Deliverable II Report

Patterns chosen:

- **Singleton:** The user authentication information is stored in a singleton. This information is constant throughout the application lifetime and does not change. This data class is lazy-loaded and referenced throughout the application which is ideal for the singleton pattern.
- **Abstract Factory:** The abstract factory is used to create components that can be parsed from JSON to be rendered by the Swing GUI. The User and Item information are hierarchical and have a complex construction. Separating the creation of an object helps to obey the Single Responsibility Principle when object creation is complex. This makes code more readable and comprehensible. Without this pattern we would have many different constructors for different purposes in the component classes.
- **Proxy:** HTTP Calls to the backend service can be expensive, so the proxy pattern is deployed to cache frequent results. The results are evicted after a short time frame. In this case this is a caching-remote-proxy. This pattern aggregates an object with “expensive” or time-consuming method calls, while also implementing the same interface. This way we can extend and modify the behavior of the original request object while adhering to the “open for extension closed for modification” principles.
- **Chain Of Responsibility:** Chain of Responsibility is used to process some of the requests to the server. Some chains include checking email and password with regex for proper formatting, making a request to the server, checking if a user has permissions to perform an action, etc. At any point in the chain, an error or decision to leave the action may occur, and so a JOptionPane is created instead, and the rest of the chain links are cancelled. This is ideal for modularizing the requests to be processed which can be reordered in different ways, and that can be canceled at any specific stage along the way. The JOptionPanes are paired and dependent on the filters they correspond to by using this pattern.
- **Bridge:** The bridge pattern separates an abstraction from an implementation and allows the implementation to be swapped out at runtime. Depending on the size of the search, we may want to display search results in a grid or a list with a single dimension. The bridge pattern encapsulates this decision in its own class, keeping client code clean and delegating the decision of which style to use as the choice of the bridge class.
- **Flyweight:** When an item is clicked on the search panel, a JFrame is activated with information relating to the item. Depending on several factors like the type of the item, the item’s stock, etc. These frames may include different options or display different information or options. Some parts of the frame are common to

multiple items, while others are unique to the individual item. Rather than creating a brand-new frame for each item in the search results, the common features of all JFrames are extracted into base classes, with the unique elements of each item frame kept in their own collection. This saves on memory and avoids code/data duplication.

System Decomposition:

The system follows an MVC architecture, consisting of a view built in swing, RESTful controllers built in the Spring framework, and a model in the form of a MySQL database hosted on Microsoft Azure. One Maven module contains the swing application, and the other contains the Spring backend. In production, the backend would be kept on a virtual machine in the cloud.

The advantages to this component structure are as follows:

Security and Safety: The model and controller are held on their own computer, and credentials are required to access the data throught http calls. This makes the data more secure by keeping database credentials out of the individual user swing app. This serves as a kind of two-layered architecture, where the GUI interface and code beneath has no knowledge of an existing database, using only requests to get the information it needs. This way components also localize their safety critical features: the database is far removed from the GUI.

Performance:

The remote proxy pattern caches requests to minimize requests to the server. The GUI actions are wrapped in SwingWorkers (Threads) to keep long requests and GUI updates off the main thread, to prevent an unresponsive interface.

Availability:

The backend is kept live and is a constant throughout every user application. The integrity of the data is high this way, all users adhere to the same database. Fault tolerance can be dealt with using database tools provided by MySQL.

Maintainability:

The frontend and backend are completely separated, allowing them to be decoupled and tested/replaced individually. Both frontend and backend contain low coupling high cohesion components that provide services for the GUI and data model respectively.

Requirements achieved:

Systems in the backend controllers process information, such as late book returns, book requests, etc. The database model contains timestamps of when a user either purchased, rented, or subscribed to items. Timestamps are also taken when such an action is taken. The management team uses the database to check the verification of users and view requests. When a book request is sent, a priority integer is sent in return to display to the user their request priority. Notifications are gathered whenever the homepage is loaded, displaying which courses the user is within, their books, and any information about due dates or overdue books.

Team Contribution Breakdown:

- **Alexander Odorico: 25%**
- **Uzziah King Lardizabal: 25%**
- **Robert Cultraro: 25%**
- **Garnett Condeno: 25%**