Test Document

Team PB-PK

8 April 2018

Name	ID Number
Noemi Lemonnier	40001085
Genevieve Plante-Brisebois	40003112
Han Gao	40053734
Theo Grimond	27276044
Real Nguyen	27566263
Ornela Bregu	26898580
William Prioriello	27080956
Tiantian Ji	27781083
Dong-Son Nguyen-Huu	40014054
Ashesh Patel	40018519
Sabrina Rieck	40032864

Table 1: Team

Contents

1 Introduction

The application that has been developed is a one user application that allow its user to track their expenses and to help maintain their budget.

1.1 Purpose

The purpose of the Test Document is to document the testing phase of our application. It provides the test plan and test results of the application including input and output files. The testing was implemented to insure that the requirements are followed. The requirements that are present for the application are:

- User can manage his cards
- User can manage a budget
- User can update his cash spending
- Only registered users are allowed to use the application
- source code is self documented with Javadoc-readable
- No unhandled exceptions from incorrect user input
- Main use cases must be accessible from the top screen
- Program is available 24/7
- Finished program can handle new user without needing to be rewritten or recompiled
- The application is failure free

1.2 Document Structure

The document is written within the limits of the following sections: the test plan, the test results, the references, the description of the input files and the description of the output files. Furthermore, in the test plan section, there are the following subsections: system level test cases, subsystem level test cases and the unit tests cases.

2 Test Plan

The following section describes the tests done on three different levels: system level, subsystem level and unit level. Each level was tested according to different requirements appropriate for that level and that were based on the assumption that the tests from the level below have passed.

It is to be noted that there are many possible types of tests that could have been executed in order to test the correctness of our program. However, due to lack of time and resources, these tests were not executed. Test that could have been execute consist of, but is not limited to, the following: performance profiling, load testing, configuration testing, installation testing, stress testing, volume testing, and security and access control testing.

2.1 System Level Test Cases

The tests that are shown in the current document for the system level aim to test that all the required actions are possible to take, they flow properly and to insure non functional requirements of the overall system are met.
The requirements for the system level testing are:

- Subsystem requirements are met
- User may navigate in between the subsystems with no issues
- Source code is self documented with Javadoc-readable
- Main use cases must be accessible from the main screen
- Program is available 24/7
- Finished program can handle new user without needing to be rewritten or recompiled
- The application is failure free

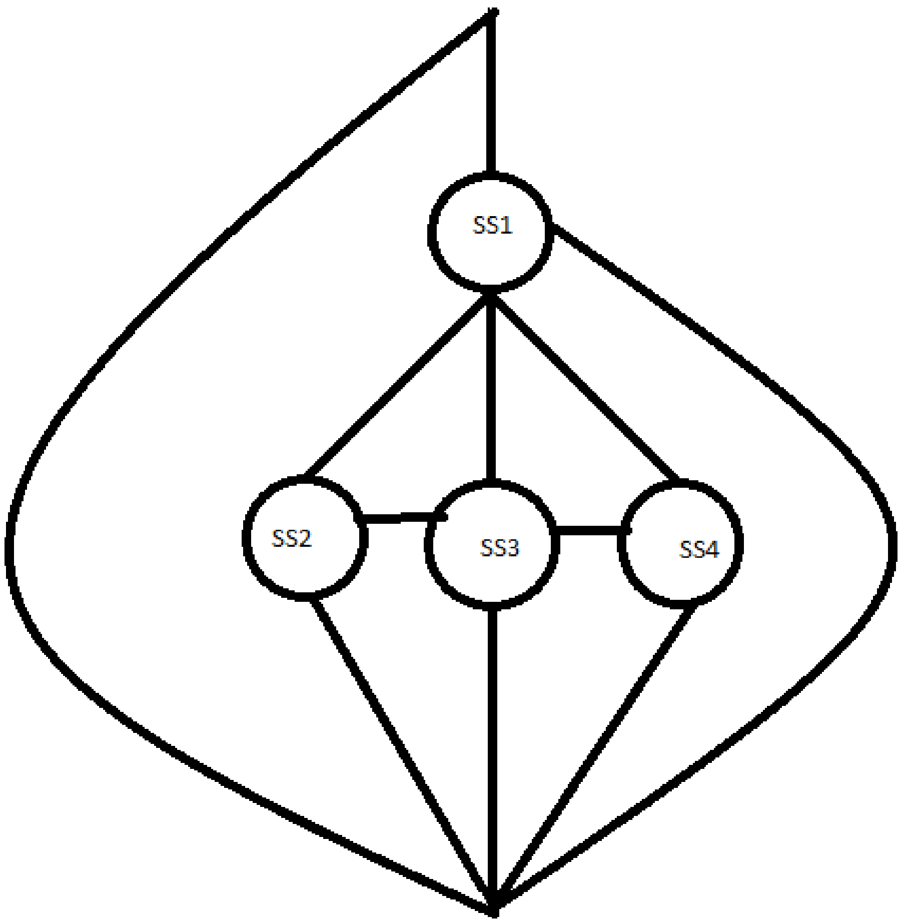
Listed below are all the potential paths a user can take within the system. The user has a choice between various actions. He can close the application from the start or identify himself and continue to the other features of the application. In each subsystem it is possible to close the application.

Note that each of the subsystem specific actions will be thoroughly defined in the next section about the subsystem testing.

1. Open the application
   - SS1. Authentication feature allows the user to identify himself
     - SS2. Clicking on the My Cards button in order to reach the My Cards subsystem
     - SS3. Clicking on the Cash Spending button in order to reach the Cash pending subsystem
     - SS4. Clicking on the Budget button in order to reach the Budget subsystem

It is also possible to navigate from one subsystem to another once passed the authentication subsystem. The authentication subsystem is the only subsystem to which it is not possible to go back to once the user has identified himself.

The possible paths are also here represented in this decision tree:



In the subsections below some test case have been put up with the steps to follow. As the testing is very extended and there is an infinite amount of possible paths of back and forth between the subsystem the main ones have been put in the documentation. For further details on each subsystem behavior the same process is covered for each subsystem further in the test document.

2.1.1 Subsystem requirements met

The specific subsystem requirements are going to be tested in details in an individual manner below.

2.1.2 Principle test cases for user navigation between subsystems

Test Case	Opening the application and closing it
Test Case Description	Clicking on the ”close” button of the application before going through the identification process
Input (Steps to produce test)	1. Launch the application 2. Click ”close” button
Output (Expected results)	The application closes

Test Case	Identify oneself then close the application
Test Case Description	Clicking the ”close” button immediately after identifying oneself
Input (Steps to produce test)	1. Launch the application 2. Go through the Authentification subsystem 3. Click on the ”close” button.
Output (Expected results)	The user has been identified but no further action has been taken

Test Case	Identify oneself, go through MyCards subsystem and close
Test Case Description	Clicking the ”close” button after going through the MyCards subsystem
Input (Steps to produce test)	1. Launch the application 2. Go through the Authentification subsystem 3. Click on the ”MyCards” button in order to reach the MyCards subsystem 4. Do actions in the MyCards subsystem 5. Click on the ”close” button
Output (Expected results)	The Autentification and MyCards subsystem works with no failure

Test Case	Go through Authentification and Budgetting subsystems and close
Test Case Description	Clicking the ”close” button after going through the Budgetting subsystem
Input (Steps to produce test)	1. Launch the application 2. Go through the Authentification subsystem 3. Click on the ”Budget” button 4. Do actions in the Budgetting subsystem 5. Click on the ”close” button
Output (Expected results)	The Authentification and Budget subsystems work with no failure

Test Case	Closing the application without being able to take action in the CashSpending subsystem as a new user
Test Case Description	Clicking the ”Close” after not being able to take action in the CashSpending subsystem
Input (Steps to produce test)	1. Launch the application 2. Go throught the Authentification subsystem as new user 3. Click on the ”CashSpendin” button 4. Click on the ”add expense” button 5. Close the error message asking to add a card in the system 6. Close the application
Output (Expected results)	The proper error message is displayed and no CashSpending is added.

Test Case	Go through the Authentification, MyCards and Budget subsystems
Test Case Description	Clicking the ”close” button after going through multiple subsystems.
Input (Steps to produce test)	1. Launch the application 2. Go through the Authentification subsystem 3. Click on the ”MyCards” button 4. Do actions in the MyCards subsystem 5. Click on the ”Budgeting” button 6. Do actions in Budgeting subsystem 7. Click on the ”close” button
Output (Expected results)	User is able to take actions in all mentionned subsystems with no trouble and close the application properly

Test Case	Go through the Authentification, MyCArds and CashSpending subsystems and then closing the application
Test Case Description	Clicking the ”close” button after taking some actions in the CashSpending subsystem
Input (Steps to produce test)	1. Launch the application 2. Go through the Authentification subsystem 3. Click on the ”MyCards” button 4. Do actions in the MyCards subsystem 5. Click on the ”CashSpending” button 6. If the user added a card, be able to enter expenses, else display of error message 7. Close the application
Output (Expected results)	There is a new expense only if the user previously added a card.

