|  |
| --- |
|  |
| Library system |
| A Level coursework |

|  |
| --- |
| Holly Wheeldon  [Date] |

Contents

[Analysis 4](#_Toc62477510)

[Problem identification 4](#_Toc62477511)

[Suitability to a computational approach 4](#_Toc62477512)

[Research of the problem 4](#_Toc62477513)

[Interviews 4](#_Toc62477514)

[Similar solutions 8](#_Toc62477515)

[Stakeholders 10](#_Toc62477516)

[Proposed solution 10](#_Toc62477517)

[Essential features 10](#_Toc62477518)

[Limitations 10](#_Toc62477519)

[Requirements 11](#_Toc62477520)

[Success criteria 11](#_Toc62477521)

[Design of the solution 12](#_Toc62477522)

[Structure of the solution 12](#_Toc62477523)

[Decomposition of the problem 13](#_Toc62477524)

[Description of the solution 13](#_Toc62477525)

[Design 13](#_Toc62477526)

[Selection of software 15](#_Toc62477527)

[Database structure 17](#_Toc62477528)

[Key Variables 18](#_Toc62477529)

[Classes 18](#_Toc62477530)

[Key functions 21](#_Toc62477531)

[Libraries used 26](#_Toc62477532)

[Usability features 28](#_Toc62477533)

[Approach to testing 28](#_Toc62477534)

[Developing the solution 29](#_Toc62477535)

[Justification of Methods Used 29](#_Toc62477536)

[White Box Testing 29](#_Toc62477537)

[Alpha testing 29](#_Toc62477538)

[Code analysis 29](#_Toc62477539)

[Book.cs 30](#_Toc62477540)

[ReturnsLog.cs 30](#_Toc62477541)

[User.cs 30](#_Toc62477542)

[LibraryDbContext.cs 30](#_Toc62477543)

[LibraryDataFunctions.cs 31](#_Toc62477544)

[AdressUpdateEventArgs.cs 39](#_Toc62477545)

[AdminHomeForm.cs 40](#_Toc62477546)

[CameraForm.cs 45](#_Toc62477547)

[LoginForm.cs 49](#_Toc62477548)

[RegisterForm.cs 51](#_Toc62477549)

[ReportsForm.cs 53](#_Toc62477550)

[UserHomeForm.cs 54](#_Toc62477551)

[Service1.cs 56](#_Toc62477552)

[Review with stakeholders 59](#_Toc62477553)

[Evaluation 59](#_Toc62477554)

[Success criteria 59](#_Toc62477555)

[Username and password to confirm user identity 59](#_Toc62477556)

[Register screen where user can enter their own details 59](#_Toc62477557)

[Screen where admins (and only admins) can add books to the library 60](#_Toc62477558)

[Usable barcode scanner 60](#_Toc62477559)

[All areas can be navigated easily with a screen reader 60](#_Toc62477560)

[Screen where students can sign books in and out 60](#_Toc62477561)

[Email sent to students 2 days before a book is due 61](#_Toc62477562)

[Screen for creating reports 61](#_Toc62477563)

[All books’, and registered users’ details stored in a database 61](#_Toc62477564)

[Log of all books taken out stored in a database 62](#_Toc62477565)

[Takes less than 1 minute to take out / return a book 62](#_Toc62477566)

[Students have their own logins, and can use the system unsupervised 62](#_Toc62477567)

[System recommends books, based on which are popular at the moment 62](#_Toc62477568)

[Students can search for book recommendations based on genre 62](#_Toc62477569)

[Safeguarding emailed when a number of flagged books are taken out 63](#_Toc62477570)

[Students and admins use different forms, to ensure simplicity 63](#_Toc62477571)

[Usability features 63](#_Toc62477572)

[Large text 63](#_Toc62477573)

[Screen reader compatibility 63](#_Toc62477574)

[Clear instructions 63](#_Toc62477575)

[Separated modes for different users 63](#_Toc62477576)

[Maintenance and development 64](#_Toc62477577)

[References 64](#_Toc62477578)

# Analysis

## Problem identification

Most schools, including my own, have a library. It is the central space for study, research and reading. Whether students read for entertainment or to advance their studies, the library is a critical part of school life. However, schools are subject to many financial constraints which can reduce the resources available to manage areas such as the library. My proposal will help provide for the needs of the students and staff, reducing the resources needed to manage the library, thereby improving the quality of the library while reducing the financial strain on the school of providing this service for their students.

The problem is that the library is inefficient, meaning book signing in and out is unreliable and inconsistent, and many students don’t return books in a timely fashion. This leads to a depleted supply of books in the library, as well as lots of time and effort spent in an attempt to remind students to return books. Therefore, my proposal is to make a system that is more responsive to the needs of users enabling them to use the library to it’s full potential and reduce the strain of menial tasks on the librarian.

## Suitability to a computational approach

The problem is highly suited to a computational approach, due to the nature of the issues with the current solution. For example, cataloguing of large quantities of data about books or students is highly suited to a relational database as when properly optimized this reduces storage of repeated data, and somewhat protects the data from loss or damage (however there is still the possibility of data loss through system crashes or failure of storage systems). Another benefit of a computational approach is that it enables users to be contacted remotely, which could be used to provide reminders for overdue books without requiring the user to visit the library in person. This has the potential to significantly reduce the number of unreturned books and prevent the library from becoming depleted. The system will also be able to adapt to the changing needs of students and administrators much more easily.

## Research of the problem

## Interviews

With regard to interviews, it is important to ensure that all groups who will use the system are involved, as this will help build requirements and requested features. Initial interviews will be conducted with a number of students who use the library, one with the librarian who will use the system on a day to day basis as the main admin, and one with the safeguarding lead to ensure I am not breaching any safety regulations especially with regard to data protection. As I was unable to meet with these people in person, I created a google form to collect the results. The questions, justifications for these, as well as the answers and any conclusions I made can be found below:

#### Pupils

1. Have you ever used the school library?

*Checks if their answers have relevance and are based on actual experiences, if they answered no the survey didn’t ask any of the other questions.*

1. Have you always been able to find books that you wanted?

*Determines quality of current searching systems*

1. If not, what prevented you from doing this?

*And any current issues to resolve*

1. Do you find it easy to check books in and out?

*Determines quality of current signing in systems*

1. If not, what was the issue?

*And any current issues to resolve*

1. What do you like about the current library system?

*Identifies current features to keep in the new system*

1. What would you like to improve about the current library system?

*Identifies potential improvements*

1. Would you be confident in using technology to check books in and out, e.g. a PC with a handheld barcode scanner?

*Checks if this input style is suitable for the solution*

11 students replied to this survey, however some misunderstood the questions and replied with answers that weren’t relevant to this project (such as management of behaviour in the library, or having computer resources so that they can do coursework) so I have omitted these responses.

1. Yes [10] no [1]
2. Yes [6] no [4]
3. Books were not available / Not in the library or taken out and not returned / Organisation of the books / Them not being there, usually

*My solution will address these issues, as it will help ensure books are returned effectively.*

1. Yes [10]
2. (no responses)
3. You can leave books in the box to be returned for you / you do it yourself / The ease of getting books / The books are labelled well + easy to give them back in / Symbols on the spine of the book telling you the genre / It seems to work pretty well

*Students seem to like the ease of being able to return books without teacher supervision, therefore I will make sure students can fully use the system without involvement of a teacher. They also appreciate extra information, such as the genre of the book, so having information on the book stored in the system could be an additional feature.*

1. Not have to wait for the librarian to scan in and out books / More complete book series. / Recommendations of books. / A bigger range of subject books / Perhaps a bit more specifically labelled / Suggested reading ages in or on the book / If we didn’t actually have to talk to people to check out a book lol

*As with question 6, students would like more autonomy with scanning books in and out, as well as extra information on the books in the library. One student also suggested that being recommended books by the system would help them find books they are interested in - I will try to incorporate this.*

1. Yes [10]
2. I think this is a bad idea as there is a guaranty that people will be stupid with it and/or break it, having someone doing it means that it is cheaper and their is less chance of pranks or stupidity

*This student was concerned that giving students responsibility and autonomy over checking in and out books would lead to misbehaviour and vandalism. However, as the library is a supervised space this shouldn’t be a significant issue that needs to be taken into account.*

#### Admin staff

1. What is the current system for signing books in and out?
2. Do you store a catalogue of library books?
3. What rules are there regarding returns, and use of the library? (e.g. How long can students borrow books for, what happens if they are returned late, is there a maximum number of books a student can have out at a time, etc.)
4. What information do you need to access about students and books? (e.g. Borrowing history, list of books currently out, students with overdue books, etc.)

*1-4 Gives an idea of the current functionality of the system – what the digital solution would be replacing*

1. Are there any issues that you find when using these systems?

*checks if the proposed solution is beneficial.*

1. Do all or most of your books have some unique identifier on them? And what types of books don’t have this?

*Checks if the solution is viable – if there is no way of uniquely identifying books already it may be too time consuming and expensive to create this.*

1. Do you hold multiple copies of the same book? If so, are these differentiated from each other in some way?

*Checks if duplicate books will cause an issue that we will need to consider*

1. What is the current book loss in the school?

*To measure improvements to this*

1. What would you like to gain from this new system?

*Identifies a potential wish list of features for the system*

The librarian completed this survey, as they are the main admin for the library.

1. Eclipse - an online system which can be accessed from anywhere at anytime.

*This is not ideal, as this is similar to my proposed system meaning that there may not be much benefit to using the new solution.*

1. This is via the Eclipse system, no physical catalogue as it is constantly changing.
2. Yr9-11 can borrow 3 books for 3 weeks and then have the option of renewing if needed. Sixth for can borrow 6 books for 4 weeks and again have the option of renewing if needed. One of the benefits for Yr9-11 students who use the library regularly is to become a VIP and then they have sixth form borrowing privileges. When a book becomes overdue the system will send out reminder emails to students, tutors and parent/carers. There are currently no fines in place however this is something which is being looked into.

*This will inform the rules I incorporate into the system.*

1. All of the below data needs to be able to specify the following information - girls, boys, year groups, Genre/Dewey Classifications, fiction, non-fiction, date fields. Current Loans - Past Loans - Overdue – Resources

*I will build this in.*

1. It's easy when you just want a general overview but not so when you need to narrow the information to the above specifics.

*This is something I can try to make easier.*

1. All catalogued books/items have an individual barcode placed inside with the school name on. This is what is used to create a record on the system. The barcode number is also hand written inside just in case the barcode label gets removed or damaged. Items which cannot be loaned/leave the library do not have this however that only really equates to a very small number of resources such as encyclopaedias.

*this will be well suited to my solution.*

1. There are only a few titles which we have duplicate copies of. They have individual barcode numbers.

*Therefore this will not be an issue.*

1. The number of books not returned by leavers last year was 7. Books lost because they were taken without being issued can only be measured by carrying out a stock take. This is a mammoth task and is not carried out every year. In my experience not many books are lost this way but obviously it does happen.

*This is a small number, but over a larger period of time this will become a more significant value.*

1. I would like to be able to generate my own reports. There are reports that can be run on the current system however if I want specific data not covered by those reports I can only access that information by running an advanced search. This is not ideal as I cannot save the search perimeters and it takes time to set it up.

*This is something I can incorporate.*

#### Safeguarding staff

1. What are the minimum security measures required to protect student and staff data stored on a database system in school?

*To find out what security measures I will need to take when storing the data*

1. What are the policies for storing student data locally, on the school network and in a cloud based model?

*To determine the best place to locate the database*

1. What information am I legally allowed/not allowed to store about a student?

*To ensure I am only storing relevant and legal information*

1. Does the school have data destruction requirements?

*If so, I may need to build this into my system*

1. Does a student have rights to withdraw permission to hold data within a system?

*If so, this will need to be built into the system*

1. Who is allowed to access student data (e.g. Borrowing history, account information etc.)?

*This will need to be built into the system*

1. Would it be possible to connect my solution with existing school data systems such as SIMS or the school’s login system?

*If so, this could be something I could implement, or a future improvement of the system.*

Two members of the safeguarding team responded to this survey, the head of safeguarding (A) and another member of staff who had more experience with IT systems (B).

1. (A) I'm not sure about this one, not my area of expertise!

(B) Nothing can be taken off the system and all personal data is completely secure. It has to be of the highest security possible

1. (A) GDPR policy, found on our website.

(B) Our GDPR policy from the trust (BEST) covers all of that.

1. (A) You personally? I don't think you are able to access any personal data about any student.

(B) With the Privacy policy fully read and understood, we can store what we need to lawfully. Biometrics require additional permissions.

1. (A) Yes. Detailed in our GDPR policy.

(B) Yes. All in the GDPR policy. But because of the type of data you won’t need to keep data past the child being in school, if it becomes a safeguarding issue we securely keep on another database.

1. (A) Depends on the type of data! sorry!

(B) Yes, all of our students are over 13. We also have an official subject access request process written into the policy and child can ask for information under that as they are all over 13 parents can ask, but chid has to consent as well.

1. (A) Depends on type of data! sorry!

(B) there are different levels of information for staff depending on the data. All information you need, all SWA staff can access (note, some staff are not SWA, they are contractors- the dinner staff, the site agents, etc)

1. (A) No, this will not be possible.

(B) Depends how secure your system is.

They also requested a feature to monitor concerning activity within the library, similar to the system used to notify the safeguarding team of any worrying web searches, which is something I will try to build into the system.

### Similar solutions

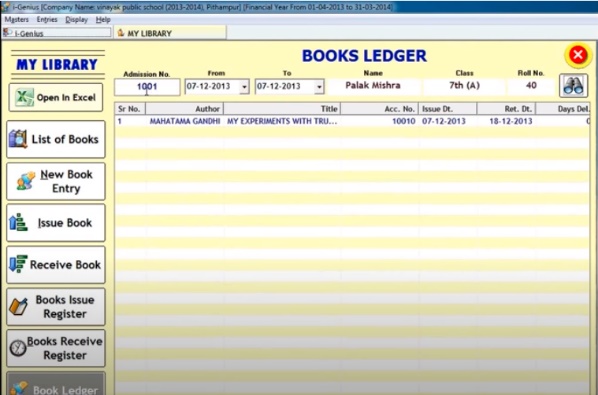
#### Example 1

(Fresh-USA, 2018)

 This is the book check-out terminal. It has a simple form based interface, with minimal functionality. However, this makes it very quick and easy to use. It features a built in barcode scanner, card reader and receipt printer which will not be available / required in my solution both due to cost and the fact that the library is free to use.

This is the book returns terminal. It is very quick to use, as users don’t need to sign in. It is also secure as returned books can’t be accessed as they are stored within the machine, important for a large commercial library. However, this style of terminal would be costly and excessive for a school library setting.

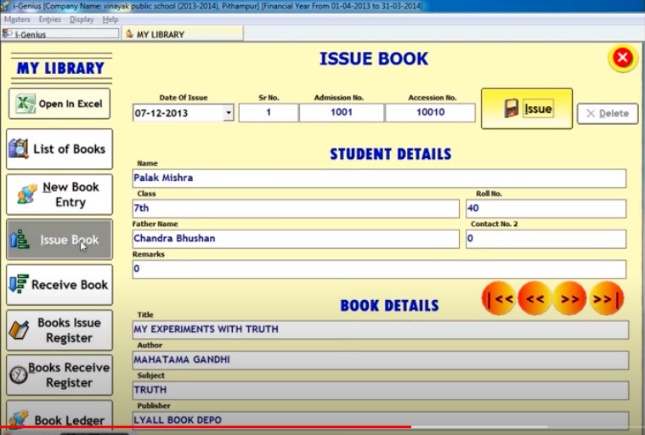
There is no functionality for admins shown in this video, so I cannot comment on the effectiveness of this aspect of the solution. However, admins would require more functionality than is shown here.



#### Example 2

(solverteam, 2013)

This is one of the report views for this solution. It is limited to viewing books issued, books returned, and a book ledger. All these can be searched by users, books or dates. However this is insufficient for our librarian, and there is no option to create or save a custom report.



This is the view for issuing a book, and there is a similar style view for adding or returning a book. Overall it looks cluttered and hard to use, however many fields auto-fill reducing the time taken to fill out this form.

