

CR SAE 3.02

Développer des Applications Communicantes

Introduction :

Après avoir participé au modules de programmation python et kivy, ainsi qu'à l'utilisation de base de données avec python, il est normal d'approfondir nos connaissances et ce en faisant de multiples projets. Ainsi parmi les projets disponibles, nous avons choisi le sujet 2 qui consiste au développement d'une application qui gère une interface d'étudiants. Comme dit dans le sujet :"La tâche principale du projet est le développement d'une application Python qui gère la situation courante d'un groupe d'étudiants." .

Prérequis :

Il y a de nombreux prérequis à la réalisation de ce projet. Il y a tout d'abord un environnement de travail requis, qui représente une machine pour programmer sur python ainsi qu'un accès à une base de données. Il y a aussi quelques outils et logiciels qui nous seront nécessaire au cours de ce projet :

- **Installer pip:**

```
sudo apt install python3-pip  
pip list  
python3 -m pip install pip --upgrade
```

- **Installer kivy**

```
python3 -m pip install kivy
```

- **Installer Pleyer**

```
python3 -m pip install pleyer
```

- **Installer WebSockets & WebSocket-Client**

```
python3 -m pip install websockets  
python3 -m pip install websocket-client
```

- **Installer Buldozer**

```
python3 -m pip install bulldozer  
pip install --user https://github.com/kivy/bulldozer/archive/master.zip  
sudo apt install git  
git clone https://github.com/kivy/bulldozer  
cd bulldozer  
python3 setup.py build  
pip install -e .  
cd ~  
dans .bashrc export PATH=~/local/bin/:$PATH  
relancer source .bashrc
```

- **Installer mysql.connector :**

```
pip3 install mysql-connector-python==8.0.29
```

Contexte :

Dans un groupe d'étudiants, on veut pouvoir les enregistrer en fonction de plusieurs critères. Ils sont, l'identifiant unique de l'étudiant qui permet de représenter son identité, et qui ne sera constitué que de chiffres. Le nom et prénom de l'étudiant qui serviront seulement à titre indicatif et qui peuvent être les mêmes qu'un autre étudiant. L'année d'étude qui comme le nom et prénom de serviront qu'à titre indicatif. Les moyennes de l'étudiant dans 4 matières : Mathématiques, Anglais, Réseaux, Informatique. Ces matières forment la moyenne générale de l'élève en question. Le mot de passe de l'étudiant pour qu'il puisse se connecter à son compte pour voir ses informations. Enfin la photo de l'étudiant. Ainsi on veut que notre projet soit composé en plus de la programmation python et kv, d'une base de donnée qui contiendra toutes ces informations pour les étudiants de la classe. Il y aura alors deux tables, l'une "étudiants" qui possède : l'identifiant unique, le nom et prénom, l'année, le mot de passe, la moyenne générale et la photo. La deuxième table possède : l'identifiant et les 4 matières. Pour consulter les élèves déjà inscrits, on utilise une liste déroulante qui effectue des requêtes à la base de données.

Dans le cadre de ce projet une VM était disponible afin d'y héberger une base de données afin de s'y connecter avec mysql.connector ce qui était notre cas au début du projet cependant après réflexion cela était d'une part plus simple pour nous d'héberger notre propre bdd autre que sur une vm en panama pour y travailler depuis chez nous mais d'autant plus sur le plan cybersécurité.

Nous sommes étudiant en parcours cybersécurité nous devons penser et réfléchir à comment sécuriser une application car il est assez simple de réaliser du "reverse engineering" avec certains logiciels et donc récolter toutes nos informations dans notre code et bdd sans aucun contrôle. C'est pour ça que nous avons eu l'idée de passer/créer une api rest qui sera gérée par une page PHP hébergée sur mon serveur étudiant ainsi qu'une bd mysql gérée par phpmyadmin sur mon péda web personnel . Et de ce fait l'application est dites plus sécurisé et peut être utilisé à distance

Résultats attendus :

Les résultats attendus seraient tout d'abord pour la partie interface soit kivy, une interface de connexion, soit pour se login ou pour s'enregistrer.



Voici ci-dessus un croquis de l'écran duquel nous nous sommes inspirés.

Ensuite pour le formulaire où l'on rentre les informations , on voulez un résultat qui ressemble à cela :

Identifiant	Année
Nom	Prénom
Moyenne	

Envoyer

Ensuite pour l'accès au informations de la base de donnée, nous voulions avoir quelque chose qui ressemble au schéma du sujet soit :

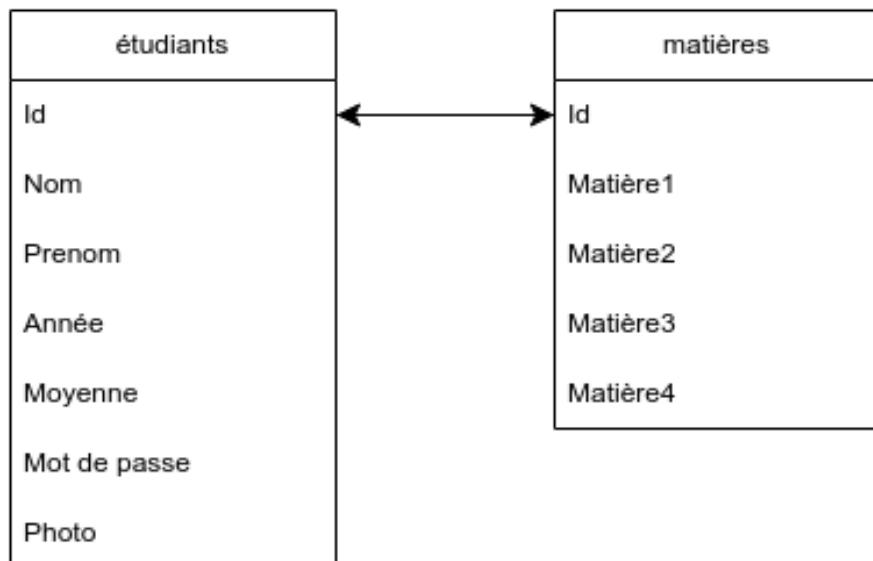
Formulaire Etudiant	
	Interroger la BD
	Aide

Liste des étudiants

Poret ▾

Nom	Année	Moyenne	Photo
Poret	2	16.75	

La base de donnée devait ressembler à cela :



Résultats obtenus :

Python :

La première étape du développement est l'import des fonctions approprié et nécessaire :

```
import kivy
from kivy.uix.widget import Widget
from kivy.app import App
from kivy.properties import ObjectProperty
from kivy.lang import Builder
from kivy.config import Config
from kivy.uix.screenmanager import ScreenManager, Screen
import base64
import io
from kivy.core.image import Image as CoreImage
from kivy.uix.image import Image
from kivy.uix.filechooser import FileChooser
from kivy.uix.image import AsyncImage
import mysql.connector
import requests
import json
import base64
import io
from kivy.core.image import Image as CoreImage
from kivy.uix.image import Image
```

