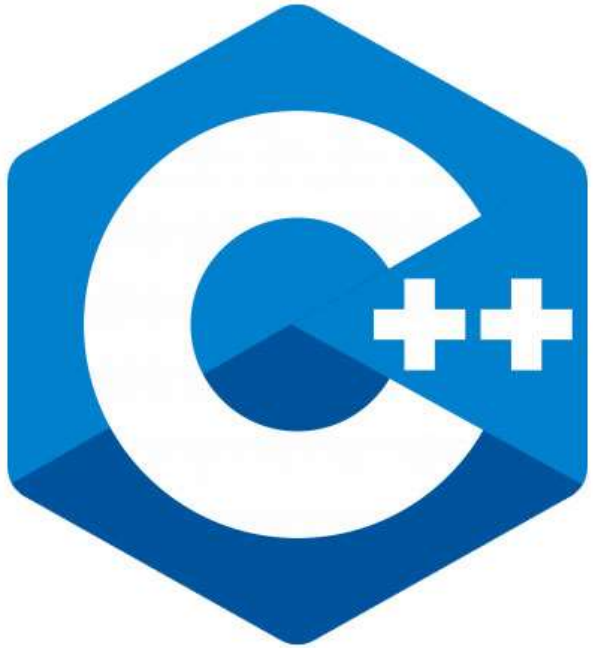
LIBRARY SYSTEM

Fabian Galasel M00860030

Library System and implementation

Introduction

In this presentation, I will be explaining the project, requirements, how I implemented it using my workspace, GitHub, and Cygwin. I am going to be talking about the design of the software, the implementation, and the testing. I will also provide a demonstration of the software and a conclusion to sum it all up.



BRIEF DESCRIPTION OF THE PROJECT

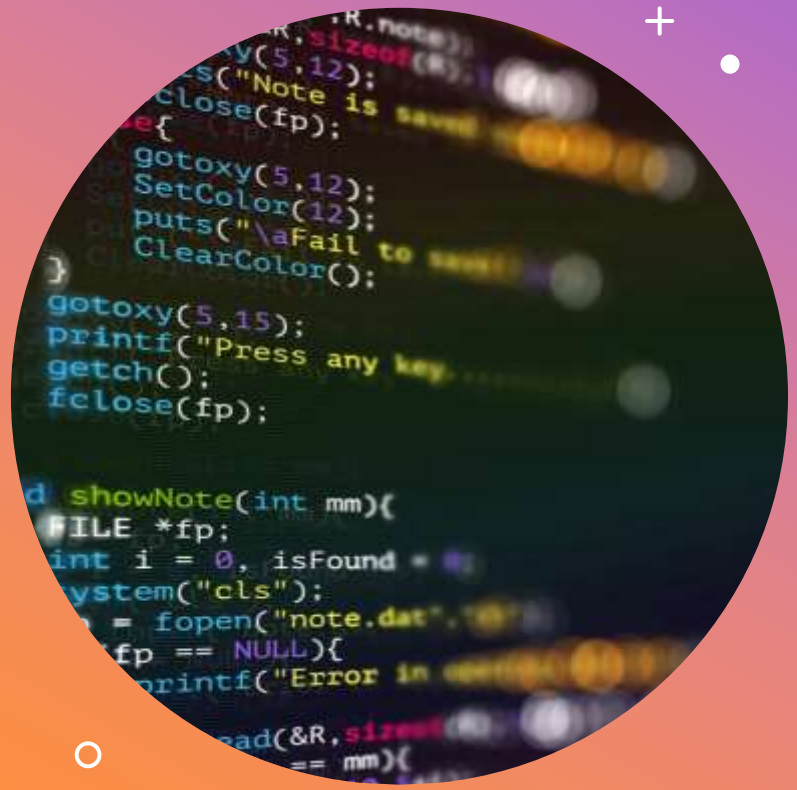
The project is all about a library system, which is used only by the librarian. They should be able to create members, or view pre-existing ones. Once they have selected a member using their ID, they should be able to do the following:

1. Assign books.
2. Return books after 3 days.
3. Calculate fine, if the member did not return the book within 3 days. £1 per day overdue.
4. Display member details.
5. And closing the software.

The software should tell them which books the user borrowed, and their fine.
The software also allows the librarian to select which file the software should read from. Ideally similar format to other files.

Progress and code was regularly uploaded to GitHub for version control. Clear messages

