**Software Requirements Specification**

**for**

**Personal Budget Manager Application**

**Version 0.1**

**Prepared by**

|  |  |  |
| --- | --- | --- |
| **Name** | **Student ID** | **e-mail** |
| **Danny Shash** | **29548912** | **danny.shash@gmail.com** |
| **Ganji Amarendher** | **25764246** | **a\_ganji@encs.concordia.ca** |
| **Siming Huang** | **40081588** | **hsim47@163.com** |
| **Tony Lac** | **40049123** | **tony.lac@polymtl.ca** |
| **Xinjie Zeng** | **27238223** | **xinjiezeng@gmail.com** |

19/2/2019

**Content**

**TABLE OF CONTENTS**

[1. Introduction 1](#_Toc1483564)

[1.1 Document Purpose 1](#_Toc1483565)

[1.2 Product Scope 1](#_Toc1483566)

[1.3 Definitions and Acronyms 1](#_Toc1483567)

[1.3.1 Definitions 1](#_Toc1483568)

[1.3.2 Acronyms 1](#_Toc1483569)

[2. Overall Description 2](#_Toc1483570)

[2.1 Product Overview 2](#_Toc1483571)

[2.2 Product Functionality 2](#_Toc1483572)

[2.3 User Classes and Characteristics 3](#_Toc1483573)

[2.4 Design and Implementation Constraints 5](#_Toc1483574)

[2.5 Analysis Models 6](#_Toc1483575)

[2.5.1 Use case model 6](#_Toc1483576)

[2.5.2 Class diagram 9](#_Toc1483577)

[2.5.3 MVC diagram 10](#_Toc1483578)

[3. Specific Requirements 10](#_Toc1483579)

[3.1 External Interface Requirements 10](#_Toc1483580)

[3.1.1 User Interfaces 10](#_Toc1483581)

[3.1.2 Hardware Interfaces 16](#_Toc1483582)

[3.1.3 Software Interfaces 16](#_Toc1483583)

[3.2 Functional Requirements 16](#_Toc1483584)

[4. Other Non-functional Requirements 17](#_Toc1483585)

[4.1 Performance Requirements 17](#_Toc1483586)

[4.2 Safety and Security Requirements 17](#_Toc1483587)

**LIST OF FIGURES**

[Figure 1. Use case model PMB 6](#_Toc1483342)

[Figure 2. Use case model: Add Expense 7](#_Toc1483343)

[Figure 3. Use case model: Modify expense 7](#_Toc1483344)

[Figure 4. Use case model: Mark paid/unpaid 8](#_Toc1483345)

[Figure 5. Use case model: Remove expense 8](#_Toc1483346)

[Figure 6. Class diagram 9](#_Toc1483347)

[Figure 7. MVC diagram 10](#_Toc1483348)

[Figure 8. PBM home window 11](#_Toc1483349)

[Figure 9. Add expense: Purchase 12](file:///C:\Users\danny\Desktop\Project1-Final1.docx#_Toc1483350)

[Figure 10. Add expense: Bill 12](file:///C:\Users\danny\Desktop\Project1-Final1.docx#_Toc1483351)

[Figure 11. Updated expense list 13](#_Toc1483352)

[Figure 12. Remove expense (a) 13](#_Toc1483353)

[Figure 13. Remove expense (b) 14](#_Toc1483354)

[Figure 14. Mark paid/unpaid (a) 15](#_Toc1483355)

[Figure 15. Mark paid/unpaid (b) 15](#_Toc1483356)

# 1. Introduction

## 1.1 Document Purpose

The purpose of this document is to analyze the requirements of implementing Personal Budget Manager Application (an application to help individuals control their expenses), including product functions, user cases, analysis model, assumptions and dependencies. The analysis model will include use case diagrams, class diagrams and MVC diagrams.

## 1.2 Product Scope

This document addresses the high-level requirements of PBM that will be used as a reference for the implementation of the system. The use case diagrams explain the interaction of the user, any individual and the result from it. The goal of the system is to provide an overview of the expenses, that are day-to-day or recurring and manage them in an efficient manner.

