RECOMMENDATION DE LIVRE

Rattrapage Projet Transverse

efrei

PARIS PANTHÉON-ASSAS UNIVERSITÉ

LAURENT CHAU
PROMO 2025

Sommaire

Sommaire 1
I. Présentation du projet 2
II. Réalisation du projet 2
1) Manipulation des lecteurs 2
2) Manipulation des livres 5
3) Notation et recommandation d'un livre 5
4) Interface graphique 7
III. Conclusion 8

I. Présentation du projet

Ce projet a pour objectif la réalisation d'un système de recommandation de livres. Pour ce faire, nous examinerons les livres lus par l'utilisateur et comparons ces lectures avec les lectures des autres lecteurs afin de déterminer quelle livre nous lui recommanderons.
En plus de recommander des livres, l'utilisateur aura accès à d'autre fonctionnalité comme : consulter la liste des livres présents dans le dépôt, ajouter, modifier ou retirer des livres à celui-ci ainsi qu'ajouter, modifier ou retirer des lecteurs dans la base de données.

Ce projet a été réaliser avec le langage Python. La librairie PySimpleGUI a été utilisé pour réaliser l'interface graphique du programme.

II. Réalisation du projet

Pour réaliser ce projet, j'ai choisi de diviser le travail en quatre axes :

- Manipulation des lecteurs
- Manipulation des livres
- Notation et recommandation d'un livre
- Interface graphique

A l'intérieur de chaque axe, j'ai choisi de représenter chaque fonctionnalité demander dans la consigne par une unique fonction. Certaines de ces fonctions feront appelés à des fonctions intermédiaires qui ont été écrites dans le but d'alléger le code et le rendre plus lisible.

1) Manipulation des lecteurs

La manipulation des lecteurs consiste en quatre fonctionnalités :

- Ajouter un lecteur
- Afficher un lecteur
- Modifier un lecteur
- Supprimer un lecteur

Pour ajouter un lecteur, j'ai commencé par définir des dictionnaires contenant les différents attributs possibles pour un lecteur. Ces dictionnaires me permettent de faciliter la modification des attributs, si nécessaire, et permettra, plus tard, de faciliter l'affichage.

```
GENDER = {
    "1": "Man",
    "2": "Woman",
    "3": "Other",
}

AGE = {
    "1": "<= 18 years old",
    "2": "Between 18 years old and 25 years old",
    "3": "> 25 years old",
}

READING_STYLE = {
    "1": "Science fiction",
    "2": "Biography",
    "3": "Horror",
    "4": "Romance",
    "5": "Fable",
    "6": "History",
    "7": "Comedy",
}
```

J'ai alors créé des fonctions intermédiaires me permettant de récupérer les attributs d'un lecteur.

```
def get_attribut(attribut: dict[str, str]) -> str:
    """Get the attribut of the reader to be added."""

    # Display the attribut dict
    for key, value in attribut.items():
        print(f"{key} - {value}")

    while True:
        attribut_number = input("Enter the corresponding number: ")
        # Check if attribut is valid
        if attribut_number not in list(attribut.keys()):
            print("Incorrect number.")
        else:
            break

    return attribut_number
```

Une fois tous les attributs récupérer, je n'ai plus qu'à écrire ces nouvelles informations dans le fichier dépôts à la fin de celui-ci.

```
def add_reader(books: str, readers: str, booksread: str,
rating_matrix: str) -> None:
    """Add a reader to the database."""

    with open(readers, "a", encoding="utf-8") as readers_file,
open(booksread, "a", encoding="utf-8") as booksread_file:
        reader_information = [
            get_username(readers),
```

```
            get_attribut(GENDER),
            get_attribut(AGE),
            get_attribut(READING_STYLE),
        ]

        reader_booksread = [reader_information[0]] +
get_booksread(books)

        readers_file.write(",".join(reader_information) + "\n")
        booksread_file.write(",".join(reader_booksread) + "\n")
    # Add row to rating matrix
    rating.add_row(books, rating_matrix)
```

Pour l'affichage, je demande à l'utilisateur le pseudo du lecteur à afficher tout en vérifiant son existence dans le dépôt. J'utilise ensuite des expressions régulières afin d'obtenir les informations le concernant sans avoir besoin de parcourir l'entièreté du fichier. Grâce aux dictionnaires définis plus haut, l'utilisateur à accès à des informations compréhensibles par tous.

```
def display_reader(books: str, readers: str, booksread: str) -> None:
    """Display a reader in the database."""

    with open(readers, encoding="utf-8") as readers_file,
open(booksread, encoding="utf-8") as booksread_file:
        readers_file_content = readers_file.read()
        booksread_file_content = booksread_file.read()

    while True:
        reader_username = input("Enter the reader's username:
").lower()

        if reader_username in readers_file_content.lower():
            break
        else:
            print("This reader is not in the database.")

    pattern = re.compile(f"{reader_username}.+", re.MULTILINE |
re.IGNORECASE)
    reader_line = pattern.search(readers_file_content)[0].split(",")
    booksread_line =
pattern.search(booksread_file_content)[0].split(",")

    print(f"Username: {reader_line[0]}")
    print(f"Gender: {GENDER[reader_line[1]]}")
    print(f"Age: {AGE[reader_line[2]]}")
    print(f"Reading style: {READING_STYLE[reader_line[3]]}")

    print("Book(s) read:")
    books_name = get_books_name(books, booksread_line[1:])
    for name in books_name:
        print(name)
```

