilisant *Mariadb,* nous avons été en mesure de stocker les données utilisateurs importants ainsi que leur image associée. Référez vous à l'annexe pour voir notre MCD.

Pour l'installation et l'utilisation, il suffisait d'installer les paquets, de créer une nouvelle *database* et un utilisateur. Pour la création des tables, ce sera l'application qui va les créer directement. Vous pouvez retrouver les tables « *user* » et « *DataPicture* » dans l'annexe, dans le fichier model.py.

2. Application :

Pour développer notre application, nous avons utilisé *Flask. Flask* est un *framework web* pour *Python,* qui permet de créer des applications web rapidement, facilement, il est léger et flexible. De plus, *Flask* est compatible avec de nombreuses extensions qui peuvent être utilisées pour ajouter des fonctionnalités supplémentaires à notre application, comme la gestion de l'authentification, la gestion de la base de données et bien d'autres.
Vous retrouverez dans l'annexe un exemple d'application simple avec flask

L'utilisation de *Flask* est simple et intuitive. Pour démarrer, il suffit d'initialiser l'application en définissant un point d'entrée pour l'application. Ensuite, nous pouvons définir les différentes fonctions qui doivent être exécutées lorsqu'un certain *URL* est donnée à l'application en utilisant les décorateurs @app.route.

```python
@views.route('/',methods=['GET'])
def acceuil():
  return render_template("acceuil.html",user=current_user)

@views.route('/galerie',methods=['GET'])
@login_required
def galerie():
  idUser = current_user.id
  result = db.session.query(DataPicture).filter_by(user_id=idUser)
  return render_template("galerie.html",result=result,user=current_user)
```

(Exemple de code *Flask*, qui définit quelle fonction utilisée en fonction de *l'URL*)

Flask possède plusieurs paquets, dont *Flask-login*. Avec *Flask-Login*, il est facile de gérer les utilisateurs authentifiés et les utilisateurs non authentifiés en utilisant des fonctions prédéfinies de *l'API*, telles que *login_user()* et *logout_user()*. Il suffit d'initialiser *Flask-Login* en définissant comment différencier les utilisateurs et en définissant les fonctions à exécuter pour l'authentification et la déconnexion. Cependant, pour comprendre comment *Flask-Login* fonctionne exactement, il est nécessaire de consulter sa documentation et de comprendre les concepts de sessions et d'authentification en général. Il est également utile de comprendre comment *Flask* gère les requêtes et les réponses *HTTP* et comment les *cookies* sont utilisés pour stocker les informations de session.

Vous retrouverez dans l'annexe, dans le fichier __init__.py l'initialisation de *Flask-login.*

Pour pouvoir utiliser les données stockées dans notre base de données depuis notre application, nous avons choisi d'utiliser le paquet *Flask-SQLAlchemy*. Il s'agit d'un module d'extension pour *Flask* qui facilite la communication entre notre application et notre base de données *MariaDB*.

Flask-SQLAlchemy est basé sur *SQLAlchemy*, un *ORM (Object-Relational Mapper)* populaire pour Python qui permet de gérer les bases de données en utilisant des objets Python. Il permet de définir des modèles de données qui correspondent aux tables de notre base de données, et de manipuler ces modèles pour effectuer des opérations sur les données. Avec *Flask-SQLAlchemy,* nous avons pu créer notre base de données en définissant les modèles de données en utilisant des objets Python. Il nous a également permis d'effectuer des *requêtes SQL* avec ces mêmes objets, sans avoir à écrire du code *SQL* brut. Cela nous a permis de maintenir notre code plus propre et plus facile à maintenir.

Vous retrouverez dans l'annexe, dans le fichier views.py des demandes utilisateurs où l'on demande des données dans la base de données.

3. Serveur :

Nous avons choisi d'utiliser *Apache* pour notre serveur web, car c'est l'un des systèmes de serveur les plus utilisés et les plus fiables dans l'industrie.