The development plan is divided into 3 iterations and the document will be updated accordingly to address the content in that phase.

## 1.3 Definitions and Acronyms

### 1.3.1 Definitions

Purchase A type of day-to-day expense

Bill A type of recurring expense

### 1.3.2 Acronyms

PBM Personal Budget Manager Application

SRS Software Requirement Specification

# 2. Overall Description

## 2.1 Product Overview

As many people don’t realize that they spend money more than they earn, PBM is developed to help individuals control their expenses and keep them out of debt. It is distinguished between two types of expenses: purchase expenses and Bill expenses. The former has a description being linked to retailor(s) (name, location, amount, date, etc.) and status ((paid (e.g. cash, debit), due date (credit card)). The latter has a description (name of the service company), status (paid, not paid) and a repetition interval (e.g. weekly, monthly, yearly… etc.). PBM will be equipped with a Java Swing User Interface. The expense table should refresh its display automatically when the expenses in the expense list change.

## 2.2 Product Functionality

PBM will have the following functionality.

* View all expenses of the spending list (in arbitrary order).
* Add a new expense.
* Update an existing expense (Mark an expense as paid or not paid).
* Remove an expense.
* The expense table should refresh its display automatically when the expenses in the expense list change.

Planned future extensions of the Personal Budget Manager are:

* Addition of new expense types.
* Support for multiple user interfaces (e.g., Web UI, Mobile UI).
* Multiple presentations of the expense list (e.g., using trees or lists).

## 2.3 User Classes and Characteristics

Use Case 1: Add a new expense

Description: User adds a new expense

Actor(s): User

Goal: User able to add 2 types of expenses

Pre-conditions: User already logged in to the system

Main scenario:

1. User chooses to add a new expense

2. User prompted to choose the expense types

3. User enters the details for the selected expense types

4. System validates the data

5. System add the expense into the list

6. System displays the main window with the list of existing expenses for the type just entered

Alternative scenarios:

1. The information is incorrect (amount entered is negative)

1.1 User will receive a prompt indicating which info is incorrect

1.2 After correcting the info, it will be sent and stored in database

Post-conditions: The list of all the expenses displayed on the main window for the selected time period.

Use Case 2: Modify an existing expense

Description: User can modify an existing expense

Actor(s): User

Goal: user is able to modify an expense type belongs to a certain period

Pre-conditions: User already logged in to the system.

Main scenario:

1. User selects a row from the list and modifies allowed details, like paid, amount, any additional details, etc...

2. System modifies the expense entry

3. System refreshes the main window with existing expense list

Alternative scenarios:

1. User changes the expense type for the displayed period

1.1 System fetched the expense list for the period chosen by the user

1.2 System displays the expense list in the main window

Same steps as in the main flow.

2. User changes the time period for the displayed expense type

2.1 System fetched the expense list for the period chosen by the user

2.2 System displays the expense list in the main window

Same steps as in the main flow.

3. User changes the expense type and the time period

3.1 System fetched the expense list for the period chosen by the user

3.2 System displays the expense list in the main window

Same steps as in the main flow.

Post-conditions: The list of all the expenses displayed on the main window for the selected time period.

Use Case 3: Mark an expense as paid or not paid

Description: User Mark an expense as paid or not paid

Actor(s): User

Goal: User able to Mark an expense as paid or not paid

Pre-conditions: User already logged in to the system

Main scenario:

1. User selects row(s) of expenses from the list of expenses
2. User selects the ‘Mark expense paid/unpaid’ button
3. The default ‘unpaid’ status is changed to ‘paid’
4. The information will be sent and stored in the database

Alternative scenarios:

1. Mark an expense as paid incorrectly

1.1 User has to identify the incorrect status.

1.2 User selects the incorrect row(s) of expenses from the list of expenses

1.3 User selects the ‘Mark expense paid/unpaid’ button

1.4 The information will be sent and changed in the database

Use Case 4: Remove an expense

Description: User chooses to remove an expense

Actor(s): User

Goal: User able to remove an entry in the selected expense list

Pre-conditions: User already logged in to the system.

