

Design Document

Team PB-PI

March 18, 2018

Table 1: Team

Name	ID Number
Alissa Bellerose	27377320
Sabrina D'Mello	27739486
Melanie Damilig	40032420
Tobi Decary-Larocque	27407645
Zain Farookhi	26390684
Giulia Gaudio	27191766
Jason Kalec	40009464
Damian Kazior	40016168
Johnny Mak	40002140
Philip Michael	40004861
Ramez Nicolas Nahas	26718108
Steven Tucci	40006014
Shunyu Wang	40043915

Contents

1	Introduction	4
1.1	Purpose	4
1.2	Scope	4
1.3	Definitions and Abbreviations	4
1.3.1	Definitions	4
1.3.2	Abbreviations	5
1.4	References	5
1.5	Overview	5
2	Architectural Design	5
2.1	Architectural Diagram	5
2.1.1	Model	8
2.1.2	View	8
2.1.3	Controller	8
2.1.4	View/Controller Interface	8
2.1.5	Controller/Model Interface	8
2.2	Rationale	8
2.3	Subsystem Interface Specifications	9
2.3.1	View Interface Subsystem	9
2.3.2	Controller Interface Subsystem	10
2.3.3	Model Interface Subsystem	11
2.4	System Topology	12
3	Detailed Design	12
3.1	Main Interface	12
3.1.1	Basic Options	13
3.1.2	Advanced Options	14
3.1.3	Withdraw Button	14
3.1.4	Deposit Button	15
3.1.5	Show History Button	15
3.1.6	Export History	16

3.1.7	Clear Account	16
3.1.8	Enter Amount Field	16
3.1.9	Type of Deposit/Withdraw	17
3.1.10	Transaction Description	17
3.1.11	Current Balance Field	17
4	Dynamic Design Scenarios	18
4.1	Deposit Amount	18
4.1.1	Deposit Amount - Sequence Diagram	18
4.2	Withdraw Amount	18
4.2.1	Withdraw Amount - Sequence Diagram	18
4.3	Show Balance	21
4.4	Show History	21
4.5	Sort History	21
4.6	Export History	22
4.7	Clear History	22
4.8	Display GUI	22
5	Conclusion	22

1 Introduction

The primary goal of this project is to create an application which allows students to keep track of their money. The MyMoney application allows students to create an account which provides them with different options to keep track of their money and spending habits through the graphical user interface. The application allows the user to create a transaction, either deposit or withdraw. It will also allow the user an option to display balance, show or export transaction history and clear history.

1.1 Purpose

The purpose of this document is to provide details on the architectural design, software design and internal design of the MyMoney application. The software architecture that was chosen for the application will be described in high level detail and a class diagram will be depicted. The software interface will have screenshots of the graphical user interface and a high level description of how the user will interact with the system.

1.2 Scope

This document is intended to provide a basis for implementation. The architecture and software processes will be explained in great detail in order to facilitate implementation and be an effective reference tool.

1.3 Definitions and Abbreviations

1.3.1 Definitions

Table 2: Definitions

Term		Definition
Model	View	The architecture used in the MyMoney application. It consists of 3 individual components the model, the view and the controller.
Controller		

1.3.2 Abbreviations

Table 3: Abbreviations

Abbreviation	Term
GUI	Graphical User Interface
MVC	Model View Controller
UML	Unified Modeling Language
ORM	Object Relational Mapping
API	Application Programming Interface
CRUD	Create, Read, Update, Delete
SQL	Structured Query Language

1.4 References

Pressman, Roger S. Software Engineering: A Practitioner's Approach. 5th ed. Toronto: McGraw-Hill, 2001.

Larman, Craig. Applying UML and patterns: an introduction to object-Oriented analysis and design and the unified process. Prentice-Hall, 2005.

