LIBRARY MANAGEMENT SYSTEM

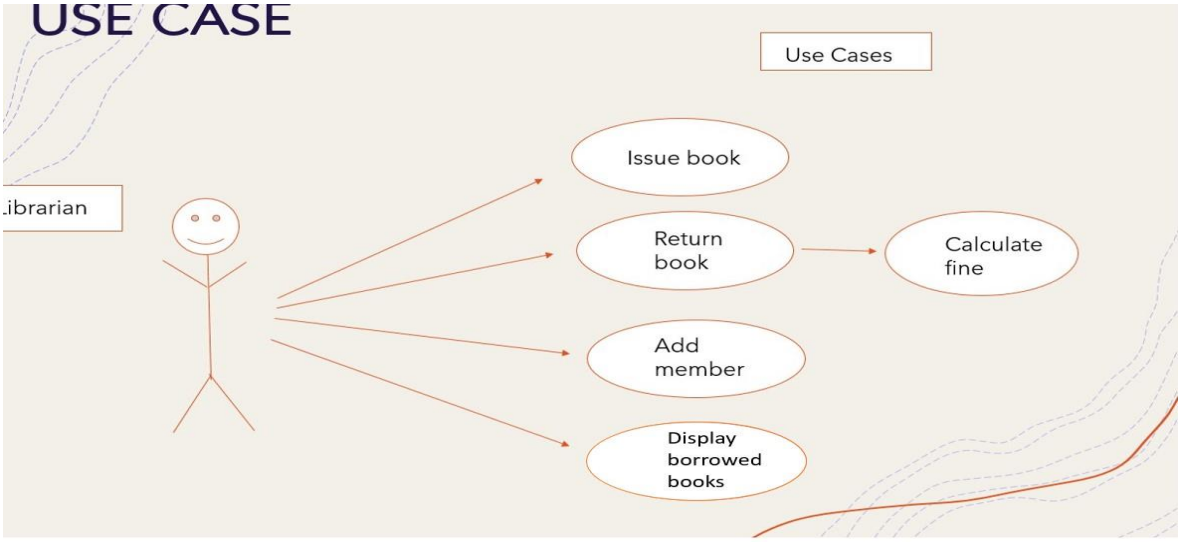
(M00897201)

The Library Management System is designed to empower librarians to efficiently track member details and book availability. Librarians can seamlessly perform various tasks, including adding members, issuing books, returning books, and displaying borrowed books.

This presentation delves into the design, implementation, testing, and demonstration aspects of the Library Management System.

USE CASE DIAGRAM:

Use Case Diagram for Library Management System:



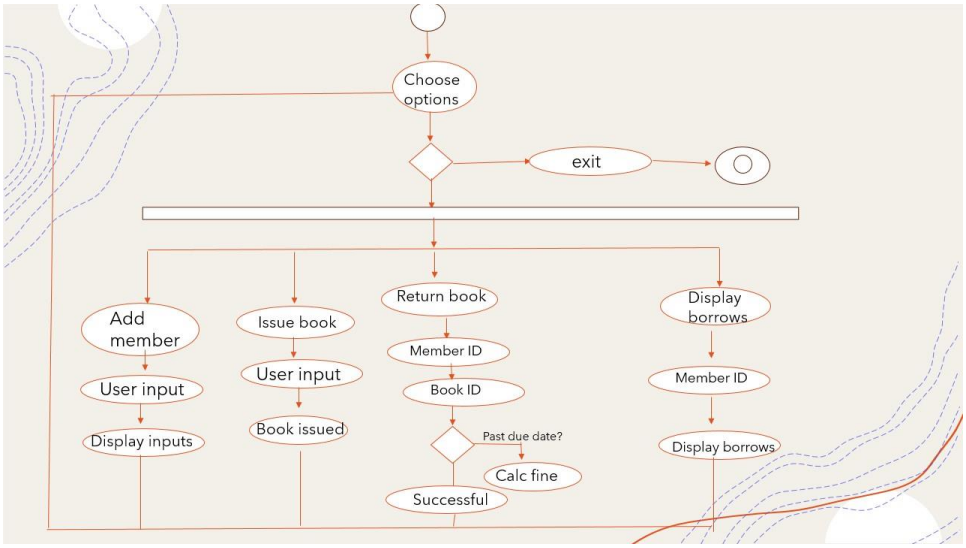
The use case diagram illustrates the interactions between the Librarian, the primary user of the Library Management System, and various tasks within the system. The Librarian serves as the central figure and engages in several crucial activities.

The Librarian can add new members into the library system, issue the members books, return the books and then display the books borrowed. The librarian can also issue fines if the book's return date is overdue.

The diagram features a boundary that encapsulates all actors and use cases, delineating the scope of the library system. In essence, this diagram functions as a visual representation, elucidating how the Librarian performs various tasks such as issuing books, returning books, adding members, calculating fine and displaying borrowed books.

ACTIVITY DIAGRAM

Library Management System Activity Diagram:



The activity diagram for our Library Management System illustrates how the system operates in real-time. It all begins with the initiation of a task, and the Librarian plays a central role in various activities.

The Librarian starts by adding a new member to the system, registering them in the process. Then, they handle lending books to members, manage returns, and may even calculate fines for late returns. The Librarian can also check and display the list of books borrowed by members.

Decision points in the diagram guide the flow based on conditions, like validating user input or addressing late returns. The entire process wraps up at an endpoint, indicating its completion.

