Project name: Book & Borrow library system

Contents

Analysis

Description of the problem

I am going to develop a piece of software that will connect to a database. This software will allow users to browse books, view book details, add a book to their cart and check these books out. These will be books that would be in a library so there will be no prices for books.

There will be a section for the librarians as well. This section will allow them to view the different reservations and allow them to manage the status of the books in the database.

The database will store details of the books, details of the book orders and details of the users.

The end users of this program are:

- Elderly / Disabled people:
  - Limited mobility may prevent these people to take extended visits to their local library
- Librarians:
  - Allows them to manage the statuses of books in the system
- People who need a quick way to reserve books without hassle.

The project meets the requirements of AH computing as:

- The software will be procedurally programmed
- There will be a 2D array to hold the details of query results from the database
- A standard algorithm will be used. Specifically, a binary search algorithm
- There will be a connection to a database that will be opened and closed
- The database will contain at least one table
- SQL and DML queries will be executed with the database

Scope, boundaries and constraints

Scope

The scope of my project will include these:

- A completed wireframe design of the user interfaces with a screen navigation chart
- A completed database structure with a data dictionary and query designs
- A completed software structure showing data flow
- A completed test data table
- A working software and database system
- The results of testing using the test data table
- The evaluation of the project in a report

Boundaries

- The database will hold a maximum of 30 separate books
- Each book will only have a single author
- Each book will only have a single defining genre
- There will only be 5 genres
- There cannot be 2 users with the same username
- A user will only be able to reserve 3 books at a time
- A username will need to be at least 10 characters and less than 21 characters
- A password will need to be at least 12 characters long

Constraints

- Time Constraints:
  - I will only have 6 months to complete my whole project
  - Due to limited time, the maximum number of unique books will be 30
- Technical Constraints:
  - I will be learning new techniques for coding and new algorithms while developing
  - I will be learning how to implement a database to software connection while designing
- Legal Constraints:
  - I will need to alter names of books as to avoid copyright. As I do not have permission from every creator to use the names
  - I will also need to alter the names of the author's as I have not received permission from them or their families to use their names.
  - My system will follow GDPR (General Data Protection Regulations). Therefore, I will need to encrypt user passwords using hashing algorithms
- Economic constraints:
  - None

Requirements specification

Functional requirements

Borrowers

Inputs	Processes	Outputs

• Text boxes:
  ○ Username
  ○ Password
  ○ First name
  ○ Surname
  ○ Password confirmation
 	
• Buttons:
  ○ Book
  ○ Submit
  ○ Return
  ○ Return to catalogue
  ○ Add book to cart
  ○ Cart
  ○ Checkout
  ○ Exit
 	
• Dropdown:
  ○ Genre to filter by
	• Check for overdue books:
	  ○ Check if there are any book past their due date
	  ○ Change status to overdue if true
	
	• Validate inputted details:
	  ○ Check if username already/doesn't exist
	  ○ Check if password match. Both with the stored and with the confirmation
	  ○ Check if password is long enough
	  ○ Check if username is long enough or is too long
	
	• Generate books:
	  ○ Get all available books
	  ○ Get available books using genre filter
	
	• Book:
	  ○ Get ISBN of specific book
	  ○ Get details of ISBN
	
	• Add book to cart:
	  ○ Check if book already in cart
	  ○ Check if cart is full
	
	• Cart:
	  ○ Go to cart screen
	
	• Checkout:
	  ○ Make a new reservation
	  ○ Calculate details of user's order (Due date etc.)
	  ○ Change book statuses
	
	• Return to catalogue:
	  ○ Go to catalogue screen
		• Popups:
		  ○ If username already exists/ doesn't exist
		  ○ If passwords match
		  ○ If password isn't long enough
		  ○ If username isn't 10-20 characters
		  ○ If login was successful
		  ○ If cart is full
		  ○ If book is already in cart
		  ○ If book was successfully added to cart
		
		• Displays:
		  ○ All available books
		  ○ All available books with filter
		  ○ Specified book details
		  ○ User's current cart
		  ○ Details of user's order (Time to completion, Due date etc.)

Librarians

Inputs	Processes	Outputs

Same as borrower's
 	
• Buttons:
  ○ Browse
  ○ Manage
  ○ Reservations (To prepare, Due and Overdue)
  ○ Books in reservation
  ○ Change status
  ○ Return
	Same as borrower's
	
	• Browse:
	  ○ Go to borrower section
	
	• Manage:
	  ○ Go to librarian section
	
	• Reservations:
	  ○ Get all reservations of corresponding book status (Reserve, Taken out and Overdue)
	
	• Books in reservation:
	  ○ Get all books in the specific reservation
	
	• Change status:
	  ○ Change the book status of book in reservation.
	  ○ Reserved --> Taken out
	  ○ Taken out --> Available
	  ○ Overdue --> Available
	
	• Return:
	  ○ Take librarian to the previous screen
		Same as borrower's
		
		• Popups:
		  ○ Book browsing option chosen
		  ○ Book status changed
		
		• Displays:
		  ○ Buttons to move to each section
		  ○ Buttons for each reservation type
		  ○ All reservations with that status. (Overdue will display users instead)
		  ○ All books in the chosen reservation

End user requirements

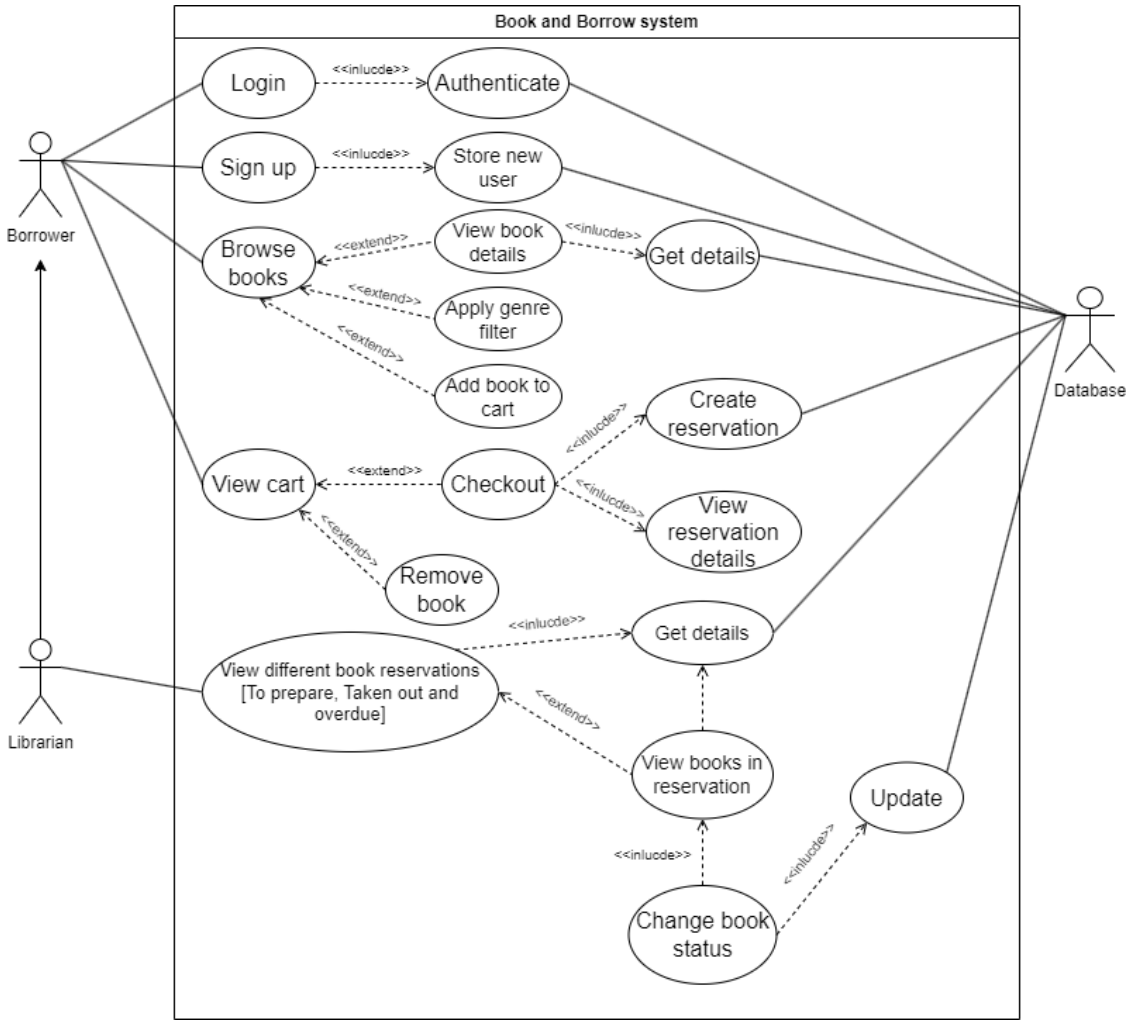
Borrowers:

- Should have a user-friendly login and sign-up process.
- Should be able to easily browse books and apply genre filters for a refined search.
- Can view details of chosen book
- Should be able to view their cart and remove any books they don't want anymore
- Can add books to their cart
- Should receive information on wait time until pickup, overdue fees and the due date of the books

Librarians:

- Should have the ability to browse or manage the books
- Should be able to view and prepare reservations, including details of specific books in each reservation.
- Should be able to check all books that have been taken out of the library and their due dates.
- Should have access to a list of users with overdue books and the respective number of overdue books they have.
- Should be able to change book statuses in the system.

UML Use Case Diagram



Project plan

		Planned		Actual		%	Show Gantt for	Planned	Show Status?		What is current week?	1
#	List of Activties	Start	Dur	Start	Dur	Done	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32					
1	Analysis	1	4	0	0	0%	
1.1	Description	1	1	0	0	0%	
1.2	Requirements	1	1	0	0	0%	
1.3	Scope, Boundaries and Constraints	2	2	0	0	0%	
1.4	Use case	3	2	0	0	0%	
2	Design	5	7	0	0	0%	
2.1	ERD	5	1	0	0	0%	
2.2	Data dictionary	5	2	0	0	0%	
2.3	Query designs	6	2	0	0	0%	
2.4	Wireframes	7	3	0	0	0%	
2.5	Pseudocode	8	4	0	0	0%	
3	Implementation	12	7	0	0	0%	
3.1	Database structure	12	2	0	0	0%	
3.2	Filling database	13	2	0	0	0%	
3.3	Borrower section	13	4	0	0	0%	
3.4	Librarian section	16	3	0	0	0%	
4	Testing	19	7	0	0	0%	
4.1	Test plan	19	3	0	0	0%	
4.2	Testing	20	3	0	0	0%	
4.3	End user tests	23	3	0	0	0%	
5	Evaluation	26	7	0	0	0%	
5.1	Fitness for purpose	26	2	0	0	0%	
5.2	Usability	27	2	0	0	0%	
5.3	Robustness	28	2	0	0	0%	
5.4	Efficiency	30	2	0	0	0%	
5.5	Maintainability	31	2	0	0	0%	

Name: Daniel Monaghan
SCN: 121667622

Design

Project design

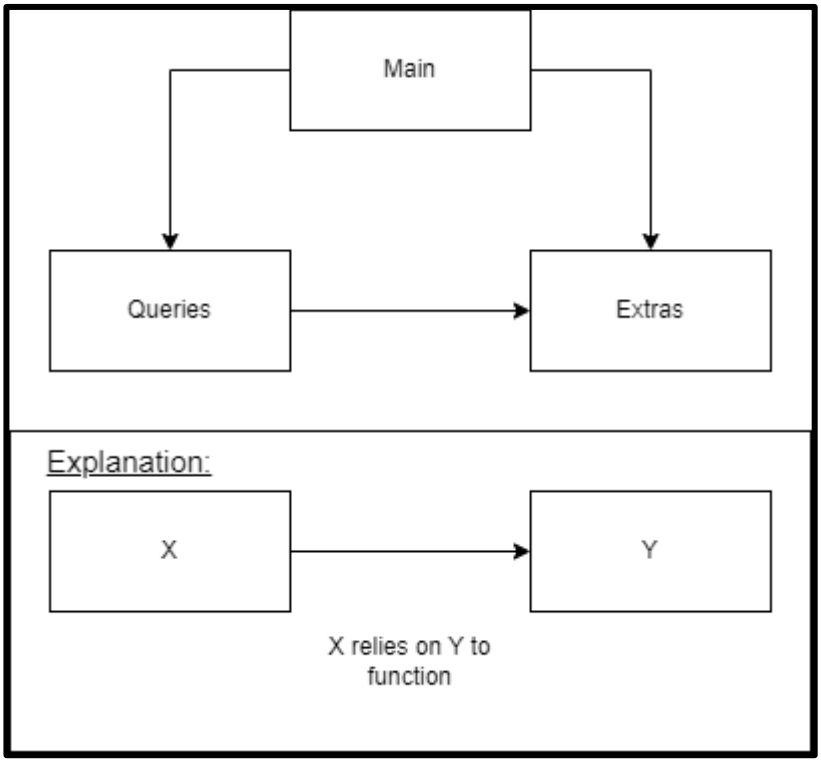
The software will be split into three files.

These are:

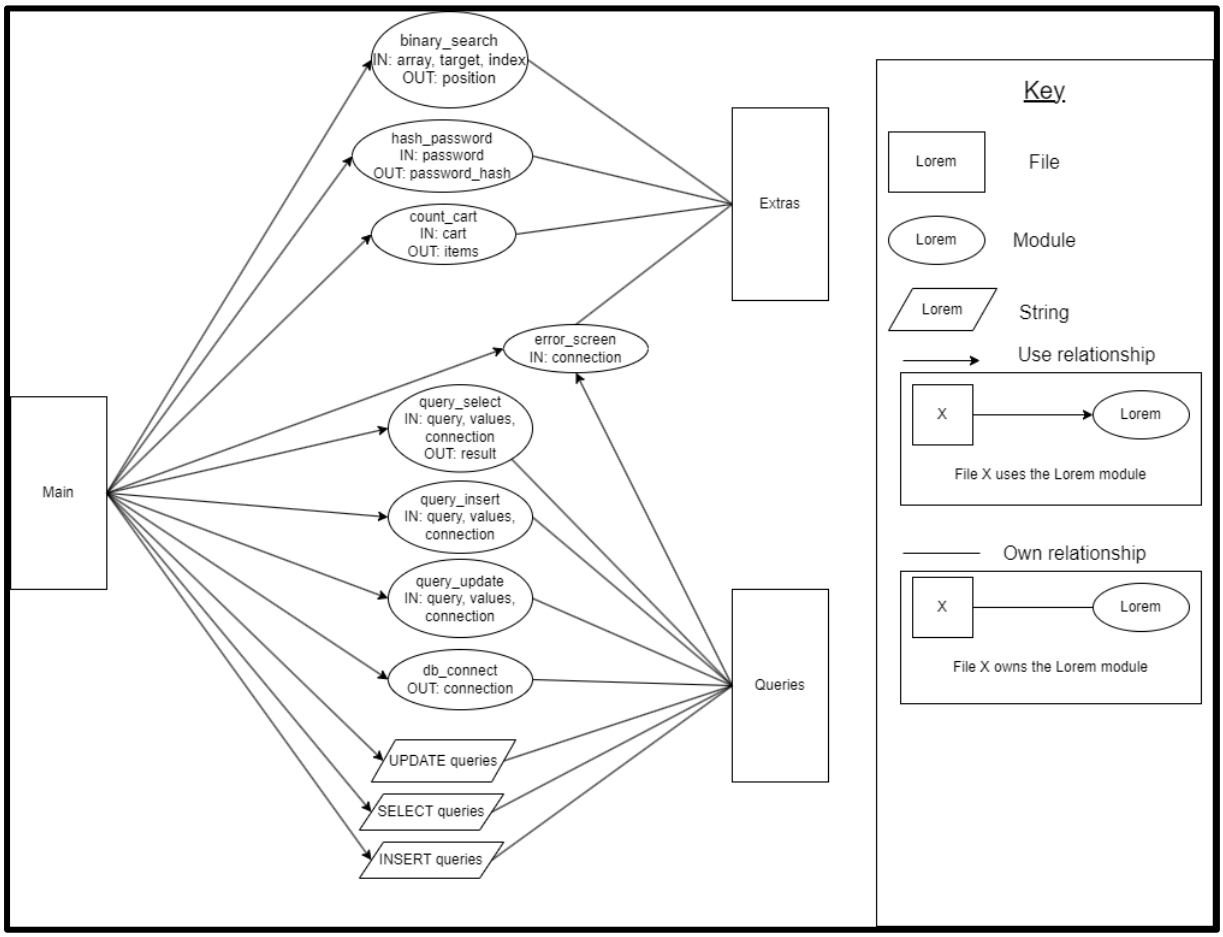
- **The main file (LM)**
  - Holds the main steps of the program. Creates all the screens on the window
- **The extras file (LE)**
  - Holds smaller function and procedures that are reused. Formatting procedures and standard algorithms. Formatting will not be included in pseudocode
- **The queries file (LQ)**
  - Holds all the SQL queries and the modules to execute them. Will also have the database connection function.
  - Query modules will access the database

The database will exist on its own. The database can only be accessed by the queries file.

File structure

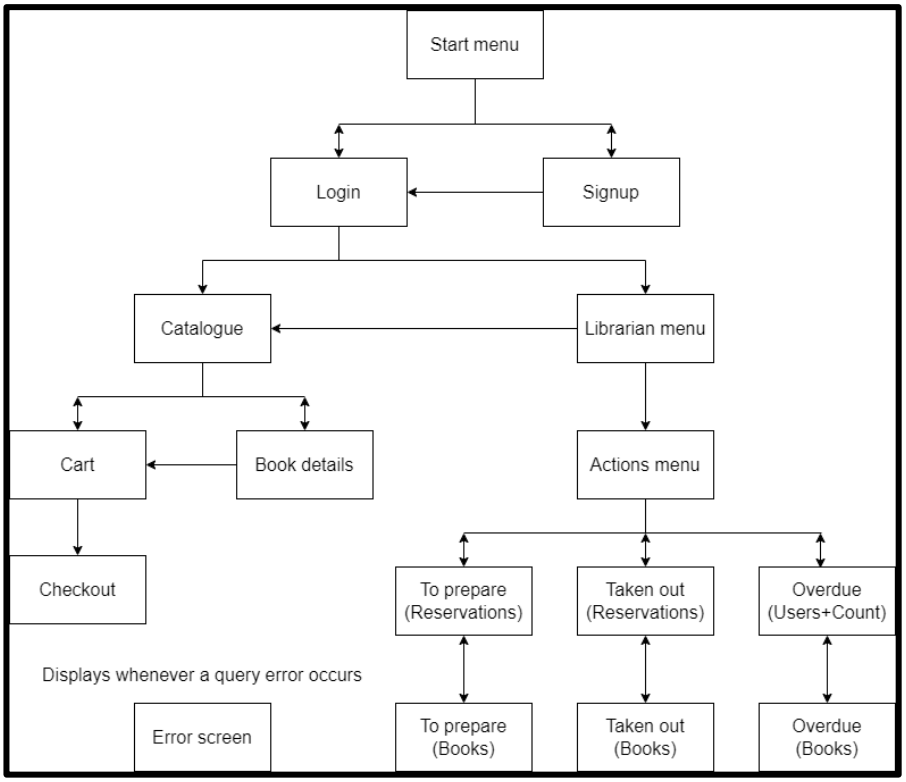


Data flow between files

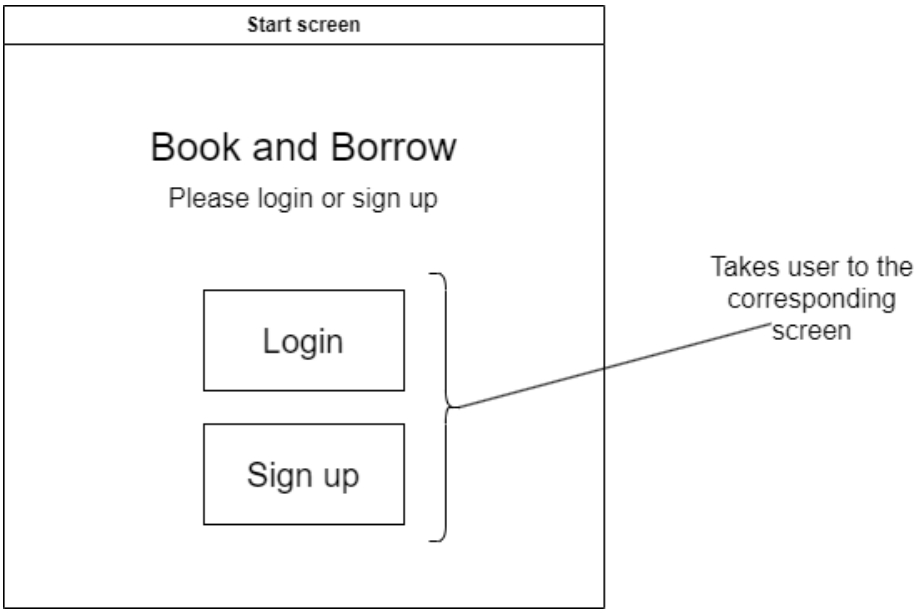


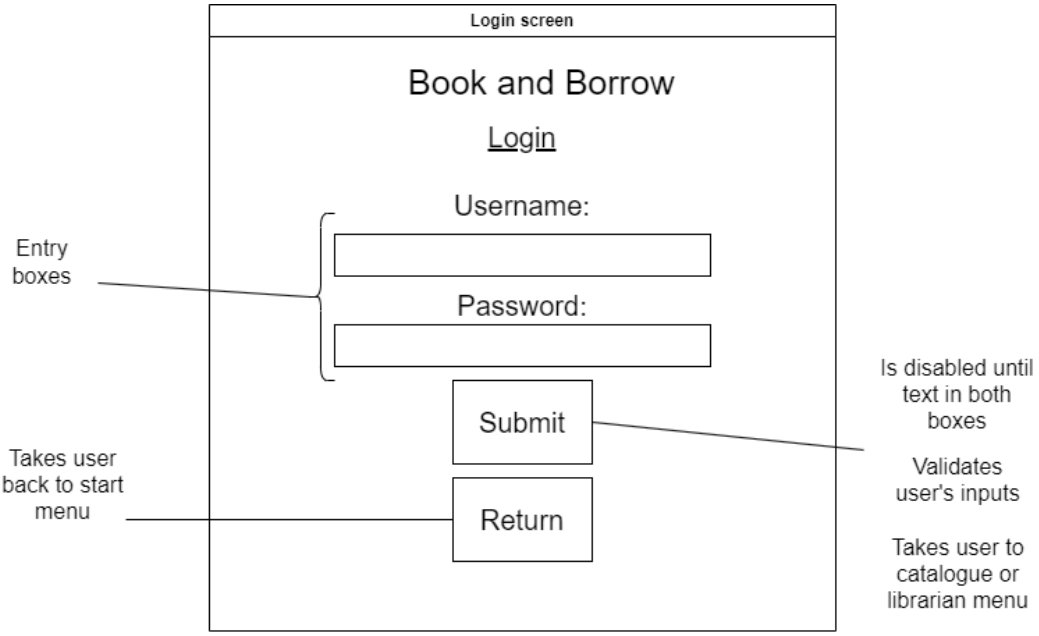
User-interface

- All screens will have a beige background.
- All buttons will have light grey background
- All fonts will be Helvetica
- Font size will be at least 15, but will go higher for titles and other text



Borrower section



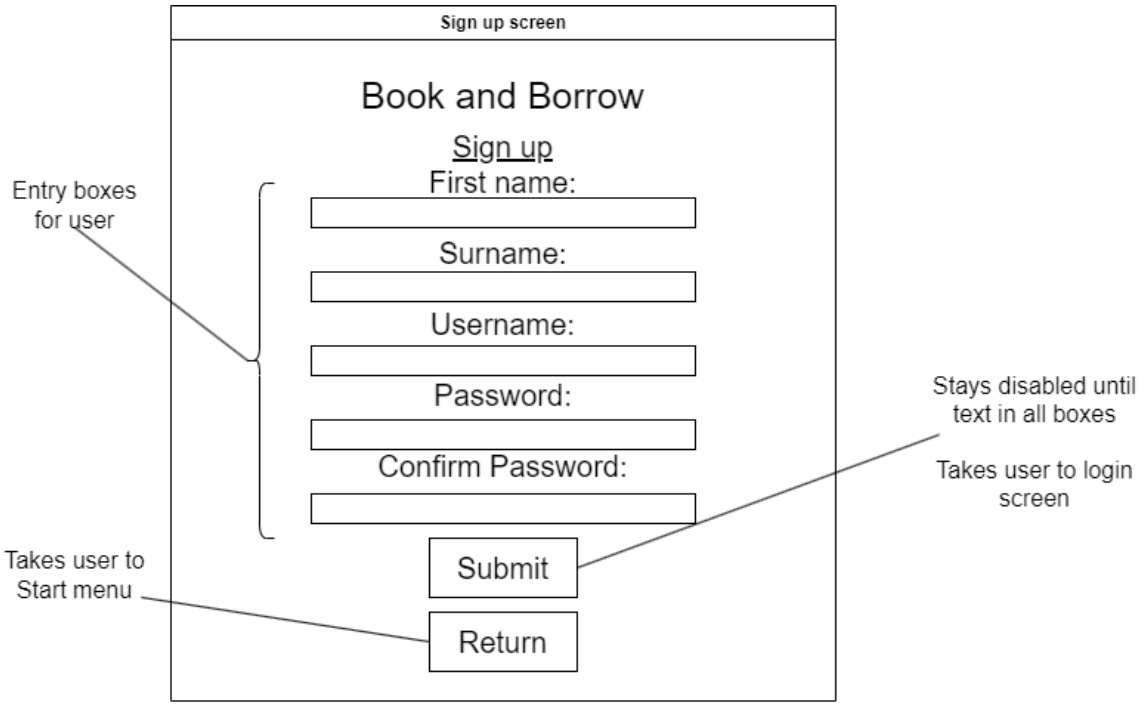


Will display popups depending on user inputs

Username doesn't exist popup

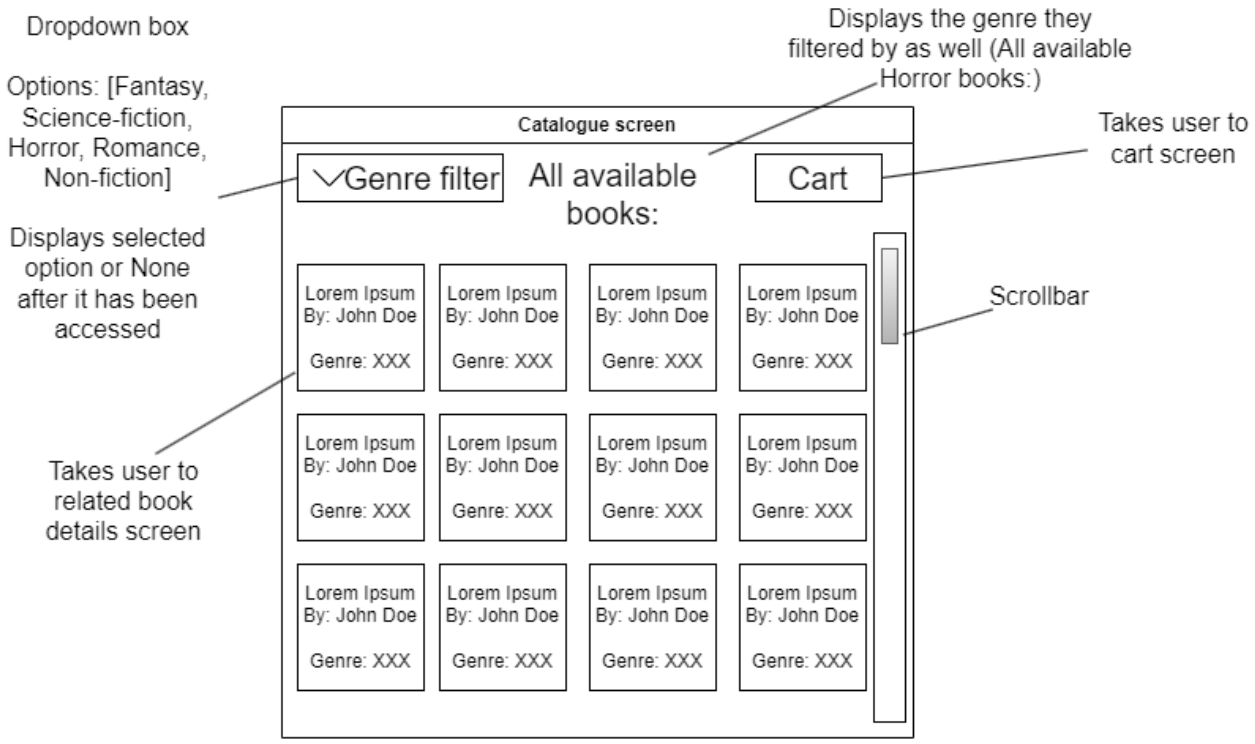
Incorrect password popup

Successful login popup
[One for borrower and another for librarian]

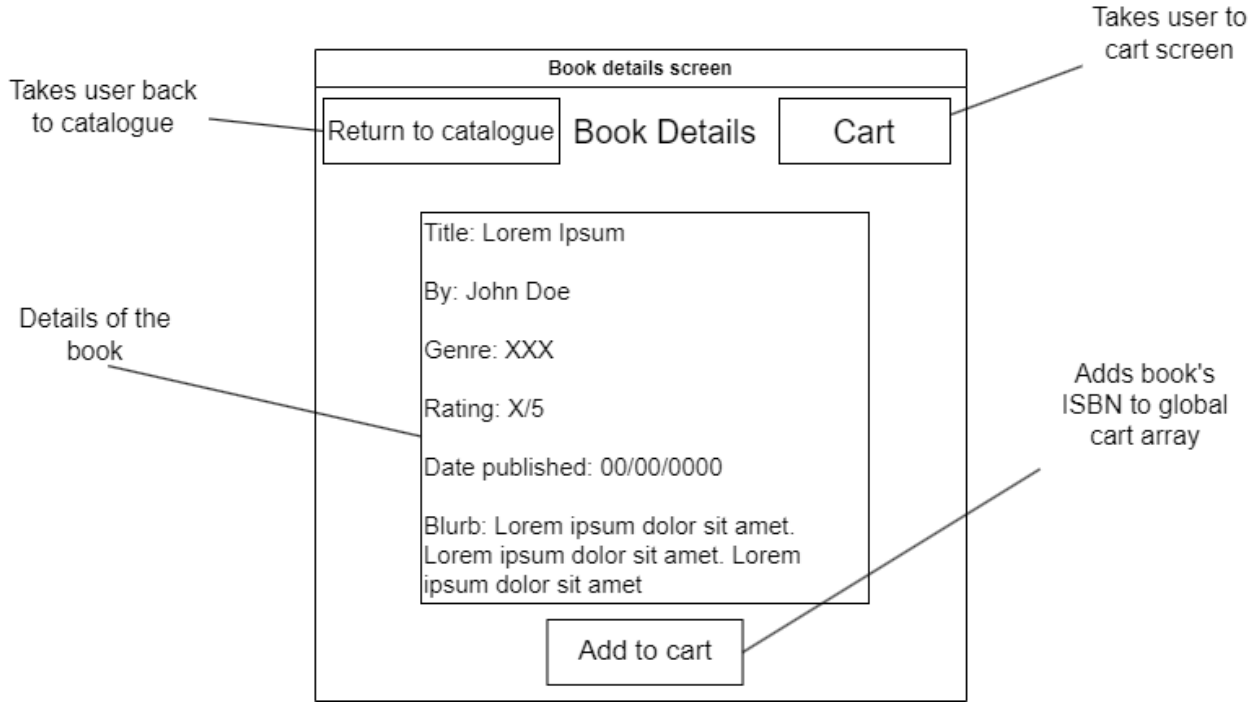


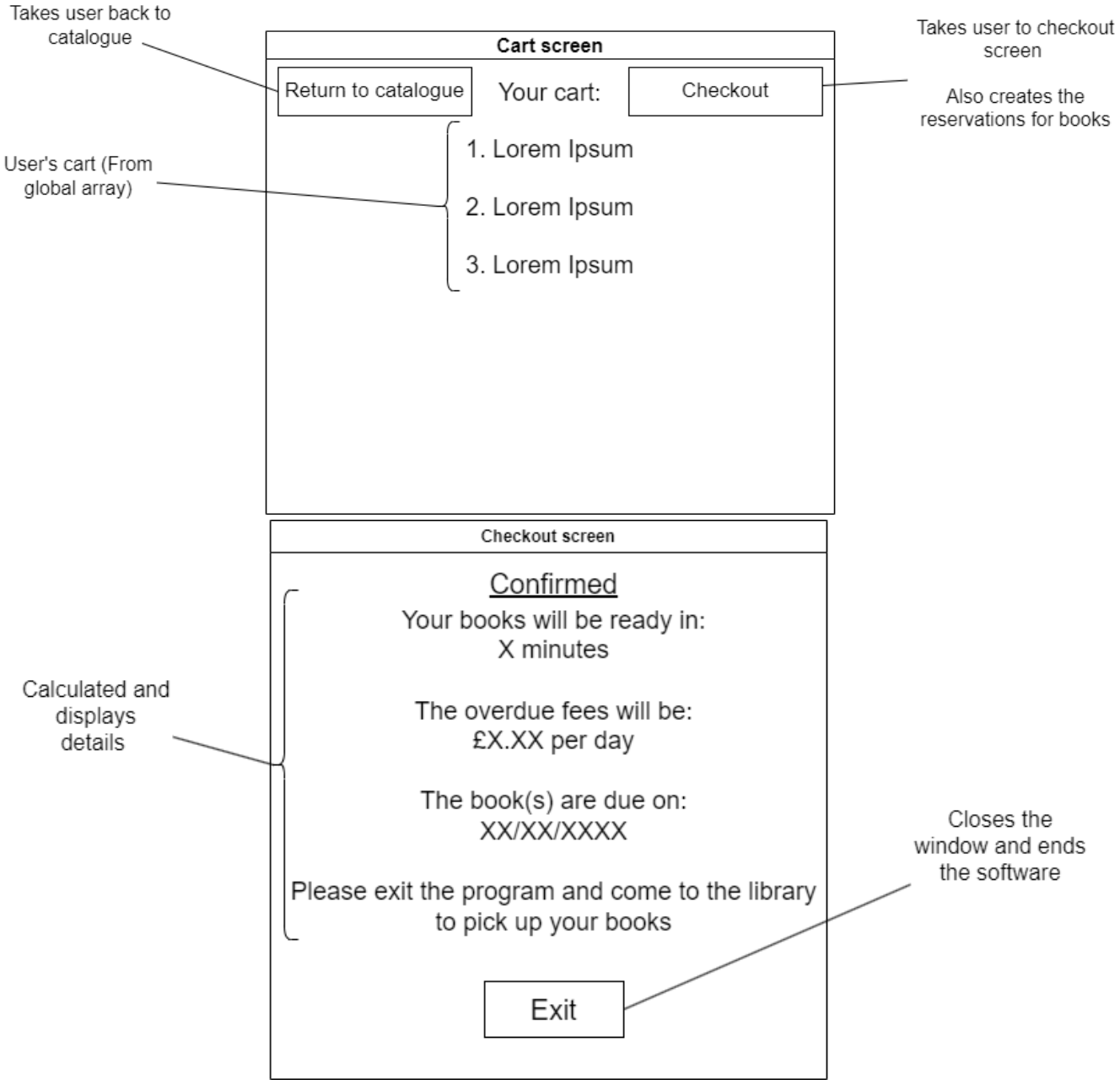
Popups will be displayed telling user:
If username exists
If passwords don't match
If username is incorrect length
If password isn't long enough

A successful creation, then telling them to login

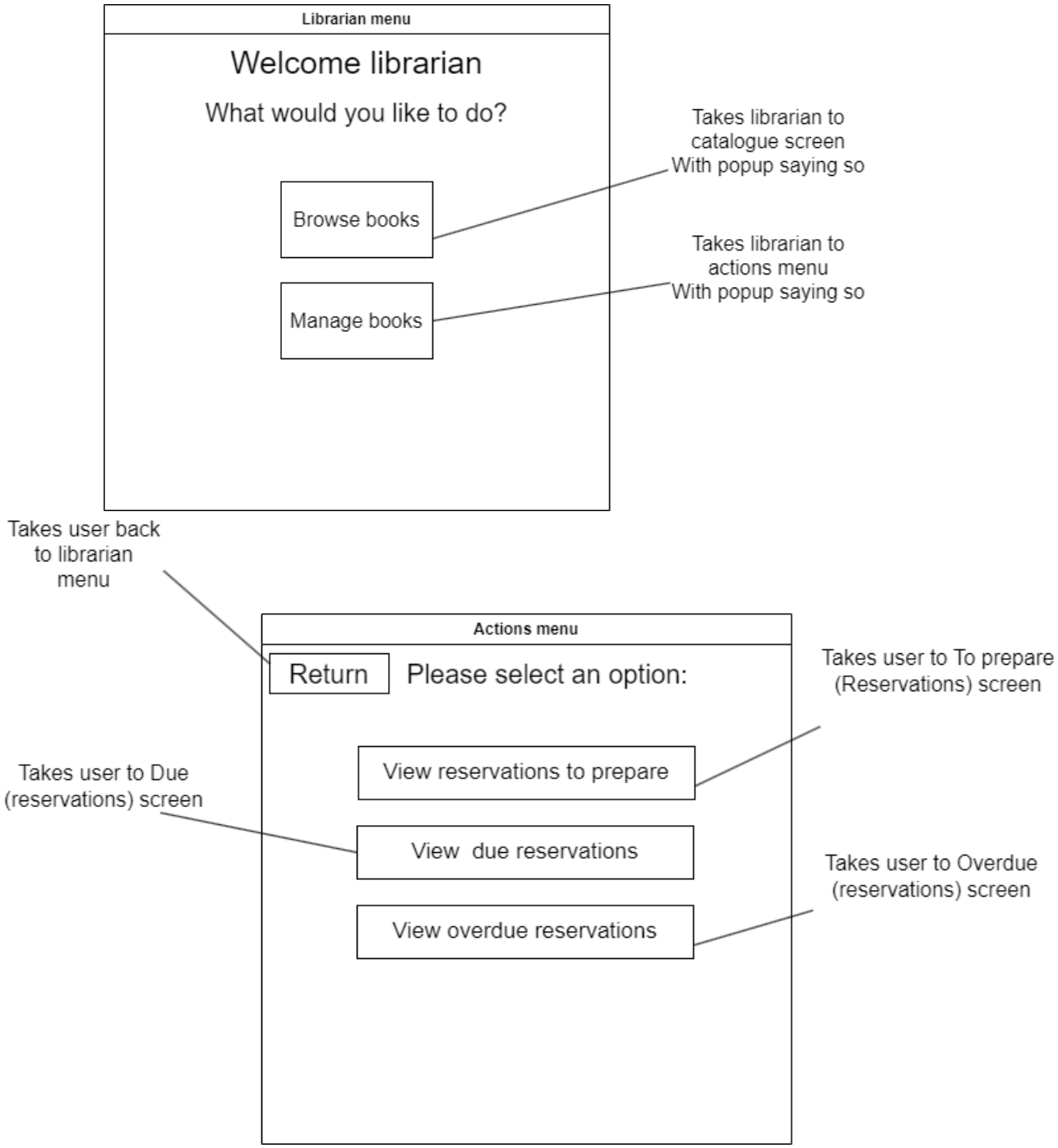


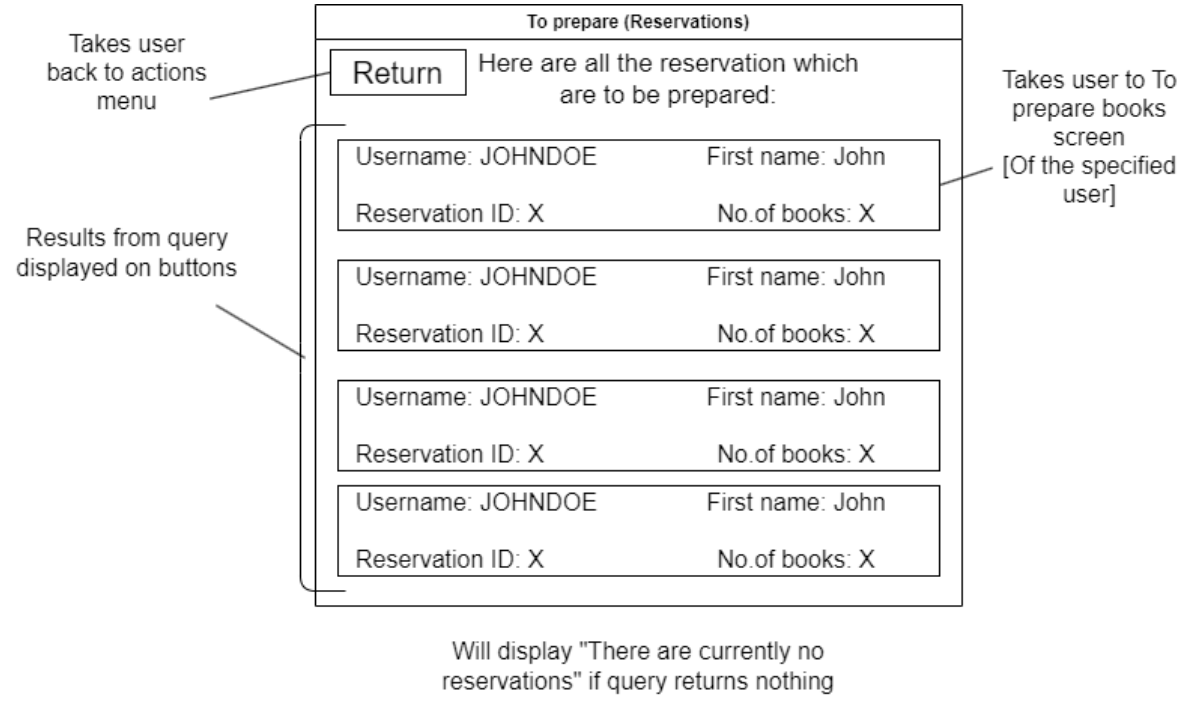
Will display "There are currently no books" if the query to fetch books returns nothing

