

A systematic investigation into ChatGPT's role in software development with an application of a Library Management System using Java and MySQL

Zainab Zafari, Hasso Plattner Institute, zainab.zafari@student.hpi.de

Abstract:

ChatGPT, a powerful language model, has received a lot of attention in academia for its ability to help developers with software engineering tasks. While existing research highlights its capabilities, empirical evidence on its practical utility in real-world software projects remains scarce. This paper addresses this gap by systematically investigating ChatGPT's role in the development of a Library Management System using Java and MySQL. Employing a Generation, Evaluation, and Repair Framework across various phases of software development, we assess ChatGPT's ability to gather requirements, design system architecture, generate code, and even generate unit tests. Our findings reveal both strengths and limitations in ChatGPT's assistance throughout the development process. While it excels in some areas such as requirement gathering, code implementation, and generating simple test cases, it struggles with complex tasks like interface design and mocking unit tests and inconsistencies, inaccuracies, and limitations are evident, particularly in complex scenarios. This study contributes to understanding ChatGPT's practical applicability in real-world software development scenarios and provides insights for leveraging its strengths while addressing its limitations.

1. Introduction:

In recent years, the integration of artificial intelligence (AI) technologies into software development processes has shown great promise in enhancing productivity and efficiency. Among these AI tools, ChatGPT, a natural language processing model developed by OpenAI, has gained significant attention for its ability to generate human-like text based on input prompts. Academic discourse has highlighted the potential of ChatGPT across various phases of software development, from requirement gathering to code generation and unit testing. However, while some practical applications of ChatGPT have been explored in simplified scenarios, empirical evidence on its performance in complex or real-world projects remains limited.

This paper aims to address this gap by conducting a systematic investigation into ChatGPT's role in practical software development scenarios, with a specific focus on the development of a Library Management System using Java and MySQL. Our research is motivated by the need to bridge the gap between simple scenario discussions and real-world applications of ChatGPT, providing nuanced insights into its strengths and limitations in complex development contexts. By leveraging a Generation, Evaluation, and Repair Framework, we aim to assess ChatGPT's efficacy across different phases of the software development lifecycle, including requirement gathering, system design, database implementation, interface design, code generation, and unit test generation.

Furthermore, our study incorporates insights from existing literature and state-of-the-art research to contextualize our findings within the broader landscape of AI-assisted software development. While previous studies have explored ChatGPT's potential in simplified scenarios, empirical evaluations in complex, real-world projects are scarce. Therefore, our research seeks to fill this gap by providing a holistic assessment of ChatGPT's capabilities in the context of a multifaceted software development project.

To provide a comprehensive assessment of ChatGPT's capabilities, we examine its performance across multiple dimensions, including consistency, correctness, understandability, and code quality. Additionally, we explore advanced techniques such as mocking, spying, and stubbing in unit testing scenarios, evaluating ChatGPT's effectiveness in generating reliable test cases for real-world applications.

Through rigorous experimentation and evaluation, this research aims to contribute practical insights for developers and researchers alike, facilitating informed decision-making and driving advancements in AI-driven development methodologies in real-world settings.

2. Motivation:

The motivation behind this research stems from the recognition in academic discourse of ChatGPT's prowess in generating accurate and understandable code. While existing research highlights its potential to aid developers throughout various phases of software development, a notable gap exists in empirical evidence regarding its practical utility in real-world projects. In this study, we concentrate on the development of a Library Management System, aiming to determine if ChatGPT can effectively contribute to the design and implementation of such real-world software applications. By investigating its efficacy directly within the

context of this project, we hope to determine whether ChatGPT may be a useful tool in actual software engineering projects.

3. Glossary

1. **Software development:** refers to the process of designing, creating, testing, and maintaining software applications and systems. It involves a series of stages, methodologies, and practices aimed at producing high-quality software that meets the needs of users or clients[1].
2. **Requirements Gathering:** This phase involves gathering and documenting the functional and non-functional requirements of the software. It includes speaking with stakeholders, assessing user demands, and creating explicit specifications that developers can use to build the system. This includes understanding the needs of users, stakeholders, and technical constraints[2].
3. **Design:** In this phase, developers create a blueprint for the software based on the gathered requirements. This includes architectural design, system design, and detailed design of individual components[3]. It also includes specifying programming language, templates, platform to use, and application security measures. In most cases it includes development of a prototype model[4].
4. **UML Diagrams:** a Unified Modeling Language diagram is a way to visualize the design, code architecture and code implementation of the system within the realm of software engineering. It includes both structural and behavioral aspects, as well as a variety of diagrams that together provide a thorough understanding of the system's structural composition and functional behavior[5].
 - a. **Class diagram:** Class diagram is a structural UML diagram and a fundamental for Object Oriented solution. It depicts classes, attributes, functions. Class diagram is used by both software engineers and business managers to show the different connections involved within a process[5].
 - b. **Use-Case diagram:** Use case diagram is a graphical overview of actors "persons or users" and use-cases"functionality of system". It outlines the functionality and behavior of the system which helps developers to analyze the relationships between use-cases and persons involved in the system[5].
 - c. **State Machine Diagram:** UML state machine diagrams, also known as state chart diagrams, depict the behaviors of various objects inside a system. You can see how elements behave differently depending on the status of the program developed[5].
5. **Implementation:** The design is translated into actual code during the implementation phase. Developers write, compile, and test the code according to the design specifications and requirements[1].
6. **Unit Testing:** Unit testing is a software testing technique where individual

units or components of a software application are tested in isolation to ensure that they function correctly as designed. In unit testing, each unit is tested independently of the rest of the application's code, typically at the function or method level.

7. **Mock Testing:** in unit testing refers to the practice of creating fake objects that simulate the behavior of real objects in a controlled manner. These fake objects, known as mocks, are used to isolate the code being tested and ensure that the unit under test is functioning correctly in isolation, without dependencies on external systems or resources[6].
8. Mocking is particularly useful when testing code that relies on external dependencies such as databases, web services, or other components that may be difficult to set up or control in a testing environment. By using mocks, developers can simulate the behavior of these dependencies and focus solely on testing the specific functionality of the unit being tested.
9. **PowerMockito:** PowerMockito is an extension of the Mockito framework in java programming. It facilitates working with java reflection in a more convenient way with its potential for mocking static methods, final methods, and constructors which are not possible by mocking framework[7].

4. State of The Art:

4.1. Requirement Gathering

Investigating ChatGPT's Potential to Assist in Requirements Elicitation Processes:

This research paper explores the potential of ChatGPT in requirement gathering and comparing the quality of output through ChatGPT with requirements formulated by five human experts. Evaluating ChatGPT's potential in seven quality attributes of abstraction, atomicity, consistency, correctness, unambiguity, feasibility, and understandability[8].

Advancing Requirements Engineering through Generative AI: Assessing the Role of LLMs:

This research paper explores the possibility of LLMs in requirement engineering processes with intention for enhancement of effectiveness and accuracy of requirements-related tasks. It proposes key directions and conducts a SWOT analysis for leveraging LLMs in requirement engineering with focus on the ability of LLMs for requirement elicitation, analysis specification, and validation[9].

4.2. Design

UML diagrams with ChatGPT

In this paper ChatGPT has been used to generate PlantUML code for uml diagram generation. ChatGPT has been used in generating UML diagrams of Use case diagram and sequence diagrams[10].

Using ChatGPT to Develop Software Systems: Alert to Software Engineers?

In this research paper, ChatGPT is used to create a class diagram using project description while evaluating ChatGPT's ability in identifying domain concepts(classes and attributes) as well as establishing association and multiplicity[11].

4.3. Code Generation/implementation

Self-collaboration Code Generation via ChatGPT:

This paper examines ChatGPT'S ability in generating codes while collaborating ChatGPT for generating better codes, both in simple tasks and complex tasks. At the same time evaluating quality and quantity of codes generated by ChatGPT[12].

Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT:

This research paper compares the performance of GitHub Copilot, Amazon CodeWhisperer, and ChatGPT code generation tools in terms of code quality metrics, such as Code Validity, Code Correctness, Code Security, Code Reliability, and Code Maintainability[13].

4.4. Unit Test

Evaluating and Improving ChatGPT for Unit Test Generation:

In this paper, the author used the generation-evaluation-repair framework in unit-test generation using ChatGPT. The evaluation focused more on correctness, sufficiency, readability and usability of

generated unit tests using ChatGPT[14].

ChatUniTest: a ChatGPT-based automated unit test generation tool:

This paper also employs the generation-evaluation-repair methodology for unit test generation. It compares ChatGPT-generated unit tests to EvoSuite, AthenaTest, A3Test, and ChatGPT in terms of test quality (correctness, line coverage, branch coverage)[15].

5. Experiments:

This section presents the experiments conducted to evaluate ChatGPT's potential in developing the Library Management System throughout the development life cycle, which includes the phases of Requirement gathering, UML diagram design (including class diagram and use case diagram), Database implementation, feature implementation"code generation" and unit testing. In each phase the results are discussed in terms of abilities and limitations with assessing the quality and effectiveness of ChatGPT's responses.

