Wallet Watcher

EECS 2311Z 23W

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.

The budget planner's main functions include a user being able to set the goal of how much they want to limit their spending for the given period. And if any expense occurs at that period exceeded the goal, the system will send the warning message to notify the user of overspending.

At any time using the application, the user could check analytic page of their financial health report of all expenses, earnings, and any budget goals they have reached.

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 we could conduct the survey on the user to determine an average taken of every single user's financial health performance which they could reach their saving goals easier through the usage of the application.

Iteration 1

Feb 17, 2023

General Planning:

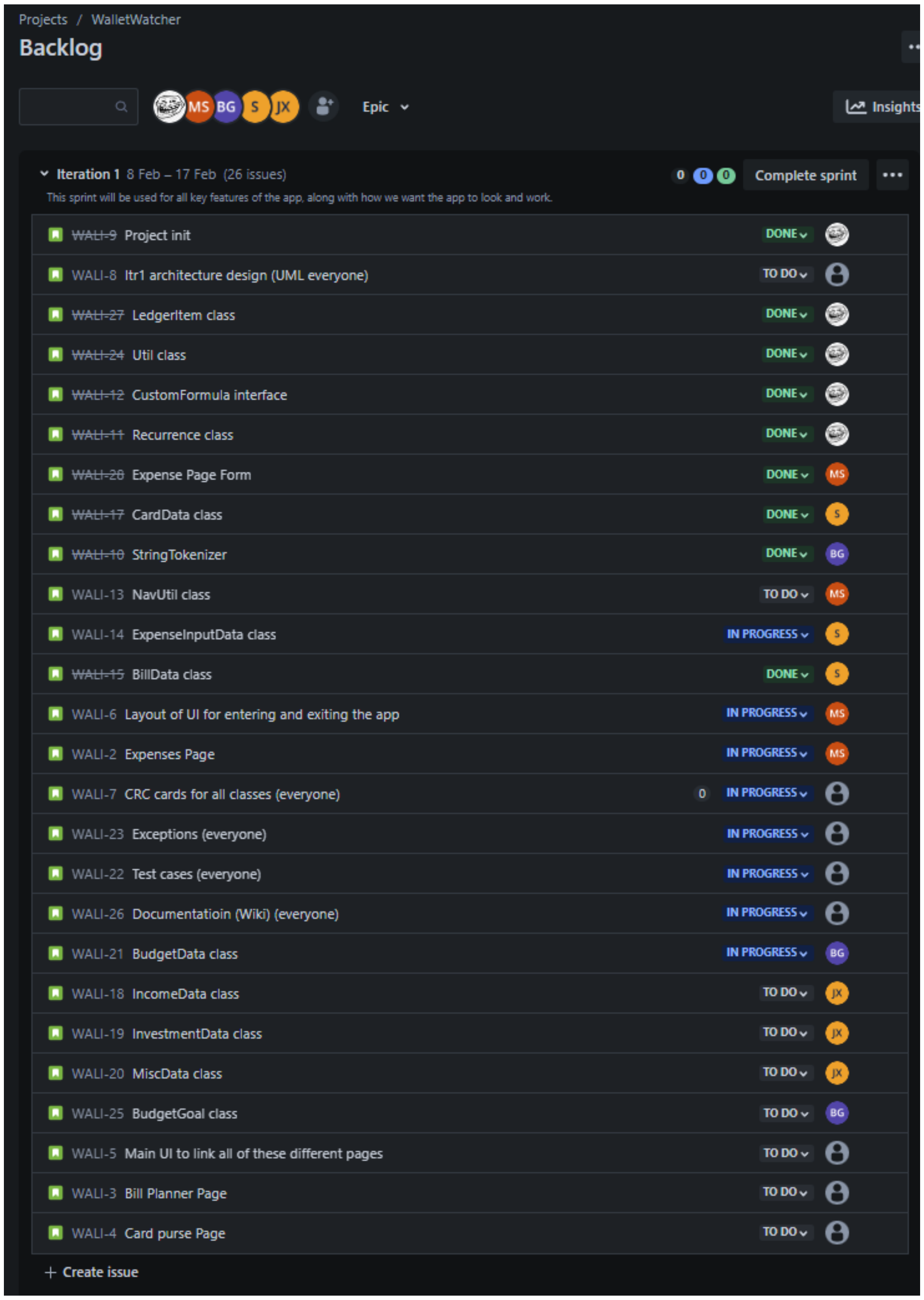
We are using an online planning tool "*Jira Board*".

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.


Projects / WalletWatcher

Backlog

Iteration 1 8 Feb – 17 Feb (26 issues) 0 0 0 Complete sprint

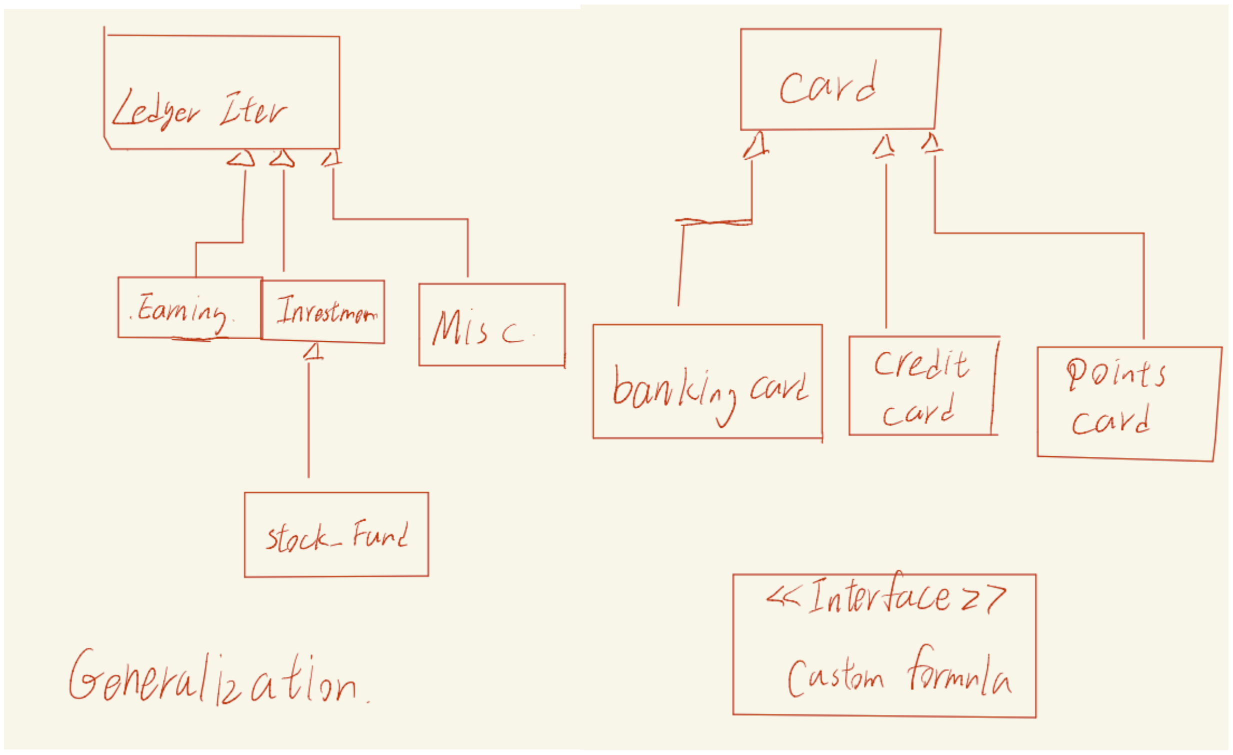
This sprint will be used for all key features of the app, along with how we want the app to look and work.

