Requirements Document

Team PA-PK

February 9, 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

List of Figures

List of Tables

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.

Brief description of the problem

At the present time, users who have more than one bank account can quickly get overwhelmed with the differing methods of access and interfaces for each account. It becomes arduous to access every account and difficult to visualise how much money one has and how one's budget changes on a day-to-day basis. Numerous applications exist with extensive budgeting functionalities, but setting them up and learning to use them requires investing some number of hours that often seems more hassle than is worth, especially for casual users who aren't interested in complex accounting functionalities. This thus costs time and frustration for the common user, and discourages them from fully taking advantage of the services that are provided by their banks. Even simply connecting to their accounts becomes a daunting task, which negatively impacts both the users and the banks. We seek to design a desktop application to solve these issues in a simple and lightweight manner.

1.2 Business Goals

- **Compete with existing solutions**
- **Target market: wide user-base composed mainly of millenials, students, new professionals**
- **Reduce total cost of ownership**
- **Future-proof, long-term robustiness of the system**

1.3 Non-Functional Requirements

- **reliability:** The current procedure which users employ benefits from near perfect uptime, as its only constraint 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 servers.
  - *Quality attribute scenario:* System operates mostly offline, and can function even without access to the internet using past data. System meets security and connectivity performance of current procedure.

Table 2: Non-Functional Requirement 1 - Reliability

Action	**Reliability**
Requirement ID:	01
Type:	Quality
Description:	We want to guarantee that our system's reliability is only constrained by the reliability of the banks servers.
Reasoning:	The current procedure which users employ benefits from near perfect uptime, as its only constraint is the reliability of the bank servers. In order to be competitive with this current solution, we must match this level of reliability

- **Simplicity and 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. Our business targets a more casual user-base which tend to favours a more simpler and more lightweight application.
  - *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. These sorts of applications tends to be popular with millenial audiences. This also allows the system to be competitive, as being less performant than the current procedures would certainly lead to a dissatisfaction in our user-base.
  - *Quality attribute scenario:* System files should be small, less than 50MB. When the system connects to the internet (with reasonable bandwidth) 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, and hence be capable of adapting to third-party changes. We also foresee a long-term use of the application, in which case the user requirements may change as our target audience change their demands. Hence, maintainability is imperative for the system. We thus advocate for a reliable and easily usable *testing suite* to reduce time and cost of debugging, and quickly conduct system diagnostics.
  - *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 our business. It is thus of upmost importance that any data coming from the banks are accessible solely with proper authentication. Improper security would certainly negatively impact user trust in our business.
  - *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 also focus on compatibility with older hardware (within reason) and support numerous operating systems, to encourage a wide user-base.
  - *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 Functional Requirements

This section will cover the functional requirements associated with the myMoney app. These are the software capabilities that must be present in order for the user to carry out the services provided by the app or to execute the use cases.

- **User account creation:** The system allows the creation of user accounts, with information (username, password, first name, and last name) provided by the user.
- **User account authentication:** The system shall grant a user access only to a properly authenticated user. If a user does not enter credentials, the system shall notify the user of the failure to authenticate.
- **User account management:** The system shall require proper authentication to modify and/or delete a user account.
- **Bank account management:** The system permits a logged-in user to add, manipulate, or remove bank accounts that are connected to the user account.
- **Transaction management:** The system permits a logged-in user to access the details of transactions associated with all bank accounts that were added to the user account. The system obtains these transactions through the banking system's API. The system allows different display formats, such as 'sorted by date' or 'sorted by type' or 'sorted by bank account.

4 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(s) system. Each bank system provides an API for accessing its bank account data. Our system will pull this data directly from the bank systems (who act as secondary actors), 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 banks holding that information in a inconvenient format.

5 Data Dictionary

6 References

A Description of File Format: Tasks

Describe input file format.

B Description of File Format: Persons

Describe output file format.

C Use Cases

C.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. For iteration 1, we have only included use case 1, 3, 5, and 6, as those are the ones we have fully implemented.

