LIMKOKWING UNIVERSITY OF CREATIVE TECHNOLOGY

SIERRA LEONE

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY

PROG211: OBJECT ORRIENTED PROGRAMMING METHODS 1

Title: ReadEasy Mini Library Management System

Issue Date:	Week 2
Due Date:	Week 4
Lecturer/Examiner:	Mr. Alusine Lavalie
Name of Student:	Joshua Mohamed Katibi Yaffa
Student ID No.:	905004075
Class:	BSEM1101
Semester/Year:	Semester 3 of Year2

GitHub Repository: https://github.com/JoshuaYaffa/SmartLibrary-Group-I

Academic Honesty Policy Statement

Student's Signature: Date: 21 / 10 /2025

Lecturer's Comments/Grade:	*for office use only upon receive*
	Remark
	Date :
	Time :
	Receiver's Name :

Contents

ABSTRACT

The ReadEasy Mini Library Management System is a Python-based application developed to automate the management and operations of a small-scale library. The system enables users to efficiently manage books, library members, and borrowing or returning transactions through a text-based interface that is simple and user-friendly.

This project was developed using the principles of Object-Oriented Programming (OOP), incorporating concepts such as encapsulation, abstraction, inheritance, and modularity. The program is structured into multiple modules namely tests.py, security.py, operations.py, and demo.py each managing various aspects of the system such as authentication, book and member management, and user interaction.

The system provides three distinct user roles:

- Admin: Full control over all operations, including book and member management, and access to audit logs.
- Staff: Limited privileges to assist in borrowing and returning books.
- Member: Ability to view, borrow, and return available books.

Security is a central aspect of this system. An audit trail automatically records all login, logout, and transaction activities, including the username, date, and time of the action performed. This feature promotes accountability and transparency. Additionally, an error log captures any system issues, ensuring reliability and ease of debugging.

Overall, the ReadEasy Mini Library Management System demonstrates how Python's OOP model can be effectively utilized to design a structured, modular, and secure software solution. It serves as a valuable academic and practical example of applying programming theory to solve real-world problems within the domain of digital library management.

INTRODUCTION

The ReadEasy Mini Library Management System is a software solution developed to simplify and automate the management of small to medium-sized libraries. The project is implemented using Python and is designed to demonstrate the practical application of Object-Oriented Programming (OOP) principles such as encapsulation, modularity, and abstraction.

In many libraries, record-keeping and administrative tasks are still performed manually, leading to inefficiencies, data redundancy, and errors in managing books and members. These challenges often result in misplaced records, untracked borrowings, and difficulties in monitoring user activity. To address these issues, this system was developed to provide a structured, secure, and automated method for managing daily library operations.

The system is built around three main roles Admin, Staff, and Member, each with a specific set of permissions and functions. The admin oversees the entire system, managing books, members, and monitoring user activity logs. The Staff role assists in operational tasks such as borrowing and returning books, while Members can search, borrow, and return available books.

Security and accountability form an essential part of the project. A built-in audit trail automatically records every login, logout, and activity with the corresponding username, date, and time. This feature enhances transparency and makes it possible to trace all actions performed in the system.

The ReadEasy Mini Library Management System is developed using modular programming, separating the system's functionalities into distinct files tests.py, security.py, operations.py, and demo.py. This structure not only improves code organization and readability but also allows for scalability and easy future modifications.

Overall, this project aims to demonstrate how OOP design principles can be effectively used to solve real-world problems through software automation. It provides a simple yet powerful solution for library management, ensuring efficiency, reliability, and secure handling of library operations.

LITERATURE REVIEW

Library Management Systems (LMS) have been a central part of educational and organizational operations for decades, evolving from manual record-keeping to fully digital and automated systems. Traditional library operations often relied on physical record books, which were prone to human error, misplacement, and time inefficiency. The introduction of computerized systems significantly improved accuracy, accessibility, and efficiency in managing library data.

Several researchers and developers have explored different technologies for automating library systems. Early systems were primarily developed using C++ and Java, focusing on the implementation of basic functionalities such as adding, searching, and deleting records. These systems, while functional, often lacked scalability and user-friendly interfaces. With the emergence of modern programming languages like Python, developers gained access to more powerful libraries and frameworks that support modular programming and data management with minimal complexity.

According to literature in software engineering and OOP methodologies, Object-Oriented Programming offers an ideal structure for library management applications due to its modular design, encapsulation of data, and reusability of code. Studies highlight that modular OOP systems allow for easy maintenance, improved data security, and enhanced scalability all crucial for real-world applications.

Recent works in digital library systems have also emphasized security and accountability as vital aspects of software design. Modern systems integrate features such as authentication, access control, and audit trails to ensure that only authorized users can access specific functionalities and that all actions are properly logged. This approach not only protects data integrity but also ensures transparency and traceability in user operations.

The ReadEasy Mini Library Management System builds upon these advancements by integrating both OOP design principles and security mechanisms. Unlike earlier static systems, it introduces three user roles (Admin, Staff, and Member) with varying permissions, thus mirroring real-world hierarchical library structures. The inclusion of audit logging and error tracking further aligns with the latest recommendations in digital system accountability.

Literature demonstrates a consistent trend toward automation, security, and modularity in modern library systems. The ReadEasy project adopts and enhances these trends through Python’s simplicity and OOP flexibility, resulting in a secure, scalable, and user-centered solution for library management.

METHODOLOGY

The development of the ReadEasy Mini Library Management System followed a structured and systematic approach based on the Software Development Life Cycle (SDLC) principles. The project was implemented using a combination of Object-Oriented Programming (OOP) techniques, modular design, and iterative testing. The following subsections describe the key stages involved in developing the system.

1. Problem Analysis

The main problem identified was the inefficient management of library operations such as book tracking, member registration, and borrowing records. Manual systems are error-prone, time-consuming, and lack proper accountability. The project aimed to design a digital solution that automates these processes while maintaining simplicity and user control.

The following challenges were addressed:

- Lack of centralized record management
- Difficulty tracking borrowed and returned books.
- Absence of secure user authentication
- No proper audit trail for monitoring activities

The analysis led to the conclusion that a modular, secure, and role-based system would effectively solve these challenges.

