**Software Design Document**

**for**

**Personal Budget Manager Application**

**Version 0.2**

**Prepared by**

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

24/3/2019

**TABLE OF CONTENTS**

[1. Introduction 1](#_Toc1483564)

[1.1 Document Purpose 1](#_Toc1483565)

[1.2 Product Scope 1](#_Toc1483566)

[1.3 Definitions and Acronyms 1](#_Toc1483567)

[1.3.1 Definitions 1](#_Toc1483568)

[1.3.2 Acronyms 1](#_Toc1483569)

[2. Architecture Design 2](#_Toc1483570)

[2.1 Rationale 2](#_Toc1483571)

[2.2 Software Architecture Diagram 2](#_Toc1483572)

[2.3 System Topology 3](#_Toc1483573)

[3. Software Interface Design 10](#_Toc1483579)

[3.1 System Interface Diagrams 10](#_Toc1483580)

[3.1.1 User Interface 10](#_Toc1483581)

[3.2 Module Interface Diagrams 10](#_Toc1483580)

[3.2.1 Model Interface 10](#_Toc1483581)

[3.2.2 View Interfaces 16](#_Toc1483582)

[3.2.3 Controller Interfaces 16](#_Toc1483583)

[3.3 Dynamic Models of System Interface 16](#_Toc1483584)

[3.3.1 Add Expense Scenario 10](#_Toc1483581)

[3.3.2 Change Payment Status Scenario 16](#_Toc1483582)

[3.3.3 Remove Expense Scenario 16](#_Toc1483583)

[4. Internal Module Design 17](#_Toc1483585)

[4.1 Module <Model> 17](#_Toc1483586)

[4.1.1 Model Class Diagram 1](#_Toc1483568)

[4.2 Module <View> 17](#_Toc1483587)

[4.2.1 View Class Diagram 1](#_Toc1483568)

[4.2 Module <Controller> 17](#_Toc1483587)

[4.3.1 Controller Class Diagram 1](#_Toc1483569)

**LIST OF FIGURES**

# Introduction

## Document Purpose

The purpose of this document is to describe the detailed structure of the components of the Personal Budget Manager Application and the precise implementation details to satisfy the requirements in the Software Requirements Document. This includes the Architectural Design (AD), Interface Specification (IS) and Detailed Class Design (DD) of the application. The AD part focuses on subsystem and module diagram, the IS part focuses on describing services provided by modules, and the DD part focuses on data description and declaration.

## Product Scope

The development plan is divided into 3 iterations and the document will be updated accordingly to address the content in that phase. This iteration of the Personal Budget Manager extends the previous iteration by adding the following features: The expense types (purchase, bill) instead of being atomic, they can be composite of sub-items expenses, which in turn may be complex expenses. Sub-items expenses are dependent on their super-expenses. If a super-item expense is removed from the expense list all its sub-items expense are also removed. Similarly, if the payment status of a super-item is set to complete or incomplete, the completion status of all sub-items will change accordingly.

In this increment implement the following:

− View expenses with their hierarchy in case of composite expenses.

− Delete expense.

− Set expense to paid/unpaid.

− Add expense: purchase or bill as single/periodic/composite.

− Hiding / unhiding paid expenses: where the user has the option to change the current view and to hide/unhide paid expenses (at the root level).

## Definitions and Acronyms

### Definitions

Purchase A type of day-to-day expense

Bill A type of recurring expense

Composite A composite of sub-item expenses

### Acronyms

PBM Personal Budget Manager Application

SRS Software Requirement Specification

SDD Software Design Document

# Architecture Design

## Rationale

## The architecture chosen for the PBM is the Model View Controller model (MVC). The MVC architecture is made up of 3 separate components. There are one model, several views and controllers. This architecture allows the three components to be developed separately from one another and they can be done in parallel. Either of the components can be updated without affecting the other components as long as their application program interface remains the same. With a common API, the 3 models can be seamlessly integrated into one App.

## C:\Users\Administrator\Desktop\1200px-MVC-Process.svg.png

## Software Architecture Diagram

## The model is the core of the PBM where the functionality and data are stored and manipulated. All of the computations that are performed are done in this component as well as all data that needs to be processed. Moreover, whenever there are changes made to the model, the model updates the view.

