ÉCOLE NATIONALE SUPÉRIEURE D'ARTS ET MÉTIERS
UNIVERSITÉ HASSAN II DE CASABLANCA

RAPPORT DE PROJET

Module : Python Avance

Realise par : Hafsa AAYACH – Lena GHALAB – CATARYA Ahlin Junior Kévin

Niveau : 1ere annee cycle d'ingenieur

Filiere : IAGI - Intelligence Artificielle et Génie Informatique

Encadré par : Le professeur MUSTAPHA HAIN

Création de la base de données avec Python

Dans le cadre de ce projet, nous avons conçu une base de données destinée à la gestion d'une pharmacie.
Cette base est créée et initialisée automatiquement à l'aide d'un script écrit en **Python**, en utilisant le module intégré **sqlite3** qui permet de manipuler des bases de données **SQLite** sans nécessiter de serveur externe.

1.Connexion à la base de données

Le programme débute par l'importation du module `sqlite3` et la création ou l'ouverture d'un fichier de base de données nommé « **pharmacie.db** »

```
import sqlite3
db=sqlite3.connect("pharmacie.db")
```

Cette commande établit une connexion avec la base de données.
Si le fichier n'existe pas encore, il est automatiquement créé dans le répertoire du programme.

2.Fonction de création des tables

La fonction `create_table(conn)` a pour rôle de définir la structure de la base de données.
Elle reçoit comme paramètre une connexion SQLite et crée plusieurs tables nécessaires au fonctionnement du système.

```
def create_table(conn: sqlite3.Connection):  1 usage
    cur = conn.cursor()
    cur.execute("PRAGMA foreign_keys = ON;")
```

L'instruction **`PRAGMA foreign_keys = ON`** permet d'activer la gestion des clés étrangères, afin d'assurer la cohérence des relations entre les tables.

3.Table des medicaments

```
    #Table des médicaments
    cur.execute("""
    CREATE TABLE IF NOT EXISTS medicaments (
        id INTEGER PRIMARY KEY,
        nom TEXT NOT NULL,
        code_barre TEXT UNIQUE,
        description TEXT,
        quantite INTEGER NOT NULL DEFAULT 0 CHECK(quantite >= 0),
        prix REAL NOT NULL CHECK(prix >= 0),
        date_expiration DATE,
        date_creation TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        mise_a_jour TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        );
    """)
```

Cette table contient les informations sur les différents médicaments disponibles dans la pharmacie.
Les principaux champs sont :

- **id** : identifiant unique du médicament (clé primaire)
- **nom** : nom du médicament
- **code_barre** : code-barres unique pour identifier le produit
- **quantite** : quantité en stock (ne peut pas être négative grâce à la contrainte `CHECK`)
- **prix** : prix du médicament (doit être positif)
- **date_expiration** : date de péremption du produit
- **date_creation** et **mise_a_jour** : dates automatiques de création et de dernière modification.

4.Table des clients

```
#Table des clients
cur.execute("""
CREATE TABLE IF NOT EXISTS clients(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nom TEXT NOT NULL,
    prenom TEXT NOT NULL,
    naissance DATE NOT NULL,
    phone TEXT,
    num_assurance TEXT,
    date_creation TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    mise_a_jour TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
""")
```

Cette table regroupe les informations personnelles des clients de la pharmacie.
Elle contient notamment leur **nom**, **prénom**, **date de naissance**, **numéro de téléphone**, et **numéro d'assurance**.
Chaque client est identifié par un **id** généré automatiquement.

5.Table des ventes

```
#Table des ventes
cur.execute("""
CREATE TABLE IF NOT EXISTS vente(
id INTEGER PRIMARY KEY AUTOINCREMENT,
id_medicament INTEGER NOT NULL,
id_client INTEGER,
quantite INTEGER NOT NULL CHECK(quantite > 0),
prix_unitaire REAL NOT NULL CHECK(prix_unitaire > 0),
prix_total REAL NOT NULL CHECK(prix_total > 0),
date_vente TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
pharmacien TEXT,
FOREIGN KEY (id_medicament) REFERENCES medicaments(id) ON DELETE RESTRICT ON UPDATE CASCADE,
FOREIGN KEY (id_client) REFERENCES clients(id) ON DELETE SET NULL ON UPDATE CASCADE
);
""")
```

Cette table enregistre les **transactions de vente** effectuées dans la pharmacie.
Chaque vente est associée à un médicament et éventuellement à un client.

