Wallet Watcher

EECS 2311Z 23W

By Team No.7:

Mustafa Syed

Muhammad Shahid

Zhenxu Wang

Jinsong Xie

Brahmjot Grewal

Table of Contents

Vision Statement **3**

Iteration 1 ..... **4**

General Planning 4

Design (architecture of the application) 5

CRC Cards 6

Major Duties of Each Member 9

Big user story 1 10

Big user story 2 11

Big user story 3 12

Iteration 2 ..... **13**

Big user story 4 13

Big user story 5 14

Big user story 6 15

Iteration 3 ..... **16**

Big user story 7 16

Big user story 8 17

Big user story 9 18

Challenges/Problems We Have Overcame **19**

Peer Evaluation Forms **20**

Peer evaluation form (itr 1) 20

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.

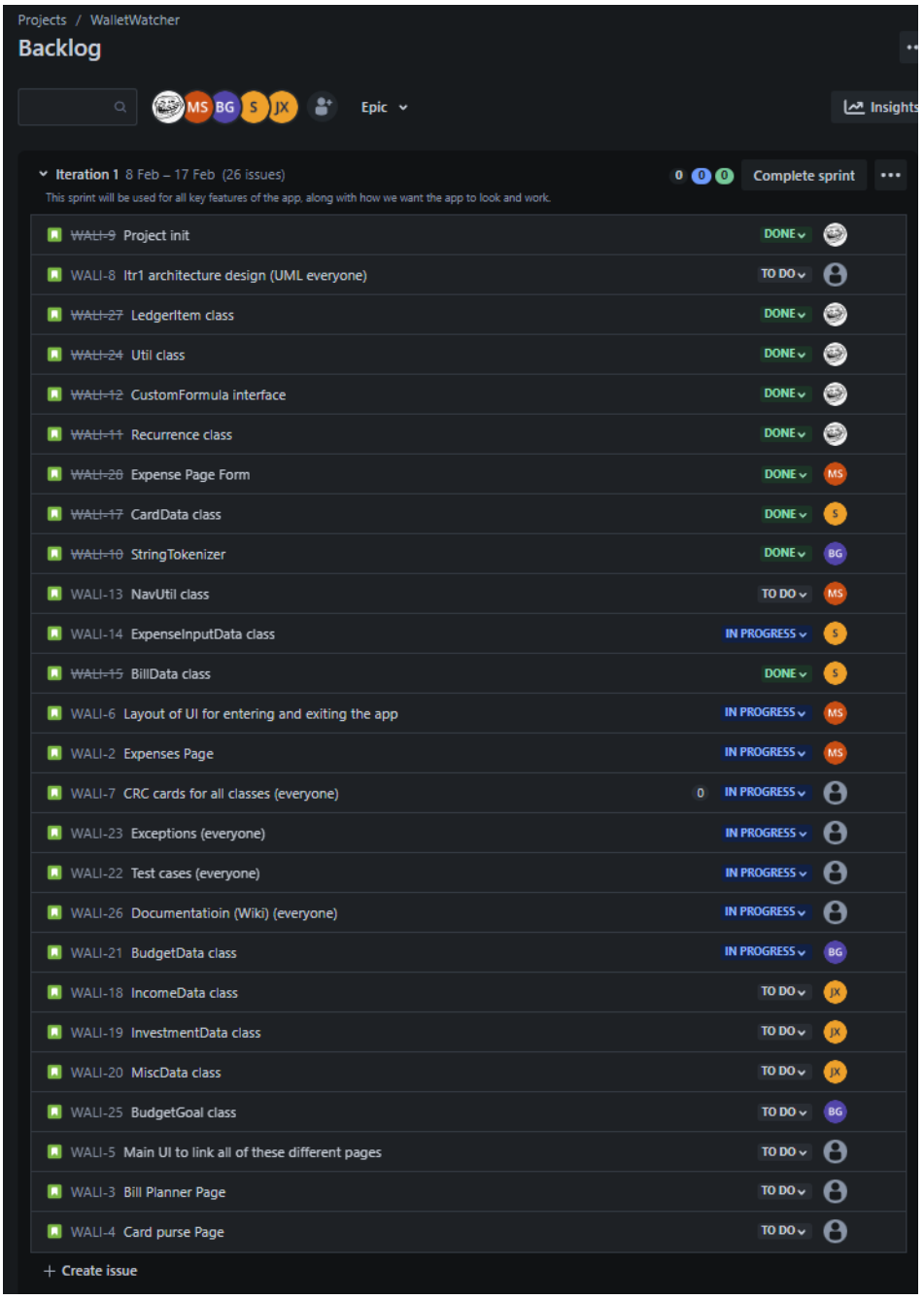
Iteration 1

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:



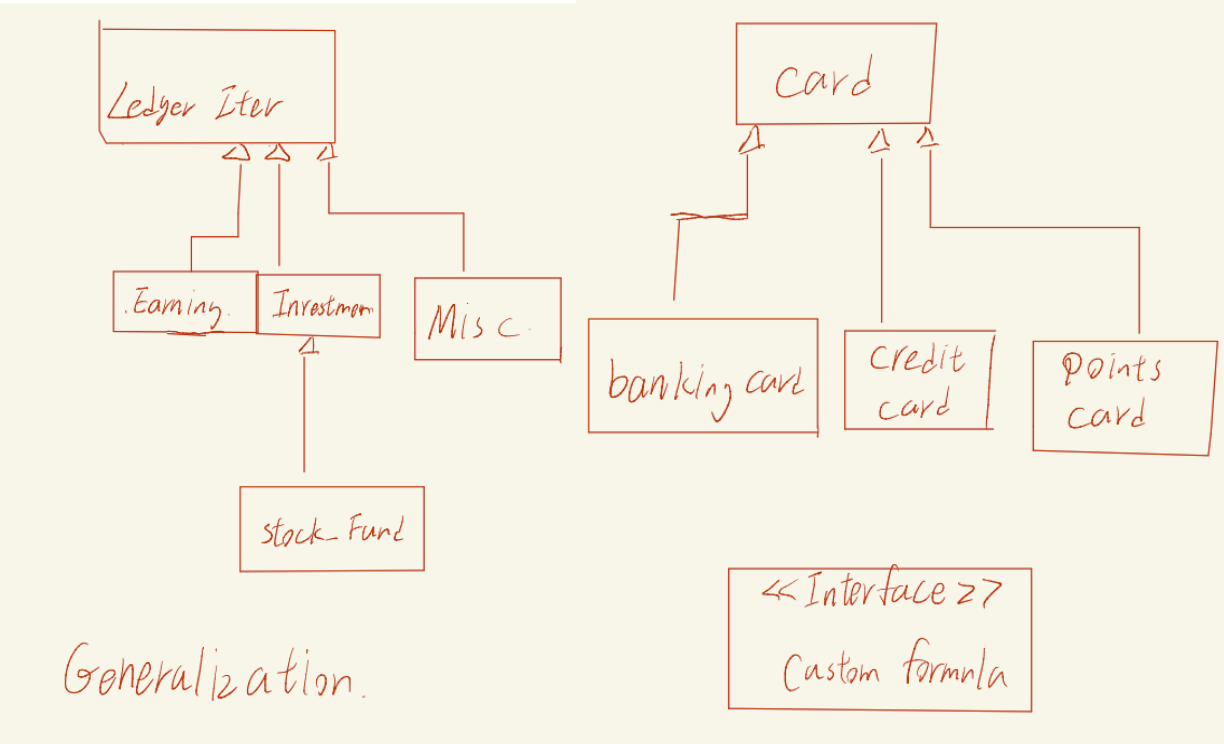
Advantages of this tool:

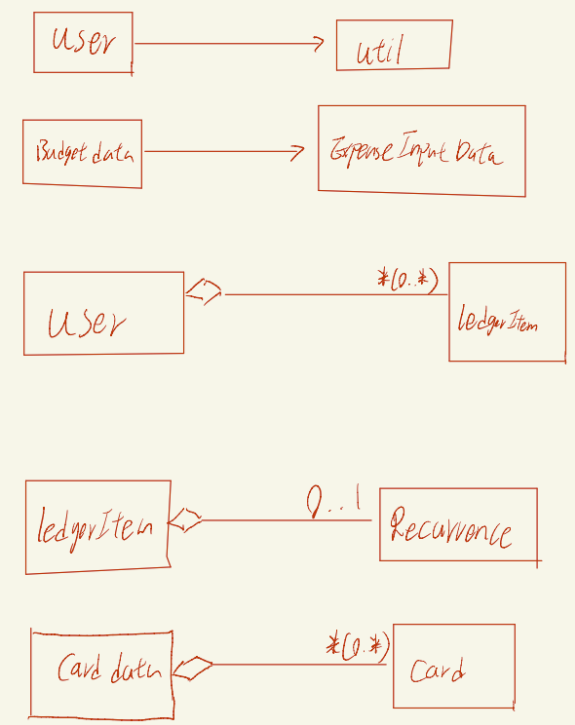
We can modify any changes anytime, and all group members have access to the same and live webpage.

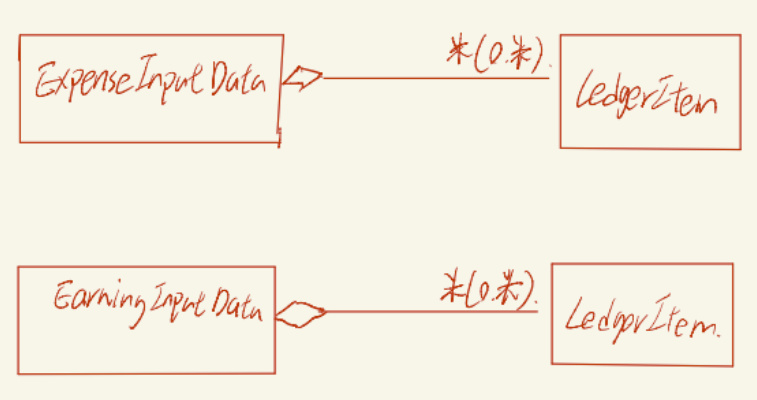
We can easily split workload and show/change the status for each individual task.

Design (architecture of the application)

This UML diagram below shows the relations between classes:







CRC Cards:

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

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

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

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

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

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

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

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

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

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

Class: User	
Responsibilities: Knows user first name 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	Collaborations: LedgerItem

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

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

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

//Stub Data Base

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

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

Class: CardData	
Responsibilities: Knows all the cards the inputs Knows how much money is stored in all the cards 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	Collaborations: Card class BankingCard class PointsCard class CreditCard class

Class: Card	
Responsibilities: 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	Collaborations:

Class: BankingCard	
Responsibilities: 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	Collaborations:

Class: CreditCard	
Responsibilities: 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	Collaborations:

Class: PointsCard	
Responsibilities: 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	Collaborations:

Major Duties of Each Member

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:

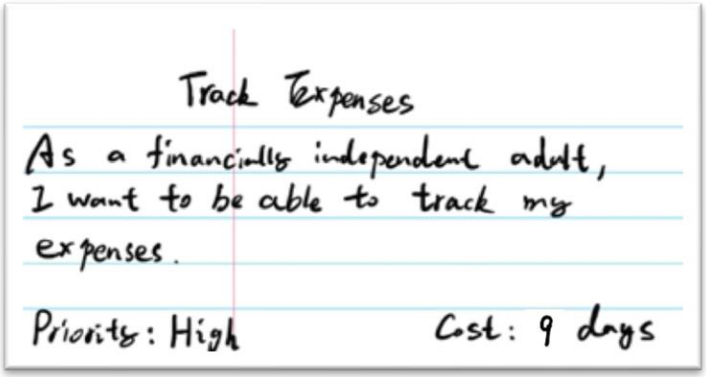
- Design and implementation of Budget related classes.
- Design and implementation of some business logic classes(DateParser).
- Finalize the processing document.

