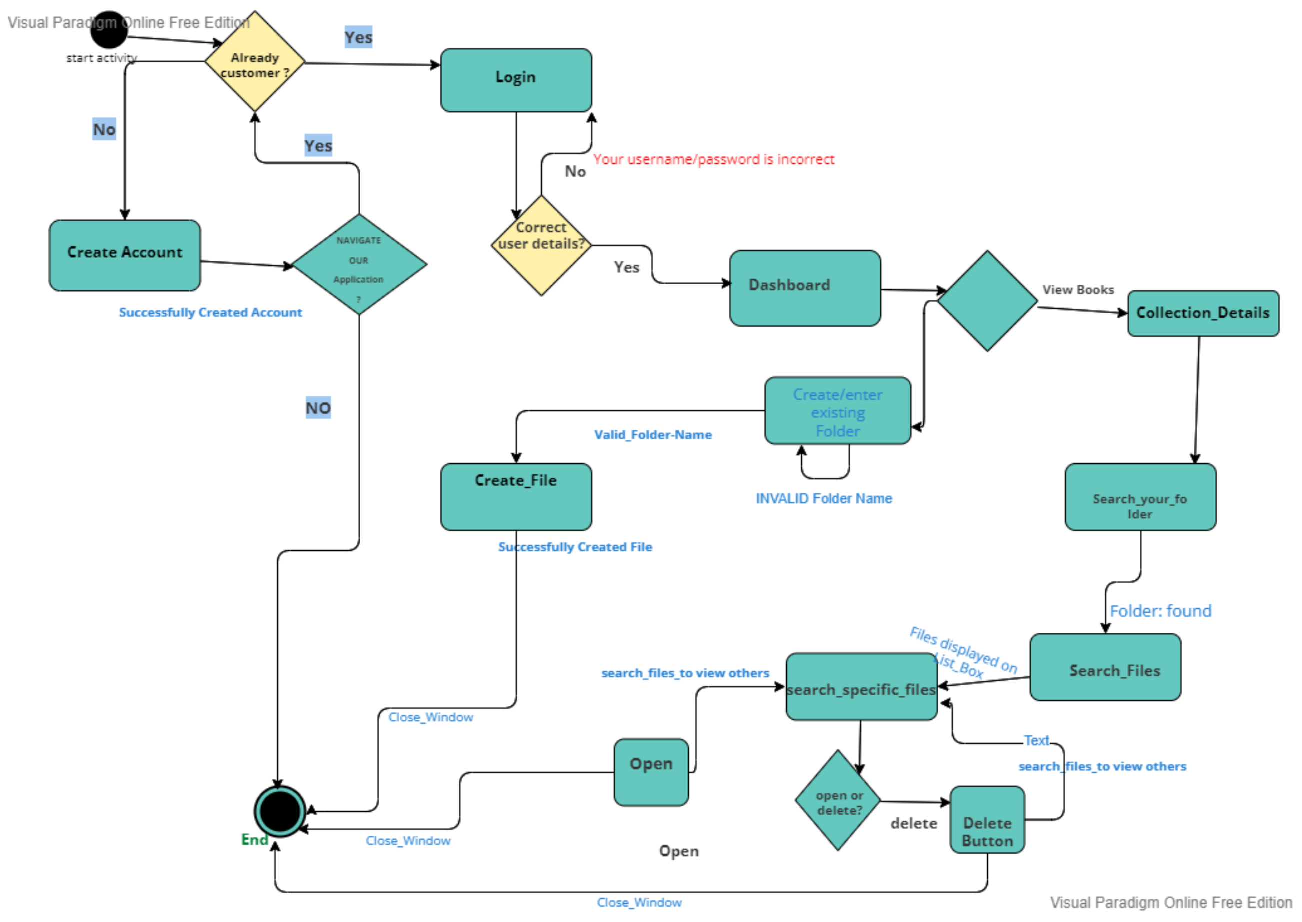
# Algorithm and Programming-Project Documentation

