The Book Cave Final Report

Verklegt námskeið 2

May 11, 2018

Group 6

Arnar Leifsson / Björgvin B. Björgvinsson

Atli Gíslason, Egill Aron Þórisson, Kjartan Reynir Ólafsson,
Nökkvi Karlsson og Sigurður Kalman Oddsson

Table of Contents

1. Introduction

This report will discuss the requirements that were implemented and not implemented in the final product. In addition, we will talk about the successes of this project as well as the parts where we could've focused more on. We made a post mortem section where we talked about what we were proud of accomplishing and what could have done better. Then we discussed the basic requirements of our system and what we could and could not include. After that we created a diary, listing what each member did for each day of the project.
Then we created updated diagrams which show the growth of our project.

2. Post Mortem

When we look back at the 3 weeks we realize how important the Requirement Analysis and the Design Report were. This surprised us as we all thought we wanted to jump straight into coding the website without a lot of planning. We are all in an agreement that the planning was extremely important and looking back, we could have improved them.

The extra features that we added to the list were a bit more than we could handle within the limited time frame that we were given to code the product. We're glad that we agreed from the beginning that everyone had to consent to all actions from the starting stages until the end. It may have taken more time to achieve certain features, however it helped us a lot in the coding stage of the project.
We were able to finish a lot of the extra features that we established, such as logging in as employee, adding books, etc. however we were not able to finish all of our employee related extra requirements.
We also decided to give up on the premium user option requirement due to time limitations.
In hindsight we feel like we might have over-complicated our code a little bit once we became more familiar with the system and structure, and unfortunately could not simplify it more due to time limitation.

The overall look of the webpage changed quite a lot in the coding stages of the project from the low-fi prototype because when we saw the layout on the site itself we noticed a lot of things that we could have designed better, for instance the navigation bar would have been way too cluttered if it had been like we designed in the low-fi prototype.

3. Discussion

We were able to accomplish all of the basic requirements in the list of demands. We added quite a lot of extra features to our requirement list, such as a premium user option, employee actions like adding books, removing comments and ship order. We also added features like currently reading user status, following system, user saving checkout information and a book rating system. Users can also change their forgotten passwords and wrap their orders in gift wrap. There were quite a lot of extra featural requirements that we were not able to complete.

The premium user features, currently reading user status, following system, user saving checkout information, book rating system, user can also change their forgotten passwords and gift wrapping option were not implemented mainly due to lack of time, these would otherwise not have been a complicated features to implement.

We also added some extra structural demands to our list. For example we wanted the site to be optimized for mobile users. We also wanted the users passwords to be hashed and stored. The cart. This is something professional websites do in real life. We realized that the structural features like retrieving books from the database takes less than 500 milliseconds and that the application is able to support 10.000 concurrent users were not realistic. Still, we were able to properly back up the cart for each user.

In hindsight we could have focused more on the planning stage in order to be more time efficient, which in turn would have given us more time to finish our extra featural requirements.

4. Diary

Date & Time	Place	Work
21.04.2018 12 hours	Room M106	Started by creating a google docs so the whole group can work on the Requirement Analysis together. The first thing we did was structuring the document together so that we would all be on the same page. After that we started to identify our user groups.
22.04.2018 11 hours	Room M106	We all focused finished the majority of the user groups and moved on to analysing the competition. We each took one competitors website.
23.04.2018 14 hours	Room M106	We took the ideas from our competitors and started to write the featural and structural requirements. We also created a draft for the introduction and about the system part.
24.04.2018 12 hours	Room M106	Egill and Nökkvi started on the interview questions while Atli, Kjartan and Kalman started on the requirement list and about the system parts.
25.04.2018 12 hours	Room M106	We all worked together on taking the interviews, then Atli, Egill and Nökkvi finished the requirement list while Kjartan and Kalman started on the conclusion.
26.04.2018 9 hours	Room M106	Here we finished tweaking the introduction and about system and finished the conclusion.
27.04.2018 10 hours	Room M106	Went over everything together, grammar structure and overall look.
28.04.2018 15 hours	Room M106	Started by creating a google docs so the whole group can work on the Design Report together. The first thing we did was to create an appropriate front page and then Atli, Egill, Kjartan and Kalman focused on the Page prototypes while Nökkvi started on the introduction.