OVERVIEW OF THE PRESENTATION



Design

Implementation

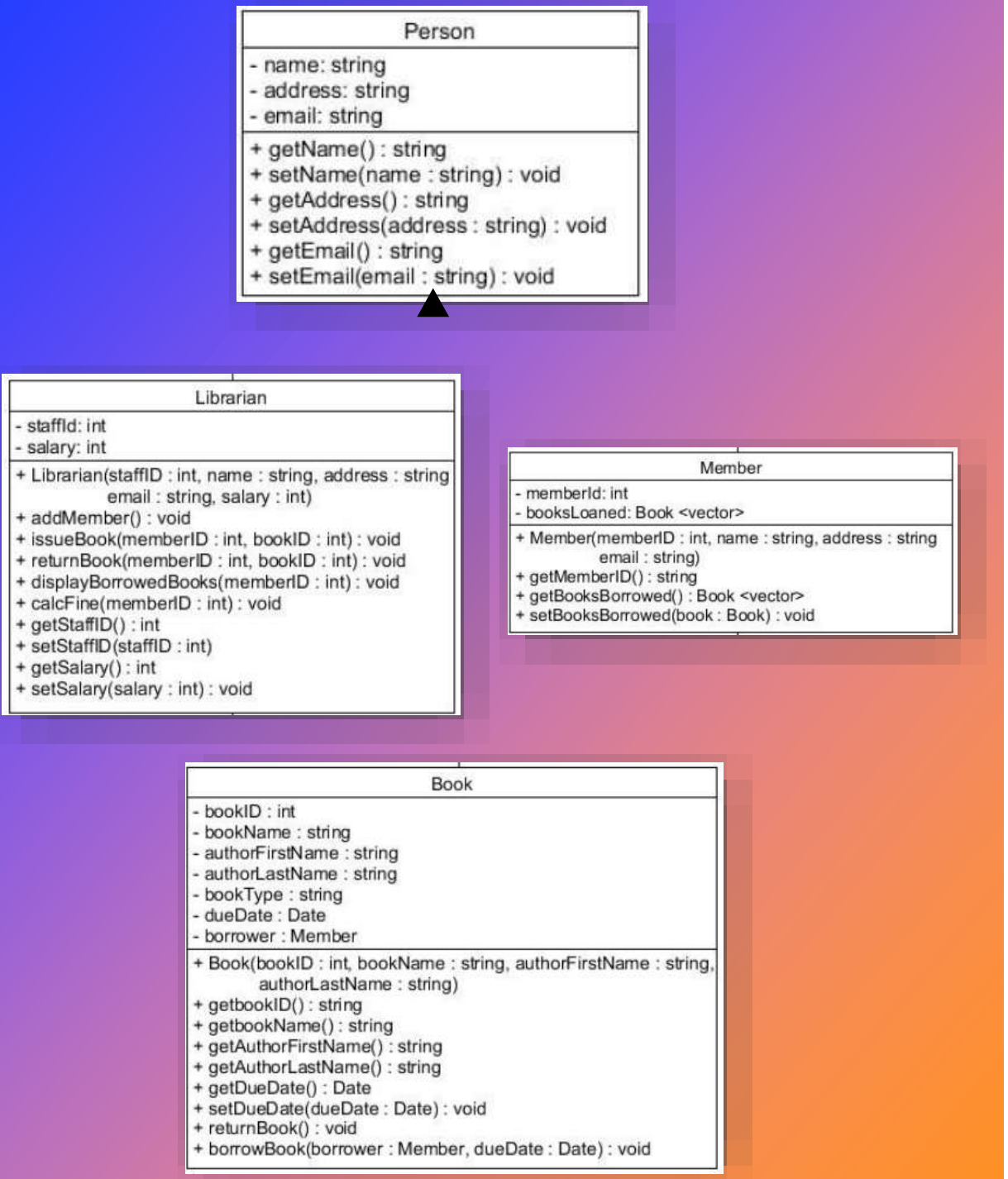
Testing approach

Software demonstration

Conclusion

DESIGN

Structure and design of software



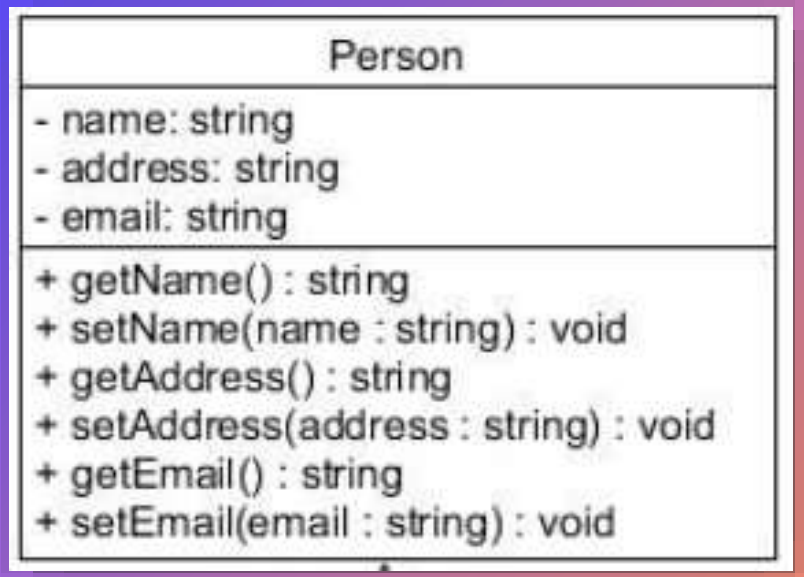
UML Diagrams

The UML diagrams are the structure of the software. This is what I used to make and compile the code.

Each class has its own attributes and methods.

And each class is used for the main library system.

I will provide more details and in the next few slides.



Person Class

The person class serves as a base class for the other classes.

It provides the structure for storing personal information like the name, address and email.

The person class, as is shown in the slide above, is inherited in both the Member Class, as well as in the Librarian Class. So, it is crucial in the code.

Attributes:

name: Represents the person's name.

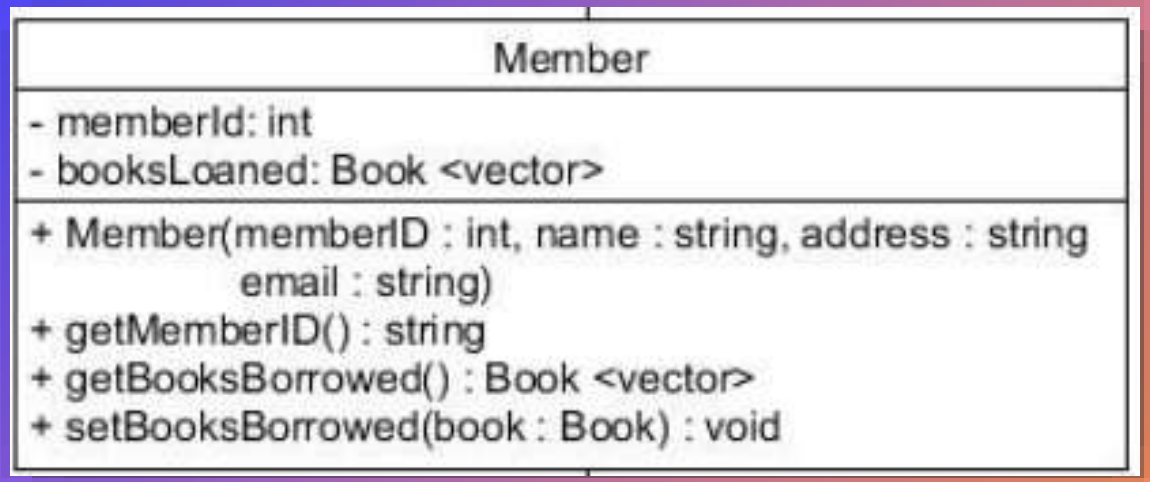
address: Represents the person's address.

email: Represents the person's email address.

