Software Requirements Specification

for

Library Management System

Version 1.0 approved

Documentation Prepared by Khalid Tarek

Project Made by The RAD Squad

For Software Engineering (2) Final Project - EELU

26 November, 2020

Table of Contents

Table of Contents ii

Revision History iii

1. Introduction 1

1.1 Purpose 1

1.2 Intended Audience and Reading Suggestions 1

1.3 Project Scope 1

1.4 References 1

2. Overall Description 2

2.1 Product Features 2

2.1.1 User Authentication 2

2.1.2 Library Member Functionality 2

2.1.3 Librarian Functionality 2

2.1.4 Book Author Functionality 2

2.2 Operating Environment 3

2.3 Design and Implementation Constraints 3

2.4 Assumptions and Dependencies 3

3. System Features 3

3.1 User Authentication 3

3.1.1 Description and Priority 3

3.1.2 Stimulus/Response Sequences 3

3.1.3 Functional Requirements 4

3.2 Library Member Functionality 4

3.2.1 Description and Priority 4

3.2.2 Stimulus/Response Sequences 4

3.2.3 Functional Requirements 4

3.3 Librarian Functionality 5

3.3.1 Description and Priority 5

3.3.2 Stimulus/Response Sequences 5

3.3.3 Functional Requirements 5

3.4 Book Author Functionality 5

3.4.1 Description and Priority 5

3.4.2 Stimulus/Response Sequences 6

3.4.3 Functional Requirements 6

4. Classes and their Characteristics 6

4.1 Abstract Generalized “User” Class 7

4.1.1 Class Attributes 7

4.1.2 Class Functions 7

4.2 “Member” Class 7

4.2.1 Class Attributes 7

4.2.2 Class Functions 7

4.3 “Librarian” Class 8

4.3.1 Class Attributes 8

4.3.2 Class Functions 8

4.4 “Author” Class 9

4.4.1 Class Attributes 9

4.4.2 Class Functions 9

4.5 “Book” Class 10

4.5.1 Class Attributes 10

4.5.2 Class Functions 10

4.6 “DatabaseUtils” Utility Class 10

4.6.1 Class Attributes 10

4.6.2 Class Functions 11

5. External Interface Requirements 12

5.1 User Interfaces 12

5.1.1 Login Screen 12

5.1.2 Main Screen 12

5.1.3 New Book Screen 12

5.1.4 Registration Screen 12

5.2 Software Interfaces 12

6. Other Nonfunctional Requirements 12

6.1 Performance Requirements 12

6.2 Safety Requirements 13

6.3 Security Requirements 13

7. Other Requirements 13

7.1 Database Requirements 13

Appendix A: Glossary 14

Appendix B: Analysis Models 14

Appendix C: Issues List 20

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Khalid | 26/11/20 | Document Creation | 1.0 |
|  |  |  |  |

# Introduction

## Purpose

This simple software is intended to simulate an imaginary library system where library members, librarians, and authors can interact with a database holding their user accounts and their respective books through a minimalistic user interface. The main purpose of this project is to practice software engineering concepts and to act as the final project for the SWE305 - Software Engineering (2) course (Egyptian E-learning University) of fall 2020-2021.

## Intended Audience and Reading Suggestions

The intended readers for this document first and foremost are the coordinators of the SWE305 course. Secondly, this document is also intended for the developers and/or users of this system, to understand the different aspects of it and how interfaces communicate with the database described.

The first section gives a brief introduction about the software, followed by a section explaining targeted users and the environment on which this system will operate will be given. Then an overview of the software’s features and its requirements will be explained in more detail, followed by the interfaces with which the targeted user will be interacting with. Finally, the nonfunctional requirements of the system will be explained, followed by a list of appendices for this document.

## Project Scope

This project’s scope encompasses the very basic functionality of a library.

* This program allows members to borrow books, add credit with which they can pay the fines that generate when the user borrows a book.
* It also allows authors to add or remove their books (while this varies greatly in reality, since the event of releasing books to the public goes through many phases, including publishing as the like, but for the purposes of this software, it was dumbed down to authors adding their books directly).
* This software also gives librarians the ability to act as an “administrator” of a sort for the system, allowing them to create new member or author accounts or remove members, authors or books from the system. In reality though, the true administrators of the system will have total control of everything, but that’s beyond the scope of this system and will not be implemented other than in a conceptual view of the business use-case model.

This software is developed purely in Java, including the user interface and the backend interface for the database.

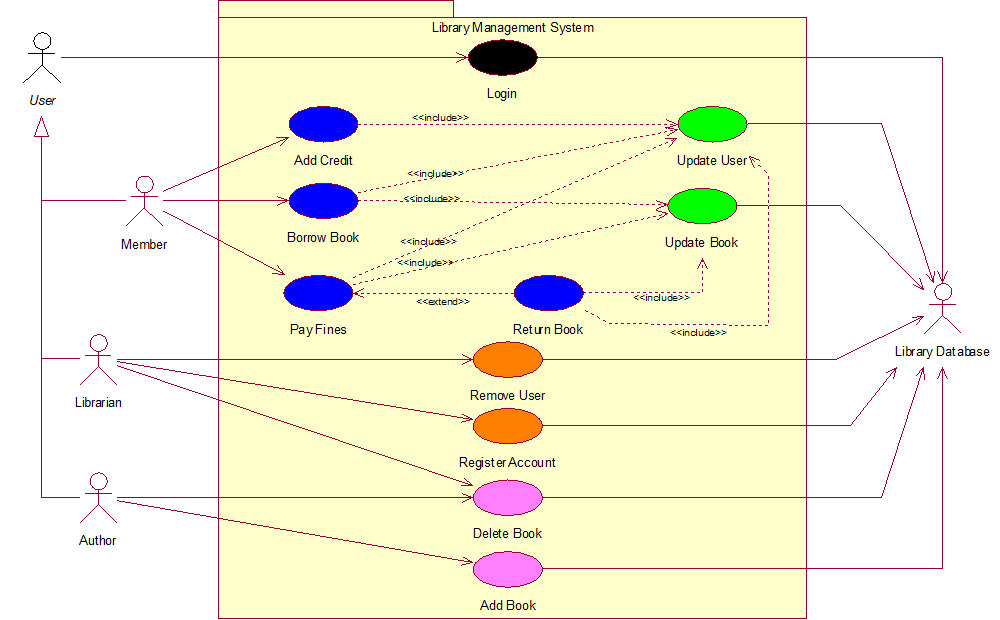
The database system used, its restraints, requirements and scope are further explained later in this document

