Technical Specification

Bookworm

By Simon Lowry

Contents:

1. Introduction

The project will be a .Net web application where you can review books, have a customizable and unique profile and gain knowledge of other books that you might be interested in using a recommender system. It be structured with 5 layers, the first three of which will be made up of the MVC pattern (model, view, controller), followed by the service layer and finally the repository layer.The service and repository layers will carry out all database interactions. This 5 layered approach will enable the system to be highly maintainable and suited to a variety of testing methods. I intend on using tools such as FakeItEasy and NUnit to enhance the testing process.

Motivation:

I want to make a system which is highly visual and visceral, which adheres to web usability standards and practices. I want the user to have an experience that has them coming back for more. To create this experience, the user will have a customizable profile where they can express all of their book related preferences. They will be able to see and view what their friends have read and connect with all kinds of book lovers.

As someone who loves reading, I can understand the frustration of the situations where you have nothing to read. The laborious process of scrolling through book covers and websites in order to find your next book can be time consuming. Having a tailored recommender system which can make suggestions would take the sting away from this situation providing the user with an abundance of options that are specifically suited to the preferences of the reader. BookWorm will provide the user with this recommender system.

Problems Solved:

A major problem encountered was importing the kaggle data set [2] into the database. This encountered a number of errors due to the quality of data not being particularly suited for a database. There was hundreds of entries which used characters which were not accepted by SQL Server Management Studio and it was as result rejecting the import and there was also a lot of missing data. This lead to having to write a script which would identify and remove any non alphabetic characters in a number of the csv files which contained the kaggle data. The script also tied to the user ids from the data to some fake users with more extensive information so that they could be used easier when evaluating and running the system. Eventually this resolved the problem and the data was able to be imported into the database.

Another big problem resolved was the selection of the recommender system. Initially I had chosen to use the recommender system MyMediaLite. This was due to the fact that was open source and there wasn’t a whole lot of free options available otherwise. The further I used the system the more I realised it’s drawbacks. This is due to the fact that it’s not extensively used, so there’s nothing on stackoverflow related to it, the documentation it is minimal and examples are very few so when it came to debugging any issues when evaluating the output of the system it became very difficult to be able to do that and ensure that it was producing quality output. This led to changing the recommender to Nreco, an apache mahout port which is native to Java. Nreco was a much better choice. With the examples abounding in apache mahout and Nreco being so similar to it’s implementation, getting up to speed in it was a lot easier. The documentation was much clearer and the structure of the code much easier to understand and the results of what it produces much easier to validate and verify.

Initially I also had great difficulty setting up SQL Server Management Studio. In order to get a local database running you had to install a number of different components and perform some operations on the command line to get the instance up and running as well as performing firewall operations to allow traffic from certain ports to be allowed and a number of other configuration settings which made this a very tedious and frustrating process which cost me a couple of days.

Research:
A vast amount of research was carried out in order to scaffold the various components of the architecture in Bookworm. In particular, setting up things like the code-first entity framework as well as the repository pattern. This wasn't something I'd done before so it required many visits to vast tutorials and use of QA forums like stackoverflow in order to get the system up and running correctly. A lot of reading was carried out on recommender systems and how to utilize and understand the concepts behind them as well as the various different flavours they can come in. When it came to using Apache Mahout I also watched various videos online from courses which gave me some inside into it's workings and made use of the book Mahout in Action [3] to get a greater understanding of how it operates and how I might go about evaluating it and what would be the best way to do so. Initially I had also spent a considerable amount of time trying to make sense of the original

Out of Scope
In the functional specification there were a few functions that were noted as ones which were not primary to the system but would be carried out if time allowed. This included functionality related to uploading and working with kindles. This was not implemented in the system due to time constraints.

2. Components

Architecture:

5 Layers:
- Model
- View
- Controller
- Service
- Repository

The selection to go with the 5 layers was based on implementing on what I had learned on my 6 month work placement where I was developing .Net web applications for a few production systems at Greenfinch Technology. Experiencing how beneficial it is to have such a good structure in place made it a must have for my own architectural choice for Bookworm. In this architecture the first 3 layers are made of the MVC pattern. MVC (Model, View Controller) provides a clear divide between model view and controller and allows for each to take up a distinct role. This keeps it so that each layer serves one functional purpose (e.g. handling client requests) sequestered from code that serves an entirely different functional purpose (e.g. representing data). While it does have some drawbacks with the lack of data hiding, it makes for a system in which code is much easier to maintain, refactor and improve upon. It makes things really simple All three layers are built to handle a specific aspect of the web applications purpose. Controller receives all requests for the application and then instructs the model to prepare any information required by the view. The view uses that data prepared by the controller to bring the final output.

Model:

The model represents the data to the user and is very important to the system. This level defines where the application’s data objects are stored. The model has no idea about anything related to either the view or the controller. The models in this system may be either a single object or a structure of objects.

