

# Test Document

## Team PB-PI

April 8, 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

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Test Plan</b>	<b>4</b>
2.1	Use Cases in the myMoney application . . . . .	5
2.2	System Level Test Cases . . . . .	6
2.2.1	[TC-1] [Withdraw Money] . . . . .	6
2.2.2	[TC-2] [Deposit Money] . . . . .	11
2.2.3	[TC-3] [Display Balance] . . . . .	16
2.2.4	[TC-4] [Clear Account] . . . . .	17
2.2.5	[TC-5] [Show History] . . . . .	18
2.2.6	[TC-6] [Sort Table] . . . . .	19
2.2.7	[TC-7] [Export History] . . . . .	25
<b>3</b>	<b>Unit Test Cases</b>	<b>26</b>
3.1	Display Balance Unit Testing . . . . .	26
3.2	Data Unit Testing . . . . .	29
<b>4</b>	<b>Test Results</b>	<b>30</b>
4.1	Deposit Money Test . . . . .	30
4.2	Withdraw Money Test . . . . .	32
<b>5</b>	<b>References</b>	<b>33</b>
<b>A</b>	<b>Description of Input Files</b>	<b>34</b>
<b>B</b>	<b>Description of Output Files</b>	<b>35</b>
<b>C</b>	<b>Figures</b>	<b>36</b>

## List of Tables

1	Team . . . . .	1
2	Use Cases (in reference to Requirements Document) . . . . .	5
3	TC-1.1 : Testing the Amount input . . . . .	6
4	TC-1.2 : Testing the type of withdraw input . . . . .	7
5	TC-1.3 : Testing the transaction description input . . . . .	8
6	TC-1.4 : Testing the Done UI Button . . . . .	9
7	TC-1.5 : Testing the Cancel UI Button . . . . .	10
8	TC-2.1 : Testing the amount input . . . . .	11
9	TC-2.2 : Testing the type of deposit . . . . .	12
10	TC-2.3 : Testing the transaction description input . . . . .	13
11	TC-2.4 : Testing the Done UI Button . . . . .	14
12	TC-2.5 : Testing the Cancel UI Button . . . . .	15
13	TC-3.1 : Testing the Display Balance . . . . .	16
14	TC-4.1 : Testing Clear Account . . . . .	17
15	TC-5.1 : Testing Show History . . . . .	18
16	TC-6.1 : Testing Sort by Date . . . . .	19
17	TC-6.2 : Testing Sort by Transaction Type . . . . .	20
18	TC-6.3 : Testing Sort by Description . . . . .	21
19	TC-6.4 : Testing Sort by Amount . . . . .	22
20	TC-6.5 : Testing Sort by Withdrawal Type . . . . .	23
21	TC-6.6 : Testing Sort by Deposit Type . . . . .	24
22	TC-7.1 : Testing Export History . . . . .	25
23	initialBalanceTest() . . . . .	26
24	updateBalanceTest() . . . . .	27
25	updateModelTest() . . . . .	28
26	CreateDummyDeposit() . . . . .	29
27	CreateDummyWithdrawal() . . . . .	30
28	Input Files . . . . .	34
29	Input Files . . . . .	35

# 1 Introduction

This document contains the overall test plan for the myMoney application. The myMoney application is a tool for young adults and students to keep track of their spending and help with their budgeting. This test plan describes the different tests performed to ensure a working program and explains the reasoning for the tests.

The purpose of this document is to provide assurance and show the reliability of the software. The test plan therefore includes:

- An overall view of the different tests performed
- A description of the tests
- References to the use cases and requirements satisfied by testing

Testing the program's different functionalities involves different types of tests. Unit, integration and user interface testing were performed for the purpose of this document. This was managed via black-box, white-box and boundary testing.

## 2 Test Plan

For the deposit and withdraw use cases, different unit tests were performed (including boundary tests) to ensure functionality. The different test cases were designed to test each part of each use case. This includes the format of what the user inputs, as well as ensuring that the database is updated or not, according to whether the transaction is legal or not.

In regards to our requirements document, the test plan references the deposit and withdrawal use cases specifically. The tests for each of these use cases are integral to the proper functionality of the application and are therefore an important part of the testing sequence. The balance display use case does not have any functional testing to ensure that it works correctly, but any modification to the user's balance should be reflected through the appropriate view. Similarly, the use cases for the "Show History" functionality as well as "Export History" do not rely on any user input except for a button click, so testing was only performed to make sure the output files are correct.

There were no tests performed to check if the application is portable to all operating systems described in the non-functional requirements, since the application is not being exported as a stand-alone app. It will function within the Eclipse environment for the time being.

## 2.1 Use Cases in the myMoney application

Table 2: Use Cases (in reference to Requirements Document)

Use Case ID	Use Case Name	Scenario
UC-1	Withdraw Money	The user indicates the amount of money they would like to withdraw from the account. The system will verify that the account has sufficient funds. Once confirmed, the user request will be processed and the system will subtract the money from the account. ALTERNATIVE SCENARIO: The system will deny the request if the user input is not valid or the account does not exist.
UC-2	Deposit Money	The user indicates the amount they would like to deposit into an account. The system will verify that the input is valid and adds that amount to the specified account. A.S.: The user enters invalid input or the account does not exist.
UC-3	Display Balance	The user can see the balance of their account. The system will retrieve the amount from the user's specified account.
UC-4	Clear Account	The user clears their current account. The system resets to its cleared default state. A.S.: The account is already in "default" state.
UC-5	Show History	The user can view all their deposits and withdrawal transactions that have been entered and processed.
UC-6	Sort Table	The user can request to sort their transaction history by certain categories (namely their entered descriptions). The system will sort the information in the way the user requests.
UC-7	Export History	The user can request to have the transaction history exported to a separate file. The system will create a .csv file for the user. A.S.: The user does not have a transaction history so the request will fail. No file will be produced.

