CIS 227 Assignment 6

Assignment Details

Begin modifying your program to introduce a GUI\*

This may mean you have to disable some functionality as we learn GUI

GUI Screen

Utilize a layout in your main screen

Implement search

Display search results

Implement Exit Button

New functionality

Implement the creation of a log file that “tracks” what is occurring.

Write to the log file when

You have a successful db operation

A user logs into your application

A user exits the application

GANTT chart

Team Roles

Lead Programmer – Christopher Rodela

UX/UI Programmer – Jacob Wiles

Functional Programmer – Michael Dolan

Program – 70

UX/UI – 35

Function - 35

Documentation – 30

Total Possible Points – 100

**Version 0.3.4**

| REVISION HISTORY | | | |
| --- | --- | --- | --- |
| DATE | VERSION | DESCRIPTION | AUTHOR |
| 2/1/2023 | 0.0.1 | Initial merging of individual code | Jacob Wiles |
| 2/1/2023 | 0.0.2 | Fixing merged code to behave as intended | Jacob Wiles |
| 2/1/2023 | 0.0.3 | Fixing bugs and reformatting the UX and UI | Jacob Wiles |
| 2/1/2023 | 0.1.0 | “Official Release” and Final turn-in for the week | Jacob Wiles |
| 2/15/2023 | 0.2.0 | Added password encryption by implementing the Boost library, Users have ability to exit program, Users can now add books to the books.csv file, users are able to create a list of books to store in their “cart” | Christopher Rodela |
| 2/22/2023 | 0.2.1 | Change over from csv to SQLite database + update database | Jacob Wiles |
| 2/22/2023 | 0.2.2 | Add admin menu | Jacob Wiles |
| 2/22/2023 | 0.2.3 | Fixing back-end queries used by the admin menu | Jacob Wiles |
| 2/23/2023 | 0.2.4 | Adding random values to QUANTITY\_ON\_HAND in the database, Made the shopping list functionality and menu, Added the SHOPPERS Table to the data. | Jacob Wiles |
| 2/24/2023 | 0.2.5 | Various bug fixes from recent versions and adding a warning for the user to the front-end when they choose to export a file. | Jacob Wiles |
| 2/25/2023 | 0.2.6 | Adding a UI prompt for a user adding a new shopper, Adding a “purchase” option to the shopping list menu, Adding basic version of CalcTotalCost() function | Michael Dolan |
| 2/26/2023 | 0.2.7 | Updating database to not accept as many NULL values and adding more database stability | Jacob Wiles |
| 2/28/2023 | 0.2.7.1 | Adding in the front-end of displaying the user’s shopping receipt and letting the user choose how they wish to search for books in the database. | Michael Dolan |
| 3/1/2023 | 0.2.8 | Adding Back-End Database functions and connecting them to the Front-End UI. | Christopher Rodela |
| 3/1/2023 | 0.2.8.1 | Fixing/Adjusting Integrations between the Front-End and Back-End | Jacob Wiles |
| 3/1/2023 | 0.2.8.2 | Various Bugfixes | Jacob Wiles |
| 3/1/2023 | 0.3.0 | “Official Release” and Final turn-in for week 7 | Jacob Wiles |
| 3/14/2023 | 0.3.1 | Made code compatible with QT. Introduced GUI | Jacob Wiles |
| 3/14/2023 | 0.3.2 | Added back-end GUI functionality | Michael Dolan |
| 3/14/2023 | 0.3.3 | Created a log file for db ops, user log ins, user exits | Michael Dolan |
| 3/14/2023 | 0.3.4 | QT GUI implementation and log file complete | Jacob Wiles/Michael Dolan |

# INTRODUCTION

## PURPOSE

Identify and describe scope of product whose technical specifications are being documented and describe desired outcome.