Views:

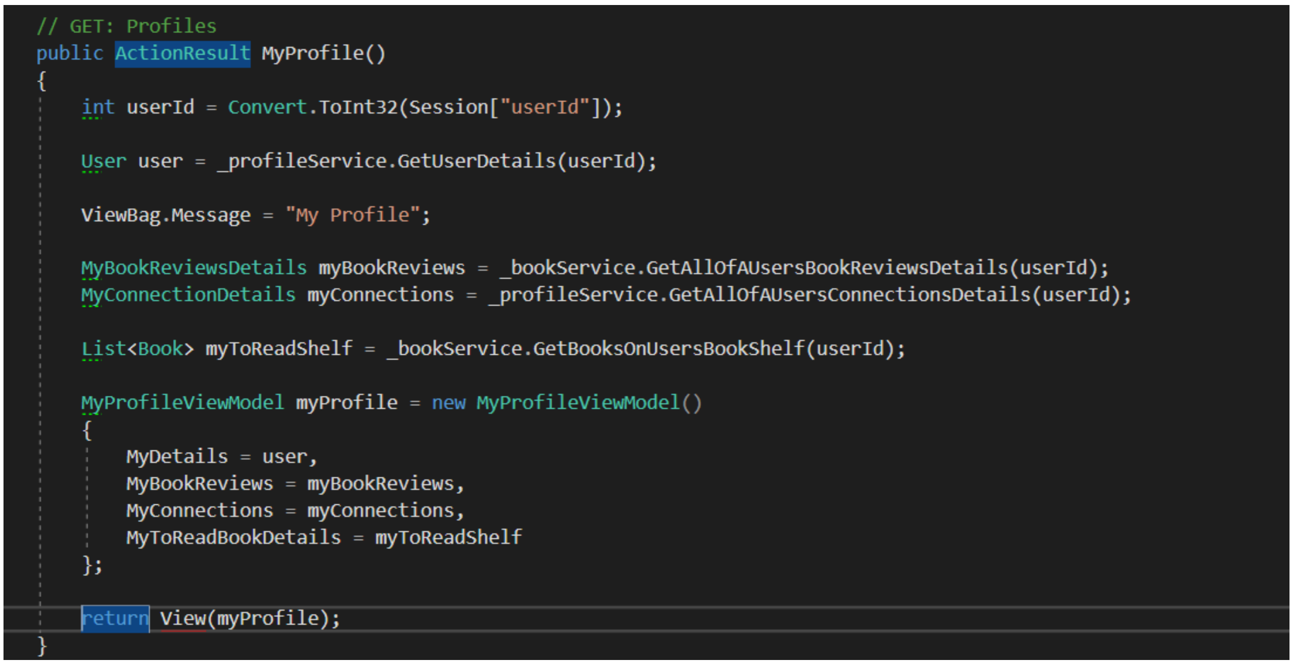
This level creates an interface to show the actual output to the user which is provided by a view model which makes use of a number of models to make up what is required to be displayed on a given screen. The view doesn’t display anything without the help of the controller directing what it displays..The view also handles requests from the user and informs the controller as to which actions to take. The actions will mostly be called through the either of use of buttons or through a form.

The UI will make use of HTML, CSS, javascript, Jquery, bootstrap and MVC Razor syntax.   Razor allows you to write mix of HTML and server side code using C#. The C# code is highly used to determine what should be displayed for the given section of the page. For example on the book profile page, the C# is determining which buttons to display, whether it’s the create review button or the edit review button. This is determined by assessing the given model which includes a property which notifies as the view as to whether or not the user has already created a view for this.

The javascript and Jquery handles most of the dynamic operations in the Views such as fading in and out alert notifications of the user as to when certain actions have taken place. It’s also used to send for the form handling operations and getting the ratings for when the user selects a particular star rating for a book review. The CSS handles all the styling for each view while the bootstrap is heavily used from the grid structure which gives the page it’s structure to the bootstrap buttons, modals and forms. This is simplified some of the UI sections and made for improving the quality of the presentation.

Controller:

Controllers act as an interface between Model and View components. It processes all the business logic and incoming requests, manipulate data using the Model component, and interact with the Views to render the final output. It receives input and initiates a response by making calls on model objec Here’s an example of a method from the profiles controller:

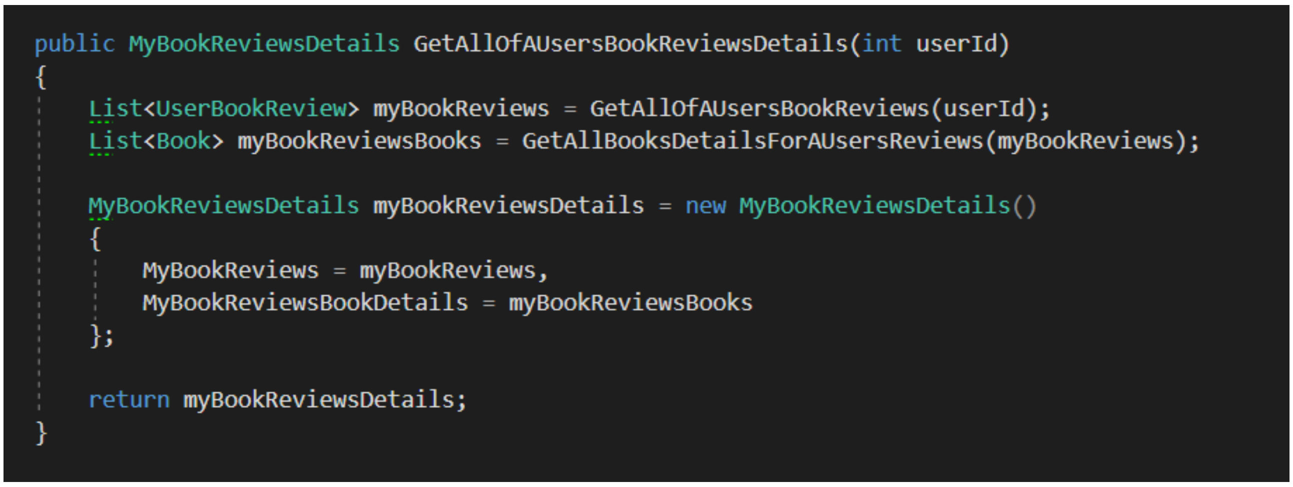


```
// GET: Profiles
public ActionResult MyProfile()
{
    int userId = Convert.ToInt32(Session["userId"]);

    User user = _profileService.GetUserDetails(userId);

    ViewBag.Message = "My Profile";

    MyBookReviewsDetails myBookReviews = _bookService.GetAllOfAUsersBookReviewsDetails(userId);
    MyConnectionDetails myConnections = _profileService.GetAllOfAUsersConnectionsDetails(userId);

    List<Book> myToReadShelf = _bookService.GetBooksOnUsersBookShelf(userId);

    MyProfileViewModel myProfile = new MyProfileViewModel()
    {
        MyDetails = user,
        MyBookReviews = myBookReviews,
        MyConnections = myConnections,
        MyToReadBookDetails = myToReadShelf
    };

    return View(myProfile);
}
```

