

LIBRARY MANAGEMENT SYSTEM

Project Part 4: Final Report

CSCI5448

Sanketh Shetty
Tauseef Indikar

Gaurav Shukla
Nicolas Neitzel

1. List the features you implemented (from part 1 and 2 where you listed the features you were planning to implement/requirements you listed) Show Final class diagram, discuss benefits of designing before coding.

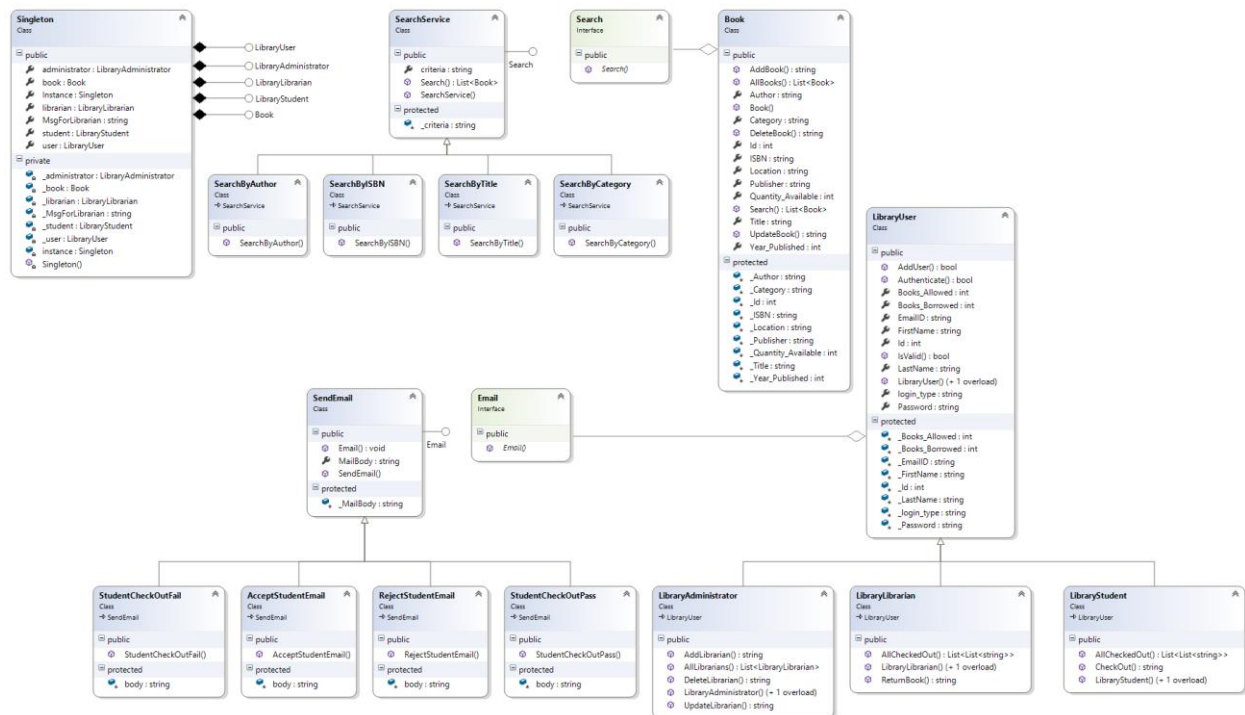
In part 1 and 2 of our Project minimum functionality requirements were:

1. Student Signup
2. Student Login
3. Librarian Login
4. Admin Login
5. Admin adding a Librarian
6. Admin can add, edit, update books
7. Student can check out books
8. Librarian can issue books to student checking his credentials

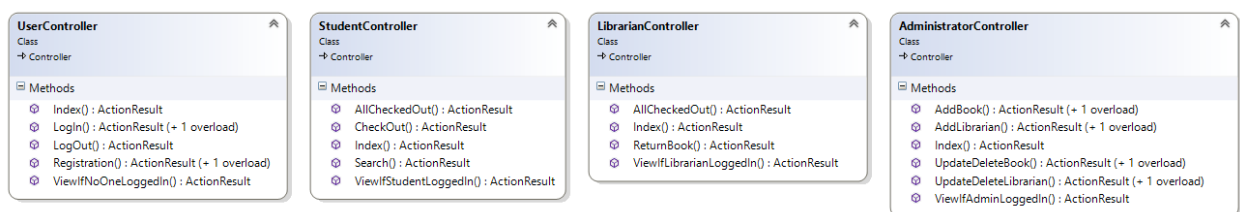
There were some stretch goals as well:

1. Student and Librarian Search functionality
2. Librarian manages Number of Available books for Student

Final Models Class Diagram (Kindly zoom for a clearer view)



Controllers Class Diagram



In our final implementation of the project, we have been able to cover all the minimum functionality requirements as stated above. We have implemented all the stretch goals. Some additional functionality that we have implemented can be considered as “changes in requirements” for the project. These changes were done keeping in mind that this is an Object Oriented Design class and the focus should be on learning and implementing design patterns as our initial design had not incorporated patterns apart from MVC framework usage.

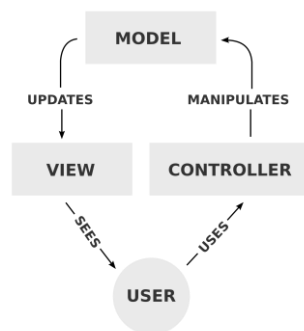
Over the period of the course our team members have spent significant time brainstorming over design and we have realized few things along the way:

1. Some of us were used to starting code as soon as we get a problem, so the design first approach was quite a departure from the usual route.
2. Design is hard. Design is a completely different ballgame compared to coding.
3. Brainstorming leads to some amazing ideas.
4. Design helps us give structure to our ideas before we translate it to code.
5. Design helps us develop software that can withstand changes in terms of requirements.
6. Design helps us develop software that is scalable.
7. Design is Programming language agnostic; therefore, we can later use the design in a different language which has same features (different OO languages).
8. Once we have a design/architecture template, it can be reused to solve different problems, thus saving a software team time and maximize productivity. Since documentation of UML/OCL diagrams becomes part of the process, it helps management give a direction to teams and lead them to complete projects within time and budget.

In the end we can say that the final code is beautiful, modular and easy to manage only because we used the design first approach. The final code is scalable and can easily be reused by others.

2. Design patterns we used in the Library Management System. Did you make use of any design patterns in the implementation of your final prototype? If so, how? If not, where could you make use of design patterns in your system?

As it can be seen from the Class Diagrams, the overarching architectural design pattern used in our project is the MVC (Model-View-Controller) pattern. We use the ASP.NET framework and MS SQL for the data persistence. The **model** contains classes that store data that is retrieved according to commands from the controller and displayed in the view. The **controller** contains classes and interfaces that can send commands to the model to update the model's state (e.g., editing a document). It can also send commands to its associated view to change the view's presentation of the model. The **view** in the ASP.NET does not contain any specific classes as such, but uses cshtml templates to render views.