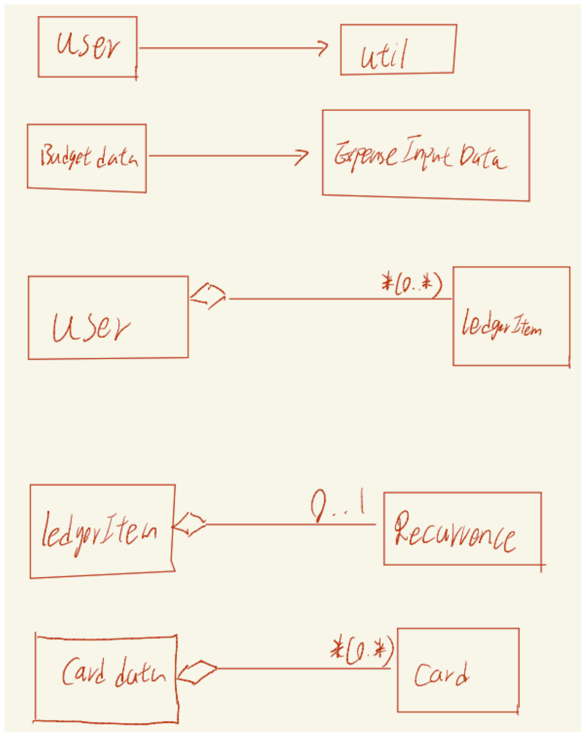
~~WALI-9~~	Project init	DONE	
WALI-8	Itr1 architecture design (UML everyone)	TO DO	
~~WALI-27~~	LedgerItem class	DONE	
~~WALI-24~~	Util class	DONE	
~~WALI-12~~	CustomFormula interface	DONE	
~~WALI-11~~	Recurrence class	DONE	
~~WALI-28~~	Expense Page Form	DONE	MS
~~WALI-17~~	CardData class	DONE	S
~~WALI-10~~	StringTokenizer	DONE	BG
WALI-13	NavUtil class	TO DO	MS
WALI-14	ExpenseInputData class	IN PROGRESS	S
~~WALI-15~~	BillData class	DONE	S
WALI-6	Layout of UI for entering and exiting the app	IN PROGRESS	MS
WALI-2	Expenses Page	IN PROGRESS	MS
WALI-7	CRC cards for all classes (everyone) 0	IN PROGRESS	
WALI-23	Exceptions (everyone)	IN PROGRESS	
WALI-22	Test cases (everyone)	IN PROGRESS	
WALI-26	Documentatioin (Wiki) (everyone)	IN PROGRESS	
WALI-21	BudgetData class	IN PROGRESS	BG
WALI-18	IncomeData class	TO DO	JX
WALI-19	InvestmentData class	TO DO	JX
WALI-20	MiscData class	TO DO	JX
WALI-25	BudgetGoal class	TO DO	BG
WALI-5	Main UI to link all of these different pages	TO DO	
WALI-3	Bill Planner Page	TO DO	
WALI-4	Card purse Page	TO DO	

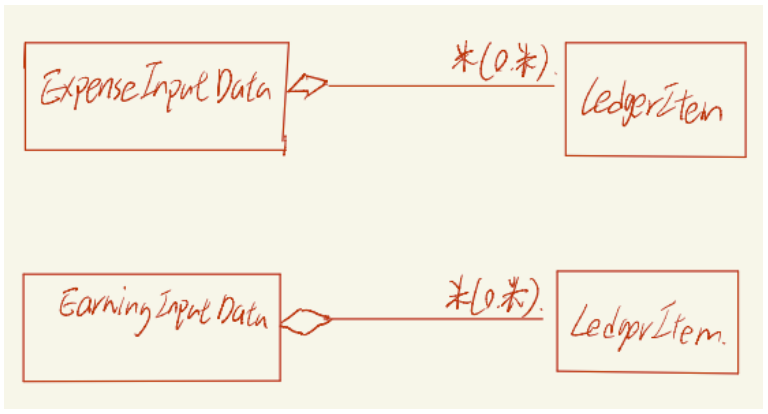
+ Create issue

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: 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	Collaborations: Java Swing Java AWT ExpensePage Class ExpenseInputData Class LedgerItem class

Class:Expense Page	
Responsibilities: 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	Collaborations: Java Swing Java AWT ExpensePageForm Class LedgerItem class

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

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

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

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

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

Class:Misc	
Responsibilities: Know name Know amount Know note Know recurrence can set amount and name	Collaborations: 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: Util	
Responsibilities: Can calculate number of years, month, biweeks, weeks between 2 days Can encrypt string	Collaborations: None

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

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

Big Story: *Track Expenses*

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

Cost: *9 days*

Priority: *High* **Actual Cost:** *9 days*

Detailed User Stories:

Detail Story: *Enter Expenses*

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

Cost: *3 days*

Priority: *High* **Actual Cost:** *3 days*

Detail Story: *Bill Planner*

Manage bills (one time or recurring), keep track of due date and if it is paid or unpaid.