29.04.2018 14 hours	Room M106	We all worked together on the page prototypes and started to move onto the think aloud interviews together.
30.04.2018 8 hours	Room M106	Atli and Nökkvi started on the Navigation diagrams, Kalman worked on the Class diagram as well as the Table Diagrams and Egill and Kjartan finished the Page prototypes.
31.04.2018 13 hours	Room M106	Atli and Nökkvi finished the Navigation Diagrams(both for the customer and for the employee), Kalman finished the think aloud interviews while Egill and Kjartan worked on the programming rules.
01.05.2018 12 hours	Room M106	We all overviewed and finished the class and table diagrams, as well as the Conclusion, while Nökkvi finished fine tuning the appendix.
02.05.2018 14 hours	Room V205	Kalman started by creating the project(initial commit), Atli set up the sql server while Egill, Nökkvi and Kjartan added the first service classes.
03.05.2018 11 hours	Room V205	Atli and Kalman worked on login and registration for user. Egill and Kjartan worked
04.05.2018 12 hours	Room V212	Kalman added icons for books. Nökkvi improved registration form Atli worked on date published filtering, partial service side validation and and a rough sketch of the catalogue. Kjartan and Egill added wishlist, partial cart.
05.05.2018 5 hours	Room M106	Nökkvi worked on book details and related functions while Atli continued working on the database. Egill, Kalman and Kjartan worked on some visual elements on the front page.
06.05.2018 15 hours	Room M106	Nökkvi and Kjartan worked on the database, Atli added book rating, add book to cart. Nd worked on catalogue Kalman and Egill added footer also worked on rating system.
07.05.2018 12 hours	Room M106	Atli and Kalman worked on catalogue Nökkvi added bestseller catalogue. Egill added a wishlist. Kjartan added books to database Kalman worked on book detail outlook.

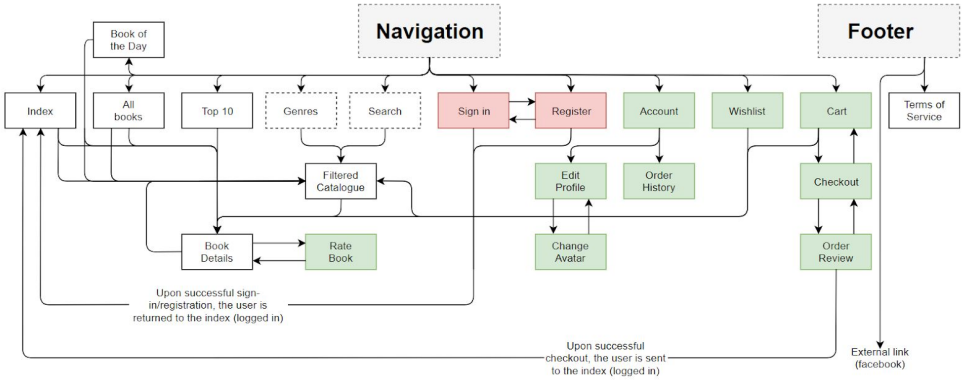
08.05.2018 13 hours	Room M106	Kjartan started the Final Report Atli built up the cart form and the input model for checkout, filter fixes. Kjartan and Nökkvi added avatar features. Kalman and Egill made additional fixes to overall look.
9.05.2018 16 hours	Room M106	Atli fixed the cart and wishlist. Kjartan worked on the Final Report. Nökkvi made major changes to checkout, cart and avatar as well as the class diagram. Kalman worked on overall improvements of the sites look.
10.05.2018 14 hours	Room M106	The whole group discussed what we wanted to have in the final report and discussed which requirements we had time to add to our system before handing it in. Most of the day went into commenting and making sure that the programming rules we set in the requirement analysis were followed.
11.05.2018 17 hours	Room M106	Nökkvi and Egill completed the client side validation and Kjartan worked on the server side validation. Kalman improved the overall look of the website Atli added the employee functions. Whole group went over the Final Report as well as the code.

5. Updated Navigation Diagram

The following diagrams are navigation diagram from the perspective of a customer and an employee, respectively. The rules used in the navigation diagram section in our Design Report apply here. Dotted bordered elements represent “pseudo-views”, all direct children of the Navigation element and the Footer element are accessible at all times, red colored views are only accessible when logged out and green colored views are only accessible when logged in.