- **import kivy** importe la bibliothèque Kivy pour utilisation dans le code.
- **from kivy.uix.widget import Widget** importe la classe de base Widget de Kivy pour créer des widgets personnalisés.
- **from kivy.app import App** importe la classe de base App de Kivy pour créer des applications.
- **from kivy.properties import ObjectProperty** importe la propriété ObjectProperty de Kivy pour lier des propriétés de widgets entre eux.
- **from kivy.lang import Builder** importe le constructeur de langue de Kivy pour créer des widgets à partir de fichiers KV.
- **from kivy.config import Config** importe la classe de configuration de Kivy pour configurer l'application.
- **from kivy.uix.screenmanager import ScreenManager, Screen** importe les classes ScreenManager et Screen de Kivy pour gérer les écrans dans une application.
- **import mysql.connector** importe la bibliothèque python pour se connecter à une base de donnée mysql.
- **import requests** importe la bibliothèque python pour envoyer des requêtes HTTP.
- **import json** importe la bibliothèque python pour travailler avec des données en format json.
- **import base64** importe la bibliothèque python pour encoder/decoder des données en base64.
- **from kivy.uix.filechooser import FileChooser** importe la classe FileChooser de Kivy pour créer un sélecteur de fichier.
- **from kivy.uix.image import AsyncImage** importe la classe AsyncImage de Kivy pour afficher des images de manière asynchrone.

```
url = "https://thomas-cecchini-etu.pedaweb.univ-amu.fr/extranet/base.php"

Config.set('graphics', 'width', '370')
Config.set('graphics', 'height', '700')
```

Ce code ci-dessus permet de définir la variable vers laquelle se trouve la base de données.

Le code en dessous permet d'initialiser la taille de la fenêtre.

Ce code définit une classe appelée "recup" qui possède quatre méthodes:

- `__init__` qui est le constructeur de la classe et qui initialise les variables d'instance `__identifiant` et `__idetu`.
- `get_identifiant` qui retourne la valeur de `__identifiant`
- `set_identifiant` qui permet de définir la valeur de `__identifiant` en enlevant les espaces.
- `get_idetu` qui retourne la valeur de `__idetu`.
- `set_idetu` qui permet de définir la valeur de `__idetu` en enlevant les espaces.

Et en dernière ligne, une variable "Ratio" est créée et est initialisée en instanciant la classe `recup`.

```
class recuperation:
    def __init__(self):
        self.__identifiant= ""
        self.__idetu=""

    def get_identifiant(self):
        print(self.__identifiant)
        print("1")
        return self.__identifiant

    def set_identifiant(self,identifiant):
        print("2",identifiant)
        self.__identifiant=str(identifiant.replace(" ", ""))

    def get_idetu(self):
        return self.__idetu

    def set_idetu(self,idetu):
        self.__idetu=str(idetu.replace(" ", ""))

Ratio = recuperation()
```

```

class Connexion(Screen):

    idetu=ObjectProperty(None)
    mdp=ObjectProperty(None)
    affi=ObjectProperty(None)
    button=ObjectProperty(None)

    def togglevisibility(self):
        if self.mdp.password == True:
            self.mdp.password = False
            self.button.text = 'Hide'

        elif self.mdp.password == False:
            self.mdp.password = True
            self.button.text = 'Show'

    def login(self):

        Ratio.set_idetu(self.idetu.text)
        mdp=self.mdp.text
        affi=self.affi.text

        etudiant={'table':'etudiants'}
        response = requests.get('https://thomas-cecchini-
etu.pedaweb.univ-amu.fr/extranet/base.php',params=etudiant)
        data = response.json()
        # print(data)
        liste= []

        for student in data:
            liste.append(student["id"])

        #print (liste)

        if self.idetu.text not in liste:
            print("Identifiant/mdp Incorrect")
            self.affi.text="Identifiant/mdp Incorrecte"
            self.idetu.text=""
            self.mdp.text=""

        else:
            print("Identifiant correcte")
            self.affi.text="Identifiant correcte"

```

Cette classe Python s'appelle "Connexion" et est utilisée pour créer une interface de connexion pour un utilisateur. Elle possède plusieurs propriétés, telles que "idetu", "mdp", "affi" et "button", qui sont toutes des propriétés de l'objet.

La méthode "togglevisibility" permet de rendre le mot de passe visible ou caché en fonction de l'état actuel. La méthode "login" est utilisée pour vérifier si les informations d'identification entrées par l'utilisateur sont correctes en les comparant à une liste d'identifiants existants. Si les informations sont correctes, un message de confirmation est affiché, sinon un message d'erreur est affiché.

Ce code définit une classe appelée "FirstWindow" qui hérite de la classe "Screen" de Kivy. La classe possède plusieurs attributs qui sont définis comme "ObjectProperty" et qui sont par défaut à "None".

Ces variables peuvent être utilisées pour stocker des informations d'identification, des informations d'année scolaire, des informations de nom et prénom, des informations de matières et un mot de passe. Ces propriétés peuvent être liées à des widgets dans l'interface utilisateur pour fournir une fonctionnalité de saisie de données à l'utilisateur.

```

class FirstWindow(Screen):

    identifiant=ObjectProperty(None)
    annee=ObjectProperty(None)
    nom=ObjectProperty(None)
    prenom=ObjectProperty(None)
    matiere1=ObjectProperty(None)
    matiere2=ObjectProperty(None)
    matiere3=ObjectProperty(None)
    matiere4=ObjectProperty(None)
    mdp=ObjectProperty(None)

```

```

    def press(self):
        url = "https://thomas-cecchini-etu.pedaweb.univ-amu.fr/extranet/base.php"
        identifiant=self.identifiant.text
        annee=self.annee.text
        nom=self.nom.text
        prenom=self.prenom.text
        matiere1=self.matiere1.text
        matiere2=self.matiere2.text
        matiere3=self.matiere3.text
        matiere4=self.matiere4.text
        mdp=selfmdp.text

        print({identifiant,annee,nom,prenom,matiere1,matiere2,matiere3,matiere4,mdp})

```

Ce code définit une méthode appelée "press" à l'intérieur de la classe "FirstWindow". Lorsque cette méthode est appelée, elle récupère le texte de chacun des attributs "ObjectProperty" définis dans la classe et les affecte à des variables locales avec le même nom.

La fonction représente la fonction qui permet l'envoie des données sur la base de données.

Ce code permet d'initialiser les variables dans lesquelles on va envoyer les informations réunies vers la base de données.