This solution is clearly designed for use by the library admins, as a user can access all areas of the system (which would be a security risk if a student were using the system themselves), also it is overly cluttered and complex, meaning that it is not intuitive to use unless you are experienced.

#### Example 3

(MacWill, 2015)

This is the home screen for this solution, as you can see there are many tabs offering a wide variety of functions – from fines, to reports to automated letter writing. However, most of this functionality is unnecessary for our library, meaning that it overcomplicates use of the system.

This is one of the reports views, where you can select to view books in the library, or all books, and this can be filtered by accession number. While there are other options for reports, none of them fit our librarians needs and there isn’t an easy way to create and save a custom report.

Like the previous example, there are no access rights to prevent a user accessing areas they are not allowed to, which poses a security risk if students are able to sign books in and out themselves.

I would like to combine the successes of both types of solution by providing a split interface, for students it is simple and easy to use similar to solution 1, but teachers and admins can benefit from the extra functionality demonstrated in solutions 2 and 3.

## Stakeholders

There are a few requirements for stakeholders:

* The stakeholder must have a quantity of books / barcode identified items that can be borrowed or used for a length of time.
* The stakeholder must allow the use of computers, including use of a barcode scanner
* All items must be uniquely identifiable.

This system could be used in a variety of settings, but my stakeholders are my school, the librarian, and students and staff who use the library.

Students require a system that is quick and easy to use, which will prevent queues and lateness to lessons when signing books in and out. And which can remind them when a book is due, as they are renowned for being unreliable and unorganised. For the staff, the biggest requirement is automation of signing in and out books, as this will free up their time to do other things.

## Proposed solution

### Essential features

* Barcode reader used to identify books – maximises efficiency of the system
* User identity confirmed securely – to create a personalised user experience and enforce access rights
* Users can sign in and out books – fulfils the purpose of the system
* Users reminded of due dates – reduces teachers chasing students for overdue books and library becoming depleted
* Books catalogued and uniquely identifiable – so signing in and out is meaningful
* Staff able to monitor and view student and book information – required by admins
* Log of books taken out by specific users stored – required to complete the preceding point.
* Use of the system is quick – doesn’t cause other issues within school (lateness due to use of the system)
* The system can be operated by students without teacher supervision – requested in interviews
* System provides additional information on books – requested in interviews
* System can recommend books to students – requested in interviews
* Admins can create custom reports – requested in interviews
* System reports any worrying activity to safeguarding – requested in interviews
* Interface is simple and intuitive to use – from similar solutions
* Split interface for students and admins – from similar solutions

### Limitations

<https://www.gov.uk/service-manual/helping-people-to-use-your-service/understanding-wcag>

The program will require users to read and respond to text displayed on screen, therefore I will use high contrast colour schemes and easy to read fonts. Also, data is to be entered via a keyboard and mouse, therefore this may be challenging for people with limited mobility in their hands. I will ensure that the program can be navigated without a mouse. And that assistive technologies work effectively with the system. Overall, I will try to follow the guidelines set out in the web content accessibility guidelines(W3C, 2018)**.** to the best of my abilities, however if the system is unusable for any reason the librarian will be able to assist the user.

There are also technical limitations to the system, as the information will be stored in a database, it will be reliant on the network being functional and online. The system would also require specialist hardware to uniquely identify books using, for example, a barcode scanner. However, if this was to break the system may be unusable, unless a manual entry system is created.

### Requirements

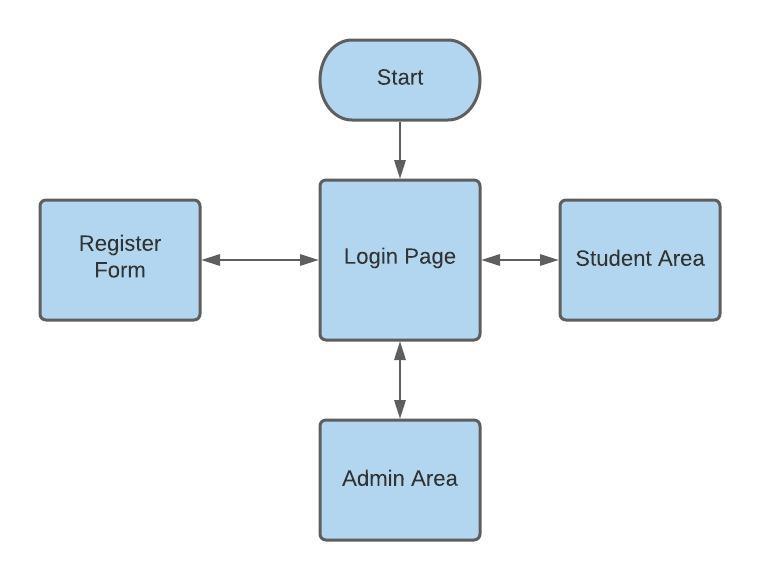
The system will be written in C#, as it is strongly typed and so reduces the likelihood of unpredictable errors at runtime. I will use the visual studio IDE and .NET core framework. The earliest compatible OS is windows 7 SP1, however this is no longer supported so I would recommend running the software on windows 10. Visual studio also recommends using a 1.8 Ghz processor ideally quad-core and having a minimum 2GB of RAM, (Microsoft, 2019) to ensure it runs efficiently. A barcode reader, mouse, keyboard, and monitor are required for input and output of information.

### Success criteria

|  |  |
| --- | --- |
| Criteria | Evidence required |
| Username and password to confirm user identity | Screenshot before and after entering these details |
| Register screen where user can enter their own details | Screenshot of register screen |
| Screen where admins (and only admins) can add books to the library | Screenshot of admin area and student area |
| Usable barcode scanner | Screenshot before and after barcode used to enter information |
| All areas can be navigated easily with a screen reader | Tests run with screen reader |
| Screen where students can sign books in and out | Screenshot of admin area |
| Email sent to students 2 days before a book is due | Screenshot of email received by a student account |
| Screen for creating reports | Screenshot of this |
| All books’, and registered users’ details stored in a database | Screenshot of book and user tables |
| Log of all books taken out stored in a database | Screenshot of returns table |
| Takes less than 1 minute to take out / return a book | Timed tests to determine speed |
| Students have their own logins, and can use the system unsupervised | Screenshot of student area |
| System recommends books, based on which are popular at the moment | Screenshot of system recommendations |
| Students can search for book recommendations based on genre | Screenshot of search recommendations |
| Safeguarding emailed when a number of flagged books are taken out | Screenshot of email received by safeguarding |
| Students and admins use different forms, to ensure simplicity | Screenshots of admin and student areas |

# Design of the solution

## Structure of the solution



Diagram

Description automatically generated

**Diagram

Description automatically generated**

## Decomposition of the problem

**First prototype**

1. Design and create the Database, using entity frameworks
2. Design and create login, register, userhome and adminhome forms for user interface
3. create system for entering data via barcodes
4. create system for entering data manually

**further prototypes and final system**

1. create service to send email when a book is nearly due
2. create custom reports forms for admins
3. create student monitoring for safeguarding
4. create recommendations system
5. create additional book information

after seeing a prototype of the system, the librarian requested that I add publication date and isbn numbers to the book table.

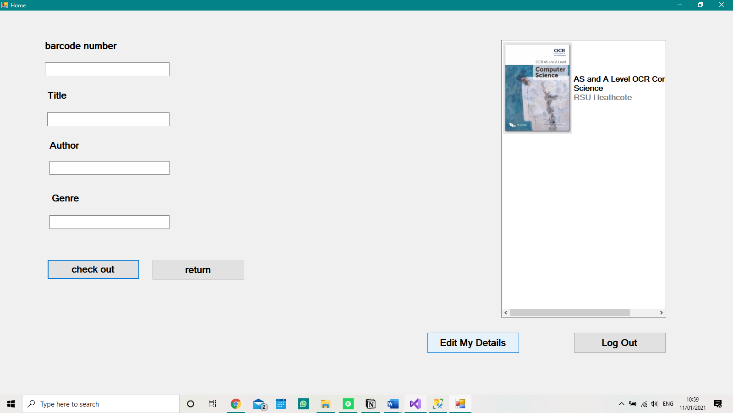
## Description of the solution

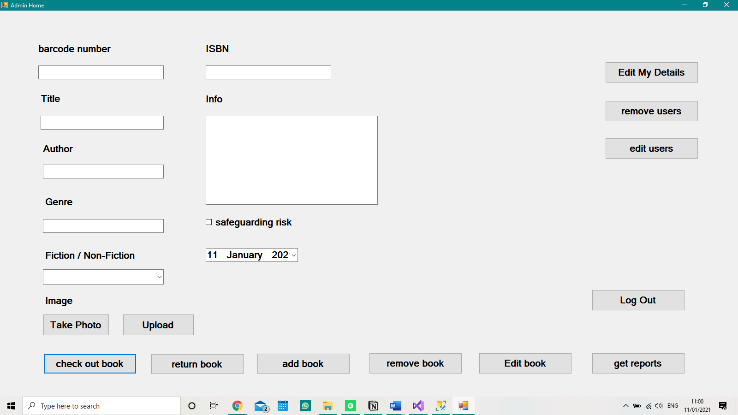
### Graphical user interface, application Description automatically generatedDesign

This is the login form, which is the first thing a user sees on loading the project. There are two textboxes (both labelled) for entering an email and password, and a button labelled “log in” to validate these inputs and move to the main forms. There is also a button labelled “register”, which loads a registration form where users can enter all of their details in order to create an account.

Graphical user interface, application

Description automatically generatedThis is the registration form, where there are 5 textboxes and 2 selection boxes for user input, all are labelled appropriately. The button labelled “done” then validates this data and creates an account for the user.

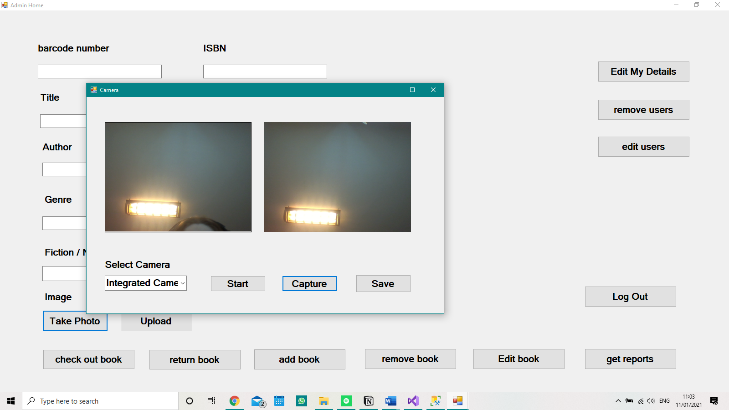
If the User’s account is a student account, after logging in they will be directed to the userhomeform. This is where books can be checked out and returned. There are 4 labelled textboxes, 3 of which autofill after the barcode is entered. The check out button then uses this input to create a log of this action, and the return button updates this log as the book has been returned. The log out button returns the user to the loginform so that the system can be used by another.

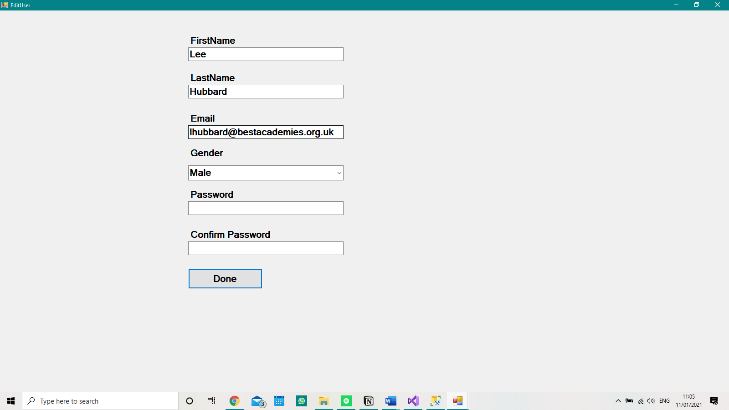
However, if the user logging in is an admin they are directed to the adminhomeform, which has all the functionality of the userhomeform, however there are some additional features. There are more textboxes and an add button, so that books can be added to the library. There is also a get reports button, which directs you to the reportsform.

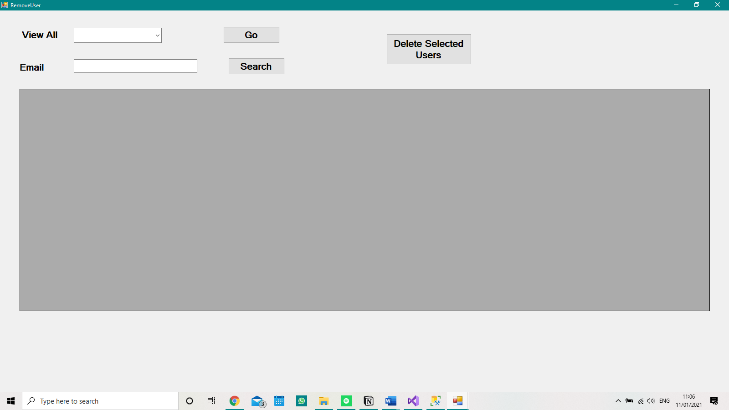
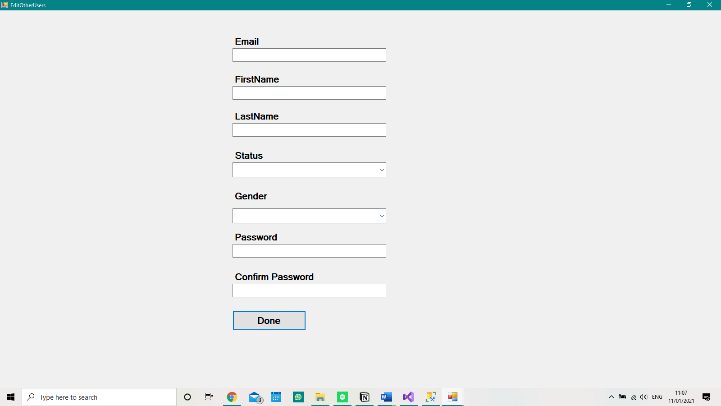
Graphical user interface, text, application

Description automatically generated

This is the reports form. There are two selection boxes, to specify the type (current loans, past loans or overdue books) and the scope (girls, boys, year groups, genre etc.), as well as a button labelled “go” which, when clicked, searches the database according to these terms. The textbox for entering the dewey decimal class is currently hidden, as the scope isn’t set to genre. The large data grid view displays the result of this search, in a read only table.

This is the camera form, there is a labelled selection box, where the user can select the appropriate webcam to use. The three buttons below this start the camera feed, take a photo from the camera feed, and save the snapshot. the top picture box shows a live feed from the computer’s webcam, and the bottom picture box shows an image captured from this webcam.

This is the Edit my details form, which allows a user to edit the information related to their account. It is laid out very similarly to the register form, however all of the data except the password is filled when the form loads.

This is the remove users form, there is a selection box, to narrow down the search results to one year group, with a button to apply this filter. As well as a textbox, for searching for an individual user and a button to apply this search. The results of both are displayed in a large data grid view below. There is also a button which deletes selected users in the data grid view from the database.

This is the edit other users form, it only appears in the admin area, and it allows them to edit details of a user other than themselves. The format is very similar to the register form.

### Selection of software

For my solution, a database will be necessary to store data about the books and students in the library, however the platform used to do this will impact how I create the rest of the program. I have narrowed down the choice to either Oracle, SQL server and Microsoft Access, and collated a table outlining some benefits and drawbacks of each potential platform.

|  |  |  |  |
| --- | --- | --- | --- |
|  | SQL Server | MS Access | Oracle |
| Size | 16 TB | 2 GB | 16 TB |
| Scalability | BETTER | WORSE | MUCH BETTER |
| Simplicity | WORSE | BETTER | MUCH WORSE |

It is clear to see that SQL server is the most suited to this project, despite it being harder to use than MS Access and is less scalable than oracle, overall it is the most suited to my project.

As well as determining the database platform software being used, I also must decide on the connection methods I use to access this data from within a C# program.

I have decided to use entity framework code first model in my program, this is a database mapper which maps between a database entities and class objects, which will enable me to interact with the database as if it were a group of classes within my program. I will also use LINQ to execute database transactions.