## Brief Description

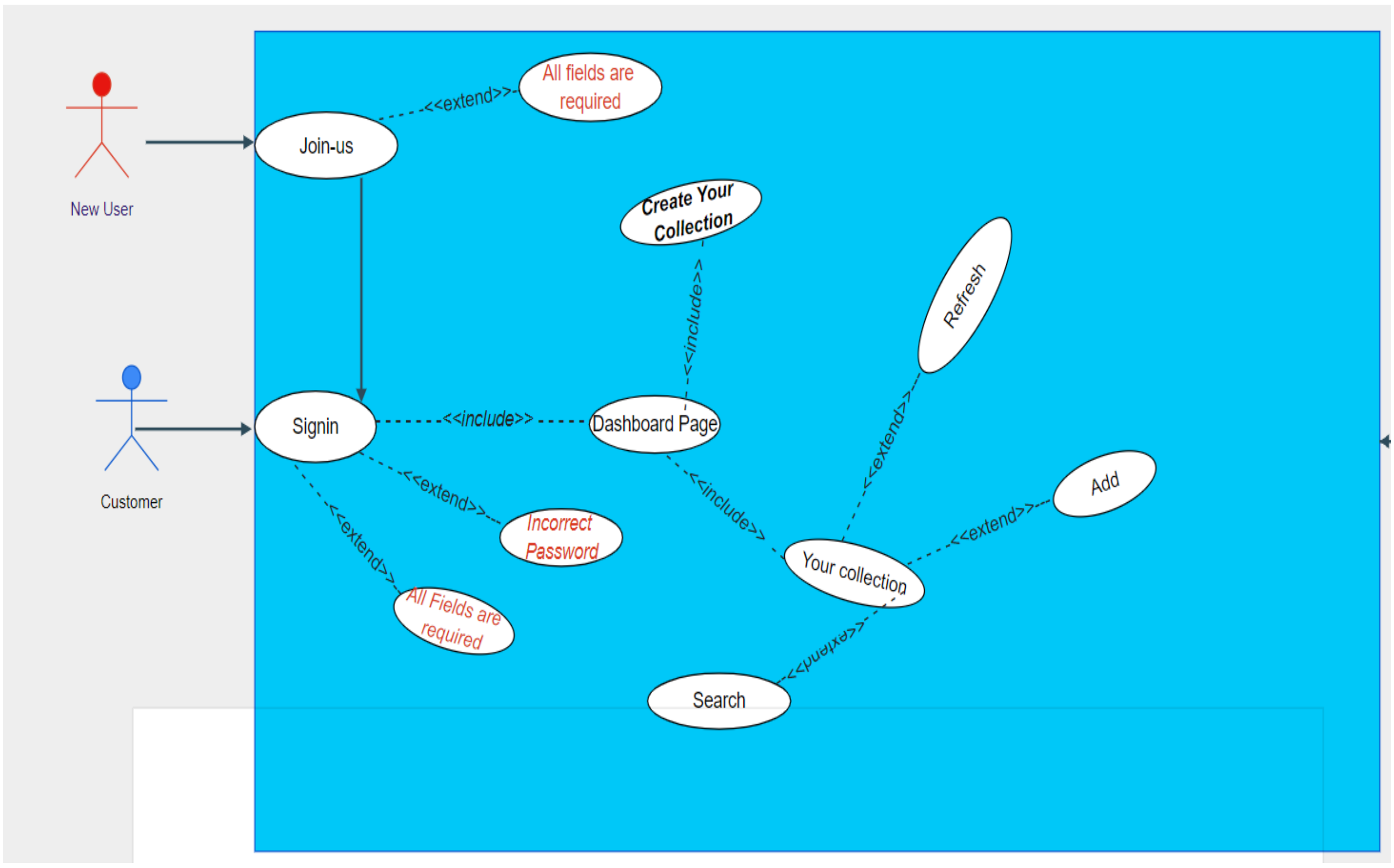
The purpose of my project was to create an application that allows users to organize their books with it. This is not the same as a library, as users can create their own collection of books and manage their collection within the application. Also, this application was not designed to have an external management system(administration) and its sole purpose is for the users to manage their collections.

This application was built with python and uses Graphical User Interface via Tkinter. It makes use of six modules(files) and classes including the main module. As a user friendly application, this program allows users to create an account that can be accessed only by the owner. After logging into their account, users will land on the dashboard page which shows a lot of options that lets users manage their account. Users can create a new collection right away in which later they can read their book collection and even can decide to remove the book they want from their collection by easily searching for it in the search box and clicking delete.