1.5 Overview

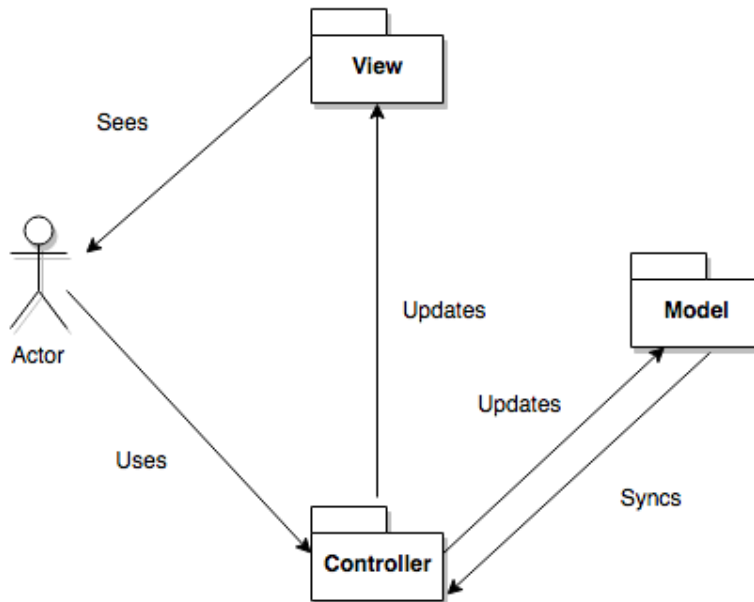
This document is divided into three major parts; the architectural design, the detailed design and the dynamic design scenarios. The architectural and software design will be described in detail in their respective individual parts.

2 Architectural Design

2.1 Architectural Diagram

The MyMoneyApp uses a variant of the MVC architecture with an observer pattern to notify of data changes. The MVC pattern is known as the Model-View-Controller pattern. Along with the MVC architecture the application also has singleton patterns. The singleton design pattern is used for only instantiating one object from a class. The singleton pattern was used for the creating the default GUI layout for the user and the initial connection to the applications database.

