# Software Design Description

**for**

# Student Budget Management System

**Version 1.0 approved**

**Sufi Nukman Bin Shakimi**

**Muhammad Alif Luqman Bin Afandi**

**Nurhamizah Bt. Husin**

**Eman Hussein**

**17th July 2020**

# Table of Contents

# 1 Introduction

## 1.1 Purpose

Student Budget Management (SBM) is focusing on managing budget and finance specifically for students. This project is related to finance field among UPM students. Nowadays, students are facing problem in managing their money in proper way due to lack of management skill and ignorance of importance of management skill. That's the reason for us to choose students as our users. Therefore, we think that through this SBM project, we can help the students to manage their finance. We believe that this project can guide them the way to manage their finance throughout their studies based on their income like loans or scholarships. This project also improve the disabilities in current existing system.

## 1.2 Scope

The SBM focuses in managing UPM students finance based on their income and pocket money every semester. The SBM gives alerts for student to be more disciplined in managing their budget and to be responsible on their expenses.

## 1.3 Definitions, Acronyms and Abbreviations

| Terms | Definition |
|---|---|
| SBM | Student Budget Management |
| SRS | Software Requirement Specification |
| UPM | Universiti Putra Malaysia |

## 1.4 References


[1] Lecture notes, Software Requirement Engineering.

[2] https://softwarekno.blogspot.my/2016/09/waterfall-model.html

[3] https://gyires.inf.unideb.hu/GyBITT/07/ch01.html

# 2 System Overview

SBM is an application for the user which comes from students to plan their budget throughout the semester. User also able to create their profile which contain name, age, email and course. Other than record pocket money, user can also record the expenses that they spent. Then, SBM will continue to allocate the recorded pocket money into three major categories: wants, needs and savings. Therefore, user can make it as a guide for them to maintain their budget and to improve their money management.

# 3 Design Consideration

## 3.1 Assumptions and Dependencies

AS1- We assume that the user will record their budget monthly only.
AS2- We assume that there are 3 major budget categorization for student in their life which are wants, needs and savings.
D1- The system is depend on the changes made by the user in their software to a latest version.

## 3.2 General Constraints

C1- The constraint that we have found is the memory requirement in the hardware. The system needs quite an high RAM as a memory requirement for its better performance on calculation and continuous changes in calculation.
C2- Another constraint is network traffic where it requires smooth internet connection in order for the calculation to be done correctly without lacks.

## 3.3 Goals and Guidelines

G1- Provides full help towards UPM students in managing their finance
G2- Create a full functioned system

## 3.4 Development Methods

The system has been created on waterfall model methodology. It is a classical model used in system development life cycle to create a system with a linear and sequential approach. It contains certain phases such as Requirement Gathering , Analysis, System Design, Implementation Development , Testing Integrating and Maintenance Fixing. The reason of using the methodology is because of its easy management for small project that has clear defined stages.

# 4 Detailed Design

The Student Budget Management class diagram consist of 4 classes which are Student, Allocation, Pocket Money and Expenses. All of these classes have their own attributes and methods as described in the below sections.

## 4.1 User Class

| **Student** |
| --- |
| +Username : String<br>+ Password : String<br>+ Name: String<br>+ Email: String |
| + prodiveUserID()<br>+ providePass()<br>+ displayProfile() |

### a. Classification

The class component consist of attributes (public modifier, attribute name, data type) and list of methods or functions.

### b. Definition

The Student class is a class that contains many attributes based on the user requirements such as Username(String), Password(String), Name(String), and Email(String). It also shows the behaviour of each attribute of the class which declared as public.

### c. Responsibilities

There are 3 methods in Profile class. The first method is provideUsername(), which will return the user's name. The second method is providePass(). This method will return

the user's password in order to login the system. The last method is diplayProfile() which is responsible for displaying profile whenever the user wants.

**Use Case 1:** Login

| |
|---|
| **Unique Identifier:** UC1 |
| **Objective:** To enable user access the system by his login information |
| **Priority:** High |
| **Source:** Arjuna Hakim (End-user) |
| **Actor:** Student |
| **Flow Of Events:**<br>**6.1. Basic Flow**<br>1. User has to press sign in button<br>2. System will display text field to insert Username and password<br>3. User has to insert Username and password<br>4. User has to press Login button<br>5. System will validate password<br>6. User can login successfully<br>**6.2. Alternative Flow**<br>6.2. Error message is prompt and ask the user to re-enter the password |
| **Pre-condition:** The use case starts when the user wants to use the application |
| **Post-condition:** The use case ends when the user successfully logged in |
| **Notes/Issues:** None |

**Diagram 4.1.1 : Flow of Event For User**

**Activity Diagram for Use Case 1:** Login

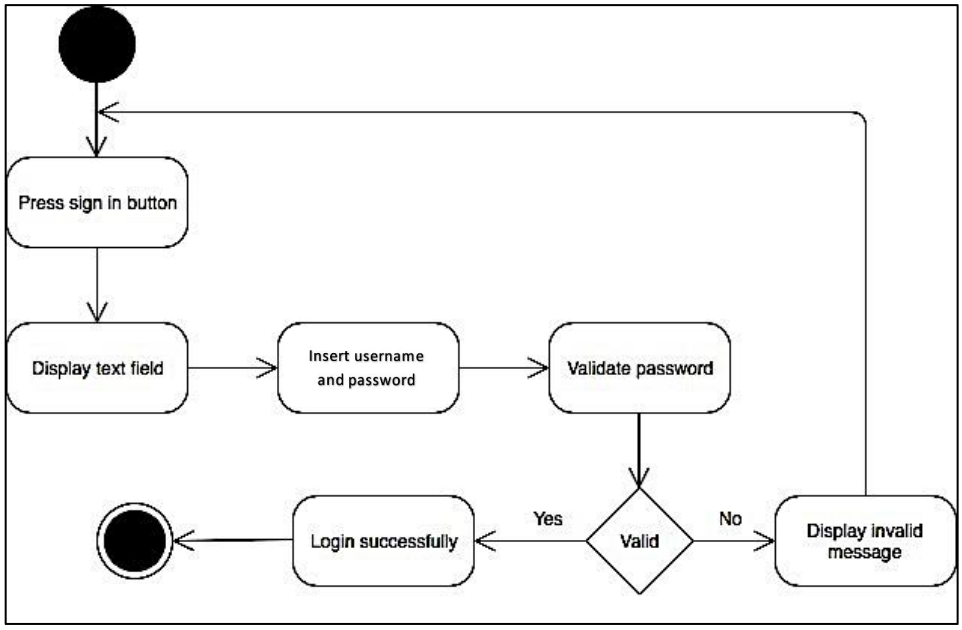



**Diagram 4.1.2: Activity Diagram For User**

User shall press sign in button to sign in. Then, the system will display text field for the user to insert Username and password. When the user has submitted their Username and password, the system will validate the password. If it is valid, the user login successfully. If invalid, the system will display an error stating that the password is invalid and return back to the beginning of the sign in activity.

**Use Case 2:** Create Profile

| |
|---|
| **Unique Identifier:** UC2 |
| **Objective:** To store student's personal information |
| **Priority:** Medium |
| **Source:** Arjuna Hakim (End-user) |
| **Actor:** Student |
| **Flow Of Events:**<br>**6.1. Basic Flow**<br>1. User has to press create profile button<br>2. System will display empty form<br>3. User has to fill in the information and press Submit button<br>4. System will ask user to confirm<br>5. User confirm the profile information<br>6. System stores the profile information<br>7. System displays the profile information<br>**6.2. Alternative Flow**<br>5.2. If the user is not confirm, he or she can continue editing the form |
| **Pre-condition:** The use case starts when the user successfully logged in |
| **Post-condition:** The use case ends when the profile is successfully edited or uploaded |
| **Notes/Issues:** None |

**Diagram 4.1.3 : Flow of Event For User**

**Activity Diagram for Use Case 2:** Create Profile

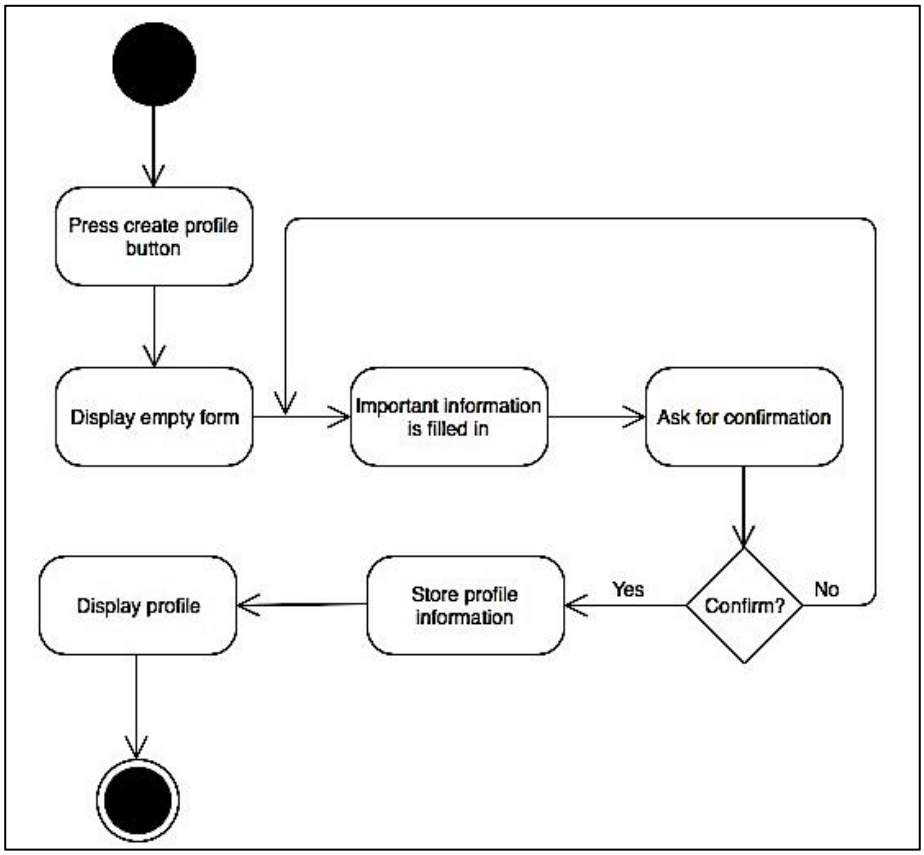



**Diagram 4.1.4: Activity Diagram For User**

User shall press create profile button to create his or her profile. Then, the system will display an empty form for the user to insert important information. After the user has finished inserting important information, the system will ask for confirmation to submit. If user confirms, the system will store the important information and display the created profile. If user not confirm, the system will stay on the edited form.

## 4.2 Pocket Money Class

| Pocket money |
| --- |
| + TotalMoney : Double<br>+ Month: String |
| + getPocketMoneyDetails() |

### a. Classification

The class component consist of attributes (public modifier, attribute name, data type) and list of methods.

### b. Definition

The Record Pocket Money class is a class that contains a few attributes such as TotalMoney(Double) and Month(String). All the attributes gave their own data type.

### c. Responsibility

The behaviour of class is define using method. Method is a subroutines that have the ability to operate a class. This operation may after state of a class or simply provide ways of accessing it. For the record pocket money class, we only have one method, which is getPocketMoneyDetails().