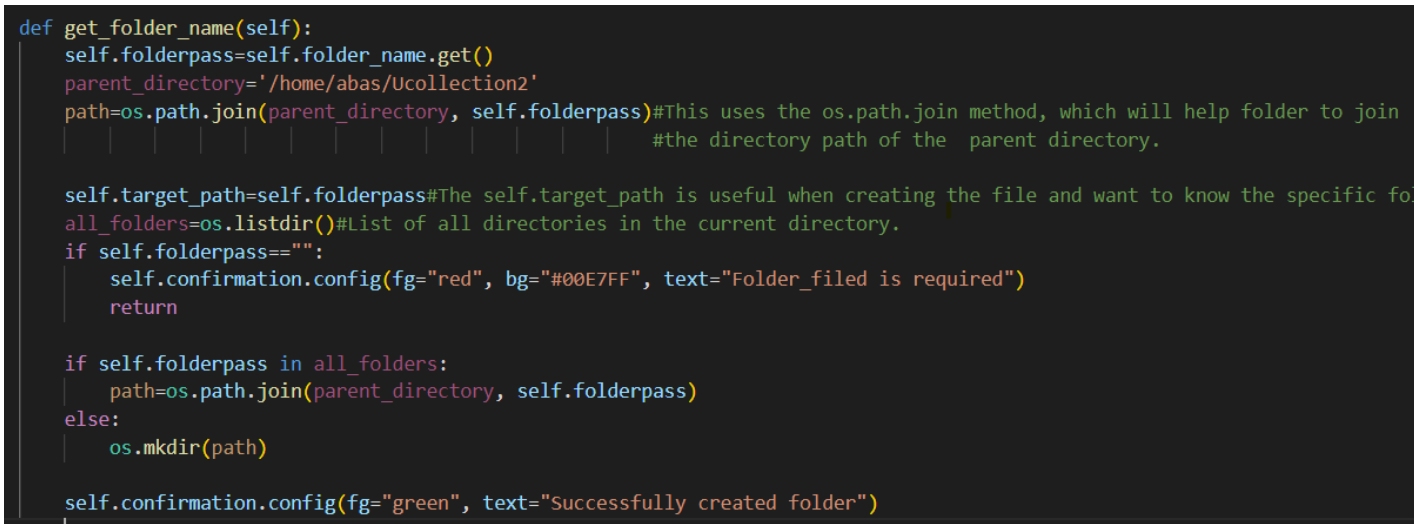
## Activity Diagram



## Use-case Diagram



## Essential Algorithms



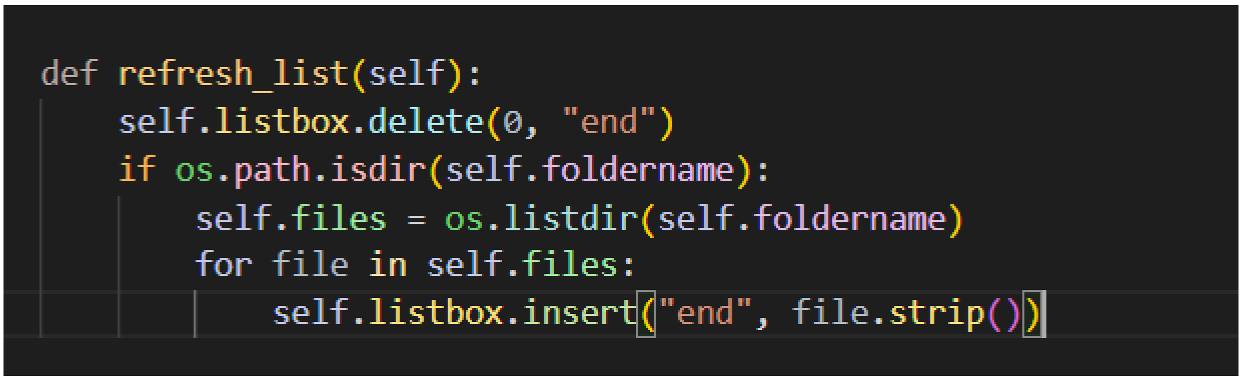
```
def get_folder_name(self):
    self.folderpass=self.folder_name.get()
    parent_directory='/home/abas/Ucollection2'
    path=os.path.join(parent_directory, self.folderpass)#This uses the os.path.join method, which will help folder to join
                                                        #the directory path of the  parent directory.

    self.target_path=self.folderpass#The self.target_path is useful when creating the file and want to know the specific fo
    all_folders=os.listdir()#List of all directories in the current directory.
    if self.folderpass=="":
        self.confirmation.config(fg="red", bg="#00E7FF", text="Folder_filed is required")
        return

    if self.folderpass in all_folders:
        path=os.path.join(parent_directory, self.folderpass)
    else:
        os.mkdir(path)

    self.confirmation.config(fg="green", text="Successfully created folder")
```

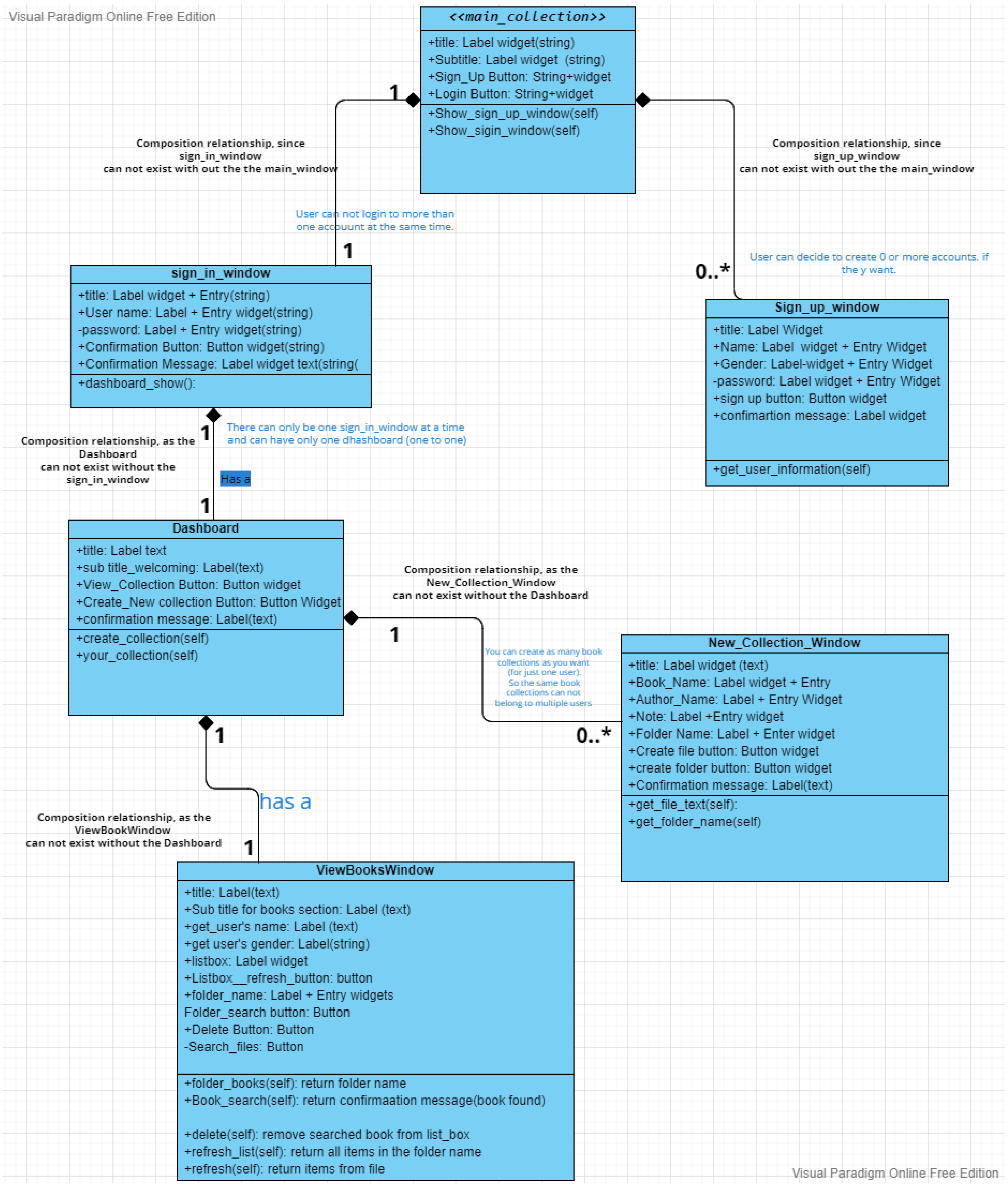
Although the function of the above code does something as simple as creating a folder, it really involves interesting behind-the-scenes work. It utilizes, Os python module methods such as `os.path.join` which takes parent_directory, and the specified folder and gets them to join the current directory path. And if the folder name the user entered in the entry box does not exist, the program uses os.mkdir(path)and passes the directory path to the method, and it creates a new folder and uses the join method to add it to the directory path.



```python
def refresh_list(self):
    self.listbox.delete(0, "end")
    if os.path.isdir(self.foldername):
        self.files = os.listdir(self.foldername)
        for file in self.files:
            self.listbox.insert("end", file.strip())
```

