**Keeper**

A Bookkeeping Application

Western Governor’s University – Software Development Capstone

Written By

Austin K. Wise

Java Developer

**Table of Contents**

**A - Keeper Application Summary**

A.1 Summary……..………….………………………………..…………………………..…………………………………………………...3

A.1.1 Business problem……………………….………………………………………………………………………..……..3

A.1.2 Customer Description………………..……..…………………………………………………………………………3

A.2 Application Gaps……………………………………………………………………..…………………………..………………………3

A.3 Software Development Methodology…………………………..…………………………..………………………………...3

A.4 Deliverables…………………………..…………………………..…………………………..……………………………………………3

A.5 Implementation and Outcomes……………………………………..…………..…………………………..…………………..4

A.6 Validation and Verification of Requirements….…………………………..……………………………………………….4

A.7 Required Resources…………………………..…………………………..…………………………..………………………………..

A.8 Timeline and Milestones……..…………………………..…………………………..…………………………….…………………

**B – Keeper Application Development**

B.1 Application Type…………………………..…………………………..…………………………..……………………………..……

B.2 Code requirements…………………………..…………………………..…………………………..……………..……………….

B.2.1 Inheritance.…………………………..…………………………..……………………………………….………………

B.2.2 Polymorphism…………………………..…………………………..………………………………….……………….

B.2.3 Encapsulation…………………………..…………………………..…………………………..……………………….

B.3 Search Functionality…………………………..…………………………..…………………………..……………………………..

B.4 Database Component…………………………..…………………………..……………………………………..………………..

B.5 Report Generation…………………………..…………………………..…………………………..……………………………….

B.6 Exception Controls…………………………..…………………………..…………………………..……………………………….

B.7 Validation Functionality…………………………..…………………………..…………………………..………………………..

B.8 Security Features…………………………..…………………………..…………………………..………………………………….

B.9 Scalable Elements…………………………..…………………………..…………………………..…………………………………

B.10 Graphical User Interface…………………………..…………………………..…………………………..……………………..

**C – Keeper Application Documentation**

C.1 Design Document…………………………..…………………………..…………………………..…………………………………

C.1.1 Class Design…………………………..…………………………..…………………………..………….………………

C.1.2 UI Design…………………………..…………………………..…………………………..………………………………

C.2 Unit Test Plan…………………………..…………………………..…………………………..……………………………………….

C.3 Unit Test Results…………………………..…………………………..…………………………..…………………………………..

C.4 Source Code and Executable File…………………………..…………………………..……………………….………………

C.5 Web API Link (not applicable) …………………………..…………………………..…………………………..………………

C.6 User Guide…………………………..…………………………..…………………………..…………………………………………..

C.6.1 Installation…………………………..…………………………..…………………………..…………..……………….

C.6.2 Maintenance…………………………..…………………………..…………………………………….……………….

C.6.3 User Guide…………………………..…………………………..……………………………………………………….

**Keeper Application Summary**

**A.1: Summary**

Across every entity type whether it be business, governmental, or even personal, one of the most important accounting functions is keeping accurate records financial transactions. The implementation of Keeper, the bookkeeping application, is a fresh take on transaction recording.

**A.1.1: Business Problem**

The record keeping of accounts and accounting transactions is an issue that pervades all accounting departments big or small. In today’s fast-paced world of economic globalization, proper record keeping, and accounting is something most business’ (especially smaller) erroneously shift downwards on their lists of priorities. This can result in catastrophic consequences when checks start bouncing and credit cards are declined due to insufficient funds available.

**A.1.2: Customer Description**

Due to timeline and resource limitation, our customer will likely be using Keeper for smaller business’, sole proprietorships, and personal use. Each of these use cases will most likely not involve complicated financial transactions, which due to our restrictions we will be unable to implement. The needs of our customers will likely revolve around a more general overlook of their current financial standing, according to their financial transactions. To fulfill this need, we will provide the two most basic, foundational reports in accounting, a balance sheet and an income statement.

**A.2: Application Gaps**

Because the application in development is not a rework of an existing application, there are no existing gaps that I will be replacing or modifying.