Methods:

getName(), getAddress(), getEmail(): Getter methods for retrieving the person's name, address, and email.

setName(const std::string&), setAddress(const std::string&), setEmail(const std::string&): Setter methods for setting the person's name, address, and email.



Member Class

The member class represents the members and inherits from the Person Class. However, it extends the Person class to include more attributes specific to the library members, like the unique member ID, information about the borrowed books, and timestamps used to keep track of the days for issuing and returning the book.

Attributes:

memberID: Represents a unique identifier for the member.

lastIssueTimestamp: Represents the timestamp of the last issued book.

bookID: Represents the ID of the book currently borrowed by the member.

borrowedBookID: Stores the ID of the borrowed book.

issueTimestamp: Stores the timestamp when a book is issued to the member.

Methods:

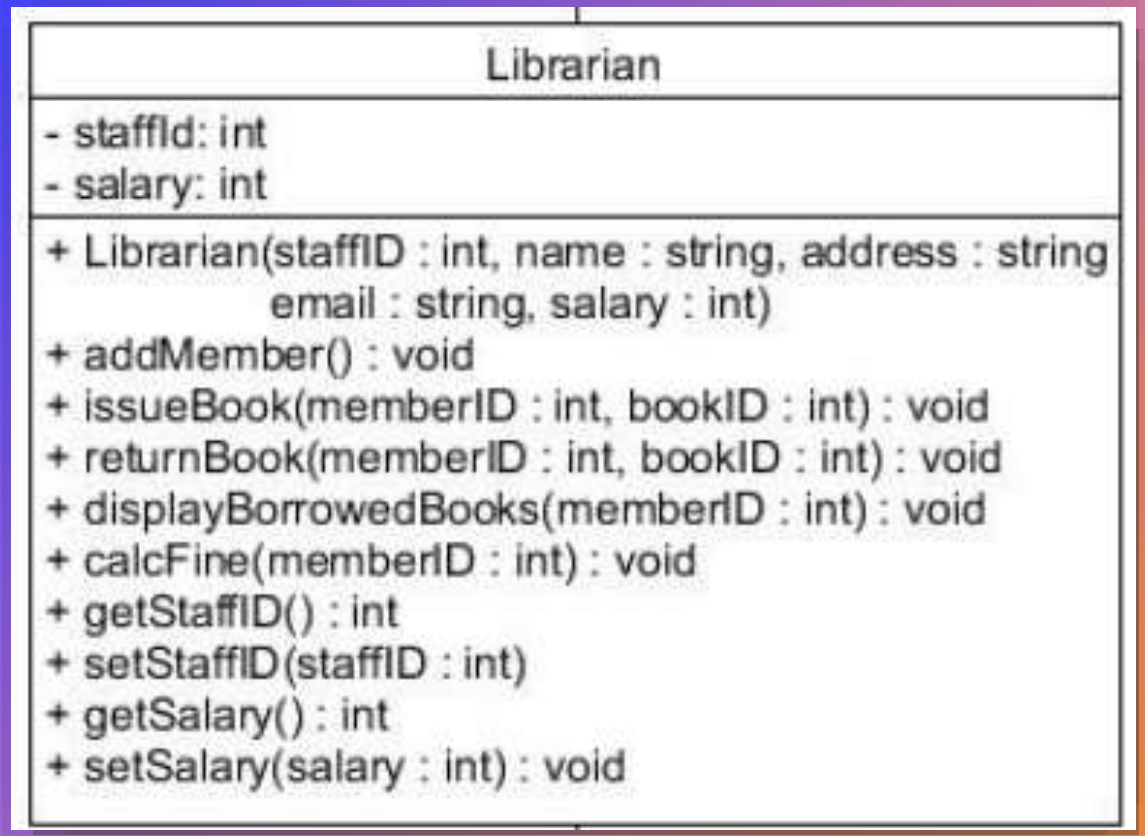
generateRandomID(): Static method to generate a random member ID.

setBooksBorrowed(int): Sets the book ID that the member has borrowed.

displayDetails(): Displays details of the member, including their ID, name, address, email, and borrowed book ID.

setLastIssueTimestamp(): Sets the last issue timestamp when a book is issued to the member.

Librarian Class



The librarian class represents the librarian and inherits from the Person Class as well. It extends the Person Class to include the attributes specific to librarians, like the staff ID, salary, managing members and so on. This class also gives them the management tasks such as adding members, issuing books, returning books and so on.

Attributes:

staffID: Represents the staff ID of the librarian.

salary: Represents the salary of the librarian.

members: A vector to store Member objects.

username, password: Credentials for the librarian to log in.

issueTimestamp: Represents the timestamp of book issuance.

Methods:

addMember(Member&): Adds a new member to the vector of members.

findMemberByID(int): Finds a member by ID in the vector.

issueBook(Member*, int): Issues a book to a member.

returnBook(Member*, int): Returns a book from a member.

displayBorrowedBooks(const Member*): Displays books borrowed by a member.