**Advantages and disadvantages of entity frameworks**

|  |  |
| --- | --- |
| advantages | disadvantages |
| Auto-generates code, which reduces development time. | Slower to load – however this is of little consequence as rate of accessing the database will be low |
| Allows developers to design and edit data models | Complicated syntax – I will be the only developer on this project so can manage this. |
|  | Not available for some relational database systems – but it is available for SQL server which I will be using |

**Advantages and disadvantages of code first model**

|  |  |
| --- | --- |
| advantages | disadvantages |
| No need to use DBMS to edit database structure | entities are created manually, which isn’t suitable for extremely large databases – my database will be relatively small |
| Easier to manage associations, foreign keys and constraints. | Any modifications made in the DBMS wont be reflected in the classes – I know this and won’t use the DBMS to edit the database structure |
| Can roll back to any version as changes are tracked | Need a knowledge of C# to create model – I have knowledge of C# so this isn’t an issue |
| No need for .edmx files or T4 scripts | Not beneficial for existing databases – the database is being created for this application |

|  |  |
| --- | --- |
| Advantages | disadvantages |
| Errors checked by compiler as query is not contained inside a string | Impacts performance on high traffic databases – there won’t be many transactions on my database |
| Works for many data platforms, including XML and SQL | Hard to write complex queries – most of the queries I will write are simple |
| It also allows debugging through .NET debugger. | Doesn’t take full advantage of features such as cached execution plan for stored procedures – I don’t plan to use this feature so it isn’t a problem |

### Database structure

To store data in a database, it must be broken up into tables, to reduce redundancy and repeated data. I will have three tables, books, returns and users which are specified below.

**Books**(Barcode, Title, Author, Genre, Fiction)

**Users**(Email, FirstName, LastName, Password, Status, Year, Gender)

**Returns**(LogId, *Barcode*, *Email*, DateOut, DateDue, Returned)

To ensure data is in third normal form, I have done several things. Firstly, I ensured there was a primary key for each table (as the barcode and email fields are unique these were used as the primary key, but a new key field was needed in the returns table), then I split the user’s name into FirstName and LastName, to make the data atomic. The author field is not atomic, however it can be treated as such, because the author of a book is only ever needed in its entire form. Furthermore, I ensured that there were no non-key dependencies or partial dependencies in my tables.

Diagram

Description automatically generated

### Key Variables

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Length | Example data | validation |
| Book.Barcode | String | 13 characters | 9780241987353 | BookExists(), to prevent duplicate rows being created |
| Book.Title | String | N/A | This Is Your Brain On Music |  |
| Book.Author | String | N/A | Daniel Levitin |  |
| Book.Genre | String | 3 characters | 781 |  |
| Book.Fiction | Boolean | 7 or 11 characters | Non-Fiction, Fiction |  |
| User.Email | String | N/A | 07hwheeldon@mybest.org.uk | UserExists() to prevent duplicate rows being created |
| User.Password | String | 16 characters | w?8???]???KHP:?? | Hashed using HMAC-SHA512 |
| User.Status | String | 10, 5, or 7 characters | Sixth-Form, Student, Staff | Selected from a combobox |
| User.Year | Integer | N/A | 2021 | Calculated from current date, and selected year group of the student. |
| User.Gender | Character | 1 character | F, M or N | Selected from a combobox |
| User.FirstName | String | N/A | Holly |  |
| User.LastName | String | N/A | Wheeldon |  |
| Returns.LogId | integer |  |  |  |
| Returns.BookBarcode | String | 13 characters | 9780241987353 |  |
| Returns.UserEmail | String | N/A | 07hwheeldon@mybest.org.uk |  |
| Returns.DateOut | DateTime | 2 bytes | 2020-09-09 10:48:51.7285658 | Current date and time when book is taken out |
| Returns.DateDue | DateTime | 2 bytes | 2020-10-07 10:48:51.7285658 | Calculated from adding a number of weeks to dateout based on the user’s status |
| Returns.Returned | Boolean | 1 bit | 1 or 0 |  |
|  |  |  |  |  |

As well as storing data in a database, I will store the current user’s username in appsettings, in order to know which user is currently logged in and pass this value to relevant functions.

### Classes

|  |  |  |
| --- | --- | --- |
| Name | Methods | Attributes |
| Book |  | string Title  string Barcode  string Genre  string Author  string Fiction  List<ReturnsLog> ReturnsLogs  string Image |
| User |  | string FirstName  string LastName  string Email  string Password  string Status  char Gender  int Year  List<ReturnsLog> ReturnsLogs |
| ReturnsLog | CloseLog() | int LogId  string BookBarcode  Book  string UserEmail  User  DateTime DateOut  DateTime DateDue  bool Returned |
| LibraryDbContext | OnConfiguring(DbContextOptionsBuilder optionsBuilder)  OnModelCreating(ModelBuilder modelBuilder) | DbSet<Book> Books  DbSet<User> Users  DbSet<ReturnsLog> Returns |
| LibraryDataFunctions | AddUser(string firstname, string lastname, string email, string password, string status, string year, char gender)  EditUser(string email, string password, string field, string value)  UserExists(string email)  LoginValid(string email, string password)  GetUser(string email)  RemoveUser(string email, string password)  AddBook(string title, string barcode, string genre, string author, string fiction, string image)  RemoveBook(string barcode)  GetBook(string barcode)  AddLog(string barcode, string email)  CloseLog(string barcode, string email)  BookExists(string barcode)  GetReport(string type, string scope)  Makehash(string password, string email)  GetReccomendations()  GetEmailBookDue() |  |
| Service1 | OnStart(string[] args)  OnStop()  ScheduleService()  SchedularCallback(object e)  WriteToFile(string text)  pop(List<string> list) | Timer Schedular  DateTime scheduledTime  TimeSpan timespan  string schedule  int dueTime  var mailMessage  var details  string name  string email  string title  string author  string days  var smtpClient  string text  string path  List<string> list  string value |
| ProjectInstaller | OnAfterInstall(IDictionary savedState) |  |
| LoginForm | GetInstance()  LoginFormInstance\_FormClosed(object sender, FormClosedEventArgs e)  btnRegister\_Click(object sender, EventArgs e)  LoginForm\_FormClosed(object sender, FormClosedEventArgs e)  btnLogin\_Click(object sender, EventArgs e) | LoginForm LoginFormInstance  RegisterForm f2  var configuration  UserHomeForm f2  var configuration  AdminHomeForm f2 |
| RegisterForm | GetInstance()  RegisterFormInstance\_FormClosed(object sender, FormClosedEventArgs e)  btnDone\_Click(object sender, EventArgs e)  RegisterForm\_FormClosed(object sender, FormClosedEventArgs e) | RegisterForm RegisterFormInstance  bool DataValid  LoginForm f1  LoginForm f2 |
| UserHomeForm | GetInstance()  UserHomeFormInstance\_FormClosed(object sender, FormClosedEventArgs e)  UserHomeForm\_FormClosed(object sender, FormClosedEventArgs e)  btn\_logout\_Click(object sender, EventArgs e)  tbxBarcode\_Leave(object sender, EventArgs e)  btn\_return\_Click(object sender, EventArgs e)  btn\_checkout\_Click(object sender, EventArgs e)  tbxBarcode\_TextChanged(object sender, EventArgs e)  logout()  UserHomeForm\_Load(object sender, EventArgs e) | UserHomeForm UserHomeFormInstance  Book currentbook  var book  var configuration  LoginForm f2  var reccomendations |
| AdminHomeForm | GetInstance()  AdminHomeFormInstance\_FormClosed(object sender, FormClosedEventArgs e)  AdminHomeForm\_FormClosed(object sender, FormClosedEventArgs e)  btnAddBooks\_Click(object sender, EventArgs e)  btn\_return\_Click(object sender, EventArgs e)  tbxBarcode\_Leave(object sender, EventArgs e)  btn\_checkout\_Click(object sender, EventArgs e)  btn\_logout\_Click(object sender, EventArgs e)  btn\_reports\_Click(object sender, EventArgs e)  btnUpload\_Click(object sender, EventArgs e)  btnTakePhoto\_Click(object sender, EventArgs e) | AdminHomeForm AdminHomeFormInstance  Book currentbook  var configuration  LoginForm f2  ReportsForm f2  DialogResult result  string source  string name  string dest  CameraForm f2 |
| ReportsForm | GetInstance()  ReportsFormInstance\_FormClosed(object sender, FormClosedEventArgs e)  ReportsForm\_FormClosed(object sender, FormClosedEventArgs e)  btnGo\_Click(object sender, EventArgs e)  cbxReportScope\_SelectedIndexChanged(object sender, EventArgs e) | ReportsForm ReportsFormInstance  AdminHomeForm f2 |
| CameraForm | btnStart\_Click(object sender, EventArgs e)  OpenCamera()  UpdateCaptureSnapshotManifast(Bitmap image)  OpenVideoSource(IVideoSource source)  getListCameraUSB()  CloseCurrentVideoSource()  btnCapture\_Click(object sender, EventArgs e)  videoSourcePlayer1\_NewFrame\_1(object sender, ref Bitmap image)  btnSave\_Click(object sender, EventArgs e) | FilterInfoCollection videoDevices  VideoCaptureDevice videoDevice  VideoCapabilities[] snapshotCapabilities  ArrayList listCamera  string pathFolder  bool needSnapshot  string \_usbcamera  string nameCapture |

### Key functions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Purpose | Input | output | Data validation |
| AddUser | Add a row to database table users, with the information in the parameters | string firstname, string lastname, string email, string password, string status, string year, char gender | Save to database |  |
| EditUser | Alter a field in the row with the specified email and password, to have the passed value | string firstname, string lastname, string email, string password, string status, string year, char gender | Save to database |  |
| User exists | Checks whether a row exists with this email | string email | boolean |  |
| LoginValid | Determines whether the email and password are associated with a registered user | string email, string password | String status, or “false” |  |
| GetUser | Get the user with the email in the parameter | string email | Var Currentuser |  |
| RemoveUser | Deletes the specified user from the database | string email, string password | Delete from database |  |
| AddBook | Add a row to database table books, with the information in the parameters | string title, string barcode, string genre, string author, string fiction, string image, bool safeguarding, string info, string isbn, string publicationdate | Save to database | Checks if the book is already registered using bookexists function. |
| RemoveBook | Deletes the specified book from the database | string barcode | Delete from database |  |
| EditBook | Edits attributes of the specified book in the database | string title, string barcode, string genre, string author, string fiction, string image, bool safeguarding, string info, string isbn, string publicationdate | Dave to database | Checks that the barcode is not null |
| GetBook | Get the user with the barcode in the parameter | string barcode | Var currentbook | Checks the book is registered using bookexists function |
| AddLog | Add a row to the Returns table, with the barcode and email linking to appropriate rows in the other tables | string barcode, string email | Save to database | Checks that returned is not null or true |
| CloseLog | Alter the appropriate row, so that returned = true | string barcode, string email | Save to database | Checks if the user associated with this log, is the same as the logged in user |
| BookExists | Determines whether the book with specified barcode exists in the database | string barcode | Boolean |  |
| GetReport | Returns a list of entities to be viewed in a report, specified by the type and scope of the search | string type, string scope | Object report |  |
| MakeHash | Makes a hash with the user’s password, and the email as the salt, using the HMAC-SHA512 function | string password, string email | String passhash |  |
| GetReccomendations | Gets a list of the most recently returned books, that may be of interest to students |  | String[,] recommendations |  |
| GetEmailBookDue | Gets a list of details of students who have books due in the next 2 days |  | List<string> name, list<string email, list<string> title, list<string> author, list<string> days |  |
| GetSafeguardingConcern | Gets a list of users who have taken out 3 or more flagged books in the past 3 months |  | List<string> concern |  |
| logout | Logs the user out of the system, by resetting the username in appsettings and closing the form. |  |  |  |
| OnConfiguring | Sets data connection, used when entitymodel is being configured | DbContextOptionsBuilder optionsBuilder |  |  |
| OnModelCreating | Configures the model (setting primary and foreign keys and creating relationships), using fluid api | ModelBuilder modelBuilder |  |  |
| OnStart | Runs when the service is started, writes to the file log and calls ScheduleService | string[] args | Text is written to log file |  |
| OnStop | Runs when the service is stopped, writes to the file log and disposes of schedular |  | Text is written to log file |  |
| ScheduleService | Finds if the current time is after the scheduled time for the service, then alters the timer to run the next day. Writing it’s progress to the log |  | Text is written to log file |  |
| SchedularCallback | Sends email to all students which have books due in the next 2 days | object e | Email sent to students |  |
| WriteToFile | Code for writing text to the log file | string text | Text is written to log file |  |
| pop | Code for popping the front item from a stack which has been implemented with a list | List<string> list | String value |  |
| OnAfterInstall | Starts the service after it has been installed on the computer | IDictionary savedState |  |  |
| GetInstance | Opens a form or gets the running instance of the form |  | LoginForm.LoginFormInstance |  |
| “”FormInstance\_FormClosed | Sets loginforminstance to null | object sender, FormClosedEventArgs e |  |  |
| btnRegister\_Click | Opens the register form | object sender, EventArgs e |  |  |
| LoginForm\_FormClosed | closes the application | object sender, FormClosedEventArgs e |  |  |
| btnLogin\_Click | Validates the data entry, then either shows an error message or saves the user details to appsettings and opens the userhomeform or the adminhomeform depending on the user which has signed in | object sender, EventArgs e | Error message, or appsettings[“Username”] | The loginvalid function is used to check if the username and password are associated with a registered user. |
| btnDone\_Click | Validates data entry, and saves the user in the database – creating an account for them | object sender, EventArgs e | Save to database | First checks all data input fields are non null, then checks the two password inputs are identical, then checks if the email address is already registered. |
| RegisterForm\_FormClosed | Opens the loginForm | object sender, FormClosedEventArgs e |  |  |
| “”HomeForm\_FormClosed | Run logout | object sender, FormClosedEventArgs e |  |  |
| btn\_logout\_Click | Run logout | object sender, EventArgs e |  |  |
| btn\_return\_Click | Runs LibraryDataFunctions.CloseLog and reset | object sender, EventArgs e |  | Checks if the barcode field is null |
| btn\_checkout\_Click | Runs LibraryDataFunctions.AddLog and reset | object sender, EventArgs e |  | Checks if the barcode field is null |
| tbxBarcode\_Leave | Auto-fills the textboxes when a barcode is entered to the system, or responds with an error message if the book cannot be found | object sender, EventArgs e |  | Checks if the book is registered, using the bookexists function |
| Logout | Removes the user’s details from appsettings and loads the loginform |  | Alters Appsettings[“username”] |  |
| Reset | Clears all data input fields |  |  |  |
| UserHomeForm\_Load |  | object sender, EventArgs e |  |  |
| btnAddBooks\_Click | Run LibraryDataFunctions.AddBook and reset | object sender, EventArgs e |  | Checks the barcode, title and author fields are not null. |
| btn\_reports\_Click | Opens reports form | object sender, EventArgs e |  |  |
| btnUpload\_Click | Opens file dialogue, and copies selected file to correct directory | object sender, EventArgs e | String dest |  |
| btnTakePhoto\_Click | Opens CameraForm, and creates an addressupdatehandler event | object sender, EventArgs e |  |  |
| CameraForm\_ButtonClicked | Sets the destination and filename, and makes the the correct buttons visible |  |  |  |
| btnRemove\_Click | Calls the removebook, and reset | Object sender, EventArgs e | Deleted from database | Checks that barcode isn’t null |
| btnEdit\_Click | Calls editbook and reset | Object sender, EventArgs e | Altered in database |  |
| btnEditDetails\_Click | Opens the edituser form | Object sender, EventArgs e |  |  |
| btn\_editusers\_Click | Opens the editotherusers form | Object sender, EventArgs e |  |  |
| ReportsForm\_FormClosed | Return to adminhomeform | object sender, FormClosedEventArgs e |  |  |
| btnGo\_Click | Gets the appropriate data and displays it in the data grid view | object sender, EventArgs e |  |  |
| cbxReportScope\_SelectedIndexChanged | Makes the dewey decimal type textbox visible | object sender, EventArgs e |  |  |
| btnStart\_Click | Search for and selected webcam device, calls openvideosource | object sender, EventArgs e |  |  |
| UpdateCaptureSnapshotManifast | Sets need snapshot to false, and fills picturebox with a bitmap image from the webcam | Bitmap image |  |  |
| OpenVideoSource | Starts the live feed from the camera in videosourceplayer | IVideoSource source |  |  |
| getListCameraUSB | Adds all available webcam devices to combobox |  |  |  |
| CloseCurrentVideoSource | Stops the webcam live feed to the videosourceplayer |  |  |  |
| btnCapture\_Click | Sets the picturebox1 to the current video frame of videosourceplayer | object sender, EventArgs e |  |  |
| btnSave\_Click | Saves the image in picturebox and closes the form | object sender, EventArgs e | addressupdateeventargs |  |
| CameraForm\_FormClosed | Closes the video source, and the camera form, and opens the adminhomeform | Object sender, EventArgs e |  |  |