## References

* [Karl E. Wiegers Software Requirements Specification Template](https://cs.gmu.edu/~dfleck/classes/cs421/spring08/srs_template.doc)
* UML Models.mdl (In the same directory as this document)

# Overall Description

## Product Features



**Appendix Figure 1:** Use Case Diagram

### User Authentication

The system can distinguish each user by their account credentials, i.e. username and password. This feature is noted in **Appendix Figure 1** as the black colored use case by the generalized actor *User*.

### Library Member Functionality

The system can help members borrow books, track their borrowed books, keep credit, and allow fine payments for the borrowed books. This feature is noted in **Appendix Figure 1** as all the green colored used cases by the actor *Member*.

### Librarian Functionality

The system can allow librarians to create new member or author accounts, remove these accounts and remove book entries. This feature is noted in **Appendix Figure 1** as all the orange colored used cases by the actor *Librarian*, and one of the prominently *Author* use-case “Delete Book”.

### Book Author Functionality

The system can allow authors to add their books into the database or remove them. This feature is noted in **Appendix Figure 1**as all the pink colored used cases by the actor *Author*.

## Operating Environment

This project was created on Windows 10, using the Net Beans IDE and Microsoft SQL Server 12. Whether or not this program can work on other versions of windows, or using a DBMS other than MSS 12 is beyond our knowledge.

## Design and Implementation Constraints

The only constraint this program has is the existence of a database with the architecture explained in **section 7.1. Database Requirements**. Otherwise, there shouldn’t be any other constraints, or at least not that we have anticipated.

## Assumptions and Dependencies

As mentioned in the previous section, this project has a dependency on Microsoft SQL Server 12, as well as the JDBC API used as an interface to communicate with the DBMS/Database.

# System Features

## User Authentication

### Description and Priority

This feature incorporates the ability for any user to authenticate themselves by entering their credentials, i.e. username and password. This feature has the highest priority as at this stage, the user gets the privileges he or she must get to use the program. As mentioned before, the use-case associated with this feature is only the “Login” Use-case denoted in the **Appendix Figure 1: Use Case Diagram**. Further details are given in the associated activity diagram in the **Appendix Figure 11: Login Activity Diagram**.

### Stimulus/Response Sequences

The only actions that stimulate this feature of the system is entering the user credentials to the system to check their validity. On doing so, two ultimate possible outcomes could happen:

1. If the entered credentials are not valid, show an appropriate error message. Invalidity of credentials could be:
   1. Username left empty, in which case the error would be “Empty Username”
   2. Password left empty, in which case the error would “Empty Password”
   3. The username and password are entered but either the username or password don’t correlate with a record in the “accounts” table. In this case the error would be “Wrong Username or Password”
2. If the entered credentials are valid, the user would get a screen with the privileges according to their user type, i.e. Member, Librarian or Author screens.

### Functional Requirements

LGN-REQ-1: The System **must** be able to distinguish between each user based on their provided credentials.

LGN-REQ-2: The System **must** reject invalid credentials and block access in that case.

LGN-REQ-3: The System **must** allow limited access in case the credentials are valid. This limitation is based on the type of user distinguished by their credentials

LGN-REQ-4: The System **could** allow guest sessions for an arbitrary member to temporarily view the system and simulate borrowing and such. Although, this arbitrary member wouldn’t get saved into the database or actually borrow any books.

## Library Member Functionality

### Description and Priority

This feature incorporates a member’s functionality. The use-cases associated with this functionality are the blue colored use-cases in **Appendix Figure 1: Use-Case Diagram.** This functionality includes:-

* Adding credit to the member’s account.
* Borrowing books
* Paying the book’s fines.
* Returning books to the library

This feature is of high priority as it is considered to be half of the consumer functionality, the author functionality being the other half.

### Stimulus/Response Sequences

The actions that a member can take and the responses the system can give are as follows:

* Add an integer number as credit to the member’s account. The system can have one of two responses (**Appendix Figure 12: Add Credit**)
  + Either the entered amount is negative, or will result in an overflow, in which case the system will reject this operation and do nothing.
  + Or the system will get a valid number and add it to the user’s credit.
* Borrow a book from a list of all possible books to borrow from the library. The system adds the book to the member’s borrowed books list, edits the book to be “borrowed by” the member, record that the book’s fine is yet to be paid, and add its fine to the member’s fines. (**Appendix Figure 4: Borrow Book Activity Diagram**)
* Selecting a book’s fine to pay. Various responses can be given by the system based on the selected book (**Appendix Figure 5: Pay Fines Activity Diagram** and **Appendix Figure 6: Return Book**)
  + If the selected book isn’t currently borrowed by the member, or has it’s fine already paid, prompt the user to return the book and exit with a message shown to the member saying “No fine to pay”
  + If the member doesn’t have enough credit, do nothing and exit a message shown to the user saying “Not enough credit”
  + Otherwise, remove the amount from member’s credit and fine to be paid by the member, mark the book as “fine paid” and prompt the user if they want to return the book.

### Functional Requirements

MEM-REQ-1: The System **must** allow the members to borrow books that aren’t already borrowed by other members.