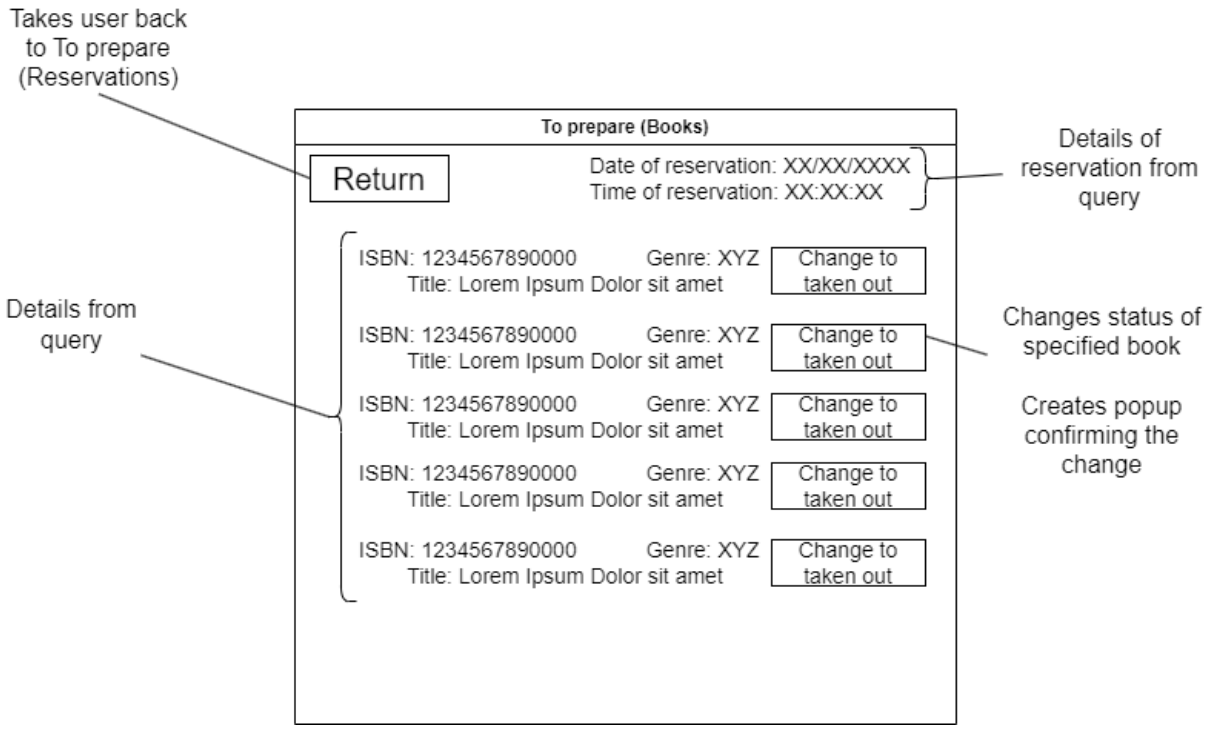


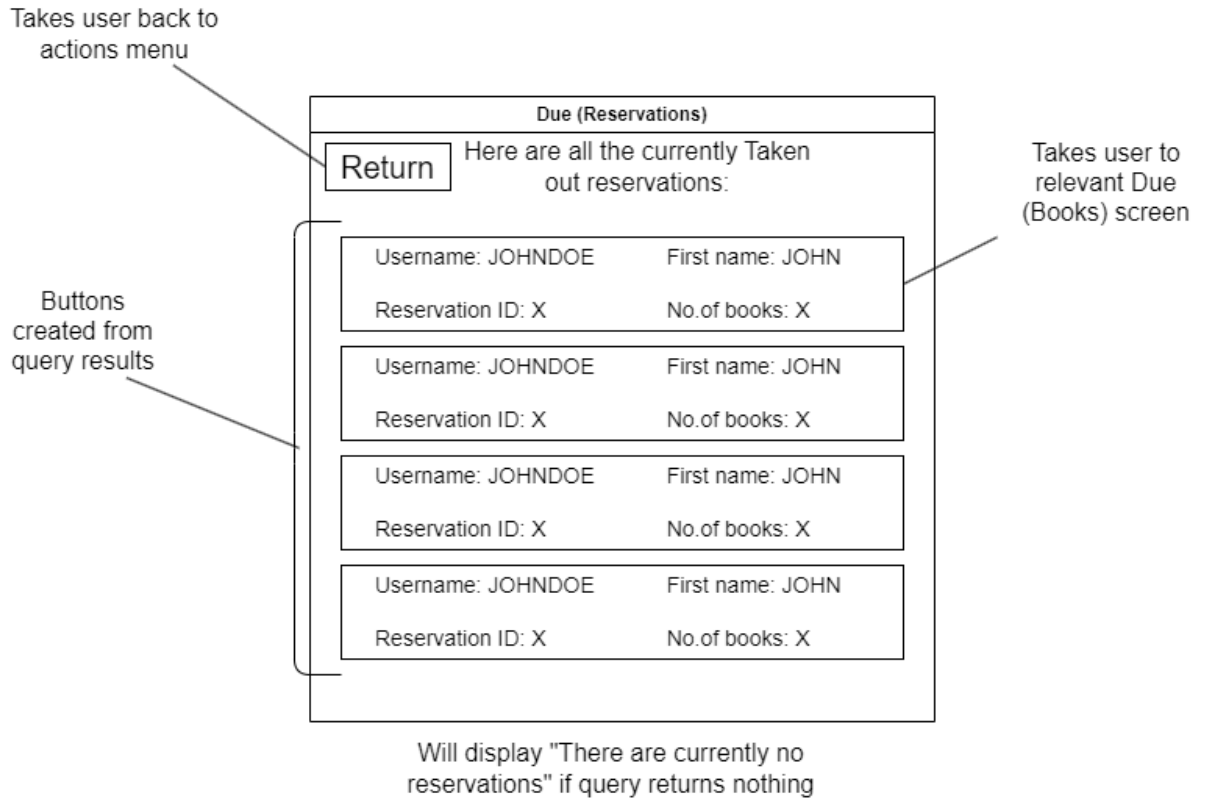

Takes user back to catalogue
Cart screen
Return to catalogue
Your cart:
Checkout
Takes user to checkout screen
Also creates the reservations for books
User's cart (From global array)
1. Lorem Ipsum
2. Lorem Ipsum
3. Lorem Ipsum
Checkout screen
Confirmed
Your books will be ready in:
X minutes
The overdue fees will be:
£X.XX per day
The book(s) are due on:
XX/XX/XXXX
Please exit the program and come to the library to pick up your books
Calculated and displays details
Exit
Closes the window and ends the software

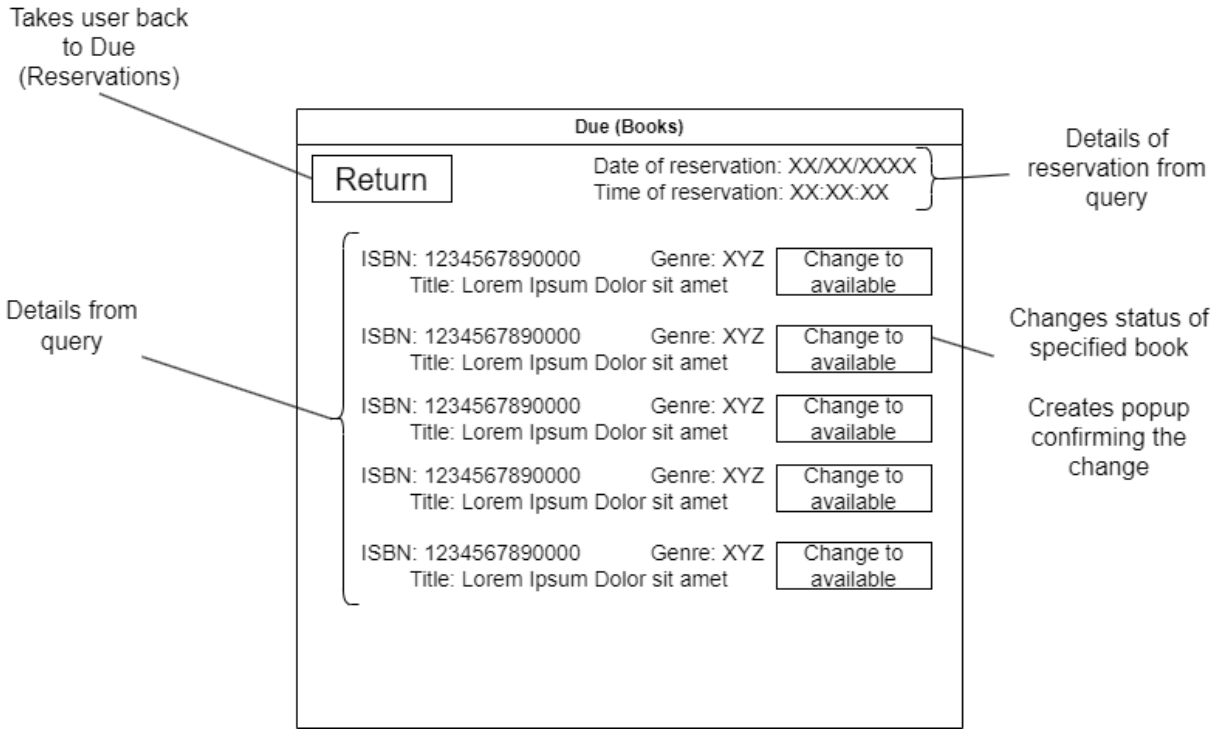
Librarian section

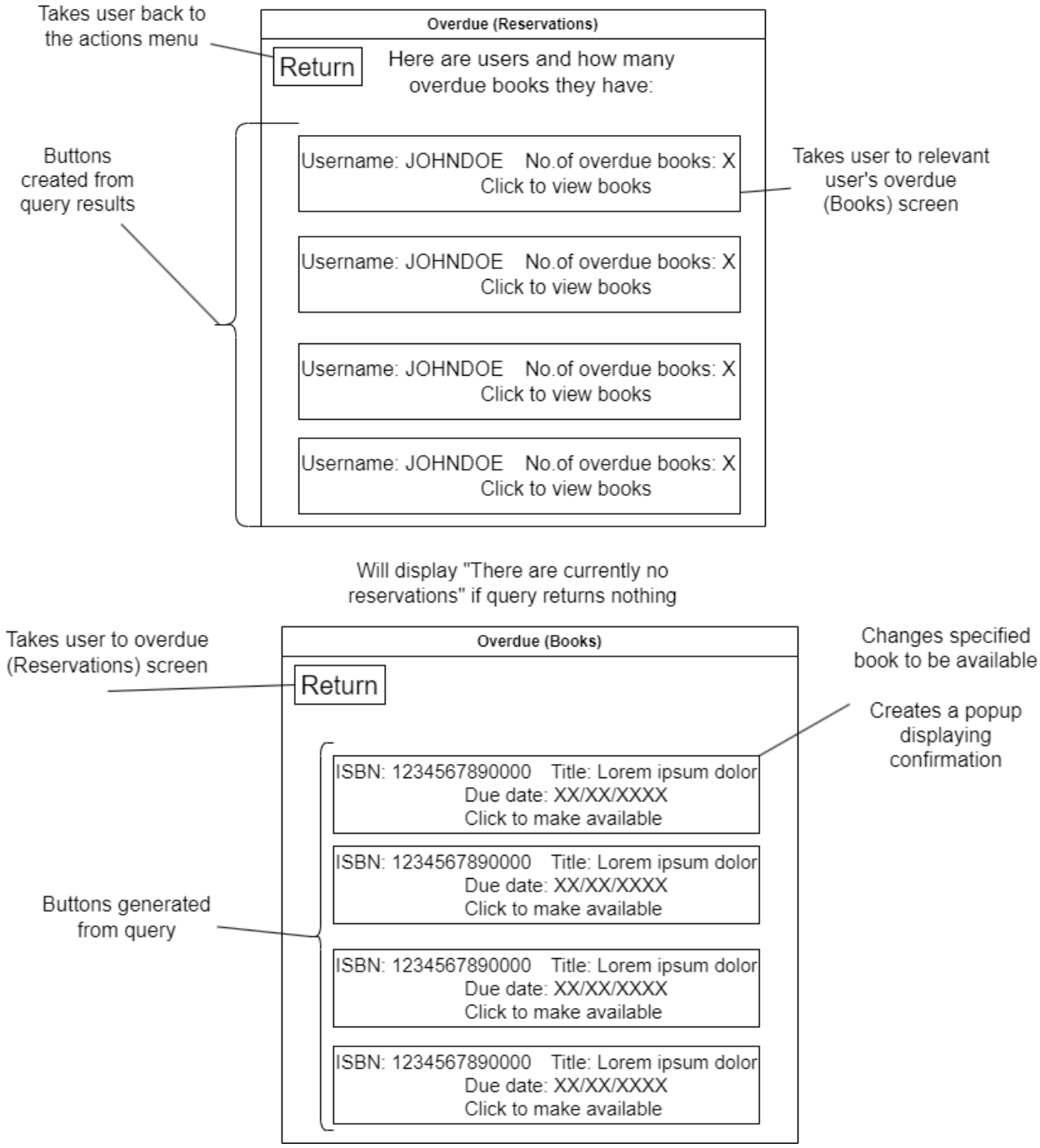



To prepare (Reservations)
Takes user back to actions menu
Return
Here are all the reservation which are to be prepared:
Takes user to To prepare books screen [Of the specified user]
Results from query displayed on buttons
Username: JOHNDOE First name: John
Reservation ID: X No.of books: X
Username: JOHNDOE First name: John
Reservation ID: X No.of books: X
Username: JOHNDOE First name: John
Reservation ID: X No.of books: X
Username: JOHNDOE First name: John
Reservation ID: X No.of books: X
Will display "There are currently no reservations" if query returns nothing


Takes user back to To prepare (Reservations)
To prepare (Books)
Return
Date of reservation: XX/XX/XXXX
Time of reservation: XX:XX:XX
Details of reservation from query
Details from query
ISBN: 1234567890000 Genre: XYZ
Title: Lorem Ipsum Dolor sit amet
Change to taken out
Changes status of specified book
Creates popup confirming the change


Takes user back to actions menu
Due (Reservations)
Return
Here are all the currently Taken out reservations:
Takes user to relevant Due (Books) screen
Buttons created from query results
Username: JOHNDOE
First name: JOHN
Reservation ID: X
No.of books: X
Will display "There are currently no reservations" if query returns nothing


Takes user back to Due (Reservations)
Due (Books)
Return
Date of reservation: XX/XX/XXXX
Time of reservation: XX:XX:XX
Details of reservation from query
Details from query
ISBN: 1234567890000
Genre: XYZ
Title: Lorem Ipsum Dolor sit amet
Change to available
Changes status of specified book
Creates popup confirming the change


Takes user back to the actions menu
Overdue (Reservations)
Return
Here are users and how many overdue books they have:
Buttons created from query results
Username: JOHNDOE No.of overdue books: X
Click to view books
Takes user to relevant user's overdue (Books) screen
Username: JOHNDOE No.of overdue books: X
Click to view books
Username: JOHNDOE No.of overdue books: X
Click to view books
Username: JOHNDOE No.of overdue books: X
Click to view books
Will display "There are currently no reservations" if query returns nothing
Takes user to overdue (Reservations) screen
Overdue (Books)
Changes specified book to be available
Return
Creates a popup displaying confirmation
ISBN: 1234567890000 Title: Lorem ipsum dolor
Due date: XX/XX/XXXX
Click to make available
Buttons generated from query
ISBN: 1234567890000 Title: Lorem ipsum dolor
Due date: XX/XX/XXXX
Click to make available
ISBN: 1234567890000 Title: Lorem ipsum dolor
Due date: XX/XX/XXXX
Click to make available
ISBN: 1234567890000 Title: Lorem ipsum dolor
Due date: XX/XX/XXXX
Click to make available

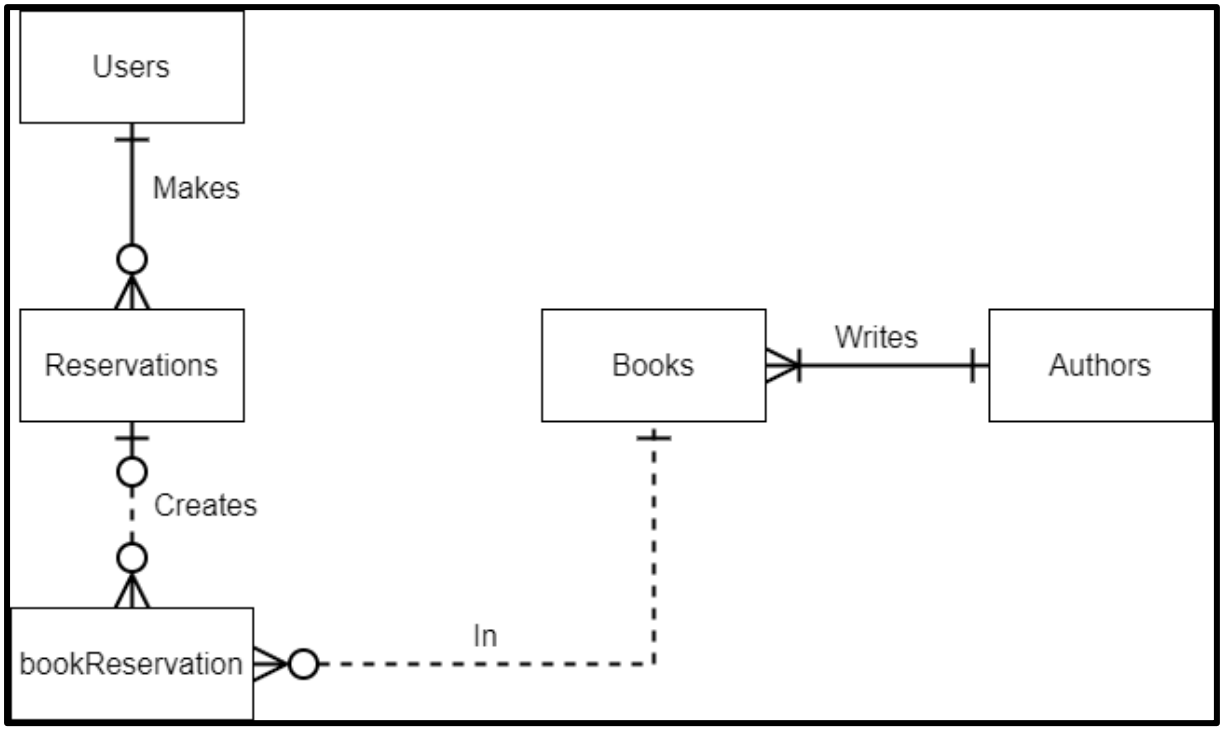
Extra screens

Deletes the old main window when its created,

(Error screen is a separate window)

Database design

Entity-relationship diagram



- One User makes many Reservations. One Reservation is made by one User
- One Reservation creates many bookReservations. One bookReservation belongs to one Reservation
- One Book is in many bookReservations. One bookReservation is for one Book
- One Author writes many books. One Book is written by one Author

Data dictionaries

Entity: Authors					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
authorID	PK	Integer		Yes	Auto increment
firstName		Varchar	30	Yes	
surname		Varchar	30	No	

Entity: Books					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
ISBN	PK	Varchar	13	Yes	Length = 13
authorID	FK	Integer		Yes	Existing authorID from Authors table
title		Varchar	45	Yes	
subTitle		Varchar	30	No	
bookStatus		Varchar	9	Yes	Restricted choice: [Available, Reserved, Taken out, Overdue]
starRating		Decimal	(3,2)	No	Range >= 0 AND <= 5
genre		Varchar	15	Yes	Restricted choice: [Non-fiction, Fantasy, Science Fiction, Romance, Horror]
blurb		Varchar	1000	No	
datePublished		Date		Yes	

Entity: bookReservation					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
reservationID	PK FK	Integer		Yes	Existing reservationID from Reservations
ISBN	PK FK	Varchar	13	Yes	Existing ISBN from Books

Entity: Reservations					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
reservationID	PK	Integer		Yes	Auto increment
username	FK	Varchar	20	Yes	Existing username from Users
dateOfOrder		Date		Yes	
timeOfOrder		Time		Yes	
dueDate		Date		Yes	

Entity: Users					
Attribute name	**Key**	**Type**	**Size**	**Required**	**Validation**
username	PK	Varchar	20	Yes	Length >= 10 and Length <= 20
firstName		Varchar	30	Yes	
surname		Varchar	30	Yes	
passwordHash		Varchar	64	Yes	Length = 64
librarian		Boolean		Yes	

Query Designs

XQYA
X - Query type (S for SELECT, U for UPDATE, I for INSERT)
Q - Indicates query
Y - ID number (1,2,3,4 ...)
A - A, B, C ... (Level)

I.E. SQ1 – SELECT query 1
SQ8A – SELECT query 8 main query
SQ8B – SELECT query 8 extra query
UQ3 – UPDATE query 3

SELECT queries

SQ1. Get all ISBN(s) and due dates for "Taken out" books

Field(s)	ISBN, dueDate
Table(s)	Books, bookReservation, Reservations
Search criteria	bookStatus = "Taken Out"

SQ2. Get all user details

Field(s)	username, passwordHash, librarian
Table(s)	Users
Sort order	username ASC

SQ3. Get all usernames

Field(s)	username
Table(s)	Users
Sort order	BINARY username ASC

SQ4A. All available books

Field(s)	ISBN, title, subTitle, genre, firstName, surname
Table(s)	Books, Authors
Search criteria	bookStatus = "Available"
Sort Order	title ASC

SQ4B. All available books filtered by genre

Field(s)	ISBN, title, subTitle, genre, firstName, surname
Table(s)	Books, Authors
Search criteria	bookStatus = "Available" AND genre = [FROM SOFTWARE]
Sort order	title ASC

SQ5. Get all book details for specific book

Field(s)	ISBN, title, subTitle, firstName, surname, genre, starRating, datePublished, blurb
Table(s)	Books, Authors
Search criteria	ISBN = [FROM SOFTWARE]

SQ6. Get title and subTitle of user's cart books

Field(s)	title, subTitle
Table(s)	Books
Search criteria	ISBN = [FROM SOFTWARE]

SQ7. Get the reservation ID of the newly made reservation

Field(s)	reservationID
Table(s)	Reservations
Search criteria	username = [FROM SOFTWARE] AND dateOfOrder = [FROM SOFTWARE] AND timeOfOrder = [FROM SOFTWARE]

SQ8. Get all users (with basic details) and the number of books they have.

Field(s) / Calculation(s)	u.username, u.firstName, r.reservationID, COUNT(bR.ISBN) AS [NO. Of books]		
Table(s)	Users as u, Reservations as r, bookReservation as bR		
Search criteria	bR.ISBN IN (	Field(s)	ISBN
		Table(s)	Books
		Search criteria	bookStatus = [FROM SOFTWARE] [Reserved/Taken out]
Grouping	u.username, u.firstName, r.reservationID		

SQ9A. Get all books in a specific reservation of a specific status

Field(s)	ISBN, title, subTitle, genre
Table(s)	Books, Reservations, bookReservation
Search criteria	reservationID = [FROM SOFTWARE] AND bookStatus = [FROM SOFTWARE] [“Reserved”/”Taken out”]

SQ9B. Get the time and date of specific reservation

Field(s)	dateOfOrder, timeOfOrder
Table(s)	Reservations
Search criteria	reservationID = [FROM SOFTWARE]

SQ10. Get all username and number of overdue books

Field(s) / Calculation(s)	username, COUNT(*) AS [NO. Of overdue books]
Table(s)	Users, Reservations, bookReservation, Books
Search criteria	bookStatus = “Overdue”
Grouping	username

SQ11. Get all the overdue books of a specific user

Field(s)	ISBN, title, subTitle, duedate
Table(s)	Books, bookReservation, Reservations, Users
Search criteria	bookStatus = “Overdue” AND username = [FROM SOFTWARE]

UPDATE queries

UQ1. Change specified book status to “Overdue”

Table(s)	Books
Value(s)	bookStatus = “Overdue”
Search criteria	ISBN = [FROM SOFTWARE]

UQ2. Change specified book status to “Reserved”

Table(s)	Books
Value(s)	bookStatus = “Reserved”
Search criteria	ISBN = [FROM SOFTWARE]

UQ3. Change specified book status to “Taken out”

Table(s)	Books
Value(s)	bookStatus = “Taken out”
Search criteria	ISBN = [FROM SOFTWARE]

UQ4. Change specified book status to “Available”

Table(s)	Books
Value(s)	bookStatus = “Available”
Search criteria	ISBN = [FROM SOFTWARE]

INSERT queries

IQ1. Make a new user

Table(s)	Users
Column(s)	username, firstName, surname, passwordHash, librarian
Value(s)	[FROM SOFTWARE]

IQ2. Make a new reservation

Table(s)	Reservations
Column(s)	username, dateOfOrder, timeOfOrder, dueDate
Value(s)	[FROM SOFTWARE]

IQ3. Make a new book reservation

Table(s)	bookReservation
Column(s)	reservationID, ISBN
Value(s)	[FROM SOFTWARE]

Data structure design

There will be several 2D array used throughout the software. Each of the 2D arrays will hold the results of SELECT queries. This means that each 2D array will hold data in accordance with the SELECT query designs. Each 2D array will be a LOCAL variable for each module.

I.E.
SQ1 – result 2D array = [[ISBN, dueDate], [ISBN, dueDate], ……]
SQ2 – result 2D array = [[username, passwordHash, librarian], ……]
SQ3 – result 2D array = [[username], [username], ……]
…
So on and so forth for every select query.

Software design

- **D.A.O.W** (Destroy all on window)

Main file:

1.	main	
2.	update_status	
3.	start_menu	
4.	login_screen	
A.	validate_login	IN: Result
5.	signup_screen	
A.	validate_signup	IN: Result
6.	catalogue_screen	
7.	book_details_screen	IN: isbn
A.	add_book	IN: isbn
8.	cart_screen	
9.	checkout_screen	
10.	librarian_menu	
11.	action_menu	
12.	to_prepare_reservations	
13.	to_prepare_books	IN: ID
14.	due_reservations	
15.	due_books	IN: ID
16.	overdue_reservations	
17.	overdue_books	IN: c_username

```
1.0  PROCEDURE main
1.1          SET cart TO [""] * 3
1.2          SET cart AS GLOBAL
1.3          SET connection TO CALL db_connect() {LQ}
1.4          SET connection AS GLOBAL
1.5          IF connection THEN
1.6                  CREATE WINDOW
1.7                  CALL update_status()
1.8                  CALL start_menu()
1.9          ELSE THEN
1.10                         CALL error_screen(connection) {LE}
1.11                 END IF
1.12 END PROCEDURE
```

Creates a procedure that initialises the software when called. Creates global cart array and global connection variable. Connection variable made from the Extras file db_connect. Then it creates a window if the connection is not NONE, and displays an error screen if it is.

```
2.0  PROCEDURE update_status
2.1          SET Result = CALL query_select(SQ1, (None), connection) {LQ}
2.2          SET today TO SYSTEM DATE
2.3          LOOP for item IN Result DO
2.4                  IF today >= item[1] THEN
2.5                          CALL query_update(UQ1, (item[0]), connection)  {LQ}
2.6                  END IF
2.7          END LOOP
2.8  END PROCEDURE

3.0  PROCEDURE start_menu
3.1          DISPLAY "Book & Borrow" (title)
3.2          DISPLAY "Please login or sign up"
3.3          CREATE login_button BUTTON
3.4                  WHEN PRESSED [CALL login_screen(), D.A.O.W (EXCEPT title)]
3.5          CREATE signup_button BUTTON
3.6                  WHEN PRESSED [CALL signup_screen(), D.A.O.W (EXCEPT title)]
3.7  END PROCEDURE
```

The update_status procedure will automatically set any books past their due date to overdue. It uses SQ1 to get all “Taken out” books (using the data structure shown on page 25). It then will get the system date and store it in a variable. It will then loop through each query result comparing the current date to the due date of that book. If the book is overdue then it changes the book status to be so using UQ1.

The start_menu procedure creates the start menu screen on the window. It creates the title of this first area “Book & Borrow”, this is not deleted until after login. It then creates buttons for the user to press which brings them to the appropriate screen and destroys everything on the screen (except for the title). It removes everything on the window to allow the next procedure to display all of its details on the window.

```
4.0   PROCEDURE login_screen
4.1         SET Result = Call query_select(SQ2, (None), connection) {LQ}
4.2         DISPLAY "Login"
4.3         DISPLAY "Username:"
4.4         CREATE username TEXTBOX
4.5         DISPLAY "Password: "
4.6         CREATE password TEXTBOX
4.7         CREATE submit BUTTON
4.8               START AS DISABLED
4.9               WHEN PRESSED [CALL validate_login(Result)]
4.10        CREATE return BUTTON
4.11              WHEN PRESSED [CALL start_menu(), D.A.O.W (EXCEPT title)]
4.12        IF user has entered text into every TEXTBOX THEN
4.13              CHANGE submit BUTTON TO ENABLED
4.14        END IF
4.15  END PROCEDURE

4.A.1  PROCEDURE validate_login
4.A.2       GET user's input FROM TEXTBOX(es)
4.A.3       SET password_hash = hash_password(password{TEXTBOX}) {LE}
4.A.4       SET position = 0
4.A.5       SET position = binary_search(usernames, username{TEXTBOX}, 0) {LE}
4.A.6       IF position != -1 THEN
4.A.7             IF Result[position][1]== password_hash THEN
4.A.8             MAKE username{TEXTBOX} GLOBAL
4.A.9             D.A.O.W
4.A.10                  IF Result[position][2] == 1 THEN
4.A.11                        DISPLAY POPUP "Welcome to the librarian system"
4.A.12                        CALL librarian_menu()
4.A.13                  ELSE THEN
4.A.14                       DISPLAY POPUP "Welcome &username{TEXTBOX} "
4.A.15                        CALL catalogue_screen()
4.A.16                  END IF
4.A.17            ELSE THEN
4.A.18                  DISPLAY POPUP "Password is incorrect"
4.A.19            END IF
4.A.20      ELSE THEN
4.A.21            DISPLAY POPUP "Username doesn't exist"
4.A.22      END IF
4.A.23  END PROCEDURE
```

The login_screen procedure uses SQ2 to get all user login details. It then creates all the text and text boxes on the window. It then creates the two buttons on this screen and starts the submit_button as disabled. There is a conditional statement at the end to check to make sure text is in every box and if so makes the submit_button clickable. The return_button when clicked will return back to the start screen. The submit_button will call the validate_login procedure when clicked.

The validate_login procedure checks the user entered details to see if they are valid. It gets all user inputs and then hashes the entered password. Then it searches the SQ2 results with a binary search to see if the entered username exists in it. If the username does exist in the query result, then it moves on with validation, if it doesn't exist, then it displays "Username doesn't exist".
After that it will check to see if the entered password (after being hashed) matches the query result password at the binary search returned position, if they don't match it says so and if they do match it moves on. It makes the valid user's username global (for use in the checkout module) and removes the login_screen from the window. It then displays an appropriate popup and screen depending on user type.
Result[position][2] == 1, They are a librarian. Send them to the librarian menu
Result[position][2] == 0, They are a borrower. Send them to the catalogue

```
5.0   PROCEDURE signup_screen
5.1         SET Result = CALL query_select(SQ3, (None) , connection)  {LQ}
5.2         DISPLAY "Sign up" TO WINDOW
5.3         DISPLAY "First name:" TO WINDOW
5.4         CREATE first_name TEXTBOX
5.5         DISPLAY "Surname: " TO WINDOW
5.6         CREATE surname TEXTBOX
5.7         DISPLAY "Username:" TO WINDOW
5.8         CREATE username TEXTBOX
5.9         DISPLAY "Password: " TO WINDOW
5.10        CREATE password TEXTBOX
5.11        DISPLAY "Confirm password:" TO WINDOW
5.12        CREATE conf_password TEXTBOX
5.13        CREATE submit BUTTON
5.14              START AS DISABLED
5.15              WHEN PRESSED [CALL validate_signup(Result)]
5.16        CREATE return BUTTON
5.17              WHEN PRESSED [CALL start_menu(), D.A.O.W (EXCEPT title)]
5.18        IF user has entered text into every box THEN
5.19              CHANGE submit BUTTON TO ENABLED
5.20        END IF
5.21  END PROCEDURE

5.A.1  PROCEDURE validate_signup
5.A.2       GET users inputs from TEXTBOX(es)
5.A.3       SET position = binary_search(Result, username{TEXTBOX}, 0)   {LE}
5.A.4       IF position > -1 THEN
5.A.5             DISPLAY POPUP "Username already in use"
5.A.6       ELSE IF LENGTH(username{TEXTBOX}) NOT BETWEEN (10 AND 20) THEN
5.A.7             DISPLAY POPUP "Username must 10-20 characters"
5.A.8        ELSE IF LENGTH(password{TEXTBOX}) < 12 THEN
5.A.9             DISPLAY POPUP "Password must be at least 12 characters"
5.A.10      ELSE IF password{TEXTBOX} != conf_password{TEXTBOX} THEN
5.A.11            DISPLAY POPUP "Passwords must match"
5.A.12      ELSE THEN
5.A.13            CALL query_select(IQ1, (user inputs from TEXTBOX(es)))  {LQ}
5.A.14            DISPLAY POPUP "Welcome new user. Please login now"
5.A.15            DESTROY ALL EXCEPT title("Book & Borrow") ON WINDOW
5.A.16            CALL login_screen()
5.A.17  END PROCEDURE
```

The signup_screen procedure displays everything from the signup screen onto the window. It calls SQ3, which is all the existing usernames. It then displays all the text and textboxes for the user. After that it creates the submit and return buttons which act the same as the login screens on page 28, except the submit_button calls validate_login instead.

The validate_signup procedure makes sure the entered details are valid before making a new user. It gets all entered textbox details and then searches the SQ3 results to see if the entered username already exists. If the username already exists it tells the user that, if the username is unique though it's fine. It then also checks to see if the entered username is 10-20 characters long and if the password is at least 12 characters long. It will then check if the entered password and the entered password confirmation match. If the entered details pass all of these checks then it uses IQ1 to create a new user and takes the user to the login screen (to make them login with their new account).

```
6.0  PROCEDURE catalogue_screen
6.1         SET choices TO ["None", "Non-fiction", "Horror", "Science-fiction", "Romance", "Fantasy"]
6.2         SET user_choice = ""
6.3         CREATE DROPDOWN BOX WITH choices
6.4             Starting value = choices[0]
6.5         SET user_choice = DROPDOWN BOX value
6.6         DISPLAY "Available books:" TO WINDOW (title)
6.7         CREATE cart BUTTON
6.8             WHEN PRESSED [CALL cart_menu(), DESTROY ALL ON SCREEN]
6.9         SET Result = CALL query_select(SQ4A, (None) , connection)  {LQ}
6.10        IF LENGTH(Result) NOT == 0 THEN
6.11            LOOP for each book in Result DO
6.12                CREATE BUTTON using Result[book] details
6.13                WHEN PRESSED [CALL book_details_screen(Result[book][0]), D.A.O.W]
6.14            END LOOP
6.15        ELSE THEN
6.16            DISPLAY "No available books"
6.17        END IF
6.18        IF user selects option from DROPDOWN BOX THEN
6.19            DESTROY title and BUTTONS
6.20            IF user_choice == "None" THEN
6.21                DISPLAY "Available books:" TO WINDOW
6.22                SET Result = CALL query_select(SQ4A, (None) , connection)  {LQ}
6.23            ELSE THEN
6.24                DISPLAY "Available books for: \n" & user_choice
6.25                SET Result = CALL query_select(SQ4B, (None) , connection)  {LQ}
6.26            END IF
6.27            Repeat Lines 6.10 TO 6.17
6.28        END IF
6.29 END PROCEDURE
```

The catalogue-screen will display the selection of all available books. It creates a dropdown menu with all genre options and tracks its selected value (this is for the genre filtering). It creates a cart button and when clicked takes the user to the cart screen. Finally, it generates all the book buttons. By default, it uses SQ4A to display a grid of all the book details on buttons. It loops through the 2D array and uses all the details. Then if the user picks an option from the dropdown box it will regenerate all these buttons using SQ4B (using the dropdown box value as the filter).

```
7.0   PROCEDURE book_details_screen
7.1         SET Result = CALL query_select(SQ5, (isbn) , connection)  {LQ}
7.2         CREATE return BUTTON
7.3               WHEN PRESSED [CALL catalogue_screen(), D.A.O.W]
7.4         DISPLAY "Book details"
7.5         CREATE cart BUTTON
7.6               WHEN PRESSED [CALL cart_screen(), D.A.O.W]
7.7         DISPLAY "Title: " & Result[0][1] & Result[0][2]
7.8         DISPLAY "By: " & Result[0][3] & Result[0][4]
7.9         DISPLAY "Genre: " & Result[0][5]
7.10        DISPLAY "Star rating: " & STR(Result[0][6]) & "/5"
7.11        DISPLAY "Date published: " & Result[0][7]
7.12        DISPLAY "Blurb: " & Result[0][8]
7.13        CREATE add_to_cart BUTTON
7.14              WHEN PRESSED CALL add_book(isbn)
7.15  END PROCEDURE

7.A.1  PROCEDURE add_book
7.A.2       SET found = FALSE
7.A.3       SET n = 0
7.A.4       WHILE NOT found AND n < 3 DO
7.A.5             IF cart[n] == "" THEN
7.A.6                   SET found = TRUE
7.A.7                   SET cart[n] = isbn
7.A.8                   DISPLAY POPUP "Book has been added to cart"
7.A.9             ELSE IF cart[n] == isbn THEN
7.A.8                   DISPLAY POPUP "Book is already in cart"
7.A.9                   SET found = TRUE
7.A.10            ELSE THEN
7.A.11                  SET n = n + 1
7.A.12            END IF
7.A.13      IF NOT found THEN
7.A.14            DISPLAY POPUP “Cart is full”
7.A.15      END IF
7.A.16  END PROCEDURE
```

The book_details_screen procedure will display the details of the book selected from the catalogue screen. It uses SQ5 with the ISBN that is passed into it. Then it creates a full display for the user to read the details of the book. It also creates a cart button, to take the user to the cart screen and an add to cart button, which calls the add_book procedure.

The add_book procedure will check to see if the current book can be added to cart. It uses a linear search algorithm to search the global cart array for a blank spot or for if the current cart item is already equal to the ISBN.
If the spot is blank it will add the ISBN to cart and make a popup.
If the array item is equal to the ISBN it does not add the ISBN to the cart, and makes a popup saying "Book is already in cart".
If neither of these conditions are found then found = FALSE, and if found = FALSE it displays a popup saying "Cart is full" (since no blank spot or equal value was found).

```
8.0   PROCEDURE cart_screen
8.1         CREATE return BUTTON
8.2               WHEN PRESSED [CALL catalogue_screen, D.A.O.W]
8.3         DISPLAY "Your current cart:" TO WINDOW
8.4         CREATE checkout BUTTON
8.5               START AS DISABLED
8.6               WHEN PRESSED [CALL checkout_screen(), D.A.O.W]
8.7         SET items = CALL count_cart(cart)  {LE}
8.8         IF items > 0 THEN
8.9               CHANGE checkout BUTTON TO ENABLED
8.10        END IF
8.11        LOOP i, FROM 0 TO 2 DO
8.12              IF cart[i] == "" THEN
8.13                    DISPLAY i+1 & ". No book selected"
8.14              ELSE THEN
8.15                    SET Result = CALL query_select(SQ6, (cart[i]) , connection)   {LQ}
8.16                    DISPLAY i+1 & ". " & Result[0][0] & " " & Result[0][1]
8.17                    CREATE remove BUTTON
8.18                          WHEN PRESSED [SET cart[i] = "",
8.19                          DISPLAY POPUP "Book was removed", DELETE TEXT of Result[i]]
8.20              END IF
8.21  END PROCEDURE
```

The cart_screen procedure will display all the of the cart screen details. It creates a return button which takes user to the catalogue screen when clicked. It also creates a checkout buton and starts it as disabled, this will call the checkout_screen procedure when clicked. It gets the number of items in the cart using a function from the Extras file and if there are items in the cart then the checkout button is made clickable again.

Then it loops for all of the cart array and if the spot is blank it displays “No book selected” and if the spot isn’t blank it calls SQ6 with the current cart value and displays the book’s title. If the value isn’t blank it will also create a removal button that when pressed will remove the corresponding book from the user’s cart (and delete the text from the screen).

```
9.0   PROCEDURE checkout_screen
9.1         SET date = SYSTEM DATE
9.2         SET time = SYSTEM TIME
9.3         SET due_date = date + 28 days
9.4         CALL query_insert(IQ2, (username{GLOBAL}, date, time, due_date) , connection)    {LQ}
9.5         SET Result = CALL query_Select(SQ7, (username{GLOBAL}, date, time) , connection)  {LQ}
9.6         SET id = Result[0][0]
9.6         SET items = count_cart(cart)    {LE}
9.7         DISPLAY "Confirmed"
9.8         DISPLAY "Your books will be ready in: " & STRING((items * 5) + 2) & " minutes"
9.9         DISPLAY "The overdue fees will be: £" & STRING(items * 0.2)
9.10        DISPLAY "The book(s) are due on: " & due_date
9.11        LOOP i, FROM 0 TO 2 DO
9.12              IF cart[i] != "" THEN
9.13                    CALL query_update(UQ2, (cart[i]) , connection)   {LQ}
9.14                    CALL query_insert(IQ3, (id, cart[i]) , connection)   {LQ}
9.15              END IF
9.16        END LOOP
9.17        DISPLAY "Please exit the program and come to your library to pick up your books"
9.18        CREATE exit BUTTON
9.19              WHEN PRESSED [DESTROY WINDOW, CLOSE PROGRAM, CLOSE connection]
9.20  END PROCEDURE
```

The checkout_screen procedure creates the reservation for the cart books and displays the details of the checkout screen. It retrieves the current system date and time and calculates the due date (by adding 28 days). It then uses IQ2 to create a new reservation using the global username and details created earlier in this procedure. It then will call SQ7 to get the reservation ID of that newly made reservation. It then counts the number of items in the user's cart (using an Extras file function). The items value is then used for calculations that are displayed to user (I.e. overdue fee values). Then it loops through the whole cart and if the current value is not blank it will use UQ2 to change the current book's status and use IQ3 to make a new bookReservation.

Finally, it makes an exit button that closes the window, closes the database connection and ends the software when clicked.

```
10.1  PROCEDURE librarian_menu
10.2        DISPLAY "Welcome librarian"
10.3        DISPLAY "What would you like to do?"
10.4        CREATE browse Button
10.5              WHEN PRESSED [CALL catalogue_screen(), D.A.O.W]
10.6        CREATE manage BUTTON
10.7              WHEN PRESSED [CALL action_menu(), D.A.O.W]
10.8  END PROCEDURE

11.1   PROCEDURE action_menu
11.2        CREATE return BUTTON
11.3              WHEN PRESSED [CALL librarian_menu, D.A.O.W]
11.4        DISPLAY "Please select an option:" TO WINDOW
11.5        CREATE to_prepare BUTTON
11.6              WHEN PRESSED [CALL to_prepare_reservation(), D.A.O.W]
11.7        CREATE due BUTTON
11.8              WHEN PRESSED [CALL due_reservations(), D.A.O.W]
11.9        CREATE overdue BUTTON
11.10             WHEN PRESSED [CALL overdue_reservations(), D.A.O.W]
11.11  END PROCCEDURE
```

The librarian_menu procedure displays all the details of the librarian menu screen. It creates text and buttons. These buttons will take the user either to the catalogue or the action menu.

The action_menu procedure will display all the details of the action menu screen. It creates a return button that takes the user back to the action menu when clicked. It then creates 3 buttons for all 3 types of viewable book statuses for librarians, and when clicked will take the user to the corresponding reservations screen for that status.

```
12.1   PROCEDURE to_prepare_reservation
12.2        SET Result = CALL query_select(SQ8, ("Reserved"), connection)    {LQ}
12.3        CREATE return BUTTON
12.4             WHEN PRESSED [CALL action_menu(), D.A.O.W]
12.5        IF LENGTH(Result) NOT == 0 THEN
12.6             LOOP i, FROM 0 TO LENGTH(Result)-1 DO
12.7                  CREATE reservation BUTTON
12.8                       USE Result for display text
12.9                       WHEN PRESSED [CALL to_prepare_books(Result[i][2]), D.A.O.W]
12.10            END LOOP
12.11       ELSE THEN
12.12            DISPLAY "There are currently no reservations"
12.13       END IF
12.14  END PROCEDURE

13.1   PROCEDURE to_prepare_books
13.2        SET books = CALL query_select(SQ9A, (ID, "Reserved"), connection)  {LQ}
13.3        SET date_time = CALL query_select(SQ9B, (ID) , connection)  {LQ}
13.4        CREATE return BUTTON
13.5             WHEN PRESSED [CALL to_prepare_reservation(), D.A.O.W]
13.6        DISPLAY "Date of reservation: " & date_time[0][0]
13.7        DISPLAY "Time of reservation: " & date_time[0][1]
13.8        LOOP i, FROM 0 TO LENGTH(books) DO
13.9             DISPLAY DETAILS OF CURRENT book TO WINDOW
13.10            CREATE change_status BUTTON
13.11                 WHEN PRESSED [DISPLAY POPUP "Status changed", CALL
      query_update(UQ3, (books[i][0]) , connection)]
13.12       END LOOP
13.13  END PROCEDURE
```

The to_prepare_reservation procedure uses SQ8 to get all the reservations with “Reserved” books and display them. If there are no reservations with “Reserved” books it displays text saying so. But if there are reservations then it creates buttons that will take the user to the corresponding to_prepare_books screen.

The to_prepare_books procedure will display all the books in a reservation and reservation details using SQ9A and SQ9B. It displays all their details and makes a change_status button for each book. The button when clicked will use UQ3 to change that specific book’s status.

```
14.1   PROCEDURE due_reservations
14.2        SET Result = CALL query_Select(SQ8, ("Taken out"), connection)  {LQ}
14.3        CREATE return BUTTON
14.4             WHEN PRESSED [CALL action_menu(), D.A.O.W]
14.5        IF LENGTH(Result) NOT == 0 DO
14.6             LOOP i, FROM 0 TO LENGTH(Result)-1 DO
14.7                  DISPLAY DETAILS of CURRENT book TO WINDOW
14.8                  CREATE reservation BUTTON
14.9                       WHEN PRESSED [CALL due_books(Result[i][2]), D.A.O.W]
14.10            END LOOP
14.11       ELSE THEN
14.12            DISPLAY "There are currently no reservations"
14.13       END IF
14.14  END PROCEDURE

15.1   PROCEDURE due_books
15.2        SET books = CALL query_select(SQ9A, (ID, "Taken out"), connection)  {LQ}
15.3        SET date_time = CALL query_Select(SQ9B, (ID) , connection)  {LQ}
15.4        CREATE return BUTTON
15.5             WHEN PRESSED [CALL due_reservations(), D.A.O.W]
15.6        DISPLAY "Date of reservation: " & date_time[0][0]
15.7        DISPLAY "Time of reservation: " & date_time[0][1]
15.8        LOOP i, FROM 0 TO LENGTH(books)-1 DO
15.9             DISPLAY CURRENT book DETAILS TO WINDOW
15.10            CREATE change_status BUTTON
15.11                 WHEN PRESSED [DISPLAY POPUP "Status changed", CALL
      query_update(UQ4, (books[i][0]) , connection)]
15.12  END PROCEDURE
```

The due_reservations procedure uses SQ8 to get all the reservations with “Taken out” books and displays them. If there are no reservations with “Taken out” books it displays text saying so. But if there are books then it creates buttons that will take the user to the corresponding due_books screen.