On peut voir les tables vers lesquelles les envoies sont effectuées et les éléments dans les tables.

```

insert_etudiants = {
    "table": "etudiants", # La table qu'on veut affecté
    "id": identifiant,
    "nom": nom, # la clef et sa valeur
    "prenom": prenom, # la clef et sa valeur
    "annee": annee, # la clef et sa valeur
    "password": mdp # la clef et sa valeur
}

insert_matières = {
    "table": "matières", # La table qu'on veut affecté
    "id_etu": identifiant, # la clef et sa valeur
    "matiere1": matiere1, # la clef et sa valeur
    "matiere2": matiere2, # la clef et sa valeur
    "matiere3": matiere3, # la clef et sa valeur
    "matiere4": matiere4 # la clef et sa valeur
}

```

```

if identifiant.isdigit():
    identifiant = int(identifiant)
else:
    print("Identifiant doit être un nombre entier")

if annee.isdigit():
    annee = int(annee)
else:
    print("Année doit être un nombre entier")

if matiere1.isdigit():
    matiere1 = float(matiere1)
else:
    print("La note d'Anglais doit être un chiffre/nombre")

if matiere2.isdigit():
    matiere2 = float(matiere2)
else:
    print("La note de Mathématiques doit être un chiffre/nombre")

if matiere3.isdigit():
    matiere3 = float(matiere3)
else:
    print("La note d'Informatique doit être un chiffre/nombre")

if matiere4.isdigit():
    matiere4 = float(matiere4)
else:
    print("La note de Réseaux doit être un chiffre/nombre")

```

Ce code si représente tous les tests dans les variables, pour être sûr que les données récupérées soient bien valides, par exemple que l'ID soit bien un entier, que l'année soit bien un entier, que les noms et prénoms soient bien des chaînes de caractères, etc...

```
response_etudiants = requests.post(url, json=insert_etudiants)
response_matieres = requests.post(url, json=insert_matieres)
```

Ces lignes de code utilisent la bibliothèque requests de Python pour effectuer une demande HTTP POST à l'URL spécifiée. La première ligne effectue une demande pour insérer des données dans la table "étudiants", tandis que la seconde ligne effectue une demande pour insérer des données dans la table "matières". La méthode requests.post() est utilisée pour envoyer les données dans les tables spécifiées en utilisant les paramètres json=insert_etudiants et json=insert_matieres. Il s'agit donc d'une requête SQL INSERT INTO pour insérer les données dans les tables correspondantes.

Ce code ci permet l'initialisation des variables pour qu'elles soient vides au moment où on les affiche, et que l'on puisse écrire dedans.

```
self.identifiant.text=""
self.annee.text=""
self.nom.text=""
self.prenom.text=""
self.matiere1.text=""
self.matiere2.text=""
self.matiere3.text=""
self.matiere4.text=""
self.mdp.text=""

identifiant2=identifiant
return identifiant2
```

```

class SecondWindow(Screen):

    def selected(self, filename):

        self.ids.my_image.source = filename[0]
        chemin = filename[0]
        print(filename[0])

        # Define the URL of the API
        url = "https://thomas-cecchini-etu.pedaweb.univ-
amur.fr/extranet/base.php"
        # Open the image file and read its content
        with open(chemin, "rb") as image_file:
            print(image_file)
            encoded_string = base64.b64encode(image_file.read()).decode("utf-8")

        # Define the data for the insert request
        data = {
            "table": "profile",
            "id_etu": "21220018",
            "image": encoded_string
        }

        # Make a POST request to insert data
        response = requests.post(url, json=data)
        print(response.status_code)
        print(response.text)

```

Ce code définit une classe appelée "SecondWindow" qui hérite de la classe "Screen" de Kivy. Il contient une méthode appelée "selected" qui prend un seul argument "filename".

Cette méthode utilise le nom de fichier passé en paramètre pour définir la source d'un widget Image avec un id "my_image" au fichier sélectionné. Il ouvre ensuite le fichier image en utilisant la déclaration "with open" et lit son contenu, puis il encode le contenu du fichier image en une chaîne de caractères base64 à l'aide de la méthode "base64.b64encode" et décode en utf-8.

Il définit ensuite un dictionnaire de données avec des clés "table", "id_etu" et "image" et des valeurs "profile", "21220018" et encoded_string respectivement. Il envoie ensuite une requête POST à l'URL spécifiée avec des données json, et le code de statut et le texte de la réponse sont imprimés.

Cette classe est utilisée pour envoyer une image à un serveur via une requête HTTP en utilisant la bibliothèque "requests".

Cette classe Python s'appelle "Interroger_la_bd" et est utilisée pour interroger une base de données en ligne via une requête HTTP GET. Elle possède plusieurs propriétés, telles que "idetu", "annee", "nom", "prenom", "mdp", "statut" et "image", qui sont toutes des propriétés de l'objet.

La méthode "intero_photo" est utilisée pour récupérer l'image associée à l'identifiant de l'étudiant, en utilisant la méthode requests.get() pour récupérer les données de l'image à partir de la base de données en ligne. Les données de l'image sont ensuite décodées en utilisant base64 et utilisées pour mettre à jour la propriété "image" de l'objet. La méthode "intero" est utilisée pour récupérer les informations de l'étudiant, en utilisant la méthode requests.get() pour récupérer les données de l'étudiant à partir de la base de données en ligne, puis en utilisant ces données pour mettre à jour les propriétés de l'objet.

Les données récupérées sont ensuite utilisées pour mettre à jour les champs de l'interface utilisateur pour afficher les informations de l'étudiant.

```

class Interroger_la_bd(Screen):
    idetu=ObjectProperty(None)
    annee=ObjectProperty(None)
    nom=ObjectProperty(None)
    prenom=ObjectProperty(None)
    mdp=ObjectProperty(None)
    statut=ObjectProperty(None)
    image=ObjectProperty(None)

    def intero_photo(self):
        trille=Ratio.get_idetu()
        etudiant={'table':'profile','id':trille}
        response = requests.get('https://thomas-cecchini-
etu.pedaweb.univ-amu.fr/extranet/base.php',params=etudiant)
        print(response)
        data = response.json()
        print(data)

        blob=data["image"]

        self.image.source=""
        self.image.data= io.BytesIO(base64.b64decode(blob))
        self.image.coreimage=CoreImage(self.image.data, ext ="jpg")
        self.image.texture=self.image.coreimage.texture

    def intero(self):
        trille=Ratio.get_idetu()
        # self.idetu.text=Ratio.get_idetu()
        idetu=self.idetu.text
        annee=self.annee.text
        nom=self.nom.text
        prenom=self.prenom.text
        statut=self.statut.text

        etudiant={'table':'etudiants','id':trille}
        response = requests.get('https://thomas-cecchini-
etu.pedaweb.univ-amu.fr/extranet/base.php',params=etudiant)
        data = response.json()
        print(data)

        self.idetu.text=data["id"]
        self.nom.text=data["nom"]
        self.prenom.text=data["prenom"]
        self.annee.text=data["annee"]
        self.statut.text=data["statut"]

        self.intero_photo()