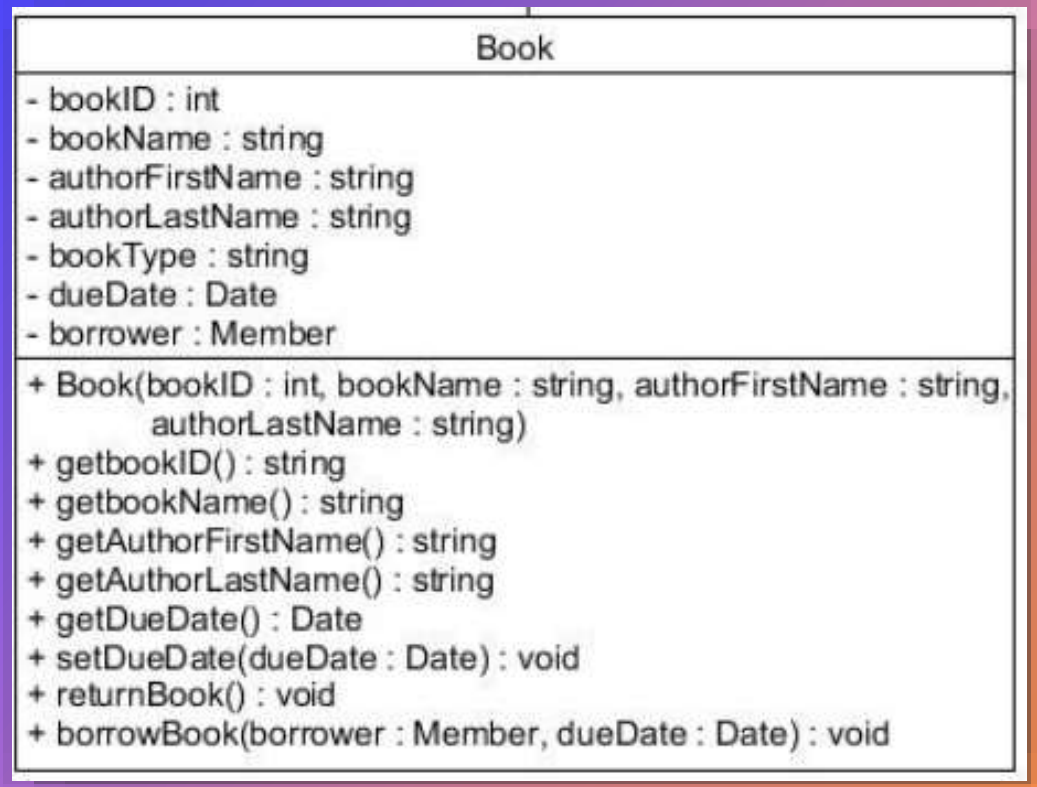
displayMemberDetails(const Member*): Displays details of a member.

validateCredentials(int, const std::string&): Validates provided staff ID and password.

getStaffID(), setStaffID(int): Getter and setter methods for staff ID.

getSalary(), setSalary(int): Getter and setter methods for salary.

Book Class



The book class represents the books. It provides the structure to store information about the books, including their unique IDs, the name of the book, page count, author details, and the book type. This class is used to create book objects when reading information from the .CSV file.

Attributes:

bookID: Represents a unique identifier for the book.

bookName: Represents the name of the book.

pageCount: Represents the number of pages in the book.

authorFirstName, authorLastName: Represents the first and last names of the book's author.

bookType: Represents the type or genre of the book.

Methods:

getBookID(), getBookName(), getPageCount(), getAuthorFirstName(),

getAuthorLastName(), getBookType(): Getter methods for retrieving book attributes.

IMPLEMENTATION

Implementation approach ,Makefile, Version Control, Repository, Commits

My implementation approach (Part 1)

Understanding the requirements

- My initial step was to understand the design requirements, which include modelling a library management system with librarians, members, and books.
- Based on the requirements, I identified the key classes are identified (Person, Member, Book, Librarian) with attributes and methods that represent the functionalities in the software.

Class implementation

- The design follows object-oriented principles, using classes to encapsulate related attributes and behaviours.
- Each class is implemented with its specific attributes and methods, representing the properties and behaviours associated with the corresponding entity (for example, the librarian managing members).

Functionality implementation

- The Librarian class is a central component responsible for library management tasks. It includes methods to add members, issue and return books, and validate credentials. The class interacts with Member objects and manages book-related operations.
- The Member class represents library members, storing information about borrowed books and timestamps. It includes methods for setting books as borrowed and displaying member details.
- The Book class represents books and includes attributes for book details. It is used to create Book objects when reading information from a CSV file.

My implementation approach (Part 2)

File reading

- CSV File Reading: The program includes a function *(readBooksFromCSV)* to read book information from a CSV file. This demonstrates the ability to integrate external data into the system.

User Interface

- The main function serves as the entry point to the program, providing a simple console-based user interface for a librarian to log in and interact with the system. It uses the Librarian class to perform library management tasks.

Creating Makefile

- The Makefile is used to create the build process of a software project. It includes instructions for compiling and linking source code files, classes, and producing the executable program.