MEM-REQ-2: The System **must** allow members to add credit to their accounts.

MEM-REQ-3: The System **must** let members pay the fines originating from borrowing books by deducting them from the credit they add to their accounts.

MEM-REQ-4: The System **must** allow members to return books to the library as long as their fines have been paid

MEM-REQ-5: The System **could** allow the members to renew the borrowing periods of the books (this would require the addition of borrowing dates later on in the system)

## Librarian Functionality

### Description and Priority

This feature incorporates the administrative-like behavior of a librarian. The librarian can view books and other users and remove these books and/or users as they please. The librarian’s most important activity is registering new members or authors to the system. The use-cases associated with this functionality are the orange use-cases in addition to one of the author use-cases “Remove Book” in **Appendix Figure 1: Use-Case Diagram**. This feature isn’t as important and therefore its priority is medium as the system administrator could perform these functionalities manually if they so please, albeit it might be a bit tedious.

### Stimulus/Response Sequences

The librarian is the only type of user that can create new accounts for other users, but they can’t create new librarian accounts, this is kept for the system administrator (not included in this project).

The system responds by alerting the librarian if they forgot to enter one of the details of the user, rejecting the new user if their username already exists in the system or successfully adding a new user to the system if no errors happened. (**Appendix Figure 8: Register User Activity Diagram**)

The librarian can remove a user by selecting them in a list. Removing a user is allowed by the system if the following conditions are met:

* If the user is a member, his fine must be equal to 0 and has returned all books.
* If the user is an author, all his books must not be borrowed.

Removing a member doesn’t affect anything other than the member’s table in the database, but removing an author also removes their books with them. (**Appendix Figure 7: Remove User Activity Diagram**)

### Functional Requirements

LIB-REQ-1: The System **must** allow librarians to create new accounts for members and/or authors

LIB-REQ-2: The System **must** allow librarians to remove accounts of members and/or authors

LIB-REQ-3: The System **must** allow librarians to remove books.

## Book Author Functionality

### Description and Priority

This feature incorporates an author’s functionality. The use-cases associated with this functionality are the pink colored use-cases in **Appendix Figure 1: Use-Case Diagram.** This functionality includes adding new books or removing books previously added by the author.

This feature is of high priority as it is considered to be half of the consumer functionality, the member functionality being the other half.

### Stimulus/Response Sequences

<List the sequences of user actions and system responses that stimulate the behavior defined for this feature. These will correspond to the dialog elements associated with use cases.>

The author can add their new books by entering the books details into the system. The system then adds this book and automatically assigns it a new book ID. (**Appendix Figure 9: Add Book Activity Diagram**)

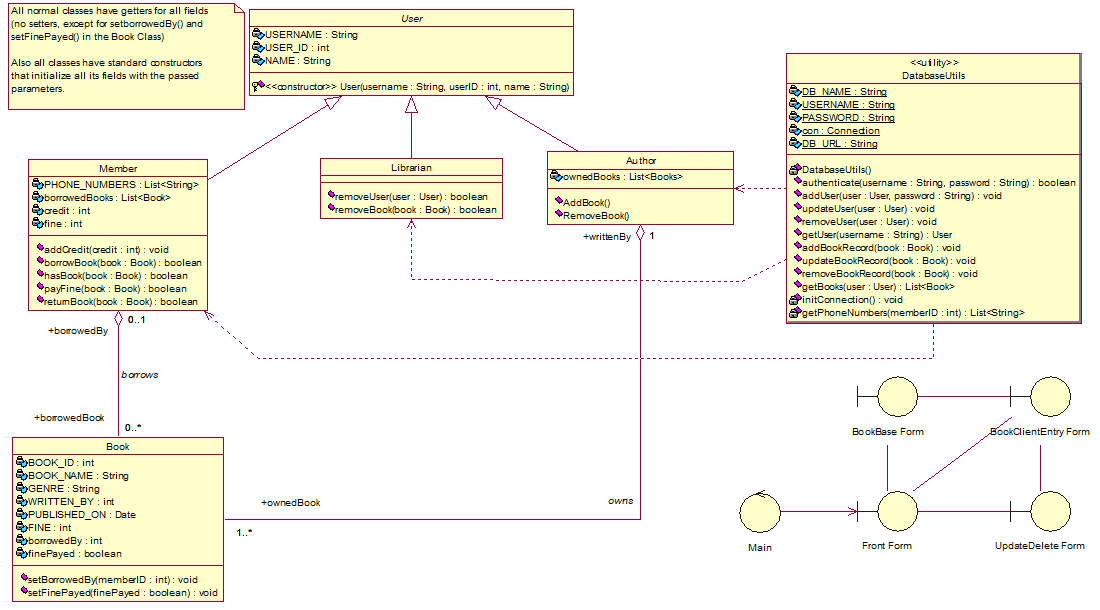
The author can remove their books if these books aren’t borrowed by any user. (**Appendix Figure 10: Remove Book Activity Diagram**)

### Functional Requirements

ATH-REQ-1: The System **must** allow authors to add new books

ATH-REQ-2:The System **must** allow authors to remove their books

# Classes and their Characteristics



**Appendix Figure 2:** Class Diagram

The user classes are incorporated into the systems logic and not as an abstract concept for the developer to keep in mind. This is done to differentiate user privileges based on their class. This has the advantage that no user will get privileges that aren’t given to him, but has the disadvantage that to change the privileges of these user classes, the source code must be edited. For the purposes of this project, we can work with such disadvantages.

The other classes include: A “book” class, a utility class “DatabaseUtils”, a control class “main” and the user interface classes used to communicate between the user and the database. The interfaces will be explained in detail in **section 5.1 User Interfaces** and the rest of the classes will also be explained more in this section.

