PROJET PYTHON AVANCE : APPLICATION DE GESTION DE BIBLIOTHEQUE

Avec Sqlite3 et Tkinter

REALISE PAR:
Mallahi Mohammed
Nait Abderrahmane Hamza
Sayerh Abdelhak

Github link : mallahimohammed-code/python-project

1. Introduction et Objectifs du Projet

Ce projet s'inscrit dans le cadre du module Python avancé. Il a pour objectif de mettre en œuvre les notions de programmation orientée objet, de gestion de base de données SQLite et de conception d'interfaces graphiques avec Tkinter. Le thème choisi — la gestion d'une bibliothèque — permet de regrouper ces compétences dans une application concrète et fonctionnelle. L'application offre la possibilité de gérer les étudiants, les livres et les emprunts via une interface intuitive.

2. Idées Générales et Architecture du Code

Le projet repose sur deux fichiers principaux :

- db.py : gère la création et la manipulation de la base de données SQLite.
- main.py : contient la logique principale et l'interface Tkinter.

L'approche modulaire a permis de séparer clairement la logique de la couche graphique. Cette conception rend le code plus lisible et plus facile à maintenir.

Voici les idées générales les plus intéressantes du projet :

- Utilisation d'une base de données SQLite embarquée, sans dépendance externe.
- Organisation claire des fonctions CRUD (Create, Read, Update, Delete) pour chaque table.
- Interface Tkinter découpée en trois onglets grâce à la classe Notebook de ttk.
- Chaque bouton de l'interface est relié à une fonction spécifique qui interagit directement avec la base de données via des appels à db.py.
- Utilisation du widget Treeview pour afficher les listes d'étudiants, de livres et d'emprunts de façon structurée.

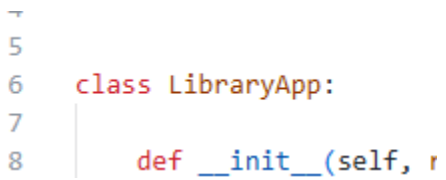
3. Logique Fonctionnelle

Vue d'ensemble de la classe LibraryApp

La classe LibraryApp constitue le cœur de l'application graphique de gestion de bibliothèque.

Elle contrôle la fenêtre principale, les onglets, les boutons et toutes les interactions entre l'utilisateur et la base de données.

Son rôle est de permettre à l'utilisateur d'effectuer les opérations principales du projet — ajouter, supprimer, emprunter et retourner un livre — de manière intuitive et fluide.



Initialisation de la fenêtre principale

Le constructeur __init__ de la classe initialise la fenêtre principale et crée les trois grands onglets de l'application.

```
def __init__(self, root):
    self.root = root
    self.root.title("ENSAM Library Management")
    self.root.geometry("1000x700")

    db.init_database()

    self.notebook = ttk.Notebook(root)
    self.notebook.pack(fill='both', expand=True, padx=10, pady=10)

    self.create_students_tab()
    self.create_books_tab()
    self.create_borrow_tab()
```

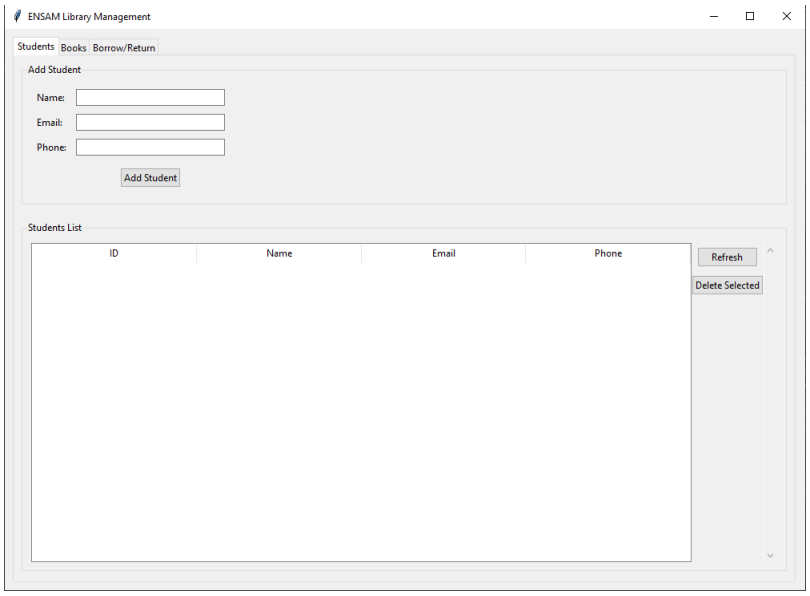
Étapes principales :

- La fenêtre Tkinter (root) est transmise au constructeur.
- Le titre et la taille de la fenêtre sont définis (ENSAM Library Management, 1000x700).
- Une instance de ttk.Notebook est créée : c'est le conteneur qui gère les différents onglets.
- La base de données SQLite est initialisée via db.init_database() (création automatique des tables si elles n'existent pas).