In this method, initially we obtain the userId from the session information. The session information is set for a logged in user when they successfully enter in their login details on the login screen. The rest of this method is constructing the view model which will used to display all the information for a given user's profile.

Service

The remaining two layers handle all of the database related operations. Again these layers are linked to the other layers while sustaining their independence. The service layer provides a unique implementation of a particular set of database related operations which are related to one particular part of the project. For example, the Book Service handles all operations that are related to books, that includes the models Book, ToRead and UserBookReviews. Here's an example of a method from the Book Service.



```
public MyBookReviewsDetails GetAllOfAUsersBookReviewsDetails(int userId)
{
    List<UserBookReview> myBookReviews = GetAllOfAUsersBookReviews(userId);
    List<Book> myBookReviewsBooks = GetAllBooksDetailsForAUsersReviews(myBookReviews);

    MyBookReviewsDetails myBookReviewsDetails = new MyBookReviewsDetails()
    {
        MyBookReviews = myBookReviews,
        MyBookReviewsBookDetails = myBookReviewsBooks
    };

    return myBookReviewsDetails;
}
```

In this method a particular user id is sent to the method to create a view model called MyBookReviewDetails. MyBookReviewDetails acts as a container for all of a users book reviews as well as the book's data itself in order to display the Book's image for example as well.

Repository

The service layer utilizes a set of generic operations that can be carried out on a particular entity or table in the database. The use of the repository pattern makes for a significant reduction in code duplication. The repository contains operations that are used time and time again such as create and delete. Each service is able to make use of the repositorty as it's made to be generic and type independent.

Database
The database itself is stored in MSSQLServerManagement Studio.

Database Schema Changes
All changes to the database structure are carried with migrations. All of these migrations are stored to be viewed or rolled back at any time in the project folder called migrations.

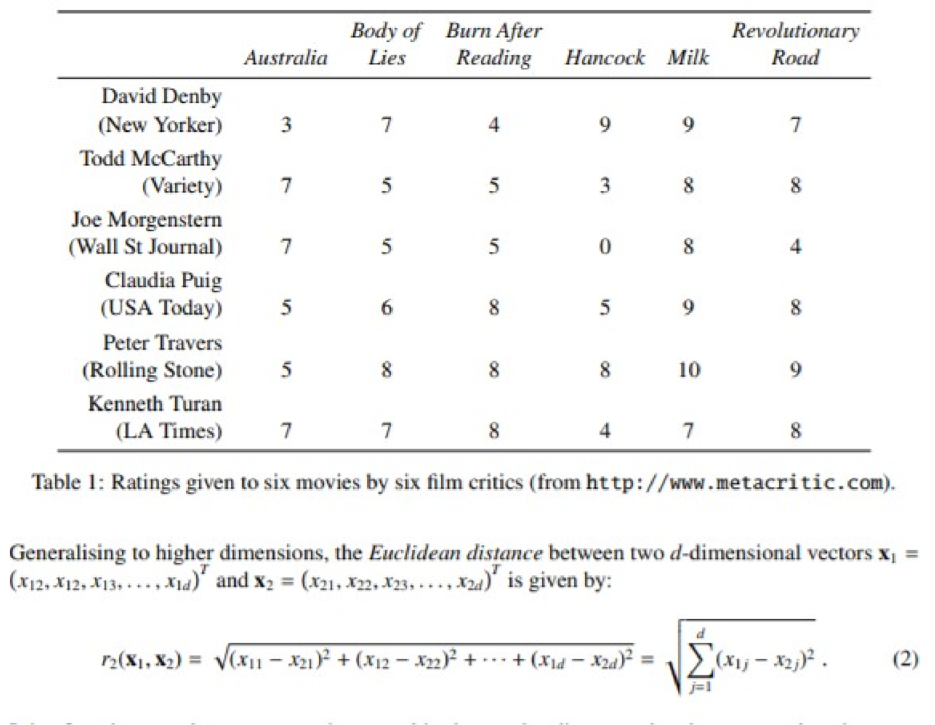
Recommender System:
This was a core part of the system.

Collaborative filtering methods analyze large amount of information about preferences of users and predict preferences of similar users for recommending items. In collaborative filtering method an accurate prediction of preferences of a user and recommendation of items is possible without any need for detailed analysis of item features. A basic assumption in collaborative filtering is that users would like similar kinds of items as they have liked in past. Collaborative filtering methods suffer from issues like – cold start, scalability and sparsity.