Les clés étrangères garantissent la cohérence des données :

- Si un **médicament** est supprimé, la suppression de la vente est **interdite** (`ON DELETE RESTRICT`),
- Si un **client** est supprimé, le champ correspondant devient **NULL** (`ON DELETE SET NULL`).

Les contraintes `CHECK` assurent que les valeurs numériques (prix et quantité) soient toujours positives.

7.Mise à jour automatique des enregistrements

Deux **déclencheurs (triggers)** ont été créés afin de mettre à jour automatiquement la colonne `mise_a_jour` dès qu'une modification est effectuée sur les tables **médicaments** ou **clients** :

```
#cette partie concerne la mise a jour
cur.execute("""
    CREATE TRIGGER IF NOT EXISTS trg_medicaments_updated_at
    AFTER UPDATE ON medicaments
    FOR EACH ROW
    BEGIN
        UPDATE medicaments SET mise_a_jour = CURRENT_TIMESTAMP WHERE id = OLD.id;
    END;
    """)
```

Ainsi, la date de dernière modification est automatiquement enregistrée sans intervention manuelle.

- ## Validation des changements

Pour finaliser la création des tables et enregistrer toutes les modifications dans le fichier de base de données, la commande suivante est utilisée :

```
conn.commit()
```

Sans cette instruction, les tables ne seraient pas effectivement enregistrées dans le fichier **pharmacie.db**.

- ## Conclusion

Ce script permet d'automatiser entièrement la création d'une base de données relationnelle pour la gestion d'une pharmacie.
Grâce à l'utilisation de **clés étrangères**, de **contraintes d'intégrité**, et de **triggers**, la base de données garantit une cohérence et une fiabilité optimales des informations.
Cette structure servira ensuite de fondation pour le développement d'une application complète de gestion pharmaceutique (ajout, vente et suivi des stocks).

Gestion des Fonctions Python pour la Base de Données

1. Introduction

Dans le cadre de ce projet de gestion d'une pharmacie, mon rôle a été de concevoir et implémenter les fonctions Python permettant de manipuler la base de données. Ces fonctions assurent les opérations de base telles que l'ajout, la modification, la suppression et l'affichage des données relatives aux médicaments, clients et ventes. L'objectif est de fournir une logique fiable et sécurisée, prête à être utilisée par l'interface graphique.

2. Description des Fonctions Développées

Voici les principales fonctions qu'on a implémentées, organisées par catégorie.

2.1 Gestion des Médicaments

- **ajouter_medicament**(nom, code_barre, description, quantite, prix, date_expiration)

- Cette fonction permet d'ajouter un nouveau médicament dans la table `medicaments`. Elle vérifie les données entrées par l'utilisateur et empêche les doublons en utilisant la clé `code_barre`, qui est unique dans la base de données.

- `cursor.execute()` exécute la requête SQL, `try...except` attrape l'erreur `sqlite3.IntegrityError` en cas de doublon. Utilisation des requêtes paramétrées pour eviter les injections SQL.
Le bloc try est un bloc protégé contre les erreurs, le code à l'intérieur sera surveillé pour détecter les erreurs

- Extrait de code :

```
try:
    cursor.execute("""
        INSERT INTO medicaments (nom, code_barre, description, quantite, prix, date_expiration)
        VALUES (?, ?, ?, ?, ?, ?)
    """, (nom, code_barre, description, quantite, prix, date_expiration))
```

- **modifier_medicament**(id, nom=None, quantite=None, prix=None)

- Cette fonction met à jour un médicament existant en fonction de son `id`. Seuls les paramètres non nuls sont pris en compte, ce qui permet de modifier un seul champ sans avoir à tout redonner.

- `cursor.execute()` exécute une requête `UPDATE`, la construction dynamique de la requête SQL est faite avec une boucle sur les paramètres.

- Extrait de code :

```
updates = []
params = []
if nom is not None:
    updates.append("nom = ?")
    params.append(nom)
```

- **supprimer_medicament(id)**

Cette fonction supprime un médicament par son id. La suppression est bloquée si des ventes sont liées à ce médicament, grâce à la contrainte RESTRICT sur la clé étrangère dans la base de données.

cursor.execute() exécute la commande DELETE, gestion d'erreur sqlite3.IntegrityError.