### Libraries used

|  |  |
| --- | --- |
| Library | Purpose |
| System | Contains fundamental classes and base classes that define commonly-used value and reference data types, events and event handlers, interfaces, attributes, and processing exceptions. |
| System.Configuration | Contains the types that provide the programming model for handling configuration data. |
| System.Reflection | Contains types that retrieve information about assemblies, modules, members, parameters, and other entities in managed code by examining their metadata. |
| System.Windows.Forms | Contains classes for creating Windows-based applications that take full advantage of the rich user interface features available in the Microsoft Windows operating system. |
| System.IO | Contains types that allow reading and writing to files and data streams, and types that provide basic file and directory support. |
| System.Drawing | Provides access to GDI+ basic graphics functionality |
| Aforge.Video | contains interfaces and classes to access different video sources. |
| Aforge.Video.DirectShow | contains classes, which allow access to video sources using DirectShow interface. |
| System.Collections | Contains interfaces and classes that define various collections of objects, such as lists, queues, bit arrays, hash tables and dictionaries. |
| System.Drawing.Imaging | Provides advanced GDI+ imaging functionality. Builds on functionality in system.drawing |
| System.ServiceProcess | Provides classes that allow you to implement, install, and control Windows service applications. Services are long-running executables that run without a user interface. Implementing a service involves inheriting from the servicebase class and defining specific behavior to process when start, stop, pause, and continue commands are passed in, as well as custom behavior and actions to take when the system shuts down. |
| System.ComponentModel | Provides classes that are used to implement the run-time and design-time behavior of components and controls. This namespace includes the base classes and interfaces for implementing attributes and type converters, binding to data sources, and licensing components. |
| System.Collections.Generic | Contains interfaces and classes that define generic collections, which allow users to create strongly typed collections that provide better type safety and performance than non-generic strongly typed collections. |
| System.Linq | Provides classes and interfaces that support queries that use Language-Integrated Query (LINQ). |
| System.Threading | Provides classes and interfaces that enable multithreaded programming. In addition to classes for synchronizing thread activities and access to data ([Mutex](https://docs.microsoft.com/en-us/dotnet/api/system.threading.mutex?view=dotnet-plat-ext-3.1), [Monitor](https://docs.microsoft.com/en-us/dotnet/api/system.threading.monitor?view=dotnet-plat-ext-3.1), [Interlocked](https://docs.microsoft.com/en-us/dotnet/api/system.threading.interlocked?view=dotnet-plat-ext-3.1), [AutoResetEvent](https://docs.microsoft.com/en-us/dotnet/api/system.threading.autoresetevent?view=dotnet-plat-ext-3.1), and so on), this namespace includes a [ThreadPool](https://docs.microsoft.com/en-us/dotnet/api/system.threading.threadpool?view=dotnet-plat-ext-3.1) class that allows you to use a pool of system-supplied threads, and a [Timer](https://docs.microsoft.com/en-us/dotnet/api/system.threading.timer?view=dotnet-plat-ext-3.1) class that executes callback methods on thread pool threads. |
| Mimekit | provides classes that are used to implement the core MIME parsing services of the mailkit framework. |
| MailKit.Net.Smtp | provides classes that are necessary for sending messages to an SMTP server. |
| Microsoft.EntityFrameworkCore | a lightweight and extensible version of the popular Entity Framework data access technology. |
| System.Text | Contains classes that represent ASCII and Unicode character encodings; abstract base classes for converting blocks of characters to and from blocks of bytes; and a helper class that manipulates and formats [String](https://docs.microsoft.com/en-us/dotnet/api/system.string?view=netcore-3.1) objects without creating intermediate instances of [String](https://docs.microsoft.com/en-us/dotnet/api/system.string?view=netcore-3.1). |
| Microsoft.AspNetCore.Cryptography.KeyDerivation | Infrastructure for ASP.NET Core cryptographic packages and ASP.NET Core utilities for key derivation. |
| Microsoft.EntityFrameworkCore.Internal |  |

### Algorithm designs

**LibraryDataFunctions - Login valid**

FUNCTION LoginValid(email, password)

Hash🡨 MakeHash(password, email)

IF there is a record with this email address THEN

User🡨 user from database where email and hashed password are correct

IF user is NULL THEN

RETURN “false”

ELSE

RETURN status of user

ENDIF

ENDIF

RETURN “false”

ENDFUNCTION

**LibraryDataFunctions - Add Log**

PROCEDURE AddLog(barcode, email)

returned 🡨 log from database where barcode is correct

IF returned is NOT NULL THEN

IF Returned attribute of returned is NOT TRUE THEN

OUTPUT(error message)

ELSE

NEW returnsLog(datetime.now, barcode, email)

ENDIF

ELSE

NEW returnsLog(datetime.now, barcode, email)

ENDIF

ENDPROCEDURE

**LibraryDataFunctions - Get Recommendations**

FUNCTION GetReccomendations()

Recommendations 🡨 array of string

Recents 🡨 list of returned books from returns sorted in descending order

Max 🡨10

IF recents.count <10 THEN

Max 🡨recents.count

ENDIF

FOR i = 0 to max

Item 🡨 recents(i)

Bookisout🡨the most recent log of the book, if it has been returned

IF bookisout is NULL THEN

Recommendations 🡨item

ENDIF

NEXT

RETURN recommendations

ENDFUNCTION

**CameraForm – btnSave\_Click**

PROCEDURE btnSave\_Click()

Name🡨 “BookCover\_” + datetime.now + “.bmp”

Path 🡨 pathfolder + name

IF directory exists(pathfolder) THEN

Picturebox1.Save

ELSE

Create directory(pathfolder)

Picturebox1.save

ENDIF

Stop video source

New addressupdateeventargs(path, name)

New adminhomeform

ENDPROCEDURE

**Login form – btnLogin\_Click**

PROCEDURE btnLogin\_Click()

String LoginValid 🡨LoginValid(email, password)

IF LoginValid = “Student” OR LoginValid = “Sixth-Form” THEN

APPSETTINGS[“username”] 🡨 email

Clear textboxes

Open userhomeform

ELSE IF LoginValid 🡨 “staff” OR LoginValid 🡨 “VIP” THEN

APPSETTINGS[“username”] 🡨 email

Clear textboxes

Open adminhomeform

ELSE IF LoginValid = “false” THEN

OUTPUT(error message)

Clear textboxes

END IF

END PROCEDURE

**Register Form – btnDone\_Click**

PROCEDURE btnDone\_Click()

Bool DataValid 🡨 true

IF any textboxes are empty THEN

OUTPUT(error message)

DataValid 🡨 false

ENDIF

IF passwords do not match THEN

OUTPUT(error message)

DataValid 🡨 false

Clear password textboxes

ENDIF

IF DataValid = true THEN

Set status and gender attributes in the correct format

IF userexists(email) = false THEN

AddUser()

Open login form

ELSE

OUTPUT(error message)

Open login form

END IF

END IF

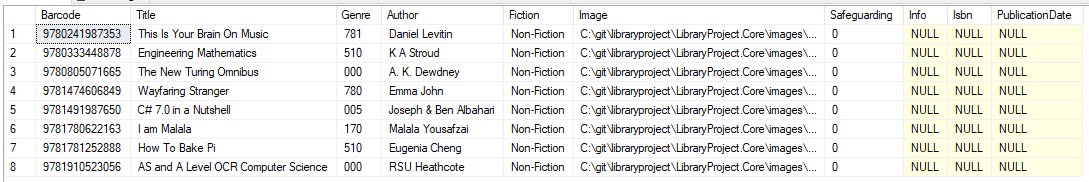
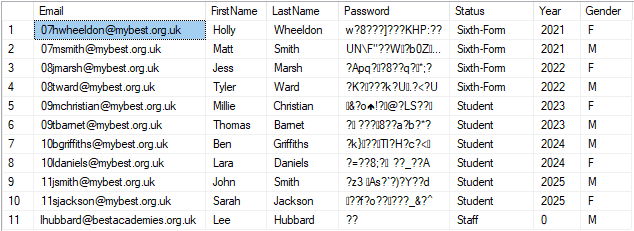
END PROCEDURE

## Usability features

To ensure that the software is usable by as many users as possible, several additions have been made to the user interfaces. These being:

* **Large text.** By having large font, people who have visual impairments will be able to read the instructions in the program. This will also make it easy to read for any student.
* **Screen reader compatibility.** text-to-speech algorithms will be able to interpret text and read it aloud to blind users, as the user interface will be organised in a clear and easily understandable way.
* **Clear instructions.** The designs of the user interfaces have been made to include minimal, but concise, instructions. This helps to minimise the amount of training required to use the program.
* **Separated modes for different users.** Since pupils will not need to use the administrator section of the system, the two have been separated. This helps to ensure that confidential information remains confidential and helps to ‘clean’ the user interface of any unnecessary information for each user type.

## Approach to testing

I will test the system using a small sample of books as testing on the scale of a library will be time consuming, and unnecessary. These books will be taken from my personal bookshelf, as I am unable to access the school library during the coronavirus outbreak causing schools to close. However, it would be more appropriate to use a sample of books from the library. I will also create a small group of sample users, which will be selected to cover all different groups and maximise the efficiency of testing.

I used my own email, and 9 fake emails as test accounts for students, ensuring I had a girl and a boy from every year group so that this data covered all groups and combinations of these. I needed to ensure that at least one email was valid in order to test the email reminder system. I also used my teacher’s email address, to test the staff areas of the system. For ease of use all accounts have a password of “password”, however as they are stored using a salted hash this is not evident from the database table.

To complete the email reminders and safeguarding features, I have created two email accounts so that a real email can be sent to test this as I was unable to get access to the email accounts currently used in the school. These were [samuelwhitbreadlibrary@gmail.com](mailto:samuelwhitbreadlibrary@gmail.com), and [samuelwhitbreadsafeguarding@gmail.com](mailto:samuelwhitbreadsafeguarding@gmail.com).

# Developing the solution

It is important, before the system is made operational, to test if the program has any glitches or bugs. The program was tested in two different ways; these being white box testing and alpha testing.

## Justification of Methods Used

White box testing has been used to test the system due to the system’s size. There are a large amount of routines and subroutines to test, and black box testing may not test all of these functions.

Alpha testing has been used to test the program as it is important to check that the system has the required functionality to complete the task it has been set out to do. It may be found that the requirements of the system were not specified carefully enough, or there are functions that have not been included in the program that are required for it to function properly.

By using both of these methods of testing together, all areas of the program are tested at least once.

## White Box Testing

The results collected, and the changes made due to this testing, can be found within the code analysis section of the report for each separate ‘block’ of code.

## Alpha testing

This was carried out with a group of volunteers, who joined a ‘library’ where they could borrow and return books over a period of a few months, the library was managed using the software I have developed. This is especially significant for the service aspect of the program, as this cannot be debugged conventionally through the IDE, it must be run by the operating system.

From this testing it was discovered that entering books one at a time can be time consuming, if a large number of books are required. This is a feature that could be developed in the future, however it requires a complete redesign of the home forms, and the situation didn’t occur frequently, so it wasn’t an urgent issue.

Another thing I learnt from this testing was that the service appeared to work as expected. However it sent a reminder email for every book, rather than one email that showed all books relevant for that user, this could be another potential improvement, as students disliked receiving many emails from the library.

## Code analysis

The analysis of the code was completed in parts, a document of code at a time. This was found to be a much easier method of analysis than analysing the code in chronological order as it was easier to differentiate analysed code from unanalysed code. An explanation of the program’s code, along with analysis on each section, can be found below:

### Book.cs

The book class stores data associated with the books table in the database. It contains attributes for each column in the table, and a constructor to assign values to each of these.

### ReturnsLog.cs

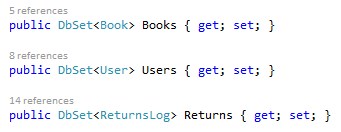
The book class stores data associated with the returns table in the database. It contains attributes for each column in the table, and a constructor to assign values to each of these.

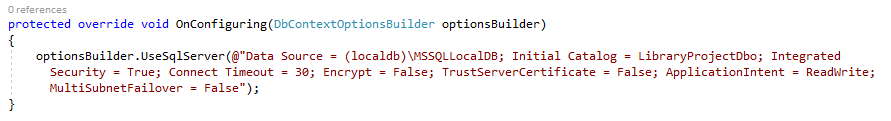
### User.cs

The book class stores data associated with the users table in the database. It contains attributes for each column in the table, and a constructor to assign values to each of these.

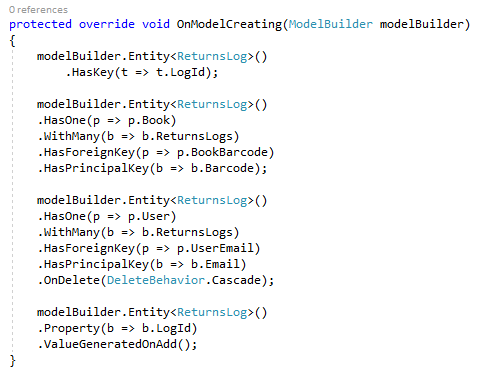
### LibraryDbContext.cs

This class manages connections to the database. The context opens and closes connections as needed. For example, the context opens a connection to execute a query, and then closes the connection when all the result sets have been processed.

First properties are declared, which represent collections of the specified entities in the context. These properties relate directly to tables within the database.

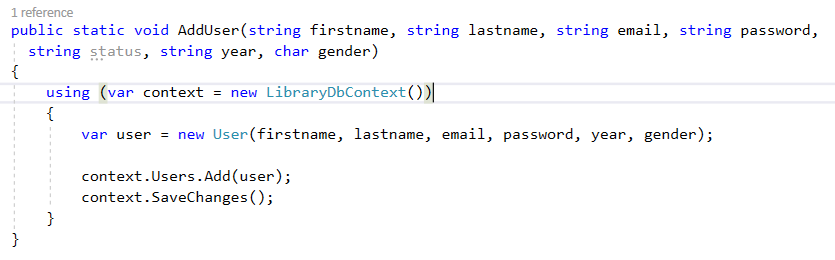


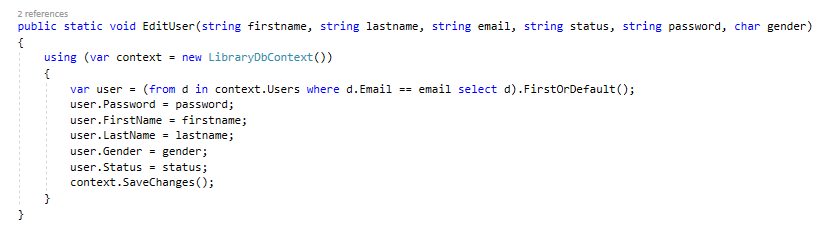
This function is run when the context is configured, and contains the connection string for the database.

This function is run when the database model is created, and helps map properties in the classes onto the database, including the relationships between entities, and auto generated values (such as the LogId attribute).

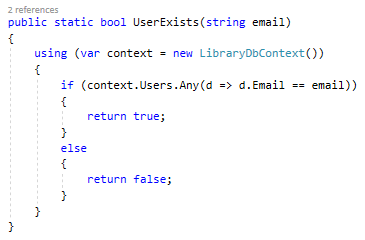
### LibraryDataFunctions.cs

This class contains all the functions for interacting with the database, including adding, editing and removing entries, as well as returning values from the database. It also contains the function for hashing the password, as it is the most accessible location for this function.