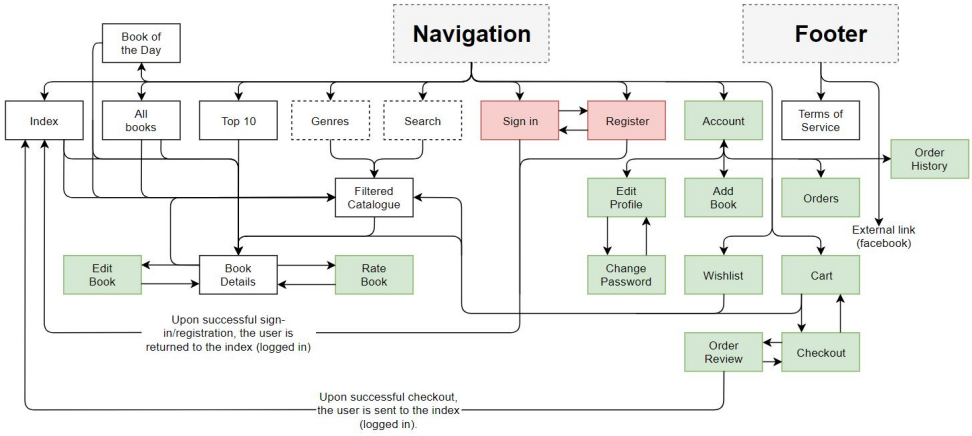
5.1 Updated Customer Navigation Diagram

The main difference from the old navigation diagram and the new one is that the views are more cross-linked. For example all views that contain a list of books contain a link to the book's details view and all of them except the 'Top 10 Rated' lead to a filtered catalogue (by author). Two additional views have also been added, the 'Book of the Day' view and the 'Order History' view.