My implementation approach (Part 3)

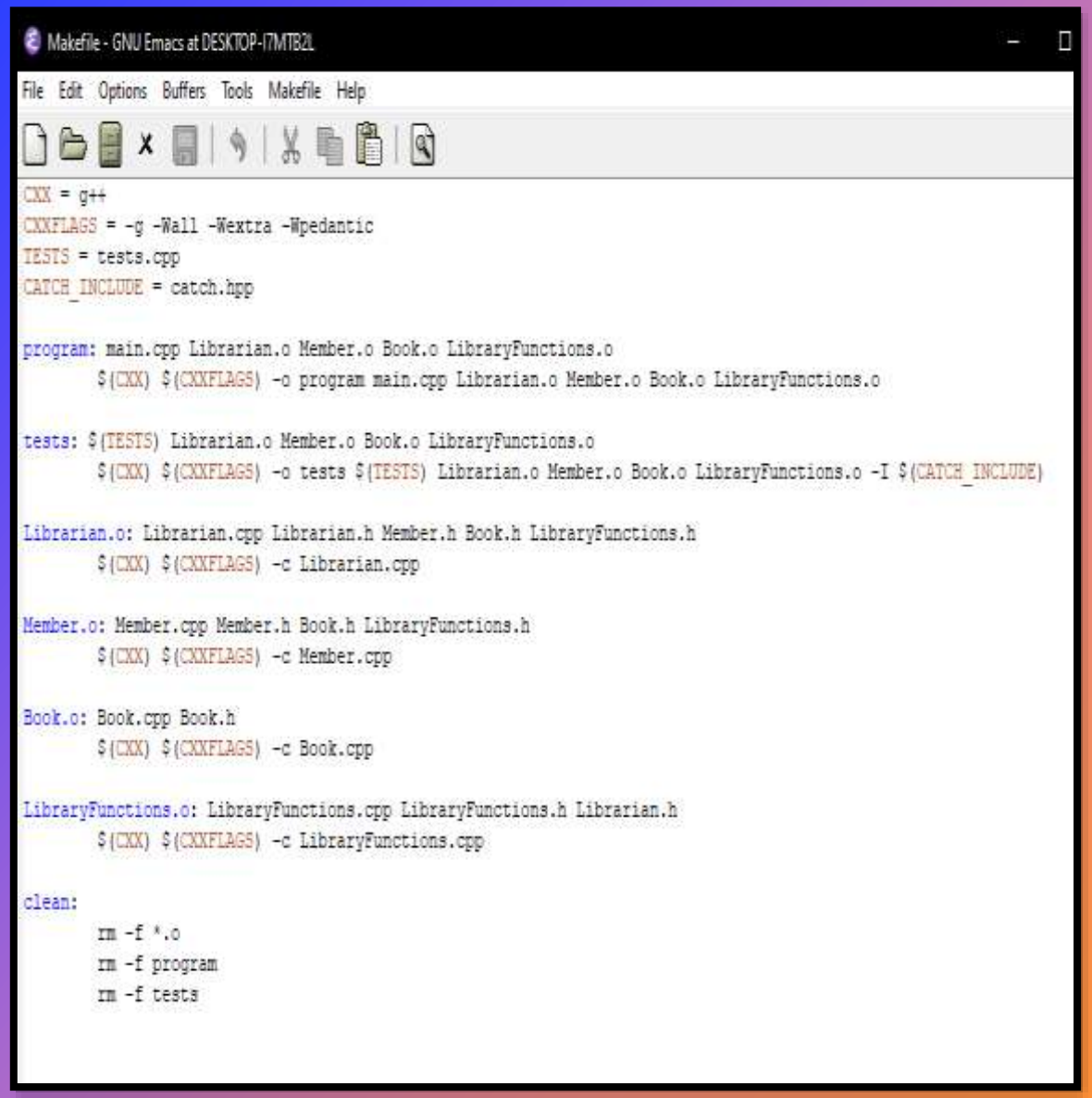
Testing

- Catch2 Tests: To ensure the correctness of the implemented functionalities, Catch2 tests are added. The tests cover specific scenarios, such as adding a member and displaying their details, validating that the implemented features work as expected.
- However, I did not get the chance to finish off the test cases, so the Catch2 Tests aren’t complete.

Version Control/Repository

- I'll explain it briefly here, but more details in slide 16
- I’ve made regular commits to the repository where all my work is, which means that the work can be checked, tracked, and preview any changes made.
- It allows history tracking, backups, recovery of files, and so on.

Creation of Makefile



The Makefile is used for creating the build process of the software. It contains a set of instructions that specify how the source code should be compiled. There are a few components:

- CXX = g++: Defines the compiler to be used (in this case, g++ for the GNU Compiler Collection).
- CXXFLAGS = -std=c++11: Specifies the compiler flags, setting the C++ standard to C++11.
- Program: Specifies the target named program, which depends on source files (main.cpp) and object files (Librarian.o, Member.o, Book.o, LibraryFunctions.o).
- Tests: Specifies the target named tests, which depends on the test source file (tests.cpp) and object files.
- Targets like Librarian.o, Member.o, Book.o, and LibraryFunctions.o define rules for compiling individual C++ source files into object files.

Version control

- I will provide screenshots of the GitHub repository which clearly shows all the commits and commit messages made.
- Version control is incredibly useful because it maintains a complete history of changes made to a project.
- This means that developers can review the project's history to understand how the code evolved and progressed over time, who made specific changes, and why those changes were made.
- This history is crucial for debugging and understanding the project's development trajectory and goal.

Timeline (with links to GitHub repository)

January 6, 2024

Implemented logic to issuing books to members, and made the program more user-friendly. Having trouble reading from .csv file

FINALLY made software read properly from any .CSV file (with similar format). It also displays the first 20 books when first opening the file

January 7, 2024

added error messages to opening .CSV, and it displays up to 300 books, if the file is opened successfully. Also added logic to issuing books and returning them (still working on it)

January 8, 2024

debugged & fixed errors, improved usability and readability. Finished working on issuing books. Almost finished on returning and calculating fines for each member.