Cependant, pour pouvoir utiliser notre application développée en *Python*, nous avons dû installer et utiliser un module d'*Apache appelé WSGI (Web Server Gateway Interface)*. *WSGI* est une spécification qui définit comment les applications Python peuvent communiquer avec les serveurs web pour gérer les requêtes et les réponses. Nous avons juste eu besoin de modifier le fichier de configuration d'*Apache* afin qu'il lance notre application.

```
ServerAdmin webmaster@localhost
        #DocumentRoot /var/www/htm
        WSGIScriptAlias / /var/www/html/flask/app.wsgi
        <Directory /var/www/html/flask>
        Order allow,deny
        Allow from all
        </Directory>
```

(Code qui a été ajouté dans le fichier de configuration d'*apache*)

F. Application Web : Front-end

Pour notre projet, la partie front-end est cruciale, car c'est cette partie qui est directement visible et interactive pour l'utilisateur final. Elle est responsable de la création de l'interface utilisateur, qui est le point de contact entre l'utilisateur et le système. Il est donc important que l'interface soit intuitive, facile à utiliser et à comprendre pour les utilisateurs. Un bon front-end permet également d'améliorer l'expérience utilisateur en rendant le système plus efficace et plus agréable à utiliser. En ce qui concerne les technologies, nous avons utilisé *HTML*, *CSS* et *Bootstrap*.

HTML (Hypertext Markup Language) est utilisé pour créer la structure de la page, en définissant les différents éléments de la page tels que les titres, les paragraphes, les images, etc.

CSS (Cascading Style Sheets) est utilisé pour définir la mise en forme des éléments HTML, en définissant les styles tels que les couleurs, les tailles, les polices, etc.

Bootstrap est une bibliothèque de styles *CSS* prédéfinis qui permet de créer des interfaces utilisateur modernes et réactives rapidement. Il fournit un ensemble de classes *CSS* prédéfinies qui peuvent être utilisées pour créer des boutons, des formulaires, des grilles, etc.

Grâce à ces langages, nous avons pu créer une interface qui est graphiquement agréable et facile à utiliser. *Bootstrap* nous a également permis de créer une interface responsive, c'est-à-dire qui s'adapte à la taille de l'écran de l'utilisateur, ce qui est important, car de plus en plus d'utilisateurs accèdent à internet via des appareils mobiles.

Il est important de noter que la partie front-end ne se limite pas simplement à la création d'une interface utilisateur belle et fonctionnelle, mais il est également important pour elle de fournir un accès à toutes les fonctionnalités de l'application. Pour faire une réelle différence dans l'expérience utilisateur, il est essentiel que la partie front-end soit en mesure de travailler en étroite collaboration avec la partie back-end pour fournir un accès aux données et aux fonctionnalités de l'application de manière transparente et efficace.

La page *d'accueil* est le point d'entrée de l'utilisateur, elle permet à ce dernier de comprendre brièvement ce que notre caméra peut faire, les différentes fonctionnalités qu'elle offre, et comment utiliser notre caméra.

