

SAE 3.02 Développer des Applications Communicantes

Sujet 2 - Gestion simplifiée du dossier étudiant

Formulaire Etudiant
Interroger la BD
Aide

Formulaire Etudiant

Nom:

Année

Moyenne

Photo


Envoyer

Formulaire Etudiant
Interroger la BD
Aide

Liste des étudiants
Camous ▼
Information sur votre choix...

Formulaire Etudiant
Interroger la BD
Aide

Liste des étudiants
Poret ▼

Nom	Année	Moyenne	Photo
Poret	2	16.75	

INTRODUCTION	3
CAHIER DES CHARGES	3
Contexte et objectif :	3
Description des utilisateurs et de leurs besoins :	3
Fonctionnalité du programme :	4
Contrainte et solutions :	4
Critères de qualité et de performance :	5
Plan de développement et de déploiement :	6
Documentation :	6
Sécurité :	7
Performance et scalabilité :	7
Impact environnemental :	8
Technologies utilisées :	9
Idee d'ajout de fonctionnalité :	9
DEVELOPPEMENT	10
Création du Programme principal :	10
Hadji :	10
Hatim :	21
Maxime :	47
Difficultés rencontrées :	96
Test et validation :	96
FONCTIONNALITE BONUS	96
CONCLUSION ET PERSPECTIVES	96
ANNEXES	96
Manuel d'utilisation :	96
Code source :	101
Sae.py :	101
sae.kv :	120
recup_photo.py :	136
Prof.py :	137
personne.py :	137
get_bina.py :	138
vidage_table.py :	138
Etudiant.py :	139
Evaluation des performances :	Erreur ! Signet non défini.

INTRODUCTION

CAHIER DES CHARGES

Contexte et objectif :

Dans le cadre de la SAÉ 302 Applications communicantes, nous avons été amenés à développer une application qui permettra de gérer des dossiers de différents étudiants. Elle doit contenir un certain nombre de fonctionnalités obligatoire mais l'application peut contenir d'autres fonctionnalités pensées et développées par nous-même.

À termes, l'application devra être fonctionnelle avec toutes les fonctionnalités demandées ainsi que celles qui ont été pensées devront être fonctionnelles utilisables. Le tout devra être accompagné d'un guide qui permettra aux utilisateurs de soit reproduire l'application ou bien de l'utiliser en tant que telle.

Description des utilisateurs et de leurs besoins :

L'application aura donc pour objectif de gérer la situation d'un groupe d'étudiants. Les utilisateurs de l'application seront les étudiants eux-mêmes, les professeurs ainsi que les administrateurs qui eux auront le contrôle de l'accès au formulaire d'inscription et de la gestion des données des étudiants dans la base de données. Les étudiants pourront ainsi consulter leurs résultats obtenus et leurs moyennes dans chacun des enseignements suivis.

Ils auront également accès à leur photo de profil stockée dans la base de données. Les administrateurs pourront effectuer des requêtes sur les données des étudiants via une liste déroulante, ainsi que gérer les données de la base de données, y compris la création, la mise à jour et la suppression de données. Les données pour les étudiants seront stockées dans une base de données contenant les tables "étudiants" et "matières", avec un minimum de 10 étudiants et 5 matières. Les photos des étudiants seront stockées dans la table "étudiants" dans un champ de type blob.

Fonctionnalité du programme :

L'application développée dans ce projet offre une interface simple d'utilisation pour les utilisateurs. Elle permet aux étudiants de s'inscrire en remplissant tous les champs nécessaires, y compris le choix de leur spécialité (matière) et la sélection de leur photo directement depuis leur ordinateur.

Les informations saisies lors de l'inscription sont ensuite envoyées vers une base de données MariaDB où elles sont stockées et traitées. Les utilisateurs peuvent ensuite se connecter à l'application en utilisant leur adresse mail et mot de passe saisis lors de l'inscription.

Les professeurs ont accès à un formulaire permettant de saisir ou de modifier la moyenne d'un étudiant dans sa matière, ils ont aussi accès aux informations de tous les étudiants. Quant à eux, les étudiants n'ont accès qu'à leurs propres informations. Ils peuvent aussi avoir accès au détail de leurs notes dans la matière sélectionnée.

Sur la page des informations, il y a une fonctionnalité qui permet aux professeurs de choisir un étudiant parmi une liste et d'afficher ses informations. Il y a également une autre fonctionnalité, disponible les professeurs, qui permet de choisir une matière parmi les 5 matières proposées (+ la spécialité) et d'afficher les informations de l'élève choisi, y compris la moyenne de la matière sélectionnée.

Contrainte et solutions :

Pour ce projet nous avons identifié plusieurs contraintes qui doivent être prises en compte pour garantir la réussite du projet. Tout d'abord il y a la contrainte du temps qui lui est limité et une date limite à respecter. Nous avons prévu une organisation claire avec des dates butoirs pour certains éléments de développement et une répartition des tâches en fonction des points faibles et forts des membres pour maximiser l'efficacité de l'équipe.

Il y a aussi le fait que les développeurs de cette application ont des compétences basiques et non poussées c'est pour cela que cette application nécessitera un travail de recherche et d'apprentissage en développant pour améliorer les compétences de l'équipe au fur et à mesure.

La sécurité des données est une contrainte primordiale pour les utilisateurs c'est pour cela que nous avons prévu de crypter les données sensibles telles que les mots de passe et de limiter l'accès aux données en fonction des utilisateurs.

Enfin il y aura des normes à respecter qui sont les besoins impératifs pour les utilisateurs c'est pour cela que nous avons prévu de nous baser d'abord sur les besoins nécessaires pour garantir le respect des normes et ensuite selon l'avancée et les possibilités, on pourrait songer à ajouter de nouvelles fonctionnalités.

Critères de qualité et de performance :

1. Exigences fonctionnelles :

- La possibilité pour un étudiant de s'inscrire via un formulaire en fournissant des informations personnelles telles que : le nom, le prénom, l'adresse Email, l'année d'étude, la spécialité choisie, la photo et le mot de passe
- La possibilité pour un professeur de s'inscrire via un formulaire en fournissant des informations personnelles telles que : le nom, le prénom, l'adresse Email et le mot de passe
- La possibilité pour un étudiant et un professeur inscrit de se connecter à l'application en utilisant leur adresse Email et leur mot de passe
- La possibilité pour un étudiant une fois connecté, de consulter uniquement ses propres informations telles que son nom, son prénom, son adresse email, sa spécialité, sa moyenne générale et sa photo.
- La possibilité pour un professeur de consulter les informations de tous les étudiants telles que leur nom, leur prénom, leur adresse email, leur spécialité, leur moyenne générale et leur photo.
- La possibilité pour un étudiant ou un professeur connecté de se déconnecter de l'application.

2. Exigences non fonctionnelles :

- Une sécurité solide pour protéger les informations des utilisateurs et les protéger contre les violations de données
- Une fiabilité élevée pour garantir que l'application est disponible pour les utilisateurs à tout moment
- Une facilité d'utilisation pour que les utilisateurs puissent naviguer et utiliser l'application efficacement

3. Critères de performance

- Temps de réponse inférieur à 2 secondes pour l'affichage des informations après la connexion de l'utilisateur
- Taux de disponibilité de 99,5% pour garantir que l'application est accessible pour les utilisateurs la plupart du temps
- Taux d'erreur inférieur à 0,1% pour garantir que l'application fonctionne correctement la plupart du temps

Plan de développement et de déploiement :

1. Recueil des exigences
 - Réunir les exigences fonctionnelles et non fonctionnelles de l'application pour s'assurer que les besoins de l'application sont bien compris (Voir partie précédente)
2. Conception de l'architecture
 - Concevoir l'architecture de l'application, y compris la conception de la base de données et les interfaces utilisateur
3. Développement
 - Créer la base de données, les tables et les colonnes nécessaires
 - Développer l'application, d'abord les formulaires d'inscription, puis la page de connexion et enfin la page post-connexion
4. Tests
 - Tester l'application pour s'assurer que l'application fonctionne correctement et répond aux critères de qualité et de performance énoncés dans le cahier des charges

Documentation :

1. Documentations Python

- La documentation officielle de Python est disponible sur le site web python.org. Elle inclut des informations sur les fonctionnalités de base de Python, ainsi que des exemples de code et des tutoriels pour les développeurs débutants et avancés. Elle inclut également des informations sur les différentes classes et fonctions de la bibliothèque standard de Python, ainsi que des exemples de code et des explications détaillées

2. Documentations Kivy

- La documentation officielle de Kivy est disponible sur le site web kivy.org. Elle inclut des informations sur les fonctionnalités de base de Kivy, ainsi que des exemples de code et des tutoriels pour les développeurs débutants et avancés. Elle inclut également des informations sur les

différentes classes et fonctions de la bibliothèque Kivy Language, ainsi que des exemples de code et des explications détaillées

3. Sites communautaires

- **Reddit** : Il existe une communauté Python active sur Reddit qui peut être une source d'aide pour les développeurs qui rencontrent des problèmes ou qui cherchent des conseils.
- **Stackoverflow** : C'est un site communautaire de questions-réponses pour les développeurs où vous pouvez poser des questions et obtenir des réponses de la part de la communauté de développeurs
- **GitHub** : Il est aussi possible de trouver des codes exemples sur des dépôts GitHub qui peuvent aider à comprendre certaines fonctionnalités

Sécurité :

1. Identification et authentification

- L'application doit inclure des mécanismes d'identification et d'authentification pour s'assurer que seuls les utilisateurs autorisés peuvent accéder aux informations de l'application. Cela inclut un mécanisme d'enregistrement par email et mot de passe

2. Cryptage des données

- Les données sensibles de l'application, telles que les mots de passe des utilisateurs, doivent être cryptées pour protéger contre les violations de données

3. Autorisation

- L'application doit inclure des mécanismes d'autorisation pour s'assurer que les utilisateurs ne peuvent accéder qu'aux informations auxquelles ils ont été autorisés à accéder. Par exemple, les étudiants ne devraient pas avoir accès aux informations des autres étudiants et les professeurs ne devraient pas avoir accès aux informations sensibles des étudiants telles que les mots de passes

Performance et scalabilité :

- Utiliser des bases de données relationnelles efficaces : Pour stocker les informations sur les professeurs et les étudiants ainsi que leurs notes, il est préférable d'utiliser une base de données relationnelle comme MySQL ou PostgreSQL. Ces bases de données

permettent de stocker et de récupérer des données de manière rapide et efficace, même lorsque la base de données est volumineuse.

- Optimiser les requêtes SQL : Pour récupérer les informations sur les professeurs et les étudiants ainsi que leurs notes, il est important d'optimiser les requêtes SQL pour éviter les requêtes inutiles et les jointures inutiles. Il est également important de limiter les données récupérées pour réduire les temps de chargement.
- Utiliser des widgets recyclés : Pour afficher une grande quantité de données, il est préférable d'utiliser des widgets recyclés pour éviter de surcharger la mémoire. Cela permet de maintenir une bonne performance même lorsque le nombre d'éléments à afficher est élevé.
- Utiliser un système de cache : Pour réduire les temps de chargement des données, il est préférable d'utiliser un système de cache pour stocker temporairement les données récemment utilisées. Cela permet de réduire les requêtes vers la base de données et de réduire les temps de chargement des données.
- Utiliser des animations pour les transitions : Pour une meilleure expérience utilisateur, il est préférable d'utiliser des animations pour les transitions entre les pages de l'application. Cela permet de rendre l'application plus fluide et plus agréable à utiliser.
- Utiliser des outils de performance : Il est également recommandé d'utiliser des outils de performance pour évaluer les performances de l'application. Ces outils peuvent aider à identifier les zones qui nécessitent des améliorations et à optimiser les performances de l'application.
- Utiliser une architecture de l'application bien structurée : Il est important d'utiliser une architecture de l'application bien structurée pour faciliter la maintenance et l'évolution de l'application. Il est également important de séparer clairement les différentes fonctionnalités de l'application pour faciliter le débogage et l'optimisation.

Impact environnemental :

En ce qui concerne l'impact environnemental et économique, l'utilisation de Python et d'une base de données est généralement considérée comme étant relativement peu impactante, tant sur le plan environnemental que sur le plan économique.

Python est une technologie open source largement utilisée dans l'industrie, ce qui signifie qu'il existe de nombreuses ressources en ligne pour apprendre et utiliser ce langage de programmation. De plus, la plupart des bases de données sont également largement utilisées et sont généralement considérées comme étant peu coûteuses à utiliser d'autant plus que cette dernière pour le projet tournera sous une VM donc ce qui réduit encore plus l'impact carbone.

Technologies utilisées :

- Logiciel :
 - Logiciel Visual Studio Code pour le développement du code.
 - One drive pour la synchronisation des documents de l'équipe.
 - Teams pour rassembler nos idées et communiquer.
- Programmation
 - Python avec la bibliothèque Kivy pour développer l'interface Utilisateur.
 - Module mysqlconnector pour la communication entre le programme et la BD.
 - La bibliothèque de socket sera utilisée pour la transmission de données entre deux appareils.
- Machine :
 - VM Linux sous VirtualBox pour faire tourner le serveur Maria DB qui servira à stocker les données.
 - PC sur lequel tournera le programme principal.

Idée d'ajout de fonctionnalité :

- Afficher les informations des étudiants sur une page web
- Faire une inscription pour l'étudiant et les professeurs pour accéder à leur compte et visualiser uniquement ce à quoi ils sont autorisés
- Mettre un système de notification sur le compte du professeur pour l'informer quand un nouvel étudiant s'inscrit
- Faire un service de chat en ligne entre professeur et étudiant
- Système de modification des notes étudiants
- Spécialité en fonction de l'étudiant

DEVELOPPEMENT

Création du Programme principal :

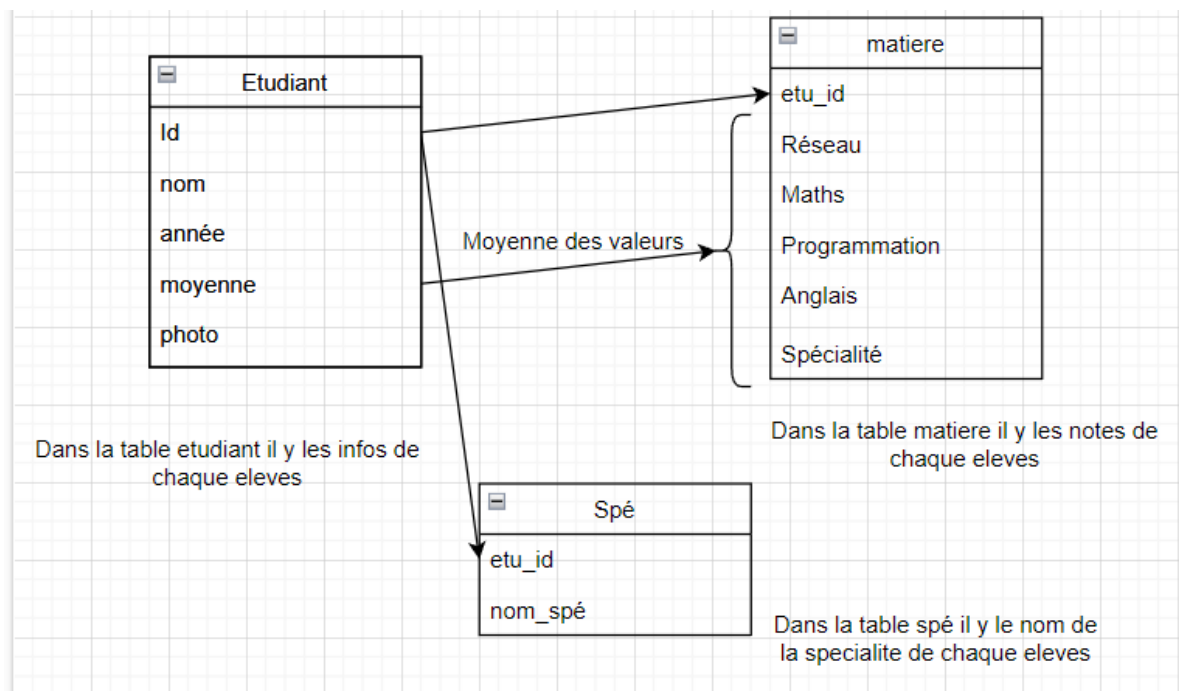
Hadji :

Le projet avait besoin d'une base de données pour pouvoir stocker puis exploiter différentes informations je m'en suis donc chargé.

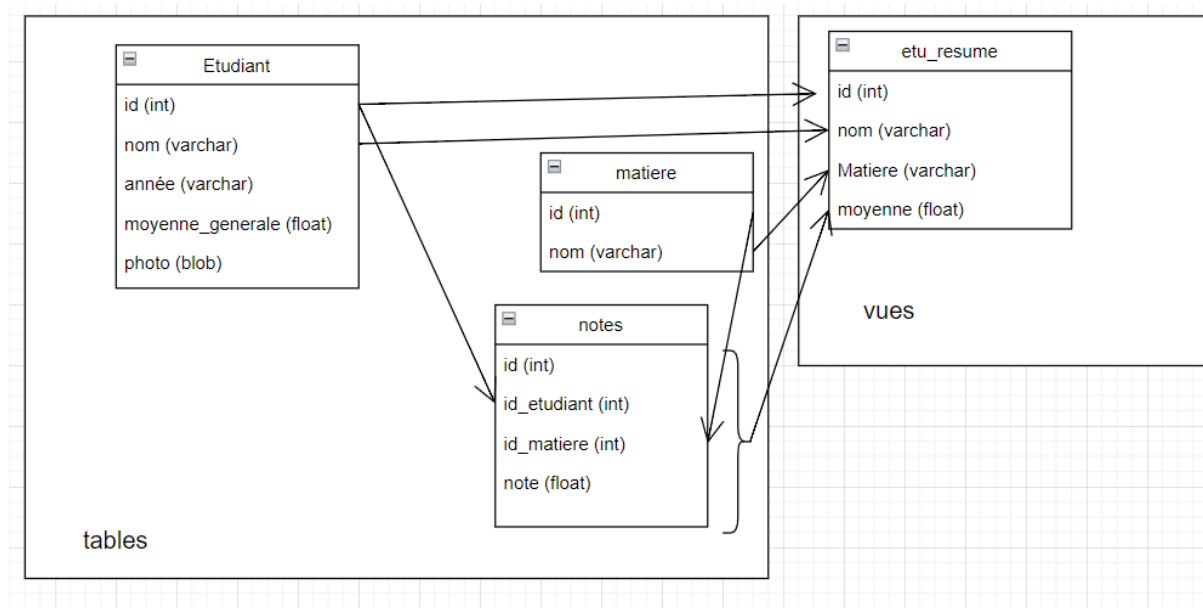
Tout d'abord il fallait avoir une machine qui hébergerait le serveur MariaDB, j'ai donc créer une machine virtuelle sur VirtualBox et je lui ai administrer les services nécessaire.

Il a fallu installer : Pip, kivy, plyer, websockets et websockets-clients, bulldozer, mysql.connector

J'ai ensuite commencé a prendre en main l'outil mysql et la base de donnée et commencer à lui insérer des données notamment les tables demandées (etudiant et matiere). Puis j'ai commencé à étudier plus en détail les liaisons entre tables (clés étrangères et primaire). Voici un premier schéma qui montre ce que j'essayais de faire au tout début du projet :



Puis en avançant j'ai complètement recommencer la base de donnée car je me retrouvais avec des problèmes que je n'arrivais pas à résoudre dont le calcul de la moyenne ce qui a m'a donner le rendu suivant :



A ce moment-là il y a donc 4 tables, dont une qui est en fait une vue. Une vue est une sorte de table virtuelle constituée de données des tables de la base. Elle se comporte ensuite comme une table lorsqu'on l'interroge. Sa modification reste différente d'une table basique.

La vue contient le résumé des moyennes de chaque élèves dans chaque matières.

Résultats, si l'on veut :

Les infos en général ==> ***select * from etudiants;***

Les infos détaillées des notes ==> ***select * from notes;***

Le nom et l'id des matieres ==> ***select * from matieres;***

Les notes de chaque étudiant dans chacune des matieres = ***select * from etu_resume ;***

Pour préciser un étudiant on rajoute à la requête : ***where id = 'num_id' ;***

Avec tout cela on peut jouer avec les requêtes pour extraire les infos voulues.

Je me suis ensuite penché sur comment les notes seraient données et comment calculer la moyenne générale en fonction de ces notes. Tout d'abord pour l'insertion des notes, comme la table notes avait les attributs : id_etudiant, id_matiere, note, le système de note était un peu spécial et la commande à entrer était donc :

INSERT INTO notes (id_etudiant, id_matiere, note) values(1,4,18)

L'étudiant n°1 aura la note de 18 dans la matière N°4 en l'occurrence programmation

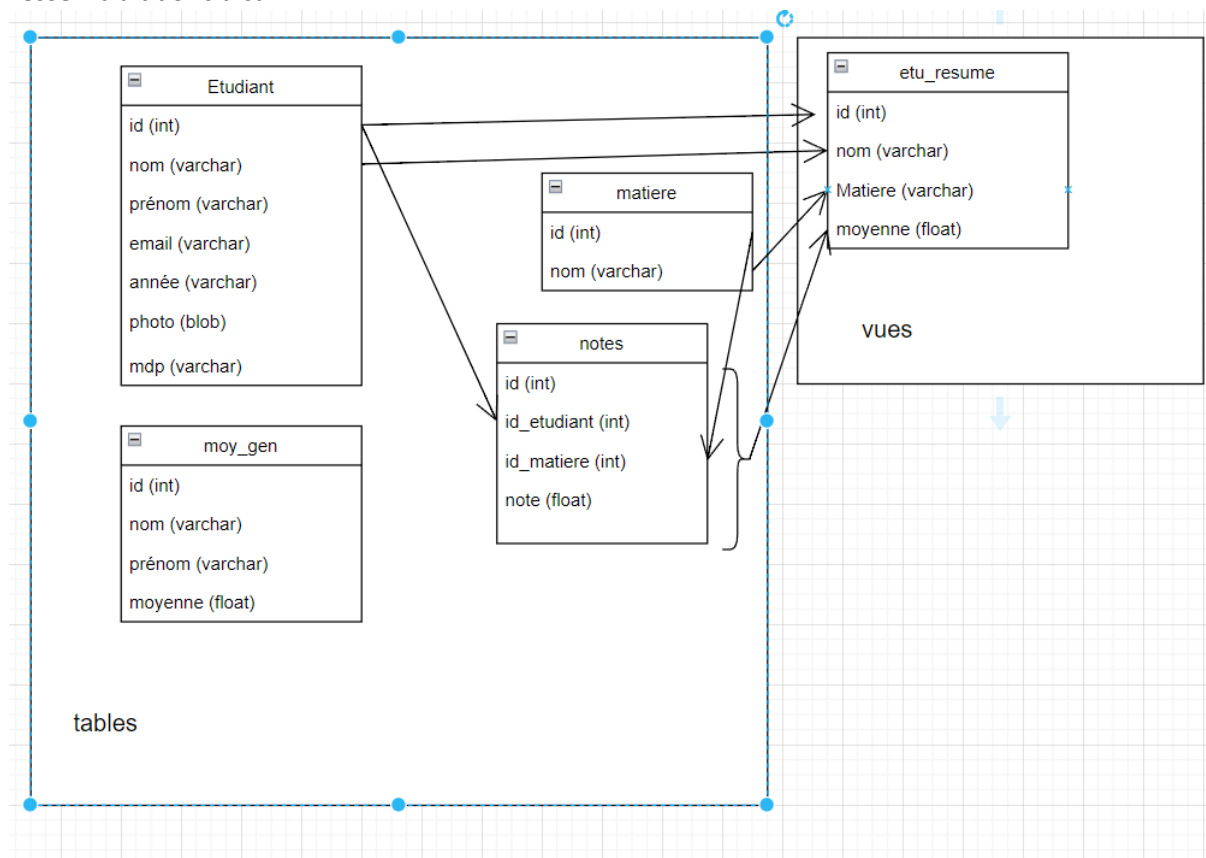
Les tables sont reliées par des contraintes de clés primaires et étrangères pour que les id correspondent. (ex : id_etudiant de la table notes = id de la table etudiant)

Ensuite il fallait que la table moyenne générale soit en accord avec les notes de la tables notes il fallait donc à chaque fois lancer la commande suivante :

UPDATE etudiants SET moyenne = (SELECT AVG(note) FROM notes WHERE id_etudiant = etudiants.id)

J'ai voulu créer un 'trigger' qui sert à comme son nom l'indique, déclencher une action lorsqu'un évènement à lieu. Mais je me suis rendu compte que cela pourrait être fait plus tard et plus facilement avec des requêtes python.

Puis l'avancement dans le projet à fait que j'ai du modifier certaines choses pour les besoins de mon groupe notamment pour le moyennes générales. J'ai du faire une nouvelle table et la base de donnée ressemblait donc à ca :



Cela à entrainer certains changements dont celle pour « mettre à jour » les moyennes qui est devenue :

UPDATE moy_gen m SET m.moyenne = (SELECT AVG(n.note) FROM notes n WHERE n.mail_etu = m.mail_etu)

Ensuite le moment de faire une première mise en commun est arrivée et donc de faire communiquer la base de données avec les codes python. La connexion à distance sur le serveur MySQL était impossible il a donc fallu faire des modifications sur la machine :

- Il à d'abord fallu chercher le fichier « conf50-server » puis modifier la ligne bind_address et la régler à 0.0.0.0 (127.0.0.1 par défaut). Le 0.0.0.0 signifie que toutes les adresses sont autorisées.
- Puis ensuite il a fallu entrer dans la base de données en root et puis accorder les pleins privilèges à l'utilisateur voulu à la table voulue. Dans notre cas nous l'avons fait pour l'utilisateur « root » et « admin » :

GRANT ALL PRIVILEGES ON doss_etu.* TO 'admin'@'%' IDENTIFIED BY 'sae302';

A partir de là nous pouvions commencer à faire fonctionner les différents éléments. La partie fonctions python, la partie KV puis la base de données.

La mise en commun a permis de prendre connaissance du travail des autres et de tirer profit du travail des uns des autres c'est-à-dire d'utiliser ce que les autres ont fait pour pouvoir améliorer voir développer ce qu'on a déjà fait.

Pour ma part les fonctions python et le module mysql.connector m'ont beaucoup aidé notamment pour créer des scripts utiles ou alors de lancer plusieurs commande n même temps.

J'ai notamment créer un script qui permet de vider toutes les tables de la base de données ce qui nous a été utile à certains moment du projet :

```
import mysql.connector

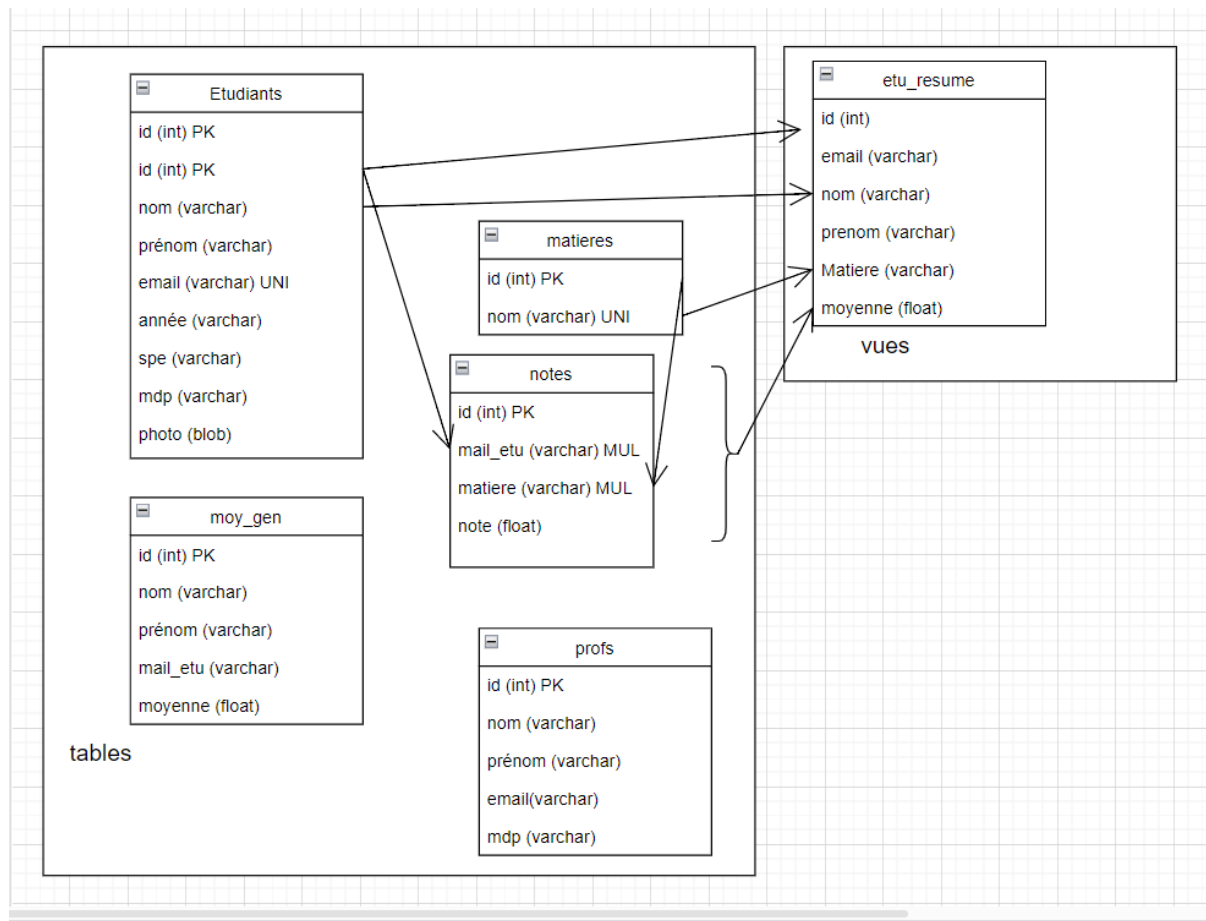
try:
    connection_params = {
        'host': "192.168.56.101",
        'user': "admin",
        'password': "sae302",
        'database': "doss_etu"
    }
except mysql.connector.Error as e:
    print("Exception : ", e)
else:
    with mysql.connector.connect(**connection_params) as db :
        with db.cursor() as c:
            c.execute("SET FOREIGN_KEY_CHECKS = 0;")
            c.execute("truncate table etudiants;")
            c.execute("truncate table moy_gen;")
            c.execute("truncate table notes;")
            c.execute("truncate table profs;")
            c.execute("truncate table test_blob;")
            c.execute("SET FOREIGN_KEY_CHECKS = 1;")

        db.commit()
        db.close()
```

La commande "SET FOREIGN_KEY_CHECKS = 0;" permet de désactiver les contraintes de clé étrangère afin de pouvoir vider les tables sans se soucier de problèmes de contraintes.