Main scenario:

1. User selects a row in the list to be removed

2. User selects the ‘Remove expense’ button

3. System refreshes the main window with existing expense list

## 2.4 Design and Implementation Constraints

PBM is a desktop application implemented using Java Swing as user interface.

## 2.5 Analysis Models

### 2.5.1 Use case model

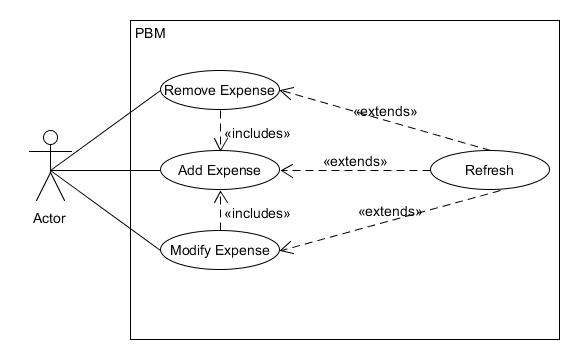


Figure 1. Use case model PMB

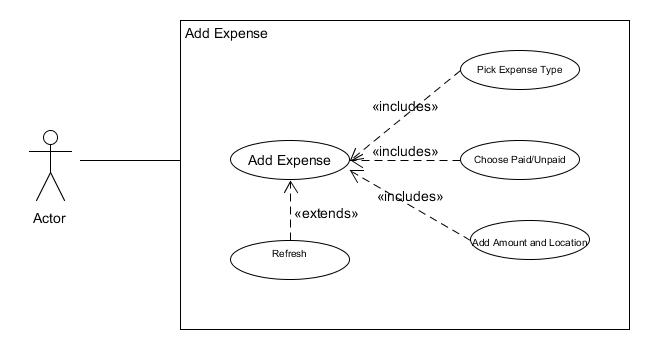


Figure 2. Use case model: Add Expense

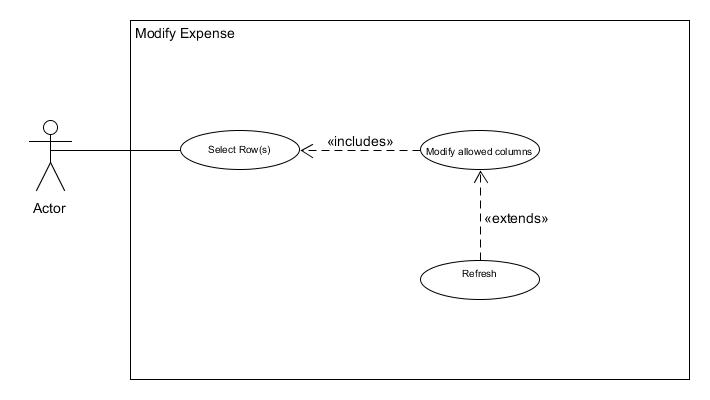


Figure 3. Use case model: Modify expense

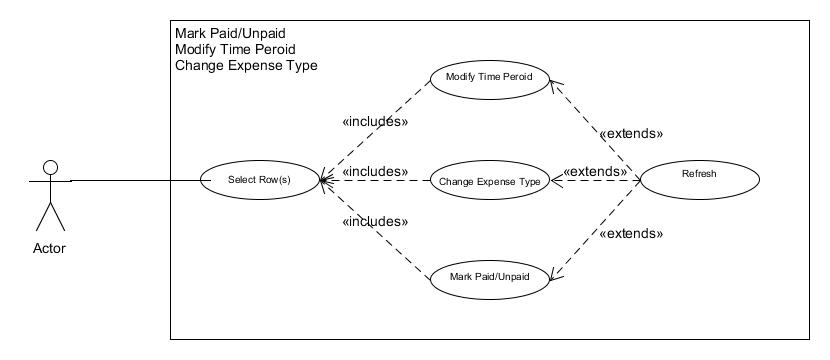


Figure 4. Use case model: Mark paid/unpaid

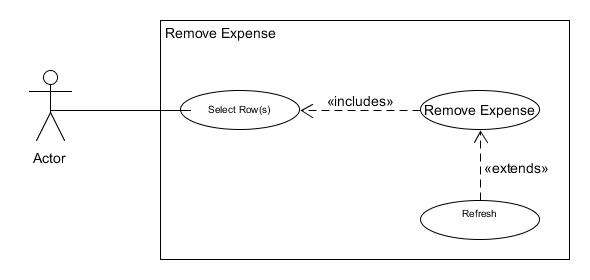


Figure 5. Use case model: Remove expense

### 2.5.2 Class diagram

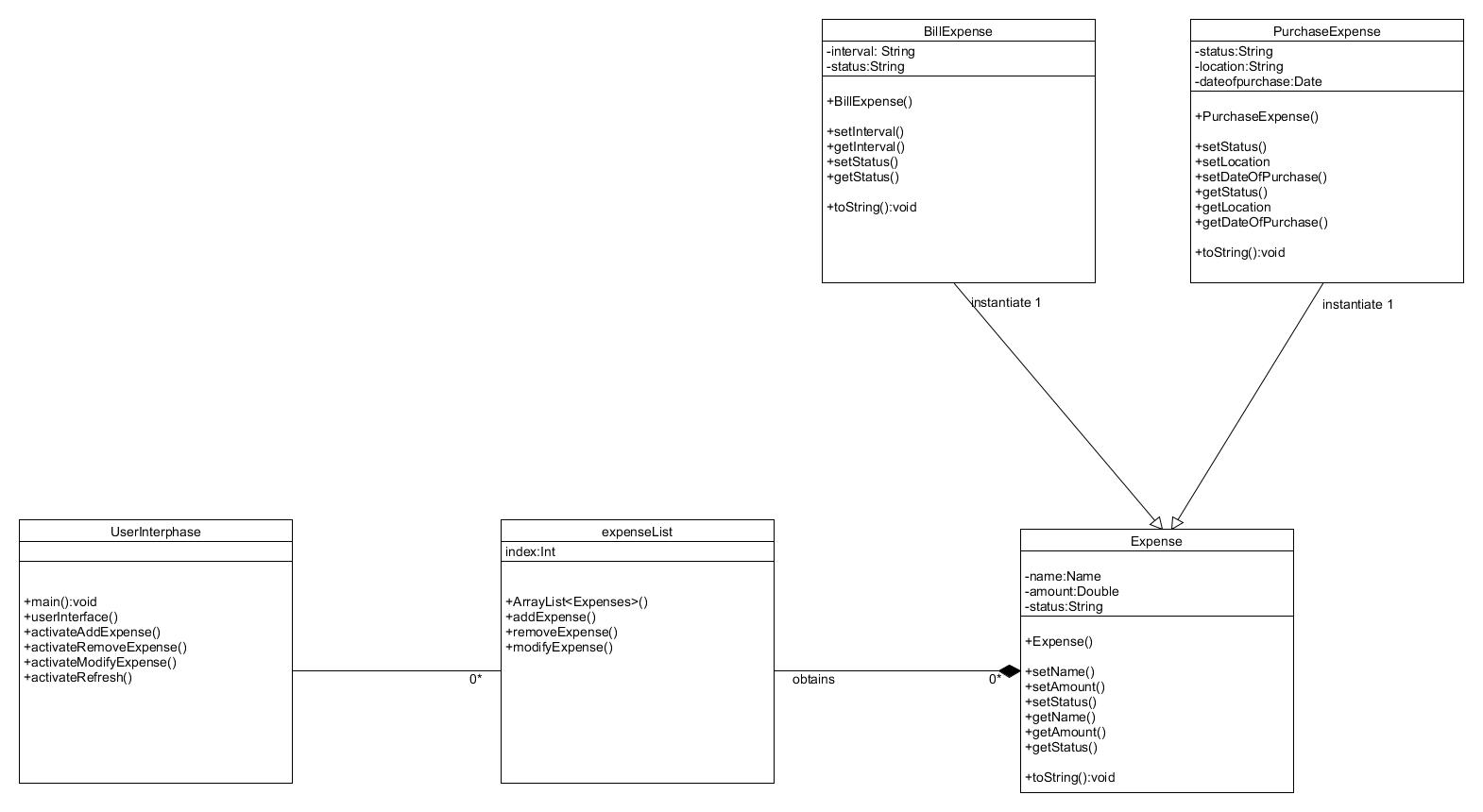


