Iteration 3 - Test Document

Team PA-PI-a

8 April 2018

Table 1: Team

Name	ID Number
Melanie Taing	40009850
Laurie Gagnon	22943433
Wayne Yiel Leung	26586988
Jordan Rutty	27300107
Michael Foo	40000225
Pierre-Andre Leger	40004010
Colin Greczkowski	40001600

Contents

List of Figures

1 Introduction

The purpose of this document is to gather all information necessary for testing of the MyMoney application. This document describes the testing approach and overall framework that will be used to test the MyMoney application.

The following pages will identify the requirements that will be tested, the testing strategy used, the test cases and their results, and the description of input files.

2 Test Plan

We will be performing system level based on the use case scenarios and unit testing on select classes. We are performing black box testing for system level testing, as we are more concerned with the whole system working as a whole, and not necessarily "how" it works. Unit testing will focus on testing specific methods within *TransactionRepository*, *RepositoryContainer*, *AccountRepository*, *BudgetRepository* as well as the controllers: *TransactionController*, *AccountController*, *BudgetController*.

The following functional requirements will be tested:

- Add Account
- Update Account
- Delete Account
- Add Transaction
- Import Transaction
- Update Transaction
- Delete Transaction
- View Account Transactions
- Add Budget
- Update Budget
- Delete Budget
- Apply Transaction to Budget
- View Budget Transactions

The unit tests will include the following classes:

- TransactionRepository
  - Function saveItem
  - Function deleteItem
- RepositoryContainer
  - Function saveItem
- AccountRepository
  - Function saveItem
  - Function deleteItem
- BudgetRepository
  - Function saveItem
  - Function deleteItem
- TransactionController
- AccountController
- BudgetController

3 System Level Test Cases

3.1 Test Case 1 - Add Account

Purpose
The purpose of the test is to verify the user is able to add a bank account into the application's database. It satisfies the requirement of the user being able to create a bank account.

Input Specification
The application displays a graphical user interface on the screen. Optionally, it shows a list of pre-existing bank accounts in the window's top-right corner. MyMoney accepts any kind of characters and of any length as input in the *Bank* and *Nickname* fields while the *Balance* field accepts non-negative integers. The *Bank*, *Nickname* and *Balance* fields cannot be empty. The user presses *Add* on the interface to add the account.

Expected Output
The application displays the window. A created bank account with the information the user entered now exists in the application.

Traces to Use Cases
This test case satisfies the main scenario of use case 1 - *AddAccount*.

3.1.1 Test-Case 1-1 Alternate Flow 1 - Empty Field

A user omits input in any of the required fields (*Bank*, *Nickname* or *Balance*) If any field is left empty, an Input Error window pops up, asking the user to complete the missing fields. User must press "OK" and fill the required fields.

3.1.2 Test Case 1-2 Alternate Flow 2 - Non-Numerical Balance

A user enters a non-numerical value in the *Balance* field. An Input Error window pops up, asking the user to complete the field with a number.

3.1.3 Test Case 1-3 Alternate Flow 3 - Negative Balance

A user enters a negative value in the *Balance* field. An Input Error window pops up, asking the user to complete the field with a number.

3.2 Test Case 2 - Update Account

Purpose
The test verifies that for an existing bank account its bank name, nickname and balance can be modified. This satisfies the requirement that the user is able to adjust account information in case of an account transfer to another financial institution.

Input Specification
MyMoney displays a graphical user interface on the screen. For this operation, MyMoney accepts any kind of characters and of any length as input in the *Bank* and *Nickname* fields while the *Balance* field accepts non-negative integers. The *Bank*, *Nickname* and *Balance* fields cannot be empty. The user presses the *Update* to update the account information.

Expected Output
The application displays the window. The system displays updated information of the bank account in the top-right window.

Traces to Use Cases
This satisfies the main scenario of use case 2 - *UpdateAccount*.

