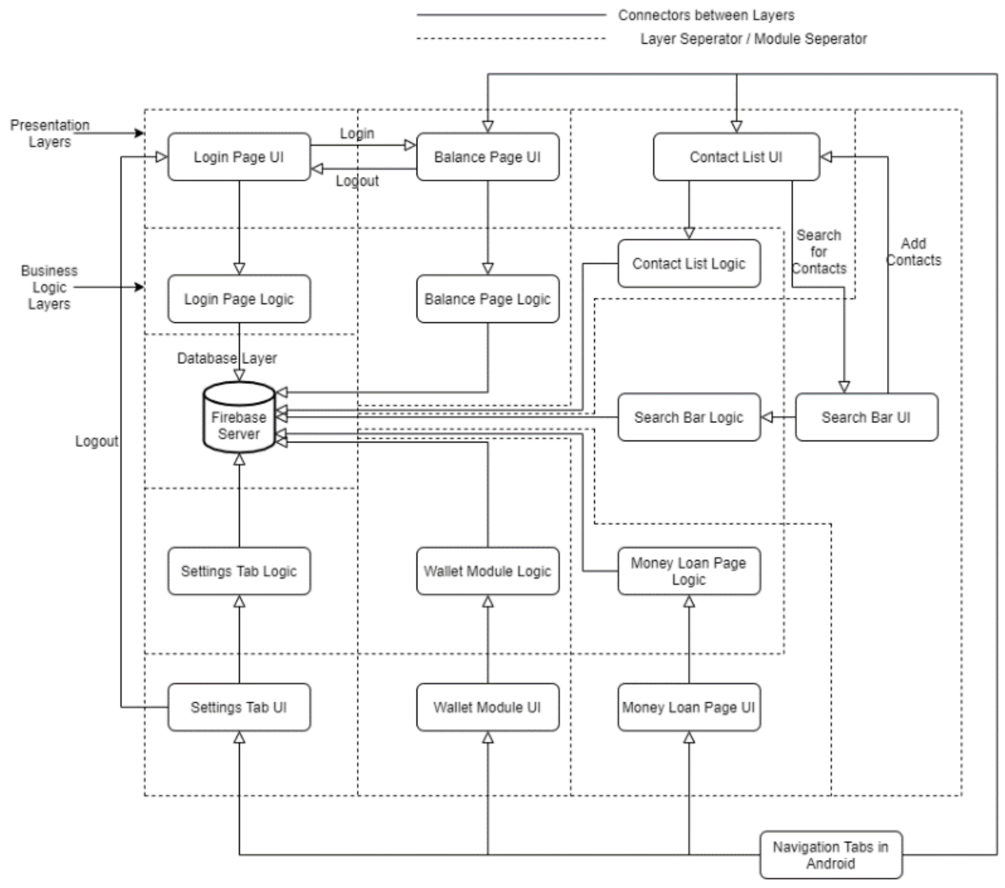
CS 446 Deliverable 5

Shanglin Ye, Zijian Wang, Shuangyou Huang, Haoqing Tian
slye, z2495wan, s257huan, h33tian

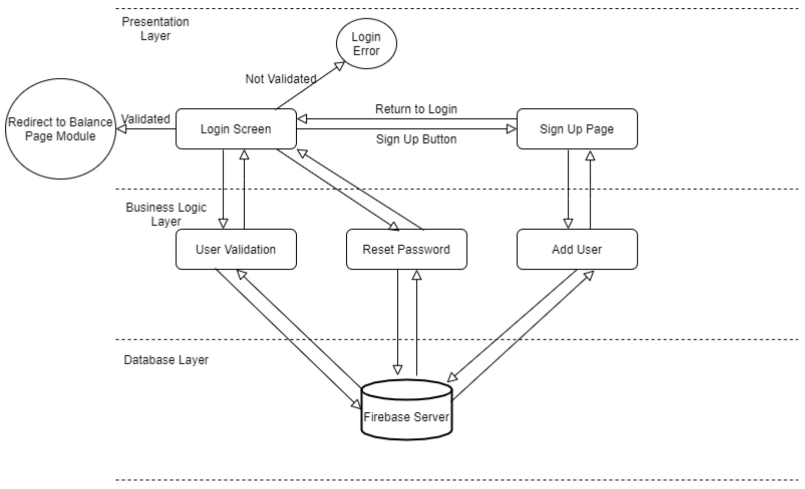
Architecture

While researching the best architectural style for our project, we decided that the layered architecture or more specifically the 3-tiered architecture is best suited for our needs. More specifically, our system will be split up into 3 distinct layers. Our top layer will be the presentation layer and will consist of all the UI components that any user will be able to interact with. Our second layer will be our business logic. This will consist of all the logic that is required by the system to meet it's functional and non-functional requirements. The business layer is also coupled with the persistence layer because we decided that it is more efficient for our business logic to interact with database objects rather than making calls to database objects in another layer. The final layer is the database layer. This is the layer that stores all our data is an external service that our business layer interacts with.



