Wallet Watcher

By Team No.7:

Mustafa Syed

Muhammad Shahid

Zhenxu Wang

Jinsong Xie

Brahmjot Grewal

Table of Contents

Wallet Watcher Vision Statement

The Wallet Watcher app allows users to be able to track their monthly expenses, along with showing outcomes of money they can save, budget planning, and tracking the earnings they make through investments of any kind. The user will make an account and input any expenses, along with the monthly or annual income and the expenses that they have. By doing this, it will allow the user to be able to hold all expenses into one area, along with those expenses being categorized, and show the amount of money they have, and what they can do to increase their financial health by budgeting or saving their money.

The app will include many features that help the user be able to budget and save their income accordingly. The biggest features include a budget planner, expense tracker, and an earnings tracker. The budget planner's main functions include a user being able to set the goal of how much they want to save monthly or yearly or control how much , and suggested expenses that they can drop in order to achieve that goal (ex. A user is spending too much money on entertainment, and can drop a subscription if they do not use it anymore).

The expense tracker feature will allow users to be able to visualize recurring bills or other that they have to pay monthly or annually, along with a calendar which shows when each bill needs to be paid or each expense took place, so that they are never late on a payment. Furthermore, the user can also see any other expenses that they have including a mortgage, line of credit, etc. so that they can know exactly how much is left to be paid. These expenses can also be seen on a chart of the user's choice to be able to show how much they spend in each category and where they can spend less to save more.

The earnings tracker feature shows the user any investments that they have inputted, and what the losses or gains on those investments are and how they can impact the users' savings or budget goals.

At the end of the month the user is given a monthly report of all expenses, earnings, and any budget goals they have reached along with a "savings score" which shows the user how much they have saved compared to the last month. At the end of the year, the user will be able to see the average "savings score" and how well they did over an annual period of time.

The app will be considered a success based on two criteria. First, after some thorough use, financial managers, financial advisors will be asked if they can continue to use an app like this, and how this app is beneficial to them and if they would use this app over another app. Secondly, at the end of the year there will be an average taken of every single user's "savings score" and if it is above a certain percentage, then that means that users have had a positive impact from the app, and have been saving more than before.

Iteration 1

Feb 17, 2023

General Planning:

We are using an online planning tool "*Jira Board*".