3.2 Similarity Measures

A similarity measure or similarity function is a real-valued function that quantifies the similarity between two objects. Although no single definition of a similarity measure exists, usually similarity measures are in some sense the inverse of distance metrics: they take on large values for similar objects and either zero or a negative value for very dissimilar objects.

The measures for similarity that I used to compare in the system are Euclidean distance, Pearson Coefficient, Spearman, log likelihood, tanimoto. The Euclidean distance is the one that was selected for this system due to it's performance values as outlined in the blog.



	Australia	*Body of Lies*	*Burn After Reading*	*Hancock*	*Milk*	*Revolutionary Road*
David Denby (New Yorker)	3	7	4	9	9	7
Todd McCarthy (Variety)	7	5	5	3	8	8
Joe Morgenstern (Wall St Journal)	7	5	5	0	8	4
Claudia Puig (USA Today)	5	6	8	5	9	8
Peter Travers (Rolling Stone)	5	8	8	8	10	9
Kenneth Turan (LA Times)	7	7	8	4	7	8

Table 1: Ratings given to six movies by six film critics (from `http://www.metacritic.com`).

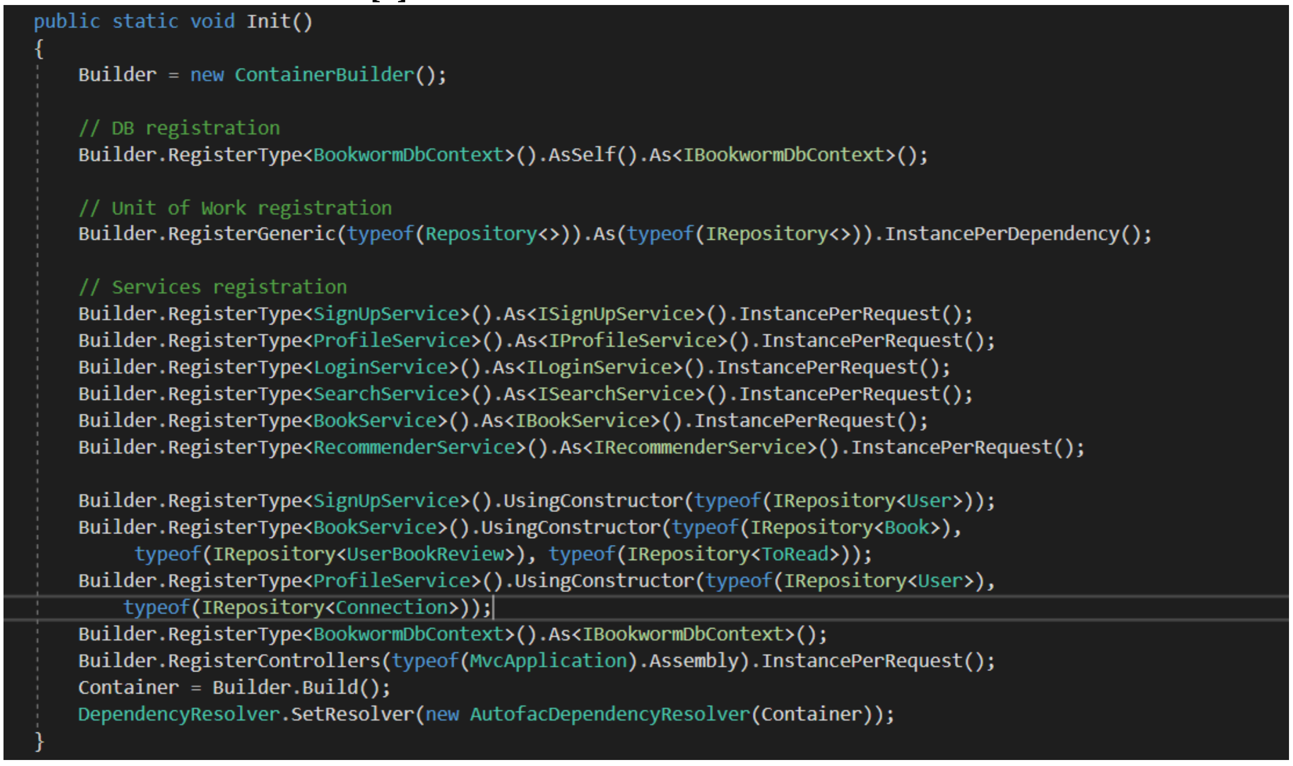
Generalising to higher dimensions, the *Euclidean distance* between two d-dimensional vectors $\mathbf{x}_1 = (x_{12}, x_{12}, x_{13}, \ldots, x_{1d})^T$ and $\mathbf{x}_2 = (x_{21}, x_{22}, x_{23}, \ldots, x_{2d})^T$ is given by:

$$r_2(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{(x_{11} - x_{21})^2 + (x_{12} - x_{22})^2 + \cdots + (x_{1d} - x_{2d})^2} = \sqrt{\sum_{j=1}^{d} (x_{1j} - x_{2j})^2} \, . \qquad (2)$$

Euclidean similarity is the opposite of the Euclidean distance. It is proportional to the length of a line drawn between the edges of the two vectors. This similarity metric favors distance over direction and makes no adjustments for the actual ratings. We derive Euclidean distance from the dot product and the norms of the two vectors.