High Level MVC



MVC Architecture

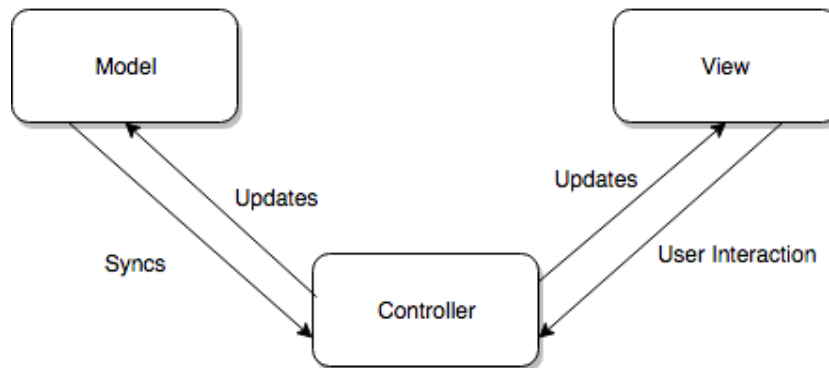


Figure 1: MVC Diagrams

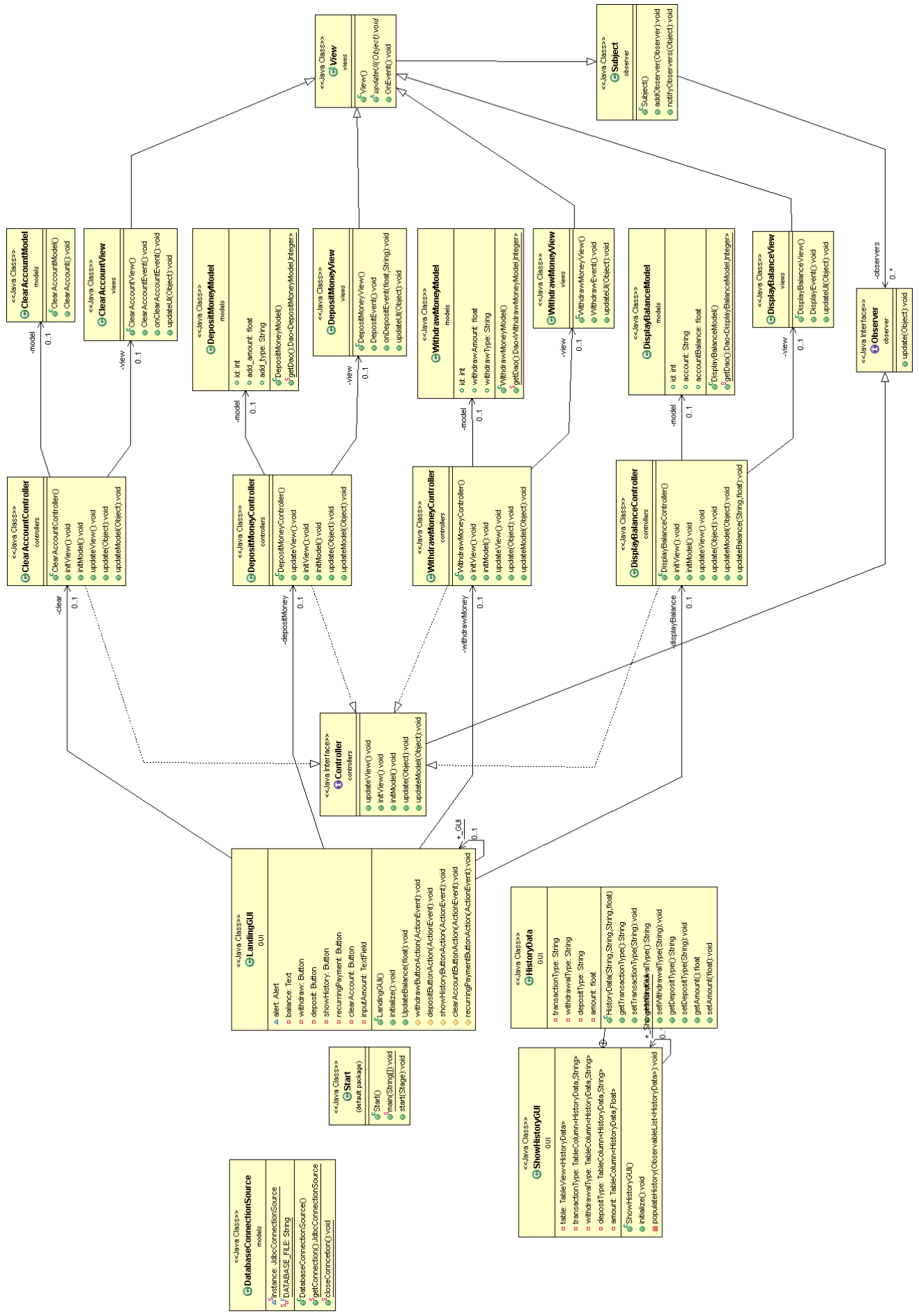


Figure 2: Class diagram

2.1.1 Model

The model component is where we store our business models and data. For that, we use an abstraction known as ORM (Object Relational Mapping). The ORM allows us to perform CRUD operations on our data without writing hardcoded specific database queries. This allows us to perform queries on a higher abstraction level and be able to switch our data store without having to rewrite new query code.

2.1.2 View

The view component layer is where the user interacts with the system. In our case this is the GUI, but the view layer does not necessarily have to be a GUI. The view displays our models, collection of models, or any mix of model data. Since the user interacts with the view, all view logic at the view layer. As a result, when a user presses a button, the view captures the event, then it creates a message and passes it to the controller to handle the event.

2.1.3 Controller

The controller component acts as an intermediary between the view and model. When an event occurs in the view, the view passes the message to the controller. The controller reads the message, and reflects the changes from the view across the model. The controller then updates the model data where it gets automatically updated into our database. Once the model gets updated, the controller takes any changes from the model and sends a message back to the view with the new updated model data. The view then takes this message and reflects the view/gui changes with the data.

2.1.4 View/Controller Interface

In the classic version of MVC, the model is the subject, and the view is the observer that subscribes to the model. Our version does not do this and will be further explained in the rationale.

2.1.5 Controller/Model Interface

The controller does not directly manipulate the model through database queries, but instead it manipulates the models high level ORM API.

