# Update GUI

## Complete Splash/Login Screen

We were able to make *A* splash screen, but it was just a blank white square with nothing in it that we had to throw together right at the end just to barely meet the requirements.

1. Using the pinned comments and examples in the ui channel of our Discord, finish making the splash screen for our program to have it include the required info (Program Name, icon/logo, program info). If you use the noted artwork, make sure to give the proper credit for it.
2. Make it so that the main window appears on the user’s primary/1st screen and the splash screen appears on the user’s 2nd screen if they have one, otherwise have the splash screen display on their 1st screen. (If you don’t have a way to test with multiple monitors, I do so I can assist if needed.)
3. Set the splash screen to be the appropriate size. (Functional is tasked to actually calculate this, but if you get done with all your other tasks before they have a commit available for this, then go right ahead. See functional’s section on this for more info.)
4. (Optional) Get the splash screen working as a Login screen. Keep in mind that the size of the splash screen is going to depend on the user’s screen size since we’ve been tasked with making it only 15% of the user’s screen.

## Display Hardware Information

The functional programmer is going to get the user’s hardware information using the Windows.h header file. It is the UX/UI programmer’s task to take this info and to display it to the user.

1. (Optional) Our main window’s top menu bar system is currently setup within MainWindow.cpp’s code instead of it being visually present in its .ui file. It would probably be easier to have the menu bar set up in the .ui file instead of in the code, but this step may take up precious time that we may not be able to afford. Regardless either way, a few of the buttons in the current menu are useless and unneeded while other buttons that we need are not included yet, including Import and Export, so editing the menu bar while you can during this task is encouraged.
2. If we don’t have a “Help” drop down in our menu, then create one and add it. Give the Help menu an option saying, “Hardware Info.”
3. Have it so that when the user clicks on “Hardware Info” the program opens up a new window on the user’s 2nd screen if they have one, otherwise just open it on their 1st screen. This new window will display the information gathered by the functional programmer’s code. (Can place dummy info in here until that function is ready.)

## Create an Invalid Entry Message Box

If a user enters an incorrect entry, then we need to use and display a qMessageBox in our application to inform the user as such. The plan is to first try to catch the invalid entry with our own validation methods, then if we don’t catch the invalid entry we will likely get one or more error messages from the database which we’ll use the qMessageBox to tell the user of said database error message.

Here are the list of items that would involve a user’s entry that may be invalid. Note that some of these items may not be implemented yet.

1. Whenever the user creates a new Book to add as a new record into the database.
   1. Within the Book Class, we are going to check and validate that the values inputted into the constructors are actual valid values for each of its elements.
   2. After you create the new Book, the getIsValid() method should return if the Book is indeed valid.
   3. If getIsValid() returns true, then continue the logic as normal and send the Book off to be added to the database. Otherwise, if false then display the qMessageBox.
   4. (Functional will work on implementing these functions, but the \_isValid variable within the Book Class should be available for you to test with.)
2. Displaying any Database Errors that we get.
   1. Now that I am actually looking at our code base right now, this one may need be optional since it would require us to adjust how we set up our database management.
   2. But, if we are somehow able to give the front-end our database error messages, then we should try to display them if we can.
   3. The first idea that came to my mind is to have the dbManager.cpp save the database error message(s) into an external file, probably not the log file since we’re using it for actually logging but it is not out of the question. Then the front-end can read the external file, either all of it or the most recent line depending on how it is done, and use that to display the database error. If that sounds good to you, UX/UI, then go ahead and do it all since I don’t have plans for Functional to assist in that.
3. (Not Yet Implemented) Whenever the user enters in an empty Username and/or Password when attempting to log in.
4. (Not Yet Implemented) Whenever an admin enters in an empty Username and/or Password when attempting to add a New User into the database.
5. (Not Yet Implemented) Whenever an admin enters in an empty Username and/or Password when attempting to Change a User’s Password in the database.

## Making UI For Previous Functionality

We need to make sure that we get ALL our previous functionality from our console-based application up and running by the end of the year! You don’t need to get it all done by this week, but try to work on getting as much of the visual UX/UI set up for ourselves as time allows.

For the most part, you can follow along with what is already created in the Main Window with the main exceptions being Import and Export which should be part of the top Menu Bar.

# Update Functionality

## Get Hardware Info

The functional programmer is going to get the user’s hardware information using the Windows.h header file. It is the UX/UI programmer’s task to take this info and to display it to the user.