La commande "SET FOREIGN_KEY_CHECKS = 1;" les réactive.

Ensuite pendant un certain moment je répondais en direct aux demandes de mon groupe qui me demandait de modifier certains éléments de la table pour leurs besoins cela à entrainer un bon nombre de changements car la base de données évolue en fonction du code python. Cela à entrainer la refonte totale de la base de donnée mais qui reste semblable à celle du début :



Les données PK indiquent les clés primaires des tables, une clé primaire est la donnée qui permet d'identifier de manière unique un enregistrement dans une table (en général des 'id')

Les données UNI elles assurent l'unicité des entrées c'est-à-dire que les données contenues dans cette colonne sont uniques et ne peuvent pas être insérées de nouveau.

Les données MUL représentent les données liées aux UNI.

Ce schéma de la base de données est le schéma final et il ne bougera donc pas.

Ensuite il a fallu s'occuper de la partie stockage de la photo puis la réception. Cette étape à occuper une grosse partie de mon temps car j'ai du utiliser le fichier python et le fichier « .kv », domaines ou je ne suis pas très à l'aise.

Après des heures de recherches, de tests et d'essais j'ai (avec l'aide de mon groupe) enfin réussi à trouver une solution en utilisant un sélectionneur de fichier graphique.

Pour cette partie sur laquelle j'ai eu le plus de mal, j'ai dû intervenir dans le code python, le fichier '.kv' et la base de données. J'ai surtout eu besoin du code python et de créer les fonctions qui me permettront de réaliser cela. Il faut savoir que toutes les méthodes se trouvent dans la classe « sae(app) ». Il y a donc une première méthode appelée « remplacer_backslash » est la suivante :

```
#Permet de remplacer les backslash par deux backslash pour éviter des erreurs
def remplacer_backslash(self,chaîne):
    return chaîne.replace("\\\\", "\\")
```

Comme indiquée dans le commentaire cette méthode, cette dernière va permettre de remplacer un backslash par deux car sur python le caractère « \ » est spécial et donc si on veut l'utiliser il faut entrer « \\ ». Donc la fonction transforme tous les « \ » en « \\ » à la chaîne de caractère passée en paramètre.

Ensuite nous avons la méthode « recup_chemin » qui elle va se servir de la méthode créée juste avant :

```
#Permet de remplacer les backslash et d'affecter le chemin au self.e.photo
def recup_chemin(self, path, selection):
    selected_file_path = selection[0]
    self.root.get_screen("ImgEtu").ids.img.source= selected_file_path
    #print(self.remplacer_backslash(selected_file_path))
    self.e.set_photo(self.remplacer_backslash(selected_file_path))
```

Cette méthode va en fait récupérer l'élément d'index 0 de la liste renvoyée par le selectionneur de fichier (Voir suite), cet élément contient donc le chemin de l'image sélectionnée. Ensuite on indique que la source de l'image affichée dans le kv sera le chemin tout juste récupéré. Puis modifie le contenu de l'attribut photo de la classe « Etudiant » et on y met le chemin qui lui-même est passé en paramètre dans la fonction précédente.

Ensuite il y a une méthode créée dans un fichier à part et qu'on a donc importé qui s'appelle « get_binary_data » :

```
def get_binary_data(image_path):
    with open(image_path, 'rb') as image_file:
        binary_data = image_file.read()
        image_file.close()
        #print(binary_data)

    return binary_data
```

Cette méthode va prendre en paramètre le contenu de « photo », puis renvoyer ses données binaires grâce à la méthode d'ouverture de fichier « rb ». Le programme va ensuite stocker ces données dans une variable qui est retournée par le programme. Cette méthode est utilisée lors de l'envoi des données vers la base (commande détaillée plus tard dans le document) :

```
params = (self.e.get_nom(), self.e.get_prenom(), self.e.get_annee(),
self.e.get_matiere(), get_binary_data(self.e.get_photo()), self.e.get_email(),
self.e.get_mdp())
```


Maintenant que les données sont envoyées il faut pouvoir les récupérer, pour cela on a créé une fonction que l'on réutilisera dans une méthode de la classe 'sae' Voici le code de cette fonction :

```
import mysql.connector
def recup_photo(nom):
    # Établissement de la connexion
    cnx = mysql.connector.connect(user='admin', password='sae302',
                                   host='172.20.10.3', database='doss_etu')

    requete = "SELECT photo FROM etudiants WHERE nom = %s"
    values = (nom,)

    # Exécution de la requête
    cursor = cnx.cursor()
    cursor.execute(requete, values)

    # Récupération des données binaires du fichier
    data = cursor.fetchone()[0]

    # Écriture des données binaires dans un nouveau fichier
    with open(nom+'.png', 'wb') as f:
        f.write(data)

    # Fermeture de la connexion
    cnx.close()

    # print(nom+".png")
    return nom+".png"
```

Donc le début va, permettre de se connecter à la base de données puis on met en place la requête qui va sélectionner la photo de l'étudiant dont le nom a été passé en paramètre.

Ensuite il stock les données binaires contenues dans la donnée et ouvre un fichier en mode écriture binaire puis il y écrit la donnée récupérée. Cela crée le fichier « .png » au nom de l'étudiant dans le répertoire courant, puis la photo sera 'reconstituée' sous le nom : <'nom'.png>. La fonction renvoie le nom du fichier.

Ceci va donc permettre lors du rassemblement d'informations de l'étudiants d'utiliser cette fonction comme ceci (cadre rouge) :

```
def page_infos_etu(self):
    inf = self.e.get_infos_etu()
    nom=inf["Nom"]
    prenom=inf["Prenom"]
    ...
    self.root.get_screen("EtuHome").ids.specialite.text=str(specialite)
    self.root.get_screen("EtuHome").ids.photo.source=recup_photo(nom)
    moy = round(((math+reseau+telecom+prog+anglais+specialite)/6),2)
    self.root.get_screen("EtuHome").ids.moyenne.text= str(moy)
```

Comme j'avais aussi travaillé dessus sur la base de données, il fallait afficher la moyenne générale que l'on a finalement calculée en python. J'ai simplement créé une variable qui prenait toute les notes et calculait la moyenne et l'affichait donc dans l'endroit prédestiné (cadre vert ci-dessus) grâce à l'id présent dans le fichier « .kv » :

```
<EtuHome>:
    name : "EtuHome"
    moyenne:moyenne

    ...

    Label:

        id:moyenne
        text:""
        markup: True
        font_size:30
        size_hint: 0.2,0.2
        pos_hint:{'center_x':.75, 'center_y':.10}
```

Pour finir voici un aperçu des tables de la base de données :

Table Etudiants (sans la photo et les mots de passe)

id	nom	prenom	email	annee	spe
1	MOUIGNI	Hadji	hadji@univ.fr	2A	Cybersécurité
2	FOURNIER	Jeremy	jeremy@univ.fr	1A	IA
3	RIPIEGO	Jorane	jorane@univ.fr	2A	IA
4	MEYLAN	Kenan	kenan@univ.fr	1A	Cybersécurité
5	OCANA	Kevin	kevin@univ.fr	2A	IA
6	BAPTISTE	Maxime	maxime@univ.fr	2A	Cybersécurité
7	ROBERT	Nicolas	nicolas@univ.fr	2A	IA
8	DELCHIAPPO	Quentin	quentin@univ.fr	1A	Cybersécurité
9	POITOU	Theo	theo@univ.fr	2A	Cybersécurité

Vue Etu_resume : (détail des notes de chaque élève dans chaque matière):

id	email	nom	prenom	matiere	moyenne
1	hadji@univ.fr	MOUIGNI	Hadji	Réseau	14
1	hadji@univ.fr	MOUIGNI	Hadji	Télécom	7.25
1	hadji@univ.fr	MOUIGNI	Hadji	Maths	18
1	hadji@univ.fr	MOUIGNI	Hadji	Programmation	6.25
1	hadji@univ.fr	MOUIGNI	Hadji	Anglais	12
1	hadji@univ.fr	MOUIGNI	Hadji	Cybersécurité	19
2	jeremy@univ.fr	FOURNIER	Jeremy	Réseau	9.5
2	jeremy@univ.fr	FOURNIER	Jeremy	Télécom	17.75
2	jeremy@univ.fr	FOURNIER	Jeremy	Maths	7.75
2	jeremy@univ.fr	FOURNIER	Jeremy	Programmation	2.5
2	jeremy@univ.fr	FOURNIER	Jeremy	Anglais	3.75
2	jeremy@univ.fr	FOURNIER	Jeremy	IA	8.25

Tables matières :

id	nom
5	Anglais
7	Cybersécurité
6	IA
3	Maths
4	Programmation
1	Réseau
2	Télécom

Table moy_gen :

id	nom	prenom	mail_etu	moyenne
1	MOUIGNI	Hadji	hadji@univ.fr	12.75
2	FOURNIER	Jeremy	jeremy@univ.fr	8.25
3	RIPIEGO	Jorane	jorane@univ.fr	12.5417
4	MEYLAN	Kenan	kenan@univ.fr	7.33333
5	OCANA	Kevin	kevin@univ.fr	16.875
6	BAPTISTE	Maxime	maxime@univ.fr	8.375
7	ROBERT	Nicolas	nicolas@univ.fr	6.20833
8	DELCHIAPPO	Quentin	quentin@univ.fr	12.75
9	POITOU	Theo	theo@univ.fr	6.875

Table notes :

id	mail_etu	matiere	note
1	hadji@univ.fr	Maths	18
2	hadji@univ.fr	Programmation	6.25
3	hadji@univ.fr	Réseau	14
4	hadji@univ.fr	Télécom	7.25
5	hadji@univ.fr	Anglais	12
6	hadji@univ.fr	Cybersécurité	19
7	jeremy@univ.fr	Maths	7.75
8	jeremy@univ.fr	Programmation	2.5
9	jeremy@univ.fr	Réseau	9.5
10	jeremy@univ.fr	Télécom	17.75
11	jeremy@univ.fr	Anglais	3.75

Tables profs : (sans le mot de passe crypté)

id	nom	prenom	email
1	kwa	coubeh	kwacoubeh@univ.fr

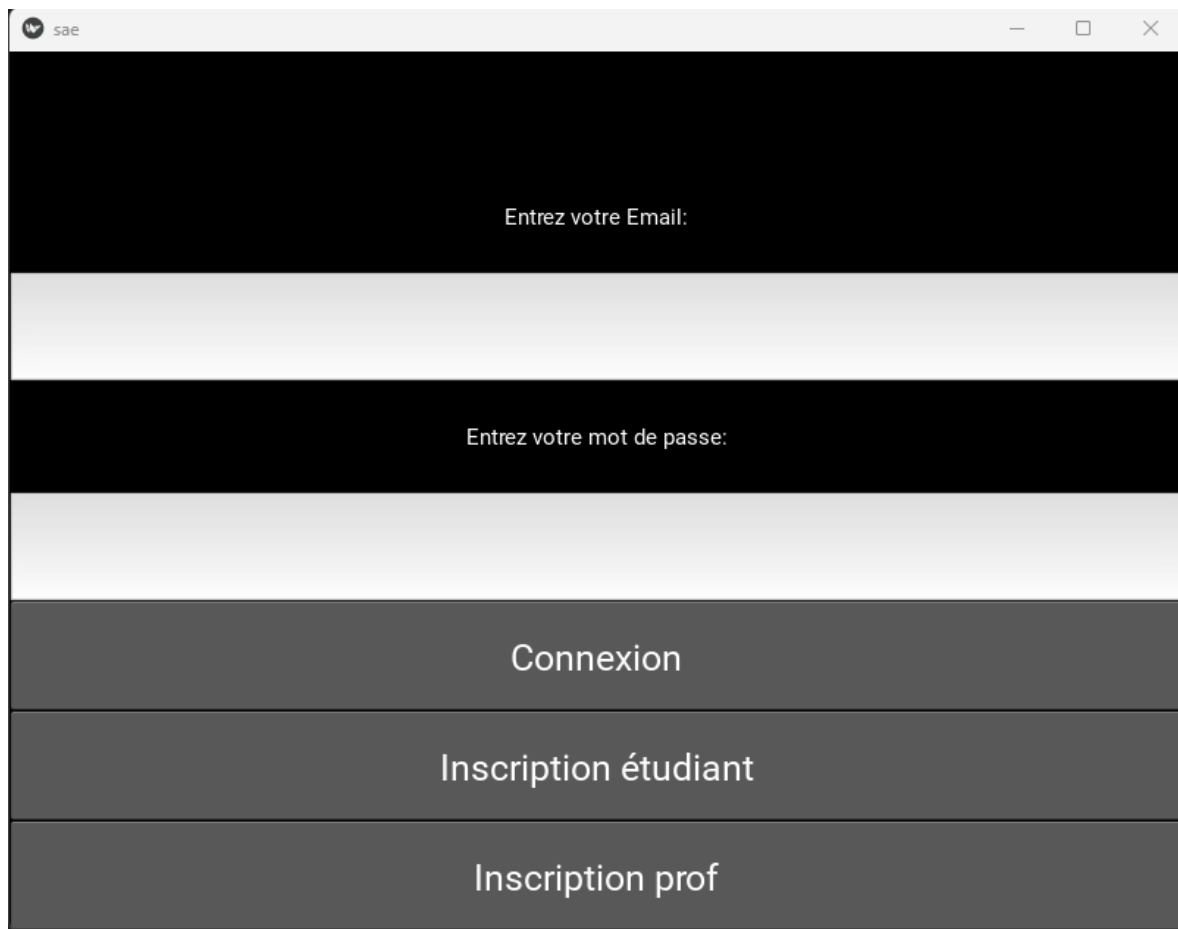
Hatim :

Avoir une application fonctionnelle est certes primordial, mais si cette dernière est dépourvue d'interface graphique simple et agréable à utiliser, personne n'en voudra.

C'est pour cela que j'ai décidé de m'occuper de cette partie plus que primordiale au bon déploiement de notre application Kivy.

Tout d'abord la page de connexion :

Avant :



The screenshot shows a web browser window with the title 'sae'. The page has a black background. At the top, there is a text input field labeled 'Entrez votre Email:'. Below it is another text input field labeled 'Entrez votre mot de passe:'. At the bottom, there are three buttons stacked vertically: 'Connexion', 'Inscription étudiant', and 'Inscription prof'.

J'ai d'abord réalisé une maquette pour mieux visualiser comment placer les widgets dans mon interface :

Mail

Mdp

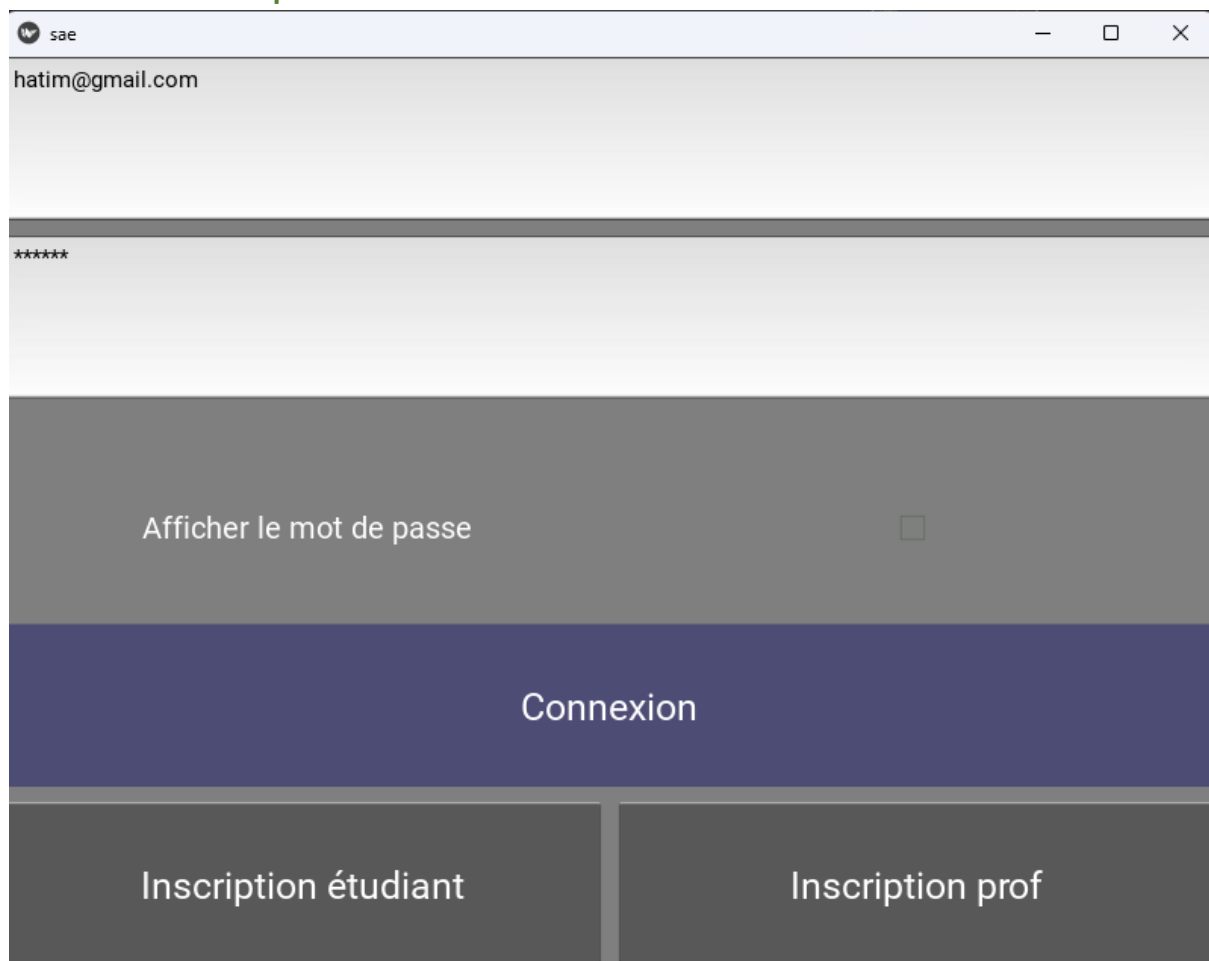
Affichage mdp☒

Connexion

Inscription étudiant

Inscription Prof

Et voilà le résultat après modification :



Les changements les plus marquant sont au niveau de la disposition des différents bouton et texte. J'ai ici comparé à la première version intégrer le texte mail et mdp dans la zone de saisie pour éviter de prendre de la place inutilement.

Au niveau du code kivy voilà la différence :

Ici on peut voir qu'il n'y a qu'un seul GridLayout avec 1 seule colonne ce qui prend énormément de place lorsque l'on a beaucoup de widgets à ajouter.

```
<Connexion>:
    name : "Connexion"
    email: email
    passwd: passwd

GridLayout:
    cols: 1
    size: (root.width,root.height)

    Label:
        text: root.my_text
        font_size: 15
```

```
Label:
    text: "Entrez votre Email:"
    font_size: 15

TextInput:
    id: email
    multiline: False

Label:
    text: "Entrez votre mot de passe:"
    font_size: 15

TextInput:
    id: passwd
    multiline: False
    password: True

Button:
    text: "Connexion"
    font_size: 25
    on_press: root.accept() if passwd.text == root.mdp_value() else
root.refuse()
    on_release:
        app.root.current = "second" if passwd.text == root.mdp_value()
else "Connexion"
        root.manager.transition.direction = "left"

Button:
    text: "Inscription étudiant"
    font_size: 25
    on_release:
        app.root.current = "InscriptionEtu"
        root.manager.transition.direction = "left"

Button:
    text: "Inscription prof"
    font_size: 25
    on_release:
        app.root.current = "InscriptionProf"
        root.manager.transition.direction = "left"
```


Maintenant si on regarde le nouveau code kivy :

```
<Connexion>:
    name : "Connexion"
    email: email
    passwd: passwd
    log:log
    vision: vision


    canvas.before:
        Color:
            rgba: 0.5, 0.5, 0.5, 1
        Rectangle:
            pos: self.pos
            size: self.size
```

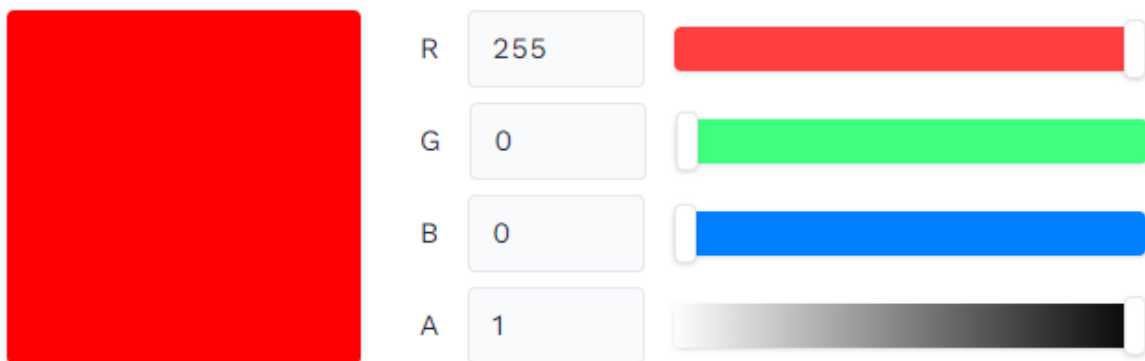
J'ai ajouté un canvas.before qui me permet d'appliquer une couleur de fond global à ma fenêtre.

Pour changer la couleur de fond j'utilise la commande rgba dans laquelle j'indique chaque couleur ici on a quatre valeur à insérer : Red, Green, Blue et alpha


Les valeurs sont comprises entre 0 et 1 qui correspond à la quantité que l'on souhaite ajouter. Pour alpha cela correspond à l'opacité de notre fond de couleur plus il est bas plus la couleur sera moins visible.

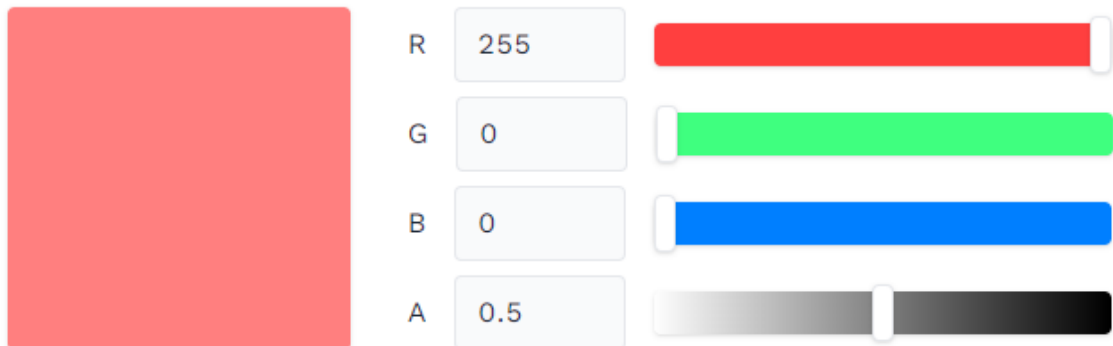
Voici un exemple avec la couleur rouge (255=1) et l'alpha à 1 :

rgba(255, 0, 0, 1) 



Et maintenant la même chose mais avec un alpha à 0.5 :

rgba(255, 0, 0, 0.5) 



```
GridLayout:
  cols: 1
  size: (root.width,root.height)
  spacing: 10

  TextInput:
    id: email
    hint_text:"Email"
    multiline: False

  TextInput:
    id: passwd
    hint_text:"mot de passe"
    multiline: False
    password: True

  Label:
    text: root.my_text
    font_size: 35
    size_hint: (1, 0.1)
    background_color: (1, 1, 1, 0)
```

J'ai fait un premier GridLayout à une colonne pour afficher l'entrée de texte pour pouvoir insérer l'email ainsi que le mot de passe.

J'ai ajouté un spacing de 10 pour aérer l'interface ainsi qu'un multiline : False pour éviter d'avoir plusieurs lignes.

Pour le password j'ai ajouté le module password :True qui permet de cacher le password.

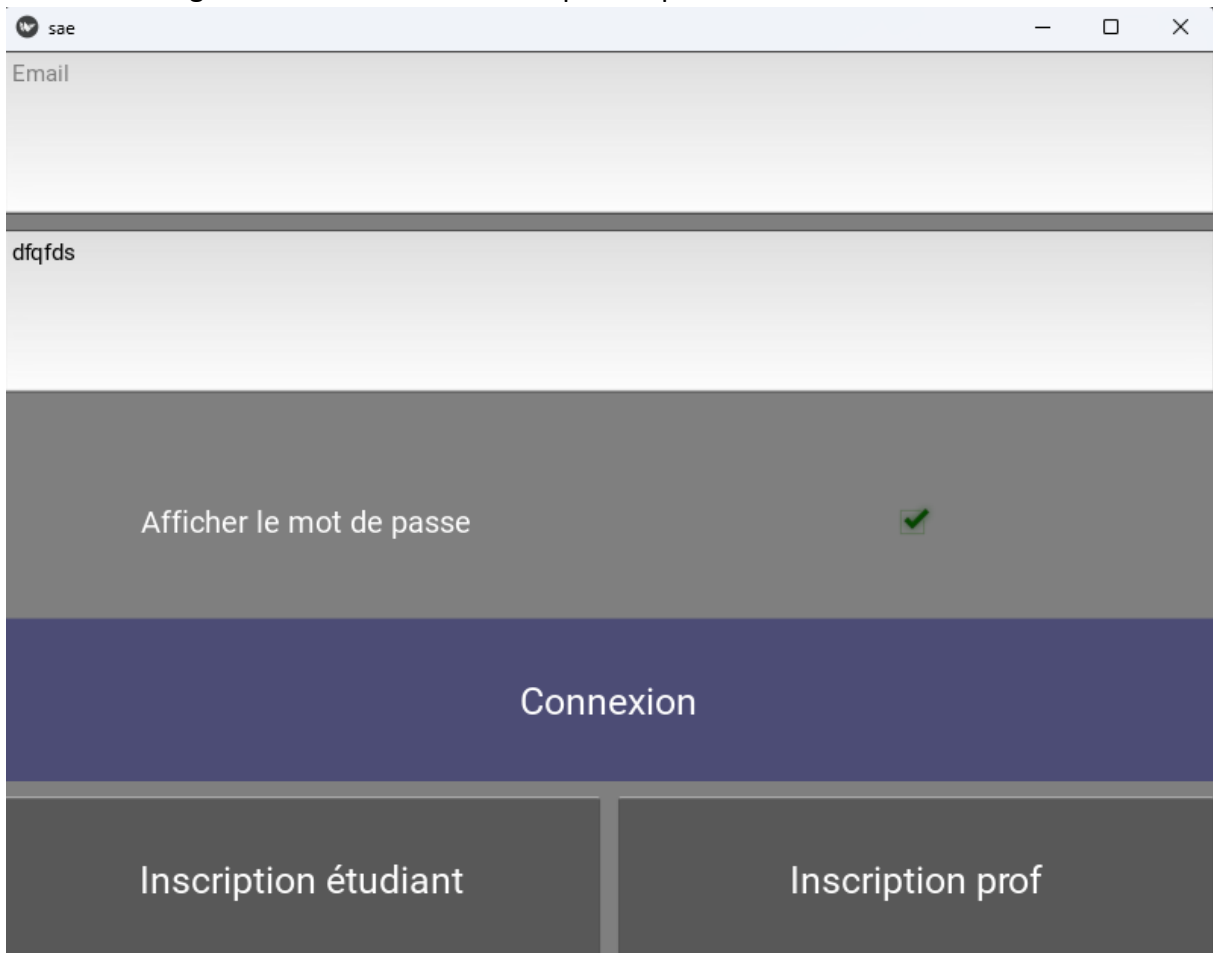
```
BoxLayout:

  Label:
    text:"Afficher le mot de passe"
```

```
font_size: 20
CheckBox:
    size_hint: (1, 1)
    on_press: app.toggle_password_visibility(self)
    active: False
    id: vision
    background_normal: ""
    background_color: (0, 0, 0.4, 0.4)
    border: (8, 8, 8, 8)
    border_radius: [20]
    color: (0.1, 0.4, 0, 1)
```

Le boxLayout intégrer au GridLayout précédent me permet d'afficher ma checkbox qui permet à l'utilisateur de voir le password qu'il rentre dans la zone de texte. Cela est possible grâce à un module python qui va changer la valeur de password : True en password :False .

J'ai aussi changé la couleur de la checkbox pour la passer en vert.



```
Button:
    id:log
```

```

        text: "Connexion"
        font_size: 25
        on_press:
            app.connex()
            app.recup_matieres_notes() if app.can_connect else
app.inactif()
            app.page_infos_etu() if app.can_connect and app.is_etu else
app.inactif()

        background_normal: ""
        background_color: (0, 0, 0.4, 0.4)
        border: (8, 8, 8, 8)
        border_radius: [20]

```

Pour le bouton de connexion je lui ai attribué une couleur mais celle-ci peut changer si la personne insère le mauvais password grâce à une fonction python :

```

#Méthode lorsque la connexion est acceptée
def accept(self):
    self.root.get_screen("Connexion").ids.my_text = "Connexion Réussie !"
    self.root.get_screen("Connexion").ids.log.background_color=(0, 0, 0.4,
0.4)

#Méthode lorsque la connexion est refusée
def refuse(self):
    self.root.get_screen("Connexion").ids.my_text = "Connexion Refusée !"
    self.root.get_screen("Connexion").ids.log.background_color=(0.6, 0, 0,
1)

def reset_color(self):
    self.root.get_screen("Connexion").ids.my_text = ""
    self.root.get_screen("Connexion").ids.log.border=(8, 8, 8, 8)

```

Ici il y a trois fonctions qui réagissent en fonction de si le password correspond bien ou pas à la base de données, en fonction de ceci elles changeront la couleur du bouton connexion.

Il est important de noter que pour agir sur une variable du fichier kv depuis le fichier python il vous faut utiliser la syntaxe suivante :

```
self.root.get_screen("Connexion").ids.my_text = "Connexion Refusée !"
```

Ici le début est le même jusqu'à "get_screen" ou vous devez spécifier dans quelle fenêtre de kivy vous voulez interagir. Enfin ids permet de spécifier que l'on veut spécifier un id et donc juste après on rentre l'id à modifier.

Et enfin pour les deux bouts d'inscription j'ai fait un GridLayout avec 2 colonnes :

```

GridLayout:
    cols: 2
    size: (root.width, root.height)
    spacing: 10
    Button:

```

```

        text: "Inscription étudiant"
        font_size: 25
        on_release:
            app.root.current = "InscriptionEtu"
            root.manager.transition = NoTransition()

    Button:
        text: "Inscription prof"
        font_size: 25
        on_release:
            app.root.current = "InscriptionProf"
            root.manager.transition.direction = "left"

```

Pour naviguer entre plusieurs fenêtres j'ai utilisé screenManager qui permet de spécifier plusieurs fenêtres et de naviguer entre ces dernières.