We first add all detailed user stories as individual classes to the backlog, we are stilling adding more classes if we need as we go. Example shown below:
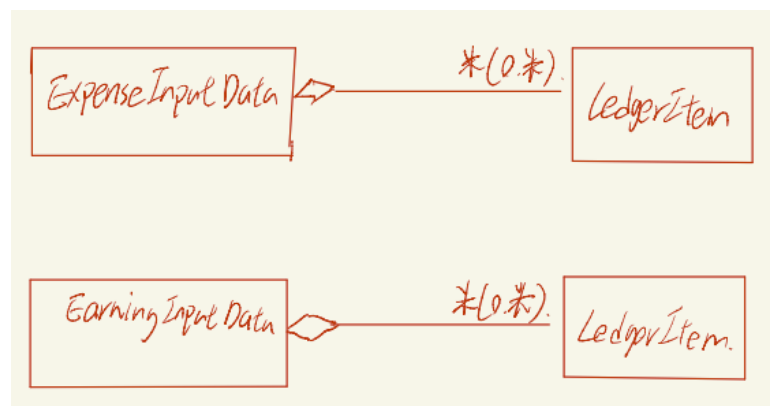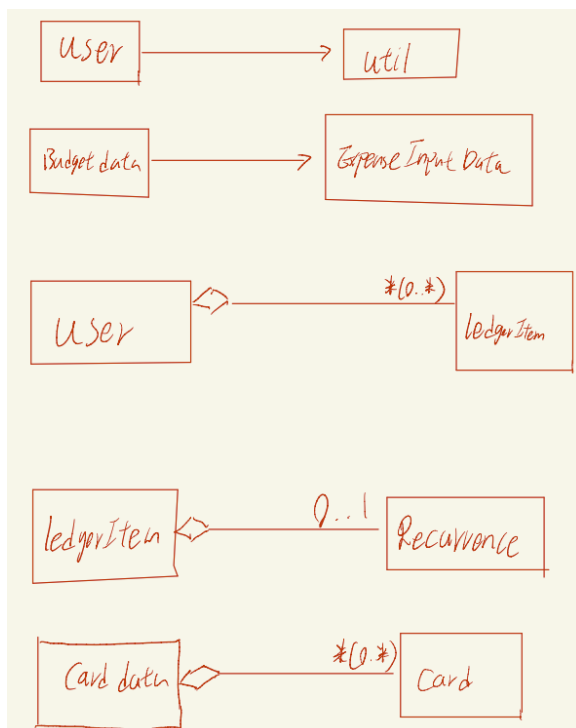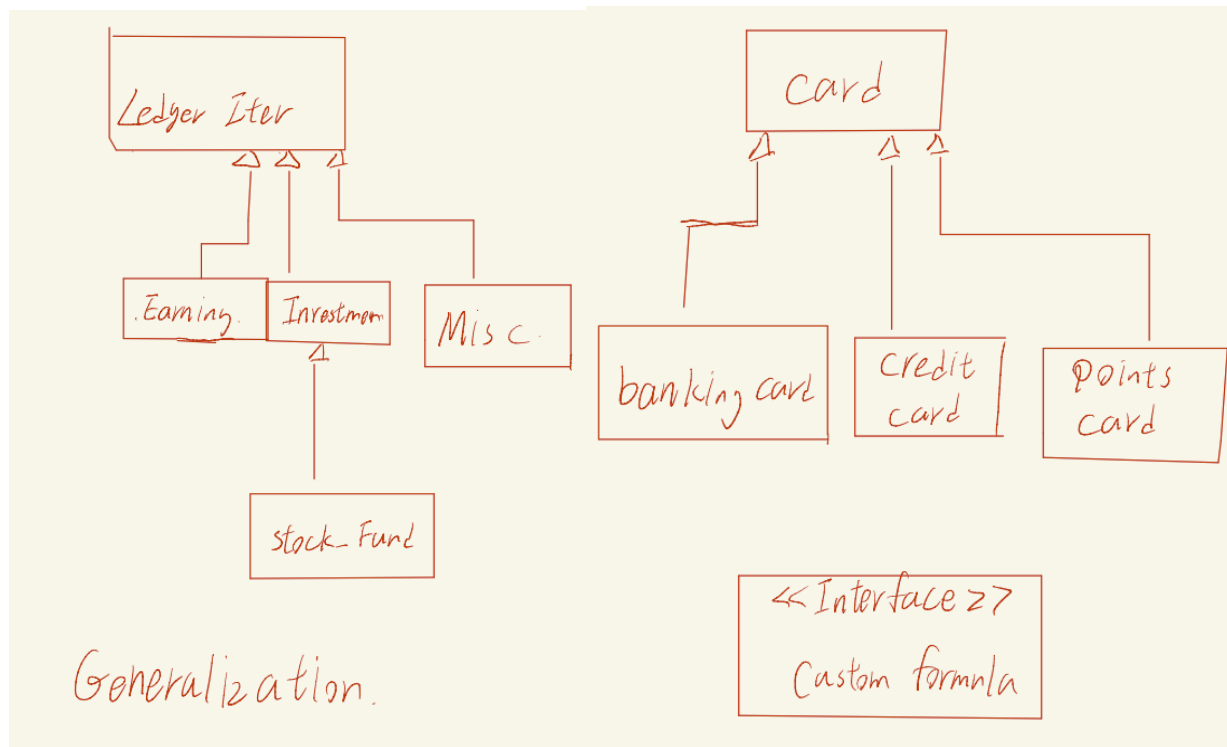
Advantages of this tool:

We can modify any changes anytime, and all group members have access to the same and live webpage.

We can easily split workload and show/change the status for each individual task.

Design (Outdated from itr2 please see new design in itr2)

This UML diagram below shows the relations between classes:

CRC Cards:

Class:Expense Page Form	
Responsibilities:	Collaborations:
Holds main window for the expense form	Java Swing
Knows name of the expense	Java AWT
Knows cost of expense	ExpensePage Class
Knows description of expense	ExpenseInputData Class
Knows date of expense payment	LedgerItem class
Passes this information to the expense page class	

Class:Expense Page	
Responsibilities:	Collaborations:
Gets data from expense page form	Java Swing
Display data for user	Java AWT
Uses ledger for data information holding	ExpensePageForm Class
Allows user to add expenses	LedgerItem class
Allows user to delete expenses	

Class:Investment	
Responsibilities:	Collaborations:
Know amount	Inherited from ledgerItem
Know date,	Extends to Stock_Fund
Know item name	Interactive with
Know note	ExpenseInputData class and
Know Recurrence	EarningInputData class
Know rate	Composite of recurrence class
Know interest	
Can cashOut	