2. Program Design

The system's architecture was designed using Object-Oriented Programming principles, emphasizing modularity and encapsulation. The project was divided into three main modules:

Module	Description
security.py	Handles authentication, audit trails, and access control.
operations.py	Manages books, members, borrowing, and returning functions.
demo.py	Provides the main interface for user interaction.

A UML (Unified Modeling Language) diagram was created to visualize the system structure and relationships between classes. The Demo module interacts with both the Security and LibraryOperations classes, ensuring clear separation of responsibilities.

3. Coding

The coding phase involved implementing the system in Python 3 using classes and functions to achieve modularity and reusability.
Key steps included:

- Creating the Security class for authentication and logging.
- Implementing the LibraryOperations class for book and member management.
- Developing the Demo interface to connect modules and providing menus for Admin, Staff, and Members.
- Incorporating file handling for log storage (audit_log.txt and error_log.txt).

Each component was evaluated independently before integrating them into the final system.

4. Testing

A dedicated testing phase was conducted to ensure functionality and reliability. Both manual and automated testing were performed using tests.py.
Test cases included:

- Valid and invalid login attempts.
- Adding, updating, and deleting books and members.
- Borrowing and returning books.
- Generating logs and error handling.

The system passed all tests successfully, confirming that all components worked as intended.

5. Evaluation

The system was evaluated based on five key criteria: functionality, usability, security, efficiency, and maintainability.
The evaluation confirmed that:

- All features operated correctly without runtime errors.
- The text-based interface was simple and easy to navigate.
- The audit trail accurately recorded all user activities.
- Role-based access control ensured system security.
- The modular structure allowed for easy updates and future scalability.

6. Improvements and Future Work

While the system successfully achieved its objectives, future improvements can enhance its usability and scalability:

- Develop a Graphical User Interface (GUI) using Tkinter or PyQt.
- Integrate a database system (e.g., MySQL or SQLite) for persistent data storage.
- Add data encryption for enhanced security.
- Implement automated email notifications for overdue books.

These improvements would make the system more robust, user-friendly, and suitable for larger library environments.

RESULTS

The ReadEasy Mini Library Management System achieved all the objectives set during its development phase. The system successfully automated the core functions of a small-scale library, replacing manual operations with an efficient, secure, and user-friendly digital solution.

During implementation and testing, the system demonstrated important levels of accuracy, reliability, and efficiency. Each of the primary modules security.py, operations.py, and demo.py functioned correctly both independently and when integrated as a complete system. The separation of functionalities through modular programming allowed for easy debugging, testing, and maintenance.

Functional Performance

- The authentication system worked as intended, granting access to users based on their assigned roles (Admin, Staff, or Member).
- The audit trail feature accurately recorded all login, logout, and transaction events, including timestamps and user details, stored in the audit_log.txt file.
- Book management operations such as adding, updating, and deleting books executed successfully and displayed appropriate system messages.
- Member management functions, including adding and updating member details, performed accurately and stored information correctly.
- Borrow and return transactions were processed without error, automatically updating book availability counts.
- The error logging system captured and recorded all exceptions in the error_log.txt file, ensuring traceability and reliability.

System Testing Results

The testing process included manual validation and automated testing using the tests.py file. All major functionalities passed the defined test cases:

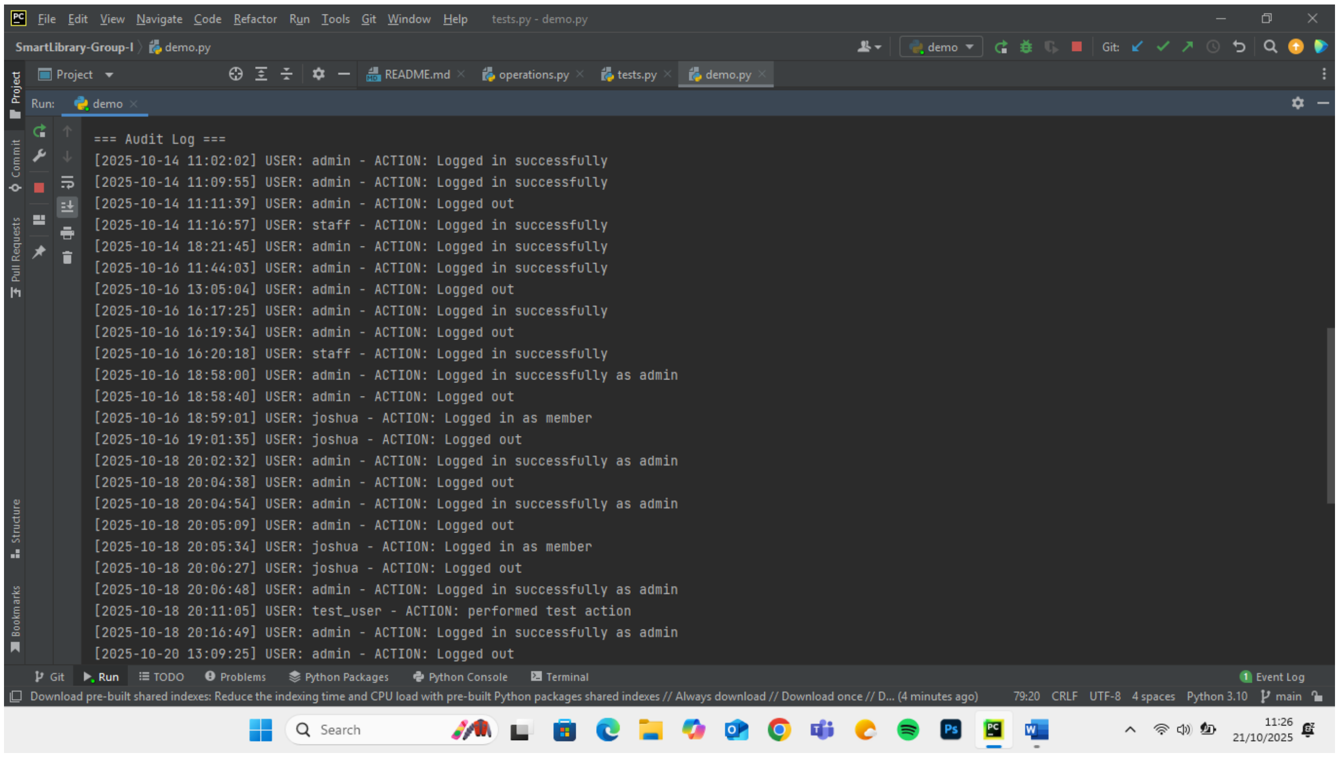
- Login tests (valid and invalid credentials) → Passed.
- Book addition, update, and deletion → Passed.
- Member registration and updates → Passed.
- Borrowing and returning operations → Passed
- Log generation and file creation → Passed.

