

# Design Document: MyMoneyApp

March 19, 2018

Name	ID Number
Fabian Vergara	40006706
Jean-Loup	40006259
Vincent Boivin	27419694
Marc-Antoine Jette-Leger	27895038
Michael Xu	27206356
Kisife Giles	40001926

Team: PA-PJ

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Purpose . . . . .	3
1.2	References . . . . .	3
<b>2</b>	<b>Architecture Design</b>	<b>4</b>
2.1	Rationale . . . . .	4
<b>3</b>	<b>System Interface Design</b>	<b>6</b>
3.1	Main Panel Interface . . . . .	6
3.1.1	User Registration Interactions . . . . .	7
3.2	Registration Panel Interface . . . . .	8
3.2.1	User Interactions . . . . .	8
3.2.2	User Login Interactions . . . . .	8
3.3	Bank Panel Interface . . . . .	9
3.3.1	User Interactions . . . . .	10
3.4	Budget/Finace Panel . . . . .	10
3.4.1	User Interactions . . . . .	10
3.4.2	User Interactions . . . . .	11
<b>4</b>	<b>Module Class Diagram</b>	<b>12</b>
<b>5</b>	<b>Dynamic Model of System</b>	<b>12</b>
5.0.1	User Registration Dynamic Sequence . . . . .	13
5.0.2	User Login Dynamic Sequence . . . . .	14
5.0.3	Set Goal Dynamic Sequence . . . . .	15
<b>6</b>	<b>Internal Module Design</b>	<b>15</b>
<b>7</b>	<b>Appendix</b>	<b>24</b>
	List of Figures	24

# 1 Introduction

The goal of this project is to develop a application to provide financial advice to students and young professionals. This iterative project is in the second phase (iteration 2) and has the following use cases:

- Use case 1. A new user can create an account in the MyMoney application.
- Use case 2. An existing user can log into their own account in the My-Money application.
- Use case 3. A user can have an overview of their financial transactions and set financial goals.
- Use case 4. The application can analyze the user's actual spending and compare it with the set goals. The categories chosen for user goals are: Home, Food, Savings, Hobbies, Loans, Others.
- Use case 5. The application provides financial advise to the user based on the information obtained from the analysis in use case 4.

Ideally, the application would connect to a user's bank account and monitor their transactions. In this project however, the user information is read from local files in the system. The aim of this document is to explain the design model used in this application.

## 1.1 Purpose

The objective of this document is to present a design of how the MyApp application is build. This document provides the following design details:

- Architecture design which provides details about how the application is designed,
- Software interface design, including the snapshots of the application,
- The class diagram,
- System sequence diagrams and
- The design of internal modules in the application.

This document is intended primarily for the members of PA-PJ group and instructors.

## 1.2 References

Course website: [1]G. Butler, Professor, Year Published. [Online]. Available: <https://users.encs.concordia.ca/gregb/home/comp354-w2018.html>. [Accessed: 17- March- 2018].

## **2 Architecture Design**

The software architecture chosen for this application is the Model - View - Controller (MVC) architecture. This architecture decouples the modeling of an application from its views and controllers.

### **2.1 Rationale**

This choice of architecture is based on the fact that the MVC model separates the views which present or accept information from the user, from the internal representation of the data. This facilitates reuse of code and concurrent development. It also enables consistent presentation of information to the user on the views.