Other Tools and Components Used in the System:

IOC Containers – Autofac [1]



```
public static void Init()
{
    Builder = new ContainerBuilder();

    // DB registration
    Builder.RegisterType<BookwormDbContext>().AsSelf().As<IBookwormDbContext>();

    // Unit of Work registration
    Builder.RegisterGeneric(typeof(Repository<>)).As(typeof(IRepository<>)).InstancePerDependency();

    // Services registration
    Builder.RegisterType<SignUpService>().As<ISignUpService>().InstancePerRequest();
    Builder.RegisterType<ProfileService>().As<IProfileService>().InstancePerRequest();
    Builder.RegisterType<LoginService>().As<ILoginService>().InstancePerRequest();
    Builder.RegisterType<SearchService>().As<ISearchService>().InstancePerRequest();
    Builder.RegisterType<BookService>().As<IBookService>().InstancePerRequest();
    Builder.RegisterType<RecommenderService>().As<IRecommenderService>().InstancePerRequest();

    Builder.RegisterType<SignUpService>().UsingConstructor(typeof(IRepository<User>));
    Builder.RegisterType<BookService>().UsingConstructor(typeof(IRepository<Book>),
         typeof(IRepository<UserBookReview>), typeof(IRepository<ToRead>));
    Builder.RegisterType<ProfileService>().UsingConstructor(typeof(IRepository<User>),
        typeof(IRepository<Connection>));
    Builder.RegisterType<BookwormDbContext>().As<IBookwormDbContext>();
    Builder.RegisterControllers(typeof(MvcApplication).Assembly).InstancePerRequest();
    Container = Builder.Build();
    DependencyResolver.SetResolver(new AutofacDependencyResolver(Container));
}
```

Code for the autofac based IOC container in Bookworm

Autofac is an open source tool that allows you to build up containers with lambdas, types, or pre-built instances of components. In Bookworm it injects the constructor parameters for the system. By making use of dependency injection it can produce an architecture which is less tightly coupled and therefore becomes implementation independent. The implementation I’m referring to is whatever class expands from the interfaces which outlines a particular setting of requirements that are to be implemented. It alsp leads to a reduction in code, makes it much easier to perform testing and this approach also favours simplicity.

Test Plan

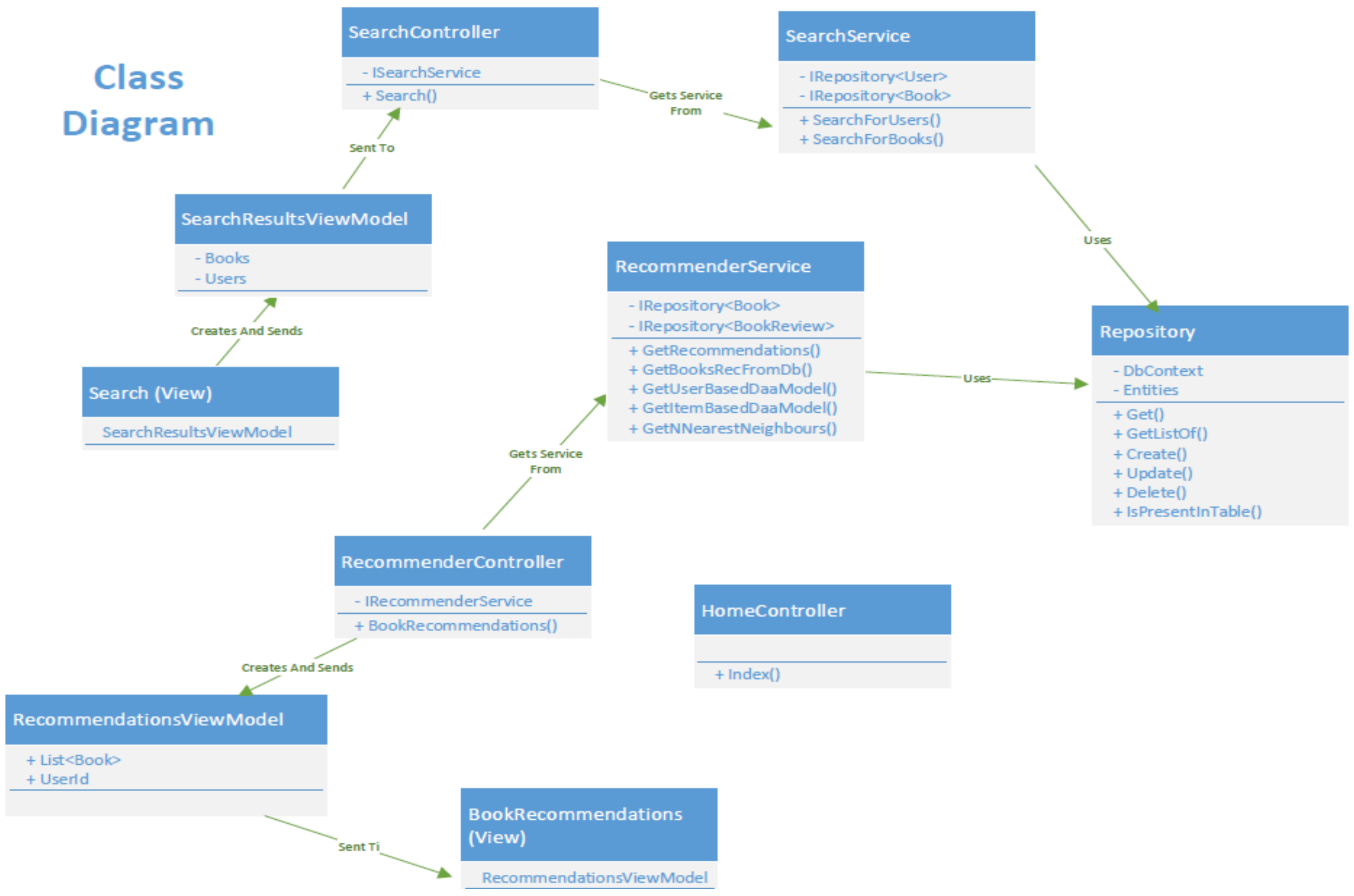
The types of testing carried out on the system was predominantly automated testing in the form of Unit Testing, Integration Testing and Component Testing. Some user testing was also carried. In the user testing users were asked to basically use the system and just play around with the system and report back what they liked and didn't like. This was very fruitful especially for the user interface design. It allowed me to get good critical feedback on the parts of the system that might be improved upon. One such example of the feedback I received which lead to a change in the system was the criticism that the book profile UI looked too plain and needed more too it. As a result of this I made changes to that UI and . They were then asked to perform a list of certain primary functions of the system and this also led one or two UI bugs being highlighted and then fixed (a few links were pointing to the wrong folder).