Voici toutes mes fenêtres :

```

WindowManager:
    Connexion:
    SecondWindow:
    InscriptionEtu:
    InscriptionProf:
    EtuHome:
    ImgEtu:
    ProfHome:

```

Pour aller par exemple dans la fenêtre InscriptionEtu, il me suffit d'entrer la commande suivante : `app.root.current= « InscriptionEtu »`

Concernant les transitions j'ai choisi de ne pas en mettre car cela ne rendait pas aussi bien que je l'imaginer j'ai donc mis un NoTransition pour ne pas en avoir.

Pour que cela fonctionne il faut impérativement dans le fichier python faire une classe pour chaque fenêtre voici un exemple de classe pour Connexion et InscriptionEtu:

```

#Classe qui gère la fenêtre de connexion
class Connexion(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
    # déclaration des variables
    my_text = StringProperty("")
    email = ObjectProperty(None)
    passwd = ObjectProperty(None)
    log = ObjectProperty(None)
    vision = ObjectProperty(None)
    bvn= ObjectProperty(None)

```

```
class SecondWindow(Screen):
    pass

#Classe pour gérer la fenêtre d'inscription pour un étudiant
class InscriptionEtu(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        #initialisation des variables
        nom = ObjectProperty(None)
        prenom = ObjectProperty(None)
        email = ObjectProperty(None)
        annee = ObjectProperty(None)
        matiere = ObjectProperty(None)
        photo = ObjectProperty(None)
        mdp = ObjectProperty(None)
        file_path= ObjectProperty(None)
        erreur = ObjectProperty(None)
```

Pour la page d'inscription Etudiant j'ai aussi utilisé un gridlayout.

Voici l'ancienne version :

Entrez votre nom:

Entrez votre prénom:

Votre Email:

Votre année d'étude:

Selectionnez votre année d'étude

Selectionnez votre spécialité :

Selectionnez une matière

Déposez une photo:

Entrez un mot de passe

Valider

Se connecter

Et la nouvelle :

Voici le code de cette fenêtre :

```
<InscriptionEtu>:
    name: "InscriptionEtu"
    nom: nom
    prenom: prenom
    email: email
    annee: annee
    matiere: matiere
    mdp: mdp
    erreur: erreur
    img:img

    canvas.before:
        Color:
            rgba: 0.5, 0.5, 0.5, 1
        Rectangle:
            pos: self.pos
            size: self.size

    GridLayout:
```

```
cols: 2
size: (root.width,root.height)
spacing: 20
padding: 20

Label:
    text: "Entrez votre nom:"
    font_size: 20
    size_hint_y: None
    height: 30

TextInput:
    id: nom
    multiline: False

Label:
    text: "Entrez votre prénom:"
    font_size: 20
    size_hint_y: None
    height: 30

TextInput:
    id: prenom
    multiline: False

Label:
    text: "Votre Email:"
    font_size: 20
    size_hint_y: None
    height: 30

TextInput:
    id: email
    multiline: False

Label:
    text: "Votre année d'étude:"
    font_size: 20
    size_hint_y: None
    height: 30

Spinner:
    id: annee
    text: "Sélectionnez votre année d'étude"
    values: ["1A", "2A"]

Label:
    text: "Sélectionnez votre spécialité :"
    font_size: 20
```



```
        size_hint_y: None
        height: 30

Spinner:
    id: matiere
    text: "Sélectionnez une matière"
    values: ["IA", "Cybersécurité"]

Label:
    text: "Sélectionnez votre image:"
    font_size: 20
    size_hint_y: None
    height: 30

Button:
    id:img
    text: "Sélectionnez votre image"
    font_size: 25
    on_release:
        app.root.current = "ImgEtu"
        root.manager.transition.direction = "right"

Label:
    text: "Entrez un mot de passe"
    font_size: 20
    size_hint_y: None
    height: 30

TextInput:
    id: mdp
    multiline: False
    password: True

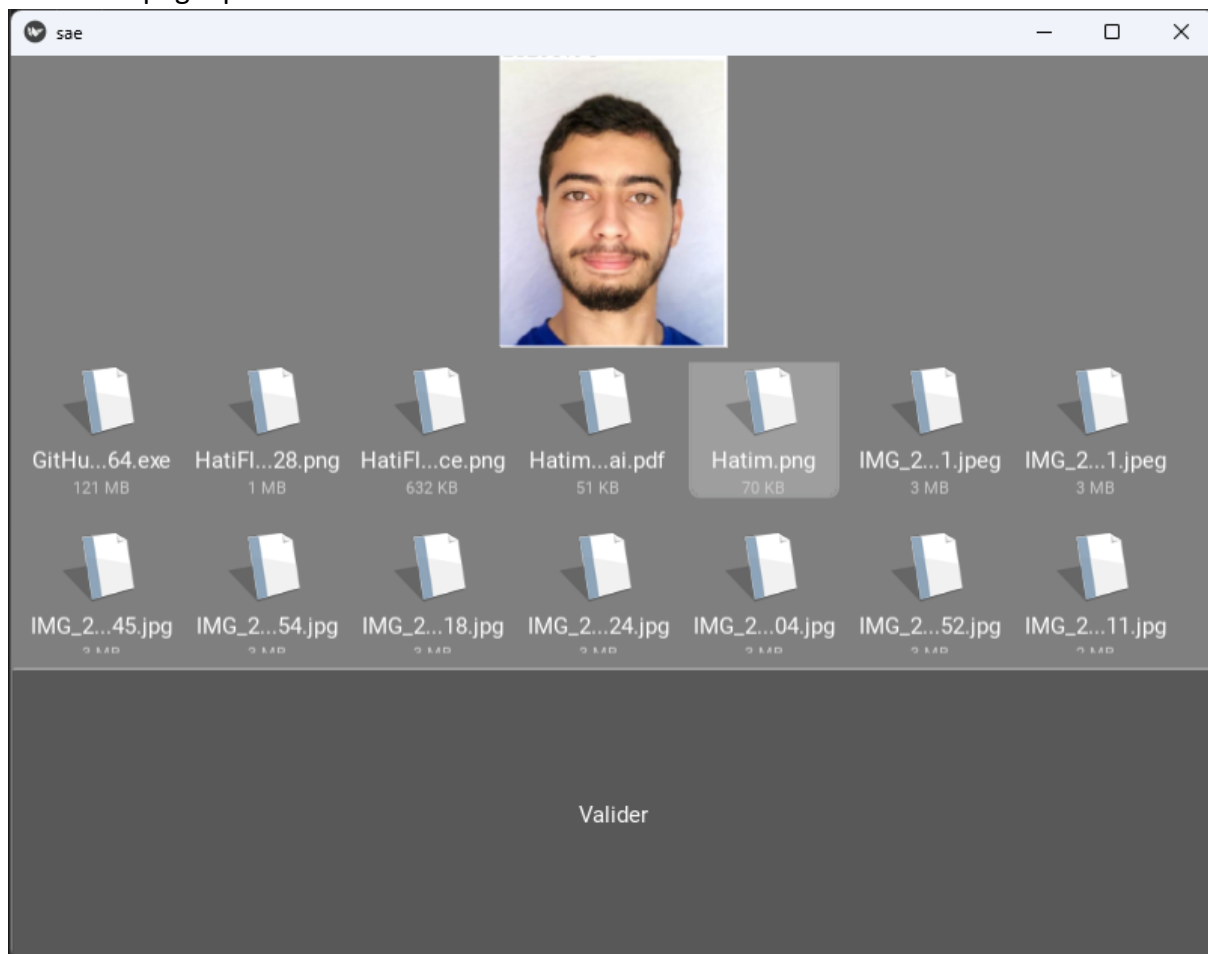
Button:
    text: "Retour"
    font_size: 25
    size_hint: 0.5, None
    height: 50
    background_color: 1, 1, 1, 1
    on_release:
        app.root.current = "Connexion"
        root.manager.transition.direction = "right"
    on_press:
        app.reinitialise_form_etu()

Button:
    text: "Valider"
```

```
        font_size: 25
        size_hint: 0.5, None
        height: 50
        background_color: 1, 1, 1, 1
        on_release:
            app.verif_formulaire_etu() #vérifie si tous les champs du
formulaire sont remplis
            app.verif_mail_etu()
            app.enregistrement_etudiant() if app.verif_etu and
app.verif_email_etu else app.inactif() #si tout est rempli, envoie des champs
vers la bdd
            app.envoi_notes() if app.verif_etu and app.verif_email_etu
else app.inactif()
            app.reinitialise_form_etu() if app.verif_etu and
app.verif_email_etu else app.inactif() #si tous est rempli, reinitialise tous
les champs
            app.same_mail_etu() if not app.verif_email_etu and
app.verif_etu else app.inactif()
            #root.copy_file(self)

    GridLayout:
        cols: 1
        Label:
            id: erreur
            text: ""
```

Ici on peut voir qu'on a un bouton « Sélectionner une image » ce bouton nous redirige vers une autre page qui est la suivante :



Nous avons un système de sélection de fichier graphique qui permet à l'utilisateur de choisir sa photo et de la prévisualiser.

```
GridLayout:
    cols: 1
    size: (root.width,root.height)
    spacing: 10
    Image:
        id: img
        source: ""
    FileChooserIconView:
        id: filechooser
        on_submit: app.on_submit(filechooser.path, filechooser.selection)
```

La source de l'image est par défaut vide mais elle est remplacée dès que l'utilisateur double clique sur une image grâce à au module on_submit, qui permet d'appeler un module et de remplacer la source par le chemin du fichier.

Voici le module en question :

```
#Permet de remplacer les backslash et d'affecter le chemin au self.e.photo
def on_submit(self, path, selection):
    selected_file_path = selection[0]
    self.root.get_screen("ImgEtu").ids.img.source= selected_file_path
    print(f'Selected file path: {selected_file_path}')
    print(self.remplacer_backslash(selected_file_path))
    self.e.set_photo(self.remplacer_backslash(selected_file_path))
    print(self.e.get_photo())
```

Lorsque l'utilisateur clique sur le bouton Valider le choix est enregistré et le texte du bouton est remplacé par le nom de l'image choisie :

The screenshot shows a web application window titled 'sae'. It contains a registration form with the following elements:

- Input field for 'Entrez votre nom:'
- Input field for 'Entrez votre prénom:'
- Input field for 'Votre Email:'
- Dropdown menu for 'Votre année d'étude:' with the text 'Sélectionnez votre année d'étude'
- Dropdown menu for 'Sélectionnez votre spécialité:' with the text 'Sélectionnez une matière'
- Dropdown menu for 'Sélectionnez votre image:' which currently shows 'Hatim.png'. A red arrow points to this dropdown.
- Input field for 'Entrez un mot de passe'
- 'Retour' button
- 'Valider' button

Voici le module qui permet de récupérer le nom du fichier en splitant le chemin et en récupérant uniquement le dernier élément :

```
def change_text(self):
    chemin = self.e.get_photo().split("\\")

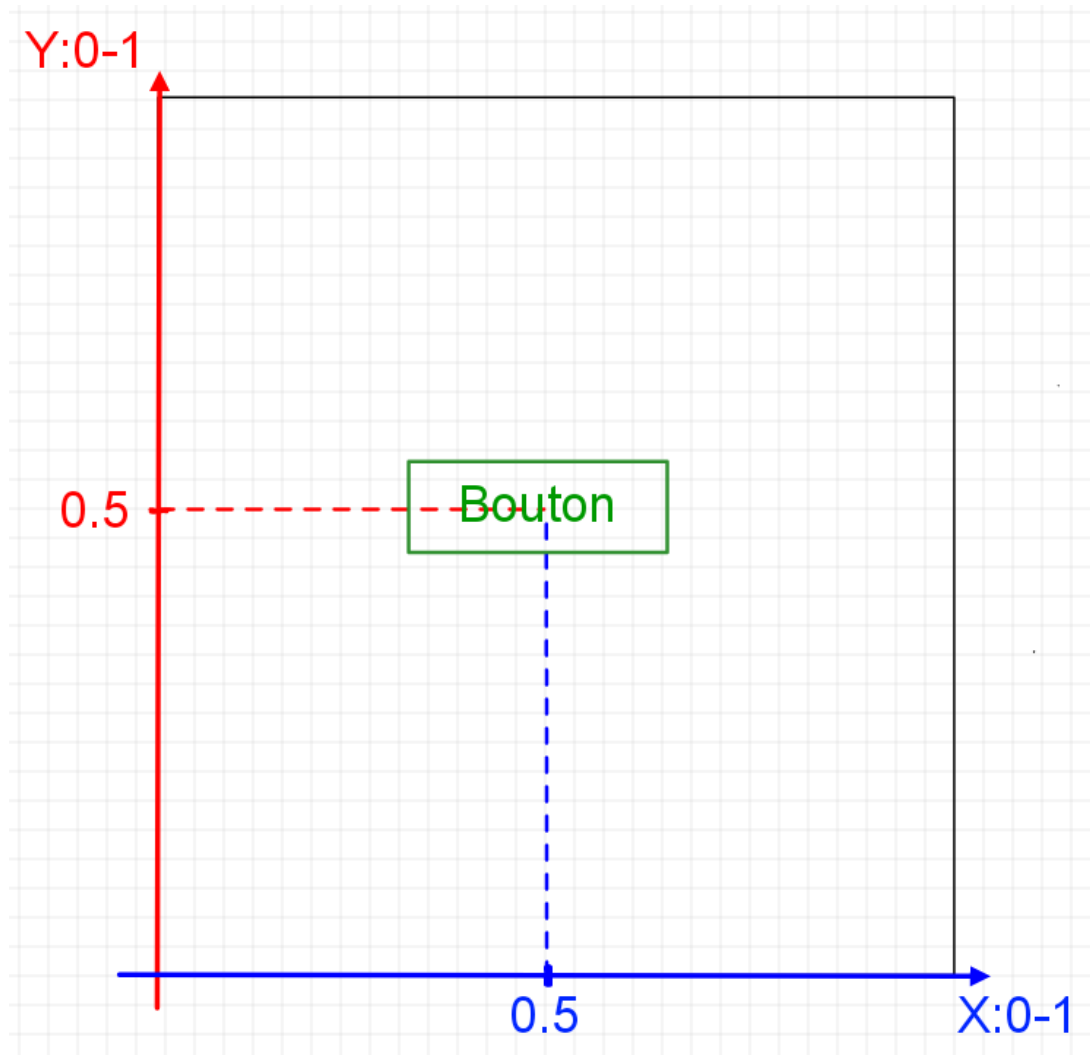
    self.root.get_screen("InscriptionEtu").ids.img.text = chemin[-1]
```

Pour la page d'inscription des profs je n'ai pas utilisé de GridLayout mais plutôt un RelativeLayout.

Comment fonctionne le RelativeLayout?

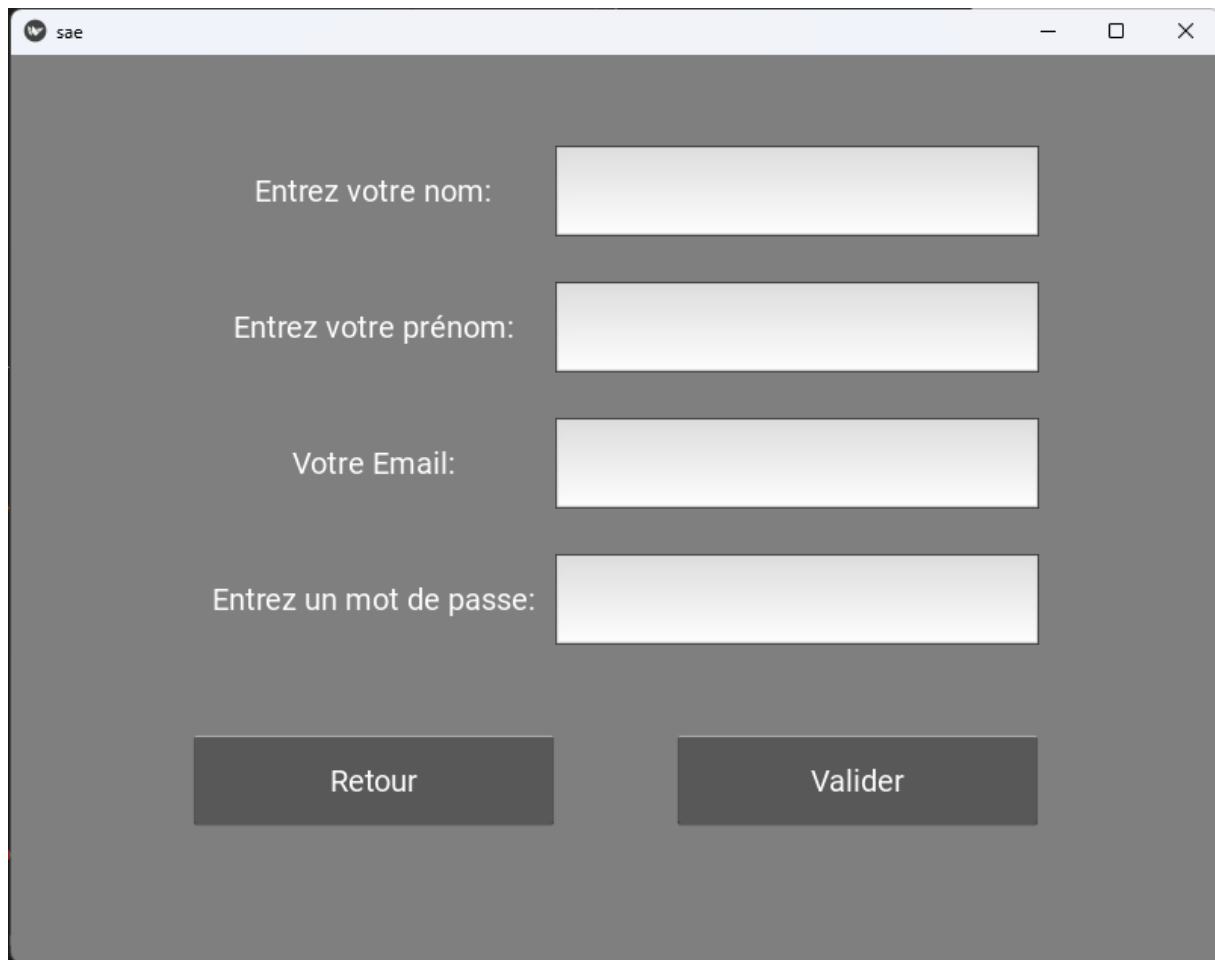
Le RelativeLayout permet de placer avec un axe X et Y un widget ou l'on souhaite dans la fenêtre Kivy les valeurs sont comprise entre 0 et 1 ce sont des pourcentages ce qui permet à ces derniers de ne pas être adapter qu'à un type de resolution mais à plusieurs écrans.

Voici un schema pour illustrer mes propos:



Comme vous pouvez le voir pour placer un bouton au centre de ma fenêtre il me suffit de mettre les valeurs X et Y à 0.5. L'avantage de cette méthode est donc de pouvoir placer à notre guise.

Voici donc à quoi ressemble la page d'inscription prof:



Entrez votre nom:

Entrez votre prénom:

Votre Email:

Entrez un mot de passe:

Voici le code kv de cette page :

```
<InscriptionProf>:
  name: "InscriptionProf"
  nom: nom
  prenom: prenom
  email: email
  mdp: mdp
  erreur: erreur

  canvas.before:
    Color:
      rgba: 0.5, 0.5, 0.5, 1
    Rectangle:
      pos: self.pos
      size: self.size

  RelativeLayout:
    orientation: 'vertical'
```

```
pos : self.pos
size : root.size
Label:
    text: "Entrez votre nom:"
    font_size:20
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.30, 'center_y':.85}

TextInput:
    id: nom
    multiline: False
    markup: True
    font_size:20
    size_hint: 0.4,0.1
    pos_hint: {'center_x':.65, 'center_y':.85}

Label:
    text: "Entrez votre prénom:"
    font_size:20
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.30, 'center_y':.70}

TextInput:
    id: prenom
    multiline: False
    font_size:20
    size_hint: 0.4,0.1
    pos_hint: {'center_x':.65, 'center_y':.70}

Label:
    text: "Votre Email:"
    font_size:20
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.30, 'center_y':.55}

TextInput:
    id: email
    multiline: False
    font_size:20
    size_hint: 0.4,0.1
    pos_hint: {'center_x':.65, 'center_y':.55}

Label:
    text: "Entrez un mot de passe:"
    font_size:20
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.30, 'center_y':.40}
```

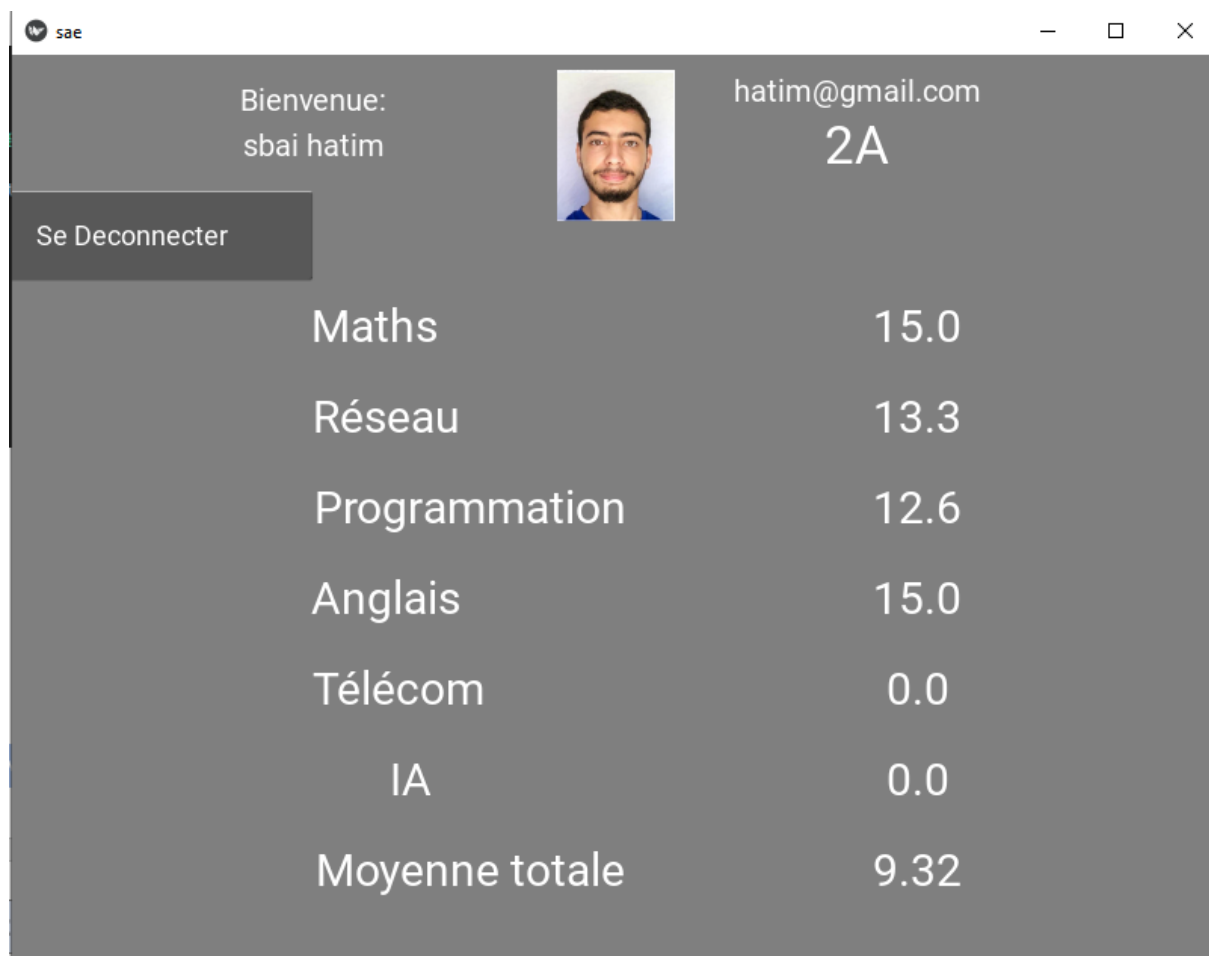
```
TextInput:
    id: mdp
    multiline: False
    password: True
    font_size:20
    size_hint: 0.4,0.1
    pos_hint:{'center_x':.65, 'center_y':.40}

Button:
    text: "Retour"
    font_size:20
    size_hint: 0.3,0.1
    pos_hint:{'center_x':.3, 'center_y':.2}
    background_color: 1, 1, 1, 1
    on_release:
        app.root.current = "Connexion"
        root.manager.transition.direction = "right"
    on_press:
        app.reinitialise_form_prof()

Button:
    text: "Valider"
    font_size:20
    size_hint: 0.3,0.1
    pos_hint:{'center_x':.7, 'center_y':.2}
    background_color: 1, 1, 1, 1
    on_release:
        app.verif_formulaire_prof() #vérifie si tous les champs du
formulaire sont remplis
        app.verif_mail_prof()
        app.enregistrement_prof() if app.verif_prof and
app.verif_email_prof else app.inactif() #si tout est rempli, envoie des champs
vers la bdd
        app.reinitialise_form_prof() if app.verif_prof and
app.verif_email_prof else app.inactif() #si tous est rempli, reinitialise tous
les champs
        app.same_mail_prof() if not app.verif_email_prof and
app.verif_prof else app.inactif()

Label:
    id: erreur
    text: ""
```


Quand un étudiant se connecte voici son interface :



Sa photo est affichée ainsi que ses données personnelles comme son nom, prénom, mail et année d'étude. Bien sûr on retrouve les matières de l'étudiant et ses notes avec une moyenne générale. Il y a également un bouton pour se déconnecter de sa session.

Voici le code kv de cette interface vous remarquerez qu'elle comporte plus de ligne que les autres fenêtres. Cela est dû au fait qu'il y a beaucoup plus d'éléments à afficher ainsi que les positions exactes du RelativeLayout :

```
<EtuHome>:
    name : "EtuHome"
    bvn:bvn
    math:math
    reseau:reseau
    telecom:telecom
    prog:prog
    anglais:anglais
    name_specialite: name_specialite
    specialite: specialite
    photo: photo
    mail:mail
    annee:annee
```

```
moyenne:moyenne
canvas.before:
    Color:
        rgba: 0.5, 0.5, 0.5, 1
    Rectangle:
        pos: self.pos
        size: self.size
RelativeLayout:
    orientation:'vertical'

    pos : self.pos
    size : root.size

    Image:
        id: photo
        source: ''
        size_hint:0.4,None
        pos_hint:{'center_x':.5, 'center_y':.9}

    Label:
        text: "Bienvenue:"
        markup: True
        font_size:20
        size_hint: 0.2,0.2
        pos_hint:{'center_x':.25, 'center_y':.95}

    Label:
        id:bvn
        text: ""
        markup: True
        font_size:20
        size_hint: 0.2,0.2
        pos_hint:{'center_x':.25, 'center_y':.90}

    Label:
        id:mail
        text: ""
        markup: True
        font_size:20
        size_hint: 0.2,0.2
        pos_hint:{'center_x':.7, 'center_y':.96}

    Label:
        id:annee
        text: ""
        markup: True
        font_size:35
        size_hint: 0.2,0.2
        pos_hint:{'center_x':.7, 'center_y':.90}
```

```
Button:
    text: "Se Deconnecter"
    font_size: 18
    size_hint: .3,.1
    pos_hint: {'center_x':.1, 'center_y':.8}
    height: 20
    background_color: 1, 1, 1, 1
    on_release:
        app.reinitialise_champs_connexion()
        app.root.current = "Connexion"
        root.manager.transition.direction = "right"

Label:
    text:"Maths"
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.30, 'center_y':.70}
Label:
    id:math
    text:""
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.75, 'center_y':.70}
Label:
    text:"Réseau"
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.31, 'center_y':.60}
Label:
    id:reseau
    text:""
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.75, 'center_y':.60}
Label:
    text:"Programmation"
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.38, 'center_y':.50}
Label:
    id:prog
    text:""
    markup: True
```

```
font_size:30
size_hint: 0.2,0.2
pos_hint:{'center_x':.75, 'center_y':.50}
Label:
    text:"Anglais"
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint:{'center_x':.31, 'center_y':.40}
Label:
    id:anglais
    text:""
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint:{'center_x':.75, 'center_y':.40}
Label:
    text:"Télécom"
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint:{'center_x':.32, 'center_y':.30}
Label:
    id:telecom
    text:""
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint:{'center_x':.75, 'center_y':.30}

Label:
    id: name_specialite
    text:""
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint:{'center_x':.33, 'center_y':.20}
Label:
    id:specialite
    text:""
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint:{'center_x':.75, 'center_y':.20}
Label:
    text:"Moyenne totale"
    markup: True
    font_size:30
    size_hint: 0.2,0.2
```

```
pos_hint: {'center_x': .38, 'center_y': .10}
Label:
    id: moyenne
    text: ""
    markup: True
    font_size: 30
    size_hint: 0.2, 0.2
    pos_hint: {'center_x': .75, 'center_y': .10}
Label:
    id: moyenne
    text: ""
    markup: True
    font_size: 30
    size_hint: 0.2, 0.2
    pos_hint: {'center_x': .75, 'center_y': .10}
```

Concernant la fenêtre de la page des profs c'est celle qui intègre le plus de fonctionnalités :

The screenshot shows a web application window titled 'sae'. The interface is dark-themed. It features a search bar with the label 'Entrez le nom d'un étudiant' and a 'Valider' button. Below this, there's another search bar with the same label. In the center, there's a section for 'Modifiez une note:' with an input field, a 'Sélectionnez la matière' dropdown menu, and a 'modifier' button. At the bottom, there's a 'Se déconnecter' button.

Ici on retrouve un module de recherche de l'étudiant avec son nom. Si on ne rentre rien et qu'on valide on aura un message d'erreur comme ci-dessus.

Si celui-ci n'est pas présent dans la base de données alors on aura le message d'erreur suivant :

The screenshot shows a web application window titled 'sae'. The interface has a dark gray background. At the top, there is a form with the label 'Entrez le nom d'un étudiant' and a text input field containing 'sbai'. To the right of the input field is a 'Valider' button. Below this, a message is displayed: 'Cet étudiant n'est pas présent dans la base de données'. Further down, there is another form with the label 'Modifiez une note:', an empty text input field, a 'Sélectionnez la matière' button, and a 'modifier' button. At the bottom center, there is a 'Se déconnecter' button.

