

# 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

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Test Plan</b>	<b>1</b>
2.1	System Level Test Cases . . . . .	3
2.1.1	Test Case 1 - Add Account . . . . .	3
2.1.2	Test Case 2 - Update Account . . . . .	3
2.1.3	Test Case 3 - Delete Account . . . . .	4
2.1.4	Test Case 4 - Add Transaction . . . . .	4
2.1.5	Test Case 5 - Import Transaction . . . . .	5
2.1.6	Test Case 6 - Update Transaction . . . . .	5
2.1.7	Test Case 7 - Delete Transaction . . . . .	6
2.1.8	Test Case 8 - View Account Transactions . . . . .	6
2.1.9	Test Case 9 - Add Budget . . . . .	6
2.1.10	Test Case 10 - Update Budget . . . . .	7
2.1.11	Test Case 11 - Delete Budget . . . . .	7
2.1.12	Test Case 12 - Apply Transaction to Budget . . . . .	7
2.1.13	Test Case 13 - View Budget Transactions . . . . .	8
2.2	Unit Test cases . . . . .	9
2.2.1	Unit Test Case 1 - Transaction: saveItem . . . . .	9
2.2.2	Unit Test Case 2 - Transaction: deleteItem . . . . .	10
2.2.3	Unit Test Case 3 - Transaction: deleteItem (all items) . . . . .	11
2.2.4	Unit Test Case 4 - RepositoryContainer: saveItem (all types) . . . . .	12
2.2.5	Unit Test Case 5 - AccountRepository: saveItem, deleteItem . . . . .	13
2.2.6	Unit Test Case 6 - BudgetRepository: saveItem . . . . .	14
2.2.7	Unit Test Case 7 - BudgetRepository: deleteItem . . . . .	15
2.2.8	Unit Test Case 8 - AccountController: addOrUpdateAccount, deleteAccount . . . . .	16
2.2.9	Unit Test Case 9 - TransactionController: addOrUpdateTransaction, deleteTransaction . . . . .	17
2.2.10	Unit Test Case 10 - BudgetController . . . . .	18
2.3	Test Case Templates . . . . .	19
2.3.1	System Level Test Case Template . . . . .	19

2.3.2	Unit Test Case Template . . . . .	19
<b>3</b>	<b>Test Results</b>	<b>20</b>
3.1	System Level Tests . . . . .	20
3.2	Unit Level Tests . . . . .	21
3.3	Requirement Test Results . . . . .	21
<b>4</b>	<b>References</b>	<b>22</b>
<b>5</b>	<b>Addendum</b>	<b>22</b>
<b>6</b>	<b>Description of Input Files</b>	<b>22</b>

## List of Figures

1	Updated use case diagram . . . . .	22
---	------------------------------------	----

# 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

## 2.1 System Level Test Cases

### 2.1.1 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* 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 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* 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 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.

#### **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 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.

#### **Traces to Use Cases**

This satisfies the main scenario of use case 4 - *AddTransaction*.



### 2.1.5 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, 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 - 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.

#### **Traces to Use Cases**

This satisfies use case 6 - *ImportTransactions*

### 2.1.7 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 change/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*

### 2.1.8 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*

### 2.1.9 Test Case 9 - Add Budget

#### **Purpose**

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

#### **Input Specification**

The 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*

### **2.1.10 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*

### **2.1.11 Test Case 11 - Delete Budget**

#### **Purpose**

#### **Input Specification**

#### **Expected Output**

#### **Traces to Use Cases**

This satisfies use case 11 - *DeleteBudget*

### **2.1.12 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*

### **2.1.13 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*

## 2.2 Unit Test cases

### 2.2.1 Unit Test Case 1 - Transaction: saveItem

**Table 2:** UT-1

<b>Test Case Number</b>	UT-1
<b>Test Case Description</b>	This test case is used to ensure that transactions are properly saved or updated to their repository
<b>Input</b>	<ol style="list-style-type: none"><li>1. A Transaction object populated with generic data</li><li>2. A second Transaction object with the ID of the first one.</li><li>3. A test transaction database.</li></ol>
<b>Expected Output</b>	<ol style="list-style-type: none"><li>1. Transaction details are printed to console.</li></ol>
<b>Expected Post-Conditions</b>	A transaction database is created and a transaction is inserted. The balance of this transaction is then updated to a new value.
<b>Execution History</b>	<ol style="list-style-type: none"><li>1. 04/03/2018 — Colin Greczkowski — Test pass.</li></ol>

