Trần Lê Hiền Đức

2131200129

Library Management System Project Report

1. Introduction

The Library Management System is designed to facilitate the management of books and readers in a library setting. It provides core functionalities such as adding books, searching for books, lending and returning books, and generating reports. The system is built following SOLID principles to ensure maintainability, scalability, and ease of extension.

2. System Architecture

The system is divided into four main components: Models, Repositories, Services, and the Main Program Interface. Each component plays a crucial role in the overall functionality of the system.

a. Models

- `Book`:

The `Book` class represents a book in the library. It includes properties such as:

- `Id`: A unique identifier for each book.

- `Title`: The title of the book.

- `Author`: The author of the book.

- `Category`: The category or genre of the book.

- `Quantity`: The number of copies available in the library.

- `Reader`:

The `Reader` class represents a library reader. It includes properties such as:

- `Id`: A unique identifier for each reader.

- `Name`: The name of the reader.

- `BorrowedBooks`: A list of books currently borrowed by the reader.

b. Repositories

- `IBookRepository`:

This interface defines the methods for accessing book data. It includes methods for:

- Adding new books to the library.

- Updating existing book information.

- Retrieving books by various criteria, such as ID, title, or category.

- `IReaderRepository`:

This interface defines the methods for accessing reader data. It includes methods for:

- Adding new readers to the library.

- Updating existing reader information.

- Retrieving readers by various criteria, such as ID.

- `BookRepository`:

This class is a concrete implementation of the `IBookRepository` interface. It manages the data storage and retrieval for books, ensuring that all book-related operations are handled efficiently.

- `ReaderRepository`:

This class is a concrete implementation of the `IReaderRepository` interface. It manages the data storage and retrieval for readers, ensuring that all reader-related operations are handled efficiently.

c. Services

- `ILibraryService`:

This interface defines the methods for library operations. It includes methods for:

- Adding new books to the library.

- Lending books to readers.

- Returning books to the library.

- Searching for books by title or category.

- `IReportService`:

This interface defines the methods for generating reports. It includes methods for:

- Generating a report of all readers and the books they have borrowed.

- Generating a book inventory report.

- `LibraryService`:

This class is a concrete implementation of the `ILibraryService` interface. It handles all library operations, ensuring that books and readers are managed effectively.

- `ReportService`:

This class is a concrete implementation of the `IReportService` interface. It handles the generation of various reports, providing valuable insights into the library's operations.

d. Main Program

- The `Program` class provides a console-based user interface for interacting with the library system. It allows users to perform various operations through a menu-driven interface. Users can:

- View all books in the library.

- Search for books by category or title.

- Add new books to the library.

- Lend and return books.

- View reports on readers and book inventory.

3. Application of SOLID Principles

a. Single Responsibility Principle (SRP)

- Each class in the system has a single responsibility. For example, `LibraryService` handles library operations, while `ReportService` is responsible for generating reports. This separation ensures that each class has a clear and focused purpose, making the system easier to maintain and extend.

b. Open/Closed Principle (OCP)

- The system is designed to be open for extension but closed for modification. Interfaces like `ILibraryService` and `IReportService` allow for new functionalities to be added without altering existing code. For instance, if a new type of report is needed, a new class implementing `IReportService` can be created without modifying the existing `ReportService`.

c. Liskov Substitution Principle (LSP)

- The system adheres to LSP by ensuring that derived classes can be substituted for their base classes without affecting the correctness of the program. For instance, any class implementing `IBook` or `IReader` can be used interchangeably with the existing `Book` and `Reader` classes.

d. Interface Segregation Principle (ISP)

- Interfaces in the system are designed to be specific and focused. For example, `ILibraryService` and `IReportService` provide distinct sets of functionalities, ensuring that classes implementing these interfaces are not forced to implement methods they do not use.

e. Dependency Inversion Principle (DIP)

- High-level modules, such as `LibraryService` and `ReportService`, depend on abstractions rather than concrete implementations. This is achieved through dependency injection, where repositories are passed as interfaces (`IBookRepository`, `IReaderRepository`) to the services, promoting loose coupling and enhancing testability.

4. Core Functionalities

a. Add New Books

- Administrators can add new books to the library by providing details such as Title, Author, Category, and Quantity. This functionality is implemented in the `LibraryService` class.

b. Search for Books

- Users can search for books by Title or Category. The search functionality is implemented in the `LibraryService` class, allowing users to find books based on their preferences.

c. Lend Books

- The system allows readers to borrow books if the stock is available. Each reader is allowed to borrow up to 3 books. This functionality is implemented in the `LibraryService` class, which checks for book availability and reader borrowing limits.

d. Return Books

- Readers can return books, and the system updates the stock quantity accordingly. This functionality is implemented in the `LibraryService` class, ensuring that returned books are added back to the library's inventory.

e. Generate Reports

- The system generates various reports, including a report of readers and the books they have borrowed, and a book inventory report. These reports are generated by the `ReportService` class.

5. Proposed Extensions for Future Development

a. Support for Additional Book Types

- To accommodate new book types like eBooks or audiobooks, the system can be extended by creating new classes that implement the `IBook` interface. This allows for the addition of new properties specific to these book types without altering existing code.

b. Enhanced Reservation System

- Implement a reservation system where readers can reserve books that are currently unavailable. This could involve adding a `Reservation` class and updating the `Reader` class to manage reservations.

c. Integration with External Systems

- Consider integrating the system with external library databases or online book retailers to provide real-time availability and pricing information. This could be achieved by implementing APIs that communicate with these external systems.

d. User Authentication and Authorization

- Introduce a user authentication system to manage different user roles, such as administrators and regular users. This would involve creating a `User` class and implementing role-based access control.

e. Mobile Application Support

- Develop a mobile application interface for the system, allowing users to access library services on the go. This would require creating a RESTful API to facilitate communication between the mobile app and the backend system.

6. Conclusion

The Library Management System is designed with a strong adherence to SOLID principles, ensuring a robust, maintainable, and extensible architecture. By following these principles, the system is well-prepared for future enhancements and can easily adapt to new requirements as they arise.