Figure 6. Class diagram

### 2.5.3 MVC diagram

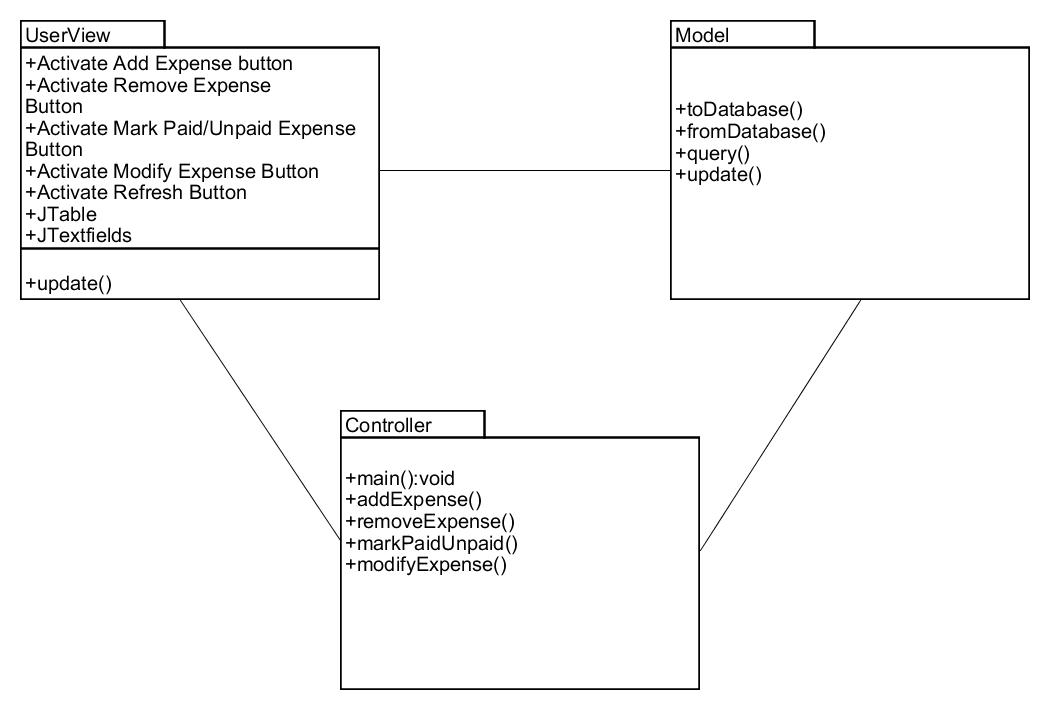


Figure 7. MVC diagram

# 3. Specific Requirements

## 3.1 External Interface Requirements

### 3.1.1 User Interfaces

This is the basic interface of the PBM. There is an expense list and three buttons, those functions are adding expense, removing expense and marking expense as paid or unpaid.