Class:Stock_fund class	
Responsibilities:	Collaborations:
Know amount	Inherit from Investment class
Know date	ExpenseInputData class and
Know item name	EarningInputData class
Know note	Earning class
Know current_amount	Composite of recurrence class
Can calculate difference	
Can cashout	
Can Insert current value	

Class:Earning	
Responsibilities:	Collaborations:
Know date	Extends from ledgerItem class
Know amount	Composite of Recurrence
Know event	Used by stock_fund
Know note	Aggregated to
Know recurrence	EarningInputData class
Can compute sumincome	
Can computed deducted income	

Class:EarningInputData	
Responsibilities:	Collaborations:
Know list of earning	Interactive with Investment and stock_fund
Know total amount	Aggregative of earning
Can add income to list	
Can add info convert to income then add to list	

Class: CustomFormula interface	
Responsibilities:	Collaborations:
Can be implemented to calculate interest.	

Class:Misc	
Responsibilities:	Collaborations:
Know name	Inherited from LedgerItem
Know amount	Composite of recurrence
Know note	
Know recurrence	
can set amount and name	

Class: LedgerItem	
Responsibilities:	Collaborations:
Knows date	Income class
Knows amount	DateParser class
Knows item name	Investment class
Knows ref number	User class
Knows recurring status	
Can set and get all attributes	

Class: Recurrence	
Responsibilities:	Collaborations:
Knows start date	LedgerItem and its subclasses
Knows frequency	
Knows end date (optional)	
Can find all recuring ledger item objects in a list	
Can set and get all attributes	

Class: User	
Responsibilities:	Collaborations:
Knows user first name	LedgerItem
Knows user last name	
Knows username for login	
Knows password for login	
Knows the ledger items this user has	
Can set and get all the attributes	
Can encrypt the password	

Class: BudgetData	
Responsibilities:	Collaborations:
Knows budget	ExpenseInputData
Can find the amount left between budget and cost of expenses	

Class: DateParser	
Responsibilities:	Collaborations:
Verifies if a String is a valid date	LedgerItem
Returns a date object from a String containing a date	

Class: Util	
Responsibilities:	Collaborations:
Can calculate number of years, month, biweeks, weeks between 2 days	None
Can encrypt string	

//Stub Data Base

Class: ExpenseInputData	
Responsibilities:	Collaborations:
Knows purchases the user makes	Stores ledgerItems class as expenses
Knows the total amount the user has spent	
Can add an expense	
Can generate a receipt	

Class: BillData	
Responsibilities:	Collaborations:
Stores user input as Bill Data	Uses expenseInputData as a means of storage

Class: CardData

Responsibilities:	Collaborations:
Knows all the cards the inputs	Card class
Knows how much money is stored in all the cards	BankingCard class
	PointsCard class
	CreditCard class
Can add credit cards to the list	
Can add banking cards/debit cards to the list	
Can add points cards to the list	
Get a specific card in the list	

Class: Card

Responsibilities:	Collaborations:
Knows the name of the card	
Knows the total amount stored within the card	
Knows a note provided by the user about the card	
Can remove and add any amount of money to the card	

Class: BankingCard

Responsibilities:	Collaborations:
Knows the name of the card	
Knows the total amount stored within the card	
Knows a note provided by the user about the card	
Knows a user spending Limit	
Can remove and add any amount of money to the card	

Class: CreditCard

Responsibilities:	Collaborations:
Knows the name of the card	
Knows the total amount stored within the card	
Knows a note provided by the user about the card	
Knows a user spending Limit	
Knows the payment date	
Knows the amount of interest on the card for going over the date	
Can remove and add any amount of money to the card	

Class: PointsCard

Responsibilities:	Collaborations:
Knows the name of the card	
Knows the total amount stored within the card	
Knows a note provided by the user about the card	
Knows the ratio between points to actual money	
Can remove and add any amount of points to the card	

Major Duties of Each Member (itr1):

Mustafa Syed:

- Design and implementation of Expense related classes.
- Connections between Expense classes and GUI.
- Finalize GitHub Wiki.

Muhammad Shahid:

- Design and implementation of Main(program starter).
- Design and implementation of GUI for big user story 1 (Track expenses).
- Connections between Expense classes and GUI.

Zhenxu Wang:

- Design and implementation of some business logic classes(Util, Recurrence).
- Design and implementation of the super classes LedgerItem and User.
- Finalize the planning document and user manual.