## 2.2 System Level Test Cases

### 2.2.1 [TC-1] [Withdraw Money]

Table 3: TC-1.1 : Testing the Amount input

<b>Test Case Number</b>	TC-1.1
<b>Description</b>	The user enters the amount to withdraw in the Amount input text field.
<b>Input</b>	<ol style="list-style-type: none"><li>1. Click Withdraw Money button</li><li>2. Enter the amount as a positive number in Amount input text field environment.</li><li>3. Click Done button</li></ol>
<b>Expected Output</b>	<ol style="list-style-type: none"><li>1. The system records the number from input as Amount in a row of transaction history table.</li><li>2. New balance will be updated after deducting the input number from the current Account Balance.</li></ol>
<b>Expected Post-Condition</b>	Account balance will be presented as a negative value with two decimal places if it belows 0. Error message is displayed if the input is not valid, and Current Balance and database will not be updated.
<b>Execution History</b>	[04/14/2018—Shunyu Wang] Executed test successfully.
<b>Trace to Use Case</b>	UC-1

Table 4: TC-1.2 : Testing the type of withdraw input

<b>Test Case Number</b>	TC-1.2
<b>Description</b>	Takes the type of withdraw defined by the user.
<b>Input</b>	<ol style="list-style-type: none"> <li>1. Click Withdraw Money button</li> <li>2. Enter any positive number in Amount input text field.</li> <li>3. Enter any type of withdraw defined by the user, such as Bill, Check, or empty string</li> <li>4. Click Done button</li> </ol>
<b>Expected Output</b>	<ol style="list-style-type: none"> <li>1. The system records the number from input as Amount and the type of withdraw as Type of Withdraw in a row of transaction history table.</li> <li>2. New balance will be updated after deducting the input number from the current Account Balance.</li> </ol>
<b>Expected Post-Condition</b>	The same string will be taken from input as the output for Type of Withdraw in transaction history table. Empty string is allowed.
<b>Execution History</b>	[04/14/2018—Shunyu Wang] Executed test successfully.
<b>Trace to Use Case</b>	UC-1

Table 5: TC-1.3 : Testing the transaction description input

<b>Test Case Number</b>	TC-1.3
<b>Description</b>	Take any user-defined descriptions, such as purposes for transactions and other special notes.
<b>Input</b>	<ol style="list-style-type: none"> <li>1. Click Withdraw Money button</li> <li>2. Enter any positive number in Amount input text field</li> <li>3. Enter any user-defined description in Transaction Description text field, such as Pay electricity bill of June, Pay off the debt owed to Jack , or empty string</li> <li>4. Click Done button</li> </ol>
<b>Expected Output</b>	<ol style="list-style-type: none"> <li>1. The system records the number from input as Amount and the user-defined description as Transaction Description in a row of transaction history table.</li> <li>2. New balance will be updated after deducting the input number from the current Account Balance.</li> </ol>
<b>Expected Post-Condition</b>	The same string will be taken from input as the output for Transaction Description in transaction history table. Empty string is allowed.
<b>Execution History</b>	[04/14/2018—Shunyu Wang] Executed test successfully.
<b>Trace to Use Case</b>	UC-1



Table 6: TC-1.4 : Testing the Done UI Button

<b>Test Case Number</b>	TC-1.4
<b>Description</b>	When clicked, the UI and the database are updated according to the values entered by the user. In some cases, error messages are displayed.
<b>Input</b>	<ol style="list-style-type: none"> <li>1. Click Withdraw Money button</li> <li>2. Refer to TC-1.1, TC-1.2, and TC-1.3, input testing data appropriately</li> <li>3. Click Done button</li> </ol>
<b>Expected Output</b>	<ol style="list-style-type: none"> <li>1. The system records the number from input as Amount, the type of withdraw as Type of Withdraw, and the user-defined description as Transaction Description in a row of transaction history table.</li> <li>2. New balance will be updated after deducting the input number from the current Account Balance.</li> </ol>
<b>Expected Post-Condition</b>	Account balance will be presented as a negative value with two decimal places if it falls belows 0. All testing data should be recorded together in database as a row transaction history, which can be examined by clicking Show History
<b>Execution History</b>	[04/14/2018—Shunyu Wang] Executed test successfully.
<b>Trace to Use Case</b>	UC-1

Table 7: TC-1.5 : Testing the Cancel UI Button

<b>Test Case Number</b>	TC-1.5
<b>Description</b>	When clicked, the transaction is canceled: the Account Balance and database are not updated.
<b>Input</b>	<ol style="list-style-type: none"> <li>1. Click Withdraw Money button</li> <li>2. Input any testing data into text fields</li> <li>3. Click Cancel button</li> </ol>
<b>Expected Output</b>	<ol style="list-style-type: none"> <li>1. Data are cleared and all input text fields are collapsed.</li> </ol>
<b>Expected Post-Condition</b>	The system state doesnt change, meaning Current Balance is not changed and no change in database.
<b>Execution History</b>	[04/14/2018—Shunyu Wang] Executed test successfully.
<b>Trace to Use Case</b>	UC-1

...