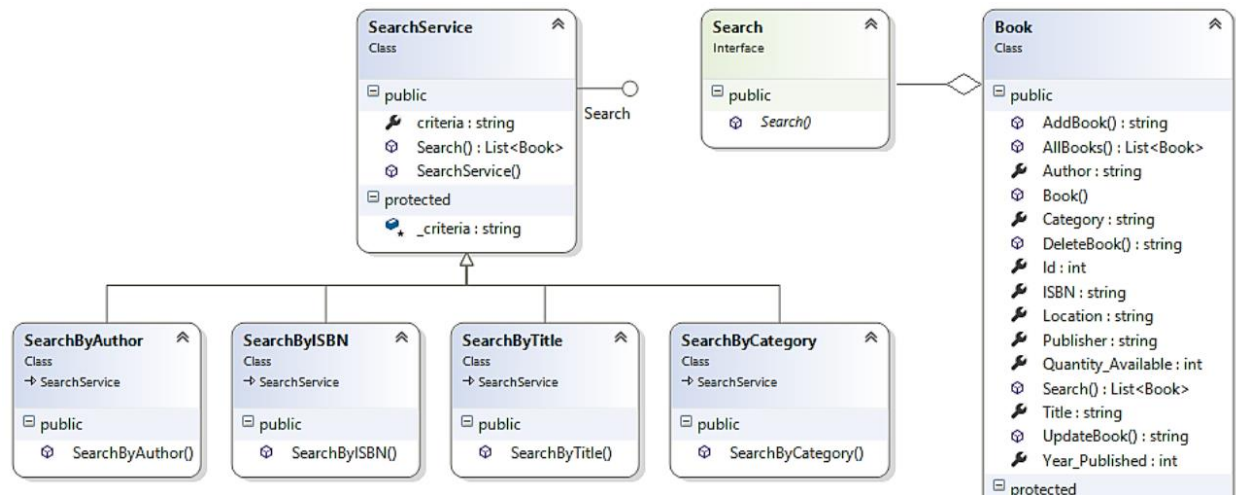
We learnt C# and the ASP.NET MVC framework. Using this framework made our project very Modular. We have created four controllers for one for each type of website i.e. User, Administrator, Librarian and Student. The routing is done so as to prevent access to other type of user's controller actions. The models used are Book, LibraryUser, LibraryAdministrator, LibraryLibrarian and LibraryStudent. A singleton class is also used. We have two interfaces. One for Search which is concretely implemented in SearchService and the other for Email which is concretely implemented in SendEmail.

We have used **Strategy pattern** in Search interface and **Bridge pattern** in Email interface. Although implementation wise the design for both the interfaces is similar the intent is different. The Search interface abstracts behavior based on user request at runtime while the Email interface just provides different functionalities which are predefined to be used in methods of LibraryUser class.

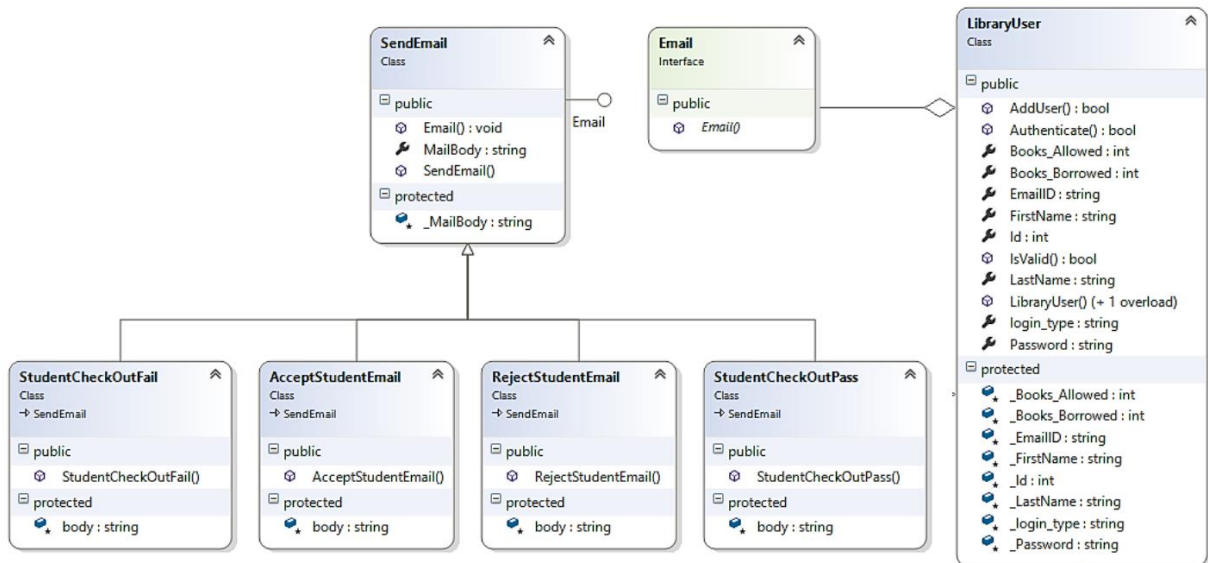
```

public List<Book> Search(String term, String criteria)
{
    Search search;
    switch (criteria)
    {
        case "Author" : search = new SearchByAuthor(); break;
        case "ISBN" : search = new SearchByISBN(); break;
        case "Category" : search = new SearchByCategory(); break;
        default : search = new SearchByTitle(); break;
    }
    return search.Search(term);
}

```



Search Interface – Strategy Pattern – Behavioral because algorithms are selected on the fly



' Email Interface – Bridge Pattern – Structural because algorithms abstracted from implementation for decoupling and neater code

We realized after we had code finalized that we could have used the **Factory pattern** in the code, but we decided to leave it at that as that would make some changes in the Strategy pattern.

