Fiona Fauvelle

APPLI-VISITEURS

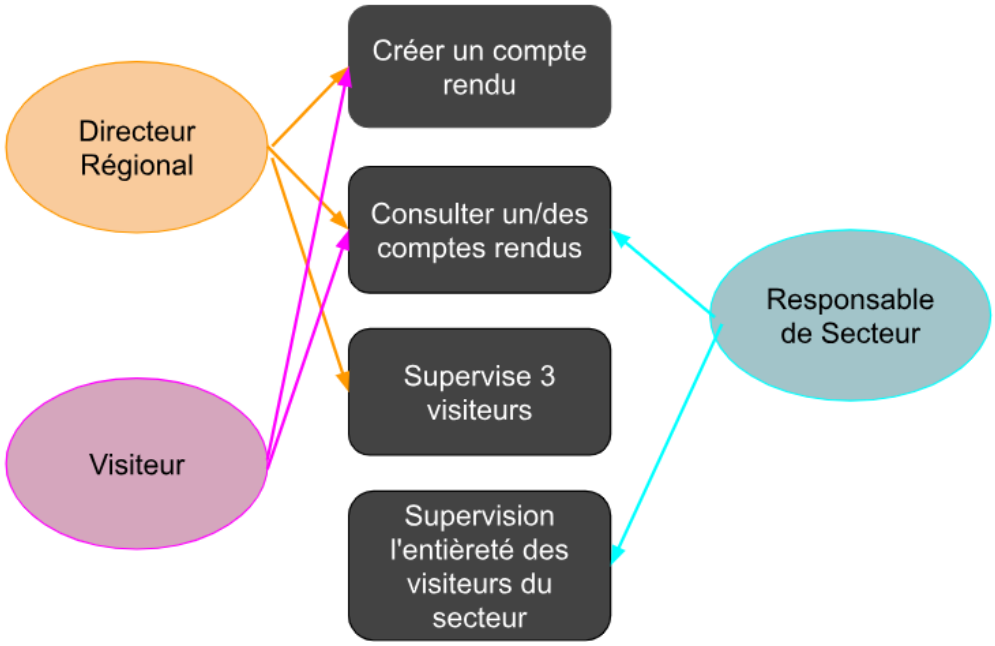
I. CONTEXTE

Le Laboratoire Galaxy Swiss Bourdin (GSB) est une entreprise fictive. Il souhaite mettre à disposition des visiteurs médicaux une application permettant notamment de centraliser les comptes-rendus de visite. Cette base d'information sera utilisée à des fins d'élaboration de la démarche de communication auprès des praticiens et donnera une vision individuelle et synthétique de l'activité de représentation. Pour permettre une aide au renseignement des rapports et à la création de présentation produits, l'application fournira une description des produits du laboratoire, les coordonnées précises des praticiens et des informations détaillées les concernant. Elle servira aussi à la mise en relation de la hiérarchie de la force commerciale, des visiteurs aux responsables de secteur en passant par les délégués régionaux.

II. SOLUTION TROUVÉE

Pour répondre à ces exigences et afin de garantir un développement efficace et moderne de l'application, nous avons choisi de créer l'application avec Python 3.

III. DIAGRAMME



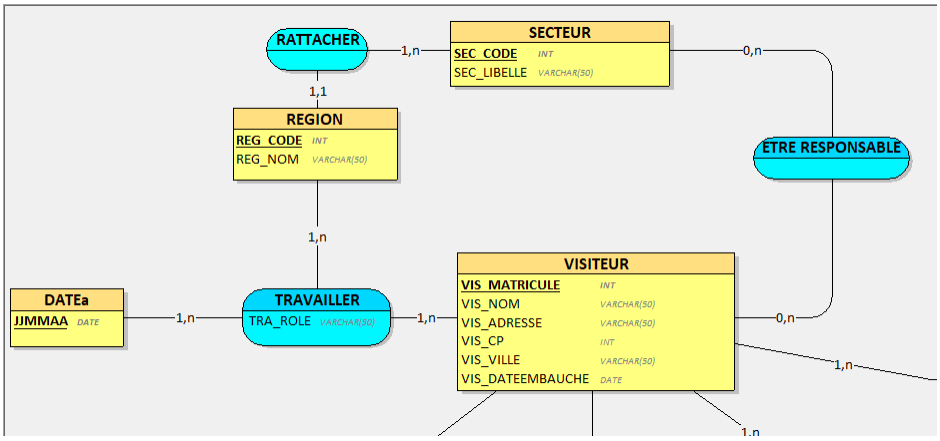
IV. L'APPLICATION

A. BASE DE DONNÉES

Pour ce projet, l'application Applivisiteurs avait déjà créé une première version de la base de données. L'objectif était d'à partir de cette BDD, reprendre l'essentiel et apporter des modifications en fonction des nouvelles attentes du client. Sur la base de données existante, on a dû retirer certaines tables n'ayant plus d'utilité pour la suite du projet : Spécialité des praticiens, Département des visiteurs, Activité Complémentaires des visiteurs, … Et encore d'autres.
On m'a demandé de me charger essentiellement de la base de données.
Pour reprendre la base de l'ancienne structure, on a reproduit la BBD sur Looping, une application de MCD. (Modélisation Conceptuelle de Données) Ainsi on reprend à partir de là.