Have the user attempt to log in within either the regular login menu or the admin menu. They do this by entering a username and password and our system checks if the provided username/password pair is found in our list of users, and in the case of the admin login also check if said pair is also listed as an admin. If not, then inform the user that their log in attempt has failed and prompt them if they would like to try again or exit the program.

Once the user successfully logs in, they are prompted with either the main menu or the admin menu depending on which login method they chose along with that menu’s provided options. The options for both the main menu and admin menu include searching the database of books, adding a book to the database (either manually or by importing a .csv file), editing the user’s “book list” or “shopping list,” adding a new user to the database, or changing a user’s password.

## DOCUMENT CONVENTIONS

Describe any naming or structural conventions employed throughout document and how they benefit reader.

Camel case will be used for all variables, functions, and file names, except for any class constructor functions.

UX/UI will be separated from the back-end logic in a separate .cpp file.

Header files will contain function declarations and library includes.

All variables will be declared at the top of the relevant file or function, including lamda functions, and listed in the approximate order that they are used in within their scope.

## REFERENCES

List any referenced document names or links.

<https://cplusplus.com/reference/> - C++’s documentation and reference.

<https://github.com/d99kris/rapidcsv> - Used to help parse large .csv files.

# DESCRIPTION

## FEATURES

List main features with brief description.

Login Menu – Being able to enter in a username and password and entering in a correct pair will gain you access to the rest of the system. Uses MD5 encryption and hashing to protect the passwords.

.CSV File Reader – A library that can read a given .csv file and parse its data for specific information. The library also helps us writing and exporting .csv files in a similar style.

Menu Display – A UI that will display the current menu, take in user input, and return the appropriate information when required.

Database – A database that stores all the book, user, and shopper data.

Querying and Searching the Database – A library that lets us query our database to either get and receive data from it or to send and set data in it. We can search for books either by their ISBN, Title, Author, Year of Publication, Publisher, MSRP, or the Quantity of that book we currently have on hand.

User Lists – A user has 1 or more list of books that they can either pull from the database or “purchase” from the database.

## USER OVERVIEW

Define groups and describe user characteristics.

Bookstore workers, admins, &/or owners using a Windows PC that need to maintain an inventory for a bookstore.

## ASSUMPTIONS / DEPENDENCIES

Detail all assumed factors (not known facts) that could potentially impact technical specifications set forth. Include external factors. (What other things do users have to have in order to operate this application?)

The user is going to be using a Windows PC.

This is a console-based application (for now we assume), so the user will have to have access to the Windows command prompt. This accessibility requirement is referring both in terms of having the appropriate user permissions within the system as well as having and being able to operate a functioning keyboard or a suitable equivalent that can interact with the command prompt.

The user will need to have the RapidCSV, SQLite3, and MD5 libraries installed onto their machine(s).

Not all input validation has been completed for this project yet, so it is assumed that the user will only enter in valid inputs and data from the input prompts and in the files that this project attempts to import.

Although MSRPs, tax, and total cost are all displayed to the user with a precision of two decimal places, the actual total and then recorded “total spent” in the shoppers table is tracked to a percentage of a cent. It is assumed that this level of precision for the shoppers’ total spent is desired to be tracked to such a degree instead of rounding it.

# SYSTEM FEATURES

## 

## SYSTEM FEATURE 1: The Login Menu(s)

|  |  |
| --- | --- |
| **DESCRIPTION AND PRIORITY** | The user must enter a username and password in order to log in and utilize the system |
| **STIMULUS / RESPONSE SEQUENCES** | First the user selects which login menu they wish to log in with, the regular menu or the admin menu. After the user makes their selection there is a prompt for the username and then it will prompt for the password. After checking the username/password pair against the database’s table of users, the user will either be told that the login was successful and send them off to that login’s next menu or they will be informed that they entered an invalid username/password pair. |
| **FUNCTIONAL REQUIREMENTS** | We take the username and password and after passing the password through a MD5 encryption hashing algorithm we check if the given pair is valid using the database users table. If the pair is valid, which means it was found in the database and if we’re logging into the admin menu that the user record is marked as being an admin, then we “log in” the user. Else if it is not valid we display an error. |