**A.3: Software Development Methodology**

Throughout the development of this application, I will be using the Agile Development Life Cycle Methodology to guide my activities and milestones. The reason behind choosing this methodology reflects more of a personal preference than a specific justification. In past projects, I’ve found that an excessive amount of planning will result in an overextension and overestimation of one’s resources and abilities. In contrast, the goal of Agile Development is to get to the actual creation of deliverables, which results a more realistic idea of what deliverables can actually be produced to satisfactory quality.

**A.4: Deliverables**

Due to the nature of the Agile Development Life Cycle Methodology, I question the possible existence of Agile deliverables. The definition of a deliverable is a specific measurable output created as a result of the work put into a project. By this definition in an Agile environment, the deliverable of this project is the application itself. This is due to the fact that Agile tends to leave more room for on the spot creative thinking than more traditional methodologies. Considering this, it becomes increasingly difficult to plan for a specific measurable output that constitutes as a deliverable.

However, because of the small nature of this project, we have the ability to look forward and define the deliverables as follows. We will go into more detail about each individual deliverable in the next section.

* Non-Functioning Graphical User Interface
* Login System
* Profile Page
* Chart of account System
* Transaction System
* Report Generation

**A.5: Implementation and Outcomes**

The first deliverable will be a non-functioning graphical user interface. The GUI will be made up of several screens, all of which shall have no real usable functionality except to get an idea of the general flow and experience of the application. Through personal experience, I’ve found this expedites the actual programming of the application because the developer begins to think more deeply about the functionality needed to be built out within each screen. This provides the developer with a clearer picture of what can and can’t be done within the parameters of the project.

As we move on to developing the real functionality of the application, our deliverables are set out in a very specific order. The first real piece of functionality will be a simple login system. A user will be able to create an account and assign various personal information to that account with which they will be able to login to the main interface of the application. While they will be able to view the various screens already created, functionality within the main interface will still be minimal if any at all.

Our next deliverable will be a Profile page. Found at the upper left-hand corner of the main user interface, a user will be able to click on the profile button and be taken to the profile page screen. On this screen the user will be able to view, update and save the information they inputted when signing up for the application.

Finally, we will begin production of the true functionality of the application. To begin, the user will need to have the ability to create a Chart of Accounts. Each account will have the distinction of being an Asset, Liability, Income, Expense or Equity account. Other attributes will also be available to assign to each account type, such as an identification number, a name, a description, and if the user would like to archive the account. On the main Chart of Accounts page the user will have the ability to view their created accounts on separate table according to the type of account they are.

After the Chart of Account functionality is created, we can then move onto Transaction recording functionality. The reason the Chart of Accounts needs to be developed first is because each transaction will be assigned accounts that were created on the Chart of Accounts page according to the nature of the transaction. Transactions will also have other attributes assigned to them such as date, amount and a description. These user created transactions will populate into a main transaction where the user can view all their recorded transaction and edit them as needed.

Finally, the last bit of functionality will be the Report generation feature. Once transactions have been recorded, reports can be generated based on the transactions. The two fundamental reports of accounting are the income statement and the balance sheet. In the income statement a user will be able to see a summed view of their income and expenses based on specific dates given by the user. The balance sheet will be a general overview of their assets, liabilities and equity transactions based on a specific date inputted by the user.

**A.6: Validation and Verification of Requirements**

To verify that the developed application meets the requirements and needs of our customers, we will need to create dummy data that will be inputted into the application.

First, we will create a user through the sign-up portal with which we will be testing all of the functionality with. Once this user is created, we should be able to sign into the application with the username or email and password given. When signed in we should be able to navigate to the profile page, where we must verify that all the data that populates into the user fields matches up with the original data we inputted.

With this user we will be checking the actual functionality of the application. We will add, edit and delete accounts and transactions. We also generate reports based on the dates given making sure that the transactions given are properly generated into the reports. As long as the reports are properly generated, that will be our main test of whether the functionality of the application is correct or not.

As a final check that all data is being properly inputted and outputted by the correct user, we will create another user through the sign-up portal. With this user, we will add various accounts and transactions. If the functionality is working correctly, only data inputted by the current user should show up in all tables and reports. When we sign out and sign in with the original user we created, there should no trace of our other user that we just created in any tables or reports.