Et enfin quand on rentre le nom d'un étudiant qui est bien présent dans la base de données on obtient toute ces données qui sont récupérer depuis le serveur sql :

The screenshot shows a web application window titled 'sae'. It features a search bar with the text 'Entrez le nom d'un étudiant' and a button 'Valider'. Below this, the student's details are displayed: '2A', 'kenan@gmail.com', and 'kenan' next to a profile picture. There is a section for 'Modifiez une note:' with an input field and a 'Sélectionnez la matière' dropdown, followed by a 'modifier' button. A table lists the student's grades for various subjects: Maths (13.0), Réseau (5.25), Programmation (10.25), Anglais (12.75), Télécom (1.75), and Cybersécurité (11.75). At the bottom, there is a 'Se déconnecter' button.

Matière	Note
Maths	13.0
Réseau	5.25
Programmation	10.25
Anglais	12.75
Télécom	1.75
Cybersécurité	11.75

On peut voir les note de chacune des matières ainsi que son prénom avec sa photo son année ainsi que son adresse mail.

Vue que nous somme sur le compte d'un professeur il en va de pouvoir modifier la note de l'étudiant. J'ai donc ajouté des widgets qui permet de modifier la note de la matière choisi directement depuis la fenêtre du programme.

Voici le code de l'interface kivy :

Maxime :

Je me suis occupé de la majeure partie du code python et des méthodes qui permettent la bonne fonctionnalité de l'application. Ces méthodes sont déclenchées lors d'événements, dans notre cas c'est-à-dire lorsqu'un utilisateur appuie sur un bouton.

Dans cette partie du rapport je vais donc expliquer le code que j'ai développé.

Explication du code :

```
# importations
from kivy.app import App
import shutil
from kivy.uix.screenmanager import ScreenManager, Screen, NoTransition
from kivy.lang import Builder
from kivy.properties import ObjectProperty, StringProperty
from kivy.core.window import Window
from Etudiant import Etudiant
from Prof import Prof
import mysql.connector
from hashlib import sha512
import random
from get_bina import get_binary_data
from recup_photo import recup_photo
```

J'ai commencé par importer les modules, les classes et les fonctions dont j'allais avoir besoin

`from kivy.app import App` importe la classe `App` du module `kivy.app`, qui est utilisée pour créer une application Kivy.

`import shutil` importe le module `shutil` qui fournit des fonctionnalités pour travailler avec les systèmes de fichiers (par exemple pour copier, déplacer ou supprimer des fichiers).

`from kivy.uix.screenmanager import ScreenManager, Screen, NoTransition` importe les classes `ScreenManager`, `Screen` et `NoTransition` du module `kivy.uix.screenmanager`. Ces classes sont utilisées pour gérer les différents écrans d'une application Kivy.

`from kivy.lang import Builder` importe la classe `Builder` du module `kivy.lang`, qui permet de créer des interfaces utilisateur Kivy à partir de fichiers KV (Kivy Language).

`from kivy.properties import ObjectProperty, StringProperty` importe les classes `ObjectProperty` et `StringProperty` du module `kivy.properties`, qui sont utilisées pour définir des propriétés liées aux widgets Kivy.

`from kivy.core.window import Window` importe la classe `Window` du module `kivy.core.window`, qui permet de gérer les fenêtres de l'application.

`from Etudiant import Etudiant` importe la classe `Etudiant` depuis le fichier `Etudiant.py`

`from Prof import Prof` importe la classe `Prof` depuis le fichier `Prof.py`

`import mysql.connector` importe le module `mysql.connector` qui permet de se connecter à une base de données MySQL.

`from hashlib import sha512` importe la fonction `sha512` du module `hashlib` qui permet de créer des hachages (empreintes numériques) à partir de données.

`import random` importe le module `random` qui fournit des fonctions pour générer des nombres aléatoires.

from get_bina import get_binary_data importe la fonction get_binary_data depuis le fichier get_bina.py

from recup_photo import recup_photo importe la fonction recup_photo depuis le fichier recup_photo.py

```
class sae(App):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        #initialisation des variables globales
        self.can_connect = False
        self.whos_connect = ""
        self.is_etu = False
        self.is_prof = False
        self.verif_etu = True
        self.verif_email_etu = True
        self.specialisation = ""

    #instanciations
    e = Etudiant()
    p = Prof()
```

Je définis une classe « sae » qui hérite de la classe « App ». Cette classe est la classe principale du code, elle contient toutes les fonctions qui sont appelés lors des différents événements

J'initialise ensuite plusieurs « variables globales ». Ce que j'entends par ce terme c'est que ces variables sont utilisées dans plusieurs fonctions de la classe.

Puis je crée des instances de 2 classes.

La classe Etudiant :

```
from personne import Personne

class Etudiant(Personne):
    #constructeur de la classe etudiant
    def __init__(self):
        Personne.__init__(self)
        self.__annee = 0
        self.__matiere = ""
        self.__photo = ""
```

```
self.__mdp = ""
self.__infos_etu = {}

#accesseurs et mutateurs
def get_annee(self):
    return self.__annee

def set_annee(self, nv_annee):
    self.__annee = nv_annee

def get_matiere(self):
    return self.__matiere

def set_matiere(self, nv_matiere):
    self.__matiere = nv_matiere

def get_photo(self):
    return self.__photo

def set_photo(self, nv_photo):
    self.__photo = nv_photo

def get_mdp(self):
    return self.__mdp

def set_mdp(self, nv_mdp):
    self.__mdp = nv_mdp

def get_infos_etu(self):
    return self.__infos_etu

def set_infos_etu(self, nv_infos_etu):
    self.__infos_etu = nv_infos_etu

#méthode pour afficher les infos d'un étudiant
def Affiche(self):
    liste_personne = Personne.Affiche(self)
    return [liste_personne[0], liste_personne[1], liste_personne[2],
self.__annee, self.__matiere, self.__photo, self.__mdp]
```

La classe Prof

```
from personne import Personne

class Prof(Personne):
    #constructeur de la classe etudiant
    def __init__(self):
        Personne.__init__(self)
        self.__mdp = ""
        self.__infos_etu = []

    #accesseurs et mutateurs
    def get_mdp(self):
        return self.__mdp

    def set_mdp(self, nv_mdp):
        self.__mdp = nv_mdp

    def get_infos_etu(self):
        return self.__infos_etu

    def set_infos_etu(self, nv_infos_etu):
        self.__infos_etu = nv_infos_etu

    #méthode pour afficher les infos d'un étudiant
    def Affiche(self):
        liste_personne = Personne.Affiche(self)
        return [liste_personne[0], liste_personne[1], liste_personne[2],
self.__mdp]
```

Ces classes héritent de la classe Personne

La classe Personne :

```
class Personne :
    #constructeur de la classe personne
    def __init__(self):
        self.__nom = ""
        self.__prenom = ""
        self.__email = ""

    # accesseurs et mutateurs
    def get_nom(self):
        return self.__nom

    def set_nom(self, nv_nom):
        self.__nom = nv_nom
```

```
def get_prenom(self):  
    return self.__prenom  
  
def set_prenom(self, nv_prenom):  
    self.__prenom = nv_prenom  
  
def get_email(self):  
    return self.__email  
  
def set_email(self, nv_email):  
    self.__email = nv_email  
  
#méthode pour afficher les infos de la personne  
def Affiche(self):  
    return [self.__nom, self.__prenom, self.__email]
```

La classe Etudiant permet de créer une instance qui va stocker les informations d'un étudiant qui s'inscrit (nom, prénom, email, année d'étude, spécialité, photo, mot de passe) pour les envoyer dans la base de données

La classe Prof permet de créer une instance qui va stocker les informations d'un professeur qui s'inscrit (nom, prénom, email, mot de passe) ainsi que les informations de tous les étudiants présents dans la base de données

Méthodes de la classe

Page d'inscription étudiant

Entrez votre nom:

Entrez votre prénom:

Votre Email:

Votre année d'étude:

Sélectionnez votre spécialité :

Sélectionnez votre image:

Entrez un mot de passe:

Bouton « Valider »

Entrez votre nom:

Entrez votre prénom:

Votre Email:

Votre année d'étude:

Sélectionnez votre spécialité :

Sélectionnez votre image:

Entrez un mot de passe

Button:

```
text: "Valider"
font_size: 25
size_hint: 0.5, None
height: 50
background_color: 1, 1, 1, 1
on_release:
    app.verif_formulaire_etu() #vérifie si tous les champs du
formulaire sont remplis
    app.verif_mail_etu()
    app.enregistrement_etudiant() if app.verif_etu and
app.verif_email_etu else app.inactif() #si tout est rempli, envoie des champs
vers la bdd
    app.envoi_notes() if app.verif_etu and app.verif_email_etu
else app.inactif()
```

```
app.reinitialise_form_etu() if app.verif_etu and
app.verif_email_etu else app.inactif() #si tous est rempli, reinitialise tous
les champs
app.same_mail_etu() if not app.verif_email_etu and
app.verif_etu else app.inactif()
```

Quand le bouton « Valider » est relâché, plusieurs méthodes sont enclenchées :

Verif formulaire etu() :

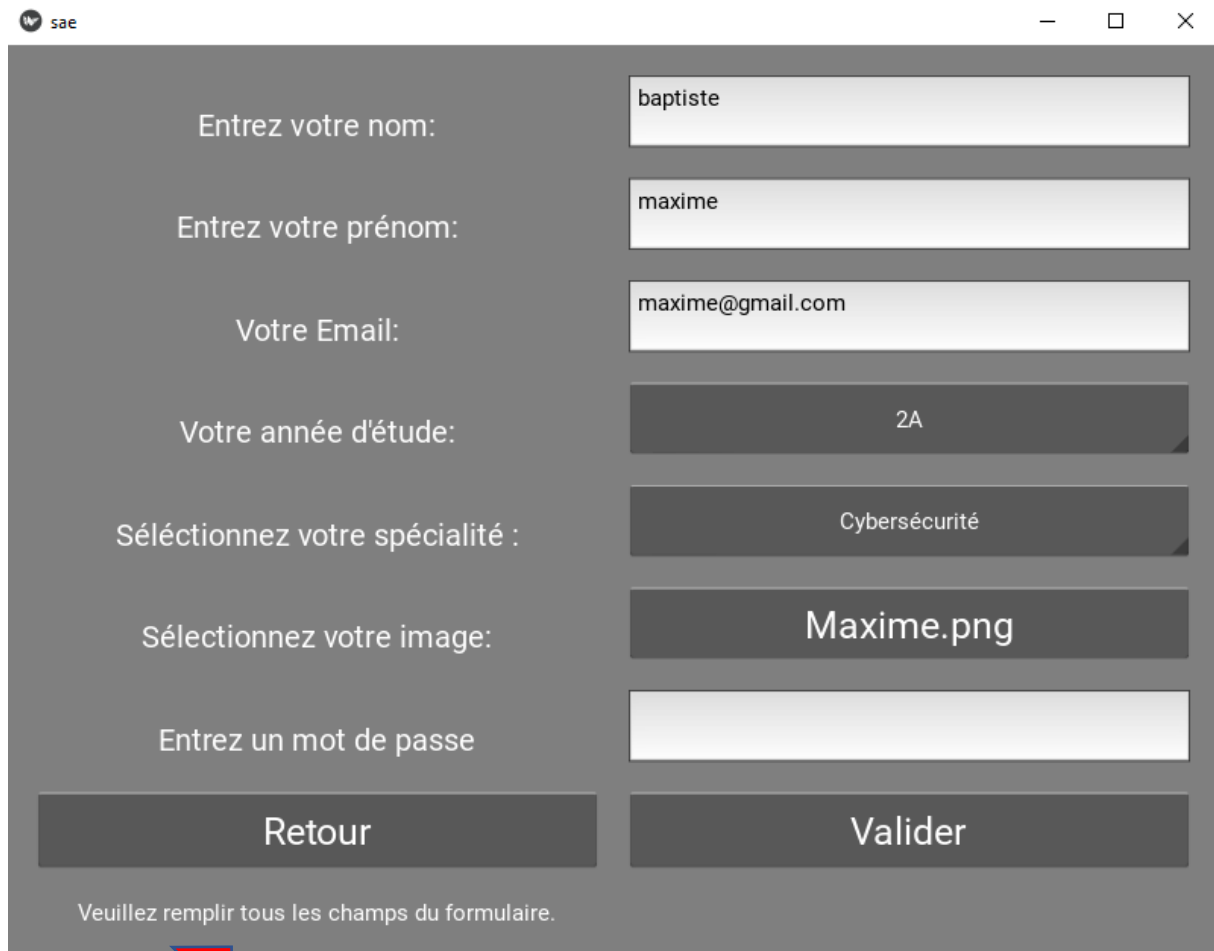
```
#Méthode pour vérifier que tous les champs du formulaire sont remplis
def verif_formulaire_etu(self):
    #initialisation à True de la variable verif à chaque vérification
    self.verif_etu = True
    #si un seul des champs n'est pas rempli
    if self.root.get_screen("InscriptionEtu").ids.nom.text == "" or
self.root.get_screen("InscriptionEtu").ids.prenom.text == "" or
self.root.get_screen("InscriptionEtu").ids.email.text == "" or
self.root.get_screen("InscriptionEtu").ids.annee.text == "Sélectionnez votre
année d'étude" or self.root.get_screen("InscriptionEtu").ids.matiere.text ==
"Sélectionnez une matière" or
self.root.get_screen("InscriptionEtu").ids.mdp.text == "" or
self.root.get_screen("InscriptionEtu").ids.img.text == "Sélectionnez votre
image":
        #envoi d'un message d'erreur
        self.root.get_screen("InscriptionEtu").ids.erreur.text = "Veuillez
remplir tous les champs du formulaire."
        #variable verif à False
        self.verif_etu = False
```

Cette méthode permet de vérifier que tous les champs soient bien remplis par l'étudiant lors de son inscription. Dans le cas contraire, un message d'erreur s'affiche sur la page. La variable self.verif.etu permet au programme de savoir si tous les champs ont bien été remplis. Elle vaut True si c'est le cas sinon elle vaut False

Pour vérifier ceci, on récupère la valeur du texte de chaque champ du formulaire lorsque l'étudiant clique sur « Valider ».

Si la valeur du texte d'un des champs parmi (Nom, Prénom, Email, Mot de passe) est égal à "" cela veut dire qu'il est vide et donc que l'étudiant ne l'a pas rempli.

Si la valeur du texte du champ année vaut « Sélectionnez votre année d'étude » ou si la valeur du texte du champ matière vaut « Sélectionnez une matière » ou si la valeur du texte du champ image vaut « Sélectionnez votre image », cela veut dire que l'utilisateur n'a pas sélectionné les informations demandées.



sae

Entrez votre nom: baptiste

Entrez votre prénom: maxime

Votre Email: maxime@gmail.com

Votre année d'étude: 2A

Sélectionnez votre spécialité : Cybersécurité

Sélectionnez votre image: Maxime.png

Entrez un mot de passe

Retour Valider

Veuillez remplir tous les champs du formulaire.

Message d'erreur

Verif_mail_etu() :

```
#Méthode pour vérifier que le mail entré par l'utilisateur n'est pas déjà
utilisé
def verif_mail_etu(self):
    #initialisation de la variable de verif à True
    self.verif_email_etu = True
    try:
        connection_params = {
            'host' : "172.20.10.3",
            'user' : "admin",
            'password' : "sae302",
            'database' : "doss_etu"
        }
    except mysql.connector.Error as e:
```

```
        print("Exception : ", e)
    else:
        #requête pour obtenir les mails des etudiants et des profs
        request = """select email from etudiants;"""
        request2 = """select email from profs;"""

        with mysql.connector.connect(**connection_params) as db :
            with db.cursor() as c:
                c.execute(request)
                rq = c.fetchall()
                c.execute(request2)
                rq2 = c.fetchall()
            db.commit()
            db.close()

        #parcours de chaque mail, si le mail existe dans la BDD : variable
de verif passe à FALSE
        for elem in rq:
            if elem[0] ==
self.root.get_screen("InscriptionEtu").ids.email.text:
                self.verif_email_etu = False

        #parcours de chaque mail, si le mail existe dans la BDD : variable
de verif passe à FALSE
        for elem in rq2:
            if elem[0] ==
self.root.get_screen("InscriptionEtu").ids.email.text:
                self.verif_email_etu = False
```

Cette méthode permet de vérifier que le mail entré par l'étudiant lors de l'inscription n'est pas déjà utilisé. En effet, on a décidé que le mail étudiant serait ce qui permettrait d'identifier uniquement un étudiant au niveau de la base de données. Par conséquent il ne peut pas y avoir 2 même adresses email dans la table étudiant ou dans la table prof de la base de données.

Si le mail entré par l'étudiant est déjà présent dans la base de données, il y a un nouveau message d'erreur qui s'affiche.

La variable `self.verif_email_etu` permet au programme de savoir si le mail est unique. Il vaut `True` de base.

Pour vérifier ceci, on récupère tous les mails des étudiants et des profs au travers de deux requêtes sql, on récupère ce que l'utilisateur a entré comme mail et on le compare avec tous les mails étudiants et profs. S'il est égal à un de ces mails, la variable `self.verif.email_etu` passe à False

Entrez votre nom: baptiste

Entrez votre prénom: maxime

Votre Email: maxime@gmail.com

Votre année d'étude: 2A

Sélectionnez votre spécialité : Cybersécurité

Sélectionnez votre image: Maxime.png

Entrez un mot de passe: *****

Retour Valider

Ce mail est déjà utilisé

Message d'erreur

Ce message d'erreur est généré par la fonction `same_mail_etu()`

```
#Méthode d'envoi d'un message d'erreur si le mail est déjà utilisé
def same_mail_etu(self):
    self.root.get_screen("InscriptionEtu").ids.erreur.text = "Ce mail est déjà utilisé"
```

Cette fonction se déclenche uniquement si `self.verif_mail_etu = False` et que `self.verif_etu = True`, c'est-à-dire que tous les champs ont été remplis et que le mail entré par l'utilisateur est déjà présent dans la BDD

Enregistrement_etudiant() :

```

#Méthode pour la création d'un compte étudiant
def enregistrement_etudiant(self):
    #ajout des informations de l'étudiant
    self.e.set_nom(self.root.get_screen("InscriptionEtu").ids.nom.text)
    self.e.set_prenom(self.root.get_screen("InscriptionEtu").ids.prenom.te
xt)
    self.e.set_email(self.root.get_screen("InscriptionEtu").ids.email.text
)
    self.e.set_annee(self.root.get_screen("InscriptionEtu").ids.annee.text
)
    self.e.set_matiere(self.root.get_screen("InscriptionEtu").ids.matiere.
text)
    mdp_crypt =
sha512(self.root.get_screen("InscriptionEtu").ids.mdp.text.encode()).hexdigest
()
    self.e.set_mdp(mdp_crypt)
try:
    connection_params = {
        'host' : "172.20.10.3",
        'user' : "admin",
        'password' : "sae302",
        'database' : "doss_etu"
    }
except mysql.connector.Error as e:
    print("Exception : ", e)
else:
    #envoi des données de l'étudiant vers la BDD
    request = """insert into
etudiants(nom,prenom,annee,spe,photo,email,mdp)
values(%s,%s,%s,%s,%s,%s,%s)"""
    params = (self.e.get_nom(), self.e.get_prenom(),
self.e.get_annee(), self.e.get_matiere(), get_binary_data(self.e.get_photo()),
self.e.get_email(), self.e.get_mdp())

    #ajout dans la table 'moy_gen'
    request2 = """insert into moy_gen(nom,prenom,mail_etu)
values(%s,%s,%s)"""
    params2 = (self.e.get_nom(), self.e.get_prenom(),
self.e.get_email())

    with mysql.connector.connect(**connection_params) as db :
        with db.cursor() as c:
            c.execute(request,params)
            c.execute(request2,params2)
        db.commit()
        db.close()

```

Cette méthode permet la création d'un compte étudiant. Elle est lancée uniquement si les variables `self.verif_etu` et `self.verif_mail_etu` sont égales à `True`, c'est-à-dire que lors de son inscription, l'étudiant a rempli tous les champs et a renseigné un email qui n'a pas déjà été utilisé pour la création d'un autre compte.

Lors de la création du compte, on envoie les informations de l'étudiant dans l'instanciation de la classe `Etudiant` avec la méthode `set` puis on récupère ces informations pour les envoyer dans la table « étudiants » de la base de données.

On envoie également les informations nom, prénom et email dans la table « moy_gen »

Avant d'être envoyé dans la base de données, le mot de passe de l'étudiant est crypté avec l'algorithme de hachage « sha512 » pour qu'il n'apparaisse pas en clair dans la base de données.

C'est la ligne `sha512(self.root.get_screen("InscriptionEtu").ids.mdp.text.encode()).hexdigest()` qui permet ceci :

La fonction `sha512()` est utilisée pour créer un objet de hachage SHA-512.

`self.root.get_screen("InscriptionEtu").ids.mdp.text` récupère la valeur saisie dans le champ du mot de passe

`.encode()` est utilisé pour encoder la chaîne en bytes.

`.hexdigest()` est utilisé pour renvoyer la valeur de hachage sous forme de chaîne hexadécimale.

Il faut importer la fonction `sha512` du module `hashlib`

```
from hashlib import sha512
```

nom	prenom	email	annee	spe
baptiste	maxime	maxime@gmail.com	2A	Cybersécurité

```
mdp
e99da17eeab04d10efc086ad71d672c4cab6c29f7768047cf2a3470886ec2a25381eb0fa892be0adc473750000dc921865a283b83222014a5bc3b46bbaf970d8
```

On peut voir que les informations sont bien envoyées vers la table étudiants et que le mot de passe est bien haché. (Ici je n'ai pas affiché la colonne photo car les données sont beaucoup trop grandes)

Envoie_notes() :

```
def envoie_notes(self):

    #récupère le mail et la spé de l'étudiant
    email = self.e.get_email()
    spe = self.e.get_matiere()

    #liste de toutes les matières de l'étudiant
    matieres = ["Maths", "Programmation", "Réseau", "Télécom", "Anglais",
spe]

    try:
        connection_params = {
            'host' : "172.20.10.3",
            'user' : "admin",
            'password' : "sae302",
            'database' : "doss_etu"
        }
    except mysql.connector.Error as e:
        print("Exception : ", e)
    else:
        with mysql.connector.connect(**connection_params) as db :
            with db.cursor() as c:
                for i in range(0, len(matieres)):

                    #envoi des données de l'étudiant vers la BDD
                    request = """insert into notes(mail_etu, matiere,
note) values(%s,%s,%s)"""
                    params = (email, matieres[i],
round(random.uniform(0, 20) * 4) / 4)
                    c.execute(request,params)
                    db.commit()
            db.close()
```

Cette méthode permet de générer des notes aléatoires dans toutes les matières de l'étudiant et de les envoyer dans la table « notes » de la BDD lors de l'inscription. Elle permet le bon fonctionnement du code.

On veut générer des notes aléatoires qui sont des nombres décimaux allant de 0 à 20 pouvant contenir après la virgule que [,25 ,5 ,75]

Ceci se fait avec la ligne : `round(random.uniform(0, 20) * 4) / 4)`

random.uniform(0, 20) génère un nombre flottant aléatoire entre 0 et 20.

round() arrondi le nombre à l'entier le plus proche

Ensuite on multiplie par 4 et divise par 4 pour avoir les nombres après la virgule voulu.

Reinitialise form etu() :

```
#Méthode pour réinitialiser les champs du formulaire
def reinitialise_form_etu(self):
    self.root.get_screen("InscriptionEtu").ids.nom.text = ""
    self.root.get_screen("InscriptionEtu").ids.prenom.text = ""
    self.root.get_screen("InscriptionEtu").ids.email.text = ""
    self.root.get_screen("InscriptionEtu").ids.annee.text = "Sélectionnez
votre année d'étude"
    self.root.get_screen("InscriptionEtu").ids.matiere.text =
"Sélectionnez une matière"
    self.root.get_screen("InscriptionEtu").ids.mdp.text = ""
    self.root.get_screen("InscriptionEtu").ids.erreur.text = ""
    self.root.get_screen("InscriptionEtu").ids.img.text = "Sélectionnez
votre image"
```

Cette méthode permet de remettre d'origine la page inscription d'étudiant lorsqu'un étudiant s'inscrit et que son inscription est validée.

Inactif() :

```
#Méthode pour le else après la condition if dans le fichier kv
def inactif(self):
    pass
```

Cette méthode empêche d'avoir des erreurs lorsqu'on utilise une condition if dans le fichier kv. Il faut obligatoirement un else après le if.

Si on ne veut rien faire si la condition if n'est pas respectée. On rajoute cette fonction après le else

Bouton « Retour »

Entrez votre nom:

Entrez votre prénom:

Votre Email:

Votre année d'étude:

Sélectionnez votre spécialité :

Sélectionnez votre image:

Entrez un mot de passe

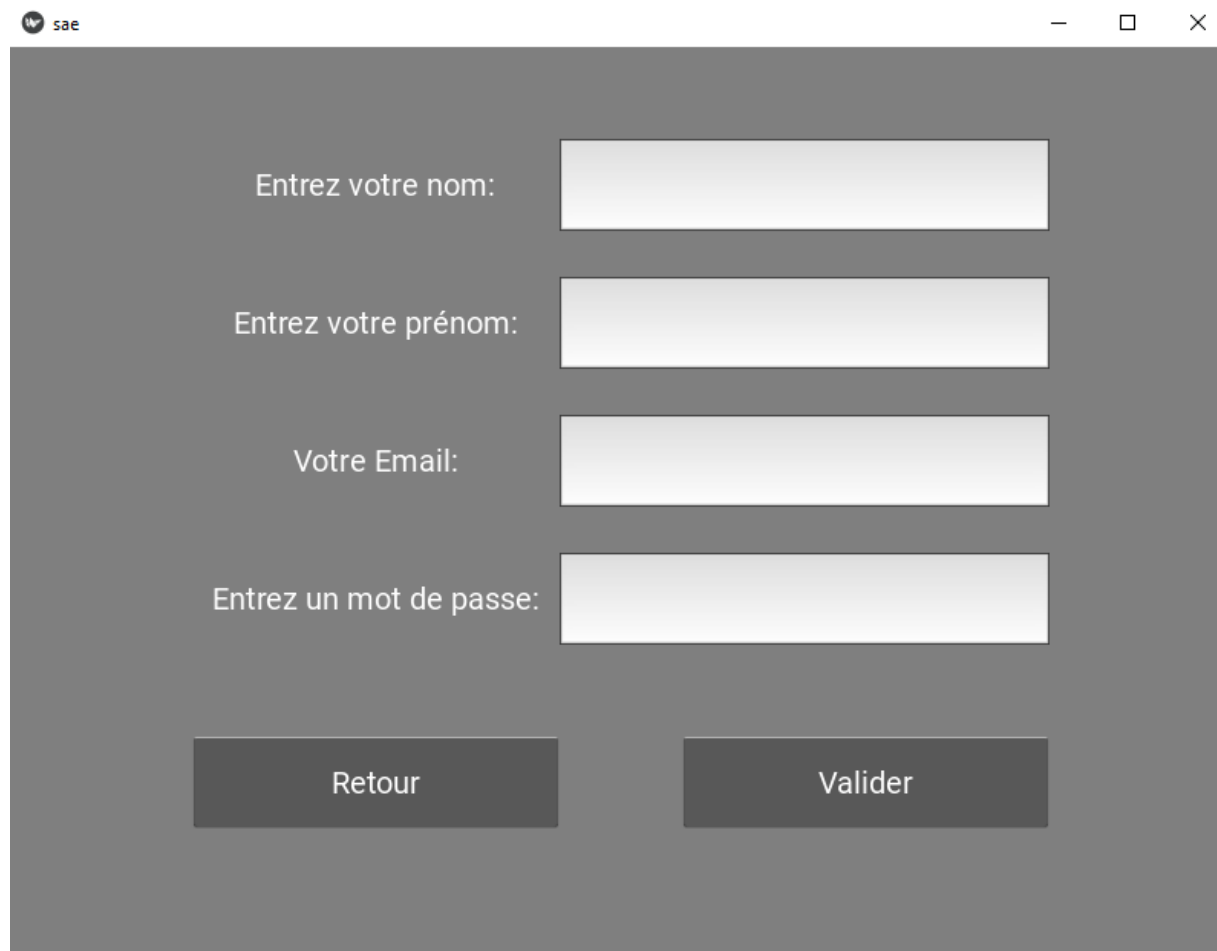
Button:

```
text: "Retour"
font_size: 25
size_hint: 0.5, None
height: 50
background_color: 1, 1, 1, 1
on_release:
    app.root.current = "Connexion"
    root.manager.transition = NoTransition()
on_press:
    app.reinitialise_form_etu()
```

Lorsque le bouton « Retour » est pressé : il n'y a que la méthode `reinitialise_form_etu()` qui s'active. Cette méthode est déjà expliquée précédemment.