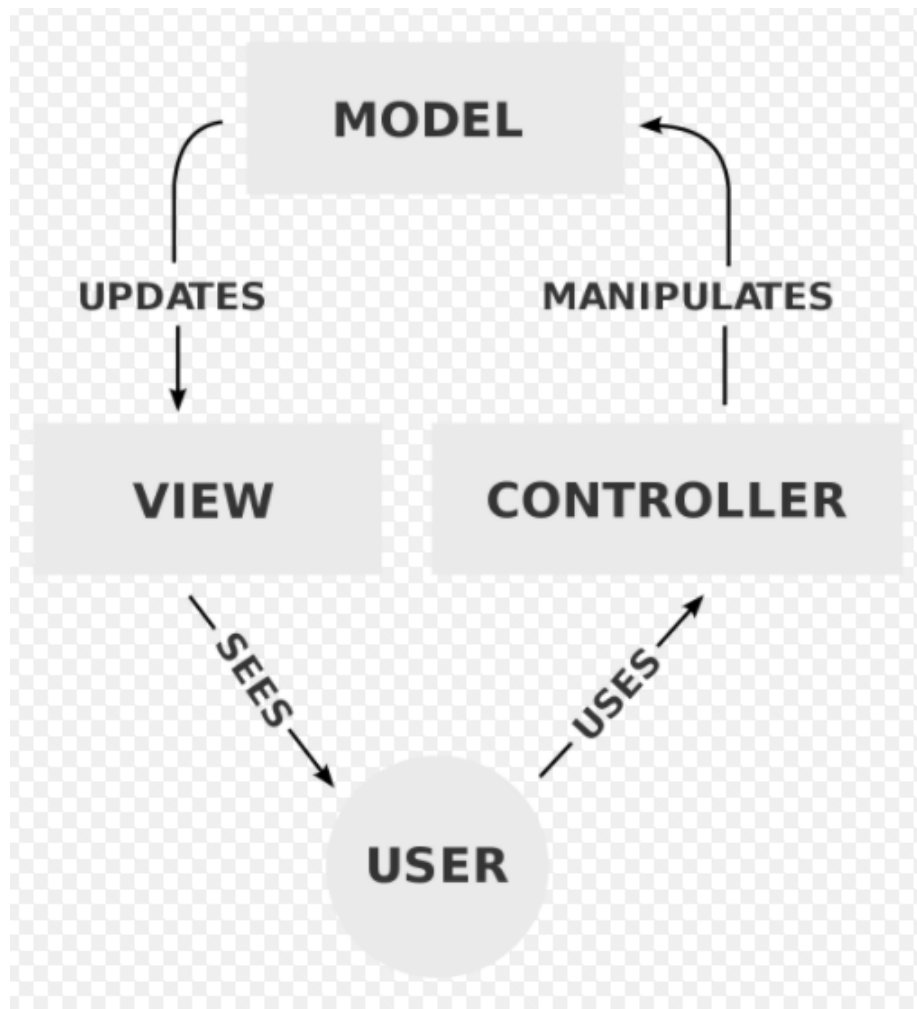


Figure 1: M-V-C architecture

??

The Model in the MyMoneyApp application represents the core functionality (user stories) and the data in the application. All the manipulation and computation on the data is done within the model.

The Views represents all the graphical user interfaces for providing information to the user and for the user to input/communication information to the application. The view is broken down into the main panel for login, the registration panel for registration of a new user, the bank panel for displaying user transactions, the finance panel for budgeting and charting of user expenses and

providing the user with financial advice.

The Controller bridges the gap between the views and the model by passing information or requests from the view to the model and from the model back to the view. This way, the MVC model allows for a consistent presentation of information to the user, independent of the model.

### 3 System Interface Design

This section provides information about the external interfaces. The MyMoneyApp application provides the following graphical user interfaces for interaction with the user:

- The Main Panel which is the home interface, for the user to login or request to register an account,
- Registration panel for registration of new users,
- The Bank Panel for user transaction details and financial advice,
- The Finance Panel for the user to set or evaluate a budget. In the following section, we present each of these interfaces and their user actions.

#### 3.1 Main Panel Interface

The main panel presents the home Graphical User Interface (GUI) for the application. User interactions with the application begins on this interface. This interface presents a user with a menu bar, a username and password fields. The menu bar has the options Registration, Help and About.