## SYSTEM FEATURE 2: Importing, Exporting, and Parsing .CSV Files

|  |  |
| --- | --- |
| **DESCRIPTION AND PRIORITY** | We are going to be both importing and exporting .csv files that contains book data. This will occasionally use the RapidCSV library to help with reading and parsing incoming files. |
| **STIMULUS / RESPONSE SEQUENCES** | For importing .CSV files, the user is asked to ensure that the .CSV file that they want to import is named “import.csv” and is in the correct directory and to his confirm or cancel. If they hit confirm, then the program will attempt to find the file named “import.csv” and to parse through the file for importing.  For exporting, the user is warned that if they already have a file named “exportedBookList.csv” in the folder that we export to then that file will be deleted and overwritten with the new data from their books list. If they confirm after this warning, then the program will do exactly this. |
| **FUNCTIONAL REQUIREMENTS** | For importing, once the user hits “confirm,” then then the program will attempt to find the file named “import.csv” and to parse through the file and adding its data to the BOOKS table in the database.  For Exporting, after the user confirms with the warning, then the program will open/create the export file and name it “exportedBookList.csv” and populate the .CSV file with the data from the user’s book list. It’ll then clear the user’s book list of all its books. |

## SYSTEM FEATURE 3: Menu System

|  |  |
| --- | --- |
| **DESCRIPTION AND PRIORITY** | A menu system that allows for easy and clean navigation of all the program’s functionality. |
| **STIMULUS / RESPONSE SEQUENCES** | For each menu “page,” that menu displays its name and all the options available for that menu, including navigating to other menus. The user can then input the number of the option that they wish to use, and the menu will “run” that option. |
| **FUNCTIONAL REQUIREMENTS** | All the menus are setup as class objects beforehand using lambda functions when the user loads up the program. When a user selects an option to do, that lambda function is triggered. In order to start up a menu, you need to call its “run()” function. You can also call other menu’s “run()” functions within their menu options. |

## SYSTEM FEATURE 4: The Database

|  |  |
| --- | --- |
| **DESCRIPTION AND PRIORITY** | We’ll be using SQLite to make our database to store and track our books, users, and shoppers data. |
| **STIMULUS / RESPONSE SEQUENCES** | Outside of “purchasing” 1 or more books from the database inventory, the only users who will be allowed to edit the database will be those logged in as “Admins.” Admins can add and remove book and user records to/from the database as well as editing user’s passwords. |
| **FUNCTIONAL REQUIREMENTS** | We have a BOOKS, USERS, and SHOPPERS tables with all their columns, except for DESCRIPTION and GENRE, requiring that they cannot be NULL. |

## SYSTEM FEATURE 5: Querying and Searching the Database

|  |  |
| --- | --- |
| **DESCRIPTION AND PRIORITY** | Whenever the user chooses to perform an action that involves querying/searching the database for book data, we will give the user the option of choosing which part of the books’ data they want to search for, namely ISBN, Title, Author, Publication Year, Publisher, MSRP, or the Quantity on Hand. |
| **STIMULUS / RESPONSE SEQUENCES** | For the general-purpose search, we give the user the freedom to choose to search using any of the above criteria. As for when the user is asking to put a single result into one of their lists, they are limited to only searching by ISBN and Title. |
| **FUNCTIONAL REQUIREMENTS** | Once the user selects how they want to search and what they want to search for, the back-end will construct the appropriate query to give to the database to search its records for any matches. It’ll then construct a list of Book objects to hold the data for the user to use later. |

