<u>LIBRARY MANAGEMENT SYSTEM</u>

<u>Created By: Sohil Rashtrapal Chavhan</u>
<u>Registration number: 25BAI10012</u>
<u>Submitted to: Prof. G.Prabu Kanna</u>

INTRODUCTION

I built a Library Management System because keeping track of books manually is a huge pain. You know how traditional libraries use those messy paper registers? They are slow, and it's super easy to make mistakes. I wanted to create a desktop application that digitizes that whole process, making it faster and error-free for a librarian to manage their inventory.

What the Code Actually Does Basically, I created a GUI (Graphical User Interface) where the user can click buttons to manage the library instead of typing commands. The system is broken down into four main jobs:

1. Adding Books: You can enter a book title and pick a genre (like *Fantasy, Classics, or Thriller*) from a dropdown menu. The code automatically files it into the right category.
2. Viewing Inventory: I built a "View" window that sorts all our books into a grid. It shows exactly what is on the shelf right now, categorized by genres like *Science Fiction* or *Self Help*.
3. Issuing Books: This is the cool part—you can select multiple books at once (using Ctrl+Click) and "check them out." The code moves them from the available list to a temporary "Issued" list so no one else can take them.

4. Returning Books: When a book comes back, the system puts it right back into the correct genre slot automatically.

The Tech Stack I Used

I wrote the whole thing in Python because it handles data really well.

Tkinter: I used the `tkinter` library to build the windows, buttons, and the main interface. It allows for the pop-up alerts (message boxes) that tell you if you forgot to select a genre or if a book was issued successfully.

Data Structures: Instead of a complex database, I used Python Dictionaries and Lists to handle the data in real-time. The books are stored in a dictionary where the "Key" is the genre, and the "Value" is a list of book titles. This makes searching and sorting super fast.

Problem Statement

Traditional library management often relies on manual record-keeping using physical ledgers or disparate spreadsheets.Searching for a specific book or checking its availability status manually takes a lot of time. This manual method makes the system very prone to mistakes which leads to books being lost.
This project solves this problem by reducing the manual labour of keeping records by using a digital interface which makes the entire process of adding, viewing, issuing and returning books.

Functional Requirements

The functional requirements of my project are::
1. Use of Tkinter
2. Use of Messagebox
3. Use of Tkk

This project has a clean UI design which helps the uer to easily navigate through the system.

The Logical Workflow

Think of this program as a traffic controller for data. It doesn't just store text; it constantly moves strings between two main storage containers: the **Main Library Dictionary** (`buk_gener_list`) and the **Issued List** (`buk_out_list`).

Here is the step-by-step flow of how the logic handles the user's actions:

1. The "Main Hub" (Startup)

- **Action**: The script starts and launches `da_big_win` (the root Tkinter window).
- **Logic:** It sits idle waiting for an event (a button click). It doesn't load the heavy data grids yet; it just presents the four main control buttons hooked up to trigger functions like `open_put_win` or `open_see_win`.

2. The "Add Book" Logic (Validation First)

- **Action:** User clicks "Add Book" and hits the `put_new_buk_in` function.
- **Input Check:** Before saving anything, the code runs a security check using `if` statements:
  - *Is the name empty?* (`if not nam_buk`) -> Throw error.
  - *Is the genre "Select Genre"?* -> Throw error.

    - *Is the book already there?* -> It calls `get_all_dem_buk()` to scan for duplicates.
- **Success:** If everything is clean, it pushes the new title into the specific list inside the `buk_gener_list` dictionary (like `buk_gener_list['Fantasy'].append(...)`) and sorts it alphabetically so it looks neat.

3. The "View" Logic (Dynamic Generation)

- **Action:** When we open the view window, the `mak_da_grid` function kicks in.
- **The Loop:** Instead of manually coding every row, the code iterates through the dictionary keys (`for k in buk_gener_list`).
- **Dynamic UI:** For every genre key found, it creates a new `Frame` and a `Listbox` on the fly. This means if we added a new genre in the code later, the GUI would automatically adjust without us rewriting the window logic.