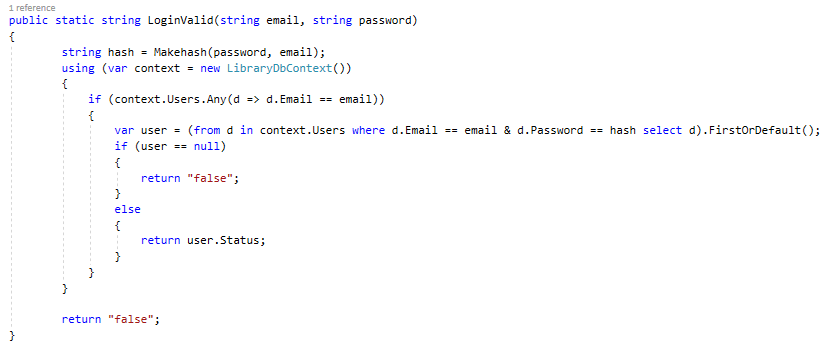
This function is used to add a user to the Users table, by creating a new instance of the User class, and adding it to the dbset Users from the LibraryDbContext class. It is called when the user registers.

This function is used to edit the properties associated with a user after it has been created, the current values associated with the user are retrieved using a LINQ query to return the user with the correct email, and then all of the other properties are updated to their new values.

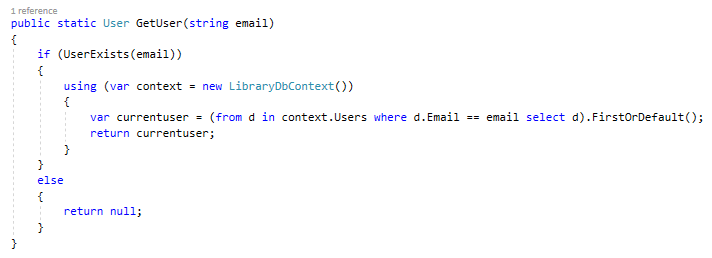
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Purpose | Data | Expected result | Actual result | action |
| Determine if a user can be altered using the edituser function | LibraryDataFunctions.EditUser("Holly", "Wheeldon", "07hwheeldon@mybest.org.uk", "sixth-form", "password", "F") | First name of record in user table is changed from sample to changed | As expected |  |



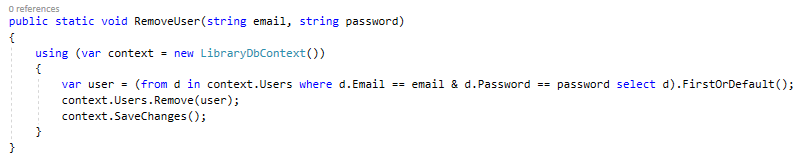
This function is used to determine if there is a row in the database, with a corresponding primary key to the email parameter using a linq query.



This function is used to determine if the email and password provided match those of a row in the database, using a Linq query. It returns the status of the user if this is true (allowing the user to be directed to the correct home page) or false if they do not match.

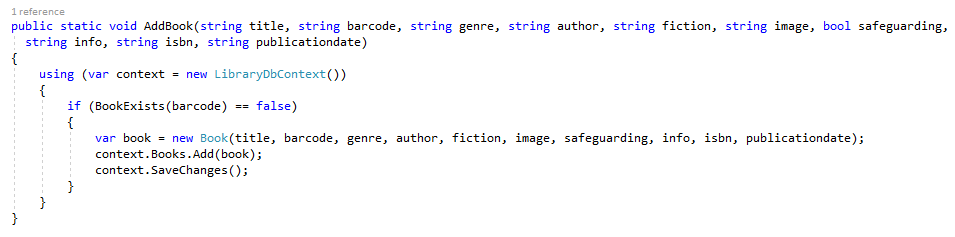


This function returns the user with the matching email attribute to the parameter using a linq query.

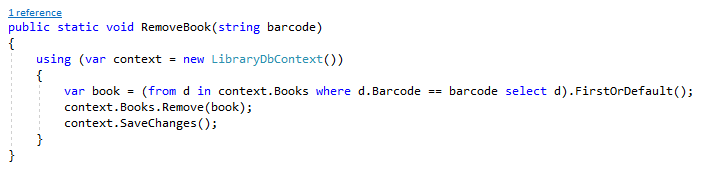


This uses a linq query to retrieve the user associated with the email and password parameters, and then deletes this user from the dbset users.

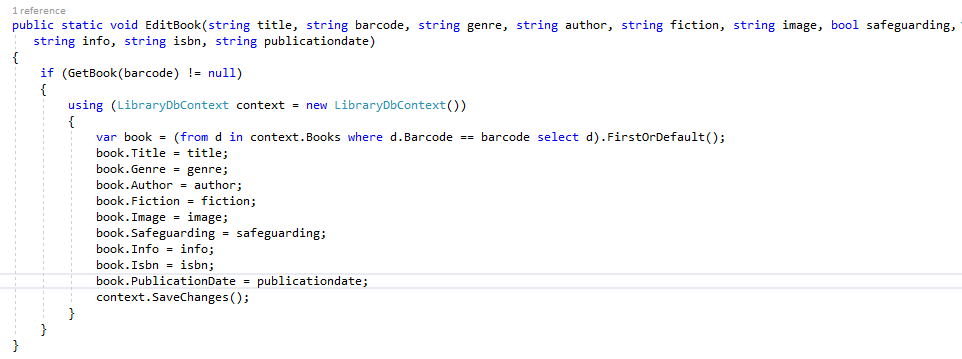
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| purpose | data | Expected result | Actual result | action |
| Determine if a user can be deleted by calling removeuser | LibraryDataFunctions.RemoveUser("myemail@email.com", "password"); | User removed from users table in database | **System.InvalidOperationException**  due to there being two identical rows in the table | change .single() to .FirstOrDefault() |
| Determine if a user can be deleted by calling removeuser | LibraryDataFunctions.RemoveUser("myemail@email.com", "password"); | User removed from users table in database | As expected |  |



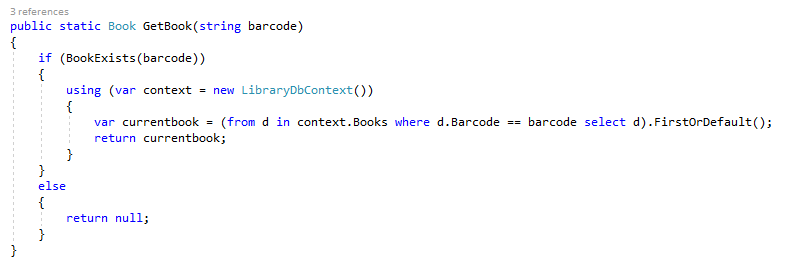
This function is used to add a book to the database, first it checks if the book already exists and then creates a new instance of the book class and adds this to the dbset books.



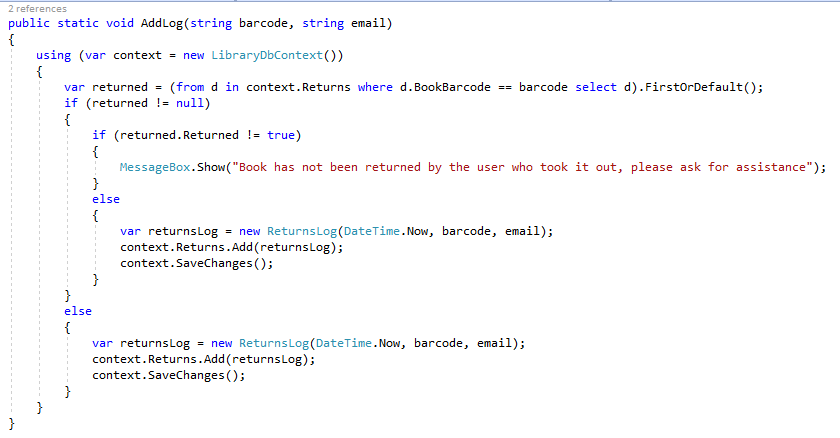
This function retrieves the book with the matching barcode attribute to the parameter, and deletes this record.



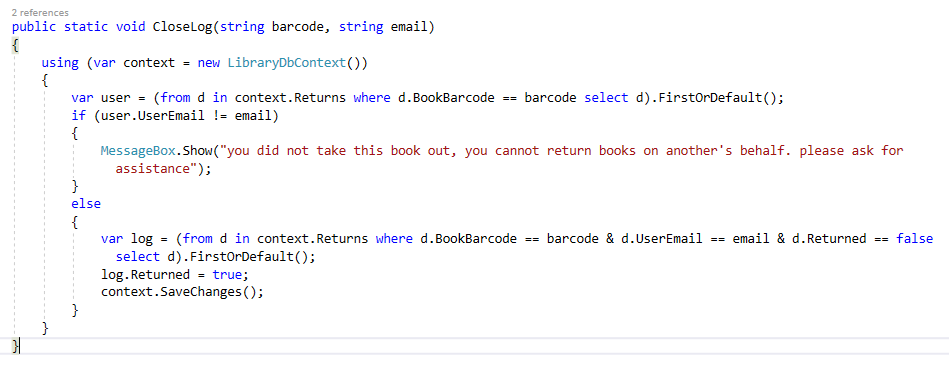
This function is used to edit the properties associated with a book after it has been created, the current values associated with the book are retrieved using a LINQ query to return the book with the correct barcode, and then all the other properties are updated to their new values.



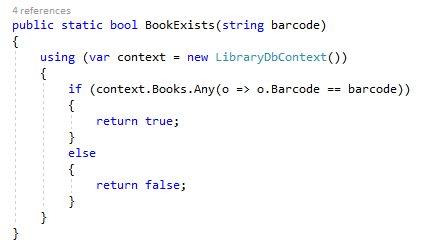
this function is used to retrieve the associated record with the barcode in the parameter, and returns this book.



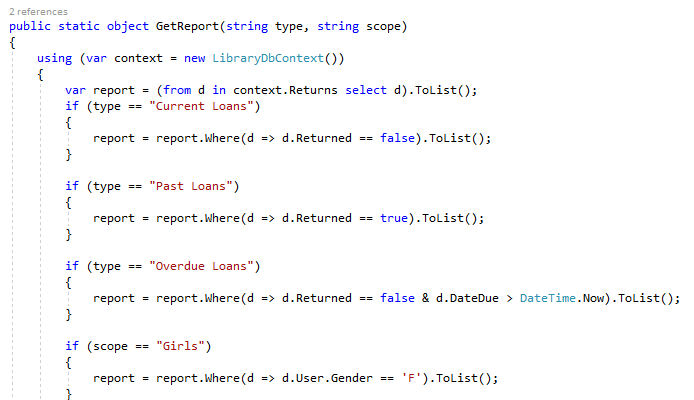
This function checks if the book is available to be taken out, and if so it creates an instance of returnsLog and adds this to the dbset returns.



This function checks if this user took this book out, and then uses a linq query to return the open log of this user taking this book out, and sets returned to true.



This function checks if there is a record with the barcode in the parameter, returning true if this is the case.

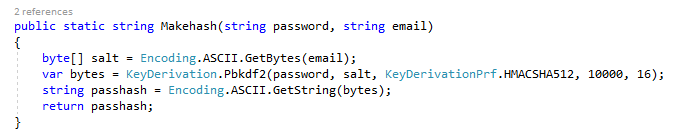


+ many more if statements

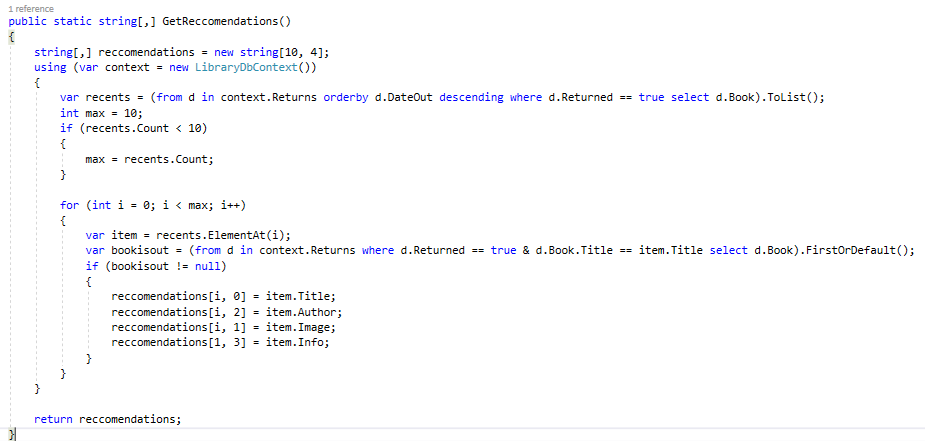
This function is used to retrieve records from the returnslog table, according to two categories (type and scope). Type refers to whether the loan is current, overdue or past, and scope narrows this down depending on the group of users (girls, boys, year groups) or by the genre of the book……..



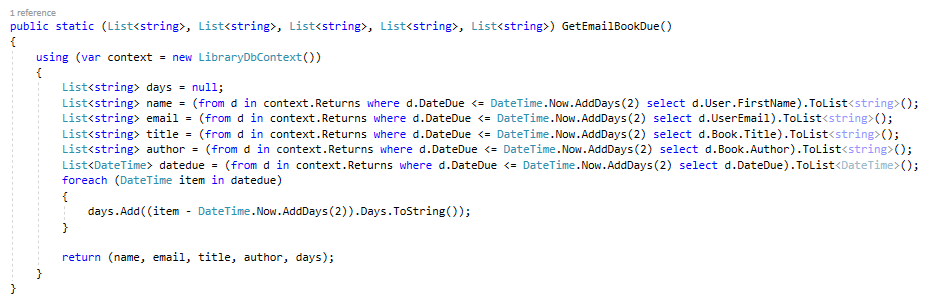
This function is used to return a list of all users from each year-group, which is used when deleting users. First the constant used to determine year groups is set dependant on whether it is September to December, or January to August. Then if statements are used to alter the linq query used to select users.



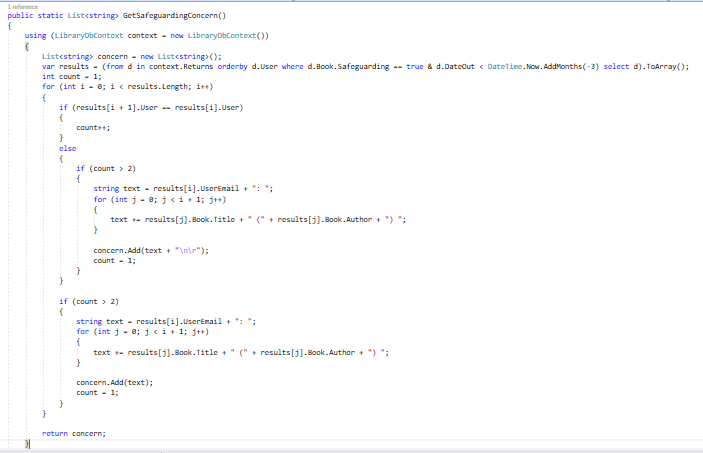
This function is used to hash passwords to be stored in the database, this is done using a salted hash (where the primary key is used in the hash as well as the data to be stored in order to eradicate duplicate hashes being stored for identical passwords). I used the SHA-512 function, considered to be very secure as there are no known attacks on this algorithm, which I accessed through the keyderivation.pbkdf2 library I installed, as it is faster than the inbuilt hashing libraries in .net framework.



This function returns a list of books which have been recently read and returned by users, as last recently used items is a good indicator of what will be wanted in the near future and is used in situations such as computer caching. This was much easier to implement than a recommendations system based on data analysis or machine learning, and may be satisfactory in providing books students may be interested in.

First the most recently returned books are retrieved using a linq query, then these are iteratively added to a 2 dimensional array of max length 10, this array is then returned.

This function is used to return all users with books overdue, so they can be emailed. 4 linq queries retrieve the name, email address, title and author of the book that is overdue, and the date it was due (which is then used to calculate the number of days overdue the book is). Then these are all returned as lists of type string.