### 2.2.2 [TC-2] [Deposit Money]

Table 8: TC-2.1 : Testing the amount input

<b>Test Case Number</b>	TC-2.1
<b>Description</b>	The user enters the amount to deposit in the Amount input text field.
<b>Input</b>	<ol style="list-style-type: none"><li>1. Click Deposit Money button</li><li>2. Enter the amount as a positive number in Amount input text field.</li><li>3. Click Done button</li></ol>
<b>Expected Output</b>	<ol style="list-style-type: none"><li>1. The system records the number from input as Amount in a row of transaction history table.</li><li>2. New balance will be updated after adding the input number to the current Account Balance.</li></ol>
<b>Expected Post-Condition</b>	Account balance will be presented as a positive value with two decimal places. Error message is displayed if the input is not valid, and Current Balance and database will not be updated.
<b>Execution History</b>	[04/14/2018—Shunyu Wang] Executed test successfully.
<b>Trace to Use Case</b>	UC-2

Table 9: TC-2.2 : Testing the type of deposit

<b>Test Case Number</b>	TC-2.2
<b>Description</b>	Takes the type of deposit defined by the user.
<b>Input</b>	<ol style="list-style-type: none"> <li>1. Click Deposit Money button</li> <li>2. Enter the amount as a positive number in Amount input text field.</li> <li>3. Enter any type of deposit defined by the user, such as Direct Deposit, Cash, or empty string</li> <li>4. Click Done button</li> </ol>
<b>Expected Output</b>	<ol style="list-style-type: none"> <li>1. The system records the number from input as Amount and the type of deposit as Type of Deposit in a row of transaction history table.</li> <li>2. New balance will be updated after adding the input number to the current Account Balance.</li> </ol>
<b>Expected Post-Condition</b>	The same string will be taken from input as the output for Type of Deposit in transaction history table. Empty string is allowed.
<b>Execution History</b>	[04/14/2018 — Shunyu Wang] Executed test successfully.
<b>Trace to Use Case</b>	UC-2

Table 10: TC-2.3 : Testing the transaction description input

<b>Test Case Number</b>	TC-2.3
<b>Description</b>	Take any user-defined descriptions, such as purposes for transactions and other special notes.
<b>Input</b>	<ol style="list-style-type: none"> <li>1. Click Deposit Money button</li> <li>2. Enter the amount as a positive number in Amount input text field.</li> <li>3. Enter any user-defined description in Transaction Description text field, such as Bi-weekly pay, Receive the bonus for working overtime, or empty string.</li> <li>4. Click Done button</li> </ol>
<b>Expected Output</b>	<ol style="list-style-type: none"> <li>1. The system records the number from input as Amount and the user-defined description as Transaction Description in a row of transaction history table</li> <li>2. New balance will be updated after adding the input number to the current Account Balance</li> </ol>
<b>Expected Post-Condition</b>	The same string will be taken from input as the output for Transaction Description in transaction history table. Empty string is allowed.
<b>Execution History</b>	[04/14/2018 — Shunyu Wang] Executed test successfully.
<b>Trace to Use Case</b>	UC-2

Table 11: TC-2.4 : Testing the Done UI Button

<b>Test Case Number</b>	TC-2.4
<b>Description</b>	When clicked, the UI and the database are updated according to the values entered by the user. In some cases, error messages are displayed.
<b>Input</b>	<ol style="list-style-type: none"> <li>1. Click Deposit Money button</li> <li>2. Refer to TC-2.1, TC-2.2, and TC-2.3, input testing data appropriately. string.</li> <li>3. Click Done button</li> </ol>
<b>Expected Output</b>	<ol style="list-style-type: none"> <li>1. The system records the number from input as Amount, the type of deposit as Type of Deposit, and the user-defined description as Transaction Description in a row of transaction history table.</li> <li>2. New balance will be updated after adding the input number to current Account Balance.</li> </ol>
<b>Expected Post-Condition</b>	Account balance will be presented as a negative value with two decimal places if it falls belows 0. All testing data should be recorded together in database as a row transaction history, which can be examined by clicking Show History
<b>Execution History</b>	[04/14/2018 — Shunyu Wang] Executed test successfully.
<b>Trace to Use Case</b>	UC-2

Table 12: TC-2.5 : Testing the Cancel UI Button

<b>Test Case Number</b>	TC-2.5
<b>Description</b>	When clicked, the transaction is canceled: the UI and database are not updated.
<b>Input</b>	<ol style="list-style-type: none"> <li>1. Click Deposit Money button</li> <li>2. Input any testing data into text fields</li> <li>3. Click Cancel button</li> </ol>
<b>Expected Output</b>	<ol style="list-style-type: none"> <li>1. All data are cleared and all input text fields are collapsed.</li> </ol>
<b>Expected Post-Condition</b>	The system state doesnt change, meaning Current Balance is not changed and no change in database
<b>Execution History</b>	[04/14/2018 — Shunyu Wang] Executed test successfully.
<b>Trace to Use Case</b>	UC-2