4. The "Issue" Logic (The Transfer)

This is where the code does the heavy lifting in the `take_buk_home` function.

- **Selection:** The user selects one or multiple books from the listbox (`take_box_list`).
- **The Move:** The system doesn't just copy the name; it **cuts and pastes**:
    1. It identifies the book and its genre key.
    2. It **removes** the book from the main dictionary (`buk_gener_list[g].remove(n)`).
    3. **String Formatting:** It creates a special string format: `f"{n} | {g}"` (e.g., "Dune | Fantasy").
    4. It appends that formatted string to the `buk_out_list`.
- **Why the format?** We attach the genre (`| Genre`) to the name so that when the book comes back, the system remembers which shelf to put it back on.

5. The "Return" Logic (The Parsing)

- **Action:** User selects a book from the "Return" list (`buk_out_list`) and clicks "Give Back".
- **The Split:** The code reads the string `"Dune | Fantasy"`. It uses the `.split(" | ")` method to separate the name (*Dune*) from the home (*Fantasy*).
- **Restoration:**
    1. It deletes the entry from `buk_out_list`.

2. It sends *Dune* back to `buk_gener_list["Fantasy"]`.
3. It calls `.sort()` again to keep the inventory organized.

6. The Refresh Cycle (`do_updat_evryting`)

- **Final Step:** After *any* add, issue, or return, the system forces a screen refresh by calling `do_updat_evryting`.
- **Sync:** It loops through every listbox in `see_box_list` and `take_box_list`, clears them (`delete(0, END)`), and re-inserts the data from the dictionaries. This ensures what you see on the screen is 100% synced with what's in the computer's memory

Non-Functional Requirements

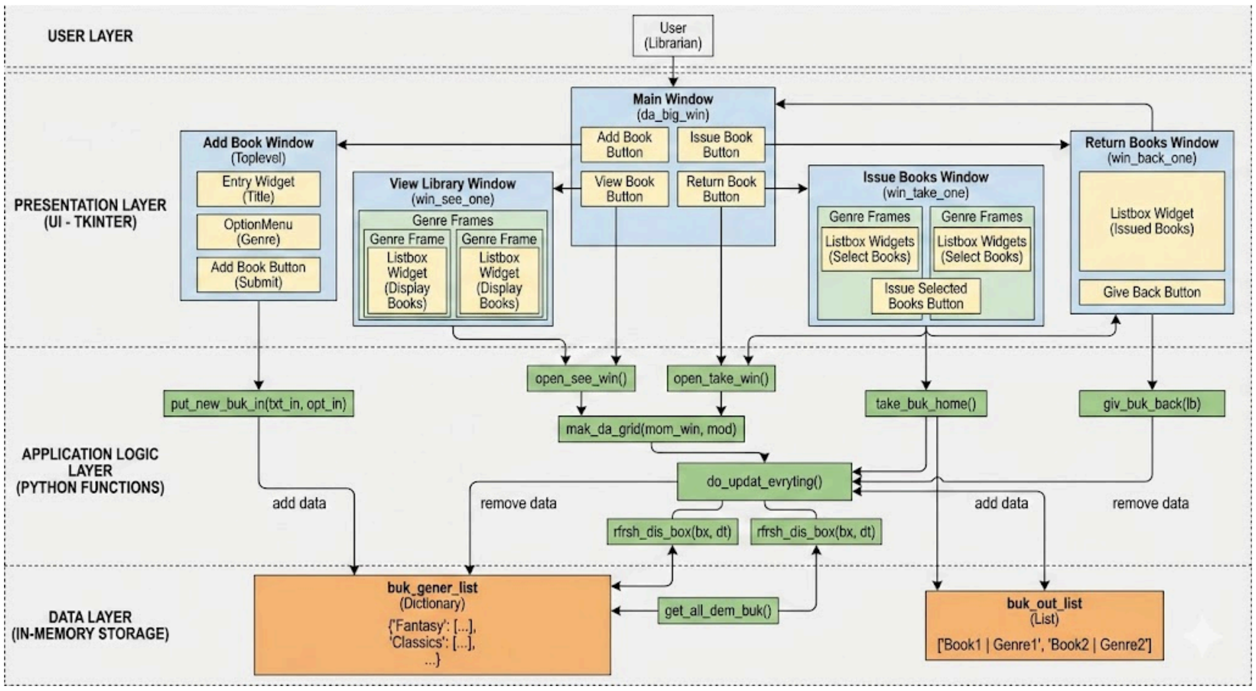
The project contains the following Non-Functional Requirements:
1. Performance: The project insures a smooth performance