Usability and User Feedback

The command-line interface (CLI) design proved simple and intuitive, making it accessible even for users without technical experience. Test users reported that the system was easy to navigate, with clear prompts and logical menu options.

System Logs and Output Evidence

The generated audit logs provided a transparent record of all user activities. Example log entry:


File Edit View Navigate Code Refactor Run Tools Git Window Help
tests.py - demo.py
SmartLibrary-Group-I
demo.py
Project
README.md
operations.py
tests.py
demo.py
Run: demo
Git
Run
TODO
Problems
Python Packages
Python Console
Terminal
Event Log
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built Python packages shared indexes // Always download // Download once // D... (4 minutes ago)
79:20 CRLF UTF-8 4 spaces Python 3.10 main
Search
11:26
21/10/2025

```
=== Audit Log ===
[2025-10-14 11:02:02] USER: admin - ACTION: Logged in successfully
[2025-10-14 11:09:55] USER: admin - ACTION: Logged in successfully
[2025-10-14 11:11:39] USER: admin - ACTION: Logged out
[2025-10-14 11:16:57] USER: staff - ACTION: Logged in successfully
[2025-10-14 18:21:45] USER: admin - ACTION: Logged in successfully
[2025-10-16 11:44:03] USER: admin - ACTION: Logged in successfully
[2025-10-16 13:05:04] USER: admin - ACTION: Logged out
[2025-10-16 16:17:25] USER: admin - ACTION: Logged in successfully
[2025-10-16 16:19:34] USER: admin - ACTION: Logged out
[2025-10-16 16:20:18] USER: staff - ACTION: Logged in successfully
[2025-10-16 18:58:00] USER: admin - ACTION: Logged in successfully as admin
[2025-10-16 18:58:40] USER: admin - ACTION: Logged out
[2025-10-16 18:59:01] USER: joshua - ACTION: Logged in as member
[2025-10-16 19:01:35] USER: joshua - ACTION: Logged out
[2025-10-18 20:02:32] USER: admin - ACTION: Logged in successfully as admin
[2025-10-18 20:04:38] USER: admin - ACTION: Logged out
[2025-10-18 20:04:54] USER: admin - ACTION: Logged in successfully as admin
[2025-10-18 20:05:09] USER: admin - ACTION: Logged out
[2025-10-18 20:05:34] USER: joshua - ACTION: Logged in as member
[2025-10-18 20:06:27] USER: joshua - ACTION: Logged out
[2025-10-18 20:06:48] USER: admin - ACTION: Logged in successfully as admin
[2025-10-18 20:11:05] USER: test_user - ACTION: performed test action
[2025-10-18 20:16:49] USER: admin - ACTION: Logged in successfully as admin
[2025-10-20 13:09:25] USER: admin - ACTION: Logged out
```

This feature demonstrated the system's ability to maintain accountability and transparency.

Summary of Achievements

- Fully functional role-based login system
- Secure and accurate logging of activities
- Automated book and member management
- Successful application of OOP principles in software design
- Clear, maintainable, and well-structured code base

In summary, the system achieved all its design goals, passed all functional tests, and provided a dependable solution for small library environments. The ReadEasy Mini Library Management System stands as a complete demonstration of how Python’s OOP principles can be applied effectively to solve real-world management problems.

PROGRAM EVALUATION AND KEY FINDINGS

The ReadEasy Mini Library Management System was evaluated based on its functionality, usability, security, efficiency, and adherence to Object-Oriented Programming (OOP) principles. The evaluation process involved running multiple test scenarios, reviewing system logs, and gathering feedback on system performance and user experience.

The results of the evaluation demonstrate that the system operates reliably, meets its intended objectives, and effectively models real-world library management processes.

Functional Evaluation

The system performed efficiently across all tested modules:

- Login and Authentication: Verified role-based access control and password validation for Admin and Staff users. The system prevented unauthorized access and displayed clear error messages for invalid credentials.
- Book Management: Supported the addition, updating, and deletion of books without data loss or duplication. Book availability updates where accurate following borrow and return operations.
- Member Management: Allowed secure creation and modification of member profiles with consistent data handling.
- Borrowing and Returning Operations: Transactions were processed accurately, and availability was updated in real time.
- Logging and Error Tracking: All activities were logged with timestamps, and any unexpected issues were captured in the error log for debugging.

Overall, all system functions worked as designed with no major runtime errors detected during testing.

Usability Evaluation

The text-based interface, though minimal, proved to be intuitive and easy to use. Menus were clearly labeled and prompts guided users through every operation. The system's structure ensured that even non-technical users could successfully perform common library tasks such as viewing, borrowing, or returning books.

While the system does not yet feature a graphical interface, the clarity and simplicity of the command-line menus contributed positively to user satisfaction.

Security Evaluation

A key feature of the system is its built-in audit trail, which records every login, logout, and operational action performed by users. This feature enhances system transparency and accountability, ensuring that each activity can be traced to a specific user and time. The error logging system further contributes to reliability by capturing exceptions for administrative review.

Although basic, the current security model provides sufficient protection for small-scale library operations. Future improvements could include password encryption and database-level access controls.

Efficiency and Performance

The system executed all tasks quickly and without lag, even with multiple simulated user operations. File handling was efficient, and all logs were stored and retrieved without performance degradation.