3.2.1 Test Case 2-1 Alternate Flow 1 - Empty Field

When updating a field, the user leaves the field blank. If any field is left empty, an Input Error window pops up, asking the user to complete the missing fields. User must press "OK" and fill the required fields.

3.2.2 Test Case 2-2 Alternate Flow 2 - Non-Numerical Balance

When updating a field, the user enters a non-numerical value in the *Balance* field. An Input Error window pops up, asking the user to complete the field with a number.

3.2.3 Test Case 2-3 Alternate Flow 3 - Negative Balance

When updating a field, the user enters a negative value in the *Balance* field. An Input Error window pops up, asking the user to complete the field with a number.

3.3 Test Case 3 - Delete Account

Purpose
This test verifies the user is able to delete their own account. This satisfies the requirement that the user is able to remove their account when their is no longer associated with a bank. This will also delete the associated transactions and budgets with the account as well.

Input Specification
MyMoney displays a graphical user interface on the screen. For this operation, the user selects their account with the mouse and presses *Delete*.

Expected Output
The application displays the window. Also it display a list of accounts in the top-right corner except the one that was deleted.

Traces to Use Cases
This satisfies the main scenario of use case 3 - *DeleteAccount*.

3.3.1 Test Case 3-1 Alternate Flow 1 - No Account

A user attempts to delete an account by pressing the *Delete* button. If there is no account to delete, nothing happens in the application.

3.4 Test Case 4 - Add Transaction

Purpose
This test verifies the user is able to add a transaction into their existing account. This satisfies the requirement that the user can complete an addition of a transaction into their account.

Input Specification
MyMoney displays a graphical user interface on the screen. For this operation, MyMoney accepts a transaction *type* - withdraw or deposit, a *date* which is selected via the date picker, an *Amount*, a integer, a *Budget* which is chosen from a drop-down list, and a *description* - a string composed of any characters and of non-negative length. To register the action, the user presses *Add* located in the bottom left of the window.

Expected Output
The application displays the window. The transaction is added to the account. If the user selects his account in the top-right corner of the window then the bottom-right window displays the newly created transaction. In addition, this will modify the account balance and also the budget balance, if the transaction was tagged.

Traces to Use Cases
This satisfies the main scenario of use case 4 - *AddTransaction*.

3.4.1 Test Case 4-1 Alternate Flow 1 - Empty Amount Field

When adding an amount for the transaction, the user leaves the field blank. An Input Error window pops up, asking the user to complete the field. User must press "OK" and fill the required field.

3.4.2 Test Case 4-2 Alternate Flow 2 - Non-Numerical Balance

When adding an amount for the transaction, the user enters a non-numerical value in the *Amount* field. An Input Error window pops up, asking the user to complete the field with a number.

3.4.3 Test Case 4-3 Alternate Flow 3 - Negative Balance

When adding an amount for the transaction, the user enters a negative value in the *Amount* field. An Input Error window pops up, asking the user to complete the field with a number.

3.5 Test Case 5 - Import Transaction

Purpose
This test verifies the user is able to import a transaction into their existing account. This satisfies the requirement that the user can import a transaction into their account.

Input Specification
MyMoney displays a graphical user interface on the screen. For this operation, the user selects their account with the mouse, clicks the import button, and chooses the .csv file.

Expected Output
The transaction is added to the user's account. The bottom-right corner displays the transaction.

Traces to Use Cases
This satisfies use case 5 - *ImportTransactions*

3.5.1 Test Case 5-1 Alternate Flow 1 - Invalid Date Format

The user attempts to import a file that has an invalid date format found inside. An error window pops up, and displays the message "Invalid date format found in import file at line " with the line number. No transactions are imported.

3.5.2 Test Case 5-2 Alternate Flow 2 - File Not Found

The user attempts to import a file that is not found in the system. An error window pops up and displays "Error."

3.5.3 Test Case 5-3 Alternate Flow 3 - Invalid Input Format

The user attempts to import a file that contains an invalid numerical value. An error window pops up and displays "Invalid input for amount column."