Jinsong Xie:

- Design and implementation of Earning related classes.
- Finalize the UML diagram.
- Finalize GitHub Wiki.

Brahmjot Grewal:

- Design and implementation of Budget related classes.
- Design and implementation of some business logic classes(DateParser).
- Finalize the processing document.

Each group member is responsible for:

- Design and implement test cases upon completion of implementation.
- Make CRC cards upon completion of implementation.
- Coordinate with other members if there is a relationship between classes.
- Keep a personal development log.
- Implement exceptions, and design where and how to handle them.
- Implement related stub database if needed.

Iteration 1: Save and Spend management (No change from iteration 0)

Big user story 1:

Track Expenses

As a financially independent adult, I want to be able to track my expenses.

Priority: High   Cost: 9 days

Detailed User Stories:

Enter Expenses

Enter and edit general expense with detailed information (amount, date, notes).

Priority: High   Cost: 3 days

Bill Planner

Manage bills (One-time and recurring), keep track of billing date, due date and if it is paid or not.

Priority: High   Cost: 4 days

Card Purse

Editable "purse" for storing credit cards, debit cards, cash and gift card info. And keep track of their balances.

Priority: High   Cost: 2 days

Big user story 2:

Track Earnings

As a financially independent adult, I want to be able to track all kinds of my earnings.

Priority: High   Cost: 11 days

Detailed User Stories:

Enter Income

Enter and edit income information. Such as amount, pay day.

Question: Is it an one-time set up for all future ledgers or manual entry each pay?
Answer: one-time setup.

Priority: High   Cost: 3 days

Track Investments

Manage investments and their earnings or losses for an user.

Priority: High   Cost: 4 days

Miscellaneous Tracker

Manage and track RESP, RRSP, TFSA, tax returns and other earnings.

Priority: High   Cost: 4 days

Big user story 3:

> **Budget Planner**
>
> As a financially independent adult, I want to be able to view and manage monthly and yearly budget plans.
>
> Priority: High   Cost: 6 days

Detailed User Stories:

> **Goals on savings**
>
> Set a months or yearly goal on how much money to save.
>
> Priority: High   Cost: 3 days