2. Usability : The project can be used by anyone because of it’s user friendly UI

3. Maintainability: The project can easily be maintained because the code gives the admin a lot of changes to tweak the library

4. Reliability: Because the code has many exception cases covered the chances of an error rising is very less thus making the project reliable

SYSTEM ARCHITECTURE AND DESIGN


USER LAYER
User (Librarian)
PRESENTATION LAYER (UI - TKINTER)
Main Window (da_big_win)
Add Book Button
Issue Book Button
View Book Button
Return Book Button
Add Book Window (Toplevel)
Entry Widget (Title)
OptionMenu (Genre)
Add Book Button (Submit)
View Library Window (win_see_one)
Genre Frames
Genre Frame
Listbox Widget (Display Books)
Issue Books Window (win_take_one)
Listbox Widgets (Select Books)
Issue Selected Books Button
Return Books Window (win_back_one)
Listbox Widget (Issued Books)
Give Back Button
APPLICATION LOGIC LAYER (PYTHON FUNCTIONS)
put_new_buk_in(txt_in, opt_in)
open_see_win()
open_take_win()
mak_da_grid(mom_win, mod)
take_buk_home()
giv_buk_back(lb)
do_updat_evryting()
rfrsh_dis_box(bx, dt)
add data
remove data
DATA LAYER (IN-MEMORY STORAGE)
buk_gener_list (Dictionary)
{'Fantasy': [...], 'Classics': [...], ...}
get_all_dem_buk()
buk_out_list (List)
['Book1 | Genre1', 'Book2 | Genre2']

Design Decisions & Rationale

Reasons to choose this specific design:
1.The interface should be precise.
2.The background should be subtle to make the options clearer.
3. The buttons should pop out for easier ease of access.
4. The font was also chosen in such a way so that the menu and the buttons are clearly visible.
5. The exit buttons are marked red for easier access

Implementation Details

Objective:
The objective of the project is to make it easier for the customers to place their orders and also make it easier for the admin/owners to manage it.

Scope of the Project

This project can be further expanded to be used in bigger organizations to manage and keep track of their data. The

current version focuses on administrative backend operations (staff-facing) rather than a public-facing catalog for students/members.
The scope of this project is to make it accessible for the customer as well so that issuing and returning a book becomes less cumbersome.

Inventory Management (Keeping Track of Stuff)

> **Cataloging:** This is just logging the books. We type in the details—like the title, who wrote it, and what genre it is—so the computer knows what we actually have.
>
> **Database Integrity:** This stops us from adding the same book twice. Everything gets its own specific ID, so the system doesn't get confused about how many copies we have.
>
> **Status Tracking:** This part is a lifesaver. It updates the status of the book automatically. It tells us if it's sitting on the shelf or if someone checked it out, so we don't have to actually walk over and look for it.

Here is the "Operational Scope" section rewritten in those same two styles.

Option 1: The "Class Project Report" Tone

Professional enough for a grade, but simple enough to be clear.

Operational Scope (How it Works)

Admin Access Only: This system is designed just for the library staff, like the librarians or admins. We didn't build a login for students in this version, so they can't access it remotely to reserve books from home yet.

Saving Data: The system makes sure that all the records—like book lists and who borrowed what—are actually saved. This way, when you close the program, the data doesn't just disappear; it’s still there when you open it again.

Who uses it and how?

Just for the Librarian: Basically, this is for internal use only. We didn't make a user side for this, so random students can't log in or reserve stuff online. It's just for the admin to manage things at the desk.