Fiona Fauvelle

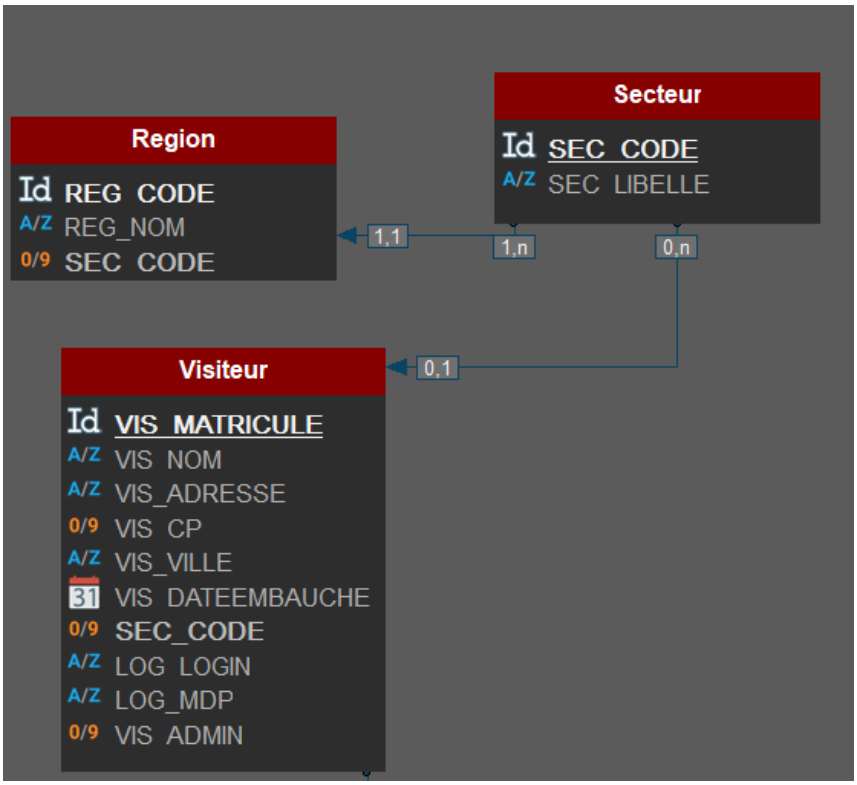
1. COTÉ LOGIN



Pour la première partie de la nouvelle base données, il fallait commencer pour la conception de la nouvelle version de la Table Visiteurs, sur l'ancienne version, on peut voir que la Table Visiteurs est liée par 2 autres tables : Secteur et Région (Les plus importants pour la nouvelle version, V1).

Sur cette partie du MCD, c'est la partie Login.

Où on va ajouter des champs : Login et Mot De Passe.

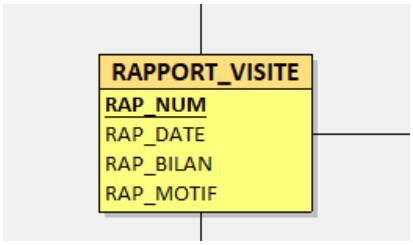


Dans la nouvelle version que j'ai créée sur **Windev**, un éditeur de Logiciel, j'ai retiré certaines tables qui nous semblaient inutiles.

Dans la V1, j'ai modifié la table visiteur et ajouté trois champs :
Login-> pour les pseudos des visiteurs,
MotDePasse-> Les mots de passe qui seront cryptés par la suite,
Admin-> Pour permettre de différencier nos Visiteurs et nos Responsable de Secteur.

2. COTÉ RAPPORTS

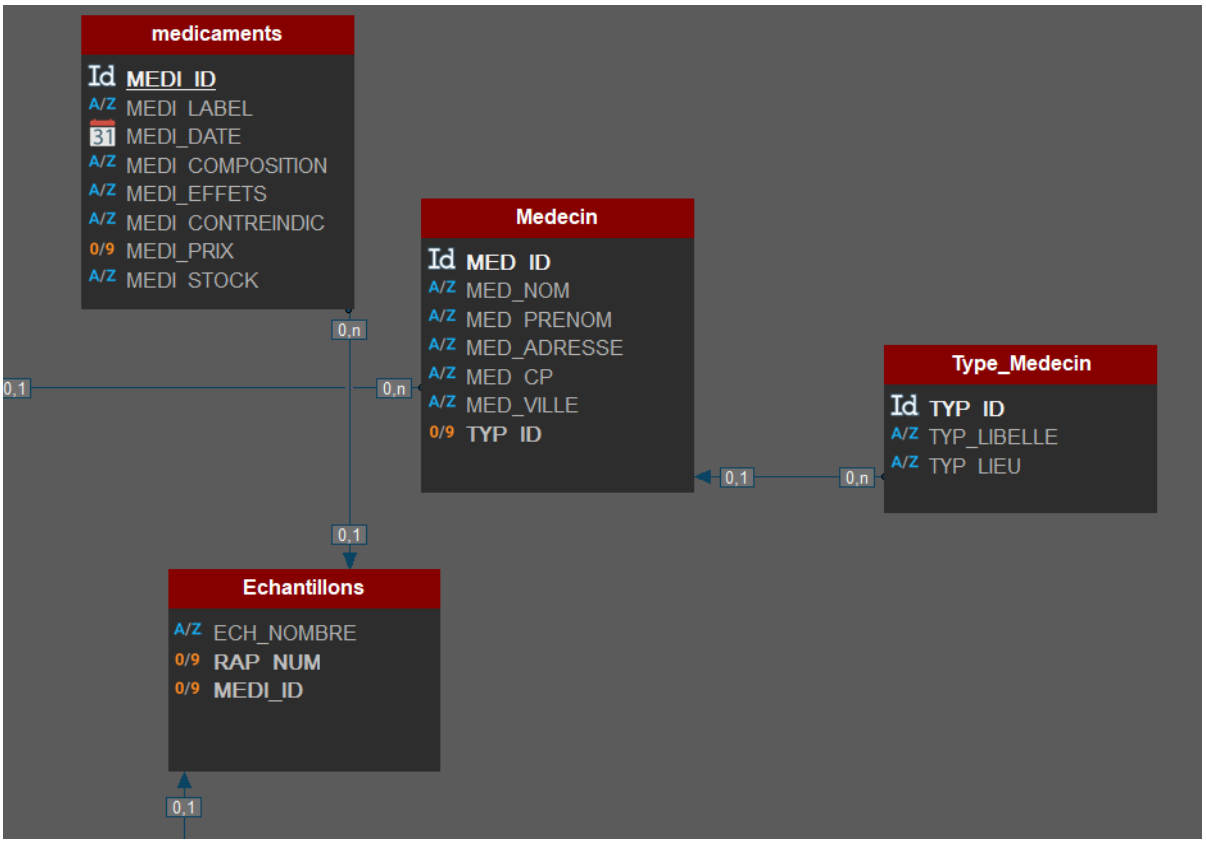
Voici la table existante Rapports Visite, dans la nouvelle table j'ai ajouté le champ Commentaire, si on prend en compte sur les autres ne sont que des listes déroulantes, il faut une section où le visiteur puisse écrire dans son compte-rendu.



Voici la V1 de la table, déjà liée à la table Visiteur. Il y a également l'ID Médicaments qui sera expliqué sous peu.

3. COTÉ MÉDICAMENTS

Sur cette partie de la BDD, il fallait repartir de zéro. D'un côté se trouve la table Médecin, où vont se répartir les médecins attitrés,et la table médicaments où on regroupe les médicaments qu'un visiteur va devoir présenter durant ses rendez-vous. J'ai ajouté la table Échantillons qui nous permet de récupérer l'ID du médicaments et l'ID Du Rapport de Visite.