The code above is about a function that refreshes files in a folder and displays them on a window. It first clears off everything in the window, and then the program uses OS method `os.path.isdir` and passes the name of the folder, and the method checks the folder name with its path and returns if the path directory exists. The previous function put the folder name in the right directory path, it should return it if the folder name is correct. Next, the program will check if the folder name is in the list directory: first if the directory path is correct then, if the folder itself is in the directory list. And then, the program loops through the items in the  specified folder name, and the Listbox widget uses the insert  method to display the list of files in the specified directory. So, what seems simple, creating a folder and opening it and printing items in the file go through such complex steps behind the scenes.

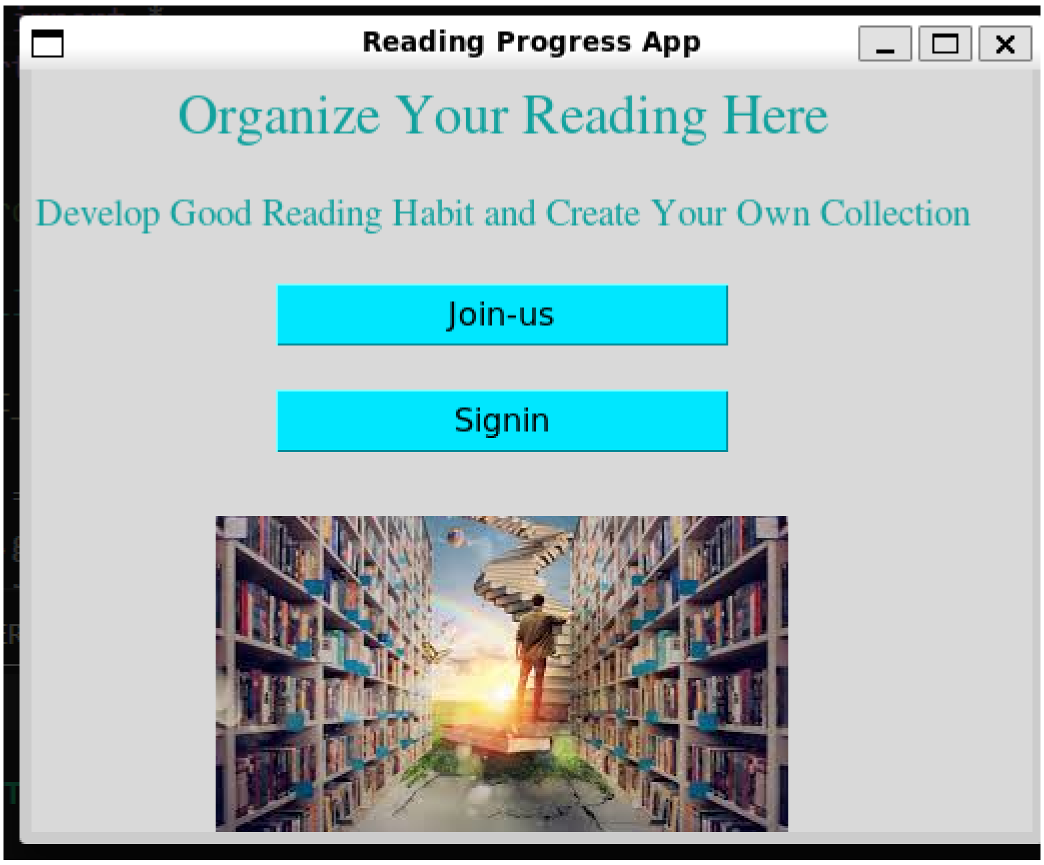
## Class Diagram


Visual Paradigm Online Free Edition
<<main_collection>>
+title: Label widget(string)
+Subtitle: Label widget (string)
+Sign_Up Button: String+widget
+Login Button: String+widget
+Show_sign_up_window(self)
+Show_sigin_window(self)
1
Composition relationship, since sign_in_window can not exist with out the the main_window
Composition relationship, since sign_up_window can not exist with out the the main_window
User can not login to more than one accouunt at the same time.
1
0..*
User can decide to create 0 or more accounts. if the y want.
sign_in_window
+title: Label widget + Entry(string)
+User name: Label + Entry widget(string)
-password: Label + Entry widget(string)
+Confirmation Button: Button widget(string)
+Confirmation Message: Label widget text(string(
+dashboard_show():
Sign_up_window
+title: Label Widget
+Name: Label widget + Entry Widget
+Gender: Label-widget + Entry Widget
-password: Label widget + Entry Widget
+sign up button: Button widget
+confimartion message: Label widget
+get_user_information(self)
1
There can only be one sign_in_window at a time and can have only one dhashboard (one to one)
Composition relationship, as the Dashboard can not exist without the sign_in_window
Has a
1
Dashboard
+title: Label text
+sub title_welcoming: Label(text)
+View_Collection Button: Button widget
+Create_New collection Button: Button Widget
+confirmation message: Label(text)
+create_collection(self)
+your_collection(self)
Composition relationship, as the New_Collection_Window can not exist without the Dashboard
1
You can create as many book collections as you want (for just one user). So the same book collections can not belong to multiple users
0..*
New_Collection_Window
+title: Label widget (text)
+Book_Name: Label widget + Entry
+Author_Name: Label + Entry Widget
+Note: Label +Entry widget
+Folder Name: Label + Enter widget
+Create file button: Button widget
+create folder button: Button widget
+Confirmation message: Label(text)
+get_file_text(self):
+get_folder_name(self)
1
has a
Composition relationship, as the ViewBookWindow can not exist without the Dashboard
1
ViewBooksWindow
+title: Label(text)
+Sub title for books section: Label (text)
+get_user's name: Label (text)
+get user's gender: Label(string)
+listbox: Label widget
+Listbox__refresh_button: button
+folder_name: Label + Entry widgets
Folder_search button: Button
+Delete Button: Button
-Search_files: Button
+folder_books(self): return folder name
+Book_search(self): return confirmaation message(book found)
+delete(self): remove searched book from list_box
+refresh_list(self): return all items in the folder name
+refresh(self): return items from file
Visual Paradigm Online Free Edition