Assumptions and Constraints

Financial Management: This version does not calculate late fees or fines for overdue books.

User Account Management: The system does not currently register students/members with passwords; it likely uses just a name or ID to issue the book.

Barcode Integration: Data entry is manual rather than automated via barcode scanners.

Targeted Users

The primary daily users of the system are librarians. They utilize the software to issue books to members, process returns, and check the immediate availability of titles requested by visitors.

TESTING APPROACH

Steps that I took while testing the project:

1. I first checked if the windows were correctly popping up.

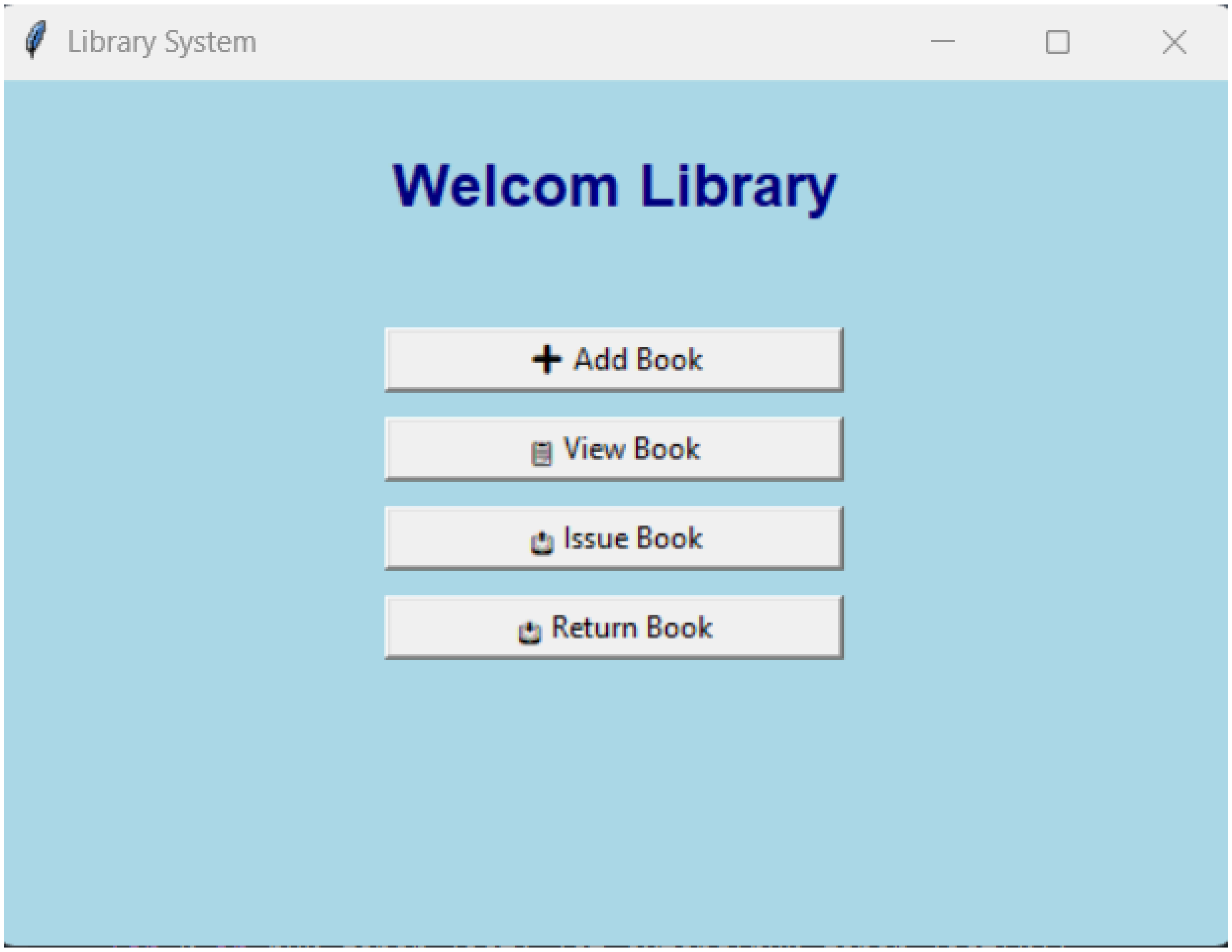
2. I made the UI without any functional code to ensure that the placement of the objects are correct.
3. I made a main code that I executed to get an idea about the execution with the UI