2.2 Rationale

MyMoneyApp uses an evolved version of the MVC architecture with the observer pattern. It allows us to independently manage business logic from the view logic. We could create

new views without ever worrying about how the controller and model work. This allows for new controllers and models to be created, tested, debugged, and integrated in parallel. Though our architecture may not be the same MVC as stated in Larman's book, it uses a newer modern version variant of MVC, in which the controller acts as a mediator between the model and view. The view causes an event, delegates to the controller, the controller modifies the model, then the controller notifies the view of the model changes. The diagrams are a good visual representation of this new structure. We can see that we don't have a connection between our model and view. That is because the controller handles it. In the original MVC pattern, the model directly notifies the view through the observer pattern. For our structure, our controller notifies the view through the observer pattern. This removes the coupling between our model and view and allows the controllers and models to be tested much easier. Given that the model isn't in charge of notifying the view, one would think that the views would become out of date. However it does not, because it is now the controller notifying its dependant views.

2.3 Subsystem Interface Specifications

All the subsystems interact with a message passing interface. Each view has a custom data type message that is passed to the other subsystems. The corresponding controller should expect the custom message and handle it appropriately. When messages are passed around, they are passed as high level Object class types, the views and controllers must down cast the object to its expected message type/class.

2.3.1 View Interface Subsystem

Table 4: updateUi method spec

Method:	void updateUI(Object data)
Purpose:	To notify the view that it should update its view state with the current data message. This is the controller to view interface. This is used when the controller sends a message to the view.
Parameters:	Object data. The data/message that the current view should unpack/cast and update its state with. The view should cast the object to its own object type.
Valid data:	An object of type that is known to the view and controller
Invalid data:	Null, or an unknown data message type. The view should handle invalid data in an expected manner, and should not throw any exceptions up the system interface.

Table 5: notifyObservers method spec

Method:	void notifyObservers(Object data)
Purpose:	Notify all dependent observers/controllers with a new message/data. This is the view to controller interface. This is used when the view wants to send a message to the controller.
Parameters:	Object data. The data/message that the current view sends to its controller when the view wants to notify the controller of a view event or change.
Valid data:	An object of type that is known to the view and controller
Invalid data:	Null, or an unknown data message type. The view should handle invalid data in an expected manner, and should not throw any exceptions up the system interface.

Table 6: addObserver method spec

Method:	void addObserver(Observer observer)
Purpose:	Add an observer/controller to the current subject/view.
Parameters:	Observer observer. The controller that subscribes to the view/subjects events or changes.
Valid data:	An observer object that should be expected to handle the views messages.
Invalid data:	Null, or an unknown controller type that would not know how to handle the views messages.

2.3.2 Controller Interface Subsystem

Table 7: initModel method spec

Method:	void initModel()
Purpose:	Initialize any model or models that the controller needs to update or create.
Parameters:	None.

Table 8: initView method spec

Method:	void initView()
Purpose:	Initialize the view and setup any needed view logic.
Parameters:	None.

Table 9: updateView method spec

Method:	void updateView()
Purpose:	Tells the controller's attached view to update its ui. This method is called after any model changes have happened, and the view needs to reflect these changes.
Parameters:	None.

Table 10: update method spec

Method:	void update(Object data)
Purpose:	Update the controller/observer data/state. Update the model with the new changes from the views data/message.
Parameters:	Object - data, this is the corresponding data message passed from the view's notifyObserver(Object data) call the object is usually going to be type casted to the specified data type depending on the what the view's data is. the data reflects the state of the view
Valid data:	An object of type that is known to the view and controller
Invalid data:	Null, or an unknown data message type. The controller should handle invalid data in an expected manner, and should not throw any exceptions up the system interface.

2.3.3 Model Interface Subsystem

For our model, we are using an ORM called ORMLite. It allows us persist our business objects to a database using a high level api and avoid us having to write our own custom and insecure SQL queries.

2.4 System Topology

The MyMoneyApp is to be used by a single user and ran on a single computer. There will be no need for networked communications or internet connections for the app to work. This allows us to easy distribute the app and integrate all the components into a single executable.

3 Detailed Design

The primary User Interface used in the system design, was created using Java FX, which is quite similar to Java Swing. This GUI gives users the ability to interact with the core necessary aspects of the system in order to obtain a satisfactory user experience.

User interactions:

The user may press Withdraw Amount to reveal the withdrawal interface where the user may then enter an amount, a type of withdrawal and an optional transaction description, then click Done to confirm the withdraw. A user may alternatively press Cancel to discard the withdraw.