### 2.2.3 [TC-3] [Display Balance]

Table 13: TC-3.1 : Testing the Display Balance

<b>Test Case Number</b>	TC-3.1
<b>Description</b>	This test is used to show the balance is properly displaying the correct amount from the user's account.
<b>Input</b>	<ol style="list-style-type: none"><li>1. The user opens the application.</li><li>2. The user sees the balance displayed in the lower center area of the default application window.</li></ol>
<b>Expected Output</b>	<ol style="list-style-type: none"><li>1. The system will query the database.</li><li>2. The balance is displayed on the screen.</li></ol>
<b>Expected Post-Condition</b>	The application will display the balance in green if the amount in the account is positive, and in red if the account is negative.
<b>Execution History</b>	[04/14/18 — Sabrina D'Mello] Executed test successfully.
<b>Trace to Use Case</b>	UC-3



#### 2.2.4 [TC-4] [Clear Account]

Table 14: TC-4.1 : Testing Clear Account

<b>Test Case Number</b>	TC-4.1
<b>Description</b>	Ensure that all transaction records for the account are deleted after user clicks on “Clear Account”.
<b>Input</b>	<ol style="list-style-type: none"><li>1. The user makes at least one deposit or withdrawal transaction from the account (see UC-1 and UC-2).</li><li>2. The user clicks on “Clear Account”.</li></ol>
<b>Expected Output</b>	<ol style="list-style-type: none"><li>1. The account balance is reset to 0.00\$.</li><li>2. All transactions are removed from the account database.</li></ol>
<b>Expected Post-Condition</b>	The entire account is reset to a default state. If the system was already in a default state, no observable changes will take place.
<b>Execution History</b>	[04/14/2018 — Shunyu Wang] Executed test successfully.
<b>Trace to Use Case</b>	UC-1, UC-2, UC-4.

### 2.2.5 [TC-5] [Show History]

Table 15: TC-5.1 : Testing Show History

<b>Test Case Number</b>	TC-5.1
<b>Description</b>	This test case ensures the functionality of Show History, which shows a history of all deposit and withdrawal transactions. Additionally, information such as the type of transaction, the reason and dates should all be present.
<b>Input</b>	<ol style="list-style-type: none"><li>1. User clicks “Show History” button.</li></ol>
<b>Expected Output</b>	<ol style="list-style-type: none"><li>1. New GUI window opens.</li><li>2. All information for a given transaction is presented on a line.</li></ol>
<b>Expected Post-Condition</b>	Separate window opens.
<b>Execution History</b>	[04/15/2018 — Philip Michael] Executed test successfully.
<b>Trace to Use Case</b>	UC-5

### 2.2.6 [TC-6] [Sort Table]

Table 16: TC-6.1 : Testing Sort by Date

<b>Test Case Number</b>	TC-6.1
<b>Description</b>	This test case ensures that sorting by date functions properly in the Transaction History GUI window that is accessed through the “Show History” button on the landing GUI.
<b>Input</b>	<ol style="list-style-type: none"><li>1. User opens Transaction History</li><li>2. User clicks on “Date” column header</li></ol>
<b>Expected Output</b>	<ol style="list-style-type: none"><li>1. Transactions are sorted by descending date, if arrow is pointing up.</li><li>2. Transactions are sorted by date ascending, if arrow is pointed down.</li><li>3. If no arrow is shown, no sorting is performed.</li></ol>
<b>Expected Post-Condition</b>	The data is sorted according to the state it is in.
<b>Execution History</b>	[04/15/2018 — Philip Michael] Executed test succesfully.
<b>Trace to Use Case</b>	UC-5, UC-6

Table 17: TC-6.2 : Testing Sort by Transaction Type

<b>Test Case Number</b>	TC-6.2
<b>Description</b>	This test case ensures that sorting by transaction type functions properly in the Transaction History GUI window that is accessed through the “Show History” button on the landing GUI.
<b>Input</b>	<ol style="list-style-type: none"> <li>1. User opens Transaction History</li> <li>2. User clicks on “Transation Type” column header</li> </ol>
<b>Expected Output</b>	<ol style="list-style-type: none"> <li>1. Transactions are sorted by deposit first, if arrow is pointing up.</li> <li>2. Transactions are sorted by withdrawal first, if arrow is pointed down.</li> <li>3. If no arrow is shown, no sorting is performed.</li> </ol>
<b>Expected Post-Condition</b>	The data is sorted according to the state it is in.
<b>Execution History</b>	[04/15/2018 — Philip Michael] Executed test succesfully.
<b>Trace to Use Case</b>	UC-5, UC-6

Table 18: TC-6.3 : Testing Sort by Description

<b>Test Case Number</b>	TC-6.3
<b>Description</b>	This test case ensures that sorting by description functions properly in the Transaction History GUI window that is accessed through the “Show History” button on the landing GUI.
<b>Input</b>	<ol style="list-style-type: none"> <li>1. User opens Transaction History</li> <li>2. User clicks on “Description” column header</li> </ol>
<b>Expected Output</b>	<ol style="list-style-type: none"> <li>1. Transactions are sorted alphabetically, if arrow is pointing up.</li> <li>2. Transactions are sorted in reverse alphabetical order, if arrow is pointed down.</li> <li>3. If no arrow is shown, no sorting is performed.</li> </ol>
<b>Expected Post-Condition</b>	The data is sorted according to the state it is in.
<b>Execution History</b>	[04/15/2018 — Philip Michael] Executed test succesfully.
<b>Trace to Use Case</b>	UC-5, UC-6

