Budgeter Project Proposal (Original)

Project Title: Expense Manager

Group members: Xi Luo, Eason Qin, Yixiao Li, Shikun Wang, Jiahang Zhang, Hanfei Yang

Project Proposal:

This project aims at creating a personal expenditure manager web application that records the expenses reported in each account. Separate registration and log-in pages will be implemented to allow incoming users to sign up for an official account or log in to their existing account. After registration, users can connect their accounts and form a single unit where all of their expenses will be kept track of by a single expense manager. A profile page will be created for each account. If the account contains multiple users, all users will be able to see who else is associated with this account. On the first day of each month, each account will set a budget for the month where the goal is to not exceed this target amount. When the total expense reaches 90% of the target budget, the application will pop up a message, notifying the user (or all users associated with this specific account) that they are close to exceeding the amount. Finally, the manager will let the user(s) know when the sum of all expenses surpasses the target set for the month. A detailed information page will be included for each recorded expense where the user will input the amount spent, the type of transaction (dining, gas, online shopping, etc), and the place where the transaction occurs. Search functionality will also be implemented where, given any user-input keyword (amount spent, type, location), the manager will retrieve a list of expenses (with all details attached) that match the search criteria. Lastly, the expense manager will

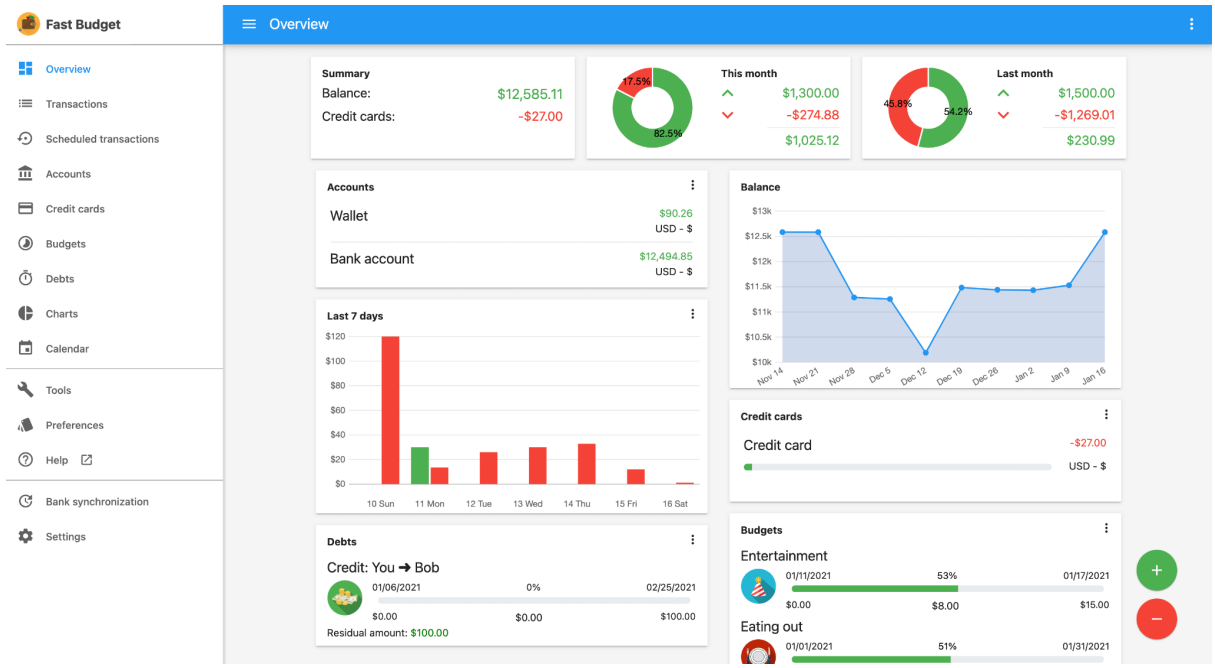
implement a sharing feature where all users on the platform can post about their savings so that everyone can share their joy while encouraging one another to save money.

Budgeter Technical Specifications

Group members: Xi Luo, Eason Qin, Yixiao Li, Shikun Wang, Jiahang Zhang, Hanfei Yang

Project Description: This project aims at creating a personal expenditure manager web application that records the expenses reported in each account.

Expected sample page:



Features:

1. Home page (JSP): Web application name, navbar, charts (Default to Guest login / implemented by using the role id distinction).
2. Login page (JSP): Allow to log in, sign up, or use as a guest to start the application.
    a. User login: login for registered user: ID & password
        i. Google login (jsp, if successful prompts to home page)
        ii. Normal login (login dispatcher extends HttpServlet in java, check DB for password matching)
3. Registration page (JSP): Allow user to register, connect to database
    a. Register dispatcher (extends HttpServlet, java)
    b. Database connection (connect to MySQL, include mysql-connector-java-8.0.28.jar)
4. Group link page (JSP): Create group (Use database to create relations between users)
5. Single User Detail page (JSP):
    a. Amount spent

    b. Type of transaction
    c. Place of transaction
    d. Warning bar when total expense reaches 90% of the target budget
6. Group Detail page (JSP):
    a. Display same set of data as aforementioned in a group
7. Search page: Retrieve all relevant information with a keyword typed in. (Category, Name, …)
    a. Search dispatcher (extends HttpServlet, java), can store search criteria in HttpSession
    b. connect to MySQL to search for matched information
8. Sharing page: Registered users can share their current expenditures on the web application.

Budgeter Design Document

1. Data structures Carolyn

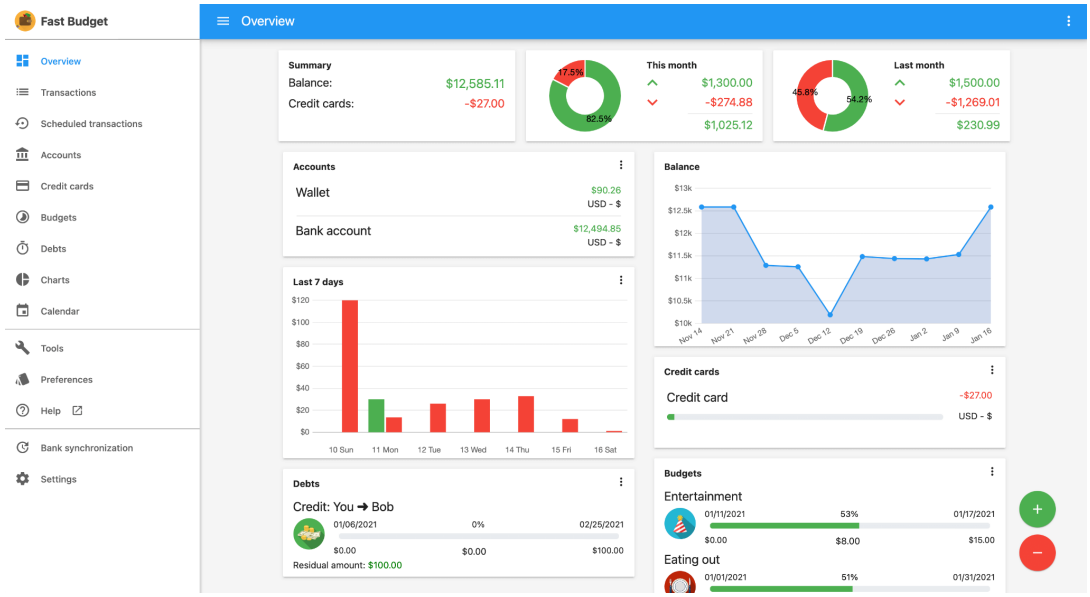
- ArrayList<Expenditure>: since there is an unknown amount of users that could potentially use the application, we need an ArrayList with unlimited length to store all of the expenditures that each user has (which might not be useful).
- ArrayList<User>: we intend to implement a group feature, so we want to create a data structure for each group that stores all users in the group.
- HashMap<String, ArrayList<String>>: This data structure stores all of the associated information in a given category. For example, if we want to search by name, we would input a string name, and the data structure will return all of the expenses associated with that name. Using a hashmap allows fast retrieval of a given key. Given a string keyword, the data structure returns an Arraylist of associated expenses because there can be any amount of expenses associated with a given keyword.