```

```

class FourWindow(Screen):

    matiere1=ObjectProperty(None)
    matiere2=ObjectProperty(None)
    matiere3=ObjectProperty(None)
    matiere4=ObjectProperty(None)
    average=ObjectProperty(None)

    def intero_mat(self):

        trille=Ratio.get_idetu()
        # self.idetu.text=Ratio.get_idetu()
        matiere1=self.matiere1.text
        matiere2=self.matiere2.text
        matiere3=self.matiere3.text
        matiere4=self.matiere4.text
        label=self.label.text

        etudiant={'table':'matieres','id':trille}
        response = requests.get('https://thomas-cecchini-
etu.pedaweb.univ-amu.fr/extranet/base.php',params=etudiant)
        data = response.json()
        print(data)

        self.matiere1.text=data["matiere1"]
        self.matiere2.text=data["matiere2"]
        self.matiere3.text=data["matiere3"]
        self.matiere4.text=data["matiere4"]

        average= (int(self.matiere1.text) + int(self.matiere2.text) +
        int(self.matiere3.text)+ int(self.matiere4.text)) / 4
        self.label.text = str(average)

```

Ce code définit une classe appelée "MyApp" qui hérite de la classe "App" de Kivy. Il contient une méthode "build" qui est utilisée pour construire l'interface utilisateur de l'application. Il charge un fichier .kv qui contient les instructions de disposition de l'interface utilisateur.

La méthode "run()" est ensuite appelée pour lancer l'application.

Ce code est utilisé pour lancer l'application en utilisant la bibliothèque Kivy, en chargeant le fichier .kv qui contient les instructions de disposition de l'interface utilisateur.

Cette classe Python s'appelle "FourWindow" et est utilisée pour afficher les informations de quatre matières pour un étudiant particulier en utilisant une requête HTTP GET. Elle possède plusieurs propriétés, telles que "matiere1", "matiere2", "matiere3", "matiere4" et "average" qui sont toutes des propriétés de l'objet. La méthode "intero_mat" est utilisée pour récupérer les informations de 4 matières pour l'étudiant en utilisant la méthode requests.get() pour récupérer les données des matières à partir de la base de données en ligne, puis en utilisant ces données pour mettre à jour les propriétés de l'objet. Il calcule également la moyenne des notes obtenues dans ces 4 matières et l'affiche dans le label average.

```

kv = Builder.load_file("My.kv")

class MyApp(App):
    def build(self):
        return kv

if __name__ == "__main__":
    MyApp().run()

```

Kivy :

Ceci est un fichier de langage Kivy qui définit la mise en page et le comportement de l'écran "Interroger_la_bd" dans une application basée sur Kivy. La classe WindowManager est utilisée pour gérer différents écrans dans l'application, et l'écran "Interroger_la_bd" est l'un des écrans répertoriés. La mise en page de l'écran est définie par le bloc <Interroger_la_bd>, qui inclut des propriétés telles que "idetu", "nom", "prenom", "statut", "annee" et "image" qui sont toutes définies comme des propriétés d'objet et liées aux widgets correspondants dans le fichier kv. Le bloc canvas.before ajoute un arrière-plan gris semi-transparent à l'écran.

Ce fichier KV est utilisé pour créer la mise en page pour l'écran "Interroger_la_bd" dans l'application et pour définir le comportement des différents widgets dans l'écran.

```

WindowManager:
    Authentification:
    Connexion:
    FirstWindow:
    SecondWindow:
    FourWindow:
    Interroger_la_bd:

<Interroger_la_bd>:

    name: "interroger"
    idetu:idetu
    nom:nom
    prenom:prenom
    statut:statut
    annee:annee
    image:image

    canvas.before:
        Color:
            rgba:(0.7,0.7,0.7,0.7)
        Rectangle:
            pos: self.pos
            size: self.size

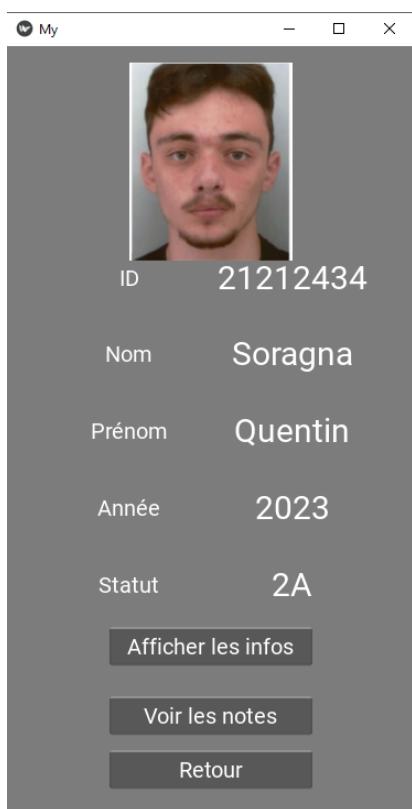
```

Ceci est la mise en page pour l'écran "Interroger_la_bd" dans une application basée sur Kivy. Il utilise un RelativeLayout avec une orientation verticale, qui positionne les éléments selon leurs propriétés pos_hint et size_hint.

Un widget Image est utilisé pour afficher l'image de l'étudiant, qui est récupérée depuis la base de données dans la fonction "intero_photo()".

Il y a plusieurs widgets Label utilisés pour afficher les informations de l'étudiant, comme son ID, son nom, son prénom, son année et son statut. Ces informations sont récupérées depuis la base de données dans la fonction "intero()". Il y a également 2 boutons, un pour afficher les informations et l'autre pour afficher les notes.

ex:



```

RelativeLayout:
    orientation: "vertical"
    pos : self.pos
    size: root.size

    Image:
        id:image
        source: ..
        size_hint: 0.4, 0.4
        pos_hint: {"center_x": 0.5, "center_y": 0.85}

    Label:
        text:'ID'
        font_size:"20"
        size_hint:1, 0.1
        pos_hint:{'center_x':.30, "center_y":.70}

    Label:
        id:idetu
        text:''
        font_size:"30"
        size_hint:1, 0.1
        pos_hint:{'center_x':.70, "center_y":.70}

    Label:
        text:'Nom'
        font_size:"20"
        size_hint:1, 0.1
        pos_hint:{'center_x':.30, "center_y":.60}

    Label:
        id:nom
        text:''
        font_size:"30"
        size_hint:1, 0.1
        pos_hint:{'center_x':.70, "center_y":.60}

    Label:
        text:'Prénom'
        font_size:"20"
        size_hint:1, 0.1
        pos_hint:{'center_x':.30, "center_y":.50}

    Label:
        id:prenom
        text:''
        font_size:"30"
        size_hint:1, 0.1
        pos_hint:{'center_x':.70, "center_y":.50}

    Label:
        text:'Année'
        font_size:"20"
        size_hint:1, 0.1
        pos_hint:{'center_x':.30, "center_y":.40}

    Label:
        id:annee
        text:''
        font_size:"30"
        size_hint:1, 0.1
        pos_hint:{'center_x':.70, "center_y":.40}

    Label:
        text:'Statut'
        font_size:"20"
        size_hint:1, 0.1
        pos_hint:{'center_x':.30, "center_y":.30}

    Label:
        id:statut
        text:''
        font_size:"30"
        size_hint:1, 0.1
        pos_hint:{'center_x':.70, "center_y":.30}

    Button:
        text: "Afficher les infos"
        font_size:"20"
        size_hint:0.5, 0.05
        pos_hint:{'center_x':.5, "center_y":.22}
        on_press:root.intero()
        #,root.intero_photo()

    Button:
        text: "Voir les notes"
        font_size:"20"
        size_hint:0.5, 0.05
        pos_hint:{'center_x':.5, "center_y":.13}
        on_release:
            app.root.current = "quatre"