3.6 Test Case 6 - Update Transaction

Purpose
This test verifies the user is able to update an existing transaction in their account. This satisfies the requirement that the user can change information of a transaction in their account.

Input Specification
MyMoney displays a graphical user interface on the screen. For this operation, the user selects their account with the mouse, chose the appropriate transaction to modify in the bottom-right window. The user can modify the transaction fields on the transactions pane. In the pane, transaction *type* - withdraw or deposit, a *date* which is selected via the date picker, an *Amount*, a integer, a *Budget* which is chosen from a drop-down list, and a *description* - a string composed of any characters and of non-negative length. The register the action, the user presses *Add* located in the bottom left of the window.

Expected Output
The application displays the window. The fields in the transaction are updated. If selected, the bottom-right corner displays the updated transaction. The associated budget and account balance will be updated accordingly.

Traces to Use Cases
This satisfies use case 6 - *UpdateTransactions*

3.6.1 Test Case 6-1 Alternate Flow 1 - Empty Amount Field

When updating a transaction, the user leaves the field blank. An Input Error window pops up, asking the user to complete the field. User must press "OK" and fill the required field.

3.6.2 Test Case 6-2 Alternate Flow 2 - Non-Numerical Balance

When updating a transaction, the user enters a non-numerical value in the *Amount* field. An Input Error window pops up, asking the user to complete the field with a number.

3.6.3 Test Case 6-3 Alternate Flow 3 - Negative Balance

When updating a transaction, the user enters a negative value in the *Amount* field. An Input Error window pops up, asking the user to complete the field with a number.

3.7 Test Case 7 - Delete Transaction

Purpose
This test verifies the user is able to delete an existing transaction in their account. This satisfies the requirement that the user can remove a transaction from their account.

Input Specification
MyMoney displays a graphical user interface on the screen. For this operation, the user selects their account with the mouse, chose the appropriate transaction to remove in the bottom-right window. The user presses *Delete* in the Transactions pane.

Expected Output
The application displays the window. The transaction does not show in the bottom-right window.

Traces to Use Cases
This satisfies use case - 7 *DeleteTransactions*

3.7.1 Test Case 7-1 Alternate Flow 1 - No Transaction

A user attempts to delete a transaction by pressing the *Delete* button. If there is no transaction to delete, nothing happens in the application.

3.8 Test Case 8 - View Account Transactions

Purpose
This test verifies that the user is able to view the selected account's transactions.

Input Specification
MyMoney displays a graphical user interface on the screen. For this operation, the user selects their account with the mouse.

Expected Output
The application displays the window. The transaction window is visible when the account is selected. The selected account's transactions shows up in the transaction table.

Traces to Use Cases
This satisfies use case 8 - *ViewTransactions*

3.8.1 Test Case 8-1 Alternate Flow 1 - No Transactions

There are no transactions to view.

3.9 Test Case 9 - Add Budget

Purpose
This test verifies that the user is able to add a budget to their account.

Input Specification
The user must select the *Budget* tab. The budget takes a Name - string of any length and of any characters and an Amount - a non-negative integer.

Expected Output
The application displays the window. The budget is added to the user's account and is displayed in the top-right window under the budget tab.

Traces to Use Cases
This satisfies use case 9 - *AddBudget*

3.9.1 Test Case 9-1 Alternate Flow 1 - Empty Field

When adding a budget, the user leaves the *Name* or *Amount* fields blank. An Input Error window pops up, asking the user to complete the missing field. User must press "OK" and fill the required field.

3.9.2 Test Case 9-2 Alternate Flow 2 - Non-Numerical Amount

When adding a budget, the user enters a non-numerical value in the *Amount* field. An Input Error window pops up, asking the user to complete the field with a number.

3.9.3 Test Case 9-3 Alternate Flow 3 - Negative Amount

When adding a budget, the user enters a negative value in the *Amount* field. An Input Error window pops up, asking the user to complete the field with a number.