4. EXPORTER SUR PHPMYADMIN

Lorsque la base de données était enfin terminée sur Windev, j'ai pu l'importer sur notre serveur commun afin que le reste de l'équipe puisse l'utiliser à leurs tours. L'intégralité de la base de données

Depuis Windev j'ai demandé d'exporter ma BDD en .sql, j'ai ainsi pu retranscrire l'intégralité de ma base. Par la suite j'ai dû modifier quelques détails pour que la base soit fonctionnelle, il y avait quelques options de Windev que SQL ne connaissait pas, j'ai dû revenir sur le code afin de modifier les clés primaires.

Table	Action	Lignes	Type	Interclassement	Taille	Perte
Echantillons	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_general_ci	48,0 kio	-
Medecin	Parcourir Structure Rechercher Insérer Vider Supprimer	10	InnoDB	utf8mb4_general_ci	32,0 kio	-
medicaments	Parcourir Structure Rechercher Insérer Vider Supprimer	196	InnoDB	utf8mb4_general_ci	80,0 kio	-
Rapport_Visite	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_general_ci	48,0 kio	-
Region	Parcourir Structure Rechercher Insérer Vider Supprimer	2	InnoDB	utf8mb4_general_ci	16,0 kio	-
Secteur	Parcourir Structure Rechercher Insérer Vider Supprimer	5	InnoDB	utf8mb4_general_ci	32,0 kio	-
Type_Medecin	Parcourir Structure Rechercher Insérer Vider Supprimer	4	InnoDB	utf8mb4_general_ci	16,0 kio	-
Visiteur	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_general_ci	32,0 kio	-
8 tables	Somme	217	InnoDB	utf8mb4_general_ci	304,0 kio	0 o

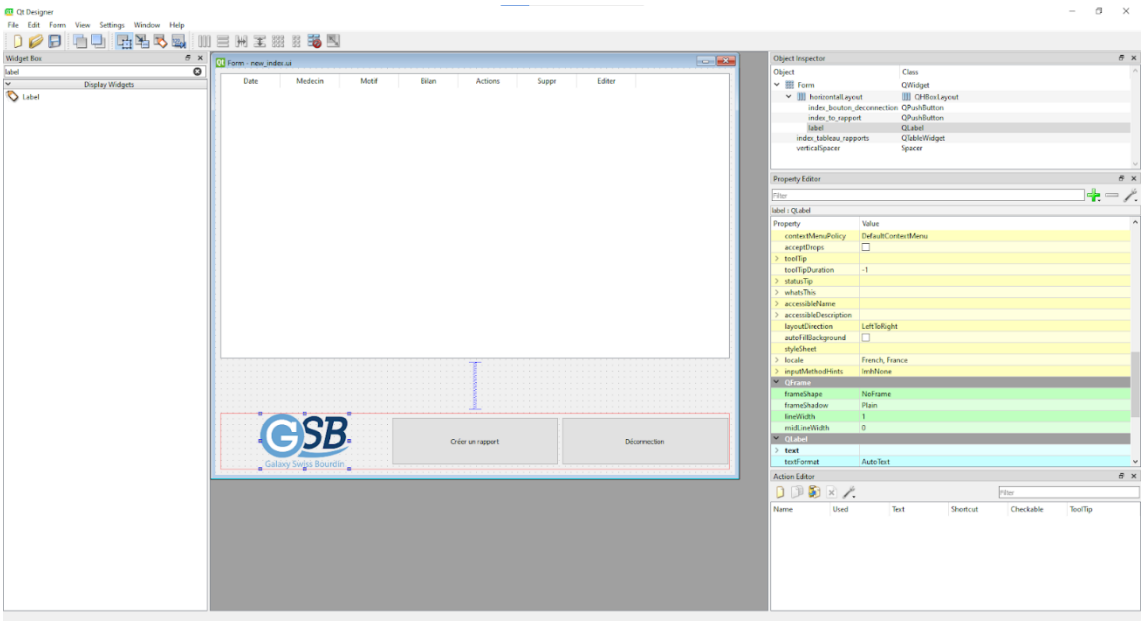
La base est maintenant apte à être utilisée et testée au reste de l'équipe.

A. FRONT

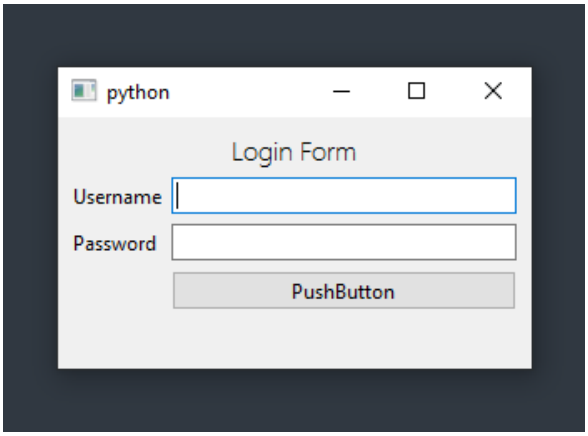
1. PYQT

Pour la mise en place du front pour l'application, nous utilisons le langage de programmation Python, avec la Bibliothèque PyQt et Requests. Ces bibliothèques Python sont faciles d'utilisation et permettent de faire l'interface graphique de notre software en liant la partie front avec l'API grâce à l'utilisation de la bibliothèque Requests.

Pour utiliser PyQt, nous avons choisi la version 6 avec Python 3.11. Afin de nous faire gagner du temps dans la création du client lourd de GSB, nous utilisons QT Designer, un logiciel permettant de faire une ou plusieurs fenêtres avec le module PyQt à sur une interface à l'aide de widgets et modules.



