Personal Finance System

Dawson Dauphinais

CS480 – Northern Michigan University

11/28/2021

Introduction

Throughout my time in College, I have learned many things- I might even like to consider myself well-rounded! But, if there is one thing that I have learned, it is that I am *terrible* at keeping track of my finances. Of course, over the years I have tried to get better at it and have tried several different financial applications, but, none of them really clicked with me. So, when it came time to think of a senior project idea, I finally decided that this would be the (nearly) perfect project for me.

My original idea was to create a Windows-based application that would allow the user to keep track of their finances, as well as provide several different tools for them. I wanted to use a PostgreSQL database to store all of the user's data, and then figure out a way for the application to interact with Microsoft Excel. This project definitely evolved as time went on, and I eventually moved away from both the idea to use a PostgreSQL database, as well as the idea of having the application work with Microsoft Excel.

My application was originally going to be written in C++ (it still is), have a graphical user interface for interacting with the application, and then use some sort of PostgreSQL library for connecting to a database, as well as a library for communicating with Excel. After doing a fair bit of research, I made a few course-altering decisions, which I think worked out for the best in the long run. First, I decided that since there was only ever going to be one user using the application at a time, a PostgreSQL database would be a bit overkill. Instead, I decided that I could store the user's data as a JSON file, where the main JSON object is an array of user-created accounts. Second, I decided that I really didn't need excel for my project, because I could

implement most of the features that I wanted to use from Excel in my GUI just by writing it myself. Lastly, I finally decided on a GUI library for my project—wxWidgets.

Overview

This application is meant to be ran like any other application you might have on your machine. You start the program, an application window appears, and then presents you with options at each different "panel." First, the user is prompted to choose between using an existing save file or creating a new one. This file is where the JSON array of accounts is stored. If the user chooses to use an existing file, a file-picker window opens and allows the user to select a file anywhere on their machine, as long as it is a JSON file (for simplicity and the sake of time, I did not implement a way to ensure that the JSON file contains the correct information—at the time of writing this, I assume the program would just hard-crash. The user is then presented with a list of the different names of accounts that were found within the file, which they may choose to use. If the user opts to create a new save instead, the window changes to a new page where the user is prompted to enter several different fields (name of the account, balance on the account, file name), as well as select a directory to save the JSON file in (this opens a similar window as choosing an existing file). After the user has either loaded a data file or created a new one, they are brought to the same page in the next step in the flow of execution. The arrive at a simple homepage, where they are presented with several options (View Account, Update Account, Create Account, Switch Accounts, Tools, and Exit). These options are buttons that the user may click to interact with the application.

Technology Overview

As mentioned previously, this application was written in C++. However, many technologies were used, and almost of all of them had quite a significant learning curve.

Since this is a C++ project at its core, I needed to figure out a way to allow the application to be built easily without having to write my own makefile. I came across this technology called CMake, which, from my understanding, is similar to Make. However, CMake allows you to configure a file called CMakeLists.txt, which essentially is where CMake looks for the specified configurations of your project. Here, you can link libraries, set different compiler flags, and many other things that I didn't get to test out. For my project, specifically, I used CMake to link the two external libraries that I used (more on that later) to my project, as well as define all of the files needed. When CMake is called with a specific generator, in my case I used the MinGW Makefiles generator, it generates a Makefile based solely on the CMakeLists.txt file, which can then be built.

This is not the only use I had for CMake, though. Since I was using two external libraries, which I did not have any experience with at all (I had never used external libraries before this), I had to compile and install these libraries on my machine so that they could be used by my project. So, CMake was used to compile and install the wxWidgets library that I used for the GUI, but I discovered a useful feature when going to build the other library that I used for JSON—you can directly specify a GitHub repo to be cloned at built at compile time using CMake! So, the JSON library that I used is pulled directly from GitHub, and I never had to manually compile it.

Going into this project, I very vaguely knew what JSON was. I had worked with it a tiny bit during my internship, but never directly. So, my knowledge of it at the beginning of this was rather limited—I knew that it was the JavaScript Object Notation, and that it was used very commonly to store what are essentially key-value pairs. So, I had to do a fair amount of research on how to effectively store data as JSON. Now, C++ doesn't have a built-in JSON library, so as I mentioned earlier, I had to find one myself. The library that I used for JSON is the <nlohmann/json> library that can be found on GitHub. So, not only did I have to learn JSON, but I had to learn how to effectively use this library within my project for storing and retrieving data. This included building a JSON object from objects that I had created in my project. For example, in my project I create an Account object that holds the user's account information. So, when the program is started (assuming they use existing data), I have to parse the JSON file using the <nlohmann/json> library and construct my Account object from only that data. The same applies in reverse if I want to save data—I convert my entire Account object into a JSON object, and then write it out to the file. You can find the proper citations for this library in my source code.