3.10 Test Case 10 - Update Budget

Purpose
This test verifies that the user is able to modify a previously added budget.

Input Specification
The budget takes a Name - string of any length and of any characters and an Amount - an integer.

Expected Output
The application displays the window. The updated budget is added to the user's account and is displayed in the top-right window under the budget tab.

Traces to Use Cases
This satisfies use case 10 - *UpdateBudget*

3.10.1 Test Case 10-1 Alternate Flow 1 - Empty Field

When updating a budget, the user leaves the *Name* or *Amount* fields blank. An Input Error window pops up, asking the user to complete the missing field. User must press "OK" and fill the required field.

3.10.2 Test Case 10-2 Alternate Flow 2 - Non-Numerical Amount

When updating a budget, the user enters a non-numerical value in the *Amount* field. An Input Error window pops up, asking the user to complete the field with a number.

3.10.3 Test Case 10-3 Alternate Flow 3 - Negative Amount

When updating a budget, the user enters a negative value in the *Amount* field. An Input Error window pops up, asking the user to complete the field with a number.

3.11 Test Case 11 - Delete Budget

Purpose
The purpose of the test is to verify that the user is able to delete a budget from their account.

Input Specification
The app's main GUI contains a tab called Budget. A list of existing budgets is visible on the right. By clicking on a budget in this list, and then clicking the Delete button in the left section of the GUI, the selected budget is marked for deletion.

Expected Output
The selected budget is deleted. The list of budgets on the right hand side of the GUI is refreshed, with the deleted budget no longer present.

Traces to Use Cases
This satisfies use case 11 - *DeleteBudget*

3.11.1 Test Case 11-1 Alternate Flow 1 - No Budget

A user attempts to delete a budget by pressing the *Delete* button. If there is no budget to delete, nothing happens in the application.

3.12 Test Case 12 - Apply Transaction to Budget

Purpose This test verifies that the budget balance is updated according to transactions added.

Input Specification
MyMoney displays a graphical interface on the screen. The user has Accounts, Transactions and Budgets already created. Under the Budget tab, the user selects a Budget, and selects the month and year to display transactions under the selected Budget.

Expected Output
The application displays the main window. The selected budget's balance is updated according to the transactions.

Traces to Use Cases
This satisfies use case 12 - *ApplyTransactionToBudget*

3.12.1 Test Case 12-1 Alternate Flow 1 - No Transactions

There are no transactions to view.

3.13 Test Case 13 - View Budget Transactions

Purpose
This test verifies that the user is able to view the selected budget's transactions across all accounts.

Input Specification
MyMoney displays a graphical user interface on the screen. The user has Accounts, Transactions and Budgets already created. Under the Budget tab, the user selects a Budget, and selects the month and year to display transactions under the selected Budget.

Expected Output
The application displays the main window. The selected budget's transactions are listed in the Budget Transactions table.

Traces to Use Cases
This satisfies use case 13 - *ViewBudgetTransactions*

3.13.1 Test Case 13-1 Alternate Flow 1 - No Transactions

There are no budget transactions to view.

3.13.2 Test Case 13-2 Alternate Flow 2 - No Budget

There is no budget to view.

4 Unit Test cases

4.1 Unit Test Case 1 - Transaction: saveItem

Table 2: UT-1

Test Case Number	UT-1
Test Case Description	This test case is used to ensure that transactions are properly saved or updated to their repository
Input	1. A Transaction object populated with generic data 2. A second Transaction object with the ID of the first one. 3. A test transaction database.
Expected Output	1. Transaction details are printed to console.
Expected Post-Conditions	A transaction database is created and a transaction is inserted. The balance of this transaction is then updated to a new value.
Execution History	1. 04/03/2018 — Colin Greczkowski — Test pass.

