SAE 501 et 601 VCOD - Analyse et conception d'un outil décisionnel

MOULIN Vincent

BUT 3A VCOD

Table des matières

I- Introduction

Avec l'avènement des objets connectés, de nombreuses technologies ont émergé, soulevant plusieurs problématiques, comme la gestion des données en temps réel et comme la surveillance des espaces. L'un de nos défis majeurs est de pouvoir surveiller une salle et détecter toute intrusion en temps réel de manière fiable et efficace.
Dans ce contexte, notre projet vise à développer une solution de surveillance combinant différents capteurs et technologies. Ce système permettra de détecter des anomalies et d'alerter les utilisateurs en cas d'intrusion ou de situation critique. Nous avons donc choisi d'aborder ce sujet avec cette problématique.
Comment concevoir un système de surveillance en temps réel d'une pièce, capable de détecter les intrusions ?
Pour répondre à cette problématique, nous commencerons par présenter le cahier des charges du projet, puis nous détaillerons son organisation et les différentes phases de développement. Nous aborderons ensuite l'architecture, détaillent les capteurs utilisés et les différentes interfaces utilisateur. Par la suite, nous explorerons les protocoles de communication adoptés, la base de données mise en place. Et finalement, nous présenterons les différents tableaux de bord permettant la visualisation des données et décrirons les différentes perspectives d'amélioration du projet.

A. Contexte

Dans cette SAE, notre objectif initial était de concevoir un projet reposant sur plusieurs stations interconnectées, chacune ayant un rôle spécifique dans le fonctionnement du système. L'objectif était de développer une solution intégrant différents procéder afin de capturer les données, les traiter, les stocker et les exploiter. Pour cela, nous avons pu nous appuyer sur nos différentes connaissances ainsi que celle acquises dans les divers cours.
Par manque de temps nous avons choisi de basé notre solution sur une seule station de captation, avec un Raspberry Pi avec divers capteurs.

II- Projet

A- Phases

Pour ce projet, nous avons suivi plusieurs cours et exercices pratiques, nous permettant d'acquérir différentes compétences nécessaires à sa réalisation. Le projet s'est déroulé en plusieurs phases, organisées de manière de nous familiariser avec les différentes technologies utilisées.

- **Phase 1 : Veille technique** – Cette première étape consistait à effectuer des recherches sur les technologies et outils lié au problématique de big data et des objets connectés.
- **Phase 2 : Station capteur et génération de données simulé** – Dans ce second temps nous avons travaillé sur la mise en place d'un script python permettant de générer des données de capteurs simulé afin de créer un premier tableau de bord avec Power Bi. Dans un même temps nous avons commencé à travailler sur la station de capteurs, afin de mettre en place les différents scripts et connecté les capteurs pour récupérer des données.
- **Phase 3 : Kafka** – Cette phase nous a permis d'aborder Kafka, un outil de gestion de flux de données, afin de mieux comprendre son fonctionnement et ses applications dans ce projet.
- **Phase 4 : Intégration des technologies (GitHub, MQTT, Base de données, Dashboard)** – lors de cette dernière phase nous avons pu mettre en place la gestion des données récupérer ainsi que de les analyser et visualiser avec l'utilisation de différentes technologies.

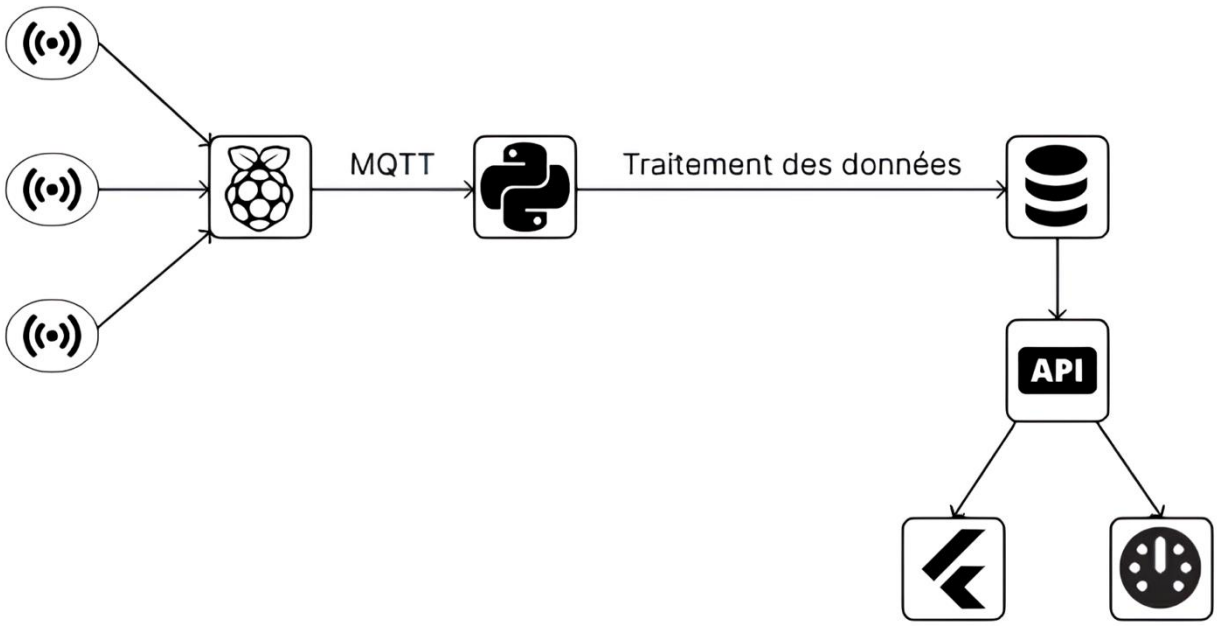
Les premières phases nous ont donc permis de nous familiariser avec les différents concepts et outils, tandis que la dernière phase a été consacrée à la mise en place du projet

B- GitHub

Lors de ce projet GitHub s'est révélé être un outil utile tout au long de notre projet, car il nous a permis de travailler sur plusieurs aspects du projet en parallèle. En utilisant GitHub, chacun a pu se concentrer sur des tâches spécifiques, comme la création des Dashboard ou de l'API, sans risquer de conflits ou de perturbations dans le travail de l'autre en s'assurant d'avoir le même environnement de travail. Malheureusement, nous avons remarqué qu'il aurait plus utile de mettre en place GitHub dès le début du projet.