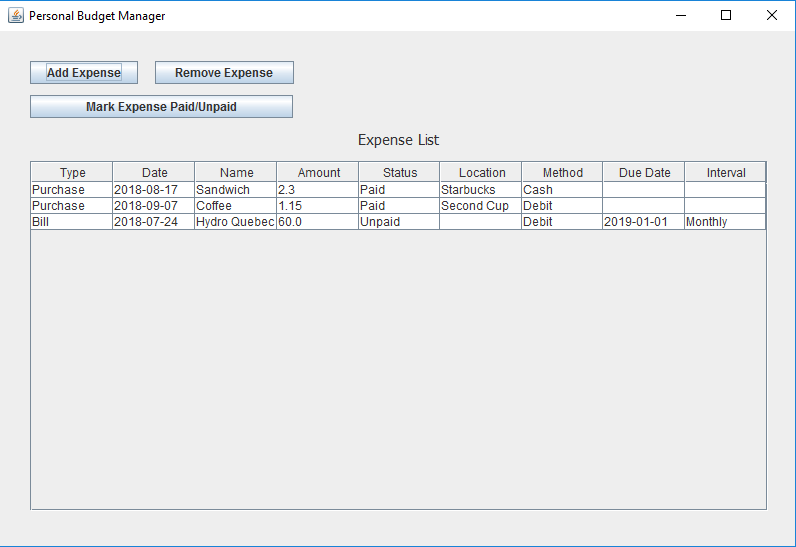


Figure 8. PBM home window

When users click the button ‘add expense’, another interface will pop up. Here they can input description of their expenses. There is a drop down buttons where users can change types of expenses: purchase and bill. With different expense types, they may input different details.

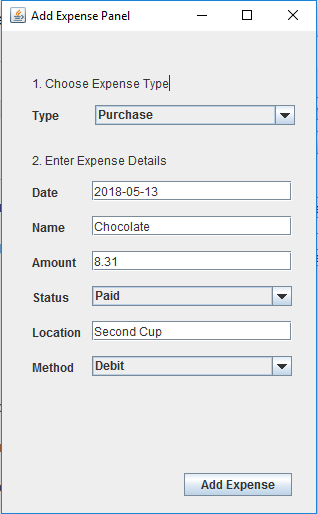
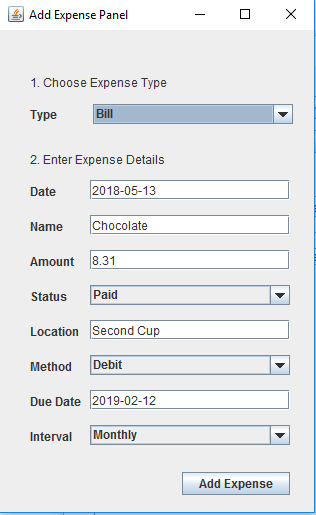


Figure 9. Add expense: Purchase

Figure 10. Add expense: Bill

After completing all the information, click the button ‘add expense’ then the data will be stored in database and added on the list.

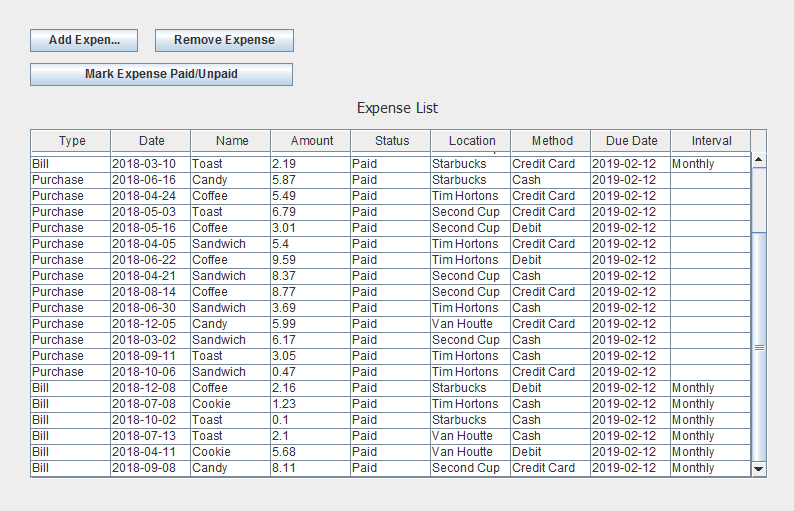


Figure 11. Updated expense list

If users want to delete an expense, just chose a line, click the button ‘Remove expense’, then the data will be deleted from the list and database.

