# Assignment Template Form

## Book class implemented

|  |  |
| --- | --- |
| 🗹 | **Book class fully implemented and tested** |

## Customer class implemented

|  |  |
| --- | --- |
| 🗹 | **Customer class fully implemented and tested**  Created customer table, class and all relevant CRUD operations |
| 🗹 | **Customer class includes an address**  As with OOP in block 1, implemented the customer class with a separate address class that the customer utilises as a field. Address has its own table in the DB with a composite primary key of house number and post code, which is referenced as a composite foreign key in the customer table. |

## BookDAO CRUD methods implemented

|  |  |
| --- | --- |
| 🗹 | **Method to connect to the books database** |
| 🗹 | **Method to return all the books stored in the database** |
| 🗹 | **Method to return a single book based on ID**  Generated an advanced search function to search by multiple/some fields and return table of matching books also |
| 🗹 | **Method to insert a book into the database** |
| 🗹 | **Method to update details of a book stored in the database based on ID** |
| 🗹 | **Method to delete a book in the database based on ID**  Added ON DELETE CASCADE to the books table – I understand this isn’t the best solution as this will remove any historic orders from the order\_line table, but without the book reference, that information is useless anyway. |
| 🗹 | **Use of prepared statements**  Had a lot of fun building the prepared statements for the advanced search function! |

## Console menu system incorporated

|  |  |
| --- | --- |
| 🗹 | **Application has a menu system to allow CRUD methods to be fully tested**  Incorporated for Customer and Book as Controller class. Can be called in main to work alongside the website code also. |

## Customer Functionality

|  |  |
| --- | --- |
| 🗹 | **Customer class fully implemented and tested** |
| 🗹 | **CRUD db functionality to add/update/delete Customer** |
| 🗹 | **Menu functionality to add/update/delete Customer**  update functionality for the console menu is limited to 1 field just to test the ability for the db to update – this is true for both book and customer |
| 🗹 | **Web functionality to add/update/delete Customer** |

## Web Front-End

|  |  |
| --- | --- |
| 🗹 | **Server connects to database and books displayed using an HTML table** |
| 🗹 | **Server incorporates administrator log-in**  Some assumptions made in this. Database as a “user type” which could have been filtered by, but its not clear if the system is going to ever have any other users or what the function of the users is, and given the two in the current db are both type 1, I just assumed anyone logging in with a username and password is an admin.  Added MD5 hash to the password – didnt apply on already stored passwords in the db, but added when retrieving from the db. Also applied the hash to the password that the user enters in the front end.  Was unsure on the aspects of the temporary password for when the user forgets their password.  No security really applied – no cookies or sessions used. For all extent and purpose the login is just a simple operation checking the password entered against the db and doesn’t stop anyone who has any of the page URLs just going through. |
| 🗹 | **Administrator can add new book** |
| 🗹 | **Administrator can update existing book** |
| 🗹 | **Administrator can delete book** |

## Advanced Features

|  |  |
| --- | --- |
| 🗹 | **Design pattern used and identified in code comments**  Use of the singleton pattern is mentioned in the top of the BookDAO class. |
| 🗹 | **Order functionality exposed through web interface**  Lots of work done implementing the order system.  Order implemented as in OOP in block 1, where the order can contain multiple books. Would have been much easier to do 1 book per order, but in reality a system probably wouldn’t be designed like that as it would create a lot of redundancy, also not as fun a challenge to figure out!  Used order table with weak entity order\_line between book and order. Created crud operations for both tables and a front end to apply.  Took by far the longest out of anything in the assessment!  Didn’t implement a stock system as OOP in block 1, but did add a check to make sure any item being added to an order didn’t already exist in the order\_line table. Assumption being each book is a single product as things like condition and notes are specific to a single book object and not a group of books. |
| 🗹 | **Appropriate use of advanced java features (e.g. lambda expressions) [identified using comments]**  Some Lambda functions have been added into the SearchBookHandler and SearchCustomerHandler classes and have been noted as such. Could have used a lot more extensively throughout. |