```

```
<FirstWindow>
    name: "first"
    spinner:spinner
    identifiant:identifiant
    annee:annee
    nom:nom
    prenom:prenom
    matiere1:matiere1
    matiere2:matiere2
    matiere3:matiere3
    matiere4:matiere4
    mdp:mdp

    canvas.before:
        Color:
            rgba:(0.7,0.7,0.7,0.7)
        Rectangle:
            pos: self.pos
            size: self.size
```

Ceci est la mise en page de l'écran "FirstWindow" dans une application basée sur Kivy. Le bloc `<FirstWindow>` définit les propriétés de l'écran, telles que "spinner", "identifiant", "annee", "nom", "prenom", "matiere1", "matiere2", "matiere3", "matiere4" et "mdp" qui sont toutes des propriétés de l'objet liées aux widgets correspondants dans le fichier kv.

Le `canvas.before` block ajoute un arrière-plan gris semi-transparent à l'écran. Cette partie de code est utilisée pour créer la mise en page pour l'écran "FirstWindow" dans l'application et pour définir les propriétés de l'écran et les widgets qui y sont contenus.

```

RelativeLayout:
    orientation:"vertical"
    pos : self.pos
    size : root.size

Spinner:
    id:spinner
    text:"Choisit une options"
    font_size:"20"
    size_hint:0.88,0.05
    values:['IA', '2A', 'Profs']
    size:[140, 400]
    pos_hint:{'center_x': .5, 'center_y': .99}

RelativeLayout:
    orientation:"vertical"
    pos : self.pos
    size : root.size
    Label:
        text:"ID"
        font_size:"20"
        size_hint:0.1
        pos_hint:{'center_x': .25, 'center_y': .75}

TextInput:
    id:idnum
    multiline:False
    font_size:"20"
    size_hint:0.42, 0.05
    pos_hint:{'center_x': .75, 'center_y': .78}

Label:
    text:"Année"
    font_size:"20"
    size_hint:0.1
    pos_hint:{'center_x': .75, 'center_y': .75}

TextInput:
    id:year
    multiline:False
    font_size:"20"
    size_hint:0.42, 0.05
    pos_hint:{'center_x': .75, 'center_y': .78}

Label:
    text:"Nom"
    font_size:"20"
    size_hint:0.1
    pos_hint:{'center_x': .25, 'center_y': .65}

TextInput:
    id:nom
    multiline:False
    font_size:"20"
    size_hint:0.42, 0.05
    pos_hint:{'center_x': .75, 'center_y': .68}

Label:
    text:"Prénom"
    font_size:"20"
    size_hint:0.1
    pos_hint:{'center_x': .75, 'center_y': .65}

TextInput:
    id:prenom
    multiline:False
    font_size:"20"
    size_hint:0.42, 0.05
    pos_hint:{'center_x': .75, 'center_y': .68}

Label:
    text:"Anglais"
    font_size:"20"
    size_hint:0.1
    pos_hint:{'center_x': .25, 'center_y': .55}

TextInput:
    id:materiel
    multiline:False
    font_size:"20"
    size_hint:0.42, 0.05
    pos_hint:{'center_x': .75, 'center_y': .58}

Label:
    text:"Mathématiques"
    font_size:"20"
    size_hint:0.1
    pos_hint:{'center_x': .25, 'center_y': .45}

TextInput:
    id:materiere2
    multiline:False
    font_size:"20"
    size_hint:0.42, 0.05
    pos_hint:{'center_x': .75, 'center_y': .58}

Label:
    text:"Informatique"
    font_size:"20"
    size_hint:0.1
    pos_hint:{'center_x': .25, 'center_y': .45}

TextInput:
    id:materiere3
    multiline:False
    font_size:"20"
    size_hint:0.42, 0.05
    pos_hint:{'center_x': .75, 'center_y': .48}

Label:
    text:"Réseaux"
    font_size:"20"
    size_hint:0.1
    pos_hint:{'center_x': .25, 'center_y': .45}

TextInput:
    id:materiere4
    multiline:False
    font_size:"20"
    size_hint:0.42, 0.05
    pos_hint:{'center_x': .75, 'center_y': .48}

Label:
    text:"Mot de passe"
    font_size:"20"
    size_hint:0.1
    pos_hint:{'center_x': .5, 'center_y': .35}

TextInput:
    id:mdp
    multiline:False
    font_size:"20"
    size_hint:0.5, 0.05
    pos_hint:{'center_x': .5, 'center_y': .38}
    on_release:
        app.root.current = "second"
        root.manager.transition.direction = "left"

Button:
    text: "Choisir Image"
    font_size:"20"
    size_hint:0.58, 0.05
    pos_hint:{'center_x': .5, 'center_y': .78}
    on_press:root.press()

Button:
    text: "Enregistrer"
    font_size:"20"
    size_hint:0.58, 0.05
    pos_hint:{'center_x': .5, 'center_y': .82}
    on_press:root.press()

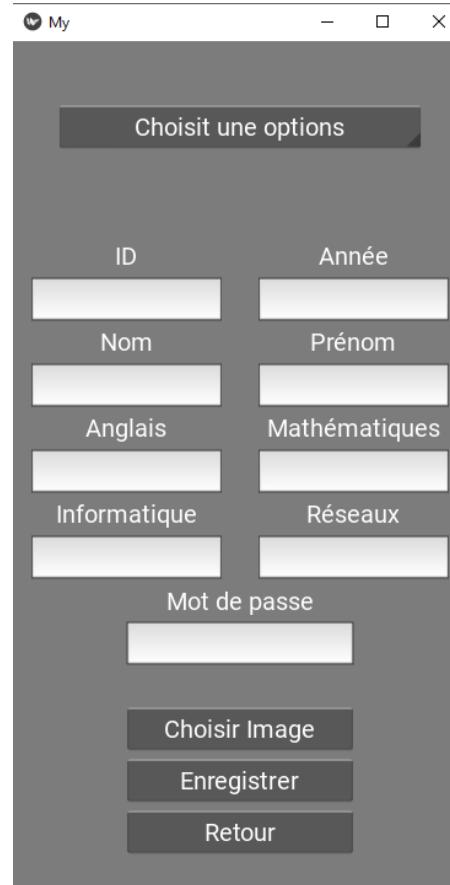
Button:
    text: "Retour"
    font_size:"20"
    size_hint:0.58, 0.05
    pos_hint:{'center_x': .5, 'center_y': .86}
    on_press:root.press()

Button:
    text: "Inregistrer"
    font_size:"20"
    size_hint:0.58, 0.05
    pos_hint:{'center_x': .5, 'center_y': .88}
    on_release:
        app.root.current = "auth"
        root.manager.transition.direction = "right"