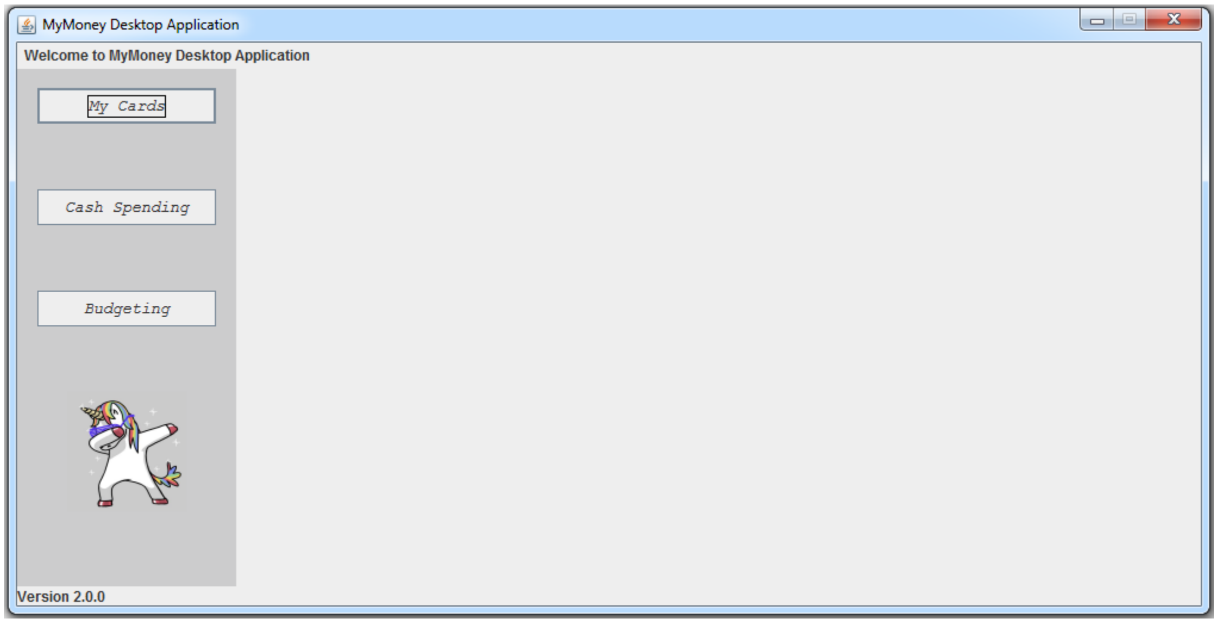
2.1.3 Non-functional requirements testing

The non-functional requirement testing, here is what has been done in order to ensure that they are met.

For the self documentation requirement, the Javadocs readable are produced by the embedded comments in the code which allows to create the Javadoc-readable documentation.

For the availability as well as the capacity for the a new user to use the program, the program has been tested with individuals outside of the development team. All results have been positive.

For the accessibility of the subsystem from the top screen, the team has designed the layout of the application in order to inherently respect that requirement. The user interface is as shown below:


MyMoney Desktop Application
Welcome to MyMoney Desktop Application
My Cards
Cash Spending
Budgeting
Version 2.0.0

No matter which subsystem the user enters, the column on the left will remain, allowing the user to access any subsystem.

As for the failure free, JUnit testing has been done for all elements in the application, allowing the development team to detect the failures, if any, and correct them.

2.2 Subsystem Level Test Cases

The following section contains the subsystem tests. Each subsystem is composed of one feature: Cards, Budget, Cash Spending and Authentication.

2.2.1 Subsystem Cards

The requirements for the Cards subsystem are as follows:

- User may add a card
- User may remove a card
- User may select a card to view transactions
- User may make a payment

Listed below are all the potential paths a user can take within the MyCards feature. The user has the choice between three main scenarios: adding a card to system, removing a card, or selecting a card. By clicking on the ”Cancel” button, the user will cancel whatever action they took and be returned to the main page for the subsystem. It is assumed that the user inputs correct input. Further details about correct input can be found in section 2.3 Unit Tests

1. Add a card
   2. Select add a debit card
      3. Click ”Cancel” button
      4. Click ”Okay” button
         5. Enter account number
         6. Enter card number
         7. Enter current amount of money on card
            8. Click ”Cancel” button
            9. Click ”Okay” button

   10. Select add a credit card
       11. Click ”Cancel” button
       12. Click ”Okay” button
           13. Enter account number
           14. Enter card number
           15. Enter amount already spent
           16. Enter credit limit
               17. Click ”Cancel” button
               18. Click ”Okay” button

19. Select add a loyalty card
    20. Click ”Cancel” button
    21. Click ”Okay” button
        22. Enter account email
        23. Enter card number
        24. Enter current amount of points
            25. Click ”Cancel” button
            26. Click ”Okay” button

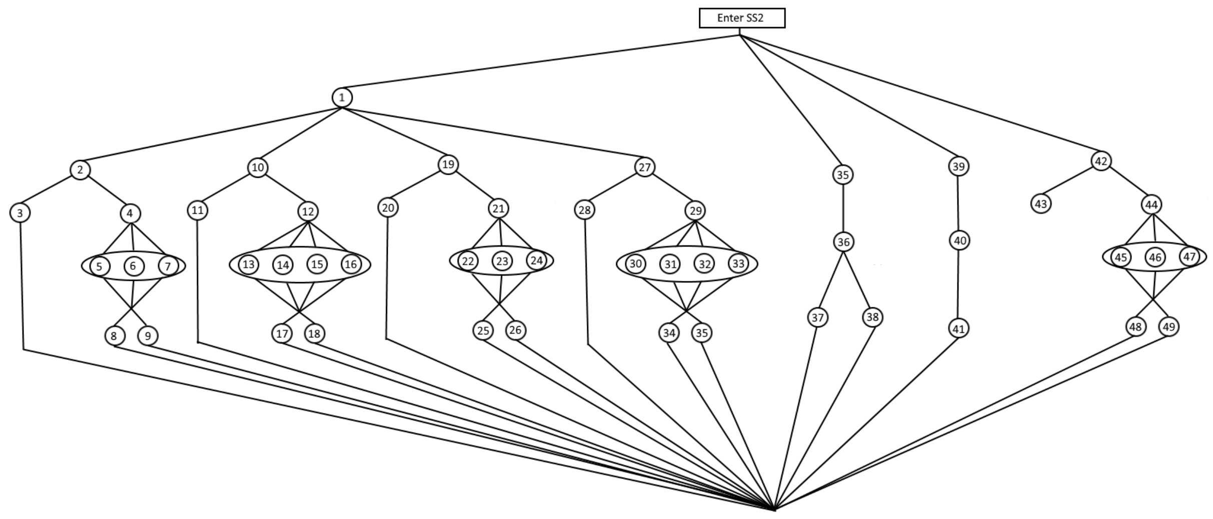
27. Select add a bitcoin card
    28. Click ”Cancel” button
    29. Click ”Okay” button
        30. Enter exchange account number
        31. Enter amount of mBTC already spent
        32. Enter card limit
            33. Click ”Cancel” button
            34. Click ”Okay” button

35. Remove a card
    36. Select card to remove
        37. Click ”Okay” button
        38. Click ”Cancel” button

39. Select a card
    40. Use scroll bars to view transactions done with this card
        41. Click ”Okay” button

42. Make a payment
    43. Click ”Cancel” button
    44. Click ”Okay” button
        45. Select a debit card
        46. Enter amount to transfer
        47. Select a credit card
            48. Click ”Cancel” button
            49. Click ”Okay” button

The following decision tree illustrates the above options. Once the user reaches the end of the tree, they may start at the beginning once more or move on to another subsystem.



The test cases described below all assume correct input as per the unit testing done in section 2.3 Unit Testing. If input is incorrect at any stage, an error message appears.

It is also important to note that when it comes to "selecting" an option, a default option is already preselected, meaning that the user cannot continue without a selected input. There is therefore no possibility of error there.

2.2.1.1 Adding a card

Test Case	Adding a credit card - with first cancellation
Test Case Description	Clicking the "Cancel" button immediately using the default selected card type
Input (Steps to produce test)	1. Click "Add a card" 2. Click "Cancel" button
Output (Expected results)	There is no new card added to the list

Test Case	Adding a card - with first input and cancellation
Test Case Description	Selecting the type of card wanted and clicking the "Cancel button"
Input (Steps to produce test)	1. Click "Add a card" 2. Select any type of card to add 3. Click "Cancel" button
Output (Expected results)	There is no new card added to the list

2.2.1.1.1 Adding a debit card

Test Case	Adding a debit card - full
Test Case Description	Going through the entire process of adding a debit card
Input (Steps to produce test)	1. Click "Add a card" 2. Select add a debit card 3. Click "Okay" button 4. Enter account number 5. Enter card number 6. Enter current amount of money on card 7. Click "Okay" button
Output (Expected results)	A new debit card is added to the list

Test Case	Adding a debit card - with all input and cancellation at the end
Test Case Description	Going through the entire process of adding a debit card and clicking the ”Cancel” button at the end
Input (Steps to produce test)	1. Click ”Add a card” 2. Select add a debit card 3. Click ”Okay” button 4. Enter account number 5. Enter card number 6. Enter current amount of money on card 7. Click ”Cancel” button
Output (Expected results)	There is no new card added to the list

Test Case	Adding a debit card - with only some input and second cancellation
Test Case Description	Only entering some of the required information to add a debit card and clicking the ”Cancel” button at the end
Input (Steps to produce test)	1. Click ”Add a card” 2. Select add a debit card 3. Click ”Okay” button 4. Enter only one or two of the following: (a) account number (b) card number (c) current amount of money on card 5. Click ”Cancel” button
Output (Expected results)	There is no new card added to the list

Test Case	Adding a debit card - with only some input and clicking ”Okay”
Test Case Description	Only entering some of the required information to add a debit card and clicking the ”Okay” button at the end
Input (Steps to produce test)	1. Click ”Add a card” 2. Select add a debit card 3. Click ”Okay” button 4. Enter only one or two of the following: (a) account number (b) card number (c) current amount of money on card 5. Click ”Okay” button
Output (Expected results)	An error message appears and there is no new card added to the list

2.2.1.1.2 Adding a credit card

Test Case	Adding a credit card - full
Test Case Description	Going through the entire process of adding a credit card
Input (Steps to produce test)	1. Click ”Add a card” 2. Select add a credit card 3. Click ”Okay” button 4. Enter account number 5. Enter card number 6. Enter amount already spent 7. Enter credit limit 8. Click ”Okay” button
Output (Expected results)	A new credit car is added to the list

Test Case	Adding a credit card - with all input and cancellation at the end
Test Case Description	Going through the entire process of adding a credit card and clicking the ”Cancel” button at the end
Input (Steps to produce test)	1. Click ”Add a card” 2. Select add a credit card 3. Click ”Okay” button 4. Enter account number 5. Enter card number 6. Enter amount already spent 7. Enter credit limit 8. Click ”Cancel” button
Output (Expected results)	There is no new card added to the list