## Abstract Generalized “User” Class

This user class is abstract, therefore it can’t instantiated as a direct object. This class represents the general user that uses this system. This class is inherited by “Member”, “Librarian” and “Author” classes.

### Class Attributes

* **Username:** A final string representing the unique username of a given user in the “accounts” table in the database.
* **User ID:** A final integer number representing the ID of the user in the database. The “accounts” table doesn’t include this field, rather it’s stored in one of the following tables: “members”, “librarians” or “authors” based on type of the user. Member user IDs will be “1XXXXX” with the X’s being any number (restraining 99,999 members per database), similarly librarian user ID’s will be “2XXXXX” and author user ID’s will be “3XXXXX”
* **Name:** A final string representation of the name of the user, similarly to user ID, the names of the users are stored in the respective table of the type of the user.

### Class Functions

* **Constructor Function:** Protected to only allow calls from subclasses. Accepts the parameters: username, user ID and name. Initializes the respective attributes.
* **Accessor Functions:** getUsername(), getUserID() and getName().
* **Mutator Functions: Has no mutator functions as all its attributes are final.**

## “Member” Class

This class represents the member user. It is a subclass of the class “User”.

### Class Attributes

* Inherits **Username**, **User ID** and **Name** from the super class
* **Phone Numbers:** A final list of strings representing the phone numbers of the member. These phone numbers can be found in the members-phone numbers relationship table. Can be an empty list.
* **Borrowed Books:** A list of book objects representing the books borrowed by the member. The “Book” class is specified more in **section 3.5 “Book” Class**.
* **Credit:** An integer number representation of the amount of money the member has added to the system. On new member creation, credit is initialized to 0.
* **Fine:** An integer representation of the amount the member owes to library. On new member creation, fine is initialized to 0.

### Class Functions

* **Constructor Function:** Accepts the parameters: username, member ID and name, phone numbers, borrowed books, credit, fine. Calls the super’s constructor with the arguments username, member ID and name.
* **Accessor Functions:**
  + **Inherited**: getUsername(), getUserID() and getName().
  + getPhoneNumbers(), getBorrowedBooks(), getCredit(), getFine().
* **Mutator Functions: Has no mutator functions.**
* **Other Functions:**
  + **addCredit(integer)**
    - **Parameters:** An integer number
    - **Returns:** Nothing
    - **Description:** Adds the passed in integer then updates the member’s record in the “members” table. For a visual representation refer to **Appendix Figure 12: Add Credit**
  + **borrowBook(book)**
    - **Parameters:** A book object
    - **Returns:** Boolean (true if no errors happened, false otherwise)
    - **Description:** Borrows the passed in book. I.e. adds this book to the member’s “Borrowed Books” then updates book record in the “books” tables. For a visual representation refer to **Appendix Figure 4: Borrow Book Activity Diagram.**
  + **hasBook(book)**
    - **Parameters:** A book object
    - **Returns:** Boolean
    - **Description:** Returns true if the passed book is in the list “Borrowed Books”, or false if it is not.
  + **payFine(book)**
    - **Parameters:** A book object
    - **Returns:** Boolean (true if no errors happened, false otherwise)
    - **Description:** Pays the book’s fine then updates the member’s record and book’s record in the “members” and “books” tables respectively. For a visual representation refer to **Appendix Figure 5: Pay Fine Activity Diagram.**
  + **returnBook(book)**
    - **Parameters:** A book object
    - **Returns:** Boolean (true if no errors happened, false otherwise)
    - **Description:** Returns the passed in book. I.e. removes the book from the user’s attribute “Borrowed Books”. Then updates book’s record in the “books” table. For a visual representation refer to **Appendix Figure 6: Return Book Activity Diagram.**

## “Librarian” Class

This class represents the librarian user. It is a subclass of the class “User”.

### Class Attributes

* Inherits **Username**, **User ID** and **Name** from the super class

### Class Functions

* **Constructor Function:** Accepts the parameters: username, librarian ID and name. Calls the super’s constructor with the arguments username, librarian ID and name.
* **Accessor Functions:** **Inherited**: getUsername(), getUserID() and getName().
* **Mutator Functions: Has no mutator functions.**
* **Other Functions:**
  + removeUser(user)
    - **Parameters:** A user object
    - **Returns:** Boolean (true if no errors happened, false otherwise)
    - **Description:** Removes the passed in user from the database (if the user is a member remove them from the “members” table, or if the user is an author remove them from the “authors” table). Also remove the account associated with this user from the “accounts” table. . For a visual representation refer to **Appendix Figure 7: Remove User Activity Diagram.**
  + removeBook(book)
    - **Parameters:** A book object
    - **Returns:** Boolean (true if no errors happened, false otherwise)
    - **Description:** Removes the passed in book from the database. For a visual representation refer to **Appendix Figure 10: Remove User Activity Diagram.**

## “Author” Class

This class represents the author user. It is a subclass of the class “User”.

### Class Attributes

* Inherits **Username**, **User ID** and **Name** from the super class
* **Borrowed Books:** A list of book objects representing the books owned by the author. The “Book” class is specified more in **section 3.5 “Book” Class**.