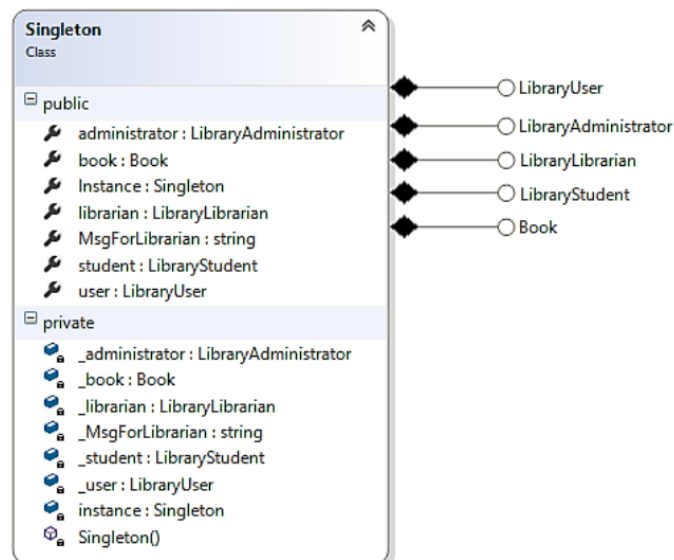
Also we have used the **Singleton pattern** to have a single ready instance of all our basic models available at runtime so that we would not have to instantiate each of them every time the controller is called. Also the singleton instance is used to keep track of when the last Student user logged in and show this to the librarian. This is useful as it helps write code like:

```
ViewBag.Results = Singleton.Instance.book.Search(Request.QueryString["Term"],
Request.QueryString["Criteria"]);
return ViewIfStudentLoggedIn();
```

As opposed to:

```
Book book= new Book();
ViewBag.Results = book.Search(Request.QueryString["Term"],
Request.QueryString["Criteria"]);
return ViewIfStudentLoggedIn();
```