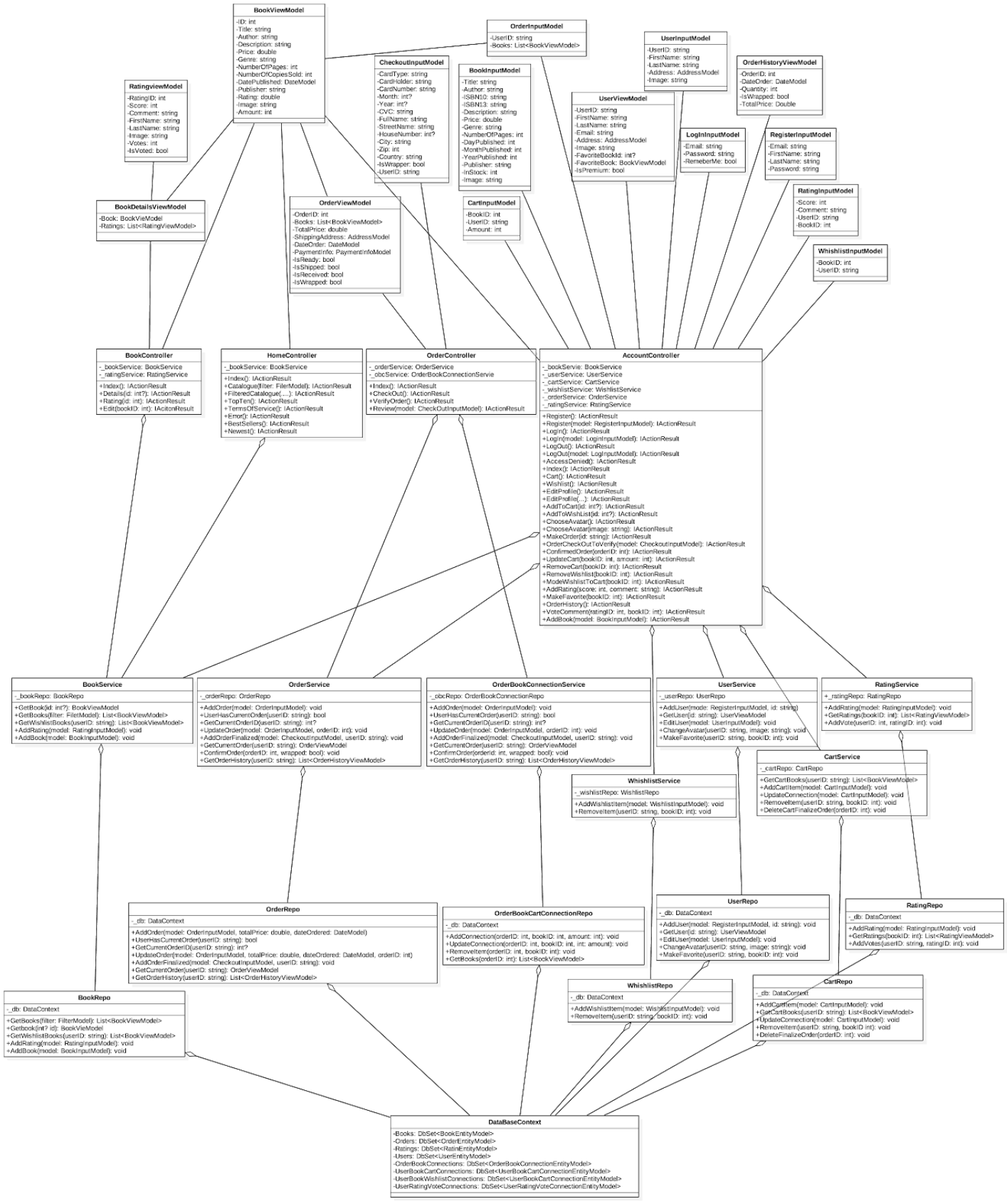
5.2 Updated Employee Navigation Diagram

The same differences from the previous diagram apply here. Another noticeable difference from the old employee navigation diagram is that the 'Cart' and 'Wishlist' views are now also available by employee. In other words, employees can now buy books. Employees can also browse all orders made in the 'Orders' view.