C- Schéma

Notre projet repose sur une architecture permettant de collecter, traiter et exploiter des données issues de capteurs connectés. Le Raspberry Pi, permet grâce aux différents capteurs de récupérer les données d'une pièce et de détecter les mouvements de celle-ci, ces capteurs mesurent différentes informations, comme la température, l'humidité, le niveau sonore et la qualité de l'air. Une fois les données collectées, elles sont transmises via le protocole MQTT à un ordinateur qui traitera ces données en Python.
Le traitement des données consiste à analyser avec la reconnaissance faciale, et à formaliser les données afin de le stocker dans la base de données. Cette base regroupe toutes les données envoyées par le Raspberry Pi ainsi que les données de la reconnaissance faciale.
Afin de rendre ces données accessibles et exploitables pour les deux tableaux de bord, une API a été mise en place. Celle-ci permet de récupérer les informations stockées dans la base et de transmettre aux différentes interfaces. Nous avons choisi de développé deux tableaux de bord un sur mobile avec une interface réduite afin d'avoir les informations principales, et l'autre sur un ordinateur offrant une interface web pour visualiser les différentes informations.
Grâce à cette architecture choisie, nous avons pu concevoir un système modulaire et évolutif. Ce schéma représente ainsi l'architecture que nous avons choisie



III- Architecture

Pour ce projet nous avons utilisé un Raspberry Pi 3 avec Twister OS. Le Raspberry Pi est un élément central de notre projet, car il permet de gérer les capteurs en collectant et transmettant des données en temps réel. Son faible coût, sa faible consommation énergétique et sa flexibilité en font un choix idéal pour des applications embarquées. De plus, avec l'utilisation de HAT ceci nous permet d'ajouter des capteurs ou des modules facilement.

A- Capteurs

Dans notre projet, plusieurs capteurs ont été utilisés pour surveiller l'état d'une pièce et s'assurer qu'elle reste dans des conditions saines. Chaque capteur joue un rôle spécifique dans l'analyse et en cas de détection d'une anomalie, on augmente une variable état afin d'indiquer le niveau de l'état critique de la pièce à l'utilisateur.

- **Grove DHT Sensor (Température et Humidité)**
- **Grove Sound Sensor (Niveau sonore)**
- **Grove Air Quality Sensor (Qualité de l'air)**
- **Grove Move Sensor (Détection de mouvement)** : Ce capteur est utilisé pour détecter des mouvements dans la pièce. Il est essentiel pour repérer d'éventuelles intrusions. Lorsqu'un mouvement est détecté, la variable état est incrémenté afin d'alerter l'utilisateur d'une activité inhabituelle. Dès qu'un mouvement est enregistré, une caméra est activée pour capturer l'image de la personne entrant dans la pièce. Nous utilisons alors un algorithme de reconnaissance faciale afin d'identifier l'individu.

Tous les capteurs permettent d'évaluer l'état de la pièce en fonction de seuils définis. Lorsqu'un capteur détecte une valeur en dehors des limites considérées comme saines, la variable état est incrémenté afin d'alerter l'utilisateur sur une éventuelle risque et dégradation. Grâce à ces capteurs, notre système permet une surveillance en temps réel de l'état de la pièce.

B- Sorties

Dans un premier temps pour donner une visualisation rapide à l'utilisateur de ces données nous avons intégré plusieurs éléments lumineux et sonores.

Le LedStick est utilisé pour afficher différentes informations. Une LED est allumée en fonction de la saison (la saison modifie les seuils des différents capteurs), les autres LED varient en fonction de l'état, afin de représenter à quelle niveau et l'état critique d'une pièce et celle varie en fonction de ci les seuils sont en dessous ou au-dessus. Nous avons également intégré les boutons LED, qui permettent de régler la luminosité du LedStick.

Le buzzer, quant à lui, est activé lorsque la variable état atteint un niveau trop élevé. Il sert d'alerte sonore pour signaler que la pièce n'est plus saine.

Puis, nous avons décidé de ne plus utiliser les sorties du Raspberry Pi. Le Raspberry Pi a été dédié à la station de captation et à la gestion des intrusions. Il est ainsi chargé de récupérer les données des capteurs et de prendre des photos en cas de détection de mouvement suspect.

IV- Reconnaissance Faciale

A- Problème Juridique

Dans ce projet, la gestion des images nous en mener à plusieurs questions comme celle des données personnelles. L'utilisation de la reconnaissance faciale implique la capture et l'analyse d'images contenant donc des informations sensibles, ce qui donne des problèmes de confidentialité et de protection des données. Avec ces enjeux, nous avons choisi de nous informer sur les recommandations de la CNIL afin de garantir le respect de ces données personnelles.

B- Mise en place

Lorsque le capteur de mouvement détecte une présence, une photo est capturée lorsque l'appareil a pu effectuer son cadrage ceci avec la caméra du Raspberry Pi. Cette image est ensuite encodée en base64 et transmise via le protocole de communication. Une fois reçue par l'autre ordinateur le programme de reconnaissance faciale s'exécute, elle est alors décodée, stockée et analysée à l'aide de la bibliothèque FaceNet en Python, qui est spécialisée dans la reconnaissance faciale basée sur l'extraction d'embeddings vectoriels (représentation numérique unique d'un visage sous forme de vecteurs, permettant de comparer et d'identifier les individus).

