LIBRARY MANAGEMENT SYSTEM

Project Part 4: Final Report

CSCI5448

Sanketh Shetty

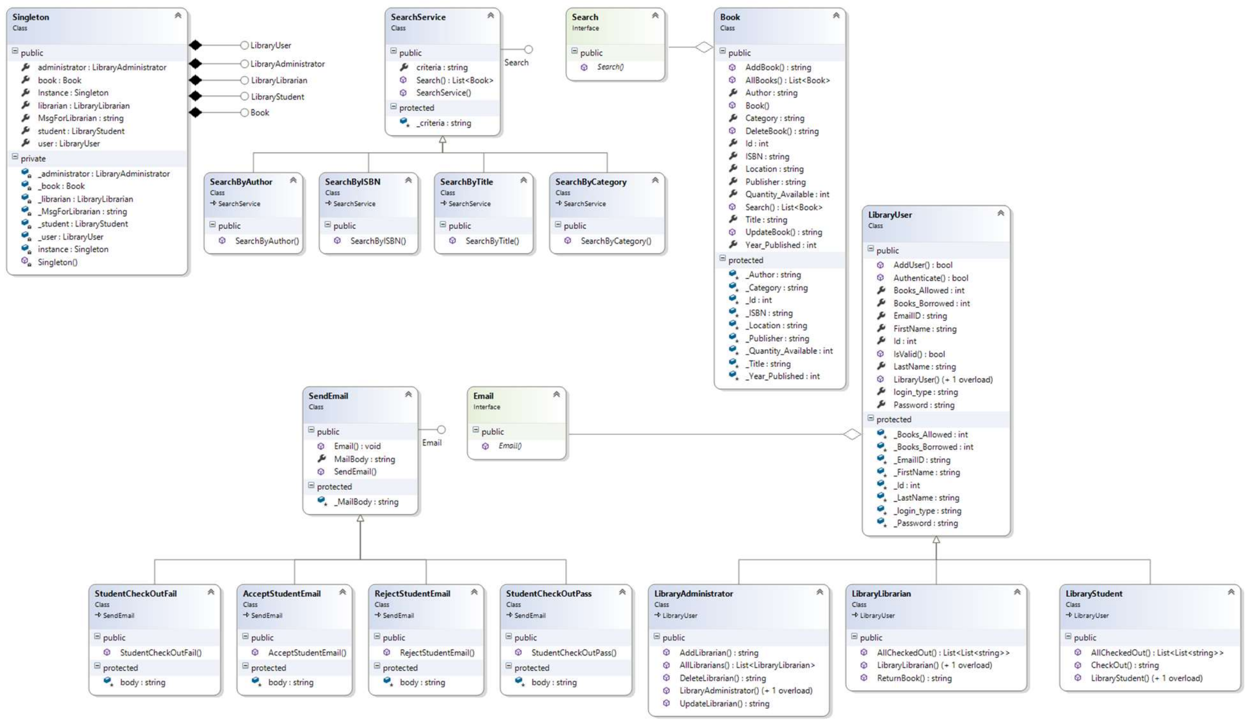
Gaurav Shukla

Tauseef Indikar

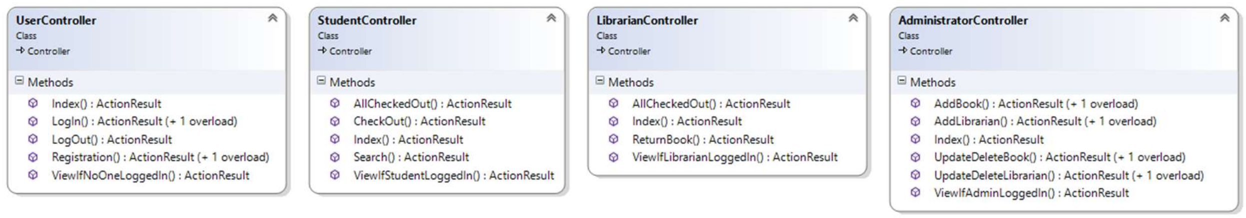
Nicolas Neitzel

1. What features were implemented and a class diagram showing the final set of classes and relationships of the system. *(This may have changed from what you originally anticipated from earlier submissions).* Discuss what changed in your class diagram and why it changed, or how it helped doing the diagrams first before coding if you did not need to change much.