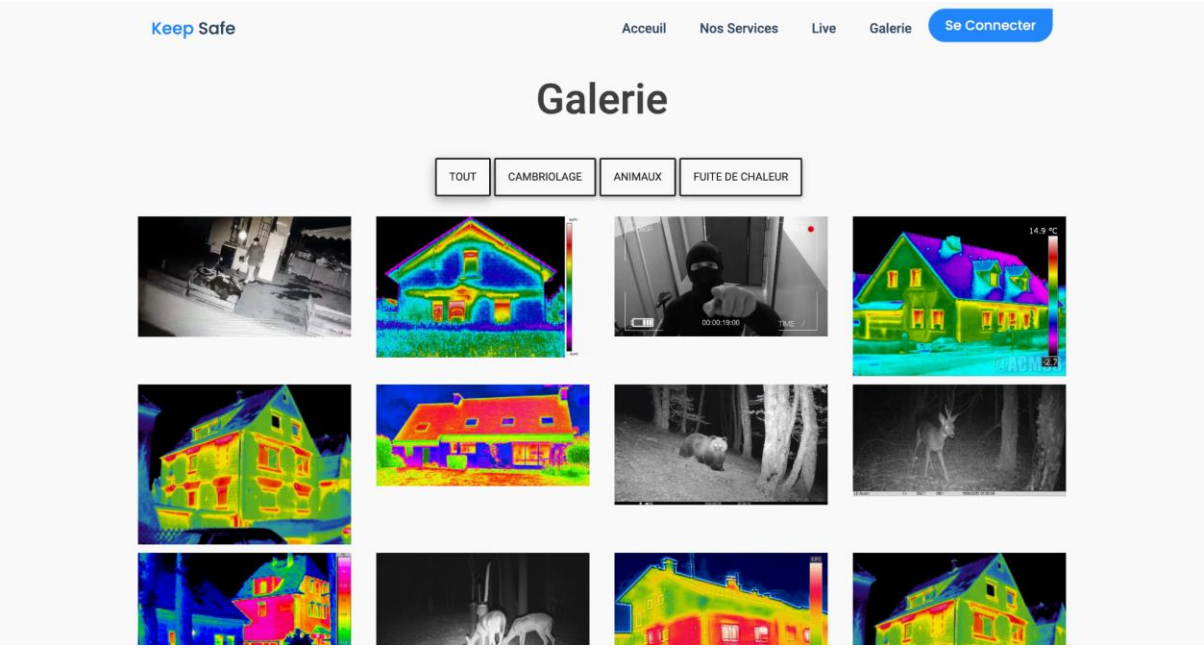


Figure 15 - Page galerie

La partie *galerie* est utilisée pour référencer toutes les photos prises par notre caméra, en les catégorisant en différentes sections, cela permet à l'utilisateur de facilement naviguer et de trouver les photos qu'il cherche.

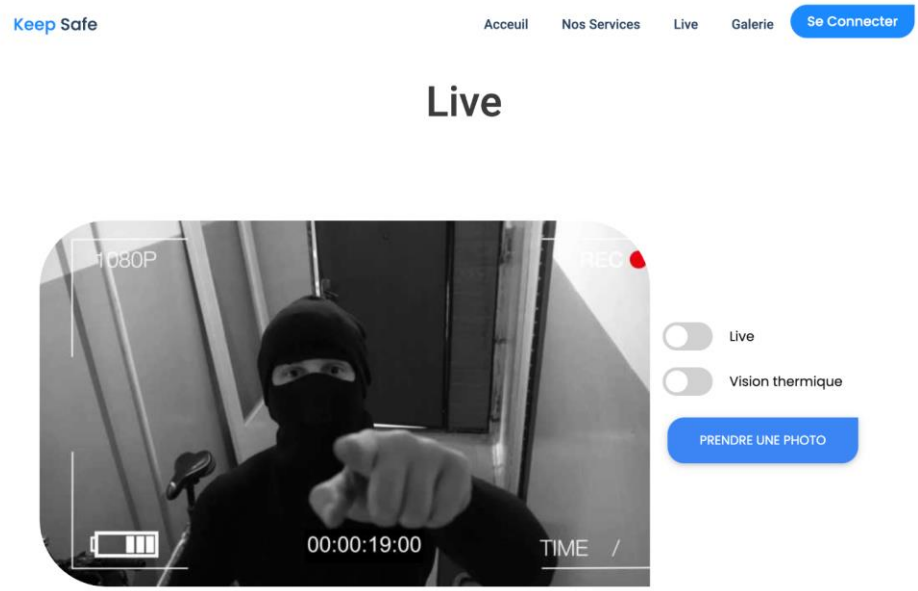


Figure 16 - Page live

La partie *live* est la fonctionnalité la plus interactive de notre caméra, elle permet à l'utilisateur de regarder en direct ce qui est filmé par notre caméra, cela permet à l'utilisateur de voir les animaux en direct, et de ne pas manquer un instant précieux.

La partie *se connecter* permet à l'utilisateur de se connecter à son compte personnel, une fois connecté, l'utilisateur peut accéder à toutes les fonctionnalités de notre caméra, mais avec plus de flexibilité.

La partie *créer un compte* permet à un utilisateur de créer un compte personnel, cela permet à l'utilisateur de sauvegarder ses photos et ses paramètres, et de personnaliser son expérience sur notre camera.

En résumé, nous offrons une grande variété de fonctionnalités, qui permettent à l'utilisateur de tirer le meilleur parti de notre caméra, en offrant une expérience utilisateur complète et interactive.

III. Conclusion

Notre projet a très bien avancé tout le long. Nous avons créé une application qui est en relation avec une base de données. Cette application est hébergée sur un serveur apache. Les utilisateurs pourront donc y aller afin de recevoir les différentes pages web réalisées.

Ils pourront aussi consulter les photos qui ont été prises par un Raspberry Pi qui fait office de caméra. La Raspberry contient plusieurs scripts afin de prendre des photos ainsi que de les travailler avec de l'intelligence artificielle.

Pour pouvoir relier la PI au serveur, nous avons mis en place un service *SSH*, mais nous avons aussi modifié la PI pour qu'elle puisse devenir un point d'accès.

Notre projet était très conséquent, nous n'avons donc pas pu faire toutes les fonctionnalités que nous voulions. Avec un peu plus de temps, nous pouvons implémenter une fonction live ainsi qu'un système de notification.

Cependant, nous sommes assez fiers de notre travail, ce vaste projet nous a permis de découvrir beaucoup de connaissances sur plusieurs aspects du développement informatique.