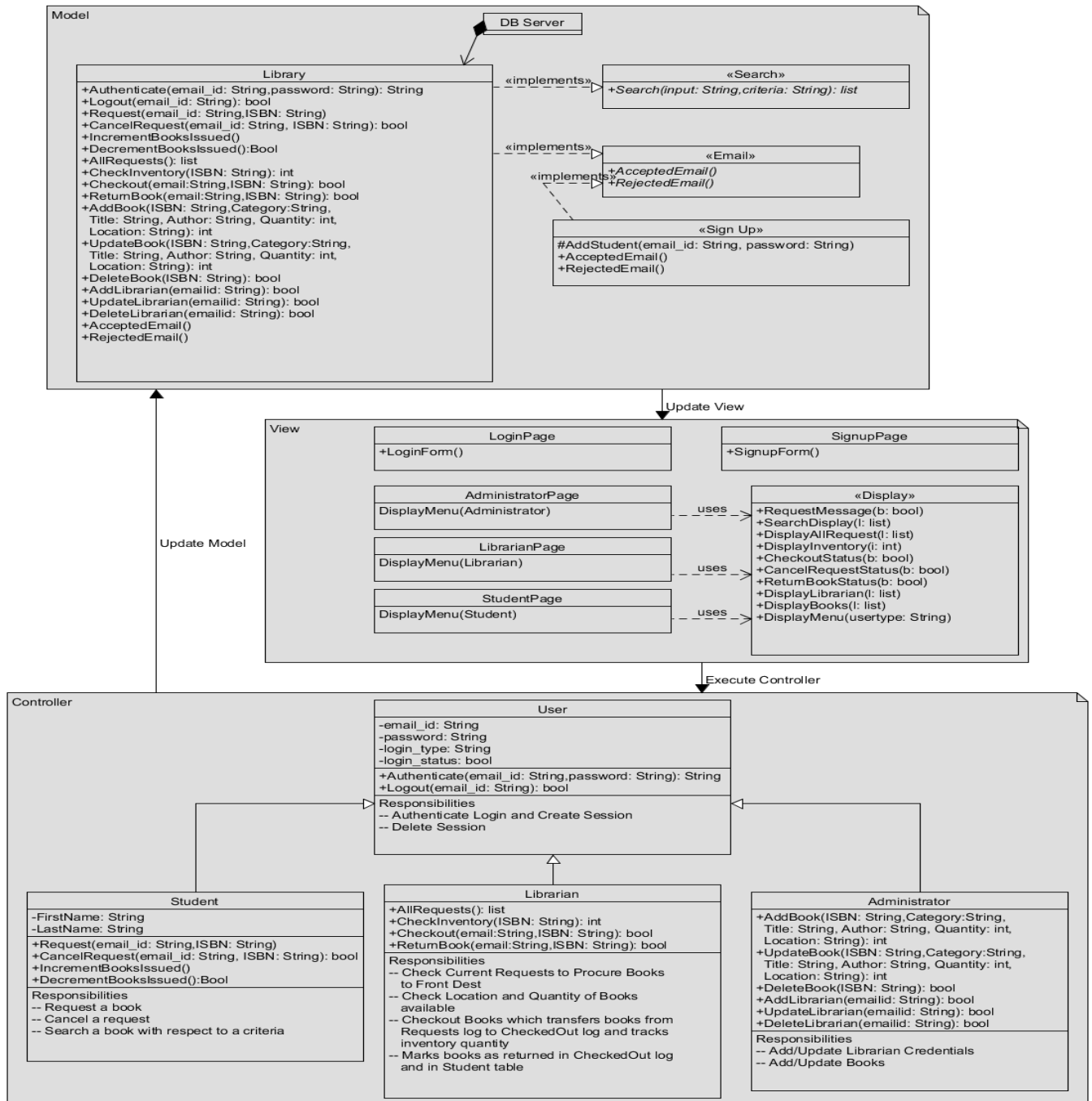
So we can avoid unwanted creation of objects.



Singleton Pattern – Composed of single instances of other models

3. Compare your Part 2 class diagram and your final class diagram - include part 2 class diagram and point out the necessary changes. You can answer why you didn't realize you needed things during the design and/or how next time you will be able to make a better design after this learning experience.

Shown below is the class diagram from Part 2 of our project



From the old class diagram, we had to change a couple of things:

1. We had a monster Library class which violated many object oriented design principles for example the single responsibility principle.
2. Though we used the inheritance in the Controller right with the User being the parent class and the Student and others being child classes, we added in a lot of unnecessary methods which was not specified in the Requirements.
3. The View classes had to change because the classes defined did not make sense in the context of a framework such as ASP.NET as opposed to DJANGOs class based views
4. There was no clear distinction between student, admin and librarian in the Model part of the design.

During the first run of the design, we didn't realise these because :

1. We were not sure about using the ASP.NET framework which is a framework for MVC that has its own classes which our defined classes can inherit and that implements a lot of functionality for us. This changed a lot of the design decisions.
2. We were not exposed to many design patterns, and couldn't recognise the need for one immediately at that point of time.
3. It was a first run for us and we were ok with making mistakes so that we could revise and learn from them.

Having taken this course and had first hand experience trying to implement a project with design as focus and since we have spent a lot of time discussing design decisions in brainstorm sessions, we think we will have a better eye for software design in the future. Some of us are cursed with the design eye forever.

4. What did you learn about analysis and design?

Over the period of this course and working on the project, our team learnt some valuable lessons:

1. We can avoid some common mistakes that we could make during coding by thinking about the design beforehand
2. Mastering analysis and design is more about practice rather than listening to lectures.
3. We could all agree that the analysis and design part was harder than we thought it would be.
4. Analysis and design gives us a proper structure on paper as well as in our brains to tackle software engineering projects better.