2. GUI design Allen

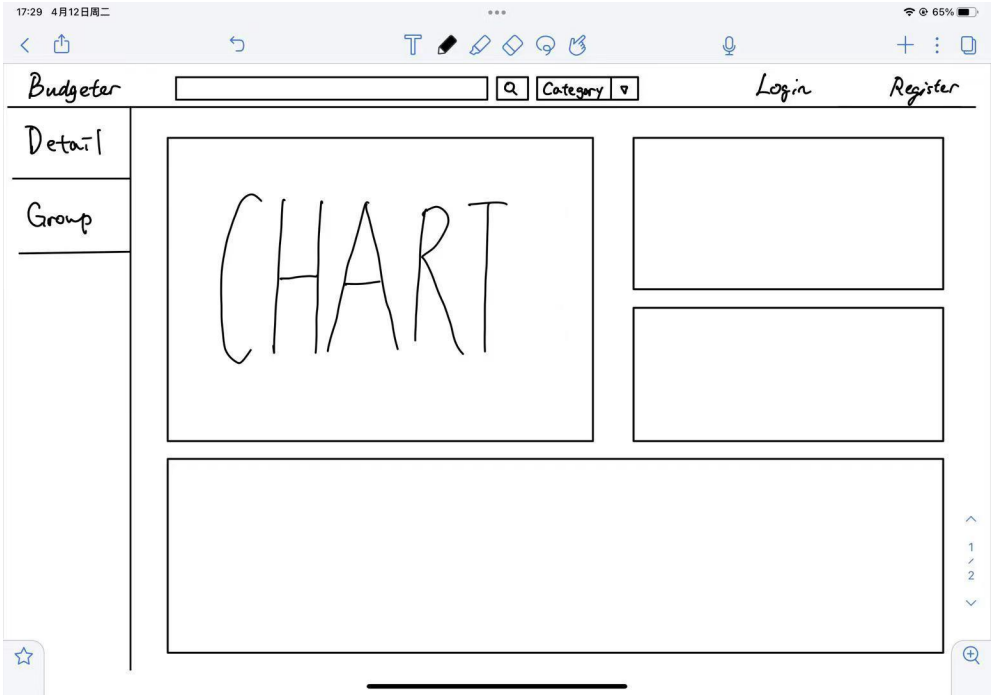
We will use the link below to build the template for the web application's pages.

https://www.figma.com/

The desired style of our web application is simple and concise. The desired impression given to the user should look like this:



Home page (JSP): Web application name, navbar, charts (Default to Guest login / implemented by using the role id distinction).

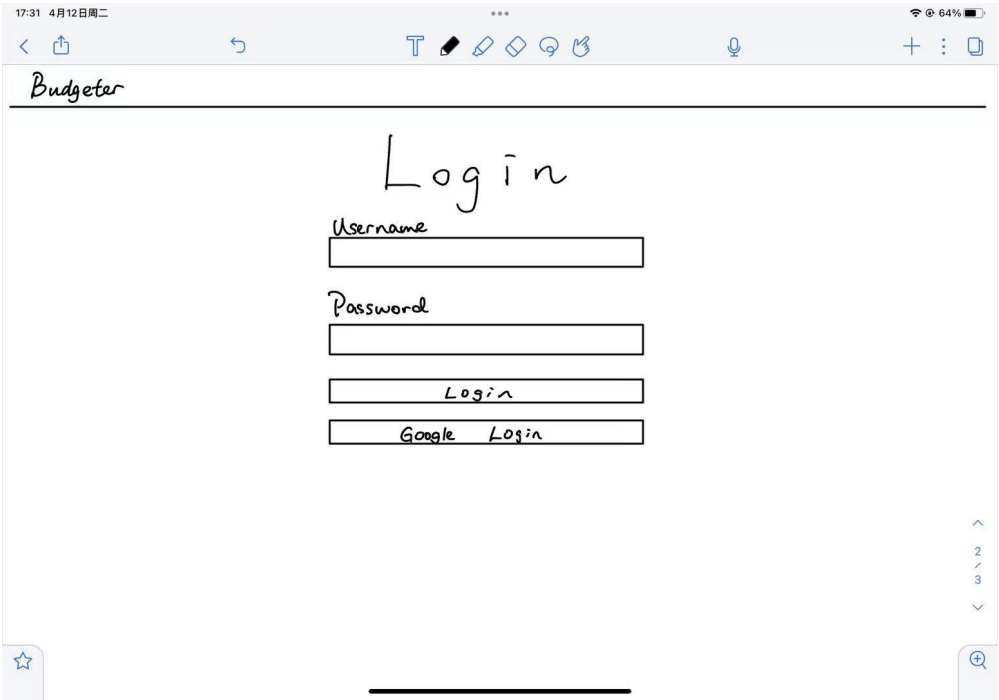


Login page (JSP): Allow to log in, sign up, or use as a guest to start the application.

User login: login for registered user: ID & password

Google login (jsp, if successful prompts to home page)

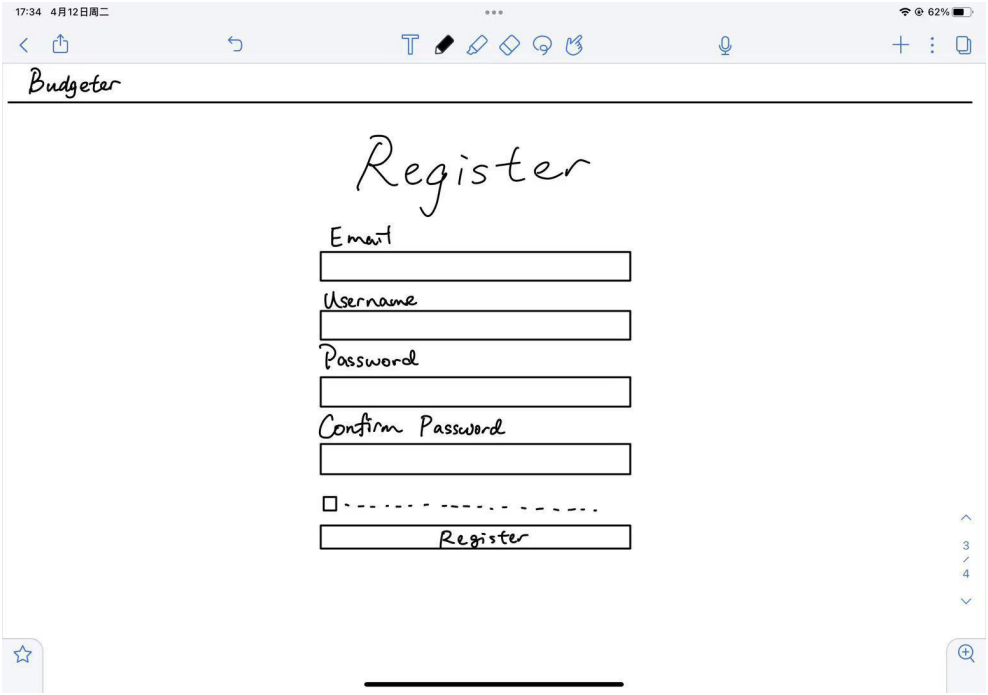
Normal login (login dispatcher extends HttpServlet in java, check DB for password matching)