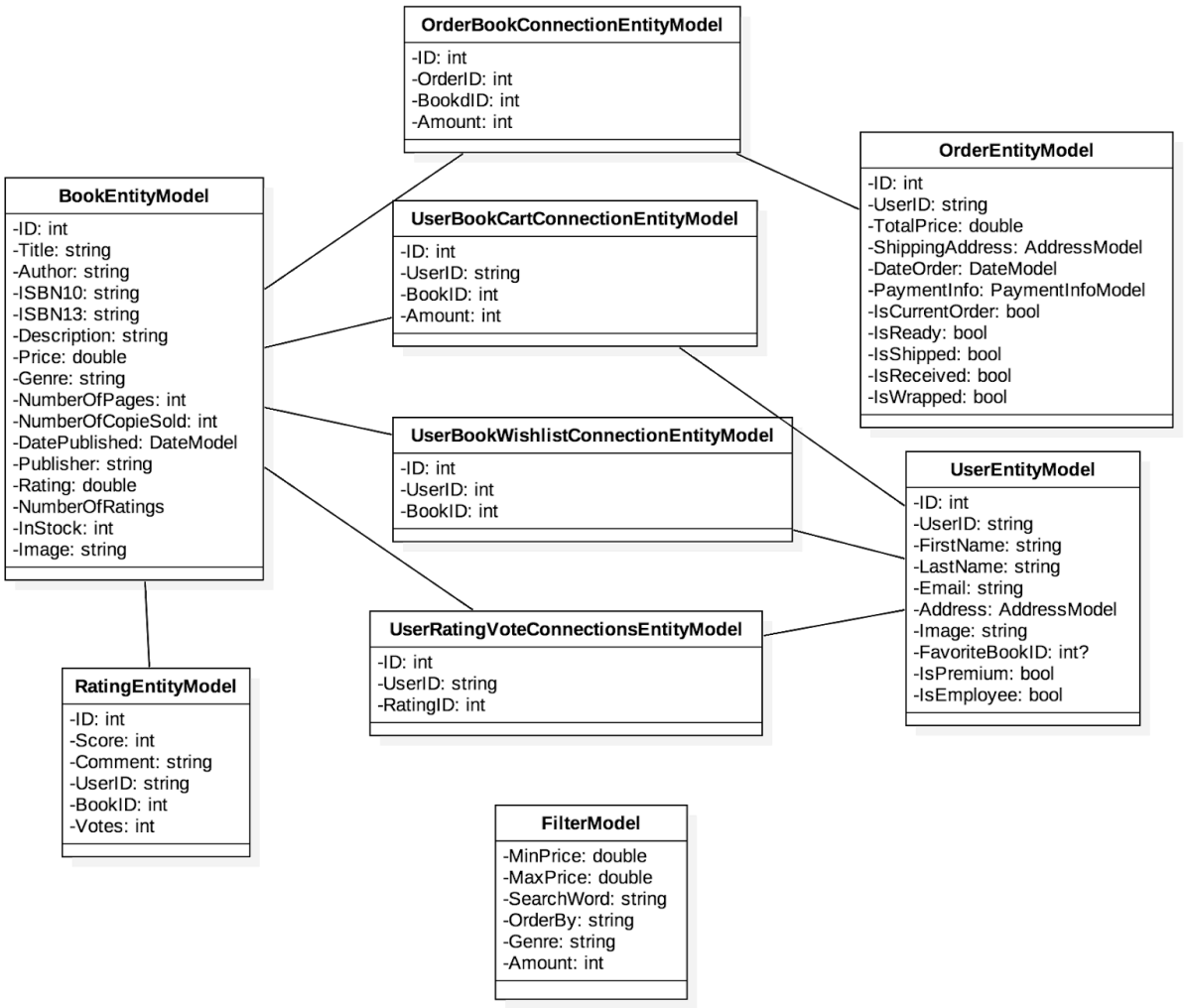
6. Updated Class Diagram

We updated the class diagram from the design report and it can be found below. It is split into 3 layers: Controller, Service, and Repository and each layer has a different role just like in the design report. The top row consists of Input and View Models which is what the user will be interacting with. The Controller's job is to send these to the user and to receive them when the user sends in a form. The Controller then talks to the Service layer to write and read data. The Controller requests access to data and sends data, and it is the Service layer's job to handle this data by sending the requests forward to the Repository classes or to throw exceptions. The repository layers handles Linq queries and reads and writes to the Database. The database is an SQL server which we connect to through the DataContext class. We did not include the Entity Model classes in the class diagram to make it more readable. The entity classes and how they interact with each other can be found in the table diagram in chapter 7. The updated class diagram is considerably larger than the one we made in the design report and each layer has significantly more classes. We could possibly combine classes and functions to make the project less cluttered if we had more time.