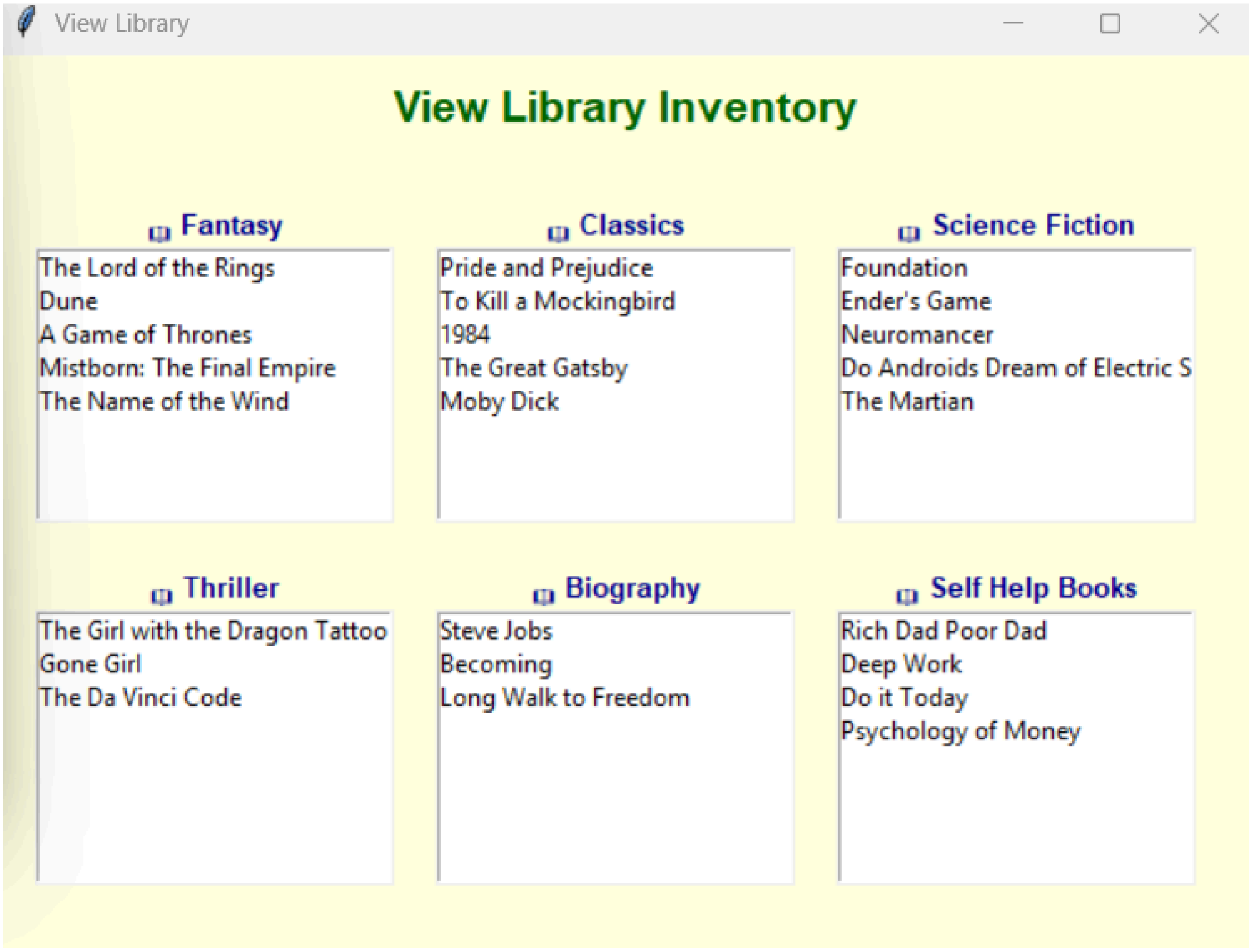
4. I made two separate codes for the two separate windows and checked them separately.