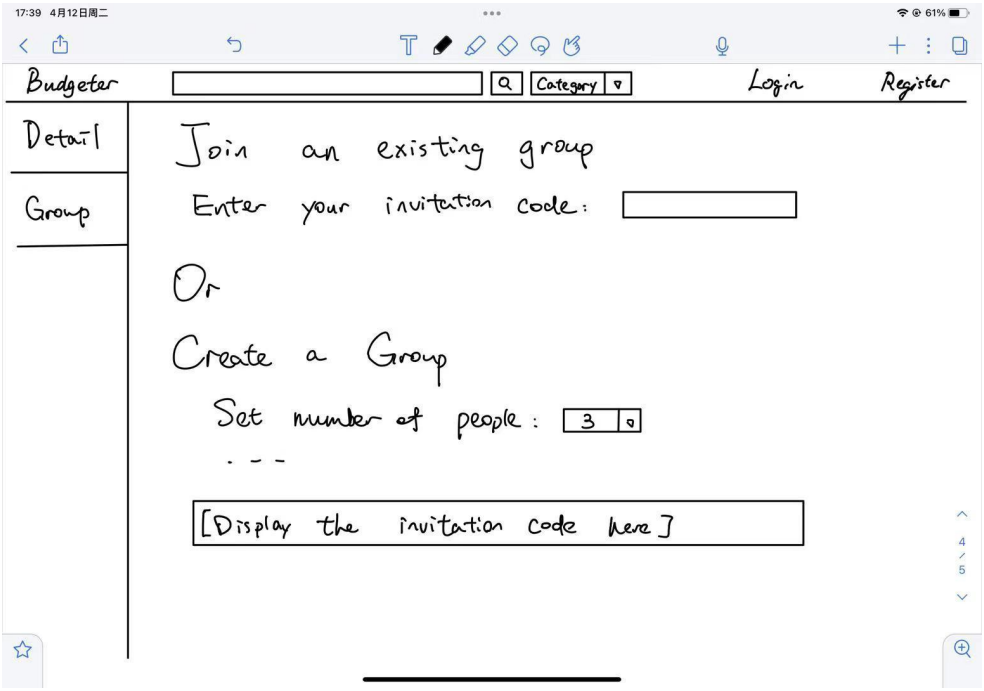
Registration page (JSP): Allow user to register, connect to database

Register dispatcher (extends HttpServlet, java)

Database connection (connect to MySQL, include mysql-connector-java-8.0.28.jar)

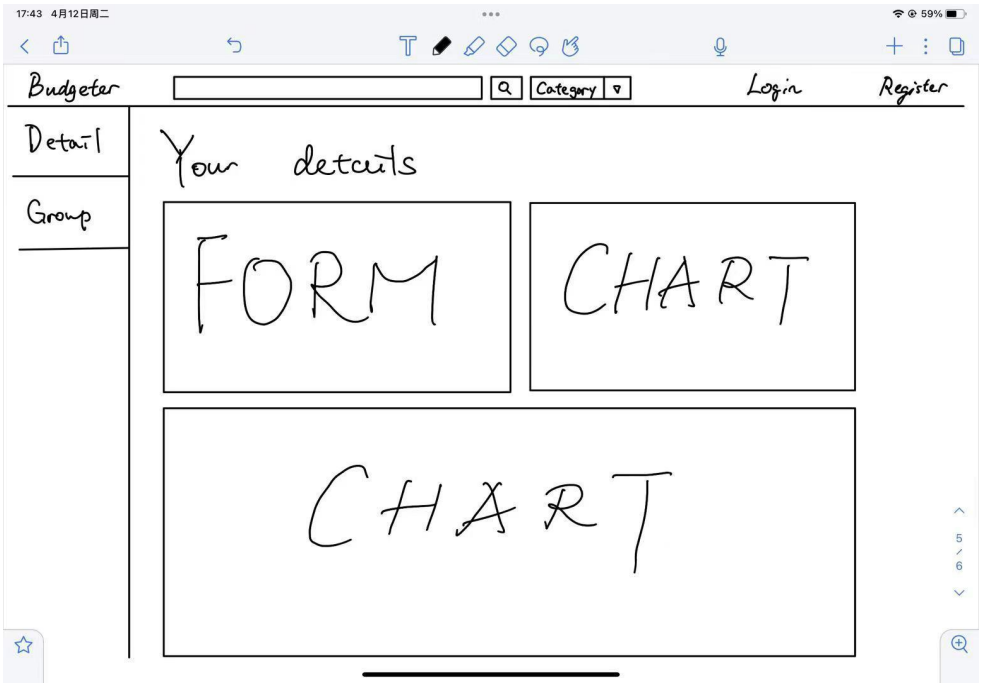


Group link page (JSP): Create group (Use database to create relations between users)



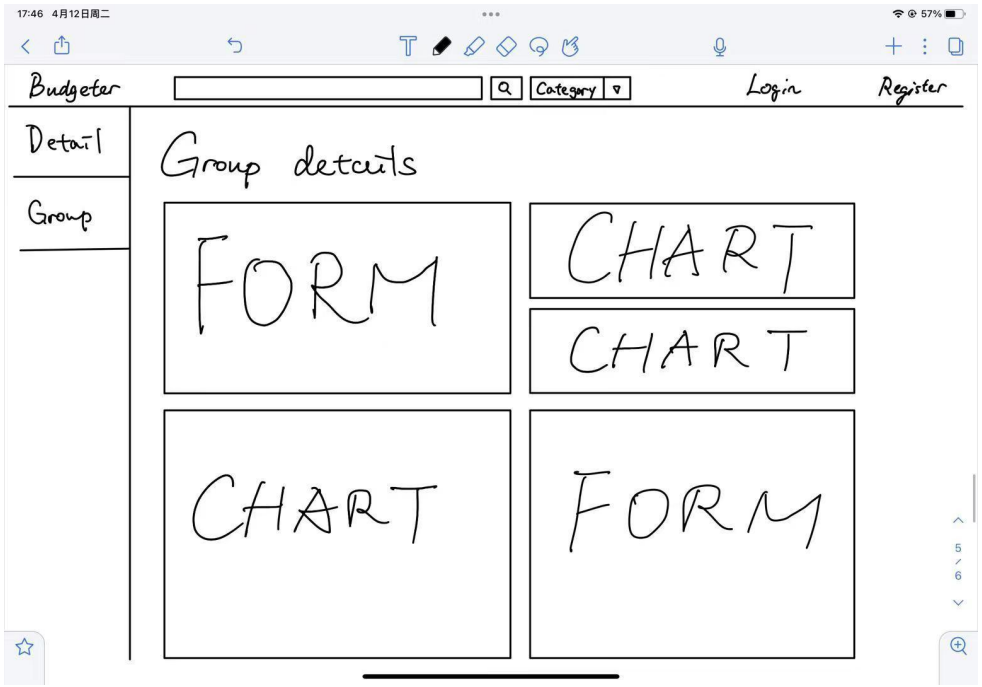
Single User Detail page (JSP):

a. Amount spent
b. Type of transaction
c. Place of transaction
d. Warning bar when total expense reaches 90% of the target budget