Test Case	Adding a credit card - with only some input and cancelling at the end
Test Case Description	Only entering some of the required information to add a credit card and clicking the ”Cancel” button at the end
Input (Steps to produce test)	1. Click ”Add a card” 2. Select add a credit card 3. Click ”Okay” button 4. Enter only one, two or three of the following: (a) account number (b) card number (c) amount already spent (d) credit limit 5. Click ”Cancel” button
Output (Expected results)	There is no new card added to the list

Test Case	Adding a credit card - with only some input and clicking ”Okay”
Test Case Description	Only entering some of the required information to add a credit card and clicking the ”Okay” button at the end
Input (Steps to produce test)	1. Click ”Add a card” 2. Select add a credit card 3. Click ”Okay” button 4. Enter only one, two or three of the following: (a) account number (b) card number (c) amount already spent (d) credit limit 5. Click ”Okay” button
Output (Expected results)	An error message appears and there is no new card added to the list

2.2.1.1.3 Adding a loyalty card

Test Case	Adding a loyalty card - full
Test Case Description	Going through the entire process of adding a loyalty card
Input (Steps to produce test)	1. Click ”Add a card” 2. Select add a loyalty card 3. Click ”Okay” button 4. Enter account email 5. Enter card number 6. Enter current amount of points 7. Click ”Okay” button
Output (Expected results)	A new loyalty car is added to the list

Test Case	Adding a loyalty card - with all input and cancellation at the end
Test Case Description	Going through the entire process of adding a loyalty card and clicking the "Cancel" button at the end
Input (Steps to produce test)	1. Click "Add a card" 2. Select add a loyalty card 3. Click "Okay" button 4. Enter account email 5. Enter card number 6. Enter current amount of points 7. Click "Cancel" button
Output (Expected results)	There is no new card added to the list

Test Case	Adding a loyalty card - with only some input and cancellation at the end
Test Case Description	Only entering some of the required information to add a loyalty card and clicking the "Cancel" button at the end
Input (Steps to produce test)	1. Click "Add a card" 2. Select add a loyalty card 3. Click "Okay" button 4. Enter only one or two of the following: (a) account email (b) card number (c) current amount of points 5. Click "Cancel" button
Output (Expected results)	There is no new card added to the list

Test Case	Adding a loyalty card - with only some input and clicking ”okay”
Test Case Description	Only entering some of the required information to add a loyalty card and clicking the ”Okay” button at the end
Input (Steps to produce test)	1. Click ”Add a card” 2. Select add a loyalty card 3. Click ”Okay” button 4. Enter only one or two of the following: (a) account email (b) card number (c) current amount of points 5. Click ”Okay” button
Output (Expected results)	An error message appears and there is no new card added to the list

2.2.1.1.4 Adding a bitcoin card

Test Case	Adding a bitcoin card - full
Test Case Description	Going through the entire process of adding a bitcoin card
Input (Steps to produce test)	1. Click ”Add a card” 2. Select add a bitcoin card 3. Click ”Okay” button 4. Enter exchange account number 5. Enter card number 6. Enter amount of mBTC already spent 7. Enter card limit 8. Click ”Okay” button
Output (Expected results)	A new bitcoin card is added to the list

Test Case	Adding a bitcoin card - with all input and cancellation at the end
Test Case Description	Going through the entire process of adding a bitcoin card and clicking the "Cancel" button at the end
Input (Steps to produce test)	1. Click "Add a card" 2. Select add a debit card 3. Click "Okay" button 4. Enter exchange account number 5. Enter card number 6. Enter amount of mBTC already spent 7. Enter card limit 8. Click "Cancel" button
Output (Expected results)	There is no new card added to the list

Test Case	Adding a bitcoin card - with only some input and second cancellation
Test Case Description	Only entering some of the required information to add a bitcoin card and clicking the "Cancel" button at the end
Input (Steps to produce test)	1. Click "Add a card" 2. Select add a bitcoin card 3. Click "Okay" button 4. Enter only one or two of the following: (a) Enter exchange account number (b) Enter card number (c) Enter amount of mBTC already spent (d) Enter card limit 5. Click "Cancel" button
Output (Expected results)	There is no new card added to the list

Test Case	Adding a bitcoin card - with only some input and clicking ”Okay”
Test Case Description	Only entering some of the required information to add a bitcoin card and clicking the ”Okay” button at the end
Input (Steps to produce test)	1. Click ”Add a card” 2. Select add a bitcoin card 3. Click ”Okay” button 4. Enter only one or two of the following: (a) Enter exchange account number (b) Enter card number (c) Enter amount of mBTC already spent (d) Enter card limit 5. Click ”Okay” button
Output (Expected results)	An error message appears and there is no new card added to the list

2.2.1.2 Removing a card

Test Case	Removing a card - full
Test Case Description	Removing a card
Input (Steps to produce test)	1. Click ”Remove a card” 2. Select a card to remove 3. Click ”Okay”
Output (Expected results)	A card is removed from the list

Test Case	Removing a card - without selecting
Test Case Description	Cancelling the removal of a card while on the default option
Input (Steps to produce test)	1. Click ”Remove a card” 2. Click ”Cancel”
Output (Expected results)	No card is removed from the list

Test Case	Removing a card - with selecting
Test Case Description	Cancelling the removal of a card after selecting a card to remove
Input (Steps to produce test)	1. Click "Remove a card" 2. Select a card to remove 3. Click "Cancel"
Output (Expected results)	No card is removed from the list

2.2.1.3 Selecting a card

Test Case	Selecting a card - full
Test Case Description	Selecting a card and scrolling to view the transactions done with this card
Input (Steps to produce test)	1. Check the box of a card to select 2. Use scroll bars to view transactions done with this card 3. Click "Okay" button
Output (Expected results)	A window opens to allow the user to view transactions and closes after the "Okay" button is clicked

Test Case	Selecting a card - without scrolling
Test Case Description	Selecting a card to view the transactions done with this card, but without scrolling
Input (Steps to produce test)	1. Check the box of a card to select 2. Click "Okay" button
Output (Expected results)	A window opens to allow the user to view transactions and closes after the "Okay" button is clicked

2.2.1.4 Making a payment

Test Case	Making a payment - full
Test Case Description	Going through the entire process of adding a bitcoin card and clicking the "Okay" button at the end
Input (Steps to produce test)	1. Make a payment 2. Select a debit card 3. Enter amount to transfer 4. Select a credit card 5. Click "Okay" button
Output (Expected results)	The appropriate changes are made to the "amount" column of the table

Test Case	Making a payment - full with cancellation
Test Case Description	Going through the entire process of adding a bitcoin card and clicking the "Cancel" button at the end
Input (Steps to produce test)	1. Make a payment 2. Select a debit card 3. Enter amount to transfer 4. Select a credit card 5. Click "Cancel" button
Output (Expected results)	No changes are made

Test Case	Making a payment - incomplete
Test Case Description	Going through the partial process of adding a bitcoin card and clicking the "Okay" button at the end
Input (Steps to produce test)	1. Make a payment 2. Click "Okay" button
Output (Expected results)	An error message appears and no changes are made

Test Case	Making a payment - incomplete with cancellation
Test Case Description	Going through the partial process of adding a bitcoin card and clicking the "Cancel" button at the end
Input (Steps to produce test)	1. Make a payment 2. Click "Cancel" button
Output (Expected results)	No changes are made

2.2.2 Subsystem Budgeting

The requirements for the Budgeting subsystem are as follows:

- User may calculate budget
- User may change percentage of budgeting
- User may view existing budget
- User may remove the budget
- User may print budget to file
- User may save budget

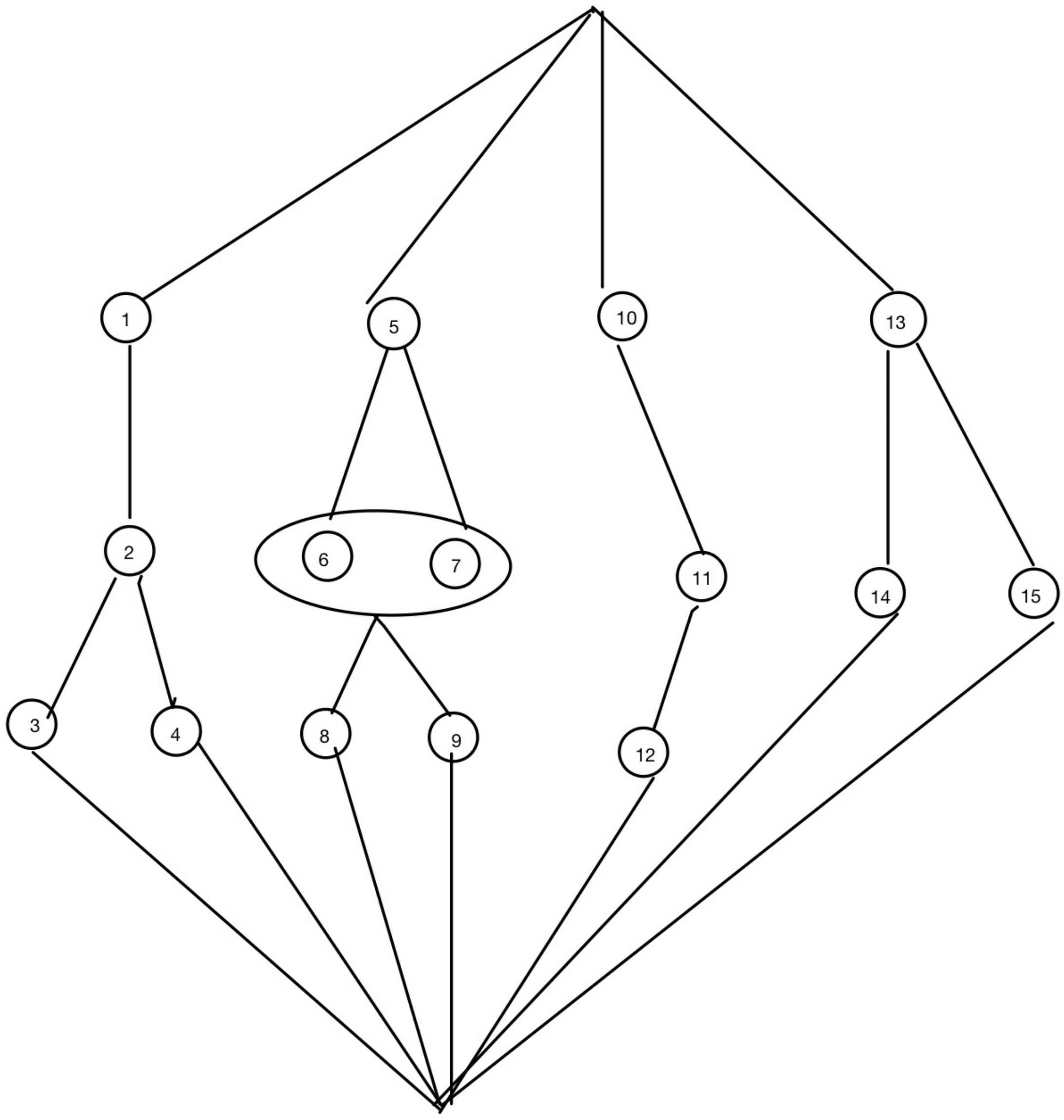
Listed below are all the potential paths a user can take within the Budgeting feature. Four main scenarios are provided for users to choose: calculate budget, modify budget, view existing budget, or clear the existing budget. By clicking on the "Cancel" button, the user will cancel whatever action they took and be returned to the main page for the subsystem.

```
1. Select "Calculate budget"
    2. Enter available funds
        3. Click "Cancel" button to not save budget
        4. Click "Okay" button to save budget

5. Select "Change percentages"
    6. Enter available funds
    7. Enter percentages for each category
        8. Click "Cancel" button
        9. Click "Okay" button

10. Select "My Budget"
    11. Invalid input exception windows appears if no budgeting exists
```