## The view is the graphical user interface (GUI) of the PBM and displays the data from the model. Whenever the model changes, the view responds to those changes by updating itself. The view also gets updated by the controller, as the controller performs simple data validations on user input and updates the view. Different views can be developed in order to present different interfaces.

## The controller is what the players use to interact with the PBM. All information is entered via the controller, which it then passes on to the model.

## C:\Users\Administrator\Desktop\PBM_MVC.jpg

## System Topology

## The PBM is to be developed for a standalone environment. All three components of the MVC model will be integrated into one executable. Moreover, the PBM does not require any third party software to run, or an Internet connection.

## Software Interface Design

## System Interface Diagrams

## The only system level interface in the PBM is the user interface, as the game does not employ any software or hardware interfaces. The user interface is GUI, which allows users to interact with the application.

## user interface

## *main panel*

This is the main panel of the PBM. There is an expense list and five buttons: add expense, remove expense, mark expense paid/ unpaid, hide/show paid expenses and create composite expense. With different expense types, they may input different details.

## 

## Figure. PBM home window

***Add expense***

When users click the button ‘add expense’, another interface will pop up. Here they can input description of their expenses. If users choose purchase expense, there are four drop down buttons where users can change types of expenses: purchase or bill, paid status: paid or unpaid, paid method: cash, debt or credit, expense category: default, food, dining, grocery, gas, parking, transportation, taxi, mortgage, saving, insurance, utilities, wireless, health, cable, internet, automobile and entertainment. If users choose bill expense, there is one more button where they can set interval due date: weekly, biweekly, monthly, quarterly or yearly.

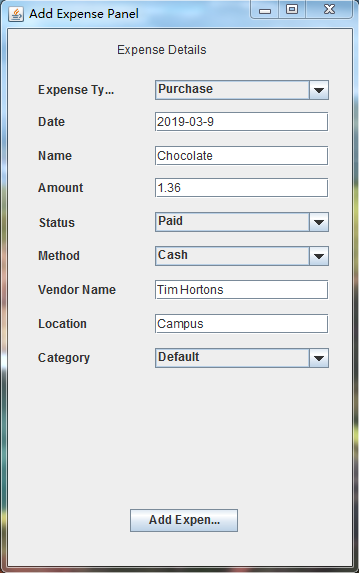
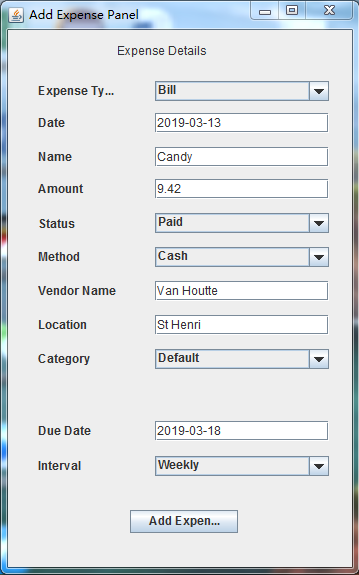
 