2.2 Gestion des Clients

- **ajouter_client**(nom, prenom, naissance, phone, num_assurance)

Cette fonction permet d'ajouter un nouveau client dans la table clients. Elle vérifie que les champs obligatoires sont remplis avant d'insérer les données dans la base.

cursor.execute() exécute la requête SQL, try...except gère les erreurs SQL.

Extrait de code :

```
try:
    cursor.execute("""
        INSERT INTO clients (nom, prenom, naissance, phone, num_assurance)
        VALUES (?, ?, ?, ?, ?)
    """, (nom, prenom, naissance, phone, num_assurance))
```

- **modifier_client**(id, nom=None, prenom=None, phone=None)

Cette fonction met à jour les informations d'un client existant. Comme pour la modification d'un médicament, seuls les champs non nuls sont mis à jour.

cursor.execute() exécute une requête UPDATE, la requête est construite dynamiquement.

- **supprimer_client(id)**

Cette fonction supprime un client par son id. Les ventes liées auront id_client = NULL, car la clé étrangère dans la table vente est définie avec SET NULL.

cursor.execute() exécute la commande DELETE.

2.3 Gestion des Ventes

- **enregistrer_vente**(id_medicament, id_client, quantite, pharmacien)

Cette fonction enregistre une vente, vérifie le stock, et met à jour la quantité du médicament. Elle récupère d'abord le prix du médicament, calcule le total, enregistre la vente, puis met à jour le stock. Elle empêche la vente si le stock est insuffisant.

cursor.execute() exécute des requêtes INSERT et UPDATE, try...except gère les erreurs (stock insuffisant, ID incorrect, etc.).

-Extrait de code :

```
cursor.execute("""
    INSERT INTO vente (id_medicament, id_client, quantite, prix_unitaire, prix_total, pharmacien)
    VALUES (?, ?, ?, ?, ?, ?)
""", (id_medicament, id_client, quantite, prix_unitaire, prix_total, pharmacien))
```

- **Afficher_ventes()**

Cette fonction récupère toutes les ventes enregistrées dans la table vente et les affiche de manière lisible, en incluant les noms des clients et des médicaments grâce à des jointures avec les tables clients et medicaments.

cursor.execute() exécute une requête SQL complexe avec des jointures (LEFT JOIN) pour combiner les données de plusieurs tables. La fonction cursor.fetchall() récupère tous les résultats de la requête.

Extrait de code :

```
cursor.execute("""
    SELECT v.id, m.nom, c.nom, c.prenom, v.quantite, v.prix_total, v.date_vente
    FROM vente v
    LEFT JOIN medicaments m ON v.id_medicament = m.id
    LEFT JOIN clients c ON v.id_client = c.id
""")
resultats = cursor.fetchall()
```

3. Gestion des Erreurs et Cas Particuliers

Toutes les fonctions intègrent une gestion des erreurs via des blocs try...except. Voici les cas gérés :

Doublons : Empêchés via la contrainte UNIQUE dans la base de données.

Données invalides : Ex. : quantité ou prix négatif.

Stock insuffisant : Impossible d'enregistrer une vente si le stock est inférieur à la quantité demandée.

Clés étrangères : Empêchent la suppression d'un médicament si des ventes y sont liées.

4. Conclusion :

Cette partie du projet met en œuvre la **logique applicative** de la gestion d'une pharmacie.
Grâce à l'utilisation de **SQLite3**, des **requêtes SQL paramétrées**, et d'une **structure modulaire** du code, elle assure :

- La fiabilité des données.
- Le respect des contraintes de la base.
- Une interaction claire entre les entités (médicaments, clients, ventes).