5.1. Requirement Gathering:

In the phase of Requirement Gathering, ChatGPT was instructed with the system's description to elicit requirements, including identifying existing requirements, suggesting new ones, and extracting both functional and non-functional requirements. The objective was to evaluate ChatGPT's understandability of the unstructured problem statement provided by users. The Evaluation was focused on firstly, to assess ChatGPT's capability in proposing comprehensive requirements from system description and if it can suggest complete requirements of the system, and secondly, to evaluate its capacity to distinguish between functional and non-functional requirements accurately. As an input, a description of the Library Management System was given to ChatGPT as follow:

"I want to design and develop a library management system with full-functionality. The front-end will be developed using Java Swing programming and back-end mysql database management system. Extract functional and nonfunctional requirements and suggest me important missing requirements

System functionality:

Admin:

-can Login

-Manage Books (add new book, delete book, search book, display books information, display list of books)

-Manage Members (add member (create accounts for them), remove member, display member information display list of members, cancel reservation of books if book is in checked out and has not returned before the reservation start date of new member, calculate and collect fine)

- Generate Reports: Generates reports on book availability, member activities, etc.

-A Member (person) can reserve a book for a desired start and end period, only if: the book exists(available in library), the member does not have any books that are being kept beyond their return date.

-A member can search a book, cancel a reservation at any time, check out a book or many books for a desired period of time, return the book, Renew/extend on a checkout duration. The book may be lost by member when checkout, The book can be find by member and return back to library, The member pay for fine of late return and for cost of lost books”

Identification of requirements and suggestion for improvement:

ChatGPT was evaluated for its ability to extract all of the requirements from system description and suggest more missing requirements to improve and complete the system's requirements. The ChatGPT's response presented in **Appendix 1**. ChatGPT demonstrated proficiency in extracting and listing requirements depicted within the system's description. Moreover, it exhibited the capacity to propose additional requirements, such as the integration of a Notification System, Authentication mechanism, and Fine Calculation Rules, which most of them were correct, comprehensive and essential for the system. Thereby contributing to the enhancement of the system's functionality.

However, despite its commendable performance, ChatGPT exhibited limitations in certain instances. Notably, it failed to identify specific requirements, such as the capability to update book information or member details, which were crucial components for the system's functionality. While these requirements were not explicitly articulated within the system description, this phase aimed to

evaluate ChatGPT's ability to infer such functionalities and subsequently recommend or identify missing requirements.

Extracting functional and nonfunctional requirements:

ChatGPT demonstrated considerable proficiency in distinguishing between functional and non-functional system requirements. As presented in Appendix1, the distinction between functional and non-functional requirements has been distinguished with accuracy, suggesting a keen understanding of their different nature within the context of software development. ChatGPT created functional requirements that were highly comprehensive, accurate, and nearly complete, indicating a good comprehension of the operating specifications required for the Library Management System.

While non-functional requirements were identified, they tended to be more general, adopting characteristics common to systems across multiple domains rather than unique characteristics essential to the Library Management System under consideration. This insight highlights the need for additional refinement in ChatGPT's discernment and articulation of non-functional criteria to ensure alignment with the unique operational and performance characteristics relevant to the system's functionality and user needs.

5.2. System Design - UML Modeling:

In this phase ChatGPT was instructed to generate UML diagrams of class diagrams and use case diagrams based on the requirements acquired by itself and some added by me. As an input the following requirements, ***Italicised Text 5.1*** were given to ChatGPT.

“I want to design and develop a library management system with full-functionality. The front-end will be developed using Java Swing programming and back-end mysql database management system. Provide me with the class diagram of the system.

Admin:

-can Login

-Manage Books (add new book, delete book, search book,display books information,

display list of books)

-Manage Members (add member (create accounts for them), remove member, display member information display list of members, cancel reservation of books if book is in checked out and has not returned before the reservation start date of new member, calculate and collect fine)

- Generate Reports: Generates reports on book availability, member activities, etc.

A Member (person) can reserve a book for a desired start and end period, only if:

- the book exists(available in library) and the member does not have any books that are being kept beyond their return date

- The member can search the book, cancel reservation at anytime, check out a book or many books for a desired period of time, return the book, Renew/extend on checkout duration

- The book may be lost by member when checkout and the book can be find by member and return back to library

- The member pay for fine of late return and for cost of lost books

A Book can be in following states for a given start and end period:

- available (it is at the library)

- checked out (it is with a member)

- reserved (has one or more open reservations on it)

- overdue(member did not return the book at the specified end date), pay fine

- expired (if a member did not pick the book at the pick up date)

-lost(member has checked out the book and lost), report the lost

-found(member has lost the book and found again), report the found

A Reservation can be in the following states:

- open (it is open for reservation other times, but it is not open for reservation when it is in checkout or another member has reserved for that period of time)
- canceled(a member will cancel his/her reservation at any time, Admin can cancel the reservation if it is in checkout state and not returned at the time of start of new reservation)
- completed (book has been checked out)
- due (member did not cancel and did not checkout the book)”

Class Diagram:

In this phase, ChatGPT was evaluated for identifying classes and attributes, determining the relationships between concepts(classes and attributes) and determining multiplicities. The above requirements were given to ChatGPT and asked to draw a class diagram of the Library Management System.

Evaluation Result:

In evaluating the class diagram generated by ChatGPT, several observations were noted. Initially, the system successfully identified essential classes such as Member, Book, Reservation, Library, and Report Generator. However, ChatGPT failed to identify classes such as BorrowedBooks, Fine, UserCredentials, as well as enumeration classes like ReservationStatus and BookStatus, as outlined in the requirements. It demonstrated limitations in recognizing all attributes associated with these classes. It comprehended the concept of inheritance, exemplified by designating User as an abstract class with Admin and Member as subclasses, a preference for a unified class structure was expressed.

An inconsistency was observed in the system's responses across different interactions, leading to variations in the produced diagrams. Moreover, while the class diagram depicted numerous methods, several of these methods were presented in incorrect classes. Furthermore, certain attributes specified in the class diagram were deemed inappropriate, raising concerns regarding the diagram's alignment with the actual system behavior.

As it is depicted in **Figure 5.1**, the class diagram is drawn from ChatGPT's response and **Figure 5.2**, the class diagram manually drawn. I used ChatGPT's responses as a draft version for drawing my own diagrams. In this diagram “**Figure**

5.1”, it is lacking classes of BorrowedBooks, Fine, UserCredentials, as well as enumeration classes like ReservationStatus and BookStatus. The Class of Library has two attributes of List of Books and list of members while lacking other attributes such as list of borrowedBook, and list of reservation. Book class has two attributes for the status of the book which are availability and state. Library class has a method of login which is appropriate for userCredentials class not library class. Furthermore it only depicts “1” and “*” multiplicity.

