**CST2550 – Software Engineering Management and Development**

**Student: Gheorghe-Florin Hasna**

**ID: M00910464**

**Module Leader: Ahmed Eissa**

Coursework1: Library System Implementation

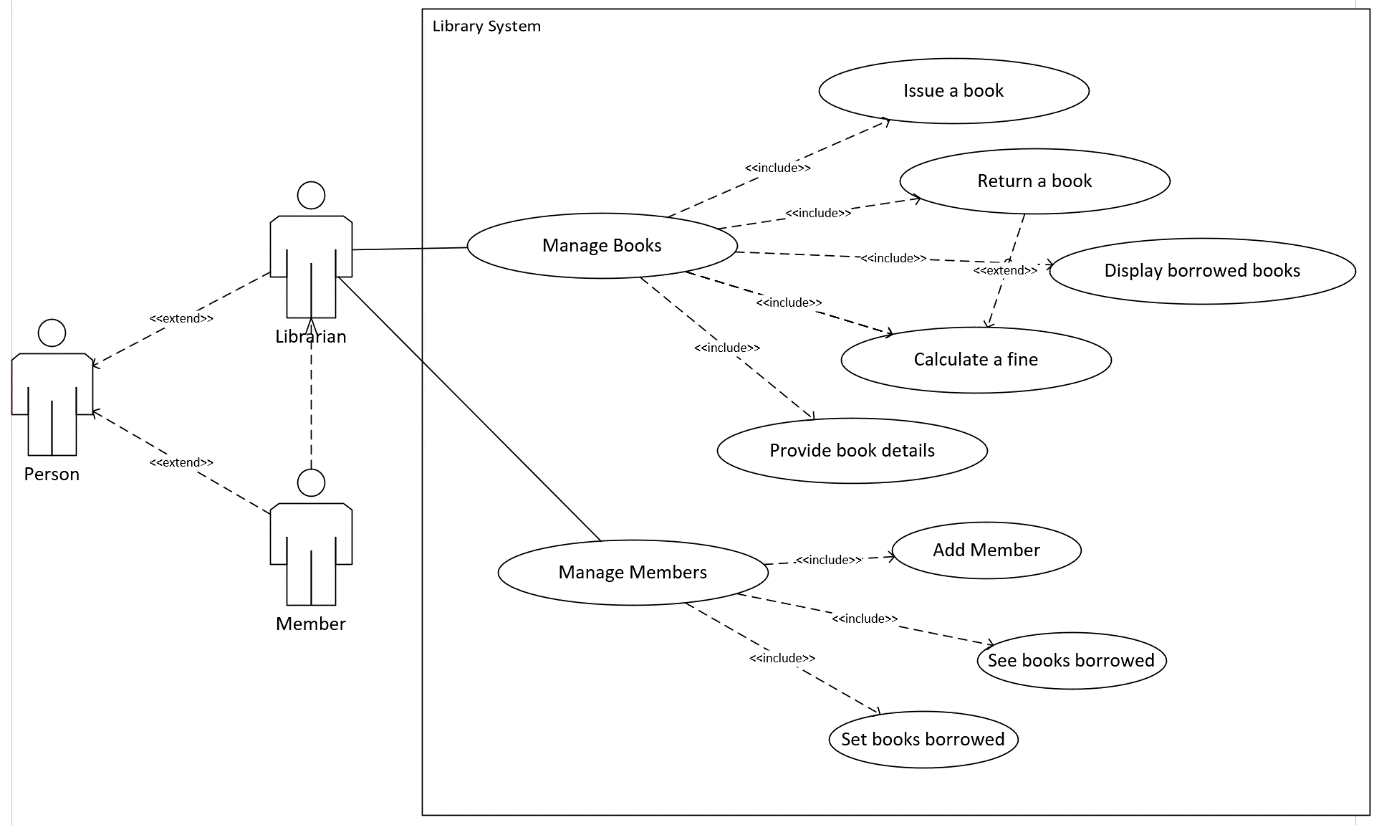
**Introduction**

The project’s goal is to deliver a functional library system that will allow a library to keep track of books that are borrowed by members and manage its members. To achieve this goal, in the implementation of the program, different strategies and libraries and implementation strategies are used to make the librarian able to add new members, issue and return books to members, retrieve the books borrowed by a member, and calculate the outstanding fines. This document presents design aspects with UML diagrams, implementation strategies, testing methods, and a step-by-step software explanation.