The user may press Deposit Amount to reveal the withdrawal interface where the user may then enter an amount, a type of withdrawal and an optional transaction description, then click Done to confirm the withdraw. A user may alternatively press Cancel to discard the withdraw.

The user may press Show History to bring up a list of past transactions as well as all relevant information regarding each transaction. The user can sort by each category such as date or type.

The user may press Clear Account to bring the app back to original settings, clear history and set the balance back to 0.

The user has the option to press Export History to write the past transactions to a CSV file which can then be viewed as a text file or it may be opened in excel

3.1 Main Interface

The systems main interface is composed of various elements required to manage ones money. There are basic options such as Withdraw Amount, Deposit Amount and Show Balance. There are also advanced options such as Clear Account and Export History. The User interface has a Amount box where one can enter the amount of money they

may wish to deposit or withdraw as well as a type of deposit or withdraw Bill Payment being an example as well as an optional Transaction Description. The app is constantly displaying the users Current Balance so they will constantly be aware of how much money they currently have in green if it is positive or in red if the user is in debt.

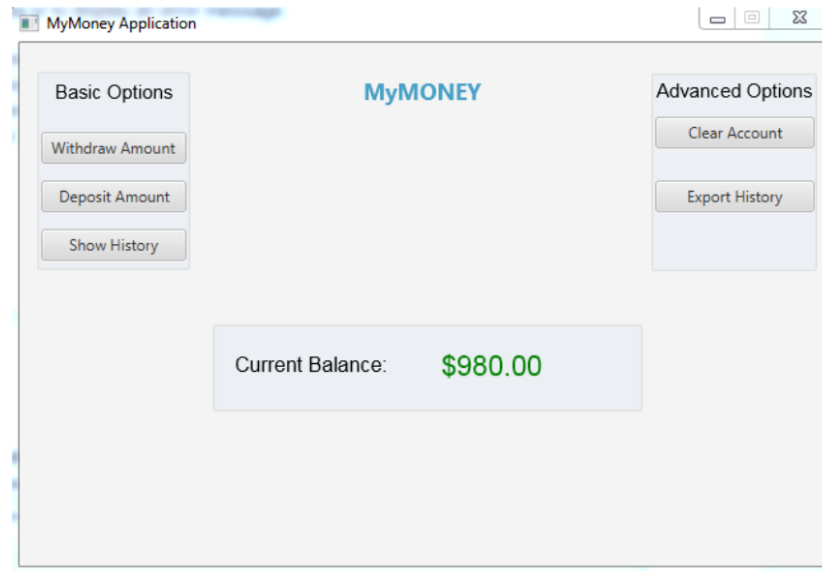


Figure 3: MyMoney Application Window

3.1.1 Basic Options

The basic options subsection of the GUI is displayed as such all functionality that the user will be using frequently such as deposit, withdraw and show history will be available here.

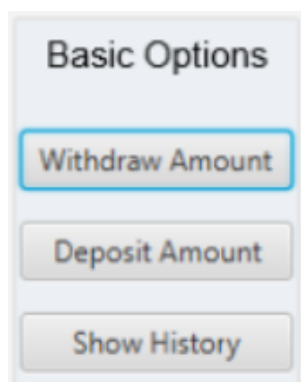


Figure 4: Basic Options

3.1.2 Advanced Options

The advanced options subsection of the GUI is displayed as such all functionality that is not frequented often such as Export History and Clear Account will be available here.

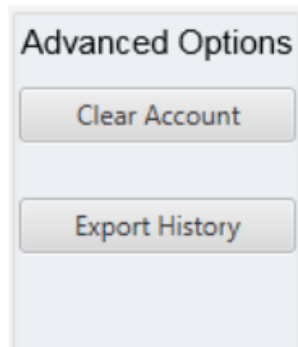


Figure 5: Advanced Options

3.1.3 Withdraw Button

The withdraw button of the basic options subsection when clicked on will display to the user the withdraw interface.

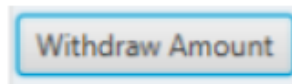


Figure 6: Withdraw Button

The Withdrawal interface is where the user will chose the properties and information for their transaction. The user must specify an amount to withdraw and the type of withdraw for example a Bill Payment and the user may enter an optional transaction description to better describe their withdrawal. Once finished, the user may click ok to confirm the withdraw or press cancel to go back to the main interface and discard the withdraw.