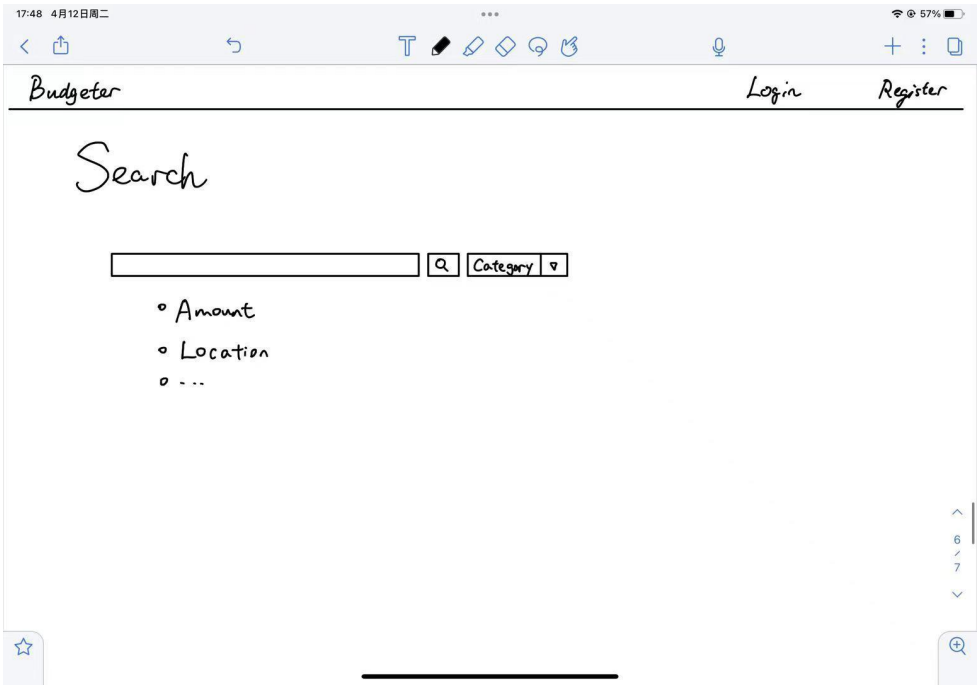
Group Detail page (JSP):

a. Display same set of data as aforementioned in a group

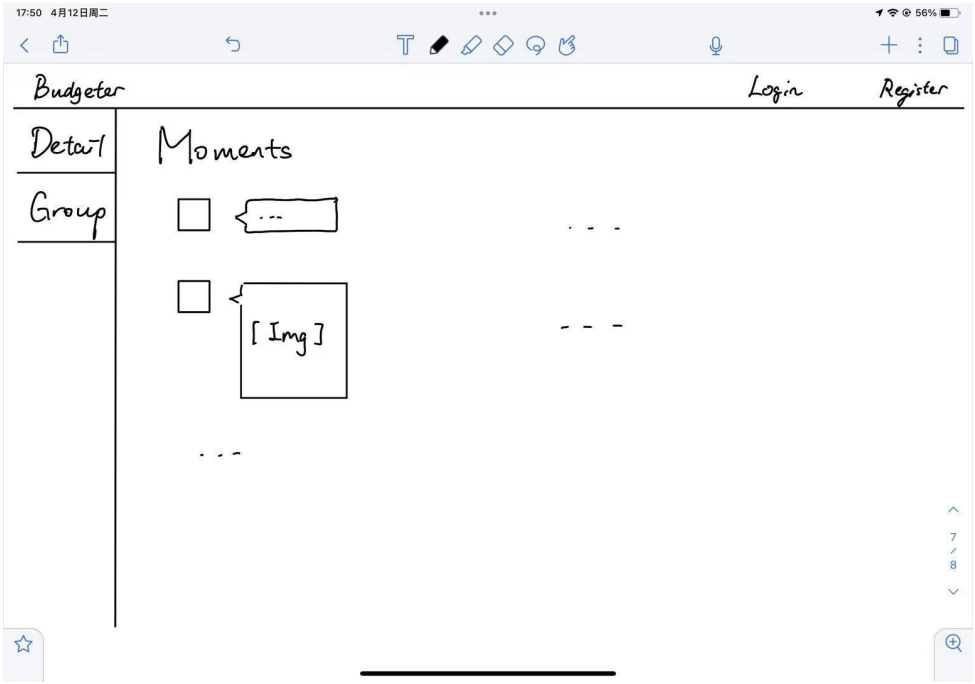


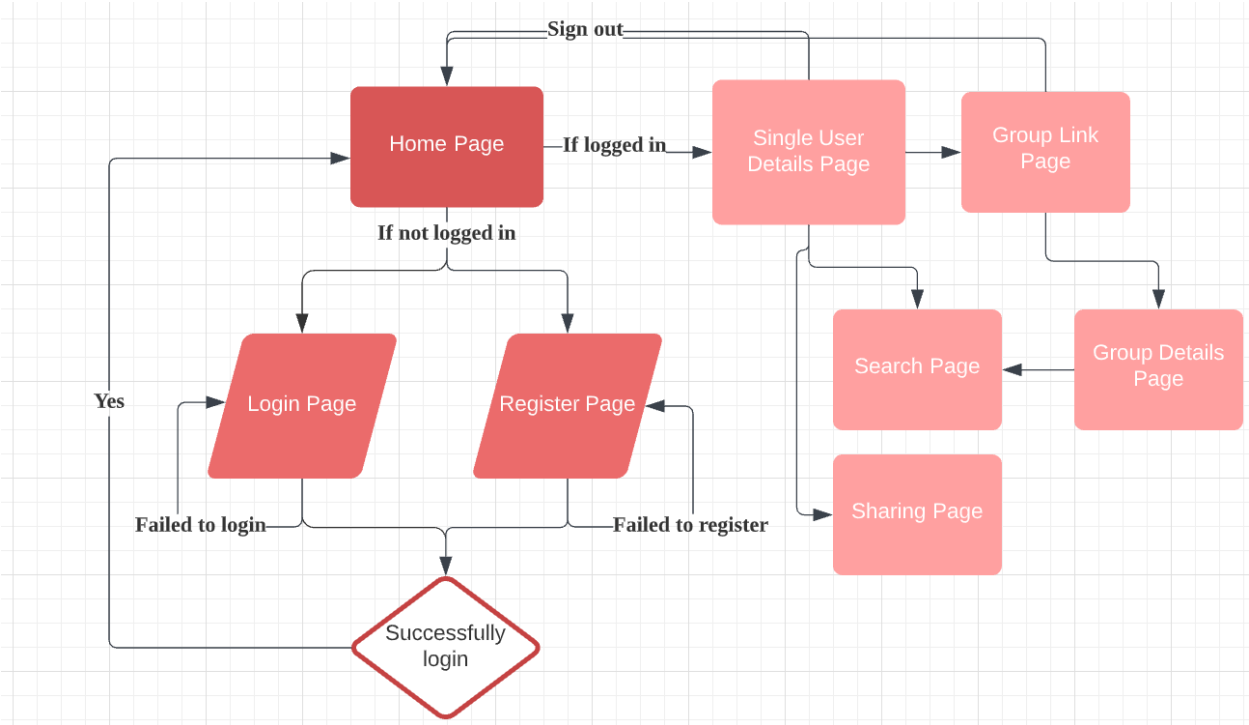
Search page: Retrieve all relevant information with a keyword typed in. (Category, Name, …)