J’ai ensuite décider de combiner les fonctions de modification et suppression car elles partagent un même principe. Comme pour l’affichage, j’utilise des expressions régulières pour accéder aux contenus dont j’ai besoin et j’effectue l’opération souhaité.

Dans le cadre de la modification, je demande à l’utilisateur la saisie d’un nouveau profil qui remplacera l’ancien. Pour la suppression, je supprime également le livre dans la liste des livres lus.

```
pattern = re.compile(f"{to_modify}.+(?=^\w)", re.MULTILINE | re.DOTALL
| re.IGNORECASE)

    if delete:
        new_readers_file = pattern.sub("", readers_file_content)
        new_booksread_file = pattern.sub("", booksread_file_content)

        rating.delete_row(rating_matrix, readers_names[to_modify])
    else:
        # Generate a new profile
        reader_informations = [
            get_alias(readers),
            get_attribut(GENDER),
            get_attribut(AGE),
            get_attribut(READING_STYLE),
        ]
        reader_booksread = [reader_informations[0]] +
get_booksread(books)

        new_readers_file = pattern.sub(",".join(reader_informations) +
"\n", readers_file_content)
        new_booksread_file = pattern.sub(",".join(reader_booksread) +
"\n", booksread_file_content)
```

2) Manipulation des livres

Les fonctions de manipulations des livres ressemblent sensiblement aux fonctions de manipulation des lecteurs. Je n’explorerais alors pas cette partie dans le rapport puisqu’elles fonctionnent de la même manière. La différence se fait dans les informations demandées à l’utilisateur pour les faire fonctionner.

3) Notation et recommandation d’un livre

Pour noter un livre, on doit s’assurer que le lecteur soit présent dans la base de données, que le livre qu’il souhaite noter soit bien présent dans sa liste des livres lus. Pour ce faire, on sécurise l’entièreté des saisies.

```
        while True:
            reader_username = input("Enter the reader's username:
").lower()
```

```
            if reader_username not in list(readers_name.keys()):
                print("Reader not in the database.")
            else:
                break

        while True:
            book = input("Enter the name of the book: ").lower()

            if book not in list(books_name.keys()):
                print("Book not in the database.")
            else:
                book = books_name[book]
                pattern = regex.compile(f"{reader_username}.+",
regex.MULTILINE | regex.IGNORECASE)

                if book not in
pattern.search(booksread_file_content)[0]:
                    print("You did not read that book.")
                else:
                    break
```

Une fois toutes ces conditions respectées, le lecteur peut entrer une note entre 1 et 5 (la saisie est aussi sécurisée ici). On insèrera cette note à sa position dans la matrice de notation.

Pour recommander un livre, on commence par calculer la matrice de similarité. Cette matrice de similarité utilise notre matrice de notation pour comparer la liste des livres lus de chaque utilisateur est dégagé un score de similarité. Ce score de similarité respecte l'expression suivante :

$$\frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \; x \; \sqrt{\sum_{i=1}^{n} b_i^2}}$$

```
def compute(reader_1: list[int], reader_2: list[int]) -> float:
    """Compute the similarities between two readers."""

    numerator = sum((a * b) for a, b in zip(reader_1, reader_2))
    denominator = math.sqrt(sum((a * a) for a in reader_1)) *
math.sqrt(sum((b * b) for b in reader_2))

    return round(numerator / denominator, 2)
```

Une fois la matrice calculée, on trouve le lecteur le plus similaire. Pour cela, on cherche le plus gros score de similarité excepté 1 (1 qui représente notre lecteur de base).