Figure. Add expense: Purchase Figure. Add expense: Bill

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

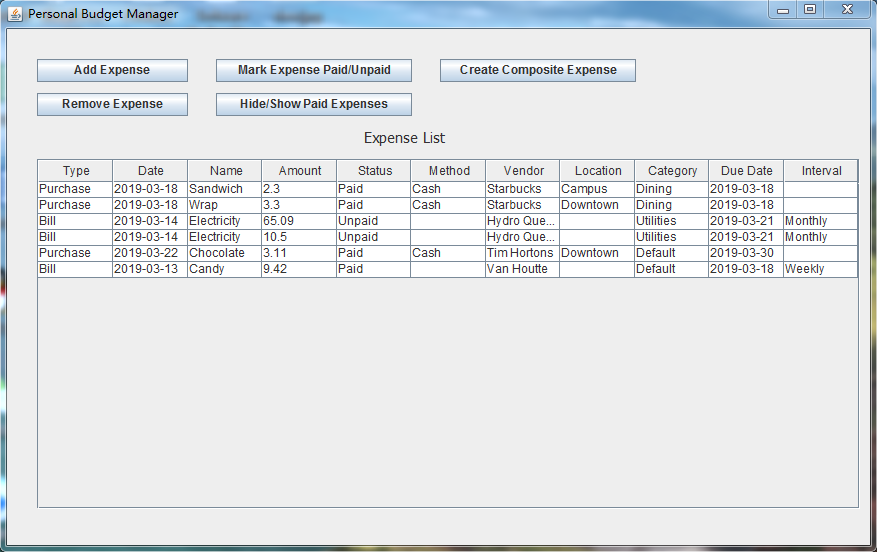


Figure. Updated expense list

***Remove expense***

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

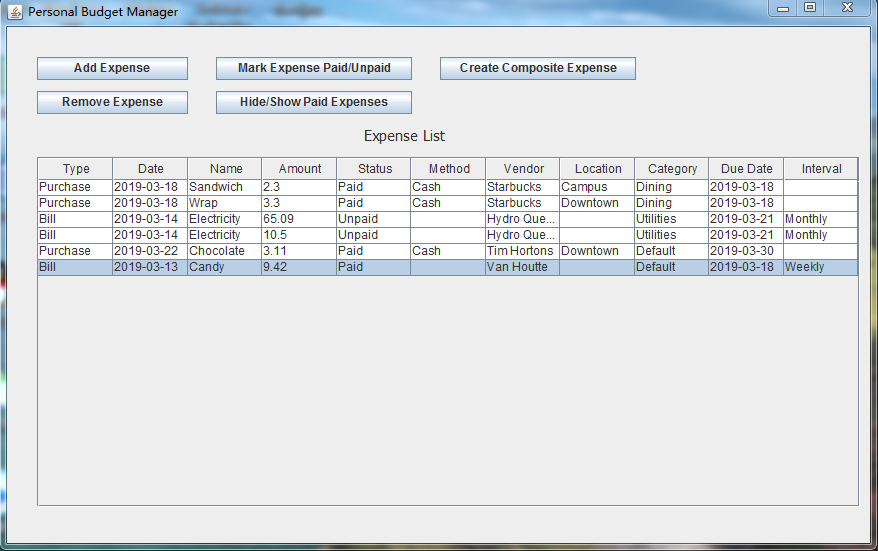


Figure. Remove expense (a)

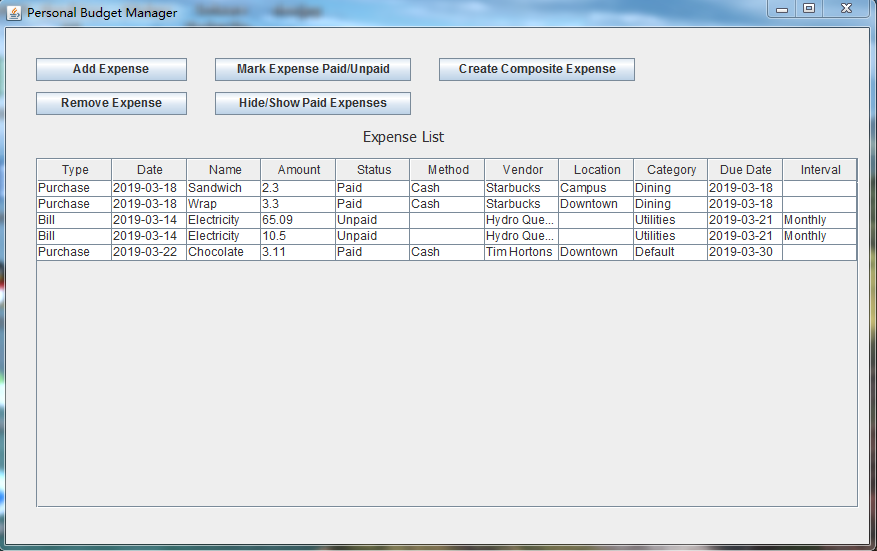


Figure. Remove expense (b)