## SYSTEM FEATURE 6: User Book List & User Shopping List

|  |  |
| --- | --- |
| **DESCRIPTION AND PRIORITY** | A user has two lists of books that they interact with, their “book list” and their “shopping list.” Users can add books from the database to these lists to perform various tasks with these lists once they are populated with that book data. |
| **STIMULUS / RESPONSE SEQUENCES** | For either of these lists, the user can search the database for exactly one matching result to add that result’s corresponding data to that list. If the user’s search wasn’t narrow enough and returns 2 or more results, then we tell them to try again with a more specific search time next time. |
| **FUNCTIONAL REQUIREMENTS** | The book list is used to create an exported .CSV file that will maybe in the future be used to remove book records from the database. Meanwhile the shopping list is for the user to “purchase” the listed books thereby lowering those books’ Quantity on Hand within the database. |

## SYSTEM FEATURE 7: QT GUI

|  |  |
| --- | --- |
| **DESCRIPTION AND PRIORITY** | We’ll be implementing a GUI into our project for better user navigation and usability |
| **STIMULUS / RESPONSE SEQUENCES** | Users will now be able to easily navigate the application easier while enhancing user experience. Improves database management by improving UI features for administrators. |
| **FUNCTIONAL REQUIREMENTS** | The application requires user authentication to access the database management functionality and allows different levels of access and permissions based on user role. |

# REQUIREMENTS OF EXTERNAL INTERFACE

## USER INTERFACES

Describe product / user interface characteristics, including standards, style guides, constraints, functionality, and sample screens if applicable.

This program is going to be a console application, which will require suitable user permissions to access and interact with the Windows command prompt as well as a functioning keyboard or a suitable equivalent that can interact with the command prompt. A mouse or a mouse equivalent will not be required for the functional operation of this application but having such equipment will allow the user an additional way to be able to close the program prematurely at their own discretion.

The user will need to have the RapidCSV library installed onto their machine(s).

The user will need to have the SQLite3 library installed onto their machine(s).

The user will need to have the MD5 library installed onto their machine(s).

# APPENDICES

## APPENDIX A: GLOSSARY OF TERMS

Define all terms and unique acronyms employed throughout document and specific to project.

## APPENDIX B: ANALYSIS DOCUMENTATION

List file / document names / provided links to all diagrams, models, additional findings pertinent to technical specification development. (Basically, just about any other documentation that we make for this project should be included here. Also include any files/filenames that would be required to compile & run our program that we’ve written ourselves, so our .cpp’s, our .h’s, and any .json and .csv that we’ve made should also be listed here, but do not include any of the files that were given to us as part of the assignment, i.e. the “books.json” and “books.csv” that are the inventory files that we were given.)

Group4Assignment5DocumentationWeek7.docx (This file)

Week 7 Gantt Chart Turn In Portrait.pdf

Week 7 Gantt Chart Turn In.pdf

Week 7 Gantt chart.xlsx

CPP2Group4Assignment2BookstoreInventory (Folder)

* BackEnd.cpp
* BackEnd.h
* Book.cpp
* Book.h
* bookstoreInventory.db
* CPP2Group4Assignment2BookstoreInventory.cpp
* hash\_password.h
* logfile.txt
* md5.cpp
* md5.h
* Menu.cpp
* Menu.h
* menuItem.cpp
* MenuItem.h
* passwordEncryptor.cpp
* rapidcsv.h
* sqlite3.c
* sqlite3.h
* Utilities.cpp
* Utilities.h

## APPENDIX B: ANALYSIS DOCUMENTATION (continued)

BSI (Folder)

* mainwindow.ui
* mainwindow.h
* mainwindow.cpp
* main.cpp
* hashpasswordencryptor.h
* hashpasswordencryptor.cpp
* dbmanager.h
* dbmanager.cpp
* CMakeList.txt

## APPENDIX C: ISSUES

List all unresolved issues, TBDs, pending decisions, findings required, conflicts, etc. (REMEMBER: If Professor Carmon runs into a bug in our application and he doesn’t see it listed and documented here, it will count as an automatic 0 points for the total “Functionality” portion for this turn in. So TEST TEST TEST!!! Everybody, test your code and communicate any problems you find! And the Lead needs to test ALL the code and ask if anyone knows of any problems with their work.)