```

Ce code définit une interface utilisateur pour une application Kivy. Il utilise des widgets tels que Spinner, Label, TextInput et Image pour créer un formulaire pour saisir des informations sur un étudiant, y compris son identifiant, son année, son nom, son prénom, ses notes de matières et un mot de passe. Il utilise également des propriétés de l'objet pour stocker les valeurs saisies dans les champs de saisie, ainsi que des propriétés de positionnement pour aligner les widgets sur l'écran. Il y a aussi une fonction intero_mat() qui permet d'interroger la base de données pour afficher les notes de matières de l'étudiant.

Ex :



La classe SecondWindow définit une interface graphique pour envoyer une image à un serveur. Il contient un champ de sélection de fichier pour choisir l'image à envoyer, un bouton "Envoyer" pour envoyer l'image sélectionnée et un bouton "Retour" pour retourner à la première fenêtre. Il utilise également un identifiant pour l'image, qui peut être utilisé pour afficher l'image sélectionnée à l'écran. La méthode selected() est appelée lorsque l'utilisateur sélectionne un fichier et pourrait être utilisée pour envoyer l'image au serveur.



```

<SecondWindow>:
    name: "second"
    id:my_widget
    my_image:my_image
    # identifiant: identifiant

BoxLayout:
    orientation: "vertical"
    size: root.width, root.height
    padding: 50
    spacing: 20

    Image:
        id: my_image
        source: ""

    # Label:
    #     id: identifiant
    #     source: ""

FileChooserIconView:
    id: filechooser
    on_selection: my_widget.selected(filechooser.selection)

Button:
    text:"Envoyer"
    font_size:"20"
    size_hint:0.50, 0.05
    pos_hint:{"center_x":.5, "center_y":.18}
    on_press:root.selected(filechooser.selection)

    #Verifie la ligne ci-dessus

Button:
    text: "Retour"
    font_size:"20"
    size_hint:0.50, 0.05
    pos_hint:{"center_x":.5, "center_y":.12}
    on_release:
        app.root.current = "first"
        root.manager.transition.direction = "right"

```

```

<FourWindow>
    name: "quatre"
    matiere4:matiere4
    matiere3:matiere3
    matiere2:matiere2
    matiere1:matiere1
    label:label

    canvas.before:
        Color:
            rgba:(0.7,0.7,0.7,0.7)
        Rectangle:
            pos: self.pos
            size: self.size

    RelativeLayout:
        orientation: "vertical"
        pos : self.pos
        size: root.size

    Image:
        source: 'notes.png'
        size_hint: 0.4, 0.4
        pos_hint: {'center_x': 0.5, 'center_y': 0.85}

    Label:
        text:'Anglais'
        font_size:"20"
        size_hint:1, 0.1
        pos_hint:{'center_x': .30, "center_y": .70}

    Label:
        id:matiere1
        text:''
        font_size:"30"
        size_hint:1, 0.1
        pos_hint:{'center_x': .70, "center_y": .70}

    Label:
        text:'Mathématiques'
        font_size:"20"
        size_hint:1, 0.1
        pos_hint:{'center_x': .30, "center_y": .60}

    Label:
        id:matiere2
        text:''
        font_size:"30"
        size_hint:1, 0.1
        pos_hint:{'center_x': .70, "center_y": .60}

    Label:
        text:'Informatique'
        font_size:"20"
        size_hint:1, 0.1
        pos_hint:{'center_x': .30, "center_y": .50}

    Label:
        id:matiere3
        text:''
        font_size:"30"
        size_hint:1, 0.1
        pos_hint:{'center_x': .70, "center_y": .50}

    Label:
        text: 'Réseaux'
        font_size:"20"
        size_hint:1, 0.1
        pos_hint:{'center_x': .30, "center_y": .40}

    Label:
        id:matiere4
        text:''
        font_size:"30"
        size_hint:1, 0.1
        pos_hint:{'center_x': .70, "center_y": .40}

    Label:
        text: 'Moyenne Générale'
        font_size:"20"
        size_hint:1, 0.1
        pos_hint:{'center_x': .30, "center_y": .30}

    Label:
        id:label
        text:''
        font_size:"30"
        size_hint:1, 0.1
        pos_hint:{'center_x': .70, "center_y": .30}

    Button:
        text: "Voir les notes"
        font_size:"20"
        size_hint:0.50, 0.05
        pos_hint:{'center_x':.5, "center_y":.20}
        on_press:root.interro_mat()

    Button:
        text: "Accueil"
        font_size:"20"
        size_hint:0.50, 0.05
        pos_hint:{'center_x':.5, "center_y":.12}
        on_release:
            app.root.current = "auth"
            root.manager.transition.direction = "right"

    Button:
        text: "Retour"
        font_size:"20"
        size_hint:0.50, 0.05
        pos_hint:{'center_x':.5, "center_y":.04}
        on_release:
            app.root.current = "interroger"
            root.manager.transition.direction = "right"

```

Ceci semble être un fichier de description de l'interface utilisateur pour une application Kivy. Il décrit l'apparence de différentes fenêtres dans l'application, telles que la première fenêtre, la deuxième fenêtre, la fenêtre d'authentification et la fenêtre d'interrogation de la base de données. Il utilise des widgets tels que des images, des boutons, des labels, des entrées de texte et des sélecteurs de fichiers pour créer l'interface utilisateur. Il utilise également des paramètres tels que les tailles de police, les positions et les dispositions pour organiser les widgets sur l'écran. Les fonctions sont appelées lorsque les boutons sont pressés pour effectuer des actions telles que l'interrogation de la base de données et l'affichage des informations.