1) FaceNet

Nous avons choisi cette bibliothèque qui est un modèle de reconnaissance faciale développé par Google qui repose sur l'apprentissage pour générer des embeddings vectoriels de visages. Contrairement aux autres FaceNet convertit chaque visage en un vecteur de caractéristiques unique, ce qui permet de comparer les visages avec une grande précision et un cout faible.

Par rapport à d'autres bibliothèques comme DeepFace, FaceNet se distingue par une précision plus élevée et meilleure rapidité d'exécution. DeepFace, est souvent plus lent et plus gourmand en ressources. FaceNet, est lui moins gourmand. De plus, il fonctionne avec de base de données de n'importe quelle taille.

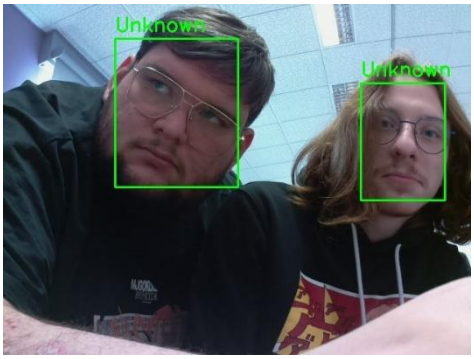
En résumé nous avons choisi FaceNet comme solution pour ces différents avantage comparé à d'autre bibliothèque telle que DeepFace.

2) Reconnaissance Faciale

Dès lors qu'un mouvement et que l'image est reçue FaceNet extrait alors l'image vectorielle, qui représente les traits caractéristiques du visage détecté ceci permet d'obtenir une signature numérique du visage, qui est ensuite stockée dans la base de données.

Lorsque le programme tente d'identifier un visage, il compare alors le nouvel embedding généré avec ceux enregistrés dans la base. Si un seuil de similarité est atteint, la personne est reconnue et son nom associé est affiché. Dans le cas contraire, le visage est considéré comme "Unknown" et devra être manuellement être modifier le nom par un administrateur dans la base de données pour être reconnu lors des prochaines analyses.

Une fois l'analyse terminée, une image annotée est générée. Sur cette image, un carré vert est affiché autour de chaque visage détecté. Si le visage correspond à une personne déjà enregistrée, son nom est inscrit au-dessus du cadre. Si non "Unknown" sera affichée.



Le programme permet de reconnaitre plusieurs visages simultanément. Si plusieurs personnes sont présentes

sur la même image, chaque visage est analysé et comparé à la base de données. Ceci permet de détecter si une intrusion avec plusieurs personnes.

En utilisant ces différentes procéder, notre système assure une surveillance en temps réel, permettant d'identifier les intrus et de suivre les entrées dans une pièce.

V- Protocole de communication

A- Kafka

Au cours de ce projet nous avons pu voir et apprendre à utiliser Kafka. Apache Kafka est une plateforme de messagerie distribuée conçue pour gérer des flux de données. Celle-ci repose sur un système de publication/souscription où des producteurs envoient des messages à des topics, et des consommateurs récupèrent ces messages. L'un des principaux avantages de Kafka est sa capacité à traiter un grand volume de données de manière efficace et fiable, en garantissant un stockage persistant des messages et une tolérance aux pannes grâce à son architecture distribuée (en 3 clusters). Contrairement à d'autres protocoles, Kafka est particulièrement adapté aux systèmes demandant une scalabilité, car il permet de se répartir sur plusieurs serveurs.

L'un des atouts majeurs de Kafka est sa gestion des files d'attente, qui permet aux consommateurs de lire les messages, pouvant voir les messages du début du topic. De plus, Kafka offre une réplication automatique, ce qui permet une disponibilité en cas de panne. Il est souvent utilisé dans les domaines du Big Data, et l'analyse en temps réel.

Mais malgré ses nombreux avantages, Kafka est une technologie complexe à mettre en place. En raison du manque de temps et de la complexité d'intégration dans notre projet, nous avons choisi s'orienter sur le protocole MQTT.

B- MQTT

Nous avons choisi MQTT pour la communication entre notre station de captation et notre base de données en raison de sa simplicité et de sa légèreté. Ce protocole est adapté aux objets connectés et aux systèmes nécessitant une

transmission de données en temps réel. Son fonctionnement repose sur un modèle de publication/souscription, où un broker centralise les échanges entre les capteurs et les clients abonnés aux topics correspondants.
L'un des principaux avantages de MQTT est sa facilité de mise en place, ce qui nous a permis de l'intégrer rapidement à notre projet.