### Class Functions

* **Constructor Function:** Accepts the parameters: username, user ID, name and ownedBooks. Initializes the respective attributes.
* **Accessor Functions:**
  + **Inherited**: getUsername(), getUserID() and getName().
  + getOwnedBooks().
* **Mutator Functions: Has no mutator functions.**
* **Other Functions:**
  + **addBook(book)**
    - **Parameters:** A book object
    - **Returns:** Boolean (true if no errors happened, false otherwise)
    - **Description:** Adds the passed in book to the author’s “Owned Books”, inserts a new record with the book’s details into the “books” table. For a visual representation refer to **Appendix Figure 9: Add Book Activity Diagram.**
  + **addBook(book)**
    - **Parameters:** A book object
    - **Returns:** Boolean (true if no errors happened, false otherwise)
    - **Description:** Removes the passed in book from the author’s “Owned Books”, deletes the book’s record from the “books” table. For a visual representation refer to **Appendix Figure 10: Remove Book Activity Diagram.**

## “Book” Class

This class represents a book that can be borrowed by the user, or can be added or removed by an author or librarian. Related Diagrams: **Appendix Figure 3:** **Book Object State chart Diagram**

### Class Attributes

* **Book ID:** A final integer number representing a book’s unique number.
* **Book Name:** A final string representing the book’s name. Doesn’t have to be unique.
* **Genre:** A final string representing the book’s genre. Doesn’t have to be unique.
* **Written By:** A final integer number representing the author ID of the author that owns this book. This ID can be used to call (or construct) the author of this book.
* **Published On:** A final date object that represents the date on which this book was published
* **Fine:** A final integer number that represents the fine that’s added when this book is borrowed.
* **Borrowed By**: An integer number representing the member ID of the member that has borrowed this book. This ID can be used to call (or construct) the member that borrowed this book. If this attribute is 0, then this book isn’t borrowed by any member.
* **Fine Payed:** A Boolean variable that represents the status of the fine of this book. If it is true, that means the member that borrowed this book has paid its fine or the book hasn’t been borrowed. If it’s false it means that the book has been borrowed but the fine hasn’t been paid.

### Class Functions

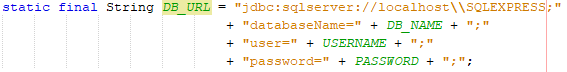
* **Constructor Function:** Parameters: Book ID, Book Name, Genre, Written By, Published On, Fine, Borrowed By and Fine Payed. Initializes the book’s attributes with the passed in parameters.
* **Accessor Functions:** getBookID(), getBookName(), getGenre(), getWrittenBy(), getPublishedOn(), getFine(), getBorrowedBy(), getFinePayed().
* **Mutator Functions:** setBorrowedBy(), setFinePayed().

## “DatabaseUtils” Utility Class

This utility class contains only static attributes and functions. This class is used to communicate between the logic of this program and the database using the JDBC API.

### Class Attributes

* **DB\_NAME:** A static final string representing the database’s name. In our case, the database we will be using will be called “LibraryDatabase”
* **USERNAME:** A static final string representing the username used to log in into SQLSERVER. In our case, the username is “sa”
* **PASSWORD:** A static final string representing the password used to log in into SQLSERVER. In our case, the password is “12345”
* **DB\_URL:** A static final string combining the previous attributes into one string to start a connection with the database. It is a fixed string with variables:



* **Con:** A static Connection object holding the connection to the database. It is initialized and re-assigned with the initConnection() function.

### Class Functions

* **Constructor Function: This class’s construction function is empty and empty, as no object can be instantiated out of a utility class.**
* **Accessor Functions: Has no accessor functions.**
* **Mutator Functions: Has no mutator functions.**
* **Other Functions:**
  + **authenticate(username, password)**
    - **Parameters:** A string Username, and a string password.
    - **Returns:** Boolean
    - **Description:** Checks if the passed username and password corresponds to a row in the “accounts table”
  + **addUser(user, password)**
    - **Parameters:** A user object, and a string password.
    - **Returns:** Nothing
    - **Description:** Inserts a user record in the appropriate table according to the user type. Then adds a new account record with the user’s username and passed password into “accounts” table.
  + **updateUser(user)**
    - **Parameters:** A user object
    - **Returns:** Nothing
    - **Description:** Updates the passed in user in the appropriate table.
  + **removeUser(user)**
    - **Parameters:** A user object
    - **Returns:** Nothing
    - **Description:** Removes the user from the appropriate table. Then removes the account record with the user’s username from the “accounts” table.
  + **getUser(username)**
    - **Parameters:** A string username
    - **Returns:** A user object
    - **Description:** Searches the three tables, if a record was found with the passed username, returns an object of the appropriate user sub-class.
  + **addBookRecord(book)**
    - **Parameters:** A book object
    - **Returns:** Nothing
    - **Description:** Inserts a new book record representing the passed in book object.
  + **updateBookRecord(book)**
    - **Parameters:** A book object
    - **Returns:** Nothing
    - **Description:** Updates the book record corresponding to the passed in book.
  + **removeBookRecord(book)**
    - **Parameters:** A book object
    - **Returns:** Nothing
    - **Description:** Removes the book record corresponding to the passed in book.
  + **initConnection()**
    - **Parameters:** Nothing
    - **Returns:** Nothing
    - **Description:** Initiates the static attribute “con”. This method should be used before using any of the functions of this utility class.
  + **getPhoneNumbers(memberID)**
    - **Parameters:** An integer memberID
    - **Returns:** A list of strings representing phone numbers.
    - **Description:** Gets the phone numbers corresponding to the user with the memberID.
  + **getBooks(user)**
    - **Parameters:** A user object
    - **Returns:** A list of book objects
    - **Description:** If the passed in user is a member, returns all books the user can borrow, or if the passed in user is a librarian, return all books.

# External Interface Requirements

## User Interfaces

### Login Screen

### Main Screen

### New Book Screen

### Registration Screen

## Software Interfaces

The only external library being used is JDBC to communicate with the database management system Microsoft SQL Server 12 used in this project.

# Other Nonfunctional Requirements

## Performance Requirements

The efficiency of this program could sometimes be redundant as it updates book’s/the user’s details.

The borrowing date “could” feature that we mentioned in MEM-REQ-5 could be implemented using a stack to boost performance.

## Safety Requirements

This system was crafted in such a way that data removal can only be done by librarians (or the actual developers of the system) to ensure that no accidental data loss could occur. That being said, some unplanned errors could result in data deletions.

## Security Requirements

No security measures were taken till now, but it is planned to implement a hashing function to encrypt passwords.