> **Enter Budget**
>
> Let user to enter or edit as a spending goal.
>
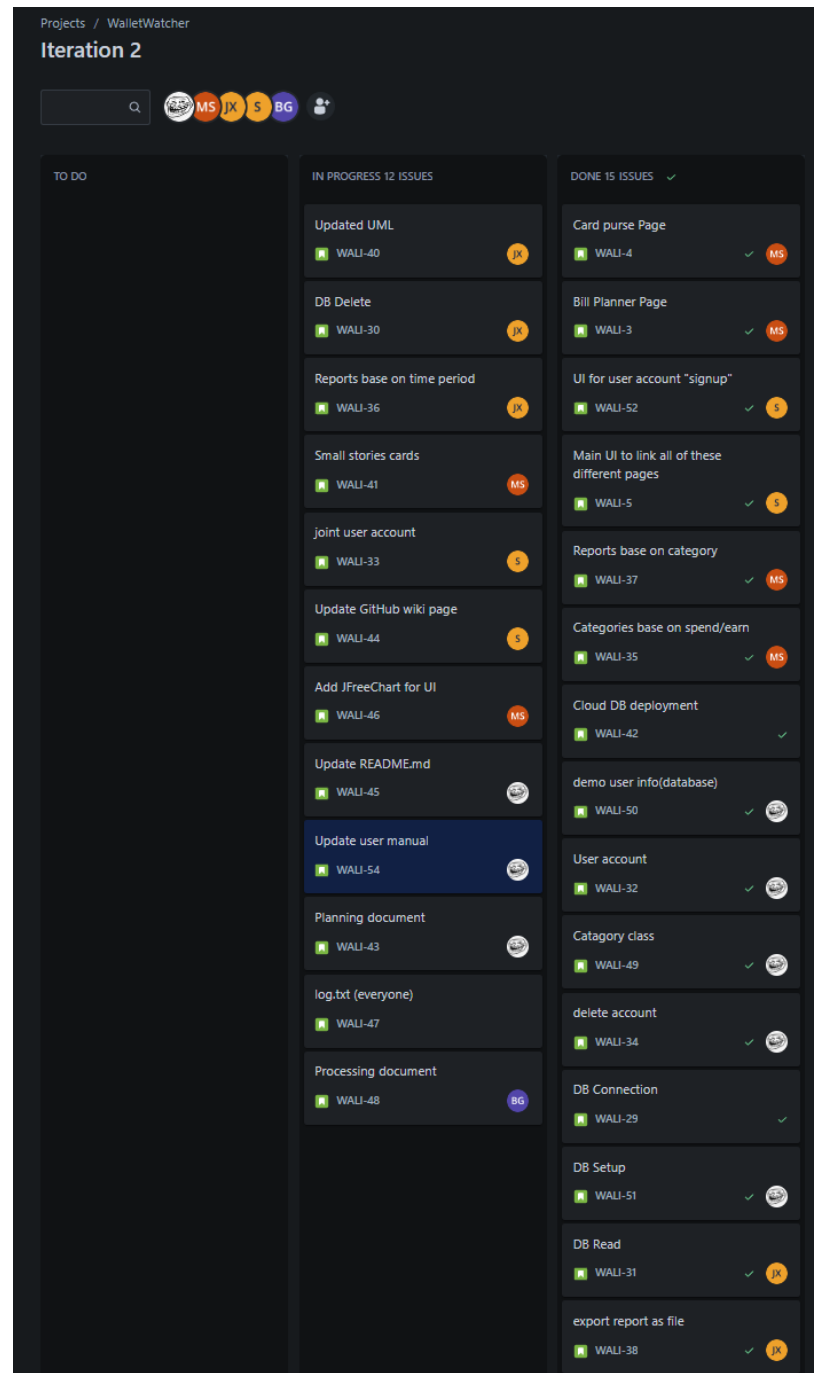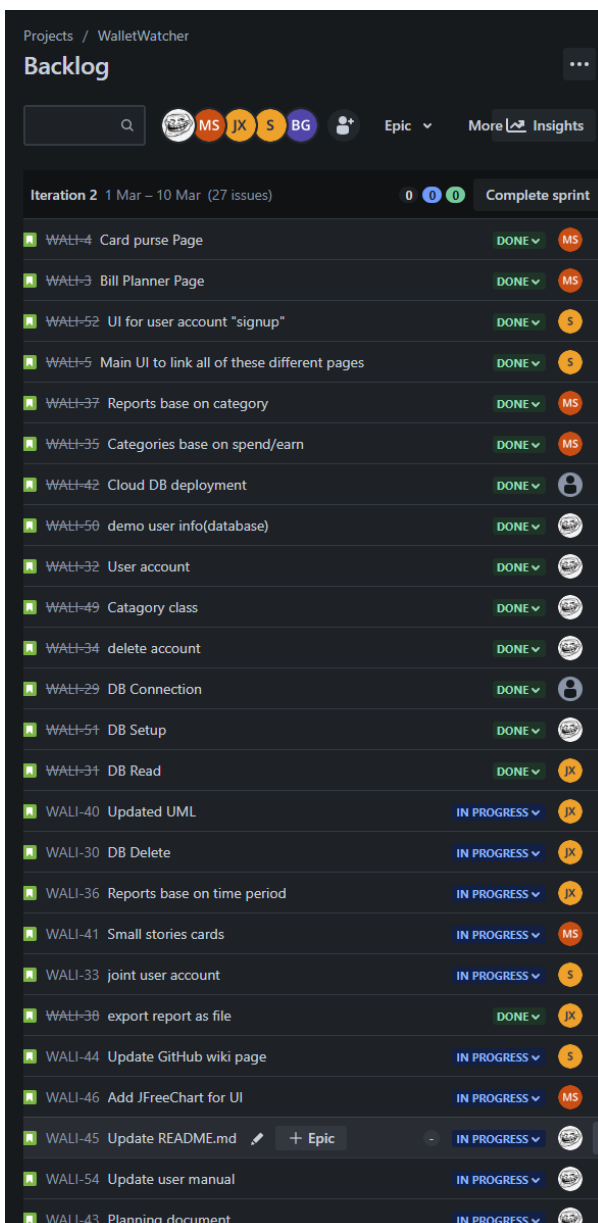> Priority: High   Cost: 3 days

Iteration 2

Personalized account(s) with customizations

March 17, 2023

General Planning:

We are using an online planning tool "*Jira Board*".

We first add all detailed user stories as individual classes to the backlog, we are stilling adding more classes if we need as we go. Example shown below:

General Planning(cont.):

Since we are bringing up the database in this iteration. We chose Microsoft Azure Cloud (Azure Database for MySQL).