**Use Case 3:** Record Pocket Money

| |
|---|
| **Unique Identifier:** UC3 |
| **Objective:** To know students pocket money to calculate allocation |
| **Priority:** High |
| **Source:** Arjuna Hakim (End-user) |
| **Actors:** Student |
| **Flow of Events**<br>• **Basic Flow**<br>1. System displays a form to user fills the pocket money they have.<br>2. User must fill the details (user`s pocket money and the month) and click 'submit' button.<br>3. System process the details.<br>4. System will save the information of pocket money details.<br>5. The use case ends.<br><br>• **Alternatives Flows**<br>4a.User desires to re-enter or change the record details |
| **Pre-condition:**<br>• User is a registered user<br>• User prompts 'record pocket money' to indicate the user want to record pocket money |
| **Post-condition:**<br>Pocket money is successfully recorded |
| **Notes/Issues:** None |

**Diagram 4.1.7: Flow of Event for Pocket Money**

**Activity Diagram for Use Case 3:** Record Pocket Money

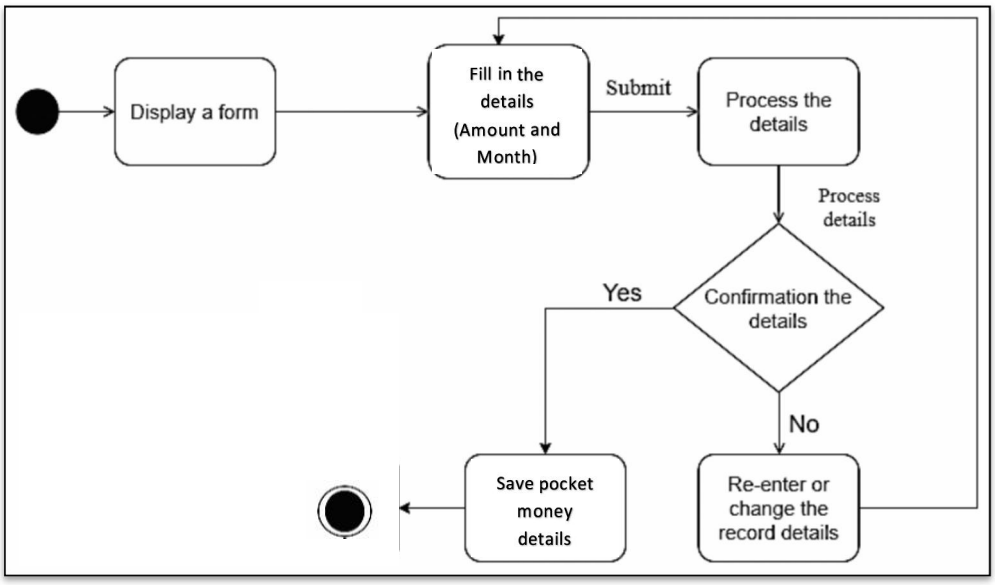



**Diagram 4.1.8: Activity Diagram for Pocket Money**

The system will display a form to user fill the details like amount of the pocket money and month. Once the user submits the form, system will process the details and the user will be asked whether they want to confirm the details or not. If the user confirms the details, the system will save the details of pocket money. If the user did not confirm the record pocket money details, user have to re-enter or change the record details.

**d. Constraint**

<u>Assumption</u>

1. User already clicked record pocket money button.

<u>Limitation</u>

None

## 4.3 Allocation Class

| Allocation |
| --- |
| + Budget:Double<br>+ WantsAllocation:Double<br>+ NeedsAllocation:Double<br>+ SavingsAllocation:Double |
| + GetTotalPocketMoney(TotalMoney:Double):Double<br>+ DisplayPocketMoney&Allocation():void<br>+ Calculate Allocation(PocketMoney:Double , Percentage:Double):Double<br>+ SaveAllocation(WantsAllocation:Double, NeedsAllocation:Double, SavingsAllocation:Double):void<br>+ DisplayAllocation():Double |

**a. Classification**

Consists of attributes(public modifier, attribute name, data type and methods.

**b. Definition**

Allocation class is an association of pocket money class. It contains 8 attributes:

I) Budget:Double- to calculate the allocation.
III) WantsAllocation:Double-to store the amount of allocation for wants.
IV) NeedsAllocation:Double- to store the amount of allocation for needs.
VI) SavingsAllocation:Double to store the amount of allocation for savings.

**c. Responsibility**

This class helps in carry out 8 tasks of the program:

I) Calculate Allocation(PocketMoney:Double, Percentage:Double):Double-is to calculate allocation for wants, needs and savings in several stages where when pocket money is being recorded and whenever expenses or transaction is being recorded. It will be getting its parameter from several classes such as Student class or Bank Management class.

II) SaveAllocation(WantsAllocation:Double, NeedsAllocation:DoubleInteger, SavingssAllocationt:Double):void-to store the allocation at all the stages that has been mentioned.

III) DisplayAllocation():Double-to send the allocation to the application function in order to be displayed.

Below is the activity diagram for calculation:

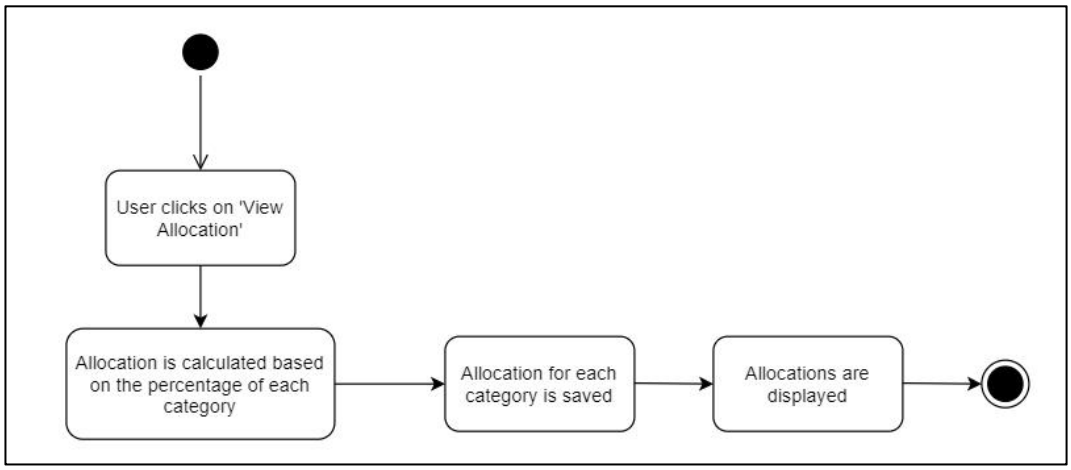



**Diagram 4.1.9: Flowchart for Allocations Calculation (UC4)**

The above diagram is an Activity diagram of UC4. It shows the activity that happens inside the use case. The activities that happen throughout the use case are pressing 'view allocation' button; calculating allocation and displaying output.

**Use Case 4:** View Allocation

| |
|---|
| **Unique Identifier:** UC4 |
| **Objective** – To show the budget allocation for every category |
| **Priority** – Medium |
| **Source** – Arjuna Hakim (End-user) |
| **Actors** – Student |
| **Flow of Events**<br>**Basic Flow**<br>1. User have to select 'View Allocation' button<br>2. System will Calculate allocation for every category |

| |
|---|
| **3.** System will display allocations |
| **Preconditions –** Starts when successfully logged in and one month of total budget has been recorded |
| **Post conditions –** When the management is displayed |
| **Notes/Issues** – None |

## 4.4 Expenses Class

| **Expenses** |
|---|
| + Expense:Double<br>+ WantsExpense:Double<br>+ NeedsExpenses:Double<br>+ SavingsExpenses:Double |
| + GetTotalPocketMoney(TotalMoney:Double):Double<br>+ GetAllocation():Double<br>+ SaveExpenses(WantsExpense:Double, NeedsExpenses:Double, SavingsExpenses:Double):void<br>+ DeductAllocation(WantsExpense:Double, NeedsExpenses:Double, SavingsExpenses:Double):Double<br>+ SaveBalance():void |

**Activity Diagram for Use Case 5:** Record Expense

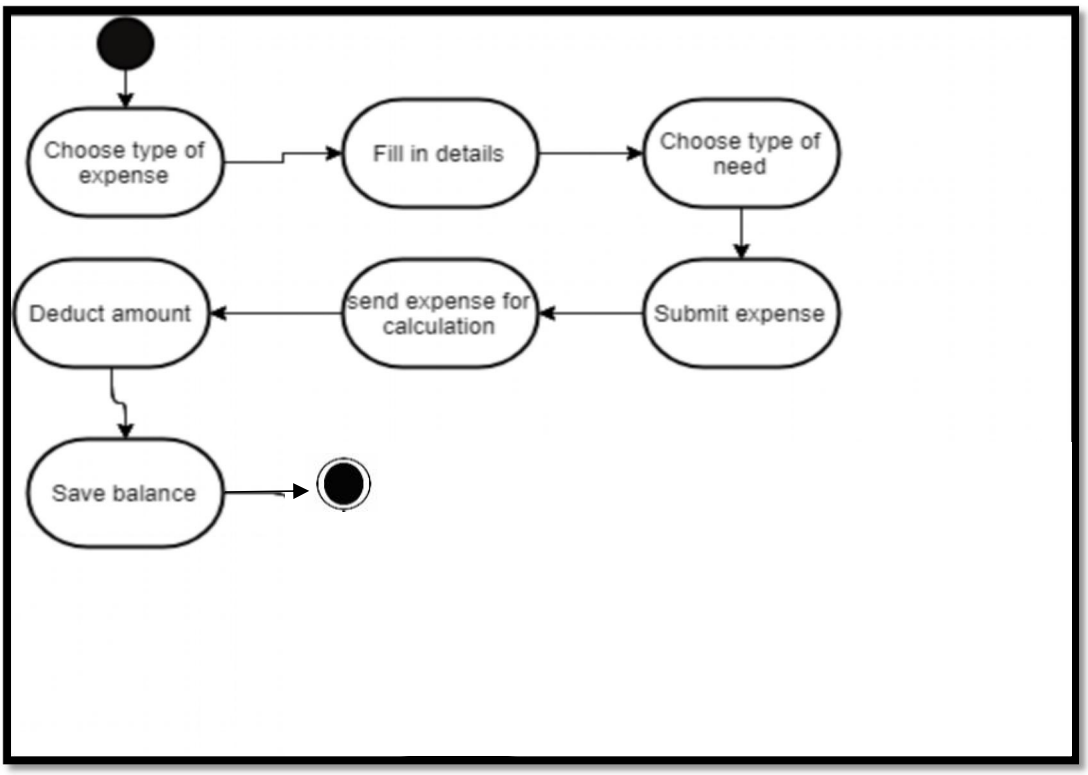



**Diagram 4.1.11: Activity Diagram for Record Expense**

The above diagram is an Activity diagram of UC5. It shows the activity that happens inside the use case. The activities that happen throughout the use case are choosing the type of expense; filling in details; deducting balance and saving balance.

