# Introduction

In this project I created an interactive GUI-based library kiosk service for librarians and patrons. Patrons can checkout and return books. Librarians can create and delete librarian and patron accounts, as well as add or remove books to the library’s collection. I built these features using object-oriented design, incorporating principles such as encapsulation, abstraction, inheritance, dependency injection, and others. I will begin by introducing the patron’s point of view using the kiosk, then I will detail these processes behind the scenes by explaining my code. I will then follow this from the librarian’s point of view. Finally, I will discuss my findings, challenges, and areas of improvement I found when working on this project.

# Opening the Application

Before the application’s GUI is activated, the application will initialize the file services. The file services include User\_File, Transaction\_File, Role\_File, and Book\_File. These are all based on custom json encoding and decoding.

## User\_File

## Transaction\_File

## Role\_File

## Book\_File

## Activating tkinter Frames

## GUI Perspective

When initially opening the application, one can see that there are multiple options. There is the option to log in so that the user can access role-based processes depending on the user’s login information. There also exists an option to view book details.

### Logging In

The login process begins with the user choosing to enter their ID information into the designated textbox, then pressing the login button. This can be viewed in the Login\_Frame under the library.py module.

#### *library.Login\_Frame*

Pressing the login button will result in the user\_id being sent into the kiosk.py module, which handles the internal affairs of the library’s processes. Specifically, the user\_id is sent to the login method which lies in the Kiosk class.

#### *kiosk.Kiosk.login*

The checkout\_item method takes two parameters as inputs, the kiosk object and the user\_id. This will then call the method from the users.py module.

#### *users.User\_File.find\_user*

The find\_user method will then check and see if there indeed exists a user in the user\_store such that the IDs will match. If it is false, then the exception User\_Not\_Found will be thrown. If there is a match, then the user object corresponding to the ID initially provided will be set to the library.py global variable logged\_user. If I was running this application as a web app, then the logged\_user data would be represented as a session. Since there is no session, I am storing the user as a global variable so it can be accessed easily by the library.py module. Now, the login process is complete. If the user had the aforementioned exception was thrown, then the user will have to redo the login process. Otherwise, depending on whether the user is a patron, or a librarian will result in different menus after logging in. This is due to role-based authentication.

#### Role Based Authentication

I designed the user object with the idea that different roles such as patrons and librarians would have different permissions. Patrons would have permission to checkout and return books, whereas the librarians would have permission to generate reports as well as add and remove patrons, librarians, and books from the library collection. I will provide more details on this in the dedicated librarian and patron point of view sections.

### Book Details

Clicking on the book details button instead of the login button will result in opening a frame in which there will be two methods: viewing the entire catalog and searching for a particular book’s details. Neither of these methods require logging in, as it only displays information about the catalog and does not manipulate any of the data. I thought this was a analogous to briefly skimming details in a library in real life.

#### *library.Book\_Details\_Frame.book\_details*

The book\_details method will take an isbn\_input and search the book\_store to retrieve the book’s details. The details include the title, author, ISBN, quantity that the library owns, and the quantity available.

#### *library.Book\_Details\_Frame.list\_books*

The list\_books method goes through the book\_store and lists every book’s title, ISBN, author, and the number of copies available for checkout.

# Patron Point of View

## Checking Books Out

The first function I want to discuss is the way patrons can check out books. Through the GUI tool, they can enter the ISBN of the book they are interested in checking out into the designated text box, then click on the button to activate the checkout process. The GUI code can be seen in the library.py module under the Checkout\_Frame class. The process itself is processed in the kiosk.py module.

### kiosk.Kiosk.checkout\_item

The checkout\_item method checks if the user is indeed a patron, as the patron is the one who has permission to checkout books. This is done by the check\_permissions method in the same class. A lack of permission will throw an exception. If permission is granted, then a new transaction receipt will be generated. The receipt will contain the patron’s ID, book’s ISBN, as well as the date and time of checkout. There will also be a custom transaction class which will additionally store the return date, due date, and fines. The standard fine I have set up is $1 per week late after the due date, which is 3 weeks after checkout date.

## Returning Books

# Librarian Point of View

## Creating New Patron Accounts

## Deleting Patron Accounts

## Creating New Librarian Accounts

## Deleting Librarian Accounts

## Adding New Books to the Library

## Removing Books from the Library

## Generating Reports

# Findings

## Challenges During Implementation

## Limitations of Project

One limitation is the adjustment of fines and due dates. Renewal processes were not programmed. Also, the limit of the fines for a book generally approaches the cost of purchase which I did not program either.

## Areas for Improvement