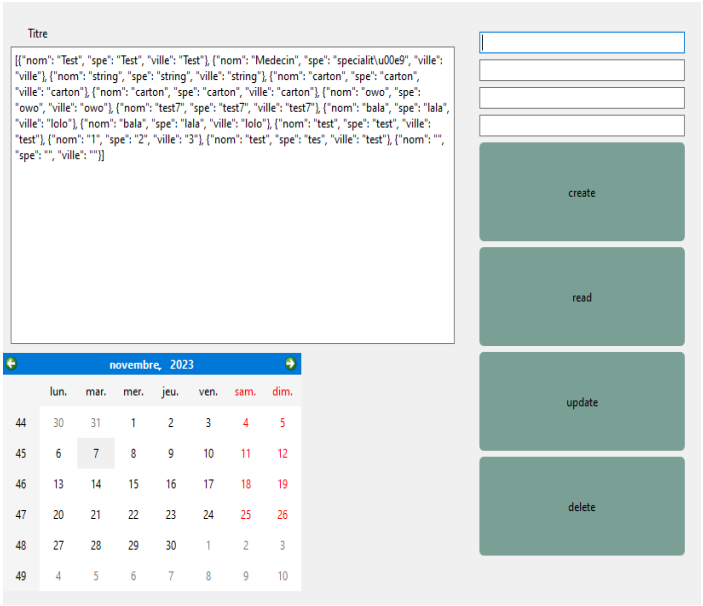
Afin de prendre en main le module PyQt nous avons commencé par créer une interface avec un CRUD (Create Read Update Delete) et un formulaire de connexion pour accéder à l'application.



Création de la page d'authentification dans un premier temps. Une fois la fenêtre créer et les widgets en place, nous nous sommes occupés du fonctionnement de la page de connexion au client lourd.

```
55  class Screen2(QMainWindow):
56      """docstring for Screen2"""
57      def __init__(self, login):
58          super().__init__()
59          super(Screen2, self).__init__()
60          loadUi("main.ui", self)
61          self.set_list()
62          self.index_button_create.clicked.connect(self.createDr)
63          self.index_button_update.clicked.connect(self.updateDr)
64          self.index_button_delete.clicked.connect(self.deleteDr)
65          self.index_button_read.clicked.connect(self.gotoCpRendu)
66
67      def set_list(self): ...
73
74      def gotoCpRendu(self): ...
76
77
78      def createDr(self): ...
90
91
92      def updateDr(self): ...
103
104     def deleteDr(self): ...
108
109
```

```
12  class Window(QWidget):
13      def __init__(self):
14          loadUi("login.ui", self)
15          button1.clicked.connect(self.login)
16
17      def gotoscreen2(self):
18          """
19          Change la fenêtre à afficher
20
21          Affiche la fenêtre suivante
22          """
23          widget.setCurrentIndex(widget.currentIndex()+1)
24
25      def login(self):
26          """
27          Envoie une demande d'authentification à l'API
28
29          - Récupére les informations rensigné dans les champs du formulaire
30          - Envoie une requête à l'API avec les informations renseignées en paramétre
31          - Si le code Http retour est "200" alors :  apelle de la méthode permettant le changememnt de fenêtre
32            Sinon renvoie un message d'erreurS
33          """
34          login    = self.input1.text()
35          mdp      = self.input2.text()
36          logedin = False
37
38          x = requests.post(f'http://127.0.0.1:8000/login',
39          data={
40              "grant_type":"",
41              "username":login,
42              "password":mdp,
43              "scope":"",
44              "client_id":"",
45              "client_secret":""
46              })
47
48          if x.status_code==200:
49              logedin = True
50              self.gotoscreen2()
51          else:
52              print('Username or Password are wrong !')
53
```



Création de la fenêtre avec un CRUD.

Un widget permet de d'afficher ce que contient la base de données, 3 champs permettent à l'utilisateur d'entrer des informations et les boutons en dessous permettent d'effectuer une certaine action du CRUD sur la DB avec les informations renseignées.

```
55  class Screen2(QMainWindow):
56      def __init__(self, login): ...
65      def set_list(self): ...
71          """
72          Renvoie la liste des médecins de la base de donnée
73          """
74      def gotoCpRendu(self): ...
76          """
77          Change la fenêtre à afficher
78          """
79      def createDr(self): ...
89          """
90          Envoie les informations renseigné afin d'ajouter un médecin dans la base de donné
91          """
92      def updateDr(self): ...
103         """
104         Modifie les informations d'un médecin désigné dans la base de donné
105         """
106     def deleteDr(self): ...
110         """
111         Supprime un médecin de la base de donnée
112         """
```

```python
class Screen2(QMainWindow):
    def __init__(self, login):
        super().__init__()
        super(Screen2, self).__init__()
        loadUi("main.ui", self)
        self.set_list()
        self.index_button_create.clicked.connect(self.createDr)
        self.index_button_update.clicked.connect(self.updateDr)
        self.index_button_delete.clicked.connect(self.deleteDr)
        self.index_button_read.clicked.connect(self.gotoCpRendu)
    def set_list(self):
        x = requests.get('http://127.0.0.1:8000/medecins')
        jason = x.json()
        fake = json.dumps(jason)
        self.List.clear()
        self.List.setText(fake)
    def gotoCpRendu(self):
        widget.setCurrentIndex(widget.currentIndex()+1)
    def createDr(self):
        nom    = self.dr_nom.text()
        spe    = self.dr_spe.text()
        ville = self.dr_ville.text()
        x = requests.post(f'http://127.0.0.1:8000/create_medecin', json={
            "nom":nom,
            "spe":spe,
            "ville":ville
            })
        self.set_list()
    def updateDr(self):
        id_dr = self.dr_id.text()
        nom    = self.dr_nom.text()
        spe    = self.dr_spe.text()
        ville = self.dr_ville.text()
        x = requests.put(f'http://127.0.0.1:8000/update_medecin/{id_dr}', json={
            "nom":nom,
            "spe":spe,
            "ville":ville
            })
        self.set_list()
    def deleteDr(self):
        id_dr = self.dr_id.text()
        x = requests.delete(f'http://127.0.0.1:8000/delete_medecin/{id_dr}')
        self.set_list()

```

Une fois le CRUD réalisé nous avons changé la structure du code de l'application.

```python
widget = QtWidgets.QStackedWidget()
login = Window()
menu = Screen2(login)
CpRendu = CpRendu()
widget.addWidget(login)
widget.addWidget(menu)
widget.addWidget(CpRendu)
```

```python
def gotoscreen2(self):
    """
    Change la fenêtre à afficher

    Affiche la fenêtre suivante
    """
    widget.setCurrentIndex(widget.currentIndex()+1)

def login(self): ...
```