Interface Graphique Tkinter pour la Gestion de Pharmacie

1. Présentation générale

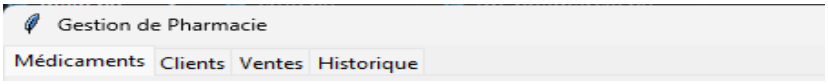
Cette partie du projet couvre **l'interface graphique** de l'application de gestion de pharmacie, développée avec **Tkinter**. Elle permet à l'utilisateur d'interagir avec les données stockées dans la base SQLite (médicaments, clients, vente) et d'effectuer toutes les opérations principales : ajout, modification, suppression, et consultation.L'interface est **structurée en onglets** grâce à ttk.Notebook

pour séparer les fonctionnalités et améliorer l'ergonomie. Un **historique des actions** est également maintenu pour tracer toutes les opérations.

2. Initialisation de l'application

```
root = tk.Tk()
root.title("Gestion de Pharmacie")
root.geometry("1200x650")

notebook = ttk.Notebook(root)
notebook.pack(fill="both", expand=True)

frame_meds = ttk.Frame(notebook)
frame_clients = ttk.Frame(notebook)
frame_ventes = ttk.Frame(notebook)
frame_histo = ttk.Frame(notebook)

notebook.add(frame_meds, text="Médicaments")
notebook.add(frame_clients, text="Clients")
notebook.add(frame_ventes, text="Ventes")
notebook.add(frame_histo, text="Historique")
```

- Création de la **fenêtre principale**.
- Définition du titre et de la taille.
- Mise en place d'un **Notebook** pour organiser l'interface en onglets



3. Gestion de l'historique

```
historique_actions = []

def ajouter_historique(action):
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    historique_actions.append(f"[{now}] {action}")
    refresh_historique()
```

- Chaque action utilisateur est enregistrée dans une **liste historique_actions**.
- Les entrées sont **horodatées**
- La fonction refresh_historique() met à jour le **Listbox** affichant l'historique dans l'onglet correspondant.

4. Onglet Médicaments

a) Formulaire d'ajout

```
frm_top = ttk.LabelFrame(frame_meds, text="Ajouter un médicament")
frm_top.pack(fill="x", padx=10, pady=5)

tk.Label(frm_top, text="Nom:").grid(row=0, column=0)
tk.Label(frm_top, text="Code-barre:").grid(row=0, column=2)
tk.Label(frm_top, text="Description:").grid(row=1, column=0)
tk.Label(frm_top, text="Prix:").grid(row=1, column=2)
tk.Label(frm_top, text="Quantité:").grid(row=2, column=0)
tk.Label(frm_top, text="Expiration:").grid(row=2, column=2)

entry_nom = tk.Entry(frm_top)
entry_code = tk.Entry(frm_top)
entry_desc = tk.Entry(frm_top)
entry_prix = tk.Entry(frm_top)
entry_qte = tk.Entry(frm_top)
entry_date = tk.Entry(frm_top)

entry_nom.grid(row=0, column=1)
entry_code.grid(row=0, column=3)
entry_desc.grid(row=1, column=1)
entry_prix.grid(row=1, column=3)
entry_qte.grid(row=2, column=1)
entry_date.grid(row=2, column=3)

tk.Button(frm_top, text="Ajouter", command=ajouter_medicament).grid(row=3, column=0, pady=5)
tk.Button(frm_top, text="Supprimer", command=supprimer_medicament).grid(row=3, column=1)
tk.Button(frm_top, text="Actualiser", command=afficher_medicaments).grid(row=3, column=2)
```

- Permet de **saisir un nouveau médicament** (nom, code-barre, description, prix, quantité, date d'expiration).
- Boutons pour **ajouter, supprimer et actualiser** le tableau.