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



Controllers Class Diagram


UserController
Class
Controller
Methods
Index() : ActionResult
LogIn() : ActionResult (+ 1 overload)
LogOut() : ActionResult
Registration() : ActionResult (+ 1 overload)
ViewIfNoOneLoggedIn() : ActionResult
StudentController
Class
Controller
Methods
AllCheckedOut() : ActionResult
CheckOut() : ActionResult
Index() : ActionResult
Search() : ActionResult
ViewIfStudentLoggedIn() : ActionResult
LibrarianController
Class
Controller
Methods
AllCheckedOut() : ActionResult
Index() : ActionResult
ReturnBook() : ActionResult
ViewIfLibrarianLoggedIn() : ActionResult
AdministratorController
Class
Controller
Methods
AddBook() : ActionResult (+ 1 overload)
AddLibrarian() : ActionResult (+ 1 overload)
Index() : ActionResult
UpdateDeleteBook() : ActionResult (+ 1 overload)
UpdateDeleteLibrarian() : ActionResult (+ 1 overload)
ViewIfAdminLoggedIn() : ActionResult

The final functionality implemented is

1. Student Signup
2. Student Login
3. Librarian Login
4. Admin Login
5. Admin can add, edit and delete a Librarian
6. Admin can add, edit and delete books
7. Student Search functionality w.r.t. criteria
8. Student can check out books
9. System emails a student on checkout and signup
10. Librarian takes a returned book
11. System manages Number of Available books for Student

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 with a few changes. Some additional functionality that we have implemented can be considered as "additions to requirements" for the project.

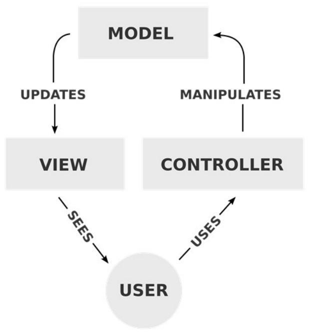
These changes were done keeping in mind that this is an Object Oriented Design class and the focus should be on learning and implementing design patterns as our initial design had not incorporated patterns apart from MVC framework usage. As per Prof. Liz Boese's suggestion for optimization we got rid of the request table and allowed the student to directly checkout books and hence the missing Librarian checking out functionality.

As these changes were evident from evaluation we had ample time to redesigned the Class Diagram. From an implementation point of view not much was changed. From the class diagram point of view, the changes made are discussed in Q3.

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?

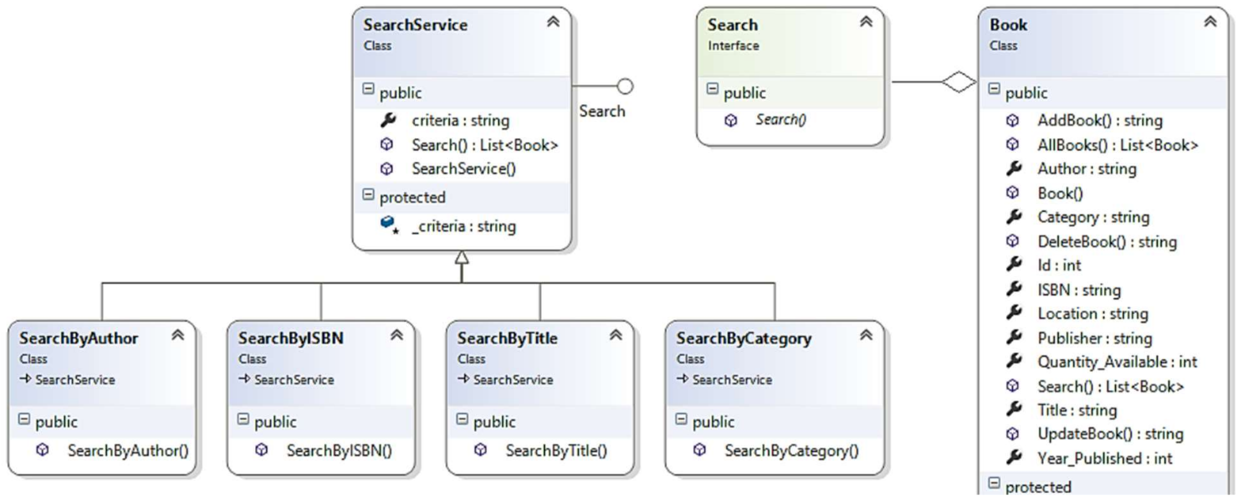
As it can be seen from the Class Diagrams, the overarching architectural design pattern used in our project is the MVC (Model-View-Controller) pattern. We use the ASP.NET framework and MS SQL for the data persistence. The model contains classes that store data that is retrieved according to commands 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.