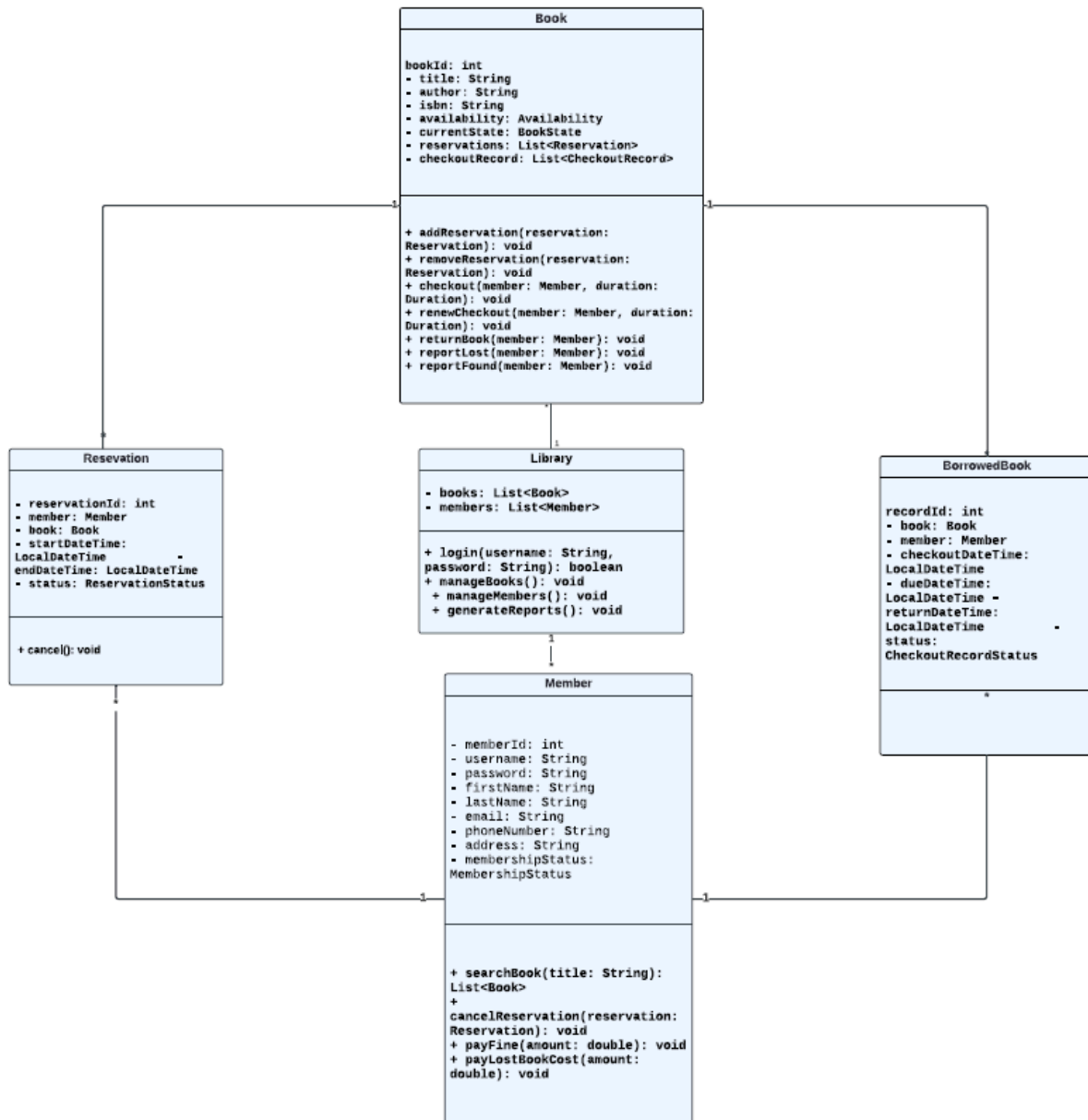


Figure 5.1 Class diagram draw from ChatGPT’s response

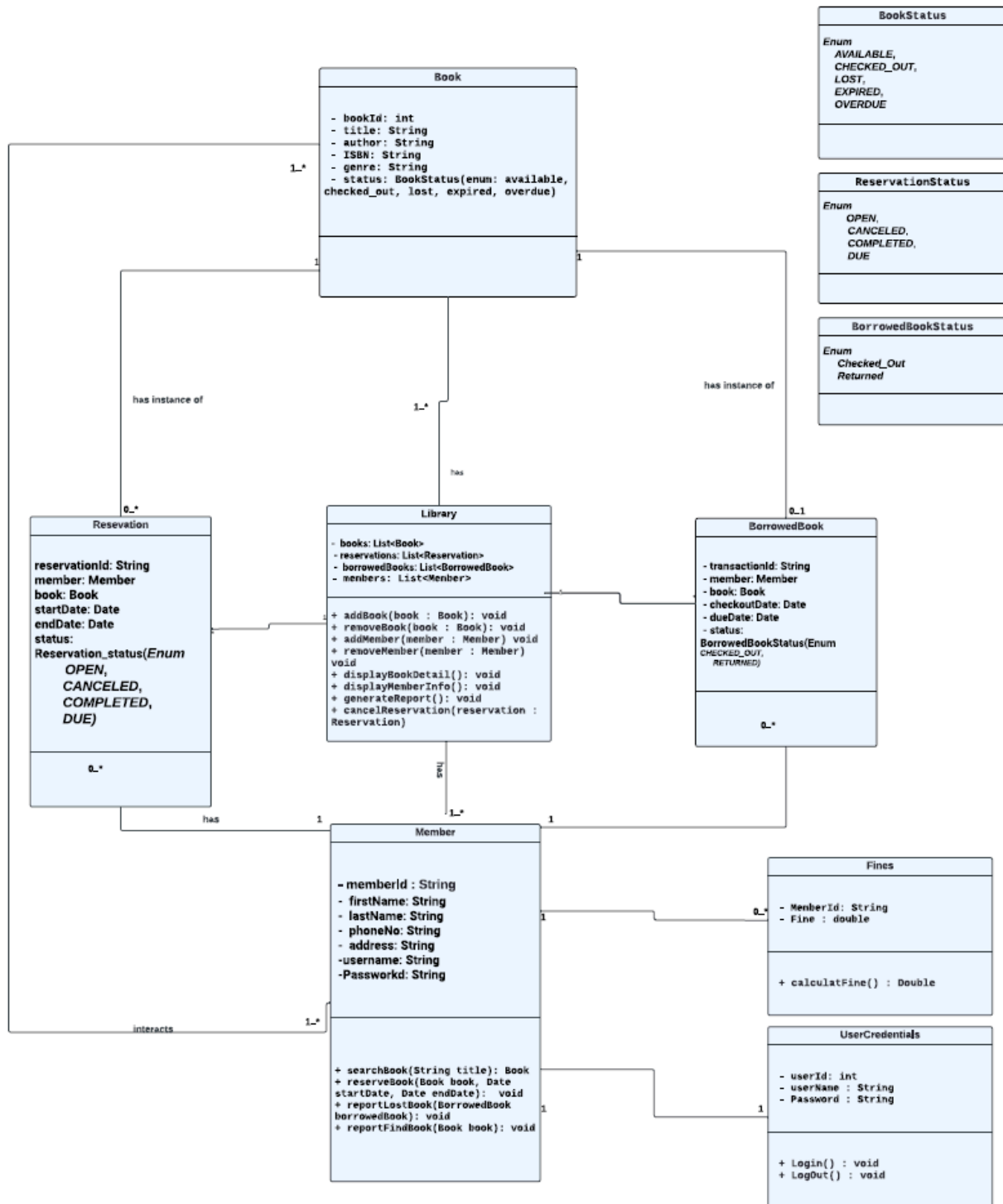


Figure 5.2 real class diagram of the project

Upon manual refinement of the diagrams and seeking clarification from ChatGPT regarding multiplicity, associations were accurately determined. However, the system's assessment was limited to verifying singular and plural relationships in multiplicity, lacking granularity in analyzing more complex associations. The *italicized text 5.2* shows the response of ChatGPT regarding association and multiplicity. I used ChatGPT's class diagram as a draft and drew my own class diagram manually. Then I gave my own class diagram to chatGPT to provide me the association and multiplicity.

“Book association with Reservation:

- 1. Relationship: Book has reservations.*
- 2. Multiplicity: One book can be associated with many reservations.*

Book association with BorrowedBook:

- 1. Relationship: Book has borrowed books.*
- 2. Multiplicity: One book can be associated with many borrowed books.*

Book association with Library:

- 1. Relationship: Library contains books.*
- 2. Multiplicity: One library contains many books.*

Reservation association with Member:

- 1. Relationship: Reservation belongs to a member.*
- 2. Multiplicity: One reservation belongs to one member.*

Reservation association with Book:

- 1. Relationship: Reservation is for a book.*
- 2. Multiplicity: One reservation is for one book.*

Member association with BorrowedBook:

- 1. Relationship: Member has borrowed books.*
- 2. Multiplicity: One member can have many borrowed books.*

UserCredentials association with Member:

1. *Relationship: User credentials belong to a member.*
2. *Multiplicity: One member has user credentials."*

Italicised Text 5.2 - ChatGPT's response of class diagram multiplicity.

As you can see in the response of ChatGPT, it only specifies one and many relationships. As an example, the relationship between member and borrowedBook is, "one member has zero or many borrowed books" the same with reservation but the response of ChatGPT is "one member can have many borrowed books". One member has only one user credential while ChatGPT does not specify explicitly that point, which seems a need for further investigation of ChatGPT in class diagrams, associations and multiplicity.

Use case diagram

During the Use Case diagram phase, ChatGPT was instructed with the same requirements as those provided for the class diagram. The focus of evaluation was its capability in identifying actors and Mapping use cases in alignment with the specified requirements. Additionally, the evaluation process aimed to determine ChatGPT's ability in establishing relationships within the diagram, including generalization, includes, and association relationships.

Evaluation Result:

The evaluation results indicate that ChatGPT could recognize the actors of Admin and Member, as well as their associated use cases, with some notable observations. Initially, ChatGPT drew a simple use case diagram which is not understandable, however, the description is well structured and comprehensive. The system primarily employed general associations without explicitly outlining the "includes" relationships between certain use cases. Notably, the use cases attributed to the Admin actor were initially limited to "manage book," "manage member," and "generate reports," with other use cases listed within parentheses as it was given in input without any changes or describing it as "includes" relationships. However, with new prompts regarding the location of "includes" relationships, the system could partially describe the "includes" relationships within the Admin actor, albeit with some omissions, while incorrectly designating the Member as a use case instead of an actor. The response of ChatGPT presented in ***Italicised Text 5.3***.

Despite these observations, ChatGPT adeptly utilized association relationships to denote connections between actors and use cases, demonstrating a sound understanding of this relationship type. Furthermore, the system successfully determined generalization relationships with the introduction of new

prompts, inferring the existence of an User actor and its subclasses, Member and Admin inheriting it.

Upon introducing prompts specifically addressing “includes” relationships, ChatGPT displayed varied performance, ***Italicised Text 5.4***. While it provided correct responses for “inclusion” relationships within the Admin actor, identifying the associated use cases somehow correct but not completely correct, the includes relationships as an example for manage member as described in system requirements consisting “add a new member, remove a new member, update information, display information, and search member” while ChatGPT suggests include relationships in manage member as “Cancel Reservation, Calculate and Collect Fine ” Furthermore, it could not identify inclusion relationships within the Member actor, incorrectly designating the Member as a use case rather than an actor and listing include relationships of member. This discrepancy underscores the need for further refinement and improvement in the system's consistency and accuracy, particularly in distinguishing between actors and use cases and accurately identifying inclusion relationships.

ChatGPT's response in use case diagram:

“Here's a simplified use case diagram based on your requirements:

In this diagram:

- *Admin: Can manage books, manage members, and generate reports.*
- *Manage Books: Admin can add, delete, search, and display books.*
- *Manage Members: Admin can add, remove, search, display members, cancel reservations, calculate and collect fines.*
- *Generate Reports: Admin can generate reports on various activities.*
- *Member: Can search, reserve, cancel reservation, checkout, return, renew checkout, report lost books, find lost books, pay fines, and pay for lost books.*

This diagram outlines the various interactions between the actors (Admin and Member) and the system.