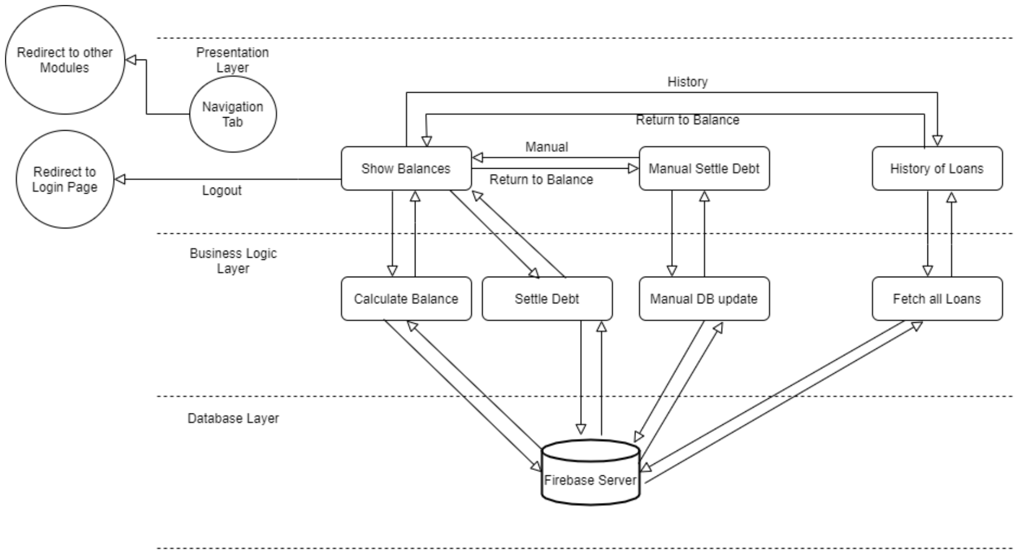
Our architecture style is can be further represented as 7 individual modules that all consist of the 3 distinct layers mentioned in the previous paragraph. Our system has 7 main functional properties that require little or no interaction with any of the other functional properties. This include the login page, the search bar, the contacts list, the balance/home page, the money loan page, the wallet module and the settings tab. Each of these modules has separate UI and each will represent the presentation layer of the respective modules. The only interactions that each presentation layer will have with another presentation layer is to redirect them to that module. For example, when you login in, you will need to be taken to the home page afterwards. The business layer of each module is separate from any other module's business layer. They operate completely independently of each other. The database layer will be shared among all 7 modules since they will be using the same data. The rest of this document will be going into more detail about each individual module. Refer to the diagram to see how our system is structured.

Login Page



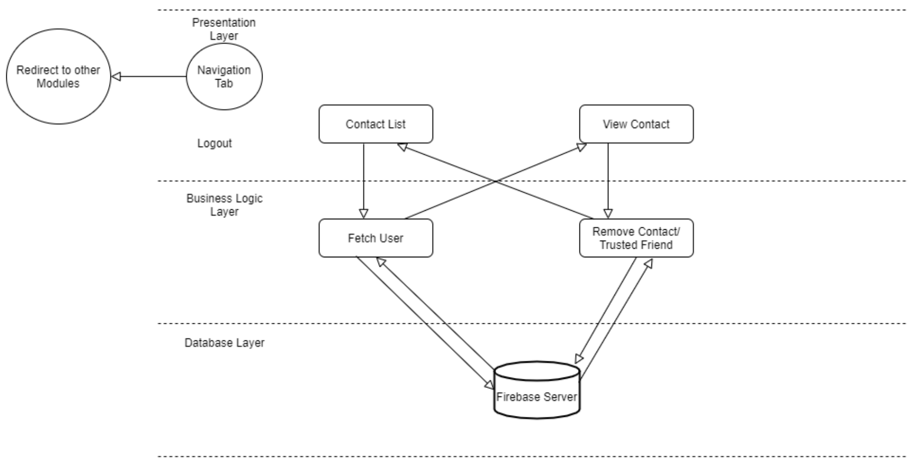
Our login module has two separate pages, each represented as a component in the presentation layer. When a user opens the application for the first time, they will open the Login Screen. They will enter their username and password and click on the Login button. This button will make a call to the User Validation component which will check if the login information is correct. Then it will send a validation flag back to the Login Screen which will redirect the user to the balance page or report a login error. If the user forgot their password, then can click on the Forgot Password link which will send a request to the Reset Password component. This will handle the logic of sending a reset email to the user and subsequently update the database. On the login screen, there is a button to take you to the Sign-Up Page if the user doesn't have an account yet. On this page, they will enter their new account information and click on Sign Up. This will send all the information to the Add User component which will store the new user in the database before returning to the Sign-Up Page with a success or failure message.