Cost: *4 days*

Priority: *High* **Actual Cost:** *3 days*

Detail Story: *Card Purse*

Editable "purse" for storing credit cards, debit cards, gift card or points card information. And keep track of their balance.

Cost: *2 days*

Priority: *High* **Actual Cost:** *3 days*

Big user story 2:

Big Story: *Track Earnings*

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

Cost: *11 days*

Priority: *High* **Actual Cost:** *10 days*

Detailed User Stories:

Detail Story: *Enter Income*

Enter and edit income information. Such as amount, date.

Question by customer: Is it a one time setup for all future ledger or manual entry after each pay?

Answer: It is manual entry.

Cost: *3 days*

Priority: *High* **Actual Cost:** *3 days*

Detail Story: *Track investment*

Manage investments and their earnings or losses for a user.

Cost: *4 days*

Priority: *High* **Actual Cost:** *4 days*

Detail Story: *Miscellaneous Tracker*

Manage and track miscellaneous earnings. Such as tax return, or any other unexpected earnings

Cost: *4 days*

Priority: *High* **Actual Cost:** *3 days*

Big user story 3:

Big Story: *Budget Planner*

As a financially independent adult, I want to be able to view and manage monthly budget plans.

Cost: *6 days*

Priority: *High* **Actual Cost:** *6 days*

Detailed User Stories:

Detail Story: *Goals on Saving*

Set a monthly goal on how much money available to spend.

Cost: *3 days*

Priority: *High* **Actual Cost:** *3 days*

Detail Story: *Enter Budget*

Let user to enter or edit as a spending goal.

Cost: *3 days*

Priority: *High* **Actual Cost:** *3 days*

Iteration 2

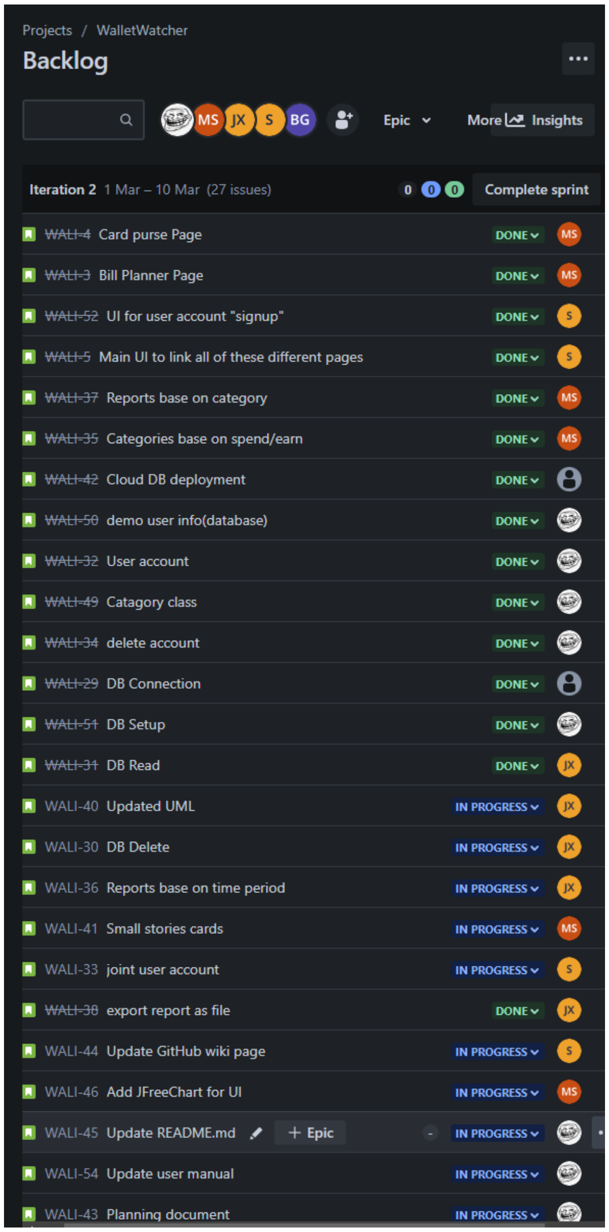
Personalized account(s) with customizations

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:



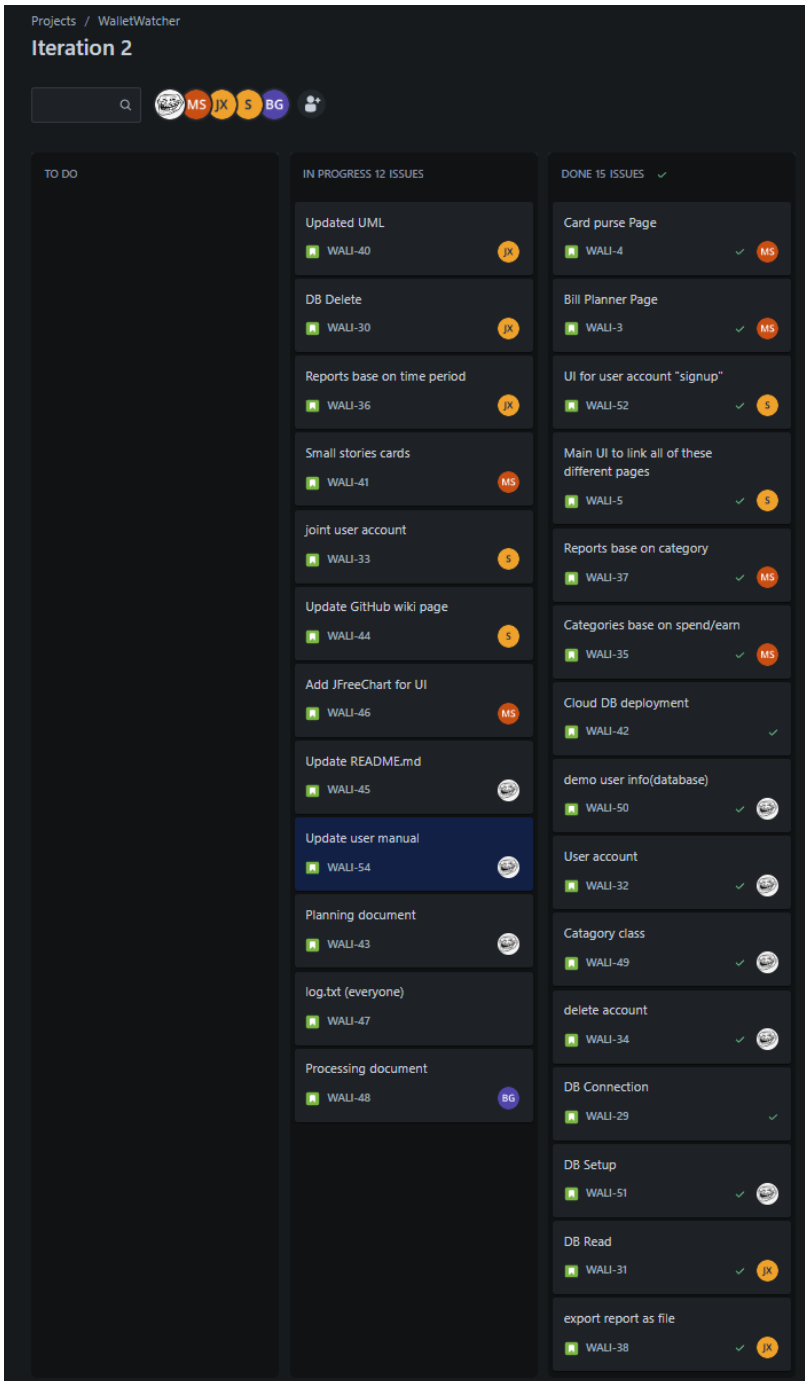
Projects / WalletWatcher

Backlog

MS JX S BG Epic More Insights

Iteration 2 1 Mar – 10 Mar (27 issues) 0 0 0 Complete sprint

~~WALI-4~~	Card purse Page	DONE	MS
~~WALI-3~~	Bill Planner Page	DONE	MS
~~WALI-52~~	UI for user account "signup"	DONE	S
~~WALI-5~~	Main UI to link all of these different pages	DONE	S
~~WALI-37~~	Reports base on category	DONE	MS
~~WALI-35~~	Categories base on spend/earn	DONE	MS
~~WALI-42~~	Cloud DB deployment	DONE	
~~WALI-50~~	demo user info(database)	DONE	
~~WALI-32~~	User account	DONE	
~~WALI-49~~	Catagory class	DONE	
~~WALI-34~~	delete account	DONE	
~~WALI-29~~	DB Connection	DONE	
~~WALI-51~~	DB Setup	DONE	
~~WALI-31~~	DB Read	DONE	JX
WALI-40	Updated UML	IN PROGRESS	JX
WALI-30	DB Delete	IN PROGRESS	JX
WALI-36	Reports base on time period	IN PROGRESS	JX
WALI-41	Small stories cards	IN PROGRESS	MS
WALI-33	joint user account	IN PROGRESS	S
~~WALI-38~~	export report as file	DONE	JX
WALI-44	Update GitHub wiki page	IN PROGRESS	S
WALI-46	Add JFreeChart for UI	IN PROGRESS	MS
WALI-45	Update README.md + Epic	IN PROGRESS	
WALI-54	Update user manual	IN PROGRESS	
WALI-43	Planning document	IN PROGRESS	



Projects / WalletWatcher

Iteration 2

MS JX S BG

TO DO

IN PROGRESS 12 ISSUES

- Updated UML — WALI-40 — JX
- DB Delete — WALI-30 — JX
- Reports base on time period — WALI-36 — JX
- Small stories cards — WALI-41 — MS
- joint user account — WALI-33 — S
- Update GitHub wiki page — WALI-44 — S
- Add JFreeChart for UI — WALI-46 — MS
- Update README.md — WALI-45
- Update user manual — WALI-54
- Planning document — WALI-43
- log.txt (everyone) — WALI-47
- Processing document — WALI-48 — BG

DONE 15 ISSUES ✓

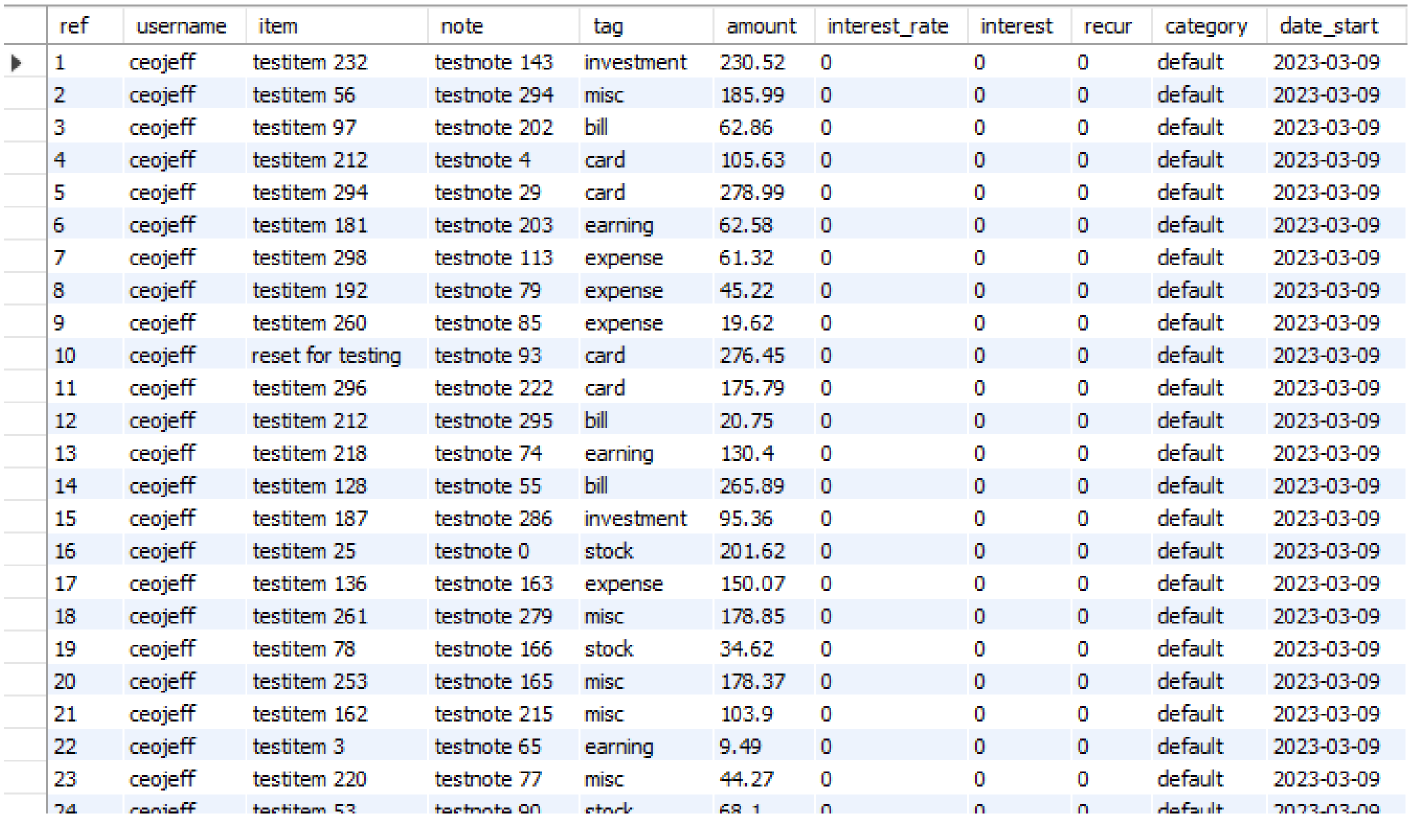
- Card purse Page — WALI-4 ✓ MS
- Bill Planner Page — WALI-3 ✓ MS
- UI for user account "signup" — WALI-52 ✓ S
- Main UI to link all of these different pages — WALI-5 ✓ S
- Reports base on category — WALI-37 ✓ MS
- Categories base on spend/earn — WALI-35 ✓ MS
- Cloud DB deployment — WALI-42 ✓
- demo user info(database) — WALI-50 ✓
- User account — WALI-32 ✓
- Catagory class — WALI-49 ✓
- delete account — WALI-34 ✓
- DB Connection — WALI-29 ✓
- DB Setup — WALI-51 ✓
- DB Read — WALI-31 ✓ JX
- export report as file — WALI-38 ✓ JX