Avant la modification, l'application fonctionnait comme une liste dans laquelle on se déplaçait de cette façon :

```
300  class Rapport_page(QtWidgets.QWidget):
301      def __init__(self): ...
310
311      def setRapportResume(self): ...
317
318
319      def setMedecins(self): ...
332
333      def setMedicaments(self): ...
342
343
344      def doSomethingNext(self): ...
349
350      def goToIndex(self): ...
355
356      def sendRapport(self): ...
402
403  class Stack(QtWidgets.QStackedWidget):
404      def __init__(self): ...
416
417      def deconnection(self): ...
426
427      def initCurrent(self): ...
430
431      def launchIndex(self,access_token,id_user): ...
444
445
446  if __name__ == "__main__":
447
448
449      today = date.today()
450      todayFr = today.strftime("%Y-%m-%d")
451      app = QApplication(sys.argv)
452      appStack = Stack()
453      appStack.show()
454
455      sys.exit(app.exec())
```

```
16   class User():
17       def __init__(self, access_token, user_id): ...
23
24       def getUserDatas(self): ...
32
33   class Login_page(QtWidgets.QWidget):
34       def __init__(self): ...
38
39       def login(self): ...
69
70       def doSomethingNext(self): ...
77
78   class Index_page(QtWidgets.QWidget):
79       def __init__(self): ...
89
90       def goToAdmin(self): ...
95
96
97       def doSomethingNext(self): ...
104
105      def setUserDatas(self): ...
115
116
117      def setRapportList(self): ...
160
161      def editRapport(self,RAP_NUM): ...
175
176      def goToRapport(self): ...
181
182      def suppr(self,id_RAP): ...
190
191  class Admin_page(QtWidgets.QWidget):
192      def __init__(self): ...
196
197      def goToIndex(self): ...
202
203      def doSomethingNext(self): ...
226
227      def setVisRapports(self,id_vis): ...
269
270      def suppr(self,id_RAP): ...
278
279      def goToRapport(self): ...
284
285      def editRapport(self,RAP_NUM): ...
299
```

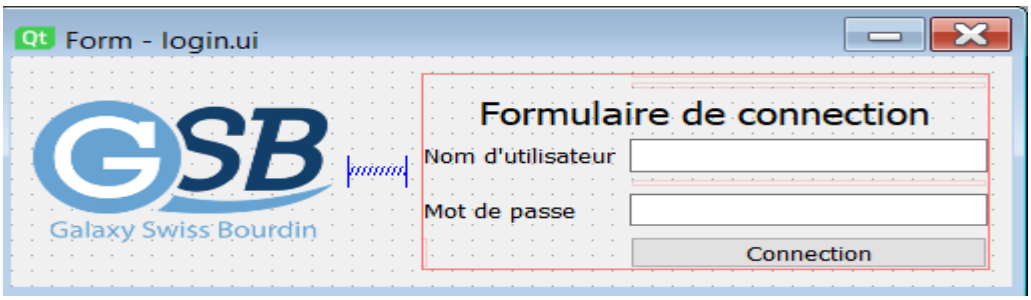
```python
    def launchIndex(self,access_token,id_user):
        """
        Initie les pages de l'application
        """
        self.user = User(access_token,id_user)
        self.admin_page = Admin_page()
        self.index_page = Index_page()
        self.rapport_page = Rapport_page()
        self.addWidget(self.admin_page)
        self.addWidget(self.index_page)
        self.addWidget(self.rapport_page)
        self.setCurrentWidget(self.index_page)

```

De cette façon, la class Stack peut être représenté comme un liste associative, désormais afin de changer de fenêtre il faut que la faut ajouter la class de la fenêtre comme attribut de la class “ Stack “ et appeler la fenêtre à l’aide de la méthode “ setCurrentWidget() ‘.

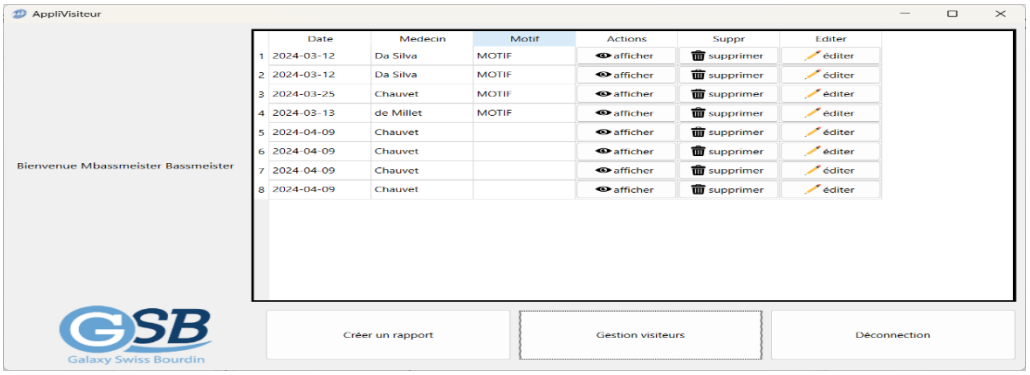
```python
    def goToAdmin(self):
        """
        Affiche la fenêtre "Admin"
        """
        appStack.setCurrentWidget(appStack.admin_page)

```

Liste des pages de l’application :

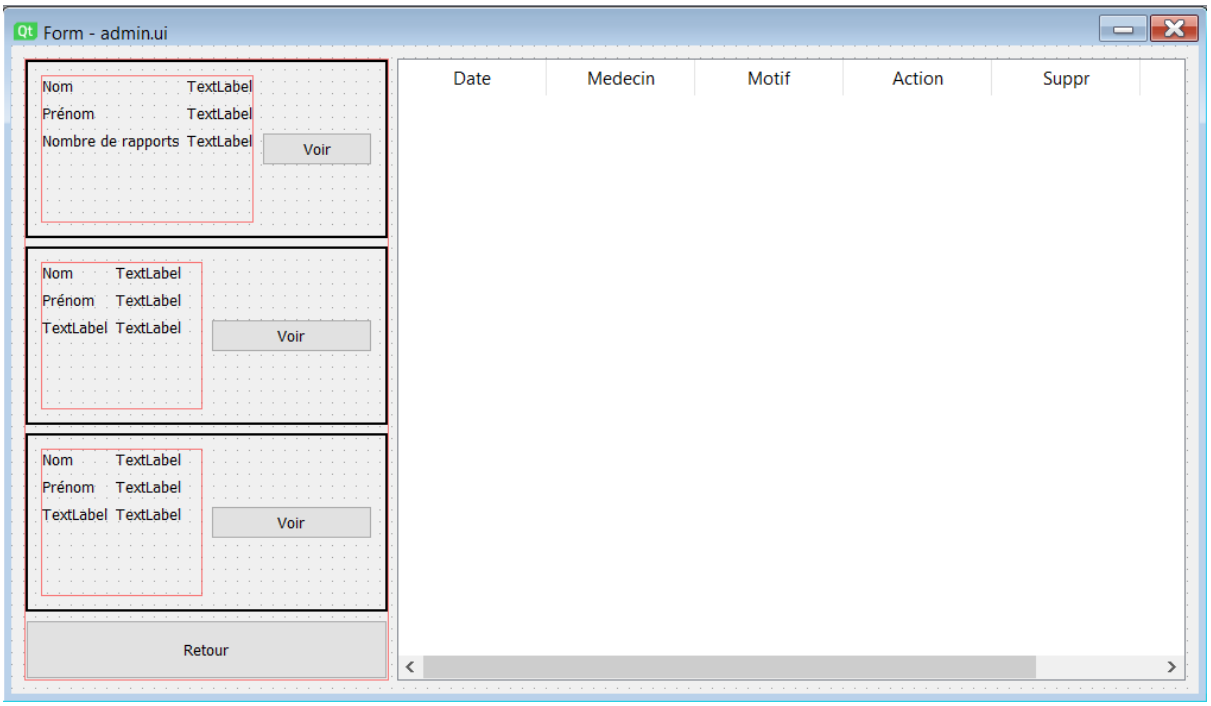
Page d’authentification :