Balance Page



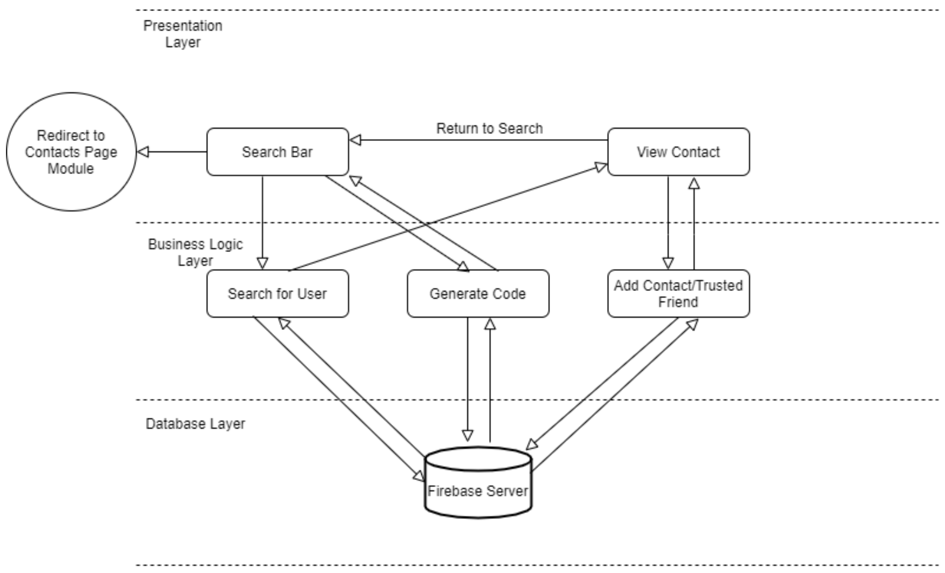
After logging into the system, it will redirect the user to the Show Balances page which also serves as the home page. The page will load the balances by calling the Calculate Balance component. When the user clicks on their outstanding balance, the system calls the Settle Debt component and updates the balance with the money stored in the user's wallet. There is a button that takes the user to the Manual Settle Debt page. They can find an individual user they owe money to and enter the amount they paid manually. This will make a call to the Manual DB Update component. Additionally, the user can click on history button that will take them to the History of Loans page. This page will call the Fetch all Loans component which grabs all the data from the database. Lastly, the logout button will redirect the user back to the login page.

Contacts List



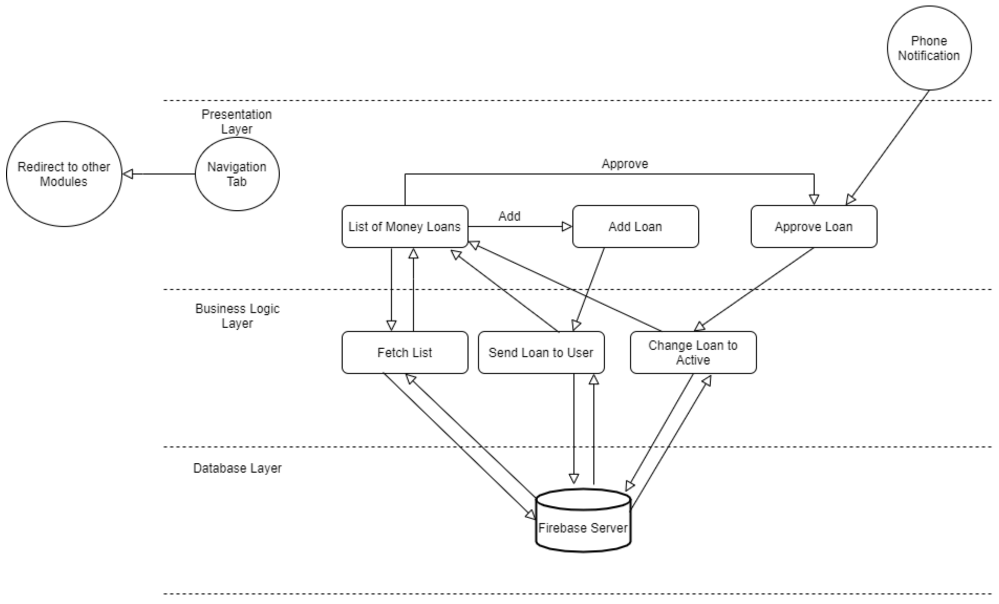
The Contact List page is the default page that is opened for the Contacts List module. On this page, all the user's contacts and close friends are listed out. This is done by making a call to the database and fetching all the friends of the logged in user. If any of the names on the contact list are clicked on, the Contact List page sends the name to the Fetch User component. The component grabs all the desired user's information from the database and return the information to the View Contact Page which will display the info. On this page, there is a button to either remove the user as a contact or trusted friend. If this button is clicked, the system sends the user's info to the Remove Contact/ Trusted Friend component which updates the database so that both user's friend lists do not contain each other anymore. After this action is complete, the component will redirect the user back to the Contact List page. Lastly, click on the search bar will take the user to the Search Bar module.