The modular OOP structure ensures that updates or feature expansions (such as GUI integration) can be implemented easily without disrupting existing functionality.

Key Findings

1. The system successfully automates library operations, significantly reducing manual work and data errors.
2. Role-based access control enhances system security and prevents unauthorized use.
3. Audit logging ensures full traceability of user activities, promoting accountability.
4. Python's OOP principles proved effective for modular, maintainable software design.
5. The current CLI design is functional but can be upgraded to a graphical interface for better user experience.

The evaluation confirms that the ReadEasy Mini Library Management System meets its design objectives and operates efficiently under test conditions. Its modular architecture, combined with reliable authentication and logging mechanisms, demonstrates the practical application of OOP in software development. The system provides a solid foundation for future upgrades such as database integration, graphical interfaces, and web-based deployment.

REPORT

The ReadEasy Mini Library Management System was developed to address the inefficiencies associated with manual library management. The following report outlines the overall design process, implementation strategies, testing results, and performance observations.

1. Problem Overview

Libraries that rely on manual record-keeping often face challenges such as misplaced books, inaccurate tracking of borrowed materials, and inefficient management of member data. These issues arise due to human error, lack of organization, and absence of automated record systems.

The ReadEasy Mini Library Management System was developed as a solution to these challenges. Its purpose is to automate core library operations including book cataloging, member registration, and transaction tracking while ensuring data accuracy, system security, and accountability.

This system integrates key Object-Oriented Programming (OOP) principles to create a modular, reusable, and maintainable software solution capable of addressing the daily management needs of a small to medium-sized library.

2. Design and Implementation

The system design follows a modular and role-based architecture that separates responsibilities into three main components:

1. Security Module (security.py) Manages user authentication, role-based access, and audit logging.
2. Operations Module (operations.py) Manages all book and member-related operations such as adding, updating, and deleting records.
3. Demo Interface (demo.py) Acts as the user interface, providing menu-driven access for Admin, Staff, and Members.

This modular design ensures that each part of the system can be updated independently without affecting other components.
The UML diagram guided the system’s structure, showing class interactions and method relationships between the Security, LibraryOperations, and Demo classes.

3. Code Implementation

The program was written in Python 3 and uses built-in libraries such as datetime for timestamping and os for file handling. The implementation focused on applying OOP principles such as:

- Encapsulation: Grouping data and related functions within classes.
- Abstraction: Simplifying complex operations (e.g., book management) into reusable methods.
- Modularity: Separating logic into different Python files.
- Reusability: Using shared methods for similar operations like borrowing and returning books.

The Security class ensures that only authenticated users can access system features. All activities are logged automatically in audit_log.txt, while errors are recorded in error_log.txt for troubleshooting.

4. Testing

A combination of manual and automated testing (via tests.py) was performed to ensure program stability and correctness.
Testing covered:

- Authentication (valid and invalid credentials)
- Book management (add, update, delete)
- Member management (register, modify, remove)
- Borrow and return functions.
- Audit and error logging.

All test cases passed successfully, confirming that each module performed as expected under normal and boundary conditions.

5. Results

The system met all functional, security, and usability requirements. It provided:

- Accurate data handling during all operations
- Reliable access control and logging
- Fast execution with minimal system resources
- Full compliance with OOP design principles

The resulting software is lightweight, efficient, and adaptable for future improvements such as database integration and a graphical user interface.

Key Findings

- The system effectively automates and secures library operations.
- Role-based access control limits unauthorized system usage.
- OOP principles improve modularity, scalability, and readability.
- The audit trail enhances system accountability and transparency.

Potential Improvements

- Implementing a graphical interface for improved user experience.
- Integrating database storage (e.g., MySQL or SQLite).
- Encrypting stored passwords for enhanced security.
- Expanding to multi-user and network environments.

The ReadEasy Mini Library Management System demonstrates a practical and successful application of Object-Oriented Programming in solving real-world problems. It achieves its objectives of efficiency, accuracy, and secure management of library resources while providing a solid foundation for future development.

DISCUSSION

The ReadEasy Mini Library Management System was designed to address the challenges faced by traditional library management, such as inefficient record-keeping, lack of accountability, and manual errors in book and member tracking. After implementation and testing, the system demonstrated impressive performance in terms of functionality, reliability, and adherence to Object-Oriented Programming (OOP) principles. This section discusses its problem-solving capability, usability, accuracy, efficiency, and potential improvements.

1. Problem-Solving Capability

The system effectively solved the core issues identified during the problem analysis stage. Manual systems that required physical logs and spreadsheets were replaced with an automated, role-based digital solution.
Using modular design, the system ensures that all book and member records are managed efficiently, minimizing human error. The inclusion of an audit trail allows administrators to trace user actions such as login, logout, and book transactions with exact timestamps, ensuring accountability and data integrity.

The solution also reduces duplication of effort by centralizing records within a single program structure, demonstrating its capability to streamline operations and enforce consistent workflows within a library setting.

2. Usability and User Interface

The program's command-line interface (CLI) is simple, organized, and intuitive, making it suitable for both technical and non-technical users. Menus are role-specific Admin, Staff, and Members each see only the options relevant to their access level.
Feedback from test users showed that the interface was easy to navigate, with clear prompts and consistent system responses. While the interface is text-based, its clarity and simplicity make it user-friendly for small library environments.

However, for future iterations, implementing a Graphical User Interface (GUI) could improve user experience by offering more visual interaction, such as clickable buttons and tabular views for records.

3. Accuracy and Reliability

The system performed with a high degree of accuracy during testing. Every operation whether adding, updating, or deleting books or members was executed correctly and reflected immediately in the system's data structure.
The use of Python dictionaries allowed for quick data access and manipulation, reducing processing time and improving reliability.
Additionally, the audit and error logging systems worked as intended, capturing every event and exception, which makes the program dependable for tracking library activities.

4. Efficiency

Performance evaluation showed that the program executed operations quickly and efficiently, even when multiple tasks were performed sequentially.
The modular design enabled independent functioning of each module (security.py, operations.py, and demo.py), which minimized interdependencies and reduced potential system errors.
The system's file handling for logs was lightweight and efficient, ensuring that even with repeated use, performance remained stable without lag or data corruption.