**Design**

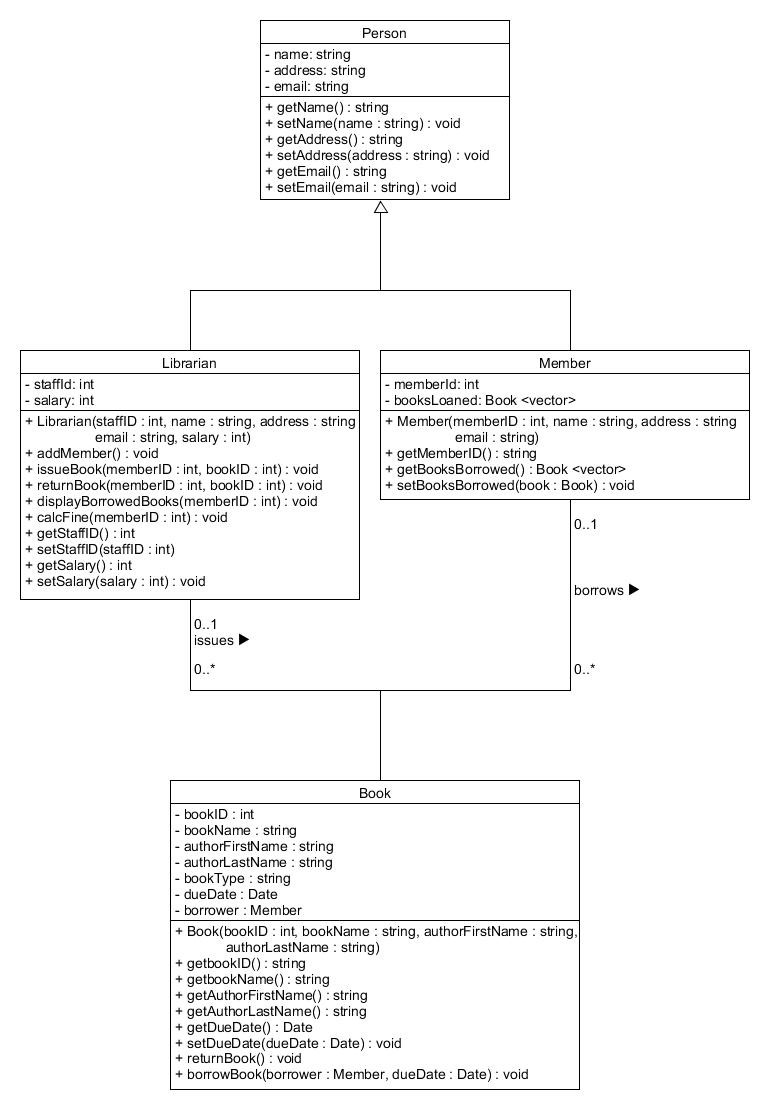
In the development of the library system, three types of diagrams have been used: use case diagrams, class diagrams, and activity diagrams. The code follows the class diagram that was given and makes no alterations to it. In the initial stage of development, a use case and activity diagram were designed for a better understanding of the development process.

Use case diagram:



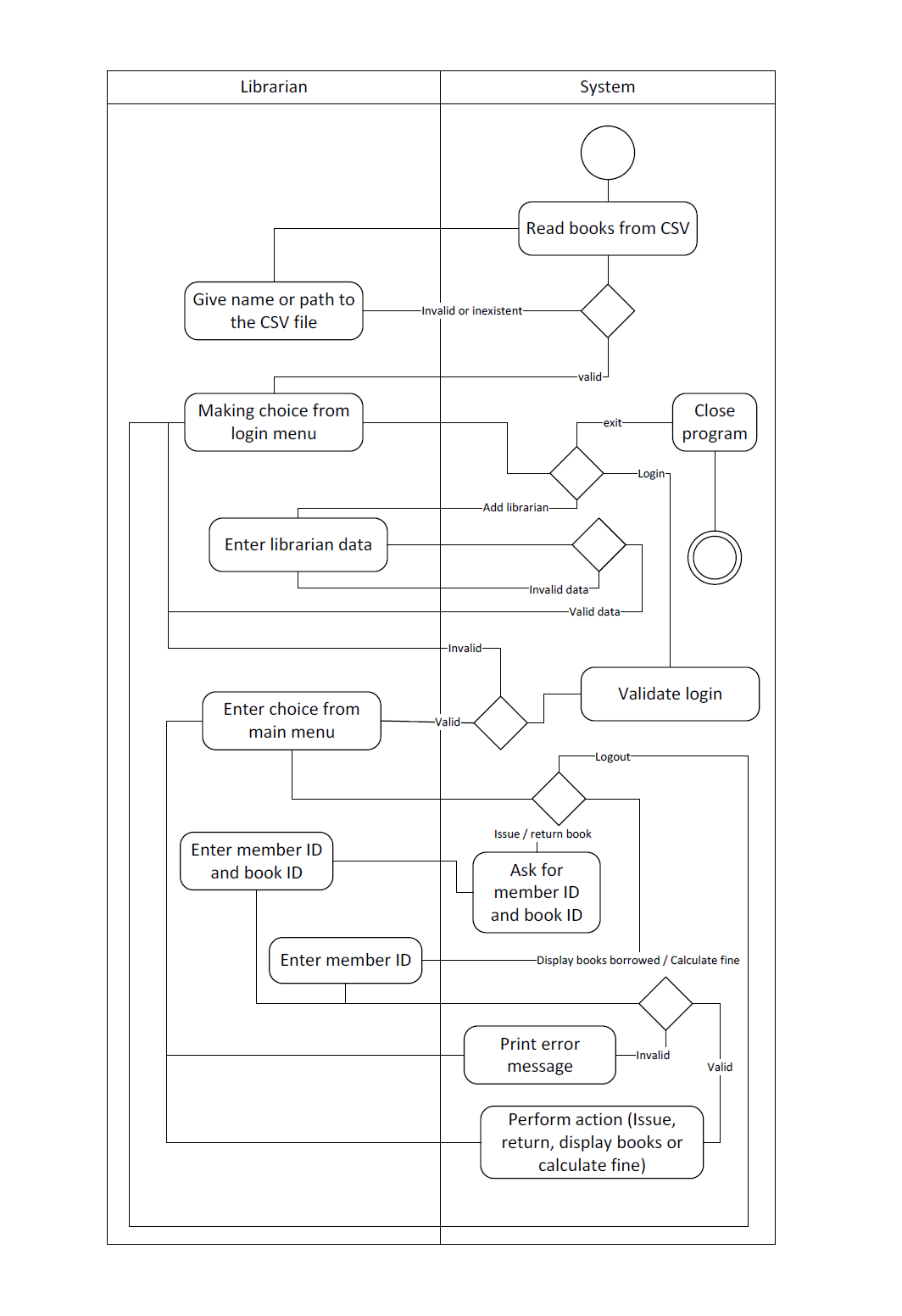
The diagram shows each use case that the program must have, it is first split into two categories, where the system is used to manage books and members. Further, the categories have subcategories, representing the main use of the system. The member is dependent on the librarian as only the librarian can access the system. The member makes the request, and the librarian is processing it.

Class diagram - provided:



The class diagram presents the structure of a relevant class, private properties, which are inaccessible in the external code, and public setter and getter methods and relevant operations that the object will be able to perform, accessible everywhere. This diagram can be read starting with the class Person, followed by classes Member and Librarian which are inherited from the class Person. The class Book holds details of a book and is being handled by the librarian who can issue the books to a member.

Activity diagram:



The activity diagram presents from the beginning, the course of actions that is performed by the system or librarian to operate, that is requested initially by the member. The member is not included in this diagram due to its lack of accessibility to the system, only the librarian can interact with the system.

**Implementation**

First and most important, ensuring that the task is understood represents one of the most vital elements of the process. Misunderstanding the task can and will lead to failure in the implementation and presentation of the program. Reading the coursework brief multiple times, and splitting the problem into smaller problems are some of the strategies used to achieve the correct implementation. Having a clear task in mind not only made everything easier and increased awareness of potential mistakes but also assisted in planning the implementation better. To implement all functionalities, several different standard libraries were imported, such as:

* “stdexcept” - for throw, try and catch exceptions;
* “regex” - for regular expressions;
* “ctime” - to import and process the current time of the system;
* “fstream” and “sstream” - to read in the CSV file