January 10, 2024

added the executable file and tried creating Makefile

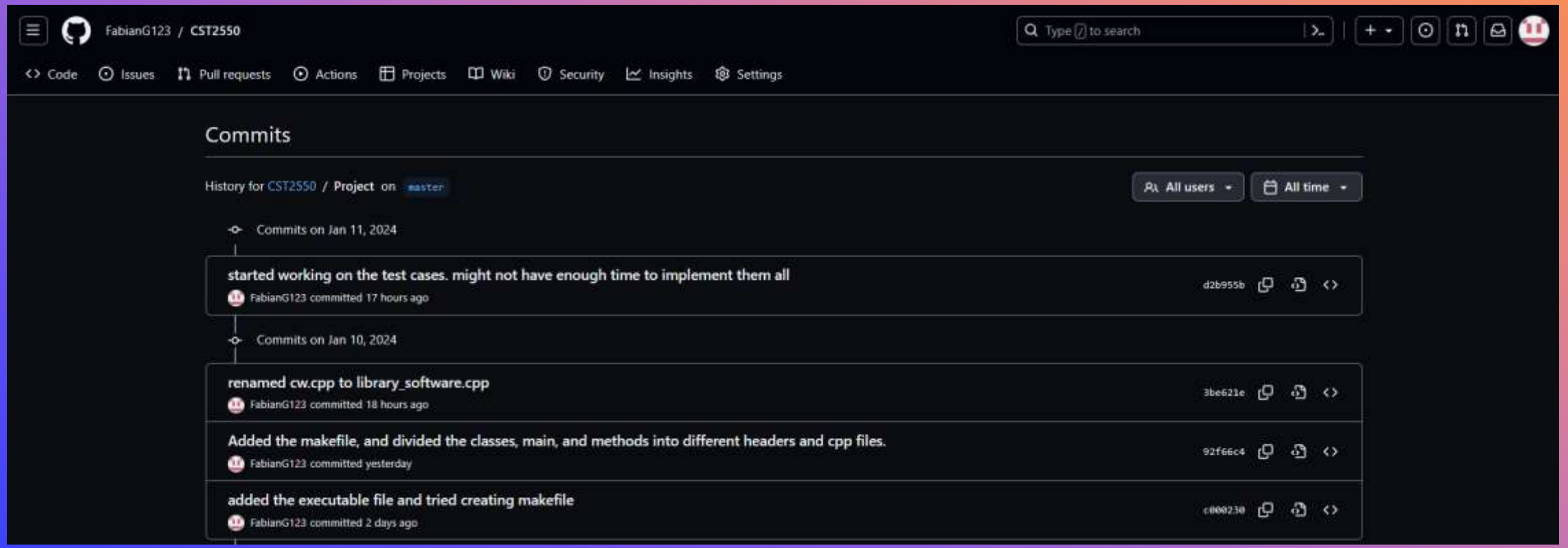
Added the Makefile, and divided the classes, main, and methods into different headers and cpp files.

renamed cw.cpp to library_software.cpp

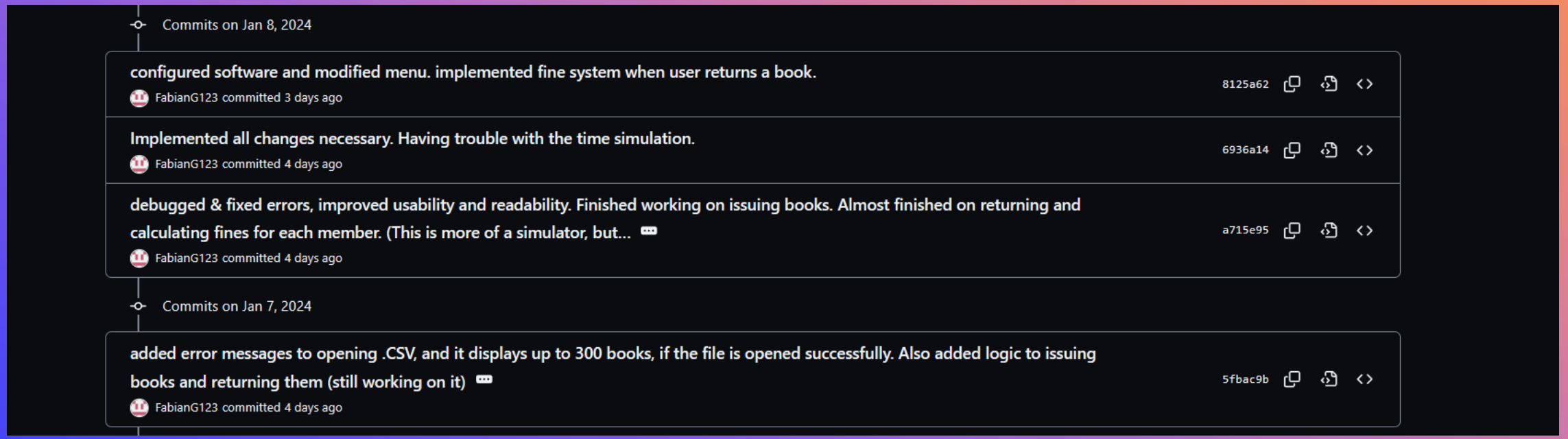
January 11, 2024

started working on the test cases. might not have enough time to implement them all