b) Tableau (Treeview) des médicaments

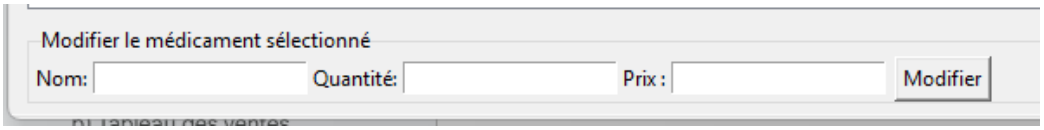
```
cols = ("ID", "Nom", "Code-barre", "Quantité", "Prix", "Expiration")
meds_tree = ttk.Treeview(frame_meds, columns=cols, show="headings")
for c in cols:
    meds_tree.heading(c, text=c)
    meds_tree.column(c, width=150)
meds_tree.pack(fill="both", expand=True, padx=10, pady=5)
meds_tree.bind("<<TreeviewSelect>>", remplir_champs_medicament)

frm_mod = ttk.LabelFrame(frame_meds, text="Modifier le médicament sélectionné")
frm_mod.pack(fill="x", padx=10, pady=5)
tk.Label(frm_mod, text="Nom:").grid(row=0, column=0)
tk.Label(frm_mod, text="Quantité:").grid(row=0, column=2)
tk.Label(frm_mod, text="Prix :").grid(row=0, column=4)
entry_nom_mod = tk.Entry(frm_mod)
entry_qte_mod = tk.Entry(frm_mod)
entry_prix_mod = tk.Entry(frm_mod)
entry_nom_mod.grid(row=0, column=1)
entry_qte_mod.grid(row=0, column=3)
entry_prix_mod.grid(row=0, column=5)
tk.Button(frm_mod, text="Modifier", command=modifier_medicament).grid(row=0, column=6, padx=5)

```

- Affiche tous les médicaments de la base.
- Sélection d'une ligne permet de **remplir le formulaire de modification**.

c) Modification d'un médicament

```
def modifier_medicament():
    selected = meds_tree.focus()
    if not selected:
        messagebox.showwarning("Sélection requise", "Sélectionnez un médicament.")
        return
    id_med = meds_tree.item(selected)["values"][0]
    nom = entry_nom_mod.get() or None
    quantite = int(entry_qte_mod.get()) if entry_qte_mod.get() else None
    prix = float(entry_prix_mod.get()) if entry_prix_mod.get() else None
    crud.modifier_medicament(id_med, nom, quantite, prix)
    ajouter_historique(f"Modification médicament ID {id_med}")
    afficher_medicaments()
```

- Permet de **mettre à jour les informations** d'un médicament sélectionné.
- Toutes les modifications sont tracées dans l'historique.



5. Onglet Clients

a) Formulaire d'ajout

- Saisie des informations : **Nom, Prénom, Date de naissance, Téléphone, Assurance**.
- Boutons pour **ajouter, supprimer, actualiser** le tableau des clients.

b) Tableau (Treeview) des clients

- Affiche les clients existants.

- Sélection d’un client remplit le formulaire de modification.

c) Modification et suppression

- Même logique que pour les médicaments : la sélection permet de **modifier ou supprimer un client**, avec historique automatique.

6. Onglet Ventes

a) Formulaire d’enregistrement d’une vente

```
frm_v = ttk.LabelFrame(frame_ventes, text="Nouvelle vente")
frm_v.pack(fill="x", padx=10, pady=5)

tk.Label(frm_v, text="ID Médicament:").grid(row=0, column=0)
tk.Label(frm_v, text="ID Client:").grid(row=0, column=2)
tk.Label(frm_v, text="Quantité:").grid(row=1, column=0)
tk.Label(frm_v, text="Pharmacien:").grid(row=1, column=2)

entry_v_med = tk.Entry(frm_v)
entry_v_cli = tk.Entry(frm_v)
entry_v_qte = tk.Entry(frm_v)
entry_v_pharma = tk.Entry(frm_v)

entry_v_med.grid(row=0, column=1)
entry_v_cli.grid(row=0, column=3)
entry_v_qte.grid(row=1, column=1)
entry_v_pharma.grid(row=1, column=3)

tk.Button(frm_v, text="Enregistrer vente", command=enregistrer_vente).grid(row=2, column=0, columnspan=4, pady=5)
```

- Permet de saisir : **ID médicament, ID client, quantité, pharmacien**.
- Bouton pour **enregistrer la vente**, qui met à jour la base et l’historique.

b) Tableau des ventes

```
cols_v = ("ID", "Médicament", "Nom Client", "Prénom", "Quantité", "Total", "Date")
ventes_tree = ttk.Treeview(frame_ventes, columns=cols_v, show="headings")
for c in cols_v:
    ventes_tree.heading(c, text=c)
    ventes_tree.column(c, width=150)
ventes_tree.pack(fill="both", expand=True, padx=10, pady=5)

tk.Button(frame_ventes, text="Actualiser", command=afficher_ventes).pack(pady=5)
```

- Affiche : ID vente, médicament, client, quantité, prix total et date.
- Actualisation automatique après chaque vente.