        12. Click "Okay" button to close window

13. Select "Clear Budget"
    14. Click "Okay" button to delete current budget
    15. Click "Cancel" button to cancel deleting current budget

The following decision tree illustrates the above options. Once the user reaches the end of the tree, they may start at the beginning once more or move on to another subsystem.



The test cases described below all assume correct input as per the unit testing done in section 2.3 Unit Testing. If input is incorrect at any stage, an error message appears.

It is also important to note that when it comes to "selecting" an option, a default option is already preselected, meaning that the user cannot continue without a selected input.

There is therefore no possibility of error there.

2.2.2.1 Calculate budget

Test Case	Calculate budget - with first cancellation
Test Case Description	Clicking the ”Cancel” button after the default window appears
Input (Steps to produce test)	1. Click ”Calculate budget” 2. Click ”Cancel” button
Output (Expected results)	There is no new budget being calculated

Test Case	Calculate budget - with first input and cancellation
Test Case Description	Entering the available amount of funds and clicking the ”Cancel button”
Input (Steps to produce test)	1. Click ”Calculate budget” 2. Enter available funds 3. Click ”Cancel” button
Output (Expected results)	There is no new budget being calculated

Test Case	Calculate budget - full
Test Case Description	Entering the available amount of funds and clicking the ”Okay button”
Input (Steps to produce test)	1. Click ”Calculate budget” 2. Enter available funds 3. Click ”Okay” button
Output (Expected results)	There is no new budget being calculated

2.2.2.2 Change percentages

Test Case	Change budget percentages - with first cancellation
Test Case Description	Change budget percentages
Input (Steps to produce test)	1. Click ”Change percentages” 2. Click ”Cancel”
Output (Expected results)	Budgeting list is not changed

Test Case	Change budget percentages - with first input and cancellation
Test Case Description	Change budget percentages
Input (Steps to produce test)	1. Click ”Change percentages” 2. Enter percentages 3. Click ”Cancel”
Output (Expected results)	Budgeting list is not changed

Test Case	Change budget percentages - full
Test Case Description	Change budget percentages
Input (Steps to produce test)	1. Click ”Change percentages” 2. Enter percentages 3. Click ”Okay”
Output (Expected results)	Budgeting list is changed

2.2.2.3 View my budget

Test Case	View My Budget - full
Test Case Description	Clicking the ”My Budget” button to view the list
Input (Steps to produce test)	1. Click ”My Budget” button
Output (Expected results)	The list of budgets is shown

2.2.2.4 Clear Budget

Test Case	Clear Budget - full
Test Case Description	Clicking the "Clear Budget" button to empty the budget list
Input (Steps to produce test)	1. Click "Clear Budget" button
Output (Expected results)	The list of budgets is cleared

2.2.3 Subsystem Cash Spending

The requirements for the Cash Spending subsystem are:

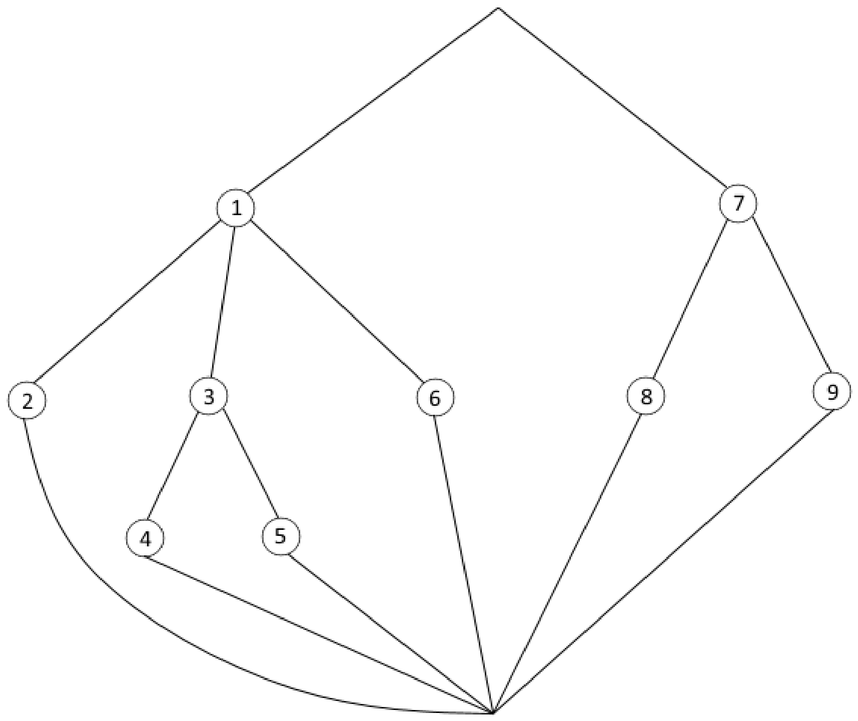
- User may add an expense
- User may be able to view all of his expenses

Listed below are all the potential paths a user can take within the Cash Spending feature. Two main scenarios are provided for users to choose: "Add an expense", and "Show all expenses". By clicking on the "Cancel" button in either of the windows shown by the "Add an expense" or "Show all expenses" scenarios, the user will cancel whatever action they took and be returned to the main page for the subsystem.

1. Select "Add an expense"
    2. Select expense type
    3. Enter amount
        4. Click "OK"
        5. Click "Cancel"

6. Select "Show all expenses"
    7. Click "OK"
    8. Click "Cancel"

The following decision tree illustrates the above options. Once the user reaches the end of the tree, they may start at the beginning once more or move on to another subsystem.



The test cases described below all assume correct input as per the unit testing done in section 2.3 Unit Testing, i.e. assume that all the values entered in the amount input field are non-empty numeric values. If input is incorrect at any stage, an error message appears.

It is also important to note that when it comes to "selecting" an option, a default option is already preselected, meaning that the user cannot continue without a selected input. In the case of the expense type, the default value is "HOUSING", and in the case of the selected card, the default value is the first card added by the user. The user cannot enter anything outside the predefined expense types, or cards that have not been created by the user. There is therefore no possibility of error there.

2.2.3.1 Adding an expense

Test Case	Add an expense - Cards not loaded yet
Test Case Description	Click on ”Add an expense” without first having loaded the cards.
Input (Steps to produce test)	1. Click on ”Add an expense”
Output (Expected results)	Error message is shown saying ”Please load your cards on clicking on My Cards tab”

Test Case	Add an expense - Amount
Test Case Description	Add an expense with a correct amount, then click ”OK”
Input (Steps to produce test)	1. Click ”Add an expense” 2. Add an amount in input field 3. Click ”OK”
Output (Expected results)	Expense successfully added and written to file

Test Case	Add an expense - Budgeting file missing
Test Case Description	Click on ”Add an expense” while the budgeting file is missing
Input (Steps to produce test)	1. Click ”Add an expense”
Output (Expected results)	System crashes

Test Case	Add an expense - Budgeting file empty
Test Case Description	Click on ”Add an expense” while the budgeting file is empty
Input (Steps to produce test)	1. Click ”Add an expense”
Output (Expected results)	All expenses will show a confirmation message to alert that the user is going over budget until budgeting is set

Test Case	Add an expense - Change type
Test Case Description	Adding an expense with a different type than the default one
Input (Steps to produce test)	1. Click ”Add an expense” 2. Change the expense type 3. Add an amount in input field 4. Click ”OK”
Output (Expected results)	Expense successfully added and written to file

Test Case	Add an expense - Change card
Test Case Description	Adding an expense with a different card than the default one
Input (Steps to produce test)	1. Click "Add an expense" 2. Change the card 3. Add an amount in input field 4. Click "OK"
Output (Expected results)	Expense successfully added and written to file