Search Bar



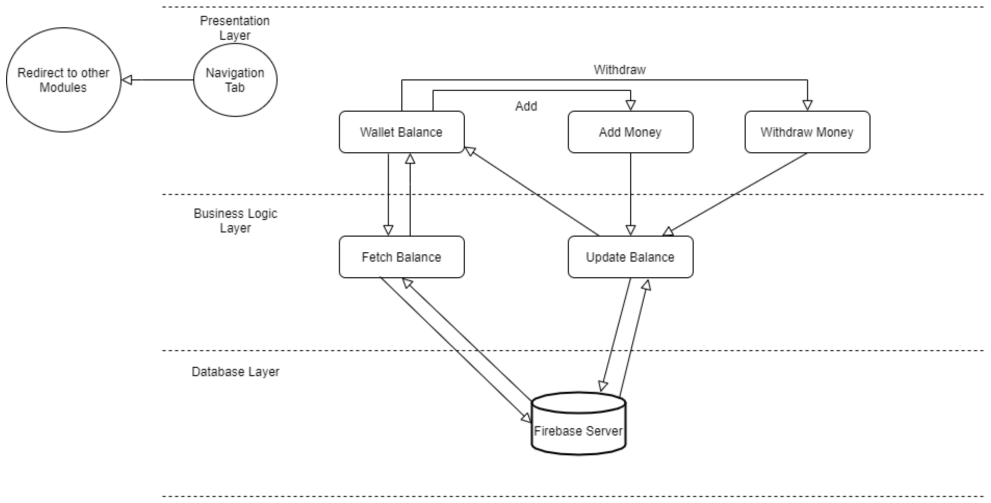
The search bar is specifically used to look up users in the system to add to your contact/trusted friend list. When you enter a name/id in the Search Bar and click on search, it will dynamically generate a list of all the users with similar names/ids. If you click on any of these users, the information will be sent to the Search for User component which looks up the user's information and redirects the user to the View Contact page which displays the user's info. On this page, there is a button to add this user to either your contacts or trusted friend list. Clicking on this button calls the Add Contact/Trusted Friend component which send a request to the user in question which they can accept or decline. The View Contact page also displays all your outstanding friend requests with the option to accept or decline. Lastly, there is a button on the View Contact page to go back to the Search Bar. If you X out of the search bar, you will go back to the Contacts Page module.

Money Loan Page



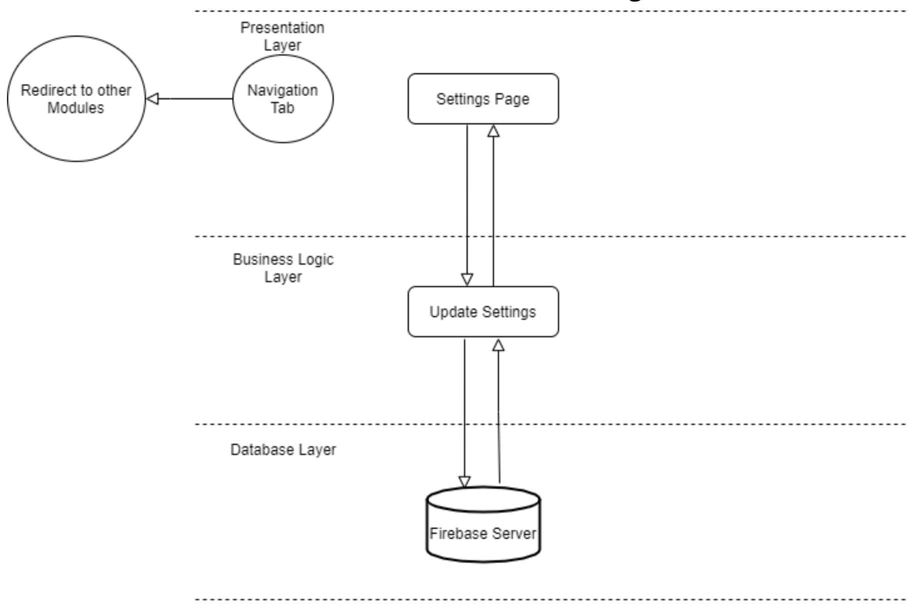
The List of Money Loans page is the default page of the Money Loan Page module. Upon loading this page, the Fetch List component will grab all the loans of the currently logged in user and send it to the List of Money Loans page which displays it to the user. There is a button that will take the user to the Add Loan page. A new loan's information can be added here and after clicking send, the page will call the Send Load to User component which will send a loan request to the other user to be approved. Afterwards, they will redirect back to the List of Money Loans page. There is also a button to go to the Approve Loan page. There is a list of all outstanding loans and clicking accept button will send the request to the Change Loan to Active component and add the loan to the database. The user will once again be redirected to the List of Money Loans page. This page can also be accessed from your phone notifications.