**A.7: Required Resources**

The programming environment this application will be developed in is the IntelliJ IDEA CE on an early 2015 MacBook Air running MacOS Mojave version 10.14.3. The language used will be Java version 8 and it will be a JavaFX desktop application. For the database we will use a MySQL relational database which will be remotely stored on RemoteMySQL.com. The entire application will be developed by myself with the possible help of C868 course instructors. All of the above resources have no associated costs to them.

**A.8: Timeline and Milestones**

Overall, the planning, development and testing of this application should not take longer than a month in total. I will be spending somewhere between 25-35 hours a week on this application which should be more than enough time to hit every milestone. My start date will be May 14th, 2019 with estimated submission date of June 14th, 2019.

Below I have inserted a table of what development milestones we will be hitting and their associated durations, dependencies and resources will be.

|  |  |  |  |
| --- | --- | --- | --- |
| Milestones | Milestone Duration | Dependencies | Resources |
| 1.Graphical User Interface | 2-3 days | None | JavaFX |
| 2. Login System | 1-2 days | Milestone 1 | Java, MySQL |
| 3. Profile Page | 2-3 days | Milestone 2 | Java, MySQL |
| 4. Chart of Accounts | 3-5 days | Milestone 2 | Java, MySQL |
| 5. Transactions | 3-5 days | Milestone 2 and 4 | Java, MySQL |
| 6. Report Generation | 3-5 days | Milestone 4 and 5 | Java, MySQL |
| 7. Documentation | 5-7 days | Completion of the application | Microsoft Word |

**Keeper Application Development**

**B.1: Application Type**

Keeper will be a platform-independent, JavaFX standalone desktop application.

**B.2: Code Requirements**

Several code requirements were put forth in the development of this application including inheritance, polymorphism, and encapsulation. All of these requirements were met at least once as shown below.

**B.2.1: Inheritance**

**B.2.2: Polymorphism**

**B.2.3: Encapsulation**

**B.3: Search Functionality**

To showcase an element of search functionality, I used the Chart of Accounts page. Accounts are divided into five distinct types, Assets, Liabilities, Income, Expenses, and Equity. As you can see in the images below, when viewing a user’s created accounts, you can choose to either view all of the account types or each account type individually.

\*\*\*\*\*\*\*\*\*\*\*\*INSERT IMAGES HERE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Image 1 – chart of accounts page with the combo box on all but showing all the available options. Description is showing how you can view all the accounts on this page

Image 2 – chart of accounts with assets selected showing only the asset accounts

**B.4: Database Component**

An excellent implementation of a database component can be seen through the transaction recording feature. In the transaction feature, you are able to record transactions made. If a transaction is incorrectly entered, the user is able to edit or delete the transaction. This functionality can be seen in the images below.

\*\*\*\*\*\*\*\*\*\*\*\*INSERT IMAGES HERE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Image 1 – image of a transaction being saved

Image 2 – image of the transaction table and the transaction that was just saved. Circle the edit button signifying we are going to edit it

Image 3 – image of transaction being edited

Image 4 – image of edited transaction on table. Circle edit button again

Image 5 – image of transaction with delete button circled

Image 6 – image showing the transaction is now deleted from the table

**B.5: Report Generation**

The income statement is a report that is generated based on specific period inputted by the user. On the income statement, the user will be able to see the income, cost of goods sold, gross profit, operating expenses, and net profit based on the period entered. This can be updated as many times as the user deems necessary to see the income statement based on weeks, months, years, or any other chosen period of time.

\*\*\*\*\*\*\*\*\*\*\*\*INSERT IMAGES HERE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Image 1 – image of the income statement showing period of January 1st to todays date.

Image 2 – image of income statement showing last months income

**B.6: Exception Controls**

In the Chart of Accounts page, a user has the ability to edit and update the accounts they have created. One option given when updating an account is the option to archive an account. When this box is checked, and the user goes to save his now updated account, a dialog box pops up asking if the user is sure he wants to archive this account as this will prevent any further usage of the account. If the user clicks yes, the account will be saved.