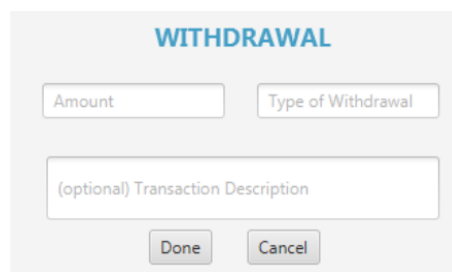
A light gray rectangular form titled "WITHDRAWAL" in bold blue text. The form contains three input fields: a text box labeled "Amount", a dropdown menu labeled "Type of Withdrawal", and a larger text box labeled "(optional) Transaction Description". At the bottom of the form, there are two buttons: "Done" and "Cancel".

Figure 7: Withdraw Interface

3.1.4 Deposit Button

The deposit button of the basic options subsection when clicked on will display to the user the deposit interface.

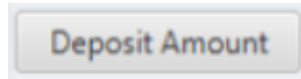


Figure 8: Deposit Button

The Deposit interface is where the user can specify amount, type of withdrawal and transaction description for their transaction. The user must specify an amount to deposit and the type of deposit for example a Paycheck and the the user may enter an optional transaction description to better describe their deposit once finished the user may click ok to confirm the deposit or press cancel to go back to the main interface and discard the deposit.

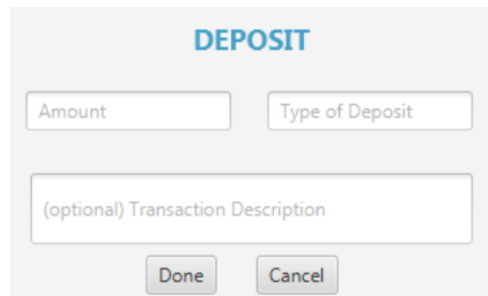
A light gray rectangular form titled "DEPOSIT" in blue. It contains three input fields: "Amount", "Type of Deposit", and "(optional) Transaction Description". At the bottom are two buttons: "Done" and "Cancel".

Figure 9: Deposit Interface

3.1.5 Show History Button

The show history button of the basic options subsection when clicked upon will show the past transactions that the user has done such as withdrawals and deposits.

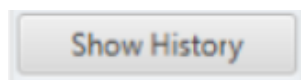


Figure 10: Show History Button

The history interface will be displayed upon clicking Show History. The history interface will display all past transactions as well as all relevant information regarding each past transaction. Each transaction will be displayed ordered by date, there is the option to order by type, description, etc. Each transaction will have information regarding the date that it was made, whether it was a withdraw or a deposit, the description of the transaction, the amount, and the type of withdraw or type of deposit.

DATE ▼	TRANSACTION TYPE	DESCRIPTION	AMOUNT	TYPE OF WITHDRAWAL	TYPE OF DEPOSIT
05/04/2018 13:22:43	Withdawal	Food delivery	20.0	Credit Card Payment	
05/04/2018 13:20:21	Deposit	Weekly Salary	400.0		Direct Deposit
05/04/2018 13:15:53	Withdawal	Paid back money owed to friend	100.0	Interac	
05/04/2018 13:15:27	Withdawal	Took out cash for bday gift	50.0	ATM	
05/04/2018 13:14:27	Deposit	Kijiji sale	250.0		Cash
05/04/2018 13:14:09	Deposit	Cheque from tenant	500.0		Cheque

Figure 11: Show History Interface

3.1.6 Export History

The Export History button when clicked on will take all past transactions and write them to a csv file which can then be imported into excel.

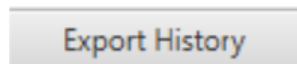


Figure 12: Export Button

3.1.7 Clear Account

The clear account button of the advanced options subsection will delete the current account and reset the application back to the clear default state.

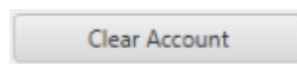


Figure 13: Export Button

3.1.8 Enter Amount Field

The amount field located in the center of the GUI in the deposit and withdraw interface is where the amount is typed to perform operations on the current balance such as typing in a certain amount to manipulate the current balance and perform the transaction.

