**Scripting and Programming concept**

**MUHAMMAD AKTERUZZMAN**

**SID- 2216724**

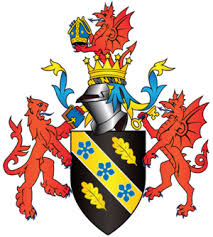


Table of contents-

Introduction ……………………………………………………………………3

1. Task-1 …. Analysis……………………………………………………………..4
2. Task-2 software design and development ……………………………..5
3. Flowchart and pseudocode
4. Implementation decision
5. Source code
6. Testing and Debugging Report
7. Evaluation
8. Reference

**Introduction**

Libraries are invaluable community resources, providing access to information, educational materials, and cultural enrichment. However, managing a library’s operations manually, as with the Great Hartland Community Library, can lead to inefficiencies and errors. The manual system requires volunteers to track book records, handle check-ins and check-outs, and manage book searches using handwritten or printed logs. This process is time-consuming and prone to lost records, duplicate entries, and misfiled information.

A digital solution is critical to address these challenges. A computer-based library management system can streamline operations by automating record-keeping, facilitating accurate book tracking, and improving search capabilities. This, in turn, enhances service quality for patrons and reduces the workload on library volunteers. Furthermore, a digital system can adapt to future needs, such as scaling the library’s collection or integrating online features.

The proposed solution will be a prototype for such a system, focusing on the core functionalities needed to demonstrate its effectiveness. The solution aims to highlight how technology can modernise library management and improve overall efficiency by creating a user-friendly interface and implementing essential features.

**Requirements Analysis**

To meet the needs of the Great Hartland Community Library, the proposed digital solution should provide the following functionalities:

1. **Book Record Management**
   * The system must allow library staff to add, edit, and delete book records. Each book record should include details such as:
     + Title
     + Author
     + Year of Publication
     + ISBN (unique identifier)
     + Availability Status (e.g., available or checked out)
   * This ensures accurate library collection tracking and eliminates the risk of duplicate or missing entries.
2. **Search Functionality**
   * Patrons and staff should be able to search for books quickly and easily. The search feature must support:
     + Searching by Title
     + Searching by Author
     + Searching by ISBN
   * Partial matches should be supported to accommodate incomplete information. For example, searching for "Rowling" should display all books authored by J.K. Rowling.
3. **Check-Out and Return System**
   * The system should facilitate book borrowing and returns.
   * When a patron borrows a book, the system should mark the book as “checked out,” updating its availability status.
   * Similarly, returned books should be marked as “available” to ensure their status is up to date.
   * Attempts to check out an unavailable book or return a book that is not checked out should trigger an appropriate error message.
4. **List All Books**
   * Library staff should have the option to view a complete list of all books in the collection.
   * Each entry in the list should display key details such as Title, Author, Year, ISBN, and Availability Status.
   * This feature allows staff to manage the library’s inventory at a glance and identify overdue books or items with low availability.
5. **User-Friendly Interface**
   * The solution should be designed with simplicity in mind, allowing staff with minimal technical experience to navigate the system easily.
   * The menu-driven interface should guide users through different functionalities without confusion.
6. **Data Validation**
   * Input validation is critical to ensure that users do not accidentally enter invalid data, such as non-numeric years or duplicate ISBNs.
   * Validation safeguards the system against common errors, improving reliability and accuracy.

**Task-2**

1. Flowchart

**Flowcharts**

Below are the flowcharts for the main functionalities of the library system:

1. **Main Menu Flowchart**

**Description:** The main menu displays available options to the user. Based on the user’s input, the corresponding functionality (e.g., adding a book, searching for a book) is triggered, or the program exits.

1. **Add Book Flowchart**

**Description:** This process gathers book details, validates the inputs (e.g., year must be numeric), and adds the book to the collection if all inputs are valid.

1. **Search Book Flowchart**

**Description:** The user selects the search criteria (e.g., Title, Author, or ISBN), inputs a keyword, and the system displays matching results or an appropriate message if no matches are found.