Trois méthodes sont ensuite appelées :

- create_students_tab() : onglet Étudiants
- create_books_tab() : onglet Livres
- create_borrow_tab() : onglet Emprunts / Retours

Ainsi, dès le lancement de l'application, tous les onglets et leurs composants sont prêts à être utilisés.



Onglet Étudiants — create_students_tab()

Cet onglet permet de gérer les étudiants de la bibliothèque : ajouter, afficher ou supprimer.

Structure :

- Un cadre de saisie (LabelFrame) pour ajouter un étudiant avec trois champs :
  - Nom
  - Email
  - Téléphone
- Un bouton "Add Student" qui déclenche la méthode add_student().
- Un second cadre contient un tableau (Treeview) affichant tous les étudiants.
- Des boutons "Refresh" et "Delete Selected" pour actualiser ou supprimer un étudiant.

Fonctionnement :

- Les données sont extraites via la fonction db.get_all_students().
- Le tableau (Treeview) est mis à jour par la méthode refresh_students(), qui vide d'abord les anciennes lignes avant d'afficher les nouvelles.
- Lorsqu'un étudiant est ajouté, supprimé ou modifié, la table est rechargée automatiquement.

```
        self.refresh_students()

    def create_books_tab(self):
        tab_frame = ttk.Frame(self.no
        self.notebook.add(tab_frame,
```

Méthodes liées aux étudiants

add_student()

- Récupère les valeurs entrées dans les champs de texte.
- Vérifie que le nom n'est pas vide (sinon une erreur apparaît via messagebox.showerror()).
- Appelle la fonction db.add_student() pour insérer l'étudiant dans la base.
- En cas de succès, affiche un message de confirmation et vide les champs.
- Rafraîchit la table d'étudiants et la liste des combobox dans l'onglet "Emprunt".

```
def add_student(self):
    name = self.student_name_entry.get().strip()
    email = self.student_email_entry.get().strip()
    phone = self.student_phone_entry.get().strip()

    if not name:
        messagebox.showerror("Error", "Name is required!")
        return

    try:
        db.add_student(name, email, phone)
        messagebox.showinfo("Success", "Student added successfully!")
        self.student_name_entry.delete(0, 'end')
        self.student_email_entry.delete(0, 'end')
        self.student_phone_entry.delete(0, 'end')
        self.refresh_students()
        self.refresh_borrow_combos()
    except ValueError as error:
        messagebox.showerror("Error", str(error))
```

delete_student()

- Récupère la ligne sélectionnée dans le tableau.
- Si aucune ligne n'est choisie, affiche un message d'avertissement.
- Sinon, demande confirmation via messagebox.askyesno() avant suppression.
- Supprime ensuite l'étudiant via db.delete_student() et recharge la table.

```
def delete_student(self):
    selected_items = self.students_table.selection()

    if not selected_items:
        messagebox.showwarning("Warning", "Please select a student to delete")
        return

    selected_row = self.students_table.item(selected_items[0])
    student_id = selected_row['values'][0]

    if messagebox.askyesno("Confirm", "Are you sure you want to delete this student?"):
        db.delete_student(student_id)
        messagebox.showinfo("Success", "Student deleted successfully!")
        self.refresh_students()
        self.refresh_borrow_combos()
```

Onglet Livres — create_books_tab()

Cet onglet gère la collection de livres de la bibliothèque.

```
def create_borrow_tab(self):
    tab_frame = ttk.Frame(self.n
```

Structure :

- Trois champs d'entrée : Titre, Auteur, ISBN.
- Un bouton "Add Book" pour ajouter un livre.
- Un tableau (Treeview) affichant la liste complète des livres avec les colonnes :
- ID, Titre, Auteur, ISBN, Disponible.
- Deux boutons : "Refresh" et "Delete Selected".

Fonctionnement :

- L'ajout se fait via la méthode add_book() qui appelle db.add_book().
- En cas d'ISBN déjà existant, une exception (ValueError) est capturée.
- Le tableau est rempli par refresh_books() qui parcourt la base (db.get_all_books()) et affiche chaque livre.
- La colonne "Disponible" affiche "Yes" ou "No" selon la valeur entière stockée en base (1 = disponible, 0 = emprunté).

Onglet Emprunts / Retours — create_borrow_tab()

Cet onglet relie les deux entités principales : étudiants et livres.

```
def create_borrow_tab(self):
    tab_frame = ttk.Frame(self.notebook)
    self.notebook.add(tab_frame, text="Borrow/Return")
```

Il permet de gérer les emprunts et retours.