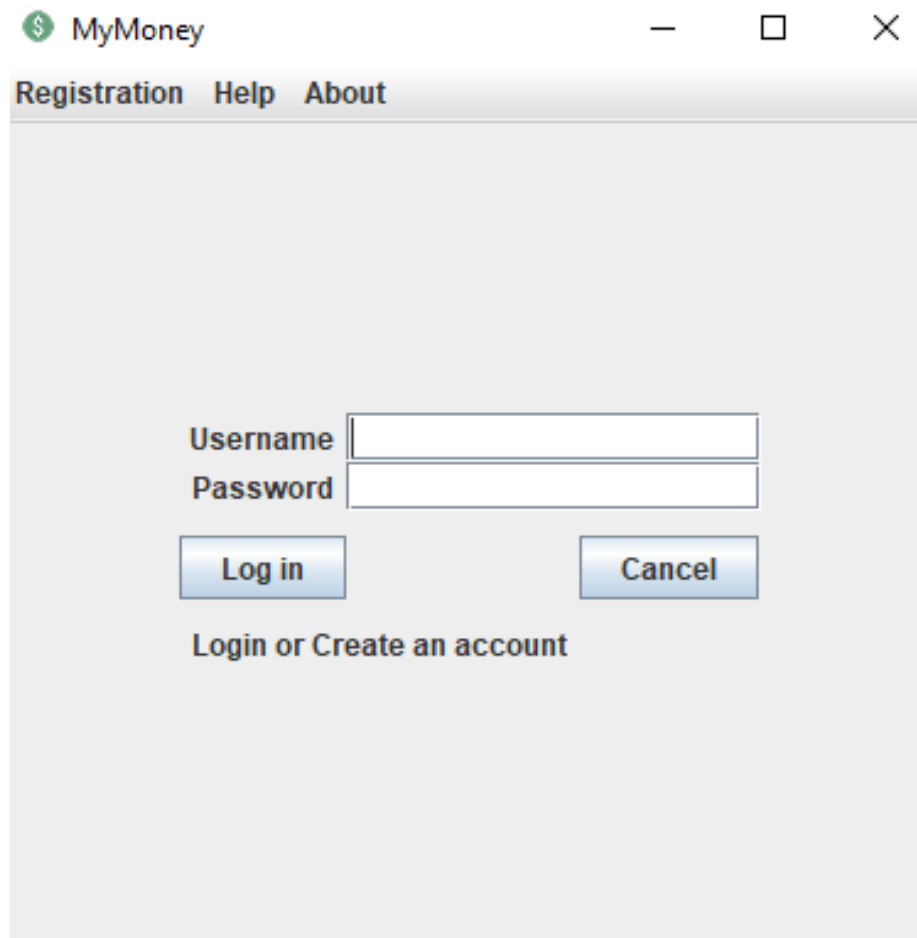


Figure 2: Main Panel Interface

This interface provides the user with an option to either log in with their username and password to an existing account or to create an account, if they do not have one. User interactions for these two scenarios are described below. There is also the Help and About options on this interface, but those are not functional at this point.

### 3.1.1 User Registration Interactions

To register a new account, the user selects the registration option from the main menu. The selection event takes the user to the registration panel.

### 3.2 Registration Panel Interface

A user without an account can create one by filling out the registration form and clicking on the register button to create an account. If the username has not been used already, the registration will be successful. Otherwise, a message directive is displayed advising the user to choose a different username.

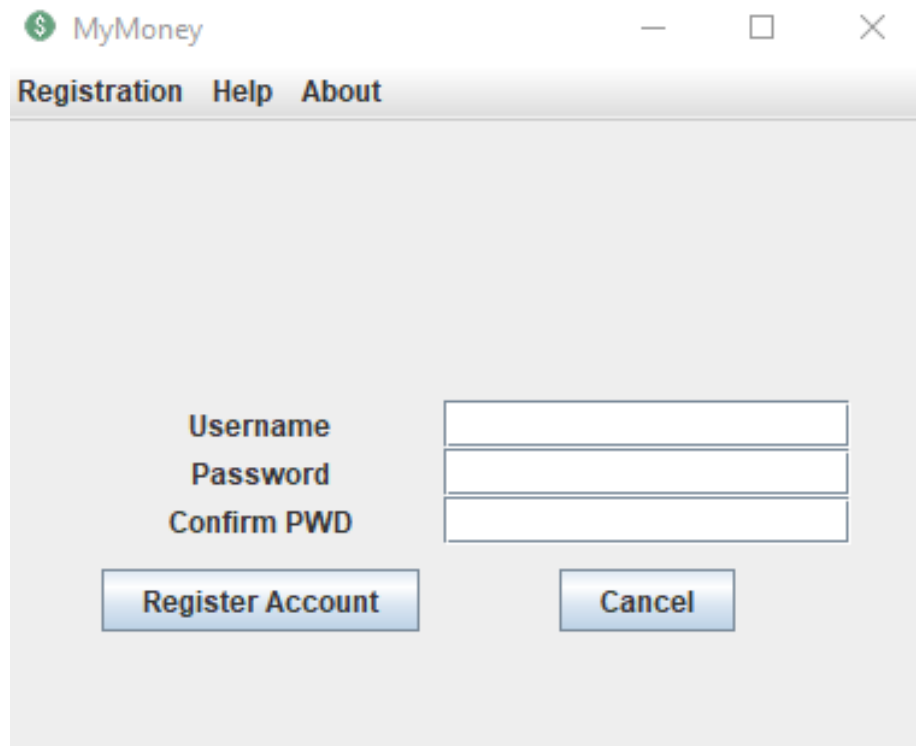
The image shows a screenshot of a software window titled "MyMoney" with a green dollar sign icon. The window has standard OS controls (minimize, maximize, close) in the top right. Below the title bar is a menu bar with "Registration", "Help", and "About". The main content area is light gray and contains a registration form. The form has three labels: "Username", "Password", and "Confirm PWD", each followed by a text input field. Below the input fields are two buttons: "Register Account" and "Cancel".

Figure 3: Registration Panel Interface

#### 3.2.1 User Interactions

The user enters a username, password, password confirmation and clicks the registration button. After a successful registration, the user is presented with the main panel to log into their account.

#### 3.2.2 User Login Interactions

On the main panel, the user enters username, password and click the login button for login, or selects the registration option on the main menu. If the



username and password do not match an existing account, a directive message is displayed asking the user to enter correct username and password. If the login information is still incorrect a second time, the directive is repeated. On the third attempt, the systems displays a directive message and exits if the login information is still incorrect.

### 3.3 Bank Panel Interface

After a successful log in, the application displays the user transactions sorted by date and provides financial advice based on this information and the goals the has set for themselves. Login is successful if the username and password match an existing account in the system.