The entire diagram is contained within the system boundary, which outlines the scope of activities within the Library Management System. To sum it up, this activity diagram visually represents how the Librarian engages in tasks like adding members, lending/returning books, and monitoring borrowed books, with the option to calculate fines for late returns.

IMPLEMENTATION

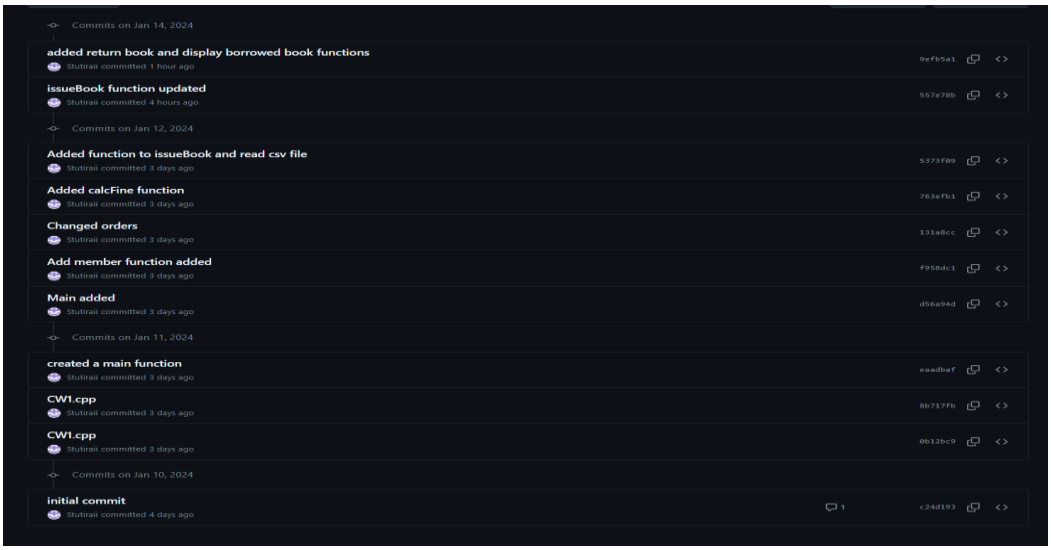
In translating the design into working software, I followed a systematic approach. I began by breaking down the high-level design into smaller, manageable tasks. Each component of the UML diagrams was addressed incrementally, ensuring that the code adhered to the design specifications. I employed an iterative development process, regularly testing and debugging as I progressed.

I used a Makefile as a tool to make compiling and running the software smoother. The Makefile had instructions on how to put together the code, link everything needed, and create the final program. Using a Makefile made the development process more efficient by automating tasks that we do repeatedly. This helped keep things consistent and made it easier to build the program. Plus, it saved time and lowered the chance of making mistakes when compiling and running the code.

Using Git with GitHub was crucial during development. It helped me keep track of changes made to the code over time. With version control, I could create separate areas for working on specific features or fixing bugs, making it easier to work on different things at once without messing up the main code. Each change I made was recorded with a short description, making it easy to understand what was done. This not only helped me see how the project evolved but also acted like a safety net, making it simple to find and undo any changes I didn't want.

SCREENSHOTS OF THE GITHUB REPOSITORY

Here's a picture of the history of changes made in the project on GitHub. The picture shows a list of changes arranged by time, each with a short but clear message describing what was done. This visual guide helps see how the project has progressed.



TESTING APPROACH:

In testing the library system using the provided Catch2 file, I followed a straightforward approach. I began with unit tests for the Librarian class, examining its staff ID and salary attributes to ensure they were set correctly. Additionally, I conducted unit tests for the Person class, validating the accuracy of the getName, getAddress, and getEmail functions. The testing process included scenarios where the Librarian class added a new member, and specific test cases focused on checking if the member ID counter incremented

as expected after adding a member. These tests aimed to guarantee the proper functioning of fundamental features within the system, with a focus on the Librarian and Person classes.

SUMMARY:

The program is a simulation of a library system, employing object-oriented programming concepts. It defines classes such as Book, Member, and Librarian, each encapsulating relevant attributes and behaviors. The main program initializes a librarian and loads book information from a CSV file into a vector called libraryBooks.

The system offers functionalities through a menu-driven interface in the main loop. Users can add members, borrow or return books, display borrowed books, and exit the program. The addMember function in the Librarian class allows for the registration of new members, generating a member ID and displaying their details.

The Librarian class also has functions like issueBook and returnBook, which simulate the processes of lending books to members and handling book returns. It calculates fines for overdue books during the return process.

LIMITATIONS

The existing code demonstrates functionality, but improvements can be made for better robustness. It lacks thorough error handling, more detailed comments, and comprehensive testing. The input handling in the menu loop is basic, and additional validation could enhance its reliability. A notable drawback is that testing requires commenting out the main function in the cpp file, limiting the effectiveness of testing procedures. Enhancing error checks, providing detailed comments for clarity, and improving testing methods would contribute to refining the code.

In future projects, I will research more on testing methodologies to address the limitations of the current code. While the existing code is functional, I recognize the importance of thorough testing for enhanced reliability and robustness. Some other key focuses will be on improving error handling to ensure the program gracefully handles unexpected situations, providing more detailed comments to enhance code readability, and conducting thorough testing to identify and fix any issues. I'll also refine the input handling in the menu loop to include robust validation for user interactions.