CSCI 4448 - Project 6

Team: Tuqa Alaithan, Nikhil Rajaram.
Project Title: Library Simulator.

Final State of System Statement:

The final state of our system was quite different than what was described in our aspirations in Project 4, with some features abandoned and some features added. The finished product is a tool that libraries would be able to use to manage their operations and expose their content digitally. At the end of development, we have developed the following features:

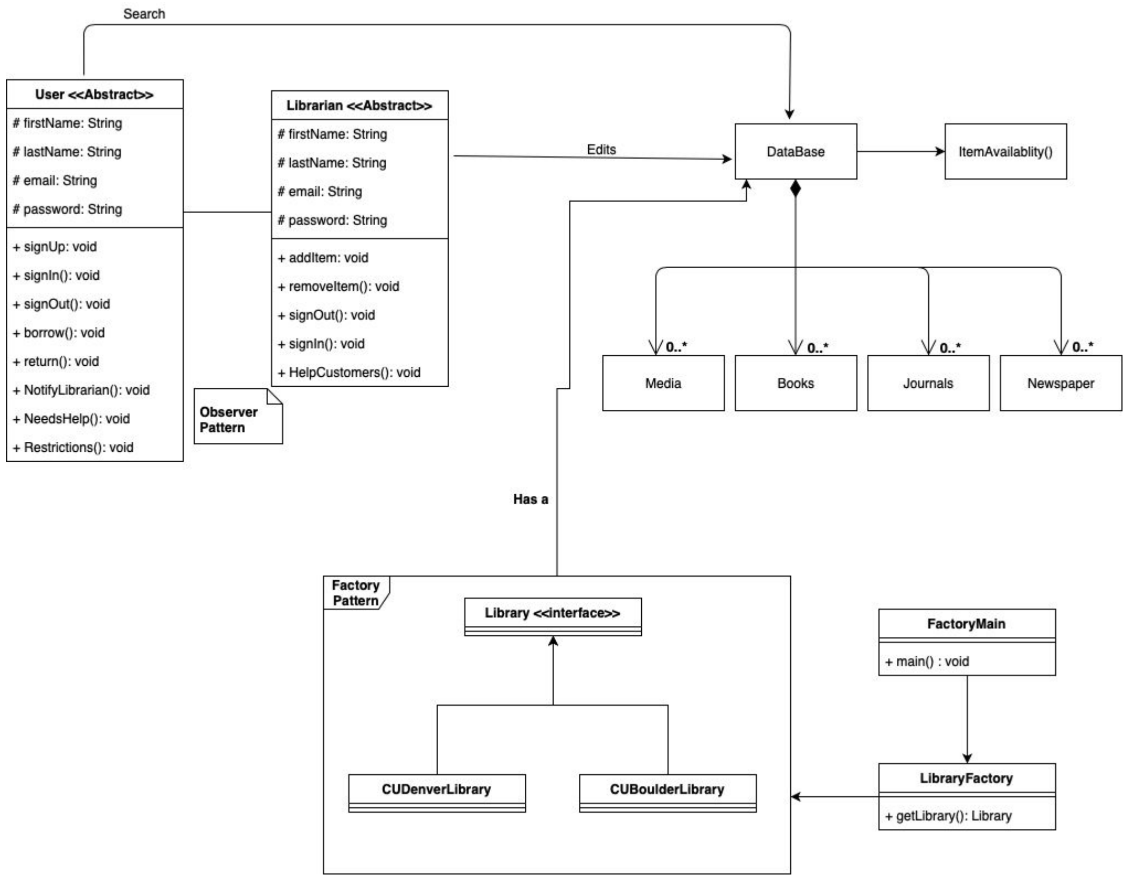
- Login/Registration
    - Users may register on the website
        - When they do so, their email and encrypted password (encrypted via the Bcrypt algorithm) are stored in a Heroku Postgres database instance
    - Once users have registered, they may log in
        - When they do so, their credentials are verified against the credentials in the database
        - If credentials are incorrect, an alert shows up that the credentials are incorrect
- Home page
    - The navigation bar at the top will allow the user to visit relevant links and sign out
    - On load, the home page will show 4 random books and 4 random movies to the user
        - 500 Books and 200 movies (items) have been parsed from the linked corpuses and persisted in the database
            - Parsing methodology in `scripts/getmovieandbooksummaries.ipynb`
    - The user will be able to click on any item and go to a summary page for the item
- Books/movies page
    - On load, the /books and /movies pages will display 12 item cards and will allow the user to access new pages with more items
- Item pages
    - Each item has its own page with dynamically loaded content from the database
    - Items that are labeled as "digital" will have a button on the page that will let the user check out the item
        - Check out for digital items was not implemented as defined in Project 4 given that we do not have access to the content and also do not have resources for third-party DRM tools
- Request help
    - When a user clicks on the navbar "Ask A Librarian" button, they are redirected to the /requestHelp page where they can ask a question
    - Librarians are related to users via a one-to-many Observer relationship
        - Librarians are observers, users are observables
        - On registration, users are assigned multiple librarians to observe their requests
        - Each user has multiple librarian observers
        - Once a user asks for help, it notifies its librarian observers

