Requirements Document

Team PA-PK

February 6, 2018

Table 1: Team

Name	ID Number
Anne-Laure Ehresmann	27858906
Marc-Antoine Dube	40029307
Kadeem Caines	26343600
Abdel Rahman Jawhar	27192142
Keith Dion	40036340

Contents

1 System **3**

1.1 Purpose 3

1.2 Business Goals 3

2 Domain Concepts **4**

3 Actors **5**

User 5

Bank 5

4 Functional Requirements **5**

5 Use Cases **5**

5.1 Overview 5

6 Non-Functional Constraints **5**

7 Data Dictionary **5**

8 References **5**

A Description of File Format: Tasks **5**

B Description of File Format: Persons **5**

List of Figures

1 Use Case Diagram 6

List of Tables

1 Team 1
2 Use Case 1 - Create User Account 7
3 Use Case 2 - Delete User Account 8
4 Use Case 3 - Add Bank Account to a User Account 9
5 Use Case 4 - Remove Bank Account from a User Account 10
6 Use Case 5 - View Transactions for Specific Bank Account 11
7 Use Case 6 - View All Transactions from all Bank Accounts 12
8 Use Case 7 - View Transaction by Type 13
9 Use Case 8 - Categorize Transaction 14
10 Use Case 9 - View Transactions by Category 15

1 System

1.1 Purpose

The purpose of this document is to define requirements for the desktop application myMoney. There exists a plethora of software for money management, each greatly varying in design due to the complex and multifarious clientele. This document may thus be to orient the development of the application. It seeks to understand the requirements of the problem, formulate the necessary functions and properties needed to answer this problem and its requirements, and then test these functions against these requirements. Hence, it may be used by our users to specify the problem and its requirements, by the developers to understand what functions their system must implement, and what to test their system against.

1.2 Business Goals

- **Meet current procedure performance:** The current procedure for users benefits from near constant *reliability*, as its only constraints is the reliability of the bank servers. We want to guarantee this same level of reliability, that is, that our system's reliability is only constrained by the reliability of the banks.
  - *Quality attribute scenario:* System operates mostly offline, and can function even without access to the internet using past data.
- **Ease-of-use:** The current procedure for users to view their bank transactions across different accounts requires them to log in each individual bank account interface, then compare each distinct format for the bank accounts manually. This is both needlessly time-consuming and difficult to navigate. Budgeting applications exist, however they can quickly become overbearing, offering a vast quantity of complex and heavy features that are designed primarly for heavyweight users.
  - *Quality attribute scenario:* Installation of the system should be easy and require very little, if any, decisions from the user beyond choosing to install the system. With no training, a user should be able to intuitively learn and use the system in a short amount of time.
- **Performance:** Ensured minimal load times to encourage frequent, short-time bursts use.
  - *Quality attribute scenario:* System files should be small, less than 100MB. When the system connects to the internet (with reasonable bandwith) to fetch transactions, they are returned to the user in less than 2 seconds.
- **Maintainability:** Due to the dependency of the system upon the banks' API, it is necessary that the system should be easy to alter and maintain to adapt to possible necessary changes. Hence, maintainability is imperative for the system. We thus advocate for a reliable and easily usable *testing suite* to reduce time spent hunting for bugs, and quickly run diagnostics on the state of the system.
  - *Quality attribute scenario:* New modifications to the system should be fully covered by unit tests before another modification is implemented.

- **Security:** A user's bank information is incredibly sensitive, and any possible security flaw could cripple the entire system and destroy any sort of trusted relationship that users may have developed with it. It is thus imperative that any data coming from the banks are accessible solely with proper authentication.
  - *Quality attribute scenario:* Database should be encrypted, and testing suite should include verification of security measures on the system.
- **Portability:** Since the system is fairly lightweight, it should be one that is independent of a user's hardware (within reason) and/or operating system.
  - *Quality attribute scenario:* System should be implemented in Java to benefit from the JVM's portability.
- **Scalability:** The system should function even on a long-term basis, and hence be capable of handling a growing number of transactions. The scaling size of the database should be managed by the system so as not to overwhelm the hardware it is running on.
  - *Quality attribute scenario:* System should use SQLite to minimize database size, and take precautions not to load an entire database in memory should the number of transactions exceed a size that may be unmanageable by the hardware.

2 Domain Concepts

3 Actors

User

Our main actor is the user. All use cases are triggered by the user, as it their requests that directly cause the bank system or our system to take action. All identification needed to access the bank system will be provided by the user. Using this information, the system will be able to answer queries made by the user as described in the use cases.

