Use Case and Architectural Design Report

In our Architectural design folder we have three files to show the functionality of our project. The three files, “Use Case DIagram Final Project.png”, “Use Case Notes Final Project1.pdf”, and “class diagram2.png”, refer to the use case diagram, the use case descriptions and the class diagram respectively.

Use Case Diagram

The use case diagram for our project includes a total of 10 use cases, with two actors. The use cases for the Employee include Add Admin, Create, Remove, Modify Entries, Register, and Search book catalog. The use cases for the End-User/Customer include Register, Renew Books, Borrow Books, Return Books, Pay Fine, and Search book catalog. It's important to note that although both the End-User and Employee can Register and search the book catalog, neither can access each other's private functions. Also, it is assumed within the diagram that in order to access the system you must either register or log in. Additionally, the only interesting thing about our use case diagram is the pay fine use case. The pay fine use case is only accessed when a user tries to return a book that has been signed out too long, therefore it extends the return book use case as it changes its behaviour. Furthermore, we never ended up including the pay fine use case specifically, we ended up adding the functionality of the pay fine use case to the renew book use case.

Use Case Notes/Descriptions

The use case notes/descriptions are the full flow/descriptions for each use case outlined in the project. It encompasses the overall description, workflow, business logic, decisions, follow-up, scenarios, details, and requirements of each use case.

As a heads-up accessing the file preview through GitHub may be wonky as the file is a pdf, try giving the file a download to access it.

Class Diagram

The Class Diagram for our project is actually quite simple. We’ve decided to include the main functionality of our project all within one principle class, the Home Controller. This decision,

though usually rash, turned out to be excellent, as it resulted in the code becoming much simpler to work with. This is largely because of the program we used to code, ASP.NET core.
Now, our project mainly consists of the Home Controller, with other classes largely being attribute placeholders for passing information through the database. This can be seen when borrowing a book for example, the home controller asks both the account class and the book class for the bookID and card number, then asks the borrowing class to borrow the book with those attributes.

As for the diagram itself, each relationship shown is an associative relationship. For example, the Home Controller class is associated with every other class, the account class is associated with the borrowing class, and the book class is associated with the borrowing class. Each of Account, Borrowing, and Book have getter and setter for each attribute.