BookViewModel
-ID: int
-Title: string
-Author: string
-Description: string
-Price: double
-Genre: string
-NumberOfPages: int
-NumberOfCopiesSold: int
-DatePublished: DateModel
-Publisher: string
-Rating: double
-Image: string
-Amount: int
OrderInputModel
-UserID: string
-Books: List<BookViewModel>
UserInputModel
-UserID: string
-FirstName: string
-LastName: string
-Address: AddressModel
-Image: string
OrderHistoryViewModel
-OrderID: int
-DateOrder: DateModel
-Quantity: int
-IsWrapped: bool
-TotalPrice: Double
RatingViewModel
-RatingID: int
-Score: int
-Comment: string
-FirstName: string
-LastName: string
-Image: string
-Votes: int
-IsVoted: bool
CheckoutInputModel
-CardType: string
-CardHolder: string
-CardNumber: string
-Month: int?
-Year: int?
-CVC: string
-FullName: string
-StreetName: string
-HouseNumber: int?
-City: string
-Zip: int
-Country: string
-IsWrapper: bool
-UserID: string
BookInputModel
-Title: string
-Author: string
-ISBN10: string
-ISBN13: string
-Description: string
-Price: double
-Genre: string
-NumberOfPages: int
-DayPublished: int
-MonthPublished: int
-YearPublished: int
-Publisher: string
-InStock: int
-Image: string
UserViewModel
-UserID: string
-FirstName: string
-LastName: string
-Email: string
-Address: AddressModel
-Image: string
-FavoriteBookId: int?
-FavoriteBook: BookViewModel
-IsPremium: bool
LogInInputModel
-Email: string
-Password: string
-RemeberMe: bool
RegisterInputModel
-Email: string
-FirstName: string
-LastName: string
-Password: string
BookDetailsViewModel
-Book: BookVieModel
-Ratings: List<RatingViewModel>
OrderViewModel
-OrderID: int
-Books: List<BookViewModel>
-TotalPrice: double
-ShippingAddress: AddressModel
-DateOrder: DateModel
-PaymentInfo: PaymentInfoModel
-IsReady: bool
-IsShipped: bool
-IsReceived: bool
-IsWrapped: bool
CartInputModel
-BookID: int
-UserID: string
-Amount: int
RatingInputModel
-Score: int
-Comment: string
-UserID: string
-BookID: int
WhishlistInputModel
-BookID: int
-UserID: string
BookController
-_bookService: BookService
-_ratingService: RatingService
+Index(): IActionResult
+Details(id: int?): IActionResult
+Rating(id: int): IActionResult
+Edit(bookID: int): IActionResult
HomeController
-_bookService: BookService
+Index(): IActionResult
+Catalogue(filter: FilerModel): IActionResult
+FilteredCatalogue(,...): IActionResult
+TopTen(): IActionResult
+TermsOfService(): IActionResult
+Error(): IActionResult
+BestSellers(): IActionResult
+Newest(): IActionResult
OrderController
-_orderService: OrderService
-_obcService: OrderBookConnectionServie
+Index(): IActionResult
+CheckOut(): IActionResult
+VerifyOrder(): IActionResult
+Review(model: CheckOutInputModel): IActionResult
AccountController
-_bookServie: BookService
-_userService: UserService
-_cartService: CartService
-_wishlistService: WishlistService
-_orderService: OrderService
-_ratingService: RatingService
+Register(): IActionResult
+Register(model: RegisterInputModel): IActionResult
+LogIn(): IActionResult
+LogIn(model: LogInputModel): IActionResult
+LogOut(): IActionResult
+LogOut(model: LogInputModel): IActionResult
+AccessDenied(): IActionResult
+Index(): IActionResult
+Cart(): IActionResult
+Wishlist(): IActionResult
+EditProfile(): IActionResult
+EditProfile(...): IActionResult
+AddToCart(id: int?): IActionResult
+AddToWishList(id: int?): IActionResult
+ChooseAvatar(): IActionResult
+ChooseAvatar(image: string): IActionResult
+MakeOrder(id: string): IActionResult
+OrderCheckOutToVerify(model: CheckoutInputModel): IActionResult
+ConfirmedOrder(orderID: int): IActionResult
+UpdateCart(bookID: int, amount: int): IActionResult
+RemoveCart(bookID: int): IActionResult
+RemoveWishlist(bookID: int): IActionResult
+ModeWishlistToCart(bookID: int): IActionResult
+AddRating(score: int, comment: string): IActionResult
+MakeFavorite(bookID: int): IActionResult
+OrderHistory(): IActionResult
+VoteComment(ratingID: int, bookID: int): IActionResult
+AddBook(model: BookInputModel): IActionResult
BookService
-_bookRepo: BookRepo
+GetBook(id: int?): BookViewModel
+GetBooks(filter: FiletModel): List<BookViewModel>
+GetWishlistBooks(userID: string): List<BookViewModel>
+AddRating(model: RatingInputModel): void
+AddBook(model: BookInputModel): void
OrderService
-_orderRepo: OrderRepo
+AddOrder(model: OrderInputModel): void
+UserHasCurrentOrder(userID: string): bool
+GetCurrentOrderID(userID: string): int?
+UpdateOrder(model: OrderInputModel, orderID: int): void
+AddOrderFinalized(model: CheckoutInputModel, userID: string): void
+GetCurrentOrder(userID: string): OrderViewModel
+ConfirmOrder(orderID: int, wrapped: bool): void
+GetOrderHistory(userID: string): List<OrderHistoryViewModel>
OrderBookConnectionService
-_obcRepo: OrderBookConnectionRepo
+AddOrder(model: OrderInputModel): void
+UserHasCurrentOrder(userID: string): bool
+GetCurrentOrderID(userID: string): int?
