Design patterns justification:

Singleton:
The singleton pattern ensures that there is only one instance of each CSV management class throughout the application. This is crucial for managing access to the CSV files to prevent concurrent reads and writes that could result in data corruption or race conditions. By having a single access point, the system ensures that all read and write operations to the CSV files are coordinated and consistent, thereby maintaining data integrity.

Factory:
Since there are four different user types each represented by their own class, the factory pattern is used to encapsulate the logic of creating user objects of those different types (students, faculty, staff, and visitors). This pattern enables high flexibility and scalability in user management by abstracting the instantiation process. It allows the system owners to introduce new user types in the future without altering the codebase significantly.

Observer:
The observer pattern enables the system to notify objects when a value changes. In this case, the need for a way to notify faculty or library staff when a new version of a course's textbook is made available was solved using an observer pattern. This allows for easy communication across objects of different types without the need to create a coupling between them. It enhances the system's responsiveness and ensures that all objects that require to be informed of changes are receiving important updates.

Decorator:
The decorator pattern provides a solution to scale the size of news popups up and down based on the current window size of the application. The class enables the system to dynamically adjust the size of the popup at runtime without the need for pre-defined values. The use of a decorator in this case allows for a more adaptive user interface without the need to control window size or worry about device screen size resulting in a robust, and flexible application.

Strategy:
The strategy pattern is used to define a family of algorithms to calculate return penalties and execute the appropriate algorithm based on the conditions of the return and whether or not it is on time. This pattern allows the system to select the appropriate algorithm at runtime based on the return's timeliness. It provides a clear structure for managing various return policies, making the system more adaptable to policy changes and enhancing its ability to handle different scenarios efficiently. It also makes it easy to implement any changes to the return policies in future updates.

Composite:
The Composite pattern is employed to manage physical items that can be individual or part of a collection (e.g., CD collection, book series, etc.). This pattern treats single items and compositions of items uniformly, simplifying the management of these entities. It allows the system to easily support collections of items as first-class entities, enhancing the system's ability to organise and represent complex item hierarchies in a straightforward manner.

Requirements justification:

Req1:For user registration and validation, the system implements a dynamic registration process where users can sign up with a unique email and a strong password. Validation steps are included to ensure that the password meets security standards. Different types of users (students, faculty, staff, and visitors) are supported, with additional validation steps for certain user types (faculty, non-faculty staff) requiring an access passcode upon signing up. This process is facilitated by a centralised user management system in the form of a factory design pattern that handles user data securely and efficiently.

Req2: The system allows registered users to rent physical items and collections, access online books, and subscribe to online newsletters. Each physical item has 20 copies available, and users face penalties for overdue items. The system tracks borrowed items and enforces limits on how many items a user can borrow and how many a user can have overdue at once. This functionality is supported by an item management system that uses csv files to track availability, borrowing history, and due dates, ensuring users can easily access and manage their rentals and subscriptions in the dashboard.

Req3: Upon login, the system queries the borrowing records and displays a list of currently rented hardcover books along with their due dates. It also alerts users about items that are nearing their return deadline or are overdue. This requirement is achieved through searching and pulling data from the user's borrowing records using their unique id in the system, enabling personalised notifications and updates.

Req4: The system enables users to subscribe to and read paid-for newsletters. The system embeds a popup that loads the content directly from the original source. This feature is carefully designed to adjust dynamically to the app's size using a decorator, ensuring a consistent and user-friendly reading experience without the need for hard-coded values.

Req5: The system features a search function that allows users to find books based on keywords. It also provides recommendations for similar books based on overlap in words throughout the title. This functionality enhances user experience by making it easier to discover new items of interest or find a book the user is looking for despite not knowing the full title of.

Req6: For faculty users, the system tracks the courses they teach and the textbooks they have used and leverages an implementation of the observer design pattern to offer notifications about new editions of the same series. The system also notifies the library management team instead if the new edition of the textbook is not available.

Req7: Each physical item stored in the library has a unique identification number along with data that makes it easier to find, rent and manage such as location in the library and whether it can be purchased. This is handled through a centralised item catalogue that stores all items and their data in a csv file and retrieves them to be displayed in the user interface. Library staff can manage each item’s information along with adding new items and enabling or disabling rentals and purchases.

Req8: The system facilitates access to the textbooks that each student needs by integrating course enrollment data with textbook management. The app provides users temporary access to a digital copy of the textbook for the duration of their enrollment. This functionality is designed to ensure that students have the necessary resources for their studies without manual intervention from their teachers.

Req9: A user can submit a request for a new book to be added into the library's catalogue. The request can be either made for the purposes of teaching a course or for self-improvement and personal use. Each request is processed through a prioritisation mechanism that categorises the request based on its type and urgency. Textbook requests for teaching purposes are typically given higher priority over other types of requests.

Req10: In some cases where an item is not available in the library for free, a special agreement with the publishers is made to provide students with a discounted price. In order to process transactions, the system includes a way to process and store payments of different types such as debit, credit, and cash. This is handled using a separate class dedicated to payment processing.

Req11: Instead of using a traditional database, the system uses CSV files to store and manage data. Access to these files is done using singleton classes that prevent data loss and race conditions while ensuring data integrity. This approach simplifies data storage and management requirements for the project, with a system design that ensures efficient data access and updates.