Table 19: TC-6.4 : Testing Sort by Amount

<b>Test Case Number</b>	TC-6.4
<b>Description</b>	This test case ensures that sorting by amount functions properly in the Transaction History GUI window that is accessed through the “Show History” button on the landing GUI.
<b>Input</b>	<ol style="list-style-type: none"> <li>1. User opens Transaction History</li> <li>2. User clicks on “Amount” column header</li> </ol>
<b>Expected Output</b>	<ol style="list-style-type: none"> <li>1. Transactions are sorted by ascending amount, if arrow is pointing up.</li> <li>2. Transactions are sorted by descending amount, if arrow is pointed down.</li> <li>3. If no arrow is shown, no sorting is performed.</li> </ol>
<b>Expected Post-Condition</b>	The data is sorted according to the state it is in.
<b>Execution History</b>	[04/15/2018 — Philip Michael] Executed test succesfully.
<b>Trace to Use Case</b>	UC-5, UC-6

Table 20: TC-6.5 : Testing Sort by Withdrawal Type

<b>Test Case Number</b>	TC-6.5
<b>Description</b>	This test case ensures that sorting by withdrawal type functions properly in the Transaction History GUI window that is accessed through the “Show History” button on the landing GUI.
<b>Input</b>	<ol style="list-style-type: none"> <li>1. User opens Transaction History</li> <li>2. User clicks on “Type of Withdrawal” column header</li> </ol>
<b>Expected Output</b>	<ol style="list-style-type: none"> <li>1. Transactions are sorted alphabetically, if arrow is pointing up.</li> <li>2. Transactions are sorted in reverse alphabetical order, if arrow is pointed down.</li> <li>3. If no arrow is shown, no sorting is performed.</li> </ol>
<b>Expected Post-Condition</b>	The data is sorted according to the state it is in.
<b>Execution History</b>	[04/15/2018 — Philip Michael] Executed test succesfully.
<b>Trace to Use Case</b>	UC-5, UC-6

Table 21: TC-6.6 : Testing Sort by Deposit Type

<b>Test Case Number</b>	TC-6.6
<b>Description</b>	This test case ensures that sorting by deposit type functions properly in the Transaction History GUI window that is accessed through the “Show History” button on the landing GUI.
<b>Input</b>	<ol style="list-style-type: none"> <li>1. User opens Transaction History</li> <li>2. User clicks on “Type of Deposit” column header</li> </ol>
<b>Expected Output</b>	<ol style="list-style-type: none"> <li>1. Transactions are sorted alphabetically, if arrow is pointing up.</li> <li>2. Transactions are sorted in reverse alphabetical order, if arrow is pointed down.</li> <li>3. If no arrow is shown, no sorting is performed.</li> </ol>
<b>Expected Post-Condition</b>	The data is sorted according to the state it is in.
<b>Execution History</b>	[04/15/2018 — Philip Michael] Executed test successfully.
<b>Trace to Use Case</b>	UC-5, UC-6



### 2.2.7 [TC-7] [Export History]

Table 22: TC-7.1 : Testing Export History

<b>Test Case Number</b>	TC-7.1
<b>Description</b>	Verify that the user can download a .csv file of their transaction history.
<b>Input</b>	<ol style="list-style-type: none"><li>1. User performs at least one deposit or withdrawal transaction.</li><li>2. User clicks on “Export History” button in the Advanced Options</li></ol>
<b>Expected Output</b>	<ol style="list-style-type: none"><li>1. A .csv file is created and saved in a default location.</li></ol>
<b>Expected Post-Condition</b>	The data from the file is sorted according to the state it is in.
<b>Execution History</b>	[04/15/2018 — Sabrina D’Mello] Executed test succesfully.
<b>Trace to Use Case</b>	UC-6, UC-7

### 3 Unit Test Cases

#### 3.1 Display Balance Unit Testing

Table 23: initialBalanceTest()

<b>Test Case Number</b>	UT-1
<b>Description</b>	This unit test case is used to ensure that when the myMoney application is initially opened that it will display the current historical balance on the account.
<b>Input</b>	<ol style="list-style-type: none"><li>1. No input</li></ol>
<b>Expected Output</b>	<ol style="list-style-type: none"><li>1. Test was executed successfully</li></ol>
<b>Expected Post-Condition</b>	The system responds to the presence or absence of an input vector and outputs a success message.
<b>Execution History</b>	<ol style="list-style-type: none"><li>1. (04/15/2018) Sabrina DMello: Executed test successfully.</li><li>2. (04/6/201) Damian Kazior Executed test successfully.</li></ol>

Table 24: updateBalanceTest()