\*\*\*\*\*\*\*\*\*\*\*\*INSERT IMAGES HERE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Image 1 – account detail page showing a check mark on the archive account button

Image 2 – the popup asking if the user is sure he wants to archive the account

**B.7: Validation Functionality**

When a user logs in, it is expected that they have the username or email associated with the account, as well as unique password that they created when signing up. If either of these are incorrect upon login, a message appears below the login stating that either the login credentials given are incorrect.

\*\*\*\*\*\*\*\*\*\*\*\*INSERT IMAGES HERE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Image – sign in screen showing an invalid login of username and password

**B.8: Security Features**

The main security feature provided in this application was developed within the login system. Each user has their own unique username and password combination that only they should have access to. With this username and password, they are able to login and view any data that they have created within that username and password.

\*\*\*\*\*\*\*\*\*\*\*\*INSERT IMAGES HERE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Image 1 – sign in page of a specific user logging in

Image 2 – main page showing the header of the circled user logged in, and transactions page with a description of only this user can see these transaction, chart of accounts, and reports.

**B.9: Scalable Elements**

In the applications current state, there is a lot of room for scalability of features and users. Currently, the database is hosted on RemoteMySQL.com which allows for one hundred megabytes of stored data. However, if the need arises for more data storage, the database can be easily migrated. First, we would need to set up our new database either in an internal or external server. Using the current database design, we will copy over all our user’s data into this new database. From there all that will need be done is update the connection in the “DBConnection” class with newly established database name, URL, user and password.

As far as features, there are many elements to which there were not enough time nor resources to implement. One would be a forgotten password element where if a user forgot their password, they can get a new password sent to them through their email. Another would be a more in-depth view of the reports that are generated. Each of these would not be difficult to implement given that I kept my code fairly clean and easy to interpret.

\*\*\*\*\*\*\*\*\*\*\*\*INSERT IMAGES HERE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Image – DBConnection class showing where to enter in the name, URL, user and password

**B.10: Graphical User Interface**

The main page of the application was built using two different FXML screens. The primary FXML screen is the header and sliding side menu as outlined below in red. Outlined in blue is the secondary FXML screen loaded into the primary screen. From these two screens various buttons can be pressed to pop up separate FXML windows, while staying keeping the main page of the application always open. Overall the interface is simple, intuitive, and functional.

\*\*\*\*\*\*\*\*\*\*\*\*INSERT IMAGES HERE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Image 1 – main page with the header and side menu outlined in red and the stackpane outlined in blue

Image 2 – main page picture circling the profile page button being pressed

Image 3 – screen shot of the profile page being opened in a separate window on the desktop

**Keeper Application Documentation**

**C.1: Design Documents**

Class Design

Below I have attached a diagram of the Model used in the Keeper application.

A screenshot of a cell phone

Description automatically generated

User Interface Design

Keeper was designed to be user-friendly and provide easy access to all the tools available within the application. Keeper was designed with a dark theme because of the growing popularity for this type of style.

Sign Up

Users access this page from the sign in page. Here is where a new user can create a login to get into the main interface of the application.

\*\*\*\*\*\*\*\*INSERT IMAGE\*\*\*\*\*\*\*\*

Sign In

When booting up the application, this is the initial landing page. From here you can navigate to the sign up window, or login if the user has already created an account.

\*\*\*\*\*\*\*\*INSERT IMAGE\*\*\*\*\*\*\*\*

Main Page

A user is presented with the main page once they have logged in. This is where the bulk of the application interface is. From here you can view the profile, chart of accounts, transactions, and reports pages.

\*\*\*\*\*\*\*\*INSERT IMAGE\*\*\*\*\*\*\*\*

Profile

From the main page, a user can navigate to the profile page. Here is where a user has the ability to view and update their personal information associated with their profile.

\*\*\*\*\*\*\*\*INSERT IMAGE\*\*\*\*\*\*\*\*

Chart of Accounts

On this page, users are able to manage their chart of accounts as well as view all of their Asset, Liability, Income, Expense, and Equity accounts.