**Use Case 5:** Record Expense and Transaction

| |
|---|
| **Unique Identifier:** UC5 |
| **Objective** – To record daily expenses and transaction to deduct it from the allocation |
| **Priority** – High |
| **Source** – Arjuna Hakim (End-user) |
| **Actors** – Student |
| **Flow of Events**<br>**Basic Flow**<br>1. User have to press the 'Record Expense' button<br>2. User have to choose the category<br>3. User have to fill in the amount spent<br>4. User have to submit<br>5. System will deduct the amount of money<br>**6.** System will save the allocation |
| **Extends**-UC6 – To see the allocation |
| **Preconditions –** Starts when the user have logged in |
| **Post conditions –** The balance amount is saved |
| **Notes/Issues** – None |

# 5 Database Design

## 5.1 Data Model

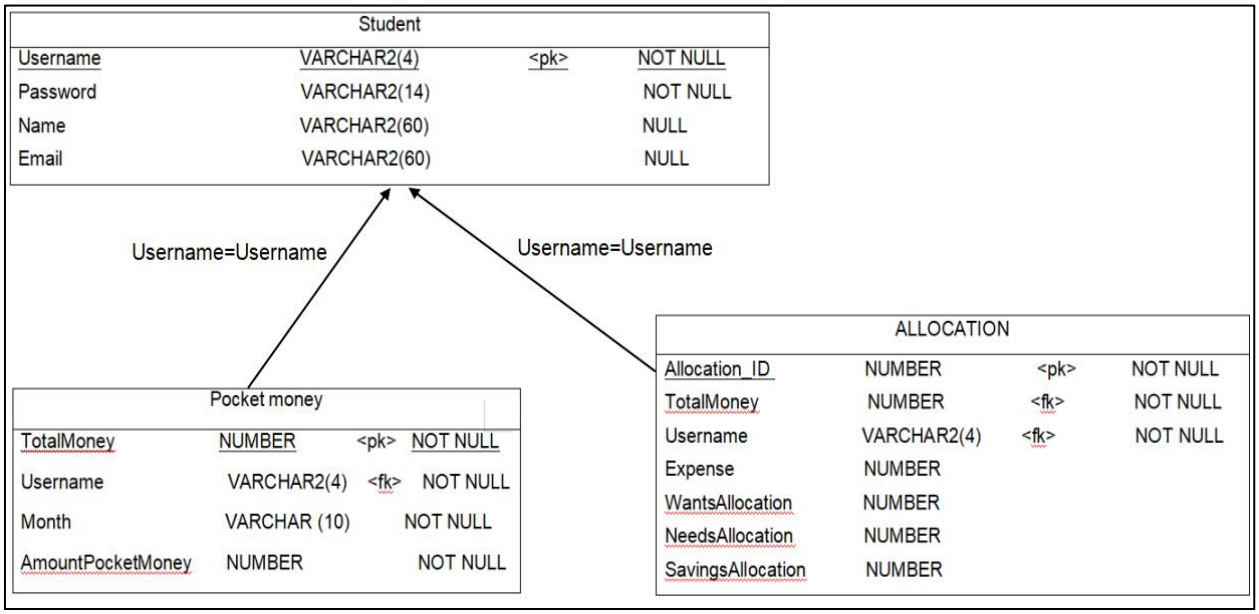



**Diagram 5.1.1: Complete Relational Table Mapping Based on The Class Diagram**

Diagram 5.1.1 represents the whole of our database system. To design this structure, we apply a series rules of thumb. Based on this approach, we convert the whole class diagram to a relational table. We list some attributes name (UserID), datatype (Varchar2(4)), primary key/foreign key<pk>/<fk> and constraints (null/not null).

## 5.2 Data Dictionary

**Use Case 1:**Login, **Use Case2:** Register

| **Student** |
|---|
| +Username : String<br>+ Password : String<br>+ Name: String<br>+ Email: String |
| + prodiveUsername()<br>+ providePass()<br>+ displayProfile() |

| Student | | | |
|---|---|---|---|
| Username | VARCHAR2(4) | <pk> | NOT NULL |
| Password | VARCHAR2(14) | | NOT NULL |
| Name | VARCHAR2(60) | | NULL |
| Email | VARCHAR2(60) | | NULL |

Student Table

| ID | Password | Name | Email |
|---|---|---|---|
| | | | |

**Diagram 5.2.1: Class Diagram and Relation Table For Student**

By using the series of rule of thumb, we map the medical record class to relational table. We convert all the string type attributes to varchar2 with different sizes and integer type to number. The primary key for this class is username and the constraint is not null. As for other attributes, we set the constraint to be null.

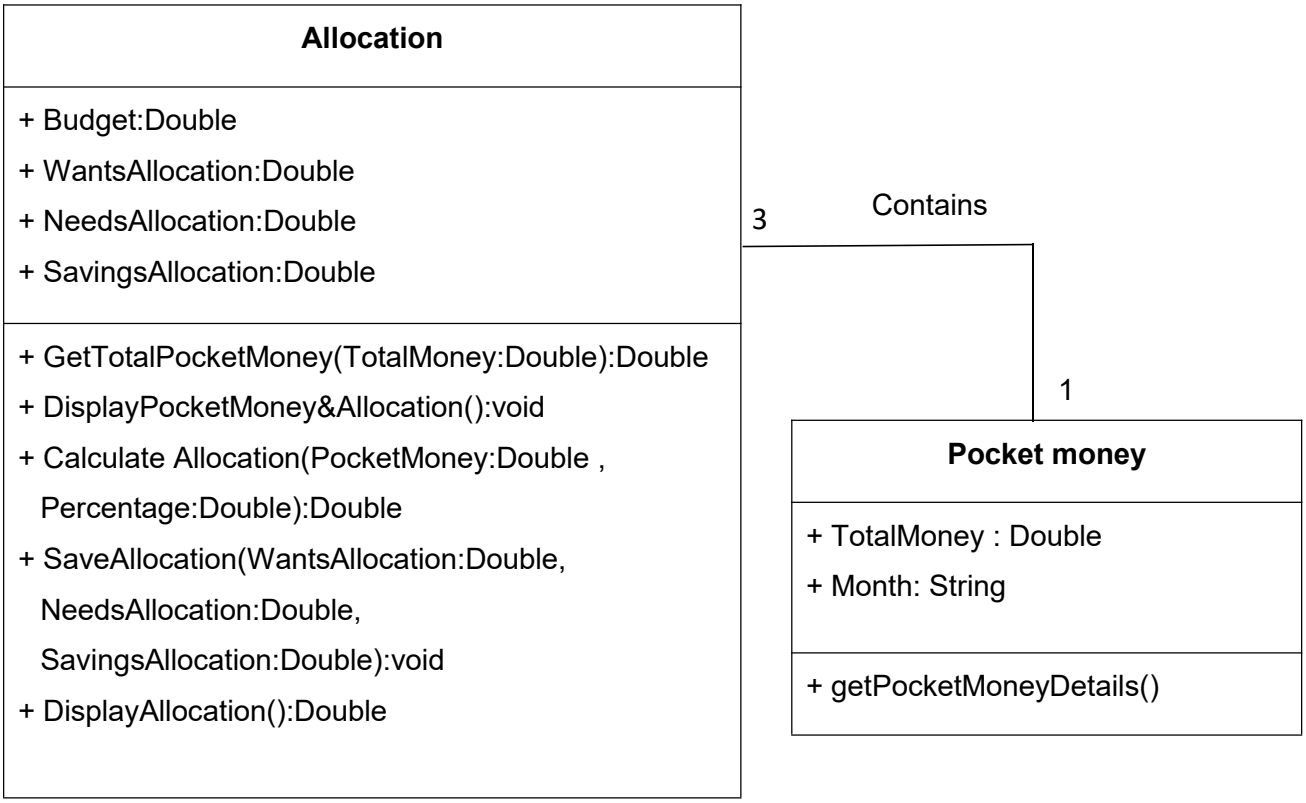
| Pocket money |
| --- |
| + TotalMoney : Double<br>+ Month: String |
| + getPocketMoneyDetails() |

| Pocket money | | | |
| --- | --- | --- | --- |
| TotalMoney | NUMBER | <pk> | NOT NULL |
| Username | VARCHAR2(4) | <fk> | NOT NULL |
| Month | VARCHAR (10) | | NOT NULL |
| AmountPocketMoney | NUMBER | | NOT NULL |

| TotalMoney | Username | Month | AmountPocketMoney |
| --- | --- | --- | --- |
| | | | |

**Diagram 5.2.2: Class Diagram and Relation Table For Pocket Money**

For this class diagram, we use the series of rule of thumb. From the diagram a, we can see clearly which is we convert all the double datatype into number, date is remain as a date in relational table. The primary key for this class is TotalMoney with datatype number and it is not null. The foreign key is Username, with datatype varchar2(4) and the constraint is not null. The datatype for data and AmountPocketMoney is date and number respectively. The constraint for both is not null.




| ALLOCATION | | | |
|---|---|---|---|
| Allocation_ID | NUMBER | <pk> | NOT NULL |
| TotalMoney | NUMBER | <fk> | NOT NULL |
| Username | VARCHAR2(4) | <fk> | NOT NULL |
| Expense | NUMBER | | |
| WantsAllocation | NUMBER | | |
| NeedsAllocation | NUMBER | | |
| SavingsAllocation | NUMBER | | |

Allocation Table

| Allocation_ID | TotalPocket Money | Username | Expense | WantsAllocation | NeedssAllocation | SavingsAllocation |
|---|---|---|---|---|---|---|
| | | | | | | |

**Diagram 5.2.3: Class Diagram and Relation Table For Allocation**

The Allocation class has been transferred into table. All the string types has been transformed into varchar2 with suitable length and all the integers and doubles has been changed into number. The primary key for this class is Allocation_ID which is not null. There are other attributes which can be null.