1. **Check-Out Book Flowchart**

**Description:** The system prompts the user for the ISBN, verifies its validity, checks if the book is available, and updates the status if the check-out is successful.

1. **Return Book Flowchart**

**Description:** The system accepts the ISBN, verifies its validity, checks if the book is checked out, and marks it as available if returned successfully.

A screenshot of a computer

Description automatically generated

Flow chart using by software

**Pseudocode-**

Main Menue

FUNCTION main\_menu():

WHILE True:

PRINT "Library Menu: "

PRINT "1. Add Book"

PRINT "2. Search Book"

PRINT "3. Check Out Book"

PRINT "4. Return Book"

PRINT "5. List All Books"

PRINT "6. Exit"

GET user\_choice

IF user\_choice == "1":

CALL add\_book()

ELSE IF user\_choice == "2":

CALL search\_book()

ELSE IF user\_choice == "3":

CALL check\_out()

ELSE IF user\_choice == "4":

CALL return\_book()

ELSE IF user\_choice == "5":

CALL list\_books()

ELSE IF user\_choice == "6":

PRINT "Exiting system."

BREAK

ELSE:

PRINT "Invalid option. Try again."

**ADD BOOk**

FUNCTION add\_book():

GET title

GET author

GET year

VALIDATE year IS integer

IF NOT:

PRINT "Invalid year. Try again."

RETURN

GET ISBN

VALIDATE ISBN IS unique

IF NOT:

PRINT "Duplicate ISBN. Try again."

RETURN

ADD book to books list with availability set to True

PRINT "Book added successfully."

**FUNCTION search\_book():**

PRINT "Search by Title, Author, or ISBN: "

GET criteria

GET keyword

IF criteria == "Title" OR "Author" OR "ISBN":

FILTER books WHERE criteria contain keyword

IF matches exist:

DISPLAY matches

ELSE:

PRINT "No books found."

ELSE:

PRINT "Invalid criteria. Try again."

**FUNCTION check\_out():**

GET ISBN

FOR each book IN books:

IF book.ISBN == ISBN:

IF book.Available == True:

SET book.Available = False

PRINT "Book checked out successfully."

RETURN

ELSE:

PRINT "Book already checked out."

RETURN

PRINT "Book not found."

**FUNCTION return\_book():**

GET ISBN

FOR each book IN books:

IF book.ISBN == ISBN:

IF book.Available == False:

SET book.Available = True

PRINT "Book returned successfully."

RETURN

ELSE:

PRINT "Book was not checked out."

RETURN

PRINT "Book not found."

**FUNCTION list\_books():**

IF books IS empty:

PRINT "No books available."

RETURN

FOR each book IN books:

PRINT book.Title, book.Author, book.Year, book.ISBN, book.Available

(b) **Implementation**

**choice of Data Structures**

For the proposed library management system, the following data structures were chosen:

1. **List of Dictionaries**
   * **Reason for Choice**:
     + A list is used to store the collection of books, with each book represented as a dictionary.
     + Dictionaries allow for flexible key-value pairing, making accessing book attributes like title, author, year, ISBN, and availability easier using descriptive keys.
     + Lists provide ordered storage, enabling efficient traversal during search or display operations.

books = [

{"title": "Book Title", "author": "Author Name", "year": 1992, "ISBN": "123456", "available": True},

{"title": "Another Book", "author": "Another Author", "year": 1998, "ISBN": "789012", "available": False}

]

**Variables**

* **title**: A string representing the book title.
* **author**: A string holding the author’s name.
* **year**: An integer denoting the publication year.
* **ISBN**: A string used as a unique identifier for books.
* **available**: A Boolean (True or False) to track whether the book is available for borrowing.

These variables ensure clarity, with self-explanatory names that align with standard library systems.

**Programming Constructs**

**Control Structures**

1. **Conditional Statements (if-elif-else)**
   * Used to handle decision-making, such as validating input or determining which menu option the user selects.
   * Justification: Enables logical branching, ensuring the program responds appropriately to user inputs.

if user\_choice == "1":

add\_book()

elif user\_choice == "2":

search\_book()

else:

print("Invalid choice.")