\*\*\*\*\*\*\*\*INSERT IMAGE\*\*\*\*\*\*\*\*

Account Detail

Navigating from the Chart of Accounts page, users use this page to add and edit their accounts. If editing an account, users have the ability to archive an account, which will prevent any further usage of the account until it is unchecked.

\*\*\*\*\*\*\*\*INSERT IMAGE\*\*\*\*\*\*\*\*

Transactions

From this page, users are able to manage their created transactions as well navigate to the transaction detail page from this screen.

\*\*\*\*\*\*\*\*INSERT IMAGE\*\*\*\*\*\*\*\*

Transaction Detail

Navigating from the Transactions page, users use this page to add and edit their financial transactions.

\*\*\*\*\*\*\*\*INSERT IMAGE\*\*\*\*\*\*\*\*

Reports

\*\*\*\*\*\*\*\*INSERT IMAGE\*\*\*\*\*\*\*\*

**C.2: Unit Test Plan**

There are several unit tests we want to execute in order to test important functionalities found within the application. The first of these tests will be the login system, testing the ability to correctly validate a user’s email or username and their corresponding password. With this unit test we will create several correct and incorrect username, email and password combinations to test the login system function.

The second test will be on the chart of accounts page, where we will test the adding of a new account. In this test we want to make sure that the account is properly created and assigned to the correct account type selected. To implement this, we will create accounts and on the back end verify the accounts table has been correctly assigned.

Our final unit test will revolve around the correlation between transaction and accounts. When updating an account name, we want to make sure that the account being changed is also changed within the transactions table to show the newly updated account name rather than the old one. To do this we will update the account names of several different account types and validate that the transaction table no longer holds any accounts of the old name.

**C.3: Unit Test Results**

After the initial testing of the application, a few testing failures occurred. However, this resulted in the finding and correcting of several bugs that would have otherwise not been found. Unit tests are especially good at finding errors such that would otherwise remain unseen.

**C.4: Source Code and Executable File**

The source code can be found alongside the executable file and this documentation in the Zip file submitted.

**C.5: Web API Link**

Because this is a stand-alone desktop application, there is no web API link available.

**C.6: User Guide**

Installation

Installation is fairly simple in the application. A user will only need enough storage on their computer, and access to the internet to download and run the application. Once the application is downloaded, the user can execute the file and be prompted by a login screen. Here, they can create an account or login with an existing one. Because this application uses an external server, users will be able to access their data from any desktop device that has internet access and room for the download.

Maintenance

The application in its current state will require little if any ongoing maintenance. Once released for public use, the first six months will be spent getting user feedback and improving upon this feedback At the six month mark, we will send out a mass update to all our users with the feedback received. After this, updates will become less frequent and be done on an as needed basis.

User Guide

After downloading and installing the application, run the application and you should be prompted with a login screen. From the login screen if you are a new user, navigate to the sign-up page and enter in all of your relevant information. Once completed, you will be redirected back to the login page where you will be able to sign in with the information you just submitted.

From here you can start using the application. Each user should start by setting up their chart of accounts on the chart of accounts page. Since this varies from business to business, it is best to consult an accountant on what a chart of accounts should look like for the business you are in. After creating your chart of accounts, it is always possible to go back and edit or archive certain accounts on an as need basis.

After setup of the chart of accounts is complete, you can now add transactions. Anytime your business performs a financial transaction of any kind, it should be recorded within the application according to the nature of the transaction. Each transaction will have a transaction type (Income or Expense), an account (which is user created in the chart of accounts), a category (also created in the chart of accounts), a date, an amount, and a description. Once each of these fields are filled out the transaction may be saved.

Once the chart of accounts is created and a few transactions have been inputted, it is now possible to use the reporting feature. Navigate to the reports page, and you should be shown the income statement report. On this page you can enter a date range at the top and click update to view your income statement for that period. The balance sheet works in a similar fashion except there is only one date to enter.

In the case where you may have mis-entered something about your personal profile, you can click Profile at the top left-hand corner to update what is necessary.

<https://capstonearchives.wgu.edu/excellence/Attachments/Anthony%20Voelker-952/Capstone%20Written%20Report-AnthonyVoelker(secure).pdf>