We were thinking of database that uses localhost. But since this Wallet Watcher application is an information storing and analysis application. It needs pre-setup databases and tables, demo-user and its data, we decided to use a cloud service. Which is easier for both customers and developers.

(Example of the ledger items table):

ref	username	item	note	tag	amount	interest_rate	interest	recur	category	date_start
1	ceojeff	testitem 232	testnote 143	investment	230.52	0	0	0	default	2023-03-09
2	ceojeff	testitem 56	testnote 294	misc	185.99	0	0	0	default	2023-03-09
3	ceojeff	testitem 97	testnote 202	bill	62.86	0	0	0	default	2023-03-09
4	ceojeff	testitem 212	testnote 4	card	105.63	0	0	0	default	2023-03-09
5	ceojeff	testitem 294	testnote 29	card	278.99	0	0	0	default	2023-03-09
6	ceojeff	testitem 181	testnote 203	earning	62.58	0	0	0	default	2023-03-09
7	ceojeff	testitem 298	testnote 113	expense	61.32	0	0	0	default	2023-03-09
8	ceojeff	testitem 192	testnote 79	expense	45.22	0	0	0	default	2023-03-09
9	ceojeff	testitem 260	testnote 85	expense	19.62	0	0	0	default	2023-03-09
10	ceojeff	reset for testing	testnote 93	card	276.45	0	0	0	default	2023-03-09
11	ceojeff	testitem 296	testnote 222	card	175.79	0	0	0	default	2023-03-09
12	ceojeff	testitem 212	testnote 295	bill	20.75	0	0	0	default	2023-03-09
13	ceojeff	testitem 218	testnote 74	earning	130.4	0	0	0	default	2023-03-09
14	ceojeff	testitem 128	testnote 55	bill	265.89	0	0	0	default	2023-03-09
15	ceojeff	testitem 187	testnote 286	investment	95.36	0	0	0	default	2023-03-09
16	ceojeff	testitem 25	testnote 0	stock	201.62	0	0	0	default	2023-03-09
17	ceojeff	testitem 136	testnote 163	expense	150.07	0	0	0	default	2023-03-09
18	ceojeff	testitem 261	testnote 279	misc	178.85	0	0	0	default	2023-03-09
19	ceojeff	testitem 78	testnote 166	stock	34.62	0	0	0	default	2023-03-09
20	ceojeff	testitem 253	testnote 165	misc	178.37	0	0	0	default	2023-03-09
21	ceojeff	testitem 162	testnote 215	misc	103.9	0	0	0	default	2023-03-09
22	ceojeff	testitem 3	testnote 65	earning	9.49	0	0	0	default	2023-03-09
23	ceojeff	testitem 220	testnote 77	misc	44.27	0	0	0	default	2023-03-09
24	ceojeff	testitem 53	testnote 90	stock	68.1	0	0	0	default	2023-03-09

(Example of the users' info table, passwords are encrypted):