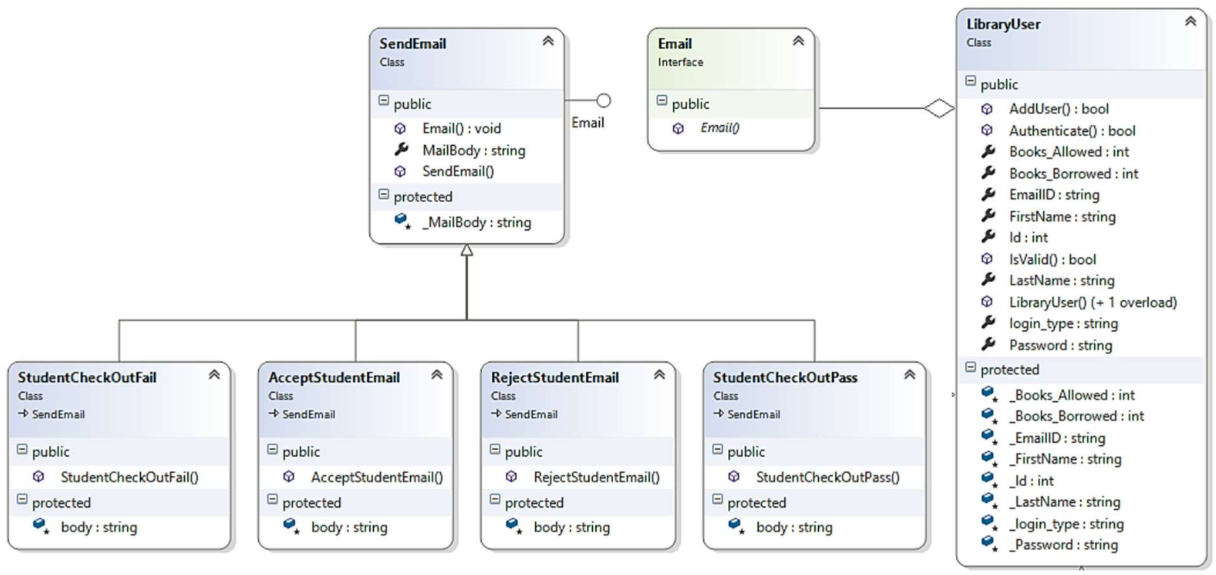
We learnt C# and the ASP.NET MVC framework. Using this framework made our project very Modular. We have created four controllers for one for each type of website i.e. User, Administrator, Librarian and Student. The routing is done so as to prevent access to other type of user's controller actions. The 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.

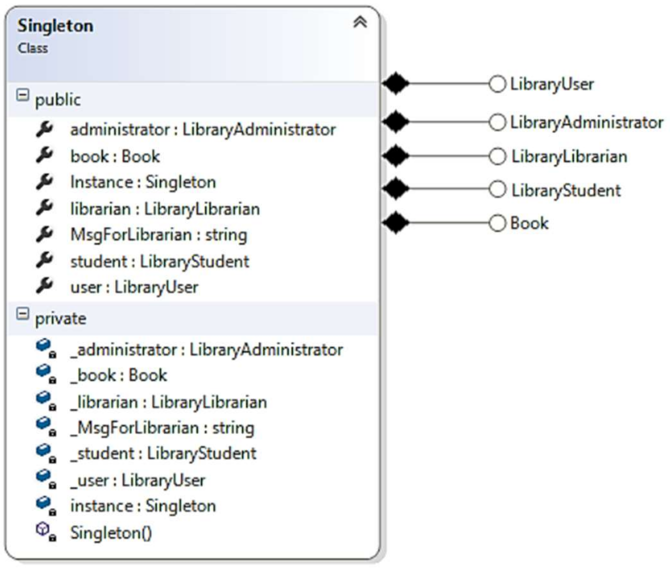
Also we have used the **Singleton pattern** to have a single ready instance of all our basic models available at runtime so that we would not have to instantiate each of them every time the controller is called. Also the singleton instance is used to keep track of when the last Student user logged in 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.