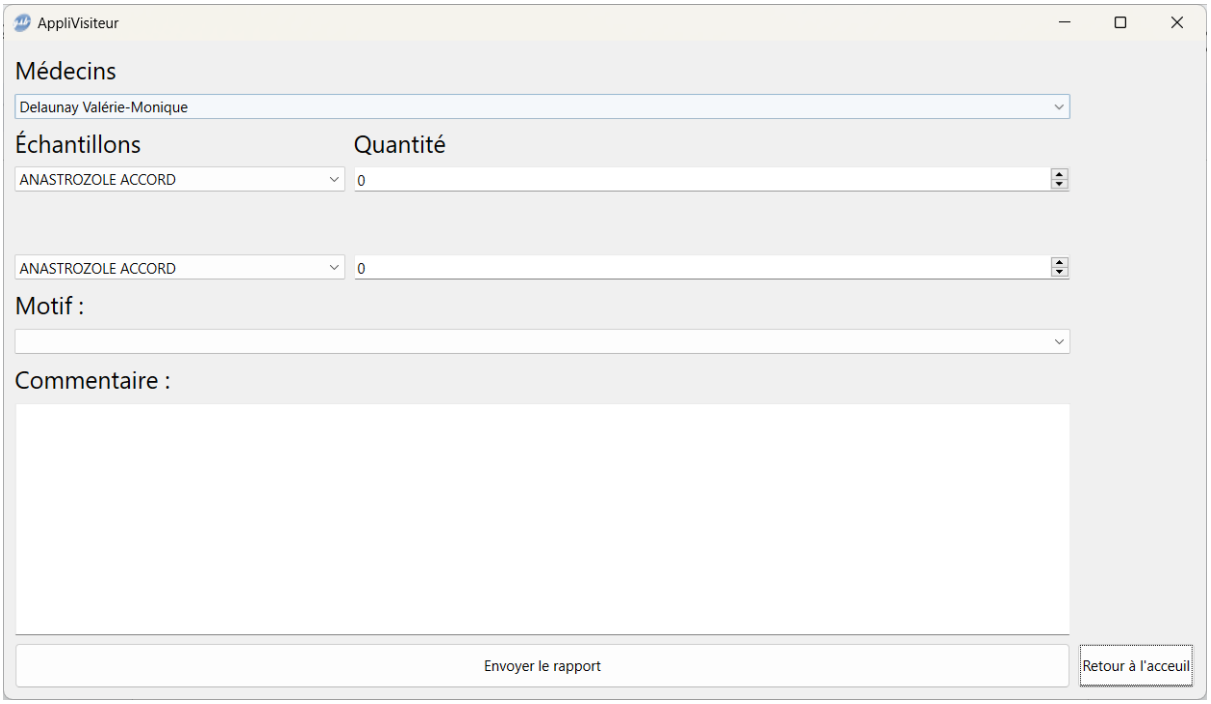
Page d’acceuil :



Page de gestion des visiteurs (pour les chefs de secteur) :



Page d'insertion de rapport :



2. PAGE D'ACCUEIL

Un tableau sur la page d'accueil de l'application permet de visualiser et de gérer les rapports insérés par l'utilisateur connecté :

	Date	Medecin	Motif	Actions	Suppr	Editer
1	2024-03-12	Da Silva	MOTIF	afficher	supprimer	éditer
2	2024-03-12	Da Silva	MOTIF	afficher	supprimer	éditer
3	2024-03-25	Chauvet	MOTIF	afficher	supprimer	éditer
4	2024-03-13	de Millet	MOTIF	afficher	supprimer	éditer
5	2024-04-09	Chauvet		afficher	supprimer	éditer
6	2024-04-09	Chauvet		afficher	supprimer	éditer
7	2024-04-09	Chauvet		afficher	supprimer	éditer
8	2024-04-09	Chauvet		afficher	supprimer	éditer

On peut y voir certaines informations du rapport qui permettent d'identifier précisément un rapport et trois boutons sont présent sur chaque ligne :

- Supprimer :

  Permet de supprimer le rapport de la base de données

- Editer :

  Permet de modifier les informations du rapport

- Afficher :

  Permet d'afficher les données du rapport en pdf et d'enregistrer le document.

GSB Galaxy Swiss Bourdin

Fiche de présentation rapport 9

Praticien : Dr. Chauvet Michelle

Médicament(s) :