Wallet Module



The Wallet Balance page is the default page of the Wallet Module. Upon opening the page, the Fetch Balance component will be called which grabs the logged in user's wallet balance which is sent back to UI to be displayed. There are two other buttons on this page. One for adding money and one for withdrawing money. These buttons will take the user to the Add Money and Withdraw Money pages respectively. When you enter an amount on either page, they will send the request to the Update Balance component. The money information in the database is updated and the user is redirected back to the Wallet Balance page

Settings Tab



The Settings Page is the only page in the Settings Tab Module. This page lists out all the user's settings. If any of the settings are changed, the system automatically calls the Update Settings component which updates the user's settings in the database. The user can also logout of their account from this page.

Navigation Tab

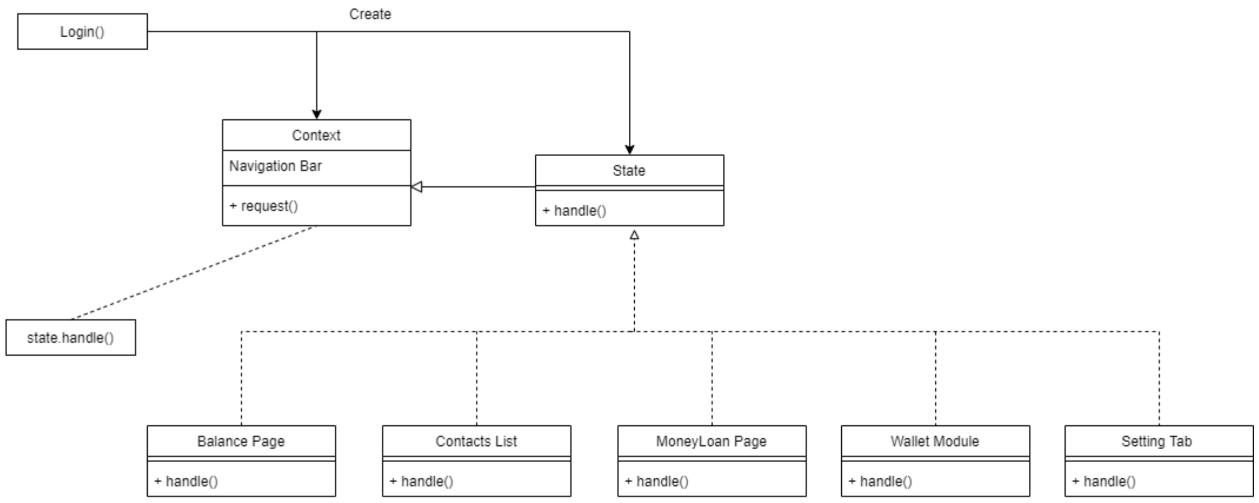
As you may have noticed, there is a navigation tab in the presentation layer of most the modules which can redirect you to other modules. This tab is static and remains on screen while navigating between the different modules/pages. It is displayed at the bottom of every page. There tab consists of 5 buttons (5 different modules) and clicking on any of the buttons brings you to the module in question. It is easy to modularize the different sets of functions since they mostly go through this tab to switch to a different set of functions.

Firebase Server

All the different modules share a database layer which is labelled as Firebase Server in all the diagrams. Firebase is an external service that we use to store our database information. In the business logic layer, we make API calls that directly add, update, delete or select information from the database. We can also directly access the firebase database from their website and view or make changes to the data there as necessary. Even though this is an external service, we still included it in our architecture since it is crucial for our system to run.

Design

As described in the architecture section of this document, our system can be split up into 7 distinct modules which are independent of each other. There are buttons in each of the presentations layers that allow the user to go from one module to another module. Thus, the State Design Pattern is the best pattern to capture the overall design of the system.