Ceci est un fichier KV qui décrit un écran d'authentification pour une application. Il utilise le layout RelativeLayout pour organiser les différents éléments graphiques sur l'écran. Il utilise également des images pour les boutons de connexion et d'inscription. Lorsque les boutons sont pressés, ils déclenchent des transitions vers des écrans de connexion et d'inscription respectivement.



```

● ○ ●

<Authentification>:
    name: "auth"

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

    RelativeLayout:
        orientation:"vertical"
        pos : self.pos
        size: root.size

    Image:
        source: 'iut.jpg'
        size_hint: 1, 1
        pos_hint: {'center_x': 0.5, 'center_y': 0.80}

    Button:
        background_normal: 'login.png'
        background_down: 'login.png'
        size_hint:0.80,0.20
        pos_hint:{'center_x":0.5, "center_y":.43}
        on_release:
            app.root.current = "Connexion"
            root.manager.transition.direction = "left"

    Button:
        background_normal: 'Inscription.png'
        background_down: 'Inscription.png'
        size_hint:0.80,0.20
        pos_hint:{'center_x":0.5, "center_y":.27}
        on_release:
            app.root.current = "first"
            root.manager.transition.direction = "left"

```

```

<Connexion>
    name: "Connexion"
    idetu:idetu
    mdp:mdp
    affi:affi
    button:button

    canvas.before:
        Color:
            rgba:(0.7,0.7,0.7,0.7)
        Rectangle:
            pos: self.pos
            size: self.size

    RelativeLayout:
        oriental:"vertical"
        pos : self.pos
        size: root.size

    Image:
        source: 'iutrt2.png'
        size_hint: 1, 1
        pos_hint: {'center_x': 0.5, 'center_y': 0.86}

    Label:
        text:"Identifiant"
        font_size:"30"
        size_hint:0.90, 0.5
        pos_hint:{'center_x':.50, "center_y":.70}

    TextInput:
        id:idetu
        multiline:False
        font_size:"20"
        size_hint:0.90, 0.05
        hint_text: "Enter votre identifiant"
        pos_hint:{'center_x':.50, "center_y":.63}

    Label:
        text:"Mot de passe"
        font_size:"30"
        size_hint:0.90, 0.5
        pos_hint:{'center_x':.50, "center_y":.53}

    TextInput:
        id:mdp
        multiline:False
        font_size:"20"
        size_hint:0.90, 0.05
        pos_hint:{'center_x':.50, "center_y":.45}
        hint_text: "Enter votre mot de passe"
        password: True

    Button:
        id: button
        size_hint: .2, .05
        pos_hint: {"center_x": .5, "center_y": .40}
        text:"Hide"
        on_press: root.togglevisibility()

    Label:
        id:affi
        text:''
        font_size:"20"
        size_hint:1, 0.1
        pos_hint:{'center_x':.50, "center_y":.35}

    Button:
        text:"Vérifier"
        font_size:"20"
        size_hint:0.50, 0.05
        pos_hint:{'center_x':.5, "center_y":.28}
        on_press:root.login()

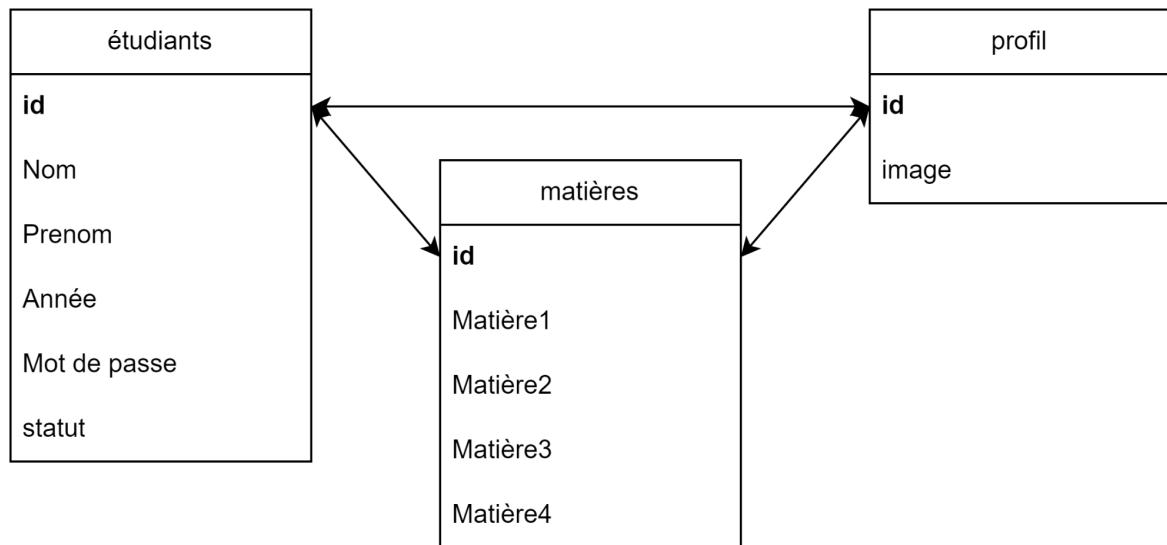
    Button:
        text:"Connexion"
        font_size:"20"
        size_hint:0.50, 0.05
        pos_hint:{'center_x':.5, "center_y":.20}
        on_release:
            app.root.current = "interroger"
            root.manager.transition.direction = "right"

    Button:
        text: "Retour"
        font_size:"20"
        size_hint:0.50, 0.05
        pos_hint:{'center_x':.5, "center_y":.12}
        on_release:
            app.root.current = "auth"
            root.manager.transition.direction = "right"

```

Ceci est un script de langage Kivy qui crée une interface de connexion. Le script utilise la classe Connexion et configure une mise en page à l'aide du widget RelativeLayout, qui organise les widgets enfants dans une orientation verticale. La mise en page inclut une image, des étiquettes, des zones de saisie de texte pour le nom d'utilisateur et le mot de passe, des boutons pour cacher/afficher le mot de passe, vérifier et connecter, ainsi qu'un bouton pour retourner à l'écran précédent.





Voici à quoi ressemble la base de données après modification, comme on peut le voir on a enlevé la photo de la première base de données pour créer une autre table "profil". Et on a ajouté un "statut" à la table étudiants

```

<?php
header('Content-Type: application/json; charset=utf-8');
// Connexion
$conn = new mysqli("mysql.pedaweb.univ-amu.fr",
"c21220018", "wsCVnewDzfp7FYv", "c21220018");

// Error handler
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Obtenir la requête dans l'url
$method = $_SERVER['REQUEST_METHOD'];

$input = null;
switch($method) {
    case 'PUT':
        $input =
json_decode(file_get_contents('php://input'),true);
        break;
    case 'DELETE':
        $input =
json_decode(file_get_contents('php://input'),true);
        break;
    case 'GET':
        $input = $_GET;
        break;
    case 'POST':
        $input =
json_decode(file_get_contents('php://input'),true);
        break;
}

$table = $input["table"];
$key = isset($input["id"]) ? $input["id"] : null;

// Découpage du chemin
$columns = preg_replace('/[^a-zA-Z0-
9_]+/i','',array_keys($input));
$values = array_map(function ($value) use ($conn) {
    if ($value==null) return null;
    return mysqli_real_escape_string($conn,(string)$value);
},array_values($input));

// Construction de la requête
$set = '';
array_shift($columns); // On enlève l'index "table" qui
nous est inutile
array_shift($values); // Et la valeur de l'index table

for ($i=0;$i<count($columns);$i++) {
    $set.=($i>0?',':'').$columns[$i].'=';
    if($columns[$i] == "password") {
        $set.=
($values[$i]==null?'NULL':"'".password_hash($values[$i],
PASSWORD_DEFAULT)."'");
    } else {
        $set.=
($values[$i]==null?'NULL':"'".$values[$i]."'");
    }
}