”

Italicised text 5.3 - ChatGPT's response of use-case diagram of the project

When asked for association, includes and generalization relationship, could accurately understand two relationships of association and generalization but failed to recognize correctly the “includes” relationships. Below you can see ChatGPT's

response.

“Include Relationship:

- *Include relationships are used in use case diagrams to show that one use case includes another use case's behavior. In your system, include relationships can be applied to depict scenarios where one use case includes the functionality of another use case. For example:*
 - *The "Manage Members" use case might include the "Cancel Reservation" and "Calculate and Collect Fine" functionalities.*
 - *The "Member" use case might include "Search Book", "Cancel Reservation", "Renew Checkout", "Report Lost Book", "Find Lost Book", "Pay Fine", and "Pay Lost Book Fee" functionalities.”*

***Italicized Text 5.4** - ChatGPT's response regarding association and multiplicity.*

The use-case diagram of the project is depicted in **Figure 5.3** ChatGPT's has been used as a draft version and I drew the diagrams with LucidChart manually.

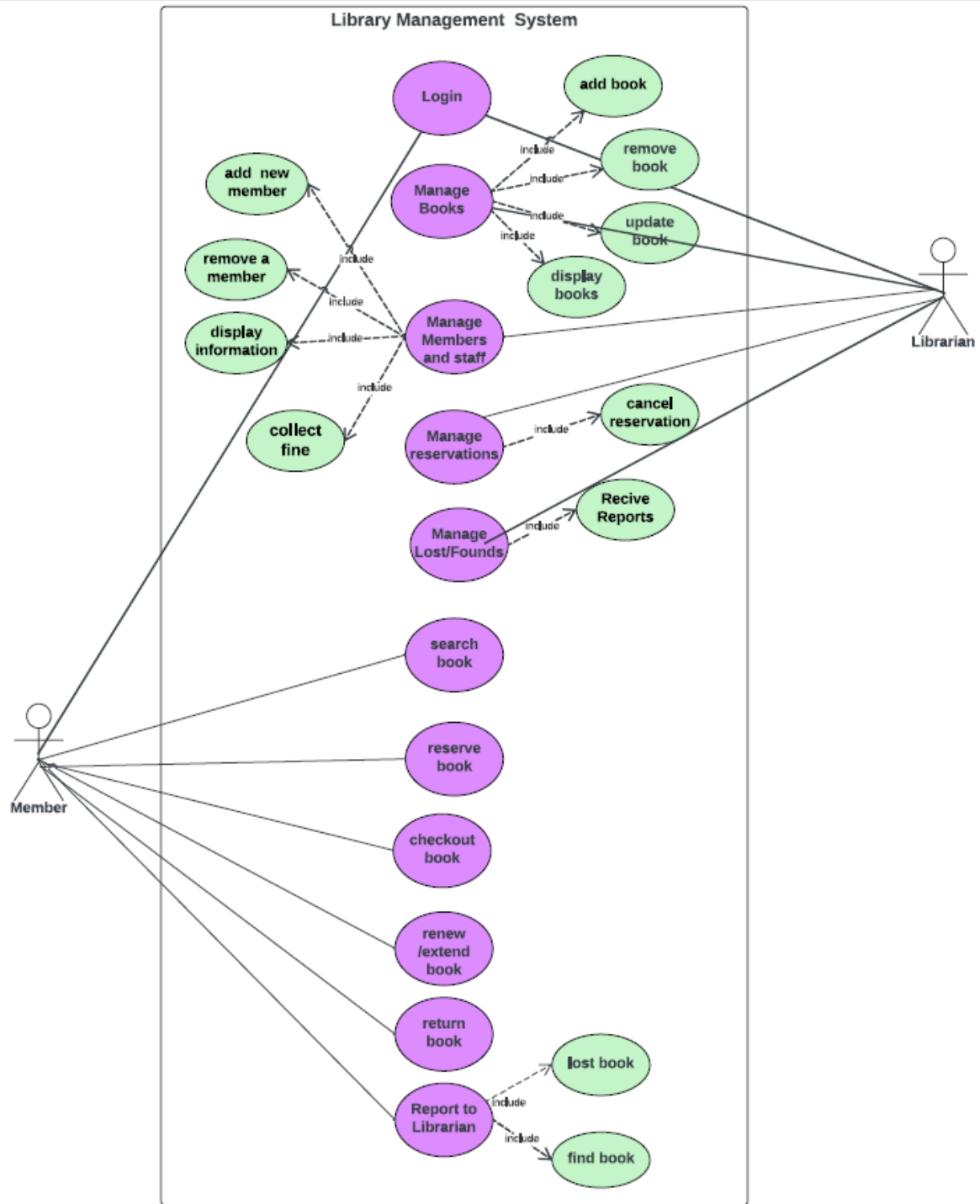


Figure 5.3 use-case diagram of the project Library Management System.

5.3. Database implementation

In the database implementation phase, ChatGPT was instructed with the class diagram of the Library Management System and tasked with transforming it

into an Entity-Relationship (ER) model, followed by generating the corresponding database implementation using MySQL commands. The evaluation process aimed to assess ChatGPT's proficiency in accurately producing an ER diagram based on the provided class diagram and subsequently translating it into MySQL commands in a coherent manner.

Evaluation Result:

The evaluation of ChatGPT's performance revealed a generally accurate extraction of entities and attributes from the provided specifications. However, a notable discrepancy arose where "library" was mistakenly designated as an entity in both the Entity-Relationship (ER) model and the subsequent database implementation.

Furthermore, during the initial prompt, ChatGPT encountered challenges in appropriately handling certain elements of the database schema. Specifically, the system failed to extract the "status" attribute from the enum class as intended, instead defining it as a boolean type. Additionally, it overlooked defining the foreign key constraint for the "memberId" attribute in the reservation table, indicating areas for improvement in accurately capturing relational dependencies within the database schema.

Despite these initial shortcomings, ChatGPT demonstrated proficiency in generating MySQL commands that adhered to syntactical standards. The system effectively translated the conceptual database design into executable commands, ensuring compatibility with MySQL database management system requirements and facilitating seamless implementation.

Upon receiving a second prompt for refinement, ChatGPT successfully addressed the identified issues in both the database schema and the accompanying ER diagram. Through iterative refinement, the system rectified the misclassification of "library" as an entity and appropriately defined the "status" attribute as an enum type. Additionally, ChatGPT corrected the oversight concerning the foreign key constraint for the "memberId" attribute, thereby enhancing the relational integrity of the database schema.

Overall, while encountering initial challenges in accurately capturing certain schema elements, ChatGPT demonstrated adaptability and responsiveness to refinement prompts, ultimately achieving a refined and coherent implementation of both the database and ER diagram. This iterative refinement process highlights

ChatGPT's potential for continuous improvement in database design and implementation tasks. ***Italicised Text-5.5*** shows ChatGPT's response in database implementation.

“Database implementation:

```
CREATE TABLE Book (  
  
    bookId INT PRIMARY KEY,  
  
    title VARCHAR(255),  
  
    author VARCHAR(255),  
  
    ISBN VARCHAR(255),  
  
    genre VARCHAR(255),  
  
    status ENUM('AVAILABLE', 'CHECKED_OUT', 'LOST', 'EXPIRED', 'OVERDUE'));  
  
CREATE TABLE Reservation (  
  
    reservationId VARCHAR(50) PRIMARY KEY,  
  
    startDate DATE,  
  
    endDate DATE,  
  
    bookId INT,  
  
    memberId VARCHAR(50),  
  
    FOREIGN KEY (bookId) REFERENCES Book(bookId),  
  
    FOREIGN KEY (memberId) REFERENCES Member(memberId));  
  
CREATE TABLE BorrowedBook (  
  
    transactionId VARCHAR(50) PRIMARY KEY,  
  
    checkoutDate DATE,  
  
    dueDate DATE,  
  
    returned BOOLEAN,  
  
    bookId INT,
```

```

    memberId VARCHAR(50),

    FOREIGN KEY (bookId) REFERENCES Book(bookId),

    FOREIGN KEY (memberId) REFERENCES Member(memberId)

);

CREATE TABLE Member (

    memberId VARCHAR(50) PRIMARY KEY,

    userId INT,

    firstName VARCHAR(255),

    lastName VARCHAR(255),

    phoneNo VARCHAR(20),

    address VARCHAR(255),

    userType ENUM('Admin', 'Member'));

CREATE TABLE Fines (

    memberId VARCHAR(50) PRIMARY KEY,

    Fine DOUBLE,

    FOREIGN KEY (memberId) REFERENCES Member(memberId));

CREATE TABLE UserCredentials (

    memberId VARCHAR(50) PRIMARY KEY,

    userName VARCHAR(255),

    Password VARCHAR(255),

    FOREIGN KEY (memberId) REFERENCES Member(memberId));”

```

Italicised Text-5.5 -ChatGPT’s response in database implementation

ER Model as follow:

“Book (bookId PK, title, author, ISBN, genre, status)

Reservation (reservationId PK, startDate, endDate, bookId FK, memberId FK)

BorrowedBook (transactionId PK, checkoutDate, dueDate, returned, bookId FK, memberId FK)

Member (memberId PK, userId, firstName, lastName, phoneNo, address, userType)

Fines (memberId PK FK, Fine)

UserCredentials (memberId PK FK, userName, Password)”

Italicised Text 5.6 - ChatGPT's response regarding ER model: presented in text based

5.4. Interface Design and functionality implementation(code generation):

During the Interface Design and code generation phase, the final project requirements, ***Italicised Text 5.1*** along with the class diagram, ***Figure 5.2*** and the implemented database, ***Italicised Text-5.5*** were presented to ChatGPT. Subsequently, the system was prompted and directed to provide guidance on the step-by-step development process for the project.

