|  |
| --- |
| **CMP205 Application Design**  **Comic Book Store Project**  **Application Design Report** |
|  |
| **BEN FLEUTY 1900040** |

|  |  |
| --- | --- |
| **Design (40%)** | |
| **Analysis**  First, I split the information in the brief into functional and non-functional requirements. This information allowed me to plan out the program effectively as I could see everything that I needed to do at once.  From this planning I deducted that I would need several classes to effectively store all the data the application needs.  Classes:   * User   + Customer   + Staff * Order * Basket * Product   The base user class contains attributes that Customer and Staff share, such as name, address, email, etc. I opted to implement using interfaces instead of inheritance as it allows for more flexible code to be written. For example, a GetUser function can be used to get either a customer or a member of staff as IUser will be returned along with any class-specific data.  Relationships:  One user has one or many orders  One order has one basket  One basket has one or many products  **This application uses dummy data generated at** [**https://www.mockaroo.com/**](https://www.mockaroo.com/)  **Model**    **User Interface** | |
| Login  Register  Landing Page / Product Search Page – Registered User  Landing Page / Product Search Page – Guest User  Basket – Empty  Basket – With Items  Search Example – “lorem” in Title OR Description    View Orders Page  Checkout Page  Checkout as Guest – Address input  Checkout Confirmation – Staff required    Product View Window  Staff Landing Page  Entity Editor – Select Entity  Customer Record Editor | |
| Order Record Viewer  Staff Record Editor  Product Record Editor | |
| Stock Levels Report | |
| Sales Viewer | |
| **Implementation (40%)** | |
| **C# Source Code**  **Executable File**  In .zip and at <https://github.com/benfleuty/CMP205U1ComicBookStoreWPF>  Private repo will require you to request access @ [1900040@abertay.ac.uk](mailto:1900040@abertay.ac.uk)  **Sample Input**  U: admin@dundeecomicbookstore.com P: spiderman  This will give you access to an admin account.  You can register non-admins through the registration page.  To make a customer record a staff record:   1. Log in as an admin with Access Employee Data permission 2. Entity Editor 🡪 Customer Record 3. Select the customer to make staff 4. Click Make Staff 5. Click on home 6. Entity Editor 🡪 Staff Record 7. Select the new staff member 8. Use the checkboxes in the form to apply relevant permissions.   **Functionality**  The application doesn’t display any tools to close the program as it is intended to be displayed customer facing. Alt-F4 and Alt-Tab will allow you to move away from the application.  The application opens to the login page. Here the user can:   * Log in with an account * Continue as a guest * Navigate to the registration page   The email and password provided are processed and checked against the database. A positive result will take the user to the landing search page.  A negative result will result in error messages.  From the landing page the customer can search for items and add them to their basket.  From the basket screen, they can open the items they have added to their basket so that they can view the information, alter the quantity of the item, or remove it entirely.  From the basket screen they can also save their order and view any saved orders.  When viewing a saved order, the user can choose to open it or delete it. If opened, this will make the newly opened order the current order and allow the customer to manipulate the order. If they had a different order open at the time of opening the new order, the user will be prompted to save their current order.  The checkout page displays the subtotal (raw price of basket) and the total (raw price less any discount and plus any shipping). If the user is a guest, they will be prompted to enter their address. Non-staff users will require a staff member to authorise the transaction by logging in with their credentials since payments are handled outside of the program. | |
| **Reflection (20%)** |
| **Development Methodology**  The approach that was opted for was agile as it suits this project well. This is because agile allows for frequent test and changing of code to ensure the product is developed correctly. Doing this allowed for prototypes of ways of meeting the given requirements could be developed and the approach could be altered as the application developed. This allowed for flexible code to be developed which reduced the overall development time and increased reliability as the algorithms are stored in one place. This makes maintaining the codebase easier and more efficient as altering one algorithm affects everything using, meaning that a change doesn’t have to be copied over throughout the program.  With each requirement, part of the solution was developed and tested, ensuring the rest of the application functioned as expected. This allow for errors to be identified and rectified early to prevent large changes having to be made later in the development cycle.  **Evaluation**  Throughout the development cycle changes had to be made to the project to ensure it would meet the requirements. The planned use of inheritance for the Customer and Staff objects that was selected since they share attributes that can be contained in a superclass User was scrapped since interfaces are more flexible. This allows the Staff and Customer classes to, in most cases, be used interchangeably.  A database interaction helper class was developed to assist in the development of the program. This allowed me to keep all my database interactions consistent and centrally stored. This was useful as it prevented the copying of identical functions around the codebase which would lead to maintenance issues. Combining this with the use of interfaces also reduces the amount of code that must be written as, for example, GetUser can get either a staff member or a customer as it returns the IUser interface. This means, that any class implementing this interface can be returned, which adds to the scalability of the application. |
| **Personal Reflections**  I found that splitting the database and code development resulted in a database structure that is not ideal for implementation to be implemented since testing was not as accessible due to lacking knowledge of the power of the .NET framework.  I feel as though I have learned well from this assignment as it has introduced me to T-SQL, more C# and .NET (such as interfaces).  Personally, I would like to add more planned functionality to the program that was not added due to time constraints. |