Test Case	Add an expense - Over budget, confirm
Test Case Description	Add an expense that is over its respective budget, confirm that the user wants to go over budget, proceed with adding expense
Input (Steps to produce test)	1. Click "Add an expense" 2. Add an amount in input field 3. Click "OK" on "Alert Going Over Budget" window
Output (Expected results)	Expense is successfully added and written to file, and a warning message is written to file and shown in the "Show all expenses" section to alert that the user has gone over budget

Test Case	Add an expense - Over budget, cancel
Test Case Description	Add an expense that is over its respective budget, and confirm that the user wants to go over budget, cancel
Input (Steps to produce test)	1. Click "Add an expense" 2. Add an amount in input field 3. Click "Cancel" on "Alert Going Over Budget" window
Output (Expected results)	No expense is added

2.2.3.2 Show expenses

Test Case	Show all expenses - no expenses
Test Case Description	Show all expenses when no expenses have been added
Input (Steps to produce test)	1. Click "Show all expenses"
Output (Expected results)	Area where expenses are shown is empty

Test Case	Show all expenses - expenses have already been added
Test Case Description	Show all expenses when one or more expense has been added
Input (Steps to produce test)	1. Click ”Show all expenses”
Output (Expected results)	Transactions are listed from oldest to newest, with the transaction number, expense type, amount, and card used

Test Case	Show all expenses - over budget expenses have been added
Test Case Description	Show all expenses when one or more over budget expense has been added
Input (Steps to produce test)	1. Click ”Show all expenses”
Output (Expected results)	A warning message is shown with the amount, expense type, and card used, along with the regular transaction message (see test case above) below it

2.2.4 Subsystem Authentication

The requirements for the Authentification subsystem are:

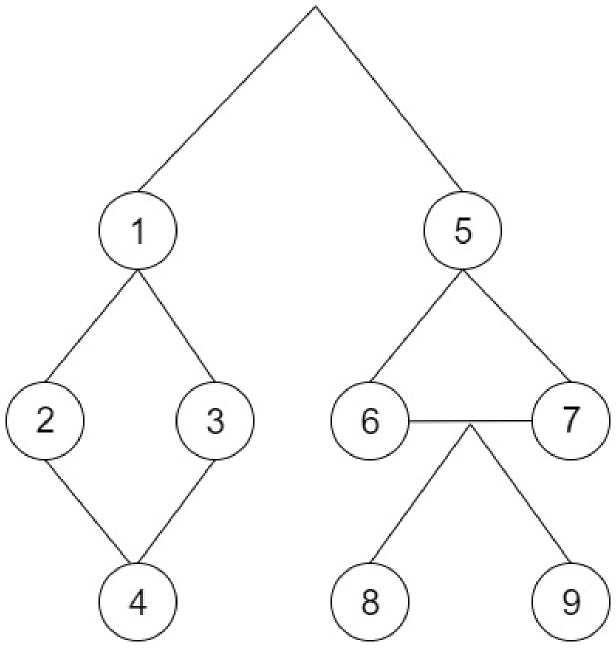
- User may sign in with an existing identification
- A new user may create his authentification parameters and create his session

Listed below are all the potential paths a user can take within the Authentication feature. Four main scenarios are provided for users to choose: Sign in, Add new user. By clicking on the ”Cancel” button, the user will cancel whatever action they took and be returned to the main page for the subsystem.

1. Try ”Sign in”
    2. Enter available user name
    3. Enter available password
        4. Click ”Sign in” button to sign in

5. Click ”New User”
    6. Enter new user name
    7. Enter new password
        8. Click ”Cancel” button
        9. Click ”Okay” button

The following decision tree illustrates the above options. Once the user reaches the end of the tree, they may start at the beginning once more or move on to another subsystem.



The test cases described below all assume correct input as per the unit testing done in section 2.3 Unit Testing. If input is incorrect at any stage, an error message appears.

It is also important to note that when it comes to "selecting" an option, a default option is already preselected, meaning that the user cannot continue without a selected input. There is therefore no possibility of error there.

2.2.4.1 Sign in

Test Case	Sign in - empty
Test Case Description	Sign in with empty user name and password
Input (Steps to produce test)	1. Enter empty user name 2. Enter empty password 3. Click "Sign In" button
Output (Expected results)	Invalid Input Alert-This user login is not exist, please try again.

Test Case	Sign in - invalid
Test Case Description	Sign in with invalid user name and password
Input (Steps to produce test)	1. Enter invalid user name 2. Enter invalid password 3. Click ”Sign In” button
Output (Expected results)	Invalid Input Alert-This user login is not exist, please try again.

Test Case	Sign in - available
Test Case Description	Sign in with correct user name and password
Input (Steps to produce test)	1. Enter right user name 2. Enter right password 3. Click ”Sign In” button
Output (Expected results)	Sign in successfully.

2.2.4.2 Add new user

Test Case	Add new user - empty
Test Case Description	Add new user with empty user name and password
Input (Steps to produce test)	1. Click ”add new user”button 2. Enter empty user name 3. Enter empty password 4. Click ”ok” button
Output (Expected results)	Invalid Input Alert-You have entered an invalid value or a duplicate, please try again.

Test Case	Add new user - duplicate
Test Case Description	Add new user with same user name and password
Input (Steps to produce test)	1. Click "add new user"button 2. Enter user name 3. Enter user name as password 4. Click "ok" button
Output (Expected results)	Invalid Input Alert-You have entered an invalid value or a duplicate, please try again.

Test Case	Add new user - available
Test Case Description	Add new user with available user name and password
Input (Steps to produce test)	1. Click "add new user"button 2. Enter available user name 3. Enter available password 4. Click "ok" button
Output (Expected results)	Add user and sign in successfully.

2.3 Unit Test cases

The unit test documented here is for boundary testing. The tests are classified per subsystem.

2.3.1 Cards

The following tables represent the tests done for all textboxes that users can input information into from the Cards feature. They tests the bounds of possible inputs a user could enter and identify both correct and incorrect input as well as how the system reacts to such input.

Class name	MyCardsUI	Method name	actionPerformed(ActionEvent 0)		
Variable name	accNb	Lower bound	0	Upper bound	9999
	Value	Expected output	Actual output	Bug found?	
Below lower bound	-1	Throws NumberFormatException()	Throws NumberFormatException()	No	
On lower bound	0	Accepted	Accepted	No	
Between bounds	5555	Accepted	Accepted	No	
On upper bound	9999	Accepted	Accepted	No	
Above upper bound	10000	Throws NumberFormatException()	Throws NumberFormatException()	No	
String input	abcd	Throws NumberFormatException()	Throws NumberFormatException()	No	

Class name	MyCardsUI	Method name	actionPerformed(ActionEvent 0)		
Variable name	cardNum	Lower bound	0	Upper bound	99999999
	Value	Expected output	Actual output	Bug found?	
Below lower bound	-1	Throws NumberFormatException()	Throws NumberFormatException()	No	
On lower bound	0	Accepted	Accepted	No	
Between bounds	5555	Accepted	Accepted	No	
On upper bound	99999999	Accepted	Accepted	No	
Above upper bound	100000000	Throws NumberFormatException()	Throws NumberFormatException()	No	
String input	abcd	Throws NumberFormatException()	Throws NumberFormatException()	No	

Class name	MyCardsUI	Method name	actionPerformed(ActionEvent 0)		
Variable name	moneySpent	Lower bound	0	Upper bound	None
	Value	Expected output	Actual output	Bug found?	
Below lower bound	-1	Throws NumberFormatException()	Throws NumberFormatException()	No	
On lower bound	0	Accepted	Accepted	No	
Between bounds	5555	Accepted	Accepted	No	
String input	abcd	Throws NumberFormatException()	Throws NumberFormatException()	No	

Class name	MyCardsUI	Method name	actionPerformed(ActionEvent 0)		
Variable name	limitCard	Lower bound	0	Upper bound	None
	Value	Expected output	Actual output	Bug found?	
Below lower bound	-1	Throws NumberFormatException()	Throws NumberFormatException()	No	
On lower bound	0	Accepted	Accepted	No	
Between bounds	5555	Accepted	Accepted	No	
String input	abcd	Throws NumberFormatException()	Throws NumberFormatException()	No	

Class name	MyCardsUI	Method name	actionPerformed(ActionEvent 0)		
Variable name	pointsAvaila	Lower bound	0	Upper bound	None
	Value	Expected output	Actual output	Bug found?	
Below lower bound	-1	Throws NumberFormatException()	Throws NumberFormatException()	No	
On lower bound	0	Accepted	Accepted	No	
Between bounds	5555	Accepted	Accepted	No	
String input	abcd	Throws NumberFormatException()	Throws NumberFormatException()	No	

Class name	MyCards	Method name	isValid(String email)	
Variable name	email	Requires	at least one char, followed by @, min 1 char, a period, min 1 char	
	Value	Expected output	Actual output	Bug found?
Incomplete input	a	False	False	No
Incomplete input	a@	False	False	No
Incomplete input	a@b	False	False	No
Incomplete input	@b	False	False	No
Incomplete input	a@b.	False	False	No
Incomplete input	a@b.c	True	True	No

Class name	MyCardsUI	Method name	actionPerformed(ActionEvent 0)		
Variable name	AmountToP	Lower bound	0	Upper bound	Depends on selected debit's card available funds (Say 500)
	Value	Expected output	Actual output	Bug found?	
Below lower bound	-1	Throws NumberFormatException()	Throws NumberFormatException()	No	
On lower bound	0	Accepted	Accepted	No	
On lower bound	100	Accepted	Accepted	No	
Between bounds	500	Accepted	Accepted	No	
Between bounds	501	Throws NumberFormatException()	Throws NumberFormatException()	No	
String input	abcd	Throws NumberFormatException()	Throws NumberFormatException()	No	

2.3.2 Budget

The following tables represent the tests done for all textboxes that users can input information into from the Budgeting feature. They tests the bounds of possible inputs a user could enter and identify both correct and incorrect input as well as how the system reacts to such input.