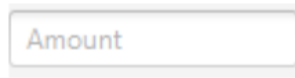


Figure 14: Enter Amount Field

3.1.9 Type of Deposit/Withdraw

The Type of Deposit and Type of Withdraw is where the user would specify what type of withdraw or deposit they intend to apply, for example Pay Check or Bill Payment which will then be inserted into the database to quickly describe the deposit or withdraw.

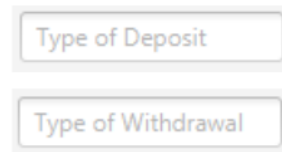


Figure 15: Type of Deposit/Withdraw Field

3.1.10 Transaction Description

The optional Transaction Description text box is used to describe a transaction if the type of deposit or withdrawal is not enough. This will be inserted into the database to thoroughly describe the transaction being inserted.

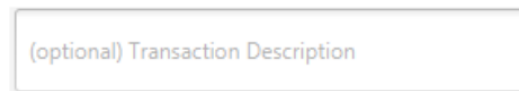


Figure 16: Transaction Description

3.1.11 Current Balance Field

The current balance field at the bottom of the GUI is where the users current balance is displayed. The field will display green if the current balance is greater than 0 and red if the current balance is less than 0.

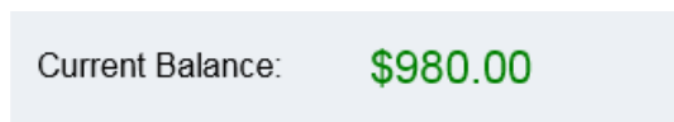


Figure 17: Current Balance Field

4 Dynamic Design Scenarios

As explained, our software offers many functionalities that allows the user to interact with it. These functionalities were developed based on the gathered user stories. It includes: Deposit Amount, Withdraw Amount, Show Balance, Show History, Clear History, and Display GUI.

4.1 Deposit Amount

Our software provides the user with basic functionality such as Deposit Amount which simply allows the user to input an amount along with certain details to be saved as a deposit as explained above in the Main Interface section. Once stored, the data can be used by other functionalities to extend the user intractability with the system.