Nunit, fakeiteasy, mock objects, fake calls, modularity

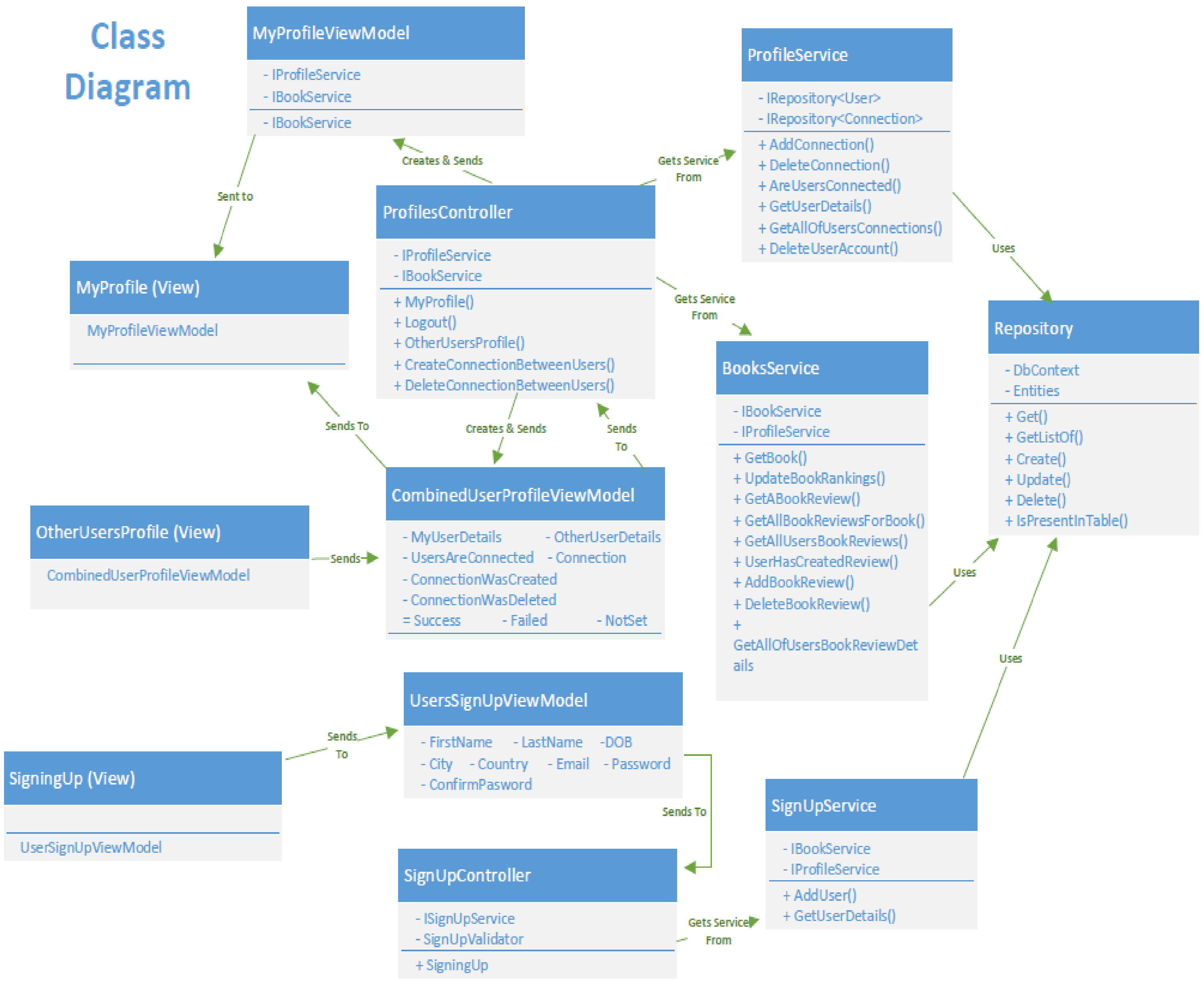
Ways to Extend/Improve the System Beyond Project Deadline:

- Kindle Highlights upload, presentation and ability to modify from Bookworm UI.
- Improved profile page
    - ability to select favourite quotes and have them displayed on an accordion.
    - ability to select user's favourite genre, author's
- A separate section for Item Based Recommendations.
- Messaging between users
- Dynamic form validation which validates the form in real time using Javascript
- Comments on people's reviews they've left
- Ability to chat between connections
- "Forgot your password" functionality
- Bookworm email subscription which could keep the user in touch book recommendations or book releases of their favourite authors
- Encryption for logins