As ChatGPT prompted with the class diagram, database implementation and requirement of the system, it started step by step implementation of the system. The System first created a class of Library for connection to database. Then write down the skeleton of every class, both interface classes and backend classes with possible methods which were blank and I completed all of the relevant methods step by step with ChatGPT.

5.4.1. Interface design:

In the evaluation of Interface Design, particular attention was directed towards ChatGPT's capacity to accurately recognize components as specified by the requirements and generate comprehensive components inclusive of complete labels, textFields, and adherence to appropriate naming conventions. This phase sought to assess the system's ability to facilitate the creation of user interface elements that align with the prescribed functionalities and design standards. Moreover, the evaluation aimed to ascertain whether ChatGPT could offer assistance in addressing complex or intricate scenarios encountered during the interface design process, emphasizing the system's capability to provide guidance and support in

navigating challenging aspects of interface development.

Evaluation Result:

In the evaluation of ChatGPT's performance in Interface Design, the system exhibited proficiency in recognizing and generating various interface components, including text fields, labels, and tables, thereby facilitating the creation of a comprehensive user interface. However, a few components were inadvertently omitted, such as a text field for the "searchBook" function within the "BookManagementPanel" during the initial prompt. Despite these minor discrepancies, ChatGPT proved adept at providing valuable insights for brainstorming and outlining interface designs, showcasing its utility as a tool for conceptualizing interface layouts and functionalities. Notably, the system consistently adhered to commendable naming conventions throughout the project, demonstrating a high level of consistency and organization in component identification and labeling.

Moreover, ChatGPT effectively supported the design of complex interface elements, offering guidance and assistance in navigating intricate scenarios encountered during the design process. While not aligned towards professional-grade design, ChatGPT proved instrumental in facilitating learning by identifying essential components and functionalities required for the project. For instance, the system provided insights into implementing features such as card-layout and title-bar, contributing to the project's educational value and facilitating the acquisition of design principles and techniques.

However, despite these strengths, ChatGPT's interface design exhibited limitations in terms of sophistication and attention to detail. Specifically, the system's designs were characterized by a basic aesthetic and lacked consideration for size and layout optimization. This observation suggests an area for improvement, emphasizing the importance of refining ChatGPT's capabilities to encompass more nuanced design considerations and aesthetic enhancements for interface development.

An example of interface design given in the following *Italicised text-5.7*, I just prompted ChatGPT with "based on the requirement let's start with designing BookManagementPanel." As everything was instructed in the first prompt of

Implementation, it could understand the requirement and produced a basic interface design in which every component was complete except one textField of search functionality.

```
"import javax.swing.*;
```

```
import java.awt.*;
```

```
import java.awt.event.ActionEvent;
```

```
import java.awt.event.ActionListener;
```

```
import java.util.List;
```

```
public class BookManagementPanel extends JPanel {
```

```
    private JTextField titleField, authorField, genreField, isbnField;
```

```
        private JButton addButton, deleteButton, updateButton, searchButton,  
refreshButton;
```

```
    private JTable bookTable;
```

```
    private JScrollPane tableScrollPane;
```

```
public BookManagementPanel() {
```

```
    // Initialize Swing components
```

```
    titleField = new JTextField(20);
```

```
    authorField = new JTextField(20);
```

```
    genreField = new JTextField(20);
```

```
    isbnField = new JTextField(20);
```

```

addButton = new JButton("Add Book");

deleteButton = new JButton("Delete Book");

updateButton = new JButton("Update Book");

searchButton = new JButton("Search Book");

refreshButton = new JButton("Refresh");

bookTable = new JTable();

tableScrollPane = new JScrollPane(bookTable);

// Set layout

setLayout(new BorderLayout());

// Create input panel for adding and searching books

JPanel inputPanel = new JPanel(new GridLayout(5, 2));

inputPanel.add(new JLabel("Title:"));

inputPanel.add(titleField);

inputPanel.add(new JLabel("Author:"));

inputPanel.add(authorField);

inputPanel.add(new JLabel("Genre:"));

inputPanel.add(genreField);

inputPanel.add(new JLabel("ISBN:"));

inputPanel.add(isbnField);

inputPanel.add(addButton);

inputPanel.add(searchButton);

// Create button panel for delete, update, and refresh operations

```



```

JPanel buttonPanel = new JPanel(new FlowLayout());

buttonPanel.add(deleteButton);

buttonPanel.add(updateButton);

buttonPanel.add(refreshButton);


// Add input panel, button panel, and table to the main panel
add(inputPanel, BorderLayout.NORTH);

add(buttonPanel, BorderLayout.CENTER);

add(tableScrollPane, BorderLayout.SOUTH);

addButton.addActionListener(new ActionListener() {

    @Override

    public void actionPerformed(ActionEvent e) {

        });

deleteButton.addActionListener(new ActionListener() {

    @Override

    public void actionPerformed(ActionEvent e) {

        }

});

updateButton.addActionListener(new ActionListener() {

    @Override

```

```

    public void actionPerformed(ActionEvent e) {

        // Implement logic to update a book

    }

});

```

```

searchButton.addActionListener(new ActionListener() {

    @Override

    public void actionPerformed(ActionEvent e) {

        // Implement logic to search for a book

    }

});

```

```

refreshButton.addActionListener(new ActionListener() {

    @Override

    public void actionPerformed(ActionEvent e) {

        // Implement logic to refresh the book table}});}

// Method to populate the book table with data

public void populateBookTable(List<Book> bookList) {

    // Implement logic to populate the JTable with book data}}

```

5.4.2. Implementation(Code Generation):

In this phase, the evaluation was more focused on assessing the correctness, completeness, readability, and effectiveness of code generation and the effectiveness on error handling throughout the project of Library Management System.

Evaluation Result:

The evaluation results indicate that ChatGPT exhibited a proficient ability to explain the step-by-step implementation process of various functionalities, providing comprehensive explanations for each developmental stage. Although initial prompts yielded incomplete code segments, subsequent refinement requests and additional prompts facilitated the generation of complete code solutions for every functionality, showcasing ChatGPT's adaptability and responsiveness to iterative refinement. In my Library Management System Application, more than 90% of the codes are ChatGPT codes, with the remaining less than 10% being manually added in case ChatGPT did not provide effective code.

One of the significant characteristics of ChatGPT during the development process was its ability to provide advice and answers when faced with difficulties in describing specific component requirements or functionality. On multiple occasions, when I was not able to properly describe the needed components or functions, ChatGPT displayed a surprising capacity to deliver suggestions comparable to human-like intuition based on the natural language input. Two illustrative examples highlight this capability.

Firstly, when tasked with implementing reservation functionality within the system, whereby members could reserve books and access only their reserved items, members can checkout a book if it is reserved by him/her or it is available. If it was reserved by another member the member can not check it out. When it was illustrated in human natural language, ChatGPT advised using a session manager to track user sessions and provided guidance on the necessary methods, such as "isReservedbyThisMember" and "isReservedByOtherMember." Despite initial flaws, additional refinement prompts allowed difficulties to be resolved, resulting in the program running properly.

Secondly, when prompted for the creation of a functionality similar to YouTube's navigation, wherein clicking on a list of pages or panels would display the corresponding content in the center of the frame, ChatGPT swiftly identified the

need for a card layout and efficiently implemented the required functionality. Although the resultant design was basic, the functionality was executed flawlessly, showcasing ChatGPT's proficiency in translating conceptual requirements into actionable implementation steps.

Despite ChatGPT's potential in code generation, several drawbacks were identified throughout the evaluation process. Inconsistencies were observed in the codes generated by ChatGPT in response to new prompts, resulting in different outputs across different interactions. Furthermore, class attributes and methods, as specified in the class diagram and initial project skeleton, frequently changed during development of the project, introduced complexities and necessitated continual modifications during development. Additionally, the practice of providing a new database connection for each new method, despite the existence of a specific database connection class, resulted in redundant code generation.

Error handling:

The system proved highly effective in error detection and resolution, offering insightful explanations and guidance for error correction. Errors encountered during development were predominantly addressed through the systematic process of copying and pasting error messages, alongside detailed descriptions of their occurrence, to ChatGPT. While most errors were successfully resolved using this method, there were occasional instances where more extensive code segments were required. In such cases, all of the relevant codes along with the error message, and the precise line where error was happening were given as input to ChatGPT. Through this methodological approach, ChatGPT effectively resolved errors throughout the entirety of my project.

Although ChatGPT's assistance was invaluable in resolving most errors, there were instances where errors persisted, notably in cases such as the TitleBar Maximize functionality. Despite implementing a custom-designed title bar, the system encountered difficulties in achieving the desired maximize functionality, ultimately requiring manual intervention to address the issue. These findings underscore the need for enhancements in ChatGPT's consistency, adaptability to changing project specifications, and error resolution capabilities, particularly in addressing complex functionalities and ensuring correctness throughout the development process.

5.5. Unit Test Generation:

In this section I used ChatGPT in two scenarios. 1. simple test cases when do not have external dependencies. 2. in a complex real world project using Mocking Unit Test.

5.5.1 Simple Test Cases

I had a simple implementation of a library management system which I used for practical implementation of Object Oriented Programming. In this framework, the system's source codes with some manually implemented unit tests were provided, and ChatGPT was instructed to generate the remaining unit test cases to improve code coverage to 100%. The evaluation criteria encompassed examining the generated test cases for correctness, readability, completeness, and code coverage.