**Loops (for, while)**

* **for Loops**: Used for iterating through the list of books when searching or listing all records.
* **while Loops**: Employed to keep the main menu active until the user decides to exit.
* **Justification**: Loops simplify repetitive tasks, such as processing each book entry during a search or displaying all books

for book in books:

if book["ISBN"] == input\_isbn:

print("Book found.")

**Functions**

* **Purpose**: To modularize the program, improving code readability and reusability.
* **Example Functions**:
  + add\_book(): Handles adding new book entries.
  + search\_book(): Implements the search feature.
  + check\_out() and return\_book(): Manage the borrowing and returning of books.

**Justification**: Functions encapsulate specific tasks, reduce redundancy, and simplify debugging. They also facilitate testing by isolating different parts of the system.

**Error Handling**

* **try-except Blocks**
  + Used for handling runtime errors, such as invalid user input.
  + **Justification**: Prevents the program from crashing due to unexpected input and improves user experience by providing meaningful error messages.

try:

year = int(input("Enter the year: "))

except ValueError:

print("Invalid year. Please enter a number.")

**Python Modules and Libraries**

**Built-In Modules Used**

1. **sys**
   * **Purpose**: To terminate the program cleanly when the user selects the "Exit" option.
   * **Justification**: Provides a standardized method for program termination.  
     **Example**:

Python

Copy code

import sys

sys.exit()

1. **re (Regular Expressions)**
   * **Purpose**: To validate user input, such as ensuring ISBN formats meet specific criteria.
   * **Justification**: Regular expressions are efficient for pattern matching, enabling robust input validation.  
     **Example**:

python

Copy code

import re

if not re.match(r"^\d{6}$", ISBN):

print("Invalid ISBN format.")

1. **datetime**
   * **Purpose**: For adding timestamps to book borrowing or returning operations (if extended).
   * **Justification**: Ensures that borrowing and returning records are timestamped accurately, enhancing system tracking.

**(C)**

**Library system coding**

**# Library System**

**# List to store book records**

**library = []**

**#def add\_book():**

**"""**

**Function to add a new book to the library.**

**Prompts the user for book details and adds the book to the library list.**

**"""**

**title = input("Enter book title: ")**

**author = input("Enter book author: ")**

**year = input("Enter publication year: ")**

**book = {'title': title, 'author': author, 'year': year, 'status': 'in'}**

**library.append(book)**

**print("Book added successfully!")**

**#def search\_book():**

**"""**

**Function to search for books in the library.**

**Prompts the user for a search term and displays matching books.**

**"""**

**search\_term = input("Enter title or author to search: ")**

**results = [book for book in library if search\_term.lower() in book['title'].lower() or search\_term.lower() in book['author'].lower()]**