For the GUI, I was torn between using Qt and wxWidgets. It seemed to me that both were rather popular, and both had their pros and cons. After a lot of tinkering and experimenting with both Qt and wxWidgets, I finally decided to use wxWidgets. This is because after all of the experimenting I did with Qt, I could not get it to cooperate with the <nlohmann/json> library, whereas I could get wxWidgets to. So, I did end up using wxWidgets for my project. At first, I found this library *incredibly overwhelming*. I did eventually come to figure out the very basics of

it, which I'm proud of even doing just that. I would say that overall, the GUI and wxWidgets was the most difficult part of my project, as it was the furthest I had to go out of my comfort zone. I don't know how I did it, but after reading through an incredible amount of documentation, I was able to get something working, and for that I am very proud of myself. Learning how to go from a "while(run==true)" loop to an event-based application was also rather challenging, but it definitely was not as difficult as just learning the different functions and classes belonging to wxWidgets.

Another technology that I used, although wasn't very necessary, was Doxygen. Doxygen parses a given source directory looking for any JavaDoc comments. It then generates documentation for all that it has found as HTML files. While this wasn't necessary for my project, and wasn't very difficult to use, I think using it made my project both look and feel much more professional. You can find the documentation for all of my code inside the "docs" folder. The index.html file will bring you to the home page of my documentation.

Organization

From the beginning, I knew that I was going to have to stay as organized as possible in all aspects, as this was undoubtedly going to be the largest project that I have ever created. So, what I have done is I have organized the source code into two different categories. The main source code, which contains the classes that I designed for the application, and the app source code, which contains the classes that are for the main application—these classes specifically pertain to wxWidgets.

The actual application is organized a little differently. I organized it in a way so that there are different panels shown according to which functionality is being used at the moment. For example, the main page is a panel, the account home page is a panel, the view account page is a panel, etc. This isn't the case for the entire application, as creating a panel for everything wasn't necessary, and honestly probably isn't the correct way of doing it anyways. The different functions of the application are organized based on the request of the user. For example, the financial equations are all located within the Tools panel, which can be accessed from the account home panel. See the graphic below to see a bit more into the organization.

Difficulties and Improvements

For the majority of my project, I would say that the difficulty was about what I expected—just slightly more difficult than I was comfortable with (intentionally!) As I mentioned earlier, the hardest part for me was the GUI by far. It was the most I had to research, experiment with, and it definitely required the most amount of time. I have never created an actual GUI before this project, and had only brief exposure to JavaFX (did not take Java at NMU). It was harder than I expected to convert a command-line application into a GUI application—however, I am glad that I wrote the entirety of my application as a command-line app first, as it allowed me to get a better feel for my application, and was very useful as a reference.

The JSON library wasn't incredibly difficult to work with, but it did require some experimentation, as the documentation that I found wasn't quite what I needed. There were several examples, however, so I really just had to figure it out on my own. I would say that this