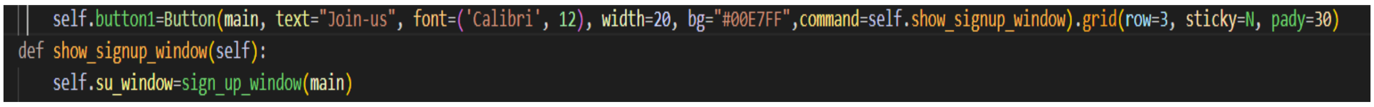
## Modules (list down and short description)

1. Driver.py

Is the main module that contains the main window of the application as well as the mainloop() which runs the event loop of the main window and the other child windows(events such as buttons).
In the Driver module, we have the parent class main_collection that has the parameter of tk module and Tk class(tk.Tk) which the main_collection class takes as an argument. In the main_collection class, there are just two widgets as the class object: 2 Labels and 2 Buttons.There are two functions for sign_up_window and sign_in_window that are commanded by the buttons but opens windows from sign_up and sign_in classes in different modules. So, Driver.py is the main module which contains the main window as (main=Tk(), the absolute root of the application), and the rest of the classes in the other modules use the Toplevel window which is a child of the main window and controls child widgets.


Reading Progress App
Organize Your Reading Here
Develop Good Reading Habit and Create Your Own Collection
Join-us
Signin

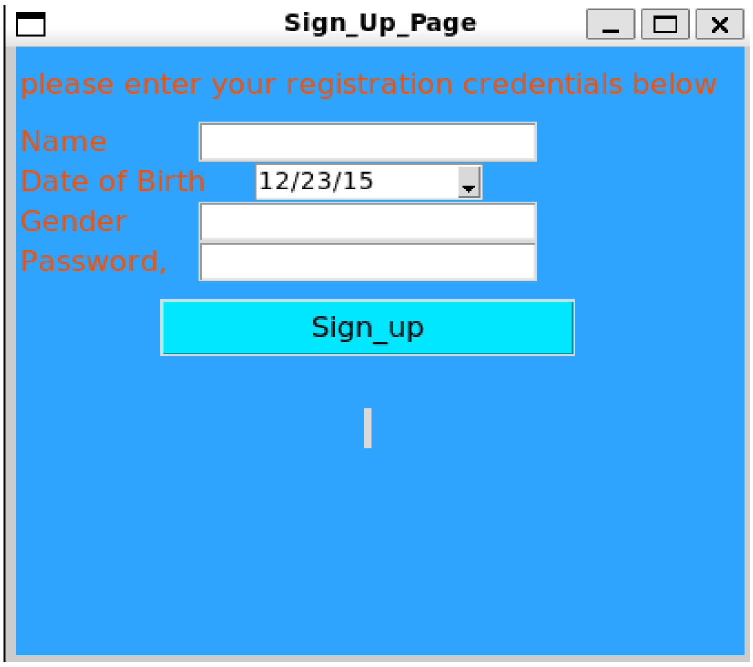
2. sign_up.py
   Is the module that hosts the sign_up window. The sign_up window is controlled by a function in the Driver module. So it is imported in the Driver module. This module is commanded by the following button and  function in the Driver module.



```
    self.button1=Button(main, text="Join-us", font=('Calibri', 12), width=20, bg="#00E7FF",command=self.show_signup_window).grid(row=3, sticky=N, pady=30)
def show_signup_window(self):
    self.su_window=sign_up_window(main)
```

And this function takes ‘main’ as an argument, and `main=Tk()` in the Driver module, which means the sign_in window is a child window of the main window.
Sign_up_window class is in the sign_up module, and takes tk.Toplevel as its parameter. The Toplevel window that’s used by the sign_up module is directly controlled by the main window in Driver.py. In the class of sign_up_window, the Toplevel is defined as `sign_up_window=Toplevel(main),`
So, the `sign_up_window(tk.Toplevel)` class contained in this module uses the following widgets: Label for just labeling and naming, Button for commanding, and Entry for inputting.
Example: The sign_up_window will prompt the user to enter the following credentials: name, date of birth, gender, and new_password. These are all Labels that include the title and the subtitle in the page.

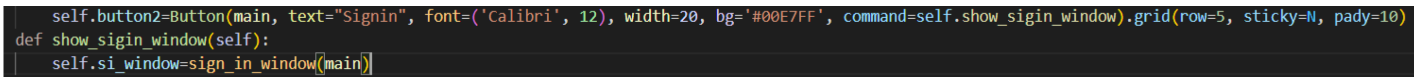
The Entry Boxes are the Entry widgets, so that the user can type in them. While the buttons are to be clicked in order to proceed and that's the Button widget.