a. Search dispatcher (extends HttpServlet, java), can store search criteria in HttpSession
b. connect to MySQL to search for matched information



Sharing page: Registered users can share their current expenditures on the web application.





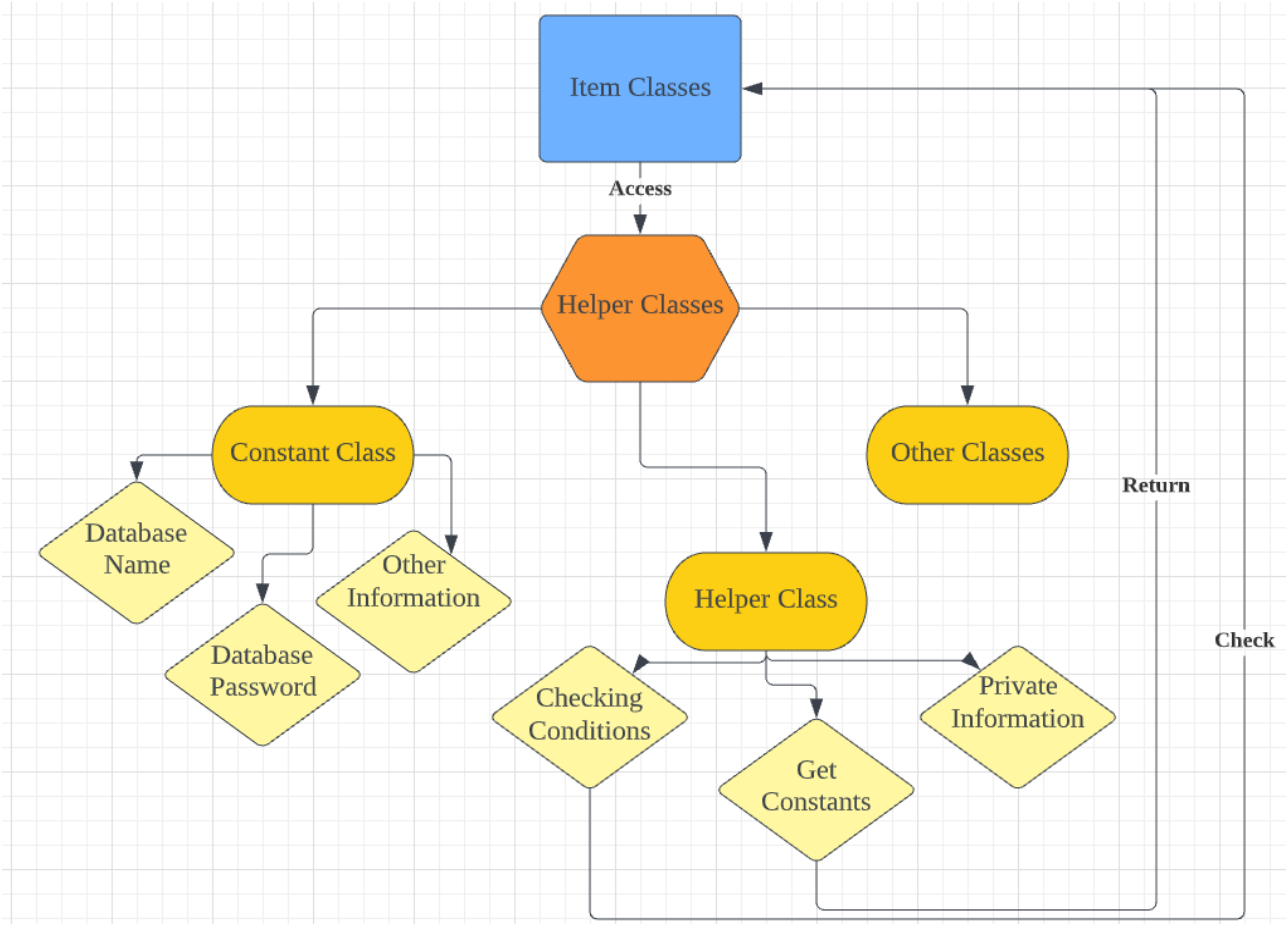
3. Class diagrams Yixiao + Shikun

[Online UML diagram tool | Lucidchart]

Detailed description of each classes:

HelperFunctions:

a. Constant Class (public class): store the constant variables and information
   i. Public data member: Database name
   ii. Public data member: Database password
   iii. Public data member: Name/Email pattern
   iv. Other public/private data members/methods if needed, etc.
b. Helper Class (public class): help checking info resiger/login connecting from DB
   i. Public method: check if name/email valid during login/registration
   ii. Public method: getUserName from database based on entered email
   iii. Public method: checkPassword for user login
   iv. Public method: check if email is already registered for user registration
   v. Private data member: N/A (will add if needed)
   vi. Other public/private methods (will add if needed)
c. Other helper functions



ItemClasses:

a. User Expenditure Class (store one expenditure detail of a user);
   i. Public method: get amount spent for one expenditure
   ii. Public method: get Type of transaction for one expenditure
   iii. Public method: get Place of transaction for one expenditure
   iv. Public method: get expense percentage of target budget
   v. Private data member: amount spent, transaction type, transaction place, expense percentage