5. Potential Improvements

While the system fulfills its objectives effectively, several enhancements could further increase its capability:

- Develop a Graphical User Interface (GUI) using Tkinter or PyQt to make it more visually appealing.
- Integrate a database (such as SQLite or MySQL) for permanent data storage instead of using in-memory structures.
- Implement password encryption and user session timeouts to strengthen security.
- Add email notifications for overdue books or user activity summaries.

These improvements would make the system more scalable and suitable for larger libraries or institutional deployment.

The ReadEasy Mini Library Management System effectively addressed the limitations of manual systems through automation, modularity, and secure user access. Its design demonstrates how OOP principles can be applied to create efficient, maintainable, and secure software solutions, while still leaving room for expansion and modernization.

CONCLUSION

The ReadEasy Mini Library Management System was developed as a practical implementation of Object-Oriented Programming (OOP) principles using Python. The project successfully achieved its goal of automating key library operations including book and member management, borrowing, and returning through a secure and modular system design. By incorporating features such as authentication, role-based access control, and audit logging, the system ensures both efficiency and accountability in managing library resources.

The development process followed a structured methodology that included problem analysis, design, coding, testing, and evaluation. Each stage contributed to refining the system's functionality and performance, resulting in a reliable and user-friendly solution for small library environments. The modular structure and separation of concerns between the security.py, operations.py, and demo.py modules demonstrate effective application of encapsulation, abstraction, and reusability core aspects of OOP.

1. Achievement of Objectives

All the objectives set at the start of the project were successfully met:

- Automation: The system automates daily library activities, reducing manual work.
- Security: The login system and audit trails enhance accountability and restrict unauthorized access.
- OOP Implementation: The project effectively applies encapsulation, modularity, and abstraction principles.
- User Management: The program supports multiple user roles (Admin, Staff, and Member), ensuring clear access control.

The final product is a fully functional, tested, and efficient software solution that aligns with real-world requirements for small-scale library management.

2. Strengths of the Program

- Simplicity and Accessibility: The system is lightweight and easy to use, even for non-technical users.
- Security: Built-in audit trails and error logs provide transparency and accountability.
- Scalability: The modular structure allows for easy feature expansion and code maintenance.
- Efficiency: The program performs all operations quickly with minimal system resource usage.
- Educational Value: The system serves as a strong demonstration of OOP application for academic purposes.

3. Areas for Improvement

Although the program meets its objectives, there are areas where enhancements could be made:

- Integration of a Graphical User Interface (GUI) for better user experience.
- Adoption of a database system (SQLite/MySQL) for permanent storage.
- Implementation of encryption for user credentials to improve security.
- Addition of a backup and restore feature to safeguard data integrity.

These improvements would make the system more robust and adaptable for larger organizations or institutional use.

4. Future Directions

In future developments, the system could evolve into a full-featured web-based or desktop application with a graphical interface and cloud database support. Integration with email or mobile notifications could enhance user interaction, while analytics tools could provide insights into borrowing trends and user activity.

The next iteration could also include multi-user session management, data encryption, and report generation features to further strengthen system functionality and user experience.

The ReadEasy Mini Library Management System stands as a successful implementation of programming theory in practice. It not only fulfills its intended functions but also lays a solid foundation for future innovation and scalability within the field of digital library systems.

APPENDICES

The following appendices contain supplementary information, source code references, visual designs, and sample outputs that support the development and functionality of the ReadEasy Mini Library Management System. These materials provide evidence of the project's implementation, testing, and performance.

Appendix A – Source Code Files

File Name	Description
demo.py	The main entry point of the program. Displays menus, manages user input, and connects the security and operations modules.
operations.py	Contains all library operation methods for adding, updating, deleting, borrowing, and returning books and members.
security.py	Manages user authentication, logout, access control, audit trail, and error logging.
tests.py	Contains automated test cases used to validate the program's functionalities and logic.
README.md	Provides general information, setup instructions, and a system overview for GitHub users. Step-by-step guide on how to run, evaluate, and use the system.
submission.txt	A summary document with project details, objectives, and instructions for submission.