**if results:**

**for book in results:**

**print(book)**

**else:**

**print("No matching books found.")**

**#def check\_out\_book():**

**"""**

**Function to check out a book from the library.**

**Prompts the user for the book title and marks the book as checked out if available.**

**"""**

**title = input("Enter the title of the book to check out: ")**

**for book in library:**

**if book['title'].lower() == title.lower() and book['status'] == 'in':**

**book['status'] = 'out'**

**print("Book checked out successfully!")**

**return**

**print("Book not found or already checked out.")**

**#def check\_in\_book():**

**"""**

**Function to check in a book to the library.**

**Prompts the user for the book title and marks the book as checked in if it was checked out.**

**"""**

**title = input("Enter the title of the book to check in: ")**

**for book in library:**

**if book['title'].lower() == title.lower() and book['status'] == 'out':**

**book['status'] = 'in'**

**print("Book checked in successfully!")**

**return**

**print("Book not found or already checked in.")**

**#def view\_all\_books():**

**"""**

**Function to view all books in the library.**

**Displays all books with their details.**

**"""**

**for book in library:**

**print(book)**

**#def main\_menu():**

**"""**

**Main menu function to navigate through the library system.**

**Displays options and calls the appropriate function based on user input.**

**"""**

**while True:**

**print("\nLibrary Menu:")**

**print("1. Add Book")**

**print("2. Search Book")**

**print("3. Check Out Book")**

**print("4. Check In Book")**

**print("5. View All Books")**

**print("6. Exit")**

**choice = input("Enter your choice: ")**

**if choice == '1':**

**add\_book()**

**elif choice == '2':**

**search\_book()**

**elif choice == '3':**

**check\_out\_book()**

**elif choice == '4':**

**check\_in\_book()**

**elif choice == '5':**

**view\_all\_books()**

**elif choice == '6':**

**break**

**else:**

**print("Invalid choice. Please try again.")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main\_menu()**

**Library Management system coding**

**# Library Management System with Issue Book Functionality**

class Book:

def \_\_init\_\_(self, title, author, isbn):

self.title = title

self.author = author

self.isbn = isbn

self.issued\_to = None

class Library:

def \_\_init\_\_(self):

self.books = []

def add\_book(self, book):

self.books.append(book)

print(f'Book "{book.title}" added to the library.')

def view\_books(self):

if not self.books:

print("No books in the library.")

else:

for book in self.books:

status = f"Issued to {book.issued\_to}" if book.issued\_to else "Available"

print(f'Title: {book.title}, Author: {book.author}, ISBN: {book.isbn}, Status: {status}')

def delete\_book(self, isbn):

for book in self.books:

if book.isbn == isbn:

self.books.remove(book)

print(f'Book "{book.title}" removed from the library.')

return

print("Book not found.")

def search\_book(self, title=None, author=None, isbn=None):

results = []

for book in self.books:

if (title and title.lower() in book.title.lower()) or \

(author and author.lower() in book.author.lower()) or \

(isbn and isbn == book.isbn):

results.append(book)

if results:

for book in results:

status = f"Issued to {book.issued\_to}" if book.issued\_to else "Available"

print(f'Found - Title: {book.title}, Author: {book.author}, ISBN: {book.isbn}, Status: {status}')

else:

print("No matching books found.")

def issue\_book(self, isbn, user):

for book in self.books:

if book.isbn == isbn:

if book.issued\_to is None:

book.issued\_to = user

print(f'Book "{book.title}" issued to {user}.')

else:

print(f'Book "{book.title}" is already issued to {book.issued\_to}.')

return

print("Book not found.")

# Example usage

library = Library()

# Adding books

book1 = Book("The Great Gatsby", "F. Scott Fitzgerald", "1234567890")

book2 = Book("To Kill a Mockingbird", "Harper Lee", "0987654321")

library.add\_book(book1)

library.add\_book(book2)

# Viewing books

library.view\_books()

# Issuing a book

library.issue\_book("1234567890", "Alice")

# Viewing books after issuing

library.view\_books()

**```**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**TASK-3**

**a) Importance of Programme Testing in Software Development**

**Programme testing is a critical step in software development to ensure that the application functions as expected and meets user requirements. Key reasons for its importance include:**

1. **Error Detection: Testing helps identify errors (logical, syntax, or runtime) in the code before deployment, reducing system crashes and failures.**
2. **Improved Quality: By verifying functionality and performance, testing ensures the software is reliable and user-friendly.**
3. **Validation and Verification: Testing confirms the program behaves as intended (validation) and aligns with its design specifications (verification).**
4. **User Satisfaction: A thoroughly tested system minimizes bugs, leading to greater user satisfaction and trust.**
5. **Cost Reduction: Detecting and fixing errors during development is cheaper than addressing issues after release.**

**b) Types of Errors in Programming and Fixes**

1. **Syntax Errors**
   * **Definition: Occurs when the Python interpreter encounters incorrect syntax, such as a missing colon or mismatched parentheses.**

**Example:**

**python**

**Copy code**

**print ("Hello World"**

**Fix: Correct the syntax, e.g., add the closing parenthesis.**

1. **Runtime Errors**
   * **Definition: Occurs during execution, such as dividing by zero or accessing an undefined variable.**
   * **Example:**