General Planning(cont.):

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

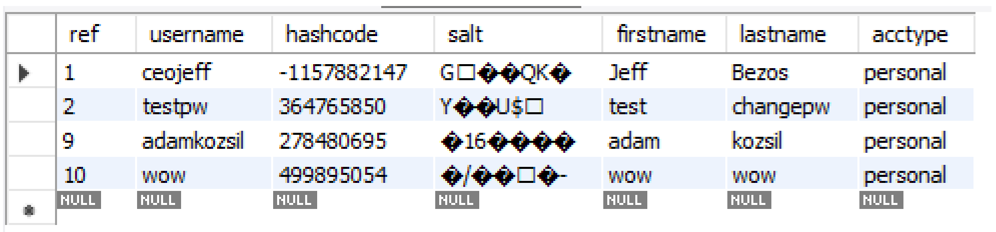
We were thinking of database that uses localhost. But since this Wallet Watcher application is an information storing and analysis application. It needs pre-setup databases and tables, demo-user and its data, we decided to use a cloud service. Which is easier for both customers and developers.

(Example of the ledger items table):



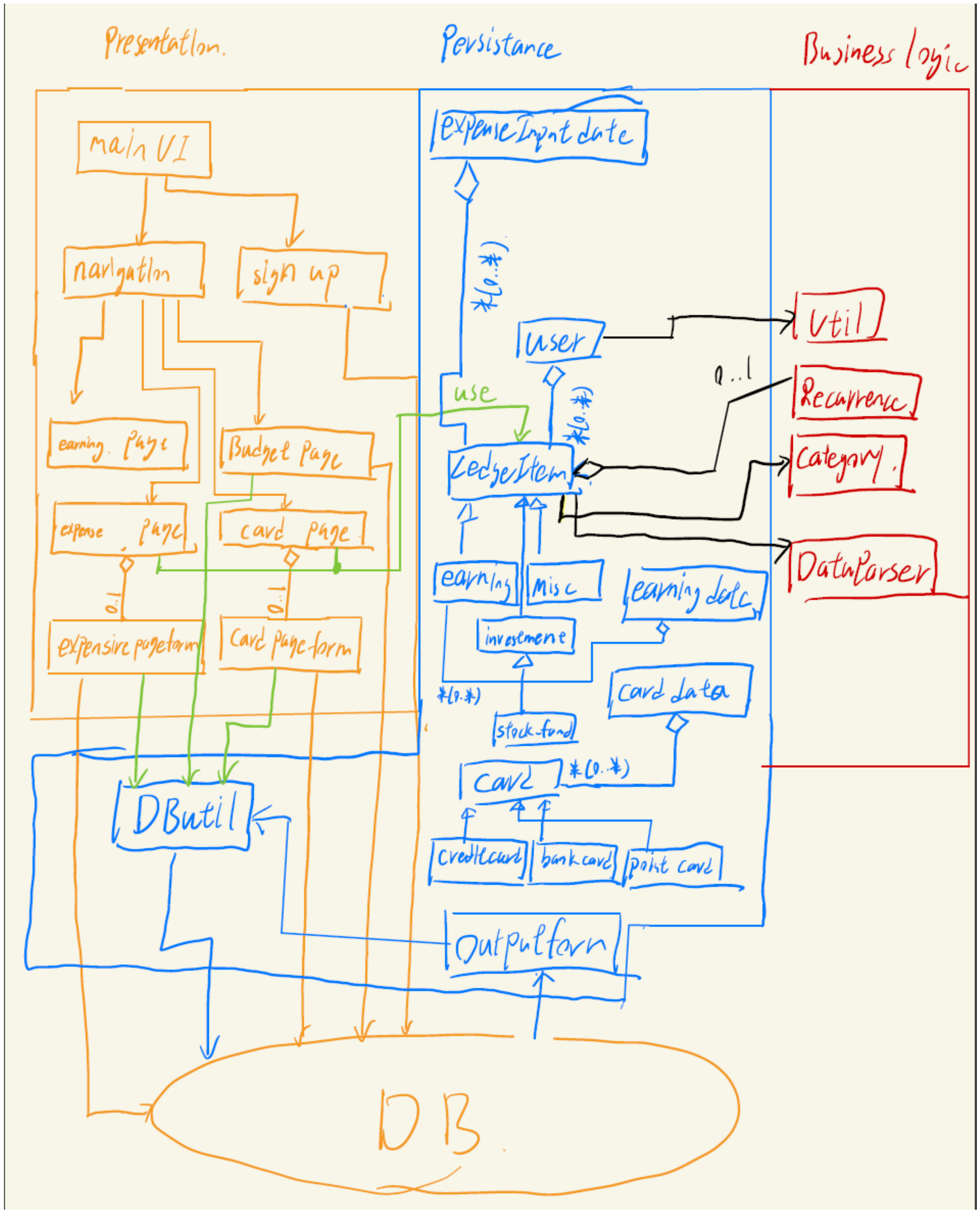
ref	username	item	note	tag	amount	interest_rate	interest	recur	category	date_start
1	ceojeff	testitem 232	testnote 143	investment	230.52	0	0	0	default	2023-03-09
2	ceojeff	testitem 56	testnote 294	misc	185.99	0	0	0	default	2023-03-09
3	ceojeff	testitem 97	testnote 202	bill	62.86	0	0	0	default	2023-03-09
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 (itr2):

