Report on Library Management System

Library management system majorly uses the below files:

1. check.py
2. book.py
3. user.py
4. storage.py

The above files majorly concentrate on performing the major CRUD operations required to manage the library system. It followed below approach where, every layer is discussed thoroughly

Layered Architecture: The overall structure of our system exhibits a Layered Architecture, where the application is organized into layers with specific responsibilities. This architecture promotes separation of concerns, making the system more organized and manageable. The key layers in your application include:

- **Presentation Layer:** This is represented by your command-line interface in main.py, where interactions with the system occur (like a **Command pattern** used to encapsulate a request as an object, letting us parameterize clients with different requests, queue or log requests, and support undoable operations.)
- **Major Logic Layer:** This includes classes like Books, Users, and Check which handle the core functionality and rules of the application ((like a **Singleton Pattern** to instantiate only once throughout your application's lifecycle.)
- **Data Access Layer:** The Storage class manages reading and writing to CSV files, abstracting the details of data persistence from the rest of the application (like a Repository Pattern, which is used to encapsulate the logic required to access data sources.)

check.py:

- Here we are using a class called Record. We are initializing a method with parameters book_management that is assigned to none. This works like a constructor where the values can be assigned.
- checkout_book() and checkin_book() method takes self, user_id and isbn parameters. The former is used to check out books and the latter to check in the books.

- During the checkout, it checks if the user_id exists or not and if if exists then it checkouts the book. If the user_if doesn't exist then it will throw a message that the given user_id doesn't exist or not active.

- During checkin function, it checks for valid user_id and see the maximum limit of 30 days for renewal of the book to avoid overdue/fine amount. If it crosses that 30 days, it will charge a fine of Rs 1 per each day.

book.py:

- Here we are using Book class. The constructor initializes the title, author, ISBN, and quantity values. eq() compares two books and returns true if both are the same or false.

- Class Books has a constructor that initializes the books array and a few methods to perform the CRUD operations.

- add_book() method adds a book. update_book() updates the current details of the book. delete_book() deletes and list_book() lists all the books in the library.

- We can perform search operations based on title, author, or ISBN.

- Also in this it checks whether the ISBN exists and avoids duplicate entry.

user.py:

- Class User has a constructor that initializes the user name and ID. eq() compares two users and returns true if both are the same or false.

- Users class performs the required CRUD operations. The constructor initializes the users and headers.

- add_user() method adds a user. update_user() updates the current details of the user. delete_user() deletes and list_user() lists all the users in the library.

- We can perform search operations based on name and ID.

storage.py:

- Class Storage has a constructor that initializes books_storrage, user_storage, and records_storage. The instances of the books, users, and records are also initialized.

- This file is generally used for read and write operations and to load and save the data.

- read_books(), read_users() and read_records() methods are used to read from the file storage and populate them.

- write_books(), read_users() and write_records() methods are used to write fro the file storage and populate them.

- load() and save() methods are used to load from the file storage into respective models and to save the data from respective models to file storage respectively.

main.py:

- Encapsulation of all the files and performs the major operations.

- **Run main.py** from the path and perform all the actions accordingly.