+UpdateOrder(model: OrderInputModel, orderID: int): void
+AddOrderFinalized(model: CheckoutInputModel, userID: string): void
+GetCurrentOrder(userID: string): OrderViewModel
+ConfirmOrder(orderID: int, wrapped: bool): void
+GetOrderHistory(userID: string): List<OrderHistoryViewModel>
UserService
-_userRepo: UserRepo
+AddUser(mode: RegisterInputModel, id: string)
+GetUser(id: string): UserViewModel
+EditUser(model: UserInputModel): void
+ChangeAvatar(userID: string, image: string): void
+MakeFavorite(userID: string, bookID: int): void
RatingService
+_ratingRepo: RatingRepo
+AddRating(model: RatingInputModel): void
+GetRatings(bookID: int): List<RatingViewModel>
+AddVote(userID: int, ratingID: int): void
CartService
-_cartRepo: CartRepo
+GetCartBooks(userID: string): List<BookViewModel>
+AddCartItem(model: CartInputModel): void
+UpdateConnection(model: CartInputModel): void
+RemoveItem(userID: string, bookID: int): void
+DeleteCartFinalizeOrder(orderID: int): void
WhishlistService
-_wishlistRepo: WishlistRepo
+AddWishlistItem(model: WishlistInputModel): void
+RemoveItem(userID: string, bookID: int): void
OrderRepo
-_db: DataContext
+AddOrder(model: OrderInputModel, totalPrice: double, dateOrdered: DateModel)
+UserHasCurrentOrder(userID: string): bool
+GetCurrentOrderID(userID: string): int?
+UpdateOrder(model: OrderInputModel, totalPrice: double, dateOrdered: DateModel, orderID: int)
+AddOrderFinalized(model: CheckoutInputModel, userID: string): void
+GetCurrentOrder(userID: string): OrderViewModel
+GetOrderHistory(userID: string): List<OrderHistoryViewModel>
OrderBookCartConnectionRepo
-_db: DataContext
+AddConnection(orderID: int, bookID: int, amount: int): void
+UpdateConnection(orderID: int, bookID: int, int: amount): void
+RemoveItem(orderID: int, bookID: int): void
+GetBooks(orderID: int): List<BookViewModel>
UserRepo
-_db: DataContext
+AddUser(model: RegisterInputModel, id: string): void
+GetUser(id: string): UserViewModel
+EditUser(model: UserInputModel): void
+ChangeAvatar(userID: string, image: string): void
+MakeFavorite(userID: string, bookID: int): void
RatingRepo
-_db: DataContext
+AddRating(model: RatingInputModel): void
+GetRatings(bookID: int): List<RatingViewModel>
+AddVotes(userID: string, ratingID: int): void
WhishlistRepo
-_db: DataContext
+AddWishlistItem(model: WishlistInputModel): void
+RemoveItem(userID: string, bookID: int): void
CartRepo
-_db: DataContext
+AddCartItem(model: CartInputModel): void
+GetCartBooks(userID: string): List<BookViewModel>
+UpdateConnection(model: CartInputModel): void
+RemoveItem(userID: string, bookID int): void
+DeleteFinalizeOrder(orderID: int): void
BookRepo
-_db: DataContext
+GetBooks(filter: FilterModel): List<BookViewModel>
+Getbook(int? id): BookVieModel
+GetWishlistBooks(userID: string): List<BookViewModel>
+AddRating(model: RatingInputModel): void
+AddBook(model: BookInputModel): void
DataBaseContext
-Books: DbSet<BookEntityModel>
-Orders: DbSet<OrderEntityModel>
-Ratings: DbSet<RatinEntityModel>
-Users: DbSet<UserEntityModel>
-OrderBookConnections: DbSet<OrderBookConnectionEntityModel>
-UserBookCartConnections: DbSet<UserBookCartConnectionEntityModel>
-UserBookWishlistConnections: DbSet<UserBookCartConnectionEntityModel>
-UserRatingVoteConnections: DbSet<UserRatingVoteConnectionEntityModel>

7. Updated Table Diagram

Below is the updated table diagram for the project. All of the Entity classes shown here are mapped to our database. We hold on to information about Users, Books, Ratings and Orders. The classes are also connected; every user has a cart and wishlist of books. Every book has multiple ratings, and every order has a list of books. These connections are written down in the ConnectionEntityModel classes. Additionally these is a class called FilterModel which is not written down in the database, but merely exists as a encapsulation of multiple data which is required for when users try to filter books. The updated table diagram does not differ greatly from the original in the design report, the main differences are that the RatingEntityModel connects directly to the BookEntityModel instead of going through a Connection class and the user EntityModel contains fewer variables than we thought it would when we made the original diagram in the design report.



8. Conclusion

The final product was overall a success as we were able to accomplish all of the basic featural requirements as well as to add extra features to the product. As we progressed with the project we realized that we had added quite a lot of extra featural requirements that would take more time than we had before the due date.
If we were to create a similar product in the future we would make sure to focus more on the planning stages. We now realise that pouring time and effort into the planning stages can save time and frustration when we actually start creating the project.