# 6 Interfaces Design

## 6.1 Sequence Diagram and Boundary Control Entity Approach

**Use Case 1:** Login

**SEQUENCE DIAGRAM 1: LOGIN**

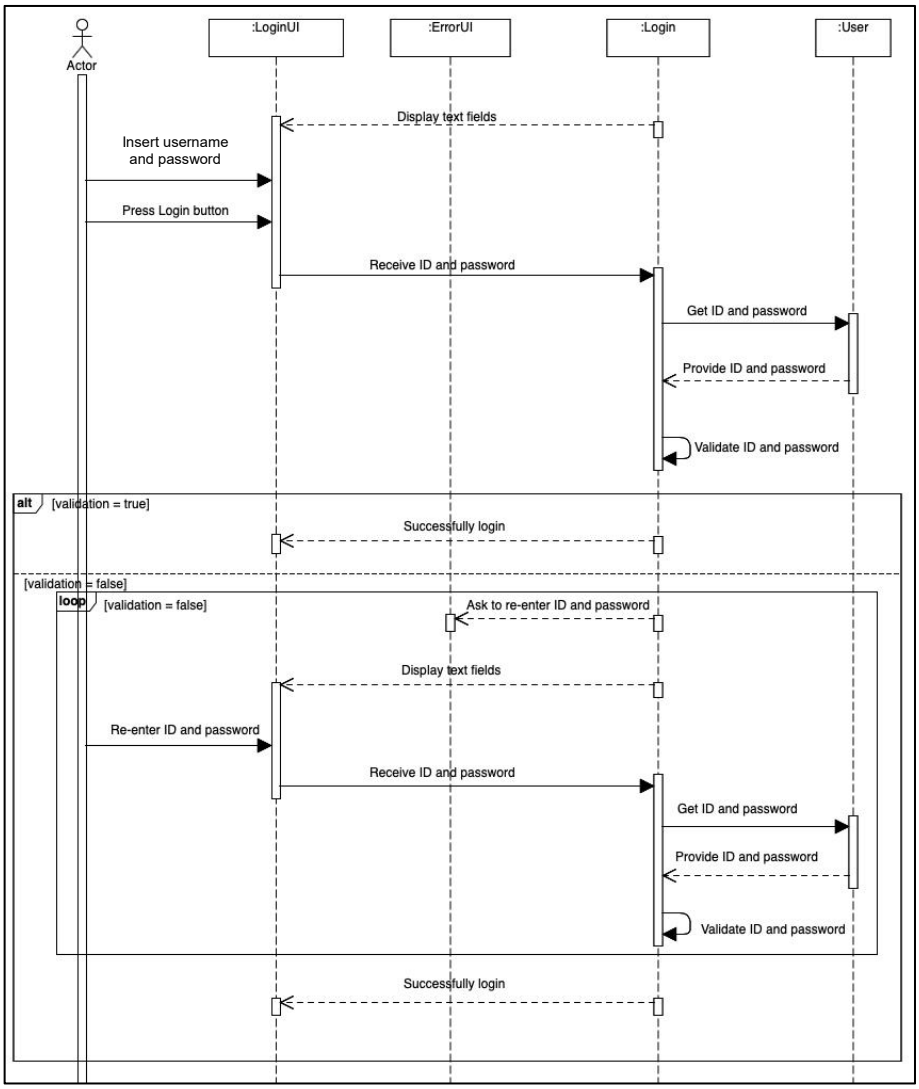



**Diagram 6.1.1: Sequence Diagram for Login**

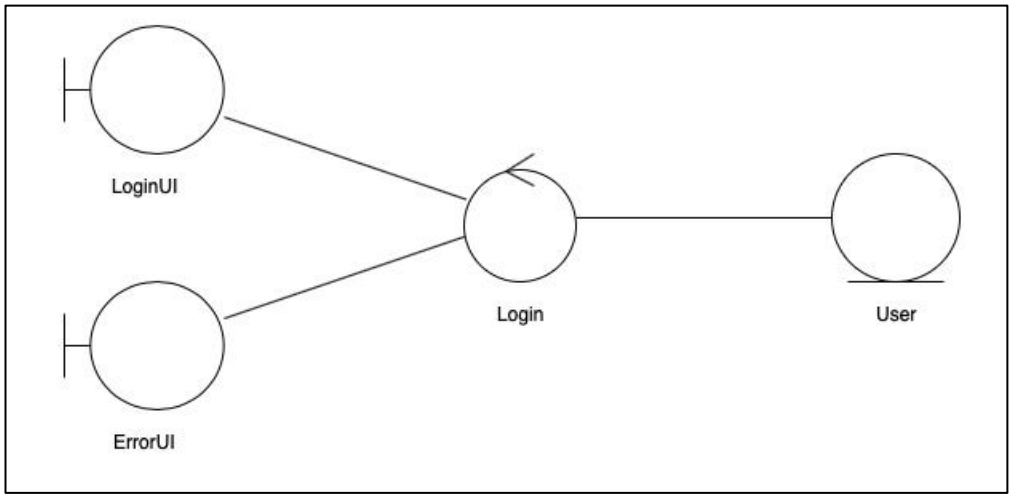



**Diagram 6.1.2: Communication Diagram for User**

Based on diagram 6.1.1 and 6.1.2, empty text fields in Login UI are displayed and user needs to enter the following attributes which are username and Password. Upon user clicked Login button, the boundary class act by submitting the username and Password to Login (control class) and proceed to get the username and Password the data from User entity. Then, the User class will provide username and Password for validation. Login UI will display a message on Login UI informing user that he or she successfully login. On the other hand, the alternatives when the validation result is invalid, Error UI will pop out informing that the user has entered wrong username or Password while asking user to re-enter username and Password. The system will validate again after user enters Login button.

**Use Case 2:** Register

## SEQUENCE DIAGRAM 2: REGISTER

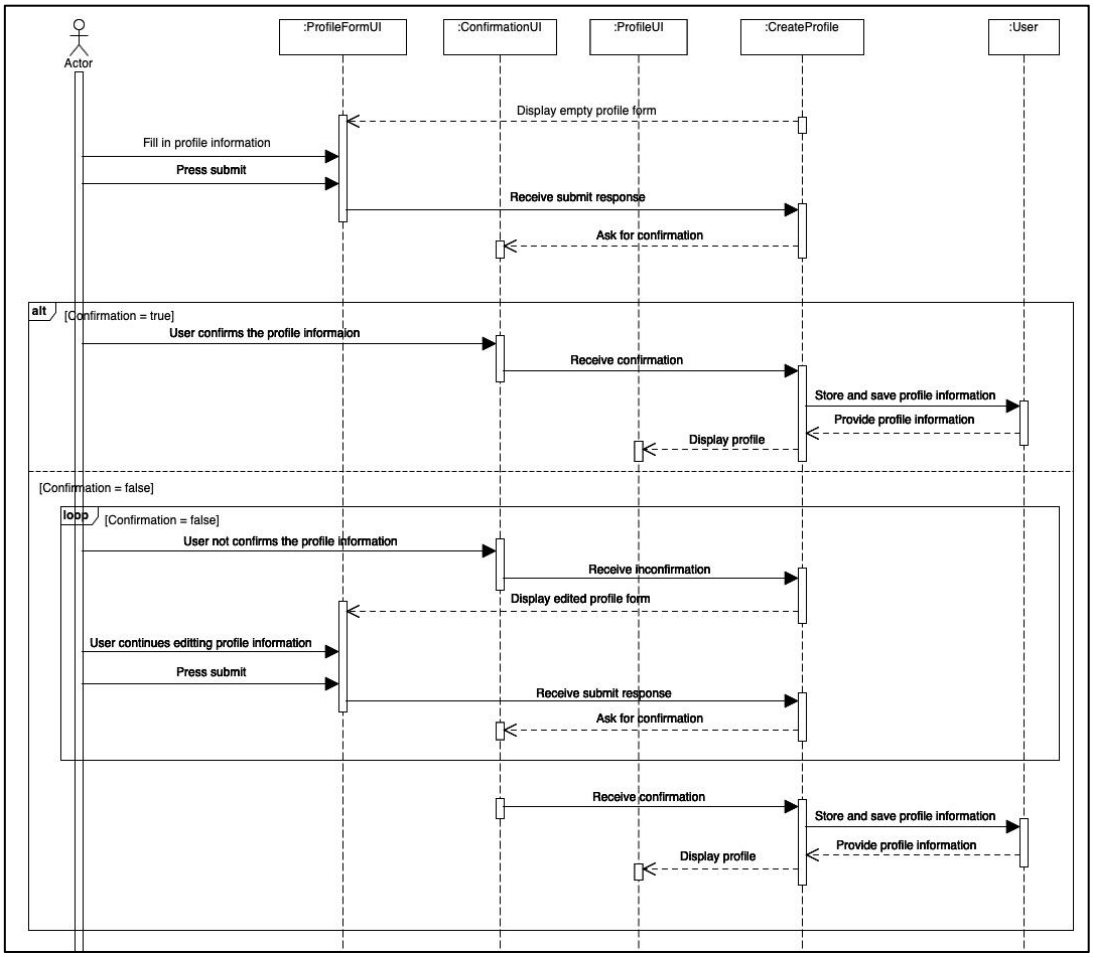



**Diagram 6.1.3: Sequence Diagram for Register**

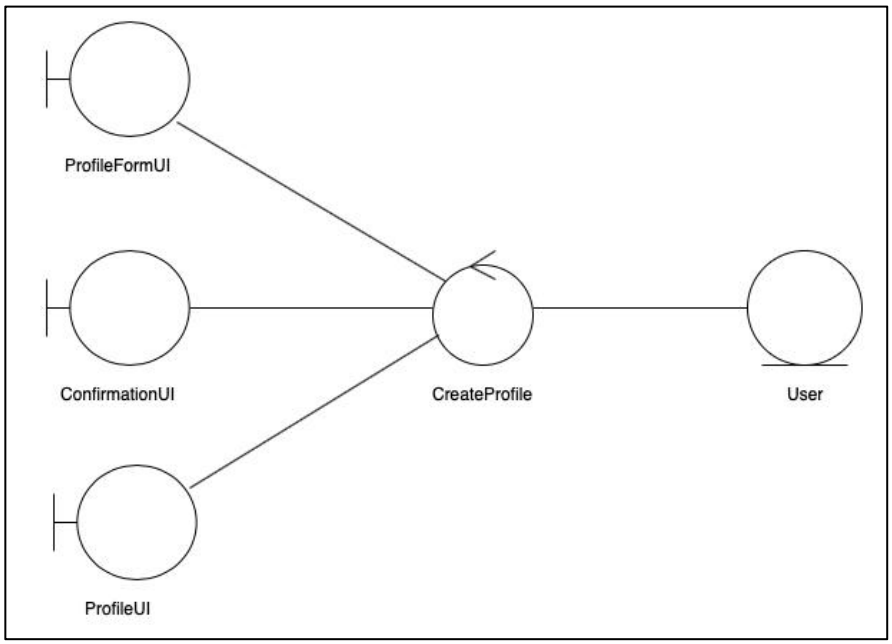



**Diagram 6.1.4: Communication Diagram for User**

Based on diagram 6.1.3 and 6.1.4, empty form in ProfileFormUI is displayed and user needs to enter the following attributes which are Username, Name, Password and Email. Upon user clicked Submit button, the boundary class act by submitting the submit response to CreateProfile (control class) and the system will prompt ConfirmationUI for confirmation. The user confirms and ConfirmationUI will store and save all the informations needed to User (entity class). On the other hand, the alternatives when the user not confirm, CreateProfileUI will display the edited form. User will continue editing the form at ProfileFormUI. When the user press submit button and confirms at ConfirmationUI, ConfirmationUI will store and save all the informations needed to User.