<b>Test Case Number</b>	UT-2
<b>Description</b>	This unit test case is used to ensure that the system updates the balance positively or negatively, depending on the transaction type.
<b>Input</b>	1. No Input
<b>Expected Output</b>	1. Test was executed successfully
<b>Expected Post-Condition</b>	The system responds to the presence or absence of an input vector and outputs a success message.
<b>Execution History</b>	<ul style="list-style-type: none"> <li>1. (04/15/2018) Sabrina DMello: Executed test successfully.</li> <li>2. (04/6/201) Damian Kazior Executed test successfully.</li> </ul>

Table 25: updateModelTest()

<b>Test Case Number</b>	UT-3
<b>Description</b>	This unit test case was created to ensure that the model updates once a change has been made to the view.
<b>Input</b>	<ol style="list-style-type: none"> <li>1. No Input</li> </ol>
<b>Expected Output</b>	<ol style="list-style-type: none"> <li>1. Test was executed successfully</li> </ol>
<b>Expected Post-Condition</b>	The system responds to the presence or absence of an input vector and outputs a success message.
<b>Execution History</b>	<ol style="list-style-type: none"> <li>1. (04/15/2018) Sabrina DMello: Executed test successfully.</li> <li>2. (04/6/201) Damian Kazior Executed test successfully.</li> </ol>

### 3.2 Data Unit Testing

Table 26: CreateDummyDeposit()

<b>Test Case Number</b>	UT-4
<b>Description</b>	This unit test case was to ensure that the database fields in the deposit money model is being set and the amount is being added it correctly.
<b>Input</b>	<ol style="list-style-type: none"><li>1. Integer Amount</li></ol>
<b>Expected Output</b>	<ol style="list-style-type: none"><li>1. Test was executed successfully</li></ol>
<b>Expected Post-Condition</b>	The system responds to the presence or absence of an input vector and outputs a success message.
<b>Execution History</b>	<ol style="list-style-type: none"><li>1. (04/15/2018) Sabrina DMello: Executed test successfully.</li><li>2. (04/6/201) Damian Kazior Executed test successfully.</li></ol>

Table 27: CreateDummyWithdrawal()

<b>Test Case Number</b>	UT-5
<b>Description</b>	This unit test case was to ensure that the database fields in the withdraw money model is being set and the amount is being deducted correctly.
<b>Input</b>	1. Integer Amount
<b>Expected Output</b>	1. Test was executed successfully
<b>Expected Post-Condition</b>	The system responds to the presence or absence of an input vector and outputs a success message.
<b>Execution History</b>	1. (04/15/2018) Sabrina DMello: Executed test successfully. 2. (04/6/201) Damian Kazior Executed test successfully.

## 4 Test Results

In this section we will be outlining the test results for both of our main test cases while providing in depth information regarding regular, special, and boundary cases.

### 4.1 Deposit Money Test

1. Expected Output (when “Done” button is clicked, and tests are implemented in order

Regular Cases

- Value 100: Works as expected.  
Account: 100.00 Current Balance: 100.00
- Value 200.78: Works as expected.  
Account: 300.78 Current Balance: 300.78

Special Cases

- Value -50: Does not work as expected. Subtracts 50 from the account and from the “Current Balance” field.  
No error message is displayed.
- Value “Hello”: Works as expected. An error message appears.  
Error: Value entered must be a positive number.
- Value: 3.2222: Works as expected.  
Account: 304.0022 Current Balance: 304.00
- Value -30.55: Subtracts 30.55 from the account and from the “Current Balance” field.  
No error message is displayed.

## 2. Type of Deposit” input text field

### Boundary Cases

- Pass nothing (leave the field empty). Works as expected. The deposit concludes with default reason.

### Regular Cases

- Pass a string “Cash Entry”. Works as expected and provides deposit reason.
- Pass a string containing a number “Money Transfer #3”. Works as expected and provides deposit reason.

## 3. “(optional) Transaction Description” input text field

### Boundary Cases

- Pass nothing (leave the field empty). Works as expected with no description.

### Regular Cases

- Pass a string “Money found in couch”. Works as expected and provides description.
- Pass a string containing a number “Sold Xbox 360”. Works as expected and provides description.

## 4. “Done” button

All tests above satisfy this case as they require the use of the Done button. Please refer to sections 1 and 2.

## 5. “Cancel” button

### Regular Cases

- Transaction is not recorded. That is:
  - Account (database) does not contain a record corresponding to the input. Works as expected.
  - “Current Balance field” is unchanged. Works as expected.

## 4.2 Withdraw Money Test

### 1. Expected Output (when “Done” button is clicked, and tests are implemented in order)

#### Boundary Cases:

- Passing a 0 or a 0.00: Works as expected, no actual change is done to the balance.

#### Regular Cases:

- Value of 100: Works as expected, withdrawal is performed.
- Value of 200.78: Works as expected. Withdrawal is performed.

#### Special Cases:

- Value of -50: Does not work as expected. It adds 50 to the account and to the “Current Balance” field. No error message is shown.
- Value “Hello”: Works as expected. An error message is shown.
- Value 3.2222: Works as expected by being rounded to the nearest hundredth when performing the withdrawal.

### 2. Expected Output (when “Show History” is clicked).

#### Boundary Cases:

- “” is under the “TYPE OF WITHDRAW”: Works as expected. It is left blank.

#### Regular Cases:

- “bill” is under the “TYPE OF WITHDRAW”: Works as expected. Bill is shown as the type of the withdrawal.