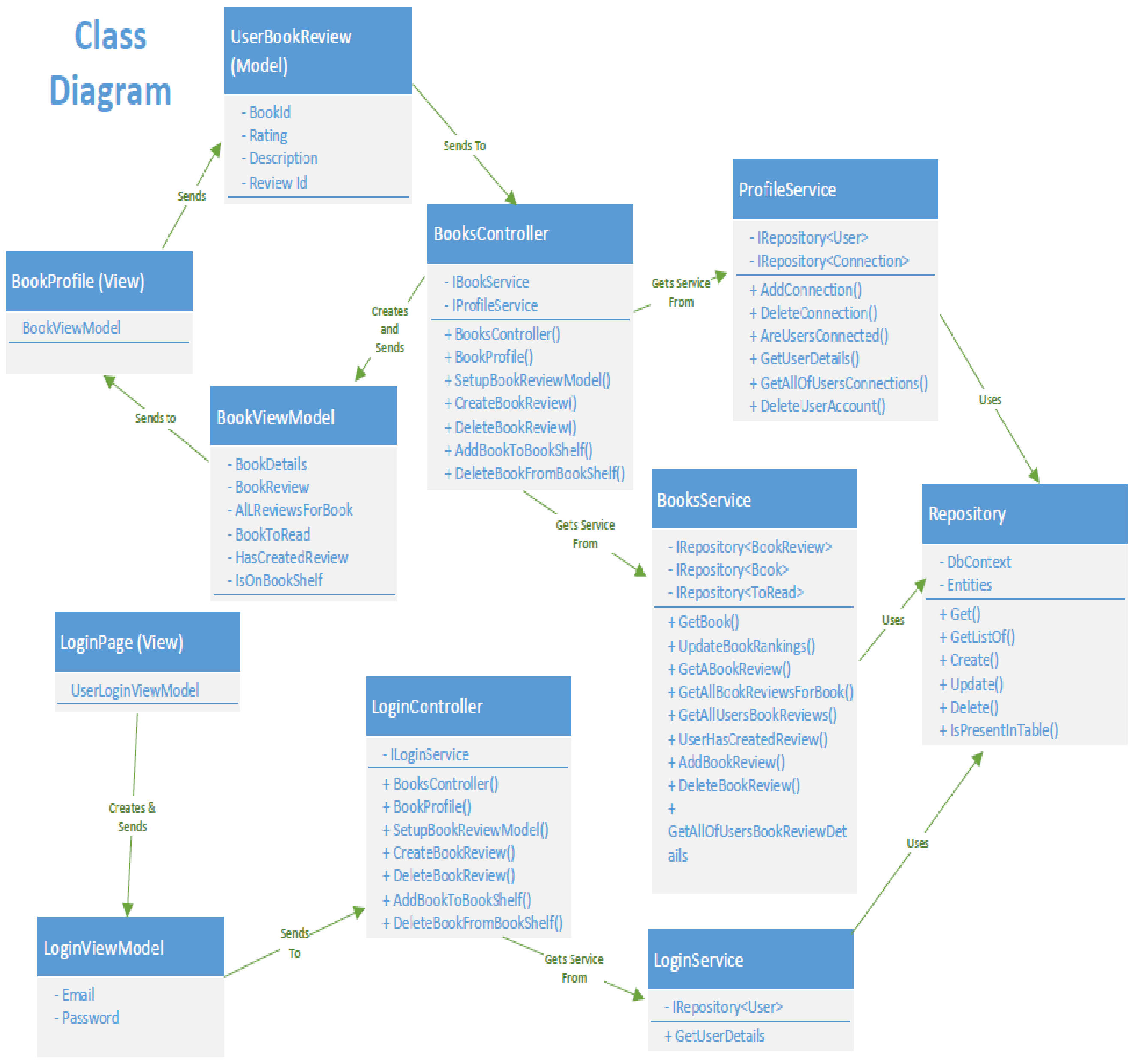
3. Detailed Class Design



Class Diagram



Class Diagram



4. Use Case View

This section contains all the use cases for the system.

USE CASE	**Sign Up**	
Goal in Context	Sign up for the service Bookworm	
Success End Condition	User has successfully signed up to use Bookworm.	
Failed End Condition	User is unable to fill in the signup for correctly.	
Trigger	A person decides they would like to sign up the web application Bookworm.	
DESCRIPTION	**Step**	**Action**
	1.	User navigates to the sign up page.
	2.	User enters all of the details in the sign up form.
	3.	User hits the sign up button.
User is presented with their newly made profile.		
EXTENSIONS	**Step**	**Branching Action**
	3 (b)	User fails to enter in the details correctly and is prompted as to what is wrong with the sign up details. The user re-enters their details and signs up for Bookworm.

USE CASE	**Login**	
Goal in Context	User is attempting to login to Bookworm.	
Success End Condition	The user successfully logs into the web application.	
Failed End Condition	The user is unable to locate login or fails to enter valid login details.	
Trigger	The user wishes to login to their Bookworm profile.	
DESCRIPTION	**Step**	**Action**

	1.	User navigates to the login page from the home page by clicking on the login link in the nav bar.
	2.	The user enters both their email and password.
	3,	The user hits the login button.
The user is presented with their profile.		
EXTENSIONS	**Step**	**Branching Action**
	3 (b)	User is unable to enter valid details. User is prompted to rectify their mistake. User re-enters their login details.