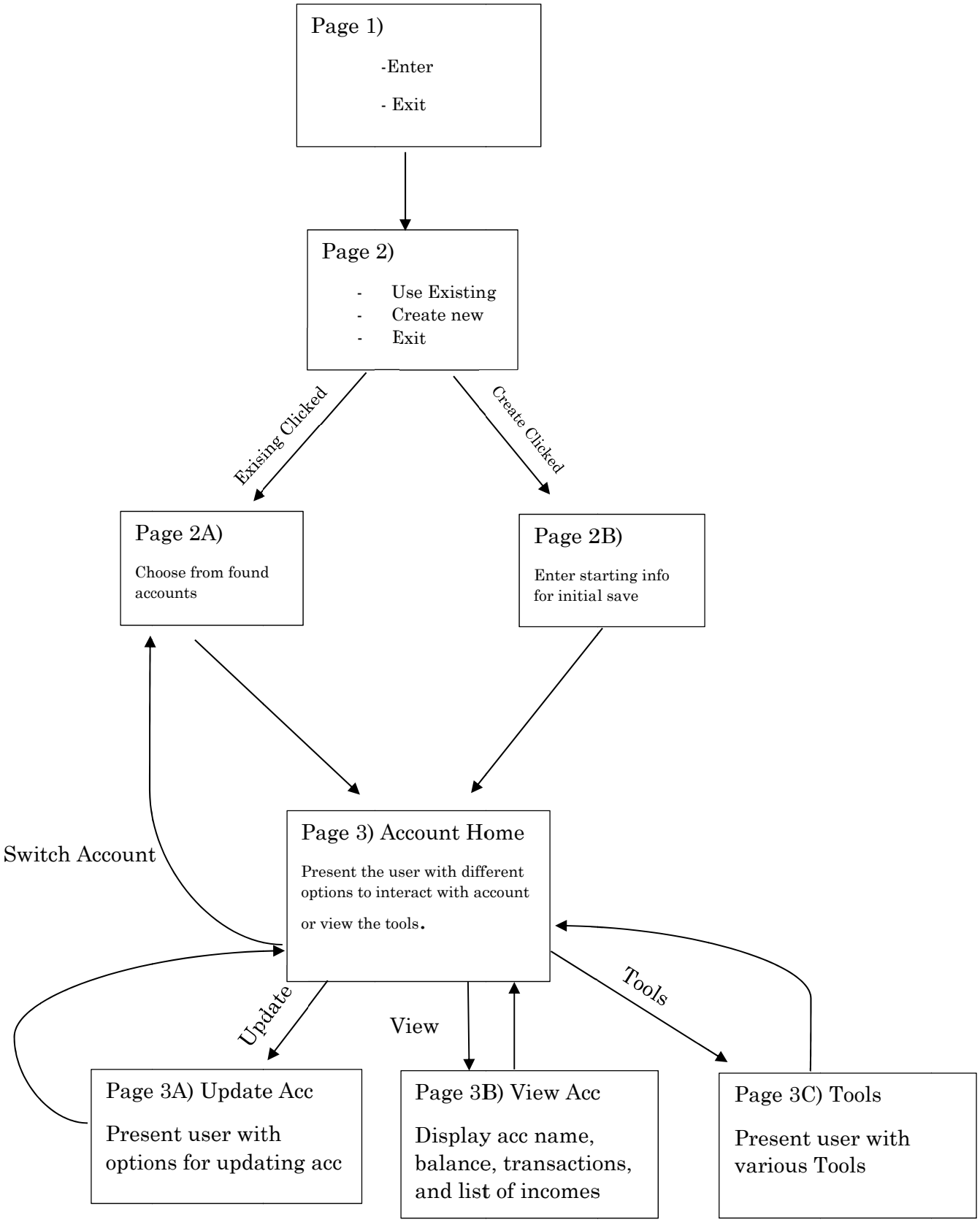
was about as difficult as I expected, maybe slightly harder, but overall, it wasn't horrible to work with by any means.

There are several improvements that I would like to have made if I had more time, but this project came right down to the wire as it was. First of all, I would have liked to make the GUI look much better… At the time of writing this, I basically just have an almost barebones GUI with simple buttons and text entries. I'm not artistic at all, though, so this would probably have taken me another 6-8 weeks. I would have also implemented more features than I already have—maybe a calculator that told me how to become a millionaire based on my most recent paycheck. It would also have been really neat to have some graphs that show spending vs. time and income vs. time. I will say that I am happy with the charts that allow the user to see a list of both their expenses and their incomes, but it would have been cool to give some more visual representations of their account(s). Also, rather than using wxPanels I would probably refactor my code to use wxSizers, which would probably decrease the number of variables used within my MainWindow class.

Conclusion

In conclusion, I would say that this project has been one of the most mentally demanding projects that I have ever done. It required me to truly test my ability to adapt as a programmer and learn new technologies on the fly. As a whole, I think that I accomplished what I set out to do, and although there are a few things that I think I could have done better, I know that I did my best with the time I had. Like I mentioned earlier, this project came right down to the wire—as I

spent most of my Thanksgiving break working on the GUI. I have a great sense of pride in this project, as there were times I didn't think I could do it—but I did! I can honestly say that I feel like a better programmer after building this project. I did accomplish almost everything that I wanted to do, but I think that I definitely could have managed my time better.


Page 1)
-Enter
- Exit
Page 2)
- Use Existing
- Create new
- Exit
Exising Clicked
Create Clicked
Page 2A)
Choose from found accounts
Page 2B)
Enter starting info for initial save
Switch Account
Page 3) Account Home
Present the user with different options to interact with account or view the tools.
Update
View
Tools
Page 3A) Update Acc
Present user with options for updating acc
Page 3B) View Acc
Display acc name, balance, transactions, and list of incomes
Page 3C) Tools
Present user with various Tools