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

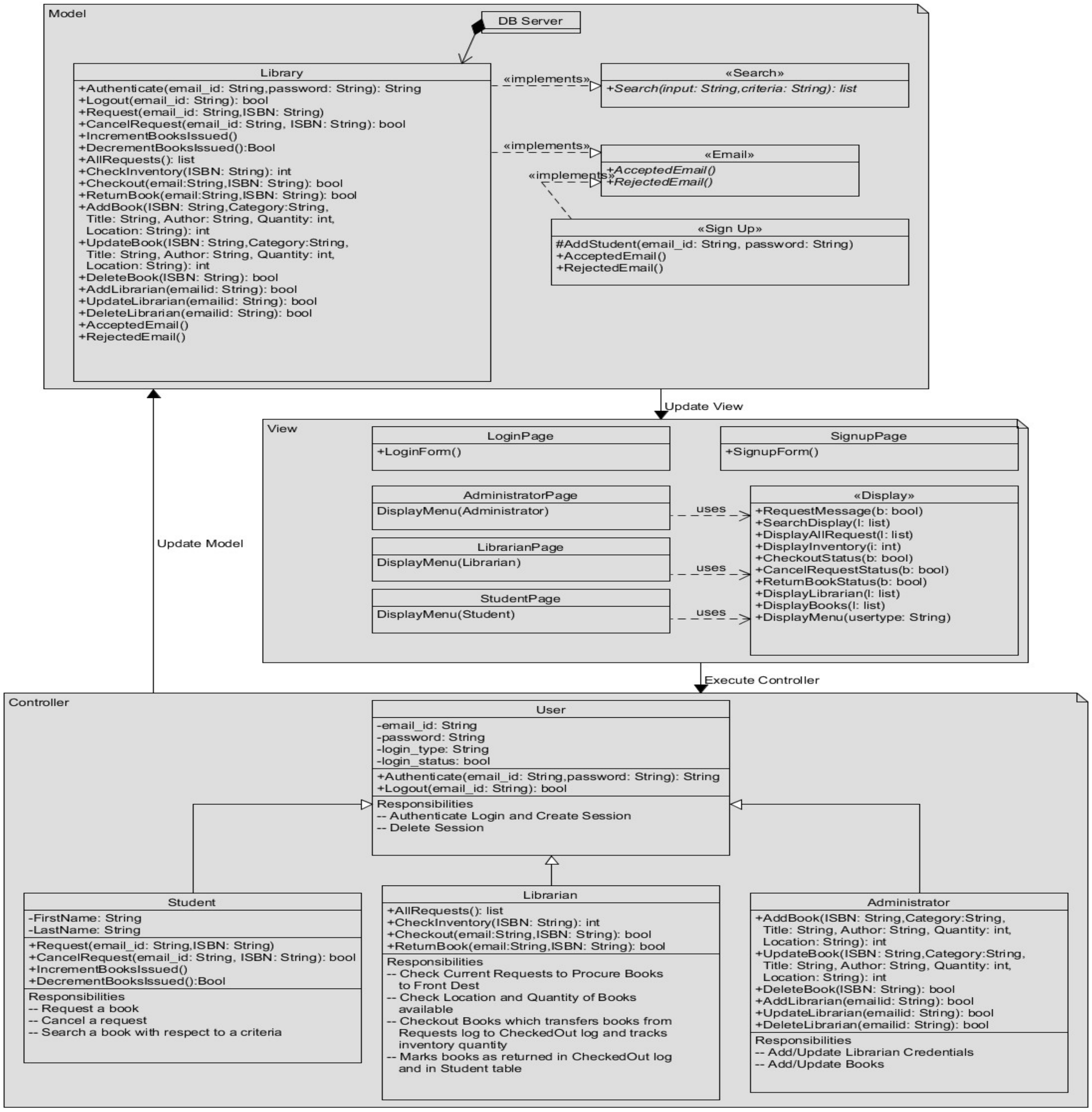
1. We had a monster Library class which violated many object oriented design principles for example the single responsibility principle.
2. Though we used the inheritance in the Controller right with the User being the parent class and the Student and others being child classes, we added in a lot of unnecessary methods which were not required to meet 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. As per Prof. Liz Boese’s suggestion for optimization we got rid of the request table and allowed the student to directly checkout books and broke the Library model down.

We didn’t realize 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 recognize the need for one immediately at that point of time.

If we had to do this project again we would definitely make use of the entity framework along with ASP.NET MVC so as to have an ORM talk to the database instead of writing Stored procedures and calling them each time. Also we would make the code more design centric so that future extensions are easier to implement and people use them as a basis for startup websites.