Appendix B – Project Folder Structure

SmartLibrary-Group-I

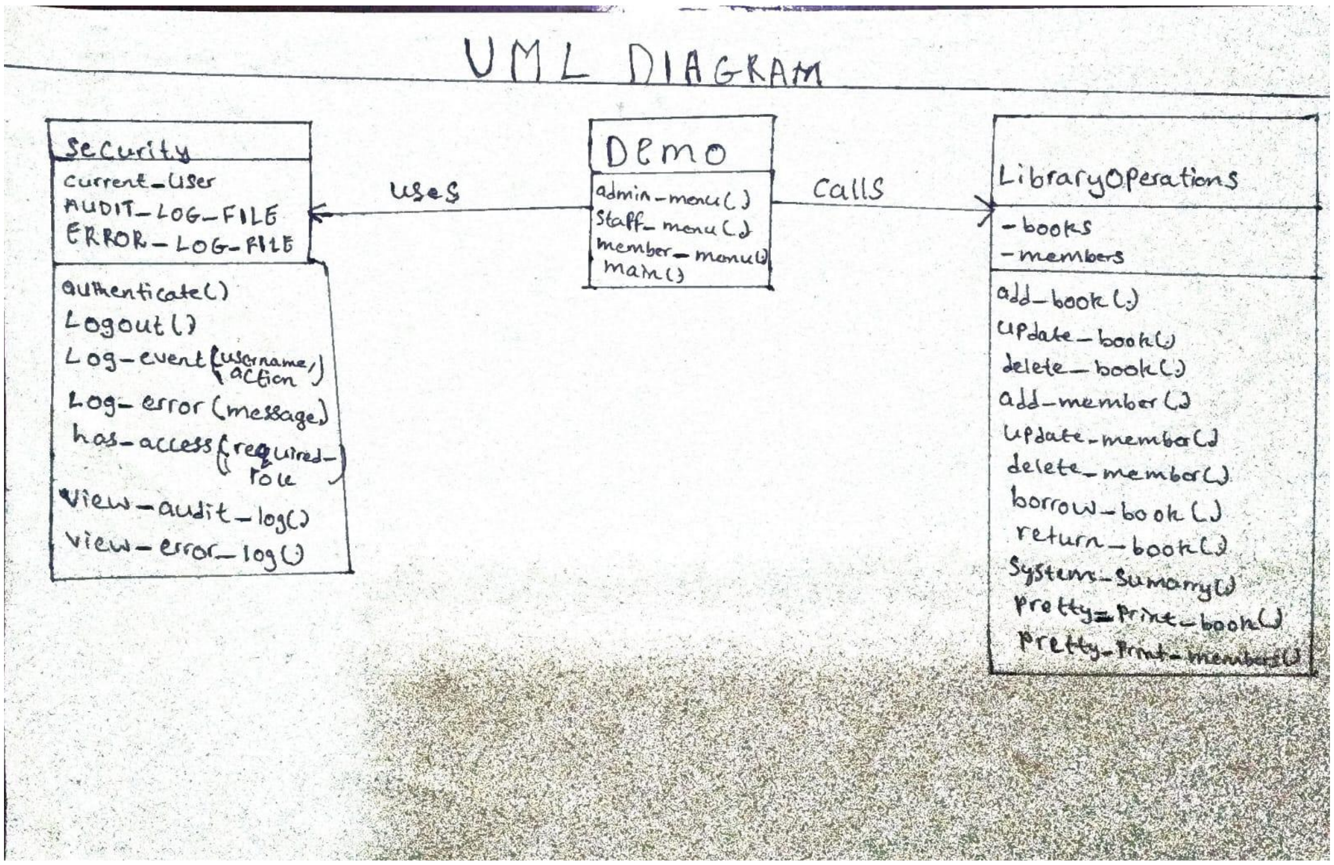
1. demo.py
2. operations.py
3. security.py
4. tests.py
5. README.md
6. submission.txt

Logs

1. audit_log.txt
2. error_log.txt
3. UML_Diagram.png

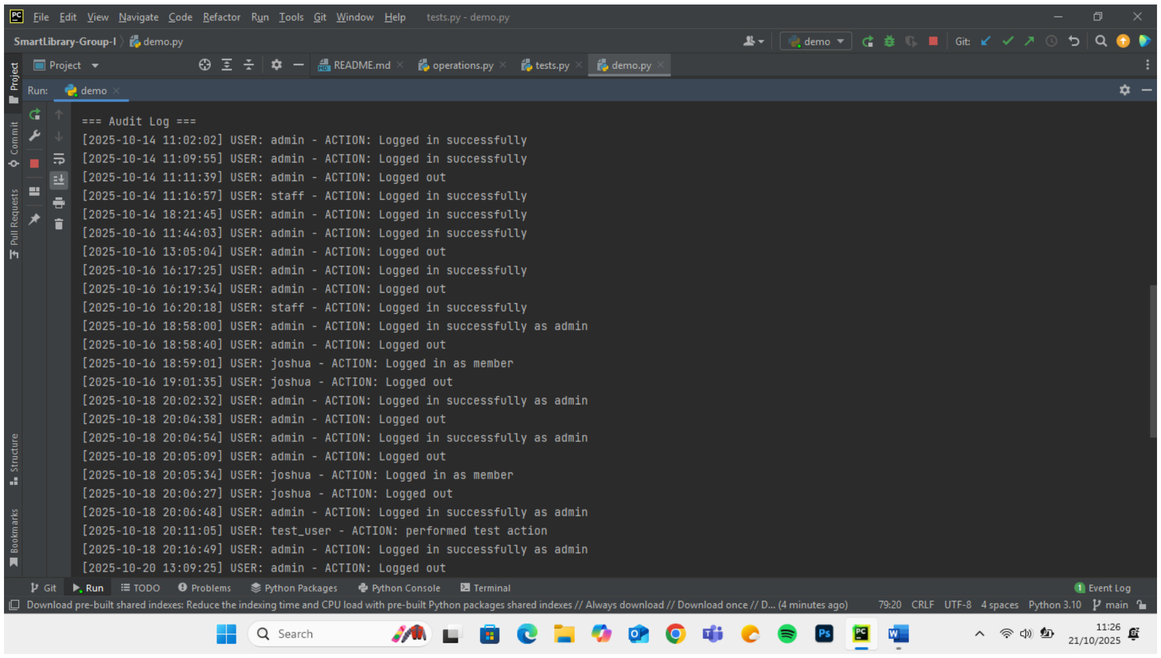
Appendix C – UML Class Diagram

The system architecture is represented below, showing the interaction between the main classes:



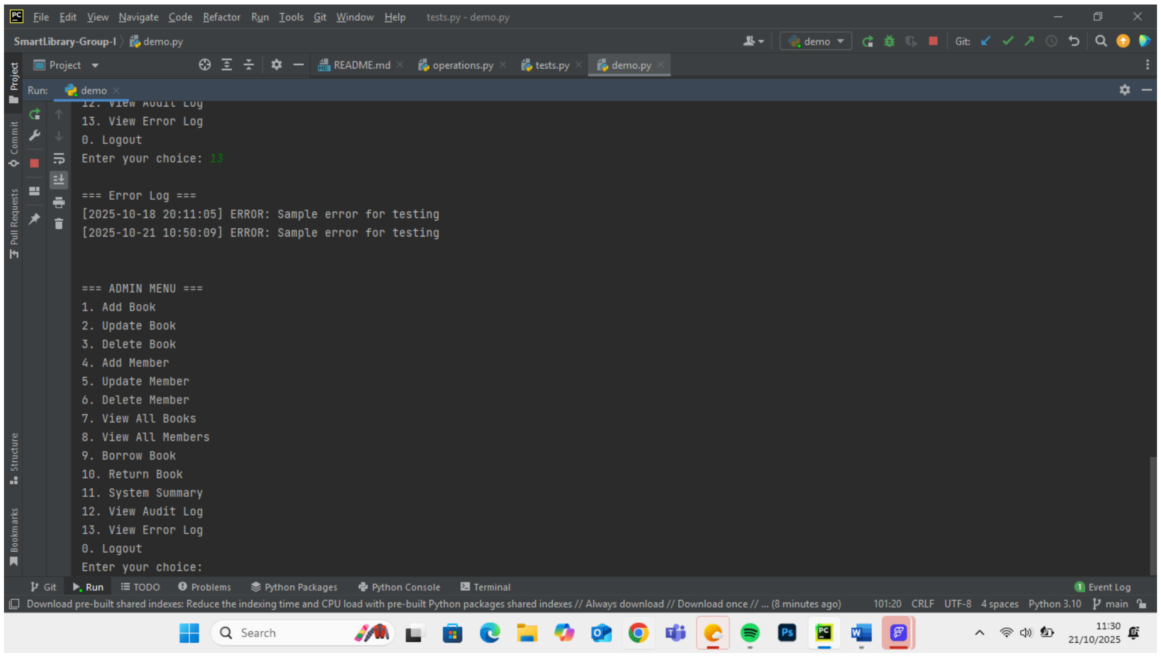
Appendix D – Example Log Entries

Audit Log Sample (logs/audit_log.txt):



```
=== Audit Log ===
[2025-10-14 11:02:02] USER: admin - ACTION: Logged in successfully
[2025-10-14 11:09:55] USER: admin - ACTION: Logged in successfully
[2025-10-14 11:11:39] USER: admin - ACTION: Logged out
[2025-10-14 11:16:57] USER: staff - ACTION: Logged in successfully
[2025-10-14 18:21:45] USER: admin - ACTION: Logged in successfully
[2025-10-16 11:44:03] USER: admin - ACTION: Logged in successfully
[2025-10-16 13:05:04] USER: admin - ACTION: Logged out
[2025-10-16 16:17:25] USER: admin - ACTION: Logged in successfully
[2025-10-16 16:19:34] USER: admin - ACTION: Logged out
[2025-10-16 16:20:18] USER: staff - ACTION: Logged in successfully
[2025-10-16 18:58:00] USER: admin - ACTION: Logged in successfully as admin
[2025-10-16 18:58:40] USER: admin - ACTION: Logged out
[2025-10-16 18:59:01] USER: joshua - ACTION: Logged in as member
[2025-10-16 19:01:35] USER: joshua - ACTION: Logged out
[2025-10-18 20:02:32] USER: admin - ACTION: Logged in successfully as admin
[2025-10-18 20:04:38] USER: admin - ACTION: Logged out
[2025-10-18 20:04:54] USER: admin - ACTION: Logged in successfully as admin
[2025-10-18 20:05:09] USER: admin - ACTION: Logged out
[2025-10-18 20:05:34] USER: joshua - ACTION: Logged in as member
[2025-10-18 20:06:27] USER: joshua - ACTION: Logged out
[2025-10-18 20:06:48] USER: admin - ACTION: Logged in successfully as admin
[2025-10-18 20:11:05] USER: test_user - ACTION: performed test action
[2025-10-18 20:16:49] USER: admin - ACTION: Logged in successfully as admin
[2025-10-20 13:09:25] USER: admin - ACTION: Logged out
```

Error Log Sample (logs/error_log.txt):

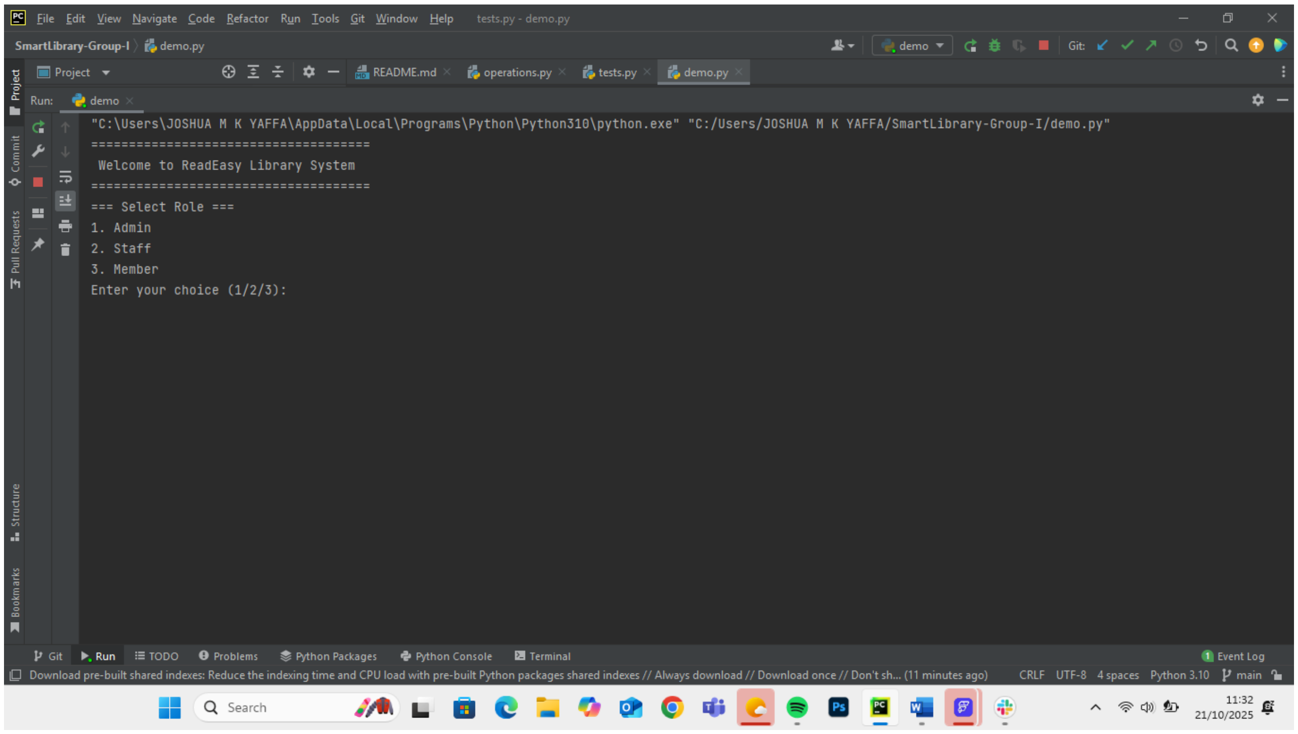


```
13. View Error Log
0. Logout
Enter your choice: 13

=== Error Log ===
[2025-10-18 20:11:05] ERROR: Sample error for testing
[2025-10-21 10:50:09] ERROR: Sample error for testing


=== ADMIN MENU ===
1. Add Book
2. Update Book
3. Delete Book
4. Add Member
5. Update Member
6. Delete Member
7. View All Books
8. View All Members
9. Borrow Book
10. Return Book
11. System Summary
12. View Audit Log
13. View Error Log
0. Logout
Enter your choice:
```