## SEQUENCE DIAGRAM 3: RECORD POCKET MONEY

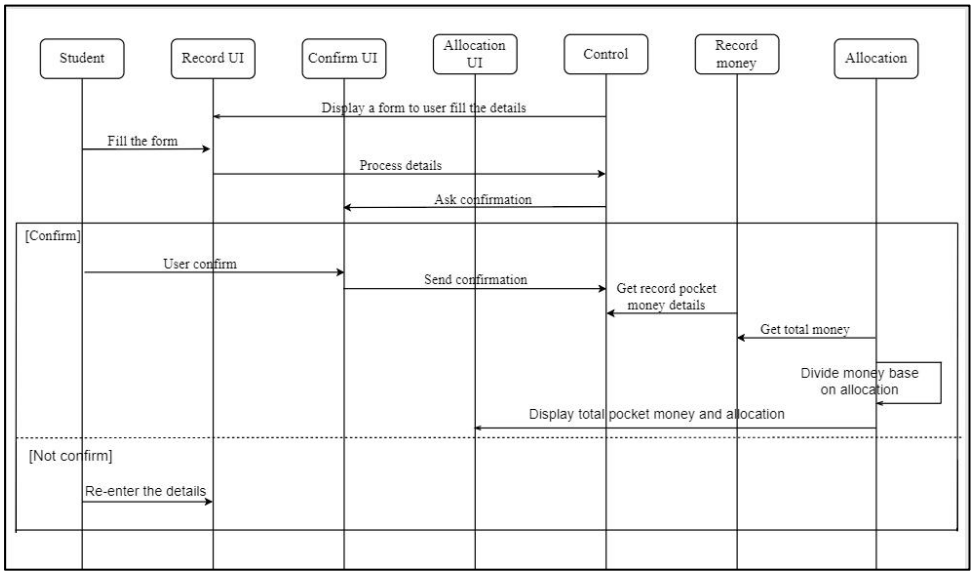



**Diagram 6.1.7: Sequence Diagram for Record Pocket Money**

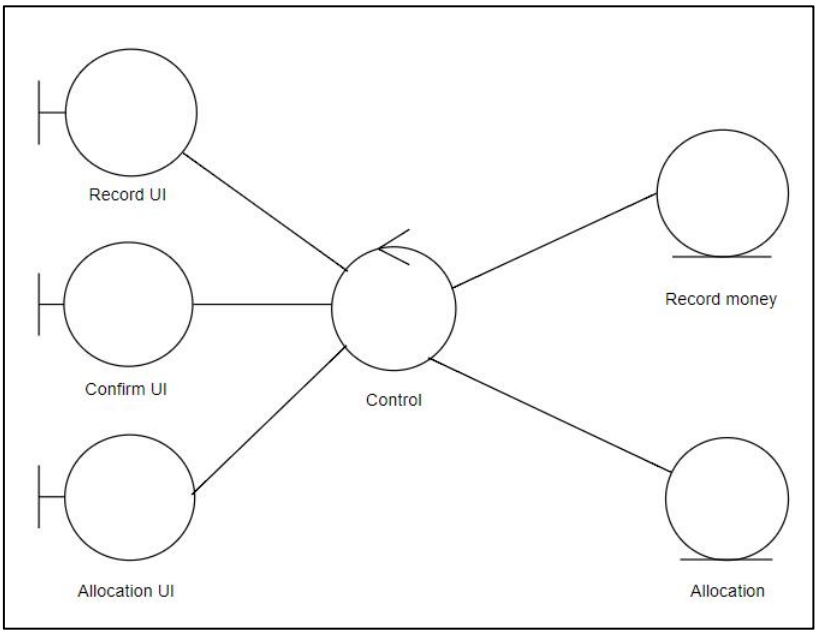



**Diagram 6.1.8: Communication Diagram for Allocation**

Based on the Diagram 6.1.7 and Diagram 6.1.8, the system will display a form to user fill the details about their pocket money such as amount the money that they have. Then the user submits the form. Once the user submits the form, the system will ask the confirmation from user, weather the user confirm the amount that they have. Then, the data will be saved in record money entity. The amount of money that they entered will divide into a few allocations and save in system. Once the system save the details of allocations, the system will display the allocation to user.

**SEQUENCE DIAGRAM 4: VIEW ALLOCATION**

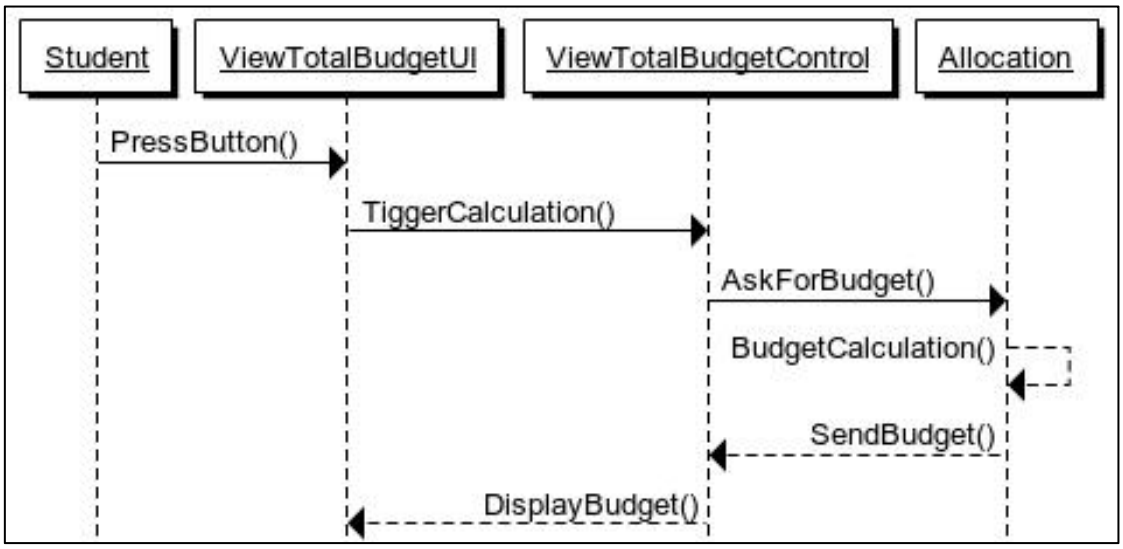



**Diagram 6.1.11: Sequence Diagram for View Total Money Management**

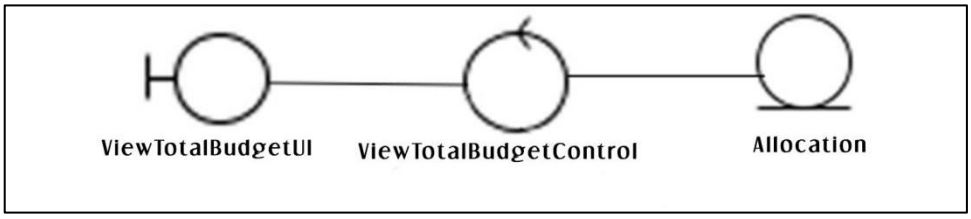



**Diagram 6.1.12: Communication Diagram for View Total Money Management**

The above diagram is a sequence diagram of View Total Budget. It involves one entity class which is allocation class; one interface class ViewTotalBudgetUI; and one control class which is ViewTotalBudgetControl. At first user have to press total budget button and the ViewTotalBudgetUI will trigger the ViewTotalBudgetControl. ViewTotalBudgetContol continues to ask Budget from Allocation Class.The Allocation class will calculate the budget and will send the budget to ViewTotalBudgetControl. ViewTotalBudgetControl will display the Budget at ViewTotalBudgetUI.

## SEQUENCE DIAGRAM 5: RECORD EXPENSE

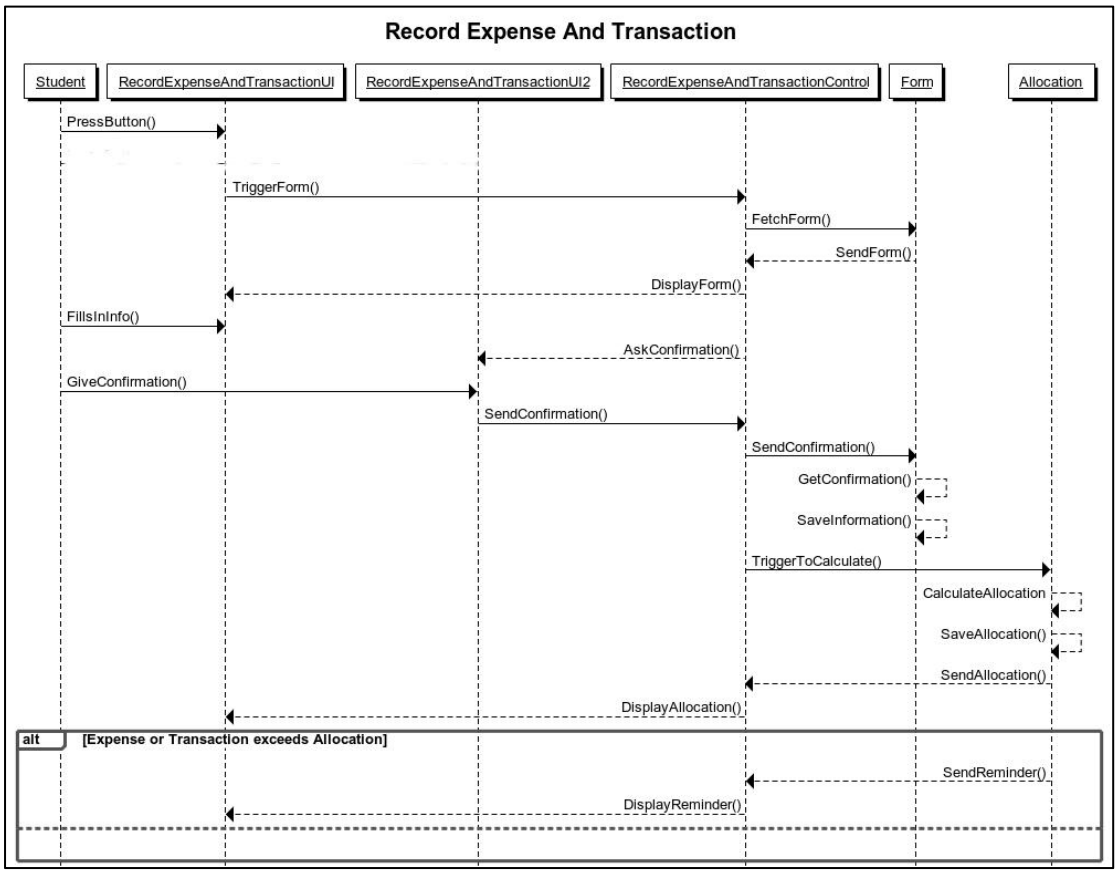



**Diagram 6.1.9: Sequence Diagram for Record Expense and Transaction**

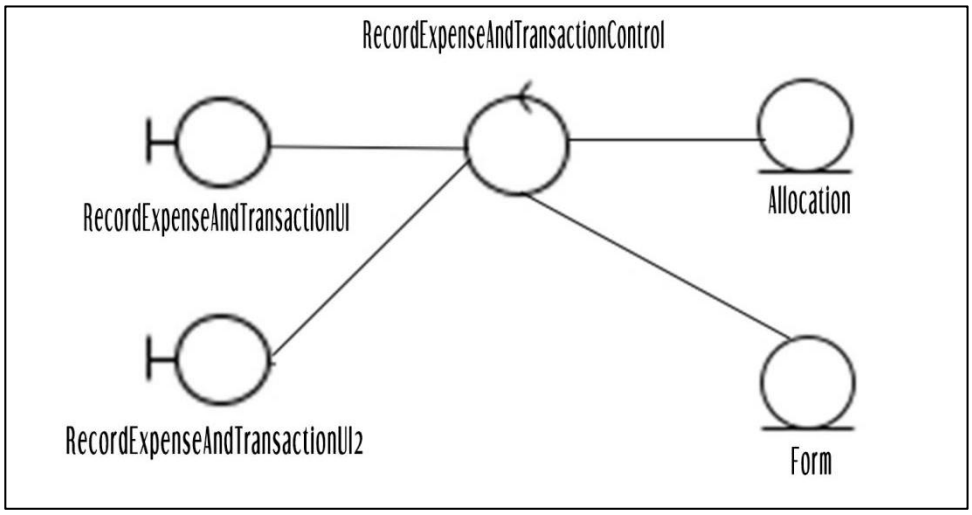



**Diagram 6.1.10: Communication Diagram for Record Expense and Transaction**

The above diagram is a sequence diagram of record expense. It shows the flow of the event to record expense and transaction. It involves two entity classes which are user class and allocation class; two interface classes record which RecordExpenseAndTransactionUI and RecordExpenseAndTransactionUI2 and one control class which is RecordExpenseAndTransactionControl. There are few steps will be carried out to complete this Function.