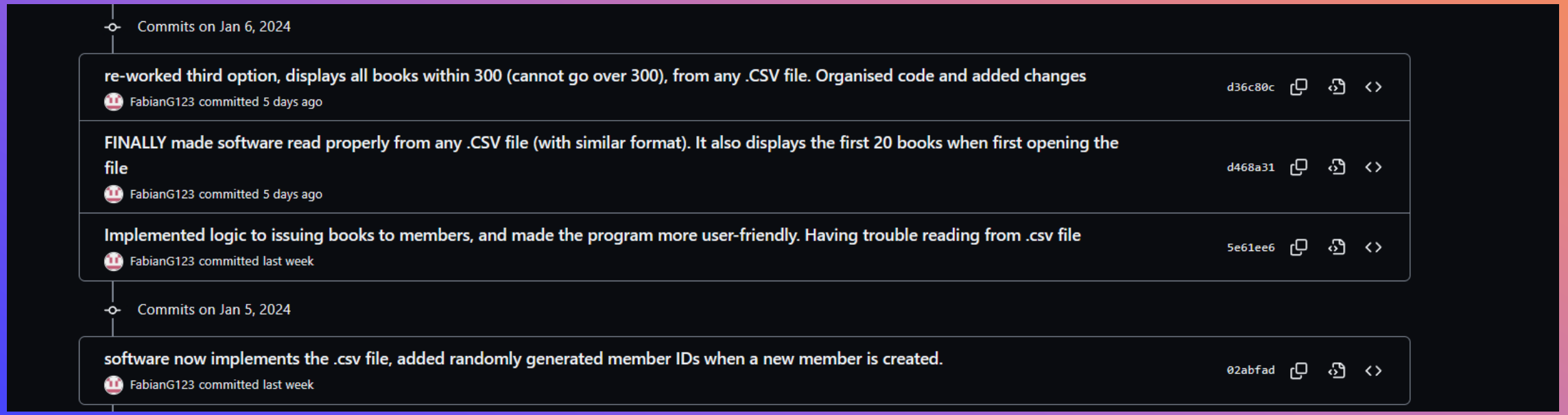
GITHUB REPOSITORY SCREENSHOTS (TOP TO BOTTOM)



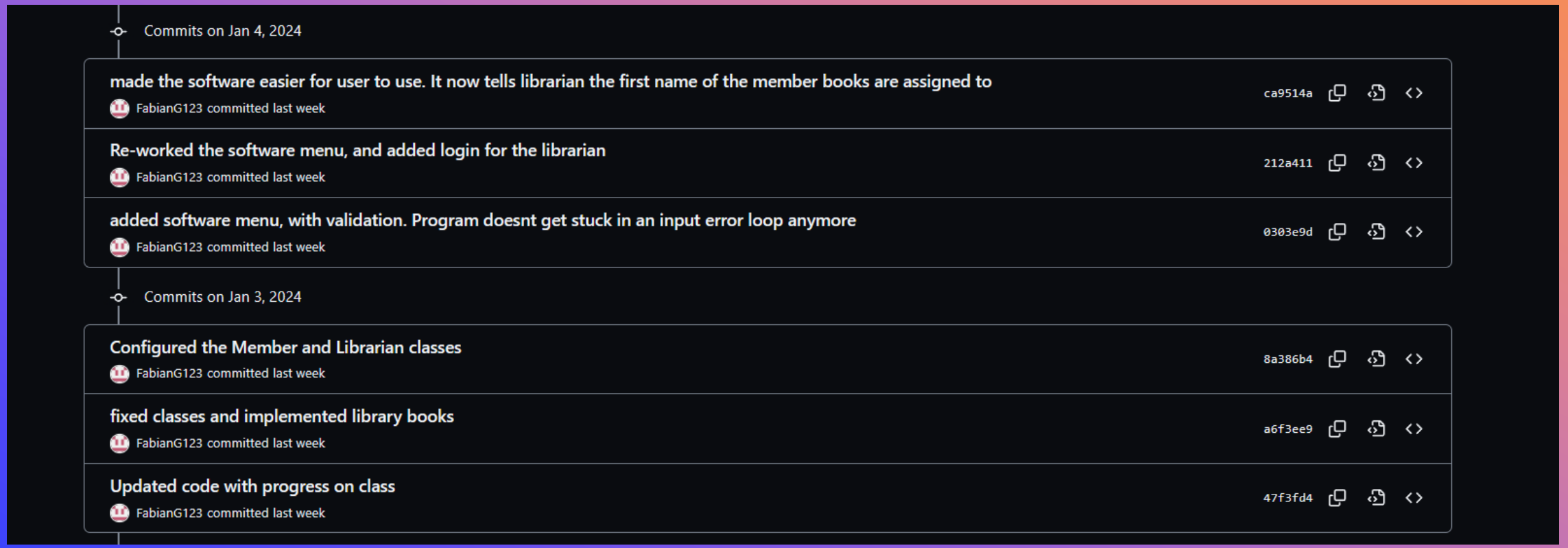
GITHUB REPOSITORY SCREENSHOTS (TOP TO BOTTOM)



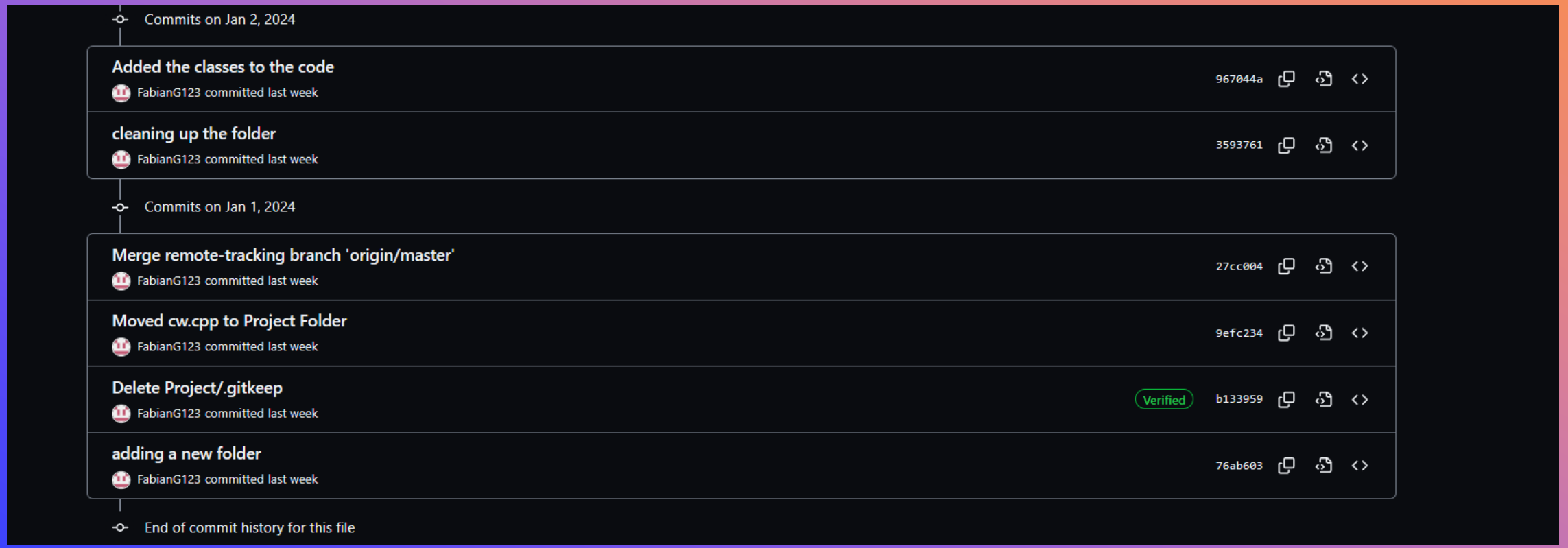
GITHUB REPOSITORY SCREENSHOTS (TOP TO BOTTOM)