La pression du bouton « Retour » permet de retourner à la page de connexion

Page d'inscription prof

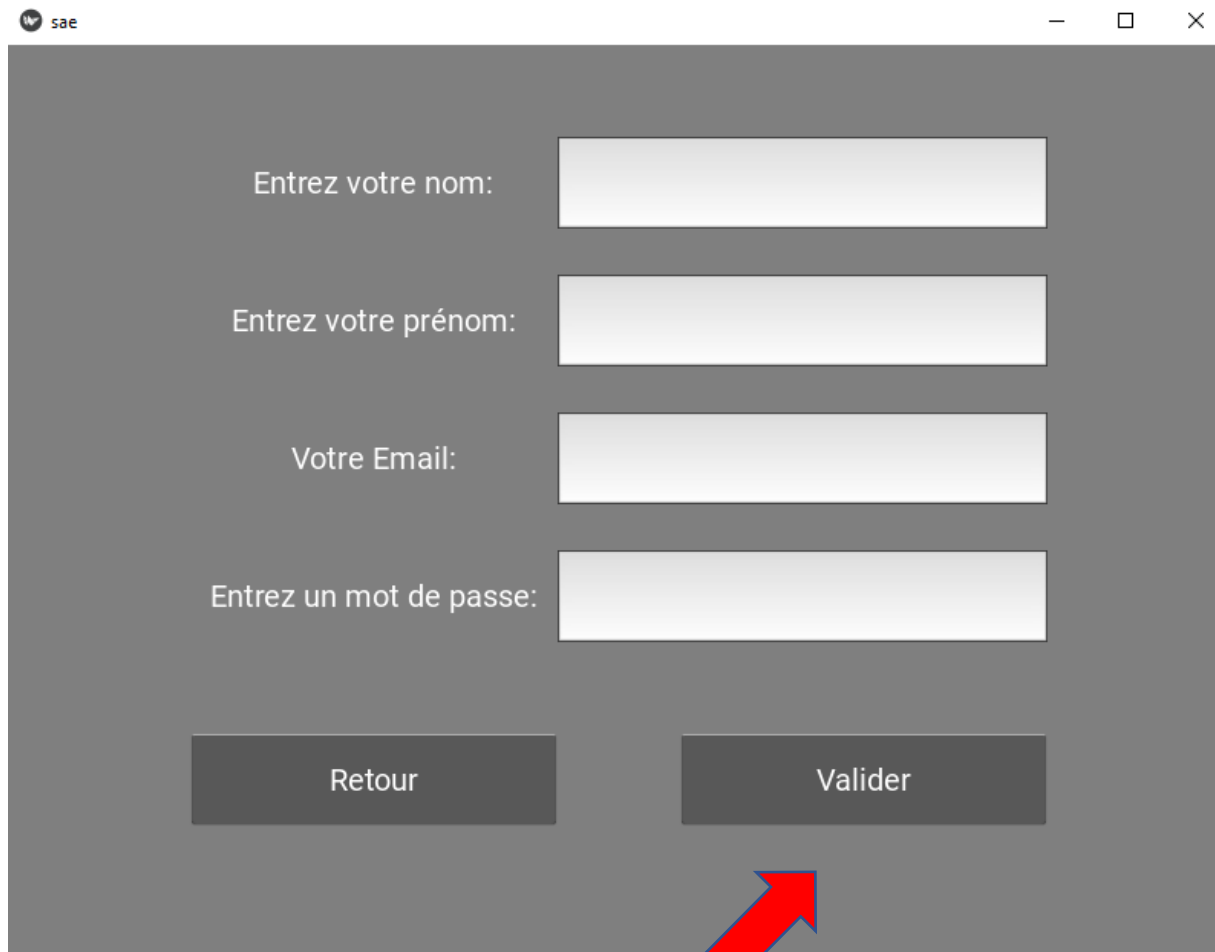


Entrez votre nom:

Entrez votre prénom:

Votre Email:

Entrez un mot de passe:

Bouton « Valider »

Entrez votre nom:

Entrez votre prénom:

Votre Email:

Entrez un mot de passe:

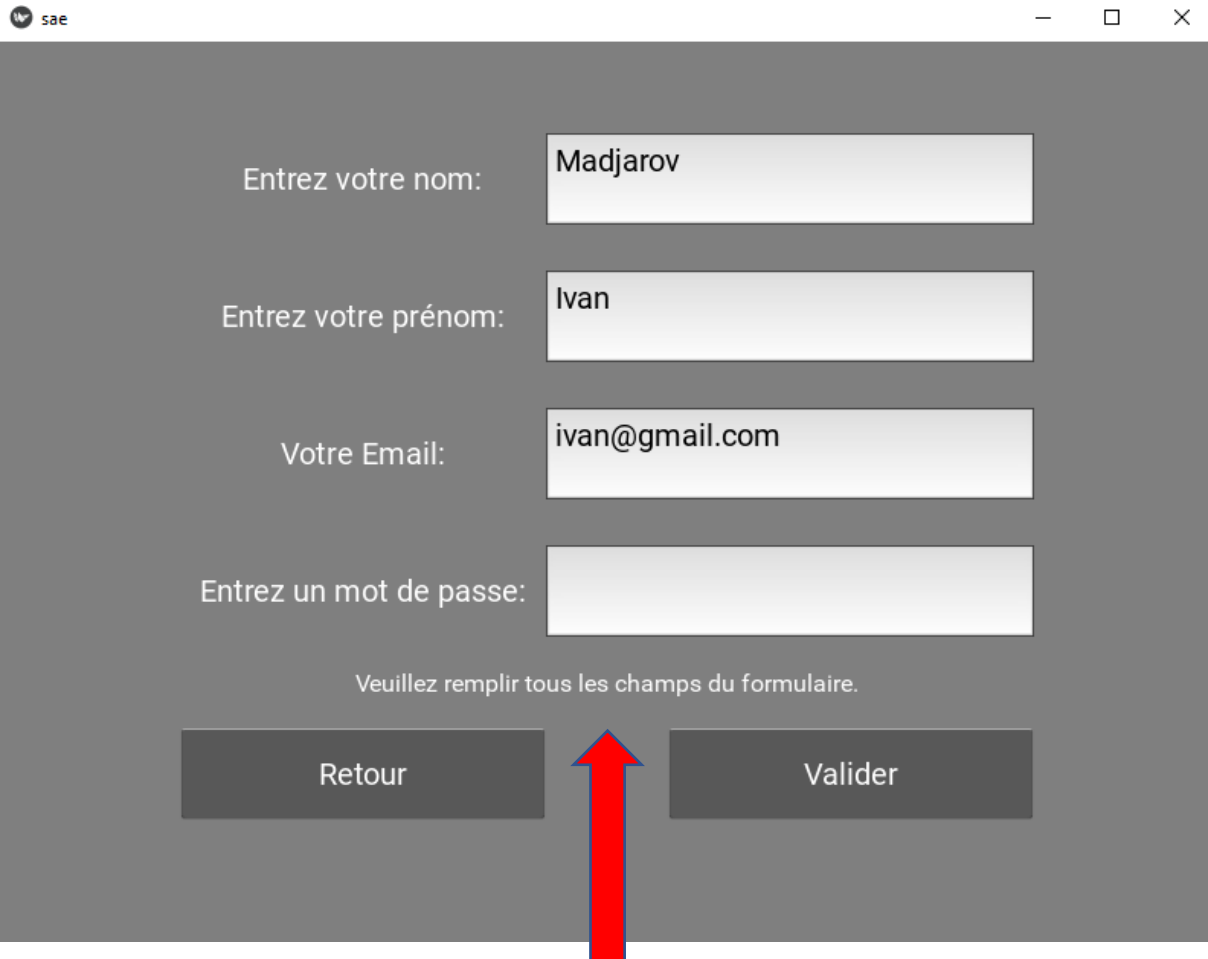
Button:

```
text: "Valider"
font_size:20
size_hint: 0.3,0.1
pos_hint:{'center_x':.7, 'center_y':.2}
background_color: 1, 1, 1, 1
on_release:
    app.verif_formulaire_prof() #vérifie si tous les champs du
formulaire sont remplis
    app.verif_mail_prof()
    app.enregistrement_prof() if app.verif_prof and
app.verif_email_prof else app.inactif() #si tout est rempli, envoie des champs
vers la bdd
    app.reinitialise_form_prof() if app.verif_prof and
app.verif_email_prof else app.inactif() #si tous est rempli, reinitialise tous
les champs
```



```
app.same_mail_prof() if not app.verif_email_prof and  
app.verif_prof else app.inactif()
```

Les méthodes présentes ici ont exactement les mêmes fonctionnalités que pour l'inscription d'un étudiant mais adaptée à un professeur. Il y a juste la méthode `envoie_notes()` qui n'est pas présente car un professeur n'a pas à avoir de notes.



Entrez votre nom: Madjarov

Entrez votre prénom: Ivan

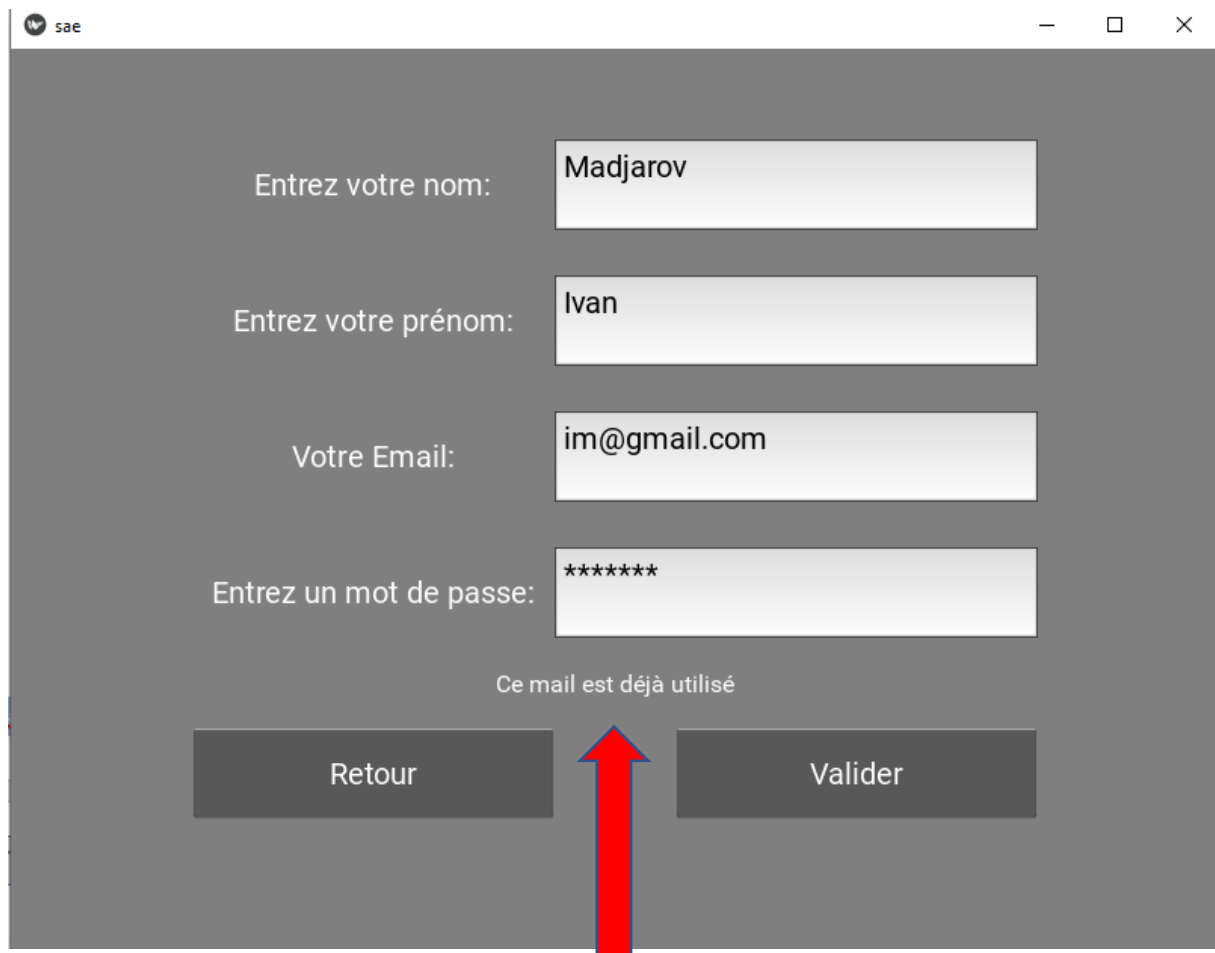
Votre Email: ivan@gmail.com

Entrez un mot de passe:

Veuillez remplir tous les champs du formulaire.

Retour Valider

Message d'erreur si tous les champs ne sont pas remplis



Entrez votre nom: Madjarov

Entrez votre prénom: Ivan

Votre Email: im@gmail.com

Entrez un mot de passe: *****

Ce mail est déjà utilisé

Retour Valider

Message d'erreur si le mail entré par le professeur est déjà présent dans la BDD

Code des méthodes du bouton « Valider » :

```
def verif_formulaire_prof(self):
    self.verif_prof = True
    if self.root.get_screen("InscriptionProf").ids.nom.text == "" or
self.root.get_screen("InscriptionProf").ids.prenom.text == "" or
self.root.get_screen("InscriptionProf").ids.email.text == "" or
self.root.get_screen("InscriptionProf").ids.mdp.text == "":
        self.root.get_screen("InscriptionProf").ids.erreur.text =
"Veuillez remplir tous les champs du formulaire."
        self.verif_prof = False

def verif_mail_prof(self):
    self.verif_email_prof = True
    try:
        connection_params = {
            'host' : "172.20.10.3",
            'user' : "admin",
            'password' : "sae302",
            'database' : "doss_etu"
        }
        except mysql.connector.Error as e:
            print("Exception : ", e)
        else:
            request = """select email from etudiants;"""
            request2 = """select email from profs;"""

            with mysql.connector.connect(**connection_params) as db :
                with db.cursor() as c:
                    c.execute(request)
                    rq = c.fetchall()
                    c.execute(request2)
                    rq2 = c.fetchall()
                db.commit()
                db.close()

                for elem in rq:
                    if elem[0] ==
self.root.get_screen("InscriptionProf").ids.email.text:
                        self.verif_email_prof = False

                for elem in rq2:
                    if elem[0] ==
self.root.get_screen("InscriptionProf").ids.email.text:
                        self.verif_email_prof = False

def same_mail_prof(self):
```

```
        self.root.get_screen("InscriptionProf").ids.erreur.text = "Ce mail est  
déjà utilisé"  
  
def inactif(self):  
    pass  
  
def enregistrement_prof(self):  
    self.p.set_nom(self.root.get_screen("InscriptionProf").ids.nom.text)  
    self.p.set_prenom(self.root.get_screen("InscriptionProf").ids.prenom.t  
ext)  
    mdp_crypt =  
sha512(self.root.get_screen("InscriptionProf").ids.mdp.text.encode()).hexdigest()  
    self.p.set_mdp(mdp_crypt)  
    self.p.set_email(self.root.get_screen("InscriptionProf").ids.email.tex  
t)  
  
def reinitialise_form_prof(self):  
    self.root.get_screen("InscriptionProf").ids.nom.text = ""  
    self.root.get_screen("InscriptionProf").ids.prenom.text = ""  
    self.root.get_screen("InscriptionProf").ids.email.text = ""  
    self.root.get_screen("InscriptionProf").ids.mdp.text = ""  
    self.root.get_screen("InscriptionProf").ids.erreur.text = ""
```

Bouton « Retour »

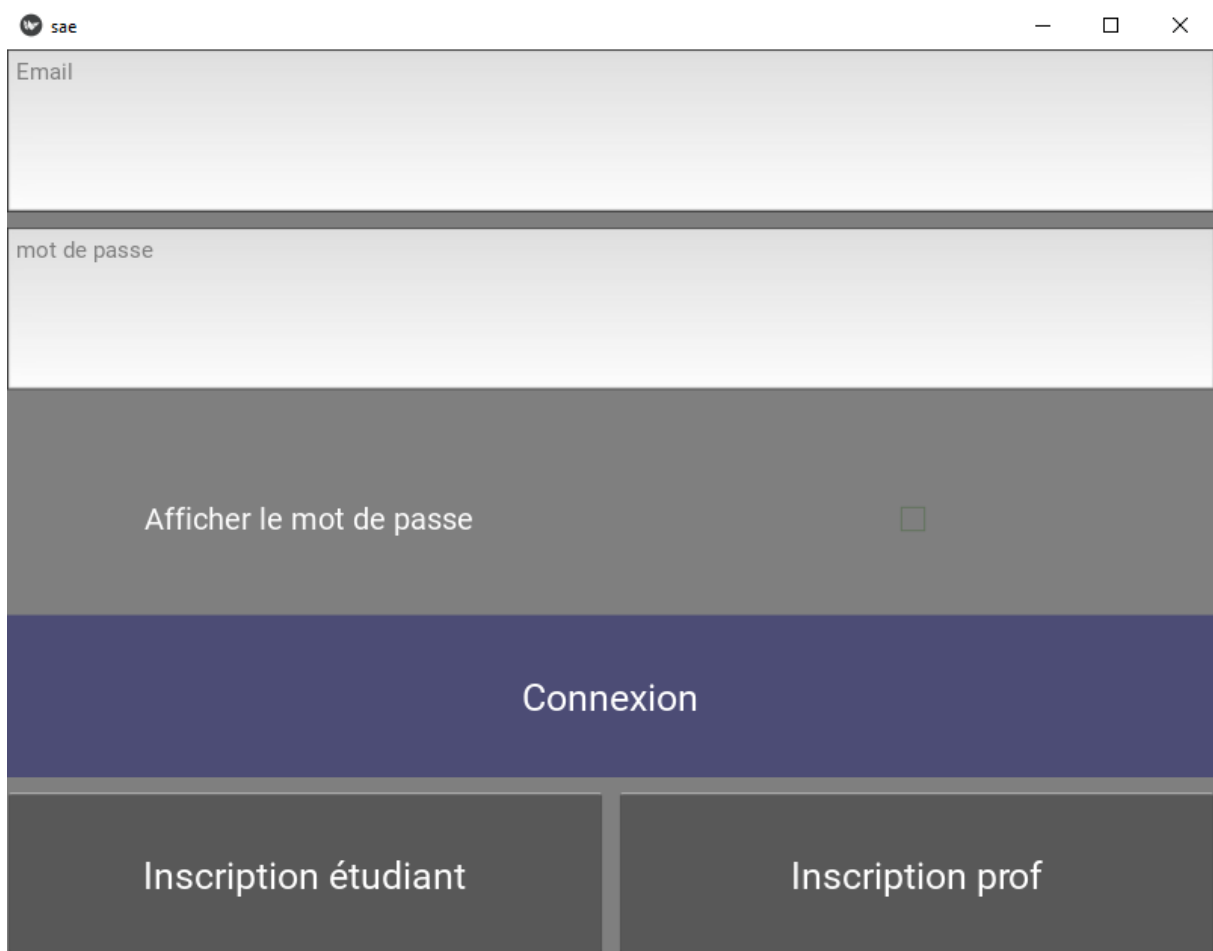
The screenshot shows a web form with a title bar containing 'sae' and standard window controls. The form has four text input fields stacked vertically, each preceded by a label: 'Entrez votre nom:', 'Entrez votre prénom:', 'Votre Email:', and 'Entrez un mot de passe:'. Below these fields are two buttons: 'Retour' on the left and 'Valider' on the right. A large red arrow points from the bottom right towards the 'Retour' button.

Button:

```
text: "Retour"
font_size:20
size_hint: 0.3,0.1
pos_hint:{'center_x':.3, 'center_y':.2}
background_color: 1, 1, 1, 1
on_release:
    app.root.current = "Connexion"
    root.manager.transition = NoTransition()
on_press:
    app.reinitialise_form_prof()
```

Le bouton « Retour » contient la même méthode que le bouton « Retour » de la page étudiant mais adaptée par la page professeur. Elle permet de remettre à l'état d'origine la page professeur.

La pression du bouton « Retour » permet de retourner à la page de connexion

Page de Connexion :

```
Button:
    id:log
    text: "Connexion"
    font_size: 25
    on_press:
        app.connex()
        app.recup_matières_notes() if app.can_connect else
app.inactif()
        app.page_infos_etu() if app.can_connect and app.is_etu else
app.inactif()
```

Il y a plusieurs méthodes exécutées lors de la pression du bouton « Connexion » par l'utilisateur que je vais détailler ci-dessous.

Connex():

Cette méthode permet d'authentifier ou non un utilisateur lors de la tentative de connexion

```
def connexion(self):
    lst_etu = []
    lst_prof = []
    self.whos_connect = ""
    self.is_etu = False
    self.is_prof = False

    try:
        connection_params = {
            'host' : "172.20.10.3",
            'user' : "admin",
            'password' : "sae302",
            'database' : "doss_etu"
        }
    except mysql.connector.Error as e:
        print("Exception : ", e)
    else:
        #requête pour obtenir chaque mail et mdp étudiant de la bdd
        request = """select email, mdp from etudiants;"""
        #requête pour obtenir chaque mail et mdp prof de la bdd
        request2 = """select email, mdp from profs;"""
        #requête pour obtenir l'id de l'étudiant

        with mysql.connector.connect(**connection_params) as db :
            with db.cursor() as c:
                c.execute(request)
                rq = c.fetchall()
                c.execute(request2)
                rq2 = c.fetchall()
            db.commit()
            db.close()

        #parcourt chaque mail et mdp étudiant
        for element in rq :
            mail = element[0]
            pwd = element[1]

            #ajoute les mails et mdp étudiant à une liste
            mlpswd = [mail, pwd]
            lst_etu.append(mlpswd)

        #parcourt chaque mail et mdp prof
        for element in rq2:
            mail = element[0]
            pwd = element[1]
```

```
#ajoute les mails et mdp prof dans une liste
mlpswd = [mail, pwd]
lst_prof.append(mlpswd)

#variable mail2 prend pour valeur le mail entré par l'utilisateur lors
de la connexion
mail2 = self.root.get_screen("Connexion").ids.email.text
#variable mail2 prend pour valeur le mdp entré par l'utilisateur lors
de la connexion
pwd2 = self.root.get_screen("Connexion").ids.passwd.text

pwd2_crypt = sha512(pwd2.encode()).hexdigest()

#teste si le mail et le mdp entrée par l'utilisateur correspond à une
paire mail mdp de la BDD
for elem in lst_etu:
    if elem[0] == mail2 and elem[1] == pwd2_crypt:
        self.can_connect = True
        self.is_etu = True
        self.whos_connect = "etu"

for elem in lst_prof:
    if elem[0] == mail2 and elem[1] == pwd2_crypt:
        self.can_connect = True
        self.is_prof = True
        self.whos_connect = "prof"

#si c'est le cas : CONNEXION + changement d'écran
if self.can_connect and self.is_etu:
    self.accept()
    self.root.get_screen("Connexion").manager.current = "EtuHome"
    self.reset_color()

elif self.can_connect and self.is_prof:
    self.accept()
    self.root.get_screen("Connexion").manager.current = "ProfHome"
    self.reset_color()

#si ce n'est pas le cas : CONNEXION REFUSEE
else:
    self.refuse()
```

Pour vérifier l'authentification d'un utilisateur :

On définit une liste pour stocker les informations d'authentification des étudiants et des professeurs.

On définit des variables pour stocker les informations sur la connexion de l'utilisateur (si elle est acceptée, si l'utilisateur est un étudiant ou un professeur).

On tente de se connecter à la base de données en utilisant les paramètres de connexion (host, user, password, database)

On utilise des requêtes SQL pour récupérer les informations d'identification des étudiants et des professeurs de la base de données.

On parcourt ces informations pour les stocker dans les listes appropriées.

On récupère les informations d'authentification saisies par l'utilisateur lors de la tentative de connexion

On crypte le mot de passe saisi par l'utilisateur car il faut comparer les haches des mots de passes.

On parcourt les listes pour vérifier si les informations saisies correspondent à une entrée de la base de données.

Si c'est le cas, on accepte la connexion, change l'écran en fonction de l'utilisateur (étudiant ou professeur) et réinitialise les couleurs de la page de connexion.

Si ce n'est pas le cas, on refuse la connexion et change la couleur du bouton en rouge

Cas où la connexion est refusée

sae

jen'existepas@gmail.com

Afficher le mot de passe ☐

Connexion

Inscription étudiant

Inscription prof

Si la connexion est acceptée, la méthode `recup_matières_notes` est déclenchée :

```
app.recup_matières_notes() if app.can_connect else app.inactif()
```

Cette fonction permet :

- De récupérer les notes et matières d'un étudiant si l'utilisateur connecté est un étudiant
- De récupérer les notes et matières de tous les étudiants si l'utilisateur connecté est un étudiant

```
def recup_matiere_notes(self):
    infos_etu = []
    try:
        connection_params = {
            'host' : "172.20.10.3",
            'user' : "admin",
            'password' : "sae302",
            'database' : "doss_etu"
        }
    except mysql.connector.Error as e:
        print("Exception : ", e)
    else:
        if self.whos_connect == "etu":
            #requête pour obtenir chaque mail et mdp étudiant de la bdd
            request = """select matiere, moyenne from etu_resume where
email = %s;"""
            request2 = """select nom, prenom, email, annee, spe, photo
from etudiants where email = %s;"""

            elif self.whos_connect == "prof":
                #requête pour obtenir chaque mail et mdp prof de la bdd
                request = """select email, matiere, moyenne from
etu_resume;"""
                request2 = """select nom, prenom, email, annee, spe, photo
from etudiants;"""

        with mysql.connector.connect(**connection_params) as db :
            with db.cursor() as c:
                if self.whos_connect == "etu":
                    c.execute(request,
(self.root.get_screen("Connexion").ids.email.text,))
                    rq = c.fetchall()
                    c.execute(request2,
(self.root.get_screen("Connexion").ids.email.text,))
                    rq2 = c.fetchall()
                elif self.whos_connect == "prof":
                    c.execute(request)
                    rq = c.fetchall()
                    c.execute(request2)
                    rq2 = c.fetchall()
            db.commit()
            db.close()

        if self.whos_connect == "etu":
            notes = {"Reseau": rq[0][1], "Telecom": rq[1][1], "Maths" :
rq[2][1], "Programmation" : rq[3][1], "Anglais" : rq[4][1], rq[0][4] :
rq[5][1]}
```

```
        infos_etu = {"Nom": rq2[0][0], "Prenom": rq2[0][1], "Email" :  
rq2[0][2], "Annee" : rq2[0][3], "Photo" : rq2[0][5], "notes": notes}  
  
    elif self.whos_connect == "prof":  
        infos_etu = []  
        rq_list = []  
        rq2_list = []  
  
        #transforme la liste de tuples en une liste de liste  
        for note in rq:  
            rq_list.append([note[0], note[1], note[2]])  
  
        for elem in rq2:  
            rq2_list.append([elem[0], elem[1], elem[2], elem[3], elem[4]])  
  
        #ajoute à une liste les dictionnaires contenant les infos des  
étudiants  
        for elem in rq2:  
            infos = {"Nom": elem[0], "Prenom": elem[1], "Email" : elem[2],  
"Annee" : elem[3], "Spe" : elem[4], "Photo" : elem[5], "Notes": None}  
            infos_etu.append(infos)  
  
        for i in range(len(infos_etu)):  
            notes = {}  
            for y in range(len(rq_list)):  
  
                if infos_etu[i]['Email'] == rq_list[y][0]:  
                    notes[rq_list[y][1]] = rq_list[y][2]  
  
            infos_etu[i]["Notes"] = notes  
  
    self.e.set_infos_etu(infos_etu)
```

Pour se faire :

On définit une liste pour stocker les informations sur les étudiants.

On tente de se connecter à la base de données en utilisant les paramètres de connexion (host, user, password, database)

On utilise des requêtes SQL pour récupérer les informations sur les matières et les notes des étudiants de la base de données.

On parcourt ces informations pour les stocker dans les listes appropriées.

On vérifie si l'utilisateur connecté est un étudiant ou un professeur pour utiliser les requêtes appropriées

Si l'utilisateur connecté est un étudiant, il récupère les informations sur les matières et les notes de cet étudiant. Il ajoute ces informations dans un dictionnaire et l'ajoute à la liste d'informations des étudiants.

Si l'utilisateur connecté est un professeur, il récupère les informations sur les matières et les notes de tous les étudiants. Il ajoute ces informations dans un dictionnaire pour chaque étudiant et l'ajoute à la liste d'informations des étudiants.

Il y a ensuite 2 cas de figures :

1^{er} cas : L'utilisateur connecté est un étudiant

L'étudiant est redirigé vers une page pour les étudiants

```
app.page_infos_etu() if app.can_connect and app.is_etu else app.inactif()
```

Si l'utilisateur connecté est un étudiant, la fonction page_infos_etu() se lance.

Page infos_etu():

Cette fonction permet de récupérer les matières et les notes de l'étudiant connecté et de les afficher dans la page étudiant

```
def page_infos_etu(self):
    inf = self.e.get_infos_etu()
    nom=inf["Nom"]
    prenom=inf["Prenom"]
    annee=inf["Annee"]
    mail=inf["Email"]
    math=round(inf["notes"]["Maths"],2)
    reseau=round(inf["notes"]["Reseau"],2)
    telecom=round(inf["notes"]["Telecom"],2)
    prog=round(inf["notes"]["Programmation"],2)
    anglais=round(inf["notes"]["Anglais"],2)
    name_specialite = list(inf["notes"].keys())[5]
    specialite = round(inf["notes"][name_specialite],2)
    self.root.get_screen("EtuHome").ids.bvn.text=str(nom+" "+prenom)
    self.root.get_screen("EtuHome").ids.annee.text=str(annee)
    self.root.get_screen("EtuHome").ids.mail.text=str(mail)
    self.root.get_screen("EtuHome").ids.math.text=str(math)
    self.root.get_screen("EtuHome").ids.reseau.text=str(reseau)
```

```

self.root.get_screen("EtuHome").ids.prog.text=str(prog)
self.root.get_screen("EtuHome").ids.telecom.text=str(telecom)
self.root.get_screen("EtuHome").ids.anglais.text=str(anglais)
self.root.get_screen("EtuHome").ids.name_specialite.text=name_specialite

self.root.get_screen("EtuHome").ids.specialite.text=str(specialite)
self.root.get_screen("EtuHome").ids.photo.source=recup_photo(nom)
moy = round(((math+reseau+telecom+prog+anglais+specialite)/6),2)
self.root.get_screen("EtuHome").ids.moyenne.text= str(moy)

```


Pour se faire, on utilise la méthode "get_infos_etu()" pour récupérer les informations sur l'étudiant depuis l'instance. Ensuite, on utilise les informations récupérées pour mettre à jour les champs de texte de l'écran étudiant avec les informations appropriées (nom, prénom, année, email, notes, etc.). On calcule également la moyenne des notes et on l'affiche à l'écran. On utilise également une fonction "recup_photo" pour récupérer la photo de l'étudiant

Page Étudiant

sae_maxime

Bienvenue:

baptiste maxime



maxime@gmail.com

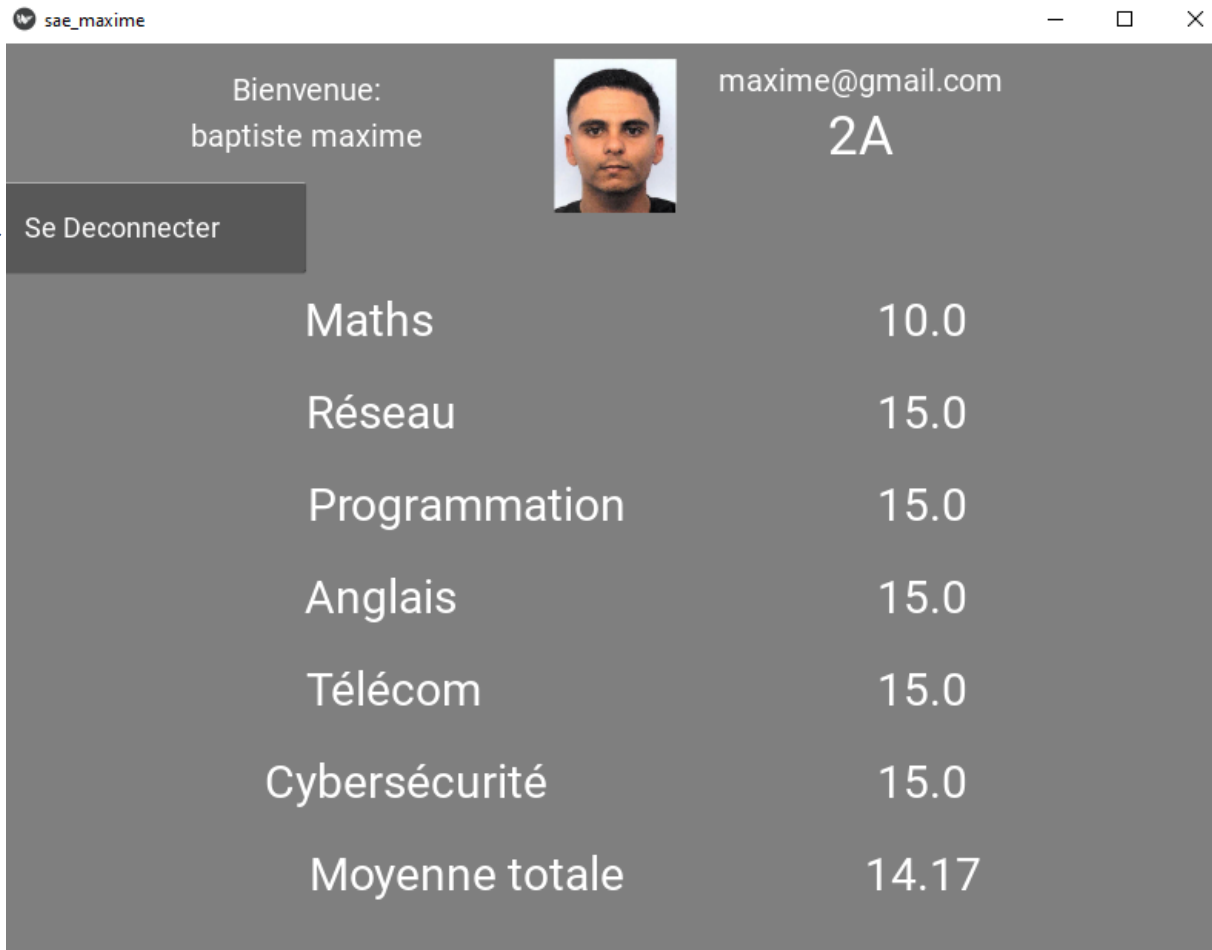
2A

Se Deconnecter

Maths	10.0
Réseau	15.0
Programmation	15.0
Anglais	15.0
Télécom	15.0
Cybersécurité	15.0
Moyenne totale	14.17

Cette page permet à l'étudiant de visualiser ses informations (Nom, Prénom, mail, année d'étude), ses notes et sa moyenne générale

Bouton « Se Déconnecter »



sa_e_maxime

Bienvenue:
baptiste maxime

maxime@gmail.com
2A

Se Deconnecter

Maths	10.0
Réseau	15.0
Programmation	15.0
Anglais	15.0
Télécom	15.0
Cybersécurité	15.0
Moyenne totale	14.17

Button:

```
text: "Se Deconnecter"  
font_size: 18  
size_hint: .3,.1  
pos_hint: {'center_x':.1, 'center_y':.8}  
height: 20  
background_color: 1, 1, 1, 1  
on_release:  
    app.reinitialise_champs_connexion()  
    app.root.current = "Connexion"  
    root.manager.transition = NoTransition()
```

Lorsque le bouton est relâché la méthode `reinitialise_champs_connexion()` est enclenchée

`reinitialise_champs_connexion()` :

```
def reinitialise_champs_connexion(self):  
    self.root.get_screen("Connexion").ids.email.text = ""  
    self.root.get_screen("Connexion").ids.passwd.text = ""
```

Cette méthode permet de réinitialiser les champs de connexion lorsque l'on se déconnecte

2^{ème} cas : l'utilisateur est un professeur

Si l'utilisateur est un professeur, il est redirigé vers la page de base qui est la page professeur. La page est comme ci-dessous

sae_maxime

Entrez le nom d'un étudiant Valider

Modifiez une note: Sélectionnez la matière Modifier

Se déconnecter

Bouton « Valider »**Button:**

```
text: "Valider"
markup: True
font_size:15
size_hint: 0.1,0.08
pos_hint:{'center_x':.65, 'center_y':.9}
on_release:
    app.reset_infos()
    app.page_infos_prof()
```

Pour pouvoir valider, il faut que le professeur entre le nom d'un étudiant valide et qui à une entrée dans la base de données, sinon un message d'erreur est affiché.