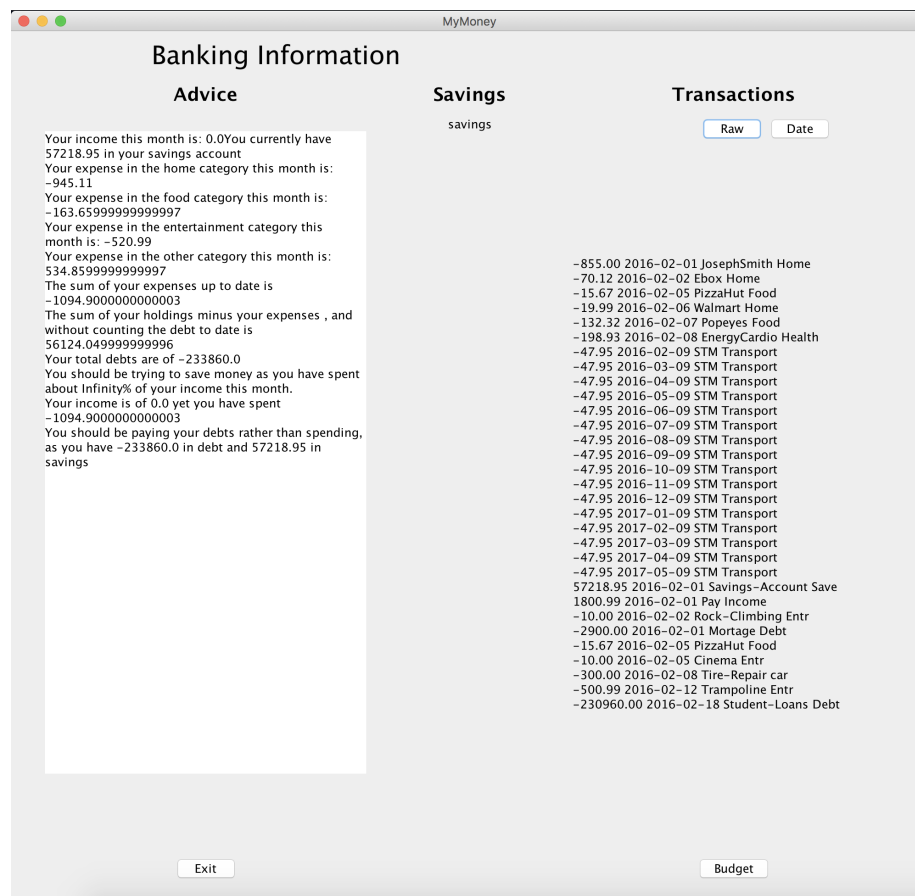


Figure 4: Bank Panel Interface

### 3.3.1 User Interactions

The user reviews their transactions and advice given and can choose to either click on the budget button to proceed to review their budget or to exit the application.

## 3.4 Budget/Finace Panel

After login into a user account, the user can review their budget or set a new budget by clicking on the budget button. This click event takes the user to the budget panel.

The screenshot shows the 'MyMoney' application window. It features two main panels: 'Expected Spendings' (Budget) and 'Real Spendings' (Actuals). Each panel has a 'Reset' button at the top. Below the buttons, the data is presented in a table-like format with categories and their corresponding values. At the bottom of each panel, there is a dropdown menu for selecting a category, a text input field for a value, and two buttons: 'Add to budget'/'Add to spendings' and 'Create chart'.

Expected Spendings		Real Spendings	
Home :	1001.00	Home :	1000.00
Food :	500.00	Food :	650.00
Hobbies :	300.00	Hobbies :	400.00
Savings :	700.00	Savings :	500.00
Other :	200.00	Other :	300.00
<b>Total Budget :</b>	<b>2700.00</b>	<b>Total Spendings :</b>	<b>2850.00</b>

At the bottom of the interface, there are two sets of controls:

- Budget Section:** A dropdown menu showing 'Other', a text input field containing '200', an 'Add to budget' button, and a 'Create chart' button.
- Spending Section:** A dropdown menu showing 'Other', a text input field containing '300', an 'Add to spendings' button, and a 'Create chart' button.

Figure 5: Finance Panel Interface

### 3.4.1 User Interactions

The user can set a budget by selecting spending categories and entering budget limits on how much they want to spend for each of the categories.