### 2.2.2 Unit Test Case 2 - Transaction: deleteItem

**Table 3:** UT-2

<b>Test Case Number</b>	UT-2
<b>Test Case Description</b>	This test verifies that the deleteItem method works as intended, and deletes a Transaction record for a given ID
<b>Input</b>	<ol style="list-style-type: none"><li>1. A generic account ID</li><li>2. A Transaction object populated with generic data, associated to the generic account.</li><li>3. A test transaction database.</li></ol>
<b>Expected Output</b>	<ol style="list-style-type: none"><li>1. "Delete Transaction 1"</li></ol>
<b>Expected Post-Conditions</b>	The test transaction database should be empty.
<b>Execution History</b>	<ol style="list-style-type: none"><li>1. 04/07/2018 — Colin Greczkowski — Test pass.</li></ol>

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

**Table 4:** UT-3

<b>Test Case Number</b>	UT-3
<b>Test Case Description</b>	This test case is used to make sure all Transactions associated to an account are properly purged from the repository.
<b>Input</b>	<ol style="list-style-type: none"><li>1. A generic account ID</li><li>2. Two Transaction objects populated with generic data, associated to the generic account.</li><li>3. A test transaction database.</li></ol>
<b>Expected Output</b>	<ol style="list-style-type: none"><li>1. "Delete Transaction 1"</li><li>2. "Delete Transaction 2"</li></ol>
<b>Expected Post-Conditions</b>	The test transaction database does not contain the two transactions that had the generic account ID.
<b>Execution History</b>	<ol style="list-style-type: none"><li>1. 04/03/2018 — Colin Greczkowski — Test pass.</li></ol>

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

**Table 5:** UT-4

<b>Test Case Number</b>	UT-4
<b>Test Case Description</b>	This tests the RepositoryContainer's ability to save a variety of types of objects (Transactions, Accounts, Budgets).
<b>Input</b>	<ol style="list-style-type: none"><li>1. Test Transaction, Budget and Account Databases</li><li>2. A test Transaction</li><li>3. A test Account</li><li>4. A test Budget</li></ol>
<b>Expected Output</b>	<ol style="list-style-type: none"><li>1. The test transaction's details are printed to console.</li></ol>
<b>Expected Post-Conditions</b>	The account, transaction and budget items are saved to their respective test databases. Balances are updated correctly.
<b>Execution History</b>	<ol style="list-style-type: none"><li>1. 04/07/2018 — Colin Greczkowski — Test pass.</li><li>2. 04/11/2018 — Michael Foo — Test fail.</li><li>3. 04/11/2018 — Melanie Taing — Test pass.</li></ol>



### 2.2.5 Unit Test Case 5 - AccountRepository: saveItem, deleteItem

**Table 6:** UT-5

<b>Test Case Number</b>	UT-5
<b>Test Case Description</b>	This test case is used to ensure created accounts are saved in the database. Also it verifies that accounts can be deleted from it.
<b>Input</b>	<ol style="list-style-type: none"><li>1. An account database</li><li>2. An account repository</li><li>3. An account with non-null values for <i>nickname</i>, <i>bankName</i> and a non-negative value for <i>balance</i></li></ol>
<b>Expected Output</b>	<ol style="list-style-type: none"><li>1. Tuples in Account repository test before delete: 2</li><li>2. Tuples in Account repository test after delete: 1</li><li>3. Current items loaded in repo:1</li><li>4. 1</li></ol>
<b>Expected Post-Conditions</b>	The system has a single account in the account database.
<b>Execution History</b>	<ol style="list-style-type: none"><li>1. 04/07/2018 — Wayne Yiel Leung — Test fail.</li></ol>

### 2.2.6 Unit Test Case 6 - BudgetRepository: saveItem

**Table 7:** UT-6

<b>Test Case Number</b>	UT-6
<b>Test Case Description</b>	This test case verifies that the saveItem method functions properly and saves the Budget object to the selected account's BudgetRepository.
<b>Input</b>	<ol style="list-style-type: none"><li>1. A Budget object populated with generic data</li><li>2. A test Account database populated with generic transactions</li><li>3. A test Budget database</li></ol>
<b>Expected Output</b>	<ol style="list-style-type: none"><li>1. Test budget added into repo: (Name, Amount, Balance) = (Test Budget, 100, 1000)</li><li>2. Budget amount before update: 100</li><li>3. Budget amount after update: 10000</li></ol>
<b>Expected Post-Conditions</b>	The test budget is added to the budget repository. The test budget's amount is updated to 1000.
<b>Execution History</b>	<ol style="list-style-type: none"><li>1. 04/08/2018 — Melanie Taing — Test fail</li><li>2. 04/10/2018 — Melanie Taing — Test pass</li></ol>

### 2.2.7 Unit Test Case 7 - BudgetRepository: deleteItem

**Table 8:** UT-7

<b>Test Case Number</b>	UT-7
<b>Test Case Description</b>	This test case verifies that the deleteItem method functions properly and deletes the Budget record in the repository for a given account ID.
<b>Input</b>	<ol style="list-style-type: none"><li>1. A generic account ID</li><li>2. A Budget object populated with generic data</li><li>3. A test Budget database</li></ol>
<b>Expected Output</b>	<ol style="list-style-type: none"><li>1. Tuples in Budget repository test before delete: 3</li><li>2. Tuples in Budget repository test after delete: 2</li><li>3. Current items loaded in repo: 2</li></ol>
<b>Expected Post-Conditions</b>	The test budget repository should contain one less budget.
<b>Execution History</b>	<ol style="list-style-type: none"><li>1. 04/08/2018 — Melanie Taing — Test fail</li><li>2. 04/10/2018 — Melanie Taing — Test pass</li></ol>