1. Using the techniques we learned in class, namely the Windows.h header file, get the hardware information of the user’s machine. (I honestly don’t know what to include here, but I guess things along the lines of CPU speed, number of cores/processors, amount of total & available RAM, amount of total & available hard drive disk space, etc.)
2. Format this information into a container (Vector/QVector, Map, List, etc.) of QStrings for easy access. (My recommendation is a Map for the Key/Value Pairs.) Make sure to write a comment on how you structured your container’s data.

## Book Class Data Validation

We need some basic data validation within the book class for the class to use to check if an inputted class element is a valid value for that class element. For example, a Book’s ISBN must be a string that is not NULL and must have values containing only numbers and the letter X and is 16 characters or less.

1. In the Book Class, make at least 9 total validation functions, one for each of the Book class’s separate elements. These functions can be public functions that take in the appropriate data type and returns a Boolean. Below is an explanation of what each element’s validation function must check for:
   1. ISBN: Use the example above.
   2. Title: The string can’t be a NULL.
   3. Author: The string can’t be a NULL and must be 255 characters or less.
   4. Year: The int can’t be NULL and must be 0 or higher and 2023 or lower.
   5. Publisher: The string can’t be a NULL and must be 255 characters or less.
   6. Description: This string CAN be NULL or a valid string.
   7. Genre: This string CAN be NULL or a valid string with 255 characters or less.
   8. MSRP: This double can’t be NULL and must be 0 or higher.
   9. Quantity: The int can’t be NULL and must be 0 or higher.
2. For each Set function, change it so that they first pass their argument value into the appropriate validation function. If the validation function returns true, then set the value as normal. If false then do nothing.
3. For all of the Book constructors, check if any of the set functions return a false. If so, then set the local result variable to false.

## Calculate Screen Size

We need to set the splash screen to take up no more than 15% of the user’s screen. Your job is to get the hardware information of the user’s screen(s) and then calculate what size the splash screen should be.

1. If a user has 2 or more screens, then calculate using the user’s 2nd screen since we are going to be displaying this on the user’s 2nd screen if possible. Otherwise, if the user has only 1 screen, then use their first screen.
2. So, there is 2 different ways to interpret 15% of the screen. There is the simple way of 15% of the screen’s height and 15% of its width, and there is the complicated way of 15% of the screen's area. I think we are supposed to do the complicated way and to calculate 15% of the area as that is the closer definition of when he says “15% of the screen.” As such, here is a link to how to solve this geometry problem: <https://www.wyzant.com/resources/answers/63901/the_ratio_of_length_to_width_in_a_rectangle_is_2_3_find_the_length_of_the_rectangle_when_the_area_is_150_sq_in>

## Restoring Previous Functionality

We need to make sure that we get ALL our previous functionality from our console-based application up and running by the end of the year! You don’t need to get it all done by this week, but try to work on getting as much of the Functional logic set up for ourselves as time allows. This is my current check list:

* Adjusting the "Quantity On Hand" of a book already in the database
* Make a User's Shopping List and save it in a variable (can save it to an external file instead if desired)
* Add a book to the Shopping List from the database.
* Display your Shopping List to the screen.
* Purchase Items in Shopping List
  + Make sure to check the Quantity in the database to ensure that we don't accidentally sell what we don't have.
* Add a new user to the database.
* Change a User's Password
* Import a .CSV file into the database.
* Make a Book List and save it in a variable.
* Add a book to your Book List from the inventory.
  + When a book is added to the Book List from the inventory, remove that record from the database.
  + If the user closes the program while the Book List is populated, then add those books back into the database.
* Print your Book List to the screen.
* Export your Book List to a .CSV file and empty the Book List

# Team Lead Responsibilities

## Gantt Chart Creation & Updates

Make the Gantt Chart and this document explaining each of the tasks in detail.

## Clean-Up Repo and Code Base

Our project repo has been having issues with files that were meant to be deleted not staying deleted and popping back up again so another cleanup of the repo is in order. A clean up to our code base would not be a bad idea either.

## Documentation

Update the project’s documentation from the previous turn-in(s) to reflect the new progress made in the previous week.

## Getting Back Our Past Features

We need to make sure that we get ALL our previous features from our console-based application up and running by the end of the year! Along with the UX/UI Programmer and the Functional Programmer, work on getting these features up and running when you can. Also make sure that either no one is working on the same feature at the same time, or if they are that they are separating the duties for that feature.