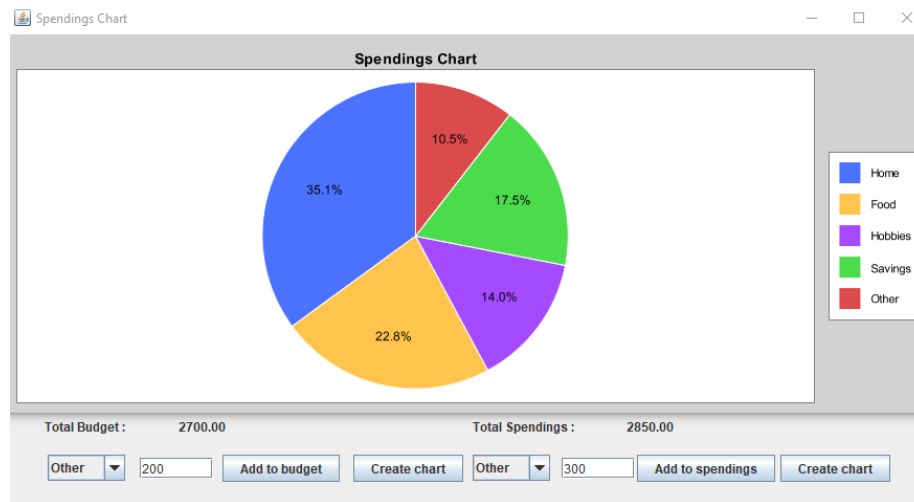


Figure 6: Pie Chart

The results of the MyMoneyApp is presented as a pie chart on the Pie Chart Panel.

### 3.4.2 User Interactions

No user interaction is needed on this interface.

## 4 Module Class Diagram

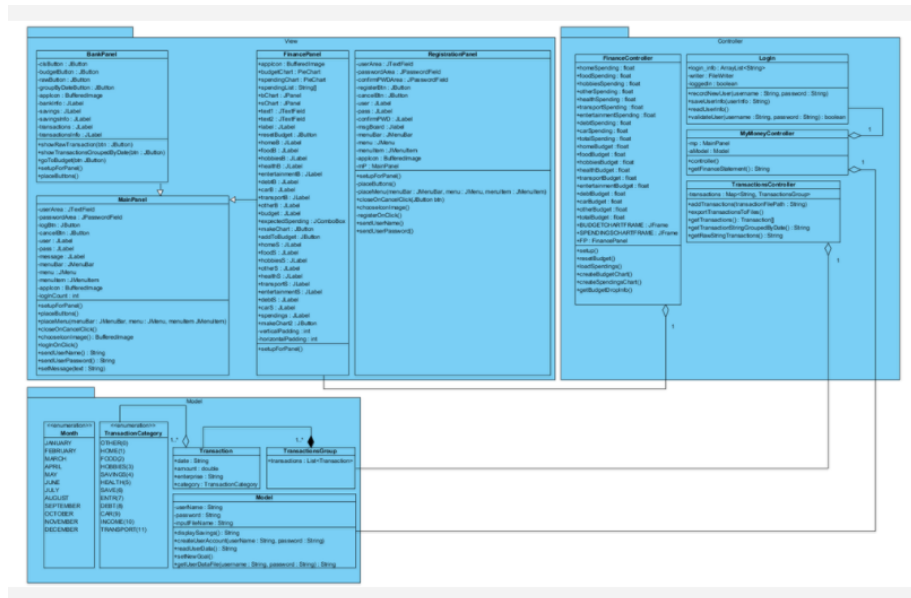


Figure 7: Class Diagram

## 5 Dynamic Model of System

This dynamic model shows interactions between components within the application. To illustrate the model interactions within the MyMoneyApp, we choose the following core scenarios:

### 5.0.1 User Registration Dynamic Sequence

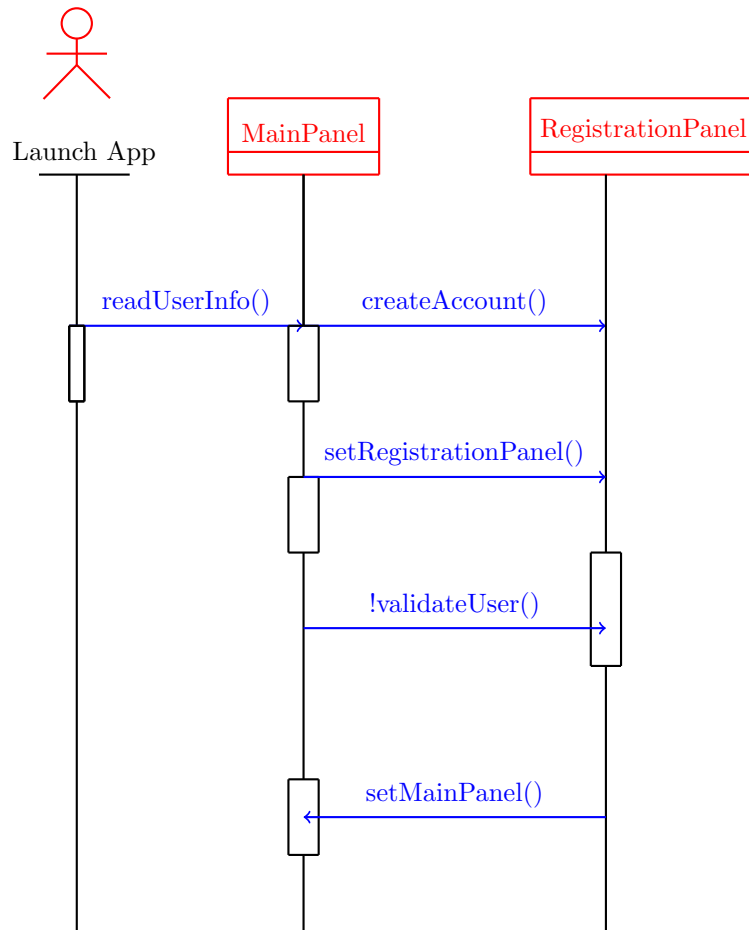


Figure 8: User Registration Dynamic Sequence

### 5.0.2 User Login Dynamic Sequence

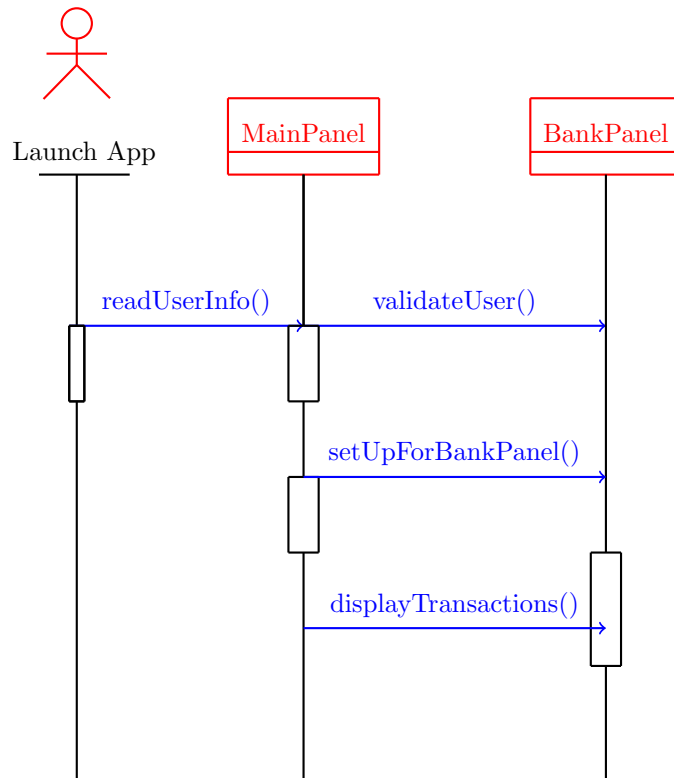


Figure 9: User Login Dynamic Sequence

### 5.0.3 Set Goal Dynamic Sequence

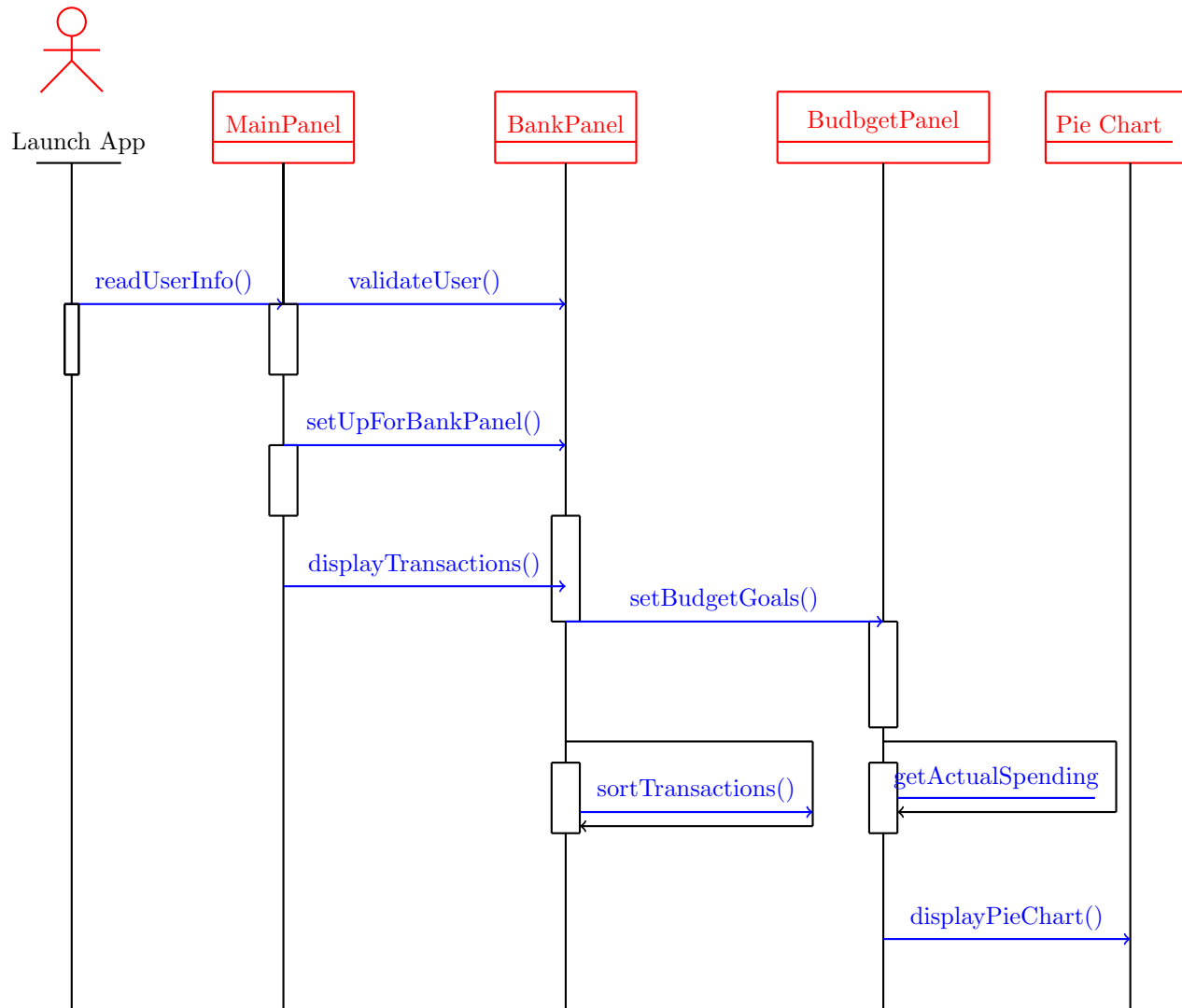


Figure 10: Goal Setting Dynamic Sequence

## 6 Internal Module Design

This section gives a detailed description of the various methods and pseudo-code implemented in the application.

Table 1	
Method Name	RecordNewUser
Class Name	LogIn
Functionality	Adds user login info to an arrayList.
Input	String username, String password
Output	none
Pseudo Code	<pre>String newUserInfo = username + "::" + password  logininfo.add(newUserInfo)  saveUserInfo(newUserInfo)  loggedIn = true</pre>

Table 2	
Method Name	saveUserInfo
Class Name	LogIn
Functionality	Method writes user login info to file.
Input	String userInfo
Output	none
Pseudo Code	<pre>file loginInfo.open  if (userInfo != "")     file.write(userInfo)</pre>

Table 3	
Method Name	readUserInfo
Class Name	LogIn
Functionality	Method reads user info from file and save in array list
Input	none
Output	none
Pseudo Code	<pre>String lineString = ""  reader = open("../datafiles//logininfo")  while( next line != NULL)     logininfo.add(lineString)</pre>



Table 4	
Method Name	validateUser
Class Name	LogIn
Functionality	return true if user info exist in file else false
Input	String username, String password
Output	boolean
Pseudo Code	<pre>String newUserInfo = username + "::" + password  return logininfo.contains(savedInfo)</pre>

Table 5	
Method Name	loadTransactions
Class Name	Model
Functionality	reads transaction info and stores them in an array
Input	none
Output	array of Transactions