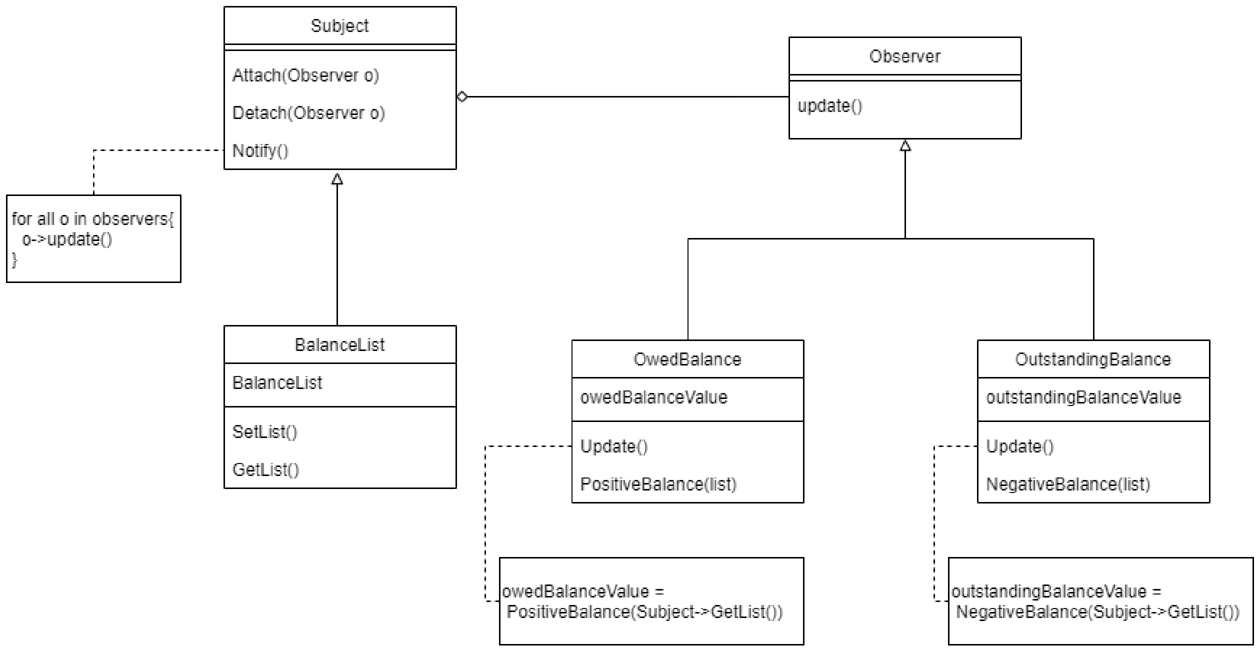


In this system, the 5 modules that contain the navigation bar, described in the architecture section of this document, each represents a concrete state. The context class in the system is an UI object that always has a navigation bar at the bottom. It also contains the current state which is one of the modules. The modules states are also UI objects and the context class will put the module UI above the navigation bar. For example, when the user first logs in a context class is created, and the default state is set as the home page/balance page. When the user clicks on another tab in the navigation bar, it will take to them that respective module by changing the state of the context class.

Each of the States is represented as one basic object in the diagram, but each consists of all the components in the presentation layer and the business logic layer of each respective state. Some of the states are simple such as the Settings Tab which has one class each for its presentation layer and its business layer. Other states may represent multiple objects with their own design patterns and will be analyzed later in this document.

This design helps minimize coupling because each of the modules is now separated into individual states that are not affected by each other. This also allows us to add more modules as we expand our system and add more items into or navigation tab. This design also has high cohesion since each of the states represents one unique module and thus all the functions are directly related to each other. It is possible that two modules will eventually become more and more dependent on each other. For example, it would be ideal to combine the Balance Page and the Money Loan Page. Once you enter a loan into the system, the balances are directly affected, and it would be more efficient to directly updates the balances on the balance page rather than make an expensive database call. However, this is done on the business logic layer of each module and those layers should be separate from each other. We would retain our current design pattern and architecture by merging the two modules as one new module. Our system runs the risk of having of having the modules become more and more dependent on each other. A solution would be to have the context class contain global variables that all the modules can share. These are a couple of issue that we foresee occurring as our system evolves and our solutions for dealing with them.

Balances - Observer Design Pattern

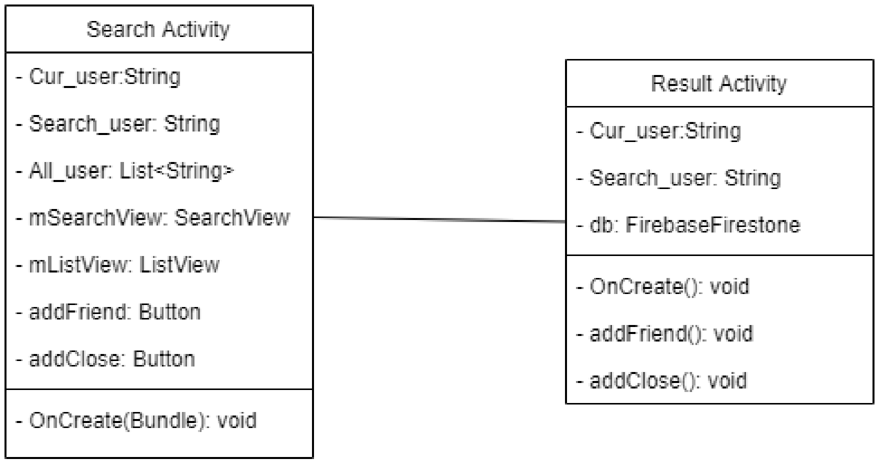


On the home page of the application, you will be able to see two balances: The OwedBalance and the OutstandingBalance. OwedBalance represents the amount of money that other people owe you and is represented by green text and the OutstandingBalance represents the amount of money that you owe other people. Each of these balances is represented by an object that is both an UI component that displays the balance number and the balance number itself (OwedBalanceValue and OutstandingBalanceValue respectively). These balances are calculated from an object called BalanceList. In our database, each user has an entry for every one of their contacts and for each entry there is a positive or negative number associated to it (positive representing owed amount and negative representing outstanding amount). The BalanceList object makes a call to the database using the SetList function for the user's contact list and stores it to limit the total number of costly calls to the database.