Bank

Our sole other actor is the bank systems. The bank system provides an API for accessing its bank account data. Our system will pull this data directly from the bank systems, using identification as given by the user for the bank's authentication system. In other words, our system will merely act as a middle man between the user demanding information in a specific format, and the bank holding that information in a inconvenient format.

4 Functional Requirements

5 Use Cases

5.1 Overview

Use cases 1 through 4 deal with the user manipulating his accounts. Use cases 5 through 7 and 9 deal with the user viewing the data in different formats. Use case 8 deals with the user manipulating the transactions' formats.

6 Non-Functional Constraints

7 Data Dictionary

8 References

A Description of File Format: Tasks

Describe input file format.

B Description of File Format: Persons

Describe output file format.

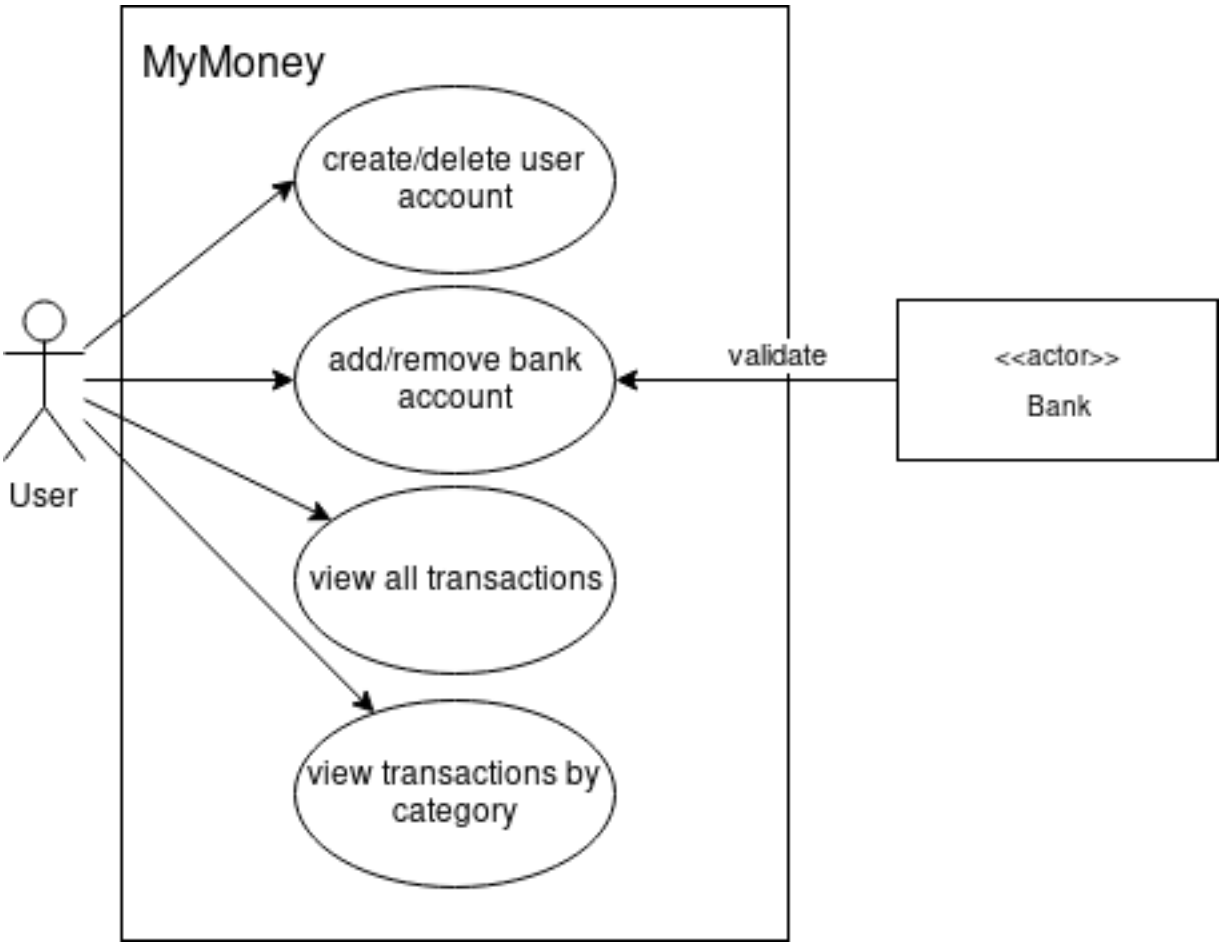

Figure 1: Use Case Diagram

Table 2: Use Case 1 - Create User Account

Action	**Create User Account**
Case ID	01
Summary	User gives information about a new user account, system validates it and creates the account.
Scope	money and budget management application
Level	user-goal
Actors	**User**
Stakeholders and Interests	1. User: Wants fast and easy account creation, clear and comprehensible display, proof of successful account creation. 2. Company: Wants user interests to be fulfilled, wants to prevent erroneous input, wants fast communication with the local account database as well as fault tolerance in case of database conflicts, issues with editing authorization, or other possible database problems.
Pre-Conditions	User has opened the application and is in the signup menu.
Success Guarantee	Account successfully saved in local account database, with name and password as specified by the user.
Main Success Flow	1. User enters a username, first name, last name, and password. 2. System validates username and password (format, whether username is already used, etc). 3. System creates new account. 4. System notifies the user of the successful account creation, then returns to home menu.
Exceptions	
Post-Conditions	
Priority	
Traces to Test Cases	

Table 3: Use Case 2 - Delete User Account

Action	**Delete User Account**
Case ID	02
Summary	User deletes a user account from the local accounts database, removing all bank accounts and information associated with that account.
Scope	money and budget management application
Level	user-goal
Actors	**User**
Stakeholders and Interests	1. User: Wants easy navigation and secure account deletion, no risk of accidental deletion, proof of successful account deletion. 2. Company: Wants user interests to be fulfilled, wants to ensure clean deletion from the database.
Pre-Conditions	User has logged in the user account he wants to delete.
Success Guarantee	user account is successfully deleted, all associated bank information is deleted, and user is returned to the home menu.
Main Success Flow	1. User enters the account settings, then selects 'delete account'. 2. System brings up a confirmation menu to ensure that this selection was not accidentally entered. 3. User confirms his choice. 4. System successfully deletes all relevant entries in the local database, then notifies the user of this successful deletion. 5. User confirms having read this notification. 6. System brings the user back to the home menu.
