# 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 My-Money 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

*Describe what forms of testing you plan to do (unit, subsystem, integration), describe briefly the schedule and resources for testing, and how you designed your test cases.*

*Indicate which qualities (from requirements) were tested and which qualities were not tested.*

## 2.1 System Level Test Cases

*All test cases for testing at the system level.*

**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.

### 2.1.1 Test Case 6.1

**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* 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*.

### 2.1.2 Test Case 6.2

**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* 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*.

### 2.1.3 Test Case 6.3

**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.

**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*.

### 2.1.4 Test Case 6.4

**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.

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

### 2.1.5 Test Case 6.5

**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, chose 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*

### 2.1.6 Test Case 6.6

**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.

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

### 2.1.7 Test Case 6.7

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

### 2.1.8 Test Case 6.8

**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. The selected account's transactions shows up in the transaction table.

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

**Table 2:** Template Test Case

| | |
|---|---|
| **Test Case Number** | UT-1 |
| **Test Case Description** | This test case is used to ensure the generated puzzle board has the same dimensions as the input width and height |
| **Input** | 1. None - Default 8-8 board size<br>2. Width/height from "input.txt" file. |
| **Expected Output** | 1. "OK" - Test executed successfully. |
| **Expected Post-Conditions** | The system responds to the presence or absence of the input vector and outputs a success message upon test successful completion in "output.txt", along with a time-stamp containing the test's execution time and date. |
| **Execution History** | 1. 05/04/2018 — Tester's name — Executed test successfully. |

## 2.2 Subsystem Level Test Cases

*All test cases for testing at the subsystem level.*

*One subsection per subsystem*

### 2.2.1 Subsystem X

## 2.3 Unit Test cases

*All test cases for testing at the unit level.*

*One subsection per unit*

### 2.3.1 Unit Test Case 1

**Table 3:** 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<br>2. A second Transaction object with the ID of the first one.<br>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 — Executed test successfully. |

### 2.3.2 Unit Test Case 2

**Table 4:** 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<br>2. A Transaction object populated with generic data, associated to the generic account.<br>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 — Executed test successfully. |

### 2.3.3 Unit Test Case 3

**Table 5:** 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<br>2. Two Transaction objects populated with generic data, associated to the generic account.<br>3. A test transaction database. |
| **Expected Output** | 1. "Delete Transaction 1"<br>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 — Executed test successfully. |

### 2.3.4 Unit Test Case 4

**Table 6:** 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<br>2. A test Transaction<br>3. A test Account<br>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 — Executed test successfully. |

### 2.3.5 Unit Test Case 5

**Table 7:** UT-5

| | |
|---|---|
| **Test Case Number** | UT-5 |
| **Test Case Description** | TO BE COMPLETED: BudgetContainer test |
| **Input** | 1. A Budget object populated with generic data<br>2. A test Account database with transactions<br>3. A test Budget database |
| **Expected Output** | 1. The test budget's details are printed to console. |
| **Expected Post-Conditions** | A budget database is created and a budget is inserted. The recorded budget amount is updated according to transactions made across all accounts for that budget. |
| **Execution History** | 1. 04/08/2018 — Melanie Taing — Not executed |

**Table 8:** UT-6

| | |
|---|---|
| **Test Case Number** | UT-6 |
| **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<br>2. An account repository<br>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<br>2. Tuples in Account repository test after delete: 1<br>3. Current items loaded in repo:1<br>4. 1 |
| **Expected Post-Conditions** | The system has a single account in the account database. |
| **Execution History** | 1. 04/07/2018 — Wayne Yiel Leung — Executed test failed. |

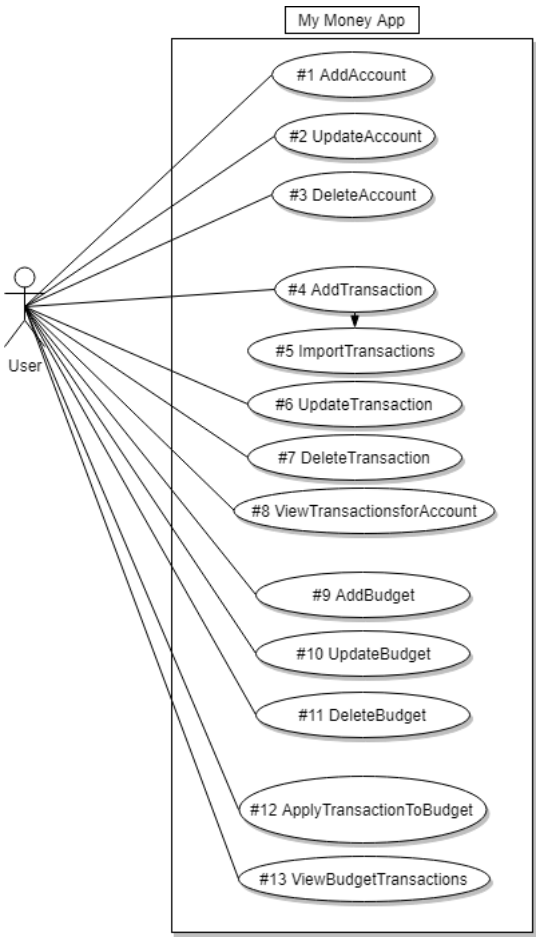
# 3 Test Results

*List the tests, indicating which passed and which did not pass. List requirements indicating the percentage of tests that passed for that requirement.*

# 4 References

# 5 Addendum

**Figure 1:** Updated use case diagram




# 6 Description of Input Files

Describe/include test data from input files.