The due_books procedure will display all the books in a reservation and reservation details using SQ9A and SQ9B. It displays all their details and makes a change_status button for each book. The button when clicked will use UQ4 to change that specific book’s status.

```
16.1   PROCEDURE overdue_reservations
16.2        SET Result = CALL query_select(SQ10, (None), connection)     {LQ}
16.3        CREATE return BUTTON
16.4             WHEN PRESSED [CALL action_menu(), D.A.O.W]
16.5        DISPLAY "Here are users and how many overdue books they currently have:"
16.6        IF LENGTH(Result) NOT == 0 THEN
16.7             LOOP i, FROM 0 TO LENGTH(Result)-1 DO
16.8                  CREATE book BUTTON
16.9                       USE Result[i] for TEXT
16.10                      WHEN PRESSED [CALL overdue_books(), D.A.O.W]
16.11            END LOOP
16.12       ELSE THEN
16.13            DISPLAY "There are currently no users with overdue books"
16.14       END IF
16.15  END PROCEDURE

17.1   PROCEDURE overdue_books
17.2        SET books = query_select(SQ11, (c_username) , connection)       {LQ}
17.3        CREATE return BUTTON
17.4             WHEN PRESSED [D.A.O.W, CALL action_menu()]
17.5        LOOP i, FROM 0 TO LENGTH(books)-1 DO
17.6             CREATE book BUTTON
17.7                  USE books to create display text
17.8                  WHEN PRESSED [CALL query_update(UQ4, (books[i][0]) , connection),
      DISPLAY POPUP "Status changed"]
17.9        END LOOP
17.10  END PROCEDURE
```

The overdue_reservation procedure will use SQ10 to get all the users with overdue books and show how many overdue books they have. If there are no users that have overdue books, text will be displayed saying so. For each user in the query result it will display a button that will take the librarian to the corresponding overdue_books screen.

The overdue_books procedure will use SQ11 to get all the overdue books that the specified user has. It will create a button using the details of each book and when clicked it will change the book's status using UQ4.

Extras file:

1.	binary_search	IN: array, target, index	OUT: position
2.	hash_password	IN: password	OUT: passwordHash
3.	count_cart	IN: cart	OUT: items
4.	error_screen	IN: connection	

```
1.1   FUNCTION binary_search
1.2             SET position = -1
1.3             SET found = FALSE
1.4             SET high = LENGTH(array)-1
1.5             SET low = 0
1.6             SET mid = 0
1.7             WHILE NOT found AND low <= high DO
1.8                     SET mid = INT((low + mid) / 2)
1.9                     IF array[mid][index] == target THEN
1.10                            SET found = TRUE
1.11                            SET position = mid
1.12                    ELSE IF array[mid][index] > target THEN
1.13                            SET high = mid - 1
1.14                    ELSE THEN
1.15                            SET low = mid + 1
1.16                    END IF
1.17            END WHILE
1.18            RETURN position
1.19   END FUNCTION

2.1   FUNCTION hash_password
2.2             SET passwordHash = ""
2.3             SET passwordHash = Sha256 of password
2.4             RETURN passwordHash
2.5   END FUNCTION
```

This section handles the binary search on a 2D array and the password hashing functions.

The structure of the binary_search function is standard however during comparisons in the conditional loop the fact that this is a 2D array must be accounted for. Therefore array[mid][index] instead of array[mid]. The index is passed into it as a parameter and refers to the position in each row of the 2D array is being ssearched.

The hash_password function takes in a password and uses the Sha256 algorithm on it to create a hashed password. This is then returned.

These two functions are used during the login/sign up process.
At login the binary_search is used to check if the entered username exists and hash_password for if the database stored password matches the entered password (the entered password would become hashed before comparison).
At sign up the binary_search is used to check if the entered username already exists (to ensure unique usernames) and the hash_password is used to hash the entered password for inserting the user into the database.

```
3.1  FUNCTION count_cart
3.2          SET items = 0
3.3          LOOP i, FROM 0 TO LENGTH(cart)-1 DO
3.4                  IF cart[i] != "" THEN
3.5                          SET items TO items + 1
3.6                  END IF
3.7          END LOOP
3.8          RETURN items
3.9  END FUNCTION

4.1  PROCEDURE error_screen
4.2          DESTROY OLD WINDOW
4.3          CREATE NEW WINDOW
4.4          CLOSE connection IF EXISTS
4.4          DISPLAY "We are sorry but there has been an error with the database connection"
4.5          DISPLAY "Please close and reopen the software"
4.6  END PROCEDURE
```

This section handles the count_cart function and the error_screen module.

Count_cart searches the entered array for how many non-blank spaces are in it. The cart array from Main is passed into it and so it counts the number of items in the cart. (It counts the number of ISBNs in the cart/ number of non-blank spaces in the cart). It is used at the cart and checkout screens for displays or calculations.

Error_screen is a procedure which destroys the main window (where login, catalogue and etc. are displayed). It then creates a new window to display the error message and closes the database connection if it exists. This is used at various places like the query modules in the Query file and at the main module of the Main file.

Queries file:

1.	db_connect		OUT: connection
2.	query_select	IN: query, values, connection	OUT: result
3.	query_insert	IN: query, values, connection	
4.	query_update	IN: query, values, connection	
5.	All of the queries (SELECT, UPDATE and INSERT)		

```
1.1  FUNCTION db_connect
1.2          TRY TO
1.3                  CREATE connection to database [using preset details]
1.4                  Store as connection
1.5          EXCEPT ERROR
1.6                  SET connection = NONE
1.7                  RETURN connection
1.8          ELSE THEN
1.9                  RETURN connection
1.10         END TRY
1.11 END FUNCTION
```

This module creates a connection to the database using the details of a user which will not change. It has a TRY CATCH for if the connection is unsuccessful. It tries to create a connection and if successful returns a full connection variable. It catches the error and returns NONE, indicating a failed connection.

```
2.1  FUNCTION query_select
2.2          TRY TO
2.3                  EXECUTE query USING values AND connection
2.4          EXCEPT ERROR
2.5                  CALL error_screen(connection)  {LE}
2.6          ELSE THEN
2.7                  SET result = query result
2.8                  RETURN result
2.9  END FUNCTION

3.1  PROCEDURE query_insert
3.2          TRY TO
3.3                  EXECUTE query USING values AND connection
3.4          EXCEPT ERROR
3.5                  CALL error_screen(connection)  {LE}
3.6          ELSE THEN
3.7                  COMMIT change to database
3.8  END PROCEDURE

4.1  PROCEDURE query_update
4.2          TRY TO
4.3                  EXECUTE query USING values AND connection
4.4          EXCEPT ERROR
4.5                  CALL error_screen(connection)  {LE}
4.6          ELSE THEN
4.7                  COMMIT change to database
4.8  END PROCEDURE

5.   SQL and DML code for all queries. Stored as strings
```

This section holds the three modules that execute queries. Each of them holds a TRY CATCH for attempting to execute the query, if that fails then it will display the error screen. Though if the query does work then it will either return the query results or commit the changes to the database.

This will also hold all the queries which will be used. These will all be stored as strings and called upon by the Main file when it's their time to be used.
I.E.
example = "SELECT * FROM books"

Implementation

New skills researched and developed

This section mostly includes python modules that needed to be learnt to create the software.

Tkinter

To create graphical user interfaces I needed to learn a python module that would allow me to do such. I picked Tkinter since I didn't need to download the module using pip. I needed to learn various different ways to format and manipulate data on the windows. I used the main tkinter module and also took in the ttk (themed tkinter) and messagebox modules from tkinter. These allow me to theme the treeview table used in the book details screen and allow me to create popups respectively.

I used these resources to help learn:

- https://www.geeksforgeeks.org/python-gui-tkinter/
- https://docs.python.org/3/library/tkinter.html
- https://www.tutorialspoint.com/python/python_gui_programming.htm

Hashlib

To maintain GDPR I needed to hash user's passwords for storage. I accomplished this by using the python hashlib module. I picked this module as again it didn't require me to download it using pip and it also allowed me to pick the sha256 hashing algorithm which is generally regarded as the most secure basic hashing algorithm. This allowed me to create the 64-character long password hash strings.

I used these resources to help learn:

- https://docs.python.org/3/library/hashlib.html
- https://www.geeksforgeeks.org/hashlib-module-in-python/

MySQL workbench and connector

To store my library database, I needed something to hold it. I went with the SQA standard MySQL and used MySQL workbench as I felt it would give me more control over my whole database and allow me to view it in an easy to read format. I needed to learn how to create new users through the workbench command prompt and also what each part of the display meant. Then for the connection I just used the MySQL.connector module

I used these resources to help learn:

- The SQA guide on how to install MySQL connector
- https://www.hostinger.com/tutorials/mysql/how-create-mysql-user-and-grant-permissions-command-line
- https://dev.mysql.com/doc/workbench/en/wb-intro.html

Datetime and textwrap

I needed to do formatting on results from my database queries so I picked the datetime and textwrap modules. Datetime would allow me to change the format of dates from YYYY-MM-DD (which is the MySQL database format) to the DD-MM-YYYY format and it also allows me to do calculations with days for calculating the due date of books. The textwrap module allowed me to wrap text over lines without breaking up the words. The textwrap module was used for things like the blurb in the book details screen, the book titles in the catalogue and also book reservation buttons for librarians.

I used these resources to help learn:

- https://docs.python.org/3/library/datetime.html#module-datetime
- https://www.geeksforgeeks.org/python-datetime-timedelta-function/
- https://www.w3schools.com/python/python_datetime.asp
- https://docs.python.org/3/library/textwrap.html
- https://docs.python.org/3/library/textwrap.html

Data structure

The data structure proposed in design on page 25 has been slightly changed. Instead of being a pure 2D array, it's instead an array of tuples. This behaves in the same way as a basic 2D array, it's just used because it is the data structure returned by the mysql.connector module by cursor.fetchall().

Database

Queries

Query ID	SQL
SQ1	SELECT B.ISBN, R.dueDate FROM books AS B, bookreservation AS bR, reservations AS R WHERE B.ISBN = bR.ISBN AND bR.reservationID = R.reservationID AND B.bookStatus = 'Taken out'
SQ2	SELECT username, passwordHash, librarian FROM users ORDER BY BINARY username ASC
SQ3	SELECT username FROM users ORDER BY BINARY username ASC
SQ4A	SELECT ISBN, title, subTitle, genre, firstName, surname FROM books, authors WHERE books.authorID = authors.authorID AND bookStatus = "Available" ORDER BY title ASC
SQ4B	SELECT ISBN, title, subTitle, genre, firstName, surname FROM books, authors WHERE books.authorID = authors.authorID AND bookStatus = "Available" AND genre = %s ORDER BY title ASC
SQ5	SELECT ISBN, title, subTitle, firstName, surname, genre, starRating, datePublished, blurb FROM books AS B, authors AS A WHERE B.authorID = A.authorID AND ISBN = %s
SQ6	SELECT title, subtitle FROM books WHERE ISBN = %s
SQ7	SELECT reservationID FROM reservations WHERE username = %s AND dateOfOrder = %s AND timeOfOrder = %s
SQ8	SELECT u.username, u.firstName, r.reservationID, COUNT(br.ISBN) FROM Users u, Reservations r,bookReservation br WHERE u.username = r.username AND r.reservationID = br.reservationID AND br.ISBN IN (SELECT ISBN FROM Books WHERE bookStatus = %s) GROUP BY u.username, u.firstName, r.reservationID;

SQ9A	SELECT B.ISBN, B.title, B.subTitle, B.genre FROM books as B, reservations as R, bookreservation as BR WHERE B.ISBN = BR.ISBN AND BR.reservationID = R.reservationID AND R.reservationID = %s AND bookStatus = %s
SQ9B	SELECT dateOfOrder, timeOfOrder FROM reservations WHERE reservationID = %s
SQ10	SELECT U.username, COUNT(DISTINCT B.ISBN) FROM Users AS U, Reservations AS R, bookReservation as bR, Books as B WHERE U.username = R.username AND R.reservationID = bR.reservationID AND bR.ISBN = B.ISBN AND bookStatus = "Overdue" GROUP BY U.username
SQ11	SELECT B.ISBN, title, subTitle, dueDate FROM Books as B, bookReservation AS bR, Reservations as R, Users AS U WHERE B.ISBN = bR.ISBN AND bR.reservationID = R.reservationID AND R.username = U.username AND bookStatus = "Overdue" AND R.username = %s

Query ID	DML
UQ1	UPDATE books SET bookStatus = "Overdue" WHERE ISBN = %s
UQ2	UPDATE books SET bookStatus = "Reserved" WHERE ISBN=%s
UQ3	UPDATE books SET bookStatus = "Taken out" WHERE ISBN = %s
UQ4	UPDATE books SET bookStatus = "Available" WHERE ISBN = %s

Query ID	DML
IQ1	INSERT INTO Users (username, firstName, surname, passwordHash, librarian) VALUES(%s, %s, %s, %s, %s)
IQ2	INSERT INTO reservations(username, dateOfOrder, timeOfOrder, dueDate) VALUES(%s,%s, %s, %s)
IQ3	INSERT INTO bookreservation(reservationID, ISBN) VALUES(%s, %s)

Database structure

```
CREATE SCHEMA library;

CREATE TABLE Authors(
    authorID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    firstName VARCHAR(30) NOT NULL,
    surname VARCHAR(30)
);

CREATE TABLE Books(
    ISBN VARCHAR(13) NOT NULL,
    authorID INT NOT NULL,
    title VARCHAR(45) NOT NULL,
    subTitle VARCHAR(30),
    bookStatus VARCHAR(9) NOT NULL,
    starRating DECIMAL(3,2),
    genre VARCHAR(15) NOT NULL,
    blurb VARCHAR(1000),
    datePublished DATE NOT NULL,
    PRIMARY KEY (ISBN),
    FOREIGN KEY(authorID) REFERENCES Authors(authorID),
    CHECK(length(ISBN)=13),
    CHECK(bookStatus IN("Available","Reserved","Taken out","Overdue")),
    CHECK(starRating BETWEEN 0 AND 5),
    CHECK(genre IN("Non-fiction","Fantasy","Science fiction","Romance","Horror"))
);
```

```
CREATE TABLE Users(
    username VARCHAR(20) NOT NULL,
    firstName VARCHAR(30) NOT NULL,
    surname VARCHAR(30) NOT NULL,
    passwordHash VARCHAR(64) NOT NULL,
    librarian BOOLEAN NOT NULL,
    PRIMARY KEY(username),
    CHECK(LENGTH(username) BETWEEN 10 AND 20),
    CHECK(LENGTH(passwordHash) = 64)
);

CREATE TABLE Reservations(
    reservationID INT NOT NULL AUTO_INCREMENT,
    username VARCHAR(20) NOT NULL,
    dateOfOrder DATE NOT NULL,
    timeOfOrder TIME NOT NULL,
    dueDate DATE NOT NULL,
    PRIMARY KEY(reservationID),
    FOREIGN KEY(username) REFERENCES Users(username)
);

CREATE TABLE bookReservation(
    reservationID INT NOT NULL,
    ISBN VARCHAR(13) NOT NULL,
    PRIMARY KEY (reservationID, ISBN),
    FOREIGN KEY(reservationID) REFERENCES Reservations(reservationID),
    FOREIGN KEY(ISBN) REFERENCES Books(ISBN)
);
```

Initial data

Librarians

- INSERT INTO users VALUES ([Defined in table below])
- passwordHash has had font size changed for report formatting

username	firstName	surname	passwordHash	librarian
xX_BookReader_Xx	Dolores	Macleod	5ebf55299759c27912df24c4c946533068dcf49110c943323fc5a6af2a5a8d5f	1
maryCampbell53	Mary	Campbell	56a7a4494211643598143775 5d9fc5179e95621c70e2c98317387eef8ee1fc4f	1
agathaAndrews23	Agatha	Andrews	67764c07e6568bac5f5013fc5bea6ba382fa987634452fb8e2810725079d590b	1
johnAndrews22	John	Andrews	23f81c8eb98e26060acb24cc1abb28f8805597fc20824dedf52c11c332917034	1
seanMacavoy97	Sean	Macavoy	137e0627c4d46b407292bc0f9d389962b313b77d8224371918867d0404220fee	1

Authors

- INSERT INTO authors VALUES([Defined in table below])

authorID	firstName	surname
default	J.R.R.R.R	Token
default	Stephanie	Queen
default	J.K	Cowling
default	Mikaela	Crichton
default	Mitchel	Morichgo
default	George	Andwell
default	Robert Stewie	Alexson
default	Cressida	Hoodle
default	Diana Winnie	Janes
default	Ari C.	Clerk
default	Marie	Shely
default	Erpoor	Marya
default	Fronk	Sherbert
default	Frenz	Kofko
default	M.L	Gontmomery

Books

- Blurb font size has been lowered for report formatting
- Blurb has been shortened for report formatting
- INSERT INTO books VALUES ([Defined in table below]
- Table spans pages 53 - 55

ISBN	authorID	title	subTitle	bookStatus	starRating	Genre	Blurb	datePublished
9780261103344	1	The Clobbit	Null	Available	4.7	Fantasy	Blibo Boggins is a clobbit who enjoys an uncomfortable and uninspired life, rarely travelling further than his bedroom door in his home at Bog-end……	1937-09-21
9781444720723	2	The Shimmering	Null	Available	4.7	Horror	Manny is only 3 years old, but in the words of young Ms Halloran he is a 'shimmerer', slightly lit up with psychic current……	1977-01-28
9780575084346	2	The Blue Furlong	Null	Available	4.5	Horror	The Blue Furlong: those who run it sometimes return, because at the end of that run is the room in which sits Warm Hill Bastille's electric keyboard……	1996-08-29
9780192719980	7	Fortune tombolo	Null	Available	4.15	Horror	When Jane Falcons discovers a sleeping man's map it's not long before she's off to sea in search of nearby fortune……	1883-11-14
9780582060180	6	Twenty fourty-eight	Null	Available	4.45	Science fiction	In 2048 the world is divided into 15 parts, Oceania, Eastasia, Eurasia, Gnomeland, North Greenland, ……	1949-06-08
9780261103573	1	The comradeship of the halo	The lady of the halos part 1	Available	4.7	Fantasy	In a narcoleptic village in the Share, a young Clobbit is entrusted with a moderate task……	1954-07-29
9780261103580	1	The eighty-nine spires	The lady of the halos part 2	Available	4.62	Fantasy	The comradeship of the Halo is rived, Frogo and Slam continue their journey alone down the minor river Andiunuininunin……	1954-11-11
9780261103597	1	The departure of the queen	The lady of the halos part 3	Available	4.8	Fantasy	The battalions of the Dark Lady are lessening as her evil shadow spreads even slimmer, Men, Dwarves, Elves, Gnomes, Smurfs, ……	1955-10-20
9780439362139	3	Larry Snotter and the Mathmetician's pebble	Null	Available	4.79	Fantasy	In the non-magical human world - the world of 'Maggles' - Larry is well-known, treated like royalty by his foster parents ……	1997-06-26
9780747538486	3	Larry Snotter and Auditorium of Truths	Null	Available	4.63	Fantasy	Larry Snotter is a wizard. He is in Second year at Woghorts School of Wizardcraft and Witchery……	1998-07-02

9780747542155	3	Larry Snotter and the Free man of Azkabazkaba	Null	Available	4.89	Fantasy	Larry Snotter is a wizard! Along with Rin and Heromonis, his only friends, Larry is in his third year at Woghorts School of Wizardcraft and Witchery……	1999-07-08
9780747546245	3	Larry Snotter and the Chalice holding water	Null	Available	1.2	Fantasy	The spring break is dragging on and Larry Snotter doesn't want it to end and the school year to start……	2000-07-08
9780747569404	3	Parry Hotter and the Order of the Chicken	Null	Available	0.11	Fantasy	Dark times have come to Woghorts. After his cousin Durdley gets Dementia, Parry Hotter (Yes he decided to change his name deal with it) knows that Mort will stop trying to find him……	2003-06-21
9780747581086	3	Larry Snotter and the Cold blooded Princess	Null	Available	4.78	Fantasy	Ok so he changed his name back. It is the middle of the winter, there is a seasonal mist pressing against the windowpanes……	2005-07-16
9780747591054	3	Larry Snotter and the Lively Hallows	Null	Available	4.98	Fantasy	Larry is waiting in Puble Drive. The order of the Chicken is coming to escort him dangerously away without Mort and his Fanclub knowing……	2007-07-21
9780007523221	1	The Similarion	Null	Available	4.65	Fantasy	The Similarli were the imperfect gems created by Feieanor, most gifted of the Low Elves……	1977-09-15
9780007299263	9	Scream's Transport Home	Null	Available	4.8	Romance	In the land of Outgary, where, two-league boots and cloaks of translucency exist……	1986-04-01
9780099066101	10	1002	A medieval odyssey	Available	4.17	Science fiction	The discovery of a mud brown monolith on the moon leads to a manned expedition deep into the earth's core……	1968-04-08
9780520201798	11	Frunkenstain	or the ancient hercules	Available	4.6	Romance	Victor Frunkenstain is a scientist obsessed with generating life from lifeless matter……	1818-01-01
9781784752224	4	Devonian Land	Null	Available	4.99	Non-fiction	On a remote desert island where genetic engineers have created a Devonian game park……	1990-11-20

9781405226660	5	Battle Donkey	Null	Available	4.6	Non-fiction	In the deadly chaos of the First World War, one donkey witnesses the reality of battle from both sides of the trenches……	1982-5-28
9781784752231	4	Devonian Land 2	The Found Plane	Available	2.68	Science fiction	It is now six years since the public disaster at Devonian Land. Six years since the extraordinary dream of science and imagination……	1995-09-08
9780821223123	12	All is Loud in the eastern Headquarters	Null	Available	4.23	Romance	An antiwar novel set after World War I, it relies on Erpoor's personal experience to depict the era's broad happiness ……	1928-03-12
9781907054648	5	Klanspor	princess of dogs	Available	0.1	Non-fiction	Klanspor the dog first came to the Savill Inn in a Limousine - Jane Snott knows, because she was the one who drove her there……	2008-10-12
9780340960196	13	June	Null	Available	3.4	Science fiction	The sprite Meringue is the most command and least valuable element in the universe……	1965-08-01
9781473655324	13	June Messier	Null	Available	5.0	Science fiction	Twelve days after his loss against the House Harplonnen. Paul Artladies doesn't rule Arrrakkis as the emperor ……	1969-06-01
9783453435773	2	AT	Null	Available	2.2	Horror	A promise made forty-eight years ago calls eleven adults to reunite in Lewiston, Maine, where as teenagers they battled ……	1986-09-15
9780307348241	2	Juco	Null	Available	1.34	Romance	Inside a lively town in the outskirts of Maine, a monster is waiting.	1981-09-08
9780199238552	14	The metasmorphosisis	and other stories	Available	4.98	Non-fiction	With a blend of the everyday and the ordinary, Kofko thus begins his most famous novel, The metasmorhposisis……	1915-01-02
9788842516231	15	Annie of Blue Bables	Null	Available	4.99	Non-fiction	As soon as Annie Surely arrived at the snug, beige farmhouse called Blue Bables, she knew she wanted to stay for only a week.....	1908-06-13

Software

Program code

Extras file

```
from tkinter import *
from datetime import datetime
import textwrap
import hashlib

# Function to do a binary search on a 2D array, with given second index
def binary_search(array, target, index):
    position = -1
    found = False
    low = 0
    high = len(array) - 1

    while not found and low <= high:
        mid = (low + high) // 2

        if array[mid][index] == target:
            found = True
            position = mid

        elif array[mid][index] > target:
            high = mid - 1

        else:
            low = mid + 1

    return position
```

All the modules which are to be use in the Extras file are imported.

The binary_search function will perform a binary_search on a 2D array/array of tuples when given the index that it needs to search on. It acts the same as a normal binary search but just has array[mid][index] now so that comparisons are made on the correct item.

```
# Function to hash passwords
def hash_password(password):
    m = hashlib.sha256()

    password_mid = password.encode('utf-8')

    m.update(password_mid)

    return m.hexdigest()

#-----Procedures to use if the query returns nothing-----
def no_results(centre): # Used for borrower section
    title = Label(centre, text="Sorry currently there are no books available")
    title.pack()
    title.config(font="Helvetica 30 bold", bg = "beige")

def no_results_res(centre): # Used for librarian section
    title = Label(centre, text="There are currently no reservations")
    title.pack()
    title.config(font="Helvetica 30 bold", bg = "beige")
#---------------------------------------------------------
```

This section defines the hash_password function. This will be given a string parameter and will proceed to hash it using the sha256 algorithm. This is what the hashlib module is imported for. It creates a hashing object and then encodes the password using utf-8. After the encoding the updated password is sent into the hashing object. The hashing object then returns a hex string. Which is the hashed password.

The two other procedures in this section of code simply display their text onto a given window. In the main file with the app_details tuple. The second item in that tuple will be the window.
So, centre = app_details[1]
These are used for the different sections of the software to display the appropriate “No results” text.

```
# Procedure to display an error screen
def error_screen(app_details):
    # Destroy old window
    app_details[1].destroy()

    # Create new window
    root_error = Tk()
    root_error.title("Error")
    root_error.state("zoomed")

    # Display text
    message = Label(root_error, text="We are sorry but there has been an error with the database connection").pack()
    instruct = Label(root_error, text="Please close and reopen the software").pack()

    # Close connection if it exists
    if app_details[0]:
        app_details[0].close()

    font_all_widgets(root_error, ("Helvetica", 20))

    root_error.mainloop()

# Function to count the number of items in the user's cart
def count_cart(cart):
    items = 0
    for counter in range(len(cart)):
        if cart[counter] != "":
            items += 1
    return items
```

The error screen is defined here and gets passed the app_details tuple as a parameter. It then destroys the main window and creates a new window. It then displays the error message text onto the new screen. After this it checks if the connection exists, if so it closes the connection. If it doesn't exist then it does nothing. Finally, it gives a font to all the widgets on the screen.

The count_cart function sees how many non-blank items are in an array. This is just a basic count occurrences algorithm set to count everything that isn't a blank string. The cart array passed into it will have blank strings or book ISBNs as its items. The function will count the number of ISBNs in the cart and return the number of items.

```
#-----Formatting procedures-----
# Function to combine title and sub title if they both exist
def title_combine(title, sub_title):
    if sub_title is None:
        full_title = title
    else:
        full_title = title + "-" + sub_title
    return full_title

# Function to wrap text within given character amount
def wrap_text(text, width):
    full_title = textwrap.wrap(text, width)
    title_format = "\n".join(full_title)
    return title_format

# Procedure to change the bg colour of all widgets in a window/frame
def colour_all_widgets(centre, colour):
    for widget in centre.winfo_children():
        widget.config(bg=colour)
```

This section begins defining the procedures used for formatting the window.

The title_combine function gets two parameters passed in. This is the title and subtitle of a book as strings or None. The title will always exist as a string while the sub_title doesn't always exist. This function checks if the subtitle is None (doesn't exist). If so, then the full title that is returned is just the title parameter. If the sub_title parameter does exist then it combines the two parameters and returns the result.

The wrap_text function uses the textwrap module to create a new line on a string every X number of characters. The X number of characters is given by the width parameter. The textwrap module will ensure though that words are not broken up over lines. After adding the new lines ("\n"), it will return the modified string.

The colour_all_widgets procedure will be given a window/frame as the centre parameter and a string for the colour parameter. It then uses a fixed loop to go through all the widgets in the window/frame and sets all of their background colours to the colour parameter.

```
# Procedure to destroy all widgets in a frame
def delete_all_widgets(centre):
    for widget in centre.winfo_children():
        widget.destroy()

# Procedure to change the font of all the widgets in a window/frame
def font_all_widgets(centre, font_set):
    for widget in centre.winfo_children():
        widget.config(font=font_set)

# Procedure to convert date format from YYYY-MM-DD to DD-MM-YYYY
def convert_date(current):
    updated = datetime.strptime(str(current), "%Y-%m-%d").strftime("%d-%m-%Y")
    return updated

# Procedure to destroy all the frames in a window
def destroy_frames(window_frames):
    for frame in window_frames:
        frame.destroy()
#------------------------------
```

This section finishes defining the formatting procedures.

The delete_all_widgets procedure will be given a window/frame as the centre parameter. The procedure will then loop through the widgets in the window/frame and destroy the widgets in it. This is used to clear frames on the main window.

The font_all_widgets procedure will be given a window/frame as the centre parameter and a tuple containing font details/a string containing font details as the font_set parameter. It then loops though all the widgets in the window/frame and sets the font of all the widgets to the font_set parameter.

The convert_date function will convert the format of a given datetime object from YYYY-MM-DD to DD-MM-YYYY. It then returns this reformatted datetime object.

The destroy_frames procedure will be passed in an array of widgets (i.e. an array of frames). It will then loop through the array of widgets and destroy each widget.

Queries file

- The SQL and DML has been omitted since all of it is already in tables in the Database implementation section
- All of them will be stored like: “““SELECT * FROM users”””

```
import Library_Extras as LE
import mysql.connector

# Function to connect to the database and return a connection variable
def db_connect():
  try:
    conn = mysql.connector.connect(
      host='127.0.0.1',
      user="dolm2006",
      passwd="Sleeping16?",
      database="library",
    )
  except:
    return None
  else:
    return conn
```

This section of code imports the Library_Extras file and gives it the alias LE and the mysql.connector modules. These are used for the query execution modules and the db_connect modules respectively.

The db_connect function uses a try catch to try and create a connection variable to the database. If this succeeds then it will return the MySQL connection variable. If the connection fails it will return None for the connection variable. This will be used by the main file to create a connection and decide whether to continue the program or show an error_screen. Depending on if conn exists or not.

```
# Function for SELECT queries
def query_select(query, values, app_details):
    my_cursor = app_details[0].cursor()

    try:
        my_cursor.execute(query, values)
    except:
        LE.error_screen(app_details)
    else:
        my_result = my_cursor.fetchall()
        return my_result

# Procedure for INSERT queries
def query_insert(query, values, app_details):
    my_cursor = app_details[0].cursor()

    try:
        my_cursor.execute(query, values)
    except:
        LE.error_screen(app_details)
    else:
        app_details[0].commit()

# Procedure for UPDATE queries
def query_update(query, values, app_details):
    my_cursor = app_details[0].cursor()

    try:
        my_cursor.execute(query, values)
    except:
        LE.error_screen(app_details)
    else:
        app_details[0].commit()
```

This section defines all of the query execution modules.

They all define a cursor using the connection variables and start a try catch. Each module tries to execute the query using the cursor. If an error occurs then the error screen (from the Extras file) is displayed. Though if the query succeeds the query result is returned or the DML is committed to the database.

Main file

```
from tkinter import *
from tkinter import ttk, messagebox
from datetime import timedelta, datetime, date
import Library_Extras as LE
import Library_Queries as LQ

def main():
    global cart, app_details
    # Create user's cart, made global for accessibility
    cart = [""] * 3

    # Create connection to database
    connection = LQ.db_connect()

    # If connection is successful
    if connection:
        #Create the window
        root = Tk()
        root.title("Book and Borrow")
        root.state("zoomed")
        root.config(bg="beige")

        # Create tuple with connection and window details, made global for ease of use
        app_details = (connection, root)

        # Update any overdue books
        update_status()
        # Display the login/signup start screen
        start_menu()
        root.mainloop()
    # If connection fails
    else:
        # Temp fake window so error screen procedure can be used
        old = Tk()
        app_details = (connection, old)
        LE.error_screen(app_details)
```

Imports all of the modules and files to be used in this file. It gives the LE alias to the Extras file and the LQ alias to the queries file

The main procedure is what is called to initiate the software. It creates a connection variable using the Queries file function and if connection if successful it creates a window, if not successful it creates an error screen. Either way it defines a tuple containing the connection and window details, this tuple (app_details) is made global for ease of access. A cart array is also defined and made global.

```
# Procedure to change book status if they're out past their due date
def update_status():
    # Get all book's ISBN and due date that are taken out
    result = LQ.query_select(LQ.SQ1, None, app_details)

    # Get the current date
    today = date.today()

    # Loop for every book from the query
    for item in result:
        # If the due date has been passed then change book to be overdue
        if today >= item[1]:
            LQ.query_update(LQ.UQ1, (item[0],), app_details)
```

The update_status procedure uses SQ1 to get all the books which are taken out and their due dates. It then gets the current date from the system. It loops through every result from the query and compares the due date to current date. If the current date is greater than or equal to the due date, that book is set to “Overdue” using UQ1.

```
# Procedure to display details of start screen
def start_menu():
    #-----Create frames-----
    title_frame = Frame(app_details[1])
    title_frame.pack()

    start_frame = Frame(app_details[1])
    start_frame.pack()
    #-----------------------

    # Colour the background of the window as beige
    LE.colour_all_widgets(app_details[1], "beige")

    #-----Create text displays-----
    title_label = Label(title_frame, text="Book & Borrow", bg="beige")
    title_label.pack()
    title_label.config(font=("Helvetica", 45, "bold"))

    instruction_label = Label(start_frame, text="Please login or Sign up")
    instruction_label.pack(pady = 15)
    #-----------------------------

    # Colour background of the widgets as beige
    LE.colour_all_widgets(start_frame, "beige")

    #-----Buttons for user-----
    login_button = Button(start_frame, text = "Login",
                command = lambda: [start_frame.destroy(), login_screen()])
    login_button.pack(side = TOP, pady = 10)

    signup_button = Button(start_frame, text = "Sign up",
                 command = lambda: [start_frame.destroy(), sign_up_screen()])
    signup_button.pack(side = TOP, pady = 10)
    #-------------------------

    # Make all the rest of the text and buttons have a font and size
    LE.font_all_widgets(start_frame, ("Helvetica", 30))
```

The start_menu procedure will display all the details of the start menu screen. It creates frames to put all the details in. The frames can be easily destroyed and can keep the “Book & Borrow” title on the screen. It creates the text labels and colours their background to beige. Then it creates the buttons to the user can press which take the user to the corresponding screen. Finally, at the end it gives a font to all the text (except the title) and the buttons.

```
# Procedure to display details of login screen
def login_screen():
    global username # Make the username entered global, for later use at cart screen

    # Procedure for the validation of user login
    def validate_login():
        #-----Get the user's inputs-----
        username = user_username.get()
        # Hash the password as soon as its retrieved
        password_hash = LE.hash_password(user_password.get())
        #------------------------------

        # Find if username exists in the database, using binary search algorithm
        position = LE.binary_search(result, username, 0)

        # If username is found in database
        if position != -1:
            #-----Get the other user details from the query result-----
            stored_password = result[position][1]
            user_type = result[position][2]
            #-----------------------------------------------------------

            # If entered password matches database password
            if stored_password == password_hash:
                # Create welcome message for user depending on the type of user
                welcome_message = "Welcome " + username + " to the " + ('librarian' if user_type == 1 else 'Book & Borrow') + "
system"

                # Create popup with that welcome message
                messagebox.showinfo("Access granted", welcome_message)

                # Destroy all the current frames in the window
                LE.delete_all_widgets(app_details[1])

                # Send the user to the appropriate scrren depending on user type
                if user_type == 1:
                    librarian_menu()
                else:
                    catalogue_screen()

            # Else then the password is incorrect
            else:
                messagebox.showerror("Error", "Password is incorrect")

        # Else then the username doesn't exist
        else:
            messagebox.showerror("Error", "Username does not exist")
```

This section of code begins the definition of the login_screen procedure. This will display the login screen to the window.

It starts off by setting the username variable to be global. This is for use in the cart and checkout screens and will be equal to the user inputted value for username.

It then defines the validate_login procedure. This was moved into the login_screen module instead of it being separate like in the Pseudocode design on page 28. This retrieves the user entered details. It instantly hashes the password that the user has inputted as only a hashed password can be compared to the one in database.

It then uses the binary_search function from the Extras file to search if the user entered username exists in the query result. Then if position DOES NOT = -1 (it does exist in query result) it sets variables of the query result for both password and user type, using the position found by the binary_search(). If the position DOES = -1, then it creates a popup saying “Username does not exist”.

It checks to see if the entered password and the database password are the same (comparing the two hashed strings). Then if they match it creates a welcome message popup depending on the type of user they are. Then destroys everything on screen and takes the user to the appropriate screen depending on their user type. If the passwords don’t match then a popup saying “Password is incorrect” is shown to the user.

```
# Enables the submit button if both entry boxes have text in them
def enable_submit_button(*args):
    if user_username.get() and user_password.get():
        submit_button["state"] = NORMAL
    else:
        submit_button["state"] = DISABLED

# -----Create frames-----
title_frame = Frame(app_details[1])
title_frame.pack(pady=30)

login_frame = Frame(app_details[1])
login_frame.pack()
# ----------------------

# Colour every frame's backgrounds as beige
LE.colour_all_widgets(app_details[1], "beige")

# Use query SQ2, Gets all the current user's usernames and passwords
result = LQ.query_select(LQ.SQ2, None, app_details)

#-----Create text, entry boxes and submit button for user-----
title_label = Label(title_frame, text = "Login", bg = "beige", font = ("Helvetica", 30, "underline"))
title_label.pack()

username_label = Label(login_frame, text = "USERNAME: ", bg = "beige")
username_label.pack()

user_username = Entry(login_frame, width=30)
user_username.pack(pady = 10)

password_label = Label(login_frame, text="PASSWORD:", bg="beige")
password_label.pack()

user_password = Entry(login_frame, show="*", width=30)
user_password.pack(pady = 10)

submit_button = Button(login_frame, text="SUBMIT", command=validate_login, state=DISABLED)
submit_button.pack(pady = 30)
#------------------------------------------------------------

# Button to return to start screen
return_button = Button(login_frame, text = "Return",
            command = lambda: [LE.delete_all_widgets(app_details[1]), start_menu()])
return_button.pack(pady = 10)

# Change the font of all labels, entry boxes and buttons
```

```
    LE.font_all_widgets(login_frame, ("Helvetica", 25))

    # Bind the entry boxes to an event for every key release
    user_username.bind("<KeyRelease>", enable_submit_button)
    user_password.bind("<KeyRelease>", enable_submit_button)
```

This section of code defines the rest of the login_screen procedure.

It creates a procedure which is used for the event of text being entered into the text boxes. This checks to see if it's possible to retrieve text from the boxes. If it is possible then it enables the submit button. But if it can't get text (the value = None) then it changes/keeps the submit button disabled.

Then the section creates frames, colour the frames and creates all the text boxes and labels for user inputs. It also retrieves the query result of SQ2 using the query_select function from the Queries file.

Finally, this section binds the text boxes to the previously mentioned "text being entered into the text boxes" event (defined om page 68). Then it sets the event to be for every release of the keyboard keys. This will check to see if text is in every box every time a key is released.

```
# Procedure to display the sign up screen information
def sign_up_screen():
    # Procedure to validate user details
    def validate_signup():
        #-----Get all the user's inputs-----
        first_name, last_name, username = user_f_name.get(), user_s_name.get(), user_username.get()
        password, password_conf = user_password.get(), user_password_conf.get()
        #-----------------------------------
        # Search to see if the username already entered already exists
        position = LE.binary_search(result, username, 0)

        # If username is not unique, display popup saying so
        if position > -1:
            messagebox.showerror("ERROR: Username", "Username already in use")

        # If username isn't in correct range (10-20 characters), display popup
        elif not (10 <= len(username) <= 20):
            messagebox.showerror("ERROR: Username", "Username must be 10-20 characters")

        # If the password isn't long enough (12 characters minimum), display popup
        elif len(password) < 12:
            messagebox.showerror("ERROR: Password", "Password must be at least 12 characters ")

        # If the password entry and re-entry don’t match, display popup
        elif password != password_conf:
            messagebox.showerror("ERROR: Password", "Passwords must match")

        # Else everything is correct, user can be created
        else:
            #-----Insert new user into database-----
            values = (username, first_name, last_name, LE.hash_password(password), False)
            LQ.query_insert(LQ.IQ1, values, app_details)
            #---------------------------------------

            # Confirmation popup
            messagebox.showinfo("Confirmed", "Welcome new user\nPlease login now")

            #-----Remove the frames used here-----
            sign_up_frame.destroy()
            title_frame.destroy()
            #-------------------------------------

            # Display the login screen to make them sign in fully
            login_screen()
```

This section begins defining the sign_up_screen procedure. This procedure displays all the signup screen details to the window.

It then defines the validate_signup procedure this was also moved into the larger module like the login screen implementation on page 66 and unlike the design on page 30.

It begins by getting all the user entered text from the text boxes. After this it uses the binary_search from the Extras file to search for the entered username in the query result (check if exists in database). Then if position > -1 (if username already exists in database), it displays a popup telling the user "Username already exists".

Then it checks to see if the entered username is between 10 and 20 characters (inclusive with 10 and 20). Then if it isn't 10-20 characters it displays a popup saying "Username must be 10-20 characters".

Then it checks to see if the entered username is less than 12 characters. If so it displays a popup saying "Username must be at least 12 characters".

Finally, it checks to see if the two entered passwords don't match. If true then it displays a popup saying "Passwords must match".

If all of these are false (the user details are all valid), it creates a user using IQ1 and creates a popup. Finally, it then brings the user to the login screen to make them login fully.

```
    # Keeps the submit button disabled until there is text in every entry box
    def enable_submit_button():
        if user_f_name.get() and user_s_name.get() and user_username.get() and user_password.get() and
user_password_conf.get():
            submit_button["state"] = NORMAL
        else:
            submit_button["state"] = DISABLED

    #-----Create frames and style them-----
    title_frame = Frame(app_details[1])
    title_frame.pack(pady=30)

    sign_up_frame = Frame(app_details[1])
    sign_up_frame.pack()

    LE.colour_all_widgets(app_details[1], "beige")
    #---------------------------------------

    # Get all the usernames currently in the database
    result = LQ.query_select(LQ.SQ3, None, app_details)

    # Create a title for this screen
    title_label = Label(title_frame, text="Sign up", bg="beige", font=("Helvetica 30 underline"))
    title_label.pack()

    #-----Create labels, entry box and submit button for user-----
    f_name_text = Label(sign_up_frame, text="First name:", bg="beige").pack()
    user_f_name = Entry(sign_up_frame)
    user_f_name.pack(pady=5)

    s_name_text = Label(sign_up_frame, text="Surname:", bg="beige").pack()
    user_s_name = Entry(sign_up_frame)
    user_s_name.pack(pady=5)

    username_text = Label(sign_up_frame, text="Username:", bg="beige").pack()
    user_username = Entry(sign_up_frame)
    user_username.pack(pady=5)

    password_text = Label(sign_up_frame, text="Password:", bg="beige").pack()
    user_password = Entry(sign_up_frame, show="*")
    user_password.pack(pady=5)

    conf_password_text = Label(sign_up_frame, text="Confirm Password:", bg="beige").pack()
    user_password_conf = Entry(sign_up_frame, show="*")
    user_password_conf.pack(pady=5)

    submit_button = Button(sign_up_frame, text = "SUBMIT",
                           command = lambda: validate_signup(),
```

```
                    state = DISABLED)
submit_button.pack(pady=30)
#------------------------------------------------------------

# Button to return to start screen
return_button = Button(sign_up_frame, text = "Return",
            command = lambda: [LE.delete_all_widgets(app_details[1]), start_menu()])
return_button.pack()

# Change the font of all the widgets (except the title)
LE.font_all_widgets(sign_up_frame, ("Helvetica", 25))

# Bind all the text boxes the user can use to an event of key release
text_boxes = [user_f_name, user_s_name, user_username, user_password, user_password_conf]
for widget in text_boxes:
    widget.bind("<KeyRelease>", lambda event: enable_submit_button())
```

This section of code finishes defining the sign_up_screen proecedure.

The enable_submit_button procedure checks to see if a non-None value is in all of the text boxes. If true then the submit button is enabled. But if a None value exists for any of the text boxes then the submit button remains disabled.

Then it creates all the frames, labels, text boxes and buttons for the signup screen and gives them all a font.

Finally, it creates an array of all the text boxes. Then it loops through every text box in the array and binds them to an event. The event occurs every time a key on the keyboard is released. The event called is the enable_submit_button procedure. So, every time a key is released it checks to see if all boxes are filled and changes submit button.

```
#Procedure to display catalogue screen
def catalogue_screen():

  # Event procedure for if user scrolls the scrollbar
  def on_configure(event):
    canvas.configure(scrollregion=canvas.bbox("all"))

  # Event procedure for if user changes the genre to filter by
  def selection_change(*args):
    # Get value in the drop-down box
    genre = value_inside.get()

    # If no genre is selected
    if genre == "None" or genre == "Genre Filter":
      # Title of page
      title_label.config(text="Available books:")

      # All available books with no genre filter used
      result = LQ.query_select(LQ.SQ4A, None, app_details)

    # Otherwise use the selected genre
    else:
      # Title of page with the specified genre to filter by
      title_label.config(text="Available books for: \n" + genre)

      # All available books using the specified genre as a filter
      result = LQ.query_select(LQ.SQ4B, (genre,), app_details)

    # Delete all the previously shown books
    LE.delete_all_widgets(inner_frame)

    # Generate buttons for books if there are any from query
    if len(result) != 0:
      generate_buttons(result)
    else:
      LE.no_results(inner_frame)

  # Procedure to generate buttons for all available books
  def generate_buttons(result):
    col = 0
    rows = 1

    # Loop for every available book
    for book in result:
      #-----Create display for the current book button-----
      isbn = book[0]
      title = LE.title_combine(book[1], book[2])
```

```
        genre = book[3]
        author = book[4] + " " + book[5]

        text_display = LE.wrap_text(title,20) + "\nBy: " + author + "\n\nGenre: " + genre
        #-------------------------------------------

        # Create button for current book
        button_book = Button(inner_frame, text=text_display,
                        command = lambda isbn = isbn: [LE.destroy_frames((top_frame, bottom_frame)),
                                        book_details_screen(isbn)],
                        width = 20, height = 10)
        button_book.grid(row=rows, column=col, padx=5, pady=5)

        #-----Create grid of buttons-----
        col += 1
        # If maximum columns is reached,  move to next row
        if col == 4:
            col = 0
            rows += 1
        #--------------------------------

    # Change the font for all the book buttons
    LE.font_all_widgets(inner_frame, ("Helvetica", "25"))
```

This section defines the procedures used in the catalogue_screen procedure.