Pseudo Code	<pre>Scanner s =TransactionFile.open  arrayList transactionList  while(file hasNextLine)     String line = file.nextLine()     readTransaction     transactionList.add(newTransaction) return transactionList.toArray</pre>

Table 6	
Method Name	displaySavings
Class Name	Model
Functionality	Displays the user's savings from their bank
Input	none
Output	String transactions
Pseudo Code	<pre>scanner s1 = ("./datafiles//savings")  String savings  savings = s1.next  return savings</pre>

Table 7	
Method Name	createUserAccount
Class Name	Model
Functionality	Create user object and save info
Input	String username, string password
Output	none
Pseudo Code	<pre>Model aModel = new Model(userName, password)  saveUserInfo(userName, password)</pre>

Table 8	
Method Name	readUserData
Class Name	Model
Functionality	Create user object and save info
Input	none
Output	String
Pseudo Code	<pre>String line  reader = open(inputFile) while(reader.nextLine != NULL) line.append(read.nextLine)  return Line</pre>

Table 9	
Method Name	getUserDataFile
Class Name	Model
Functionality	Returns file name corresponding to a user.
Input	String username, String password
Output	none
Pseudo Code	<pre>string fileName if(validate User) filename = "../datafiles//default"  else fileName = "../datafiles//usernotfound"  return filename</pre>

Table 10	
Method Name	resetBudget
Class Name	FinanceController
Functionality	Sets budget values to 0.
Input	none
Output	none
Pseudo Code	homeBudget = 0; foodBudget = 0; hobbiesBudget = 0; savingsBudget = 0; otherBudget = 0; totalBudget = 0;

Table 11	
Method Name	resetSpending
Class Name	FinanceController
Functionality	Sets spending values to 0.
Input	none
Output	none
Pseudo Code	totalSpending = 0; homeSpending = 0; foodSpending = 0; hobbiesSpending = 0; savingsSpending = 0; otherSpending = 0;