Evaluation result:

The evaluation showed positive results, as ChatGPT exhibited the capacity to generate practically comprehensive test cases that were completely correct, easy to understand, and well-structured. Only in few cases needed imports and only one incorrect assertion was identified, which was again solved by ChatGPT, indicating a high level of accuracy in the generated tests. Furthermore, the coverage of the test suite improved significantly, achieving almost 100% (99.6 %) increase in code coverage. These findings underscore ChatGPT's efficacy in facilitating the creation of robust test suites, thereby enhancing the reliability and stability of the implemented library management system.

5.5.2 Mocking Unit tests

In this phase the real project of Library Management System codes was given to ChatGPT for creating unit tests, focusing on correctness, readability, and syntactical accuracy. Additionally, the evaluation emphasized the appropriate use of Spy, Whitebox, and PowerMockito techniques to ensure comprehensive testing and validation of system functionalities.

Evaluation Result:

The outcomes of the assessment represented the identification of various errors, such as the mocking unnecessary classes, the calling of non-existent methods, incorrect parameter passing, and insufficient use of stub and spy functionalities. Despite repeated attempts to modify and correct these faults via new prompts, resolution remained unresolved. Individual lines met syntax and

reading norms, however overall syntactic coherence was not correct. Furthermore, it was determined that ChatGPT's efficacy in supporting mock testing and PowerMockito procedures was considered unsatisfactory.

As an example, **Figure-5.4** illustrates the ChatGPT's Mocking Unit Test generation, and **Figure-5.5** is my manually written Mocking Unit Test, which both of them employed on the same method.

```
@Test
void testAddBook1_Successful() throws SQLException {
    // Mocking JOptionPane
    JOptionPane optionPaneMock = Mockito.mock(JOptionPane.class);
    Mockito.doNothing().when(optionPaneMock).showMessageDialog(any(), any(), any(), any());

    // Mocking isISBNUnique to return true
    BookManagementPanel panelSpy = Mockito.spy(bookManagementPanel);
    Mockito.doReturn(true).when(panelSpy).isISBNUnique(anyString());

    // Mocking database interaction
    Connection connectionMock = Mockito.mock(Connection.class);
    PreparedStatement statementMock = Mockito.mock(PreparedStatement.class);
    Mockito.when(connectionMock.prepareStatement(any())).thenReturn(statementMock);
    Mockito.when(statementMock.executeUpdate()).thenReturn(1);
    Mockito.when(statementMock.executeQuery()).thenReturn(Mockito.mock(ResultSet.class));

    // Injecting mock connection
    DatabaseConnector.setConnection(connectionMock);

    // Test
    int result = panelSpy.addBook1("Test Title", "Test Author", "Test Genre", "123-456-789-0");
    assertEquals(1, result);
    verify(optionPaneMock).showMessageDialog(any(), eq("Book added successfully"));
}
```

Figure 5.4- ChatGPT's response in Mocking Test

This example serves as an illustration of ChatGPT's response regarding mocking unit tests, specifically within the `addBook1()` method within the `BookManagementPanel` class, the source code of `addBook1()` method inserted in **Appendix-2**. Notably, this method also includes another `addBook1()` method, invoked from the `Library` class, both of which are declared as public. For the sake of clarity and elucidation, I have supplemented this with a manually written and accurately formulated test case, as depicted in **Figure 5.5**.

In this Mocking Unit test, ChatGPT initially attempts to invoke the `JOptionPane` class, yet erroneously, as it is not required to be mocked, and the invocation is inaccurate. Notably, the `JOptionPane.showMessageDialog` method is a static

method, thus rendering it unsuitable for mocking; instead, PowerMocking would be necessary. However, the class `Library` has not been mocked, despite its significance as it interacts with the database. Conversely, irrelevant classes such as `Connection` and `PreparedStatement` have been mocked, which is unnecessary since mocking tests utilize fake objects rather than real ones; for improved analysis, I employed fake objects (e.g., title, author, genre, and ISBN) in my manually written test. Furthermore, ChatGPT references the `DatabaseConnector.setConnection` method, which does not exist. Moreover, incorrect parameters have been declared for the `BookManagementPanel.addBook1()` method by ChatGPT; these parameters relate to the `Library` class's `addBook1()` method, rather than `BookManagementPanel.addBook1()`.

```
@Test
public void TestAddBook1() {
    library = mock(Library.class);
    bmpanel = mock(BookManagementPanel.class);

    String title = "A";
    String author = "B";
    String genre = "C";
    String isbn = "D";

    when(library.addBook1(title,author,genre,isbn)).thenReturn(true);

    when(bmpanel.addBook1()).thenReturn(1);
    int succes = bmpanel.addBook1();
    Assertions.assertEquals(succes,1);

    Mockito.verify(bmpanel).addBook1();
}
```

Figure 5.5- Manually written Mocking Test

Spy:

As described in the glossary, spy mocking is similar to mocking, but it allows for the control of the internal behavior of the method being mocked. **Figure-5.6** depicts ChatGPT's spying test, while **Figure-5.7** displays my manually written spying test, both for the same method.

```

@Test
void testAddBook1_DuplicateISBN() {
    // Mocking JOptionPane
    JOptionPane optionPaneMock = Mockito.mock(JOptionPane.class);
    Mockito.doNothing().when(optionPaneMock).showMessageDialog(any(), any(), any(), any());

    // Mocking isISBNUnique to return false (duplicate ISBN)
    BookManagementPanel panelSpy = Mockito.spy(bookManagementPanel);
    Mockito.doReturn(false).when(panelSpy).isISBNUnique(anyString());

    // Test
    int result = panelSpy.addBook1("Test Title", "Test Author", "Test Genre", "123-456-789-0");
    assertEquals(0, result);
    verify(optionPaneMock).showMessageDialog(any(), eq("Duplicate ISBN found!, Please enter a new Unique ISBN"), eq("Error"), eq(JOptionPane.ERROR_MESSAGE));
}

@Test
void testAddBook1_InvalidISBNFormat() {
    // Mocking JOptionPane
    JOptionPane optionPaneMock = Mockito.mock(JOptionPane.class);
    Mockito.doNothing().when(optionPaneMock).showMessageDialog(any(), any(), any(), any());

    // Test
    int result = bookManagementPanel.addBook1("Test Title", "Test Author", "Test Genre", "invalid_isbn");
    assertEquals(0, result);
    verify(optionPaneMock).showMessageDialog(any(), eq("Invalid ISBN Format. Please enter a valid ISBN."), eq("Error"), eq(JOptionPane.ERROR_MESSAGE));
}

```

Figure-5.6 - spying addBook1() method by ChatGPT

```

@Test
public void TestUniqIsbnFalse() {
    BookManagementPanel bmpanel = new BookManagementPanel();
    Library = mock(Library.class);
    BookManagementPanel spyBmpanel = spy(bmpanel);

    when(!spyBmpanel.isISBNUnique("123-456-789-0")).thenReturn(false);

    System.out.println("Duplicate ISBN found!, Please enter a new Unique ISBN");

    when(spyBmpanel.isISBNUnique("123-456-789-0")).thenReturn(true);

    when(library.addBook1("Parental Psychology", "Noorzad", "Psychology", "123-456-789-0")).thenReturn(true);
    System.out.println("Book added Successfully!");
}

```

Figure-5.7- manually written spying addBook1() method

Both of these unit tests spy on the BookManagementPanel class, attempting to control the behavior of the addBook1() method. When the ISBN is not unique, a message is displayed; if the ISBN is unique, the Library.addBook1() method is called and the process continues.

The manually written test passed successfully. However, upon analysis of the tests generated by ChatGPT, several issues were identified. Firstly, it unnecessarily mocks classes of JOptionPane, Connection, and PreparedStatement but incorrectly, the screenshot is after the second prompt for refinement. It fails to mock the Library class, which is crucial for the mocking unit test, given its significance in interacting with the database. Additionally, it spies on the LibraryManagementPanel.addBook1() method but passes parameters from the Library class, despite neither being mocked nor called. In the second method, while checking for an invalid ISBN, it fails to mock the Library, spy on the

BookManagementPanel, or verify the validity of the ISBN, instead incorrectly calling the BookManagementPanel.addBook1() method with parameters from the Library.addBook1() method.

Despite multiple attempts, after three or four prompts, to rectify the issue by requesting ChatGPT not to mock unnecessary classes and considering the interaction of the Library class with the database, the issue remained unresolved. This led to the conclusion that mocking unit tests with ChatGPT may not be useful at all.

PowerMockito:

PowerMockito is an extension of the mocking framework with the ability to mock static methods, private methods, and constructors, which is not possible with conventional mocking.

To validate ChatGPT's capability in PowerMocking, I once again utilize the exact method addBook() (referred to as addBook1() in mocking) in both the Library and BookManagementPanel classes, albeit with different access specifiers. This time, the addBook() method in the Library class is declared as public static, while the method in the BookManagementPanel is private. Despite this difference, the code inside the addBook1() method used for mocking and the addBook() method used in PowerMocking remains the same. The source code for the method addBook() presented in *Appendix-3*.

For illustration *Figure-5.8* depicts ChatGPT's spying test, while *Figure-5.9* displays my manually written spying test, both for the same method.