#So, this the sign_up_page, which will be displayed if a button in the main_window is clicked

3.signin_m.py

This is the module for the sign_in page/window of this application. Like the sign_up_window, this module is commanded by the following button and function in the Driver module.



```
    self.button2=Button(main, text="Signin", font=('Calibri', 12), width=20, bg='#00E7FF', command=self.show_sigin_window).grid(row=5, sticky=N, pady=10)
def show_sigin_window(self):
    self.si_window=sign_in_window(main)
```

The function takes 'main' as an argument, and `main=Tk()` in the Driver module, which means the sign_in window is a child window of the main window.

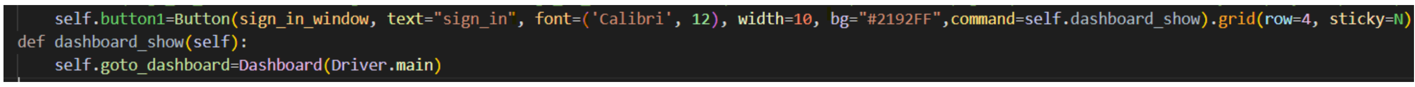
`class sign_in_window(tk.Toplevel):` Toplevel window is defined as `sign_in_window=Toplevel(main)` because the class makes use of the toplevel window as it takes `tk.Toplevel` as its parameter. When defining Toplevel in the class, the Toplevel window takes 'main' as its argument, and it is just a variable that takes in any. The sign_in class has the following widgets: 5 Labels, 2 Entry widgets, and 1 Button.

Labels: Username, Password, and notification message if fields are not filled or password/username is not correct.

Entry: Input boxes for username and password

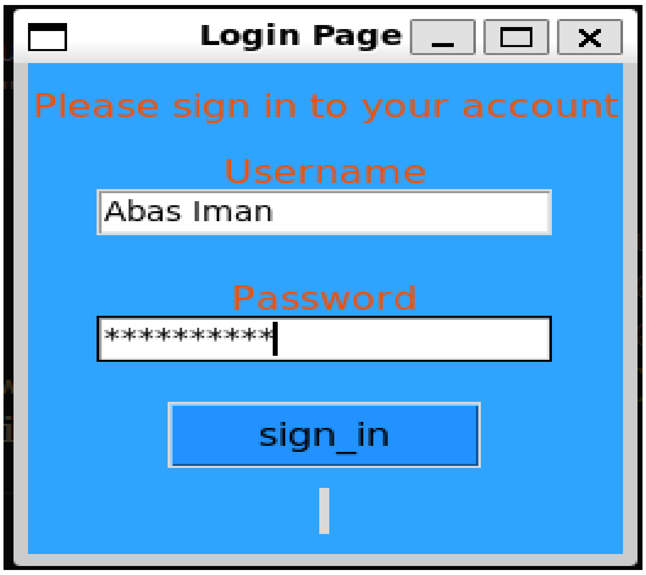
Button: one button that once is clicked will destroy the sign_in window and display the Dashboard.

The signin_window has a function as well that if the command button is clicked will open the Dashboard window:



```
    self.button1=Button(sign_in_window, text="sign_in", font=('Calibri', 12), width=10, bg="#2192FF",command=self.dashboard_show).grid(row=4, sticky=N)
def dashboard_show(self):
    self.goto_dashboard=Dashboard(Driver.main)
```

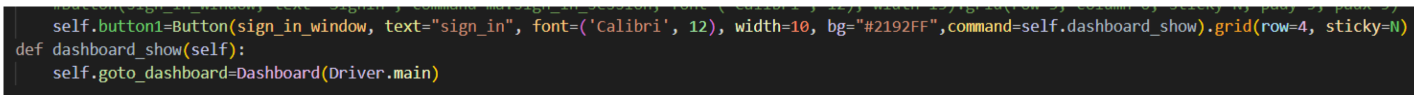
#This is the function and button widget in `class sign_in_window(tk.Toplevel)` that commands the Dashboard module, after the sign_in button is clicked. The function takes 'Driver.main' as its argument which is the main_window and because it's a sub-window of the main window.



#This is the sign_in window which will be opened when a button in the main_window(main.Tk())is clicked.

4. dashboard.py

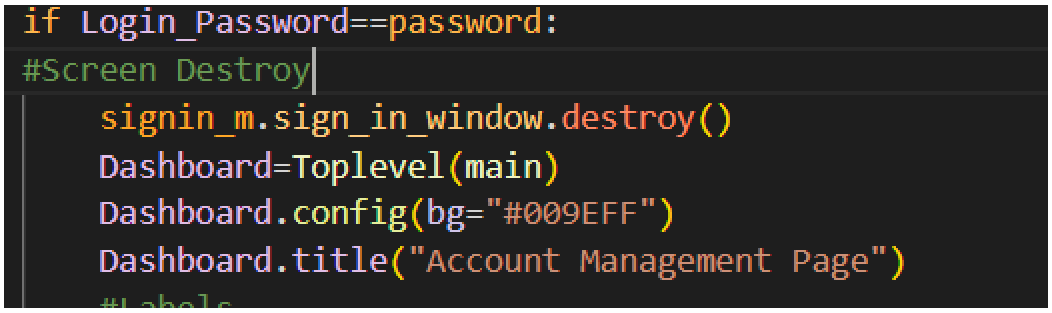
This module hosts the dashboard in this application. So, when the user enters their credentials in the sign_in page and then clicks sign_in they will land on the dashboard. This module is not directly controlled by the main_window(main=Tk()) as the following function and the button that commands the dashboard in the sign_in module.



```
    self.button1=Button(sign_in_window, text="sign_in", font=('Calibri', 12), width=10, bg="#2192FF",command=self.dashboard_show).grid(row=4, sticky=N)
def dashboard_show(self):
    self.goto_dashboard=Dashboard(Driver.main)
```

The function takes Driver.main as an argument instead of just main, and this is because we want to access the variable main=Tk()in the Driver module from the sign_up.py which is why import Driver in sign_up.py.