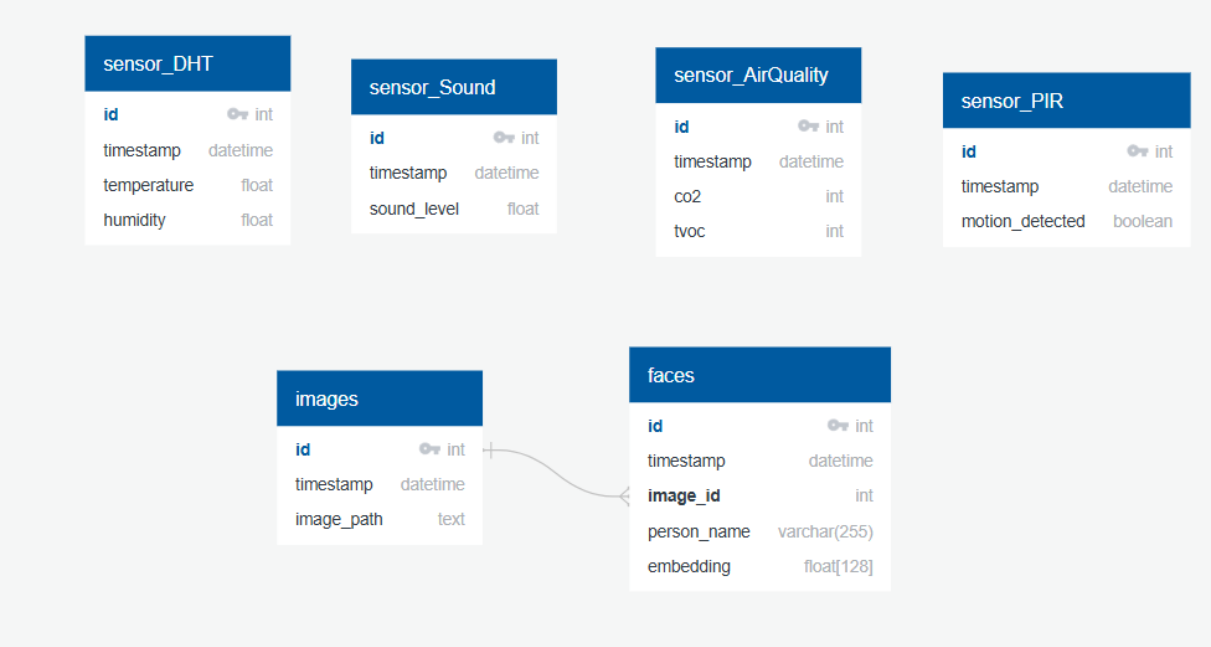
C- Mise en place

Pour la mise en place de la communication entre le Raspberry Pi et la base de données nous avons commencé par l'installation de mosquitto, un broker MQTT qui permet de gérer la communication, mosquitto fonctionne comme un serveur.

Pour la réception de ces données et l'envoie, nous avons pu utiliser la bibliothèque paho-mqtt en Python, qui nous permet d'envoyer et de recevoir facilement des messages MQTT via des topics. Nous avons fait un code client MQTT qui récupère les données en continue des différents topics liés aux capteurs et aux images envoyées par la station de captation afin de les stocké dans la base de données.

VI- Base de données

A- Schéma



Notre base de données est conçue pour stocker les informations de nos capteurs ainsi que les résultats de la reconnaissance faciale. Elle est structurée en plusieurs tables, chacune ayant un rôle.
Les tables sensor_DHT, sensor_Sound, sensor_AirQuality et sensor_PIR sont dédiées aux capteurs que nous utilisons pour surveiller une pièce et ces données seront associès avec un timestamp générer via un trigger sur les insertions de données permettant d'obtenir la date actuelle.

La table images sert à stocker les chemins d'accès aux images capturées lors des détections de mouvement. Chaque image enregistrée dans le système est aussi associée à un timestamp.
Enfin, la table faces est pour la reconnaissance faciale. Elle contient les identifiants des visages détectés, ainsi que l'image_id qui fait référence à la table images, permettant ainsi de savoir quelle image a été utilisée pour une détection donnée. Le champ person_name permet d'identifier la personne reconnue, tandis que le champ embeddingstocke la signature numérique du visage sous forme d'un vecteur généré par FaceNet.

Nous avons choisi de minimiser le nombre de tables et de colonnes dans notre base de données afin d'optimiser l'espace de stockage et d'éviter d'enregistrer des données inutiles. Les capteurs envoient des mesures toutes les 30 secondes, ce qui signifie un grand volume de données est envoyé. Pour éviter cela, nous avons limité le stockage aux informations essentielles et avons exclu les données inutiles ou calculables à partir des valeurs déjà enregistrées. Cette approche nous a permis d'éviter les données inutiles de la base tout en conservant les informations nécessaires pour notre analyse en temps réel.

B- Postgre SQL

Nous avons choisi PostgreSQL comme système de gestion de base de données pour notre projet, principalement pour sa capacité à gérer des données en temps réel.

L'autre raison de notre choix et l'avantage de PostgreSQL de sa capacité à gérer un grand volume de données et à exécuter des requêtes complexes rapidement.

C- API

Dans ce projet nous avions eu la nécessiter de créer une API car dans le Dashboard mobile nous n'avions pas d'autre choix que de passer une API afin de récupérer les données. Une API est un intermédiaire qui permet aux différentes

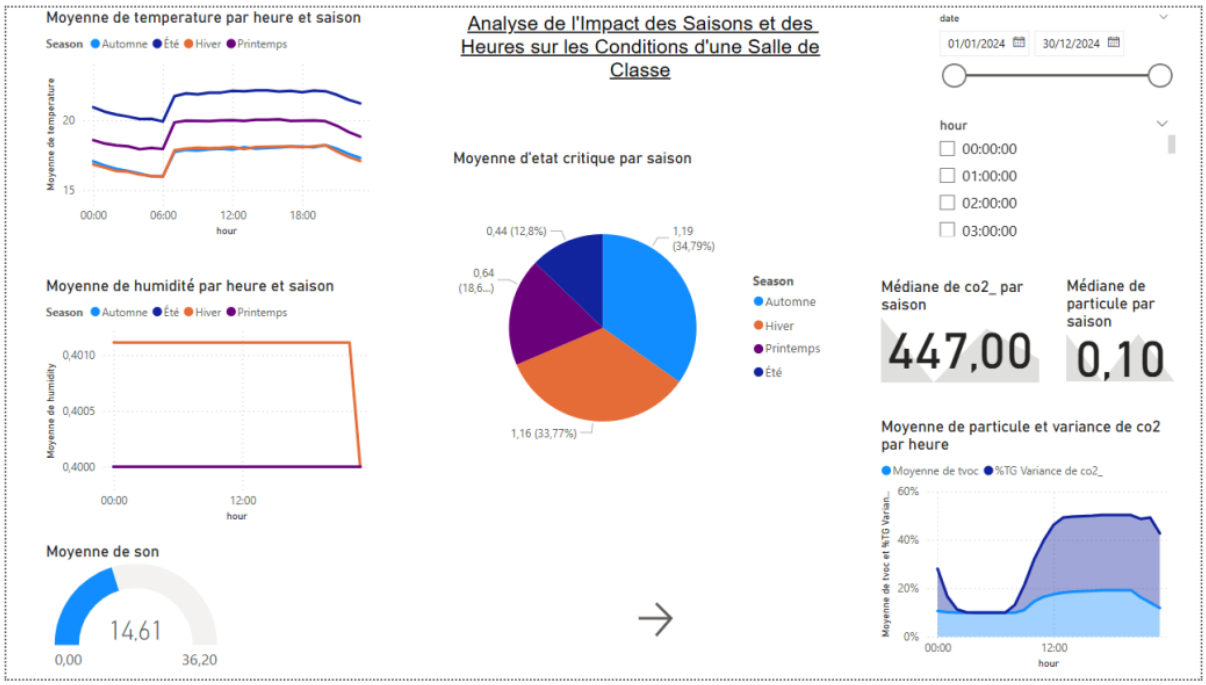
applications de communiquer entre eux. Elle définit un ensemble de règles et de méthodes permettant d'accéder aux données.