    - Once a help request is submitted, librarians are able to view their requests by viewing the /checkRequests page where their help requests are shown
        - A librarian may respond to a user by clicking the mailto link and emailing the user
- User roles
    - User roles are set by administrators (ourselves) and range from USER to LIBRARIAN
    - USER role
        - Can access home page, books/movies pages, item pages, request help pages
    - LIBRARIAN role
        - Can access all pages that user can as well as /checkRequests, /checkOutPhysicalItem, and /returnPhysicalItem
- Check out/return physical item
    - Librarians may check out physical items for users who presumably would approach them in a real library environment
    - Once a librarian inputs the user's email and the ID of the item

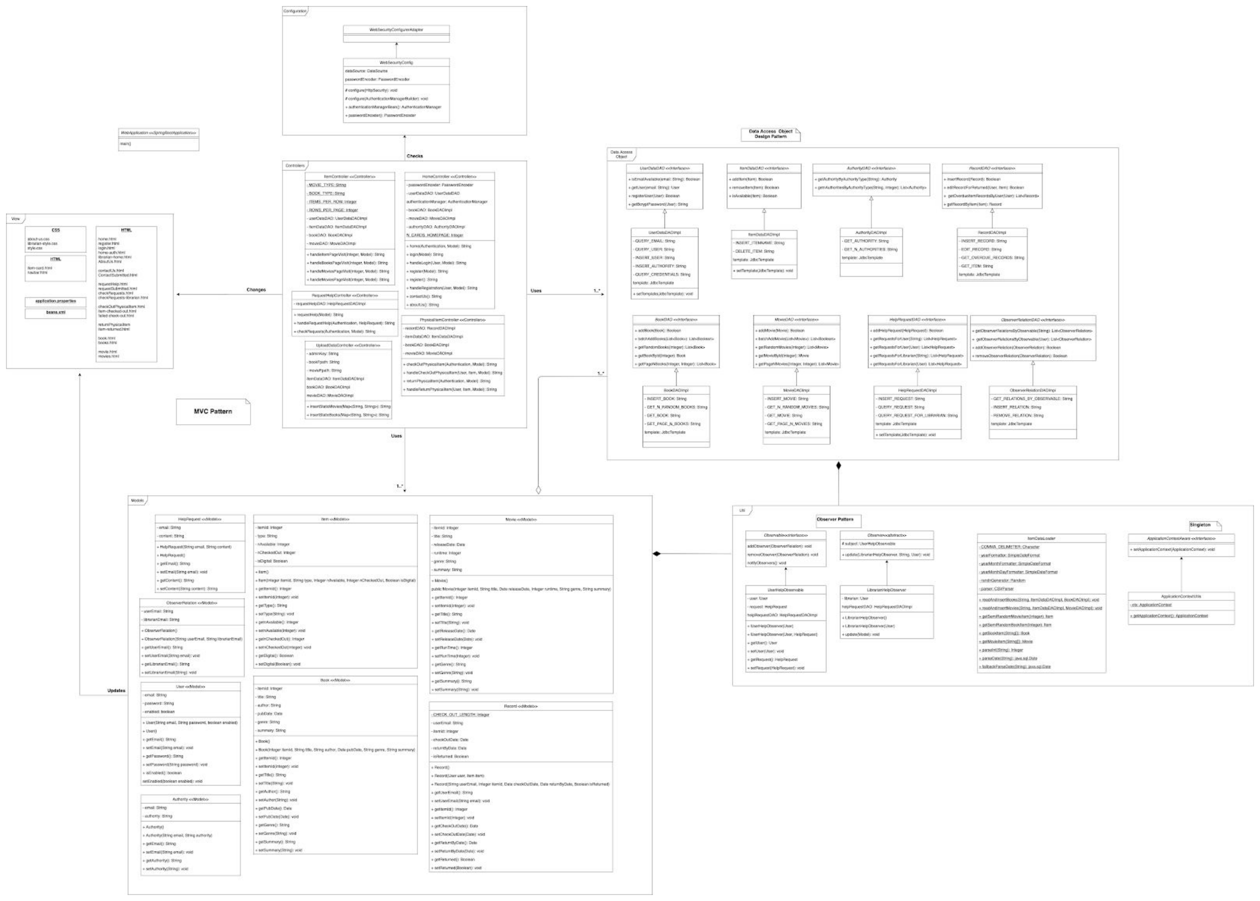
- ❖ Features in Project 4 that were not implemented in the final product:
    - Factory pattern: we were planning on implementing Factory to be able to create two types of libraries; CUDenver and CUBoulder libraries. However, we decided to create one CSCI4448 Library and so there was no need for the Factory pattern.
    - Journals and Newspapers: we did not add Journals and Newspapers to our library as intended since our dataset only has books and movies.