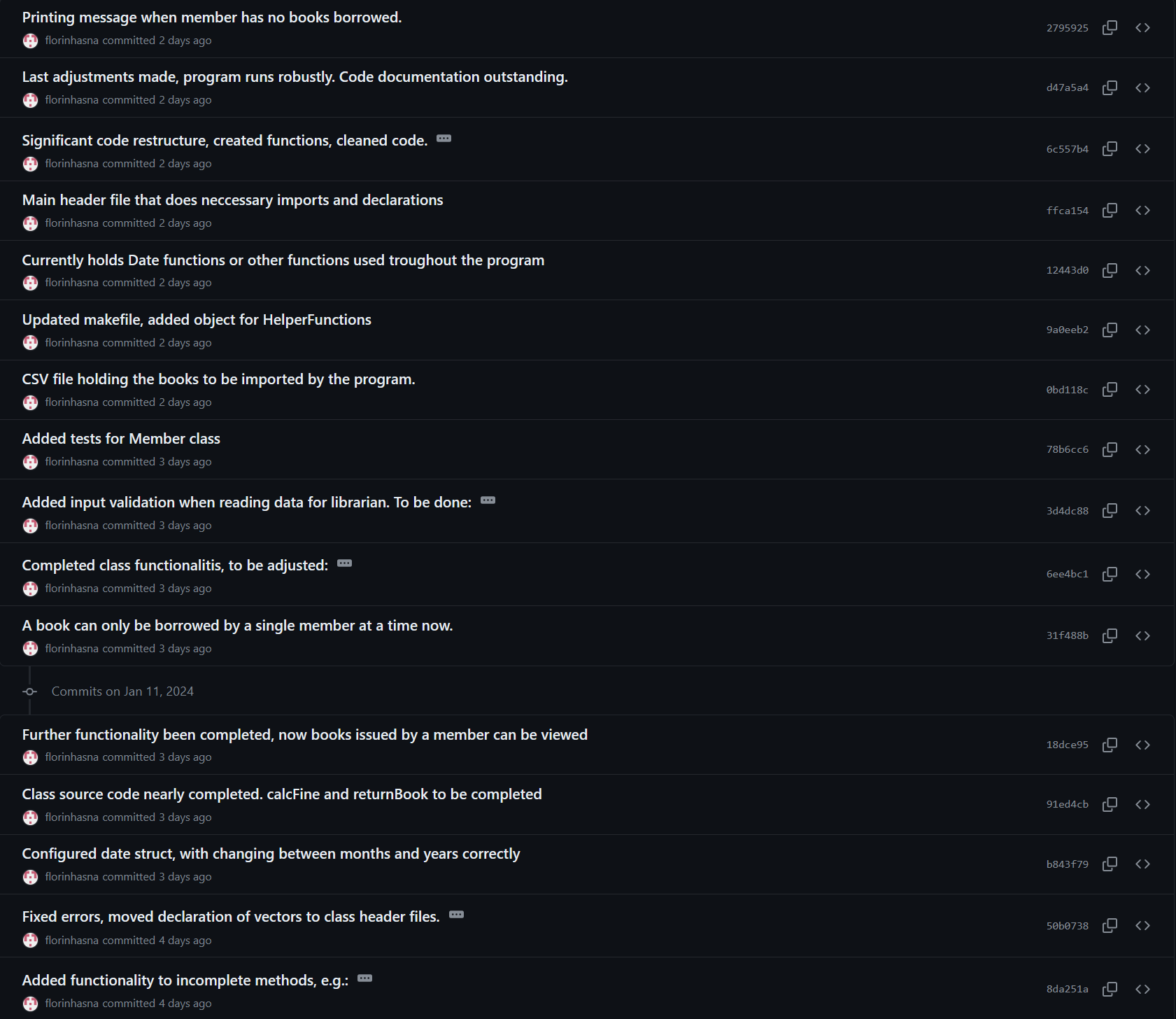
Also, online resources have been utilized, such as:

* “Big C++” by Cay Horstmann
* [How to get the current time and date in C++? - Stack Overflow](https://stackoverflow.com/questions/997946/how-to-get-current-time-and-date-in-c)
* [parsing - How can I read and parse CSV files in C++? - Stack Overflow](https://stackoverflow.com/questions/1120140/how-can-i-read-and-parse-csv-files-in-c)
* [C++ Programming Language - GeeksforGeeks](https://www.geeksforgeeks.org/c-plus-plus/)

Secondly, to reach the capability of running the program, the build tool “Make” was used to assist in building, cleaning, and compiling the project. The makefile was written to build each source code file and their headers in object files and connect them with the main source code file “LibrarySystem.cpp”. Successful execution of the makefile creates two .exe programs: “program” and “class\_tests”. The file “program” is running the main program and “class\_tests” is returning the results after executing the test cases. Files used and compiled using make:

* HelperFunctions.cpp and HelperFunctions.h (main header importing necessary libraries and some additional helper functions);
* Book. cpp and Book.h (Book class and its header);
* Person. cpp and Person.h (Person class, with derived classes Member and Librarian and its header);
* LibrarySystem.cpp (main source code file, used to control the classes);
* ClassTests.cpp and catch.hpp (source code to run tests on classes and library needed for unit testing).