4.2 Unit Test Case 2 - Transaction: deleteItem

Table 3: UT-2

Test Case Number	UT-2
Test Case Description	This test verifies that the deleteItem method works as intended, and deletes a Transaction record for a given ID
Input	1. A generic account ID 2. A Transaction object populated with generic data, associated to the generic account. 3. A test transaction database.
Expected Output	1. "Delete Transaction 1"
Expected Post-Conditions	The test transaction database should be empty.
Execution History	1. 04/07/2018 — Colin Greczkowski — Test pass.

4.3 Unit Test Case 3 - Transaction: deleteItem (all items)

Table 4: UT-3

Test Case Number	UT-3
Test Case Description	This test case is used to make sure all Transactions associated to an account are properly purged from the repository.
Input	1. A generic account ID 2. Two Transaction objects populated with generic data, associated to the generic account. 3. A test transaction database.
Expected Output	1. "Delete Transaction 1" 2. "Delete Transaction 2"
Expected Post-Conditions	The test transaction database does not contain the two transactions that had the generic account ID.
Execution History	1. 04/03/2018 — Colin Greczkowski — Test pass.

4.4 Unit Test Case 4 - RepositoryContainer: saveItem (all types)

Table 5: UT-4

Test Case Number	UT-4
Test Case Description	This tests the RepositoryContainer's ability to save a variety of types of objects (Transactions, Accounts, Budgets).
Input	1. Test Transaction, Budget and Account Databases 2. A test Transaction 3. A test Account 4. A test Budget
Expected Output	1. The test transaction's details are printed to console.
Expected Post-Conditions	The account,transaction and budget items are saved to their respective test databases. Balances are updated correctly.
Execution History	1. 04/07/2018 — Colin Greczkowski — Test pass. 2. 04/11/2018 — Michael Foo — Test fail. 3. 04/11/2018 — Melanie Taing — Test pass.

4.5 Unit Test Case 5 - AccountRepository: saveItem, deleteItem

Table 6: UT-5

Test Case Number	UT-5
Test Case Description	This test case is used to ensure created accounts are saved in the database. Also it verifies that accounts can be deleted from it.
Input	1. An account database 2. An account repository 3. An account with non-null values for *nickname*, *bankName* and a non-negative value for *balance*
Expected Output	1. Tuples in Account repository test before delete: 2 2. Tuples in Account repository test after delete: 1 3. Current items loaded in repo:1 4. 1
Expected Post-Conditions	The system has a single account in the account database.
Execution History	1. 04/07/2018 — Wayne Yiel Leung — Test fail. 2. 04/12/2018 — Melanie Taing — Test pass.

4.6 Unit Test Case 6 - BudgetRepository: saveItem

Table 7: UT-6

Test Case Number	UT-6
Test Case Description	This test case verifies that the saveItem method functions properly and saves the Budget object to the selected account's BudgetRepository.
Input	1. A Budget object populated with generic data 2. A test Account database populated with generic transactions 3. A test Budget database
Expected Output	1. Test budget added into repo: (Name, Amount, Balance) = (Test Budget, 100, 1000) 2. Budget amount before update: 100 3. Budget amount after update: 10000
Expected Post-Conditions	The test budget is added to the budget repository. The test budget's amount is updated to 1000.
Execution History	1. 04/08/2018 — Melanie Taing — Test fail 2. 04/10/2018 — Melanie Taing — Test pass

4.7 Unit Test Case 7 - BudgetRepository: deleteItem

Table 8: UT-7

Test Case Number	UT-7
Test Case Description	This test case verifies that the deleteItem method functions properly and deletes the Budget record in the repository for a given account ID.
Input	1. A generic account ID 2. A Budget object populared with generic data 3. A test Budget database
Expected Output	1. Tuples in Budget repository test before delete: 3 2. Tuples in Budget repository test after delete: 2 3. Current items loaded in repo: 2
Expected Post-Conditions	The test budget repository should contain one less budget.
Execution History	1. 04/08/2018 — Melanie Taing — Test fail 2. 04/10/2018 — Melanie Taing — Test pass