Après cela, il nous suffit de récupérer les listes des livres lus pour les deux lecteurs et de faire la différence des deux listes. Cette différence nous donnera une liste des livres non-lus par notre lecteur de base. Cette différence sera les recommandations de livres.

```
    user_1 = [int(x) for x in user_1]
    user_2 = [int(x) for x in user_2]

    books_recommended =
list(set(user_1).symmetric_difference(set(user_2)))
    books_recommended = re.get_books_name(books, books_recommended)

    print("Book(s) recommended:")
    for book in books_recommended:
        print(book)
```

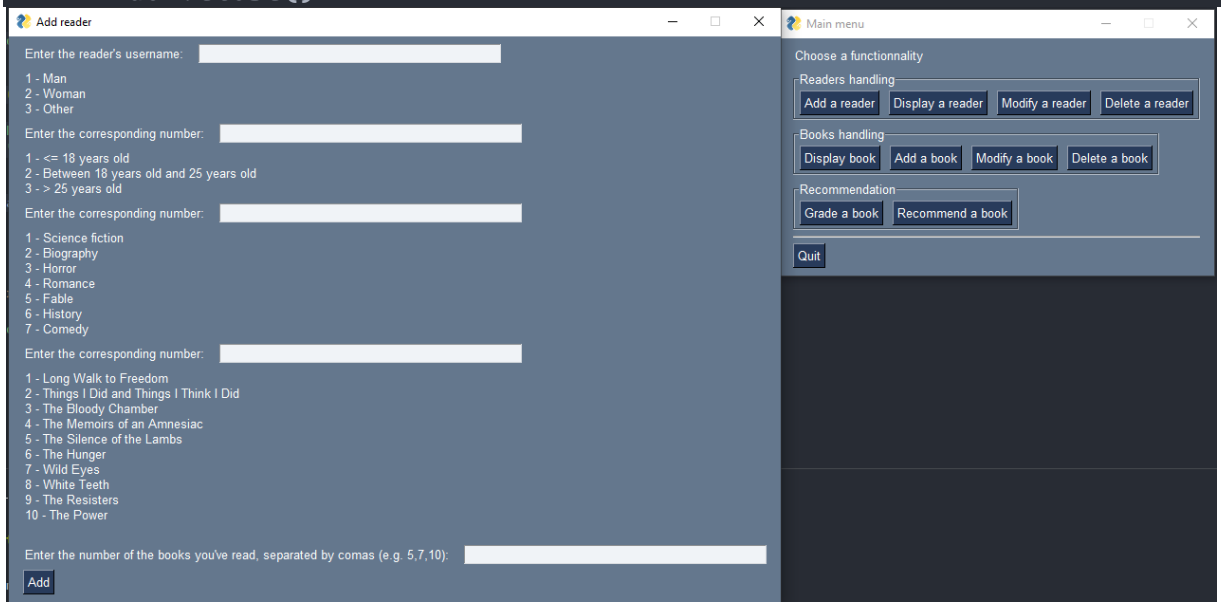
4) Interface graphique

N'ayant aucune connaissance dans la réalisation d'interface graphique, je me suis orienté vers la librairie PySimpleGUI qui combine plusieurs librairies comme Tkinter, WxPython dans une interface simplifiée.

Cette librairie ne nous permet pas de créer des fenêtres dynamiques, c'est-à-dire que le contenu de la fenêtre est figé à sa création, j'ai décidé alors d'associer chaque fonctionnalité à une fenêtre distincte.

```
def add_reader(books: str, readers: str, booksread: str,
rating_matrix: list[list[int]]) -> None:
    layout = [
        [sg.Text("Enter the reader's username: "), sg.In(key="-NAME-
")],
        [sg.Text(gender)],
        [sg.Text("Enter the corresponding number: "), sg.In(key="-
GENDER-")],
        [sg.Text(age)],
        [sg.Text("Enter the corresponding number: "), sg.In(key="-AGE-
")],
        [sg.Text(reading_style)],
        [sg.Text("Enter the corresponding number: "), sg.In(key="-
READING_STYLE-")],
        [sg.Text(b.display_books(books))],
        [sg.Text("Enter the number of the books you've read, separated
by comas (e.g. 5,7,10): "), sg.In(key="-BOOKS_READ-")],
        [sg.Button("Add")]
    ]

    window = sg.Window("Add reader", layout, finalize=True,
modal=True)
    block_focus(window)

    event, values = window.read()
```

```
    reader_profile = [values["-NAME-"], values["-GENDER-"], values["-
AGE-"], values["-READING_STYLE-"]]
    reader_booksread = [values["-NAME-"]] + values["-BOOKS_READ-
"].split(",")

    if event == "Add":
        re.add_reader(books, readers, booksread, rating_matrix,
reader_information=[reader_profile, reader_booksread])

    window.close()
```



Chaque fenêtre contient au minimum un bouton, dont je vérifie l'état grâce aux évènements de la fenêtre afin de déclencher les fonctions associées à la fonctionnalité.

III. Conclusion

La réalisation de ce projet m'a permis de renforcer mes connaissances en Python et d'obtenir une première expérience dans la réalisation d'interface graphique.
Il m'a également permis de me former au métier d'ingénieur grâce à la réalisation d'un projet avec un cahier des charges précis et des contraintes bien définis.

Ce projet peut être améliorer en sécurisant chaque saisie lors de l'utilisation d'une interface graphique ainsi qu'améliorer notre interface graphique afin de proposer une meilleur expérience graphique.