Lorsque le bouton « Valider » est relâché, les méthodes `reset_infos()` et `page_infos_prof()` sont déclenchées.

Page infos_prof()

Cette méthode permet d'afficher les informations, les matières et les notes de l'étudiant choisit par le professeur.

```
def page_infos_prof(self):
    name = self.root.get_screen("ProfHome").ids.selection.text
    inf = self.e.get_infos_etu()
    trouver = False

    for i in range(0, len(inf)):
        if name == inf[i]["Nom"]:
            trouver = True
            mail=inf[i]["Email"]
            annee=inf[i]["Annee"]
            nom=inf[i]["Nom"]
            prenom=inf[i]["Prenom"]
            math=round(inf[i]["Notes"]["Maths"],2)
            reseau=round(inf[i]["Notes"]["Réseau"],2)
            telecom=round(inf[i]["Notes"]["Télécom"],2)
            prog=round(inf[i]["Notes"]["Programmation"],2)
            anglais=round(inf[i]["Notes"]["Anglais"],2)
            spe = inf[i]["Spe"]
            self.specialisation = inf[i]["Spe"]
            special = round(inf[i]["Notes"][spe],2)

            self.root.get_screen("ProfHome").ids.name_math.text="Maths"
            self.root.get_screen("ProfHome").ids.name_reseau.text="Réseau"
            self.root.get_screen("ProfHome").ids.name_prog.text="Programma
tion"

            self.root.get_screen("ProfHome").ids.name_telecom.text="Téléco
m"

            self.root.get_screen("ProfHome").ids.name_anglais.text="Anglai
s"

            self.root.get_screen("ProfHome").ids.name_spe.text=spe
            self.root.get_screen("ProfHome").ids.matiere.values =
["Maths", "Réseau", "Programmation", "Anglais", "Télécom", spe]

            self.root.get_screen("ProfHome").ids.math.text=str(math)
            self.root.get_screen("ProfHome").ids.reseau.text=str(reseau)
            self.root.get_screen("ProfHome").ids.prog.text=str(prog)
            self.root.get_screen("ProfHome").ids.telecom.text=str(telecom)
            self.root.get_screen("ProfHome").ids.anglais.text=str(anglais)
            self.root.get_screen("ProfHome").ids.special.text=str(special)
```

```

        self.root.get_screen("ProfHome").ids.photo.source=recup_photo(
nom)

        self.root.get_screen("ProfHome").ids.mail.text=mail
        self.root.get_screen("ProfHome").ids.annee.text=annee
        self.root.get_screen("ProfHome").ids.prenom.text=prenom

    if not trouver:
        if self.root.get_screen("ProfHome").ids.selection.text == "":
            self.root.get_screen("ProfHome").ids.error.text = "Entrez le
nom d'un étudiant"
        else:
            self.root.get_screen("ProfHome").ids.error.text = "Cet
étudiant n'est pas présent dans la base de données"
        else:
            self.root.get_screen("ProfHome").ids.error.text = ""

```

Reset infos():

Cette méthode permet de remettre d'origine la page professeur pour permettre ensuite l'affichage des informations du nouvel étudiant

```

def reset_infos(self):
    self.root.get_screen("ProfHome").ids.photo.source = ""

    self.root.get_screen("ProfHome").ids.math.text=""
    self.root.get_screen("ProfHome").ids.reseau.text=""
    self.root.get_screen("ProfHome").ids.prog.text=""
    self.root.get_screen("ProfHome").ids.telecom.text=""
    self.root.get_screen("ProfHome").ids.anglais.text=""
    self.root.get_screen("ProfHome").ids.special.text = ""

    self.root.get_screen("ProfHome").ids.name_math.text=""
    self.root.get_screen("ProfHome").ids.name_reseau.text=""
    self.root.get_screen("ProfHome").ids.name_prog.text=""
    self.root.get_screen("ProfHome").ids.name_telecom.text=""
    self.root.get_screen("ProfHome").ids.name_anglais.text=""
    self.root.get_screen("ProfHome").ids.name_spe.text=""
    self.root.get_screen("ProfHome").ids.matiere.values = []
    self.root.get_screen("ProfHome").ids.annee.text = ""
    self.root.get_screen("ProfHome").ids.mail.text = ""
    self.root.get_screen("ProfHome").ids.prenom.text = ""
    self.root.get_screen("ProfHome").ids.note_error.text = ""

```

W sae_maxime

Entrez le nom d'un étudiant Valider

Modifiez une note: Entrez le nom d'un étudiant Sélectionnez la matière Modifier

Message d'erreur si le nom n'est pas rentré

Se déconnecter

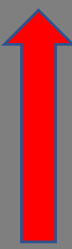
sae_maxime

Entrez le nom d'un étudiant

Cet étudiant n'est pas présent dans la base de données

Modifiez une note:

Message d'erreur si le nom n'est pas connu




sae_maxime

Entrez le nom d'un étudiant

baptiste

Valider



maxime

2A

maxime@gmail.com

Modifiez une note:

Sélectionnez la matière

Modifier

Maths	10.0
Réseau	15.0
Programmation	15.0
Anglais	15.0
Télécom	15.0
Cybersécurité	15.0

Se déconnecter

sae_maxime

Entrez le nom d'un étudiant

sbai

Valider

2A

hatim@gmail.com

hatim

Modifiez une note:

Sélectionnez la matière

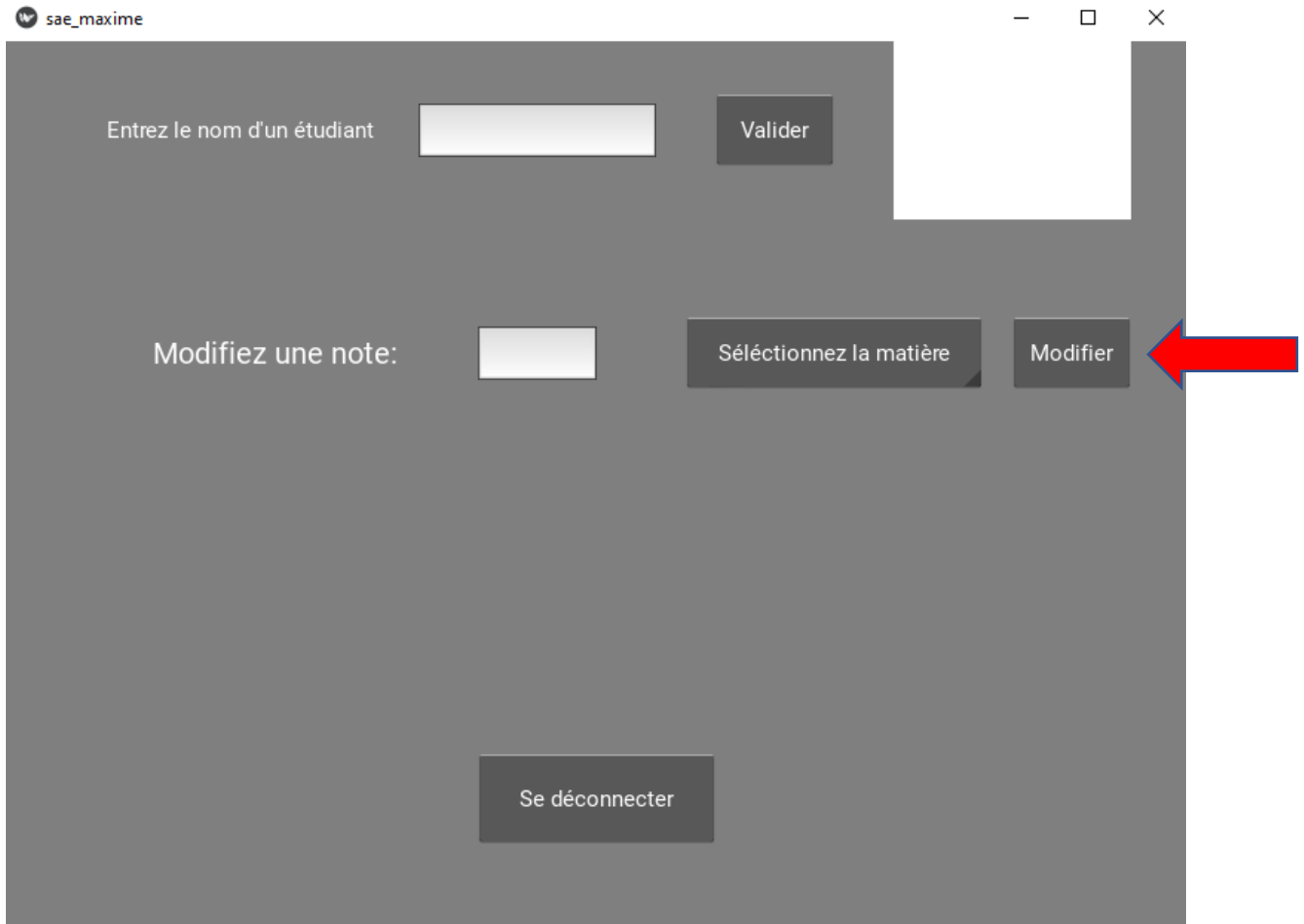
Modifier

Maths	15.0
Réseau	13.3
Programmation	12.6
Anglais	15.0
Télécom	0.0
IA	0.0

Se déconnecter

Bouton "Modifier"

Ce bouton permet à un professeur de modifier la note d'un étudiant dans la matière qu'il souhaite. Le professeur doit rentrer une note comprise entre 0 et 20 et sélectionner une matière pour pouvoir modifier une note. Dans le cas contraire, un message d'erreur s'affiche.



```
Button:
    id:bout_mod
    text: "Modifier"
    markup: True
    font_size:15
    size_hint: 0.1,0.08
    pos_hint:{'center_x':.9, 'center_y':.65}
    on_release:
        app.modify_note()
        app.recup_matières_notes()
```

Quand on relâche le bouton « Modifier », deux méthodes sont exécutées.

La méthode `recup_matières_notes()` à déjà été expliquée plus haut

Modify notes()

Cette méthode permet au professeur de modifier une note dans une matière

```
def modify_note(self):
    to_modify = self.root.get_screen("ProfHome").ids.matiere.text
    try:
        if 0 <= float(self.root.get_screen("ProfHome").ids.note_mod.text)
<= 20 and self.root.get_screen("ProfHome").ids.note_mod.text != "":
            if to_modify == "Maths":
                self.root.get_screen("ProfHome").ids.math.text =
self.root.get_screen("ProfHome").ids.note_mod.text
            elif to_modify == "Réseau":
                self.root.get_screen("ProfHome").ids.reseau.text =
self.root.get_screen("ProfHome").ids.note_mod.text
            elif to_modify == "Programmation":
                self.root.get_screen("ProfHome").ids.prog.text =
self.root.get_screen("ProfHome").ids.note_mod.text
            elif to_modify == "Anglais":
                self.root.get_screen("ProfHome").ids.anglais.text =
self.root.get_screen("ProfHome").ids.note_mod.text
            elif to_modify == "Télécom":
                self.root.get_screen("ProfHome").ids.telecom.text =
self.root.get_screen("ProfHome").ids.note_mod.text
            else:
                if to_modify == "Cybersécurité" or to_modify == "IA":
                    self.root.get_screen("ProfHome").ids.special.text =
self.root.get_screen("ProfHome").ids.note_mod.text
                else:
                    self.root.get_screen("ProfHome").ids.note_error.text
= "Veuillez choisir une matière"
            else:
                self.root.get_screen("ProfHome").ids.note_error.text =
"Veuillez entrer une note comprise entre 0 et 20"

        except ValueError:
            self.root.get_screen("ProfHome").ids.note_error.text = "Veuillez
entrer une note comprise entre 0 et 20"

    else:
        if 0 <= float(self.root.get_screen("ProfHome").ids.note_mod.text)
<= 20 and self.root.get_screen("ProfHome").ids.note_mod.text != "":
```

```
try:
    connection_params = {
        'host' : "172.20.10.3",
        'user' : "admin",
        'password' : "sae302",
        'database' : "doss_etu"
    }
except mysql.connector.Error as e:
    print("Exception : ", e)
else:
    request = """update notes set note = %s where mail_etu =
%s and matiere = %s;"""
    params =
(round(float(self.root.get_screen("ProfHome").ids.note_mod.text),2),
self.root.get_screen("ProfHome").ids.mail.text, to_modify)

    with mysql.connector.connect(**connection_params) as db :
        with db.cursor() as c:
            c.execute(request,params)
        db.commit()
        db.close()
```


Cette méthode permet de modifier une note d'un étudiant dans la base de données. Elle vérifie d'abord que la note saisie par le professeur est valide, c'est-à-dire qu'elle est comprise entre 0 et 20 et qu'elle est bien un nombre. Si c'est le cas, elle met à jour la note dans l'interface graphique en fonction de la matière choisie par le professeur. Ensuite, elle se connecte à la base de données et utilise une requête SQL pour mettre à jour la note de l'étudiant dans la table "notes" en utilisant l'adresse e-mail de l'étudiant et la matière choisie. Si une erreur se produit, un message d'erreur est affiché

sae_maxime

Entrez le nom d'un étudiant

baptiste

Valider



maxime

2A

maxime@gmail.com

Modifiez une note:

Sélectionnez la matière

Modifier

Veuillez entrer une note comprise entre 0 et 20

Maths	10.0
Réseau	15.0
Programmation	15.0
Anglais	15.0
Télécom	15.0
Cybersécurité	15.0

Se déconnecter


Message d'erreur si la note n'est pas sélectionnée ou n'est pas comprise entre 0 et 20

sae_maxime

Entrez le nom d'un étudiant

baptiste

Valider



2A

maxime@gmail.com

maxime

Modifiez une note:

12.6

Sélectionnez la matière

Modifier

Veuillez choisir une matière

Maths	10.0
Réseau	15.0
Programmation	15.0
Anglais	15.0
Télécom	15.0
Cybersécurité	15.0

Message d'erreur si la matière n'est pas sélectionnée


Se déconnecter

sae_maxime

Entrez le nom d'un étudiant

baptiste

Valider



maxime

2A

maxime@gmail.com

Modifiez une note:

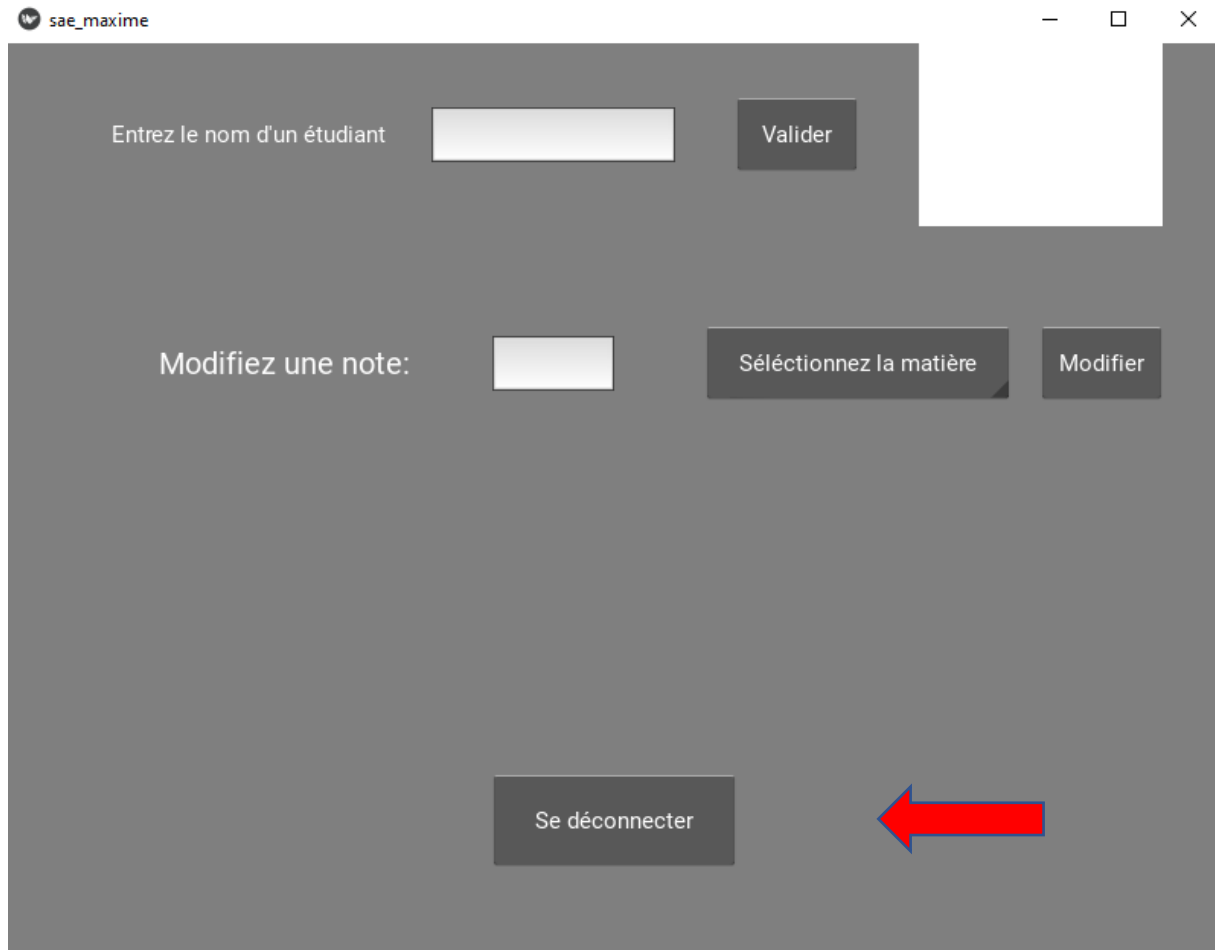
12.6

Réseau

Modifier

Maths	10.0
Réseau	12.6
Programmation	15.0
Anglais	15.0
Télécom	15.0
Cybersécurité	15.0

Se déconnecter

Bouton « Se déconnecter »**Button:**

```
text: "Se déconnecter"
font_size: 16
markup: True
font_size: 15
size_hint: 0.2, 0.1
pos_hint: {'center_x': .5, 'center_y': .15}
on_release:
    app.reinitialise_champs_connexion()
    app.reset_page_prof()
    app.root.current = "Connexion"
    root.manager.transition = NoTransition()
```

Lorsque le bouton « Se déconnecter » est relâché, deux méthodes sont exécutées :

Reinitialise_champs_connexion() est déjà expliquée plus haut

Reset_page_prof() :

```
def reset_page_prof(self):  
    self.root.get_screen("ProfHome").ids.error.text = ""  
    self.root.get_screen("ProfHome").ids.selection.text = ""  
    self.root.get_screen("ProfHome").ids.photo.source = ""  
  
    self.root.get_screen("ProfHome").ids.math.text = ""  
    self.root.get_screen("ProfHome").ids.reseau.text = ""  
    self.root.get_screen("ProfHome").ids.prog.text = ""  
    self.root.get_screen("ProfHome").ids.telecom.text = ""  
    self.root.get_screen("ProfHome").ids.anglais.text = ""  
    self.root.get_screen("ProfHome").ids.special.text=""  
    self.root.get_screen("ProfHome").ids.note_mod.text=""  
    self.root.get_screen("ProfHome").ids.annee.text = ""  
    self.root.get_screen("ProfHome").ids.mail.text = ""  
    self.root.get_screen("ProfHome").ids.prenom.text = ""  
    self.root.get_screen("ProfHome").ids.note_error.text = ""  
  
    self.root.get_screen("ProfHome").ids.name_math.text=""  
    self.root.get_screen("ProfHome").ids.name_reseau.text=""  
    self.root.get_screen("ProfHome").ids.name_prog.text=""  
    self.root.get_screen("ProfHome").ids.name_telecom.text=""  
    self.root.get_screen("ProfHome").ids.name_anglais.text=""  
    self.root.get_screen("ProfHome").ids.name_spe.text=""  
    self.root.get_screen("ProfHome").ids.matiere.text="Sélectionnez la  
matière"  
    self.root.get_screen("ProfHome").ids.matiere.values = []
```

Cette méthode permet de remettre la page professeur à l'état original lors de la déconnexion

Difficultés rencontrées :

Nous avons eu des problèmes avec :

- La conversion d'image en format blob.
- ScreenManager pour pouvoir naviguer entre plusieurs fenêtres
- La gestion des id avec ScreenManager

Test et validation :

Tout fonctionne parfaitement pour les fonctionnalités proposées dans notre application kivy et nous avons également corrigé plusieurs bugs.

FONCTIONNALITE BONUS

Nous avons réussi à proposer les fonctionnalités bonus suivantes :

- Modification de la note en temps réel
- Page d'inscription pour utilisateur étudiants et prof
- Affichage des différentes ainsi que de la spécialité de l'étudiant

CONCLUSION ET PERSPECTIVES

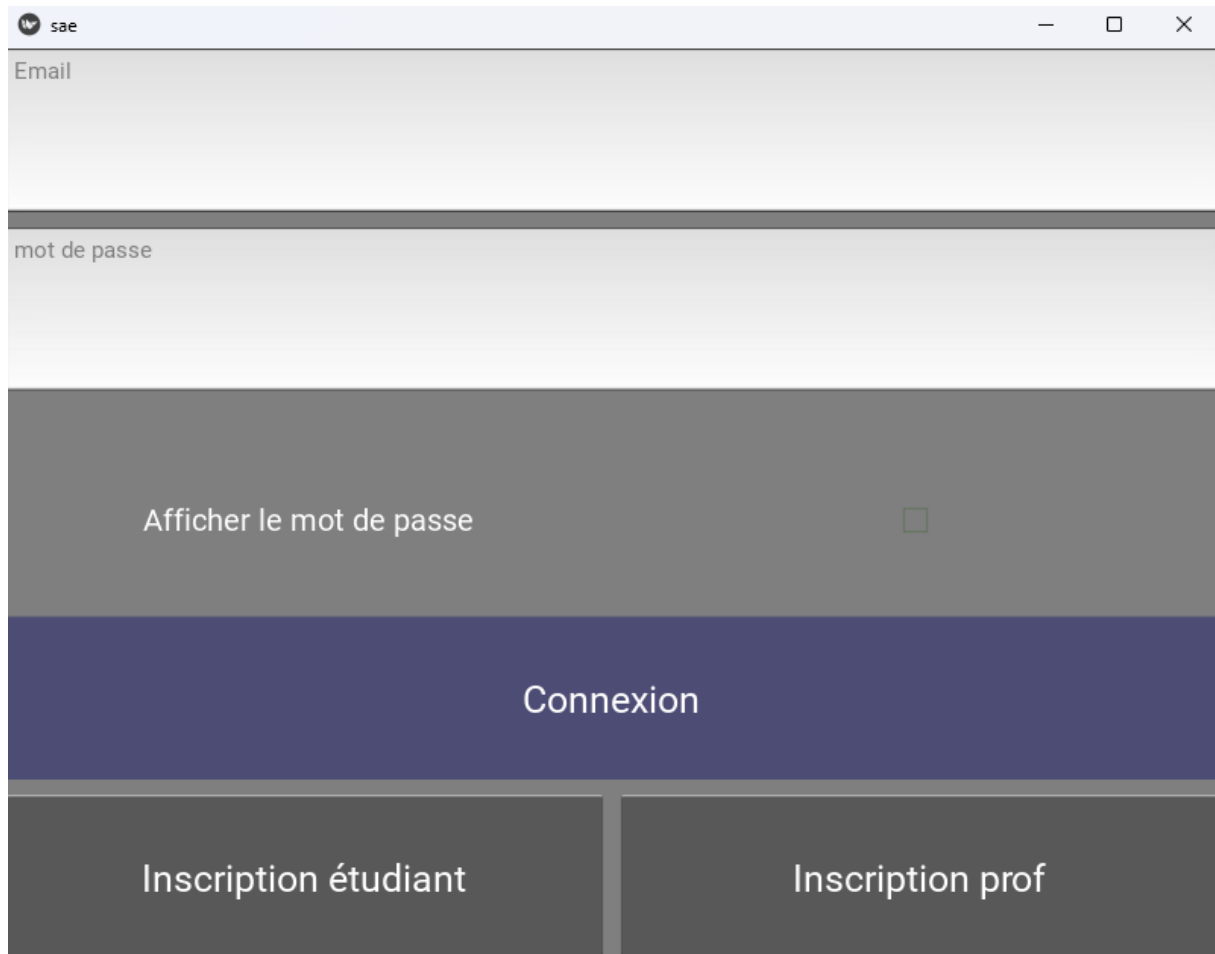
Cette SAE nous a permis de découvrir toutes les fonctionnalités de Kivy et de voir toutes les possibilités de combinaison possible de Kivy avec Python et ses autres modules. Le travail d'équipe et l'organisation dans le groupe étaient importants car nous avons majoritairement travaillé en distanciel.

Grâce à cela nous avons pu développer notre travail en autonomie et notre communication au sein de l'équipe.

ANNEXES

Manuel d'utilisation :

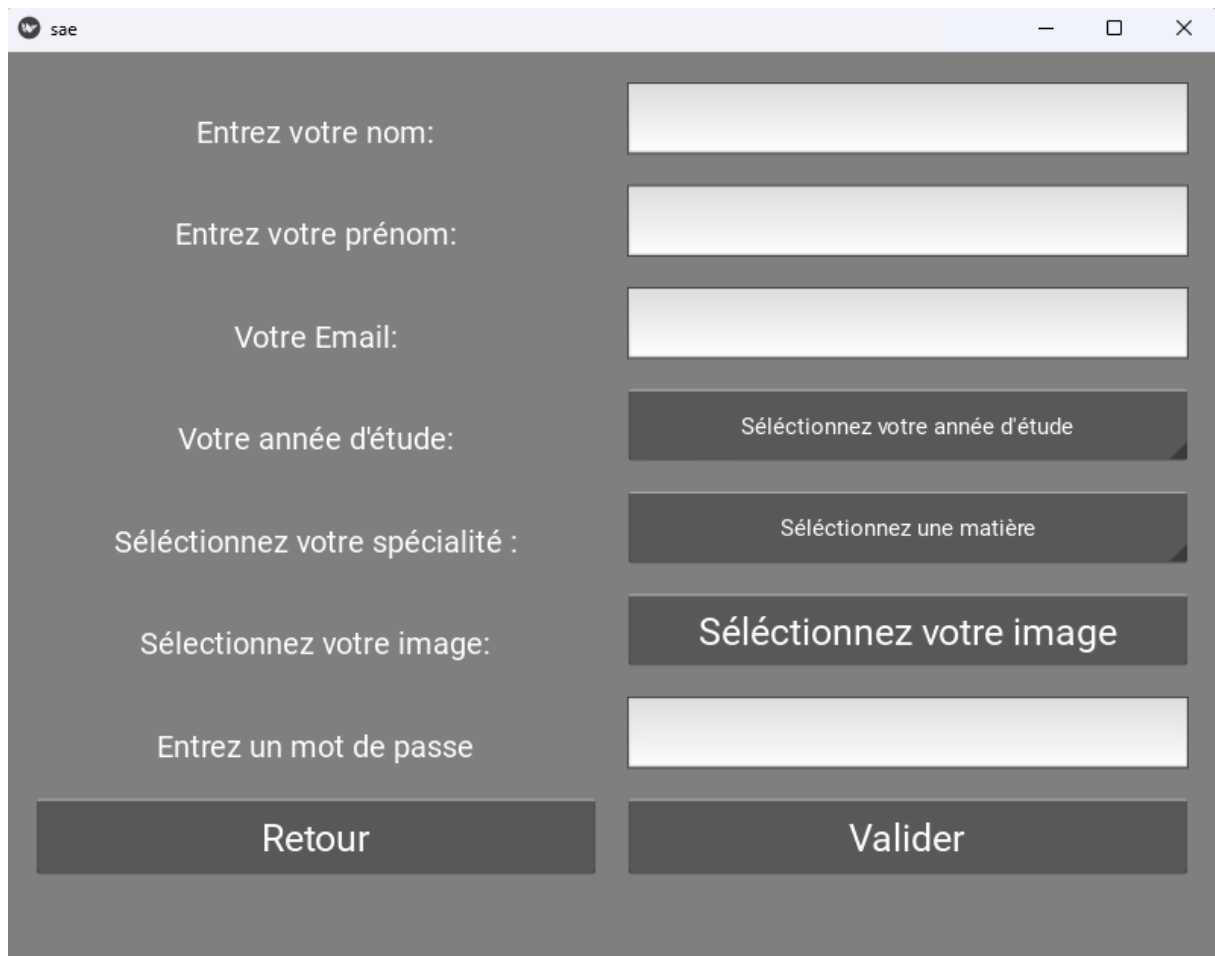
Lancer le programme sae.py pour aller avoir cette fenêtre d'affichage :



The screenshot shows a web application window titled 'sae'. It contains a login/registration form with the following elements:

- An 'Email' input field.
- A 'mot de passe' (password) input field.
- A checkbox labeled 'Afficher le mot de passe' (Show password).
- A large blue button labeled 'Connexion' (Login).
- Two buttons at the bottom: 'Inscription étudiant' (Student registration) and 'Inscription prof' (Professor registration).