Since OwedBalance and OutstandingBalance should update when the BalanceList changes, the Observer design pattern is the best suited for the design of all the classes and objects related to the balance. In this case, the BalanceList is the Subject and the OwedBalance and OutstandingBalance classes are the Observers. The Update function should get the BalanceList and update their values for both classes by calling the PositiveBalance and NegativeBalance functions respectively. The Observer pattern is ideal here because the two balances are dependent on a BalanceList and it is better than making multiple database calls in different places. In addition, the BalanceList doesn't have to make any assumptions about OwedBalance and OutstandingBalance making all the classes loosely coupled. Lastly, there might be uses in the future that also depend on the BalanceList and it would be very easy to add another class that also observers a BalanceList object.

The presentation layer of the Balance Page module will both contain instances of the OwedBalance and the OutstandingBalance classes. The PositiveBalance and NegativeBalance function calls invokes a call to the business logic layer. Therefore, the BalanceList is contained in the business logic layer. The Observer pattern allows the classes to be separate thus making it possible to decouple the presentation layer and the business logic layer. The SetList function makes a call to the database layer which is our Firebase Server

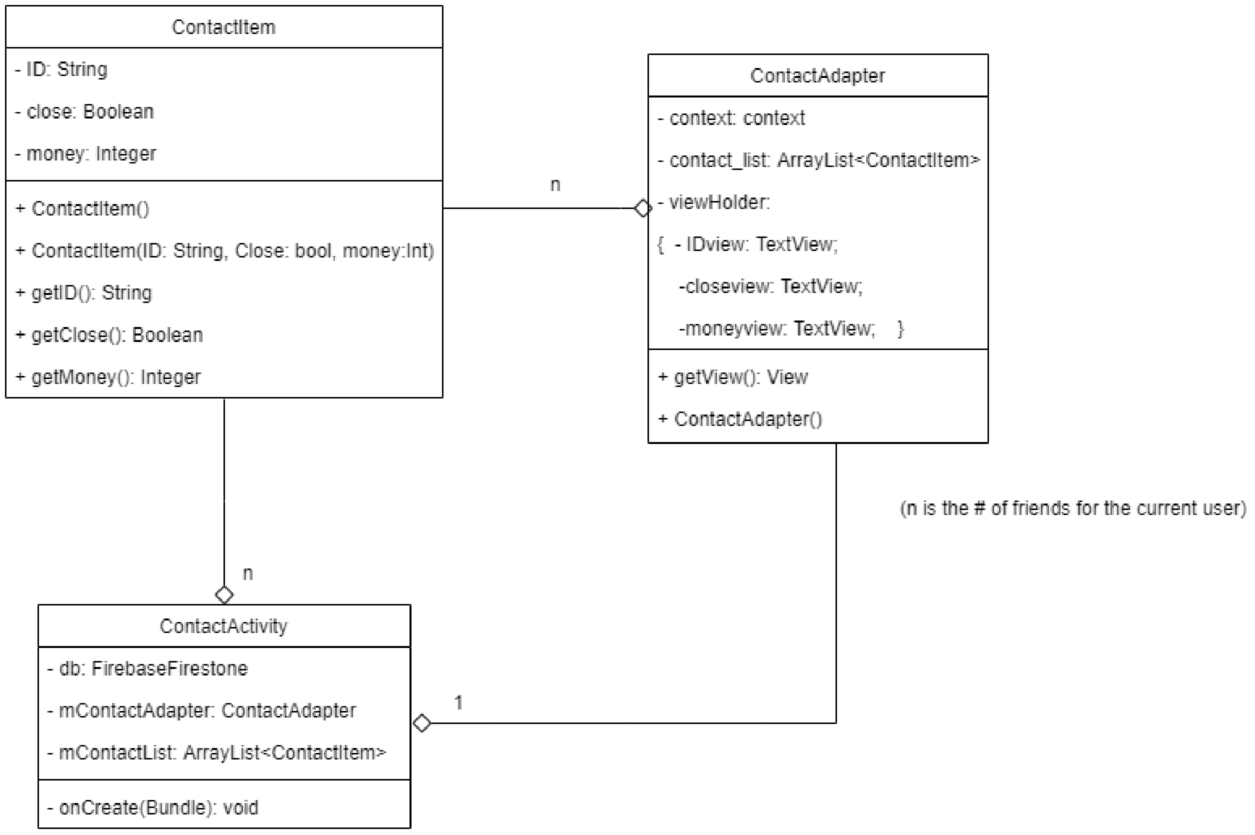
Search Bar- Search Activity and Result Activity



There are two classes associated with the Search Bar Module, Search Activity and Result Activity. In the Search Activity class, there is the ListView of usernames from the database which will update depending on the standard input. The interface for search activity contains the search bar and the dynamic ListView. Clicking the username on the list or clicking search button will bring user to the result activity which displays corresponding user portfolio page. The current user can see some information about the user that they searched up depending on their privacy settings. There are two functions in the Result Activity; one which can send the other user a Friend Request and the other a Close Friend Request. These are called by the addFriend and addClose functions respectively and will update the attribute values in database. This update is bidirectional; if a User A adds User B to their contact list, then User B's contact list will also have User A added to it

The layered architecture, the search bar, result portfolio page, and the add friend buttons are accessible for users on the presentation layer. On the business logic layer, the Search Activity class will adapt the ListView with the database collection for the user. Additionally, the Result Activity will update the database values by setting the click listener. The database layer is Firebase Server which contains the accessible contact collection and user collection.