Table 12	
Method Name	addTransactions
Class Name	TransactionsController
Functionality	Add Transactions to the user's bank info
Input	String transactionFilePath
Output	none
Pseudo Code	<pre> ArrayList entries  transactionFilePath.open  String currentline = transactionFilePath.getLine  entries.add(currentLine)  for(every line)     String[] data = entry.split("s+")  if (data.length &gt; 4) break  String amount = data[0]; String date = data[1] String enterprise = data[2]; String category = data[3];  if (!transactions.containsKey(date)) transactions.put(date, new TransactionsGroup()) </pre>

Table 13	
Method Name	exportTransactionsToFiles
Class Name	TransactionsController
Functionality	writes all transactions to a file
Input	none
Output	none
Pseudo Code	<pre> for(transactionMap.size)     String key = Map.getKey LinkedList TransactionEntries  transactionsPerGivenDate = Map.getValue  for(transactionsPerGivenDate.size)     transactionEntries.add(transaction)  transactionEntries.WriteToFile </pre>

Table 14	
Method Name	getTransacationStringGroupedByDate
Class Name	TransactionsController
Functionality	Returns transactions arranged by date
Input	none
Output	String
Pseudo Code	<pre>String temp for(Map entry)     String key = entry.getKey     TransactionsGroup transactionsPerGivenDate     = entry.getValue  for(number of Transactions)     temp += transaction  return temp</pre>

Table 15	
Method Name	getRawStringTransacations
Class Name	TransactionsController
Functionality	Returns transactions arranged by date
Input	none
Output	String
Pseudo Code	<pre>Scanner s= TransactionFile.open  String transactions  while(s.hasNextLine)     transactions += s1.nextLine  return transactions</pre>

Table 16	
Method Name	TransactionsGroup
Class Name	TransactionsGroup
Functionality	create a linked list
Input	none
Output	none
Pseudo Code	<pre>this.transactions = new LinkedList&lt;Transaction&gt;()</pre>

Table 17	
Method Name	addTransaction
Class Name	TransactionsGroup
Functionality	create a linked list
Input	Transaction Object
Output	none
Pseudo Code	this.transactions.add(transaction)

Table 18	
Method Name	savingAdvice
Class Name	Advice
Functionality	Looks at user's expenses and gives financial advice
Input	ArrayList<String> arr
Output	none
Pseudo Code	<pre> String [] arrayarr = arr.toArray double [] arraytype = 0.00,0.00,0.00,0.00,0.00,0.00,0.00 double doubleamount = 0 String advice = "Error!"  for (arrayarr.length)     String ExpenseType = arrayarr.substring(4)     String StringAmount = arrayarr.substring(0).idex(3)     doubleAmmount      if (ExpenseType== inco)         arrayType[0] += doubleAmmount     if (ExpenseType== Food)         arrayType[1] += doubleAmmount     if (ExpenseType== Entr)         arrayType[2] += doubleAmmount     if (ExpenseType== Save)         arrayType[3] += doubleAmmount     if (ExpenseType== Debt)         arrayType[5] += doubleAmmount     else         arrayType[4] += doubleAmmount  double sum =arrayType.totalSum double totalExpense = arrayType.sumAtIndex(1,2,3,4)  print all income information  if(arraytype[0]-totalExpense &gt;= arraytype[0]*.2)     advice = save money     if(arraytype[6]&lt;= sum)         advice += pay debts else     advice = pay debts print advice </pre>

## 7 Appendix

### List of Figures

1	M-V-C architecture . . . . .	5
2	Main Panel Interface . . . . .	7
3	Registration Panel Interface . . . . .	8
4	Bank Panel Interface . . . . .	9
5	Finance Panel Interface . . . . .	10
6	Pie Chart . . . . .	11
7	Class Diagram . . . . .	12
8	User Registration Dynamic Sequence . . . . .	13
9	User Login Dynamic Sequence . . . . .	14
10	Goal Setting Dynamic Sequence . . . . .	15

### List of Tables

Table 1(recordNewUser).	16
Table 2(saveUserInfo) . . . . .	16
Table 3(readUserInfo) . . . . .	16
Table 4(validateUser) . . . . .	17
Table 5(loadTransactions) . . . . .	17
Table 6(displaySavings) . . . . .	17
Table 7(createUserAccount) . . . . .	18
Table 8(readUserData) . . . . .	18
Table 9(getUserDataFile) . . . . .	18
Table 10(resetBudget) . . . . .	19
Table 11(resetSpending) . . . . .	19
Table 12(addTransactions) . . . . .	20
Table 13(exportTransactionsToFiles) . . . . .	20
Table 14(getTransactionStringGroupedByDate) . . . . .	21
Table 15(getRawStringTransactions) . . . . .	21
Table 16(transactionsGroup) . . . . .	21
Table 17(addTransaction) . . . . .	22
Table 18(savingAdvice) . . . . .	23