Final Class Diagram and Comparison Statement:

- **Project 4 UML Class Diagram:**



- **Final Project Diagram:**



<u> For a clearer diagram, please check "Project6UML.png" on GitHub.</u>**

- **Key Changes and Notes:**
  - As it is shown in Project 4 UML Diagram, we were planning on making two different models for regular users and librarians. We decided to merge them into one model "User" and differentiate between regular users and librarian by Authentication; unauthorized user (regular user) and authorized user (librarian) while using Delegation. We did this to have our models conform to the Spring framework better.
  - In Project 4 UML Diagram we can see that the Observer pattern was to be implemented as methods inside Librarian and User models. Due to the merge of users and librarians models into one model, we decided to create new classes "LibrarianHelpObserver", which has an instance of "User librarian", and "UserHelpObservable", which has an instance of "User user" and "HelpRequest" model. Additionally, "ObserverRelation" model is helpful to assign observer-observable relation between users and librarians.
  - We added "HelpRequest" model as it helps to use it for viewing the content of requests for librarian to see them.
  - We also added "Record" model to create a record for a user if they checked out a physical item. The "Record" includes user's email, the checked item ID, checkout date, and return by date.

    - We used Data Access Object Design Pattern to separate application layer from the persistence layer (Postgres relational database).
    - Used MVC pattern that has 5 controllers, 8 models, and 21 views.
    - Used Singleton pattern for the ApplicationContextUtils class to get the Spring ApplicationContext from non-Spring managed classes.
    - Factory pattern was not implemented since it is unnecessary.
    - We used implementations of the Builder pattern in:
        - WebSecurityConfig (AuthenticationManagerBuilder) to build an AuthenticationManager object to manage users and authorities
        - ItemDataLoader (CSVParserBuilder/CSVReaderBuilder) to build CSVParser and CSVReader objects to parse through the CSVs of items
        - We did not include these in our class diagram as we did not implement the pattern ourselves

Third Party code vs. Original code Statement:

- The following websites were used to get familiar on how SpringBoot works with MVC pattern:
    - https://spring.io/guides/gs/serving-web-content/
    - https://www.baeldung.com/
- The following website was used to review Bootstrap syntax/functionalities:
    - https://www.w3schools.com/bootstrap/
    - https://getbootstrap.com/docs/4.0/content/tables/
- AboutUs.html and contactUs.html templates from:
    - https://www.w3schools.com/bootstrap/bootstrap_theme_company.asp
- Exposing AuthenticationManager was done by the answer at the following link:
    - https://stackoverflow.com/questions/21633555/how-to-inject-authenticationmanager-using-java-configuration-in-a-custom-filter
- The rest of the code is original work.

Statement on the OOAD process for overall Semester Project:

- **List 3 key design process elements or issues (positive/negative):**
    - Using Spring MVC was challenging to learn but made the code more robust and secure.
    - LibrarianHelpObserver and UserHelpObservable are not managed by Spring and so we extracted bean from application context for instantiation.
    - Our code is flexible so that if in the future we needed to add new models or data access objects it won't mess up the rest of the code or change much.
    - There are many classes as shown in the UML class diagram but the creation for all classes was necessary because each class has a different responsibility.