However, the **dashboard window** is still a sub-window of the main_window. The class Dashboard(tk.Toplevel):in this module, takes the tk.Toplevel as a parameter. And same as the previous Toplevel windows, it is defined in the class Dashboard(tk.Toplevel)as Dashboard=Toplevel(main)and takes 'main' as its argument. Once the user lands on the dashboard window,keeping the sign_in window is not necessary. So, the sign_in window disappears after sign_in is clicked. To do that, circular imports were made across the signin_m.py module and the dashboard.py. Many variables defined in the signin_m.py have to be accessed in the dashboard. So, import signin_m in the dashboard. So, this how the sign_in_window=Toplevel(main) Gets destroyed in the dashboard:



```
if Login_Password==password:
#Screen Destroy
    signin_m.sign_in_window.destroy()
    Dashboard=Toplevel(main)
    Dashboard.config(bg="#009EFF")
    Dashboard.title("Account Management Page")
```

#So when the login_passoword(a variable defined in the signin_m module) is Equal to password(value stored in open file.txt), then call the `signin_m.sign_in_window.destroy()` and the sign_in window gets destroyed before these lines

```
                Dashboard=Toplevel(main)
                Dashboard.config(bg="#009EFF")
                Dashboard.title("Account Management Page")
```

Gets executed which will open the dashboard window.

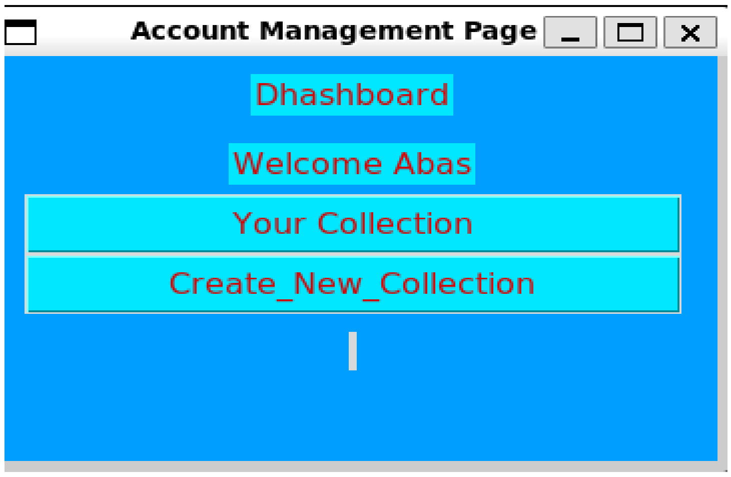
The file in which the data from the user is stored is defined as 'name', which has the value of whatever the user inputs as their name. Therefore, a Python OS module has been imported in the signup, sign_in and dashboard modules so that the program can read os.listdir() which is the list of directories of our current directory(folder that contains all the files) and get the username credentials and display on the dashboard.

The Dashboard uses the following widgets: Label, Button(view collection and create collection), and config(function to configure certain options, such as "your password is wrong",).

The dashboard module hosts two command functions:

```
    def create_collection(self):
        self.new_coll=new_collection_window(Driver.main)
    def your_collection(self):
        self.yout_list=ViewBooksWindow(Driver.main)
```

These functions and their buttons are the command for two modules: new_collection_bk.py and view_collection.py. So, they can be accessed via the Dashboard and not directly from the main_window.



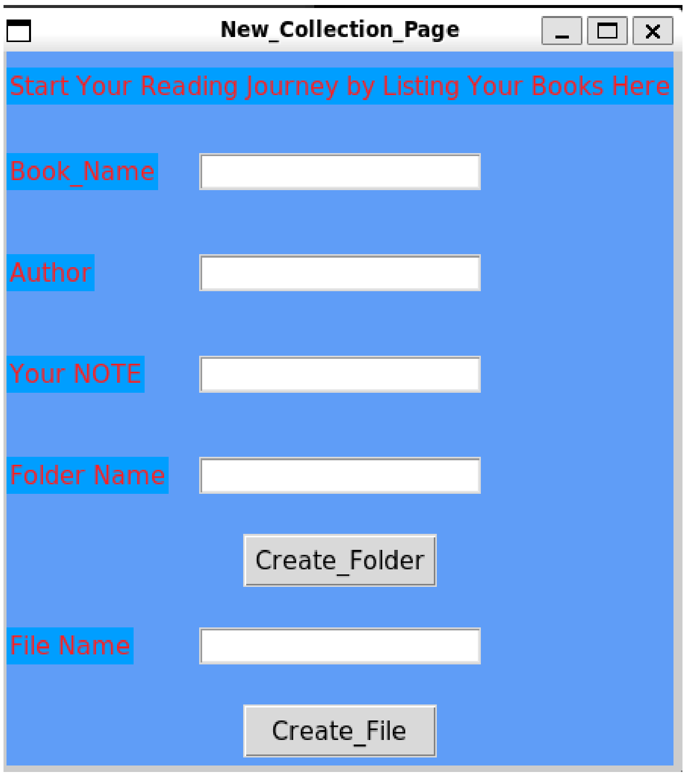
#This is the dashboard page in this application that can be accessed after sign in.