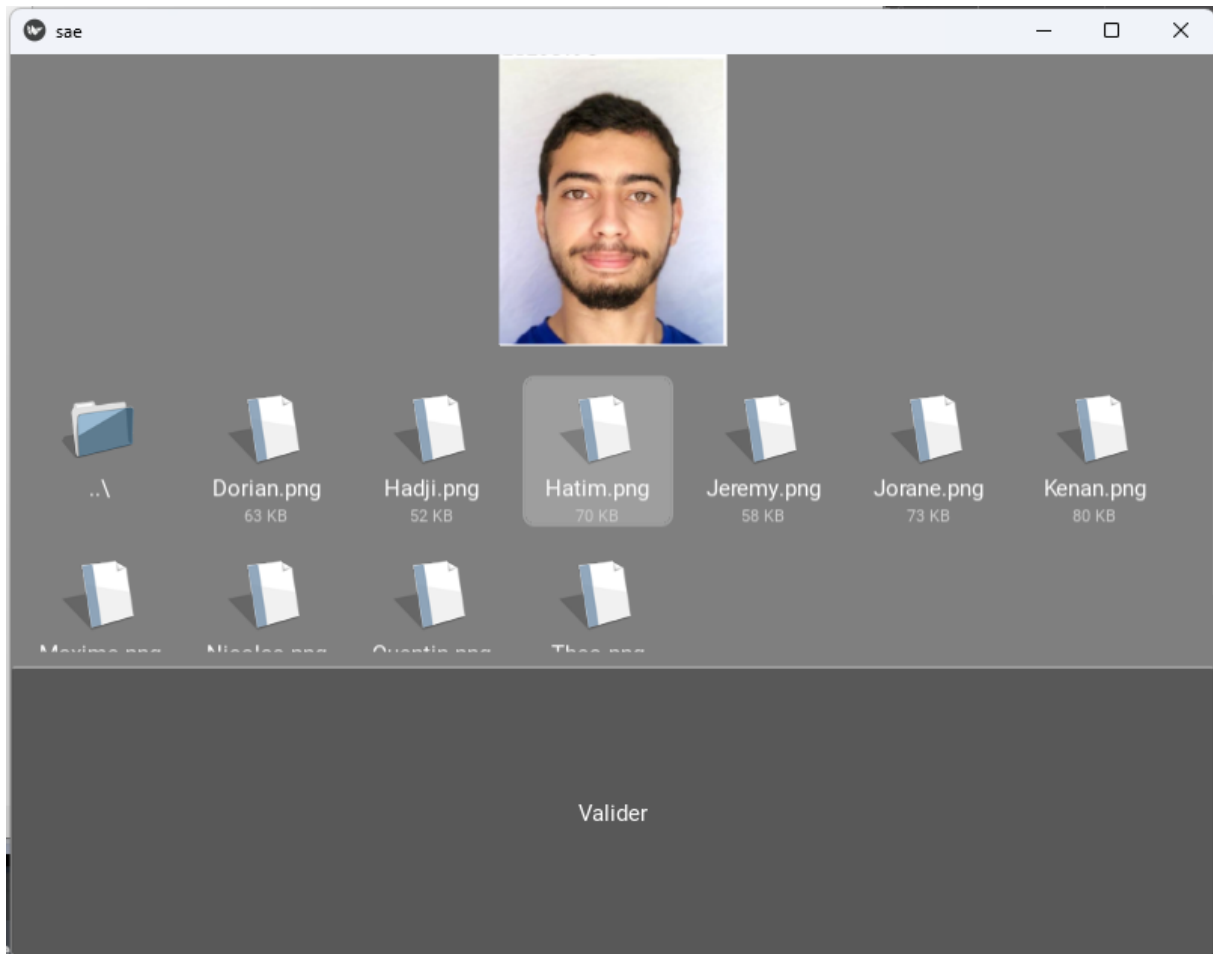
Ensuite si vous êtes étudiant cliquez sur le bouton d'inscription :



The screenshot shows a web application window titled "sae" with a registration form. The form contains the following fields and buttons:

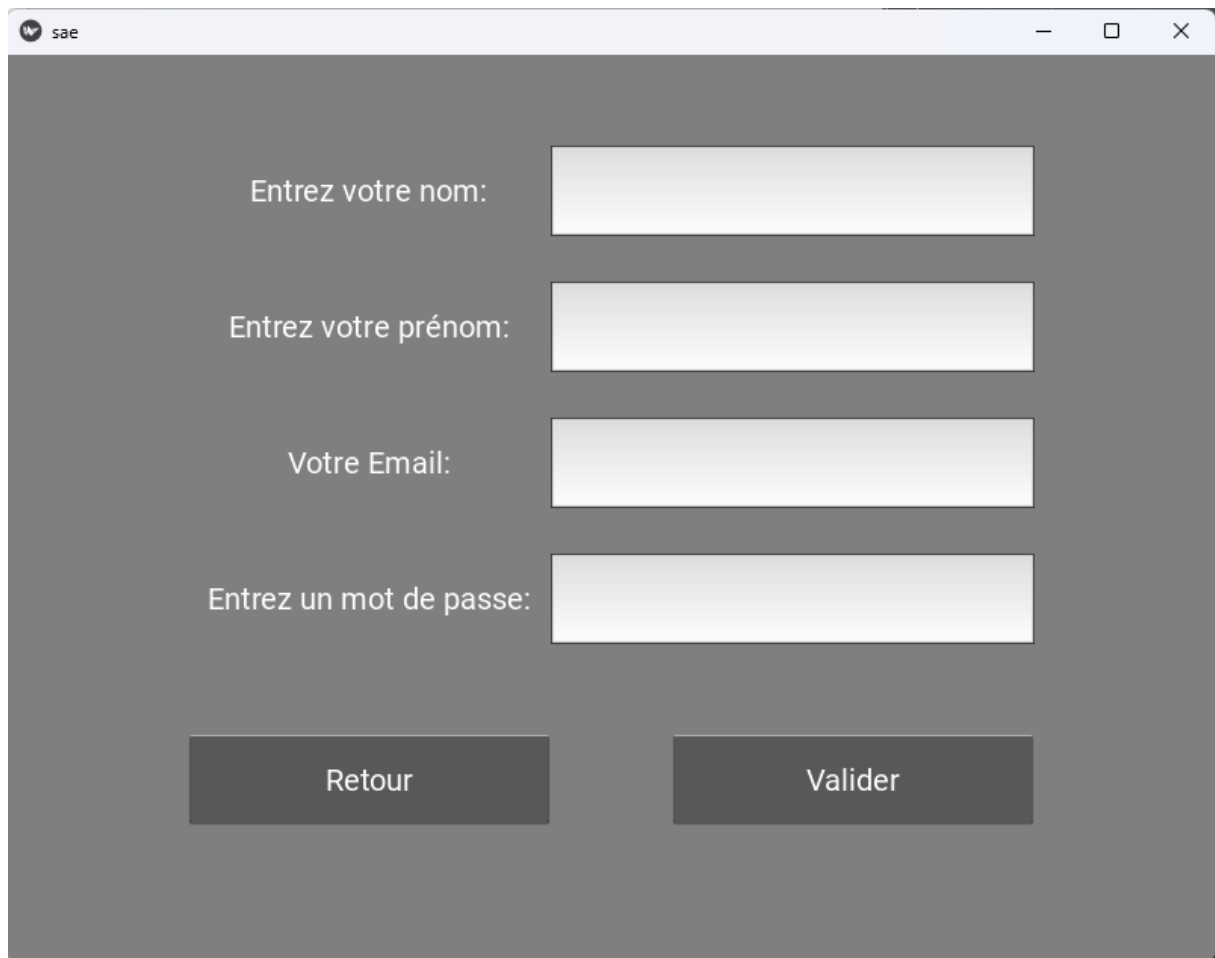
- Entrez votre nom: [Text input field]
- Entrez votre prénom: [Text input field]
- Votre Email: [Text input field]
- Votre année d'étude: [Dropdown menu with text "Sélectionnez votre année d'étude"]
- Sélectionnez votre spécialité : [Dropdown menu with text "Sélectionnez une matière"]
- Sélectionnez votre image: [Image selection button with text "Sélectionnez votre image"]
- Entrez un mot de passe: [Text input field]
- Retour [Button]
- Valider [Button]

Entrez vos données personnelles puis sélectionner votre image en cliquant sur le bouton :



Pour naviguer cliquer sur les icones de dossier avec un seul clic de souris. Pour visualiser l'image double-cliquer dessus. Une fois l'image choisie appuyer sur « valider » puis encore sur « valider » pour enregistrer votre compte.

Pour l'inscription des professeurs entrez vos données puis cliquer sur « valider » :



The screenshot shows a web application window with a title bar containing a logo and the text 'sae'. The window has standard minimize, maximize, and close buttons. The main content area is a registration form with a dark gray background. It contains four text input fields, each preceded by a label: 'Entrez votre nom:', 'Entrez votre prénom:', 'Votre Email:', and 'Entrez un mot de passe:'. Below the input fields are two buttons: 'Retour' and 'Valider'.

Une fois inscrit vous pouvez vous connecter à vos espaces et accéder à vos données.

Pour l'interface prof vous disposez d'une fonction de recherche de l'étudiant et de modification de la note :

Entrez le nom d'un étudiant

2A hatim@univ.fr hatim

Modifiez une note:

Maths	15
Réseau	6.0
Programmation	9.25
Anglais	2.25
Télécom	14.25
Cybersécurité	12.25

Code source :

Sae.py :

```
# importations
from kivy.app import App
import shutil
from kivy.uix.screenmanager import ScreenManager, Screen, NoTransition
from kivy.lang import Builder
from kivy.properties import ObjectProperty, StringProperty
from kivy.core.window import Window
from Etudiant import Etudiant
from Prof import Prof
import mysql.connector
from hashlib import sha512
import random
from get_bina import get_binary_data
from recup_photo import recup_photo

#Classe qui gère la fenêtre de connexion
```

```
class Connexion(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
    # déclaration des variables
    my_text = StringProperty("")
    email = ObjectProperty(None)
    passwd = ObjectProperty(None)
    log = ObjectProperty(None)
    vision = ObjectProperty(None)
    bvn= ObjectProperty(None)

class SecondWindow(Screen):
    pass

#Classe pour gérer la fenêtre d'inscription pour un étudiant
class InscriptionEtu(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
    #initialisation des variables
    nom = ObjectProperty(None)
    prenom = ObjectProperty(None)
    email = ObjectProperty(None)
    annee = ObjectProperty(None)
    matiere = ObjectProperty(None)
    photo = ObjectProperty(None)
    mdp = ObjectProperty(None)
    file_path= ObjectProperty(None)
    erreur = ObjectProperty(None)

#Classe qui gère la fenêtre d'inscription pour un prof
class InscriptionProf(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.verif_prof = True
        self.verif_email_prof = True

    nom = ObjectProperty(None)
    prenom = ObjectProperty(None)
    email = ObjectProperty(None)
    mdp = ObjectProperty(None)
    erreur = ObjectProperty(None)

class EtuHome(Screen):
```

```
bvn = ObjectProperty(None)
def __init__(self, **kwargs):
    super().__init__(**kwargs)

class ProfHome(Screen):
    bvn = ObjectProperty(None)
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

class ImgEtu(Screen):

    def __init__(self, **kwargs):
        super().__init__(**kwargs)

    #Va renvoyer le chemin de l'image sélectionnée
    def selected(self, filechooser):
        if filechooser:
            self.multiselect = False
            #print(type(filechooser))
            self.ids.img.source = filechooser[0]
            self.path_photo = filechooser[0]

class WindowManager(ScreenManager):
    pass

class sae(App):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        #initialisation des variables globales
        self.can_connect = False
        self.whos_connect = ""
        self.is_etu = False
        self.is_prof = False
        self.verif_etu = True
        self.verif_email_etu = True
        self.specialisation = ""

    #instanciations
    e = Etudiant()
    p = Prof()
```

```
#Permet de remplacer les backslash par deux backslash pour éviter des
erreurs
def remplacer_backslash(self,chaîne):
    return chaîne.replace("\\\\", "\\")

#Permet de remplacer les backslash et d'affecter le chemin au self.e.photo
def on_submit(self, path, selection):
    selected_file_path = selection[0]
    self.root.get_screen("ImgEtu").ids.img.source= selected_file_path
    #print(self.remplacer_backslash(selected_file_path))
    self.e.set_photo(self.remplacer_backslash(selected_file_path))

#Méthode lorsque la connexion est acceptée
def accept(self):
    self.root.get_screen("Connexion").ids.log.background_color=(0, 0, 0.4,
0.4)

#Méthode lorsque la connexion est refusée
def refuse(self):
    self.root.get_screen("Connexion").ids.log.background_color=(0.6, 0, 0,
1)

def reset_color(self):
    self.root.get_screen("Connexion").ids.my_text = ""
    self.root.get_screen("Connexion").ids.log.border=(8, 8, 8, 8)

#Méthode pour cacher ou afficher le mot de passe
def toggle_password_visibility(self, checkbox):
    if self.root.get_screen("Connexion").ids.passwd.text == "":
        return
    if checkbox.active:
        self.root.get_screen("Connexion").ids.passwd.password = False
    else:
        self.root.get_screen("Connexion").ids.passwd.password = True

def connex(self):
    lst_etu = []
    lst_prof = []
    self.whos_connect = ""
    self.is_etu = False
    self.is_prof = False

    try:
        connection_params = {
            'host' : "192.168.1.7",
            'user' : "admin",
            'password' : "sae302",
```



```
        'database' : "doss_etu"
    }
except mysql.connector.Error as e:
    print("Exception : ", e)
else:
    #requête pour obtenir chaque mail et mdp étudiant de la bdd
    request = """select email, mdp from etudiants;"""
    #requête pour obtenir chaque mail et mdp prof de la bdd
    request2 = """select email, mdp from profs;"""
    #requête pour obtenir l'id de l'étudiant

    with mysql.connector.connect(**connection_params) as db :
        with db.cursor() as c:
            c.execute(request)
            rq = c.fetchall()
            c.execute(request2)
            rq2 = c.fetchall()
        db.commit()
        db.close()

    #parcourt chaque mail et mdp étudiant
    for element in rq :
        mail = element[0]
        pwd = element[1]

        #ajoute les mails et mdp étudiant à une liste
        mlpswd = [mail, pwd]
        lst_etu.append(mlpswd)

    #parcourt chaque mail et mdp prof
    for element in rq2:
        mail = element[0]
        pwd = element[1]

        #ajoute les mails et mdp prof dans une liste
        mlpswd = [mail, pwd]
        lst_prof.append(mlpswd)

    #variable mail2 prend pour valeur le mail entré par l'utilisateur lors
    de la connexion
    mail2 = self.root.get_screen("Connexion").ids.email.text
    #variable mail2 prend pour valeur le mdp entré par l'utilisateur lors
    de la connexion
    pwd2 = self.root.get_screen("Connexion").ids.passwd.text

    pwd2_crypt = sha512(pwd2.encode()).hexdigest()
```

```
#teste si le mail et le mdp entrée par l'utilisateur correspond à une
paire mail mdp de la BDD
for elem in lst_etu:
    if elem[0] == mail2 and elem[1] == pwd2_crypt:
        self.can_connect = True
        self.is_etu = True
        self.whos_connect = "etu"

for elem in lst_prof:
    if elem[0] == mail2 and elem[1] == pwd2_crypt:
        self.can_connect = True
        self.is_prof = True
        self.whos_connect = "prof"

#si c'est le cas : CONNEXION + changement d'écran
if self.can_connect and self.is_etu:
    self.accept()
    self.root.get_screen("Connexion").manager.current = "EtuHome"
    self.reset_color()

elif self.can_connect and self.is_prof:
    self.accept()
    self.root.get_screen("Connexion").manager.current = "ProfHome"
    self.reset_color()

#si ce n'est pas le cas : CONNEXION REFUSEE
else:
    self.refuse()

def recup_matiere_notes(self):
    infos_etu = []
    try:
        connection_params = {
            'host' : "192.168.1.7",
            'user' : "admin",
            'password' : "sae302",
            'database' : "doss_etu"
        }
    except mysql.connector.Error as e:
        print("Exception : ", e)
    else:
        if self.whos_connect == "etu":
            #requête pour obtenir chaque mail et mdp étudiant de la bdd
            request = """select matiere, moyenne from etu_resume where
email = %s;"""
            request2 = """select nom, prenom, email, annee, spe, photo
from etudiants where email = %s;"""
```

```
elif self.whos_connect == "prof":
    #requête pour obtenir chaque mail et mdp prof de la bdd
    request = """select email, matiere, moyenne from
etu_resume;"""
    request2 = """select nom, prenom, email, annee, spe, photo
from etudiants;"""

    with mysql.connector.connect(**connection_params) as db :
        with db.cursor() as c:
            if self.whos_connect == "etu":
                c.execute(request,
(self.root.get_screen("Connexion").ids.email.text,))
                rq = c.fetchall()
                c.execute(request2,
(self.root.get_screen("Connexion").ids.email.text,))
                rq2 = c.fetchall()
            elif self.whos_connect == "prof":
                c.execute(request)
                rq = c.fetchall()
                c.execute(request2)
                rq2 = c.fetchall()
        db.commit()
        db.close()

    if self.whos_connect == "etu":
        notes = {"Reseau": rq[0][1], "Telecom": rq[1][1], "Maths" :
rq[2][1], "Programmation" : rq[3][1], "Anglais" : rq[4][1], rq2[0][4] :
rq[5][1]}

        infos_etu = {"Nom": rq2[0][0], "Prenom": rq2[0][1], "Email" :
rq2[0][2], "Annee" : rq2[0][3], "Photo" : rq2[0][5], "notes": notes}

    elif self.whos_connect == "prof":
        infos_etu = []
        rq_list = []
        rq2_list = []

        #transforme la liste de tuples en une liste de liste
        for note in rq:
            rq_list.append([note[0], note[1], note[2]])

        for elem in rq2:
            rq2_list.append([elem[0], elem[1], elem[2], elem[3], elem[4]])
```

```

        #ajoute à une liste les dictionnaires contenant les infos des
étudiants
        for elem in rq2:
            infos = {"Nom": elem[0], "Prenom": elem[1], "Email" : elem[2],
"Annee" : elem[3], "Spe" : elem[4], "Photo" : elem[5], "Notes": None}
            infos_etu.append(infos)

        for i in range(len(infos_etu)):
            notes = {}
            for y in range(len(rq_list)):

                if infos_etu[i]['Email'] == rq_list[y][0]:
                    notes[rq_list[y][1]] = rq_list[y][2]

            infos_etu[i]["Notes"] = notes

        self.e.set_infos_etu(infos_etu)

def reinitialise_champs_connexion(self):
    self.root.get_screen("Connexion").ids.email.text = ""
    self.root.get_screen("Connexion").ids.passwd.text = ""

def reset_connexion(self):
    self.reinitialise()

#Méthode pour la création d'un compte étudiant
def enregistrement_etudiant(self):
    #ajout des informations de l'étudiant
    self.e.set_nom(self.root.get_screen("InscriptionEtu").ids.nom.text)
    self.e.set_prenom(self.root.get_screen("InscriptionEtu").ids.prenom.te
xt)
    self.e.set_email(self.root.get_screen("InscriptionEtu").ids.email.text
)
    self.e.set_annee(self.root.get_screen("InscriptionEtu").ids.annee.text
)
    self.e.set_matiere(self.root.get_screen("InscriptionEtu").ids.matiere.
text)

```

```

        mdp_crypt =
sha512(self.root.get_screen("InscriptionEtu").ids.mdp.text.encode()).hexdigest
()

        self.e.set_mdp(mdp_crypt)

    try:
        connection_params = {
            'host' : "192.168.1.7",
            'user' : "admin",
            'password' : "sae302",
            'database' : "doss_etu"
        }
    except mysql.connector.Error as e:
        print("Exception : ", e)
    else:
        #envoi des données de l'étudiant vers la BDD
        request = """insert into
etudiants(nom,prenom,annee,spe,photo,email,mdp)
values(%s,%s,%s,%s,%s,%s,%s)"""
        params = (self.e.get_nom(), self.e.get_prenom(),
self.e.get_annee(), self.e.get_matiere(), get_binary_data(self.e.get_photo()),
self.e.get_email(), self.e.get_mdp())

        #ajout dans la table 'moy_gen'
        request2 = """insert into moy_gen(nom,prenom,mail_etu)
values(%s,%s,%s)"""
        params2 = (self.e.get_nom(), self.e.get_prenom(),
self.e.get_email())

    with mysql.connector.connect(**connection_params) as db :
        with db.cursor() as c:
            c.execute(request,params)
            c.execute(request2,params2)
        db.commit()
        db.close()

    #Méthode pour vérifier que tous les champs du formulaire sont remplis
    def verif_formulaire_etu(self):
        #initialisation à True de la variable verif à chaque vérification
        self.verif_etu = True
        #si un seul des champs n'est pas rempli
        if self.root.get_screen("InscriptionEtu").ids.nom.text == "" or
self.root.get_screen("InscriptionEtu").ids.prenom.text == "" or
self.root.get_screen("InscriptionEtu").ids.email.text == "" or
self.root.get_screen("InscriptionEtu").ids.annee.text == "Sélectionnez votre

```

```

année d'étude" or self.root.get_screen("InscriptionEtu").ids.matiere.text ==
"Sélectionnez une matière" or
self.root.get_screen("InscriptionEtu").ids.mdp.text == "" or
self.root.get_screen("InscriptionEtu").ids.img.text == "Sélectionnez votre
image":
    #envoi d'un message d'erreur
    self.root.get_screen("InscriptionEtu").ids.erreur.text = "Veuillez
remplir tous les champs du formulaire."
    #variable verif à False
    self.verif_etu = False

    #Méthode pour vérifier que le mail entré par l'utilisateur n'est pas déjà
utilisé
def verif_mail_etu(self):
    #initialisation de la variable de verif à True
    self.verif_email_etu = True
    try:
        connection_params = {
            'host' : "192.168.1.7",
            'user' : "admin",
            'password' : "sae302",
            'database' : "doss_etu"
        }
    except mysql.connector.Error as e:
        print("Exception : ", e)
    else:
        #requête pour obtenir les mails des etudiants et des profs
        request = """select email from etudiants;"""
        request2 = """select email from profs;"""

        with mysql.connector.connect(**connection_params) as db :
            with db.cursor() as c:
                c.execute(request)
                rq = c.fetchall()
                c.execute(request2)
                rq2 = c.fetchall()
            db.commit()
            db.close()

        #parcours de chaque mail, si le mail existe dans la BDD : variable
de verif passe à FALSE
        for elem in rq:
            if elem[0] ==
self.root.get_screen("InscriptionEtu").ids.email.text:
                self.verif_email_etu = False

        #parcours de chaque mail, si le mail existe dans la BDD : variable
de verif passe à FALSE
        for elem in rq2:

```

```
        if elem[0] ==
self.root.get_screen("InscriptionEtu").ids.email.text:
        self.verif_email_etu = False

#Méthode d'envoie d'un message d'erreur si le mail est déjà utilisé
def same_mail_etu(self):
    self.root.get_screen("InscriptionEtu").ids.erreur.text = "Ce mail est
déjà utilisé"

#Méthode pour le else après la condition if dans le fichier kv
def inactif(self):
    pass

#Méthode pour réinitialiser les champs du formulaire
def reinitialise_form_etu(self):
    self.root.get_screen("InscriptionEtu").ids.nom.text = ""
    self.root.get_screen("InscriptionEtu").ids.prenom.text = ""
    self.root.get_screen("InscriptionEtu").ids.email.text = ""
    self.root.get_screen("InscriptionEtu").ids.annee.text = "Sélectionnez
votre année d'étude"
    self.root.get_screen("InscriptionEtu").ids.matiere.text =
"Sélectionnez une matière"
    self.root.get_screen("InscriptionEtu").ids.mdp.text = ""
    self.root.get_screen("InscriptionEtu").ids.erreur.text = ""
    self.root.get_screen("InscriptionEtu").ids.img.text = "Sélectionnez
votre image"

#Méthode pour envoyer des notes lors de l'insription d'un étudiant
def envoie_notes(self):
    #récupère le mail et la spé de l'étudiant
    email = self.e.get_email()
    spe = self.e.get_matiere()

    #liste de toutes les matières de l'étudiant
    matieres = ["Maths", "Programmation", "Réseau", "Télécom", "Anglais",
spe]

    try:
        connection_params = {
            'host' : "192.168.1.7",
            'user' : "admin",
            'password' : "sae302",
            'database' : "doss_etu"
        }
    except mysql.connector.Error as e:
```

```

        print("Exception : ", e)
    else:
        with mysql.connector.connect(**connection_params) as db :
            with db.cursor() as c:
                for i in range(0, len(matieres)):

                    #envoi des données de l'étudiant vers la BDD
                    request = """insert into notes(mail_etu, matiere,
note) values(%s,%s,%s)"""
                    params = (email, matieres[i],
round(random.uniform(0, 20) * 4) / 4)
                    c.execute(request,params)
                    db.commit()
                db.close()

def copy_file(self, instance):
    file_path = self.file_path.text
    shutil.copy(file_path, "C:\\Users\\Hatim\\Desktop\\test")

def envoi_blob(self,fichier) :

    filename=self.selected(fichier)
    with open(filename,'rb') as f:
        data = f.read()
    self.e.set_photo= str(data)

def go_back(self, instance):
    self.manager.current = "main"

def same_mail_etu(self):
    self.root.get_screen("InscriptionEtu").ids.erreur.text = "Ce mail est
déjà utilisé"

def enregistrement_prof(self):
    self.p.set_nom(self.root.get_screen("InscriptionProf").ids.nom.text)
    self.p.set_prenom(self.root.get_screen("InscriptionProf").ids.prenom.t
ext)
    mdp_crypt =
sha512(self.root.get_screen("InscriptionProf").ids.mdp.text.encode()).hexdigest()
    self.p.set_mdp(mdp_crypt)

```



```

self.p.set_email(self.root.get_screen("InscriptionProf").ids.email.tex
t)

try:
    connection_params = {
        'host' : "192.168.1.7",
        'user' : "admin",
        'password' : "sae302",
        'database' : "doss_etu"
    }
except mysql.connector.Error as e:
    print("Exception : ", e)
else:
    request = """insert into profs(nom,prenom,email,mdp)
values(%s,%s,%s,%s)"""
    params = (self.p.get_nom(), self.p.get_prenom(),
self.p.get_email(), self.p.get_mdp())

    with mysql.connector.connect(**connection_params) as db :
        with db.cursor() as c:
            c.execute(request,params)
            db.commit()
            db.close()

def verif_formulaire_prof(self):
    self.verif_prof = True
    if self.root.get_screen("InscriptionProf").ids.nom.text == "" or
self.root.get_screen("InscriptionProf").ids.prenom.text == "" or
self.root.get_screen("InscriptionProf").ids.email.text == "" or
self.root.get_screen("InscriptionProf").ids.mdp.text == "":
        self.root.get_screen("InscriptionProf").ids.erreur.text =
"Veuillez remplir tous les champs du formulaire."
        self.verif_prof = False

def verif_mail_prof(self):
    self.verif_email_prof = True
    try:
        connection_params = {
            'host' : "192.168.1.7",
            'user' : "admin",
            'password' : "sae302",
            'database' : "doss_etu"
        }
    except mysql.connector.Error as e:
        print("Exception : ", e)
    else:
        request = """select email from etudiants;"""