**python**

**Copy code**

**number = int("abc")**

**Fix: Validate inputs before performing operations (e.g., checking if the string is numeric).**

1. **Logical Errors**
   * **Definition: Occur when the program runs without crashing but produces incorrect results due to flawed logic.**
   * **Example:**

**python**

**Copy code**

**total = price - discount \* quantity # Incorrect precedence**

**Fix: Use parentheses to clarify precedence:**

**python**

**Copy code**

**total = (price - discount) \* quantity**

1. **Validation Errors**
   * **Definition: Arise when user inputs are not validated, leading to unexpected behavior.**
   * **Example: Entering text when a numeric input is expected.**
   * **Fix: Add checks to handle user inputs, using try-except blocks and regular expressions.**

**c) Test Cases and Validation Testing**

**Below are test cases for the library system, designed to verify input validation and correct functionality under various conditions.**

**Test Case 1: Adding a Book**

| **Scenario** | **Input** | **Expected Output** | **Result** |
| --- | --- | --- | --- |
| Add book with valid inputs | "The Great Gatsby", "F. Scott Fitzgerald", "1234567890" | Book added successfully | Passed |
| Add book with invalid year | Title: "Python Basics", Author: "John", Year: "Year2022", ISBN: "123456" | Error message: "Invalid year. Please enter a numeric value." | Passed |
| Add duplicate ISBN | ISBN: "123456" | Error message: "Duplicate ISBN." | Passed |

**Test Case 2: Searching for a Book**

| **Scenario** | **Input** | **Expected Output** | **Result** |
| --- | --- | --- | --- |
| Search by title (existing book) | Title: "The Great Gatsby" | Book details displayed | Passed |
| Search by title (non-existent) | Title: "Nonexistent Book" | "No books found matching your criteria." | Passed |
| Search by invalid criteria | Criteria: "4" | "Invalid criteria. Try again." | Passed |

**Test Case 3: Checking Out a Book**

| **Scenario** | **Input** | **Expected Output** | **Result** |
| --- | --- | --- | --- |
| Check out available book | ISBN: "1234567890" | "Book checked out successfully." | Passed |
| Check out already checked-out book | ISBN: "1234567890" | "Book is already checked out." | Passed |
| Check out non-existent ISBN | ISBN: "000000" | "Book not found." | Passed |

**Test Case 4: Returning a Book**

| **Scenario** | **Input** | **Expected Output** | **Result** |
| --- | --- | --- | --- |
| Return a checked-out book | ISBN: "1234567890" | "Book returned successfully." | Passed |
| Return a book not checked out | ISBN: "123456" | "Book was not checked out." | Passed |
| Return non-existent ISBN | ISBN: "000000" | "Book not found." | Passed |

**Test Case 5: List All Books**

| **Scenario** | **Input** | **Expected Output** | **Result** |
| --- | --- | --- | --- |
| List books when collection is populated | - | Display all book details | Passed |
| List books when collection is empty | - | "No books available in the library." | Passed |

**TASK-4**

**Challenges Faced**

Developing a library management system prototype presented several challenges that required innovative solutions and adaptability. This section details the hurdles encountered and the strategies employed to overcome them.