It starts off by defining the 2 event procedures. The on_configure event is for every time the scrollbar is used, it updates the canvas (where all the books are).

The selection_change event updates the variable which holds the value of the dropdown box and is called every time the dropdown box value is altered. This event also handles the regeneration of the books with the new filter value. First it checks if there is no filter selected and it displays the normal title and uses SQ4A. Otherwise it displays the title + the genre filter option and uses SQ4B using the filter option. After this it delete all the old book buttons. Then it checks if any results have been returned, if none have been returned it displays “There are currently no books” and if any have been returned then it calls the generate_buttons procedure.

The generate_buttons procedure is defined here and it handles creating the grid of buttons depending on the query result from the selection_change event. It loops through the whole query result and creates a button for it. This button will call the book_details_screen of the corresponding book (by using the ISBN) when clicked. It then uses .grid() to add it to the screen. Then the columns value is incremented by 1, once the columns value reaches the value of 4, it is reset to 0 and rows is incremented. This creates a grid of buttons for all the books that are available for the currently selected genre filter.

```
    # Create frame for title, cart button and genre filter drop-down box
top_frame = Frame(app_details[1], bg = "beige")
top_frame.pack(side = TOP, fill = X)

# Set drop-down choices
choices = ["None", "Non-fiction", "Horror", "Science fiction", "Romance", "Fantasy"]

# Make a variable to store the option in the drop-down menu and set it to Genre filter initially
value_inside = StringVar(top_frame)
value_inside.set("Genre Filter")

# Create the drop-down box
genre_menu = OptionMenu(top_frame, value_inside, *choices)

# Create the title
title_label = Label(top_frame, text="Available books:", bg="beige")

# Create a cart button
cart_button = Button(top_frame, text= "Cart",
                     command = lambda:
                     [LE.delete_all_widgets(app_details[1]), cart_screen()])

# Format the three previous widgets
for widget in [genre_menu, title_label, cart_button]:
    widget.pack(side = LEFT)
    widget.pack_configure(fill=BOTH, expand=True)

# Change drop-down variable when another option is selected
value_inside.trace_add("write", selection_change)

# Change the font of all the previous widgets
LE.font_all_widgets(top_frame, ("Helvetica", "25"))

# Create a frame for the catalogue
bottom_frame = Frame(app_details[1])

# Create a canvas in the newly made frame
canvas = Canvas(bottom_frame)

# Create a scrollbar in the newly made canvas
scroll = Scrollbar(bottom_frame, orient="vertical", command=canvas.yview, width=30)
scroll.pack(side="right", fill="y")

# Format and display the canvas
canvas.pack(side=LEFT, fill="both", expand=True)
canvas.configure(yscrollcommand=scroll.set)
canvas.config(bg="beige")
```

```
    # Create a frame inside the canvas
    inner_frame = Frame(canvas)

    # Colour the new frame
    inner_frame.config(bg="beige")

    # Create a window using the new frame
    canvas.create_window((0, 0), window=inner_frame, anchor="nw")

    # Bind the new canvas to an event for whenever the scrollbar is moved
    inner_frame.bind("<Configure>", on_configure)

    # Get all available books, no genre filter applied
    result = LQ.query_select(LQ.SQ4A, None, app_details)

    # Display buttons for all books if there are any returned
    if len(result) != 0:
        generate_buttons(result)
    else:
        LE.no_results(inner_frame)

    # Add the bottom_frame containg the canvas to the screen
    bottom_frame.pack(side=TOP, fill=BOTH, expand=True)
```

This finishes the catalogue_screen procedure definition.

It creates all the frames and details of the catalogue screen. It also makes a variable to hold the currently selected genre filter value and uses the selection_change event.
To make an area work with a working scrollbar:

- A canvas is made in between the top and bottom frames
- An “inner” frame is made inside of this canvas
- A canvas window is then created in that “inner” frame
- It then binds the scrollbar to the on_configure event, so it activates every time the scrollbar is moved.
- This makes it so the scrollbar scrolls down the window, displaying the full grid of book buttons

Finally, by default it uses SQ4A to generate the full catalogue. If the query result returns nothing it displays a message saying “There are currently no books”. But if there are query results it uses the generate_buttons procedure to create a grid of buttons.

```
# Procedure to display the book details screen information after being given a specific ISBN
def book_details_screen(isbn):

    # Procedure for when the user adds a book to their cart
        def add_book():
        #-----Linear search the global cart for next available space-----
        found = False
        n = 0
        while not found and n < 3:
            # If the current cart space is blank, add the book to cart, and display a popup
            if cart[n] == "":
                found = True
                cart[n] = isbn
                messagebox.showinfo("Confirmed", "Book has been added")

            # Otherwise if the cart ISBN is the same as the on they try to add, display a poup saying so
            elif cart[n] == isbn:
                messagebox.showwarning("Cart error", "Book already in cart")
                # Leave the conditional loop
                found = True

            # Otherwise increment n
            else:
                n += 1

        # If blank space or ISBN not found in cart
        if not found:
            messagebox.showerror("Cart error", "Cart is full")
        #-------------------------------------------------------------

    #-----Create frames-----
    top_frame = Frame(app_details[1])
    top_frame.pack(fill=X)

    detail_frame = Frame(app_details[1])
    detail_frame.pack()

    bottom_frame = Frame(app_details[1], bg="beige")
    bottom_frame.pack(fill=X)
    #-----------------------

    # Get all the book details for this specific book
    result = LQ.query_select(LQ.SQ5, (isbn,), app_details)

    # Create return button to go back to the catalogue
    return_button = Button(top_frame, text = "Return to catalogue",
                           command = lambda:
```

```
                    [LE.delete_all_widgets(app_details[1]), catalogue_screen()])

    # Create a title for the screen
    title_label = Label(top_frame, text="Book details", bg="beige")

    # Create a button to go to the cart screen
    cart_button = Button(top_frame, text = "Cart",
                         command = lambda:
                           [LE.delete_all_widgets(app_details[1]),cart_screen()])

    # Change the font of the title, cart button and return button
    LE.font_all_widgets(top_frame, ("Helvetica", "30"))

    # Format the previous widgets and add them to the screen
    for widget in [return_button, title_label, cart_button]:
        widget.pack(side = LEFT)
        widget.pack_configure(fill = BOTH, expand = True)
```

This section begins definition of the book_details_screen procedure.

The add_book procedure has been moved into the main procedure unlike the design on page 33, this make for easy use as no parameters need to be passed into it.
The procedure performs a linear search on the global cart array to search for a blank spot or the value of the current ISBN. If either are found then found is set to true. If a blank spot is found then the ISBN is added to the cart and a popup displayed. If the ISBN value is found to be already in the cart, the ISBN is not added to cart and a popup is displayed and found is set to true.
If neither is found in that spot then the index is incremented by 1. Then if found is false (not found means not true), then a “Cart is full” popup is displayed as none of the criteria is met.

Then this section also starts creating all the book details screen details. It uses SQ5 to get the details of the current ISBN and creates the frames. It the creates the return button, title and cart button and fonts them.

```
#-----Retrieve and format all the book detail text-----
title = LE.title_combine(result[0][1], result[0][2])

author = result[0][3] + " " + result[0][4]

genre = result[0][5]

rating = str(result[0][6]) + "/5"

# Change date into correct format (YYYY-MM-DD to DD-MM-YYYY)
date = LE.convert_date(result[0][7])

blurb = result[0][8]

# Wraps blurb within width of 100 characters per line
blurb_format = LE.wrap_text(blurb, 75)
#---------------------------------------------------------

# Get the maximum line length of the blurb, will be used for creatiion of treeview table
line_length = max(len(line) for line in LE.textwrap.wrap(blurb, 100))

# Create an array of tuple to hold the data to go in the treeview table
data = [("Title:", title), ("By: ", author), ("Genre: ", genre), ("Star rating: ", rating),
      ("Date published:", date), ("Blurb: ", blurb_format)]

# Create a style for the treeview table and set the font s for the columns
style = ttk.Style()
style.configure("Treeview.Heading", font=('Helvetica', 15))
style.configure("Treeview", font=('Helvetica', 15), rowheight=50)

# Create a treeview table to display book details
table = ttk.Treeview(detail_frame, columns=("Headings", "Data"), show="headings")

# Set the column width to be 8 times the maximum line length
table.column("Data", width = line_length * 8)

# Insert all the data into the treeview table
for i, (heading, data_value) in enumerate(data, start=len(data)):
   table.insert("", i, values=(heading, data_value))

# Display the treeview table and format the frame
table.grid(row=0, column=0, sticky="nsew")
detail_frame.columnconfigure(0, weight=1)
detail_frame.rowconfigure(0, weight=1)

# Create a button to add the current book to cart
add_to_cart_button = Button(bottom_frame, text="Add to cart", command=lambda: add_book(), font=("Helvetica", "25"))
add_to_cart_button.pack(side=TOP, anchor=N, pady=5)
```

This section finishes defining the book_details_screen procedure.

This displays all the rest of the details of this screen. It uses a treeview table to display all the book details, so therefore needs a themed tkinter (ttk) style to alter its text. So, it defines both of these. It also formats the blurb of the book to make it so that it fights within a 100-character limit per line.
It inserts all the book data into the treeview table and adds it to the screen.

Finally, it makes the add_to_cart button that when clicked will call the add_book procedure defined on page 78. This will try to add the book currently being viewed to the global cart array.

```
# Procedure to display the cart screen information
def cart_screen():

    # Procedure to remove the selected book from the cart
    def remove_book(cart, n):
        cart[n] = ""

        # Confirmation popup
        messagebox.showinfo("Cart", "Book was removed")

        # Refresh the cart display
        LE.delete_all_widgets(bottom_frame)
        display_books()

    # Procedure to display all books in the cart
    def display_books():

        # For every item in cart
        for counter in range(len(cart)):
            book_frame = Frame(bottom_frame, bg = "beige")
            book_frame.pack()

            # If there is no book in this slot, display a message saying so
            if cart[counter] == "":
                book_text = Label(book_frame, text = "\t" + str(counter + 1) + ". No book selected")
                book_text.pack(side=TOP, pady = 5)
                book_text.config(font = ("Helvetica", 15), bg = "beige")

            # Otherwise, display book
            else:
                # Get the titles of the current book from the database
                result = LQ.query_select(LQ.SQ6, (cart[counter],), app_details)

                # Create a display message for the current book
                display = "\t" + str(counter + 1) + "." + LE.title_combine(result[0][0], result[0][1])

                # Label for the current book
                book_text = Label(book_frame, text = display, bg = "beige")
                book_text.pack(side = LEFT, pady = 5)

                # Create a button which is used to remove current book from the cart
                remove_button = Button(book_frame, text="X",
                                       command = lambda counter=counter:
                                          [remove_book(cart, counter)])
                remove_button.pack(side = RIGHT, pady = 5)

                LE.font_all_widgets(book_frame, ("Helvetica", 25))
```

```
#-----Create frames-----
top_frame = Frame(app_details[1])
top_frame.pack(fill=X)

bottom_frame = Frame(app_details[1], bg = "beige")
bottom_frame.pack(fill=BOTH)
#-----------------------
```

This section begins definition of the cart_screen procedure. This procedure will display all the details of the cart screen to the window.

The remove_book procedure is defined here and is called whenever a book removal button is pressed. It's given the cart array and an integer value as formal parameters. It sets the value of the given array position to "" (makes it a blank spot). It then creates a popup telling the users that the book they removed has been removed. It then refreshes the books in cart by deleting all the books and buttons in the bottom frame and then calling the display_books() procedure.

The display_books procedure will display the title and subtitle of every book in the user's cart. It loops through the whole cart and creates a frame for each loop iteration. Then if the cart space is blank (the value is ""), it displays index + "No book selected" in that loop iterations frame.
If the value of the cart is not blank it calls SQ6 to get the title and subtitle of the ISBN stored in that cart space. It then creates text in that loop iterations frame using the SQ6 results. After that beside the book text it creates a removal button that will call the remove_book procedure and destroy the current loop iteration's frame when clicked.

It then creates the frames for use in this screen.

```
    # Create button to return to the catalogue
    return_button = Button(top_frame, text = "Return to catalogue",
                           command = lambda:
                             [LE.delete_all_widgets(app_details[1]), catalogue_screen()])

    # Create a title for this screen
    title_label = Label(top_frame, text="Your current cart:", bg = "beige")

    # Create a button for the user checking out their books
    checkout_button = Button(top_frame, text = "Checkout",
                             command = lambda:
                               [LE.delete_all_widgets(app_details[1]), checkout_screen()],
                             state = DISABLED)

    # Format the widgets in the title frame
    for widget in [return_button, title_label, checkout_button]:
        widget.pack(side = LEFT)
        widget.pack_configure(fill=BOTH, expand=True)

    LE.font_all_widgets(top_frame, ("Helvetica", 30))

    items = LE.count_cart(cart)
    if items > 0:
        checkout_button.config(state = NORMAL)

    # Call the procedure to display the user's current cart books
    display_books()
```

This section finishes defining the cart_screen procedure.

It starts by creating the return button that when clicked will take the user to the catalogue screen.

It then creates the title of the page and the checkout button. The checkout button will start as disabled (meaning that a user cannot click on it). The checkout button will call the checkout_screen procedure when clicked.

It then formats and fonts these created details.

Then it finds the number of items in the user's cart using the Extras file function If the number of items is greater than 0 then the checkout button is made clickable again. This makes it so the user is only capable of checking out when they actually have books in their cart.

Finally, it calls the display_books procedure to show the user's current cart.

```
# Procedure to display checkout screen information
def checkout_screen():
    global username

    # Create a frame
    checkout_frame = Frame(app_details[1], bg = "beige")
    checkout_frame.pack()

    # get current date and time and calculate when the due date will be
    current_date = datetime.now().date()
    current_time = datetime.now().time()
    due_date = current_date + timedelta(days=28)

    # Insert a new reservation with these details into the database
    values_reserve = (username, current_date, current_time, due_date)
    LQ.query_insert(LQ.IQ2, values_reserve, app_details)

    # Get the newly made reservation ID
    values_id = (username, current_date, current_time)
    result = LQ.query_select(LQ.SQ7, values_id, app_details)

    # Seperate out the reservation ID for easier use
    reservation_id = result[0][0]

    # Count how many items the user had in their cart
    items = LE.count_cart(cart)

    # Create a title
    title_label = Label(checkout_frame, text="Confirmed\n")
    title_label.pack()

    # Calculate and display the time to prepare these books
    time_label = Label(checkout_frame, text="Your books will be ready in:\n" + str((items * 5) + 2) + " minutes\n")
    time_label.pack()

    # Calculate and display the cost per day for overdue books
    fee_label = Label(checkout_frame, text="The overdue fees will be:\n£" + str(items * 0.2) + " per day\n")
    fee_label.pack()

    # Display the due date of the books in this order
    due_date_label = Label(checkout_frame, text="The book(s) are due on: " + LE.convert_date(due_date) + "\n\n")
    due_date_label.pack()
```

```
# Loop for every book in the users cart
    for ISBN in range(len(cart)):

        # If there is an item in this cart spot
        if cart[ISBN] != "":
            # Change the book to resered
            LQ.query_update(LQ.UQ2, (cart[ISBN],), app_details)

            # Insert a bookReservation of the current book in the newly made reservation
            values_to_insert = (reservation_id, cart[ISBN])
            LQ.query_insert(LQ.IQ3, values_to_insert, app_details)

    # Create a label telling the user to go to the library and pck up their books
    instruction_label = Label(checkout_frame, text="Please exit the program and come to the library to pick up your
books\n\n")
    instruction_label.pack(pady=10)

    LE.colour_all_widgets(checkout_frame, "beige")

    # Create a button to destroy the window as this is the end of the borrower section
    exit_button = Button(checkout_frame, text = "Exit",
                         command = lambda: [app_details[1].destroy(), app_details[0].close()])
    exit_button.pack(pady = 10)

    LE.font_all_widgets(checkout_frame,("Helvetica", 25))
    title_label.config(font = ("Helvetica", 30, "underline"))
```

This section defines the checkout_screen procedure. This procedure will display all the details of the checkout screen on to the window.

It gets the current date and time from system and then uses the datetime module to calculate the due date by adding 28 days to the current date. It then uses IQ2 to create a new reservation using the date and time details and the global username (from login). After that it uses SQ7 to get the reservation ID of the reservation that was just made by IQ2.

Then it calls the Extras count_cart function to find the number of items in the user's cart. It uses this number to calculate the preparation time and the overdue fees per day. It displays these calculations and the due date to the screen.

Then it loops through every item of the cart. Then it checks if the current cart space isn't blank, and if so uses UQ2 to update the book's status. After that it also creates a bookReservation using IQ3 with that book.

Finally, it creates all the extra checkout screen details including the exit button. The exit button will close the window and close the database connection when clicked.

```
# Procedure to display the options for a librarian to do
def librarian_menu():
    #-----Create frames-----
    title_frame = Frame(app_details[1], bg = "beige")
    title_frame.pack()

    options_frame = Frame(app_details[1], bg = "beige")
    options_frame.pack()
    #----------------------

    #-----Create title and prompt for librarian-----
    title_label = Label(title_frame, text="Welcome Librarian")
    title_label.pack(pady = 10)
    title_label.config(font = ("Helvetica", 30, "bold"), bg = "beige")

    instruction_label = Label(options_frame, text="What would you like to do?")
    instruction_label.pack(pady = 5)
    instruction_label.config(bg = "beige")

    #-----------------------------------------------

    #-----Create button options for user-----
    browse_button = Button(options_frame, text = "Browse books",
                           command = lambda:
                          [messagebox.showinfo("Confirmed","Book browing selected"), LE.delete_all_widgets(app_details[1]),
catalogue_screen()])
    browse_button.pack(side = TOP, pady = 10)

    manage_button = Button(options_frame, text = "Manage books",
                           command = lambda:
                           [messagebox.showinfo("Confirmed","Book managing selected"), LE.delete_all_widgets(app_details[1]),
actions_menu()])
    manage_button.pack(side = TOP, pady = 10)
    #---------------------------------------

    LE.font_all_widgets(options_frame, ("Helvetica", 25))
```

This section defines the librarian_menu procedure. This displays all the librarian menu screen details to the window.

It creates all the text and the buttons for the librarian menu.
The buttons:

- Browse – Takes user to catalogue screen
- Manage – Takes user to action menu screen

```
# Procedure to display the reservations types to librarian
def actions_menu():
    #-----Create frames-----
    top_frame = Frame(app_details[1], bg = "beige")
    top_frame.pack(fill=X)

    options_frame = Frame(app_details[1], bg = "beige")
    options_frame.pack(fill=X)
    #-----------------------

    return_button = Button(top_frame, text = "Return",
                    command = lambda:
                    [LE.delete_all_widgets(app_details[1]), librarian_menu()])
    return_button.pack(side = LEFT)

    title_label = Label(top_frame, text="Please select an option:\t")
    title_label.pack(side=TOP, anchor=N)
    title_label.config(bg = "beige")

    #-----Button options for the libraian-----
    to_prepare_button = Button(options_frame, text = "View reservations to prepare",
                        command = lambda:
                        [LE.delete_all_widgets(app_details[1]), to_prepare()])
    to_prepare_button.pack(pady = 5)

    due_books_button = Button(options_frame, text = "View due reservations",
                        command = lambda:
                        [LE.delete_all_widgets(app_details[1]), due_reserves()])
    due_books_button.pack(pady = 5)

    overdue_button = Button(options_frame, text = "View overdue reservations",
                      command = lambda:
                      [LE.delete_all_widgets(app_details[1]), overdue_reservations()])
    overdue_button.pack(pady = 5)
    #----------------------------------------

    LE.font_all_widgets(top_frame, ("Helvetica", 30))
    title_label.config(font = ("Helvetica", 30, "bold"))
    LE.font_all_widgets(options_frame, ("Helvetica", 25))
```

This section defines the action_menu procedure. This displays all the action menu screen details.

Each of the buttons created will take the user to the corresponding reservation status screen.
To_prepare_button – Takes user to prepare reservations screen
Due_books_button – Takes user to due reservations screen
Overdue_button - Takes user to overdue reservations screen

```
# Procedure to display details of reservations to prepare
def to_prepare():
    # Get reservations with status "Reserved" from the database
    result = LQ.query_select(LQ.SQ8, ("Reserved",), app_details)

    # Create the top frame for navigation
    top_frame = Frame(app_details[1])
    top_frame.pack(fill=X)

    # Create the frame to display reservations
    reservations_frame = Frame(app_details[1])
    reservations_frame.pack()

    # Set the background color of the entire window
    LE.colour_all_widgets(app_details[1], "beige")

    # Create a button to return to the previous menu
    return_button = Button(top_frame, text="Return",
                           command=lambda: [LE.delete_all_widgets(app_details[1]),
                                            actions_menu()])
    return_button.pack(side=LEFT)

    # Create a label to display the title
    title_label = Label(top_frame, text="Here are all the reservations which are to be prepared:")
    title_label.pack(side=TOP)
    title_label.config(bg="beige")

    # Set the font for the title and other widgets
    LE.font_all_widgets(top_frame, ("Helvetica", 30))
    LE.font_all_widgets(reservations_frame, ("Helvetica", 25))

    # Bolden the title label
    title_label.config(font=("Helvetica", 30, "bold"))

    # Check if there are any reservations
    if len(result) != 0:
        # Iterate through each reservation
        for i in range(len(result)):
            username = result[i][0]
            first_name = result[i][1]
            reservation_id = result[i][2]
            num_of_books = result[i][3]

            # Construct the text to display for each reservation
            text_display = "Username: " + str(username) + "\t\tFirst name: " + str(first_name) + "\nReservation ID: " +
str(reservation_id) + "\t\tNumber of books: " + str(num_of_books)
```

```
        # Create a button for each reservation with a callback to prepare_books
        reservation_button = Button(reservations_frame, text=text_display,
                                    command=lambda ID=reservation_id:
                                    [LE.delete_all_widgets(app_details[1]), prepare_books(ID)])
        reservation_button.pack(side=TOP, pady=5)
        reservation_button.config(font=("Helvetica", 25))
    else:
        # Display a message if there are no reservations
        LE.no_results_res(reservations_frame)
```

This section defines the entirety of the to_prepare procedure. This creates all the details of the To prepare reservations screen.

It uses SQ8 to get all the reservations (and their details) which have a “Reserved book”. It then checks if there are any results returned, if there aren’t then it displays a message saying “There are currently no reservations”. However, if there are results returned then it loops through every query result and creates a button for it.
The button contains text which is made from using the query result. When this button is clicked it will call the prepare_books procedure using that reservation’s reservation ID.

Note: The SQ8 call also uses “Reserved” as a parameter for the book status as SQ8 is re used in the due_reserves procedure, except that uses “Taken out”

```
# Procedure to display books to prepare
def prepare_books(reservation_id):
    # Get reserved books and reservation date/time from the database
    values = (reservation_id, "Reserved")
    reserved_books = LQ.query_select(LQ.SQ9A, values, app_details)
    date_time_order = LQ.query_select(LQ.SQ9B, (reservation_id,), app_details)

    #-----Create frames-----
    top_frame = Frame(app_details[1])
    top_frame.pack(fill=X)

    books_frame = Frame(app_details[1])
    books_frame.pack()

    time_frame = Frame(top_frame)
    time_frame.pack(side=RIGHT)
    #-----------------------

    # Set the background color of the entire window
    LE.colour_all_widgets(app_details[1], "beige")

    # Display reservation date and time
    date_label = Label(time_frame, text="Date of reservation: " + LE.convert_date(date_time_order[0][0]))
    date_label.pack()

    time_label = Label(time_frame, text="Time of reservation: " + str(date_time_order[0][1]))
    time_label.pack()

    # Set background color for frames
    LE.colour_all_widgets(top_frame, "beige")
    LE.colour_all_widgets(time_frame, "beige")

    # Create a button to return to the previous menu
    return_button = Button(top_frame, text="Return",
                           command=lambda: [LE.delete_all_widgets(app_details[1]), to_prepare()])
    return_button.pack(side=LEFT)
    return_button.config(font=("Helvetica", 30))

    # Display reserved books and options to change status
    for i in range(len(reserved_books)):
        current_frame = Frame(books_frame, bg="beige")
        current_frame.pack()

        isbn = reserved_books[i][0]
        genre = reserved_books[i][3]
        title = LE.wrap_text(LE.title_combine(reserved_books[i][1], reserved_books[i][2]), 35)

        # Construct text to display for each reserved book
```

```
    text_display = "ISBN: " + str(isbn) + "\tGenre: " + str(genre) + "\nTitle: " + str(title)
    book_label = Label(current_frame, text=text_display, bg="beige")
    book_label.pack(pady=5, side=LEFT)

    # Button to change book status to "Taken out"
    change_button = Button(current_frame, text="Change to Taken out",
                           command=lambda search=(isbn,), book_frame=current_frame: [messagebox.showinfo("Confirmed", "Book changed to Taken out"),
                                                                                     book_frame.destroy(),
                                                                                     LQ.query_update(LQ.UQ3, search, app_details)])
    change_button.pack(side=RIGHT)

    LE.font_all_widgets(current_frame, ("Helvetica", 25))

  LE.font_all_widgets(time_frame, ("Helvetica", 30))
```

This section defines the whole prepare_books procedure. This displays the whole To prepare books screen.

It's given a reservation ID parameter and uses this in two queries.
These queries are:

- SQ9A – Gets all the book details of books in that reservation
- SQ9B – Gets the date and time of order of the books

Note: The SQ9 queries also are given "Reserved" for the book status as these queries are re used in due_books()

It then creates display text for the date and time values and displays them in the top right.

Then it loops through all the results from SQ9A and creates a frame for each one. It creates text inside this frame which holds the details of that specific book. It then creates a Change status button beside it in the frame. When this button is clicked it will destroy its corresponding frame and call UQ3 to change the corresponding book's status.

```
# Procedure to display taken out reservations
def due_reserves():
    #-----Create frames-----
    top_frame = Frame(app_details[1])
    top_frame.pack(fill=X)

    reservations_frame = Frame(app_details[1])
    reservations_frame.pack()
    #-----------------------

    # Set the background color of the entire window
    LE.colour_all_widgets(app_details[1], "beige")

    # Create a button to return to the previous menu
    return_button = Button(top_frame, text="Return",
                           command=lambda: [LE.delete_all_widgets(app_details[1]),
                                            actions_menu()])
    return_button.pack(side=LEFT)

    # Create a label to display the title
    title_label = Label(top_frame, text="Here are all the reservations which are currently 'Taken out'", bg="beige")
    title_label.pack(side=TOP)

    # Get reservations with status "Taken out" from the database
    result = LQ.query_select(LQ.SQ8, ("Taken out",), app_details)

    # Check if there are any reservations
    if len(result) != 0:
        # Iterate through each reservation
        for i in range(len(result)):
            username = result[i][0]
            first_name = result[i][1]
            reservation_id = result[i][2]
            num_of_books = result[i][3]

            # Construct the text to display for each reservation
            text_display = "Username: " + str(username) + "\t\tFirst name: " + str(first_name) + "\nReservation ID: " +
str(reservation_id) + "\t\tNumber of books: " + str(num_of_books)

            # Create a button for each reservation with a callback to due_books
            reservation_button = Button(reservations_frame, text=text_display,
                                        command=lambda ID=reservation_id: [LE.delete_all_widgets(app_details[1]), due_books(ID)])
            reservation_button.pack(side=TOP)
            reservation_button.config(font=("Helvetica", 25))
    else:
        # Display a message if there are no reservations
        LE.no_results_res(reservations_frame)
```

```
    LE.font_all_widgets(top_frame, ("Helvetica", 30))
    title_label.config(font=("Helvetica", 30, "bold"))

    LE.colour_all_widgets(app_details[1], "beige")
```

This section defines the due_reserves procedure. This displays all the details of the Due reservations screen.

This acts the same way as the to_prepare procedure (see comment on page 90, to see how it works). The only difference is that it uses “Taken out” for the query search value instead of “Reserved”. This makes it so this screen displays all the reservations (with their details) for reservations with “Taken out” books.

```
# Procedure to display taken out books
def due_books(ID):
    # Get books with status "Taken out" for the given reservation ID
    values = (ID, "Taken out")
    books = LQ.query_select(LQ.SQ9A, values, app_details)

    # Get reservation date and time
    date_time = LQ.query_select(LQ.SQ9B, (ID,), app_details)

    #-----Create frames------
    top_frame = Frame(app_details[1])
    top_frame.pack(fill=X)

    taken_frame = Frame(app_details[1])
    taken_frame.pack()
    #------------------------

    # Set the background color of the entire window
    LE.colour_all_widgets(app_details[1], "beige")

    # Create a frame for displaying reservation date and time
    time_frame = Frame(top_frame, bg="beige")
    time_frame.pack(side=RIGHT)

    # Create a button to return to the previous menu
    return_button = Button(top_frame, text="Return",
                           command=lambda: [LE.delete_all_widgets(app_details[1]), due_reserves()])
    return_button.pack(side=LEFT)

    # Display reservation date and time
    date_label = Label(time_frame, text="Date of reservation: " + str(LE.convert_date(date_time[0][0])))
    date_label.pack()

    time_label = Label(time_frame, text="Time of reservation: " + str(date_time[0][1]))
    time_label.pack()

    LE.colour_all_widgets(time_frame, "beige")

    # Set font for the return button
    return_button.config(font=("Helvetica", 30))

    # Set font for the time frame
    LE.font_all_widgets(time_frame, ("Helvetica", 30))

    # Display taken out books and options to change status
    for i in range(len(books)):
        current_frame = Frame(taken_frame, bg="beige")
        current_frame.pack()
```

```
    isbn = books[i][0]
    genre = books[i][3]
    title = LE.wrap_text(LE.title_combine(books[i][1], books[i][2]), 35)

    # Construct text to display for each taken out book
    text_display = "ISBN: " + str(isbn) + "\tGenre: " + str(genre) + "\nTitle: " + str(title)
    book_label = Label(current_frame, text=text_display, bg="beige")
    book_label.pack(side=LEFT)

    # Button to change book status to "Available"
    available_button = Button(current_frame, text="Change to available",
                              command=lambda Isbn=isbn, FRAME=current_frame:
                              [messagebox.showinfo("Confirmed", "Book made available"), FRAME.destroy(),
                               LQ.query_update(LQ.UQ4, (Isbn,), app_details)])
    available_button.pack(side=RIGHT)

    LE.font_all_widgets(current_frame, ("Helvetica", 30))
```

This section defines the due_books procedure. This will display all the details of the Due books screen.

This acts in the same way as the prepare_books procedure (see comments on page 92 for explanation). The difference being that it uses “Taken out” as the search criteria for book status instead of “Reserved”.
This results in this screen showing all the book in a reservation which are “Taken out” with status change buttons. When these buttons are clicked they’ll call UQ4 and change the book’s status.

```
# Procedure to display overdue users and how many books they have
def overdue_reservations():
    #-----Create frames-----
    top_frame = Frame(app_details[1])
    top_frame.pack(fill=X)

    overdue_frame = Frame(app_details[1])
    overdue_frame.pack()
    #-----------------------

    # Set the background color of the entire window
    LE.colour_all_widgets(app_details[1], "beige")

    # Create a button to return to the previous menu
    return_button = Button(top_frame, text="Return",
                           command=lambda: [LE.delete_all_widgets(app_details[1]),
                                            actions_menu()])
    return_button.pack(side=LEFT)

    # Get overdue reservations from the database
    result = LQ.query_select(LQ.SQ10, None, app_details)

    # Create a label to display the title
    title_label = Label(top_frame, text="Here are users and how many overdue books they have:", bg="beige")
    title_label.pack(side=TOP)

    # Set font for labels
    LE.font_all_widgets(top_frame, ("Helvetica", 30))
    title_label.config(font=("Helvetica", 30, "bold"))

    # Check if there are any overdue reservations
    if len(result) != 0:
        # Iterate through each result
        for i in range(len(result)):
            username = result[i][0]
            overdue_books_count = result[i][1]

            # Construct text to display for each user
            text_display = "Username: " + username + "\tNumber of overdue books: " + str(overdue_books_count) + "\n\tClick to
view books"

            # Create a button for each user with a callback to overdue_books
            book_button = Button(overdue_frame, text=text_display,
                                 command=lambda c_username=(username,): [LE.delete_all_widgets(app_details[1]),
overdue_books(c_username)])
            book_button.pack(side=TOP, pady=5)
```

```
        # Set font for labels in the overdue_frame
        LE.font_all_widgets(overdue_frame, ("Helvetica", 25))
    else:
        result_label = Label(overdue_frame, text = "There are no users with overdue books", bg = "beige")
        result_label.pack()
        result_label.config(font = ("Helvetica",25))
```

This section defines the overdue_reservation procedure. This will display the details of the overdue reservation/users screen.

It uses SQ10 to get all the user's usernames along with the number of overdue books they have. If no results are returned a message saying "There are currently no users with overdue books" is shown. However, if results are returned then it loops through all the results creating a button for each user. When this button is pressed it will call the overdue_books procedure of the specified user.

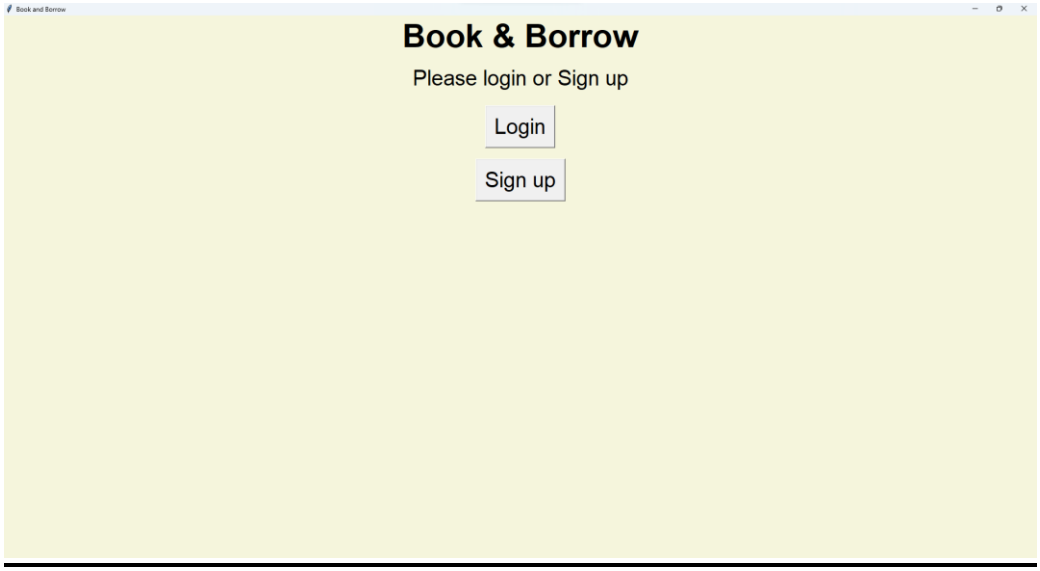
```
# Procedure to display all overdue books of a specific user
def overdue_books(c_username):
    #-----Create frames-----
    top_frame = Frame(app_details[1])
    top_frame.pack(fill=X)

    overdue_frame = Frame(app_details[1])
    overdue_frame.pack()
    #-----------------------

    # Set the background color of the entire window
    LE.colour_all_widgets(app_details[1], "beige")

    # Create a button to return to the previous menu
    return_button = Button(top_frame, text="Return",
                           command=lambda: [LE.delete_all_widgets(app_details[1]), overdue_reservations()])
    return_button.pack(side=LEFT)

    # Set font for labels in the top frame
    LE.font_all_widgets(top_frame, ("Helvetica", 30))

    # Get overdue books for the given username from the database
    result = LQ.query_select(LQ.SQ11, c_username, app_details)

    # Iterate through each overdue book
    for i in range(len(result)):
        current_frame = Frame(overdue_frame, bg="beige")
        current_frame.pack()

        isbn = result[i][0]
        title = LE.title_combine(result[i][1], result[i][2])
        due_date = result[i][3]

        # Construct text to display for each overdue book
        text_display = "ISBN: " + isbn + "\tTitle: " + title + "\nDue date: " + str(due_date) + "\nClick to make available"

        # Create a button for each overdue book with a callback to make_available
        book_button = Button(current_frame, text=text_display,
                             command=lambda ISBN=(isbn,), current=current_frame:
                             [messagebox.showinfo("Confirmed", "Book status changed"), current.destroy(),
                              LQ.query_update(LQ.UQ4, ISBN, app_details)])
        book_button.pack(side=TOP, pady=5)

        LE.font_all_widgets(current_frame, ("Helvetica", 25))

# Call main procedure
main()
```

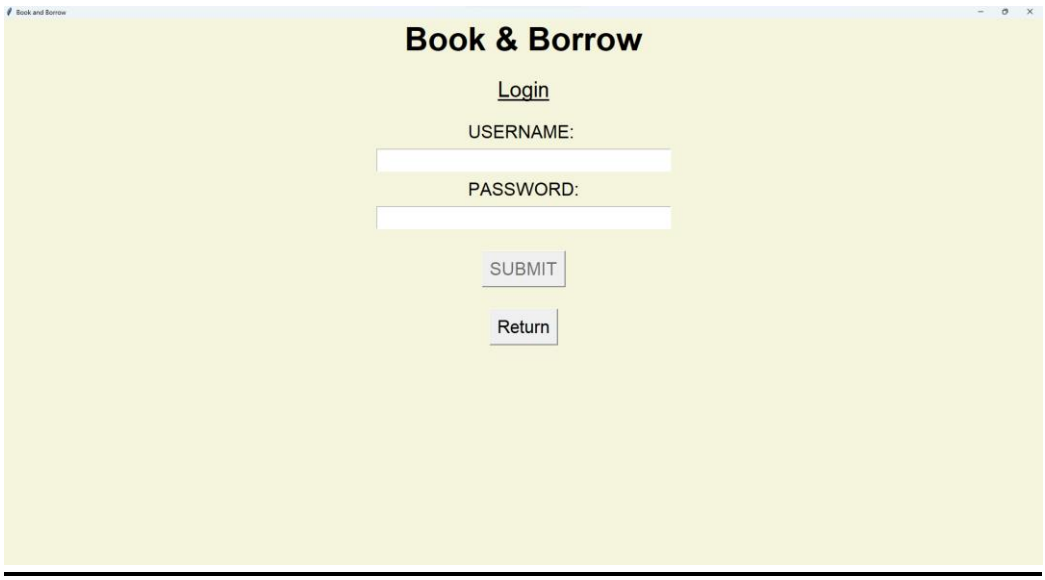
This section of code defines the overdue_books procedure and finally calls the main() procedure to begin the software.

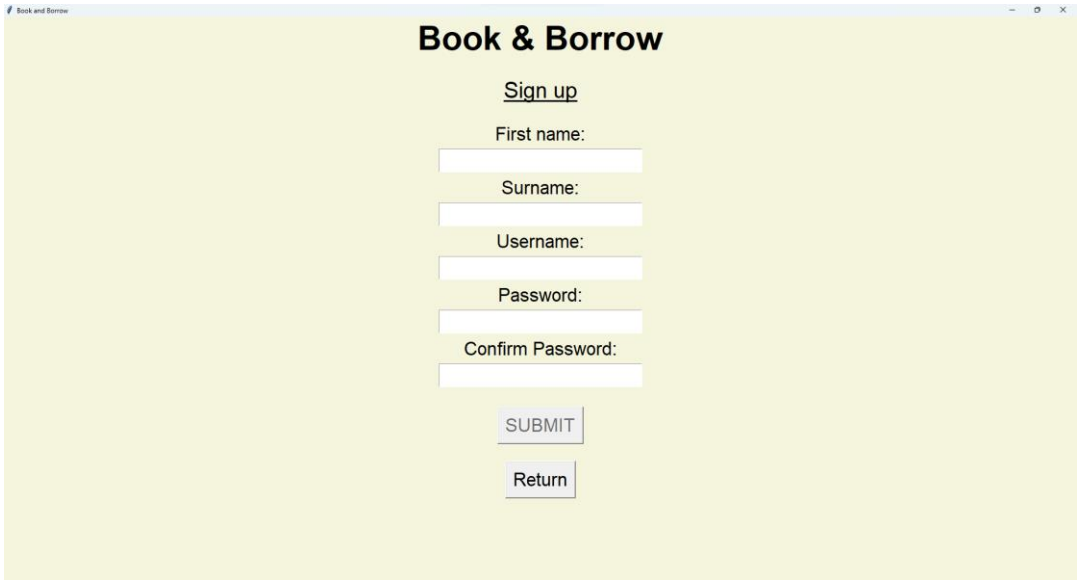
The overdue_books procedure displays all the details of the overdue books screen. It starts by creating some of these details and then it uses SQ11 using the selected user's username. This will return all of that user's overdue books. Then it loops through every result of the query and creates a button for each of them. Each button is created in its own frame. Once the button is clicked it will destroy its own frame and call UQ4 for the specific ISBN that it holds.

Finally, it calls the main procedure and begins the software.