TRAMADOL CRISTERS

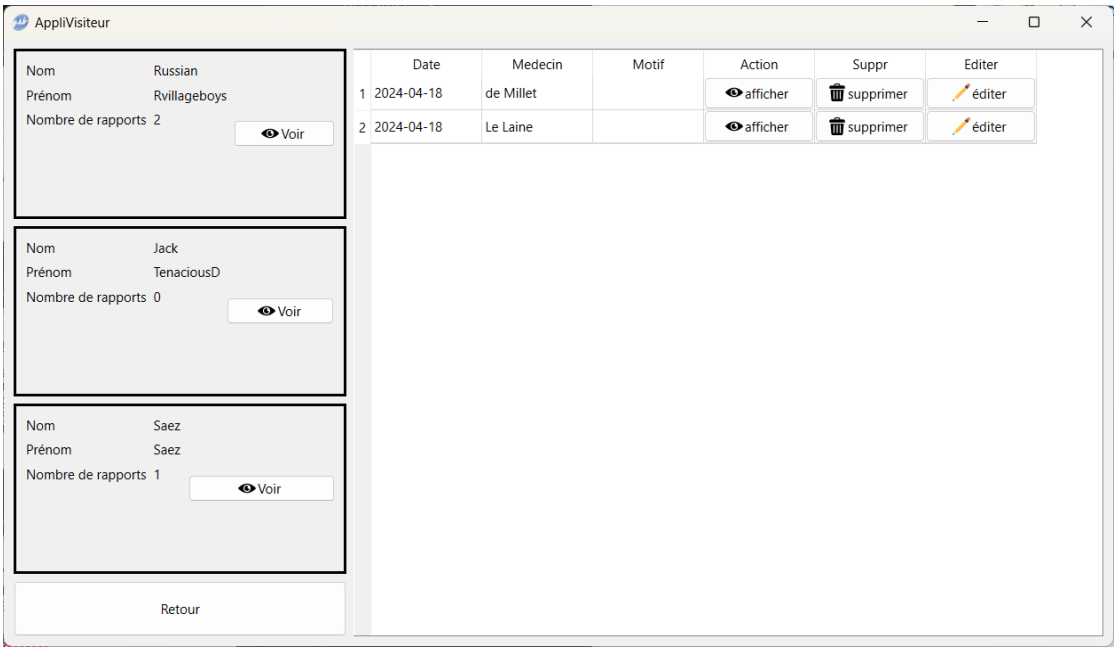
Quantité : 50
Composition : MACROGOL 3350
Effet(s) : Nisi illum inventore aspernatur velit. Molestiae in inventore
Contre-indication(s) : liste I
Prix : 10

TRAMADOL CRISTERS

Quantité : 20
Composition : MACROGOL 3350
Effet(s) : Nisi illum inventore aspernatur velit. Molestiae in inventore
Contre-indication(s) : liste I
Prix : 10

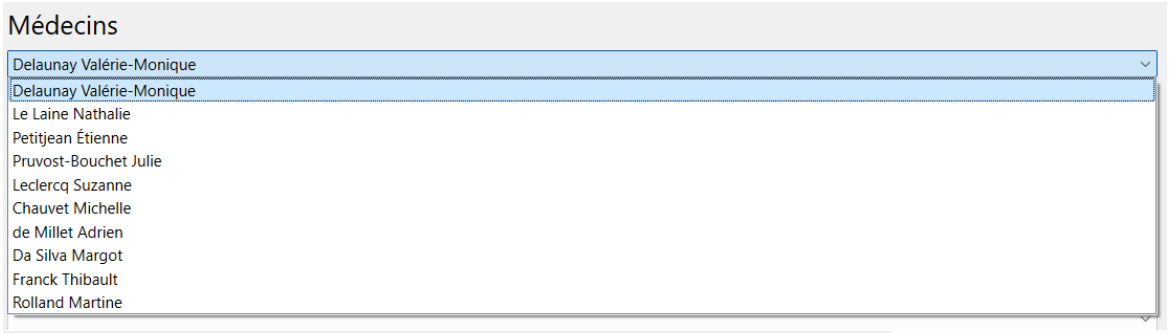
2024 GSB Applivsiteur

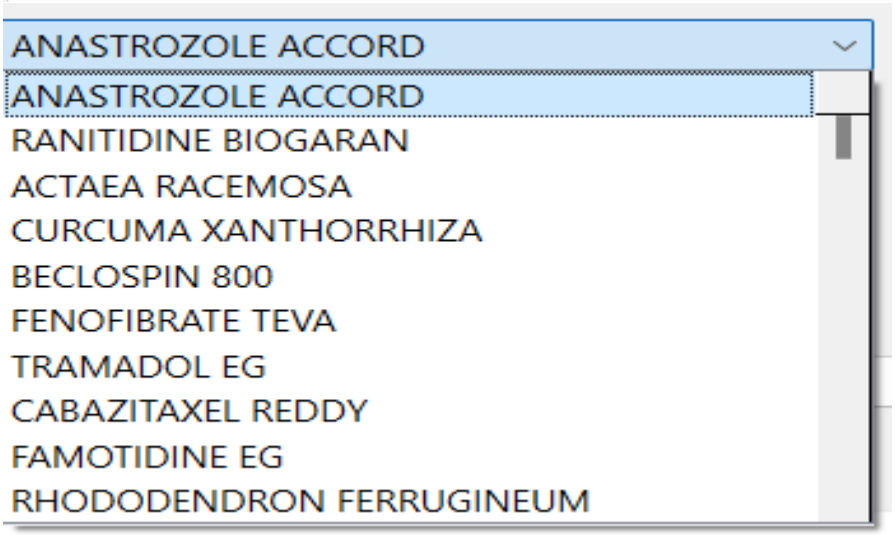
3. PAGE DE GESTION DES VISITEURS



Cette page permet aux chefs de secteur de pouvoir gérer et visualiser les rapports renseignés par les 3 visiteurs qui lui sont attribués.
On peut y voir le nom, prénom et le nombre de rapports des visiteurs, cliquer le sur bouton “voir” permet d’afficher les rapports du visiteur dans le tableau ci-dessus.

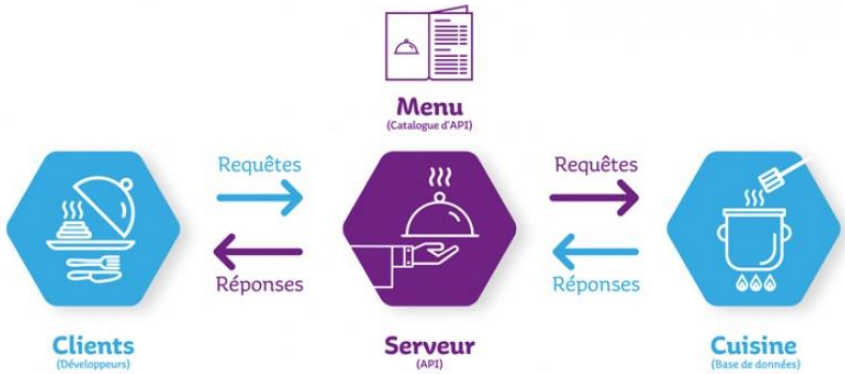
4. PAGE D’INSERTION DE RAPPORT





B. API

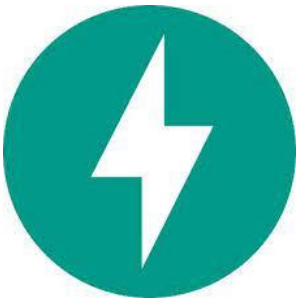
1. Qu'est-ce qu'une API ?