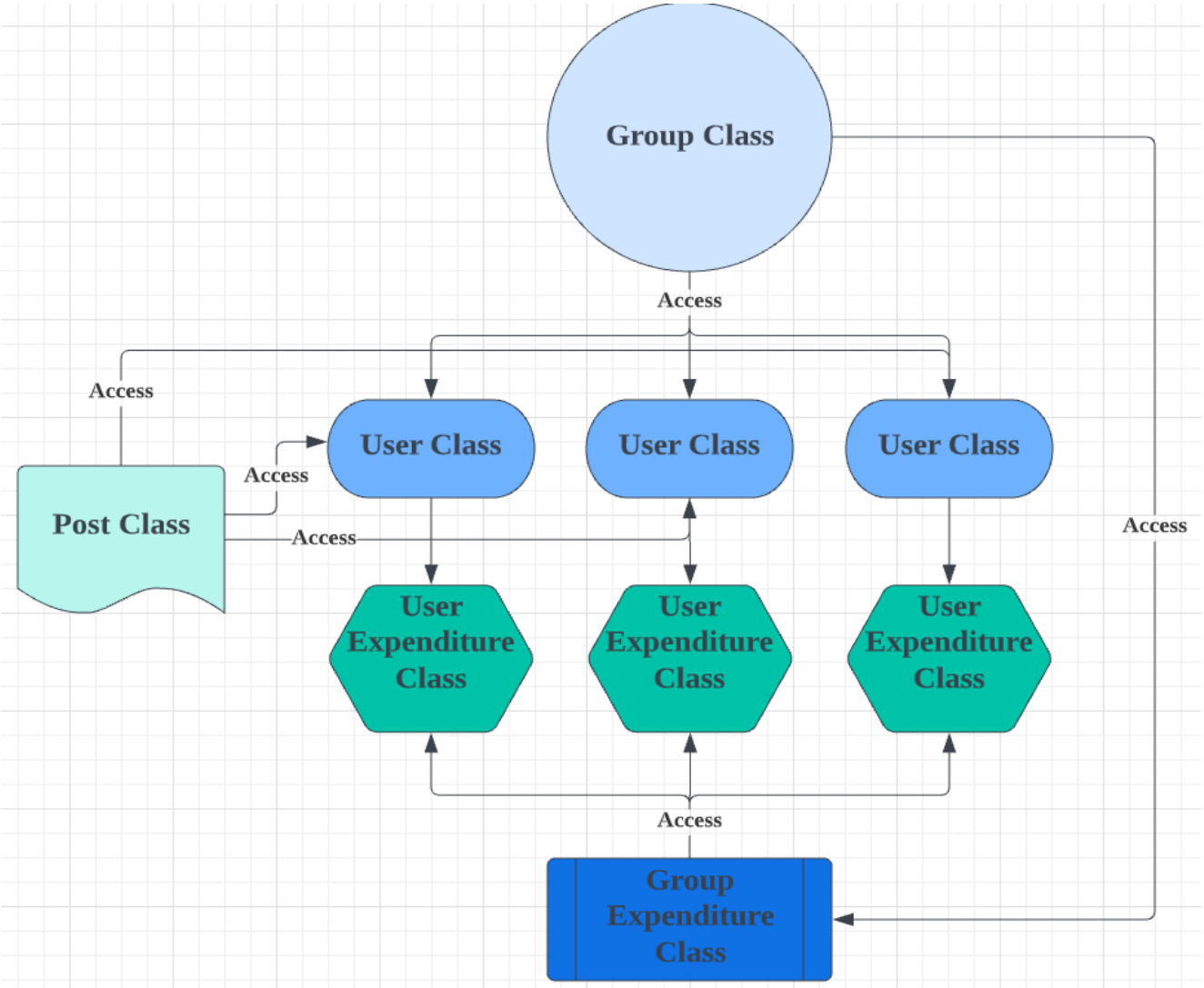
b. User Class (store all information of one user);
   i. Public method: get all expenditures of a user
   ii. Public method: get user name/email
   iii. Public method: get user current expense
   iv. Public method: get user budget
   v. Public method: search group the user belongs to
   vi. Public method: get user's post

vii. Private data member: Arraylist of expenditure for one user; one user's current total expenditure; one user's budget; user name; user email, arraylist of post a user made

c. Group Class (store information of group, containing multiple users);
   i. Public method: find/ check if exists/ get all users in the group
   ii. Public method: get group total budget
   iii. Public method: get group total expenditure
   iv. Private data member: Arraylist of users, group total budget, group total current expenditure;

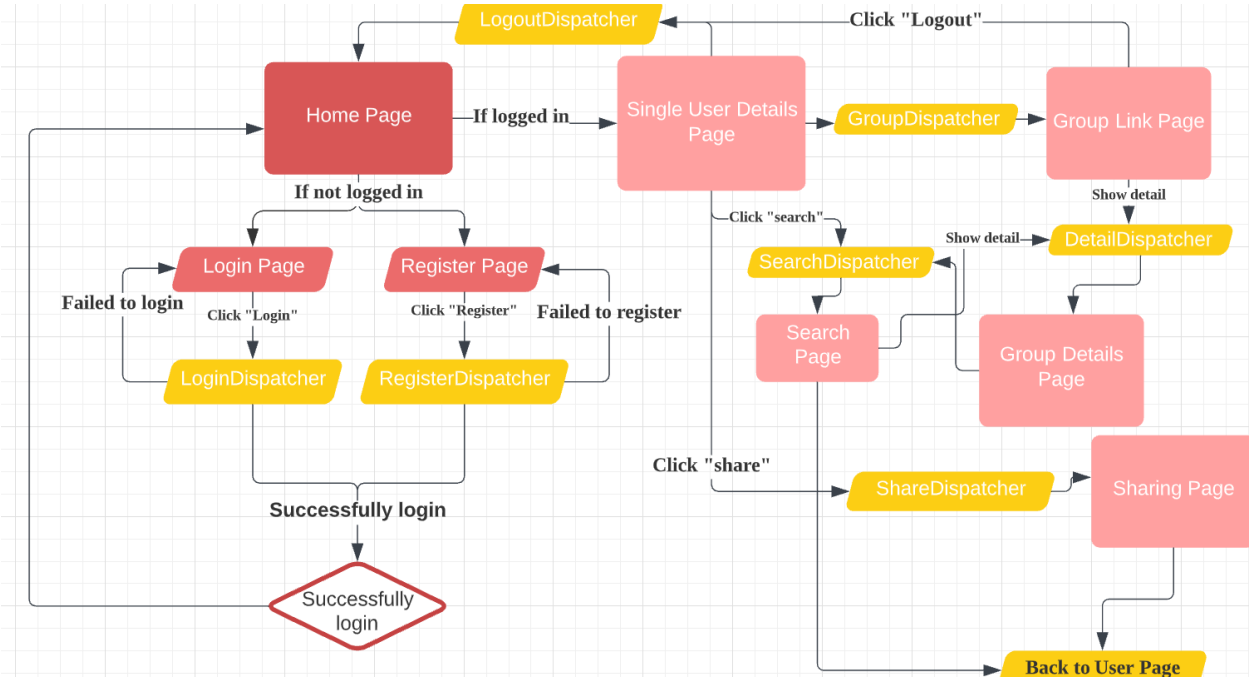
d. Group Expenditure Class: (store the expenditure of one group)

e. Post Class (store the sharing post when user shares with others);
   i. Public method: get post message
   ii. Private data member: post