Exceptions	
Post-Conditions	
Priority	
Traces to Test Cases	

Table 4: Use Case 3 - Add Bank Account to a User Account

Action	**Add Bank Account to a User Account**
Case ID	03
Summary	User gives information about a new bank account, system sends it to the bank for verification then creates necessary entries in the local database once the bank approves the information.
Scope	money and budget management application
Level	user-goal
Actors	**User**, Bank
Stakeholders and Interests	1. User: Wants fast and easy account creation, clear and comprehensible display, proof of successful account creation. 2. Company: Wants user interests to be fulfilled, wants to prevent erroneous input, wants fast communication with the local account database as well as the bank, wants fault tolerance in case of database conflicts, issues with the bank, or other possible local database problems. 3. Bank: Wants to satisfy its customer base, wants correctly formatted account information given to its API, inexpensive and non-redundant communication of bank account data to third-party applications.
Pre-Conditions	User has logged in a user account and is in the user home menu.
Success Guarantee	Bank account successfully saved in local account database, with information corresponding the data validated by the bank.
Main Success Flow	1. User enters his/her bank account number. 2. System validates input, and sends it to the bank for verification and connection. 3. Bank validates bank account number, and then responds with the bank account information. 4. System records the valid bank account details securely in its database, and notifies the user of the successful addition of the bank account in the database.
Exceptions	
Post-Conditions	
Priority	
Traces to Test Cases	

Table 5: Use Case 4 - Remove Bank Account from a User Account

Action	**Remove Bank Account from a User Account**
Case ID	04
Summary	User deletes a bank account from the local database.
Scope	money and budget management application
Level	user-goal
Actors	**User**
Stakeholders and Interests	1. User: Wants easy navigation and secure account deletion, no risk of accidental deletion, proof of successful account deletion. 2. Company: Wants user interests to be fulfilled, wants to ensure clean deletion from the database.
Pre-Conditions	User has logged in the user account whose active association with a bank account is the one the user wants to delete.
Success Guarantee	Bank account is successfully removed from that user account, all associated bank information is deleted from the local database, and user is returned to the account home menu.
Main Success Flow	1. User selects the account he wants to remove, then selects 'remove account'. 2. System brings up a confirmation menu to ensure that this selection was not accidentally entered. 3. User confirms his choice. 4. System successfully deletes all relevant entries in the local database, then notifies the user of this successful deletion. 5. User confirms having read this notification. 6. System brings the user back to the home account menu.
Exceptions	
Post-Conditions	
Priority	
Traces to Test Cases	