Class name	BudgetingUI	Method name	actionPerformed(ActionEvent 0)		
Variable name	fundsAvailable	Lower bound	0	Upper bound	none
	Value	Expected output	Actual output	Bug found?	
Below lower bound	-1	Throws NumberFormatException()	Throws NumberFormatException()	No	
On lower bound	0	Accepted	Accepted	No	
Between bounds	50	Accepted	Accepted	No	
On upper bound	100	Accepted	Accepted	No	
Above upper bound	100000000	Throws NumberFormatException()	Throws NumberFormatException()	No	
String input	abcd	Throws NumberFormatException()	Throws NumberFormatException()	No	

Class name	Budgeting	Method name	testCalculateFood()		
Variable name	percentFood	Lower bound	0	Upper bound	100
	Value	Expected output	Actual output	Bug found?	
Below lower bound	-1	Throws NumberFormatException()	Throws NumberFormatException()	No	
On lower bound	0	Accepted	Accepted	No	
Between bounds	50	Accepted	Accepted	No	
On upper bound	100	Accepted	Accepted	No	
Above upper bound	105	Throws NumberFormatException()	Throws NumberFormatException()	No	
String input	abcd	Throws NumberFormatException()	Throws NumberFormatException()	No	

Class name	Budgeting	Method name	testCalculateClothing()		
Variable name	percentClothin	Lower bound	0	Upper bound	100
	Value	Expected output	Actual output	Bug found?	
Below lower bound	-1	Throws NumberFormatException()	Throws NumberFormatException()	No	
On lower bound	0	Accepted	Accepted	No	
Between bounds	50	Accepted	Accepted	No	
On upper bound	100	Accepted	Accepted	No	
Above upper bound	100000000	Throws NumberFormatException()	Throws NumberFormatException()	No	
String input	abcd	Throws NumberFormatException()	Throws NumberFormatException()	No	

Class name	Budgeting	Method name	testCalculateMedical()		
Variable name	percentMedical	Lower bound	0	Upper bound	100
	Value	Expected output	Actual output	Bug found?	
Below lower bound	-1	Throws NumberFormatException()	Throws NumberFormatException()	No	
On lower bound	0	Accepted	Accepted	No	
Between bounds	50	Accepted	Accepted	No	
On upper bound	100	Accepted	Accepted	No	
Above upper bound	100000000	Throws NumberFormatException()	Throws NumberFormatException()	No	
String input	abcd	Throws NumberFormatException()	Throws NumberFormatException()	No	

Class name	Budgeting	Method name	testCalculateDonation()		
Variable name	percentDonatic	Lower bound	0	Upper bound	100
	Value	Expected output	Actual output	Bug found?	
Below lower bound	-1	Throws NumberFormatException()	Throws NumberFormatException()	No	
On lower bound	0	Accepted	Accepted	No	
Between bounds	50	Accepted	Accepted	No	
On upper bound	100	Accepted	Accepted	No	
Above upper bound	100000000	Throws NumberFormatException()	Throws NumberFormatException()	No	
String input	abcd	Throws NumberFormatException()	Throws NumberFormatException()	No	

Class name	Budgeting	Method name	testCalculateMisc()		
Variable name	percentMisc	Lower bound	0	Upper bound	100
	Value	Expected output	Actual output	Bug found?	
Below lower bound	-1	Throws NumberFormatException()	Throws NumberFormatException()	No	
On lower bound	0	Accepted	Accepted	No	
Between bounds	50	Accepted	Accepted	No	
On upper bound	100	Accepted	Accepted	No	
Above upper bound	100000000	Throws NumberFormatException()	Throws NumberFormatException()	No	
String input	abcd	Throws NumberFormatException()	Throws NumberFormatException()	No	

Class name	Budgeting	Method name	testCalculateTransport()		
Variable name	percentTranspo	Lower bound	0	Upper bound	100
	Value	Expected output	Actual output	Bug found?	
Below lower bound	-1	Throws NumberFormatException()	Throws NumberFormatException()	No	
On lower bound	0	Accepted	Accepted	No	
Between bounds	50	Accepted	Accepted	No	
On upper bound	100	Accepted	Accepted	No	
Above upper bound	100000000	Throws NumberFormatException()	Throws NumberFormatException()	No	
String input	abcd	Throws NumberFormatException()	Throws NumberFormatException()	No	

Class name	Budgeting	Method name	testCalculateEntertainment()		
Variable name	percentEnterta	Lower bound	0	Upper bound	100
	Value	Expected output	Actual output	Bug found?	
Below lower bound	-1	Throws NumberFormatException()	Throws NumberFormatException()	No	
On lower bound	0	Accepted	Accepted	No	
Between bounds	50	Accepted	Accepted	No	
On upper bound	100	Accepted	Accepted	No	
Above upper bound	100000000	Throws NumberFormatException()	Throws NumberFormatException()	No	
String input	abcd	Throws NumberFormatException()	Throws NumberFormatException()	No	

Class name	Budgeting	Method name	testCalculateUtilities()		
Variable name	percentUtilities	Lower bound	0	Upper bound	100
	Value	Expected output	Actual output	Bug found?	
Below lower bound	-1	Throws NumberFormatException()	Throws NumberFormatException()	No	
On lower bound	0	Accepted	Accepted	No	
Between bounds	50	Accepted	Accepted	No	
On upper bound	100	Accepted	Accepted	No	
Above upper bound	100000000	Throws NumberFormatException()	Throws NumberFormatException()	No	
String input	abcd	Throws NumberFormatException()	Throws NumberFormatException()	No	

Class name	Budgeting	Method name	testCalculateSaving()		
Variable name	percentSavingI	Lower bound	0	Upper bound	100
	Value	Expected output	Actual output	Bug found?	
Below lower bound	-1	Throws NumberFormatException()	Throws NumberFormatException()	No	
On lower bound	0	Accepted	Accepted	No	
Between bounds	50	Accepted	Accepted	No	
On upper bound	100	Accepted	Accepted	No	
Above upper bound	100000000	Throws NumberFormatException()	Throws NumberFormatException()	No	
String input	abcd	Throws NumberFormatException()	Throws NumberFormatException()	No	

2.3.3 Cash Spending

The following tables represent the tests done for all textboxes that users can input information into from the Cash Spending feature. They tests the bounds of possible inputs a user could enter and identify both correct and incorrect input as well as how the system reacts to such input

Class name	CashSpendi	Method name	addingToTheExpenditure		
Variable name	amount	Lower bound	0	Upper bound	150
	Value	Expected output	Actual output	Bug found?	
Below lower bound	-1	Error message is shown	Error message is shown	No	
On lower bound	0	Value accepted	Value accepted	No	
Between bounds	100	Value accepted	Value accepted	No	
On upper bound	150	Value accepted	Value accepted	No	
Above upper bound	151	Confirmation message is shown	Confirmation message is shown	No	

2.3.4 Authentication

The following tables represent the tests done for all textboxes that users can input information into from the Authentication feature. They tests the bounds of possible inputs a usercould enter and identify both correct and incorrect input as well as how the system reacts to such input.

Class name	Authentification	Method name	Assert.assertEquals(message,expected,actual)		
Variable name	username and password	Lower bound	none	Upper bound	none
	Value	Expected output	Actual output	Bug found?	
Default Constructor	username empty password empty	Equal	Equal	No	
Constructor	username lol password 1234	Equal	Equal	No	
Set Get username	username lol	Equal	Equal	No	
Set Get Password	password 1234	Equal	Equal	No	

Class name	Authentification List	Method name	Assert.assertEquals(message,expected,actual)		
Variable name	userslist	Lower bound	none	Upper bound	none
	Value	Expected output	Actual output	Bug found?	
Constructor Getter	userlist	get userlist	get userlist	No	
Add User	username edws	Equal	Equal	No	
Remove User	username qswf	Equal	Equal	No	
Get User Index	userlist	Equal	Equal	No	

Class name	AuthentificationUI	Method name	actionPerformed(ActionEvent 0)		
Variable name	username	Lower bound	none	Upper bound	none
	Value	Expected output	Actual output	Bug found?	
Empty input	empty	Throws FormatException()	Throws FormatException()	No	
Number input	12475	Accepted	Accepted	No	
String input	abc	Accepted	Accepted	No	
Mixed input	abc100	Accepted	Accepted	No	

Class name	AuthentificationUI	Method name	actionPerformed(ActionEvent 0)		
Variable name	password	Lower bound	none	Upper bound	none
	Value	Expected output	Actual output	Bug found?	
Empty input	empty	Throws FormatException()	Throws FormatException()	No	
Number input	12475	Accepted	Accepted	No	
String input	abc	Accepted	Accepted	No	
Mixed input	abc100	Accepted	Accepted	No	

3 Test Results

Here are the test results of the application. They are classified per system level, subsystem level and overall application.