Each group member is responsible for:

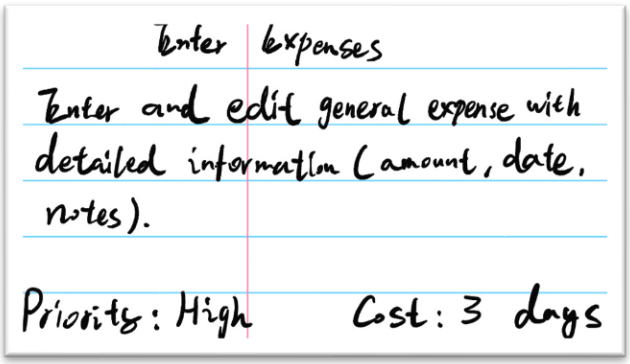
- Design and implement test cases upon completion of implementation.
- Make CRC cards upon completion of implementation.
- Coordinate with other members if there is a relationship between classes.
- Keep a personal development log.
- Implement exceptions, and design where and how to handle them.
- Implement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

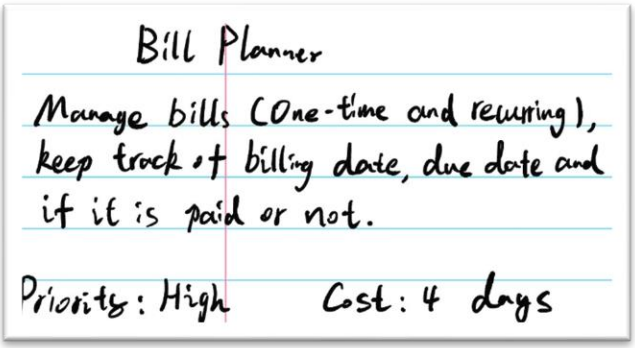
Iteration 1: Save and Spend management (No change from iteration 0)

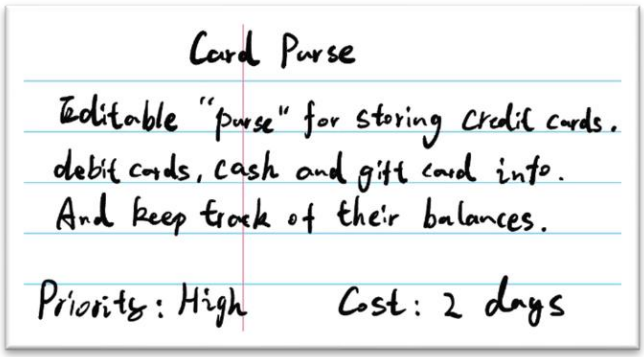
Big user story 1:



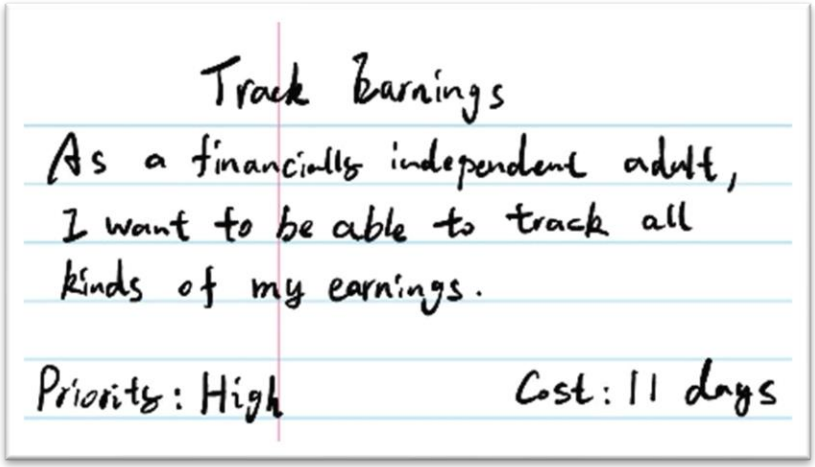
Detailed User Stories:



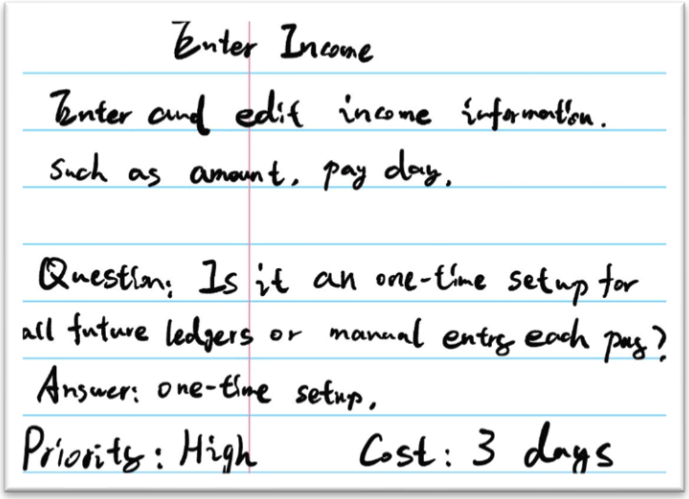


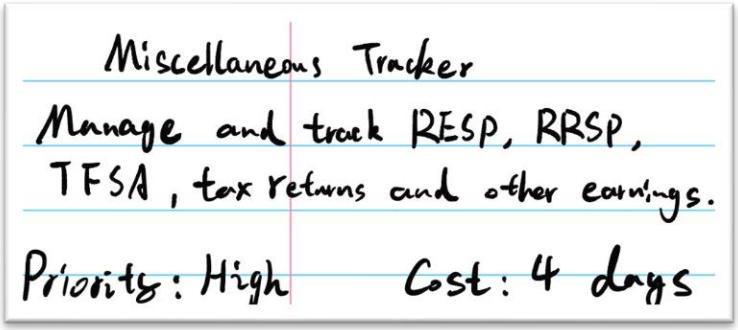


Big user story 2:



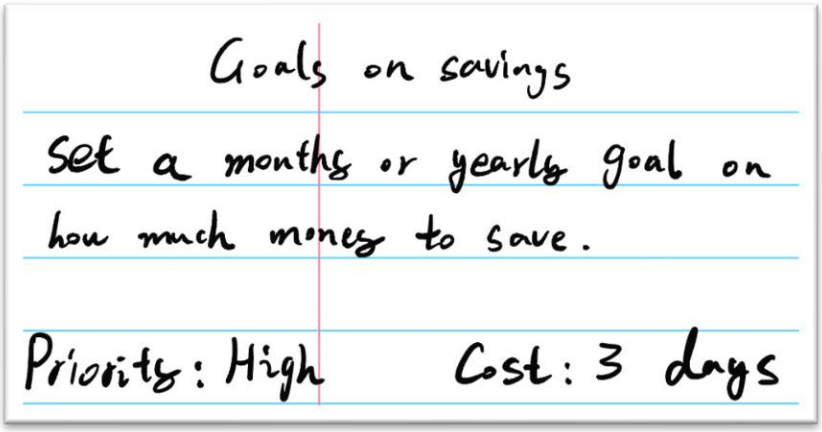
Detailed User Stories:

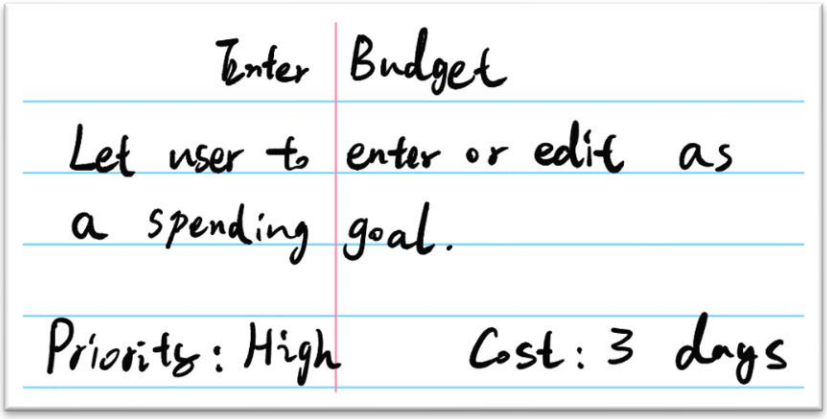




Big user story 3:

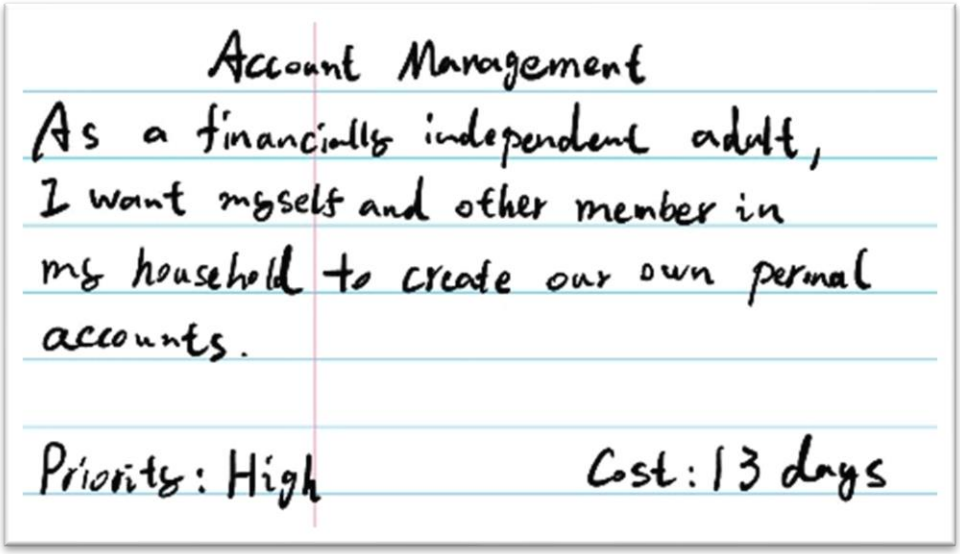
Detailed User Stories:





Iteration 2: Personalized account(s) with customizations

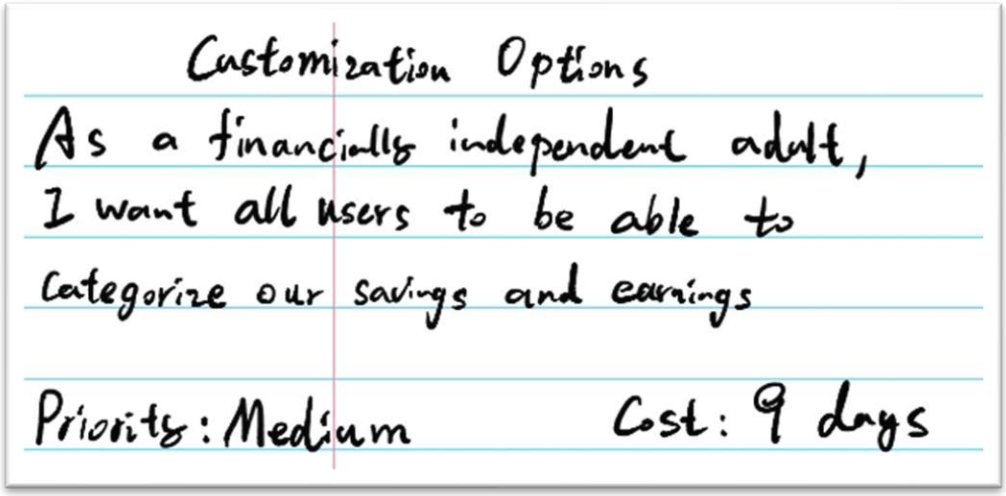
Big user story 4:



Detailed User Stories:

(To be added)

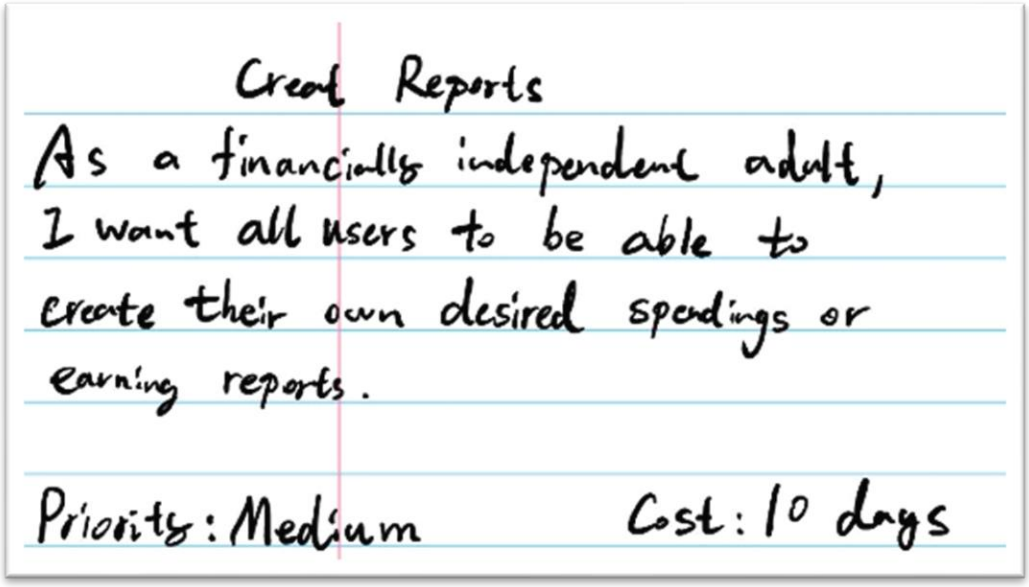
Big user story 5:



Detailed User Stories:

(To be added)

Big user story 6:

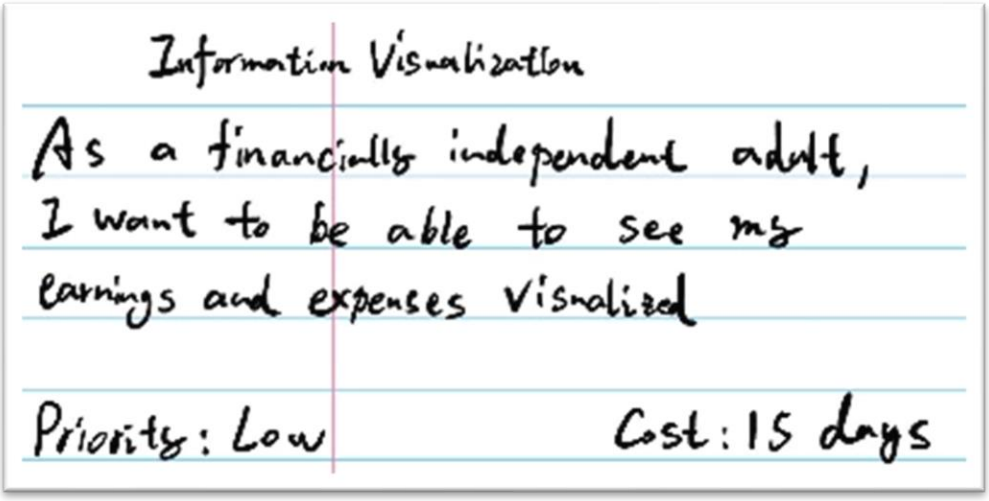


Detailed User Stories:

(To be added)