| ISSUES | | |
| --- | --- | --- |
| ID | DESCRIPTION | PARTY RESPONSIBLE |
| 1 | Input validation is not up to par with our standards or expectations yet. We’re going to want to centralize our validation for each type of data, ISBN, Year, Cost, Quantity, etc., so that both the front-end and the back-end don’t have to implement their own checks to the data. | Jacob W |
| 2 | Importing files into the database using the Rapidcsv library can lead to larger ISBNs getting converted into scientific notation. Recommended that we replace the use of the Rapidcsv library for importing .csv files with the regular ifstream and doing our parsing manually. | Christopher R |
| 3 | The export book list function doesn't include the new fields MSRP and QuantityOnHand. This might not be needed but we can ask Prof. Carmon about if we'd need it or not. | Michael D |
| 4 | Most of the back-end that deals with the database don't use the safety features of checking if the database opened correctly. See the example I show after this table. … We need to change it so that we are taking these safety steps whenever we use the database. | Jacob W |
| 5 | We are not making use of checking the errors that we get from the database at all. Specifically, the variables "rc" and "&zErrMsg". We need to change how we handle our errors and make sure to check for when any are reported and figure out a way to relay the error messages back to the front-end. | Jacob W |
| 6 | There is currently no input validation done for when we import a file into the database. We need to check that all of the data that we receive from the file is valid before attempting to add it in to the database. | Jacob W |
| 7 | The function that is supposed to remove a book from the database currently does nothing. We probably need to implement that. | Jacob W |
| 8 | Using a vector that requires contiguous memory for our "user book list" can potentially cause speed issues. We should change this from a vector to a different container type that isn't contiguous. | Jacob W |
| 9 | The TOTAL\_SPENT column of the SHOPPERS table has a check but doesn't have the "NOT NULL" constraint. Will need to add that constraint in (which involves renaming the old table, creating a brand-new table using the old table as a guideline while making the required changes, and then inserting all the records from the old table into the new table. Then if you want to delete the old table) | Michael D |
| 10 | In our current implementation, the user is only able to purchase a single copy of each book in their shopping list at a time. The back-end was actually mostly already ready to handle this behavior, but our current front-end doesn’t support any way for the user to make use of it. In the future, we will need to figure out a way to allow the user to purchase more than 1 copy of each book at a time. | Jacob W |
| 11 | The “Add a book to your ‘shopping list’ from the database” menu item currently allows the user to add a book to their shopping list when the QUANTITY of that book in the database is 0. There should be a check to make sure that the QUANTITY is 1 or more before adding that book to the user’s shopping list. | Michael D |
| 12 | Our current “purchaseShoppingList()” function in the back-end handles the increasing of our TOTAL\_SPENT and the decrease of our QUANTITY\_ON\_HAND in two separate sub-functions. This can lead to situations where if the shopping list includes a book that we have a 0 for its QUANTITY, then the shopper still gets docked for the full price of the order but we only update all of the books in the list before the book with the 0 QUANTITY and not updating any of the books that come after. The recommended solution is to combine the two sub-functions increaseTotalSpent() and decreaseBoughtBooks() into the purchaseShoppingList() function and to check that the book is available first, then add the two UPDATE queries to the bigQuery string that will run if there is no errors up to that point at the end of the function. | Michael D |
| 13 | In the Search By section you are currently only able to search a book by title | Jacob W |

Issue ID 4 Example:

Int result = sqlite3\_open(“bookstoreInventory.db”, &db);

If (result == SQLITE\_OK)

{

// Do stuff

}

Else

{

Cout << “Unable to open file” << endl; // Don’t want to use couts in the back-end, but you get the idea.

}

Sqlite3\_close(db);