To keep evidence of changes and store the program in a safer environment, GitHub was a necessity. GitHub was used primarily as a backup of the program’s source code and for keeping evidence of every change made throughout the implementation. Having the source code in the GitHub repository is mostly important for having a previous version of the program which a developer can go back to in critical situations.



**Testing approach**

The main adopted testing approach is unit testing. To assist with testing, an external library has been imported from GitHub: <https://github.com/catchorg/Catch2/tree/v2.x>. The testing framework was used to create assertions to test the class: Book, Person, Member, and Librarian. In testing the classes, the main tested methods are the setter and getter methods. The method to issue and return books “setBooksLoaned” in the Member class has also been tested using test assertions by checking the size of the vector returned.

Additionally, the code was constantly checked, and its behaviors were observed with every step of implementation.

To test the understanding of the messages, the program was handed to external individuals to execute it. By observing the process, the necessary changes were made to increase the intuition of messages.

**Software step-by-step explanation**

The program is working based on data inputs given. Is built to run robustly, and is not supposed to fail when a wrong input is typed. To achieve that, regular expressions were used to limit the input to the accepted format. When the program first runs, it will import data from an external .csv file which holds all the data required for the books the library possesses. Failing to do so, will mean that the file has been renamed or moved and the program handles the situation by requesting the user the new name or path of the file (NOTE: when giving the name or path, must not include the file format, the file format is added in the program). After this action is performed, a menu with three options will be printed. The options are simple, getting the user to perform the desired option. If the user chooses to log in, but no account is created, then the program will redirect the user to create an account, otherwise, it will simply require the staff ID to validate the user’s login. Upon creating a librarian account, the program requires five fields of data and after completion, prints confirmation and the entered details. The details required are, and checked by the program as follows:

* Staff ID is accepted only if is made of four numbers and has not been used before.
* Full name accepted if the user enters full name.
* Address, the program accepts only the first line of address and the postcode which should be typed in capitals (e.g. 35 Hendon Way, NW7 3FG).
* E-mail, any valid e-mail format is acceptable (e.g. [email@domain.com](email@domain.com%20) or [different\_email@domain.co.uk](mailto:different_email@domain.co.uk)).
* Annual salary accepted number is greater than 1,000.

Upon successful registration, the librarian can log in and can manage members and books. The librarian can add members, where member ID, full name, address, and e-mail of the member are required and are processed as mentioned above. The librarian can issue or return books to a member, but the same book cannot be borrowed by two members, a book can be borrowed by one member at a time and the due date for the book to be returned is after three days of the borrowing. Failing to return the book in time, the system will calculate a fee of £1 for every day over the due date. The ability to return the book to the library has also been implemented and when this procedure is being actioned, it will say whether there are any outstanding fines for the member returning the book. Two more functionalities can be accessed by the librarian, which will allow one to see the books that were borrowed by a member, and to calculate the fines. For issuing and returning books, the librarian will need to enter a valid member ID and book ID, whilst for seeing the books borrowed and calculating the fines, the librarian will need to provide a valid member ID only. The program will keep the librarian logged in until logging out, which will return the program to the previous state, where another librarian can log in or another can be created.

**Conclusion**

In conclusion, by using the specified approach and implementation strategies, the final product is running robustly and provides clear instructions to the user which increases usability. It is important to have well-structured and defined UML diagrams to follow from the beginning and not alter during the implementation, a situation that could cause a radical change in the whole agreed implementation plan. Testing the behaviours of classes is vital, as a small unexpected result could significantly affect the desired behaviour. As for personal limitations, I identify incomplete information over using new libraries and understand new concepts as one that impacted the implementation, a limitation which I believe requires more experience to overcome. In future projects, a better focus and better understanding of C++ concepts will be settled in my mind after the completion of these implementations, but I would like to focus more on designing clearer UML diagrams and be more thorough on testing the software.