In analyzing ChatGPT's response, we observe several drawbacks. Firstly, it successfully mocked the JOptionPane and Library classes using PowerMock, but it failed to instantiate or mock the BookManagementPanel correctly, instead relying solely on the declared variable of BookManagementPanel. Secondly, it mocked nonexistent and unnecessary methods such as getTitleField(), getAuthorField(), getGenreField(), and getIsbnField(). These methods do not exist, and even if they did, it would not be necessary to mock them. The predefined fake objects created at the beginning of the method are correct. Thirdly, it incorrectly called the BookManagementPanel.addBook() method, which is private and should be called using Whitebox.invokeMethod, as demonstrated in the manually written test case. Furthermore, it failed to verify whether the Library class was called or not.

Moreover, instead of verifying `BookManagementPanel.addBook()`, it verified `refreshBookList`. Not only did it verify the incorrect method, but the syntax used was also incorrect. Finally, although it attempted to control the behavior of the method in a manner reminiscent of spying, it failed to actually spy on the `BookManagementPanel` class. Considering these drawbacks and the inefficiency of ChatGPT in mocking, stubbing, spying, and power mocking, it appears unsatisfactory to me, and I believe there is a significant need for further investigation.

```
@Mock
private BookManagementPanel panelMock;
@Test
public void testAddBook_WithValidISBN_ShouldAddBookSuccessfully() {
    // Mocking the required fields
    String title = "Sample Title";
    String author = "Sample Author";
    String genre = "Sample Genre";
    String isbn = "123-4567890-1234567-X";

    // Mocking JOptionPane
    PowerMockito.mockStatic(JOptionPane.class);

    // Mocking Library.addBook() method to return true
    PowerMockito.mockStatic(Library.class);
    when(Library.addBook(title, author, genre, isbn)).thenReturn(true);

    // Setting up the mock objects
    when(panelMock.getTitleField()).thenReturn(title);
    when(panelMock.getAuthorField()).thenReturn(author);
    when(panelMock.getGenreField()).thenReturn(genre);
    when(panelMock.getIsbnField()).thenReturn(isbn);

    // Calling the actual method to test
    panelMock.addBook();

    // Verifying if refreshBookList() method is called
    verify(panelMock).refreshBookList();

    // Verifying if JOptionPane.showMessageDialog() is called with success message
    PowerMockito.verifyStatic(JOptionPane.class);
    JOptionPane.showMessageDialog(panelMock, "Book added successfully");
}
```

Figure 5.8 -Powermocking with ChatGPT

```

@Test
public void testAddBook() throws Exception {

    PowerMockito.mockStatic(Library.class);

    BookManagementPanel bookManagementPanel = new BookManagementPanel();
    Whitebox.invokeMethod(bookManagementPanel, "addBook");

    PowerMockito.verifyStatic(Library.class, Mockito.times(1));

    String title = "LLM in SE";
    String author = "James";
    String genre = "Study Book";
    String isbn = "123-4567890-12X";
    when(Library.addBook(title, author, genre, isbn)).thenReturn(true);
    PowerMockito.verifyPrivate(BookManagementPanel.class, Mockito.times(1)).invoke("addBook");

}

```

Figure 5.9 - Manually written test with Powermockito.

6. Future Work:

Despite its remarkable performance in various phases of the software development life cycle, our study has identified several areas for future improvement and investigation for enhancing the effectiveness and usability of ChatGPT in real-world software projects.

GUI, there is an important need for improvement in GUI design when utilizing ChatGPT. While ChatGPT has shown the ability to recognize GUI components, the designs produced are often basic and may not consider factors such as size and layout optimization which sometimes induced invisibility of components such as TextFields and Labels. To address this gap, future work could explore techniques for enhancing ChatGPT's understanding of design principles and ensuring that generated GUI designs are visually appealing and user-friendly. Specifically, efforts should be made to optimize size and layout to ensure that all components are visible and accessible to users.

In terms of system design, particularly UML diagrams, further investigation and improvement are needed. While ChatGPT has demonstrated the ability to generate UML diagrams, there is a need for refinement in terms of accuracy and completeness. Future research could focus on enhancing ChatGPT's understanding of complex system architectures and ensuring that generated diagrams accurately reflect the underlying structure of the software.

Furthermore, in the realm of mocking tests, there is a significant need for further investigation and improvement. Our study revealed challenges associated

with using ChatGPT to generate reliable mock tests, particularly in scenarios involving complex dependencies or external systems. Future research could explore advanced techniques for mocking, spying, stubbing, and powermockito framework in unit testing scenarios, aiming to improve the accuracy and effectiveness of ChatGPT-generated tests.

Finally, there is a need for consistency improvement in ChatGPT's outputs across different interactions and prompts. Our research highlighted instances where ChatGPT's responses varied depending on the input prompt, leading to inconsistencies in the generated outputs. Addressing this issue will require ongoing refinement of ChatGPT's algorithms and training data to ensure consistent and reliable performance in real-world software development projects.

7. Conclusion:

In this study, we conducted a comprehensive investigation into the role of ChatGPT in practical software development scenarios, focusing on the development of a Library Management System using Java and MySQL. Our research aimed to bridge the gap between simple scenario discussions and real-world applications of ChatGPT, providing nuanced insights into its strengths and limitations across different phases of the software development lifecycle.

Through rigorous experimentation and evaluation, we assessed ChatGPT's efficacy in various development tasks, including requirement gathering, system design, database implementation, interface design, code generation, and unit test generation. Our findings revealed both the capabilities and challenges of integrating ChatGPT into complex development projects, highlighting areas of improvement and opportunities for further research.

Despite its potential, ChatGPT exhibited limitations in certain tasks, such as understanding complex requirements, generating accurate diagrams, and producing reliable unit tests. However, it demonstrated proficiency in tasks such as generating code snippets, development of database, assisting in interface design, and providing insights during the development process.

Furthermore, our study identified the importance of incorporating advanced techniques such as mocking, spying, and stubbing in unit testing scenarios to improve ChatGPT's effectiveness in generating reliable test cases for real-world

applications. By leveraging these techniques, developers can enhance the quality and reliability of their software while leveraging ChatGPT's capabilities.

Overall, this research contributes to the growing body of knowledge surrounding AI-assisted software development, offering practical insights for developers and researchers alike. By addressing the gap between simple scenarios and practical implementation, we aim to facilitate informed decision-making and drive advancements in AI-driven development methodologies.

As the field of AI-assisted software development continues to evolve, further research is needed to address the limitations identified in this study and explore new avenues for leveraging ChatGPT's capabilities. By continuing to refine and improve AI technologies, we can unlock new opportunities for innovation and enhance the efficiency and effectiveness of software development processes.

References

- [1] "What Is Software Development? Definition and Types | Simplilearn," Simplilearn.com. Accessed: Mar. 29, 2024. [Online]. Available: <https://www.simplilearn.com/tutorials/programming-tutorial/what-is-software-development>
- [2] M. Tornqvist, "The Importance of Requirements Gathering in Software Development," Medium. Accessed: Mar. 29, 2024. [Online]. Available: <https://bootcamp.uxdesign.cc/the-importance-of-requirements-gathering-in-software-development-a8102b1f666e>
- [3] "Software Design Process - Software Engineering," GeeksforGeeks. Accessed: Mar. 29, 2024. [Online]. Available: <https://www.geeksforgeeks.org/software-engineering-software-design-process/>
- [4] "Untitled document," Google Docs. Accessed: Mar. 29, 2024. [Online]. Available: https://docs.google.com/document/d/1S_EKFhgziS7avaVcl_WnwxKUA1xMxpa6VXe3FTcjSTQ/edit?usp=embed_facebook
- [5] "What is a UML Diagram? | Different Types and Benefits | Miro," <https://miro.com/>. Accessed: Mar. 29, 2024. [Online]. Available: <https://miro.com/diagramming/what-is-a-uml-diagram/>
- [6] C. Team, "Mock Testing: Understanding the Benefits and Best Practices," CodiumAI. Accessed: Mar. 29, 2024. [Online]. Available: <https://www.codium.ai/blog/mock-testing/>
- [7] baeldung, "Introduction to PowerMockito | Baeldung." Accessed: Mar. 29, 2024. [Online]. Available: <https://www.baeldung.com/intro-to-powermock>
- [8] K. Ronanki, C. Berger, and J. Horkoff, "Investigating ChatGPT's Potential to Assist in Requirements Elicitation Processes." arXiv, Jul. 14, 2023. Accessed: Mar. 29, 2024. [Online]. Available: <http://arxiv.org/abs/2307.07381>
- [9] C. Arora, J. Grundy, and M. Abdelrazek, "Advancing Requirements Engineering through Generative AI: Assessing the Role of LLMs." arXiv, Nov. 01, 2023. Accessed: Mar. 30, 2024. [Online]. Available: <http://arxiv.org/abs/2310.13976>
- [10] L. de Ridder, "Generating PlantUML Diagrams with ChatGPT," Medium. Accessed: Apr. 01, 2024. [Online]. Available: <https://medium.com/@Empanado/generating-plantuml-diagrams-with-chatgpt-7905a34b6c14>
- [11] D.-K. Kim, *Using ChatGPT to Develop Software Systems: Alert to Software Engineers?* 2023. doi: 10.13140/RG.2.2.26388.78725.
- [12] Y. Dong, X. Jiang, Z. Jin, and G. Li, "Self-collaboration Code Generation via ChatGPT." arXiv, May 24, 2023. doi: 10.48550/arXiv.2304.07590.
- [13] B. Yetiştiren, I. Özsoy, M. Ayerdem, and E. Tüzün, "Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT." arXiv, Oct. 21, 2023. doi: 10.48550/arXiv.2304.10778.
- [14] Z. Yuan *et al.*, "No More Manual Tests? Evaluating and Improving ChatGPT for Unit Test Generation." arXiv, May 08, 2023. Accessed: Apr. 01, 2024. [Online]. Available: <http://arxiv.org/abs/2305.04207>
- [15] Z. Xie, Y. Chen, C. Zhi, S. Deng, and J. Yin, "ChatUniTest: a ChatGPT-based automated unit test generation tool." arXiv, May 08, 2023. doi: 10.48550/arXiv.2305.04764.