Dans notre projet, nous avons développé l'API avec Flask afin de récupérer les données enregistrées dans notre base PostgreSQL. Cette API permet d'accéder aux mesures de nos différents capteurs ainsi qu'aux autres tables chaque table était alors accessible avec un chemin. Grâce à ces différents chemins, nous pouvons interroger la base de données.

Nous avons choisi d'implémenter uniquement les requêtes GET dans notre API, car nous n'avions pas besoin d'ajouter ou de modifier des données dans un premier temps. Notre besoin principal était la consultation des informations enregistrées, et l'utilisation des requêtes GET était donc suffisante pour notre d'utilisation.

VII- Dashboard

A- Power BI



Notre premier tableau de bord a été réalisé avec Power BI, un outil adapté pour l'analyse et la visualisation de données et permet la gestion des données en temps réel. Les avantage de Power BI est sa simplicité d'utilisation et son interface intuitive qui permet de créer des Dashboard dynamiques sans nécessiter de connaissances en programmation.

Au début du projet, nous ne disposions pas encore de données réelles issues de nos capteurs. Pour pouvoir tester notre base de données et commencer nos premières analyses, nous avons donc généré des données simulées qui respectaient les tendances et les variations attendues en fonction des saisons et des heures. Cela nous a permis de construire un premier prototype fonctionnel du tableau de bord.

Pour ce premier tableau de bord nous disposions d'une problématique différente qui était comment influent une saison sur une salle de classe. Nous avons choisi de séparer l'analyse par capteur afin de mieux observer les tendances sur chaque type de mesure. Voici les principaux éléments présents dans le tableau de bord :

- **Température et humidité** : représentées sous forme de courbes permettant de voir l'évolution des mesures au fil des heures et selon les saisons.
- **Qualité de l'air** : affichée sous forme de médianes et de moyennes des niveaux de CO2 et de particules pour chaque saison.
- **Son** : représenté via un indicateur de jauge pour donner une idée rapide du niveau sonore moyen enregistré.
- **État** : un camembert indique la répartition des états critiques en fonction des saisons, ce qui permet d'identifier les périodes à risque et qu'elle saison influent plus sur l'état critique.

Nous avons également ajouté des filtres interactifs, notamment sur la période, l'heure de la journée et la saison, permettant de sélectionner une plage de temps spécifique.

Lors de ce tableau nous avons pu utiliser les requête DAX qui sont une des fonctionnalités de Power BI. Elles permettent de créer des champs calculés et d'effectuer des analyses avancées. Dans notre cas, nous avons utilisé DAX pour déterminer automatiquement la saison en fonction de la date des relevés.

Power BI nous a offert une première approche rapide et efficace pour visualiser les tendances et comprendre nos données. Grâce à celle-ci nous avons pu avoir une idée de comment les analyser et de choisir une meilleure problématique sur les idées que nous faisions de la station de captations.

B- Dash Plotly

Pour la création de notre tableau de bord en rapport avec notre problématique, nous avons choisi d'utiliser Dash Plotly, une bibliothèque Python permettant de concevoir des Dashboard interactifs pour le web. Notre objectif était de créer

un dashboard plus modulable, capable d'évoluer en fonction de nos besoins et offrant une meilleure personnalisation que ce que permet Power BI.

Au départ, nous avions voulions utiliser Tkinter, une bibliothèque native de Python permettant de créer des interfaces graphiques. Cependant, nous avons rapidement constaté ses limites. Tkinter est avant tout conçu pour des applications locales et ne permet pas une personnalisation avancée ni une intégration facile avec des bases de données.

Dash Plotly nous a permis de créer un Dashboard web avec HTML, CSS et Python et d'utiliser Plotly pour générer des graphiques interactifs et esthétiques.

Nous avons choisi d'utiliser psycopg plutôt que l'API car lors de nos premiers essais, nous avons tenté d'intégrer nos données via celle-ci cependant, nous avons remarqué que les temps de chargement étaient trop longs à cause des multiples requêtes envoyées à l'API. Cela créait un délai important pour l'affichage des données sur le dashboard et lors des changements de page.

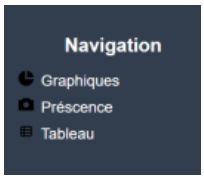
Pour résoudre ce problème de performance, nous avons fait le choix d'interroger directement notre base de données PostgreSQL en utilisant psycopg, une bibliothèque optimisée pour les interactions avec PostgreSQL.