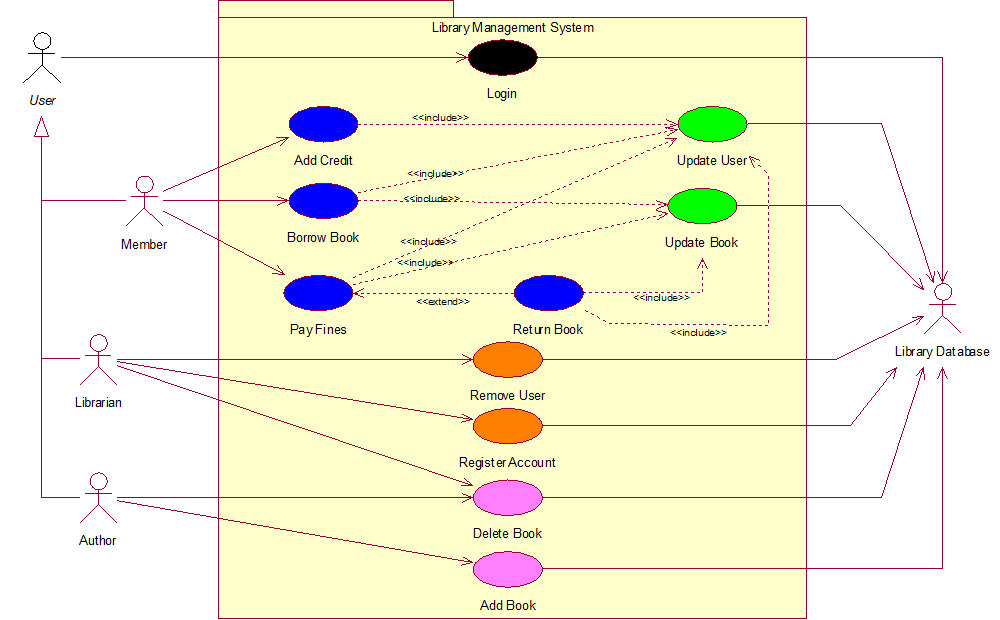
# Other Requirements

## Database Requirements

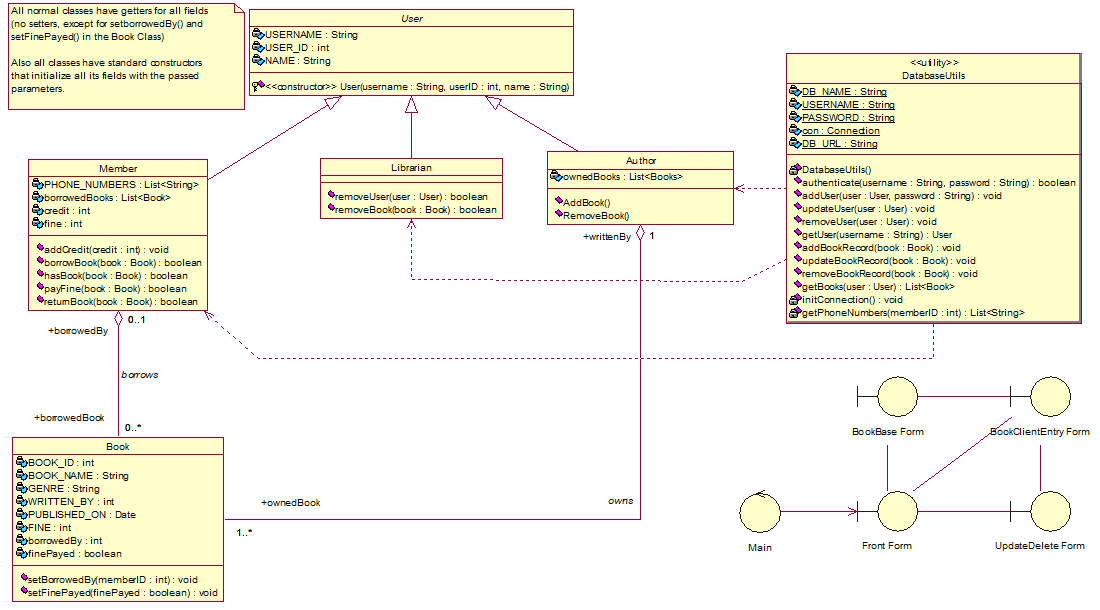
Appendix A: Glossary

* **User**:The generalized person that uses the system.
* **Member**:A person that intends to use the system to keep track of their borrowed books and paying the fines generated by borrowing said books.
* **Librarian**:A person that acts as the general administrator that can create accounts for new members or authors or delete already existing members, authors or books if like.
* **Author**:For the purposes of this project, an author is a person that has written a book, publishes it and owns it fully. This person can keep his books in the library for members to borrow and read. This is the most different term, as in reality the process that results in the library acquiring the book is quite rigorous and complicated, and hence it was dumbed down and simplified for the purposes of this project.
* **Book**: An object that is written by an author and can be borrowed by a member. Its content can’t be read in the application and is only an abstract representation of a book in reality.