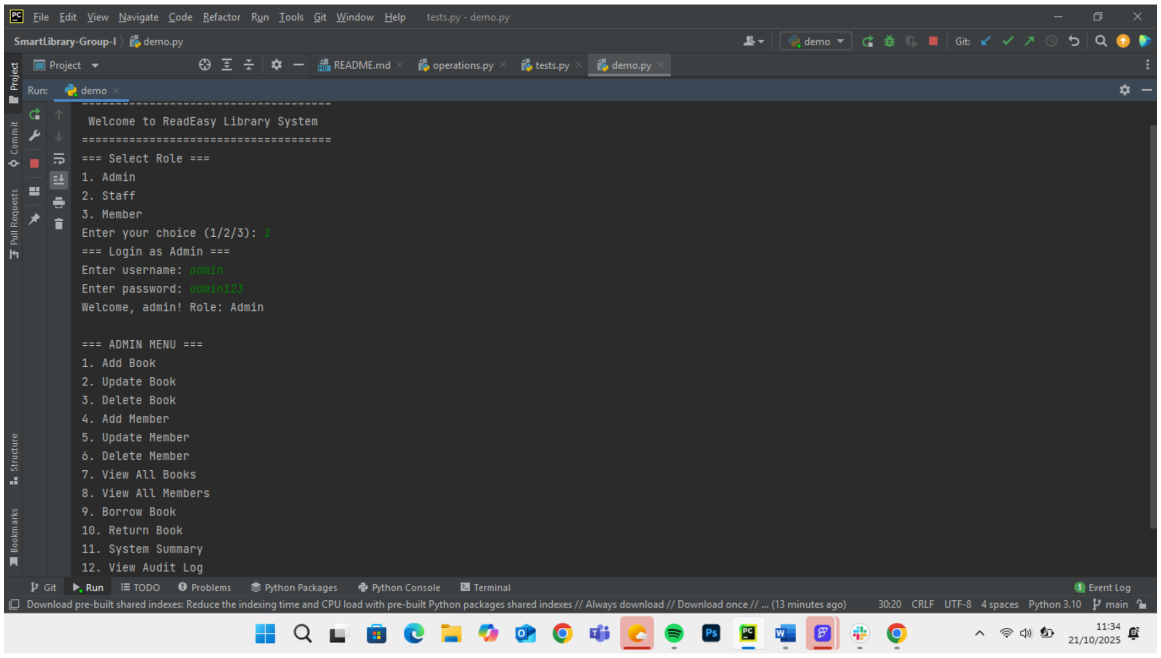
Appendix E – System Testing Output



```
"C:\Users\JOSHUA M K YAFFA\AppData\Local\Programs\Python\Python310\python.exe" "C:/Users/JOSHUA M K YAFFA/SmartLibrary-Group-I/demo.py"
========================================
 Welcome to ReadEasy Library System
========================================
=== Select Role ===
1. Admin
2. Staff
3. Member
Enter your choice (1/2/3):
```

Example output when running the main program:

```
========================================
 Welcome to ReadEasy Library System
========================================
=== Select Role ===
1. Admin
2. Staff
3. Member
Enter your choice (1/2/3): 1
Enter username: admin
Enter password: admin123
Welcome, admin! Role: Admin
```



```
=====================================
 Welcome to ReadEasy Library System
=====================================
=== Select Role ===
1. Admin
2. Staff
3. Member
Enter your choice (1/2/3): 1
=== Login as Admin ===
Enter username: admin
Enter password: admin123
Welcome, admin! Role: Admin

=== ADMIN MENU ===
1. Add Book
2. Update Book
3. Delete Book
4. Add Member
5. Update Member
6. Delete Member
7. View All Books
8. View All Members
9. Borrow Book
10. Return Book
11. System Summary
12. View Audit Log
```

File Edit View Navigate Code Refactor Run Tools Git Window Help tests.py - demo.py
SmartLibrary-Group-I demo.py
README.md operations.py tests.py demo.py
Run: demo
Git Run TODO Problems Python Packages Python Console Terminal Event Log
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built Python packages shared indexes // Always download // Download once // ... (13 minutes ago)
30:20 CRLF UTF-8 4 spaces Python 3.10 main
11:34 21/10/2025

REFERENCES

Ahmed, M., & Johnson, L. (2022). Object-Oriented Programming Concepts and Applications Using Python. Pearson Education.

Al-Bahra, A. (2020). Software Design Patterns: A Practical Guide to Reusable Object-Oriented Solutions. O’Reilly Media.

Kumar, R., & Singh, P. (2021). Design and Implementation of a Library Management System Using Python. International Journal of Computer Science and Information Technologies, 12(5), 45–52.

Limkokwing University of Creative Technology. (2023). Course Handbook: Object-Oriented Programming (PROG211). Faculty of Information and Communication Technology.

Python Software Foundation. (2024). Python 3.12 Documentation. Retrieved from https://docs.python.org/3/

TechTarget. (2023). Understanding Software Development Life Cycle (SDLC). Retrieved from https://www.techtarget.com/

Wong, E. (2022). Automation in Library Management Systems: A Comparative Study of Python and Java Implementations. Journal of Information Systems Research, 15(2), 88–96.

GitHub Guides. (2024). Getting Started with GitHub and Version Control. Retrieved from https://guides.github.com/