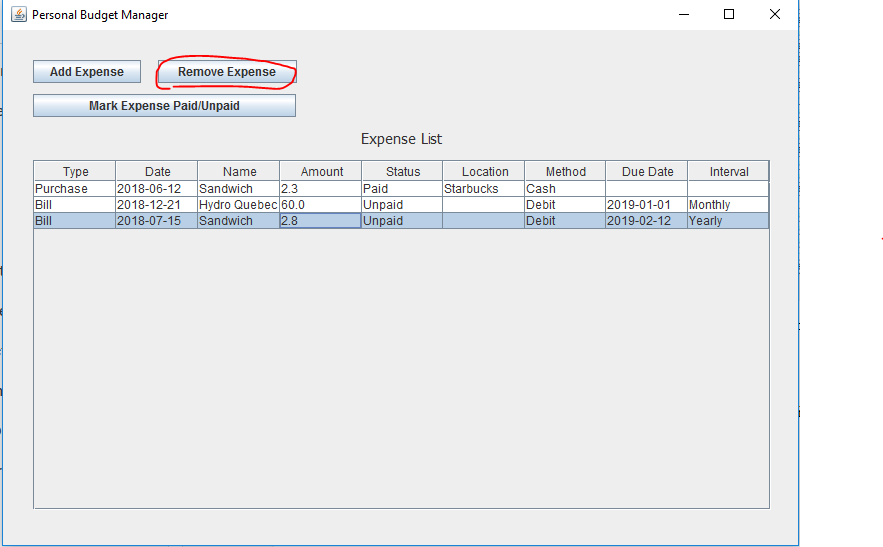


Figure 12. Remove expense (a)

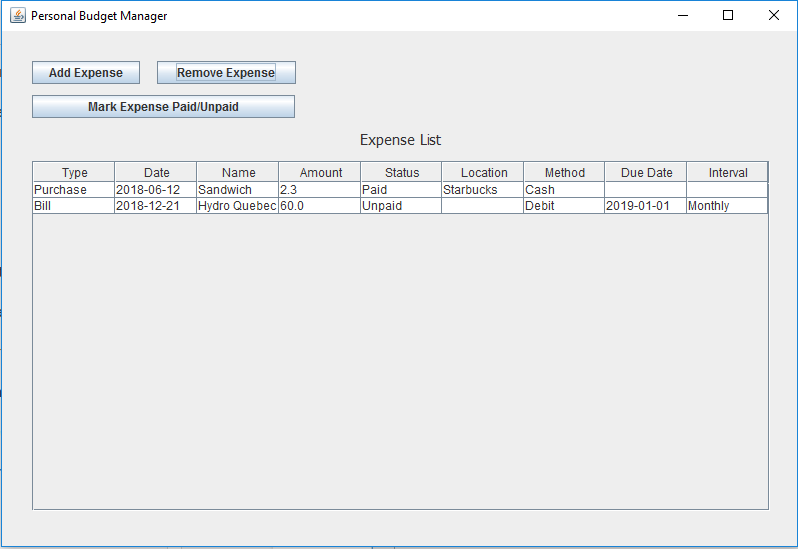


Figure 13. Remove expense (b)

If users want to change an expense status, just chose a line, click the button ‘Mark expense paid/unpaid’, then the change will be stored in the database.