5.new_collection_bk.py

This module hosts the new_collection page where the user can create a new collection of books and put them in a folder. This page can be accessed through the Dashboard as its function and button are in the Dashboard page. Like the previous pages, the function that commands the new_collection takes (Driver.main) as an argument.

```
def create_collection(self):
    self.new_coll=new_collection_window(Driver.main)
```

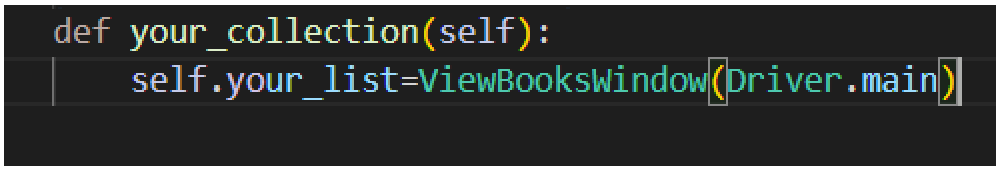
The user has to specify the name of the folder in which the books should be created as well as the file. Each book has to be in its own file, but all of them can be in one folder.



#This is the new_collection page, and the user has to put the book name, the author of the book, and a note that will describe their specific book. Afterward, the user will have to specify the Folder name(full path) and the file name.

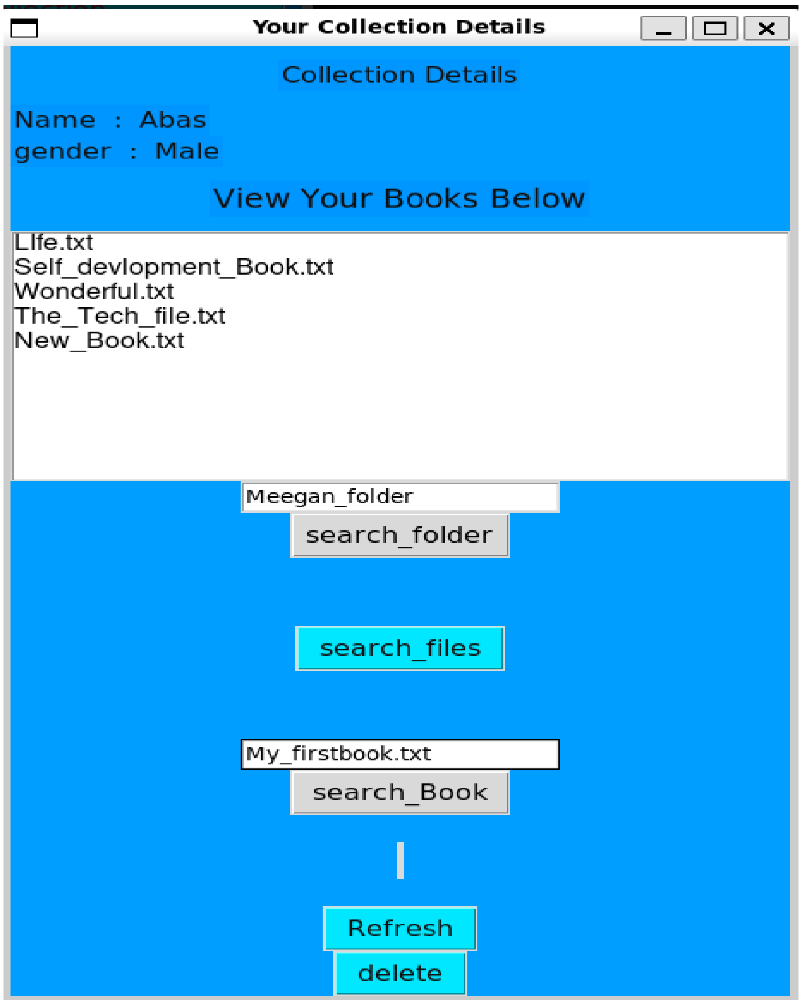
6. View_collection.py

This modulo hosts the page where the user can view a list of books, search for specific ones, and can decide whether to delete or to keep it in their collection. It can be accessed via the Dashboard and not directly from the main_window, however, it makes use of the Toplevel window and its command function takes(Driver.main) as its argument, so it is still a sub_window of the main window.



```
def your_collection(self):
    self.your_list=ViewBooksWindow(Driver.main)
```

#This is the function that manages the View_collection.py to be displayed.



This is the view collection module(window), the most complex window, as everything that has been created throughout the application has been accessed in this folder. It uses five buttons(search_folder, search_files, search_book, Refresh, and Delete).

## Reflection

This project was all about a journey of discovery and learning new material. From the day I decided to create a GUI application with Python, I knew I had to learn new materials by myself in order to make that possible. I started doing research on what to do and modules to use in order to begin working on my project. I came across Flask Python, a microweb framework, and found it a bit difficult to learn and understand about it, mainly due to the short time I had to decide what to do, how to do it, and what to use. At the same time I came across Django, also a web framework, and read about it until I discovered Tkinter, which turned out to be much simpler compared to the previous two web frameworks. I decided to go with Tkinter and then had to look up ways to store data for my application. SQLite was an option but would require me to invest substantial time to understand and be able to use it. So, I used a plain text file to store data along with an OS module so that my program interacts with my operating system and manages files saved there.

Although this was a very challenging project since I had to learn everything, it also put my ability to understand OOP in Python to the test and gave me extra knowledge regarding GUI applications using Tkinter in Python. By conducting extensive research in order to learn about Tkinter and enhance my understanding of plain text files, my research skills have improved as a result and now I know target websites for research regarding python applications.

## Notable Library

The library that I used the most throughout this application is `import os`. The os module that enabled me to access files/folders saved in my operating system.

Os methods I used include `os.listdir(), os.path.join` (helps the user's file to join their specific folder in the line of directory path). Os.path.isdir, (to check if the specified directory path exists).

## Notable Websites for reference

Pynative: https://pynative.com/python-delete-files-and-directories/
Geeksforgeeks: https://www.geeksforgeeks.org/delete-a-directory-or-file-using-python/
DigitalOcean: https://www.digitalocean.com/community/tutorials/python-os-module
freeCodeCamp: https://www.youtube.com/watch?v=YXPyB4XeYLA&t=1004s