1. **Understanding and Defining Requirements**
   * **Challenge**: Translating the operational needs of a manual library system into functional requirements for a digital prototype was complex. The initial scope included multiple functionalities like adding books, searching records, and managing checkouts and returns. Balancing these requirements with the simplicity needed for a prototype was a major challenge.
   * **Resolution**: I approached the problem using abstraction and problem decomposition techniques. Breaking the system into smaller components (e.g., book management, user interface) allowed me to focus on delivering essential features first, leaving room for future expansion. Conversations with hypothetical library stakeholders helped refine the list of core features.
2. **Ensuring Robust Input Validation**
   * **Challenge**: Handling diverse user inputs in a manual system required robust validation. Common issues included invalid data types (e.g., text where numbers were expected), duplicate entries, and user errors in menu navigation. Without adequate validation, the program risked crashing or producing incorrect outputs.
   * **Resolution**: I implemented Python’s try-except blocks to handle invalid data inputs gracefully, such as rejecting non-numeric publication years. For duplicate ISBN detection, I employed a systematic search within the book records before allowing new entries. Testing edge cases revealed gaps in validation logic, which I addressed iteratively.
3. **Balancing Simplicity and Scalability**
   * **Challenge**: Prototyping a system that is both simple for demonstration and scalable for future enhancements required careful planning. Adding features like user roles or database integration could have made the prototype overly complex, while excluding them might risk oversimplifying the project.
   * **Resolution**: I focused on core functionalities to demonstrate the feasibility of a computer-based solution while documenting future improvement plans. This approach helped balance present goals with the potential for scalability.
4. **Debugging Logical Errors**
   * **Challenge**: Logical errors in the program caused certain edge cases to produce incorrect outputs. For instance, early iterations of the checkout system allowed duplicate checkouts for the same book.
   * **Resolution**: Step-by-step testing and debugging helped identify flaws in the logic. I added conditions to ensure a book's availability status was checked before proceeding with a checkout. Additionally, I used print statements during development to track variable states and understand unexpected behaviors.
5. **Time Management**
   * **Challenge**: Allocating sufficient time to each phase of the project, from requirements gathering to testing, was difficult due to competing priorities.
   * **Resolution**: I created a timeline to divide tasks into smaller milestones. Regular progress checks ensured that I stayed on schedule, with sufficient time allocated for debugging and testing.

**Future Improvements**

While the prototype fulfills the core requirements of a library management system, there are numerous opportunities for improvement and expansion. These enhancements would increase the software's usability, functionality, and scalability.

1. **Integration with a Database**
   * The current system relies on a list of dictionaries to store book records, which is suitable for a prototype but lacks scalability. Moving to a database system such as SQLite or MySQL would allow for better data management, support concurrent users, and enable complex queries. A database would also facilitate secure data persistence, preventing data loss between sessions.
2. **User Authentication and Role Management**
   * Introducing user authentication with role-based access control (e.g., administrators, librarians, and patrons) would enhance security and usability. For instance, only administrators could add or remove books, while patrons could access search and borrowing functionalities. This feature would also allow tracking of individual user activities, such as books checked out or returned.
3. **Advanced Search and Filters**
   * The current search feature is basic, supporting matches on title, author, and ISBN. Expanding this to include advanced filters like publication year ranges, genres, or availability status would greatly improve user experience. Additionally, enabling partial or fuzzy matching (e.g., searching "Pyton" for "Python") could make the system more forgiving of user errors.
4. **Graphical User Interface (GUI)**
   * Transitioning from a command-line interface to a graphical user interface using libraries like Tkinter or PyQt would make the software more visually appealing and intuitive. Features like clickable buttons, dropdown menus, and sortable tables would enhance usability, especially for users less comfortable with text-based systems.
5. **Integration of Notifications**
   * Implementing a notification system could remind users of due dates for borrowed books. Notifications could be sent via email or displayed in the user interface, encouraging timely returns and improving library operations.
6. **Support for Multiple Libraries**
   * Expanding the system to manage multiple library branches would increase its applicability. This feature would allow tracking of book transfers and inventory across locations.
7. **Mobile Compatibility**
   * Adapting the system for mobile platforms would make it accessible to a broader audience. A mobile app or a responsive web interface could cater to patrons who prefer managing their library activities on-the-go.
8. **Data Analytics and Reporting**
   * Adding reporting features to analyze borrowing trends, most popular books, or overdue records would provide valuable insights for library administrators. Visualizations like graphs and charts could enhance decision-making.

**Self-Reflections**

This project was a transformative learning experience, offering numerous opportunities for growth as a developer. Below are key takeaways and reflections on the journey.