***Mark expense paid/unpaid***

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

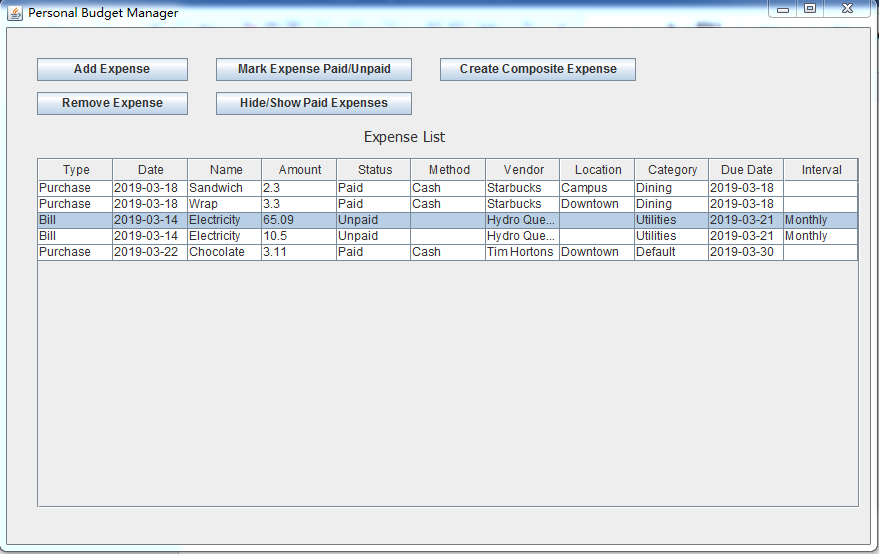


Figure. Mark paid/unpaid (a)

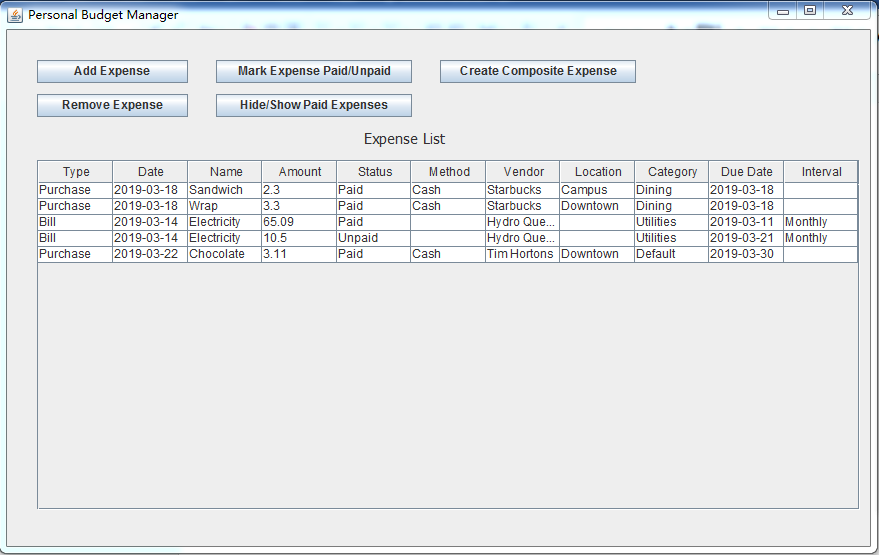


Figure. Mark paid/unpaid (b)

***Hide/show paid expenses***

Click the button ‘hide/show paid expense’, then the list will only show the unpaid expenses. Click the button again, the list will back to original look.