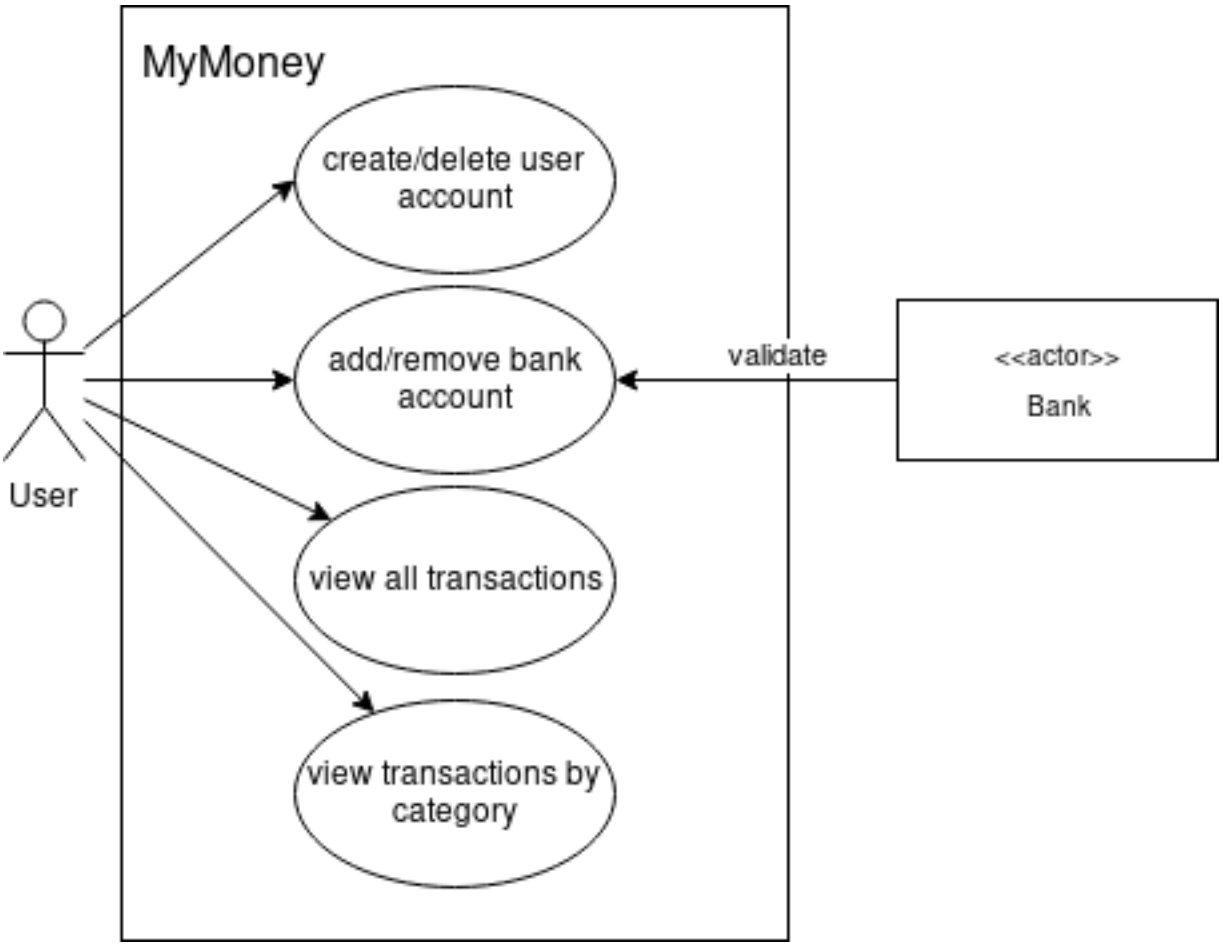

Figure 1: Use Case Diagram

Table 3: 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 4: 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.

Table 5: 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 6: 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.

Table 7: 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 8: 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 9: 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.

Table 10: Use Case 8 - Categorize Transaction

Action	**Categorize Transaction**
Case ID	08
Summary	User can select a category to represent a transaction.

Table 11: 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.