This function is used to calculate whether a student has taken out lots of concerning books, and the safeguarding team should be contacted. First a linq query is used to return all of the loans of a book where safeguarding concern is true from the last three months, then a for loop cycles through all of these, and finds if runs of length 3 or more from the same user, if so these are added to a string containing all of the users and corresponding concerning books they have taken out which is returned.

Because this function is only ever called by the service, it couldn’t be debugged using the visual studio IDE, so I completed a dry run.

Results[] contained this data:

|  |  |  |
| --- | --- | --- |
| 0 | Holly | Book1 |
| 1 | Holly | Book2 |
| 2 | Holly | Book3 |
| 3 | Rachel | Book1 |
| 4 | Rachel | Book2 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Count | i | Results(i+1).user | Results(i).user | j | Results(j) |
| 1 | 0 | Holly | Holly |  |  |
| 2 | 1 | Holly | Holly |  |  |
| 3 | 2 | Rachel | Holly | 0 | Book1 |
|  |  |  |  | 1 | Book2 |
|  |  |  |  | 2 | Book3 |
| 1 | 3 | Rachel | Rachel |  |  |
| 2 |  |  |  |  |  |

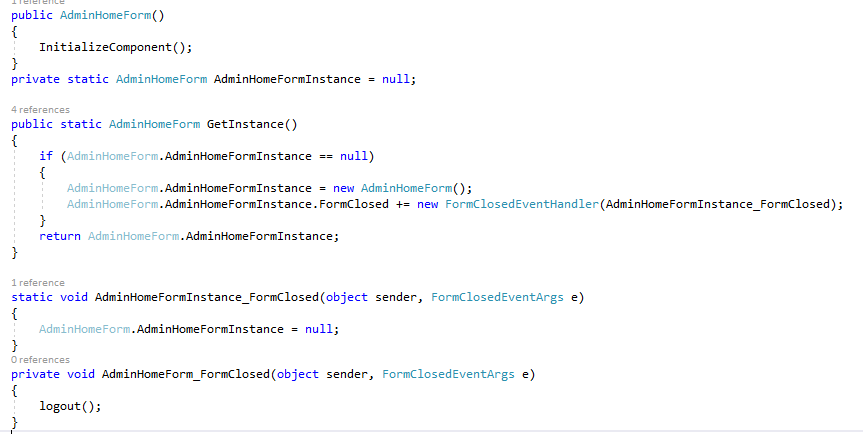
This provided the correct result, so it was successful.

### AdressUpdateEventArgs.cs

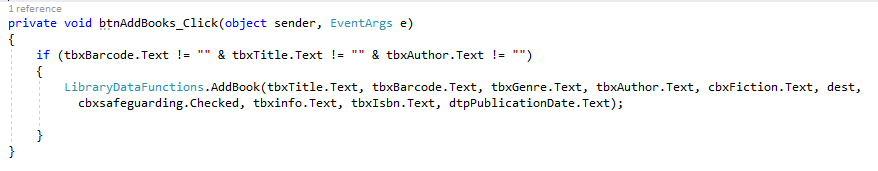
This file contains the code used to pass the name and address of a photo taken in the cameraform back to the adminhomeform. It includes a constructor which sets the values of the private variables - paddress and pname, and two read only public attributes which return the values of the private variables.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| purpose | data | Expected result | Actual result | Action |
| Passing file location and name |  | When save button on cameraform is clicked, path of file is copied to tbxfilename is adminhomeform | Doesn’t happen | Make the textbox visible |
| Passing location and name |  | When save button is clicked, path of file is copied to tbxfilename is adminhomeform | As expected |  |

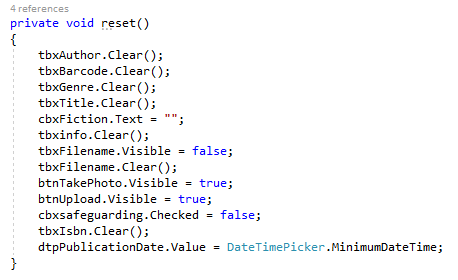
### AdminHomeForm.cs



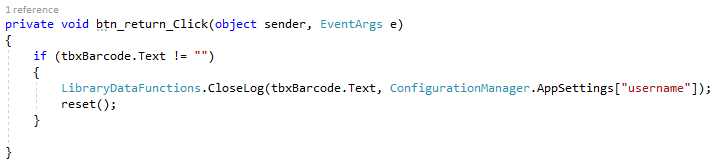
These functions are used to allow the program to return to an open form rather than opening a new one, by setting a variable called adminhomeforminstance holding the form, and using the formclosedeventhandler to set this value back to null when the form is closed.



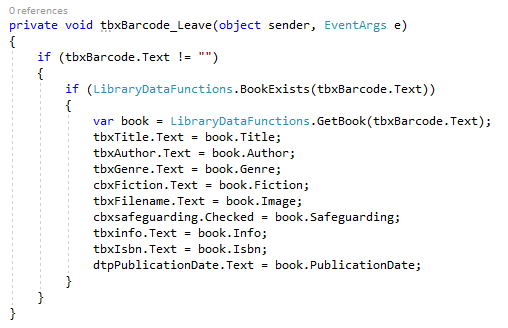
This function is called when the add button is clicked, it checks barcode, title or author are not null, and then calls the addbook function.



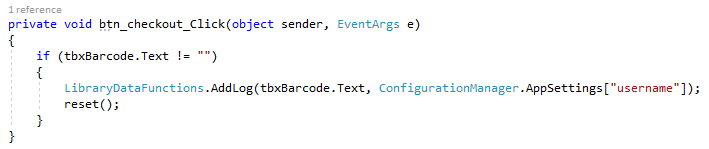
This function resets all data input fields to their null values.



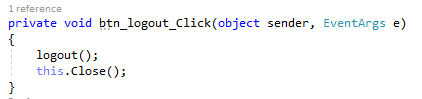
This function is called when the return button is clicked, it checks that barcode is not null and then calls closelog function, referencing the username property in appsettings to get the email parameter.



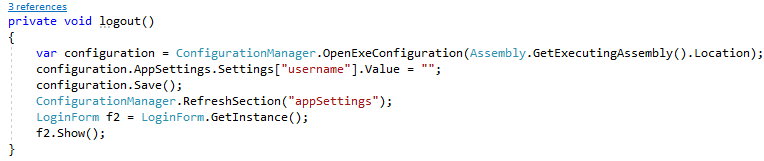
This function is called when the user leaves the barcode textbox, and it fills all the data input areas with the corresponding values associated with the barcode entered.



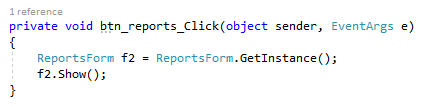
This function is called when the checkout button is clicked, if the barcode is not null, then the addlog function is called, referencing the username property in appsettings.



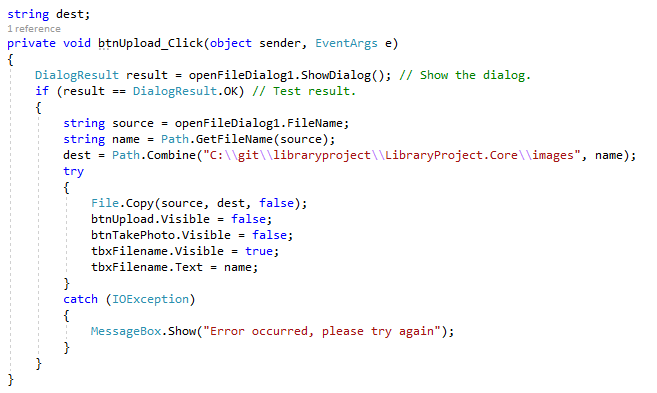
This function is called when the logout button is called, and it calls the logout function, and then closes the form.



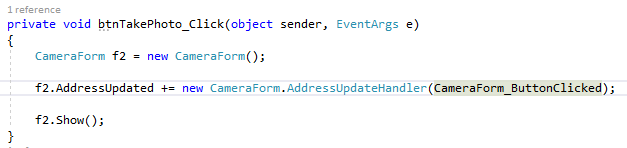
This function logs a user out, by removing their details from appsettings and loading the loginform.



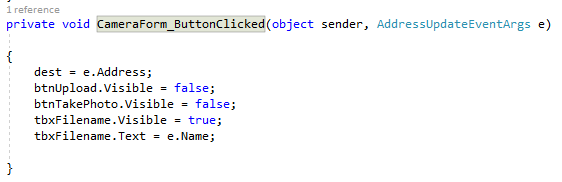
This function is called when the reports button is clicked, and it opens the reports form.



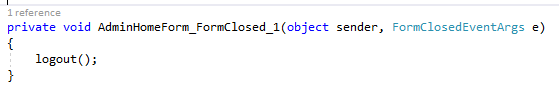
This function is called when the upload button is called, and it opens a file dialogue where the user can choose an image, and gets the location and filename, this file is then copied to the project directory, finally the upload and takephoto buttons are hidden, and the filename textbox shows the textbox. If there is an error in copying the file then a messagebox shows an error.



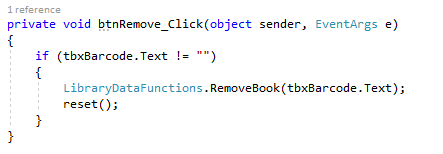
This function is called when the takephoto button is clicked, an instance of cameraform is created, and the addressupdatehandler is created (so data can be passed back to the adminhomeform), finally the form is loaded.



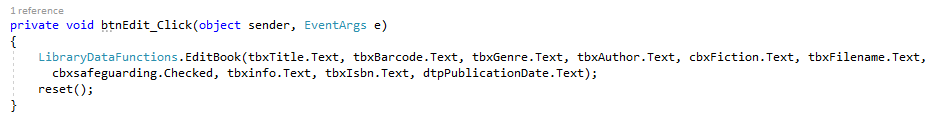
This function is called when the addressupdate event occurs, it sets the destination and filename of the image, as well as making the upload and takephoto buttons hidden and showing the filename in a textbox.



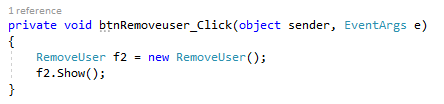
This function is called when adminhomeform is closed, and it calls the logout function.



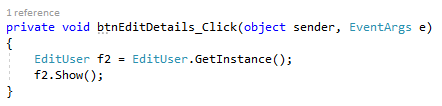
This function is called when the remove button is clicked, it checks if the barcode is null and if not calls the removebook function and the reset function.



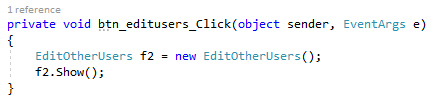
This function is called when the edit button is clicked, it call the edit book and then reset functions.



This function is called when the removeusers button is clicked and it opens the removeuser form.



This function is called when the editdetails button is clicked and it opens the edituser form.



This function is called when the editusers button is clicked, and it opens the editotherusers form.

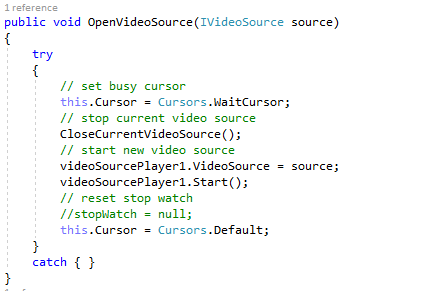
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Purpose | Data | Expected result | Actual result | action |
| Book added to database | This is your brain on music, Daniel levitin, 781, isbn barcode | Book shows in database | **Microsoft.Data.SqlClient.SqlException** (due to model changes not having been applied) | Create migration to sync model and database |
| Book added to database | This is your brain on music, Daniel levitin, 781, isbn barcode | Book shows in database | As expected |  |
| Check out books functionality | This is your brain on music, Daniel levitin, 781, isbn barcode | Log added to database | Error due to returns log entity needing migration | Try to migrate |
| Migration |  | Migration created and database updated | System.InvalidOperationException in returnslog entity | Add fluid api to dbcontext to replace data annotations |
| Check out book | a-level comsci textbook, rsu Heathcote, 000, isbn barcode | Log added to database | **System.NullReferenceException** this.user not assigned a value when useremail is assigned | Added getuser function to access status property |
| Check out book | a-level comsci textbook, rsu Heathcote, 000, isbn barcode | Log added to database | as expected |  |
| Return book | a-level comsci textbook, rsu Heathcote, 000, isbn barcode | Returned updated to true | Nothing happened | Added call to closelog function |
| Return book | a-level comsci textbook, rsu Heathcote, 000, isbn barcode | Returned updated to true | As expected |  |

### CameraForm.cs

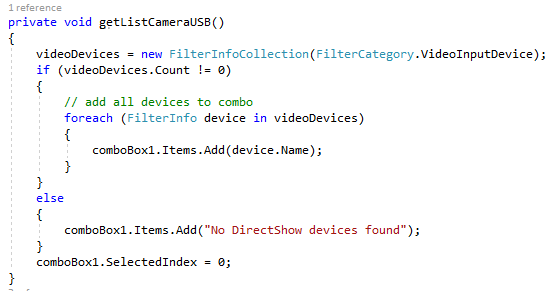
This form is used to take a photo using a webcam connected to the computer, which is then used as the cover image for a book.



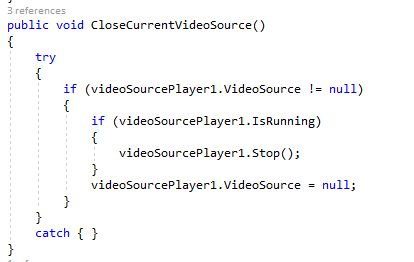
This function is run when the start button is clicked, it searches for suitable camera devices and displays them in a list. Then initialising the one selected, showing an error in a messagebox if any stage is unsuccessful.



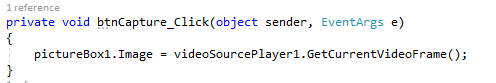
This function is used to start the video feed from the camera.



This function is used to search for available camera devices to add to the comboBox on the form. It is called by the form’s constructor.



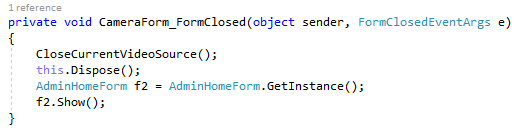
This function is used to close the running video source, first it checks if there is a video source and that it is running, and then it closes it and sets the source to null.



This function is called when the capture button is clicked, it gets the current frame of the video, and sets it to the picture box so it can be reviewed before being saved.



This function is called when the save button is clicked, it stops the video source, sets up the filename and file location, then saves a jpeg of the image from the picturebox, creating the directory if needed. Then the video source is closed, the addressupdateevent is called in order to pass the name and location of the saved jpeg to the adminhomeform, and this form is closed.

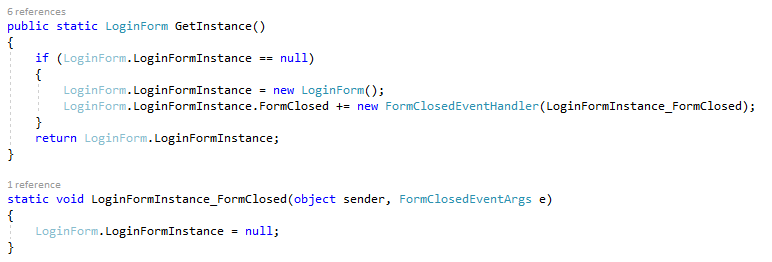


This function is used to close the video source and show the adminhomeform when the cameraform is closed.

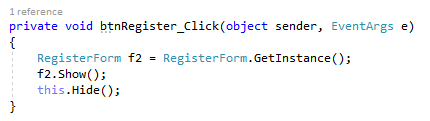
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| purpose | data | Expected result | Actual result | action |
| Camera connect and takes photo |  | Video feed shows on left, and on click of capture button photo taken and displayed on right | Video feed shown, but stretched and distorted. Camera doesn’t capture photo on clicking button | Change size of videosourceplayer to aspect ratio 4:3, linked new frame event and function in videosourceplayer |
| Camera connect and takes photo |  | Video feed shows on left, and on click of capture button photo taken and displayed on right | Capture displays, but not scaled to size of box. | Change sizemode attribute to zoom |
| Camera saves photo from capture |  | Image from capture saved to folder imagecapture | **System.ArgumentException:** 'Parameter is not valid.' | Fix errors in file path string (using escape characters) |
| Camera turns off when save button is clicked and form closes |  |  | As expected |  |

### LoginForm.cs

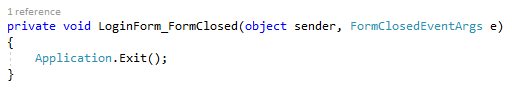
This form is the first thing the user sees on running the program, it allows users to enter their login and access the system, and has an option to register new users.