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:

Big Story: *Account Management*

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

Cost: *13 days*

Priority: *High* **Actual Cost:** *11 days*

Detailed User Stories:

Detail Story: *Create Unique Accounts*

Creating unique accounts with corresponding password. Each account has different information/data stored.

Cost: *4 days*

Priority: *High* **Actual Cost:** *4 days*

Detail Story: *Joint Accounts*

I want to be able to merge 2 accounts. So that the merged account has all information from both original accounts.

Cost: *5 days*

Priority: *Medium* **Actual Cost:** *4 days*

Detail Story: *Delete Accounts*

Allow users to remove their account and erase the data stored in the application.

Cost: *5 days*

Priority: *Medium* **Actual Cost:** *3 days*

Big user story 5:

Big Story: *Customization Options*

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

Cost: *9 days*

Priority: *Medium* **Actual Cost:** *11 days*

Detailed User Stories:

Detail Story: *Categorize Based on Earnings*

Allow users to choose premade categories, or input our own categorizes.

Cost: *4 days*

Priority: *Medium* **Actual Cost:** *4 days*

Detail Story: *Categorize Based on Expenses*

Allow users to choose premade categories, or input our own categorizes.

Cost: *4 days*

Priority: *Medium* **Actual Cost:** *4 days*

Detail Story: *Update/Delete Items*

Allow users to easily update or delete an item from the table.

Cost: *3 days*

Priority: *Medium* **Actual Cost:** *1 days*

Big user story 6:

Big Story: *Create Reports*

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

Cost: *8 days*

Priority: *Medium* **Actual Cost:** *10 days*

Detailed User Stories:

Detail Story: *Time Period*

Display user expenses/earnings based on time period.

Cost: *4 days*

Priority: *Medium* **Actual Cost:** *3 days*

Detail Story: *Category*

Display user expenses/earnings based on category.

Cost: *2 days*

Priority: *Medium* **Actual Cost:** *2 days*

Detail Story: *Export as Excel Sheet*

User can export a current report as an Excel Spreadsheet.

Cost: *2 days*

Priority: *Medium* **Actual Cost:** *5 days*

Iteration 3

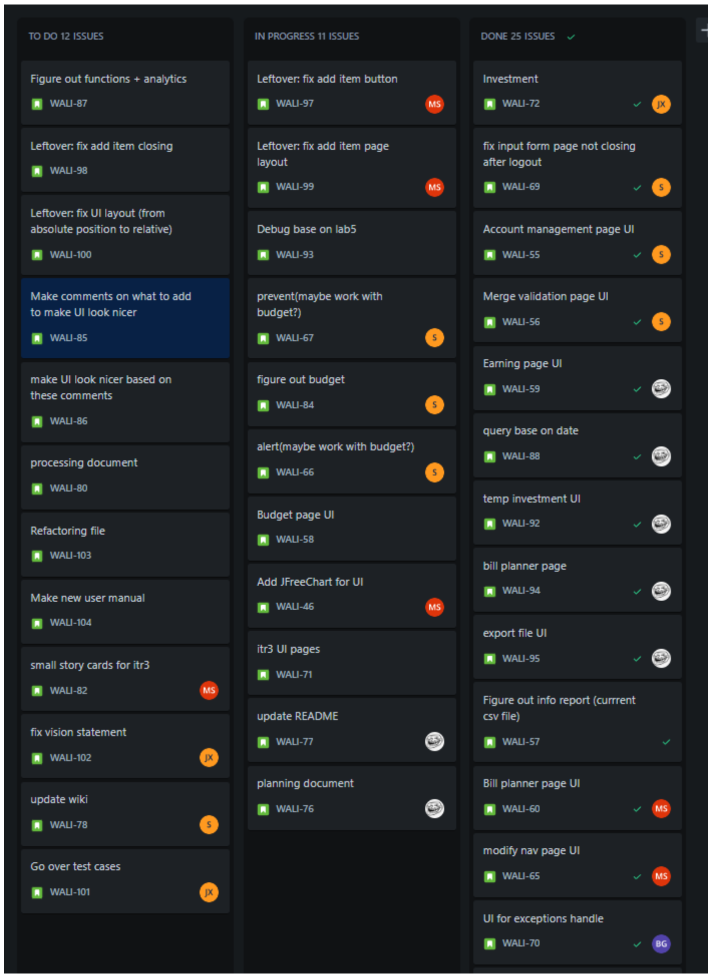
Personalized account(s) with customizations

April 10, 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:



TO DO 12 ISSUES

- Figure out functions + analytics — WALI-87
- Leftover: fix add item closing — WALI-98
- Leftover: fix UI layout (from absolute position to relative) — WALI-100
- Make comments on what to add to make UI look nicer — WALI-85
- make UI look nicer based on these comments — WALI-86
- processing document — WALI-80
- Refactoring file — WALI-103
- Make new user manual — WALI-104
- small story cards for itr3 — WALI-82 — MS
- fix vision statement — WALI-102 — JX
- update wiki — WALI-78 — S
- Go over test cases — WALI-101 — JX