5. Upon my satisfaction, I used the two codes and made them into functions in the main code.

6. I then ran the main code and ran it, giving wrong or invalid inputs to check for exception cases.

SCREENSHOTS


Library System
Welcom Library
Add Book
View Book
Issue Book
Return Book



View Library Inventory

Fantasy
The Lord of the Rings
Dune
A Game of Thrones
Mistborn: The Final Empire
The Name of the Wind

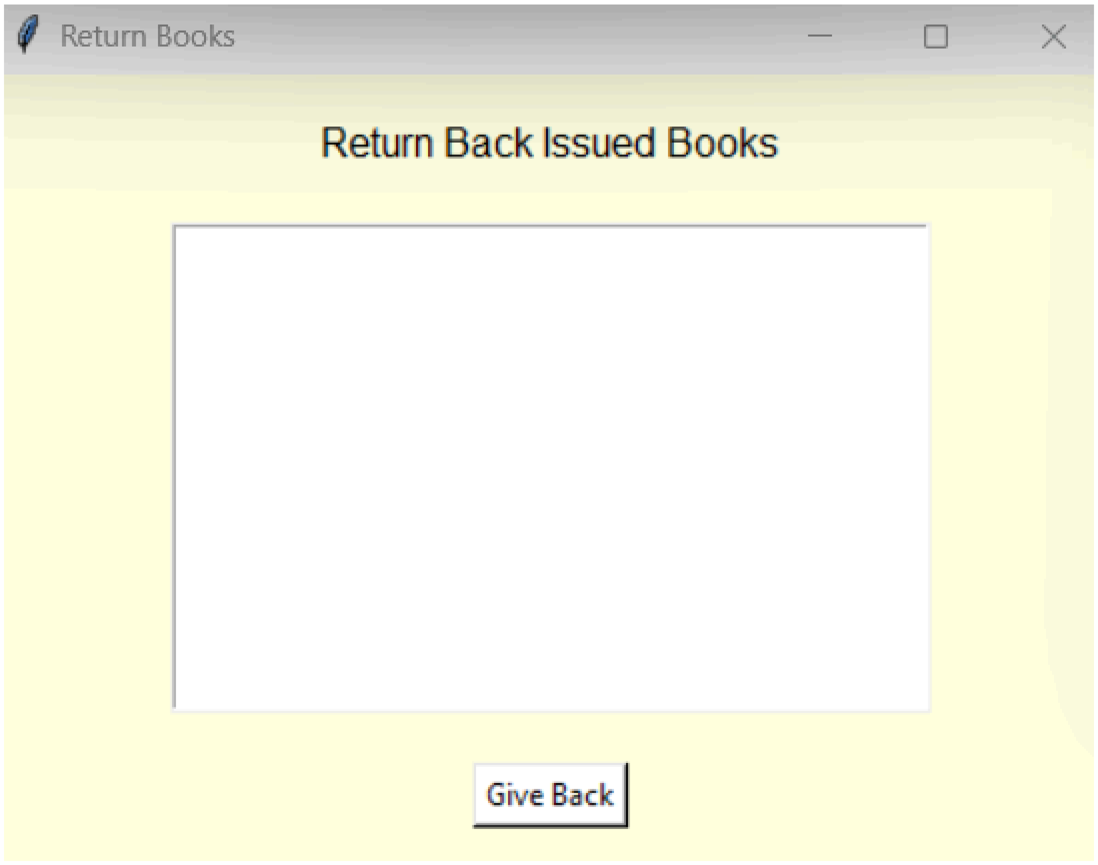
Classics
Pride and Prejudice
To Kill a Mockingbird
1984
The Great Gatsby
Moby Dick