1. At first user have to press record button
2. RecordExpenseAndTransactionUI will trigger the RecordExpenseControl
3. RecordExpenseControl will fetch the form from Form class
4. Form class will send the form to RecordExpenseControl
5. RecordExpenseControl will display the form at RecordExpenseUI
6. The user have to fill in their expenses in the displayed form at RecordExpenseUI
7. RecordExpenseControl will ask confirmation at RecordExpenseAndTransactionUI2 with a new pop up window
8. The user have to give confirmation at RecordExpenseAndTransactionUI2
9. RecordExpenseUI2 will send confirmation to RecordExpenseControl
10. RecordExpenseControl will save the expenses in order to deduct from the allocations.

## 6.2 Prototype

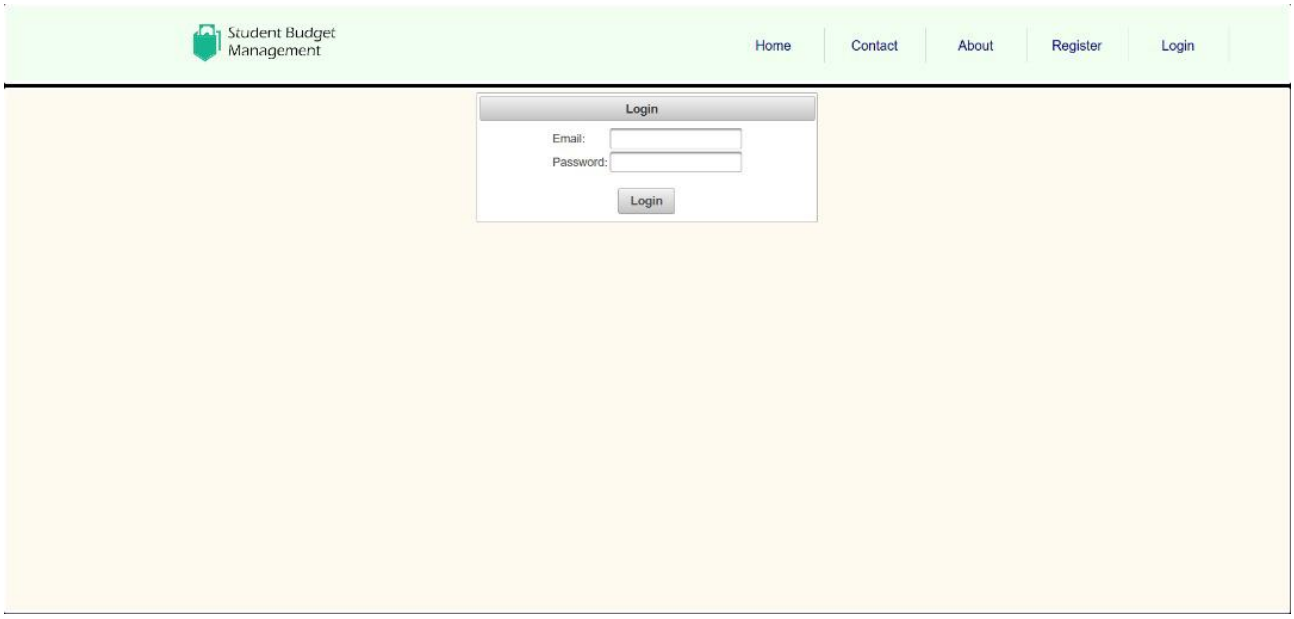



**Diagram 6.2.1: Login Interface**

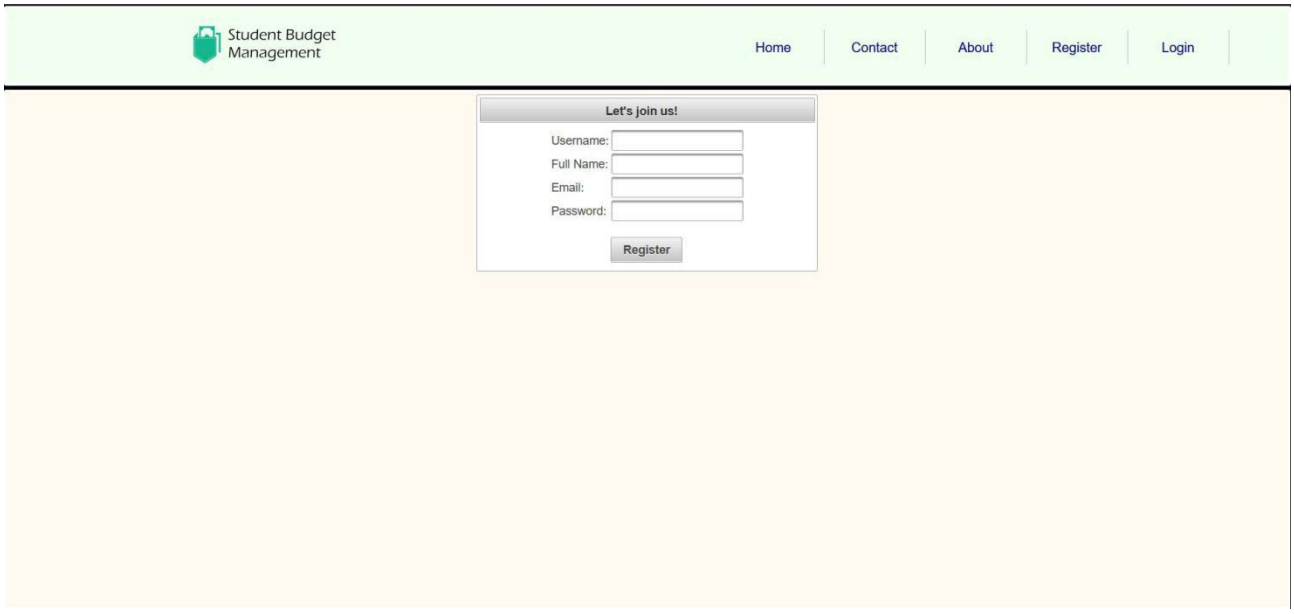



**Diagram 6.2.2: Create Profile Interface**

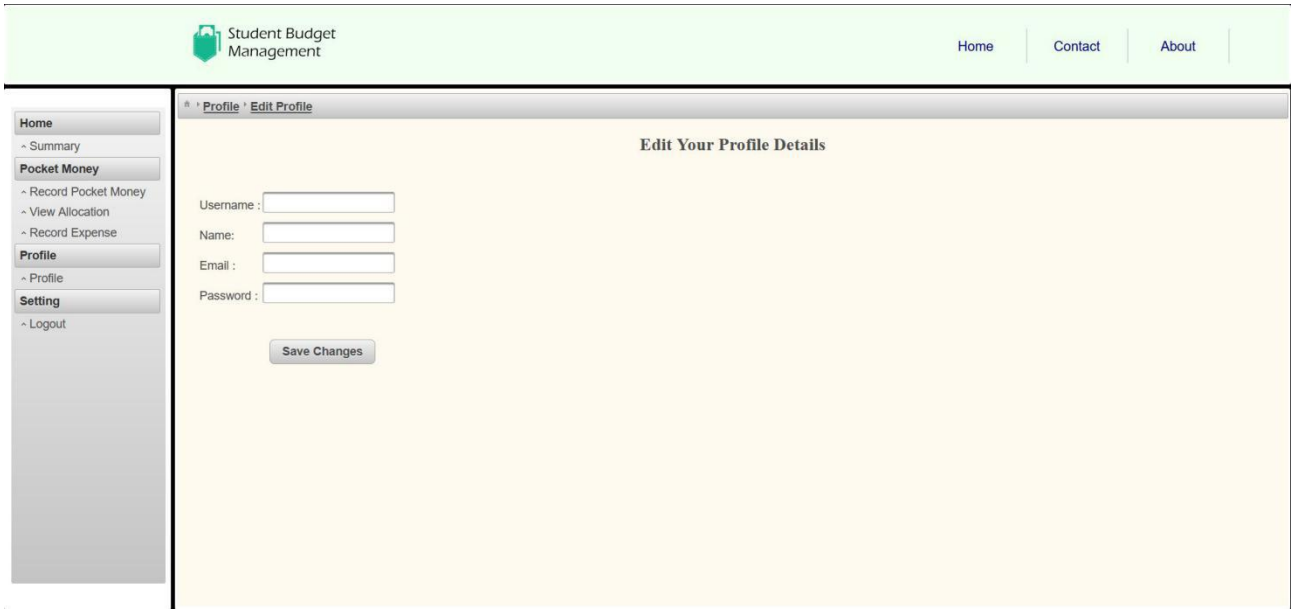



**Diagram 6.2.3: View and Edit Profile Interface**

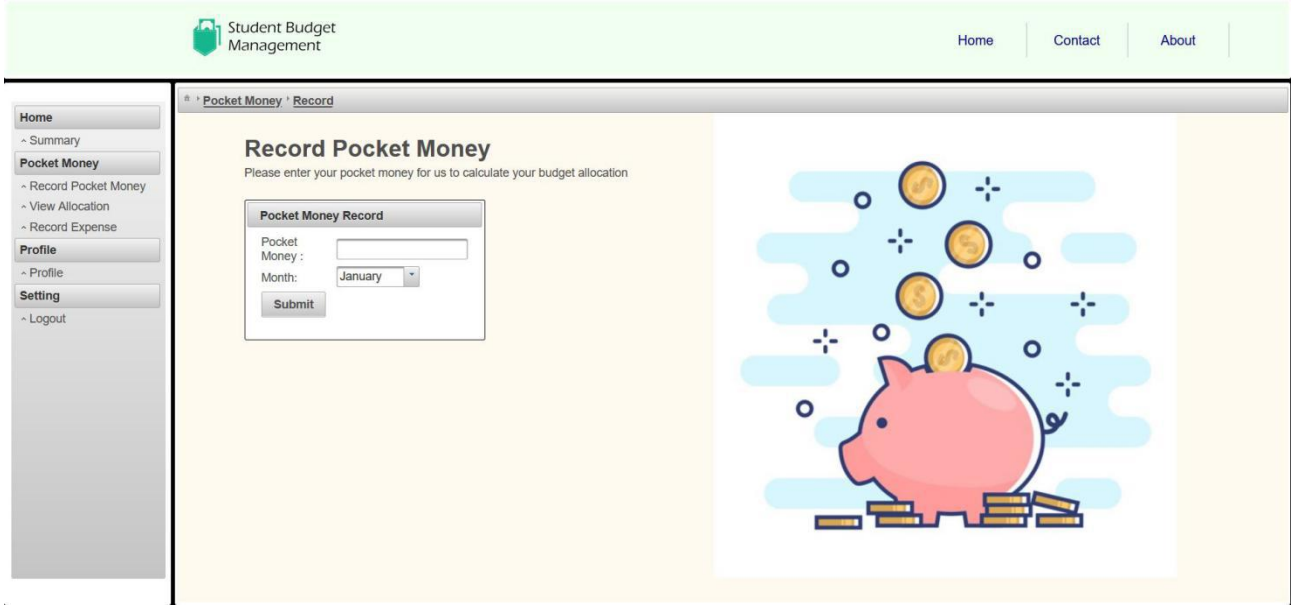