3.1 System Level Testing

Program	MyMoney	
		Percentage of tests passed
Requirements	Subsystem requirements met	100%
	Navigation between subsystems	100%
	Self documentation with Javadoc	100%
	Features easy to access	100%
	Available 24/7	100%
	Can handle new users	100%
	Failure free	100%

3.2 Subsystem Level Testing

Feature	Cards						
Requirements	Add a card				Remove a card	Make payment	Select a card
Percentage of tests passed	100%				100%	100%	100%
	Debit	Credit	Loyalty	Bitcoin			
	100%	100%	100%	100%			

Feature	Budgeting					
Requirements	Calculate budget	Custom percentages	Save budget	Remvoe budget	View budget	Print budget to file
Percentage of tests passed	100%	100%	100%	100%	100%	100%

Feature	Cash Spending				
Requirements	Add an expense				View all expenses
Percentage of tests passed	100%				100%
	Housing	Food	Utilities	Clothing	Medical
	100%	100%	100%	100%	100%
	Donations	Savings	Entertainem	Transportati	Misc
	100%	100%	100%	100%	100%

Feature	Authentication	
Requirements	Sign in with existing account	Create new account
Percentage of tests passed	100%	100%

3.3 Overall Application

Program	MyMoney				
Requirements	System				
		Features			
		Authentication	Budget	Cash Spending	Cards
Percentage of tests passed	100%	100%	100%	100%	100%

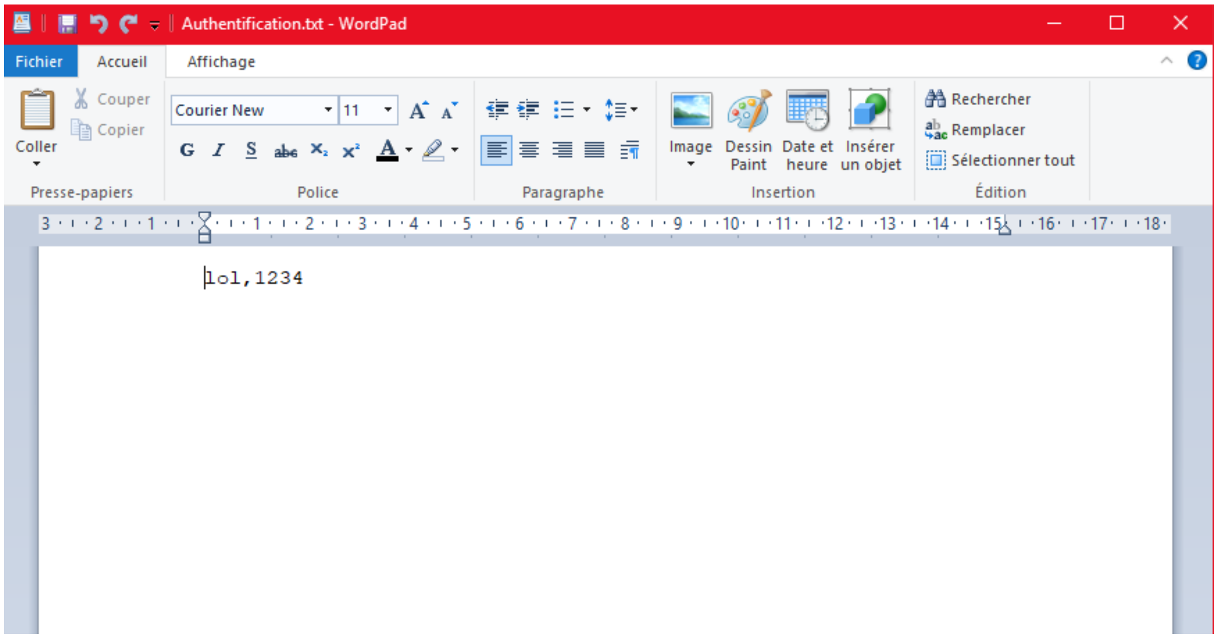
4 References

Team Redmond. *Master Test Plan.* Accessed March 21st at http://users.encs.concordia.ca/ paquet/wiki/images/3/35/Phase3final.pdf

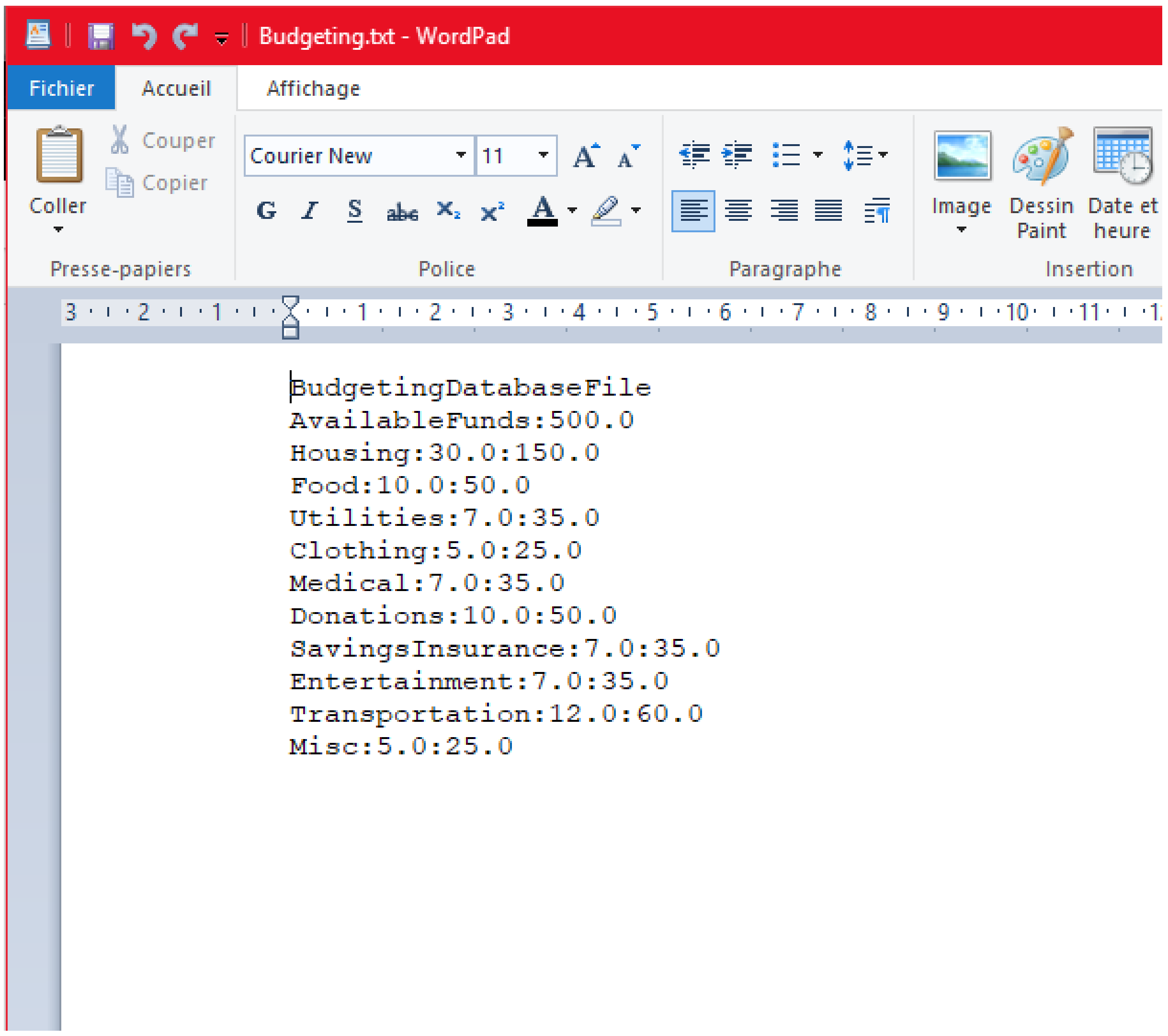
Dr. Gregory Butler. *COMP 354 Software Engineering.* Accessed March 21st at http://users.encs.concordia
w2018.html.

A Description of Input Files

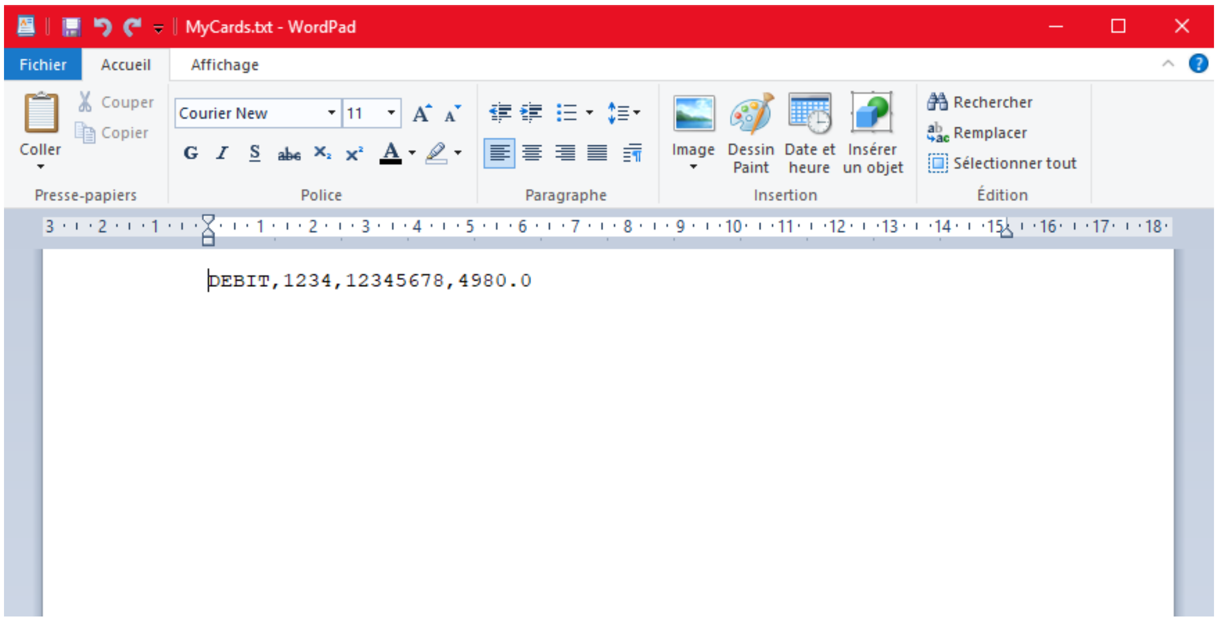
The input files for our program and testing is the information that the user, in this case the testers, enter in the application. The information the user inputs is stored on .txt files that look like the following files:



```
lol,1234
```



```
BudgetingDatabaseFile
AvailableFunds:500.0
Housing:30.0:150.0
Food:10.0:50.0
Utilities:7.0:35.0
Clothing:5.0:25.0
Medical:7.0:35.0
Donations:10.0:50.0
SavingsInsurance:7.0:35.0
Entertainment:7.0:35.0
Transportation:12.0:60.0
Misc:5.0:25.0
```