1) Navigation

Pour naviguer entre les différentes pages et avoir les informations nécessaires dans notre Dashboard nous avons intégré une navbar et une topbar.

La navbar permet un accès rapide aux différentes pages nous l'avons conçu pour permettre de passer facilement d'une page à l'autre du dashboard. Cette barre contient trois pages principales.

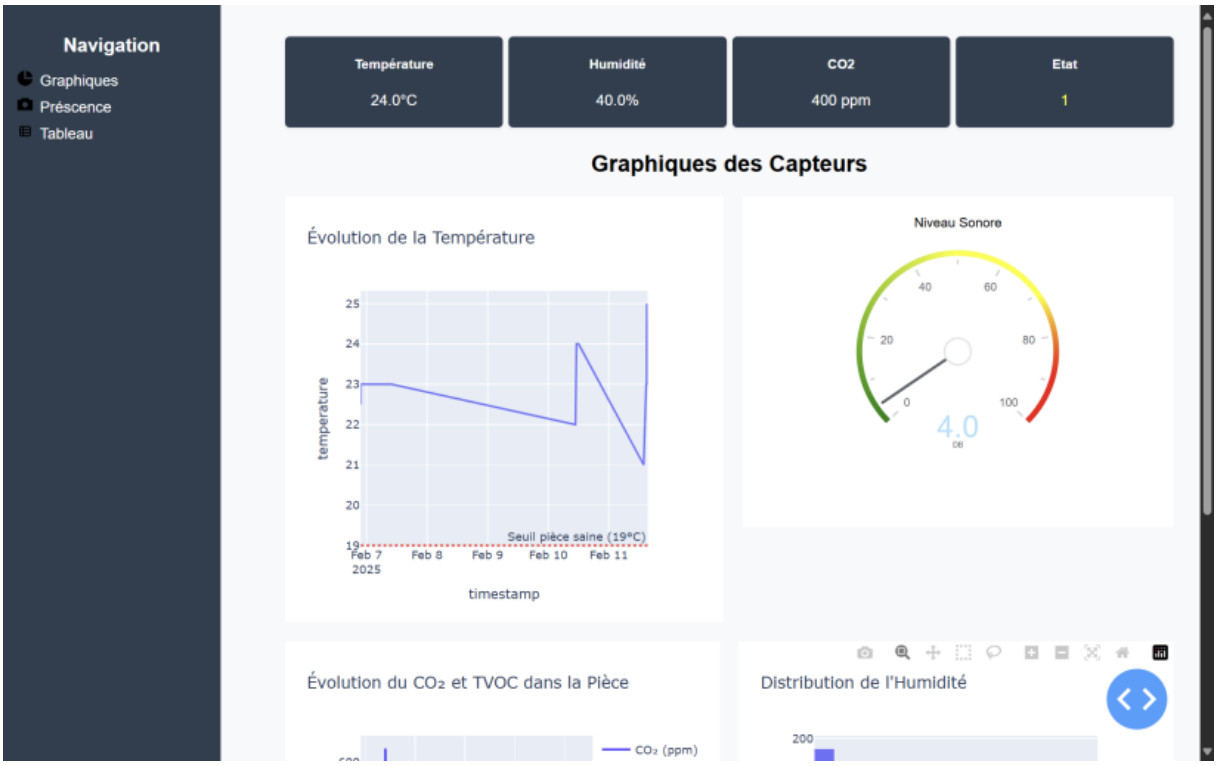
- **Graphiques** : Accès aux visualisations des données des capteurs.
- **Présence** : Accès aux informations sur la détection de présence.
- **Tableau** : Accès aux données sous forme de tableau.



Chaque lien est accompagné d'une icône en svg représentative pour faciliter la navigation.

La topbar permet un aperçu des capteurs celle-ci est une barre d'affichage située en haut de chaque page. Son rôle est de fournir un aperçu rapide des conditions de la pièce mesurées par les capteurs. Elle affiche la température avec une couleur adaptée (Rouge si elle est trop élevée, Bleue si elle est trop basse et Blanche si elle est normale), pour les autres capteurs (humidité, CO2) si la limites d'une pièce saine est dépasser alors les valeurs seront rouge. Un état global de la pièce est affiché celui-ci est calculer avec des seuils sur tous les capteurs et si ceux-ci dépassent les seuils alors l'état est incrémenter. La couleur de la valeur varie en fonction du niveau de l'état critique (blanc, jaune, orange ou rouge). Nous avons choisi de ne pas afficher le tvoc et le son pour que l'utilisateur est les informations principales. Le tvoc n'étant pas compréhensible pour tous les utilisateurs et le son pouvant être retrouver sur la première page.

2) Graphique



Dans notre page graphique, nous avons mis en place plusieurs visualisations pour permettre une analyse claire et détaillée des données environnementales captées en temps réel. Cette page est structurée de manière à offrir un aperçu rapide et efficace des tendances et des variations nous

avons pu reprendre les idées des différents graphiques avec ce qu'on a pu faire sur les premiers Dashboard Power Bi.

Le premier graphique permet de suivre l'évolution de la température, nous avons intégré un graphique en ligne permettant de visualiser les variations de la température en fonction du temps. Une ligne de seuil a été ajoutée à 19°C, indiquant une référence pour une température d'une pièce saine.