- “check” is under the “TYPE OF WITHDRAW”: Works as expected. Check is shown as the type of the withdrawal.

3. (optional) “Transaction Description” input text field

Boundary Cases:

- “” is under the “DESCRIPTION”: Works as expected. It is left blank.

Regular Cases:

- “Pay electricity bill of June” is under the “DESCRIPTION”: Works as expected. Shows this as the description for the withdrawal.
- “Pay off the debt owed to Jack” is under the “DESCRIPTION”: Works as expected. Shows this as the description for the withdrawal.

4. “Done” button

Expected output is relative to the outlined uses in the above cases. They all require the use of the “Done” button, therefore this button is tested through those same tests.

5. “Cancel” button

Regular Cases:

- Transaction is not recorded. That is:
  - Account (database) does not contain a record corresponding to input data: Works as expected.
  - “Current Balance” field is unchanged: Works as expected. No changes occur.

## 5 References

We obtained a test document sample that we used as a reference: Montrealopoly, Master Test Plan, from: <https://users.encs.concordia.ca/paquet/wiki/images/3/35/Phase3final.pdf>

## A Description of Input Files

Table 28: Input Files

File Name	File Extension	File Description	Input Description
mymoneyappdb	.db	<p>Database file used for SQLite. Contains all the database information locally, such as the different tables and their contents. It contains the following four Tables:</p> <ul style="list-style-type: none"> <li>• Withdraw_Money</li> <li>• Deposit_money</li> <li>• Display_Balance</li> <li>• sqlite_sequence</li> </ul>	<p>The application will read from this file every time it is run to get all the Database information and display them on the GUI.</p>
Transaction_History	.csv	<p>Excel file used to keep a history of all transactions executed within the application, as well as all the information relating to those transactions. It contains the following six columns:</p> <ul style="list-style-type: none"> <li>• Date</li> <li>• Transaction Type</li> </ul>	<p>The application will read from this file every time it tries to access the history of all transactions to be able to display them on the GUI.</p>

## B Description of Output Files

Table 29: Input Files

File Name	File Extension	File Description	Input Description
mymoneyappdb	.db	<p>Database file used for SQLite. Contains all the database information locally, such as the different tables and their contents. It contains the following four Tables:</p> <ul style="list-style-type: none"> <li>• Withdraw_Money</li> <li>• Deposit_money</li> <li>• Display_Balance</li> <li>• sqlite_sequence</li> </ul>	<p>Everytime a new withdraw or deposit action happens, the application will write to this file to add new entries under their respective tables.</p>
Transaction_History	.csv	<p>Excel file used to keep a history of all transactions executed within the application, as well as all the information relating to those transactions. It contains the following six columns:</p> <ul style="list-style-type: none"> <li>• Date</li> <li>• Transaction Type</li> </ul>	<p>Everytime a new withdraw or deposit action happens, the application will write to this file to add new rows containing all the information of the transaction.</p>