Appendix

Appendix -1: requirement gathering

“Functional Requirements:

User Authentication:

- *Admin should be able to log in securely.*

Book Management:

- *Admin should be able to add new books to the system.*
- *Admin should be able to delete existing books.*
- *Admin should be able to search for books by various criteria.*
- *Admin should be able to view detailed information about each book.*
- *Admin should be able to display a list of all books in the library.*

Member Management:

- *Admin should be able to add new members (create accounts).*
- *Admin should be able to remove existing members from the system.*
- *Admin should be able to search for members by various criteria.*
- *Admin should be able to view detailed information about each member.*
- *Admin should be able to display a list of all members.*
- *Admin should be able to cancel reservations of books for members.*
- *Admin should be able to calculate and collect fines from members for late returns.*

Report Generation:

- *Admin should be able to generate reports on book availability.*
- *Admin should be able to generate reports on member activities.*

Book Reservation:

- *Members should be able to reserve books for a desired start and end period, subject to availability.*
- *Members should not be able to reserve a book if they have overdue books.*
- *Members should be able to cancel their reservations at any time.*

Book Checkout:

- *Members should be able to check out one or multiple books for a desired period of time.*
- *Members should be able to renew or extend the checkout duration if no other members have reserved the book.*
- *Members should be able to report lost books and pay for them.*

Book Return:

- *Members should be able to return books.*
- *Members should be notified of overdue books and should be able to pay fines for late returns.*
- *Members should be able to report found books if previously reported lost.*

Non-functional Requirements:

Security:

- *The system should ensure secure login for both admin and members.*
- *Access to sensitive functionalities like adding/deleting books and members should be restricted to admin only.*

Performance:

- *The system should be able to handle a large number of concurrent users efficiently.*
- *Database queries should be optimized for quick retrieval of information.*

Reliability:

- *The system should be robust and reliable, with minimal downtime.*
- *Data integrity should be maintained at all times.*

Scalability:

- *The system should be designed to scale easily with an increase in the number of users and books.*

User Interface:

- *The user interface should be intuitive and easy to use for both admin and members.*
- *It should provide clear feedback on actions taken.*

Data Backup and Recovery:

- *Regular backups of the database should be performed to prevent data loss.*
- *There should be mechanisms in place for data recovery in case of system failure or corruption.*

Audit Trail:

- *The system should maintain an audit trail of all actions taken by admin and members, including login/logout activities, book reservations, checkouts, returns, fines collected, etc.*

Missing Requirements:

Email Notifications:

- *Members should receive email notifications for overdue books, upcoming reservation expirations, etc.*

Fine Calculation Rules:

- *Define specific rules for fine calculation, such as the amount charged per day for late returns.*

Maximum Number of Books Checkout:

- *Specify the maximum number of books a member can check out simultaneously.*

Reservation Priority:

- *Define rules for prioritizing reservations in case of multiple requests for the same book during the same period.*

Book Categories and Genre:

- *Implement a system for categorizing books by genre or subject for easier search and management.*

Accessibility:

- *Ensure that the system is accessible to users with disabilities, adhering to accessibility standards.*

Feedback Mechanism:

- *Provide a mechanism for users to provide feedback or suggestions for improvement.*

System Notifications:

- *Implement system-wide notifications for important events such as book availability, reservation confirmations, etc.”*

Appendix-2: addBook1() method source code

BookManagementPanel.addBook1():

```
public int addBook1(){
```

```
    String title = titleField.getText();
```

```
    String author = authorField.getText();
```

```
    String genre = genreField.getText();
```

```
    String isbn = isbnField.getText();
```

```

                                                                    if
(isbn.matches("^\\d{1,5}-\\d{1,7}-\\d{1,6}-[0-9Xx]$" | "\\d{3}-\\d{1,5}-\\d{1,7}-\\d{1,6}-[0-9Xx]$" )) {

        JOptionPane.showMessageDialog(this, "Invalid ISBN Format. Please
enter a valid ISBN.", "Error", JOptionPane.ERROR_MESSAGE);

        // return 1;

    }

    if(!isISBNUnique(isbn)) {

        JOptionPane.showMessageDialog(this,"Duplicate ISBN found!, Please enter a
new Unieque ISBN", "Error", JOptionPane.ERROR_MESSAGE);

        //return 1;

    }

    Library library = new Library();

    // Call Library method to add the book to the database

    boolean success = library.addBook1(title, author, genre, isbn);

    // If book was added successfully, refresh the book list

    if (success) {

        refreshBookList();

        // Optionally, display a message to the user

        JOptionPane.showMessageDialog(this, "Book added successfully");

        return 1;
    }

```

```

    } else {

        // Optionally, display an error message if book addition failed

        JOptionPane.showMessageDialog(this, "Failed to add book", "Error",
JOptionPane.ERROR_MESSAGE);

    }

    return 0;

}

```

Library.addBook1():

```

public boolean addBook1( String title, String author, String genre, String isbn) {

    try (Connection connection = DatabaseConnector.getConnection()) {

        String sql = "INSERT INTO Book (Title, Author, Genre, ISBN, Status)
VALUES (?, ?, ?, ?, ?)";

        PreparedStatement statement = connection.prepareStatement(sql);

        statement.setString(1, title);

        statement.setString(2, author);

        statement.setString(3, genre);

        statement.setString(4, isbn);

        statement.setString(5, BookStatus.AVAILABLE.name()); // Set the initial
status

        int rowsInserted = statement.executeUpdate();

        return rowsInserted > 0;
    }
}

```

```

    } catch (SQLException ex) {

        ex.printStackTrace();

        return false;

    }

}

```

BookManagementPanel.isUniqueISBN():

```

public boolean isISBNUnique(String isbn) {

    try (Connection connection = DatabaseConnector.getConnection()) {

        String sql = "SELECT COUNT(*) AS count FROM Book WHERE isbn =
?";

        PreparedStatement statement = connection.prepareStatement(sql);

        statement.setString(1, isbn);

        ResultSet resultSet = statement.executeQuery();

        if (resultSet.next()) {

            int count = resultSet.getInt("count");

            return count == 0; // Return true if no books with the same ISBN
found

        }

    } catch (SQLException ex) {

        ex.printStackTrace();

    }

    // If an exception occurs or the query fails, return false by default

```

```
return false;
```

```
}
```

Appendix-3: *addBook()* Source code

BookManagementPanel.addBook():

```
private void addBook(){
```

```
    String title = titleField.getText();
```

```
    String author = authorField.getText();
```

```
    String genre = genreField.getText();
```

```
    String isbn = isbnField.getText();
```

```
    // Validate input
```

```
        if (title.isEmpty() || author.isEmpty() || genre.isEmpty() ||  
isbn.isEmpty()) {
```

```
            JOptionPane.showMessageDialog(null, "Please fill in all fields.");
```

```
            return;
```

```
        }
```

```
        if
```

```
(!isbn.matches("^\\d{1,5}-\\d{1,7}-\\d{1,6}-[0-9Xx]$|^\\d{3}-\\d{1,5}-\\d{1,7}-\\d{1,6}-[0-9Xx]$")) {
```

```
            JOptionPane.showMessageDialog(this, "Invalid ISBN Format.  
Please enter a valid ISBN.", "Error", JOptionPane.ERROR_MESSAGE);
```

```

        return;

    }

    if(!isISBNUnique(isbn)) {

        JOptionPane.showMessageDialog(this,"Duplicate ISBN found!, Please
enter a new Unique ISBN", "Error", JOptionPane.ERROR_MESSAGE);

        return;

    }

    // Call Library method to add the book to the database

    boolean success = Library.addBook(title, author, genre, isbn);

    // If book was added successfully, refresh the book list

    if (success) {

        refreshBookList();

        // Optionally, display a message to the user

        JOptionPane.showMessageDialog(this, "Book added successfully");

    } else {

        // Optionally, display an error message if book addition failed

        JOptionPane.showMessageDialog(this, "Failed to add book", "Error",
JOptionPane.ERROR_MESSAGE);

    }

}

Library.addBook():

public static boolean addBook( String title, String author, String genre, String isbn)
{

```

```

    try (Connection connection = DatabaseConnector.getConnection()) {

        String sql = "INSERT INTO Book (Title, Author, Genre, ISBN, Status)
VALUES (?, ?, ?, ?, ?)";

        PreparedStatement statement = connection.prepareStatement(sql);

        statement.setString(1, title);

        statement.setString(2, author);

        statement.setString(3, genre);

        statement.setString(4, isbn);

        statement.setString(5, BookStatus.AVAILABLE.name()); // Set the
initial status

        int rowsInserted = statement.executeUpdate();

        return rowsInserted > 0;

    } catch (SQLException ex) {

        ex.printStackTrace();

        return false;

    }

}

```