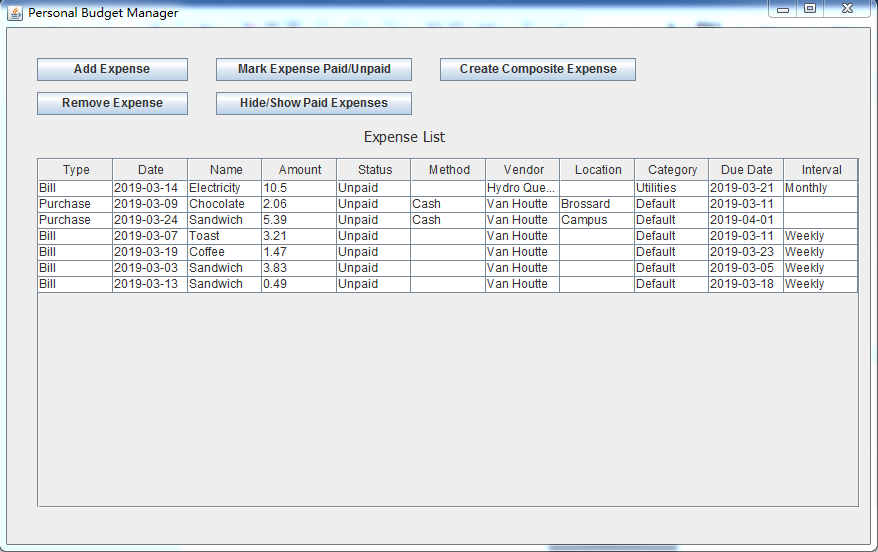


Figure. hide/show paid expense

***Create composite expense***

## Module Interface Diagrams

### Model Interface

### View Interface

### Controller Interface

## Dynamic Models of System Interface

## In order to better portray the interactions between the system modules, we have chosen some scenarios, or major functionalities of the system and will explain and depict them using sequence diagrams. We have elected to use sequence diagrams because they depict the interaction between the classes (or objects) of the system and also show the sequence of calls (or messages) that occur.

### Add Expense Scenario

## C:\Users\Administrator\Desktop\AddSimpleExpenseSeqDiagram.jpg

### Change Payment Status Scenario

## C:\Users\Administrator\Desktop\ChangePaymentStatusSeqDiagram.jpg

### Remove Expense scenario

## C:\Users\Administrator\Desktop\RemoveSimpleExpenseSeqDiagram.jpg

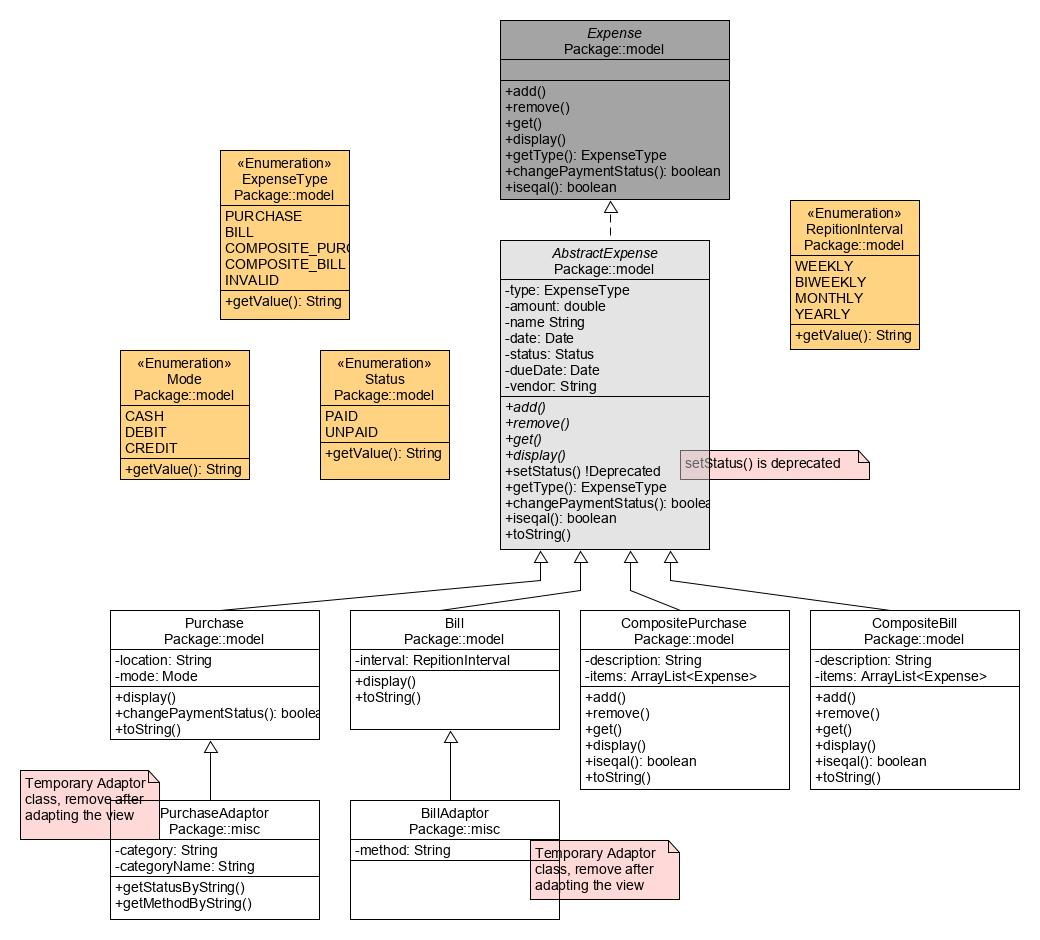
## Internal Module Design

## The system is divided into three modules, the model, the view and the controller. The modules interact together using their respective interface. In this section, we will describe the detailed design of three modules.