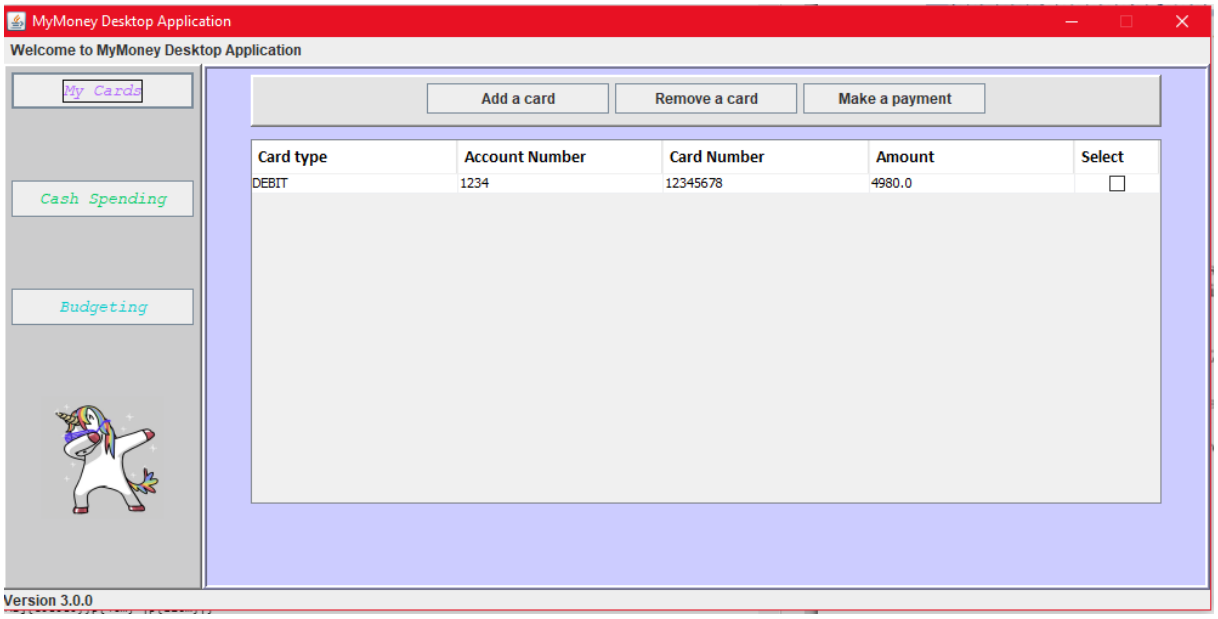


```
DEBIT,1234,12345678,4980.0
```

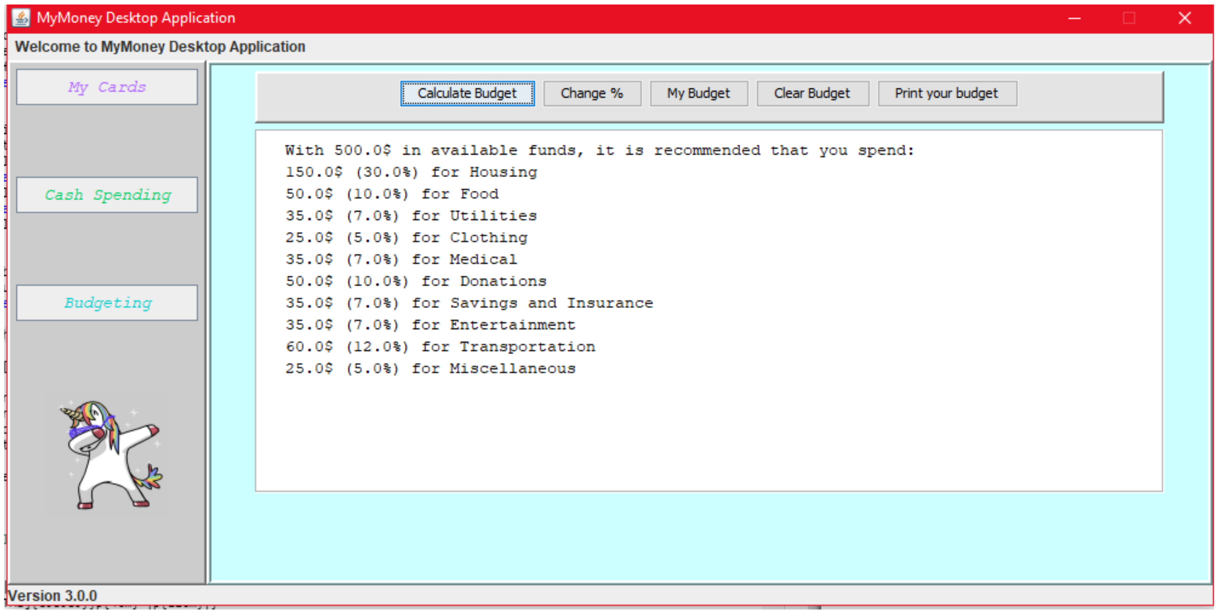
The files above are some of the files the application has in order to keep track of the information. Each subsystem has a file.

B Description of Output Files

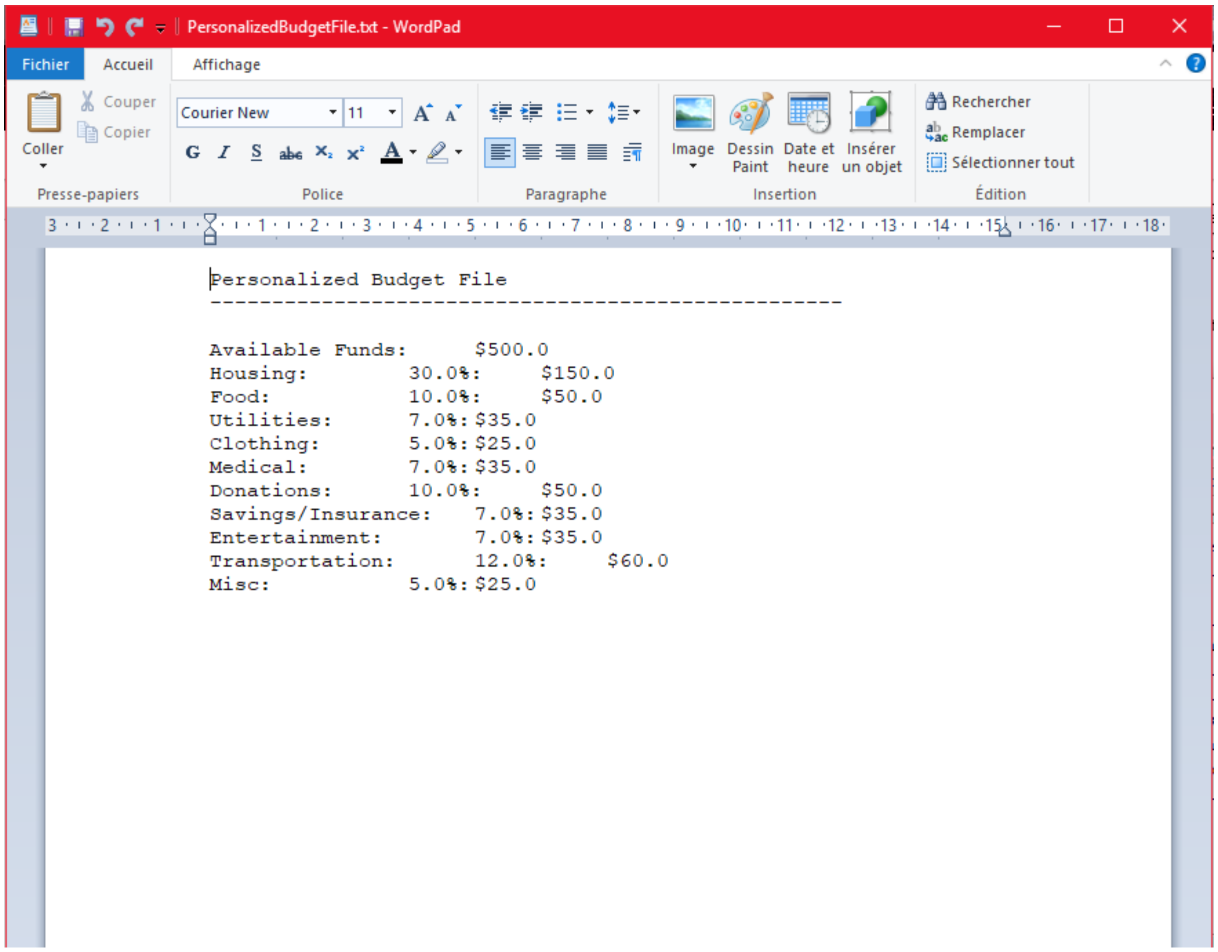
The output files of the application are the display in the graphic user interface of the information that has been entered by the user and the file that can be created by the user in the Budgeting subsystem. Here are examples of output files:



Card type	Account Number	Card Number	Amount	Select
DEBIT	1234	12345678	4980.0	☐



```
With 500.0$ in available funds, it is recommended that you spend:
150.0$ (30.0%) for Housing
50.0$ (10.0%) for Food
35.0$ (7.0%) for Utilities
25.0$ (5.0%) for Clothing
35.0$ (7.0%) for Medical
50.0$ (10.0%) for Donations
35.0$ (7.0%) for Savings and Insurance
35.0$ (7.0%) for Entertainment
60.0$ (12.0%) for Transportation
25.0$ (5.0%) for Miscellaneous
```



```
Personalized Budget File
------------------------------------------------

Available Funds:    $500.0
Housing:       30.0%:    $150.0
Food:          10.0%:    $50.0
Utilities:     7.0%:$35.0
Clothing:      5.0%:$25.0
Medical:       7.0%:$35.0
Donations:     10.0%:    $50.0
Savings/Insurance:  7.0%:$35.0
Entertainment:      7.0%:$35.0
Transportation:     12.0%:    $60.0
Misc:          5.0%:$25.0
```

The only subsystem that does not have an output file is the Authentification subsystem. It does not output to the screen the information that the user has inputed after the user enters the information. The information stays in the database text file.