ref	username	hashcode	salt	firstname	lastname	acctype
1	ceojeff	-1157882147	G□��QK�	Jeff	Bezos	personal
2	testpw	364765850	Y��U$□	test	changepw	personal
9	adamkozsil	278480695	�16����	adam	kozsil	personal
10	wow	499895054	�/��□�-	wow	wow	personal
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Revised 3-layer Design (architecture of the application)
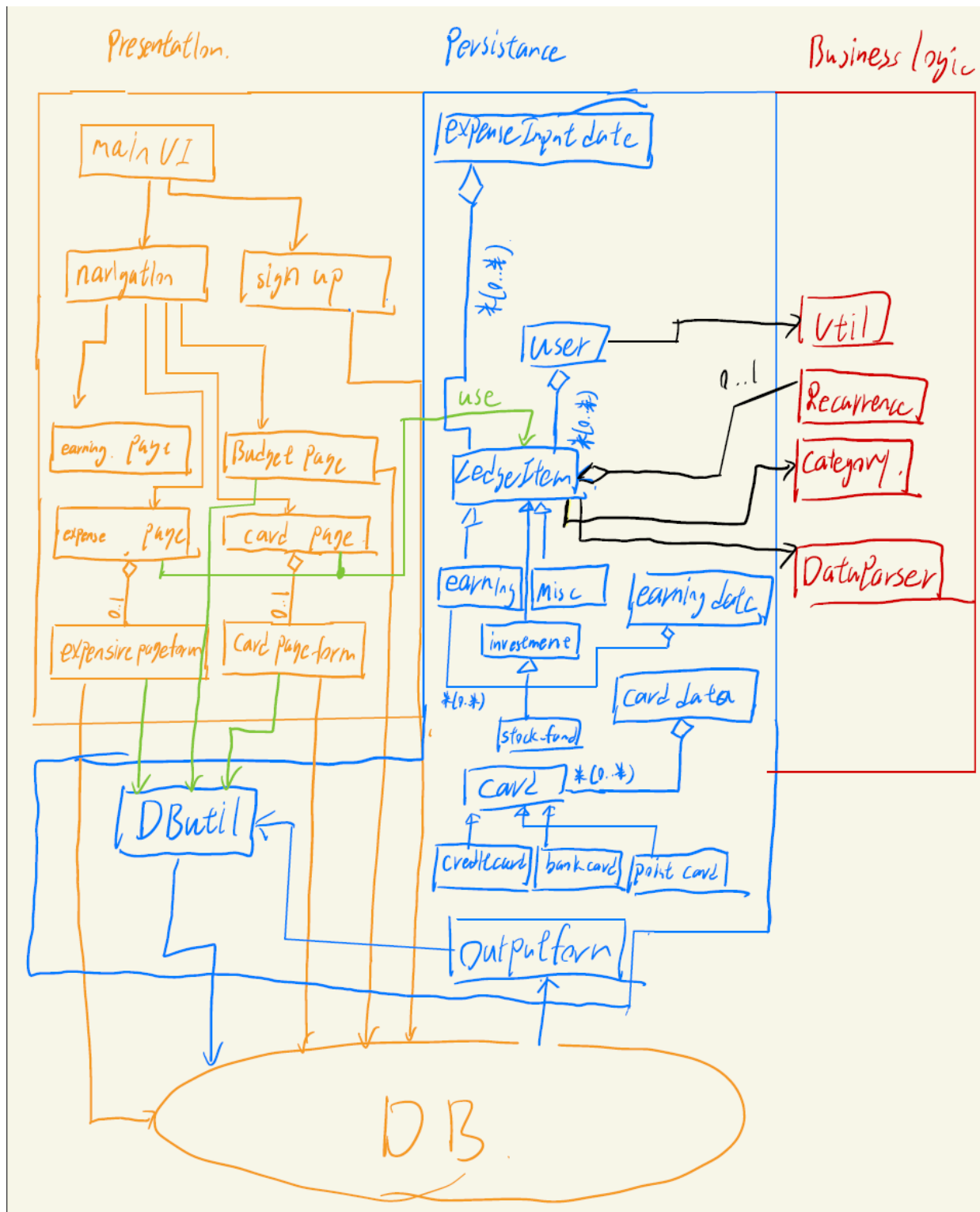
Major Duties of Each Member (itr1):

Mustafa Syed:

- Design and implementation of GUI for itr1 and itr2 features.
- Design and implementation of Joint account feature and Navigation page.
- Help with database utility methods.
- Finalize GitHub Wiki.

Muhammad Shahid:

- Design and implementation of UI for itr1 and itr2 features.
- UX and connections between GUI pages design.
- Finalize small user stories of itr2.

Zhenxu Wang:

- Design and set up databases.
- Design and implementation of database utility methods.
- Finalize the planning document and user manual.

Jinsong Xie:

- Design and implementation file export feature.
- Help with database utility methods.
- Finalize the 3-layer UML diagram.

Brahmjot Grewal:

- Gradle conversion, GitHub workflow set up.
- Design the architecture of database.
- Finalize the processing document.

Each group member is responsible for:

- Design and implement test cases upon completion of implementation.
- Coordinate with other members if there is a relationship between classes.
- Keep a personal development log.
- Implement exceptions, and design where and how to handle them.

Iteration 2: Personalized account(s) with customizations

Big user story 4:

Account Management

As a financially independent adult,
I want myself and other member in
my household to create our own permal
accounts.