Une API (application programming interface ou « interface de programmation d'application ») est une interface logicielle qui permet de « connecter » un logiciel ou un service à un autre logiciel ou service afin d'échanger des données et des fonctionnalités.

2. API

Pour faire le plein de connaissances sur ce nouveau framework que je ne connaissais pas qu'est FastAPI, j'ai suivi ce tutoriel sur Youtube qui comprenait tous les savoirs de base pour que je puisse développer cet API :



J'utilise des "models" et des "schémas" :

```python
class Medecin(Base):
    __tablename__ = "Medecin"

    MED_ID = Column(Integer, primary_key=True, index=True)
    MED_NOM = Column(String)
    MED_PRENOM = Column(String)
    MED_ADRESSE = Column(String)
    MED_CP = Column(String)
    MED_VILLE = Column(String)
    TYP_ID = Column(String, ForeignKey('Type_Medecin.TYP_ID'))

    type = relationship('Type_Medecin', back_populates="medecin")
    rapport_med = relationship('Rapport_Visite', back_populates="affiliate_med")
```

Les models sont le reflet des tables dans la base de données, ce sont des objets que je peux utiliser pour me servir de la base de données.

```python
class showMedecin(BaseModel):
    MED_ID : int
    MED_NOM: str
    MED_PRENOM: str
    MED_ADRESSE: str
    MED_CP: str
    MED_VILLE: str
```

Les schémas sont des "moules" qui vont permettent d'indiquer à la route ce qu'elle doit recevoir/renvoyer.

Les images ci-dessus sont les exemples du "model" Medecin, qui représente donc la table "Medecins" de la base de données ainsi que le schéma "showMedecin" qui est utilisée dans les routes "/medecins" et "/medecin/id".

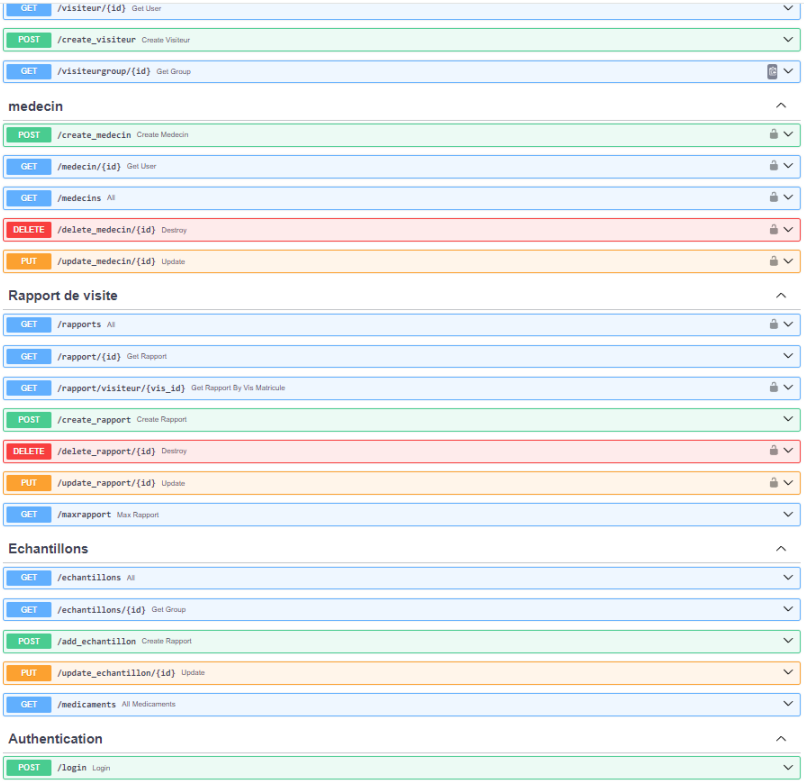
Le schéma indique donc à l'API ce qu'elle doit retourner.

3. LA PREMIÈRE ÉTAPE : L'AUTHENTIFICATION

On a besoin d'un login et d'un mot de passe qui proviennent de la table Visiteur dans les champs LOG_LOGIN et LOG_MDP si la combinaison du login et du mot de passe haché correspond à un enregistrement dans la base de données alors la route qui est "/login" nous renvoie un token qui est de type JWT (JSON Web Token).

Ce token permet d'être identifié par l'API et donc de pouvoir envoyer des requêtes et reçevoir des réponses. L'API est sécurisée par ce token c'est à dire que si un utilisateur essaie d'utiliser les routes de l'api sans être authentifier et en possession du token, l'API n'est pas accessible.

Ensuite, j'ai créé toutes les autres routes



En suivant le modèle CRUD (Create, Read, Update, Delete) pour la plupart des tables.

C. TEST UNITAIRE

```python
import pytest
from fastapi.testclient import TestClient
from .main import app
from . import models, schemas

client = TestClient(app)

# Test pour vérifier si tous les rapports sont récupérés avec succès
def test_get_all_rapports():

    # Vous pouvez vous authentifier en envoyant un nom d'utilisateur et un mot de passe valide pour obtenir le toke
    login_response = client.post("/login", data={"username": "demo", "password": "        "})
    print(login_response)
    token = login_response.json()[0]["access_token"]

    # Utilisez ce token dans l'en-tête Authorization pour les requêtes suivantes
    response = client.get("/rapports", headers={"Authorization": f"Bearer {token}"})
    assert response.status_code == 200
    assert len(response.json()) > 0

# Test pour vérifier si un rapport spécifique est récupéré avec succès
def test_get_rapport_by_id():
    login_response = client.post("/login", data={"username": "demo", "password": "password"})
    print(login_response)
    token = login_response.json()[0]["access_token"]
    # Supposons que l'ID 1 existe dans la base de données
    response = client.get("/rapport/1",headers={"Authorization": f"Bearer {token}"})
    assert response.status_code == 200

# Test pour vérifier la création d'un nouveau rapport
def test_create_rapport():
    login_response = client.post("/login", data={"username": "demo", "password": "password"})
```

Qu'est-ce qu'un test unitaire ?

Le test unitaire est une méthode de test de logiciels qui consiste à tester des éléments ou des unités d'un logiciel. L'objectif du test unitaire est de valider que chaque unité du logiciel fonctionne comme prévu.