These functions are used to allow the program to return to an open form rather than opening a new one, by setting a variable called loginforminstance holding the form, and using the formclosedeventhandler to set this value back to null when the form is closed.



This function is called when the register button is clicked, it opens the register form and hides the login form.

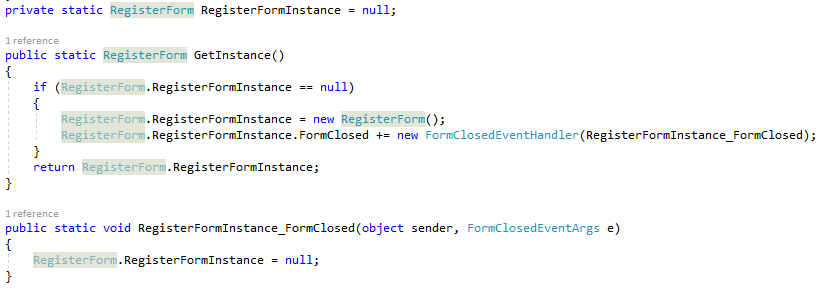


This function is called when the form is closed, and it closes the application.

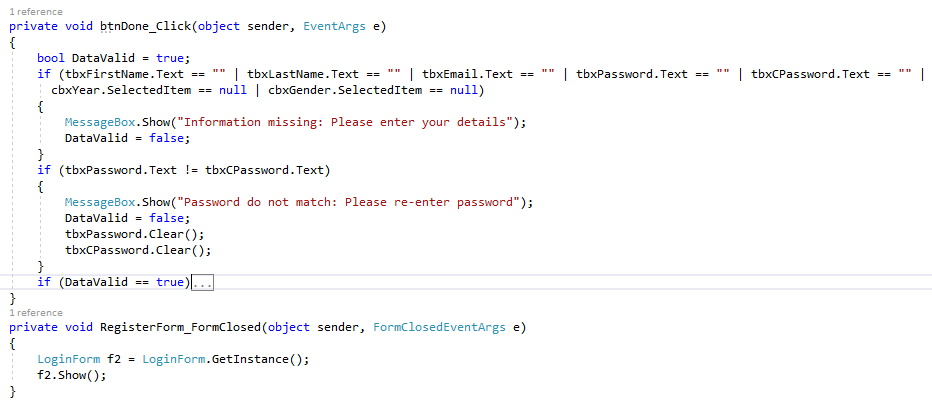
This function is called when the login button is clicked, first it checks if the login is valid and what type of user it is, then if the user is a student or a sixth-former the username is added to appsettings, the textboxes are cleared and the userhomeform is opened. If the user is a member of staff or a library vip, then their username is added to appsettings and textboxes are cleared but they are directed to the adminhomeform. If the login is not valid then an error message is shown, and the textboxes are cleared.

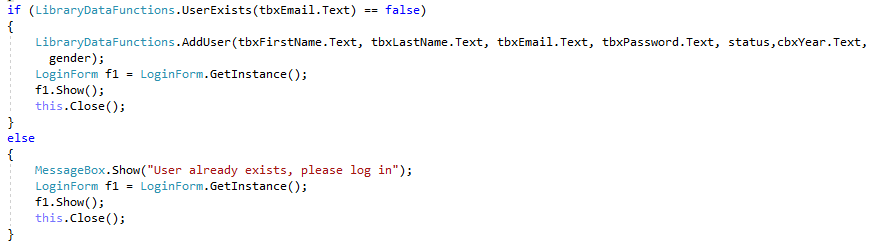
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Purpose | Data | Expected result | Actual result | action |
| Test storing of logged in user’s email in app.config | 07hwheeldon@mybest.org.uk | Email shows in app.config username | System.NullReferenceException | Add setting to app.config for data to be stored in |
| Test storing of logged in user’s email in app.config | 07hwheeldon@mybest.org.uk | Email shows in app.config username | Data not shown (as app.config compiled to bin in order to run) program accepts data from config but later causes **System.NullReferenceException**  when trying to access user | Fix accessing app.config. |
| Test storing of logged in user’s email in app.config | 07hwheeldon@mybest.org.uk | Email shows in app.config username | As expected |  |

### RegisterForm.cs

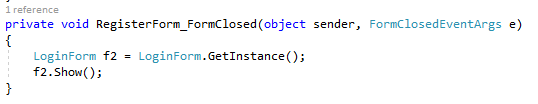


These functions are used to allow the program to return to an open form rather than opening a new one, by setting a variable called registerforminstance holding the form, and using the formclosedeventhandler to set this value back to null when the form is closed.





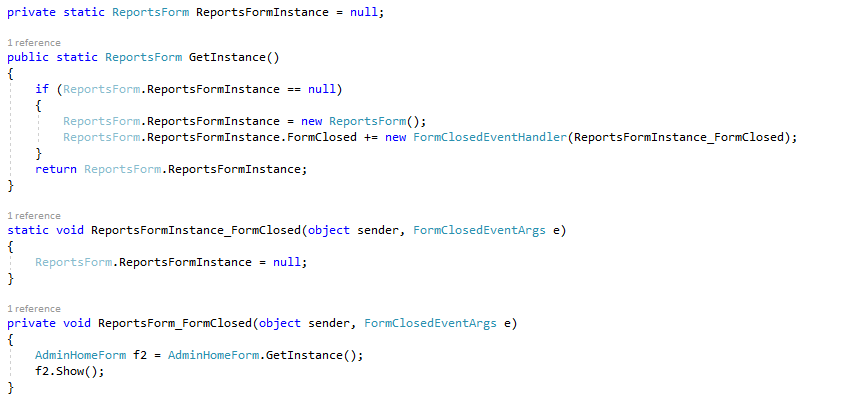
This function is called when the done button is clicked, first it checks if the data entry is valid by ensuring no fields are empty, and that the passwords match. If this is the case, then a series of if statements (not shown in the image) are used to transform the entries from the combination box to the required format for the database (year groups becoming student, sixth form or staff status, and gender displayed as a character. Then an if statement checks this user doesn’t already exist in the table, if so the user is added and the login form is opened. Otherwise an error message is shown to the user and the login form is opened.



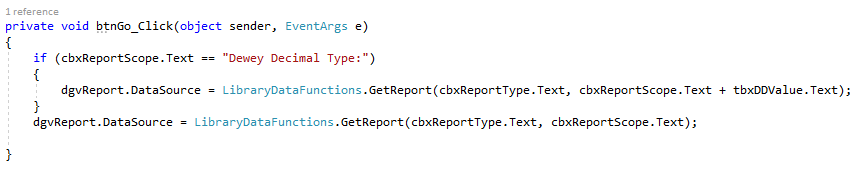
This function is called when the register form is closed, and it opens the login form.

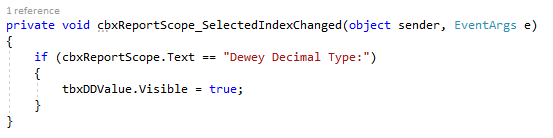
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Purpose | Data | Expected result | Actual result | action |
| Check if missing data is recognised | Holly, Wheeldon, “”, year 9, password, password | Messagebox prompts user to add missing details | As expected |  |
| Check if missing year group (as different form of entry) | Holly, Wheeldon, [myemail@email.com](mailto:myemail@email.com), “”, password, password | As above | As expected |  |
| Check if password validation works | Holly, Wheeldon, [myemail@email.com](mailto:myemail@email.com), year 9, password, 12345 | Messagebox prompts user to reenter passwords | As expected |  |
| User added to database | Holly, Wheeldon, [07hwheeldon@mybest.org.uk](mailto:07hwheeldon@mybest.org.uk), year 12, password | Return to login form, and user shows in database | **System.InvalidOperationException** | Change system.data.entity to Microsoft.entityframeworkcore and edit dbcontext accordingly |
| User added to database | Holly, Wheeldon, [07hwheeldon@mybest.org.uk](mailto:07hwheeldon@mybest.org.uk), year 12, password | Return to login form, and user shows in database | System.ArgumentException | Change usesqlserver() to useInMemorydatabase() |
| User added to database | Holly, Wheeldon, [07hwheeldon@mybest.org.uk](mailto:07hwheeldon@mybest.org.uk), year 12, password | Return to login form, and user shows in database | **System.FormatException** | Fixed data types of parameters, and reorganised constructor data interpretation |
| User added to database | Holly, Wheeldon, [07hwheeldon@mybest.org.uk](mailto:07hwheeldon@mybest.org.uk), year 12, password | Return to login form, and user shows in database | as expected | After adding connection string to config file |

### ReportsForm.cs



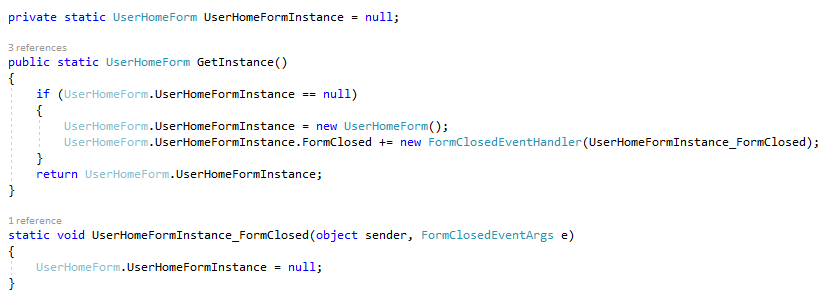
These functions are used to allow the program to return to an open form rather than opening a new one, by setting a variable called reportsforminstance holding the form, and using the formclosedeventhandler to set this value back to null when the form is closed.



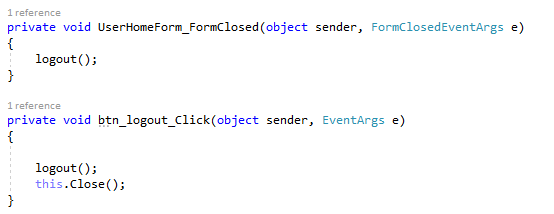
This function is called when the go button is clicked, the datagridview is filled by using different overloads of the get report function, depending on whether the scope is set to dewey decimal type

This function is called when the index of the scope combobox is changed, it uses and if statement to check if the selected option is dewey decimal type, if so setting the dewey decimal textbox to visible.

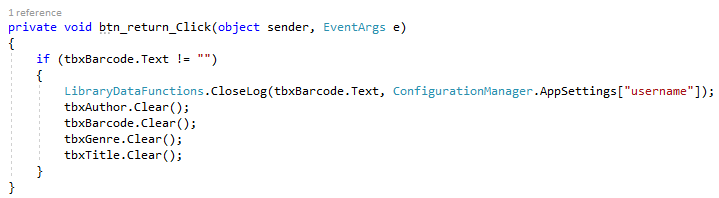
### UserHomeForm.cs



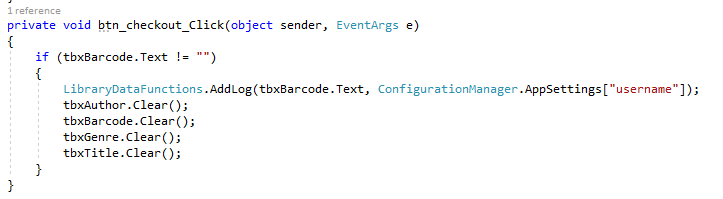
These functions are used to allow the program to return to an open form rather than opening a new one, by setting a variable called userhomeforminstance holding the form, and using the formclosedeventhandler to set this value back to null when the form is closed.



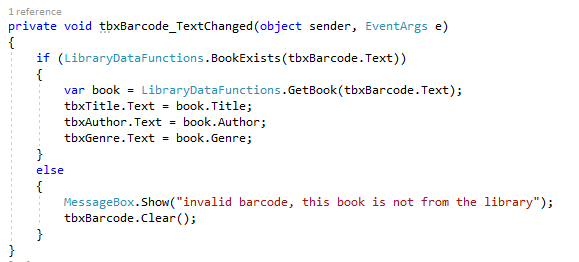
These functions are called when the form is closed, or the logout button is clicked, and they both call logout and close the form if needed.



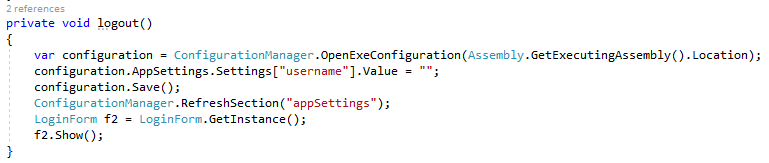
This function is called when the return button is clicked, it checks the barcode is not null and then closes the log associated with that barcode and user, before clearing all the textboxes.



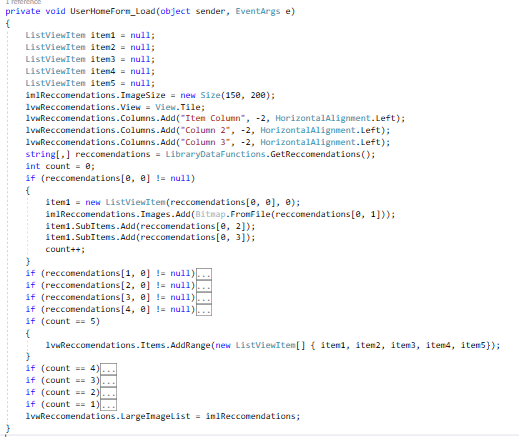
This function is called when the checkout button is clicked, it checks that the barcode is not null and then creates a log for that book and user, before clearing the textboxes.



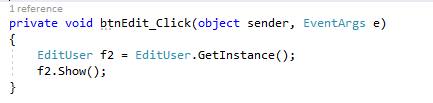
This function is called when the text is changed in the barcode textbox, and it checks if it is associated with a book, and if so the book is fetched and the other boxes are filled with with data, otherwise an error message is shown and the barcode is cleared.



This function logs a user out, by removing their details from appsettings and loading the loginform.



This function is called when the form is loaded, and it creates the recommendations of books the user may want to read. First listviewitems are created, and a title, two columns, and an image. If statements add data to each listitem and increments a count. Then another set of if statements adds the list items to the listview, using overloads depending on the number of list items.

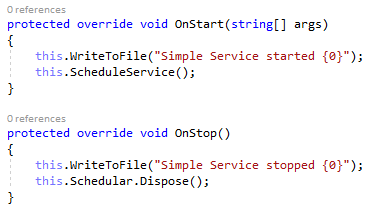


This function is called when the edit button is clicked, and it opens the edituser form.

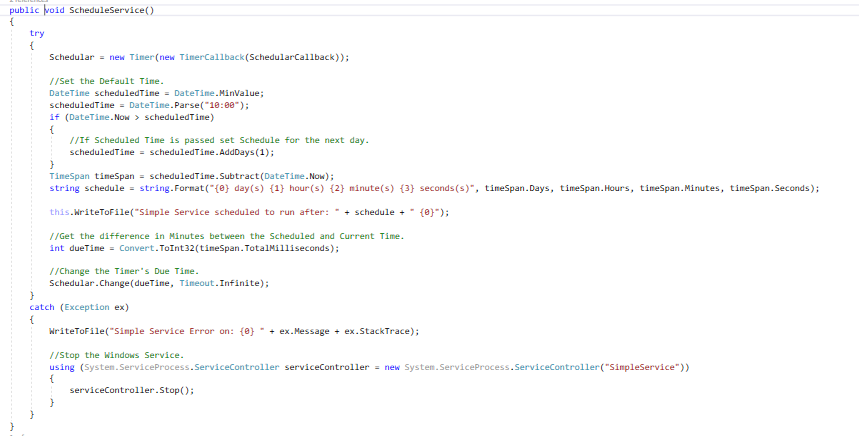
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Purpose | Data | Expected result | Actual result | action |
| Barcode data entry works | This is your brain on music, Daniel levitin, 781, isbn barcode entered via scanner | Data inputted successfully | As expected |  |

### Service1.cs

This is a service which runs each day, in order to complete certain timed functions such as sending emails.