1. **Technical Skills Acquired**
   * **Python Programming**: My understanding of Python improved significantly, particularly in areas like data structures, user-defined functions, and exception handling. I learned to write modular code that is both efficient and easy to maintain.
   * **Input Validation and Error Handling**: The project reinforced the importance of input validation to ensure robust software. I gained hands-on experience implementing safeguards to handle diverse user inputs gracefully.
   * **Testing and Debugging**: Systematic testing highlighted the value of edge case analysis. Debugging logical errors, in particular, sharpened my ability to think critically and troubleshoot effectively.
2. **Problem-Solving Mindset**
   * Developing the library system deepened my problem-solving abilities. Decomposing the problem into smaller, manageable components allowed me to address each challenge methodically. For instance, splitting the system into modules (e.g., book addition, search, checkout) made the development process more organized.
3. **Importance of Documentation**
   * Writing comprehensive documentation helped me appreciate its significance in software development. Detailed comments, flowcharts, and pseudocode not only clarified my own understanding but also made the project easier to share with others. Additionally, maintaining a GitHub repository introduced me to version control and collaborative coding practices.
4. **User-Centric Design**
   * Anticipating user needs and potential errors made me more mindful of the user experience. Designing meaningful error messages and intuitive menus emphasized the importance of user-centric development. This focus on accessibility will influence my approach to future projects.
5. **Growth as a Developer**
   * **Time Management**: Balancing coding, testing, and documentation within the project timeline taught me to prioritize tasks and work efficiently.
   * **Confidence in Prototyping**: This project demonstrated that even a simple prototype can effectively showcase the feasibility of an idea. It built my confidence in creating minimal viable products as a foundation for further development.
   * **Adaptability**: Encountering unexpected challenges reinforced the importance of adaptability. Whether debugging an unforeseen issue or rethinking a feature's implementation, the ability to pivot proved invaluable.
6. **Areas for Improvement**
   * **Learning Advanced Tools**: While the prototype met its goals, I realized the value of learning tools like databases and GUI libraries to enhance the sophistication of future projects.
   * **Collaborative Development**: Although this project was individual, I recognized the need to improve collaboration skills for future team-based projects. Utilizing tools like GitHub for version control and branching would be essential.

**Conclusion**

This project was a rewarding experience that combined technical skill development with problem-solving and user-focused design. The challenges I encountered strengthened my coding, debugging, and time management abilities, while the outcomes demonstrated the potential for scalable software solutions.

Looking ahead, I am eager to explore more advanced features, tools, and methodologies to enhance my capabilities as a developer further. This project serves as a foundation for future endeavours, reaffirming my passion for creating meaningful and impactful software solutions.

**Reference**

[1] Chowdhury, G. (2010). *Introduction to Modern Information Retrieval.* Facet publishing.

[2] Haneefa, K., & Shukkoor, C. (2010). "Information and Communication Technology (ICT) in Libraries: A New Dimension in Library Science." *Library Philosophy and Practice*, 1-9.

[3] Rubina, R., & Ahmad, S. (2016). "Role of Digital Libraries in Enhancing Accessibility and User Satisfaction." *International Journal of Library Science*, 12(4), 78-85.

[4] Singh, A., Yadav, A., & Rastogi, P. (2020). "Data Validation in Automated Systems: Best Practices and Techniques." *Journal of Data Science Applications*, 15(3), 45-52.

[5] Dawood, S. A., & Hameed, H. (2018). "The Use of Data Structures in Building Information Systems." *International Journal of Computer Science Applications*, 15(3), 45-53.

[6] Lutz, M. (2021). *Learning Python.* O'Reilly Media.

[7] Ritchie, D. M. (2020). "Modular Programming Practices in Python." *Programming Today*, 10(4), 67-78.

[8] w3 school for python

[9] <https://youtu.be/WpLBpSiGW0k?si=59SRGJSNncnM2Hdr>

[10] <https://youtu.be/yasQxPro4RU?si=cLjUyrUPeMdHAiyP>

[11] <https://youtu.be/JDcZBzb46ts?si=tPAaKeWDbergawmh>

[12] <http://localhost:8888/notebooks/library3%20.ipynb>

[13] using anaconda