## Module <Model>

## The most important module is the model. This module is used to implement application methods and store data. The model consists of six classes, including *AbstractExpense, ExpeenseKey, Bill, Purchase, CompositeBill* and *CompositePurchase.* Each class contains the attributes and methods.

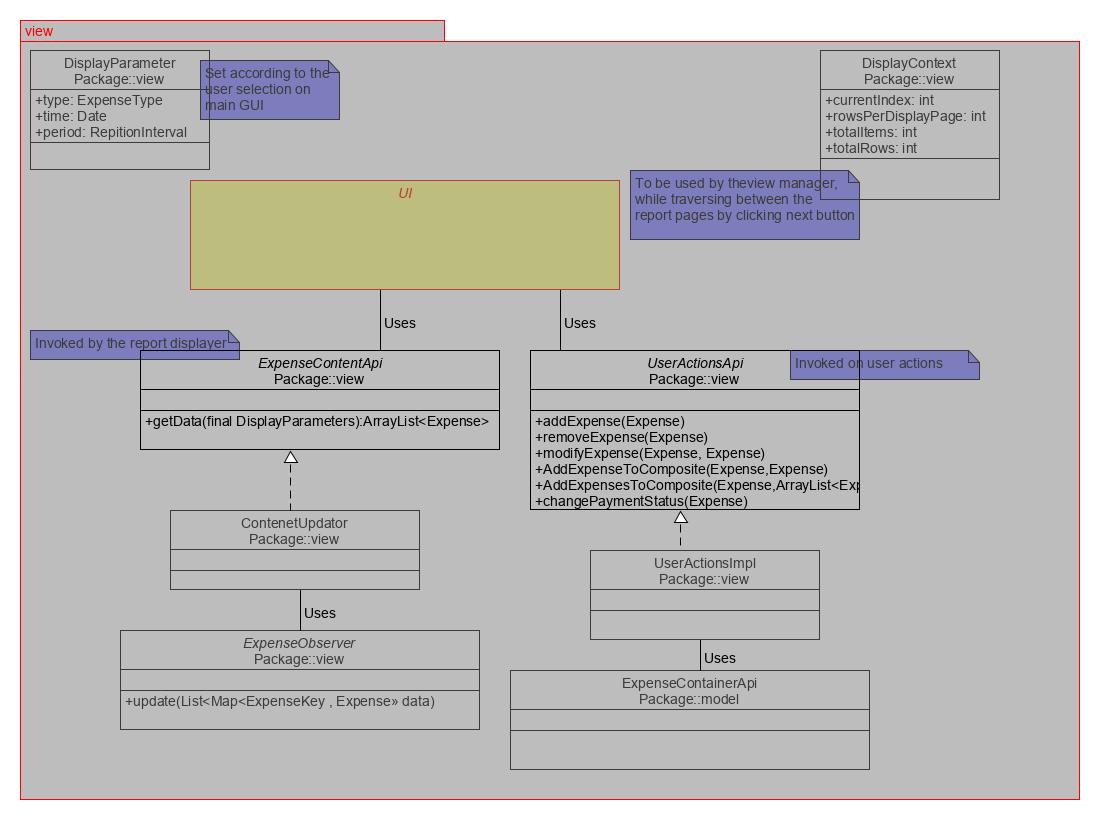
### Model Class Diagram



## Module <View>

## The module View consists of different views that users can see in the system.

### View Class Diagram



## Module <Controller>

## For the Controller module, we have five classes, including *DatabaseStore, ExpenseContainerImpl, FileLoaderImpl, FileStore* and *InMemoryStore.*

### Controller Class Diagram