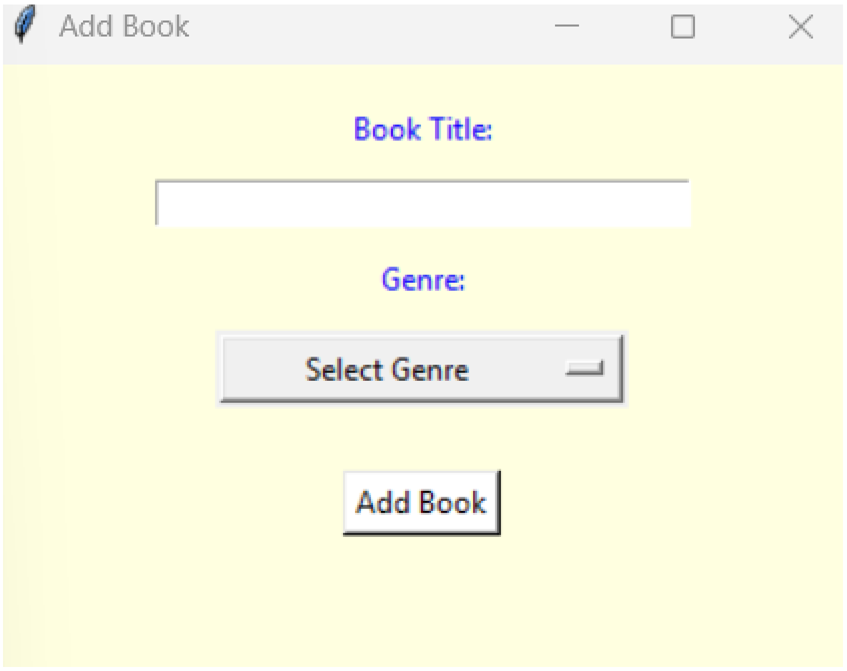
Science Fiction
Foundation
Ender's Game
Neuromancer
Do Androids Dream of Electric S
The Martian

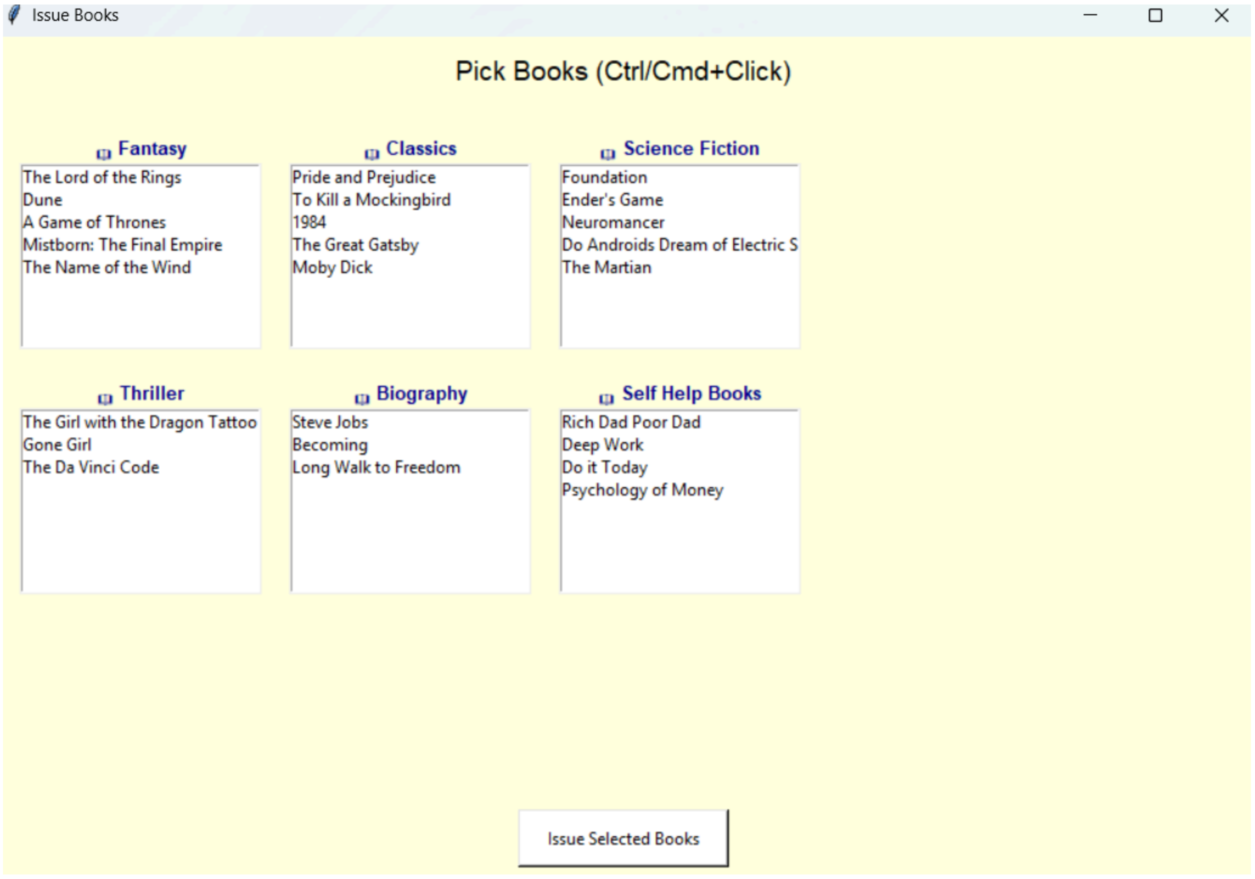
Thriller
The Girl with the Dragon Tattoo
Gone Girl
The Da Vinci Code