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


Model
DB Server
Library
+Authenticate(email_id: String,password: String): String
+Logout(email_id: String): bool
+Request(email_id: String,ISBN: String)
+CancelRequest(email_id: String, ISBN: String): bool
+IncrementBooksIssued()
+DecrementBooksIssued():Bool
+AllRequests(): list
+CheckInventory(ISBN: String): int
+Checkout(email:String,ISBN: String): bool
+ReturnBook(email:String,ISBN: String): bool
+AddBook(ISBN: String,Category:String, Title: String, Author: String, Quantity: int, Location: String): int
+UpdateBook(ISBN: String,Category:String, Title: String, Author: String, Quantity: int, Location: String): int
+DeleteBook(ISBN: String): bool
+AddLibrarian(emailid: String): bool
+UpdateLibrarian(emailid: String): bool
+DeleteLibrarian(emailid: String): bool
+AcceptedEmail()
+RejectedEmail()
«implements»
«Search»
+Search(input: String,criteria: String): list
«implements»
«Email»
+AcceptedEmail()
+RejectedEmail()
«implements»
«Sign Up»
#AddStudent(email_id: String, password: String)
+AcceptedEmail()
+RejectedEmail()
Update View
Update Model
View
LoginPage
+LoginForm()
SignupPage
+SignupForm()
AdministratorPage
DisplayMenu(Administrator)
LibrarianPage
DisplayMenu(Librarian)
StudentPage
DisplayMenu(Student)
uses
uses
uses
«Display»
+RequestMessage(b: bool)
+SearchDisplay(l: list)
+DisplayAllRequest(l: list)
+DisplayInventory(i: int)
+CheckoutStatus(b: bool)
+CancelRequestStatus(b: bool)
+ReturnBookStatus(b: bool)
+DisplayLibrarian(l: list)
+DisplayBooks(l: list)
+DisplayMenu(usertype: String)
Execute Controller
Controller
User
-email_id: String
-password: String
-login_type: String
-login_status: bool
+Authenticate(email_id: String,password: String): String
+Logout(email_id: String): bool
Responsibilities
-- Authenticate Login and Create Session
-- Delete Session
Student
-FirstName: String
-LastName: String
+Request(email_id: String,ISBN: String)
+CancelRequest(email_id: String, ISBN: String): bool
+IncrementBooksIssued()
+DecrementBooksIssued():Bool
Responsibilities
-- Request a book
-- Cancel a request
-- Search a book with respect to a criteria
Librarian
+AllRequests(): list
+CheckInventory(ISBN: String): int
+Checkout(email:String,ISBN: String): bool
+ReturnBook(email:String,ISBN: String): bool
Responsibilities
-- Check Current Requests to Procure Books to Front Dest
-- Check Location and Quantity of Books available
-- Checkout Books which transfers books from Requests log to CheckedOut log and tracks inventory quantity
-- Marks books as returned in CheckedOut log and in Student table
Administrator
+AddBook(ISBN: String,Category:String, Title: String, Author: String, Quantity: int, Location: String): int
+UpdateBook(ISBN: String,Category:String, Title: String, Author: String, Quantity: int, Location: String): int
+DeleteBook(ISBN: String): bool
+AddLibrarian(emailid: String): bool
+UpdateLibrarian(emailid: String): bool
+DeleteLibrarian(emailid: String): bool
Responsibilities
-- Add/Update Librarian Credentials
-- Add/Update Books

4. What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

Over the period of this course and working on the project, we learnt:

1) The importance and function of each UML model throughout the process of object-oriented analysis and design and explaining the notation of various elements in these models.
2) Analysis and design gives us a proper structure on paper as well as in our brains to tackle software engineering projects better.
3) Design helps us give structure to our ideas before we translate it to code.
4) Design helps us develop software that can withstand changes in terms of requirements.
5) Design helps us develop software that is scalable.
6) Design is Programming language agnostic; therefore, we can later use the design in a different language which has same features (different OO languages).
7) 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.
8) Brainstorming leads to some amazing ideas.
9) We can avoid some common mistakes that we could make during coding by thinking about the design beforehand.

Having taken this course and had firsthand experience in implementing a project with design as focus and having spent a lot of time discussing design decisions in brainstorm sessions, we believe we have a better eye for software design in the future. All of us are now cursed with the design eye forever.