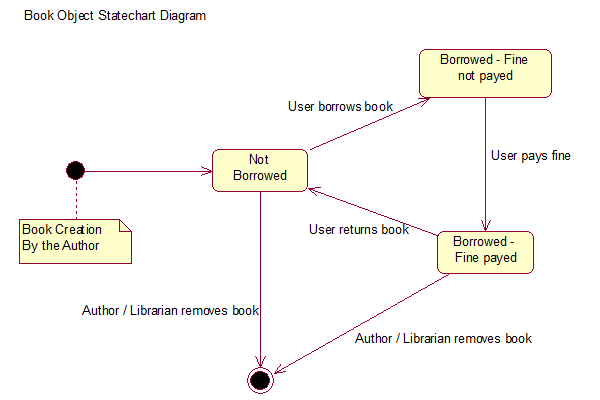
Appendix B: Analysis Models



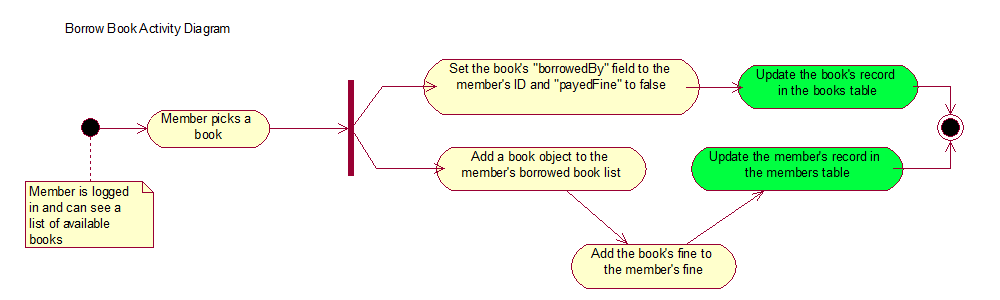
**Appendix Figure 1:** Use Case Diagram



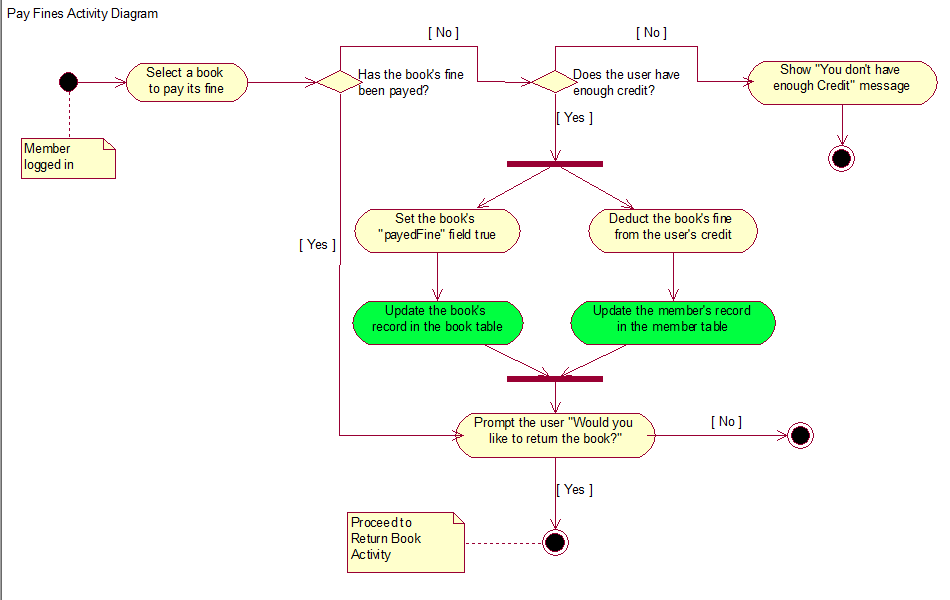
**Appendix Figure 2:** Class Diagram



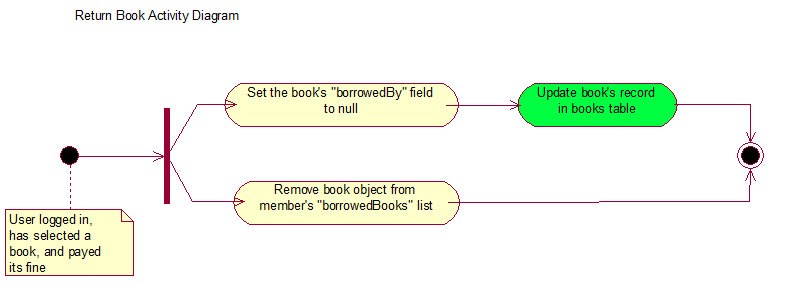
**Appendix Figure 3:** Book Object State chart Diagram



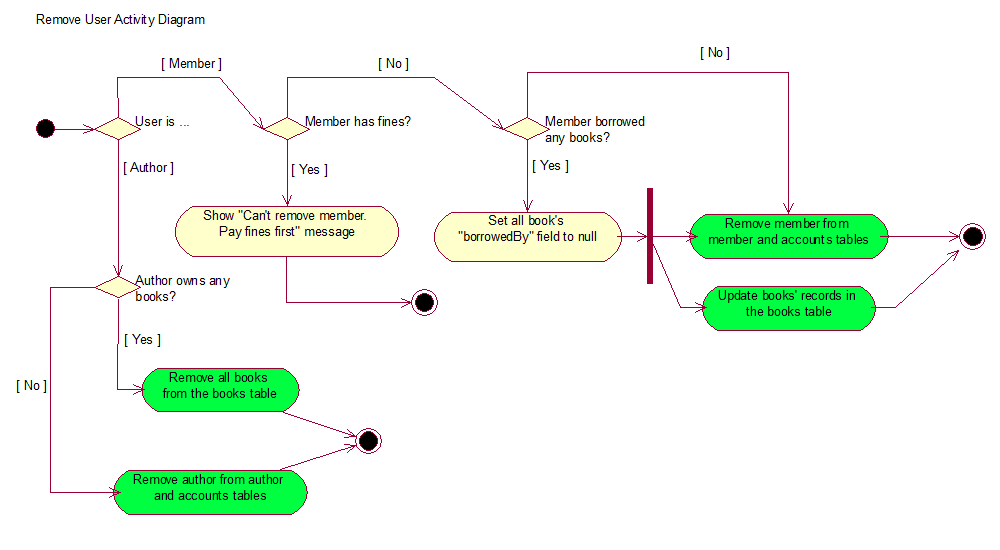
**Appendix Figure 4:** Borrow Book Activity Diagram.