IV. Résumé en anglais

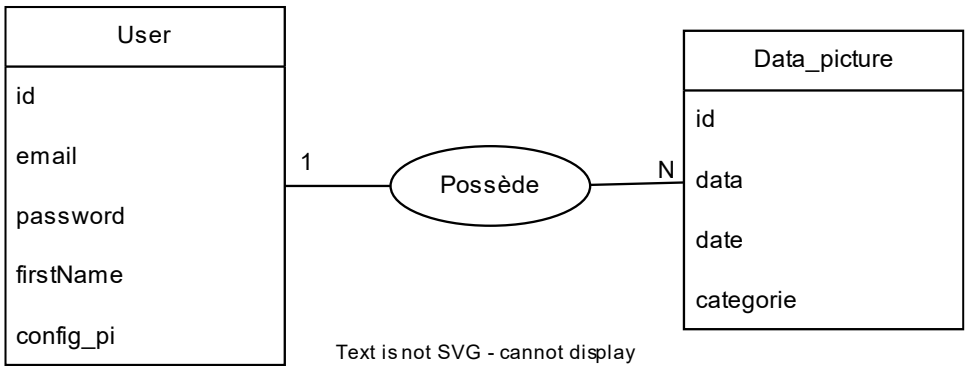
Our project KeepSafe is a surveillance camera system that allows users to effectively monitor a building or other area from a distance. We developed this system to meet the growing need for remote monitoring, especially for people who have properties located far away and cannot be present all the time.

To achieve this, we implemented a variety of features such as the ability to automatically capture photos when motion is detected, the ability to remotely control the camera and initiate captures on demand, and the option to view live footage remotely. This system allows for efficient and effective remote monitoring of a building or other area, and can be used for a variety of purposes such as wildlife monitoring or security surveillance.

In conclusion, KeepSafe is a powerful and flexible surveillance system that allows users to monitor their properties or other areas remotely with ease. With its advanced features and capabilities, it is an ideal solution for anyone looking for a reliable and effective way to keep an eye on their property or other areas.

Annexe :

MCD de notre base de données:



Exemple d'application simple avec Flask :

- **Fichier : views.py**

```
@views.route('/',methods=['GET'])
def acceuil():
  return render_template("acceuil.html",user=current_user)

@views.route('/galerie',methods=['GET'])
@login_required
def galerie():
  idUser = current_user.id
  result = db.session.query(DataPicture).filter_by(user_id=idUser)
  return render_template("galerie.html",result=result,user=current_user)
```

- **Fichier : __init__.py**

```
db = SQLAlchemy()

def create_app():
  app = Flask(__name__)
  app.config['SECRET_KEY'] = 'toto'
  app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql+pymysql://keepsafe:toto@localhost:3306/KeepSafe'
  db.init_app(app)

  from views import views

  from auth import auth

  app.register_blueprint(views,url_prefix='/')
  app.register_blueprint(auth,url_prefix='/')

  from models import User
  #create_database(app) Pour créer la database la première fois

  login_manager = LoginManager()
  login_manager.init_app(app)
  login_manager.login_view = 'auth.login'

  #Initialisation de Flask login pour le cookie
  @login_manager.user_loader
  def load_user(user_id):
    return User.query.get(int(user_id))

  return app

app = create_app()
```

- **Fichier : model.py**

```
class DataPicture(db.Model):
  id = db.Column(db.Integer,primary_key=True)
  data = db.Column(db.String(100))
  date = db.Column(db.DateTime(timezone=True),default=func.now())
  categorie = db.Column(db.String(50))

  #Foreign Key of User
  user_id = db.Column(db.Integer,db.ForeignKey('user.id'))


class User(db.Model, UserMixin):

  id = db.Column(db.Integer, primary_key=True)
  email = db.Column(db.String(100), unique=True)
  password = db.Column(db.String(100))
  firstName = db.Column(db.String(100))
  config_pi = db.Column(db.String(100))

  pictures = db.relationship('DataPicture')
```

Définitions :

*__yoloV5__ : est une famille de modèles de détection d'objets à échelle composite formés sur le jeu de données COCO, et inclut des fonctionnalités simples pour l'augmentation de temps de test (TTA), l'assemblage de modèles, l'évolution des hyperparamètres et l'exportation vers ONNX, CoreML et TFLite.

*__Mutex__: c'est un acronyme pour "mutual exclusion", c'est un type de sémaphore qui s'utilise pour garantir qu'un seul processus ou thread accède à une ressource à la fois.

Schéma de la structure de notre projet :