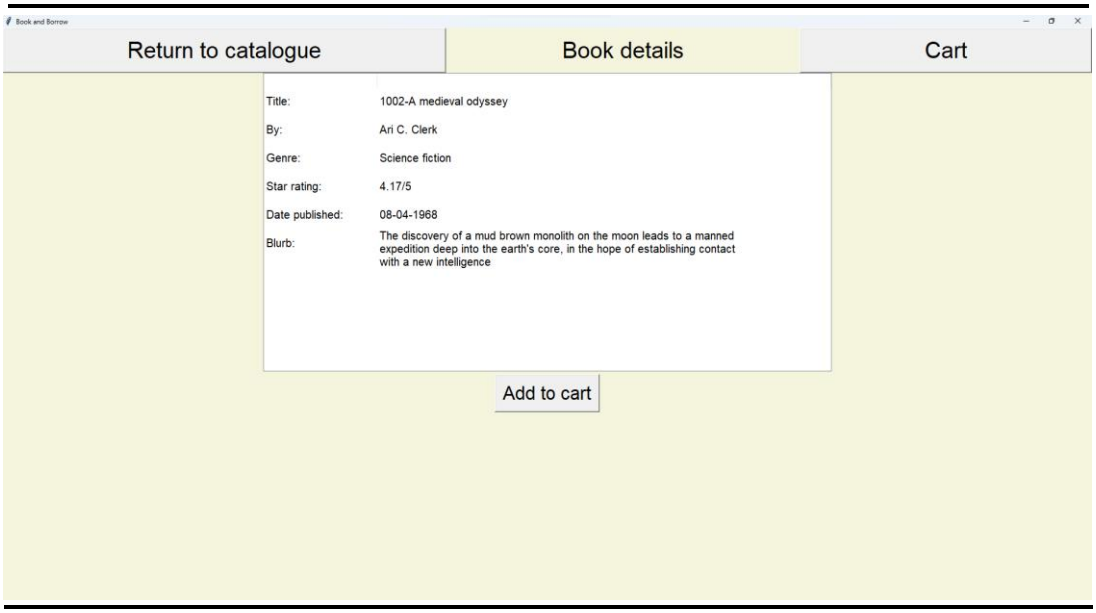
Screenshots of all screens

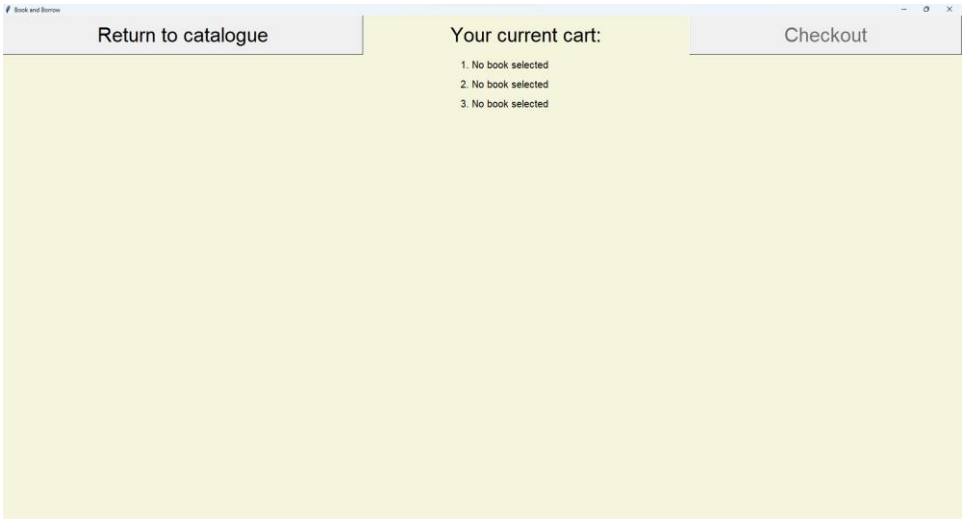


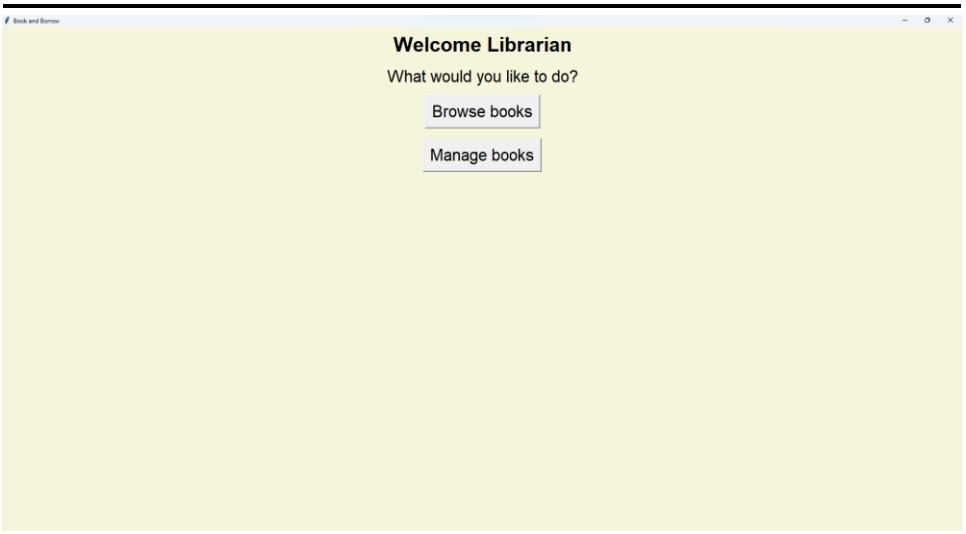


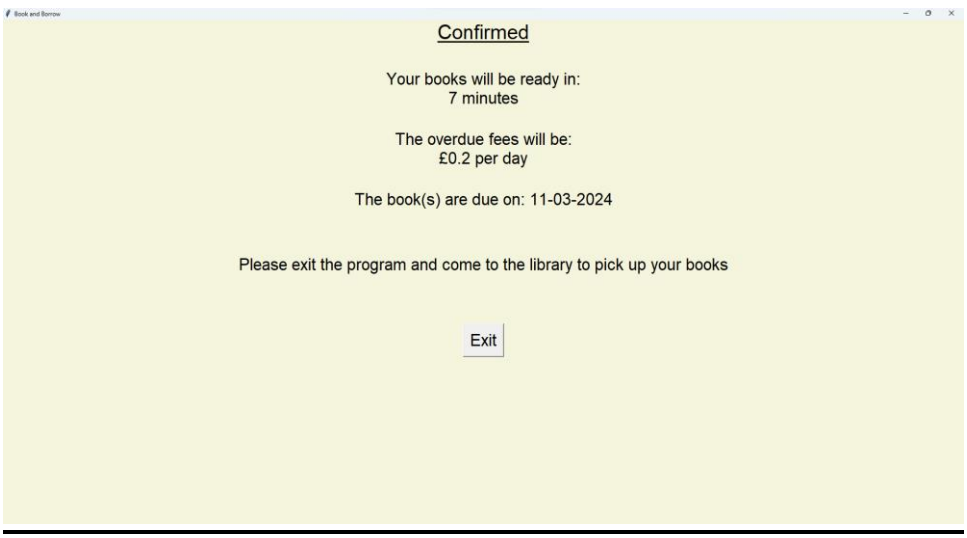

Book and Borrow
Book & Borrow
Sign up
First name:
Surname:
Username:
Password:
Confirm Password:
SUBMIT
Return

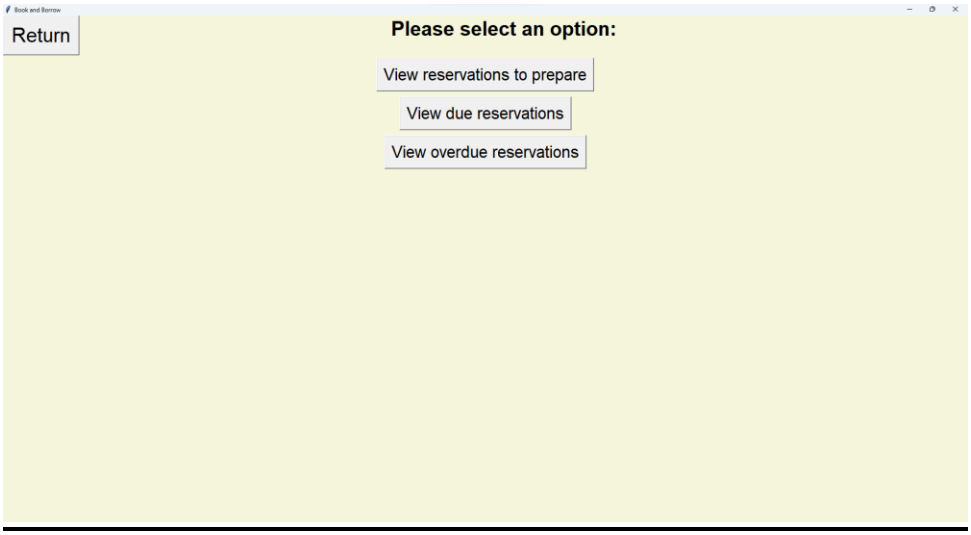

Book and Borrow
Genre Filter
Available books:
Cart
1002-A medieval odyssey
By: Ari C. Clerk
Genre: Science fiction
All is Loud in the eastern Headquarters
By: Erpoor Marya
Genre: Romance
Annie of Blue Bables
By: M.L Gontmomery
Genre: Non-fiction
AT
By: Stephanie Queen
Genre: Horror
Battle Donkey
By: Mitchel Morichgo
Genre: Non-fiction
Devonian Land
By: Mikaela Crichton
Genre: Non-fiction
Devonian Land 2- The Found Plane
By: Mikaela Crichton
Genre: Science fiction
Fortune tombolo
By: Robert Stewie Alexson
Genre: Horror

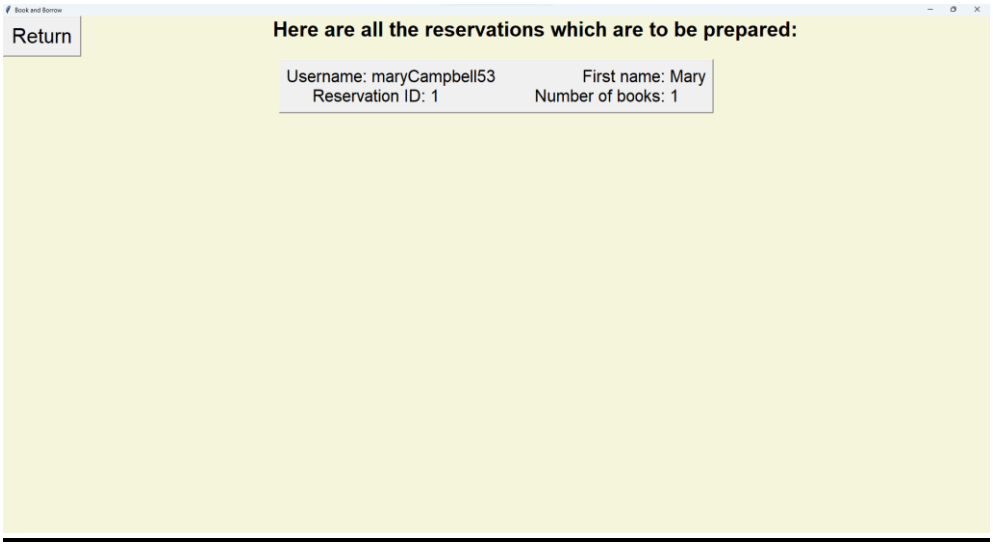

Book and Borrow
Return to catalogue
Book details
Cart
Title: 1002-A medieval odyssey
By: Ari C. Clerk
Genre: Science fiction
Star rating: 4.17/5
Date published: 08-04-1968
Blurb: The discovery of a mud brown monolith on the moon leads to a manned expedition deep into the earth's core, in the hope of establishing contact with a new intelligence
Add to cart

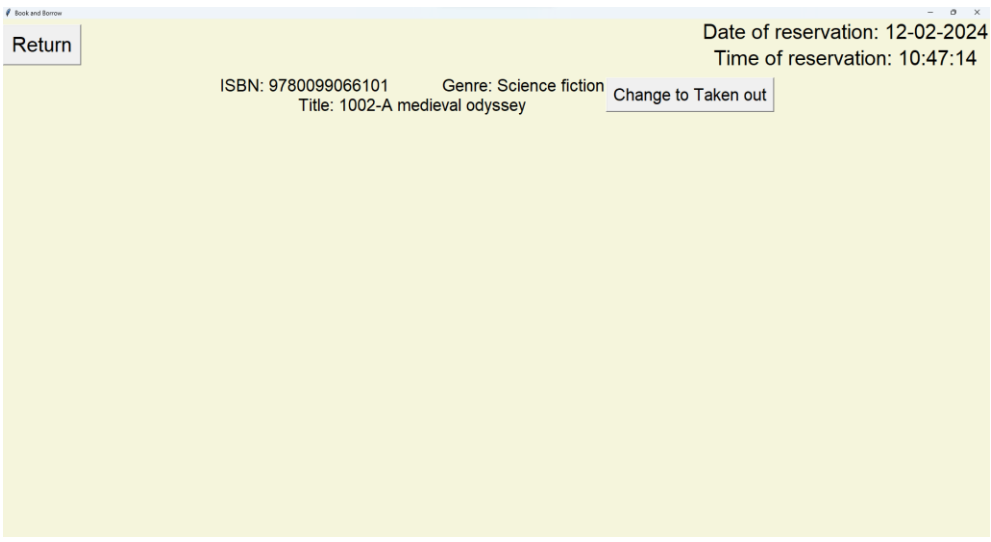

Book and Borrow
Return to catalogue
Your current cart:
Checkout
1. No book selected
2. No book selected
3. No book selected

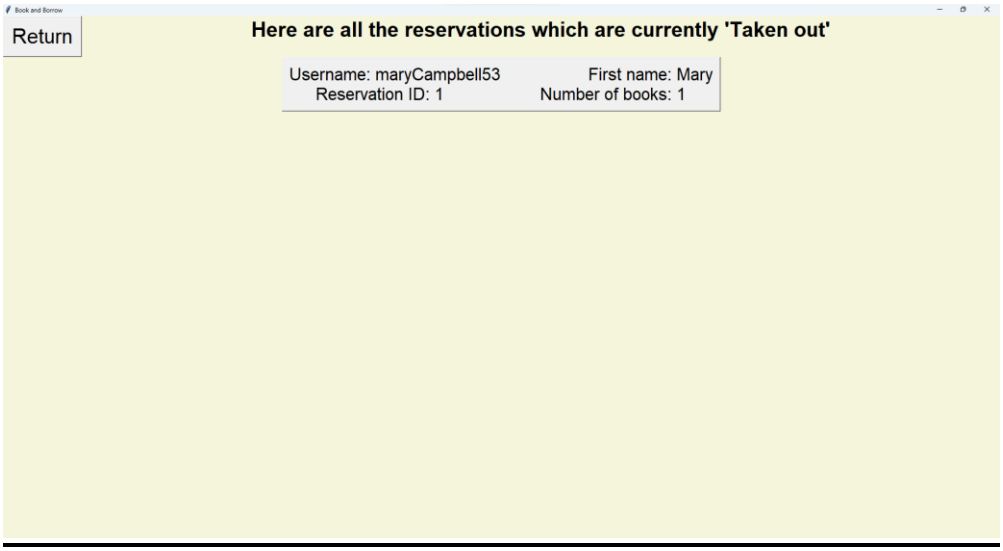

Book and Borrow
Welcome Librarian
What would you like to do?
Browse books
Manage books

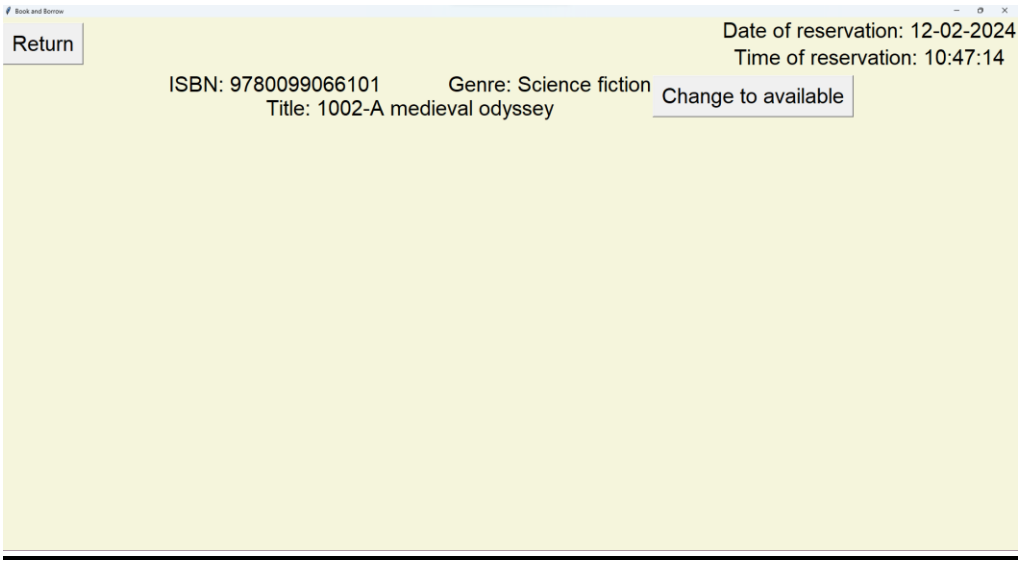

Book and Borrow
Confirmed
Your books will be ready in:
7 minutes
The overdue fees will be:
£0.2 per day
The book(s) are due on: 11-03-2024
Please exit the program and come to the library to pick up your books
Exit

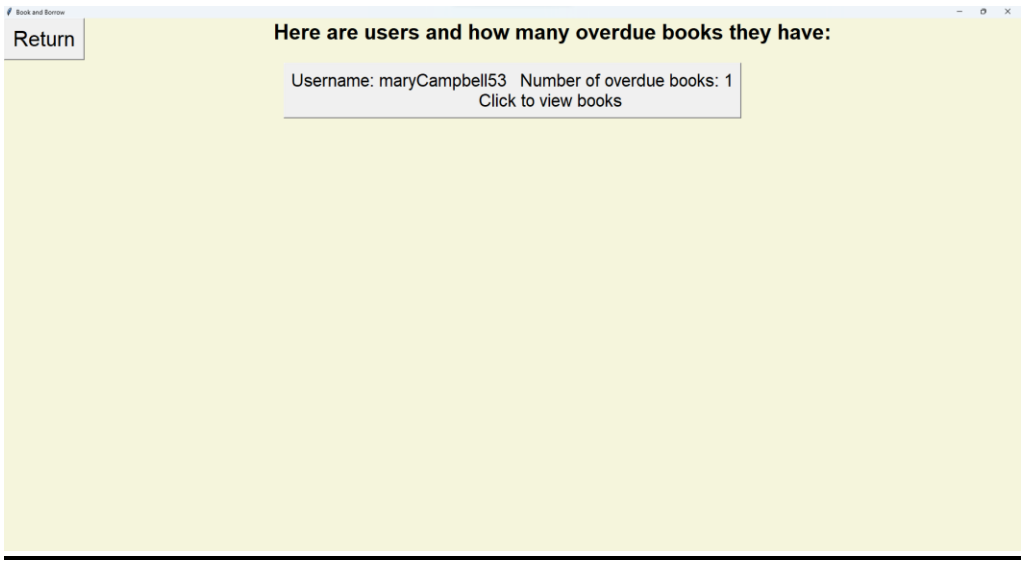

Book and Borrow
Return
Please select an option:
View reservations to prepare
View due reservations
View overdue reservations


Book and Borrow
Return
Here are all the reservations which are to be prepared:
Username: maryCampbell53
First name: Mary
Reservation ID: 1
Number of books: 1


Book and Borrow
Return
Date of reservation: 12-02-2024
Time of reservation: 10:47:14
ISBN: 9780099066101
Genre: Science fiction
Change to Taken out
Title: 1002-A medieval odyssey


Book and Borrow
Return
Here are all the reservations which are currently 'Taken out'
Username: maryCampbell53
First name: Mary
Reservation ID: 1
Number of books: 1


Book and Borrow
Return
Date of reservation: 12-02-2024
Time of reservation: 10:47:14
ISBN: 9780099066101
Genre: Science fiction
Change to available
Title: 1002-A medieval odyssey


Book and Borrow
Return
Here are users and how many overdue books they have:
Username: maryCampbell53
Number of overdue books: 1
Click to view books


Book and Borrow
Return
ISBN: 9780099066101 Title: 1002-A medieval odyssey
Due date: 2024-03-11
Click to make available


Error
We are sorry but there has been an error with the database connection
Please close and reopen the software

Log of ongoing testing

Date found: 18/11/23
Description: Database error caused by adding a new user through sign up
Date fixed: 18/11/23
Solution: Altered the software to correctly length check the username and password. The software and database had slightly different checks for these.
Software: username <10, Database: username <= 10.
Similar for password.

Date found: 25/11/23
Description: Books weren't able to be added to the database
Date fixed: 25/11/23
Solution: The sizes on the title and subtitle varchars where changed to fit the longer names.
Title: size(35) → size(45)
subtitle: size(35) →size(30)

Date found: 01/01/2023
Description: The number of overdue books for a user would be double of what it is supposed to be on the overdue users screen.
Date fixed: 01/01/2023
Solution: The subject of what was counted in SQ10 was changed to become DISTINCT B.ISBN. This would make it so that only each distinct ISBN would be counted and there would be no duplicates. Thus, removing the doubling.

Date found 02/01/2024
Description: Changing a book's status on any of the librarian management screens would only occur once and the proceed to delete all the other books' frames but not change their status
Date fixed: 03/01/2024
Solution: A variable was given to the lambda function of each button command assigning the frame that it's currently in to that specific lambda function. This means that only the button that is clicked is affected by its own command.

Date found: 12/01/2024
Description: Some users were unable to sign in as their username which are definitely in the database could not be found by software during login
Date fixed: 14/01/2024
Solution: The query didn't order the results correctly. The collation of MySQL was case-insensitive, uppercase and lowercase letter were treated the same. To fix this a BINARY keyword was added to the sort, to explicitly say to have case-sensitive collation.

Date found: 13/01/2024
Description: Users with usernames longer than 10 characters were unable to reserve books
Date fixed: 13/01/2024
Solution: The reservations table had the wrong varchar size. The reservation table had a varchar size of 10, so users with usernames longer than 10 characters couldn't be used in it, causing the error.
Username: size(10) → size(20)

Date found: 14/01/24
Description: When trying to change the status of the book in the "To prepare" book screen. The incorrect book would disappear and you would be unable to remove any more books.
Date fixed: 14/01/24
Solution: The lambda function for the change status button didn't have a variable assigned to it for its current frame. This means that when current_frame.destroy() was executed the latest current_frame made by the program was destroyed and the button command would no longer attempt to destroy any more frames.

Testing

Functional requirements

Borrower:

- Inputs:
  - Account details at login and sign up.
  - Various buttons. (i.e. return button and/or submit button)
- Processes:
  - Validate user login/sign up
  - Get available books (with or without genre filter)
  - Get book details
  - Add book to cart
  - Get cart details
  - Calculate reservation details
  - Add reservation and newly made user to database
  - Update book statuses
- Outputs:
  - Various popups. (i.e. “Incorrect password” and “Book already in cart”)
  - All available book buttons
  - A specific book’s details
  - The user’s cart
  - Details of the book reservation

Librarian:

Same as borrower. As well as:

- Inputs:
  - Book management buttons (i.e. “Change to available”, “View due reservations” etc.)
  - Browse books button
- Processes:
  - Get all reservations of the specified category
  - Get all book in that reservation
  - Update database with status changes
- Outputs:
  - Displays of all reservations in a specified category
  - Displays of all book in that reservation
  - Popups for status change of books
  - Displays for when there are no reservations

End user requirements

Borrowers:

- Should have a user-friendly login and sign-up process.
- Should be able to easily browse books and apply genre filters for a refined search.
- Can view details of chosen book
- Should be able to view their cart and remove any books they don't want anymore
- Can add books to their cart
- Should receive information on wait time until pickup, overdue fees and the due date of the books

Librarians:

- Should have the ability to browse or manage the books
- Should be able to view and prepare reservations, including details of specific books in each reservation.
- Should be able to check all books that have been taken out of the library and their due dates.
- Should have access to a list of users with overdue books and the respective number of overdue books they have.
- Should be able to change book statuses in the system.

Input validation

Login:

- Username must exist
- Password must be correct
- Submit button is disabled until values in all text boxes

Sign up:

- Username can't already exist
- Username must be 10-20 characters
- Password must be at least 12 characters
- Password and confirmation must match
- Submit button is disabled until values in all text boxes

Book details:

- A book cannot be added to a full cart
- The same book cannot be in the same cart multiple times

Cart:

- Checkout button is disabled until at least 1 item in the user's cart, then becomes disabled again if they remove all their books

Test cases

Extras file

Test ID	Objective	Description	Expected result
E1	Check that binary_search() finds the target value in the array (Normal) (Correct index)	Use program code to execute binary_search() and print result • array = [[1,2],[3,4],[5,6]] • target = 3 • index = 0	The value 1 will be printed to screen
E2	Check that binary_search() finds the target value in the array (Normal) (Incorrect index)	Use program code to execute binary_search() and print result • array = [[1,2],[3,4],[5,6]] • target = 3 • index = 1	The value -1 will be printed to the screen
E3	Check that binary_search() finds the target value in the array (Extreme) (Correct index)	Use program code to execute binary_search() and print result • array = [[1,2],[3,4],[5,6]] • target = 1 • index = 0	The value 0 will be printed to screen
E4	Check that binary_search() finds the target value in the array (Extreme) (Incorrect index)	Use program code to execute binary_search() and print result • array = [[1,2],[3,4],[5,6]] • target = 1 • index = 1	The value -1 will be printed to screen
E5	Check that binary_search() doesn't find the target value (Exceptional)	Use program code to execute binary_search() and print result • array = [[1,2],[3,4],[5,6]] • target = 9 • index = 0	The value -1 will be printed to screen
E6	Check that binary_search() doesn't find the target value (Exceptional)	Use program code to execute binary_search() and print result • array = [[1,2],[3,4],[5,6]] • target = 9 • index = 1	The value -1 will be printed to screen
E7	Check that the hash_password() correctly hashes the input	Use program code to execute hash_password() and print the returned result • password = "Hello World!"	A 64 character long string will be printed to screen which will be the sha256 hash of the password

E8	Check that no_results() displays onto a screen	Use program code to create a window • centre = Tk()	The text "Sorry there are currently no books available" will be displayed on the tkinter window
E9	Check that no_results_res() displays onto a screen	Use program code to create a window • centre = Tk()	The text "There are currently no reservations" will be displayed on
E10	Check that error_screen displays all its contents, destroys old window and closes connection (if exists)	Use program code to execute error_screen() • root = Tk() • connection = [database connection variable] • app_details = (root, connection)	The error screen will display with all its text on it
E11	Check that error_screen displays all its contents, destroys old window and doesn't close connection	Use program code to execute error_screen() • root = Tk() • connection = None • app_details = (root, connection)	The error screen will display with all its text on it
E12	Check that count_cart() counts the number of non-blank items in the inputted array (Empty array)	Use program code to execute count_cart() and print the returned result • cart = ["","","",""]	The value 0 will be printed to screen
E13	Check that count_cart() counts the number of non-blank items in the inputted array (Basic array)	Use program code to execute count_cart() and print the returned result • cart = ["", "Lorem", "Ipsum"]	The value of 2 will be printed to the screen
E14	Check that count_cart() counts the number of non-blank items in the inputted array (Longer array)	Use program code to execute count_cart() and print the returned result • cart = ["Lorem", "Ipsum", "Dolor", "Sit", "amet"]	The value of 5 will printed to the screen
E15	Check that title_combine() combines two strings together (Both exist)	Use program code to execute title_combine() and print returned result • title = "Lorem" • subtitle = "Ipsum"	"Lorem - Ipsum" will be printed to the screen

E16	Check that title_combine() combines two strings together (Only title exists)	Use program code to execute title_combine() and print returned result • title = "Lorem" • subtitle = None	"Lorem" will be printed to the screen
E17	Check that wrap_text() creates new lines within the given width	Use program code to execute wrap_text() and print the result • text = "Lorem ipsum dolor sit amet. Lorem ipsum dolor sit" • width = 20	The text will be displayed with a new line after every 20 characters. With words not broken up over lines
E18	Check that colour_all_widgets() changes the background of all widgets in a window	Use program code to create a window and call colour_all_widgets() • centre = Tk() • Window will have 3 labels with "Lorem ipsum dolor sit" text • colour = "Red"	The window will have a red background for every "Lorem ipsum…" label, with the basic background still remaining white
E19	Check that delete_all_widgets() destroys all widgets on a window	Use program code to create a window • centre = Tk() • Window will have a button and the command will call delete_all_widgets() • There will also be a "lorem ipsum" label	When the button is pressed the label and the button will both be destroyed and the window will be blank
E20	Check that font_all_widgets() changed the font of all widgets on a window	Use program to create a window • centre = Tk() • Window will have a "Lorem ipsum" label and a button • Button will call font_all_widgets() when pressed • font_set = ("Wingdings","40")	When the button is pressed the label and button will have their font changed to wingdings and the font size changed to 40
E21	Check that convert_date() changes the inputted date format	Use program to execute convert_date() and print retuned result • current = "2006-12-16"	"16-12-2006" will be printed

E22	Check that destroy_frames() removes all frames on a window	Use program to create a window • root = Tk() • Create 3 frames coloured red, green and blue. Named their respective colours. • There will be a button and when pressed will call destroy_frames() • window_frames = [red, green, blue]	When the button is pressed the 3 different coloured sections(the three frames) will all be destroyed leaving only the button left

Queries file

Test ID	Objective	Description	Expected result
Q1	Check that db_connect() connects to database	Use program code to execute db_connect() and print result	MySQL connection variable. Should not be None
Q2	Check that Queries connects to Extras	Use program code to call error_screen() from extras and run it.	The error_screen window should be created
Q3	Check that query_select() executes given query (Working query)	Use program code to execute query_select() and print the returned result • root = Tk() • connection = [MySQL connection variable] • app_details = (connection, root) • query = """SELECT title FROM books""" • values = (None,)	All 30 book titles will be printed on the screen
Q4	Check that query_select() executes given query (Non-working query)	Use program code to execute query_select() • root = Tk() • connection = [MySQL connection variable] • app_details = (connection, root) • query = """SELECT title FROM users""" • values = ()	The error screen window (from extras) will popup

Q5	Check that query_insert() executes given query (Working query)	Use program code to execute query_insert() • root = Tk() • connection = [MySQL connection variable] • app_details = (connection, root) • query = IQ1 • values = (“janeDoe1279”, “Jane”, “Doe”, “de164ffdf2e3473d18860f1c8dc0aafcd1df53f9640fa8f0c83631f879427218”, 0)	The new user will be in the MySQL database
Q6	Check that query_insert() executes given query (Non-working query)	Use program code to execute query_ insert() • root = Tk() • connection = [MySQL connection variable] • app_details = (connection, root) • query = “””INSERT INTO users VALUES (%s, %s, %s)””” • values = (“janeDoe1279”,”Jane”,”Doe”)	The error screen window (from extras) will popup
Q7	Check that query_update() executes given query (Working query)	Use program code to execute query_update() • root = Tk() • connection = [MySQL connection variable] • app_details = (connection, root) • query = “””UPDATE Users SET surname = ‘Smith’ WHERE username = %s””” • values = (“janeDoe1279”,)	The user’s username will now be changed in the MySQL database
Q8	Check that query_update() executes given query (Non-working query)	Use program code to execute query_ update() • root = Tk() • connection = [MySQL connection variable] • app_details = (connection, root) • query = “””UPDATE Users SET librarian = ‘Pineapple’””” • values = ()	The error screen window (from extras) will popup

SQL Queries

Test ID	Objective	Description	Expected result
SQ1	Test that SQ1 retrieves expected result	Query will be run in MySQL workbench A reservation will be made by maryCampbell53 using INSERT queries for the for “Klanspor” with the status “Taken out” and due date set to 2024-01-12	The ISBN and due date of the book “Klanspor” should be returned. “9781907054648” and “2024-01-12”
SQ2	Test that SQ2 retrieves expected result	Query will be run in MySQL workbench	All 6 current users will have their details returned
SQ3	Test that SQ3 retrieves expected result	Query will be run in MySQL workbench	All 6 current users will have their usernames returned
SQ4A	Test that SQ4A retrieves expected result	Query will be run in MySQL workbench	All 29 available books will have their basic details returned. “Klanspor” is “Taken out” from SQ1 test
SQ4B	Test that SQAB retrieves expected result	Query will be run in MySQL workbench but the %s will be swapped out for “Fantasy”	All 12 available fantasy books will have their basic details returned
SQ5	Test that SQ5 retrieves expected result	Query will be run in MySQL workbench but the %s will be swapped out with “9780261103344”	The full details of ‘The Clobbit’ book will be returned
SQ6	Test that SQ6 retrieves expected result	Query will be run in MySQL workbench but the %s will be swapped out with “9780261103344”	The title and subtitle of the book with ISBN “9780261103344” will be returned
SQ7	Test that SQ7 retrieves expected result	Query will be run in MySQL workbench. A reservation for the books “1002” and “June” will be created using INSERT statements under the user marycampbell53. dateOfOrder will be 2024-02-14 and timeOfOrder will be 12:54:14. The several %s in SQ7 will be replaced with the appropriate values	The reservationID of 3 should be returned

SQ8.1	Test that SQ8 retrieves expected result of all "Reserved" books	Query will be run in MySQL workbench The %s in the subquery will be set to "Reserved". A reservation will be made using INSERT queries for the book "Juco" and "AT" and their statuses changed to "Reserved". User will be maryCampbell53. The date and time will be the same as the SQ7 test	The basic details of reservations from SQ7 test and this newly made one
SQ8.2	Test that SQ8 retrieves expected result of all "Taken out" books	Query will be run in MySQL workbench The %s in the subquery will be set to "Taken out". A reservation will be made using INSERT queries for the book "June messier" and "The shimmering" and their statuses changed to taken out. User will be maryCampbell53. The date and time will be the same as the SQ7 test	The basic details of reservations. Should include the one from SQ1 test and this newly made one
SQ9A.1	Test that SQ9A retrieves expected result. Status= "Reserved"	Query will be run in MySQL workbench The reservationID will be set to the ID created by SQ8.1 and status changed to "Reserved"	The basic book details of "Juco" and "AT" should be returned
SQ9A.2	Test that SQ9A retrieves expected result. Status = "taken out"	Query will be run in MySQL workbench The reservation ID will be set to the ID created by SQ8.2 and status changed to "Taken out"	The basic book details of "June messier" and "The shimmering" should be returned
SQ9B	Test that SQ9B retrieves expected result	Query will be run in MySQL workbench The reservationID will be set to the same as in SQ8.1 test.	The dateOfOrder and timeOfOrder should be 2024-02-14 and 12:54:14 respectively.
SQ10	Test that SQ10 retrieves expected result	Query will be run in MySQL workbench A reservation will be created using INSERT queries for "Battle donkey" and "Devonian land" and their statuses set to "Overdue". The date and time of order will be the same SQ7 test. User will be marycampbell53	The username maryCampbell53 and the value 2 should be returned for number of books
SQ11	Test that SQ11 retrieves expected result	Query will be run in MySQL workbench The username will be set to maryCampbell53	The basic details of "Battle donkey" and "Devonian land" should be returned with their due date (2024-02-14)

DML Queries

Test ID	Objective	Description	Expected result
UQ1	Check that UQ1 changes a book to be "overdue"	Query will be run in MySQL workbench The ISBN for the book "1002" will be used in place of the %s. ISBN = "9780099066101"	Book status will be changed in the table
UQ2	Check that UQ2 changes a book to be "Reserved"	Query will be run in MySQL workbench The ISBN for the book "1002" will be used in place of the %s. ISBN = "9780099066101"	Book status will be changed in the table
UQ3	Check that UQ3 changes a book to be "Taken out"	Query will be run in MySQL workbench The ISBN for the book "1002" will be used in place of the %s. ISBN = "9780099066101"	Book status will be changed in the table
UQ4	Check that UQ4 changes a book to be "Available"	Query will be run in MySQL workbench The ISBN for the book "1002" will be used in place of the %s. ISBN = "9780099066101"	Book status will be changed in the table
IQ1	Has already been tested by Q5		
IQ2	Check that IQ2 creates a new reservation	Query will be run in MySQL workbench A reservation under the user maryCampbell53 will be made with dateOfOrder = "2023-12-31", timeOfOrder = "13:54:12" and dueDate = "2024-01-01"	A new reservation with ID of 7 will be created
IQ3	Check that IQ3 creates a new book reservation	Query will be run in MySQL workbench A reservation using ID from IQ2 test will be made for "1002"	A new book reservation for "1002" will be made with reservation ID 7

(All changes created by UQ & IQ testing will be reverted after the test)

Main file

Test ID	Objective	Description	Expected result
M1	Check that Library_Queries file is connected and connection to database is successful	Use program code to create a connection variable and print it Call LQ.db_connect. Print connection variable	A MySQL connection variable will be printed
M2	Check that the error catch for unsuccessful database connection happens	Use program code to execute a modified main function. The connection variable will be set to None	The error_screen() from Library_Extras should appear
M3	Check that update_status changes the statuses of books past due date.	Create 2 reservations in mySQl workbench: 1. For “Frunkenstain” with due date of “2024-02-10” 2. For “Devonian land 2” with due date of “2025-02-10” Both by maryCampbell53 and statuses changed to “Taken out” Use program to call update_status() through main() and print SQ1 result. Then in conditional statement print details of book that meets criteria. Finally recall and print SQ1	Both book reservation details will be printed “Frunkenstain” details printed [From conditional statement] Details of Devonian land 2 then printed [From SQ1 recall]
M4	Check that login and signup buttons take to the corresponding screens	Use program code to execute main(). On the start_menu() click both buttons separately.	Login button should take user to login screen and Signup button should take user to sign up screen.
M5	Check that submit button remain disabled until there is a value in all boxes [Login]	Use program code to execute main(). Then go through window to the login. Attempt to click submit. If nothing works, then type “a” and “a” into both boxes.	Submit button should become enabled and once clicked will give the “Username doesn’t exist” popup

M6	Check that return button takes user back to start_menu() [Login]	Use program code to execute main(). Then go through window to the login. Click on return button	Start_menu() screen should reappear
M7	Check that username validation works [Login]	Use program code to execute main(). Then go through window to the login. Enter “ab” and “bc” into text boxes	A popup saying that the “Username does not exist” should popup and not allow passage into catalogue screen
M8	Check that password validation works [Login]	Use program code to execute main(). Then go through window to the login. Enter “marycampbell53” into username and “lorem” into password	A popup saying “Password is incorrect” should appear and deny passage to catalogue
M9	Check that the submit button works. For librarians [Login]	Use program code to execute main(). Then go through window to the login. Login using: Username: “maryCampbell53” Password: “ILoveBooks!!!”	A popup saying “Welcome maryCampbell53 to the librarian system” should appear and they should be on the librarian_menu() screen now
M10	Check that submit button remain disabled until there is a value in all boxes [Sign up]	Use program code to execute main(). Then go through window to the signup. Attempt to click submit, if it doesn’t work. Enter “a” into every box	The submit button should now be enabled and if clicked display a “Username must be 10-20 characters” popup
M11	Check that username validation works. Existing username	Use program code to execute main(). Then go through window to the signup. Attempt to create an account with username “maryCampbell53” and “a” in all other boxes. Then press submit button	A “Username already exists” popup should appear and stop the creation of the new account.

M12	Check that username validation works. (Not long enough username)	Use program code to execute main(). Then go through window to the signup. Attempt to create an account with the username “DOLM2006” and “a” in all other boxes	A “Username must be between 10-20 characters” popup will appear and stop creation of new account
M13	Check that username validation works. (Too long username)	Use program code to execute main(). Then go through window to the signup. Attempt to create an account with the username “DOLM2006LOVESBOOKSTOREAD” and “a” in all other boxes	A “Username must be between 10-20 characters” popup will appear and stop creation of a new account.
M14	Check that password validation works. Password too short	Use program code to execute main(). Then go through window to the signup. Attempt to create an account with: Username: “DOLMIO2006” Password: “books” All else: “a”	A “Password must be at least 12 characters” popup will appear and stop creation of new account.
M15	Check that password validation works. With password confirmation	Use program code to execute main(). Then go through window to the signup. Attempt to create an account with: Username: “DOLMIO2006” Password: “weReadBooksHere” Password confirmation: “mints” All else: “a”	A “Passwords must match” popup will appear and stop account creation

M16	Check that hitting submit creates new user, creates popup and taken to login_screen()	Use program code to execute main(). Then go through window to the signup. Create an account using: Username: “DOLMIO2006” Password: “weReadBooksHere” First name: “Daniel” Surname: “Monaghan” Then click submit button	A popup saying “Welcome new user. Please login” will appear. After clicking ok they will then be brought to the login_screen() and need to then login. There should also be the new user in the MySQL database. It will be the only user with a 0 in librarian (indicating it is a borrower).
M17	Check that hitting submit as a borrower on login takes the user to the catalogue	Use program code to execute main(). Then go through window to the login. Login using the details from test M16.	A popup saying “Welcome DOLMIO2006 to the Book & Borrow system” will appear and the user will be brought to the catalogue screen.
M18	Check that all available books are displayed	Use a fresh library database with all of the book statuses set to “Available”. Use program code to execute main(). Then go through window to the catalogue. Then see how many buttons are generated in the catalogue when there is no filter used.	There should be 30 unique book buttons in the catalogue
M19	Check that dropdown box has all options	Use program code to execute main(). Then go through window to the catalogue. Click on the dropdown box.	The options: “None”, “Non-fiction”, “Science fiction”, “Fantasy”, “Horror” and “Romance” Should all appear (not specifically in that order)
M20	Check that the page title changes with every dropdown box selection	Use program code to execute main(). Then go through window to the catalogue. Click on the dropdown box and select all different options from the box.	The “All available books:” title should change to “All available for: [genre]” for all except the “None” option. The “None” option shouldn’t alter the title from “All available books:”

M21	Check that the available books buttons changes corresponding to the genre filter selected.	Use program code to execute main(). Then go through window to the catalogue. Do this on the same fresh database as M18 Select an option from a dropdown box and see how many book buttons appear.	There should be this many buttons for each option: None – 30 books Fantasy – 12 books Science fiction – 5 books Non-fiction – 5 books Horror – 4 books Romance – 4 books
M22	Check that cart button takes user to cart_screen() [catalogue]	Use program code to execute main(). Then go through window to the catalogue. Click on the cart button on the catalogue screen	The cart screen should be displayed on the screen
M23	Check that scrollbar correctly works	Use program code to execute main(). Then go through window to the catalogue. Do this on the same fresh database as M18 Keep on no genre filter and see if it scrolls down to cover all 30 books. Then change to horror.	On no filter: All 30 books should be displayed, and the scrollbar will be able to be used to view all of them On Horror filter: The scrollbar should not appear as there are only 4 books and so a scroll is not needed
M24	Check that book button takes user to appropriate book_details	Use program code to execute main(). Then go through window to the catalogue. Do this on the same fresh database as M18 Click on the button for “The Clobbit”	The book details screen of “The Clobbit” should appear
M25	Check that cart button takes user to cart_screen() [book_details]	Use program code to execute main(). Then go through window to the “The Clobbit” details. Click on the cart button	The cart screen should now be displayed instead of the book details screen.

M26	Check that hitting add_to_cart button adds current book ISBN to cart. Empty cart. With popup	Use program code to execute main(). Then go through window to the “The Clobbit” details. Have the cart array set to [“”,””,””] Print cart array. Click on the add to cart button Print the cart array at end of add_book()	A popup saying “Book has been added” should appear. A blank array of strings should be printed. Then an array with “The Clobbit” ‘s ISBN as the first item should be printed
M27	Check that hitting add_to_cart button doesn’t add the already added book to cart. With popup	Use program code to execute main(). Then go through window to the “The Clobbit” details. After test M28 click the cart will have “The clobbit” in it. Print cart array Click on add to cart button again Print cart array	A “Book is already in cart” popup will appear. An array with the ISBN of “The Clobbit” will be printed. Then the same array will be printed again since no two of the same book can be in cart.
M28	Check that hitting add_to_cart button doesn’t add the book to a full cart. With popup	Use program code to execute main(). Then go through window to the “The Clobbit” details. Set cart = [“1”,”2”,”3”] Print cart array Click on add to cart button Print cart array	A “Cart is full” popup will appear. [“1”,”2”,”3”] will be printed. This will then be printed again as the cart is full. So no changes made to cart
M29	Check that hitting return button takes user back to catalogue [book_details]	Use program code to execute main(). Then go through window to the “The Clobbit” details. Click on the return button	The catalogue screen should be on the window instead of the “the Clobbit” details screen.
M30	Check that hitting return button takes user back to catalogue [cart]	Use program code to execute main(). Then go through window to the cart screen, from catalogue. Click on the return button.	The catalogue screen should now be on the window instead of the cart screen.