```

```
request2 = """select email from profs;"""

with mysql.connector.connect(**connection_params) as db :
    with db.cursor() as c:
        c.execute(request)
        rq = c.fetchall()
        c.execute(request2)
        rq2 = c.fetchall()
    db.commit()
    db.close()

    for elem in rq:
        if elem[0] ==
self.root.get_screen("InscriptionProf").ids.email.text:
        self.verif_email_prof = False

    for elem in rq2:
        if elem[0] ==
self.root.get_screen("InscriptionProf").ids.email.text:
        self.verif_email_prof = False

def reinitialise_form_prof(self):
    self.root.get_screen("InscriptionProf").ids.nom.text = ""
    self.root.get_screen("InscriptionProf").ids.prenom.text = ""
    self.root.get_screen("InscriptionProf").ids.email.text = ""
    self.root.get_screen("InscriptionProf").ids.mdp.text = ""
    self.root.get_screen("InscriptionProf").ids.erreur.text = ""

def same_mail_prof(self):
    self.root.get_screen("InscriptionProf").ids.erreur.text = "Ce mail est
déjà utilisé"

def inactif(self):
    pass

def selected(self, filename):
    try:
        self.ids.img.source= filename[0]
        return(filename[0])
    except:
        pass
def envoi_blob(self,fichier) :
```

```

        filename=self.selected(fichier)
        with open(filename,'rb') as f:
            data = f.read()
        self.e.set_photo=data

def go_back(self, instance):
    self.manager.current = "main"

def build(self):
    pass

def page_infos_etu(self):
    inf = self.e.get_infos_etu()
    nom=inf["Nom"]
    prenom=inf["Prenom"]
    annee=inf["Annee"]
    mail=inf["Email"]
    math=round(inf["notes"]["Maths"],2)
    reseau=round(inf["notes"]["Reseau"],2)
    telecom=round(inf["notes"]["Telecom"],2)
    prog=round(inf["notes"]["Programmation"],2)
    anglais=round(inf["notes"]["Anglais"],2)
    name_specialite = list(inf["notes"].keys())[5]
    specialite = round(inf["notes"][name_specialite],2)
    self.root.get_screen("EtuHome").ids.bvn.text=str(nom+" "+prenom)
    self.root.get_screen("EtuHome").ids.annee.text=str(annee)
    self.root.get_screen("EtuHome").ids.mail.text=str(mail)
    self.root.get_screen("EtuHome").ids.math.text=str(math)
    self.root.get_screen("EtuHome").ids.reseau.text=str(reseau)
    self.root.get_screen("EtuHome").ids.prog.text=str(prog)
    self.root.get_screen("EtuHome").ids.telecom.text=str(telecom)
    self.root.get_screen("EtuHome").ids.anglais.text=str(anglais)
    self.root.get_screen("EtuHome").ids.name_specialite.text=name_speciali
te
    self.root.get_screen("EtuHome").ids.specialite.text=str(specialite)
    self.root.get_screen("EtuHome").ids.photo.source=recup_photo(nom)

```

```

moy = round(((math+reseau+telecom+prog+anglais+specialite)/6),2)
self.root.get_screen("EtuHome").ids.moyenne.text= str(moy)

def page_infos_prof(self):
    name = self.root.get_screen("ProfHome").ids.selection.text
    inf = self.e.get_infos_etu()
    trouver = False

    for i in range(0, len(inf)):
        if name == inf[i]["Nom"]:
            trouver = True
            mail=inf[i]["Email"]
            annee=inf[i]["Annee"]
            nom=inf[i]["Nom"]
            prenom=inf[i]["Prenom"]
            math=round(inf[i]["Notes"]["Maths"],2)
            reseau=round(inf[i]["Notes"]["Réseau"],2)
            telecom=round(inf[i]["Notes"]["Télécom"],2)
            prog=round(inf[i]["Notes"]["Programmation"],2)
            anglais=round(inf[i]["Notes"]["Anglais"],2)
            spe = inf[i]["Spe"]
            self.specialisation = inf[i]["Spe"]
            special = round(inf[i]["Notes"][spe],2)

            self.root.get_screen("ProfHome").ids.name_math.text="Maths"
            self.root.get_screen("ProfHome").ids.name_reseau.text="Réseau"
            self.root.get_screen("ProfHome").ids.name_prog.text="Programma
tion"

            self.root.get_screen("ProfHome").ids.name_telecom.text="Téléco
m"

            self.root.get_screen("ProfHome").ids.name_anglais.text="Anglai
s"

            self.root.get_screen("ProfHome").ids.name_spe.text=spe
            self.root.get_screen("ProfHome").ids.matiere.values =
["Maths", "Réseau", "Programmation", "Anglais", "Télécom", spe]

            self.root.get_screen("ProfHome").ids.math.text=str(math)
            self.root.get_screen("ProfHome").ids.reseau.text=str(reseau)
            self.root.get_screen("ProfHome").ids.prog.text=str(prog)
            self.root.get_screen("ProfHome").ids.telecom.text=str(telecom)
            self.root.get_screen("ProfHome").ids.anglais.text=str(anglais)

```

```

        self.root.get_screen("ProfHome").ids.special.text=str(special)
        self.root.get_screen("ProfHome").ids.photo.source=recup_photo(
nom)

        self.root.get_screen("ProfHome").ids.mail.text=mail
        self.root.get_screen("ProfHome").ids.annee.text=annee
        self.root.get_screen("ProfHome").ids.prenom.text=prenom

    if not trouver:
        if self.root.get_screen("ProfHome").ids.selection.text == "":
            self.root.get_screen("ProfHome").ids.error.text = "Entrez le
nom d'un étudiant"
        else:
            self.root.get_screen("ProfHome").ids.error.text = "Cet
étudiant n'est pas présent dans la base de données"
        else:
            self.root.get_screen("ProfHome").ids.error.text = ""

def reset_infos(self):
    self.root.get_screen("ProfHome").ids.photo.source = ""

    self.root.get_screen("ProfHome").ids.math.text=""
    self.root.get_screen("ProfHome").ids.reseau.text=""
    self.root.get_screen("ProfHome").ids.prog.text=""
    self.root.get_screen("ProfHome").ids.telecom.text=""
    self.root.get_screen("ProfHome").ids.anglais.text=""
    self.root.get_screen("ProfHome").ids.special.text = ""

    self.root.get_screen("ProfHome").ids.name_math.text=""
    self.root.get_screen("ProfHome").ids.name_reseau.text=""
    self.root.get_screen("ProfHome").ids.name_prog.text=""
    self.root.get_screen("ProfHome").ids.name_telecom.text=""
    self.root.get_screen("ProfHome").ids.name_anglais.text=""
    self.root.get_screen("ProfHome").ids.name_spe.text=""
    self.root.get_screen("ProfHome").ids.matiere.values = []
    self.root.get_screen("ProfHome").ids.annee.text = ""
    self.root.get_screen("ProfHome").ids.mail.text = ""
    self.root.get_screen("ProfHome").ids.prenom.text = ""
    self.root.get_screen("ProfHome").ids.note_error.text = ""

def reset_page_prof(self):
    self.root.get_screen("ProfHome").ids.error.text = ""
    self.root.get_screen("ProfHome").ids.selection.text = ""
    self.root.get_screen("ProfHome").ids.photo.source = ""

    self.root.get_screen("ProfHome").ids.math.text = ""
    self.root.get_screen("ProfHome").ids.reseau.text = ""

```

```

self.root.get_screen("ProfHome").ids.prog.text = ""
self.root.get_screen("ProfHome").ids.telecom.text = ""
self.root.get_screen("ProfHome").ids.anglais.text = ""
self.root.get_screen("ProfHome").ids.special.text=""
self.root.get_screen("ProfHome").ids.note_mod.text=""
self.root.get_screen("ProfHome").ids.annee.text = ""
self.root.get_screen("ProfHome").ids.mail.text = ""
self.root.get_screen("ProfHome").ids.prenom.text = ""
self.root.get_screen("ProfHome").ids.note_error.text = ""

self.root.get_screen("ProfHome").ids.name_math.text=""
self.root.get_screen("ProfHome").ids.name_reseau.text=""
self.root.get_screen("ProfHome").ids.name_prog.text=""
self.root.get_screen("ProfHome").ids.name_telecom.text=""
self.root.get_screen("ProfHome").ids.name_anglais.text=""
self.root.get_screen("ProfHome").ids.name_spe.text=""
self.root.get_screen("ProfHome").ids.matiere.text="Sélectionnez la
matière"
self.root.get_screen("ProfHome").ids.matiere.values = []

def change_text(self):
    chemin = self.e.get_photo().split("\\")

    self.root.get_screen("InscriptionEtu").ids.img.text = chemin[-1]

def modify_note(self):
    to_modify = self.root.get_screen("ProfHome").ids.matiere.text
    try:
        if 0 <= float(self.root.get_screen("ProfHome").ids.note_mod.text)
<= 20 and self.root.get_screen("ProfHome").ids.note_mod.text != "":
            if to_modify == "Maths":
                self.root.get_screen("ProfHome").ids.math.text =
self.root.get_screen("ProfHome").ids.note_mod.text
            elif to_modify == "Réseau":
                self.root.get_screen("ProfHome").ids.reseau.text =
self.root.get_screen("ProfHome").ids.note_mod.text
            elif to_modify == "Programmation":
                self.root.get_screen("ProfHome").ids.prog.text =
self.root.get_screen("ProfHome").ids.note_mod.text
            elif to_modify == "Anglais":
                self.root.get_screen("ProfHome").ids.anglais.text =
self.root.get_screen("ProfHome").ids.note_mod.text
            elif to_modify == "Télécom":
                self.root.get_screen("ProfHome").ids.telecom.text =
self.root.get_screen("ProfHome").ids.note_mod.text
            else:
                if to_modify == "Cybersécurité" or to_modify == "IA":

```

```

        self.root.get_screen("ProfHome").ids.special.text =
self.root.get_screen("ProfHome").ids.note_mod.text

        else:
            self.root.get_screen("ProfHome").ids.note_error.text
= "Veuillez choisir une matière"
        else:
            self.root.get_screen("ProfHome").ids.note_error.text =
"Veuillez entrer une note comprise entre 0 et 20"

    except ValueError:
        self.root.get_screen("ProfHome").ids.note_error.text = "Veuillez
entrer une note comprise entre 0 et 20"

    else:
        if 0 <= float(self.root.get_screen("ProfHome").ids.note_mod.text)
<= 20 and self.root.get_screen("ProfHome").ids.note_mod.text != "":
            try:
                connection_params = {
                    'host' : "192.168.1.7",
                    'user' : "admin",
                    'password' : "sae302",
                    'database' : "doss_etu"
                }
            except mysql.connector.Error as e:
                print("Exception : ", e)
            else:
                request = """update notes set note = %s where mail_etu =
%s and matiere = %s;"""
                params =
(round(float(self.root.get_screen("ProfHome").ids.note_mod.text),2),
self.root.get_screen("ProfHome").ids.mail.text, to_modify)

                with mysql.connector.connect(**connection_params) as db :
                    with db.cursor() as c:
                        c.execute(request,params)
                    db.commit()
                    db.close()

if __name__ == "__main__":

```

```
sae().run()
```

sae.kv :

```
#:kivy 1.11
#:import NoTransition kivy.uix.screenmanager.NoTransition

WindowManager:
    Connexion:
    SecondWindow:
    InscriptionEtu:
    InscriptionProf:
    EtuHome:
    ImgEtu:
    ProfHome:
<Connexion>:
    name : "Connexion"
    email: email
    passwd: passwd
    log:log
    vision: vision

    canvas.before:
        Color:
            rgba: 0.5, 0.5, 0.5, 1
        Rectangle:
            pos: self.pos
            size: self.size

    GridLayout:
        cols: 1
        size: (root.width,root.height)
        spacing: 10

        TextInput:
            id: email
            hint_text:"Email"
            multiline: False

        TextInput:
            id: passwd
            hint_text:"mot de passe"
            multiline: False
            password: True

        Label:
            text: root.my_text
            font_size: 35
```



```

        size_hint: (1, 0.1)
        background_color: (1, 1, 1, 0)

BoxLayout:
    Label:
        text:"Afficher le mot de passe"
        font_size: 20
    CheckBox:
        size_hint: (1, 1)
        on_press: app.toggle_password_visibility(self)
        active: False
        id: vision
        background_normal: ""
        background_color: (0, 0, 0.4, 0.4)
        border: (8, 8, 8, 8)
        border_radius: [20]
        color: (0.1, 0.4, 0, 1)

    Button:
        id:log
        text: "Connexion"
        font_size: 25
        on_press:
            app.connex()
            app.recup_matières_notes() if app.can_connect else
app.inactif()
            app.page_infos_etu() if app.can_connect and app.is_etu else
app.inactif()

        background_normal: ""
        background_color: (0, 0, 0.4, 0.4)
        border: (8, 8, 8, 8)
        border_radius: [20]

GridLayout:
    cols: 2
    size: (root.width,root.height)
    spacing: 10
    Button:
        text: "Inscription étudiant"
        font_size: 25
        on_release:
            app.root.current = "InscriptionEtu"
            root.manager.transition = NoTransition()
    Button:
        text: "Inscription prof"

```

```
        font_size: 25
        on_release:
            app.root.current = "InscriptionProf"
            root.manager.transition = NoTransition()

<SecondWindow>:
    name : "second"

    GridLayout:
        cols: 1
        size: (root.width,root.height)

        Label:
            text: "Seconde fenêtre"
            font_size: 32

        Button:
            text: "Retourner en arrière"
            font_size: 32
            on_release:
                app.root.current = "Connexion"
                root.manager.transition = NoTransition()

<InscriptionEtu>:
    name: "InscriptionEtu"
    nom: nom
    prenom: prenom
    email: email
    annee: annee
    matiere: matiere
    mdp: mdp
    erreur: erreur
    img:img

    canvas.before:
        Color:
            rgba: 0.5, 0.5, 0.5, 1
        Rectangle:
            pos: self.pos
            size: self.size

    GridLayout:
        cols: 2
        size: (root.width,root.height)
        spacing: 20
        padding: 20
```

```
Label:
    text: "Entrez votre nom:"
    font_size: 20
    size_hint_y: None
    height: 30

TextInput:
    id: nom
    multiline: False

Label:
    text: "Entrez votre prénom:"
    font_size: 20
    size_hint_y: None
    height: 30

TextInput:
    id: prenom
    multiline: False

Label:
    text: "Votre Email:"
    font_size: 20
    size_hint_y: None
    height: 30

TextInput:
    id: email
    multiline: False

Label:
    text: "Votre année d'étude:"
    font_size: 20
    size_hint_y: None
    height: 30

Spinner:
    id: annee
    text: "Sélectionnez votre année d'étude"
    values: ["1A", "2A"]

Label:
    text: "Sélectionnez votre spécialité :"
    font_size: 20
    size_hint_y: None
    height: 30

Spinner:
```

```
id: matiere
text: "Sélectionnez une matière"
values: ["IA", "Cybersécurité"]

Label:
    text: "Sélectionnez votre image:"
    font_size: 20
    size_hint_y: None
    height: 30

Button:
    id: img
    text: "Sélectionnez votre image"
    font_size: 25
    on_release:
        app.root.current = "ImgEtu"
        root.manager.transition = NoTransition()

Label:
    text: "Entrez un mot de passe"
    font_size: 20
    size_hint_y: None
    height: 30

TextInput:
    id: mdp
    multiline: False
    password: True

Button:
    text: "Retour"
    font_size: 25
    size_hint: 0.5, None
    height: 50
    background_color: 1, 1, 1, 1
    on_release:
        app.root.current = "Connexion"
        root.manager.transition = NoTransition()
    on_press:
        app.reinitialise_form_etu()

Button:
    text: "Valider"
    font_size: 25
    size_hint: 0.5, None
    height: 50
    background_color: 1, 1, 1, 1
```

```

        on_release:
            app.verif_formulaire_etu() #vérifie si tous les champs du
formulaire sont remplis
            app.verif_mail_etu()
            app.enregistrement_etudiant() if app.verif_etu and
app.verif_email_etu else app.inactif() #si tout est rempli, envoie des champs
vers la bdd
            app.envoie_notes() if app.verif_etu and app.verif_email_etu
else app.inactif()
            app.reinitialise_form_etu() if app.verif_etu and
app.verif_email_etu else app.inactif() #si tous est rempli, reinitialise tous
les champs
            app.same_mail_etu() if not app.verif_email_etu and
app.verif_etu else app.inactif()

        GridLayout:
            cols: 1
            Label:
                id: erreur
                text: ""

<ImgEtu>:
    name: "ImgEtu"
    canvas.before:
        Color:
            rgba: 0.5, 0.5, 0.5, 1
        Rectangle:
            pos: self.pos
            size: self.size
    GridLayout:
        cols: 1
        size: (root.width, root.height)
        spacing: 10
        Image:
            id: img
            source: ""
        FileChooserIconView:
            id: filechooser
            on_submit: app.on_submit(filechooser.path, filechooser.selection)
        Button:
            text: "Valider"
            on_release:
                app.root.current = "InscriptionEtu"
                root.manager.transition = NoTransition()
                app.change_text()

<InscriptionProf>:
    name: "InscriptionProf"

```

```
nom: nom
prenom: prenom
email: email
mdp: mdp
erreur: erreur

canvas.before:
    Color:
        rgba: 0.5, 0.5, 0.5, 1
    Rectangle:
        pos: self.pos
        size: self.size

RelativeLayout:
    orientation: 'vertical'

    pos : self.pos
    size : root.size
    Label:
        text: "Entrez votre nom:"
        font_size:20
        size_hint: 0.2,0.2
        pos_hint: {'center_x':.30, 'center_y':.85}

    TextInput:
        id: nom
        multiline: False
        markup: True
        font_size:20
        size_hint: 0.4,0.1
        pos_hint: {'center_x':.65, 'center_y':.85}

    Label:
        text: "Entrez votre prénom:"
        font_size:20
        size_hint: 0.2,0.2
        pos_hint: {'center_x':.30, 'center_y':.70}

    TextInput:
        id: prenom
        multiline: False
        font_size:20
        size_hint: 0.4,0.1
        pos_hint: {'center_x':.65, 'center_y':.70}

    Label:
```

```
text: "Votre Email:"
font_size:20
size_hint: 0.2,0.2
pos_hint:{'center_x':.30, 'center_y':.55}

TextInput:
    id: email
    multiline: False
    font_size:20
    size_hint: 0.4,0.1
    pos_hint:{'center_x':.65, 'center_y':.55}

Label:
    text: "Entrez un mot de passe:"
    font_size:20
    size_hint: 0.2,0.2
    pos_hint:{'center_x':.30, 'center_y':.40}

TextInput:
    id: mdp
    multiline: False
    password: True
    font_size:20
    size_hint: 0.4,0.1
    pos_hint:{'center_x':.65, 'center_y':.40}

Button:
    text: "Retour"
    font_size:20
    size_hint: 0.3,0.1
    pos_hint:{'center_x':.3, 'center_y':.2}
    background_color: 1, 1, 1, 1
    on_release:
        app.root.current = "Connexion"
        root.manager.transition = NoTransition()
    on_press:
        app.reinitialise_form_prof()

Button:
    text: "Valider"
    font_size:20
    size_hint: 0.3,0.1
    pos_hint:{'center_x':.7, 'center_y':.2}
    background_color: 1, 1, 1, 1
    on_release:
```

```

        app.verif_formulaire_prof() #vérifie si tous les champs du
formulaire sont remplis
        app.verif_mail_prof()
        app.enregistrement_prof() if app.verif_prof and
app.verif_email_prof else app.inactif() #si tout est rempli, envoie des champs
vers la bdd

        app.reinitialise_form_prof() if app.verif_prof and
app.verif_email_prof else app.inactif() #si tous est rempli, reinitialise tous
les champs

        app.same_mail_prof() if not app.verif_email_prof and
app.verif_prof else app.inactif()

    Label:
        id: erreur
        text: ""

<EtuHome>:
    name : "EtuHome"
    bvn:bvn
    math:math
    reseau:reseau
    telecom:telecom
    prog:prog
    anglais:anglais
    name_specialite: name_specialite
    specialite: specialite
    photo: photo
    mail:mail
    annee:annee
    moyenne:moyenne
    canvas.before:
        Color:
            rgba: 0.5, 0.5, 0.5, 1
        Rectangle:
            pos: self.pos
            size: self.size
    RelativeLayout:
        orientation:'vertical'

        pos : self.pos
        size : root.size

    Image:
        id: photo
        source: ''
        size_hint:0.4,None

```



```
pos_hint: {'center_x': .5, 'center_y': .9}

Label:
    text: "Bienvenue:"
    markup: True
    font_size: 20
    size_hint: 0.2, 0.2
    pos_hint: {'center_x': .25, 'center_y': .95}

Label:
    id: bvn
    text: ""
    markup: True
    font_size: 20
    size_hint: 0.2, 0.2
    pos_hint: {'center_x': .25, 'center_y': .90}

Label:
    id: mail
    text: ""
    markup: True
    font_size: 20
    size_hint: 0.2, 0.2
    pos_hint: {'center_x': .7, 'center_y': .96}

Label:
    id: annee
    text: ""
    markup: True
    font_size: 35
    size_hint: 0.2, 0.2
    pos_hint: {'center_x': .7, 'center_y': .90}

Button:
    text: "Se Deconnecter"
    font_size: 18
    size_hint: .3, .1
    pos_hint: {'center_x': .1, 'center_y': .8}
    height: 20
    background_color: 1, 1, 1, 1
    on_release:
        app.reinitialise_champs_connexion()
        app.root.current = "Connexion"
        root.manager.transition = NoTransition()

Label:
    text: "Maths"
    markup: True
    font_size: 30
```

```
        size_hint: 0.2,0.2
        pos_hint: {'center_x':.30, 'center_y':.70}
Label:
    id:math
    text:""
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.75, 'center_y':.70}
Label:
    text:"Réseau"
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.31, 'center_y':.60}
Label:
    id:reseau
    text:""
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.75, 'center_y':.60}
Label:
    text:"Programmation"
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.38, 'center_y':.50}
Label:
    id:prog
    text:""
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.75, 'center_y':.50}
Label:
    text:"Anglais"
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.31, 'center_y':.40}
Label:
    id:anglais
    text:""
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.75, 'center_y':.40}
Label:
```

```
text:"Télécom"
markup: True
font_size:30
size_hint: 0.2,0.2
pos_hint: {'center_x':.32, 'center_y':.30}
Label:
    id:telecom
    text:""
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.75, 'center_y':.30}

Label:
    id: name_specialite
    text:""
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.33, 'center_y':.20}
Label:
    id:specialite
    text:""
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.75, 'center_y':.20}
Label:
    text:"Moyenne totale"
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.38, 'center_y':.10}
Label:
    id:moyenne
    text:""
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.75, 'center_y':.10}
Label:
    id:moyenne
    text:""
    markup: True
    font_size:30
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.75, 'center_y':.10}
```

```

<ProfHome>:
  name : "ProfHome"
  math:math
  name_math:name_math
  reseau:reseau
  name_reseau:name_reseau
  telecom:telecom
  name_telecom:name_telecom
  prog:prog
  name_prog:name_prog
  anglais:anglais
  name_anglais:name_anglais
  special:special
  name_spe:name_spe
  selection: selection
  error: error
  photo: photo
  note_mod:note_mod
  matiere:matiere
  bout_mod:bout_mod
  mail:mail
  annee:annee
  prenom:prenom
  note_error:note_error
  canvas.before:

    Color:
      rgba: 0.5, 0.5, 0.5, 1
    Rectangle:
      pos: self.pos
      size: self.size
  RelativeLayout:
    orientation:'vertical'

    pos : self.pos
    size : root.size
    Image:
      id: photo
      source: ''
      size_hint: 0.2,0.2
      pos_hint: {'center_x':.85, 'center_y':.9}

    Label:
      text: "Entrez le nom d'un étudiant"
      markup: True
      font_size:15
      size_hint: 0.2,0.2
      pos_hint: {'center_x':.2, 'center_y':.9}

```

```
TextInput:
    id: selection
    multiline: False
    markup: True
    font_size:15
    size_hint: 0.2,0.06
    pos_hint:{'center_x':.45, 'center_y':.9}

Button:
    text: "Valider"
    markup: True
    font_size:15
    size_hint: 0.1,0.08
    pos_hint:{'center_x':.65, 'center_y':.9}
    on_release:
        app.reset_infos()
        app.page_infos_prof()

Label:
    id:mail
    text: ""
    markup: True
    font_size:20
    size_hint: 0.2,0.2
    pos_hint:{'center_x':.5, 'center_y':.75}
Label:
    id:annee
    text: ""
    markup: True
    font_size:20
    size_hint: 0.2,0.2
    pos_hint:{'center_x':.2, 'center_y':.75}
Label:
    id:prenom
    text: ""
    markup: True
    font_size:20
    size_hint: 0.2,0.2
    pos_hint:{'center_x':.85, 'center_y':.75}

Label:
    id:error
    text: ""
    font_size:15
    size_hint: 0.2,0.2
    pos_hint:{'center_x':.5, 'center_y':.75}

Button:
```

```
text: "Se déconnecter"
font_size: 16
markup: True
font_size:15
size_hint: 0.2,0.1
pos_hint: {'center_x':.5, 'center_y':.15}
on_release:
    app.reinitialise_champs_connexion()
    app.reset_page_prof()
    app.root.current = "Connexion"
    root.manager.transition = NoTransition()

Label:
    text: "Modifiez une note:"
    font_size:20
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.23, 'center_y':.65}
TextInput:
    id: note_mod
    multiline: False
    markup: True
    font_size:16
    size_hint: 0.1,0.06
    pos_hint: {'center_x':.45, 'center_y':.65}
Label:
    id:note_error
    text:""
    font_size:17
    size_hint: 0.2,0.2
    pos_hint: {'center_x':.45, 'center_y':.56}
Spinner:
    id: matiere
    text: "Sélectionnez la matière"
    values: []
    font_size:15
    size_hint: 0.25,0.08
    pos_hint: {'center_x':.7, 'center_y':.65}
Button:
    id:bout_mod
    text: "Modifier"
    markup: True
    font_size:15
    size_hint: 0.1,0.08
    pos_hint: {'center_x':.9, 'center_y':.65}
    on_release:
        app.modify_note()
        app.recup_matiere_notes()

Label:
```

```
id:name_math
text:""
font_size:20
size_hint: 0.2,0.2
pos_hint: {'center_x':.29, 'center_y':.50}
Label:
id:math
text:""
font_size:20
size_hint: 0.2,0.2
pos_hint: {'center_x':.5, 'center_y':.50}

Label:
id:name_reseau
text:""
font_size:20
size_hint: 0.2,0.2
pos_hint: {'center_x':.3, 'center_y':.45}
Label:
id:reseau
text:""
font_size:20
size_hint: 0.2,0.2
pos_hint: {'center_x':.5, 'center_y':.45}
Label:
id:name_prog
text:""
font_size:20
size_hint: 0.2,0.2
pos_hint: {'center_x':.35, 'center_y':.40}
Label:
id:prog
text:""
font_size:20
size_hint: 0.2,0.2
pos_hint: {'center_x':.5, 'center_y':.40}
Label:
id:name_anglais
text:""
font_size:20
size_hint: 0.2,0.2
pos_hint: {'center_x':.3, 'center_y':.35}
Label:
id:anglais
text:""
font_size:20
size_hint: 0.2,0.2
pos_hint: {'center_x':.5, 'center_y':.35}
Label:
```

```

        id:name_telecom
        text:""
        font_size:20
        size_hint: 0.2,0.2
        pos_hint:{'center_x':.31, 'center_y':.30}
    Label:
        id:telecom
        text:""
        font_size:20
        size_hint: 0.2,0.2
        pos_hint:{'center_x':.5, 'center_y':.30}

    Label:
        text:""
        id: name_spe
        font_size:20
        size_hint: 0.2,0.2
        pos_hint:{'center_x':.31, 'center_y':.25}
    Label:
        id:special
        text:""
        font_size:20
        size_hint: 0.2,0.2
        pos_hint:{'center_x':.5, 'center_y':.25}

```

recup_photo.py :

```

import mysql.connector

def recup_photo(nom):
    # Établissement de la connexion
    cnx = mysql.connector.connect(user='admin', password='sae302',
                                   host='192.168.1.7', database='doss_etu')

    query = "SELECT photo FROM etudiants WHERE nom = %s"
    values = (nom,)

    # Exécution de la requête
    cursor = cnx.cursor()
    cursor.execute(query, values)

    # Récupération des données binaires du fichier
    data = cursor.fetchone()[0]

```



```
# Écriture des données binaires dans un nouveau fichier
with open(nom+'.png', 'wb') as f:
    f.write(data)

# Fermeture de la connexion
cnx.close()

# print(nom+".png")
return nom+".png"
```

Prof.py :

```
from personne import Personne

class Prof(Personne):
    #constructeur de la classe etudiant
    def __init__(self):
        Personne.__init__(self)
        self.__mdp = ""
        self.__infos_etu = []

    #accesseurs et mutateurs
    def get_mdp(self):
        return self.__mdp

    def set_mdp(self, nv_mdp):
        self.__mdp = nv_mdp

    def get_infos_etu(self):
        return self.__infos_etu

    def set_infos_etu(self, nv_infos_etu):
        self.__infos_etu = nv_infos_etu

    #méthode pour afficher les infos d'un étudiant
    def Affiche(self):
        liste_personne = Personne.Affiche(self)
        return [liste_personne[0], liste_personne[1], liste_personne[2],
self.__mdp]
```

personne.py :

```
class Personne :
```

```
#constructeur de la classe personne
def __init__(self):
    self.__nom = ""
    self.__prenom = ""
    self.__email = ""

# accesseurs et mutateurs
def get_nom(self):
    return self.__nom

def set_nom(self, nv_nom):
    self.__nom = nv_nom

def get_prenom(self):
    return self.__prenom

def set_prenom(self, nv_prenom):
    self.__prenom = nv_prenom

def get_email(self):
    return self.__email

def set_email(self, nv_email):
    self.__email = nv_email

#méthode pour afficher les infos de la personne
def Affiche(self):
    return [self.__nom, self.__prenom, self.__email]
```

get_bina.py :

```
def get_binary_data(image_path):
    with open(image_path, 'rb') as image_file:
        binary_data = image_file.read()
        image_file.close()
        #print(binary_data)

    return binary_data
```

vidage_table.py :

```
import mysql.connector
```

```
try:
    connection_params = {
        'host': "192.168.1.7",
        'user': "admin",
        'password': "sae302",
        'database': "doss_etu"
    }
except mysql.connector.Error as e:
    print("Exception : ", e)
else:
    with mysql.connector.connect(**connection_params) as db :
        with db.cursor() as c:
            c.execute("SET FOREIGN_KEY_CHECKS = 0;")
            c.execute("truncate table etudiants;")
            c.execute("truncate table moy_gen;")
            c.execute("truncate table notes;")
            c.execute("truncate table profs;")
            c.execute("SET FOREIGN_KEY_CHECKS = 1;")

        db.commit()
        db.close()
```

Etudiant.py :

```
from personne import Personne

class Etudiant(Personne):
    #constructeur de la classe etudiant
    def __init__(self):
        Personne.__init__(self)
        self.__annee = 0
        self.__matiere = ""
        self.__photo = ""
        self.__mdp = ""
        self.__infos_etu = {}

    #accesseurs et mutateurs
    def get_annee(self):
        return self.__annee

    def set_annee(self, nv_annee):
        self.__annee = nv_annee

    def get_matiere(self):
        return self.__matiere
```

```
def set_matiere(self, nv_matiere):
    self.__matiere = nv_matiere

def get_photo(self):
    return self.__photo

def set_photo(self, nv_photo):
    self.__photo = nv_photo

def get_mdp(self):
    return self.__mdp

def set_mdp(self, nv_mdp):
    self.__mdp = nv_mdp

def get_infos_etu(self):
    return self.__infos_etu

def set_infos_etu(self, nv_infos_etu):
    self.__infos_etu = nv_infos_etu

#méthode pour afficher les infos d'un étudiant
def Affiche(self):
    liste_personne = Personne.Affiche(self)
    return [liste_personne[0], liste_personne[1], liste_personne[2],
self.__annee, self.__matiere, self.__photo, self.__mdp]
```