```

```

// Switch selon l'url
switch ($method) {
    case 'GET':
        if($table == "etudiants"){
            $sql = "select * from `$table`.".$key?" WHERE id=$key:'";
        }else if($table == "matieres"){
            $sql = "select * from `$table`.".$key?" WHERE id_matiere=$key:'";
        }else if($table == "profile"){
            $sql = "select * from `$table`.".$key?" WHERE id_etu=$key:'";
        }
        break;
    case 'PUT':
        if($table == "etudiants"){
            $sql = "update `$table` set $set where id=$key";
        }else if($table == "matieres"){
            $sql = "update `$table` set $set where id_matiere=$key";
        }else if($table == "profile"){
            $sql = "update `$table` set $set where id_etu=$key";
        }
        break;
    case 'POST':
        if($table == "etudiants"){
            $sql = "insert into `$table` set $set";
        }else if($table == "matieres"){
            $sql = "insert into `$table` set $set";
        }else if($table == "profile"){
            $sql = "insert into `$table` set $set";
        }
        break;
    case 'DELETE':
        if($table == "etudiants"){
            $sql = "delete from `$table` where id=$key";
        }else if($table == "matieres"){
            $sql = "delete from `$table` where id_matiere=$key";
        }else if($table == "profile"){
            $sql = "delete from `$table` where id_etu=$key";
        }
        break;
}

```

```

// exécution de la requête
$result = mysqli_query($conn,$sql);

// Error handler
if (!$result) {
    http_response_code(404);
    die(mysqli_error());
}

// Print result
if ($method == 'GET') {
    if (!$key) echo '[';
    for ($i=0;$i<mysqli_num_rows($result);$i++) {
        echo ($i>0?',':'').json_encode(mysqli_fetch_object($result));
    }
    if (!$key) echo ']';
} elseif ($method == 'POST') {
    echo mysqli_insert_id($conn);
} else {
    echo mysqli_affected_rows($conn);
}

// close mysql connection
$conn->close();
?>

```

Ce script PHP utilise une approche RESTful pour manipuler les données dans la base de données en utilisant les méthodes HTTP standard. Il utilise les méthodes GET, PUT, POST et DELETE pour effectuer des opérations CRUD (Create, Read, Update, Delete) sur les données de la base de données. Il lit les données dans l'URL ou dans le corps de la demande pour déterminer la table et les informations à utiliser pour la requête. Il utilise également la fonction password_hash pour crypter les mots de passe stockés dans la base de données afin de protéger la sécurité des informations des utilisateurs.

Il utilise la classe mysqli pour établir une connexion à la base de données MySQL et gérer les erreurs de connexion. Il utilise un switch pour effectuer des opérations différentes en fonction de la méthode HTTP utilisée. Pour GET, il construit une requête pour récupérer les données de la table spécifiée. Pour PUT, il construit une requête pour mettre à jour les données de la table spécifiée en utilisant les informations fournies. Pour POST, il construit une requête pour insérer des données dans la table spécifiée en utilisant les informations fournies. Et pour DELETE il construit une requête pour supprimer les données de la table spécifiée en utilisant les informations fournies.

Il utilise également la fonction json_decode pour traiter les données envoyées dans le corps de la demande pour assurer une compatibilité avec les format json et avec les données envoyées en GET.

Difficultés rencontrés :

- Blob / Filechooser

L'utilisation du blob fut une des parties les plus dures pour nous dans le projet. Nous y avons passé un nombre d'heures assez stratosphérique. Les problèmes rencontrés ont été les suivants. Premièrement nous avons passé beaucoup de temps sur un problème d'encodage de l'image pour au final utilisé base64, deuxièmement il a été compliqué de récupérer la valeur de l'id écrit sur l'écran 1 pour l'ajouter dans la bd de la table pour l'image, troisièmement la partie la plus compliquée, notre page php qui avait été mal configuré et qui empêcher les requêtes de passer que sa soit pour envoyer ou récupérer l'image. Et enfin le dernier point qui était facile à résoudre mais assez compliqué à comprendre, le programme ne pouvait pas ouvrir les photos en .png mais seulement les photos en .jpg ce qui est assez difficile à comprendre car quand on test le programme avec une image et qu'il marche et 20 lignes de codes plus loin on re test et qu'il ne marche plus on peut chercher pendant longtemps cette erreur

- PHP

Comme je vous est expliqué ci-dessus la page PHP a vraiment été un réelle défi dans lequel on a failli abandonner plusieurs fois c'est à dire que toutes les 20 lignes de codes on devait aller modifier dans notre fichier php la configuration car elle n'était pas bonne. Par exemple premièrement les requêtes request.post() ne marchait pas puis une fois patché cela a été au tour de requests.get() et ce encore actuellement n'arrivant pas à faire fonctionner la requests.delete() qui est dans notre programme python en commentaire car même en ayant passer énormément de temps à essayer de résoudres l'erreur je n'ai pas réussit à comprendre celle-ci, ce qui m'empêche donc de faire mon bouton qui permet de supprimer l'utilisateur de la db quand celui-ci est pressé

- Temps

L'un des facteurs les plus contraignants, en effets vue tout les tests avant d'arriver à ce programme finale cela nous a fait perdre beaucoup de temps (sqlite3,mysql.connector...), de plus n'ayant pas d'énorme compétence en PHP cela en faisait un réelle défi pour nous et peut être trop ambitieux. Du coup nous n'avons pas eu le temps de mettre en place quelques solutions que nous avions envisagé notamment la possibilité de se désinscrire, la possibilité que le professeurs puisse se connecter sur une page d'authentification différente et dans laquelle il aurait pu voir

chaque utilisateur ainsi que leurs informations sous le format d'une liste déroulante.
Ou bien encore améliorer l'interface graphique qui est loin d'être parfait

Conclusion :

Lors de ce projet de développement d'application de gestion d'étudiant, nous avons bien qu'avec peu de consignes, rencontrer des problèmes parfois compliqué à résoudre, cependant nous avons réussi à atteindre nos objectif, bien que nous n'étions pas contre un peus plus de temps pour perfectionner notre application, comme la mis en place du bulldozer pour créer une application mobile, ou certaine modification qui aurait apporté des fonctionnalité en plus. Nous sommes quand même plutôt satisfait de notre travail surtout sur le plan sécurité qui est une réelle fierté pour nous. Durant cette SAE nous avons considérablement améliorer notre niveau en programmation et en connaissance Python/Kivy/SQl et PHP