IN PROGRESS 11 ISSUES

- Leftover: fix add item button — WALI-97 — MS
- Leftover: fix add item page layout — WALI-99 — MS
- Debug base on lab5 — WALI-93
- prevent(maybe work with budget?) — WALI-67 — S
- figure out budget — WALI-84 — S
- alert(maybe work with budget?) — WALI-66 — S
- Budget page UI — WALI-58
- Add JFreeChart for UI — WALI-46 — MS
- itr3 UI pages — WALI-71
- update README — WALI-77
- planning document — WALI-76

DONE 25 ISSUES ✓

- Investment — WALI-72 ✓ — JX
- fix input form page not closing after logout — WALI-69 ✓ — S
- Account management page UI — WALI-55 ✓ — S
- Merge validation page UI — WALI-56 ✓ — S
- Earning page UI — WALI-59 ✓
- query base on date — WALI-88 ✓
- temp investment UI — WALI-92 ✓
- bill planner page — WALI-94 ✓
- export file UI — WALI-95 ✓
- Figure out info report (currrent csv file) — WALI-57 ✓
- Bill planner page UI — WALI-60 ✓ — MS
- modify nav page UI — WALI-65 ✓ — MS
- UI for exceptions handle — WALI-70 ✓ — BG

For graphs presentation we are implementing by a third part API called *JFreeChart*.

For exporting we are implementing by a third party API call *Apache Poi*.

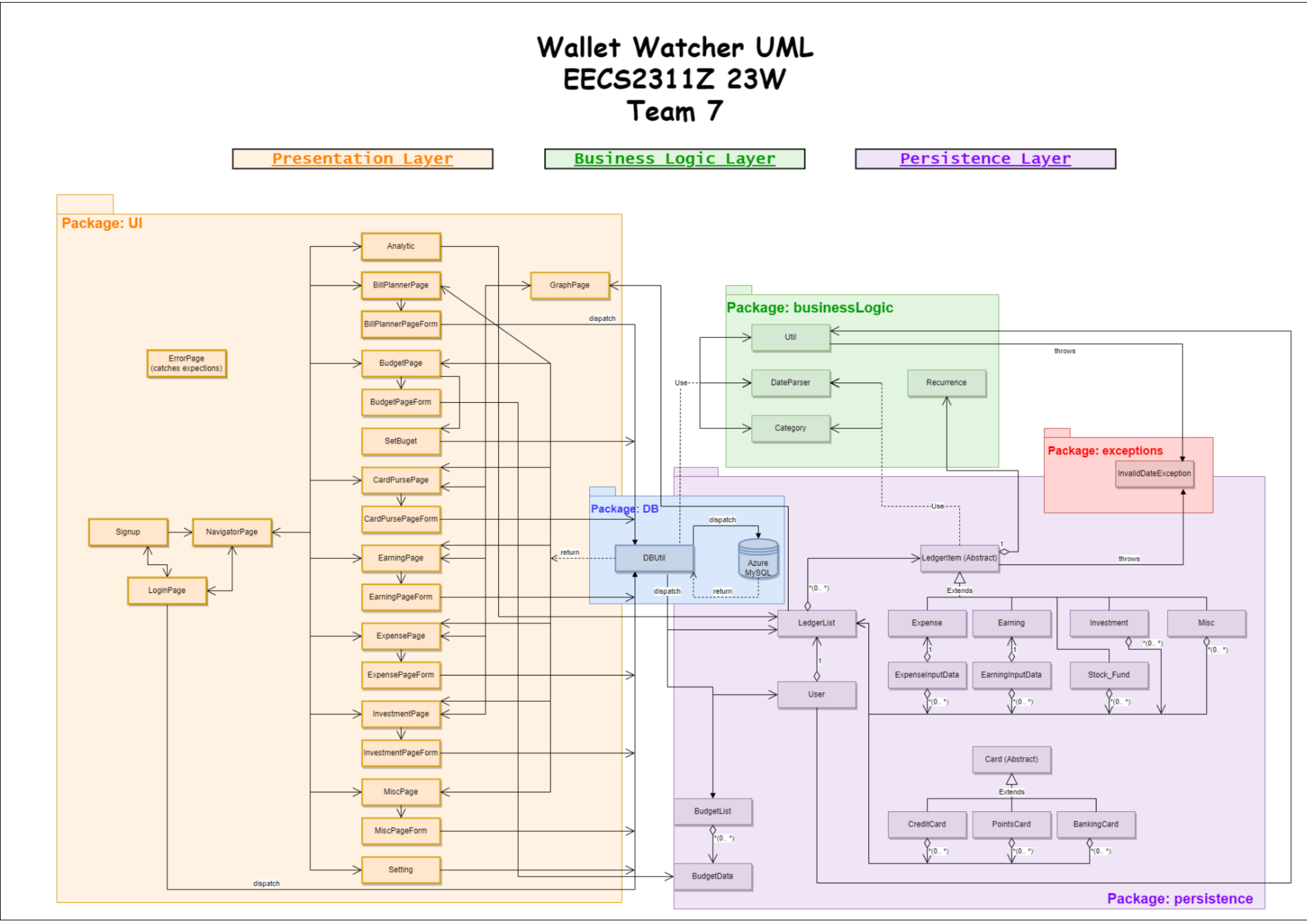
For UI appearance we are implementing by a third party API call *Flatlaf.*

For UI date selector we are implementing by a third party API call *JCalendar.*

.