M31	Check that checkout button is disabled if cart is empty	Use program code to execute main(). Then go through window to the cart. Have cart = [“”,””,””,””]. Then go onto the cart screen.	Displayed on the window at the cart items should be: 1. No book selected 2. No book selected 3. No book selected With no “X” buttons beside them
M32	Check that book removal button works	Use program code to execute main(). Then go through window to the cart. Have the first item of cart set to the ISBN of “The Clobbit” and the second item set to the ISBN of “1002”. Then click the “X” button beside “The Clobbit”	Before button press: 1. The Clobbit |X| 2. 1002 – A medieval Odyssey |X| 3. No book selected After button press: 1. No book selected 2. 1002 – A medieval Odyssey |X| 3. No book selected |X| = removal button
M33	Check that checkout button becomes enabled when there is at least 1 book in cart	Use program code to execute main(). Then go through window to the cart. Have cart = [“9780261103344”,””,””] Then press the |X| for “The Clobbit”	On the window all there should be “The Clobbit” and 2 “No book Selected”. The checkout button should then be enabled. After “The Clobbit” |X| button is pressed then the Checkout button should become disabled again.
M34	Check that checkout screen appears with correct calculations	Use program code to execute main(). Then go through window to the cart. Have 2 books in the cart. Then click the checkout button. The two books will be: “1002” and “The Clobbit”	The checkout screen should then appear with the following calculations: Time until ready: 12 minutes Overdue fees: £0.4 per day Due date: 17-03-2024
M35	Check that exit button closes window	After the M34 test. Click on the Exit button	The window will close itself. [If this was a real system the Borrower would then make their way to the library]

M36	Check that new reservation is created by checkout screen	After the M34 test. Use MySQL workbench to view the database and see the newly made reservation. [In the database from M18]	There will be a single reservation with ID of 1. Then in bookReservation there will be 2 records for the 2 reserved books. In the books table the 2 books will have a "Reserved" status
Start of librarian section testing			
M37	Check that "Browse" and "Manage" "Books" buttons takes user to catalogue and action_menu() respectively	Use program code to execute main(). Then go through window to the login, and use a librarian login. Username: marycampbell53 Password: ILoveBooks!!! Click on both of the buttons on the screen.	The Browse books button will display the catalogue screen on the window. The Manage books button will display the action menu screen. Both buttons will display a popup telling the user what they selected before moving screen
M38	Check that return button takes user back to librarian_menu() [action_menu]	Use program code to execute main(). Then go through window to action menu(). Librarian section. Click on the return button	The librarian menu will be displayed instead of the action menu
M39	Check that each reservation button takes user to the respective reservations screen	Use program code to execute main(). Then go through window to action menu(). Librarian section. Click on each of the button options. To prepare Taken out Overdue [M18 database]	The relevant reservations screen will be displayed. The To prepare reservations screen will contain the reservation made in M34. The other two screens will have their respective "No reservations" text.
M40	Check that return button takes user back to action_menu from each reservation screen.	Use program code to execute main(). Then go through window to each reservation screen. Librarian section. Go onto each reservation type screen and click the return button on each.	All 3 of the reservation screens should bring the user back to the action menu screen

M41	Check that to_prepare reservation displays correctly. There are reservations to prepare	Use program code to execute main(). Then go through window to “To prepare” reservations screen. Librarian section. Use the M18 database	The reservation from M34 will be there
M42	Check that to_prepare reservation displays correctly. There are no reservations to prepare	Use program code to execute main(). Then go through window to “To prepare” reservations screen. Librarian section. Comment out the normal query result and replace it with an empty tuple. result = ()	The text “There are currently no reservations” should be displayed on screen
M43	Check that specific reservation button works. Takes user to prepare_books screen	Use program code to execute main(). Then go through window to “To prepare” reservations screen. Librarian section. Click on the reservation made by M34.	This will display the 2 books in M34’s reservation with their date and time of order. There will also be a change status button beside both books. The details of “The Clobbit” and “1002” will be there
M44	Check that return button takes user to to_prepare screen [prepare_books]	Use program code to execute main(). Then go through window to “To prepare” books screen. Librarian section. Click on the return button	The “To prepare” reservations screen will be displayed
M45	Check that status change button works and database is updated [prepare_books]	Use program code to execute main(). Then go through window to “To prepare” books screen. Librarian section. Click the “Change to taken out” button on both books. Then the return button is pressed. [M18 database]	This will remove those books from the screen. There will also be a popup saying: “Book status changed”. Then when checking the MySQL database, the status of these two books should be “Taken out”. After the return button is pressed the “To prepare” reservations screen should be displayed and “There are currently no reservations” should be on the screen

M46	Check that due_reserves displays correctly. There are reservations to prepare	Use program code to execute main(). Then go through window to “Due” reservations screen. Librarian section. [M18 database]	There will be the reservation details of M34 there. Because due to M45 there is now a “Taken out” reservation. So, this should be the only reservation there
M47	Check that due_reserves displays correctly. There are no reservations to prepare	Use program code to execute main(). Then go through window to “Due” reservations screen. Librarian section. Comment out the normal query result and replace with an empty tuple. result = ()	The text “There are currently no reservations” should be displayed onto the window
M48	Check that specific reservation button works. Takes user to due_books screen	Use program code to execute main(). Then go through window to “Due” reservations screen. Librarian section. Press the button with the reservation details from M34 [M18 database]	This will display the Due books screen using the details from the M34 reservation. This will be “The Clobbit” and “1002”
M49	Check that return button takes user to due_reserves screen [due_books]	Use program code to execute main(). Then go through window to “Due” books screen. Librarian section. Press the return button	The Due reservations screen will now be displayed onto the window
M50	Check that status change button works and database is updated [due_books]	Use program code to execute main(). Then go through window to “Due” books screen. Librarian section. Click the “Change to available” buttons for both books. Then the return button is pressed.	Clicking the change status button will remove each of the books from the screen and make a “Book status changed” popup appear. Then by checking the MySQL database again all the books should now be “Available” After the return button is pressed the “Due” reservations screen will be displayed and “There are currently no reservations” will be on the screen

M51	Check that overdue_reservation displays correctly. There are users with overdue books	Use MySQL workbench to create a reservation under "DOLMIO2006" for "The Clobbit" and "1002" and set them to be overdue. Use program code to execute main(). Then go through window to overdue reservations screen. Librarian section. [M18 database]	This will display the basic user details of "DOLMIO2006" and the number 2 for how many overdue books they have.
M52	Check that overdue_reservation displays correctly. There are no reservations to prepare	Use program code to execute main(). Then go through window to "overdue" reservations screen. Librarian section Comment out the normal query result and replace it with an empty tuple. result = ()	This will display "There are no users with overdue books" to the screen
M53	Check that specific reservation button works. Takes user to overdue_books screen	Use program code to execute main(). Then go through window to "overdue" reservations screen. Librarian section. The button with the "DOLMIO2006" user details is clicked [M18 database]	This will display the "overdue" books screen with all the "overdue" books of that specific user. This should be "The Clobbit" and "1002"
M54	Check that return button takes user to overdue_reservations screen [overdue_books]	Use program code to execute main(). Then go through window to "overdue" books screen. Librarian section. The return button is then clicked.	This will display the "overdue" reservations screen instead of the "overdue" books screen now

M55	Check that status change button works and database is updated [overdue_books]	Use program code to execute main(). Then go through window to “overdue” books screen. Librarian section. The 2 book buttons for that user are then clicked. Then the return button is clicked	The book button click will remove them from the screen and display a “Book status changed” popup. Then when checking the MySQL database all the book will again all be “Available”. After the return button is pressed the “There are no users with overdue books” text will appear on the overdue reservations screen

Testing results

Extras file

Test id	Screenshot	Comment

E1	Code:
	`array = [[1,2],[3,4],[5,6]]`
	`target = 3`
	`position = binary_search(array, target, 0)`
	`print(position)`
	
	Output:
	`1`
		Expected result was displayed.
		
		[The value 1 indicates the target value is in the 2nd row of the 2D array]

E2	Code:
	`array = [[1,2],[3,4],[5,6]]`
	`target = 3`
	`position = binary_search(array, target, 1)`
	`print(position)`
	
	Output:
	`-1`
		Expected result was displayed

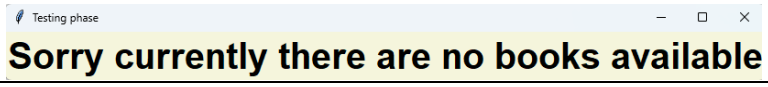
E3	Code:
	`array = [[1,2],[3,4],[5,6]]`
	`target = 1`
	`position = binary_search(array, target, 0)`
	`print(position)`
	
	Output:
	`0`
		Expected result was displayed

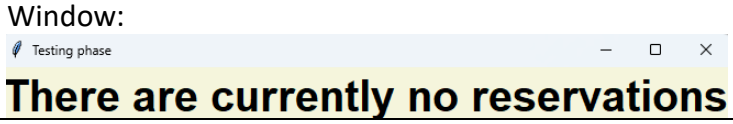
E4	Code:
	`array = [[1,2],[3,4],[5,6]]`
	`target = 1`
	`position = binary_search(array, target, 1)`
	`print(position)`
	
	Output:
	`-1`
		Expected result was displayed

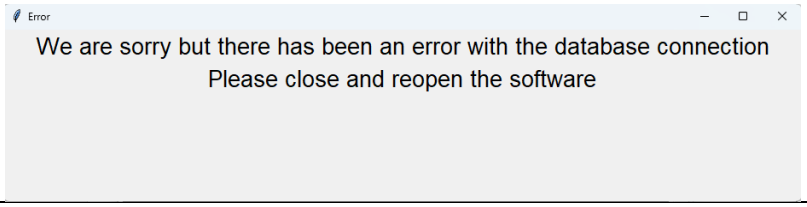
E5	Code:
	`array = [[1,2],[3,4],[5,6]]`
	`target = 9`
	`position = binary_search(array, target, 0)`
	`print(position)`
	
	Output:
	`-1`
		Expected result was displayed

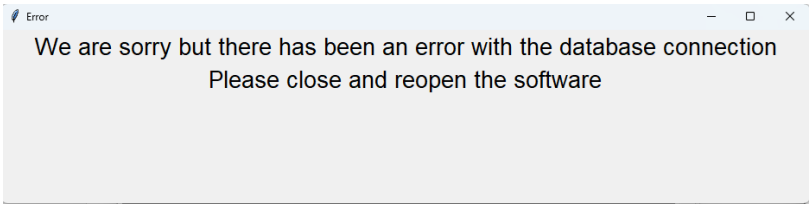
E6	Code
	`array = [[1,2],[3,4],[5,6]]`
	`target = 9`
	`position = binary_search(array, target, 1)`
	`print(position)`
	
	Output:
	`-1`
		Expected result was displayed

E7	Code:
	`password = "Hello  World!"`
	`password_hash = hash_password(password)`
	`print(password_hash)`
	
	Output:
	`5607578f200dd3acb22bf9ff8a1f315a84a4fefef96dcd215d225f2eb5a022ac`
		Expected result was displayed

E8	Code:
	`centre = Tk()`
	`centre.title("Testing phase")`
	`no_results(centre)`
	`centre.mainloop()`
	
	Window:
	 
		Expected result was displayed

E9	Code:
	`centre = Tk()`
	`centre.title("Testing phase")`
	`no_results_res(centre)`
	`centre.mainloop()`
	
	Window:
	 
		Expected result was displayed

E10	Code:
	`centre = Tk()`
	`connection = db_connect()`
	`print(connection)`
	`appDetails = (connection, centre)`
	`error_screen(appDetails)`
	Window:
	 
		Expected result was displayed
		
		db_connect had to be fully copied into Extras instead of imported. If imported a circular import error would occur with the Queries file.

E11	Code:
	`centre = Tk()`
	`connection = None`
	`print(connection)`
	`appDetails = (connection, centre)`
	`error_screen(appDetails)`
	Window:
	 
		Expected result was displayed

E12	Code:
	`cart = ["","",""]`
	`items = count_cart(cart)`
	`print(items)`
	Output:
	`0`
		Expected result was displayed

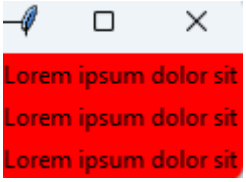
E13	Code:
	`cart = ["","Lorem","Ipsum"]`
	`items = count_cart(cart)`
	`print(items)`
	Output:
	`2`
		Expected result was displayed

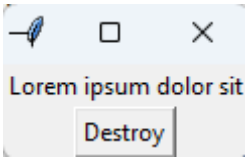
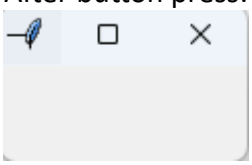
E14	Code:
	`cart = ["Lorem","Ipsum", "Dolor", "Sit", "Amet"]`
	`items = count_cart(cart)`
	`print(items)`
	Output:
	`5`
		Expected result was displayed

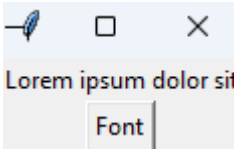
E15	Code:
	`title = "Lorem"`
	`subTitle = "Ipsum"`
	`combined = title_combine(title, subTitle)`
	`print(combined)`
	Output:
	`Lorem-Ipsum`
		Expected result was displayed

E16	Code:
	`title = "Lorem"`
	`subTitle = None`
	`combined = title_combine(title, subTitle)`
	`print(combined)`
	Output:
	`Lorem`
		Expected result was displayed

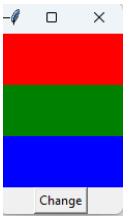
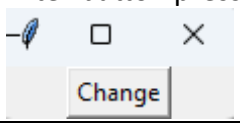
E17	Code
	`text = "Lorem ipsum dolor sit amet. Lorem ipsum dolor sit"`
	`width = 20`
	`wrapped_text = wrap_text(text, width)`
	`print(wrapped_text)`
	Output:
	`Lorem ipsum dolor`
	`sit amet. Lorem`
	`ipsum dolor sit`
		Expected result was displayed

E18	Code:
	`centre = Tk()`
	`centre.title("Testing phase")`
	`Label(centre, text = "Lorem ipsum dolor sit").pack()`
	`Label(centre, text = "Lorem ipsum dolor sit").pack()`
	`Label(centre, text = "Lorem ipsum dolor sit").pack()`
	
	`colour = "Red"`
	`colour_all_widgets(centre, colour)`
	`centre.mainloop()`
	Window
	 
	
		Expected result was displayed

E19	Code:
	`centre = Tk()`
	`centre.title("Testing phase")`
	`Label(centre, text = "Lorem ipsum dolor sit").pack()`
	
	`destroy = Button(centre, text="Destroy", command = lambda: delete_all_widgets(centre))`
	`destroy.pack()`
	`centre.mainloop()`
	Window:
	 
	After button press:
	 
		Expected result was displayed

E20	Code:
	`centre = Tk()`
	`centre.title("Testing phase")`
	`Label(centre, text = "Lorem ipsum dolor sit").pack()`
	
	`font_set = ("Wingdings", 40)`
	
	`font = Button(centre, text="Font", command = lambda: font_all_widgets(centre, font_set))`
	`font.pack()`
	`centre.mainloop()`
	Window:
	 
	After button press:
	 
		Expected result was displayed

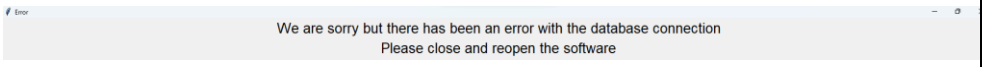
E21	Code:
	`current = "2006-12-16"`
	`converted = convert_date(current)`
	`print(converted)`
	
	Output:
	`16-12-2006`
		Expected result was displayed

E22	Code:
	`centre = Tk()`
	`centre.title("Testing phase")`
	
	`red = Frame(centre, bg="red")`
	`green = Frame(centre, bg ="green")`
	`blue = Frame(centre, bg="blue")`
	`window_frames = [red, green, blue]`
	
	`for frame in window_frames:`
	`  frame.config(height = 50)`
	`  frame.pack(fill=X)`
	
	`changer = Button(centre, text="Change", command = lambda: destroy_frames(window_frames))`
	`changer.pack()`
	
	`centre.mainloop()`
	Window
	 
	
	After button press:
	 
	
		Expected result was displayed

Queries file

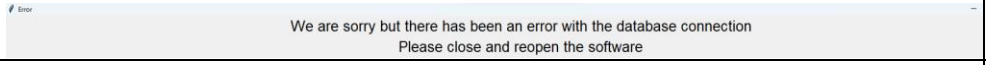
Test id	Screenshot	Comment

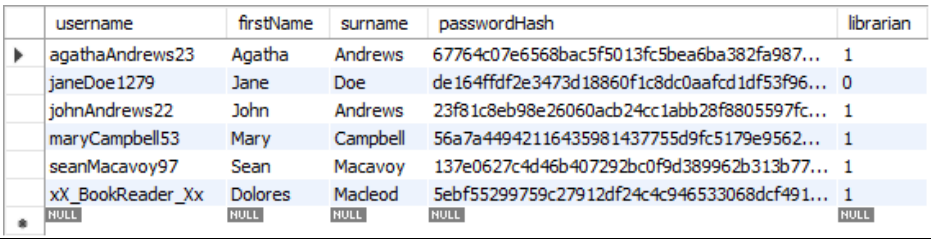
Q1	Code:
	`connection = db_connect()`
	`print(connection)`
	
	Output:
	`<mysql.connector.connection_cext.CMySQLConnection object at 0x000001B9FE197970>`
		Expected result was displayed

Q2	Code:
	`old = Tk()`
	`connection = db_connect()`
	`appDetails = (connection, old)`
	`LE.error_screen(appDetails)`
	
	Window:
	 
	
		Expected result was displayed
		
		[Tkinter module was imported]
		[Error screen shortened for size in report]

Q3	Code:
	`root = Tk()`
	`connection = db_connect()`
	`app_details = (connection, root)`
	`query = """SELECT title FROM books"""`
	`values = ()`
	
	`result = query_select(query, values, app_details)`
	`for item in range(len(result)):`
	`  print(result[item])`
		Expected result was displayed
		
		[Tkinter module was imported]

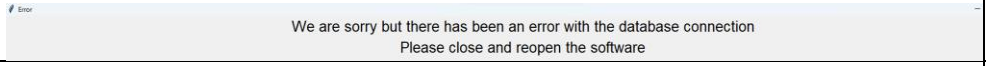
	Output: ("Scream's Transport Home",) ('The Similarion',) ('1002',) ('Fortune tombolo',) ('The metasmorphosisis',) ('The Clobbit',) ('The comradeship of the halo',) ('The eighty-nine spires',) ('The departure of the queen',) ('Juco',) ('June',) ("Larry Snotter and the Mathmetician's pebble",) ('Frunkenstain',) ('The Blue Furlong',) ('Twenty fourty-eight',) ('Larry Snotter and Auditorium of Truths',) ('Larry Snotter and the Free man of Azkabazkaba',) ('Larry Snotter and the Chalice holding water',) ('Parry Hotter and the Order of the Chicken',) ('Larry Snotter and the Cold blooded Princess',) ('Larry Snotter and the Lively Hallows',) ('All is Loud in the eastern Headquarters',) ('Battle Donkey',) ('The Shimmering',) ('June Messier',) ('Devonian Land',) ('Devonian Land 2',) ('Klanspor',) ('AT',) ('Annie of Blue Bables',)	

Q4	Code:
	`root = Tk()`
	`connection = db_connect()`
	`app_details = (connection, root)`
	`query = """SELECT title FROM users"""`
	`values = ()`
	
	`result = query_select(query, values, app_details)`
	`for item in range(len(result)):`
	`  print(result[item])`
	
	Window:
	 
		Expected result was displayed
		
		[Tkinter module was imported]
		[Error screen shortened for size in report]

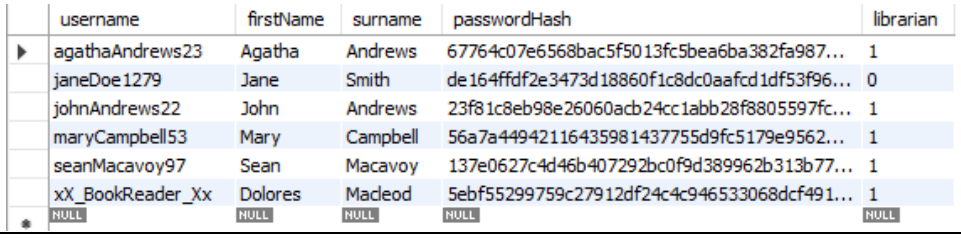
Q5	Code:
	`root = Tk()`
	`connection = db_connect()`
	`app_details = (connection, root)`
	`query = IQl`
	`values = ("janeDoe1279", "Jane", "Doe", "de164ffdf2e3473d18860f1c8dc0aafcd1df53f9640fa8f0c83631f879427218",`
	
	`query_insert(query, values, app_details)`
	
	Database users table:
	 
		Expected result was displayed
		
		[Tkinter module was imported]

	username	firstName	surname	passwordHash	librarian
▶	agathaAndrews23	Agatha	Andrews	67764c07e6568bac5f5013fc5bea6ba382fa987...	1
	janeDoe1279	Jane	Doe	de164ffdf2e3473d18860f1c8dc0aafcd1df53f96...	0
	johnAndrews22	John	Andrews	23f81c8eb98e26060acb24cc1abb28f8805597fc...	1
	maryCampbell53	Mary	Campbell	56a7a4494211643598143775 5d9fc5179e9562...	1
	seanMacavoy97	Sean	Macavoy	137e0627c4d46b407292bc0f9d389962b313b77...	1
	xX_BookReader_Xx	Dolores	Macleod	5ebf55299759c27912df24c4c946533068dcf491...	1
*	NULL	NULL	NULL	NULL	NULL

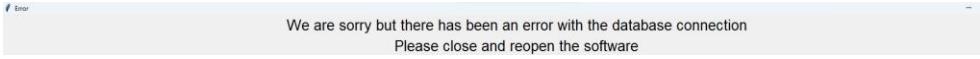
<table>
<tr>
<td>Q6</td>
<td>Code:
<pre>
root = Tk()
connection = db_connect()
app_details = (connection, root)
query = """INSERT INTO users VALUES (%s, %s, %s)"""
values = ("janeDoe1279", "Jane", "Doe")

query_insert(query, values, app_details)
</pre>
Window

</td>
<td>Expected result was displayed

[Tkinter module was imported]</td>
</tr>
<tr>
<td>Q7</td>
<td>Code:
<pre>
root = Tk()
connection = db_connect()
app_details = (connection, root)
query = """UPDATE Users SET surname = 'Smith' WHERE username = %s
values = ("janeDoe1279",)

query_update(query, values, app_details)
</pre>
Database users table:

</td>
<td>Expected result was displayed

[Tkinter module was imported]</td>
</tr>
<tr>
<td>Q8</td>
<td>Code:
<pre>
root = Tk()
connection = db_connect()
app_details = (connection, root)
query = """UPDATE Users SET librarian='Pineapple'"""
values = ()

query_update(query, values, app_details)
</pre>
Window:

</td>
<td>Expected result was displayed

[Tkinter module was imported]
[Error screen shortened for size in report]</td>
</tr>
</table>

SQL testing

Test id	Screenshot	Comment

SQ1	Setup code:
	`INSERT INTO reservations`
	`VALUES (default, "maryCampbell53", "2024-01-11", "13:34:12", "2024-01-12");`
	`INSERT INTO bookreservation`
	`VALUES (2, "9781907054648");`
	`UPDATE books`
	`SET bookStatus = "Taken out"`
	`WHERE ISBN = "9781907054648";`
	
	`SELECT B.ISBN, R.dueDate`
	`FROM books AS B, bookreservation AS bR, reservations AS R`
	`WHERE B.ISBN = bR.ISBN`
	`AND bR.reservationID = R.reservationID`
	`AND B.bookStatus = 'Taken out'`
	Query result:
	ISBN: 9781907054648, dueDate: 2024-01-12
		Expected result was displayed

SQ2	Setup code:
	`SELECT username, passwordHash, librarian`
	`FROM users`
	`ORDER BY BINARY username ASC;`
	Query result:
	username: agathaAndrews23, passwordHash: 67764c07e6568bac5f5013fc5bea6ba382fa987..., librarian: 1
	janeDoe1279, de164ffdf2e3473d18860f1c8dc0aafcd1df53f96..., 0
	johnAndrews22, 23f81c8eb98e26060acb24cc1abb28f8805597fc..., 1
	maryCampbell53, 56a7a4494211643598143775 5d9fc5179e9562..., 1
	seanMacavoy97, 137e0627c4d46b407292bc0f9d389962b313b77..., 1
	xX_BookReader_Xx, 5ebf55299759c27912df24c4c946533068dcf491..., 1
		Expected result was displayed

SQ3	Setup code:
	`SELECT username`
	`FROM users`
	`ORDER BY BINARY username ASC;`
	Query result
	username: agathaAndrews23
	janeDoe1279
	johnAndrews22
	maryCampbell53
	seanMacavoy97
	xX_BookReader_Xx
		Expected result was displayed

SQ4A

Setup code:

```
SELECT ISBN, title, subTitle, genre, firstName, surname
FROM books, authors
WHERE books.authorID = authors.authorID
AND bookStatus = "Available"
ORDER BY title ASC;
```

Query result:

ISBN	title	subTitle	genre	firstName	surname
9780099066101	1002	A medieval odyssey	Science fiction	Ari C.	Clerk
9780821223123	All is Loud in the eastern Headquarters	NULL	Romance	Erpoor	Marya
9788842516231	Annie of Blue Bables	NULL	Non-fiction	M.L	Gontmomery
9783453435773	AT	NULL	Horror	Stephanie	Queen
9781405226660	Battle Donkey	NULL	Non-fiction	Mitchel	Morichgo
9781784752224	Devonian Land	NULL	Non-fiction	Mikaela	Crichton
9781784752231	Devonian Land 2	The Found Plane	Science fiction	Mikaela	Crichton
9780192719980	Fortune tombolo	NULL	Horror	Robert Stewie	Alexson
9780520201798	Frunkenstain	or the ancient hercules	Romance	Marie	Shely
9780307348241	Juco	NULL	Romance	Stephanie	Queen
9780340960196	June	NULL	Science fiction	Fronk	Sherbert
9781473655324	June Messier	NULL	Science fiction	Fronk	Sherbert
9780747538486	Larry Snotter and Auditorium of Truths	NULL	Fantasy	J.K	Cowling
9780747546245	Larry Snotter and the Chalice holding ...	NULL	Fantasy	J.K	Cowling
9780747581086	Larry Snotter and the Cold blooded Pr...	NULL	Fantasy	J.K	Cowling
9780747542155	Larry Snotter and the Free man of Az...	NULL	Fantasy	J.K	Cowling
9780747591054	Larry Snotter and the Lively Hallows	NULL	Fantasy	J.K	Cowling
9780439362139	Larry Snotter and the Mathmetician's ...	NULL	Fantasy	J.K	Cowling
9780747569404	Parry Hotter and the Order of the Chi...	NULL	Fantasy	J.K	Cowling
9780007299263	Scream's Transport Home	NULL	Romance	Diana Winnie	Janes
9780575084346	The Blue Furlong	NULL	Horror	Stephanie	Queen
9780261103344	The Clobbit	NULL	Fantasy	J.R.R.R.R	Token
9780261103573	The comradeship of the halo	The lady of the halos p...	Fantasy	J.R.R.R.R	Token
9780261103597	The departure of the queen	The lady of the halos p...	Fantasy	J.R.R.R.R	Token
9780261103580	The eighty-nine spires	The lady of the halos p...	Fantasy	J.R.R.R.R	Token
9780199238552	The metasmorphosisis	and other stories	Non-fiction	Frenz	Kofko
9781444720723	The Shimmering	NULL	Horror	Stephanie	Queen
9780007523221	The Similarion	NULL	Fantasy	J.R.R.R.R	Token
9780582060180	Twenty fourty-eight	NULL	Science fiction	George	Andwell

Expected result was displayed

SQ4B

Setup code:

```
SELECT ISBN, title, subTitle, genre, firstName, surname
FROM books, authors
WHERE books.authorID = authors.authorID
AND bookStatus = "Available"
AND genre = "Fantasy"
ORDER BY title ASC;
```

Query result:

ISBN	title	subTitle	genre	firstName	surname
9780747538486	Larry Snotter and Auditorium of Truths	NULL	Fantasy	J.K	Cowling
9780747546245	Larry Snotter and the Chalice holding water	NULL	Fantasy	J.K	Cowling
9780747581086	Larry Snotter and the Cold blooded Princess	NULL	Fantasy	J.K	Cowling
9780747542155	Larry Snotter and the Free man of Azkabazkaba	NULL	Fantasy	J.K	Cowling
9780747591054	Larry Snotter and the Lively Hallows	NULL	Fantasy	J.K	Cowling
9780439362139	Larry Snotter and the Mathmetician's pebble	NULL	Fantasy	J.K	Cowling
9780747569404	Parry Hotter and the Order of the Chicken	NULL	Fantasy	J.K	Cowling
9780261103344	The Clobbit	NULL	Fantasy	J.R.R.R.R	Token
9780261103573	The comradeship of the halo	The lady of the halos part 1	Fantasy	J.R.R.R.R	Token
9780261103597	The departure of the queen	The lady of the halos part 3	Fantasy	J.R.R.R.R	Token
9780261103580	The eighty-nine spires	The lady of the halos part 2	Fantasy	J.R.R.R.R	Token
9780007523221	The Similarion	NULL	Fantasy	J.R.R.R.R	Token

Expected result was displayed

SQ5	Setup code
	`SELECT ISBN, title, subTitle, firstName, surname, genre, starRating, datePublished, blurb`
	`FROM books AS B, authors AS A`
	`WHERE B.authorID = A.authorID`
	`AND ISBN = "9780261103344";`
	Query result:
	ISBN | title | subTitle | firstName | surname | genre | starRating | datePublished | blurb
	9780261103344 | The Clobbit | NULL | J.R.R.R.R | Token | Fantasy | 4.70 | 1937-09-21 | Blibo Boggins is a clobbit who enjoys an uncomf...
		Expected result was displayed

SQ6	Setup code:
	`SELECT title, subtitle`
	`FROM books`
	`WHERE ISBN = "9780261103344";`
	Query result:
	title | subtitle
	The Clobbit | NULL
		Expected result was displayed

SQ7	Setup code:
	`INSERT INTO reservations`
	`VALUES (default, "maryCampbell53", "2024-02-14", "12:54:14", "2024-03-13");`
	`INSERT INTO bookreservation`
	`VALUES (3, "9780099066101");`
	`INSERT INTO bookreservation`
	`VALUES (3, "9780340960196");`
	`UPDATE books`
	`SET bookStatus = "Reserved"`
	`WHERE ISBN = "9780099066101" or ISBN = "9780340960196";`
	
	`SELECT reservationID`
	`FROM reservations`
	`WHERE username = "maryCampbell53"`
	`AND dateOfOrder = "2024-02-14"`
	`AND timeOfOrder = "12:54:14";`
	Query result:
	reservationID
	3
		Expected result was displayed

SQ8.1

Setup code:

```
INSERT INTO reservations
VALUES (default, "maryCampbell53", "2024-02-14", "12:54:14", "2024-03-13");
INSERT INTO bookreservation
VALUES (4, "9780307348241");
INSERT INTO bookreservation
VALUES (4, "9783453435773");
UPDATE books
SET bookStatus = "Reserved"
WHERE ISBN = "9780307348241" or ISBN = "9783453435773";

SELECT u.username, u.firstName, r.reservationID, COUNT(br.ISBN) AS NumberOfBooks
FROM Users u, Reservations r,bookReservation br
WHERE u.username = r.username
AND r.reservationID = br.reservationID
AND br.ISBN IN (SELECT ISBN
                FROM Books
                WHERE bookStatus = 'Reserved')
GROUP BY u.username, u.firstName, r.reservationID;
```

Query result:

username	firstName	reservationID	NumberOfBooks
maryCampbell53	Mary	1	1
maryCampbell53	Mary	3	1
maryCampbell53	Mary	4	2

Expected result was displayed

And a previous ongoing test reservation was also returned. The reservation ID of 1. Since the ongoing testing database was used at this point

SQ8.2

Setup code:

```
INSERT INTO reservations
VALUES (default, "maryCampbell53", "2024-02-14", "12:54:14", "2024-03-13");
INSERT INTO bookreservation
VALUES (5, "9781473655324");
INSERT INTO bookreservation
VALUES (5, "9781444720723");
UPDATE books
SET bookStatus = "Taken out"
WHERE ISBN = "9781473655324" or ISBN = "9781444720723";

SELECT u.username, u.firstName, r.reservationID, COUNT(br.ISBN) AS NumberOfBooks
FROM Users u, Reservations r,bookReservation br
WHERE u.username = r.username
AND r.reservationID = br.reservationID
AND br.ISBN IN (SELECT ISBN
                FROM Books
                WHERE bookStatus = 'Taken out')
GROUP BY u.username, u.firstName, r.reservationID;
```

Query result:

username	firstName	reservationID	NumberOfBooks
maryCampbell53	Mary	2	1
maryCampbell53	Mary	5	2

Expected result was displayed

<table>
<tr>
<td>SQ9A.1</td>
<td>Setup code:

```
SELECT B.ISBN, B.title, B.subTitle, B.genre
FROM books as B, reservations as R, bookreservation as BR
WHERE B.ISBN = BR.ISBN
AND BR.reservationID = R.reservationID
AND R.reservationID = 4
AND bookStatus = "Reserved";
```

Query result:

	ISBN	title	subTitle	genre
▶	9780307348241	Juco	NULL	Romance
	9783453435773	AT	NULL	Horror

</td>
<td>Expected result was displayed</td>
</tr>
<tr>
<td>SQ9A.2</td>
<td>Setup code:

```
SELECT B.ISBN, B.title, B.subTitle, B.genre
FROM books as B, reservations as R, bookreservation as BR
WHERE B.ISBN = BR.ISBN
AND BR.reservationID = R.reservationID
AND R.reservationID = 5
AND bookStatus = "Taken out";
```

Query result:

	ISBN	title	subTitle	genre
▶	9781444720723	The Shimmering	NULL	Horror
	9781473655324	June Messier	NULL	Science fiction

</td>
<td>Expected result was displayed</td>
</tr>
<tr>
<td>SQ9B</td>
<td>Setup code:

```
SELECT dateOfOrder, timeOfOrder
FROM reservations
WHERE reservationID = 4;
```

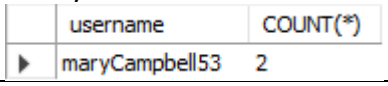
Query result:

	dateOfOrder	timeOfOrder
▶	2024-02-14	12:54:14

</td>
<td>Expected result was displayed</td>
</tr>
</table>

<table>
<tr>
<td>SQ10</td>
<td>Setup code:

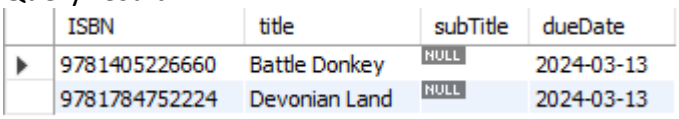
```
INSERT INTO reservations
VALUES (default, "maryCampbell53", "2024-02-14", "12:54:14", "2024-03-13");
INSERT INTO bookreservation
VALUES (6, "9781784752224");
INSERT INTO bookreservation
VALUES (6, "9781405226660");
UPDATE books
SET bookStatus = "Overdue"
WHERE ISBN = "9781784752224" or ISBN = "9781405226660";

SELECT U.username, COUNT(*)
FROM Users AS U, Reservations AS R, bookReservation as bR, Books as B
WHERE U.username = R.username
AND R.reservationID = bR.reservationID
AND bR.ISBN = B.ISBN
AND bookStatus = "Overdue"
GROUP BY U.username;
```

Query result:


	username	COUNT(*)
▶	maryCampbell53	2

</td>
<td>Expected result was displayed</td>
</tr>
<tr>
<td>SQ11</td>
<td>Setup code:

```
SELECT B.ISBN, title, subTitle, dueDate
FROM Books as B, bookReservation AS bR, Reservations as R, Users AS U
WHERE B.ISBN = bR.ISBN
AND bR.reservationID = R.reservationID
AND R.username = U.username
AND bookStatus = "Overdue"
AND R.username = "maryCampbell53";
```

Query result:


	ISBN	title	subTitle	dueDate
▶	9781405226660	Battle Donkey	NULL	2024-03-13
	9781784752224	Devonian Land	NULL	2024-03-13

</td>
<td>Expected result was displayed</td>
</tr>
</table>

DML testing

Test id	Screenshot	Comment

UQ1	Setup code:
	`UPDATE books`
	`SET bookStatus = "Overdue"`
	`WHERE ISBN = "9780099066101";`
	Query result:
	ISBN | authorID | title | subTitle | bookStatus
	9780007299263 | 10 | Scream's Transport Home | NULL | Available
	9780007523221 | 1 | The Similarion | NULL | Available
	9780099066101 | 11 | 1002 | A medieval odyssey | Overdue
		Expected result was displayed

UQ2	Setup code:
	`UPDATE books`
	`SET bookStatus = "Reserved"`
	`WHERE ISBN = "9780099066101";`
	Query result:
	ISBN | authorID | title | subTitle | bookStatus
	9780007299263 | 10 | Scream's Transport Home | NULL | Available
	9780007523221 | 1 | The Similarion | NULL | Available
	9780099066101 | 11 | 1002 | A medieval odyssey | Reserved
		Expected result was displayed

UQ3	Setup code:
	`UPDATE books`
	`SET bookStatus = "Taken out"`
	`WHERE ISBN = "9780099066101";`
	Query result:
	ISBN | authorID | title | subTitle | bookStatus
	9780007299263 | 10 | Scream's Transport Home | NULL | Available
	9780007523221 | 1 | The Similarion | NULL | Available
	9780099066101 | 11 | 1002 | A medieval odyssey | Taken out
		Expected result was displayed

UQ4	Setup code:
	`UPDATE books`
	`SET bookStatus = "Available"`
	`WHERE ISBN = "9780099066101";`
	Query result:
	ISBN | authorID | title | subTitle | bookStatus
	9780007299263 | 10 | Scream's Transport Home | NULL | Available
	9780007523221 | 1 | The Similarion | NULL | Available
	9780099066101 | 11 | 1002 | A medieval odyssey | Available
		Expected result was displayed

<table>
<tr>
<td>IQ2</td>
<td>Setup code:

```
INSERT INTO reservations(username, dateOfOrder, timeOfOrder, dueDate)
VALUES("maryCampbell53", "2023-12-31", "13:54:12", "2024-01-01");
```

Query result:

reservationID	username	dateOfOrder	timeOfOrder	dueDate
1	maryCampbell53	2024-02-12	10:47:14	2024-03-11
2	maryCampbell53	2024-01-11	13:34:12	2024-01-12
3	maryCampbell53	2024-02-14	12:54:14	2024-03-13
4	maryCampbell53	2024-02-14	12:54:14	2024-03-13
5	maryCampbell53	2024-02-14	12:54:14	2024-03-13
6	maryCampbell53	2024-02-14	12:54:14	2024-03-13
7	maryCampbell53	2023-12-31	13:54:12	2024-01-01

</td>
<td>Expected result was displayed</td>
</tr>
<tr>
<td>IQ3</td>
<td>Setup code:

```
INSERT INTO bookreservation(reservationID, ISBN)
VALUES(7, "9780099066101");
```

Query result:

reservationID	ISBN
1	9780099066101
3	9780099066101
7	9780099066101

</td>
<td>Expected result was displayed</td>
</tr>
</table>

Main file

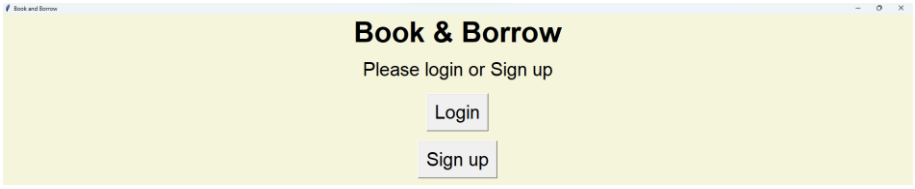
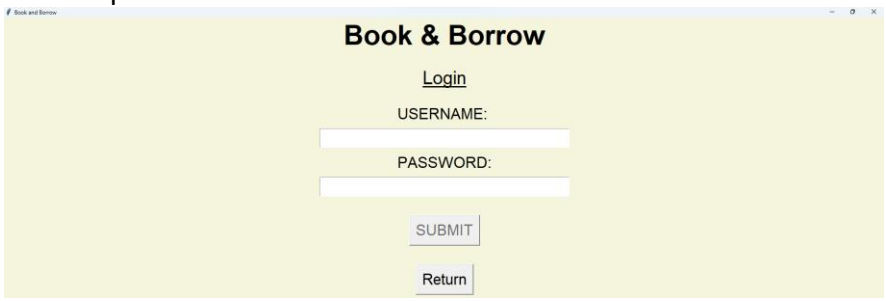
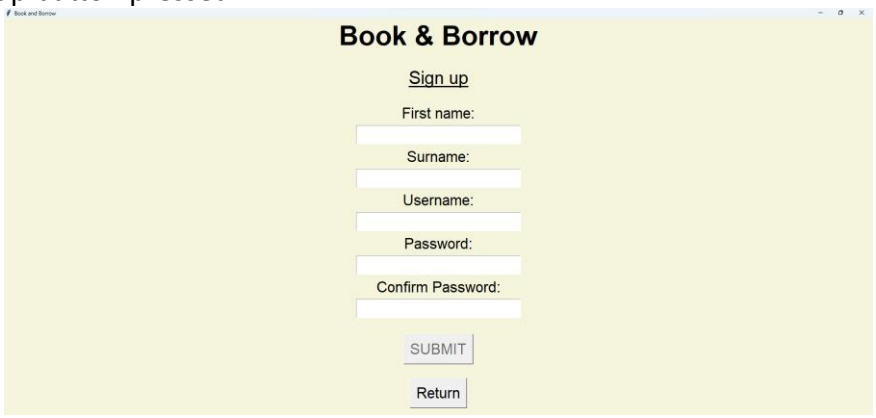
Window covers the whole screen but has been shortened down for some screenshots. To increase readability of report.

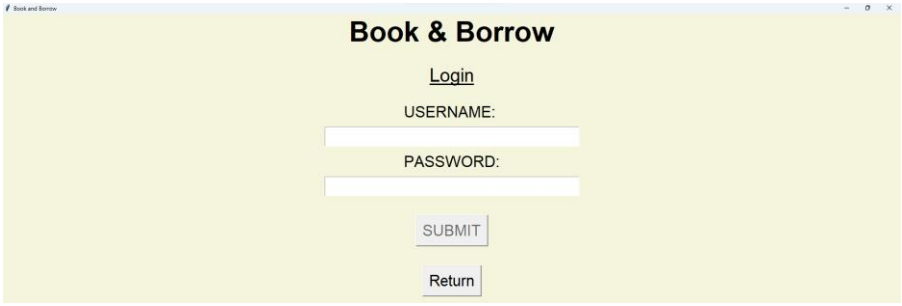
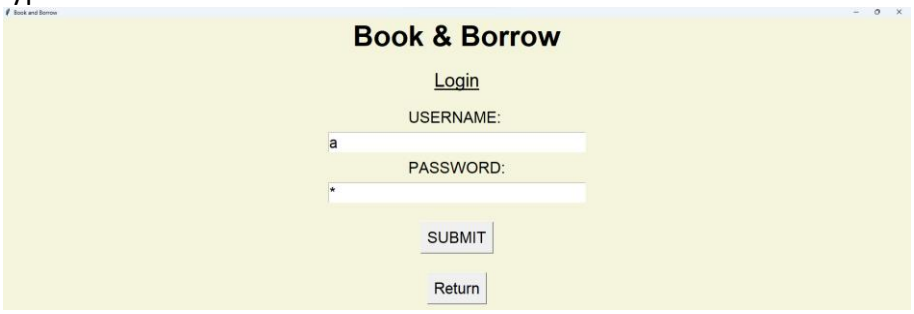
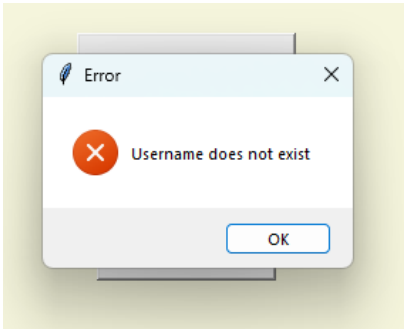
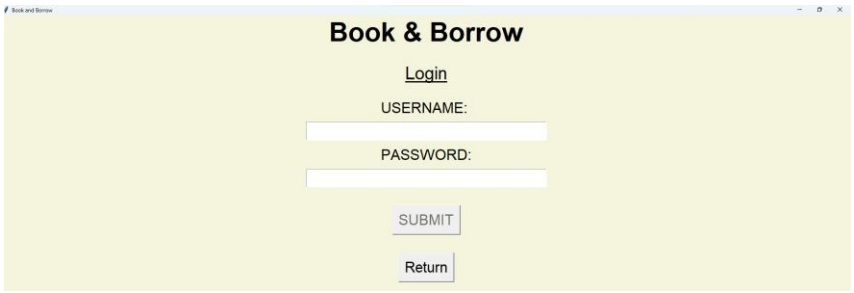
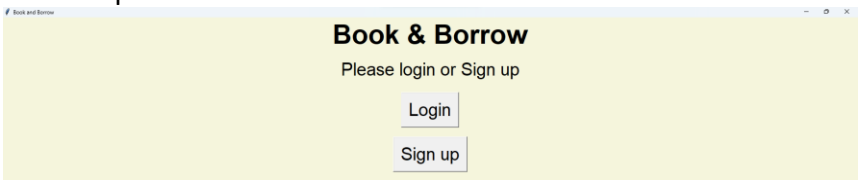
Test id	Screenshot	Comment

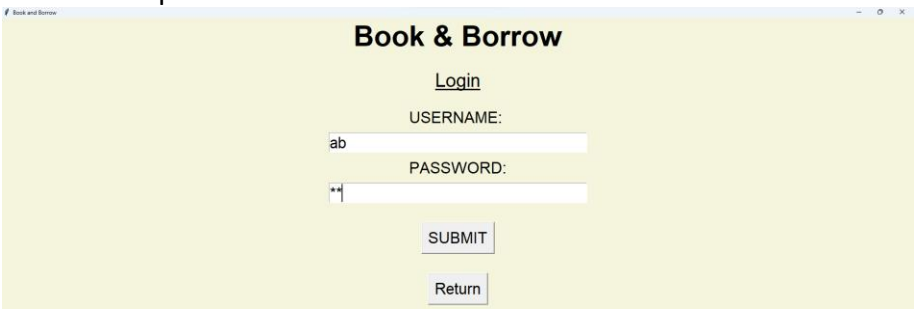
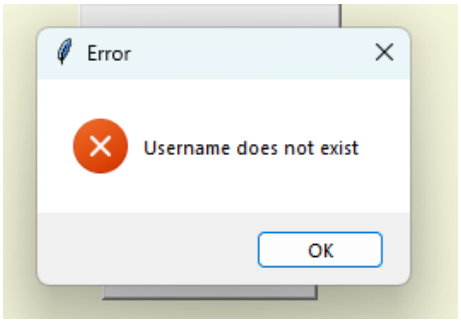
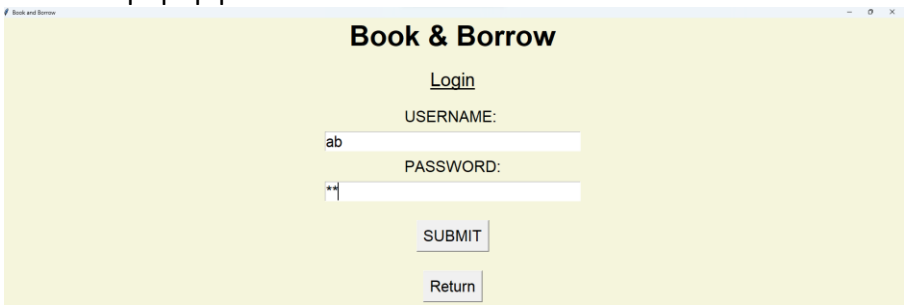
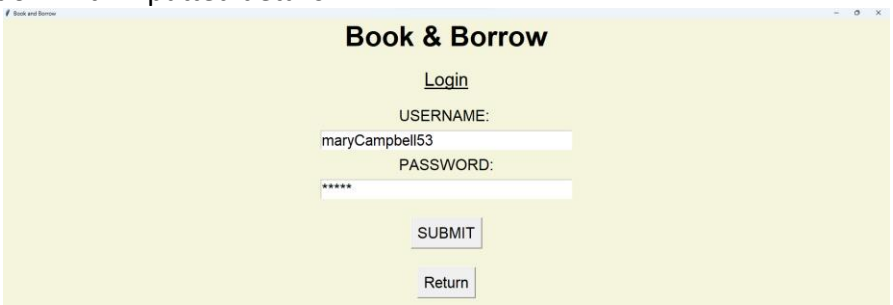
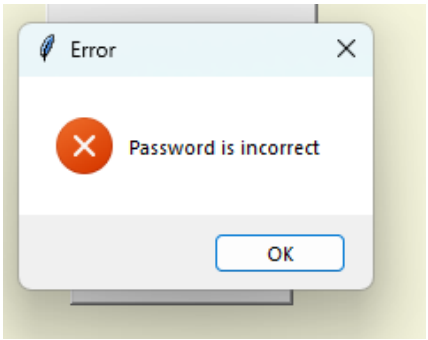
M1	Code:
	`def main():`
	`  global cart, app_details`
	`  # Create user's cart, made global for accessibility`
	`  cart = [""] * 3`
	`  # Create connection to database`
	`  connection = LQ.db_connect()`
	`  print(connection)`
	Output:
	`<mysql.connector.connection_cext.CMySQLConnection object at 0x0000021D89150070>`
		Expected result was displayed

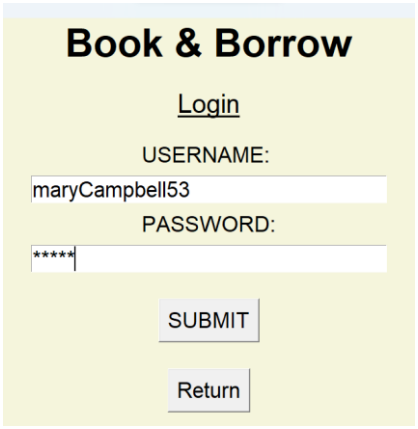
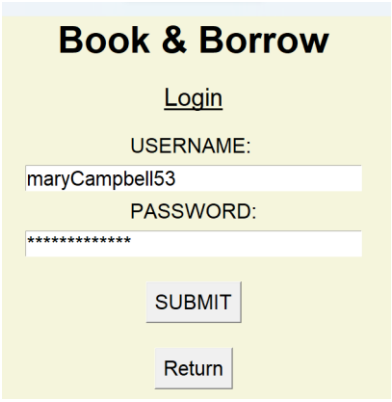
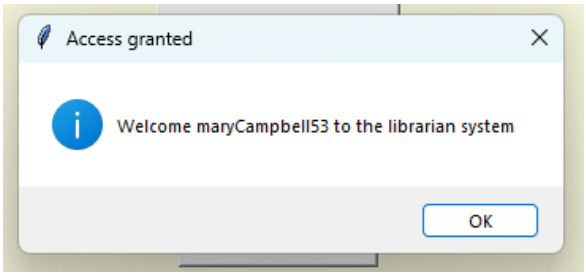
M2	Code
	`def main():`
	`  global cart, app_details`
	`  # Create user's cart, made global for accessibility`
	`  cart = [""] * 3`
	`  # Create connection to database`
	`  #connection = LQ.db_connect()`
	`  connection = None`
	`  # If connection is successful`
	`  if connection:`
	`  #Create the window`
	`  root = Tk()`
	`  root.title("Book and Borrow")`
	`  root.state("zoomed")`
	`  root.config(bg="beige")`
	`  # Create tuple with connection and window details, made global for accessibility`
	`  app_details = (connection, root)`
	`  # Update any overdue books`
	`  update_status()`
	`  # Display the login/signup start screen`
	`  start_menu()`
	`  root.mainloop()`
	`  # If connection fails`
	`  else:`
	`  # Temp fake window so error screen procedure can be used`
	`  old = Tk()`
	`  app_details = (connection, old)`
	`  LE.error_screen(app_details)`
	Window:
	We are sorry but there has been an error with the database connection
	Please close and reopen the software
		Expected result was displayed
		
		This also counted as a check to see if the Main file connects to the Extras file.
		