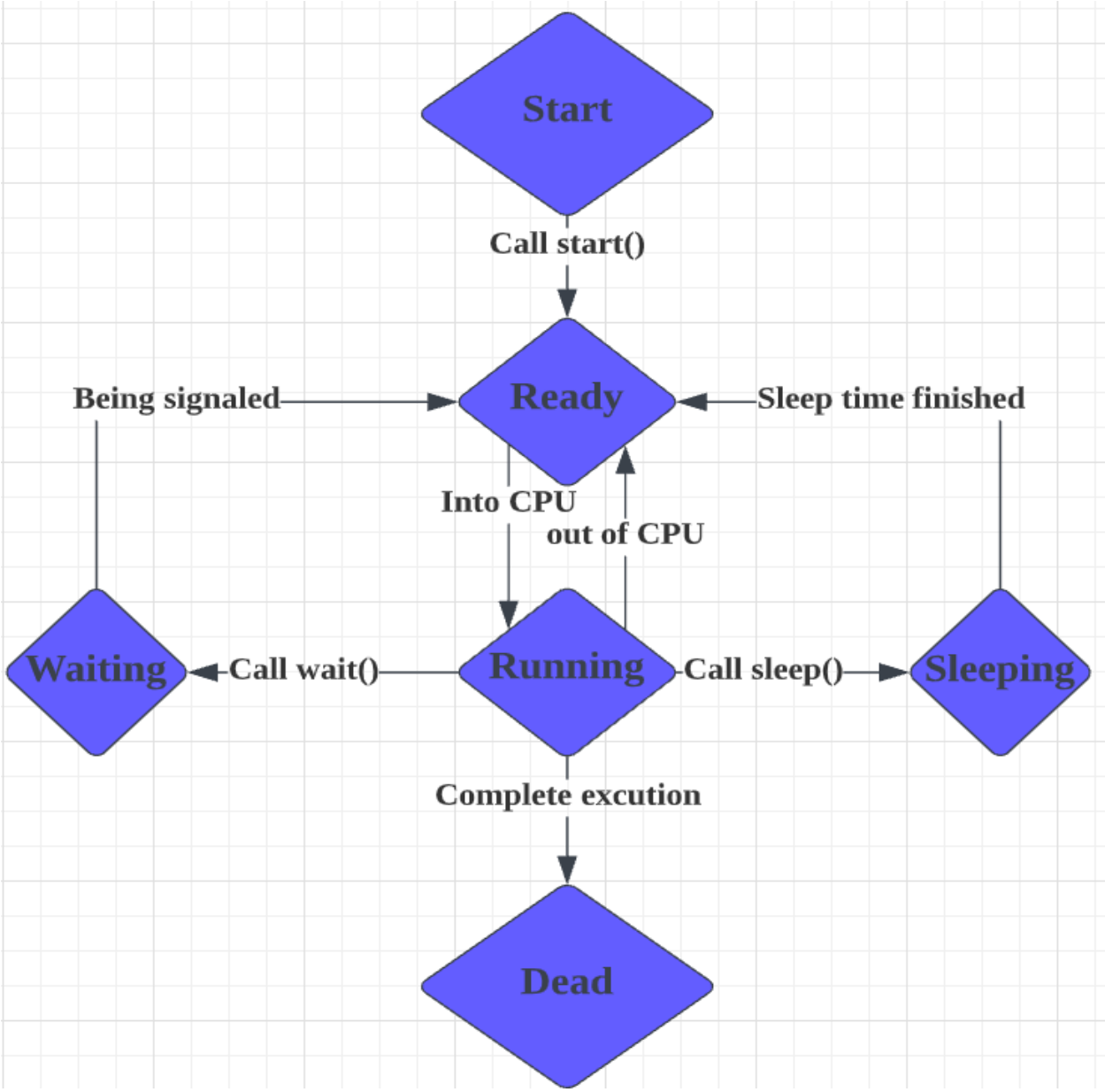
ServletDispatchers:

a. LoginDispatcher Class Extends HttpServlet
   i. with doGet() and doPost()
   ii. redirect user to home page if login successful
b. RegisterDispatcher Class Extends HttpServlet
   i. with doGet() and doPost()
   ii. store user information into database if register successful
c. SearchDispatcher Class Extends HttpServlet
   i. with doGet() and doPost()
   ii. Search for and return searched user/post/group
d. DetailDispatcher Class Extends HttpServlet
   i. with doGet() and doPost()
   ii. Direct user to the detail page of one expenditure
e. LogoutDispatcher Class Extends HttpServlet
   i. with doGet() and doPost()
   ii. redirect user from home page to login/register page
f. ShareDispatcher Class Extends HttpServlet
   i. with doGet() and doPost()
   ii. Direct user to message board page where all posts
g. GroupDispatcher Class Extends HttpServlet
   i. with doGet() and doPost()
   ii. direct one user into its group page



Multi-Thread Class

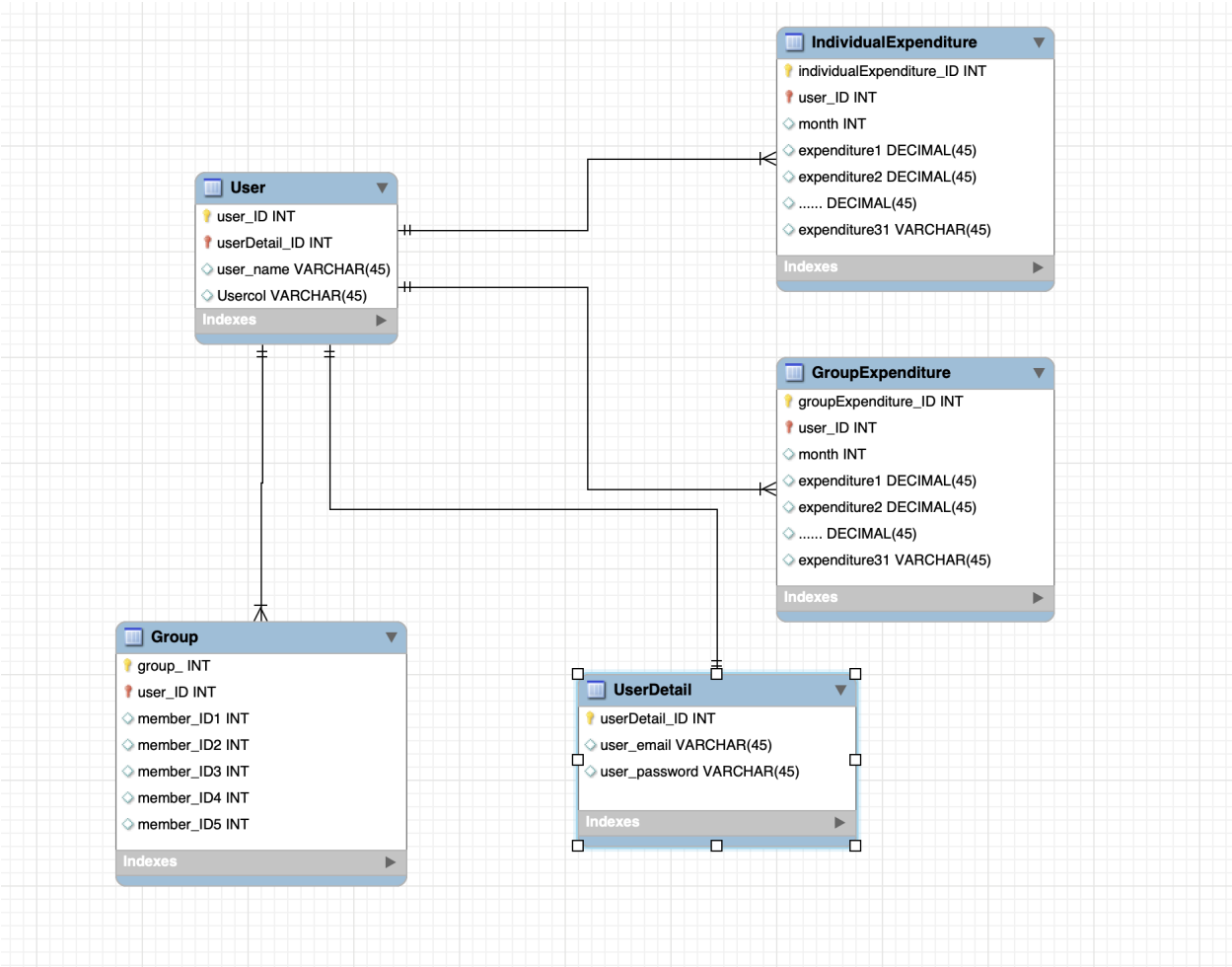
a. Class Extends Thread/Runnable (use ExecutorService to run multi-thread)
b. Synchronized Class/method
c. Private Lock, with ReentrantLock()
d. Private Condition member with newCondition()



