Book Store

Assignment 2

srof384 and udea631 | Compsci 235 | 24/09/2021

New Reading List Feature

The route we chose when deciding on our new feature was to think about what's missing; what would we like to have if we were using a bookstore/library. We realised that there was nothing to save books we were interested in. We would have to take a picture or try to remember the books. This would be manageable for a small-sized catalogue but in larger sized catalogues (which we were thinking of as a real-life situation) this would be a tedious and incontinent way to remember books. Thus, the new feature we decided to implement was a personal reading list for the user. With this feature, users can add and remove books to and from their reading list, as well as view their reading list. This will remove the need for users trying to remember book details. To use this feature the user is required to log in. The inspiration for this feature was Goodreads' Bookshelf feature where users can add books to their 'Want to Read' and 'Read' list.

To implement this feature, we modelled it after our browse books section. This was done to try to give potential users a degree of familiarity and to fit the whole look/ design of the webpage. We used the User domain model attribute read_books that is a list of Book objects. We added methods in memory_repository.py to add books to the user's read_books list, get the length of said list, remove a Book form the list and return the list. In the repository.py interface we added the corresponding abstract methods to enable the service layers to use the added functionality.

In the service layer for the reading list, we added methods to access the mentioned methods in the repository interface, as well as methods to get a Book object by its id, get a User object by the username from the current session and a method to get the list 4 books to display per page along with a reviews button and an add to reading list button. We then created a blueprint to handle requests from the website for the methods. All the methods in the blueprint require the user to be logged in. This functionality was provided by the authentication blueprint.

Our Jinja template for viewing the reading list is identical to the template for browsing books, with some added personalisation and a button to remove the selected book from the users reading list. We also added a new button to the template for browsing books so users can add a book to the user's reading list. This button redirects the user to the login page if they haven't already done so.

Overall Design

When designing our project, it was important for us to conform to the dependency inversion principle. This was important as we moved through iterations in our development, as we needed to ensure our ability to safely make changes in high-level modules without affecting our completed low-level modules and vice versa. By ensuring our modules are decoupled we can safely make changes to one module in our project without affecting the others. This allows us to easily extend the functionality of our project should the need arise.

By using the dependency inversion principle and repository pattern, we also future-proof our project for changes in the method of data storage. If later in development, we wish to change our method of storing data to using a database repository we can do so without needing to change high-level modules. This is because of our abstract repository interface being unconcerned with the details of the data storage. This also allows us to postpone committing to a database while designing the minimal viable product. In doing so this also allow us to easily respond to changes in the stakeholder's requirements for data storage, even late in development.

We adhered to the single responsibility principle at each level of our project. From the overall structure, to each class and method. By dividing the project structure into the route handlers, service layer, abstract repository, and memory repository layers we ensured that our project is easy to maintain, understand and reuse, which are the fundamental benefits of the single responsibility pattern. For each layer of the project, we divided the methods and classes into relevant packages based on the functionality they were responsible for, i.e authentication, searching, displaying books etc. The methods and classes within the packages were divided so each is responsible for a single focused task within the project.

By dividing the project in this way, we ensured our project is maintainable by keeping tasks isolated, and thereby reducing the chances of future changes causing errors throughout the project. This also aids in debugging and our ability to respond to change, as it is easier to follow the path of a single task that interacts only with methods and classes directly relating to its task.

This also ensures the program is easy to understand as each method is self-explanatory in what is tasked with and how it carries out this task.

By having a repository interface and service layers we ensure that we can reuse methods from our memory repository across the project without modification.

Visual Design

Our visual design choices were heavily influenced by (possible) user experience by focusing on usability and usefulness in a minimalistic approach. This design ensures Usability by being easy to use with little to no learning curve, Usefulness by enabling users to accomplish desired goals (browsing books, adding reviews, etc). This design aims to limit user's perception/cognitive load and decrease visual complexity.