4.8 Unit Test Case 8 - AccountController: addOrUpdateAccount, deleteAccount

Table 9: UT-8

Test Case Number	UT-8
Test Case Description	This test verifies that an user is able add or update, or remove their account.
Input	1. An account with accountId, bankName, nickname, balance
Expected Output	1. The account is deleted from the database.
Expected Post-Conditions	There is one less account in the database.
Execution History	1. 04/11/2018 — Wayne Yiel Leung — Test pass

4.9 Unit Test Case 9 - TransactionController: addOrUpdateTransaction, deleteTransaction

Table 10: UT-9

Test Case Number	UT-9
Test Case Description	This test verifies that an user is able add or update, or remove a transaction from their account.
Input	1. A transaction with transactionId, accountId, BudgetId, type, date, amount and description
Expected Output	1. The transaction is deleted from the database.
Expected Post-Conditions	There is one less transaction in the database.
Execution History	1. 04/11/2018 — Wayne Yiel Leung — Test pass

4.10 Unit Test Case 10 - BudgetController testAddOrUpdateBudget

Table 11: UT-10

Test Case Number	UT-10
Test Case Description	This test checks that new budgets can be added, and existing budgets can be updated.
Input	1. test Model 2. test Controller 3. test Budget
Expected Output	1. No output
Expected Post-Conditions	Budget is properly saved.
Execution History	1. 04/12/2018 - Colin Greczkowski - Test executed successfully.

4.11 Unit Test Case 11 - BudgetController testDeleteAccount

Table 12: UT-11

Test Case Number	UT-11
Test Case Description	This test ensures that, for a given a budget ID, the associated budget is deleted.
Input	1. test Controller 2. test Budget
Expected Output	1. No output
Expected Post-Conditions	The test budget is deleted from the repository.
Execution History	1. 04/12/2018 - Colin Greczkowski - Test executed successfully.

4.12 Test Case Templates

4.12.1 System Level Test Case Template

Purpose
State the purpose of the test. Indicate which requirement and which aspect of that requirement is being tested.

Input Specification
State the context for the test in terms of system state. State the input test data. You may need to mention operations invoked as well as data for the operation. You can cross-reference to actual file data specified in an appendix.

Expected Output
State the expected system response and output. You can cross-reference to actual file data specified in an appendix.

Traces to Use Cases
State which requirements (at the level of use case and scenario) are tested by this test case.

4.12.2 Unit Test Case Template

Table 13: UT-X

Test Case Number	UT-X
Test Case Description	
Input	1.
Expected Output	1.
Expected Post-Conditions	
Execution History	1. mm/dd/yyyy — Tester's name — Execution result

5 Test Results

5.1 Requirement Test Results

Table 14: Requirement Test Results

Requirement	**Test Passed**
Account	100%
Transaction	100%
Budget	100%

5.2 System Level Tests

Table 15: System Test Case Results

Test Case	**Test Name**	**Result**
1	Add Account	Pass
2	Update Account	Pass
3	Delete Account	Pass
4	Add Transaction	Pass
5	Import Transaction	Pass
6	Update Transaction	Pass
7	Delete Transaction	Pass
8	View Account Transactions	Pass
9	Add Budget	Pass
10	Update Budget	Pass
11	Delete Budget	Pass
12	Apply Transaction to Budget	Pass
13	View Budget Transactions	Pass

5.3 Unit Level Tests

Table 16: Unit Test Case Results

Unit Test Case	Test Name	Result
1	Transaction: saveItem	Pass
2	Transaction: deleteItem	Pass
3	Transaction: deleteItem (all items)	Pass
4	RepositoryContainer: saveItem (all types)	Pass
5	AccountRepository: saveItem, deleteItem	Pass
6	BudgetRepository: saveItem	Pass
7	BudgetRepository: deleteItem	Pass
8	AccountController	Pass
9	TransactionController	Pass
10	BudgetController	Pass

6 References

Sinnig, D., "Introduction to Software Testing" (Current April 8, 2018)
Larman, C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd edition, Prentice-Hall, 2005.

7 Addendum

Figure 1: Updated use case diagram