Actual cost: 11 days

Priority: High   Cost: 13 days

Detailed User Stories:

Create Unique Accounts

Creating unique accounts where

username can not be used more than once,

and password set

Actual cost: 4 days

Priority: high   Cost: 4 days

Joint Accounts

I want to be able to create joint accounts

where my account joins with another, and

we have all our expenses on one account

Actual cost: 5 days

Priority: High   Cost: 7 days

Delete

Allow user to remove account, and

delete all stored user information

Actual cost: 2 days

Priority: High   Cost: 2 days

Big user story 5:

Customization Options

As a financially independent adult,
I want all users to be able to
Categorize our savings and earnings

Actual cost: 8 days

Priority: Medium   Cost: 9 days

Detailed User Stories:

Categorize based on earnings

Display user categories based on earnings
that were inputted by user

Actual cost: 4 days

Priority : medium   Cost: 4 days

Categorize based on spending

Display user's ledger items based on
categories that it was spent on

Acutal cost: 4 days

Priority: High   Cost: 5 days

Big user story 6:

Creat Reports

As a financially independent adult, I want all users to be able to create their own desired spendings or earning reports.

Actual cost: 8 days

Priority: Medium   Cost: 10 days

Detailed User Stories:

Time Period

Display user spendings based on time period it was spent in

Actual cost: 3 days

Priority: medium   Cost: 4 days

Category

Display user spending based on category that was spending the most

Actual cost: 2 days

Priority: medium   Cost: 2 days

export as file

User is given option to export monthly data as a file

Actual cost: 3 days

Priority: High   Cost: 2 days

Iteration 3: Projection and analysis

Big user story 7:

Detailed User Stories:

(To be added)

Big user story 8:

Future Projection

As a financially independent adult, I want to be able to see how much money I am gaining or losing.

Priority: Low   Cost: 12 days

Detailed User Stories:

(To be added)

Big user story 9:

Suggestions

As a financially independent adult, I want this app to be able to notify me if I am spending too much money.

Priority: Medium   Cost: 12 days

Detailed User Stories:
(To be added)

Challenges/Problems We Have Overcame:

How to minimize code redundancy?

We had too many classes made for the big story Earnings. It seems all of them had a good reason why they should be stand-alone classes. For example, under Investment class, we used to have some sub-classes like GIC, Saving account, RRSP.

But soon, we noticed that the thinking process we made was based on a customer/user. If we change our perspective to a programmer, we can merge those classes base on their functionalities. For example, GIC and Saving account are similar. They both need a start date, amount, information of recurrency and maybe an end date. Thus, we can just merge them back into their superclass Investment. In addition, we just require user to declare what type of the investment is (by storing a string value). Then we generalized this idea and applied for all the designs for this project.

Having problems pushing, merging or rebasing branches on GitHub.

When we first started working on the project, we decided to branch off from the main branch. But we branched off for each team member, which each team member has an individual branch. This gave us a lot of problems if there is a conflict.

The solution is, we will just keep what branches we have so far for itr1. And from itr2, we should branch off based on the feature of the implementation. And start from itr2, we will start using GitHub pull request feature.

Project build, project configuration files and packaging.

We are still having issues with .classpath and .project file, because Eclipse build-in default Java project build may not be recognized by other IDE. And we still cannot find a proper solution to fix these files to make the project works perfectly on everyone's computer.

We ended up converting this existing Eclipse Java project to a Maven project.

Database setup:

We had problems using local servers. Since each developer has different login names, passwords, even database names. If we keep using the local servers, it's will be time consuming to resolve the git conflicts caused by database connection strings. And each time we need to run test cases on a new device or server, we need to re-initialize the database and some pre-existed tables.

Solution we found is to use a cloud service.

EECS 2311 Project Peer Evaluation

Please submit this form on <u>eClass</u> (doc or pdf) Together with your submission

Each team member is supposed to put in 100% effort in developing the class project, in each iteration. If you think that in your team there are individuals who did not put as expected. You can mention it in this form. You evaluate everybody in your team by giving them a mark out of 100. This may affect their individual marks, as explained in class. Note that:

1. 100 means satisfactory. You don't give anybody more than 100.
2. Lower than 100 marks must have an explanation, which typically is a fact, like #commits, late commits, not completing the assigned tasks, not attending meetings, etc. that I can verify in your logs/github/etc.

Group Number: 7		
Member's Name	Mark	Explanation (only if mark < 100)
Mustafa Syed	100	
Muhammad Shahid	100	
Zhenxu Wang	100	
Jinsong Xie	100	
Brahmjot Grewal	100	

Your name (printed): MUSTAFA SYED, MUHAMMAD SHAHID, ZHENXU WANG, JISONG XIE, BRAHMJOT GREWAL

Signature:

Brahmjot Grewal   Jinsong Xie   Mustafa Syed   Zhenxu Wang   Muhammad Shahid