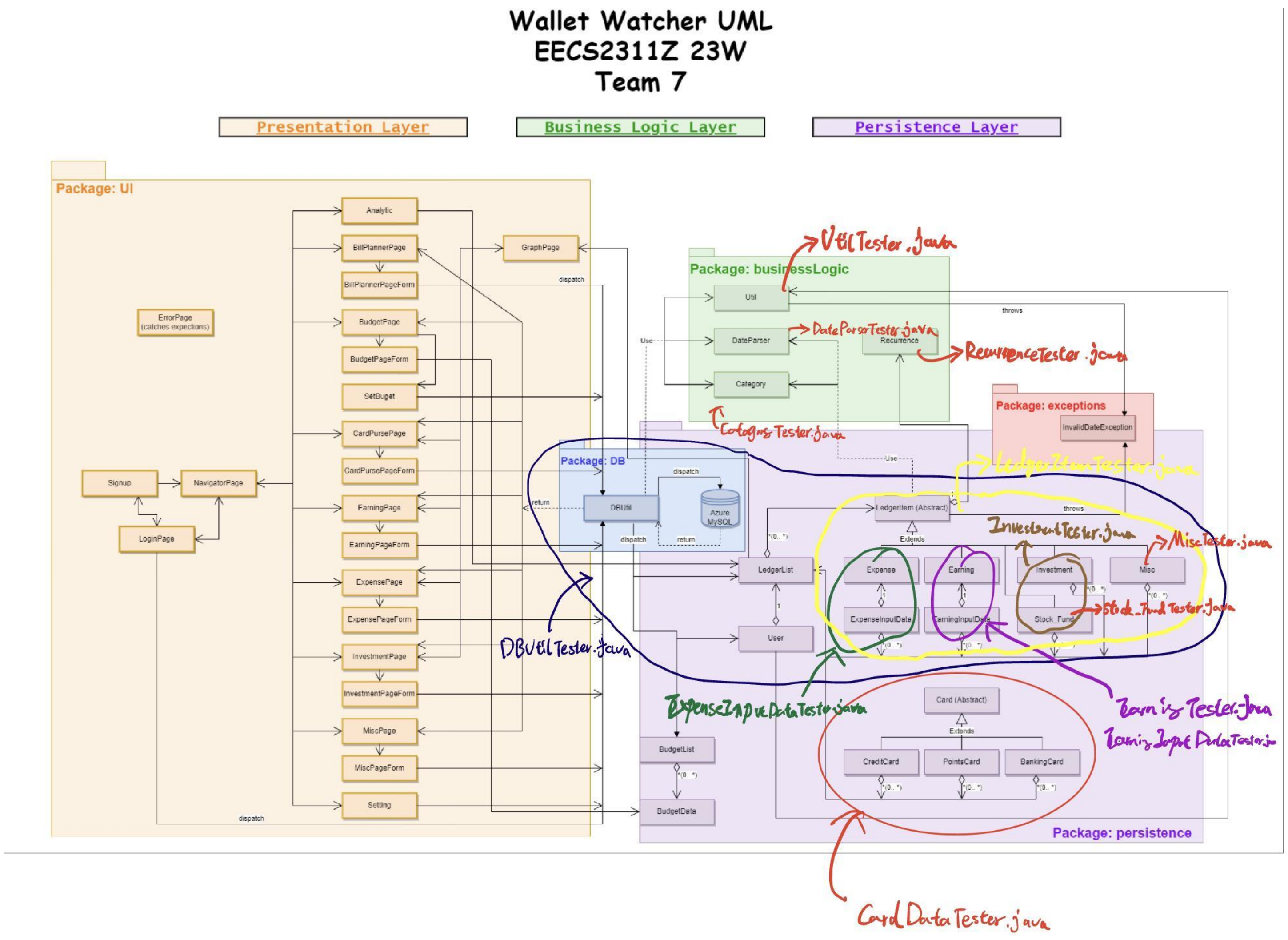
Revised 3-Layer Design (architecture of the application)

Please visit https://github.com/Weilei424/Wallet-Watcher/blob/main/uml.png for details.



Test Cases coverage UML

No integrated testing for UI as per Professor mention during iteration 3.



Major Duties of Each Member (itr3):

Mustafa Syed:

- Design and implementation of GUI for itr3 features.
- Design and implementation of big story Visualization and Analysis.
- Design and implementation of static cache class LedgerList.
- Finalize GitHub Wiki.

Muhammad Shahid:

- Finish up, debug, and refactor UI for itr1 and itr2 features.
- UX and connections between GUI pages design.
- Finalize small user stories of itr3.

Zhenxu Wang:

- Finish up and refactor UI for itr1 and itr2 features.
- Design and implementation of database utility methods.
- Finalize the planning document and user manual.

Jinsong Xie:

- Design and implementation of GUI for itr3 features.
- Help with implementations of itr3 features.
- Help with itr3 documents.

Brahmjot Grewal:

- Refactor and debug for the entire project.
- Design and implementation of exception handling.
- 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 3: Visualization and Analysis

Big user story 7:

Big Story: *Information Visualization*

As a financially independent adult, I want to be able to see my earnings and expenses visualized.

Cost: *15 days*

Priority: *Low* **Actual Cost:** *14 days*

Detailed User Stories:

Detail Story: *Pie Chart Based on Category*

I want to see a pie chart that show the percentage of categories under one report.

Cost: *5 days*

Priority: *Low* **Actual Cost:** *5 days*

Detail Story: *Line Chart Based on Time Period*

I want to see a line chart that show the amount trend during a time period under one report.

Cost: *5 days*

Priority: *Low* **Actual Cost:** *5 days*

Detail Story: *Info Visualization Comparative Graphs*

I want to see my budget goal and actual spending visualized.

Cost: *5 days*

Priority: *Low* **Actual Cost:** *4 days*

Big user story 8:

Big Story: *Analysis*

As a financially independent adult, I want to be able to see some analysis on my records/reports.

Cost: *13 days*

Priority: *Low* **Actual Cost:** *14 days*

Detailed User Stories:

Detail Story: *Monthly Spending/Earning trend*

I want to see the trending of my spending or earning.

Cost: *4 days*

Priority: *Low* **Actual Cost:** *3 days*

Detail Story: *Spending Data Analysis*

I want to see my total and average spending, also the budget data as a summary.

Cost: *4 days*

Priority: *Low* **Actual Cost:** *3 days*

Detail Story: *Spending Notifications based on budget*

I want to be able to know my spending status comparing with my budget plan. And have notification sent based on the result.

Cost: *5 days*

Priority: *Low* **Actual Cost:** *8 days*

Big user story 9:

Big Story: *Application Appearance*

The current application interface looks boring, I want it to look nicer.

Cost: *8 days*

Priority: *Low* **Actual Cost:** *7 days*

Detailed User Stories:

Detail Story: *Theme Changing*

I want to be able to change the theme of the application

Cost: *4 days*

Priority: *Low* **Actual Cost:** *3 days*

Detail Story: *Use Eye-friendly Color*

Use nicer (easy to read) colors for the designs.

Cost: *4 days*

Priority: *Low* **Actual Cost:** *4 days*

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 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.