### 2.2.8 Unit Test Case 8 - AccountController: addOrUpdateAccount, deleteAccount

**Table 9:** UT-8

<b>Test Case Number</b>	UT-8
<b>Test Case Description</b>	This test verifies that an user is able add or update, or remove their account.
<b>Input</b>	<ol style="list-style-type: none"><li>1. An account with accountId, bankName, nickname, balance</li></ol>
<b>Expected Output</b>	<ol style="list-style-type: none"><li>1. The account is deleted from the database.</li></ol>
<b>Expected Post-Conditions</b>	There is one less account in the database.
<b>Execution History</b>	<ol style="list-style-type: none"><li>1. 04/11/2018 — Wayne Yiel Leung — Test pass</li></ol>

### 2.2.9 Unit Test Case 9 - TransactionController: addOrUpdateTransaction, deleteTransaction

**Table 10: UT-9**

<b>Test Case Number</b>	UT-9
<b>Test Case Description</b>	This test verifies that an user is able add or update, or remove a transaction from their account.
<b>Input</b>	<ol style="list-style-type: none"><li>1. A transaction with transactionId, accountId, BudgetId, type, date, amount and description</li></ol>
<b>Expected Output</b>	<ol style="list-style-type: none"><li>1. The transaction is deleted from the database.</li></ol>
<b>Expected Post-Conditions</b>	There is one less transaction in the database.
<b>Execution History</b>	<ol style="list-style-type: none"><li>1. 04/11/2018 — Wayne Yiel Leung — Test pass</li></ol>

### 2.2.10 Unit Test Case 10 - BudgetController

**Table 11:** UT-10

<b>Test Case Number</b>	UT-10
<b>Test Case Description</b>	
<b>Input</b>	1.
<b>Expected Output</b>	1.
<b>Expected Post-Conditions</b>	
<b>Execution History</b>	1. mm/dd/yyyy — Tester's name — Execution result

## 2.3 Test Case Templates

### 2.3.1 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.

### 2.3.2 Unit Test Case Template

**Table 12: UT-X**

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

## 3 Test Results

### 3.1 System Level Tests

**Table 13:** 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



### 3.2 Unit Level Tests

**Table 14:** 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	Fail
6	BudgetRepository: saveItem	Pass
7	BudgetRepository: deleteItem	Pass
8	AccountController	Pass
9	TransactionController	Pass
10	BudgetController	Pass

### 3.3 Requirement Test Results

**Table 15:** Requirement Test Results

Requirement	Test Passed
Account	80%
Transaction	100%
Budget	100%

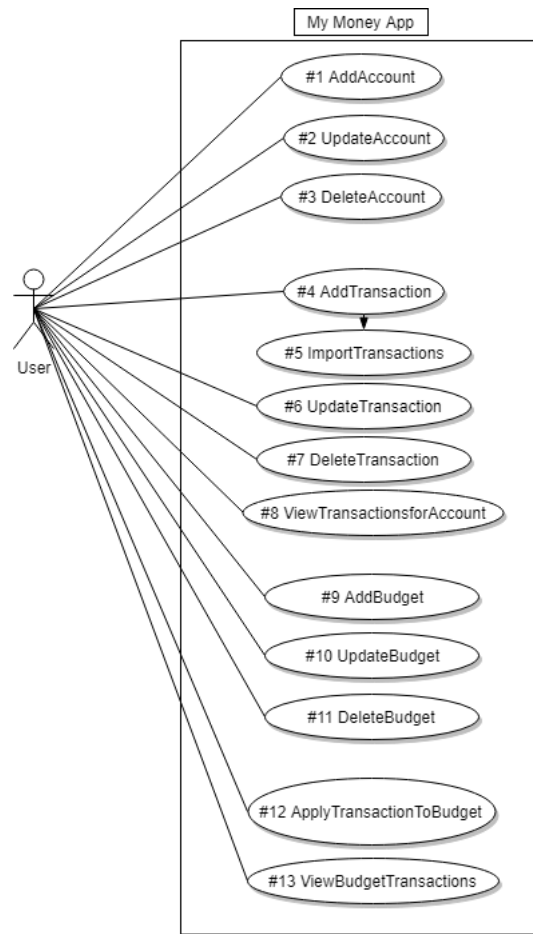
## 4 References

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.

## 5 Addendum

**Figure 1:** Updated use case diagram



## 6 Description of Input Files