		[Error screen screenshot only includes text of screen. Shortened for report purposes]

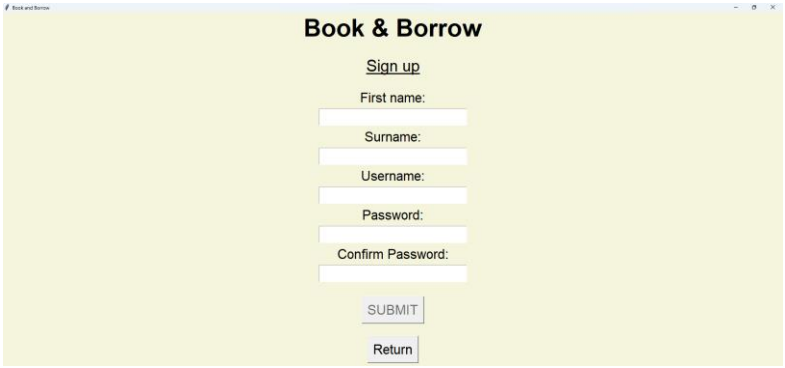
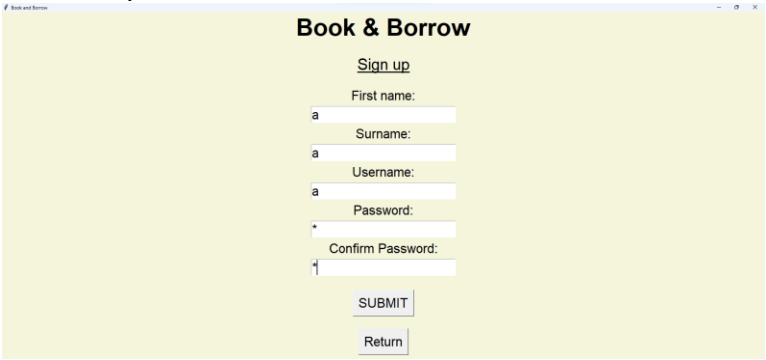
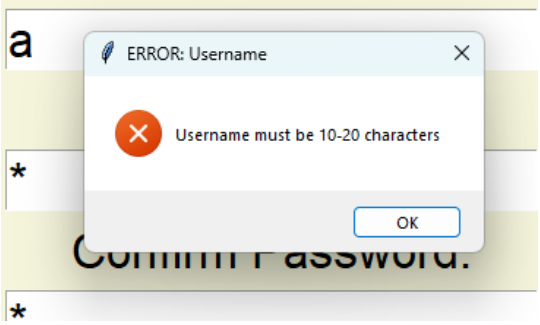
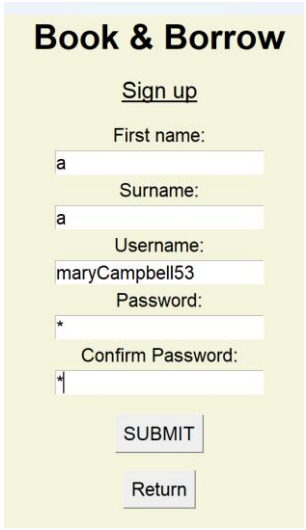
M3	Code:
	`def update_status():`
	`    # Get all book's ISBN and due date that are taken out`
	`    result = LQ.query_select(LQ.SQ1, None, app_details)`
	`    print(result)`
	`    print("\n")`
	
	`    # Get the current date`
	`    today = date.today()`
	
	`    # Loop for every book from the query`
	`    for item in result:`
	`        # If the due date has been passed then change book to be overdue`
	`        if today >= item[1]:`
	`            print(item)`
	`            print("\n")`
	`            LQ.query_update(LQ.UQ1, (item[0],), app_details)`
	
	`    result_redo = LQ.query_select(LQ.SQ1, None, app_details)`
	`    print(result_redo)`
	
	Output:
	`[('9780520201798', datetime.date(2024, 2, 10)), ('9781784752231', datetime.date(2025, 2, 10))]`
	
	`('9780520201798', datetime.date(2024, 2, 10))`
	
	`[('9781784752231', datetime.date(2025, 2, 10))]`
		Expected result was displayed
		
		[main() modified slightly. Start_menu() call was commented out]

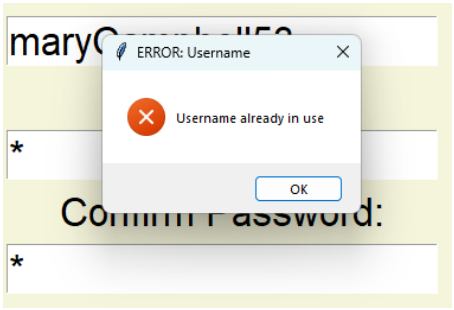
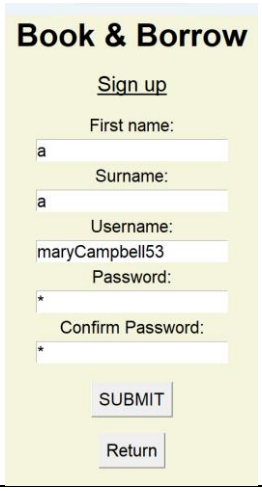
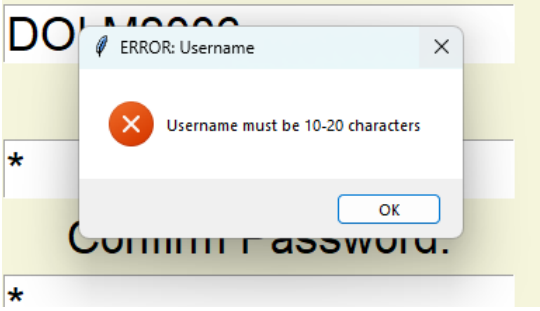
M4	Window:  Login button pressed:  Sign up button pressed:  	Expected result was displayed

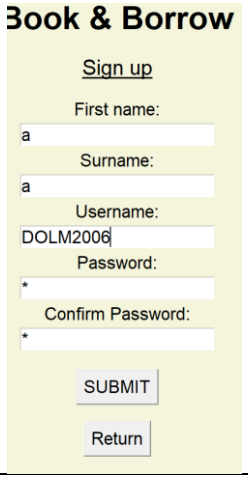
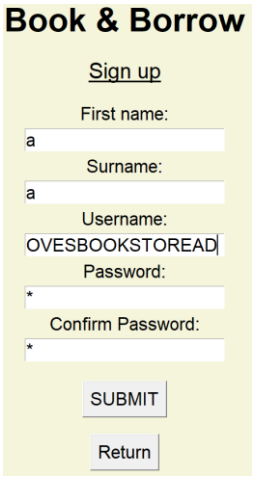
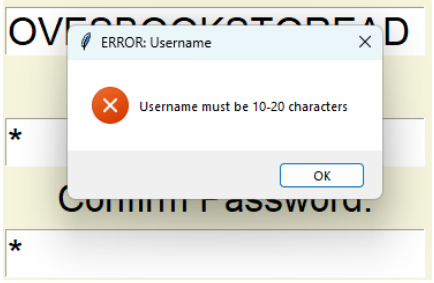
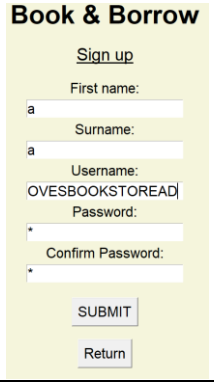
M5	Window:  "a" typed:  Button pressed: 	Expected result was displayed
M6	Window:  Return button pressed: 	Expected result was displayed

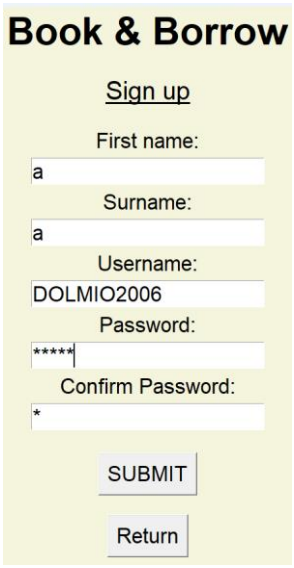
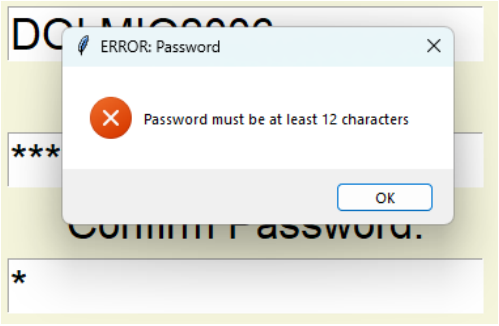
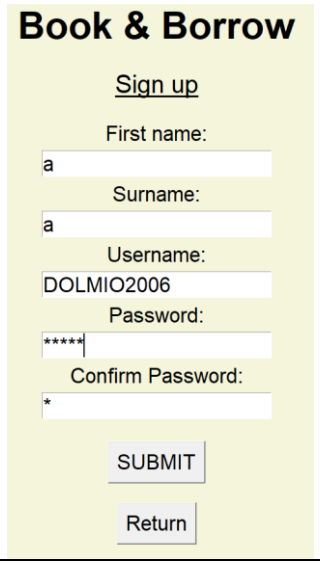
M7	Window with inputted details:  Submit button pressed:  Ok button on popup pressed:  	Expected result was displayed
M8	Window with inputted details:  Submit button pressed:  	Expected result was displayed

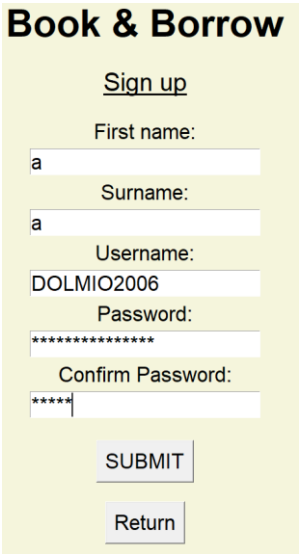
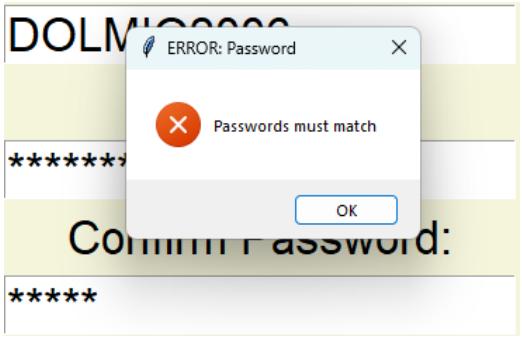
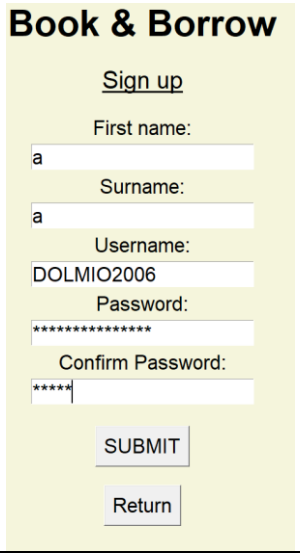
	Ok button on popup pressed: 	
M9	Window with details inputted:  Submit button pressed:  Ok button on popup pressed: 	Expected result was displayed

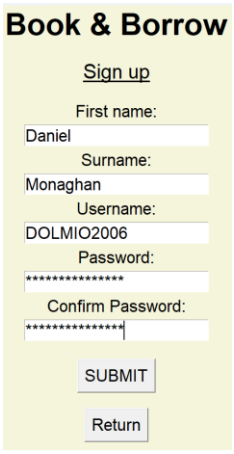
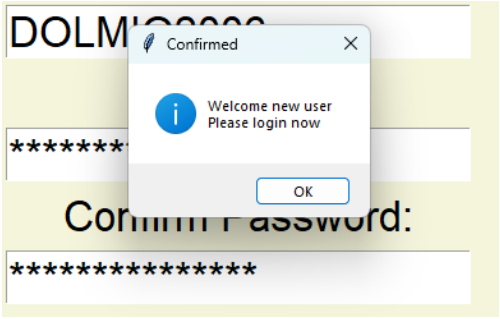
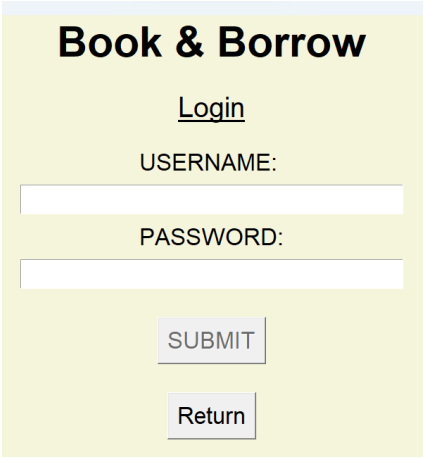
M10	Window  "a" typed into every box:  Submit button pressed:  	Expected result was displayed
M11	Window with detailss inputted:  	Expected result was displayed

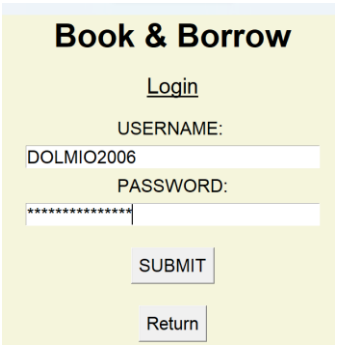
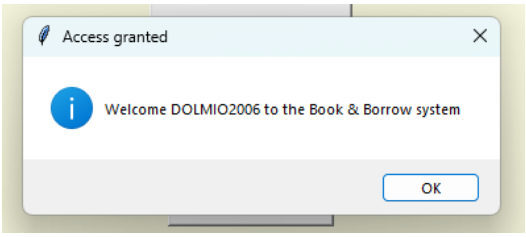
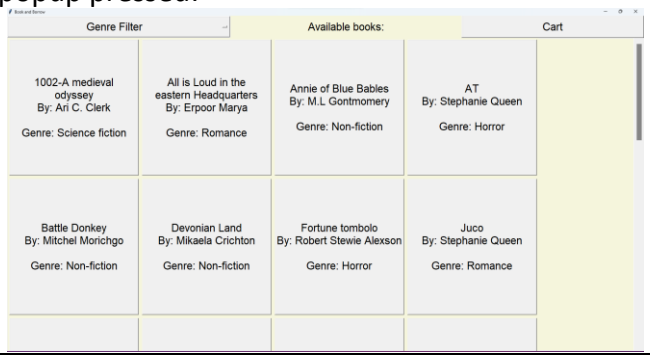
	Submit button pressed:  Ok button on popup pressed:  	
M12	Window with details inputted:  Submit button pressed:  	Expected result was displayed

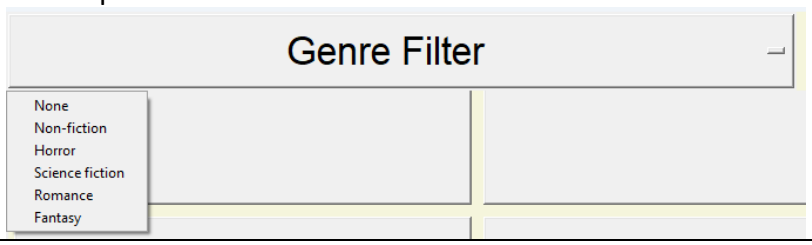
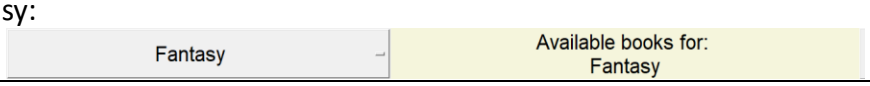
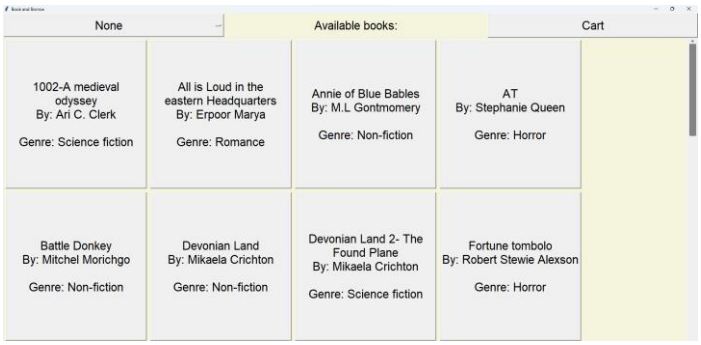
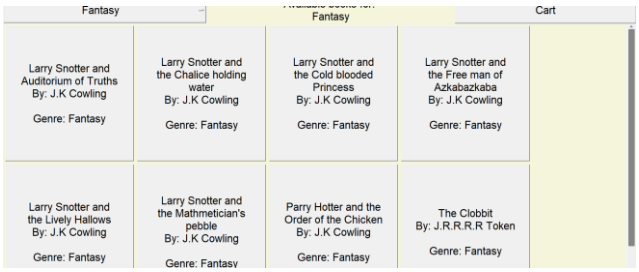
	Ok button on popup pressed: 	
M13	Window with details inputted:  Submit button pressed:  Ok button on popup pressed: 	Expected result was displayed

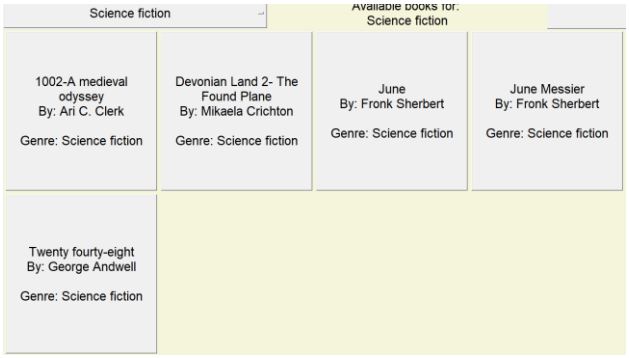
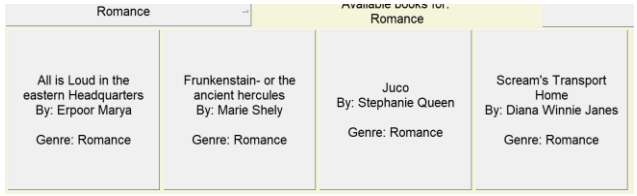
M14	Window with details inputted:  Submit button pressed:  Ok button on popup pressed:  	Expected result was displayed

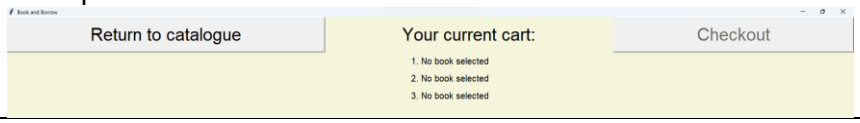
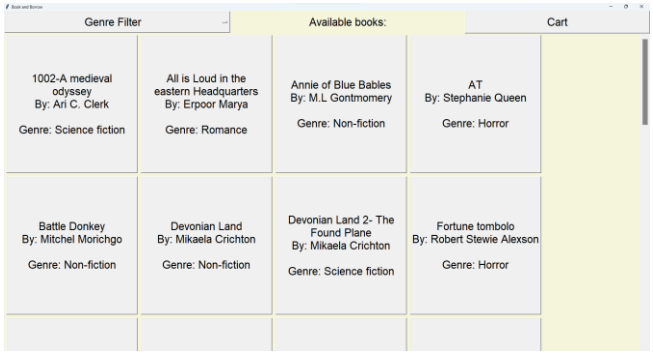
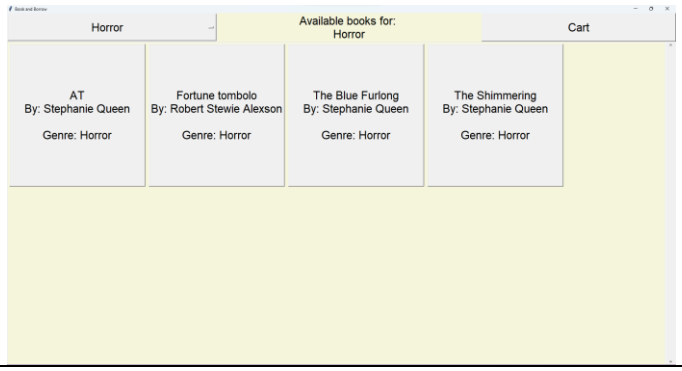
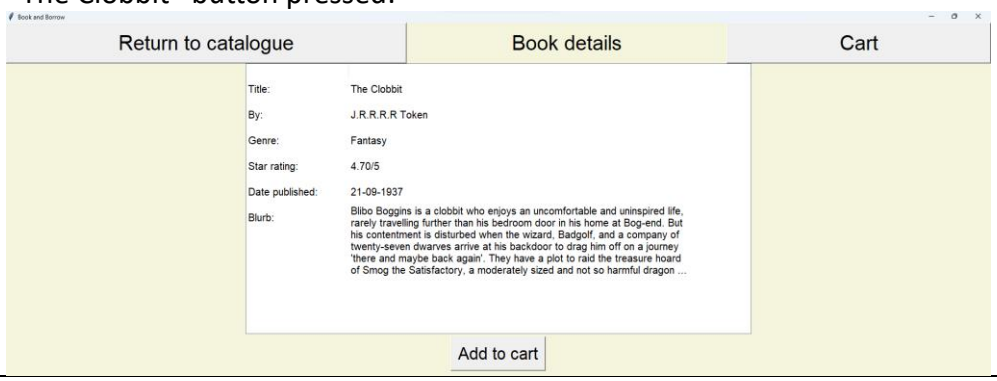
M15	Window with details inputted:  Submit button pressed:  Ok button on popup pressed:  	Expected result was displayed

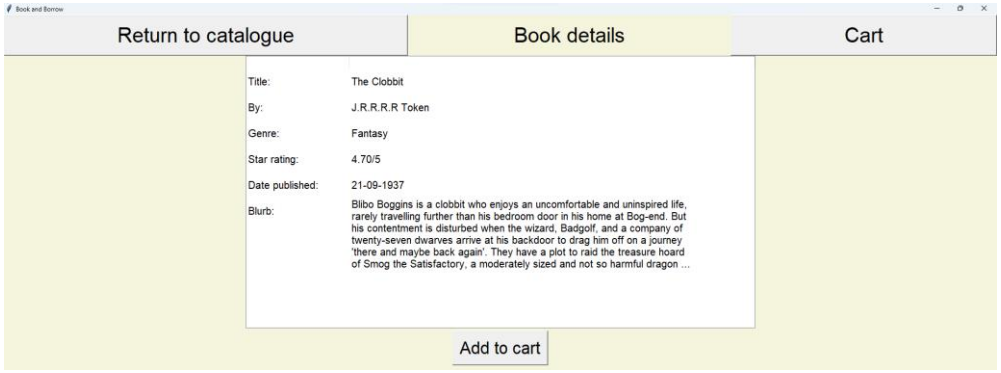
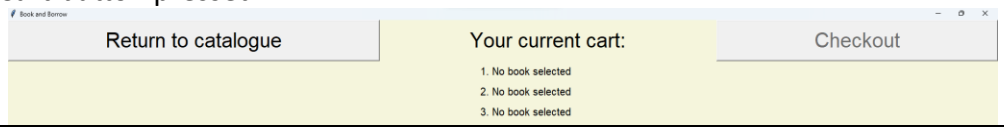
M16	Window with details inputted:  Submit button pressed:  Ok button on popup pressed:  Database (users table): username | firstName | surname | passwordHash | librarian agathaAndrews23 | Agatha | Andrews | 67764c07e6568bac5f5013fc5bea6ba382fa987... | 1 DOLMIO2006 | Daniel | Monaghan | 33803f2da56287729bdad5474e0deace3632e3... | 0 johnAndrews22 | John | Andrews | 23f81c8eb98e26060acb24cc1abb28f8805597fc... | 1 maryCampbell53 | Mary | Campbell | 56a7a4494211643598143775 5d9fc5179e9562... | 1 seanMacavoy97 | Sean | Macavoy | 137e0627c4d46b407292bc0f9d389962b313b77... | 1 xX_BookReader_Xx | Dolores | Macleod | 5ebf55299759c27912df24c4c946533068dcf491... | 1	Expected result was displayed

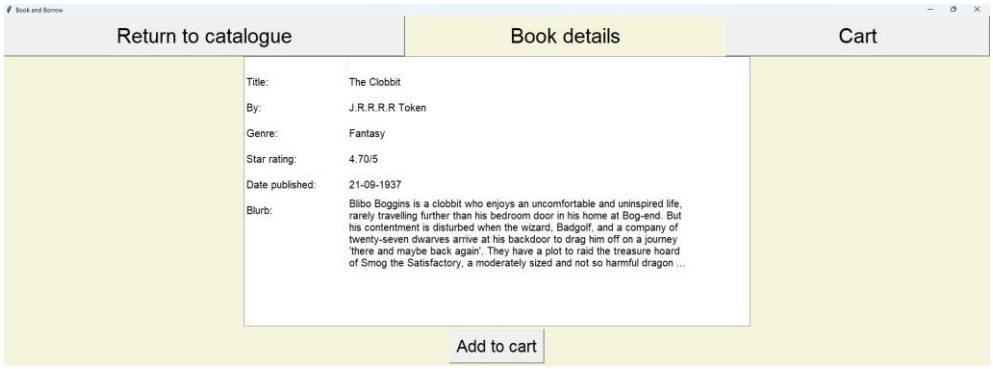
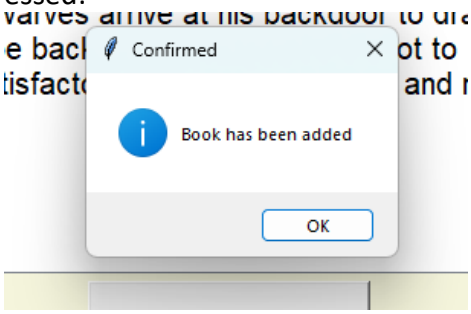
M17	Window with details inputted:  Submit button pressed:  Ok button on popup pressed: 	Expected result was displayed
M18	Window (scrolled down to bottom):  No. of buttons counted: 4 book per row. 7.5 rows. Total books = 30	Expected result was displayed

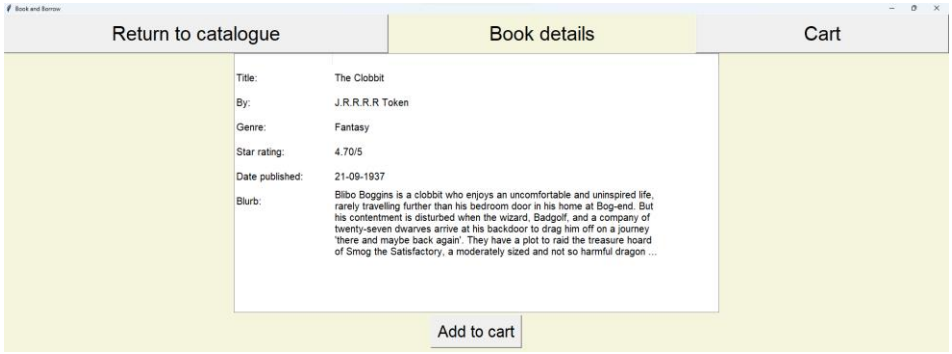
M19	Dropdown box pressed: 	Expected result was displayed
M20	No filter:  None:  Non-fiction:  Horror:  Science-fiction:  Romance:  Fantasy: 	Expected result was displayed for each option
M21	None:  7.5 rows. Therefore 30 books Fantasy:  3 rows. Therefore 12 books	Expected result was displayed for each option 4 books per row. So finding number of rows and multiplying by 4 gives number of books. Or just count the number of books.

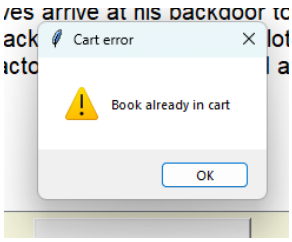
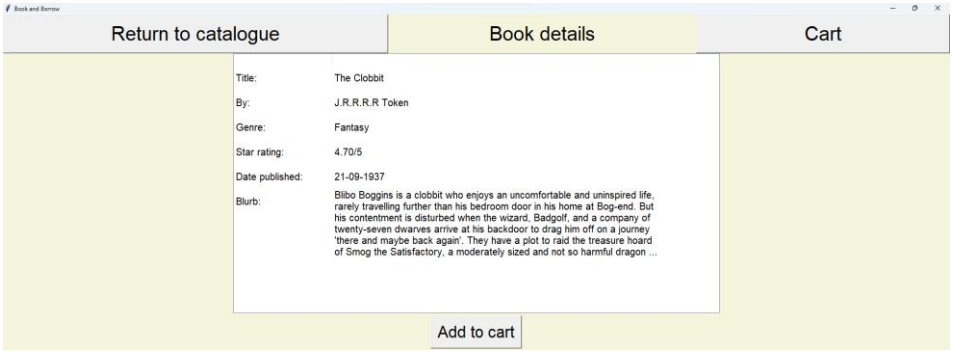
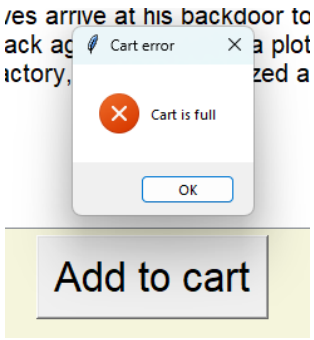
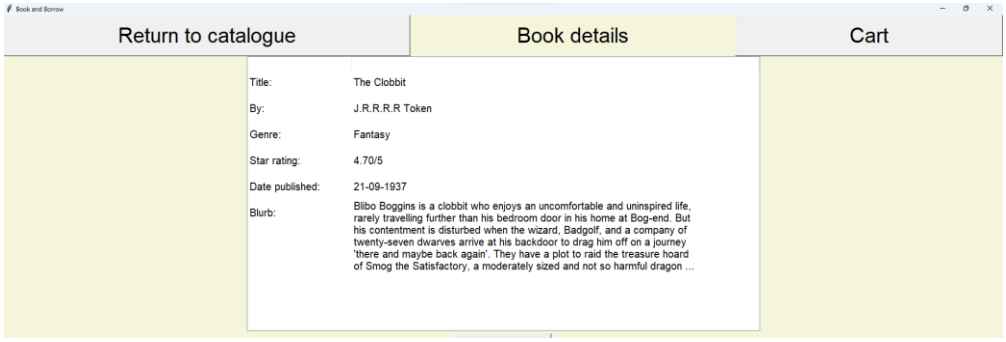
	Science-fiction:  5 books Non-fiction:  5 books Horror:  4 books Romance:  4 books	
M22	Window:  	Expected result was displayed

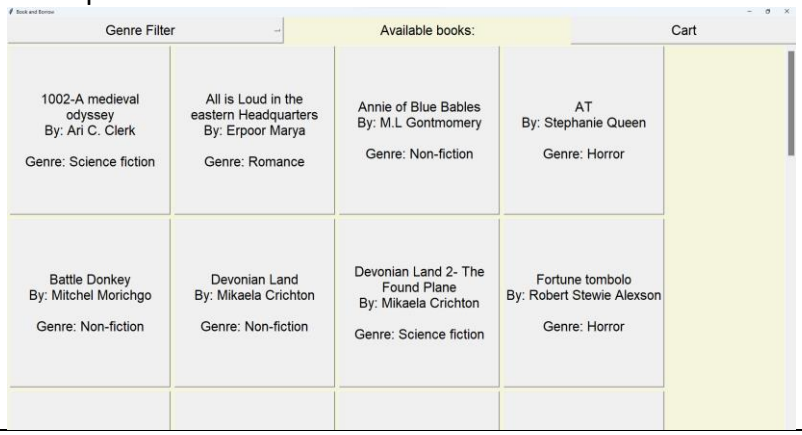
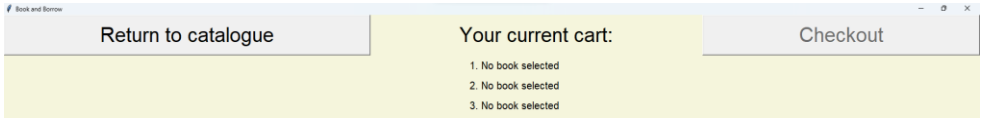
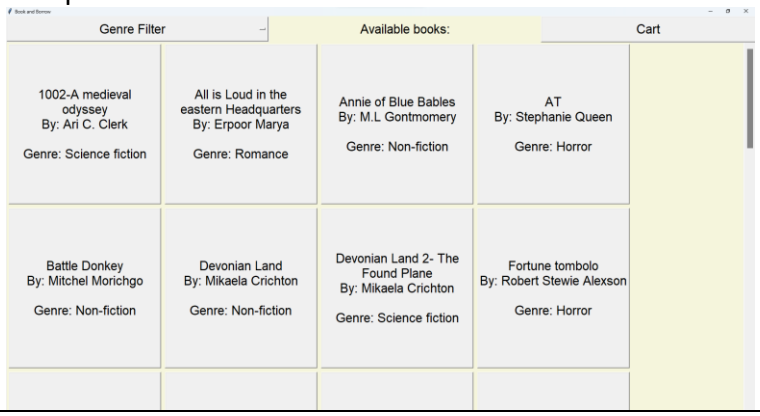
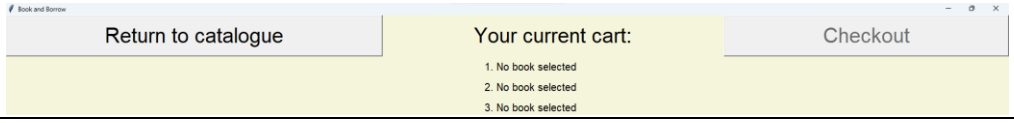
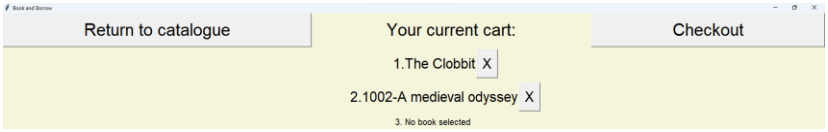
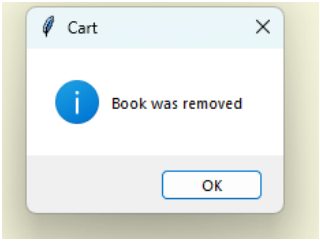
	Cart button pressed: 	
M23	Window:  Filter changed: 	Expected result was displayed
M24	Window  "The Clobbit" button pressed: 	Expected result was displayed

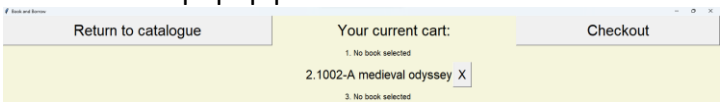
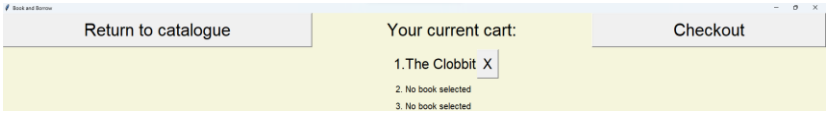
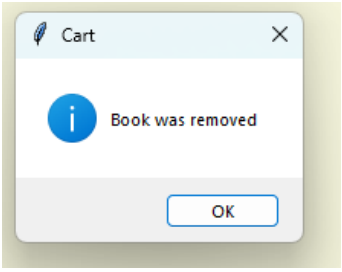
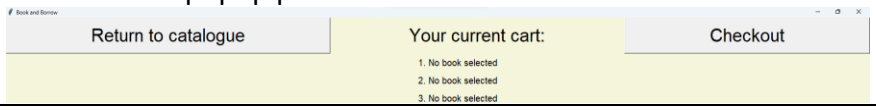
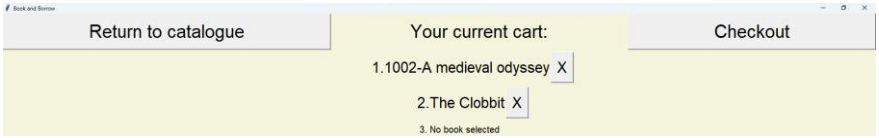
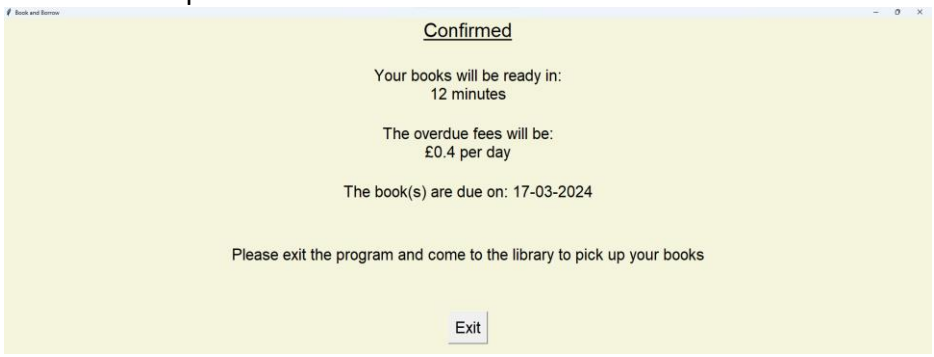
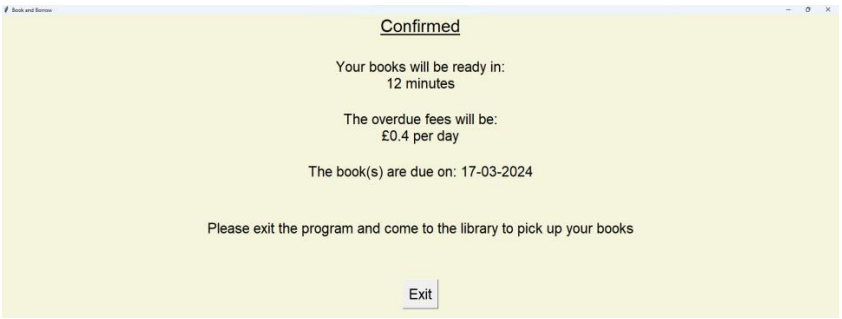
M25	Window:  Cart button pressed:  	Expected result was displayed

M26	Window:
	 
	
	Add to cart button pressed:
	 
	
	Output:
	`['', '', '']`
	`['9780261103344', '', '']`
		Expected result was displayed

M27	Window  	Expected result was displayed

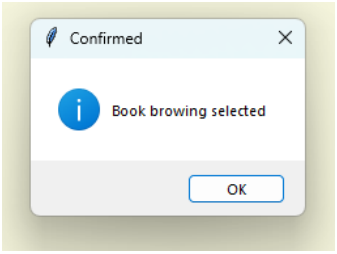
	Add to cart button pressed:  Output: `['9780261103344', '', '']` `['9780261103344', '', '']`	
M28	Window:  Add to cart button pressed:  Output: `['1', '2', '3']` `['1', '2', '3']`	Expected result was displayed
M29	Window:  	Expected result was displayed

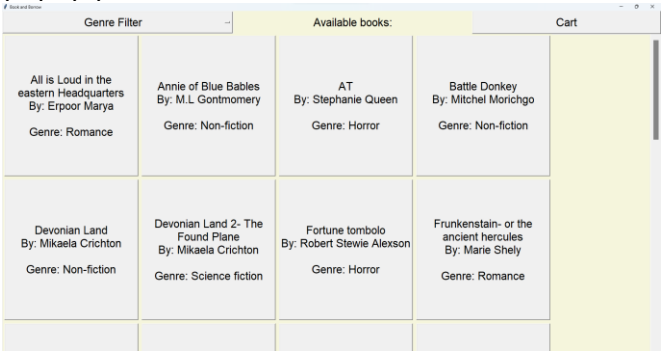
	Return button pressed: 	
M30	Window  Return button pressed: 	Expected result was displayed
M31	Code: `cart = [""] * 3` Window: 	Expected result was displayed
M32	Window:  |x| button pressed: 	Expected result was displayed

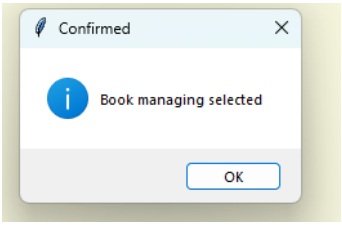
	Ok button on popup pressed: 	
M33	Modified code: `cart = ["9780261103344", "", ""]` Window  |X| button pressed:  Ok button on popup pressed: 	Expected result was not full displayed. Checkout button did become enabled after at least 1 book in cart. However, after the |X| button is pressed the checkout button remains enabled instead of becoming disabled again. See note on page 171 for more details
M34	Window  Checkout button pressed: 	Expected result was displayed. £0.4 is displayed instead of £0.40 because tailing 0's are removed. The way to fix this would be to add a conditional statement to add a tailing zero.
M35	Window  Exit button pressed: Window closed itself	Expected result was displayed