**Diagram 6.2.4: Record Pocket Money Interface**

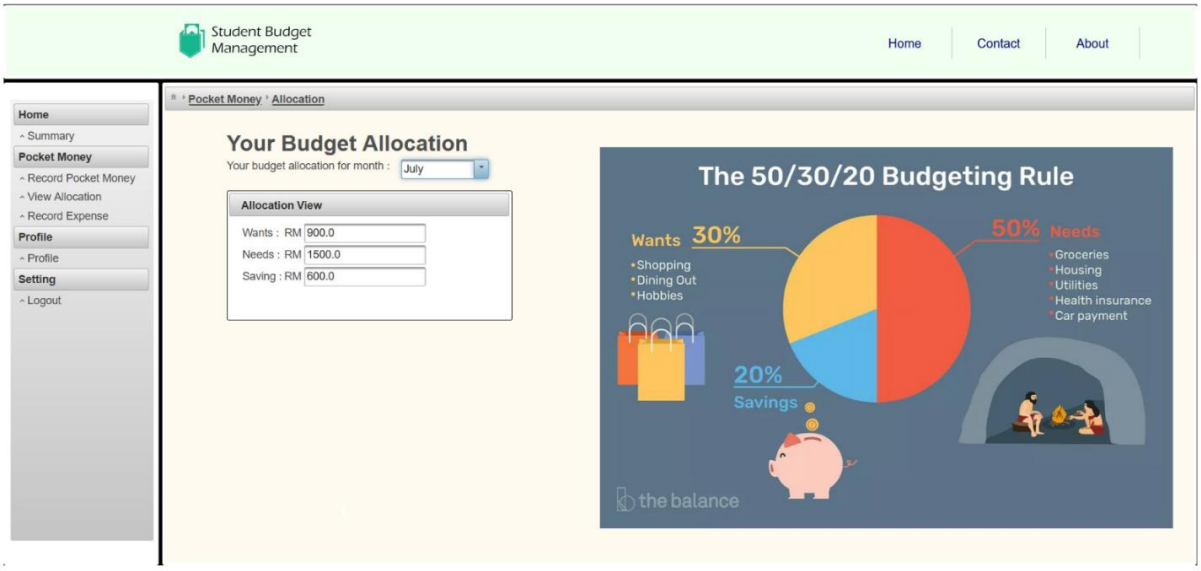



**Diagram 6.2.5: View Allocations Interface**

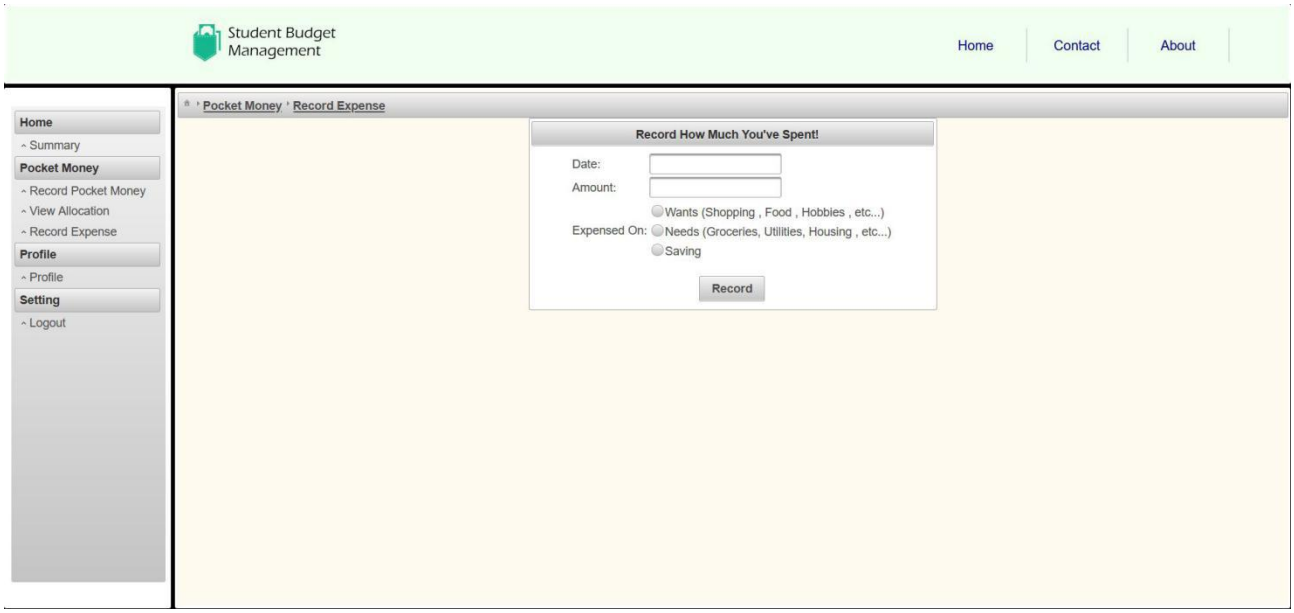



**Diagram 6.2.6: Record Expenses Interface**

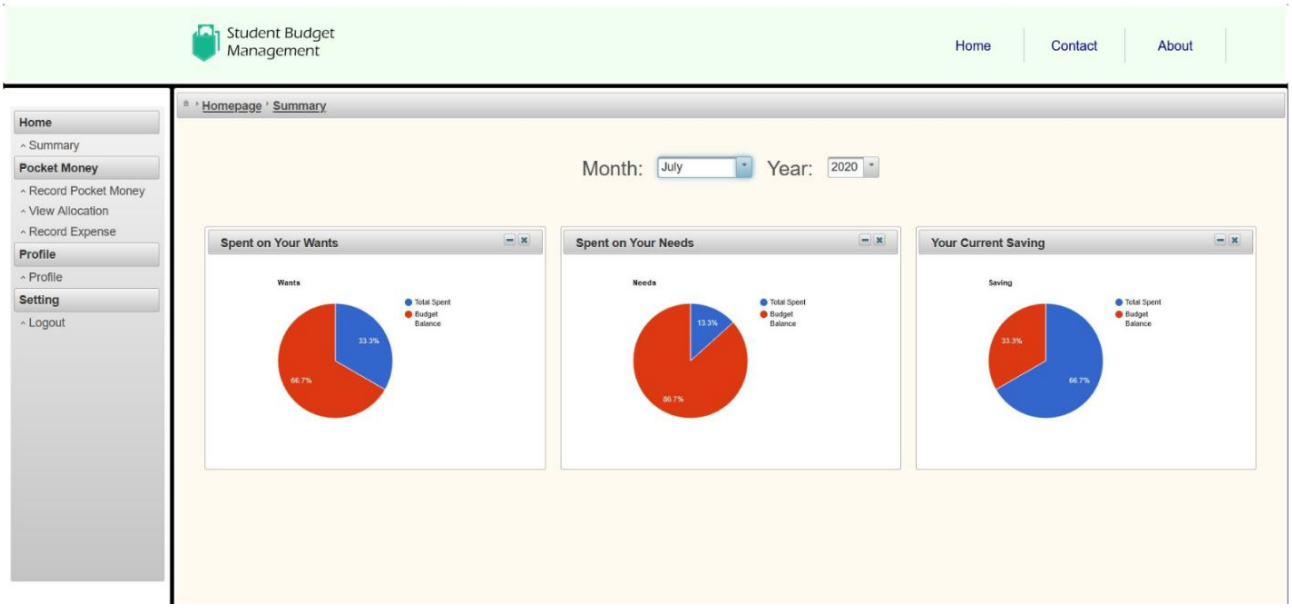



**Diagram 6.2.7: View Summary Interface**

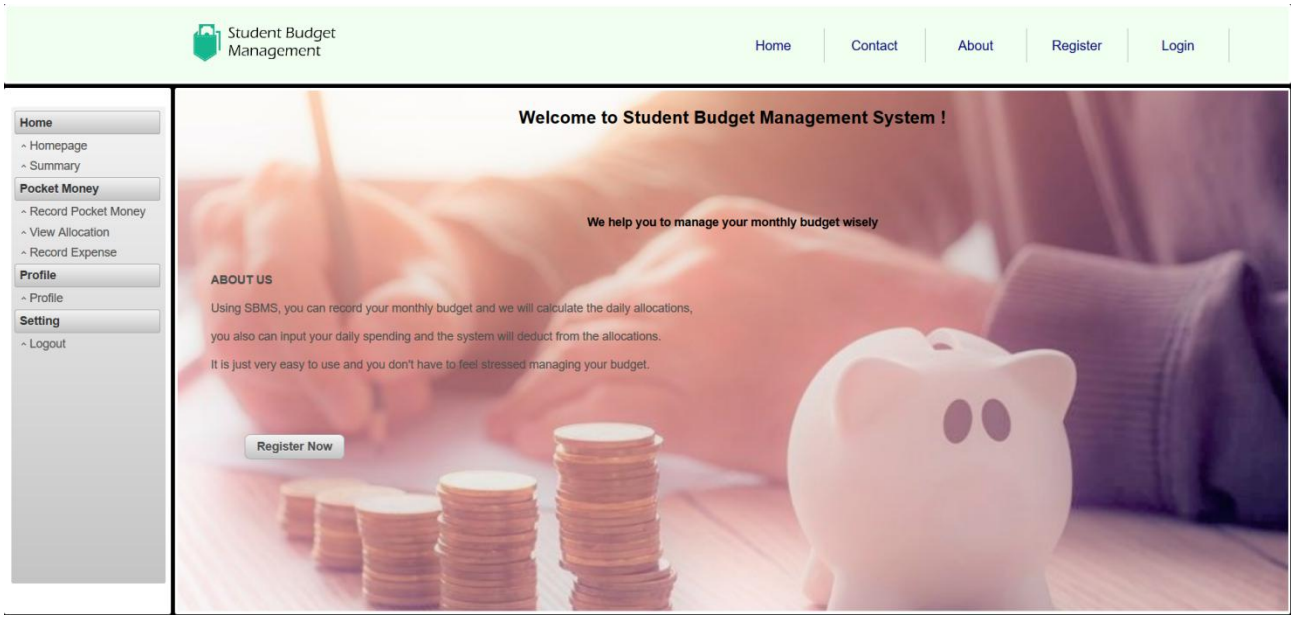



**Diagram 6.2.8: View Homepage Interface**

# 7 Other Analysis Models

## 7.1 Deployment Diagram

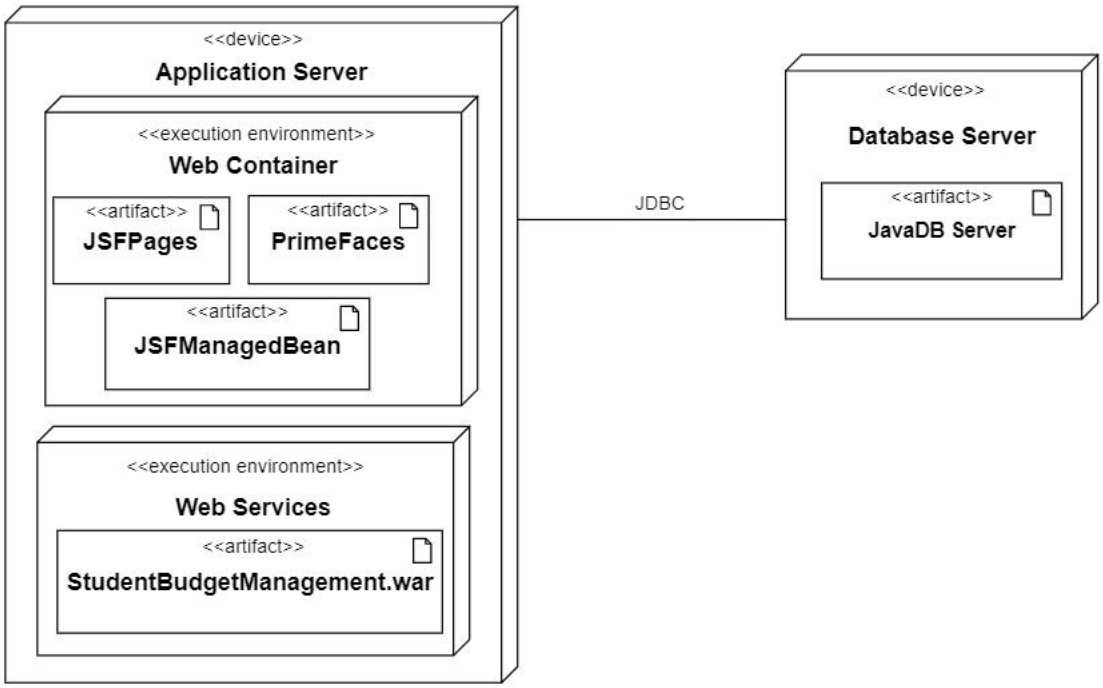



**Figure 7.1 Deployment diagram for SBM**

Figure 7.1 shows the deployment diagram which represents the structure of the run-time configuration of the system. The nodes shows the devices of execution environments in the system and consists of artifacts which are the entities used for deployment. Such as, web services, managed bean and the JSF pages. SBM system consists of two major nodes which are the databases server and the application server, where they communicate through the JDBC (Java Database Connectivity) in order for the client to access the database.