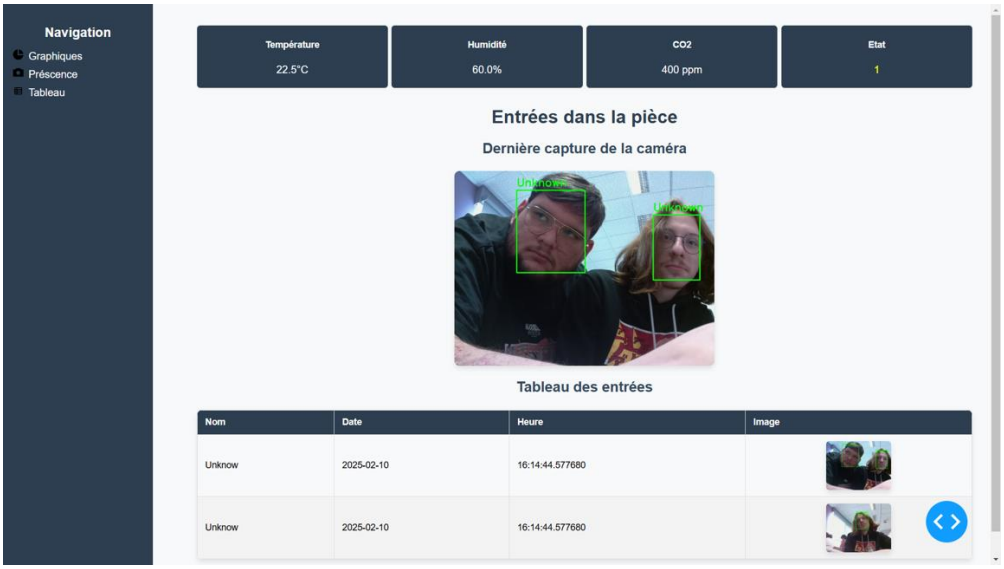
Le deuxième graphique permet de voir le niveau sonore dans une pièce avec une jauge. Pour avoir une bonne visualisation du niveau sonore dans une pièce, nous avons opté pour une jauge interactive pour afficher le niveau sonore. Elle dispose de seuil pour permettre rapidement de voir si le niveau de décibel est correct dans la pièce.

Le troisième graphique permet de voir l'évolution du CO2 et TVOC, ceci est un graphique combiné pour afficher à la fois l'évolution du CO2 et du TVOC dans l'air. Ces deux indicateurs permettent de surveiller la qualité de l'air dans une pièce

Le dernier graphique permet de surveiller l'humidité dans une pièce, nous avons choisi un histogramme pour représenter la distribution des niveaux d'humidité dans la pièce. Ce graphique nous permet de voir la répartition des mesures et d'identifier si l'humidité est généralement dans une plage saine ou si des variations sont présentes.

Nous avons disposé les visualisations en deux lignes afin de rendre la lecture plus fluide et une interface moderne.

3) Présence



Dans cette deuxième page, présence, nous avons conçu une interface permettant de suivre en temps réel les entrées dans la pièce. Cette page permet de surveiller et d'obtenir un historique détaillé des passages.

Au centre de cette page, nous affichons la dernière image capturée par la caméra. Celle-ci met en évidence les visages détectés à l'aide d'un encadrement vert et attribue un nom si une reconnaissance est disponible. Lorsque l'identité est inconnue, le système attribue le statut "Unknown".

En dessous de l'image, nous avons un tableau qui fait un récapitulatif des dernières entrées enregistrées. Ce tableau affiche plusieurs informations le nom (Si la personne est identifiée, son nom apparaît, sinon elle est marquée comme "Unknown"), la date, l'heure et l'image (photo du moment de l'intrusion avec la reconnaissance faciale).

Les éléments sont pour permettre à l'utilisateur de voir directement la dernière intrusion qui a eu lieu et avec l'historique alors il peut constater toutes les entré et sorties de la pièce surveiller.

4) Tableau

Dans cette dernière page, nous avons conçu une interface permettant de visualiser les données issues des capteurs. Cette page permet de suivre les données des capteurs de la pièce afin de surveiller.

Dans cette page, nous retrouvons un tableau qui recense les valeurs des différents capteurs. Chaque ligne du tableau correspond à une mesure enregistrée et comprend les informations suivantes capteurs (Type du capteur), donnée (Valeur mesurée avec son unité date et heure.

Pour faciliter l'analyse des données, nous avons intégré un filtre déroulant permettant de filtrer les mesures par type de capteur. Nous pouvons ainsi sélectionner un ou plusieurs capteurs pour afficher uniquement les informations de ceux-ci.

Nous avons veillé à ce que cette page soit ergonomique et intuitive pour l'utilisateurs, tout en gardant notre esthétique qui ce veux minimaliste et moderne.

5) Mise en place et Base de données

Dans notre implémentation, nous avons essayé de respecter une architecture MVC (Modèle-Vue-Contrôleur) afin d'avoir une organisation claire du code. Le modèle est représenté par les fonctions d'interrogation de la base de données, la vue correspond aux composants Dash, et le contrôleur gère les interactions et le traitement des données.

Pour l'interrogation de la base de données, nous utilisons psycopg2 afin d'établir une connexion avec notre base de données. Les requêtes SQL nous permettent d'extraire les informations, comme les dernières valeurs mesurées par les capteurs ou encore l'ensemble des données à afficher dans le tableau. Nous avons conçu des fonctions pour chaque requête qui nous permettent de récupérer uniquement les données nécessaires. Pour éviter une surcharge inutile, nous limitons le nombre de résultats affichés.

Nous avons choisi d'utiliser Pandas pour structurer et manipuler les données récupérées de la base. Plotly, la bibliothèque utilisée pour l'affichage des graphiques dans notre application, fonctionne de plus simplement avec des DataFrames Pandas. Chaque requête exécutée retourne donc un DataFrame, ce qui nous permet de facilement filtrer, trier et formater les données avant leur affichage. Pour cela une fonction fetchdata prend une requête se connecte à la base de données exécute cette requête et nous retourne un DataFrame Pandas avec les données de la requête.

C- Flutter

Nous avons choisi de créer un Dashboard mobile il y a été conçu pour fournir une interface intuitive et permettre de surveiller les capteurs. Nous avons choisi Flutter comme framework de développement en raison de sa simplicité avec le langage de programmation Dart et sa capacité à produire des applications natives sur Android et iOS.