## C Figures

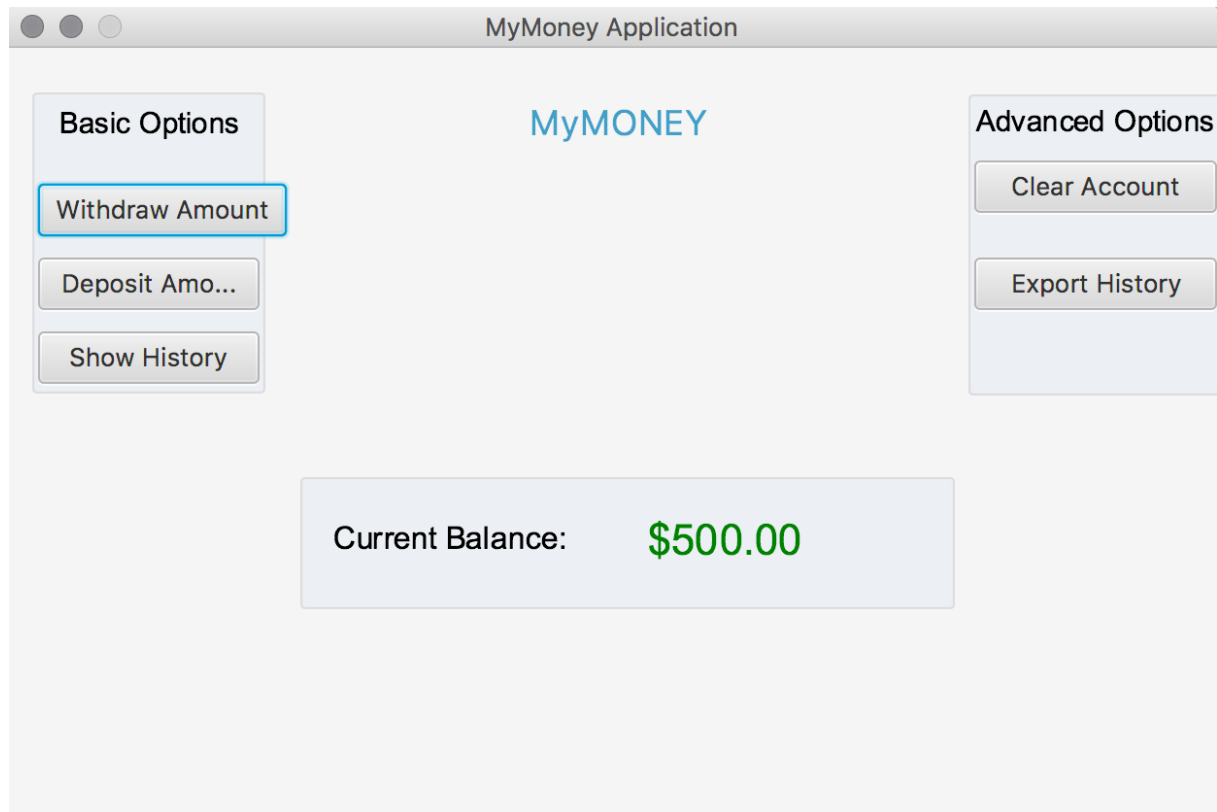
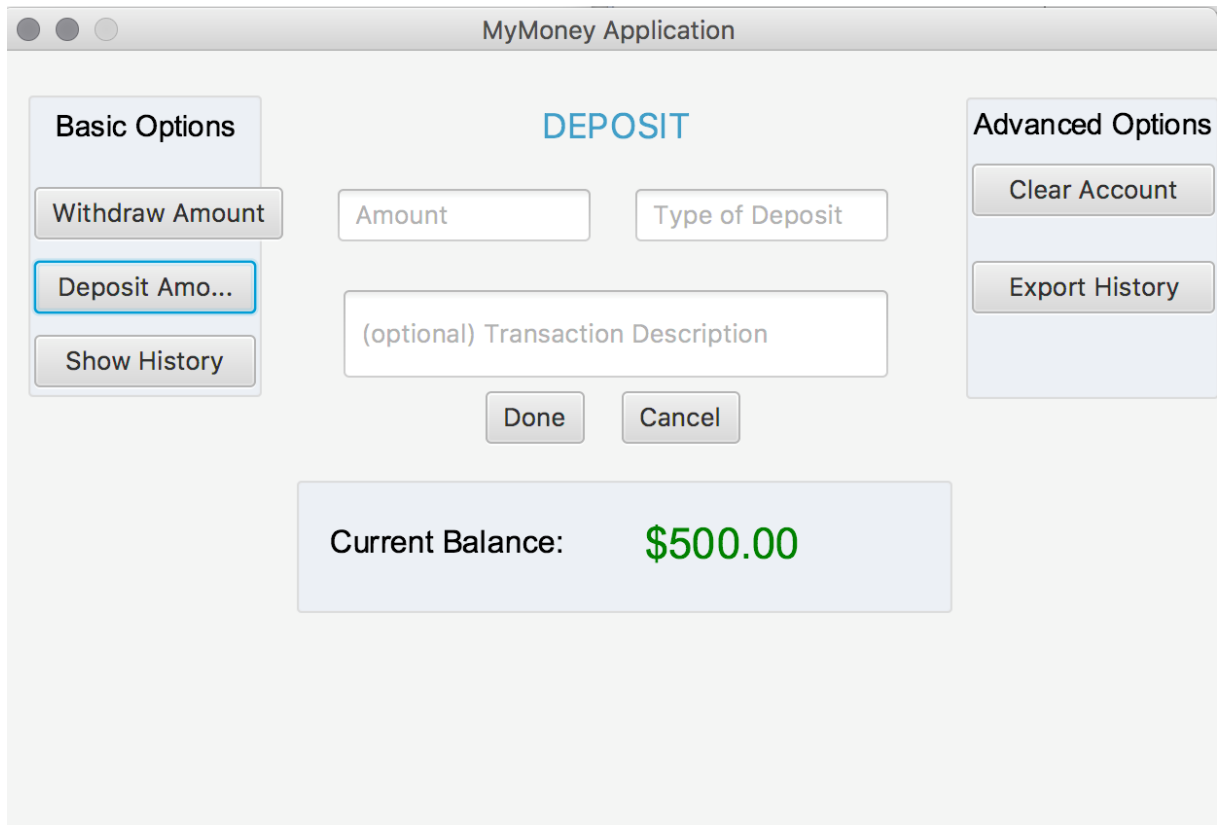


Figure 1: The user opens myMoney App.



The image shows a software window titled "MyMoney Application" with a "DEPOSIT" dialog box. The dialog is organized into three main sections: "Basic Options" on the left, a central input area, and "Advanced Options" on the right. The "Basic Options" section contains three buttons: "Withdraw Amount", "Deposit Amo..." (which is highlighted with a blue border), and "Show History". The central area features a title "DEPOSIT" in blue, followed by two input fields labeled "Amount" and "Type of Deposit", and a larger text area labeled "(optional) Transaction Description". Below these are "Done" and "Cancel" buttons. The "Advanced Options" section on the right contains two buttons: "Clear Account" and "Export History". At the bottom of the dialog, a light blue box displays the "Current Balance: \$500.00" in green text.

Section	Field/Option	Value/Label
Basic Options	Withdraw Amount	Button
	Deposit Amo...	Button (highlighted)
	Show History	Button
Central Input	Amount	Input Field
	Type of Deposit	Input Field
Advanced Options	Clear Account	Button
	Export History	Button
Current Balance		\$500.00

Figure 2: The fields mentioned in the Deposit Test Case section are shown above.