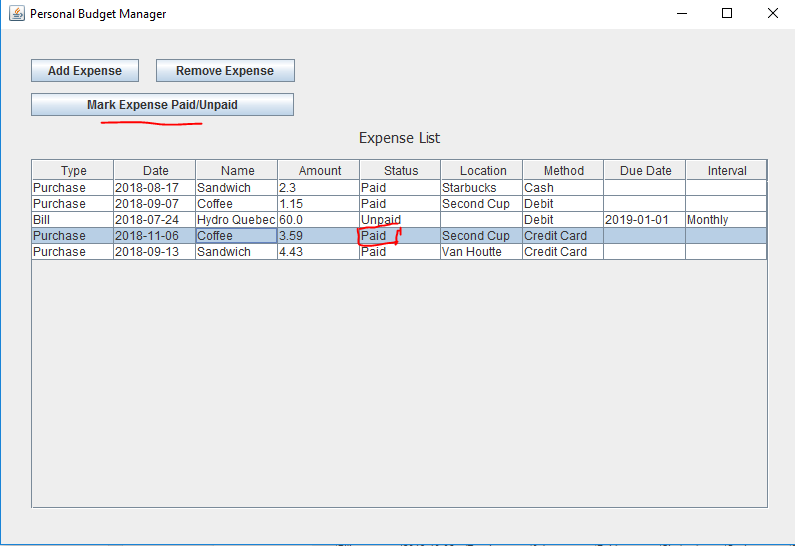


Figure 14. Mark paid/unpaid (a)

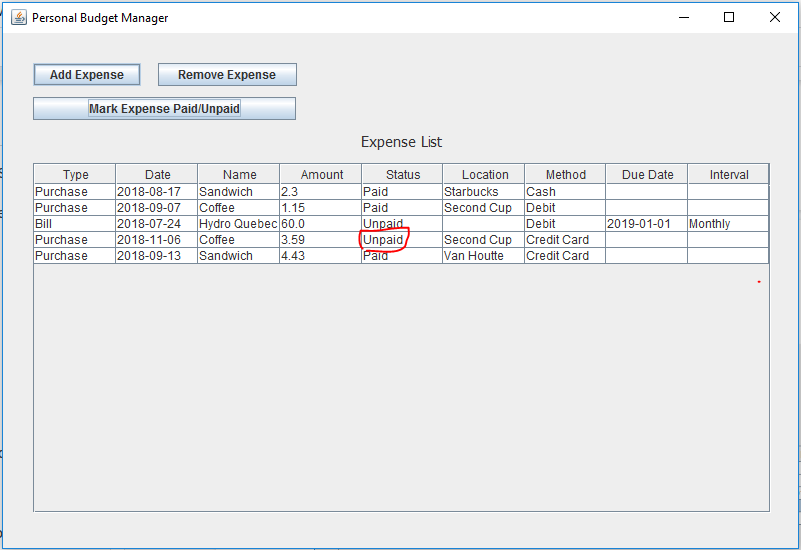


Figure 15. Mark paid/unpaid (b)

### 3.1.2 Hardware Interfaces

PC with at least 300 MHz Processors

Mouse

128 MB RAM

Storage requirements would be at least 10MB

No Internet connection required

### 3.1.3 Software Interfaces

The application should be opened on Eclipse.

## 3.2 Functional Requirements

1. Information inputted should be stored in database.
2. Information should be accessible via queries.
3. The expense table should refresh its display automatically when the expense in the list change.
4. If errors occur (amount entered is negative), there should be recovery mechanisms to solve them.

# 4. Other Non-functional Requirements

## 4.1 Performance Requirements

1. Efficiency – The product should have abovementioned functionalities. For good user experience, the system responding time must take less than 1 second.
2. Reliability – If errors occur such as wrong input and error deletion, there should be recovery mechanisms for all possible errors.
3. User friendliness – The graphical user interface should be clear and pleasing for good user experience. The operations should be designed in a way that is easy to understand and satisfy users’ needs.
4. Portability – The system should be run on different PCs with different operating systems.

## 4.2 Safety and Security Requirements

The system should have a level of security for the sensitivity of information to the database. To avoid misoperation, the confirmation request should always be provided. For further improvement, credit card information may also be included in the application, so we should make the database password-protected.