## 7.2 Package Diagram

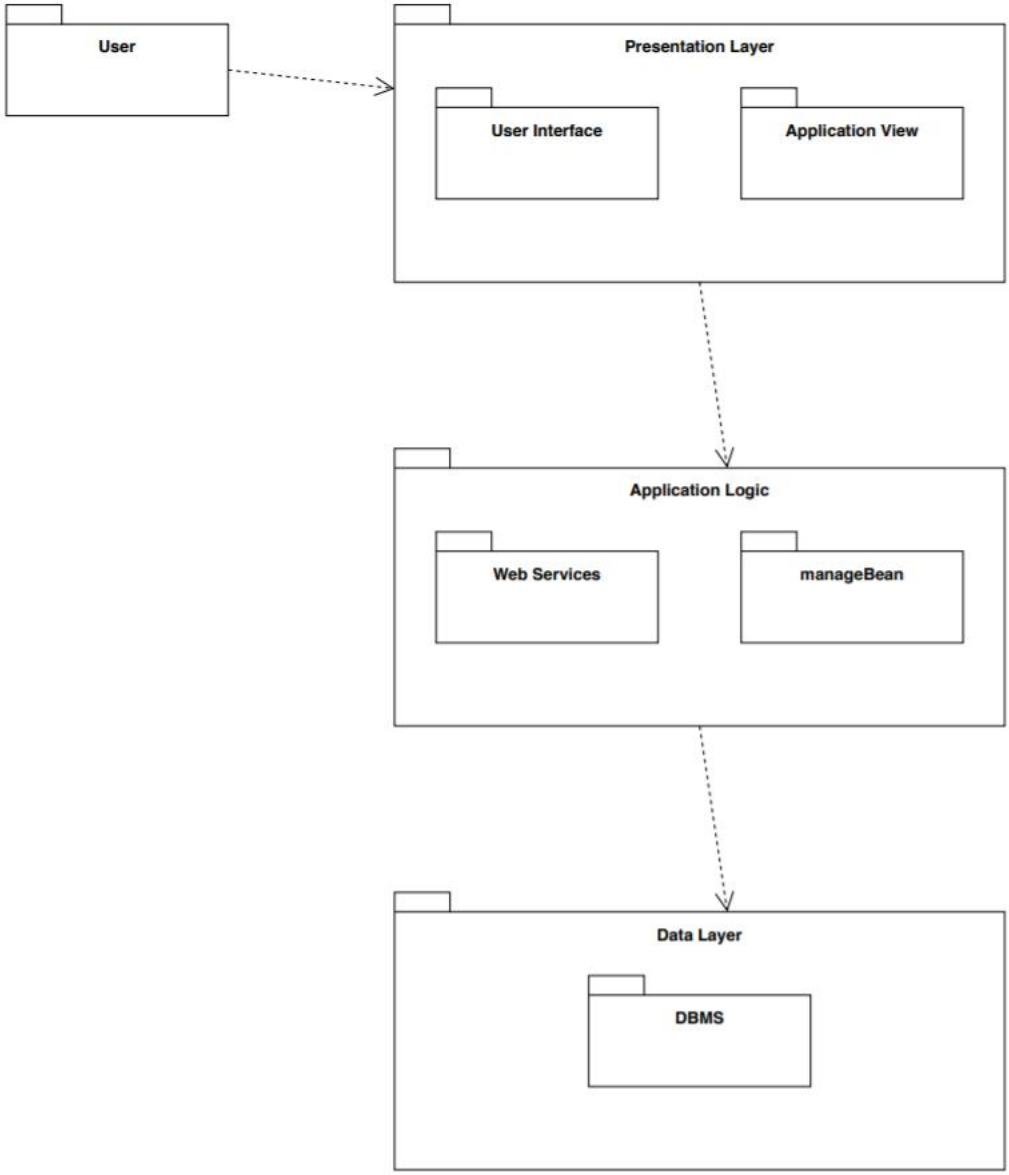



**Figure 7.2 Package diagram for SBM**

Figure 7.2 illustrates the package diagram of the system. It represents a layered architecture of three major layers (represented as packages). The layers are presentation (responsible for the view and user interface of the system which user react with), second layer is the application logic (responsible for processing user requests, and consists of the web services and managed bean), last layer is the data layer (responsible for storing the user's data).

## 7.3 Component Diagram

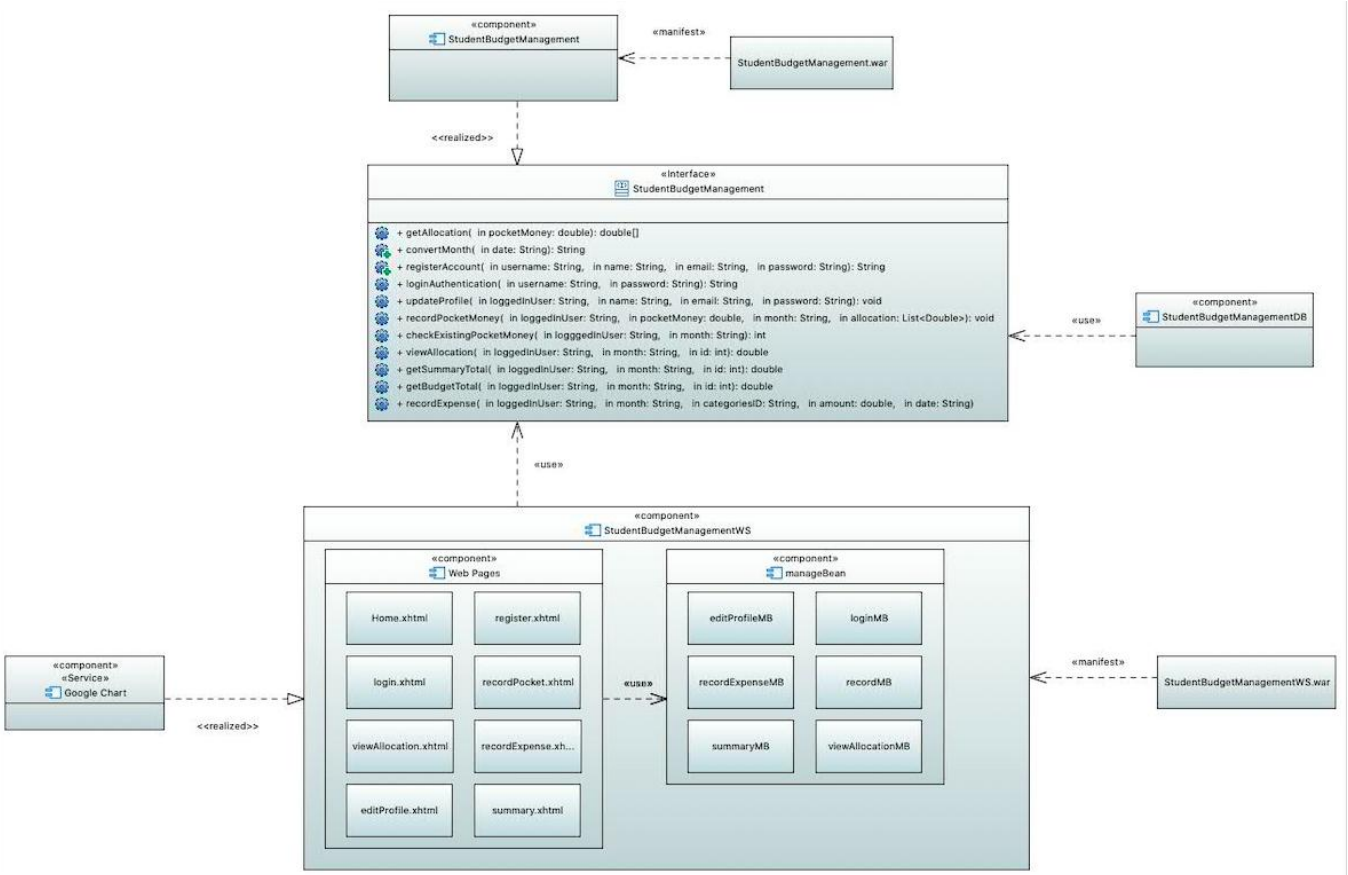



**Figure 7.3 Component diagram for SBM**

Figure 7.3 shows the component diagram of the system. It represents the structure of the system's major components and what interfaces they realize or use, as well as the interactions between them.