Biography
Steve Jobs
Becoming
Long Walk to Freedom

Self Help Books
Rich Dad Poor Dad
Deep Work
Do it Today
Psychology of Money


Return Books
Return Back Issued Books
Give Back


Add Book
Book Title:
Genre:
Select Genre
Add Book



```
The Lord of the Rings
Dune
A Game of Thrones
Mistborn: The Final Empire
The Name of the Wind
```

```
Pride and Prejudice
To Kill a Mockingbird
1984
The Great Gatsby
Moby Dick
```

```
Foundation
Ender's Game
Neuromancer
Do Androids Dream of Electric S
The Martian
```

```
The Girl with the Dragon Tattoo
Gone Girl
The Da Vinci Code
```

```
Steve Jobs
Becoming
Long Walk to Freedom
```

```
Rich Dad Poor Dad
Deep Work
Do it Today
Psychology of Money
```

CHALLENGES FACED

The challenges that I faced during the writing of this project are:

1. I had to go deeper into the topic of UIs and choose one that was correct for me
2. I chose Tkinter but because I had very less experience with it, I had to first master it’s basics.
3. Even after learning Tkinter, I ran into some issues with the button placement, confusing between functions like pack, place, grid
4. I faced the most issues when I decided to make the project a multi window interactive one.
5. I had to learn about different methods on how to do it. The one that worked for me was TopLevel
6. I also faced error for the update function as I had to do a lot of work to make it happen.

Learning Keys and Takeaways

The Learning Keys and Takeaways are:

1. I learned how to use Tkinter

2. I learned how to create buttons, scrollboxes and selections using cursors.

3. I learned how to integrate UI with python code.

4. I also learned how to work with Multiwindow interfaces and how to make them work.

5. I learned the format and how to write codes that I had to integrate in the buttons.

6. I learned about Entry and placements of objects in an interface.

FUTURE ENHANCEMENTS

Here is a list of future enhancements, categorized by priority, that would transform this from a student project into a professional software product.

1. I could replace the lists with a proper database like **SQLite** (for small libraries) or **MySQL/PostgreSQL.**

2. Create a login screen with encrypted passwords.

3. Due Date Calculation**:**
   When a book is issued, automatically calculate a return date.

4. I could add a search box that filters results in real-time by Title, Author,etc.

5. Libraries use barcode scanners to update the code so a librarian can scan a book's ISBN code to instantly pull it up on the screen instead of typing the name.

References

1. Bro Code YT Channel .
   Used it to learn Tkinter

THANK YOU