These functions are called when the service starts and stops respectively, it writes to a file documenting this, and then either calls schedule service or disposes of the schedular.

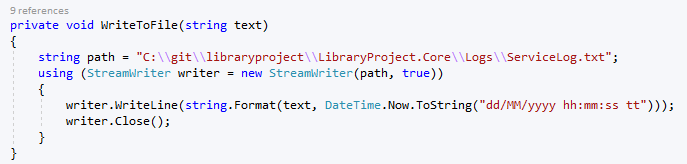


This function is called when the service starts, and it creates a timer called schedular, and checks if the current time is after the scheduled time for it to run next, if so it sets the scheduled time for the next day. Then it writes to the file documenting when it ran and when it is scheduled for, if there is an error an error message is written to the file and the service is stopped.

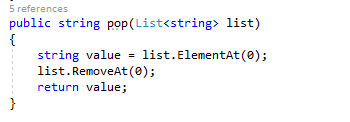




This function is run when the timer finishes, it sends an email to any students with a book overdue using the mimekit library, writing it’s progress to a file. It also sends an email to the safeguarding team showing any users who have taken out a number of books flagged concerning, similarly writing it’s progress to the file.



This function writes a string to a text file in the program directory, using a streamwriter.



This function is used to pop items from the list of users returned by librarydatafunctions.getemailbookdue(), so they can be emailed separately.

## Review with stakeholders

I completed a review after the majority of the basic functionality was complete, where I demonstrated the program, and received feedback on whether it met the needs of the library. Mostly the opinion was very positive, they were especially pleased with the reports area as it was much easier to use than the current system. However, they requested some more attributes be stored about books – including ISBN number, publication date, and an area for extra information to be stored. We also discussed the best way of deleting users effectively, and the challenges that come with students repeating years, and joining and leaving the school at non-standard times.

A final review was also completed at the end of development to aid in the evaluation, however this had to be in the form of a video call where I shared my screen as schools were closed. They were satisfied with all the additional features I had added, particularly the safeguarding feature. During this meeting they also ticked off all the success criteria they felt had been met during the project, and this allowed me to better understand where the needs had and hadn’t been met – this has been indicated in the evaluation.

# Evaluation

## Success criteria

### Graphical user interface, application Description automatically generatedUsername and password to confirm user identity

This was confirmed by the stakeholder.

### Register screen where user can enter their own details

Graphical user interface, application

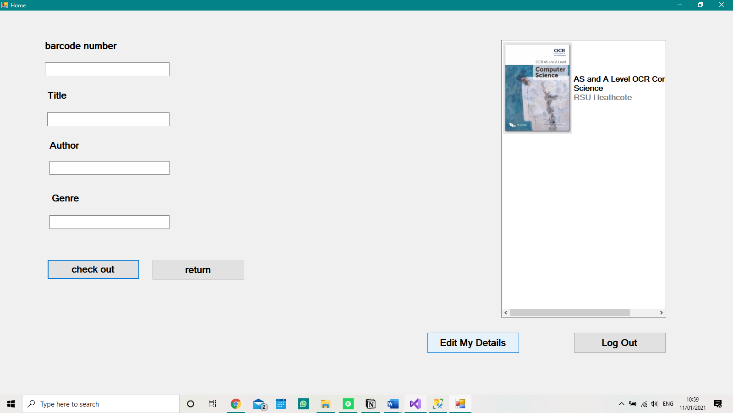
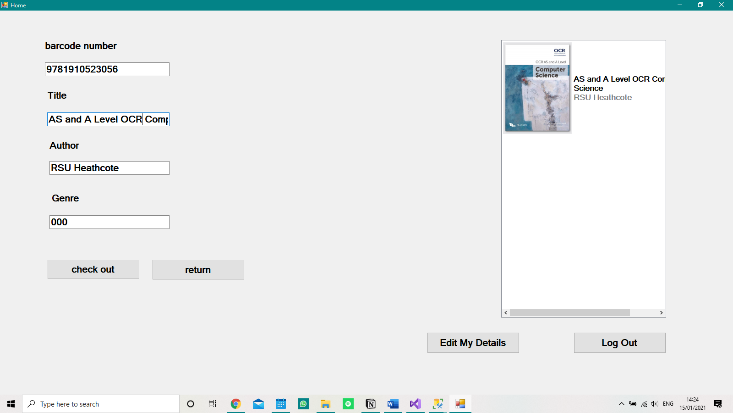
Description automatically generated

This was confirmed by the stakeholder.

### Screen where admins (and only admins) can add books to the library

It can be seen that only the admin form has an “add book” button. This was confirmed by the stakeholder.

### Usable barcode scanner



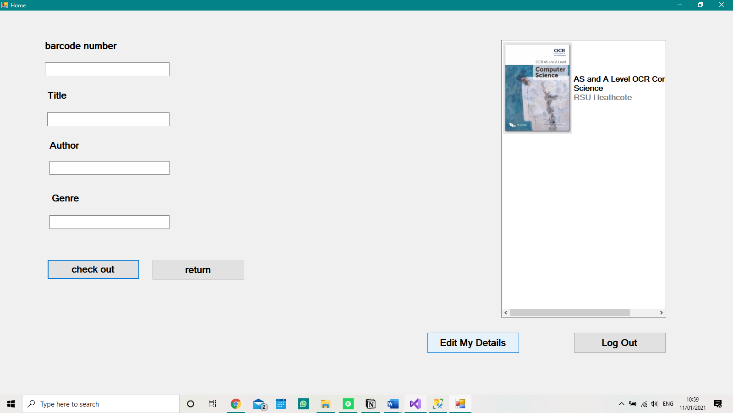
This was confirmed by the stakeholder.

### All areas can be navigated easily with a screen reader

I used the narrator feature on my pc while running the code, and it was easy to navigate and use even without use of a mouse / tracker pad (as the screen reader is designed for people with vision loss they will also not be able to use the mouse).

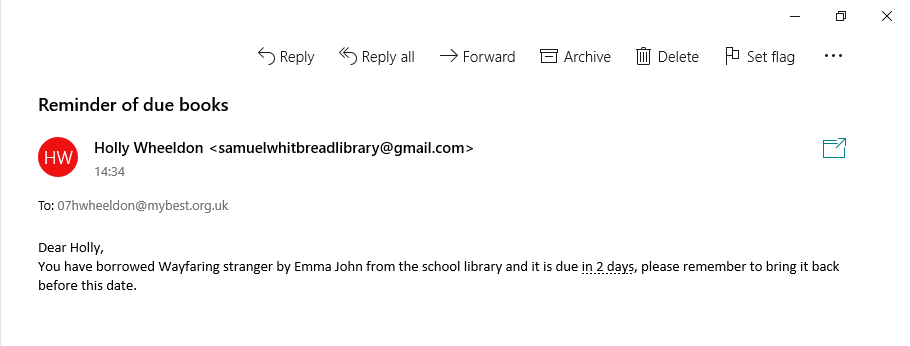
This was not applicable to the stakeholder review, as I didn’t complete a screen reader test in the meeting due to the difficulty I had using it to navigate the computer, as I wasn’t well practiced. I decided it wouldn’t be a worthwhile use of their time in the meeting.

### Screen where students can sign books in and out



This screen has a check out button, and a return button where they can sign books in and out of the library. This was confirmed by the stakeholder.

### Email sent to students 2 days before a book is due



This was not included in the stakeholder meeting, as emails are only sent to a user at 10 am when a book is due.

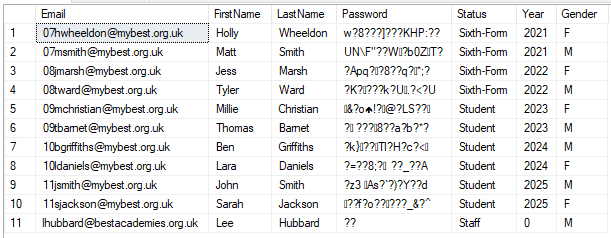
### Screen for creating reports

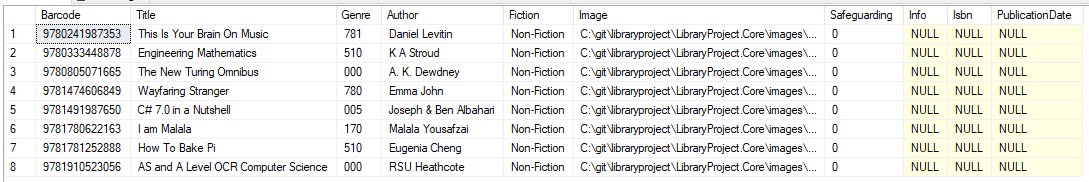
Graphical user interface, text, application

Description automatically generated

This form is used for creating reports, in the data grid view. This was confirmed by the stakeholder.

### All books’, and registered users’ details stored in a database





The last three columns are null, because they were requested to be added during a review with the librarian, therefore the database had already been populated with data before they were added. This wasn’t included in the stakeholder meeting, as the database is a back end part of the program that isn’t seen by users.

### Log of all books taken out stored in a database

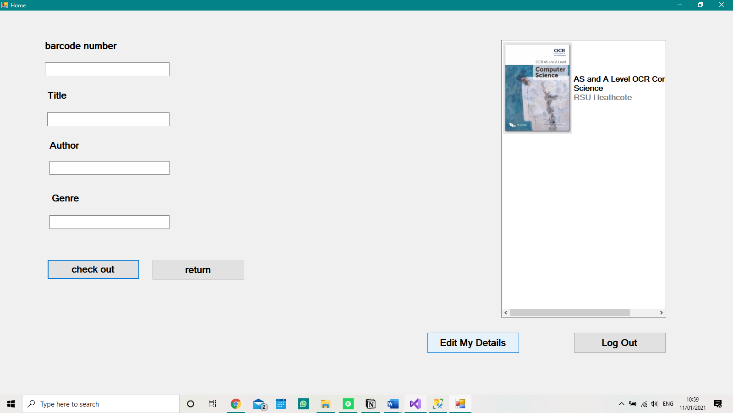


This wasn’t included in the stakeholder meeting, as the database is a back end part of the program that isn’t seen by users.

### Takes less than 1 minute to take out / return a book

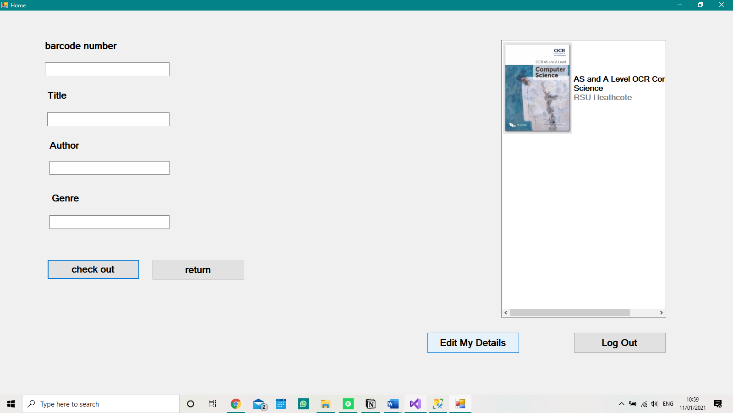
In a timed test it took me 23 seconds to take out a book, so this is well under the time requirement. This wasn’t included in the stakeholder meeting, as the test is qualitative, as well as the fact that a variable internet speed would make any tests completed invalid.

### Students have their own logins, and can use the system unsupervised



This is the student area, accessible when they log in, and which they can use unsupervised. This was confirmed by the stakeholder.

### System recommends books, based on which are popular at the moment

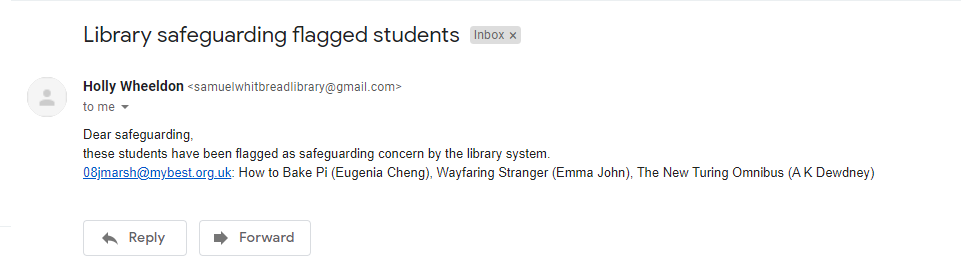


On the right is the recommended books, based on what other students are reading. This was confirmed by the stakeholder.

### Students can search for book recommendations based on genre

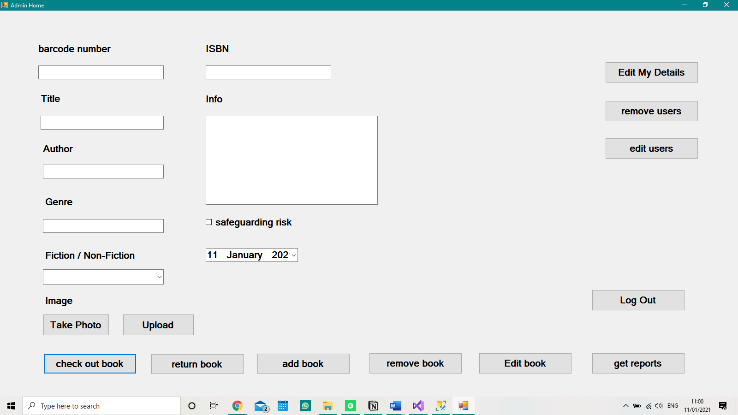
This feature was not included in the program.

### Safeguarding emailed when a number of flagged books are taken out



None of the books in my library were flagged, so in order to create this email I had to flag a number of books and make a user take them out, which is why these books do not seem concerning. This was not included in the stakeholder meeting, as emails are only sent to a user at 10 am when the service runs.

### Students and admins use different forms, to ensure simplicity

these forms are the different user and admin forms, the user one is much simpler. This was confirmed by the stakeholder.

## Usability features

### Large text

All of the forms use bold 15-point font, which makes it very easy to read.

### Screen reader compatibility

Every object on the forms is numbered in a logical order using tab indexing, which allows screen readers to navigate the form easily.

### Clear instructions

All instructions and labels are concise and autological. For example, the button labelled “return” returns a book, the button labelled “edit my details” allows you to edit the information stored about you.

### Separated modes for different users

There is a separate admin and user home form, which have different functions based on the permissions of the user.

## Maintenance and development

Over time management of the database may be required to ensure it continues to perform optimally, especially as more records are added.

It has been ensured that the code is well documented through this document, with clear explanations of the purpose of every subroutine. Due to this, the code can be accessed by another developer easily in the future. This will be especially useful if the stakeholder wants new features added to the system.

The home form could be redesigned to allow actions on multiple books at once to increase the speed of using the system, as was exposed in the alpha testing phase.

The email reminders system could be altered, in order to only send one email at a time to each user by including all relevant books in the same email.

I could also add the recommendations by searching for genre, as this was a success criteria I didn’t meet.

If it is found in the future that a certain area of the system is not used, it can be easily removed due to the modular nature of the program. This same principle can also be applied to adding new features.

# References

Fresh-USA, 2018. *Library Software.* [Online]   
Available at: https://www.youtube.com/watch?v=s2cw80tP-4E  
[Accessed 2020].

MacWill, 2015. *Bibliosoft - Library Management Software.* [Online]   
Available at: https://www.youtube.com/watch?v=IoS4SnusvWk  
[Accessed 2020].

Microsoft, 2019. *Visual Studio 2019 Product Family System Requirements.* [Online]   
Available at: https://docs.microsoft.com/en-us/visualstudio/releases/2019/system-requirements  
[Accessed 2020].

solverteam, 2013. *Library Software Demo.* [Online]   
Available at: https://www.youtube.com/watch?v=bQabhpj4ZaQ  
[Accessed 2020].

W3C, 2018. *Web Content Accessibility Guidelines 2.1.* [Online]   
Available at: https://www.w3.org/TR/WCAG21/  
[Accessed 2020].