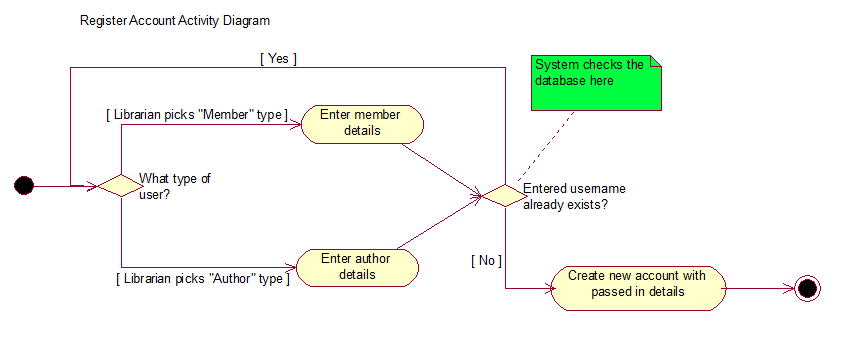
**Appendix Figure 5:** Pay Fines Activity Diagram



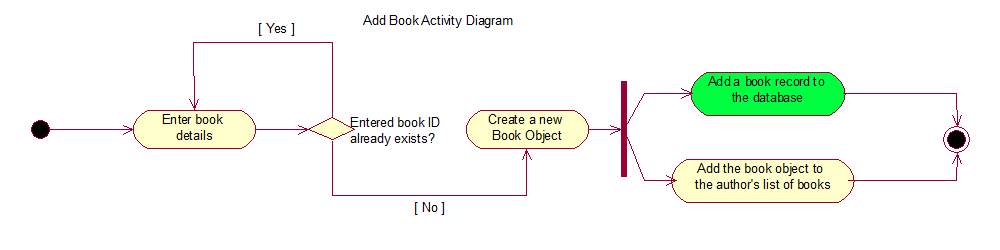
**Appendix Figure 6:** Return Book Activity Diagram



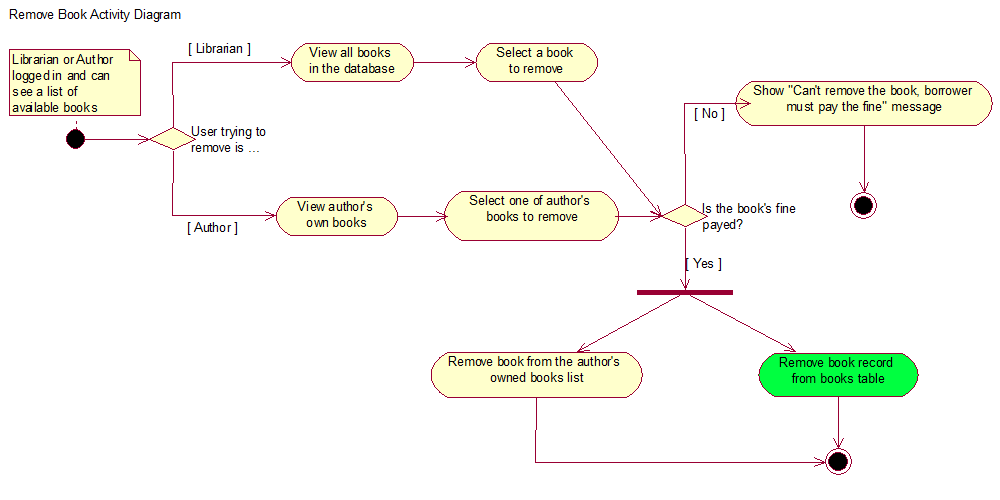
**Appendix Figure 7:** Remove User Activity Diagram



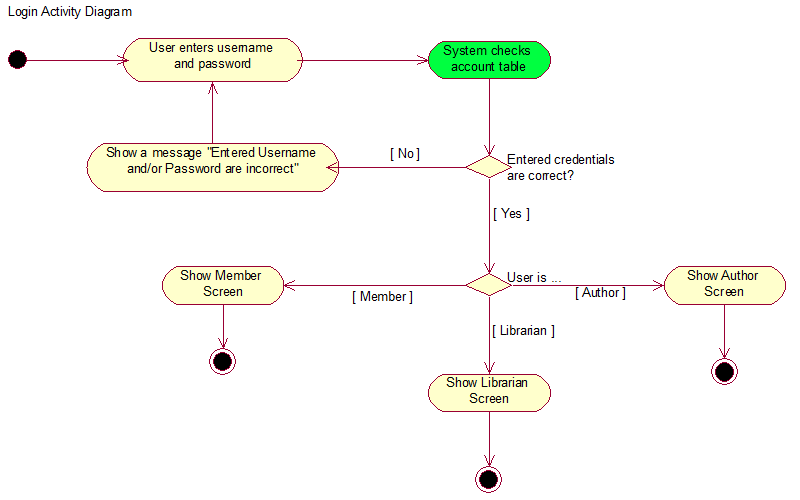
**Appendix Figure 8:** Register Account Activity Diagram



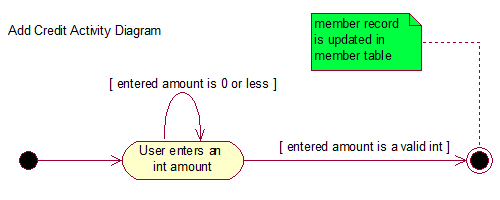
**Appendix Figure 9:** Add Book Activity Diagram



**Appendix Figure 10:** Remove Book Activity Diagram



**Appendix Figure 11:** Logic Activity Diagram



**Appendix Figure 12:** Add Credit Activity Diagram

Appendix C: Issues List

< This is a dynamic list of the open requirements issues that remain to be resolved, including TBDs, pending decisions, information that is needed, conflicts awaiting resolution, and the like.>

**TBD**