Table 6: Use Case 5 - View Transactions for Specific Bank Account

Action	**View Transactions for Specific Bank Account**
Case ID	05
Summary	User selects specific bank account and views transactions associated with with selected bank account
Scope	money and budget management application
Level	user-goal
Actors	**User**
Stakeholders and Interests	1. User: Wants quick and convenient viewing of previous transactions from one specific bank account 2. Company: Wants to give user ability to micromanage every aspect of the application down to each bank account and transaction.
Pre-Conditions	User has created and logged into a user account, and has added at least one bank account to his/her myMoney account.
Success Guarantee	User can view transaction by bank account.
Main Success Flow	1. User selects specific bank account from list of all accounts. 2. System displays all previous transactions under specified bank account. 3. User selects desired transaction. 4. System shows all information about desired transaction, such as date and amount withdrawn, deposited, or transferred.
Exceptions	
Post-Conditions	
Priority	
Traces to Test Cases	

Table 7: Use Case 6 - View All Transactions from all Bank Accounts

Action	**View All Transactions from all Bank Accounts**
Case ID	06
Summary	User can view all transactions that have been made from all bank accounts
Scope	money and budget management application
Level	user-goal
Actors	**User**
Stakeholders and Interests	1. User: Wants easy and convenient viewing of all transactions among all bank accounts. 2. Company: Wants user interests to be fulfilled.
Pre-Conditions	User has created and logged into a user account, and at least one bank account has been added to his/her myMoney account.
Success Guarantee	User is able to conveniently view all transactions from all institutions in one display.
Main Success Flow	1. User selects View All Transactions option. 2. System shows all transactions across all accounts on one display.
Exceptions	
Post-Conditions	
Priority	
Traces to Test Cases	

Table 8: Use Case 7 - View Transaction by Type

Action	**View Transaction by Type**
Case ID	07
Summary	User can view transactions by the following types: deposits, withdrawals, and transfers.
Scope	money and budget management application
Level	user-goal
Actors	**User**
Stakeholders and Interests	1. User: Wants to view transaction by type, such as withdrawals, deposits, and transfers for convenient view of the movement of his/her money throughout the accounts and accurate records of spending. 2. Company: Wants user interests to be fulfilled. Wants to facilitate user's ability to keep track of input and output of money between accounts.
Pre-Conditions	User has created and logged in a user account, added at least one bank account, and has made at least one type of transaction.
Success Guarantee	User is able to clearly view all deposits, withdrawals, and transfers that have been made.
Main Success Flow	1. User selects View All Transactions option or View By Specific Bank Account. 2. System shows all transactions associated with previous choice. 3. User selects view transactions by type 4. System prompts user to choose which type of transaction he/she would like to view: deposits, withdrawals, transfers. 5. User selects one of the options 6. System shows all information regarding selected transaction such as date, amount, to which account, and from which account.
Exceptions	
Post-Conditions	
Priority	
Traces to Test Cases	

Table 9: Use Case 8 - Categorize Transaction

Action	**Categorize Transaction**
Case ID	08
Summary	User can select a category to represent a transaction.
Scope	money and budget management application
Level	user-goal
Actors	**User**
Stakeholders and Interests	1. User: Wants to label each transaction as a specific type of spending, such as rent, bills, leisure, etc., to better manage and track spending habits. 2. Company: Wants user interests to be fulfilled. Wants to facilitate user's ability to budget and manage money through simple categorization of spending.
Pre-Conditions	User has created and logged in a user account, added at least one bank account, and has made at least one type of transaction.
Success Guarantee	User can quickly and efficiently categorize each transaction.
Main Success Flow	1. User selects desired transaction to categorize. 2. User selects the categorize option. 3. System displays the categories in a drop down menu. 4. User selects preferred category to represent the current transaction. 5. System saves transaction under chosen category.
Exceptions	
Post-Conditions	
Priority	
Traces to Test Cases	

Table 10: Use Case 9 - View Transactions by Category

Action	**View Transactions by Category**
Case ID	09
Summary	User can view transaction according to categories of spending.
Scope	money and budget management application
Level	user-goal
Actors	**User**
Stakeholders and Interests	1. User: Wants to view transactions in different categories of spending such as rent, bills, leisure, etc., to view and monitor spending habits and allocate budget to each category of spending. 2. Company: Wants to optimize the ease in which a user can allocate his/her budget, and track spending habits.
Pre-Conditions	User has created and logged in a user account, has added at least one bank account, and has made at least one transaction.
Success Guarantee	User can easily categorize and view transactions.
Main Success Flow	1. User selects View All Transactions. 2. System shows all transactions from all accounts on one display. 3. User selects to view transaction by categories. 4. System groups transactions by similar category. 5. System displays transactions grouped by category.
Exceptions	
Post-Conditions	
Priority	
Traces to Test Cases	