GITHUB REPOSITORY SCREENSHOTS (TOP TO BOTTOM)



GITHUB REPOSITORY SCREENSHOTS (TOP TO BOTTOM)



TESTING

Subtitle

Testing approach (incomplete)

I've tried to implement testing for the software, but I ran into a lot of errors, and I didn't have the time in the end.

But my planned approach was the following:

1. Librarian Class:
   - Verifying that a librarian can log in successfully with correct credentials.
   - Verify that a librarian cannot log in with incorrect credentials.
   - Testing adding a new member to the system and verify the member's details.
   - Testing adding a member with invalid details (e.g., empty name) and ensure appropriate validation.
   - Testing issuing a book to a member and verifying that the member's details are updated correctly.
   - Testing returning a book from a member and check if fines are calculated accurately.
2. Member Class:
   - Verifying that a member is created with a unique ID.
   - Testing setting and displaying member details.
   - Testing borrowing a book and ensure the member's book details are updated correctly.
   - Testing returning a book and check if the book is marked as returned.
   - Testing attempting to borrow a book when the member already has one.

Testing approach (incomplete)

I've tried to implement testing for the software, but I ran into a lot of errors, and I didn't have the time in the end.

But my planned approach was the following:

1. Book Class:
   - Testing creating a book with valid details.
   - Testing creating a book with invalid details (e.g., negative page count) and ensure appropriate validation.
2. Main Program (Integration Testing):
   - Testing the overall flow of adding a member, issuing a book, and returning a book.
   - Testing the display of member and book details in the main program.
   - Testing the handling of invalid input during user interactions.
   - Testing the CSV file reading function with both a valid and an invalid file.

TESTING SCREENSHOTS


```
Fabian@DESKTOP-I7MTB2L ~/Project
$ ./tests
Enter member details:
Name: Fabian
Address: HA28AD
Email: galaselfabian@gmail.copm

-------------------------
Member added:

Member ID: 844
-------------------------




-------------------------
Name: Fabian
Address: HA28AD
Email: galaselfabian@gmail.copm
-------------------------

Book issued to FabianBook returned within 3 days. No fine.
Thank you for returning the book within 3 days!
Time since borrowing: 0 days
Book returned by Fabian
```

TESTING SCREENSHOTS


```
Book returned by Fabian

------------------------


Displaying books borrowed by Fabian:
No books borrowed by the member.

Member ID: 844
------------------------



------------------------
Name: Fabian
Address: HA28AD
Email: galaselfabian@gmail.copm
------------------------

===============================================================================
All tests passed (19 assertions in 4 test cases)
```

SOFTWARE DEMONSTRATION

A demonstration recording has been provided in the Repository, explaining the console and how the code runs.

CONCLUSION

Subtitle

Brief summary of the work

My Library management software is a console-based app written in C++. It includes four main classes. Person, Librarian, Member, and Book. The system allows librarians to manage and create members, issue and return books, and display member details and borrowed books. It also calculates a fine upon returning a book, which is only calculated after 3 days of returning the book.

Key features of my program:

- Librarians can add members, issue and return books, and view member details.
- Books are read from a CSV file, and the system displays them along with their details.
- The system handles member and librarian authentication with staff ID and password.
- Fine calculation is implemented for books returned after a specified period.
- Librarians must login with their ID and password.
- It can open any .CSV file, as long as it has similar format

Limitations of my work

Software Testing

- I did not manage to complete the software testing for this project because of errors and lack of time.

Use real time (to return books)

- The program uses a simulation of time, but the values can be changed.
- I am using seconds to test the fine and returning function. 3 minutes = 3 days.
- This can be changed to 3 days.

Allowing members to borrow more than 1 book

- So far, the software only allows a member to borrow one book.
- And if the member borrows a book, and then borrows one without returning it, the new book will override the previous one.

How I would approach a similar project in the future

I would allow myself more time from the beginning. So, I would start earlier. I would also structure my program to begin with, meaning that before anything, I'll use the design from the project proposal or scenario, and map out exactly how the code or software needs to be laid out. I would also read over the requirements until I have completely understood what it is I need to do. And if things still aren't clear, I would ask the client/user about specifics.

If I do that, I would be able to:

- Allow more time for debugging and error handling.
- More time for software testing.
- Spend more time on details and requirements specified in the brief.
- Be more organised and efficient with my code and documentation.

THANK YOU

Fabian

M00860030