USE CASE	**Recommend Book Suggestions**	
Goal in Context	To find new books which may be suited for a given user.	
Success End Condition	The user finds books that they like which have been recommended to them.	
Failed End Condition	The user doesn’t receive any recommendations.	
Trigger	The user is looking for something new to read.	
DESCRIPTION	**Step**	**Action**
	1.	User navigates to login screen and enters login credentials.
User is presented with their profile.	2.	User clicks on “My Recommendeations” button in nav bar.
User is presented with their recommendations tailored specifically to them.	3.	User adds to book to To Read shelf.

EXTENSIONS	**Step**	**Branching Action**
	3 (b)	User is not presented with any recommendations or does not like any of the recommendations and decides to search out books themselves.

USE CASE	**Edit User Profile**	
Goal in Context	Update profile details.	
Success End Condition	User has updated their profile.	
Failed End Condition	User cancels updating their profile or an error has occurred while updating.	
Trigger	User wishes to change something about their profile.	
DESCRIPTION	**Step**	**Action**
	1.	User navigates to login page.
User is presented with login view.	2.	User enters their credentials.
User is presented with their profile.	3.	User clicks on edit profile button.
User is presented with edit profile view.	4.	User enters in updated details and hits the update profile button.
User successfully updates their profile and the details on the profile reflect this change.		
EXTENSIONS	**Step**	**Branching Action**
	4 (b)	An error occurred while updating the results and the user is notified of this.

USE CASE	**Connect With Other Users**

Goal in Context	“”	
Success End Condition	Other user accepts connection request and the two users become connected.	
Failed End Condition	User request is rejected by the other user	
Trigger	A user comes across someone they would like to connect with	
DESCRIPTION	**Step**	**Action**
	1.	User navigates to login page.
User is presented with their profile.	2.	User enters credentials and logs into their profile
	3.	User enters name of other user they want to connect with in search bar
User is presented with search results from the given query which includes the user they have selected.	4.	User finds the name of the user in the search results and clicks on that name.
	5.	User clicks on the add connection on the other user’s private profile.
Other User sees the connect request and accepts it and the two users become connected.		
EXTENSIONS	**Step**	**Branching Action**
	3 (b)	User is unable to find the other user they were looking for and stops searching
	5 (b)	The other user rejects the connection request and the two users remain unconnected.

USE CASE	**Create Book Reviews**
Goal in Context	To create a book review
Success End Condition	User successfully creates a book review.

Failed End Condition	User is unable to locate the book in question or fails to create the review.	
Trigger	User wants to leave a book review having read a book.	
DESCRIPTION	**Step**	**Action**
	1.	User navigates to login screen.
	2.	User enters credentials and logs in.
	3.	User enters the name of the book they want to leave a book review for into the search bar.
	4.	U
EXTENSIONS	**Step**	**Branching Action**
	3 (b)	The book the user wants to review is already on the user's book shelf. User navigates to their book shelf.
	3 (c)	User selects the book they want to leave a review.
User is presented with a modal where they can enter their book review details.	3 (d)	User hits the create a review button and successfully enters their review and then presses the submit button.

USE CASE	**Update Book Reviews**	
Goal in Context	Update a previously created book review.	
Success End Condition	User has updated their book review.	
Failed End Condition	User fails to update their book review and are notified of this failure.	
Trigger	User wishes to update a book review they have already created.	
DESCRIPTION	**Step**	**Action**
	1.	User logs in.
	2.	User searches out the book they are

		looking to update.
	3.	User clicks the link to the book profile in the search results.
	4.	Users hits the edit review button on the book profile.
	5.	User submits updated review.
EXTENSIONS	**Step**	**Branching Action**
	5 (b)	User hits the submit updated review button and an error occurs. The user is notified of this failure to update and their occurring.

USE CASE	**Search for Other Users**	
Goal in Context	To find other users.	
Success End Condition	Successfully found the users that the user was looking for	
Failed End Condition	User was unable to find the user they were looking for.	
Trigger	User wants to find a particular user	
DESCRIPTION	**Step**	**Action**
	1.	User Logins with their credentials
User is presented with their profile.	2.	User enters the name of the user they would like to find in the search bar.
User is presented with the search results.	3.	User selects the name of the user they were looking for within search results.
User is presented with the profile of the user they were searching for.		

EXTENSIONS	**Step**	**Branching Action**
	3. (b)	User is unable to find the user they were looking for and choose to do something else instead.

USE CASE	**Search for Books**	
Goal in Context	To find a particular book.	
Success End Condition	The user finds the book they were looking for.	
Failed End Condition	The user fails to find the book they were looking for.	
Trigger	The user has become motivated to find a particular book.	
DESCRIPTION	**Step**	**Action**
	1.	User navigates to the login page.
	2.	User enters their login credentials.
User manages to login successfully and is presented with their profile.	3.	User enters the title of the book they are looking for into the search bar.
The user is presented with search results which include the book the user was looking for.	4.	User clicks on the book they were looking for in the search results.
User is presented with the book profile for that book.		
EXTENSIONS	**Step**	**Branching Action**
	4 (b)	The user is unable to find the book they are looking for and gives up the search.

5. References:

[1] https://autofac.org/ - Autofac’s website

[2] https://www.kaggle.com/zygmunt/goodbooks-10k/data – Goodreads Dataset

[3] https://www.amazon.co.uk/Mahout-Action-Sean-Owen/dp/1935182684/ref=sr_1_1/261-3819216-0809863?ie=UTF8&qid=1526360206&sr=8-1&keywords=mahout+in+action - Apache Mahout In Action (Book)