Structure :

- Deux menus déroulants (Combobox) :
- Liste des étudiants (affichée sous forme “id : nom”)
- Liste des livres disponibles (affichée sous forme “id : titre”)
- Un bouton “Borrow Book” pour effectuer un emprunt.
- Un tableau Treeview qui affiche tous les emprunts, avec les colonnes :
- ID, Étudiant, Livre, Date d’emprunt, Date de retour.

Deux boutons supplémentaires :

- “Refresh” pour recharger la table.
- “Return Selected” pour enregistrer un retour.

Méthodes clés de l’onglet Emprunt

borrow_book_action()

- Récupère la sélection d’un étudiant et d’un livre.
- Si l’un des deux n’est pas choisi, affiche une erreur.
- Extrait l’ID du texte sélectionné (ex. “2: John Doe” → 2).
- Appelle la fonction db.borrow_book(student_id, book_id) :
    - Si le livre est disponible, il est marqué comme emprunté (available = 0).
    - Sinon, un message d’erreur s’affiche.
- Rafraîchit ensuite la liste des livres et des emprunts.

```
def borrow_book_action(self):
    student_selection = self.student_combobox.get()
    book_selection = self.book_combobox.get()

    if not student_selection or not book_selection:
        messagebox.showerror("Error", "Please select both student and book")
        return

    student_id = int(student_selection.split(':')[0])
    book_id = int(book_selection.split(':')[0])

    success = db.borrow_book(student_id, book_id)

    if success:
        messagebox.showinfo("Success", "Book borrowed successfully!")
        self.refresh_books()
        self.refresh_borrows()
        self.refresh_borrow_combos()
    else:
        messagebox.showerror("Error", "Book is not available!")
```

return_book_action()

- Récupère la ligne sélectionnée du tableau d'emprunts.
- Vérifie si le livre est déjà retourné.
- Si non, confirme l'action avec l'utilisateur.
- Appelle db.return_book(borrow_id) qui met à jour la date de retour et rend le livre disponible à nouveau (available = 1).
- Actualise toutes les tables après opération.

```
def return_book_action(self):
    selected_items = self.borrows_table.selection()

    if not selected_items:
        messagebox.showwarning("Warning", "Please select a borrow record to return")
        return

    selected_row = self.borrows_table.item(selected_items[0])
    borrow_id = selected_row['values'][0]
    return_date = selected_row['values'][4]

    if return_date != "Not returned":
        messagebox.showwarning("Warning", "This book has already been returned")
        return

    if messagebox.askyesno("Confirm", "Return this book?"):
        db.return_book(borrow_id)
        messagebox.showinfo("Success", "Book returned successfully!")
        self.refresh_books()
        self.refresh_borrows()
        self.refresh_borrow_combos()
```

Gestion du rafraîchissement des données

Chaque onglet dispose d'une méthode refresh_*() :

- refresh_students()
- refresh_books()
- refresh_borrows()

Ces méthodes suivent le même schéma :

1. Effacer toutes les lignes actuelles du tableau (Treeview.delete()).
2. Récupérer les données actualisées depuis la base.
3. Réinsérer les nouvelles lignes dans l'interface.

Cette technique évite les doublons et assure une synchronisation parfaite entre la base SQLite et l'affichage graphique.

```
def refresh_students(self):
    for row in self.students_table.get_children():
        self.students_table.delete(row)

    students = db.get_all_students()

    for student in students:
        self.students_table.insert('', 'end', values=student)
```

```python
def refresh_books(self):
    for row in self.books_table.get_children():
        self.books_table.delete(row)

    books = db.get_all_books()

    for book in books:
        book_id = book[0]
        title = book[1]
        author = book[2]
        isbn = book[3]
        available = book[4]

        if available == 1:
            available_text = "Yes"
        else:
            available_text = "No"

        self.books_table.insert('', 'end', values=(book_id, title, author, isbn, available_text))
```

```python
def refresh_borrow_combos(self):
    students = db.get_all_students()
    student_options = []
    for student in students:
        student_id = student[0]
        student_name = student[1]
        student_options.append(f"{student_id}: {student_name}")

    self.student_combobox['values'] = student_options

    books = db.get_all_books(available_only=True)
    book_options = []
    for book in books:
        book_id = book[0]
        book_title = book[1]
        book_options.append(f"{book_id}: {book_title}")

    self.book_combobox['values'] = book_options
```

Conclusion

La classe LibraryApp illustre une conception bien organisée d'une application Python complète :

- Interface graphique moderne grâce à ttk.Notebook et Treeview.
- Gestion efficace des données avec SQLite.
- Interaction fluide entre interface et base via des fonctions simples et claires.

Cette application démontre comment combiner les deux volets du module Python avancé — base de données et interface graphique — dans un projet cohérent, lisible et professionnel.

C'est un excellent exemple pour tout projet de gestion (école, bibliothèque, formation, etc.).