L'application offre une expérience utilisateur simple, avec un thème sombre et des widgets facilitant la lecture des données essentielles des capteurs. Pour ceci nous avons choisi une disposition simple avec la même topbar que notre Dashboard web et au milieu la dernière image capturer de la caméra. Nous avons également implémenté un système de rafraîchissement manuel via un bouton, permettant à l'utilisateur de mettre à jour les données vues que le Dashboard se trouve statique.

L'application mobile interagit avec notre API backend pour récupérer les dernières valeurs des capteurs. Nous avons mis en place une classe ApiService qui utilise la bibliothèque http de Dart pour effectuer des requêtes GET vers notre API. Ces requêtes permettent d'obtenir les informations des capteurs des différents capteurs et de la caméra. Chaque requête est traitée de manière asynchrone, pour ne pas bloquer l'interface utilisateur. Une fois les données reçues, elles sont décodées depuis le format JSON pour être utilisées directement dans l'application.

VIII- Conclusion

En conclusion ce projet nous a permis de mettre en place une station de captation de données connectée à un Dashboard mobile en Flutter et Web en python. Tout ceci grâce à un Raspberry Pi équipé de plusieurs capteurs ou on récupère des données d'une pièces en temps réel (avec les problématiques de Big Data et IoT qui suive). L'objectif était de faciliter la visualisation et l'analyse de ces données, tout en assurant une communication fluide entre les capteurs et les différentes applications.
Pour assurer une bonne organisation du projet, on a pu utiliser Git pour la gestion du versionning et du travail collaboratif. Côté technique, le Raspberry Pi fonctionne sous Linux, et le backend repose sur des appels API REST et requête de base de données. Pour le transport des données, on aurait pu explorer des

solutions comme Kafka afin d'améliorer la gestion des flux en temps réel mais on a préféré se diriger vers un protocole plus simple MQTT.
Le projet a suivi une logique assez simple :

-Les capteurs récupèrent des données et les envoient au serveur.
-Le backend les stocke et les met à disposition via des API.
-L'application mobile et web affiche les données

Même si l'application est fonctionnelle, il y a pas mal de pistes d'amélioration pour aller encore plus loin.

A- Amélioration

1) Dashboard

Pour les différents Dashboard, plusieurs points auraient pu être améliorés :

Pour le Dashboard web cela pourraient être Optimiser son fonctionnement en utilisant pleinement l'API pour récupérer et afficher les données. Ajouter des graphiques interactifs afin de permettre une surveillance plus détaillée d'une pièce, avec une visualisation en temps réel des variations de température, d'humidité ou de détection de mouvement.

Dashboard mobile cela auraient pu être d'ajouter un tableau récapitulatif permettant de visualiser les dernières entrées sur une journée. Intégrer un système de notifications en cas d'intrusion ou de dépassement d'un seuil critique.

Les améliorations générales possibles seraient de rendre les Dashboard dynamiques, notamment en intégrant JavaScript pour le Dashboard web et en améliorant l'expérience utilisateur sur mobile avec des animations et des mises à jour automatiques des données sans recharger la page.

2) Base de données

Pour améliorer la base de données, plusieurs solutions auraient pu être envisagées. La première consisterait à optimiser les relations en intégrant une table "Capteur" parent, qui servirait de modèle commun pour tous les capteurs. Cela permettrait d'ajouter une notion d'héritage. Une autre possibilité serait d'opter pour une base de données NoSQL comme MongoDB, qui permet de

stocker des données sans relation entre elles. Ce choix offrirait une plus grande souplesse pour gérer différents types de capteurs sans avoir à modifier constamment la structure de la base.
Concernant l'API, l'optimisation du cache permettrai de réduire le temps de réponse des requêtes et limiter la charge serveur. L'intégration d'un mécanisme de mise en cache, comme Redis ou un cache natif, permettrait d'éviter des appels inutiles à la base de données en stockant temporairement les réponses fréquentes. De plus, le choix du langage backend pourrait également impacter les performances. L'utilisation de langages plus performants et optimisés pour le traitement serveur, tels que PHP, Rust ou encore Go permettrait d'améliorer la réactivité et la robustesse du système.

IX- Annexes

- Code https://github.com/l0wwik/sae_vcod.git
- Grove
  - DHT https://wiki.seeedstudio.com/Grove-TemperatureAndHumidity_Sensor/
  - Son https://wiki.seeedstudio.com/Grove-Sound_Sensor/
  - Presence https://wiki.seeedstudio.com/Grove-PIR_Motion_Sensor/
  - Qualité de l'air https://wiki.seeedstudio.com/Grove-Air_Quality_Sensor_v1.3/
  - Ledstick https://wiki.seeedstudio.com/Grove-RGB_LED_Stick-10-WS2813_Mini/
  - Bouton led https://wiki.seeedstudio.com/Grove-LED_Button/
  - Buzzer https://wiki.seeedstudio.com/Grove-Buzzer/
- Documentation utiliser
  - Bibliothèque FaceNet https://github.com/timesler/facenet-pytorch
  - Bibliothèque PahoMqtt https://github.com/eclipse-paho/paho.mqtt.python/tree/master
  - Bibliothèque Dash https://dash.plotly.com
  - Bibliothèque Psycopg2 https://www.psycopg.org/docs/
  - Bibliothèque Pandas https://pandas.pydata.org
  - Bibliothèque Flask https://flask.palletsprojects.com/en/stable/
  - Postgre SQL https://www.postgresql.org
  - Flutter https://flutter.dev
  - Dart https://dart.dev/docs
  - Framboise 314 https://www.framboise314.fr/linternet-des-objets-iot-sur-raspberry-pi-avec-mqtt/