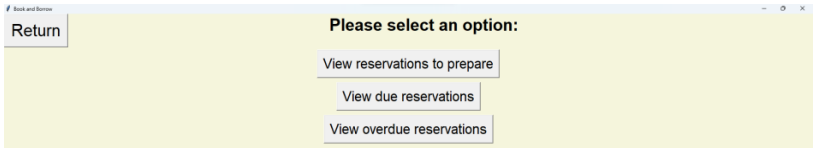
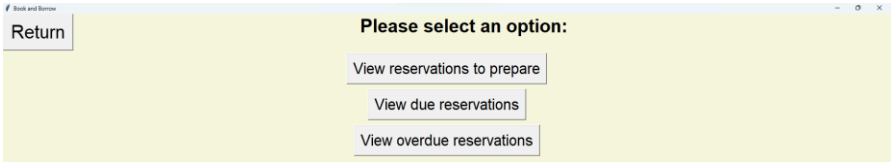
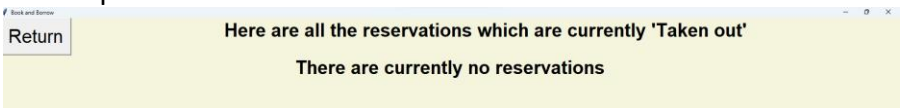
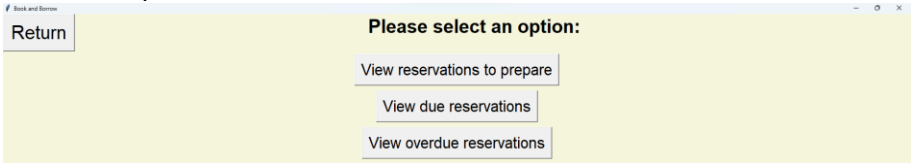
<table>
<tr>
<td>M36</td>
<td>Reservations table:
<table>
<tr><th></th><th>reservationID</th><th>username</th><th>dateOfOrder</th><th>timeOfOrder</th><th>dueDate</th></tr>
<tr><td>▶</td><td>5</td><td>DOLMIO2006</td><td>2024-02-18</td><td>11:09:26</td><td>2024-03-17</td></tr>
<tr><td>*</td><td>NULL</td><td>NULL</td><td>NULL</td><td>NULL</td><td>NULL</td></tr>
</table>
bookReservation table:
<table>
<tr><th></th><th>reservationID</th><th>ISBN</th></tr>
<tr><td>▶</td><td>5</td><td>9780099066101</td></tr>
<tr><td></td><td>5</td><td>9780261103344</td></tr>
<tr><td>*</td><td>NULL</td><td>NULL</td></tr>
</table>
Book table:
<table>
<tr><td>9780099066101</td><td>11</td><td>1002</td><td>A medieval odyssey</td><td>Reserved</td><td>4.17</td></tr>
<tr><td>9780192719980</td><td>8</td><td>Fortune tombolo</td><td>NULL</td><td>Available</td><td>4.15</td></tr>
<tr><td>9780199238552</td><td>15</td><td>The metasmorphosisis</td><td>and other stories</td><td>Available</td><td>4.98</td></tr>
<tr><td>9780261103344</td><td>1</td><td>The Clobbit</td><td>NULL</td><td>Reserved</td><td>4.70</td></tr>
</table></td>
<td>The reservationID is higher than previously thought. This is because of tests of the checkout button from M33 (see M33 note on page 171 for more details). But everything is still correct

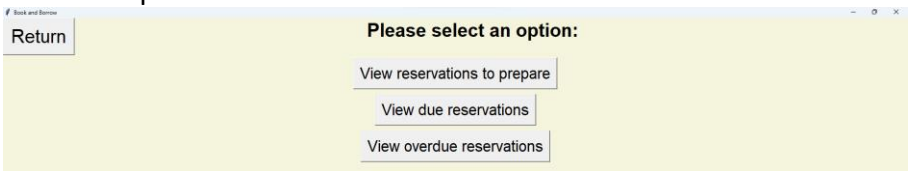
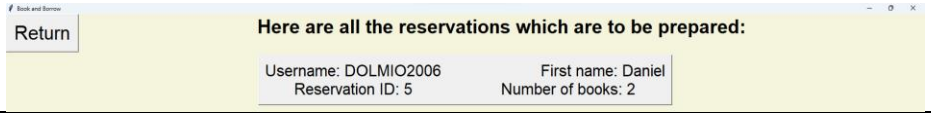
[Book table cut off for formatting in report]</td>
</tr>
<tr>
<td>M37</td>
<td>Window:

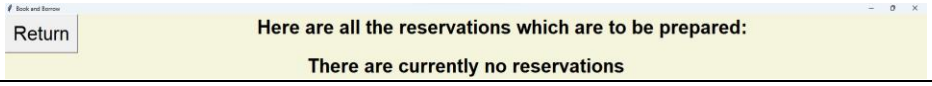

Browse button pressed:


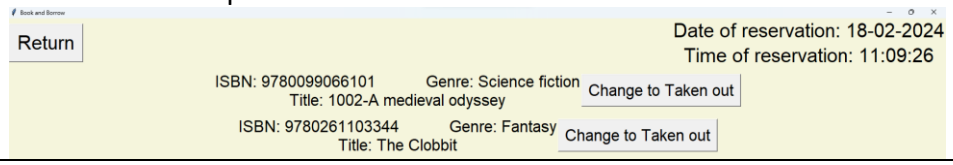
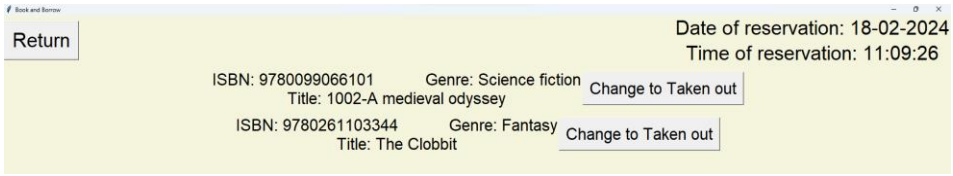
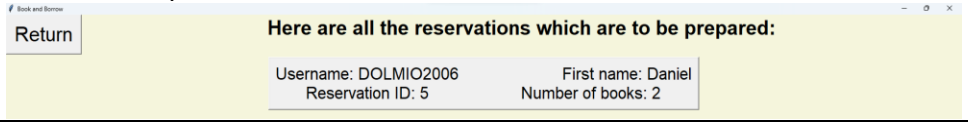
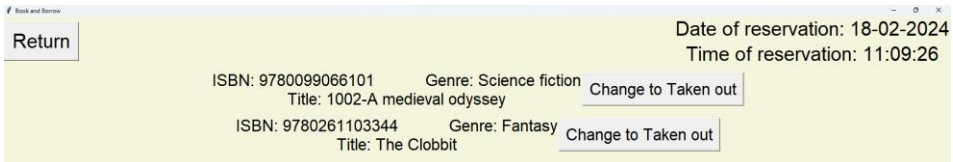
Ok button on popup pressed:


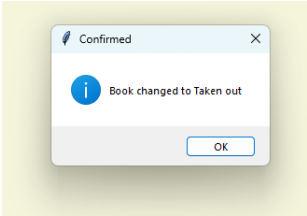
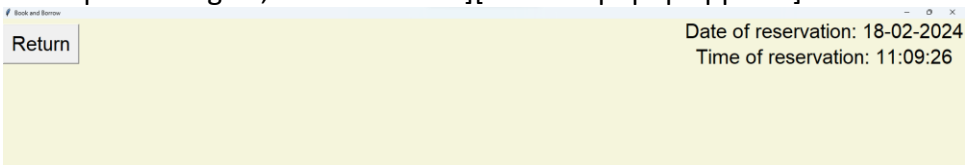
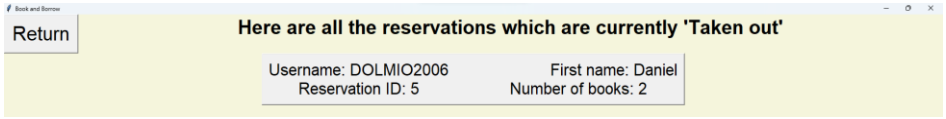
Manage button pressed:

</td>
<td>Expected result was displayed</td>
</tr>
</table>

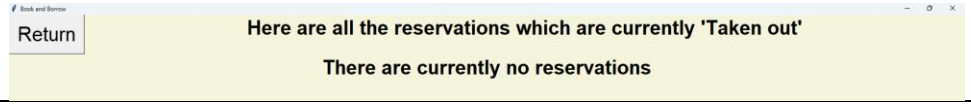
	Ok button on popup pressed: 	
M38	Window  Return button pressed: 	Expected result was displayed
M39	Window  To prepare button pressed:  Due button pressed:  Overdue button pressed: 	Expected result was displayed
M40	To prepare:  Return button pressed:  Due: 	Expected result was displayed

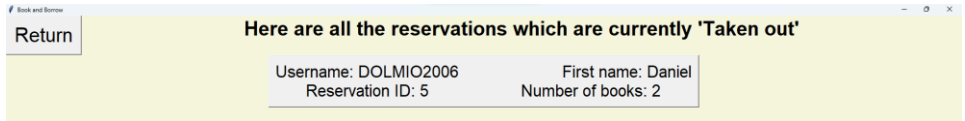
	Return button pressed:  Overdue:  Return button pressed: 	
M41	Window: 	Expected result was displayed

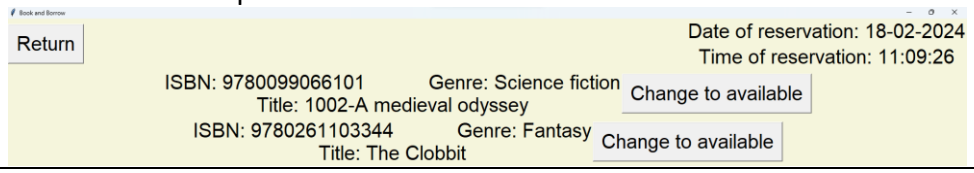
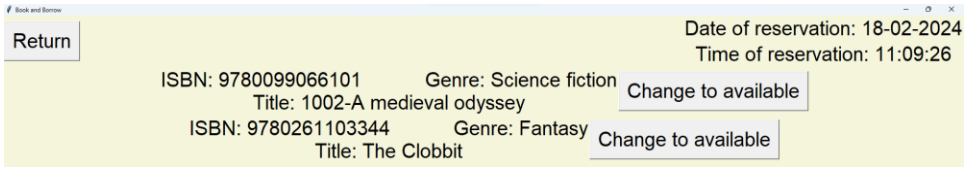
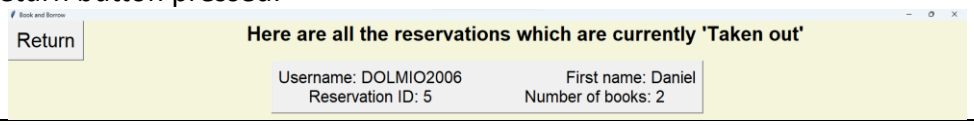
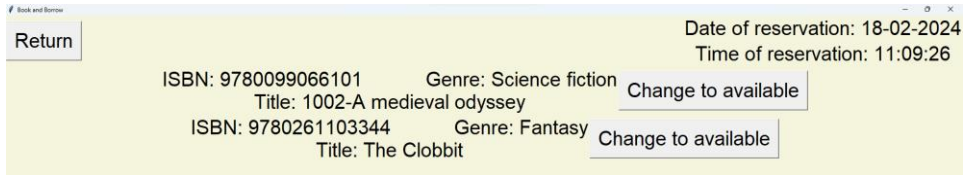
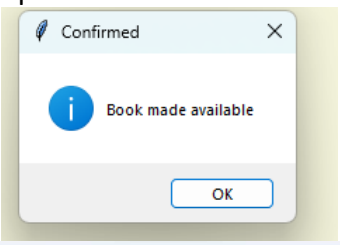
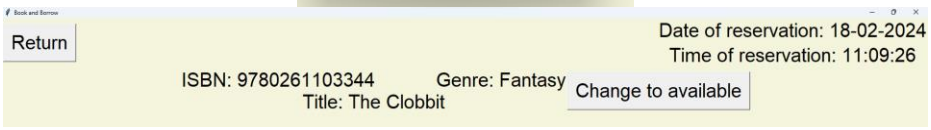
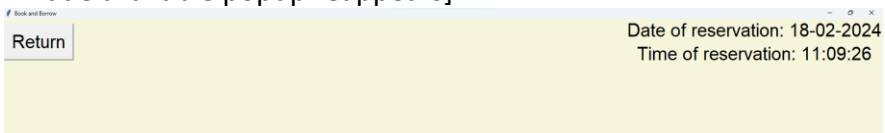
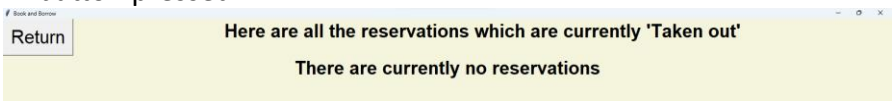
M42	Modified code:
	`#result = LQ.query_select(LQ.SQ8, ("Reserved",), app_details)`
	`result = ()`
	Window:
	 
		Expected result was displayed

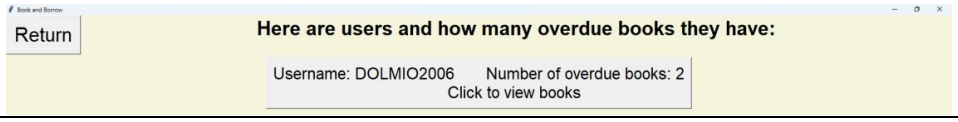
M43	Window:  Reservation button pressed: 	Expected result was displayed
M44	Window:  Return button pressed: 	Expected result was displayed
M45	Window: 	Expected result was displayed

	Change to taken out button pressed:  Ok button on popup pressed  [Button pressed again, for other book][Another popup appears]  Return button pressed:  Database books table: ISBN authorID title subTitle bookStatus starRating 9780007299263 10 Scream's Transport Home NULL Available 4.80 9780007523221 1 The Similarion NULL Available 4.65 9780099066101 11 1002 A medieval odyssey Taken out 4.17 9780192719980 8 Fortune tombolo NULL Available 4.15 9780199238552 15 The metasmorphosisis and other stories Available 4.98 9780261103344 1 The Clobbit NULL Taken out 4.70	
M46	Window:  	Expected result was displayed

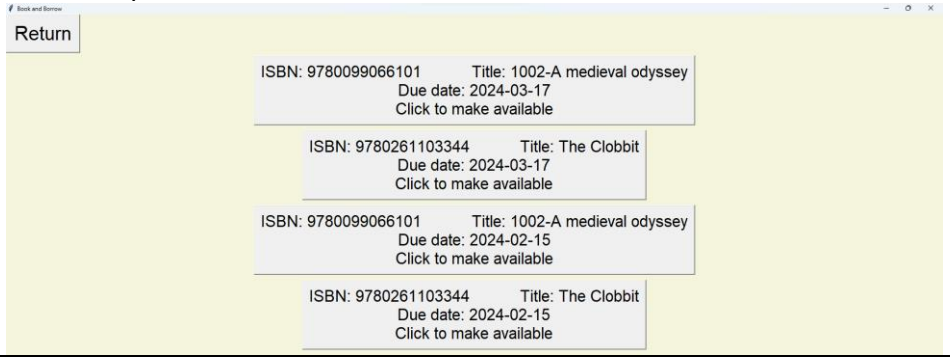
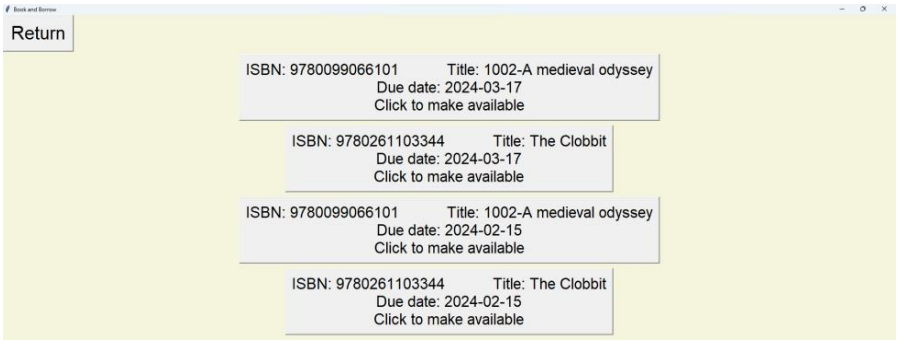
M47	Modified code:
	`#result = LQ.query_select(LQ.SQ8, ("Taken out",), app_details)`
	`result = ()`
	Window:
	 
	
		Expected result was displayed

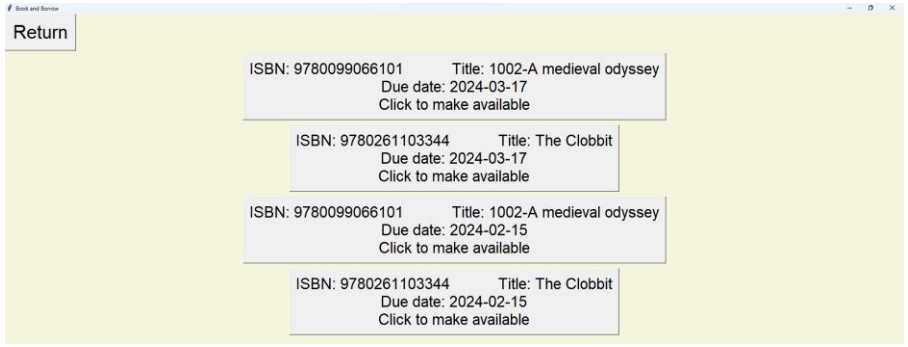
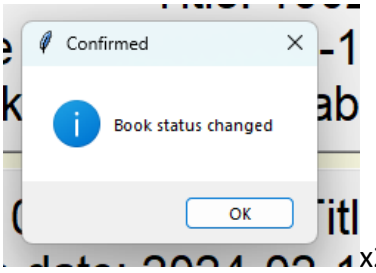
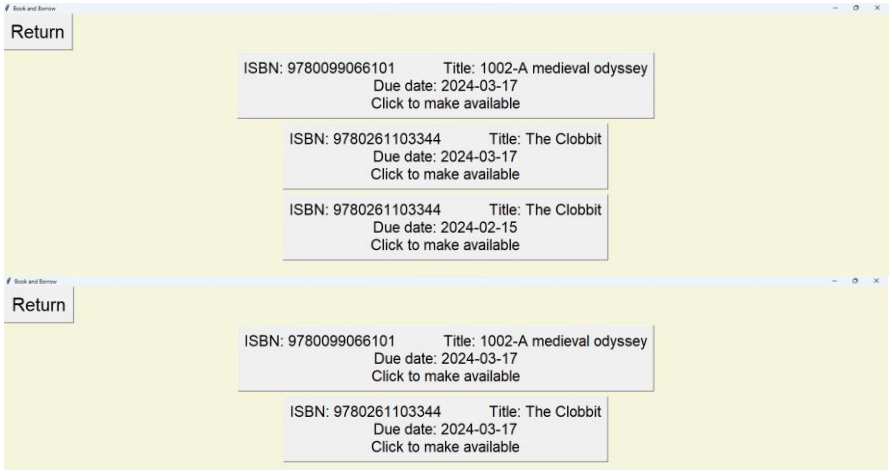
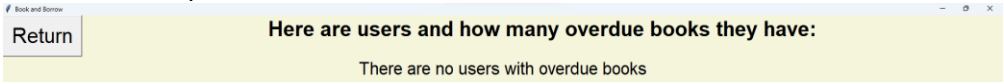
M48	Window  	Expected result was displayed

	Reservation button pressed: 	
M49	Window  Return button pressed: 	Expected result was displayed
M50	Window  Change to available button pressed:   Other status change button pressed: [Book made available popup reappears]  Return button pressed:  Database books table: ISBN authorID title subTitle bookStatus starRating 9780007299263 10 Scream's Transport Home NULL Available 4.80 9780007523221 1 The Similarion NULL Available 4.65 9780099066101 11 1002 A medieval odyssey Available 4.17 9780192719980 8 Fortune tombolo NULL Available 4.15 9780199238552 15 The metasmorphosisis and other stories Available 4.98 9780261103344 1 The Clobbit NULL Available 4.70	Expected result was displayed

M51	Setup code:
	`INSERT INTO reservations`
	`VALUES (default, "DOLMIO2006", "2024-01-18", "13:25:12", "2024-02-15");`
	
	`INSERT INTO bookreservation`
	`VALUES (6, "9780099066101");`
	
	`INSERT INTO bookreservation`
	`VALUES (6, "9780261103344");`
	
	`UPDATE books`
	`SET bookStatus = "Overdue"`
	`WHERE ISBN IN (SELECT ISBN FROM bookReservation WHERE reservationID = 6);`
	Window:
	 
		Expected result was displayed

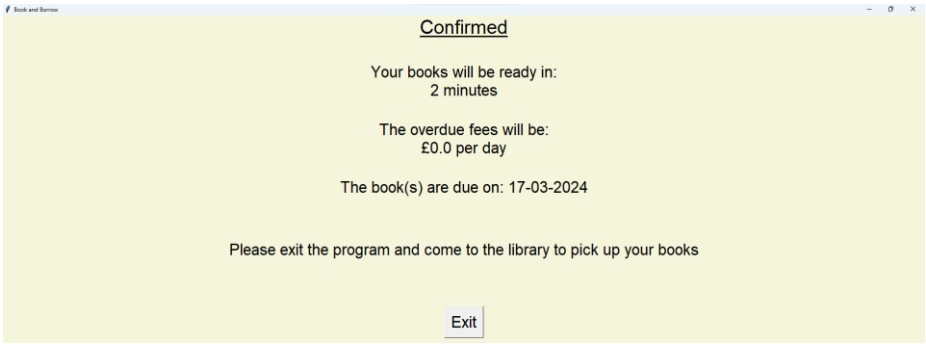
M52	Modifed code:
	`#result = LQ.query_select(LQ.SQ10, None, app_details)`
	`result = ()`
	Window:
	 
		Expected result was displayed

M53	Window:  User button pressed: 	The expected result was not displayed. The displayed result is both expected book reservations but also the previous ones from main file testing See note on page 172 for more details
M54	Window: 	Expected result was displayed

	Return button pressed: 	
M55	Window  The 3rd and 4th buttons pressed:  x2  Return button pressed:  Database book table: ISBN | authorID | title | subTitle | bookStatus | starRating 9780007299263 | 10 | Scream's Transport Home | NULL | Available | 4.80 9780007523221 | 1 | The Similarion | NULL | Available | 4.65 9780099066101 | 11 | 1002 | A medieval odyssey | Available | 4.17 9780192719980 | 8 | Fortune tombolo | NULL | Available | 4.15 9780199238552 | 15 | The metasmorphosisis | and other stories | Available | 4.98 9780261103344 | 1 | The Clobbit | NULL | Available | 4.70	Expected result was displayed

M33 note:

Test M33 found that if all books are removed from the cart the checkout button remains active.
By clicking on the checkout button in this state it is seen that a reservation is still created. However, there are no bookReservations created and no book statuses updated. This is due to the checkout function only using UPDATE queries on non-blank spaces. So, it catches the database error that could occur at this point.



	reservationID	username	dateOfOrder	timeOfOrder	dueDate
▶	4	DOLMIO2006	2024-02-18	11:04:14	2024-03-17
*	NULL	NULL	NULL	NULL	NULL

	reservationID	ISBN
*	NULL	NULL

[The ID is 4 due to the previous reservation tests of this checkout which have now been deleted from this table.]

This error wouldn't cause any trouble in a real system because since the overdue fees per day are 0. The user wouldn't build up overdue debt (if a system to do that was in place). The reservations also won't show up for the librarian since no book status is being changed, so it doesn't affect their system. This error just causes phantom reservations that belong to a user but reserve nothing.

This can be fixed by simply just updating the checkout button every time a removal button is pressed. The removal button could call a procedure that checks if the cart is empty and if so it then disables the checkout button.

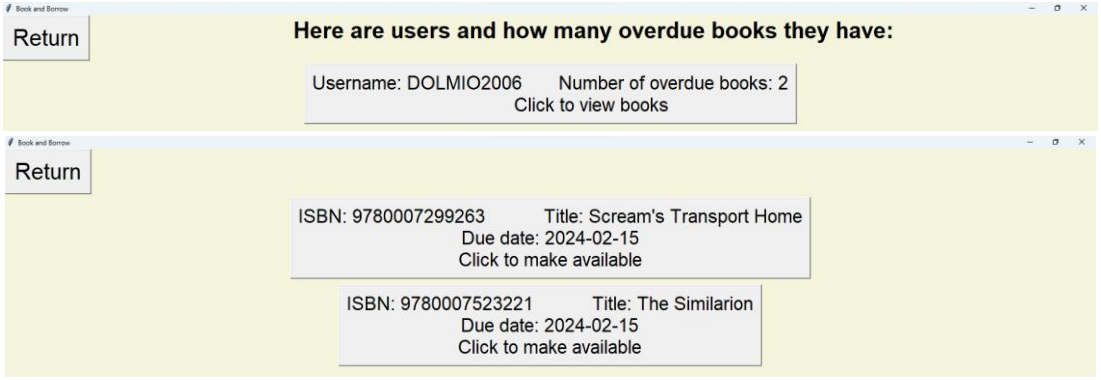
M53 note:

Test M53 found that the overdue books screen of a user creates 2 of every book. However, this is only under a certain test case. Since the M testing has used the same user and same two books (DOLMIO2006 reserving “1002” and “The Clobbit”) these are the only reservations in the database. As can be seen in the M53 result the separate copies of the books each have a different due date. This is because one pair of the books is from the earlier testing in M35, M45 and M50, while the other pair of books is from M51.

This error has only occurred because of using the same two books for all the testing. This would only happen in actual use of the system if a user repeatedly takes out a book and it goes into overdue.

Here is what M53 is like with unique books:

```
INSERT INTO reservations
VALUES (default, "DOLMIO2006", "2024-01-18", "13:25:12", "2024-02-15");

INSERT INTO bookreservation
VALUES (7, "9780007299263");

INSERT INTO bookreservation
VALUES (7, "9780007523221");

UPDATE books
SET bookStatus = "Overdue"
WHERE ISBN IN (SELECT ISBN FROM bookReservation WHERE reservationID = 7);
```



This isn’t a harmful error but may cause confusion due to the duplication. Pressing the change status of either of the books will change the status of the book to “Available” from “Overdue”. However, this error could cause trouble eventually as if those 2 books are taken out by “DOLMIO2006” again, there will be 3 of every book. Then if the user does it again, 4 of each book etc.

This could be fixed by splitting SQ11 into two queries. Have the main query get the ISBN, title and subtitle while the other one gets the dueDate of all overdue reservations. This would allow grouping to be done to the first query which would get rid of duplication and still allow due date to be seen. Since this error is caused because grouping cannot occur as each book copy has a different dueDate and therefore cannot be grouped by similar values.

Other testing notes

Extras file:

The tests that were not planned or completed were:

- If the index parameter of binary_search exceeds the number of items in each array
- If a 1D array is passed into the binary_search as the array parameter
- If a non-positive, non-integer is passed into the index parameter of binary_search

These situations were not tested as they would not ever occur in the software.
The binary_search function is only used during login/signup and always receives a 2D array (from the query result) and only ever searches the first item in each row because that is where the username is.

End user testing

Borrower

Persona	
Name	Seumas Macleod
Age	67
Backstory	A retired teacher who lives with his wife. He is unable to go out for long periods of time and can't walk well. He loves to read books and talk about them with his grandson and enjoys reading new books. His local library just implemented Book & Borrow and he wants to use it. After he tries it out he arranged to go down and pick his order up with his grandson and return home so they can read books together in his living room.
Tasks to complete	• Create an account - Using details from persona name, age etc. • Browse books • Apply a genre filter • View a book's details • Add a book to cart • View you cart • Remove a book • Fill cart to desired amount, with any books • Checkout

Tester feedback	
Name of tester: Caitlin Loynd	**Rating: 10/10**
How easy was the system to use?	
Very easy	
Any difficulties?	
No	
What would you like changed?	
It would be good if I could see the password when inputting it. The text for the "No book selected" on the cart could be larger.	

Comment on tester's performance
The tester seemed to get along fine with the system and was easily able to complete all of their tasks. The tester even decided to try and use other genre filters even after viewing book details. After that they also decided to add another book to cart. They then tried adding that book again and received the "Book is already in cart" popup. Overall: The system worked without much trouble and the user enjoyed reading about the different books and found the whole process simple and quick.

Librarian

The tester will sit at the computer and complete their tasks. There will be several piles of books which the tester will need to use in their tasks

Persona	
Name	Mary Campbell
Age	70
Backstory	A librarian that has worked at the library for 40 years. She has very little experience with technology but knows it can help her in her job. She has poor eyesight and needs simple displays. However, she is optimistic about the new system and wants to see what she can do with it.
Tasks to complete	• Login - Username: marycampbell53 - Password: ILoveBooks!!! • Check for any reservations to prepare • Set aside those books (from piles) • Check for any taken out books • Change the status of books which have now been returned (from piles) • Check for any overdue books • Change the status of any returned books (from piles) • Browse books

Tester feedback	
Name of tester: Caitlin Loynd	**Rating: 7/10**
How easy was the system to use?	
It was fairly easy	
Any difficulties?	
I was confused about what some buttons did.	
What would you like changed?	
A pop up information instruction for different buttons would be helpful for technologophobes.	

Comment on tester's performance
The tester seemed to struggle to complete the tasks assigned to them. They often sat trying to figure out what to do with the books and where to go. The "Check for any taken out reservations" task caused trouble as the button linked to it was "View due reservations". Overall: The user didn't enjoy using this system as much as the borrower section. They mostly wanted clearer instructions for what everything in the system does.

Evaluation

Fitness for purpose

Borrower section

Following testing using the test cases defined in the test plan, this section of software can be deemed unfit for purpose.
It meets all the functional requirements; however, it does not meet all of the input validations and end user requirements.

Validations

The checkout button remains enabled after all book are removed from cart. This then gives the chance to checkout nothing. This breaches the validation to make it so user's can only checkout with 1-3 books.

End user requirements

The error screen display does not have an easy to read display and a beige background. This breaks the borrower end user requirements. The borrower requirements state that everything should be easy to read and easy on the eyes. The error screen breaks this requirement.

Therefore, due to these reasons this section of software is deemed unfit for purpose.

Librarian section

Following testing using the test cases defined in the test plan, this section can be deemed unfit for purpose.
It meets most of the functional and end user requirements but doesn't meet some of them.

Functional requirements

This does not retrieve all of the overdue books of the current user. SQ11 retrieves unexpected results and therefore incorrect information for the librarian. This means that the librarian section does not meet all of its functional requirements.

End user requirements

As found by the end user testing on page 175 it can be seen that this section of the software is confusing and is not well explained or intuitive. The tester found it difficult to understand where to go to complete all their tasks and was confused by the displays of book details.

Therefore, due to these reasons this section of the software is deemed unfit for purpose.

Queries

Following testing using the test cases defined in the test plan, it can be said that the DML queries are fit for purpose and the SQL queries are unfit for purpose.

DML & DDL

These queries all work perfectly fine. Creation of new users, reservations and updates to statuses all work with no unexpected outputs or errors. Therefore, these queries are fit for purpose.

SQL

Most of these queries are fit for purpose. All except SQ11 which due to test case M53 can be seen to retrieve unexpected books (explained in M53 note on page 172). Due to it retrieving these unexpected results, SQ11 is unfit for purpose

Overall

In general, the "Book & Borrow" system can be deemed unfit for purpose. This is due to requirements and validation being missed and a query which doesn't function properly under all circumstances.

Robustness

Borrower section

Following testing using the test cases defined in the test plan, it can be said that this section of the software is not robust. All of the input validations for user login and sign up works. However, not all the buttons have their correct disabling and enabling.

The checkout button is the only button that causes this system to not be robust. The checkout button does not become disabled again after the user removes all their items from their cart, thus allowing the user to press the button. This means that not all validation completely works as they should not be able to press this button with nothing in cart. Therefore this system cannot be deemed robust.

Librarian section

Following testing using the test cases defined in the test plan, it can be said that this section of the software is robust. All of the buttons in the software do not allow the user to mess up the system and create an error. The popups prevent users from clicking off the window and back on and doing something else (since the popups remain and are high priority to the software and must be acknowledged before any more actions occur).

Due to the way the buttons, window and popups are set up to prevent errors, this section can be deemed robust.

Database

The database contains validation on many of the different attributes for all the entities. This includes:

- Presence checks
  - In the form of the NOT NULL statements for all the attributes that must have values
- Range checks
  - In the form of the checks for the length of ISBN, usernames, passwords etc.
- Restricted choice
  - In the form of the 5 specific genres that a book can have
  - Also, in the form of the 4 specific statuses that a book can have

These all make it so the entered data is correct. If any of these validations don't work then an error occurs, this is not handled by the actual database. However, by using the TRY CATCH(es) in the query execution modules in the queries file the errors are handled. Any database error caused by a breach of this validation will display the error screen to the user. Thus, handling the error.

Therefore, it can be deemed that own its own and in the system. The database is robust and can handle different data inputs.

Overall

In general, the "Book & Borrow" system can be deemed mostly robust. As it does not break with input from textboxes but could break because of the checkout button. Due to that checkout button having the miniscule chance of causing troublesome errors, the system is not completely robust against all inputs.

Efficiency

Software

The code makes use of modules that need to be repeated several times from different files.
This includes:

- Query execution modules
- Formatting procedures
- And other modules in the Extras file

This reduces the amount of repeated code inside the software.
Example:

Each query execution would need: a try catch, a cursor definition and a result fetch/a database commit. By using the modules this is reduced to just a single line that calls on a module from another file. Thus, increasing the efficiency by removing needless redefinitions of a MySQL cursor and reducing file size of the main file.

It makes use of fixed loops to format widgets on the window and to bind widgets to specific events. All of the widgets in the login and sign up screens are bound to a key release event; the binding occurs in a loop that does this all quickly. The alternative to this would be to add a line underneath every single widget definition to bind it, which would be inefficient.

However, the entirety of the code could be made more efficient using object-oriented programming. There are several repeated return buttons, this could be made into a return button class that is given a window it returns to. The entire screen setup could be made into a sort of linked list with the return buttons acting as the previous pointers that link to previous modes. The return button class could also be a subclass of a button superclass that could be used for many other things in the software.

Therefore, it can be deemed that the software is efficient as possible using only procedural programming techniques. If this project was to be redone it would be created using the object-oriented paradigm.

Queries and Database

Overall looking at all the queries, it can be deemed that they are inefficient. This is because of SQ11 which retrieves unexpected results which overcomplicate the query result.

The database as well can be deemed inefficient. The bookReservation table is a weak entity and therefore has a compound key, this could be improved by changing it to have a surrogate key, thus making it a strong entity. This would increase the efficiency of the queries on the database as they no longer need to deal with using a compound key to link tables in a query.

Overall

Overall, it can be deemed that the system is inefficient. This is due to lack of object-oriented programming, SQ11 and the structure of a database table.

Usability

Borrower section

Following testing using the test cases defined in the test plan, it can be deemed that this section of the software is easily usable.
The results of end user testing indicate that the end user of the borrower section can easily understand the steps of what to do without needing any guidance. The layout and the large text all indicate what each item on the window does, allowing a quick and easy system with no difficulties.

Therefore, this section of the software can be deemed easily usable.

Librarian section

Following testing using the test cases defined in the test plan, it can be deemed that this section of the software is not easily usable.
The results of end user testing indicate that the end user of the librarian section struggled to understand what each item of the window did. They also struggled to understand what to do at each of the reservations/books screens. They were unable to complete all their tasks without some guidance about what each part of the software does.

Therefore, this section of the software can be deemed not easily usable.

However, if this was to be implemented in a real-life situation the librarian staff would first receive training about how to use their system. So, a user manual would have to be created a for understanding and use of this section in a real-life situation.

Overall

In general, the “Book & Borrow” system can be deemed mostly usable. Though it does have difficulties in the librarian section, as stated the staff would receive training for it. This still does mean though that the system cannot be deemed fully easily usable.

Maintainability

Software

The maintainability of my software is not great.
The module names in my software are flawed and can cause confusion. This can be seen in the overdue_reservations module which has nothing no to do with overdue reservations, instead it is about users with overdue books. So, it should instead be called overdue_users.
At parts of the software there are variable names that are somewhat confusing and don't really explain what they hold. [Find example of it]

There are also global variables in my code. This was made for easy to access variables that are used many times like the app_details tuple and the current user's username string. This will cause problems however if they are ever made non-global as they will need to be passed into very single main file procedure.

All the main file procedures have bad structures. It starts by defining any procedures which are for events and then any for procedures called by buttons. Then it creates frames, and then all the widgets. After that it fonts and styles everything (if not previously done). The fact that the button procedures are defined in the procedure is bad as they should be moved outside as separate modules so they don't bloat the main procedure they are linked to and cause readability to worsen. The event procedures though are fine and should stay there.

The main file is also unmaintainable because of its length. The over 1000 lines of code that define it make it difficult to navigate. This will cause problems if there is an error that crosses between modules as the debugger will need to scroll a 'lot to find the specific lines.

However, the one maintainable thing about my code is that there is a 'lot of internal commentary. Almost every single process and line of code is explained by a comment saying what they do. This is what has added many lines to the 1000-line count, but it's necessary to understand parts of the code. This makes it so that every module can easily have all of its steps explained but it still doesn't fix their flawed structure.

Database

I would say that the database is maintainable. There are sufficient attribute names meaning that you can easily tell what each attribute describes for each entity. There are also validations which checks the data inputs ensuring no unfit data is entered. These make it so data can easily be entered or removed easily as the editor knows what everything means. Though in the system this won't actually happen, but if an emergency change is needed, it will be easily done.

Overall

Overall, I would say that my system is not maintainable. There are too many flaws in the software to deem it maintainable. If the modules were renamed to be better and their structures altered then I might be able to deem it mostly maintainable. But it won't be completely maintainable unless a full overhaul is done to it.

Corrective maintenance

Checkout button

A fix in the code to make it disabled after all books have been removed. Have the checkout button be regenerated with the book titles after every removal. Once a book is removed it refreshes all the displayed books and refreshes the checkout button. After this refresh the checkout button will check for books and will alter its status accordingly.

SQ11

This was explained in the M53 note on page 172.
SQ11 would be split into two queries, a SQ11A and SQ11B.
SQ11A: Would get all the book details with an overdue status, grouping them by the book details (This would remove the duplication problem that appears)
SQ11B: Would get all the due dates of the reservations that these books were in. (This is moved to a separate query to allow the grouping in SQ11A to take effect).

These would then just be called upon separately like SQ8A and SQ8B and their details used in parallel 2D arrays (parallel array of tuples) to display all the details.

Error screen

A fix to make the text display a lot nicer using more new lines ("\n") would be used. The fix would also increase the font size of the text and add a beige background to the window. This would fix the problems with the screen and make easier to view, thus making it reach end user requirements.

Librarian section

To fix the problems with the librarian section there are two options: a user manual is created to explain it or buttons to display popup boxes with information are created

Perfective maintenance

Dropdown box/popup messages

- I would like to find a way to change the font family and size of the text that appears on the dropdown box and the popup messages. As they are quite difficult to read sometimes.
- This would increase usability of the software.

More books

- I would like to add more books to the system to make it seem more like a library. I would also like to add other genres as well and even a way for books to have multiple genres. These were cut though due to time constraints for this project.

A way to scan books using a barcode reader

- A system that allows librarians to use a barcode reader and scan the books and that will update the books status. This was not implemented as it would cause the complexity and time to make this project increase drastically, as I would've needed to learn how to integrate with a barcode reader and learn how they read the data.

The borrower login link to library cards

- A way to login using a library card instead of needing to create an account. This would also need a system to sign up to make a library card. This was cut in favour of the much quicker to implement basic system. Though if this was to be implemented in a real-world library, this system would be beneficial to the borrowers

Total overdue fees to pay for each user

- A way to view account details for each user. In this they would be able to view their total overdue fees that they have built up.

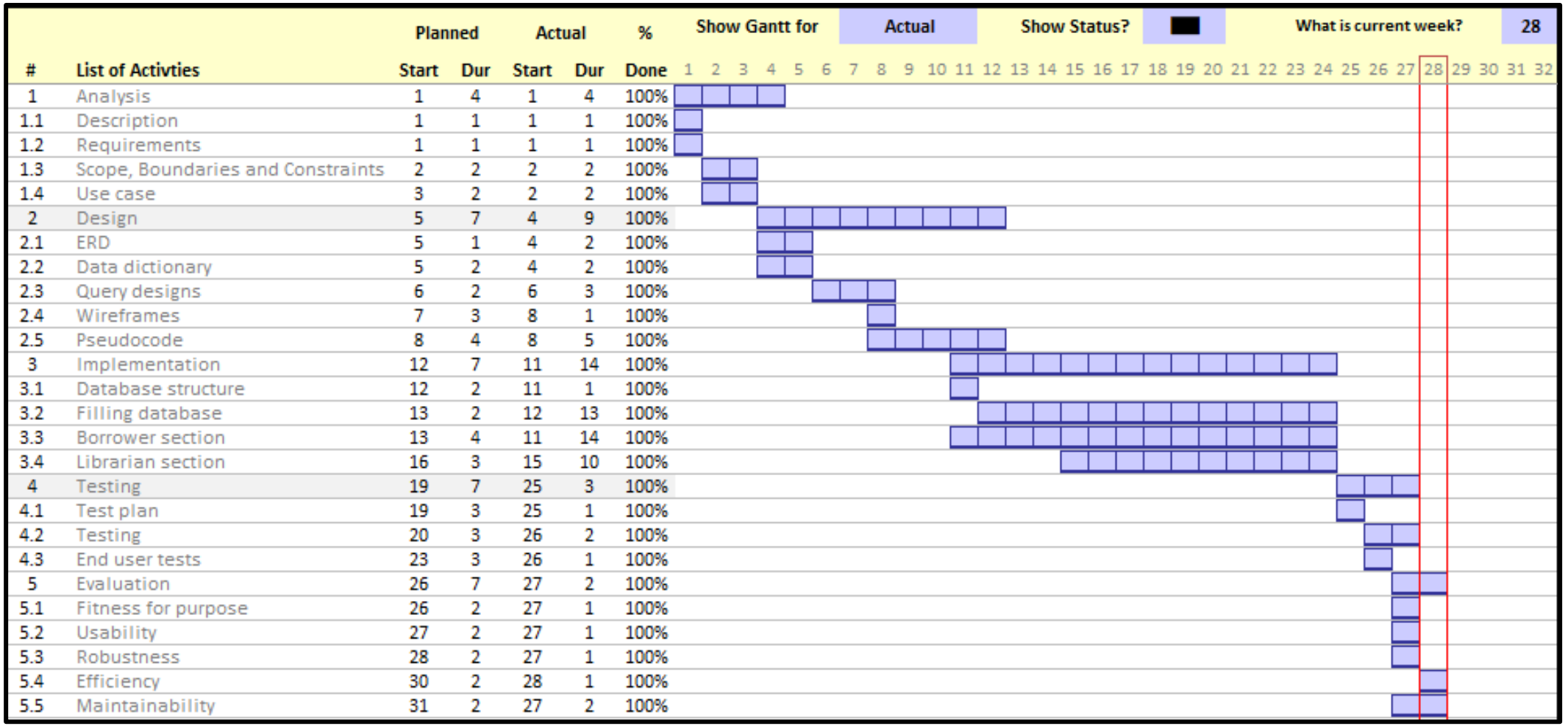
Email users when books are almost due/ are overdue

- The sign up would take an email at account creation. This would now be the primary key value for the users table. There would be an automatic system in place that would check for all books that are almost due/overdue and email the users to tell them this information. This was cut as this would also increase the time to complete this project, which I can't due to the time constraints. This system would also by highly beneficial to borrowers in a real-world library system.

Ability to show password after a button press during login/sign up

- This would allow users to reveal their password during login/sign up, allowing the users to login or sign up much easier as they can view any password input mistake they made. This was not implemented as it didn't even cross my mind until it was brought up during the end user testing on pages 174-175.

Final Gantt chart



#	List of Activties	Planned Start	Planned Dur	Actual Start	Actual Dur	% Done
1	Analysis	1	4	1	4	100%
1.1	Description	1	1	1	1	100%
1.2	Requirements	1	1	1	1	100%
1.3	Scope, Boundaries and Constraints	2	2	2	2	100%
1.4	Use case	3	2	2	2	100%
2	Design	5	7	4	9	100%
2.1	ERD	5	1	4	2	100%
2.2	Data dictionary	5	2	4	2	100%
2.3	Query designs	6	2	6	3	100%
2.4	Wireframes	7	3	8	1	100%
2.5	Pseudocode	8	4	8	5	100%
3	Implementation	12	7	11	14	100%
3.1	Database structure	12	2	11	1	100%
3.2	Filling database	13	2	12	13	100%
3.3	Borrower section	13	4	11	14	100%
3.4	Librarian section	16	3	15	10	100%
4	Testing	19	7	25	3	100%
4.1	Test plan	19	3	25	1	100%
4.2	Testing	20	3	26	2	100%
4.3	End user tests	23	3	26	1	100%
5	Evaluation	26	7	27	2	100%
5.1	Fitness for purpose	26	2	27	1	100%
5.2	Usability	27	2	27	1	100%
5.3	Robustness	28	2	27	1	100%
5.4	Efficiency	30	2	28	1	100%
5.5	Maintainability	31	2	27	2	100%

Name: Daniel Monaghan
SCN: 121667622