4.1.1 Deposit Amount - Sequence Diagram

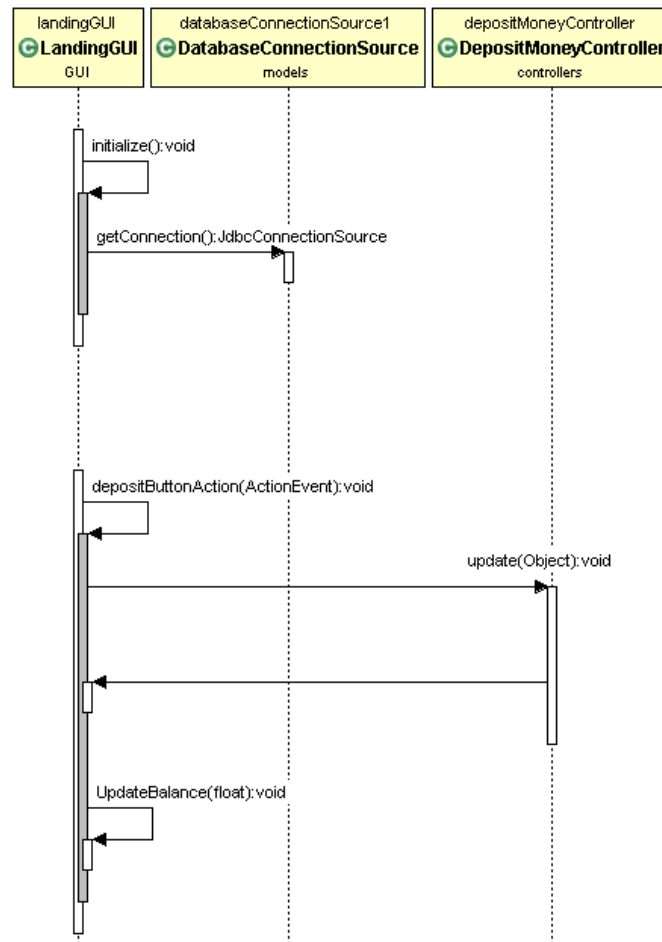


Figure 18: Deposit Sequence Diagram

The Deposit Amount use case is a basic functionality that is invoked from the GUI. When the GUI is launched, it establishes a connection with the database and then waits for user inputs. Once the user entered an amount into the Deposits input box and confirmed their action by pressing the Deposit Amount button, the function `depositButtonAction(ActionEvent)` is called which tells the `DepositMoneyController` to update the Model. When updated, the `DepositMoneyController` will signal the GUI that the information has been successfully updated and that it needs to display the new information. The GUI will then call `UpdateBalance(float)` which will finally update the displayed balance to reflect the new amount.

4.2 Withdraw Amount

Similar to the Deposit Amount use case, Withdraw Amount is another basic functionality that allows the user to interact with the system. It requires the user to input a withdrawal amount which is saved into the Withdraw section of the account.

4.2.1 Withdraw Amount - Sequence Diagram



Figure 19: Withdraw Sequence Diagram

This use case functions very similarly to the Deposit Amount use case. It establishes a database connection when the GUI is launched and waits for user inputs. Once a withdrawal amount has been entered and confirmed by the user, it triggers the `withdraw-Button(ActionEvent)` function which informs the `WithdrwaMoneyController` to update the Model. The controller will then update the model and inform the GUI that the changes has been successfully applied. The GUI will then call `UpdateBalance(float)` function which will update the displayed balance to reflect the new amount.

4.3 Show Balance

Show Balance is a feature that allows the user to see the difference between their monthly earning and spending. This amount can be negative or positive to reflect the users monthly money situation. This use case solidifies the intent of our software which allows the user to know about their spending habits for a given period of time.

Additionally, the balance amount is shown on the main window of the software and is constantly updated when specific actions are made. For instance, a successful Deposit or Withdraw amount input would update the balance once the action is complete.

4.4 Show History

Show History is a functionality that allows the user to see all previous Deposit and Withdraw inputs to review certain purchases they have made or certain deposits they have forgotten for instance. When the Show History button is pressed, a new window containing the information will pop up on top of the main window. This new window is independent to the main window and can be freely manipulated by the user.

Within this new window, it will display a table with all Deposit and Withdraw inputs since the beginning. The data is sorted by the input date from newest to oldest. The user can resume their normal activity with the software regardless of whether the Show History window is active or not. Although, if the user inputs more data with the history window still open, it will not display the new additions and would require the user to reopen the same window to see the changes.

4.5 Sort History

After opening up the Show History window, the user has the choice to sort any column of the window. By default, the data is sorted by newest date but the user can opt to sort the data by either: date, transaction type, description, amount, type of withdrawal, or type of deposit. This functionality does not affect how the data is stored. It will only manipulate how the data is displayed to the user.

4.6 Export History

Export History is a functionality very similar to Show History. Just like the other functionality, it shows to the user all previous Deposit and Withdraw inputs but instead, exports this data into a CSV file. This file containing all the information can then be opened into Microsoft Excel where the user can further analyze or archive his data.

4.7 Clear History

Clear History is another functionality that allows the user to remove all data that has ever been saved into the software. This allows any old or unnecessary information to be removed allowing the user to only see recent information. By pressing the Clear History button, it will empty all tables within the database such as Deposit and Withdraw amounts. It is recommended that the user first exports their data before clearing their history.

4.8 Display GUI

Display GUI is a feature that allows the user to easily interact with the system. The GUI simplifies the number of steps required for the user to carry out an action on the software by allowing them to simply type into input boxes and press on buttons to confirm their actions. This feature also compacts all other features and functionalities into one window which greatly improves the user's experience.

5 Conclusion

The MyMoneyApp was created for users to be able to manage their personal financials. The App allows them to constantly keep track of their money that has been deposited and withdrawn. The user can also view their history of their transactions as well as view their balance. Another feature of this application is that it allows the user to clear their history of transactions. This document was to analyse the design of the application.

The primary user interface of the MyMoneyApp was created with Java FX. The GUI allows easy and quick access for the user to deposit, withdraw, view balance, view history and clear transactions.

The MyMoneyApp design documentation was created to express the major design implementation that was used in creating this money management application for an average everyday user.