Iteration 3: Projection and analysis

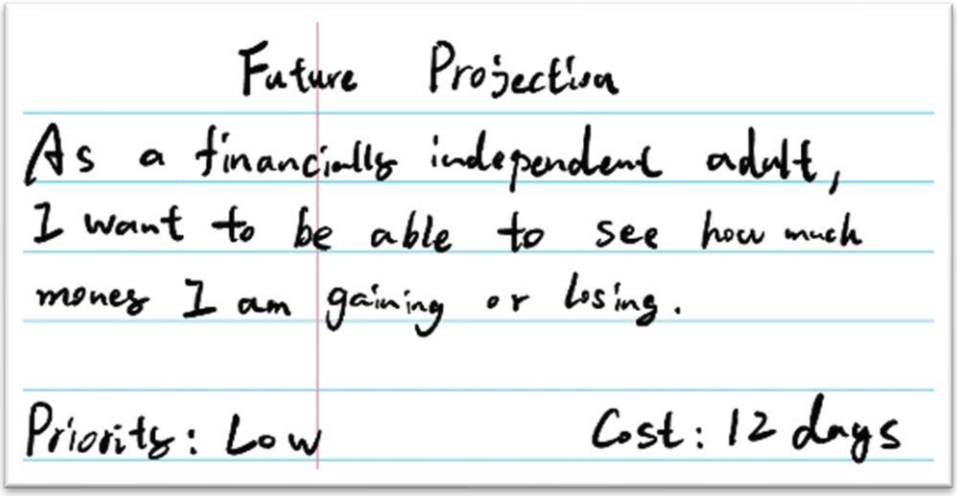
Big user story 7:



Detailed User Stories:

(To be added)

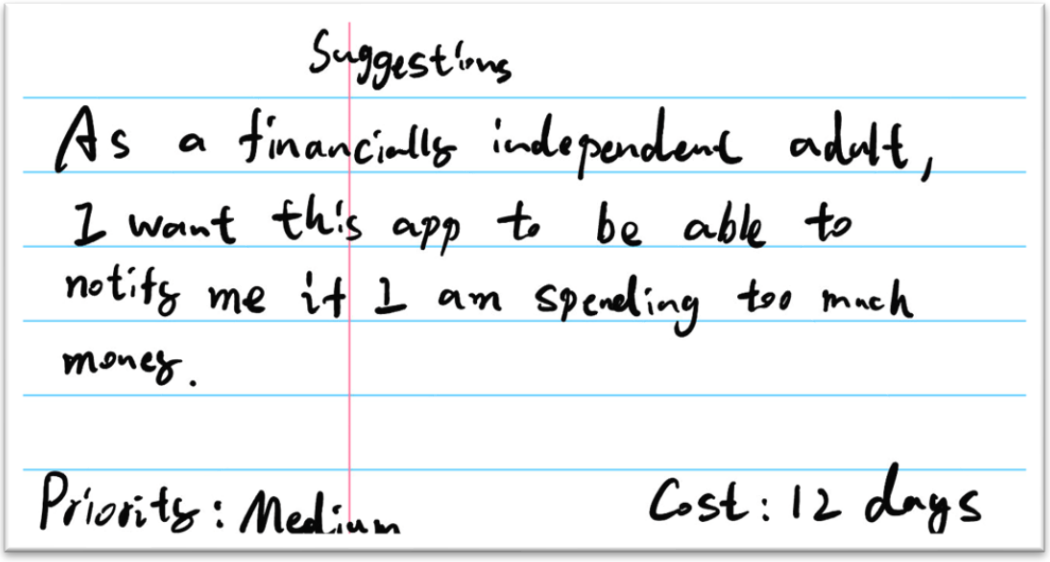
Big user story 8:



Detailed User Stories:

(To be added)

Big user story 9:



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.

EECS 2311 Project Peer Evaluation

Please submit this form on eClass (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 Mustafa Syed Muhammad Shahid Jinsong Xie Zhenxu Wang