Exception Class

a. Throw Exceptions for possible error scenarios

4. Database schemas



5. Hardware/software requirements

Hardware:

A computer that can run any mainstream browser

Software:

Environment for Java SE-19

Environment for Apache Tomcat v9.0

Google Chrome / IE browser / Safari / Opera / Firefox

6. API design (Server design, if applicable) Jiahang

Might use API(REST) to provide or retrieve data from our database without giving our database access permission to the application. Our APIs noew are designed to return a JSON list to provide required information or data.

1. Basic design: include CRUD for each database if applicable.

- details (single user's spending info)

Method	URL	Description	Request Body	Response Body
GET	/some/path/details	Get all details' list	None	A collection for details resources
GET	/some/path/details/{id}	Get a single detail	None	A single detail resource
DELETE	/some/path/details{id}	Delete a single detail	None	Null(if no such data) or the deleted detail resource
POST	/some/path/details	Create a single detail	Data for new detail	Created detail info/source
PUT	/some/path/details/{id}	Update a single detail	Update detail's info	Update detail info/resource

- User (user's profile info) - could be used to update/create user

Method	URL	Description	Request Body	Response Body
GET	/some/path/user	Get a list of all users	None	A collection for users' resources
GET	/some/path/user/{id}	Get a single user	None	A single user resource
DELETE	/some/path/user/{id}	Delete a single user	None	Null(if no such user) or the deleted user's resource
POST	/some/path/user	Create a single	Data for the new	Created user's

		user	user	resource
PUT	/some/path/user/{id}	Update a single user	Updated single user's info	Updated user's resource

- Group (group's info) - could be used to create new group or update group info (users/group information)

Method	URL	Description	Request Body	Response Body
GET	/some/path/group	Get a list of all groups	None	A collection for groups' resources
GET	/some/path/group/{id}	Get a single group	None	A single group resource
DELETE	/some/path/group/{id}	Delete a single group	None	Null(if no such user) or the deleted group's resource
POST	/some/path/group	Create a single group	Data for the new group	Created group's resource
PUT	/some/path/group/{id}	Update a single group	Updated single group's info	Updated group's resource

- shares/moments

Method	URL	Description	Request Body	Response Body
GET	/some/path/share	Get a list of all shares	None	A collection for shares' resources
GET	/some/path/share/{id}	Get a single share	None	A single share resource
DELETE	/some/path/share/{id}	Delete a single share	None	Null(if no such user) or the deleted share's resource

POST	/some/path/share	Create a single share	Data for the new share	Created share's resource
PUT	/some/path/share/{id}	Update a single share	Updated single share's info	Updated share's resource

2. Status code

Status Code	Description
200	OK
201	Created
202	Accepted
204	No Content
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
422	Unprocessable Entity
500	Internal Server Error
503	Service Unavailable

3. API key: currently considering this part. We now do not need this for verification.

Budgeter Testing document

Scenario 1: Guest user login

Test Step	Input	Expected Output
Navigate to home page of Budgeter	Start local server Tomcat, run “index.jsp”	Home page displayed
No User Login	Click on left side to expand navigation menu	Ask user to register/login, feature only includes viewing the main page and seeing the left side navigation or header navigation
Guest Login (Login with google account)	Click on the Budgeter icon, or click on navigation bar	Guest logged in. page jumps back to index.jsp with message “Hi! User” shown next to the budgeter icon. Left side navigation menu will now show add expenditure, personal profile, group page, and group profile for user to select.

Scenario 2: User Register and Registered User Login

Test Step	Input	Expected Output
Navigate to home page of Budgeter	Start local server Tomcat, run “index.jsp”	Home page displayed
User Register	Click on navigation bar “register” option	Jump to register page for user to register. After registration completes will jump back to index.jsp but not logged in
Guest Login (or login with google account)	Click on navigation bar “log in” option	Guest logged in with limited features; navigation bar asks user to register for app

Scenario 3: Registered User Performing Search on a Transaction

Test Step	Input	Expected Output
Navigate to any page	Input transaction date, amount to search all transaction related to the search keyword	Jump to search.jsp display all searched results related to the given keyword.

Scenario 4: Registered User Trying to Join a Group

Test Step	Input	Expected Output
Navigate to any page	Open the left hand side navigation menu and choose the option “group”	Jump to group.jsp and showing option for joining a group or create a group. If joining a non-existent group, there will be an alert at the top of the page.

Scenario 5: Registered User Checking Personal Profile

Test Step	Input	Expected Output
Go to any page with the left side navigation menu	Click “personal profile”	Displaying - personal budget - personal expenditure

Scenario 6: Registered User Checking Group Profile

Test Step	Input	Expected Output
Go to any page with the left side navigation menu	Click “group profile”	Displaying: - Group name - Group member names - Group budget - Group expenditure

Scenario 7: (Multithreading Testing) Multiple User Can Use Budgeter Simultaneously

Test Step	Input	Expected Output
Access Budgeter on 2 different laptop	Start local server Tomcat, run “index.jsp”	Home page displayed successfully on 2 laptop

Scenario 8: (Multithreading Testing) There will be no threading problems when multiple users in one group add expenditures simultaneously, ie, that group's total expenditure gets updated correctly without race condition.

Test Step	Input	Expected Output
Access Budgeter on 2 different laptops (should already logged in with registered account and joined the same group)	Navigate to "add expenditure" (addexpenditure.jsp) and input expenses spent	The expenditure of each individual is reflected on their own personal profile respectively; the group's total expenditure gets updated correctly.

Scenario 9: (Networking Testing) Multiple Users can get connected to the TomCat Server (a shared medium), transport and exchange data by creating and joining groups: If User A joins User B's group, User B's group profile gets updated.

Test Step	Input	Expected Output
Access Budgeter on 2 different laptops(should already login with registered account)	Navigate to "group" (group.jsp) and make one user join another existing group	The existing group will update its group members and print out all current members

Scenario 10: (Networking Testing) Multiple Users can get connected to the TomCat Server (a shared medium), transport and exchange data by adding personal expenditure to groups: If User A adds a new expenditure, the group expenditure (if User A has a group) gets updated.

Test Step	Input	Expected Output
Access Budgeter on 2 different laptops (should already login with registered account)	Navigate to "add expenditure" (addexpenditure.jsp) and input expenses spent	The total expenditure of all individuals in the group is reflected on the group profile page's group expenditure value