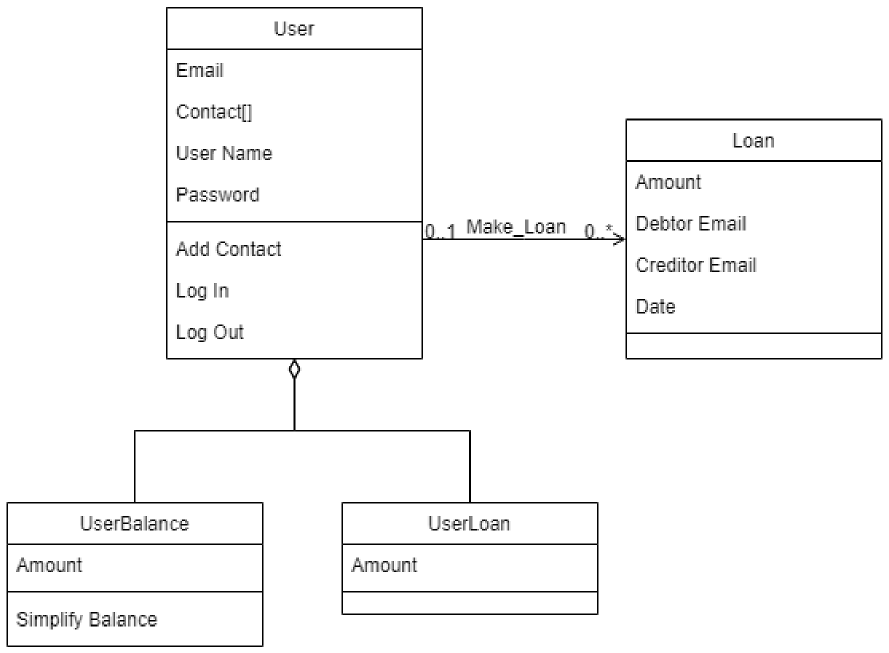
Contact List – Design Pattern



Since the Contact List is basically a social media platform, we created our design accordingly. There are three classes in the Contact List module, Contact Activity, Contact Item, and Contact Adapter. Each instance of the Contact Item class represents a specific friend of the current user. There are some attributes such as the friend's name, the settled debt to this specific friend, and if this friend is a close friend. To achieve encapsulation, these attributes are set to private and getters are provided for them. In the Contact Adapter class, the adapter (Decorator) design pattern was used with the view holder to wrap the data for the ListView to display. This class corresponding with the externally visible interface which displays each item on the ListView (Contact List). The Contact Activity get the current username from previous activity and updates all his or her friends' information from the database (Firebase Server) to the ListView using contact adapters. For each friend, there are two buttons for remove the friend from contact list and set off the close friend. For display, clicking the "Contact List" button from previous page will bring the user to his or her contact list.

The presentation layer contains the Contact List and user can their friends or close friends by clicking the button. The business logic layer corresponding to the Contact Activity which fetches their friend's information from the database and change the attribute values in database using click listener. The database layer is the Firebase server which contains the contact collection for all users and their friends' information.

Money Loan Page



The Money Loan page is the page where the user creates a money loan record in the application. Additionally, users can also check loan history on this page. Lastly, the user can choose to add new loan in this page. When user interacts with the UI to create a new loan, user will be asked to select the person in contacts who owe user money and be asked to select the amount of money owed. After the loan is created, there will be a message sent to debtor asking the debtor to confirm the loan. If the loan is confirmed, the user's and debtor's loan balance will change respectively. Users can view loan history to check past loans. Whenever a loan is confirmed, it should be shown in the loan history and user could be able to interact with It to view details of the loan: created time, debtor's name, debtor's email, amount, etc.

The Loan class has following attributes. The "Amount" is the value of the loan. The "Debtor Email" attribute shows the email of the loan debtor while the Creditor Email attribute shows the email of the loan creditor. Lastly the "Date" records when the loan happened (DD/MM/YY)

The User class has following attributes and operations. The "Email" is the registered email address of the user. The "Contact" array is the list of User who are added in the contact list. The "User Name" and "Password" are the login credentials that validate the user in question. The "Add Contact" function adds two users to each other's contact lists. The "Log In" function is used to validate the login credentials. Lastly, the "Log Out" function allows users to log out their account.

Each User object has a UserBalance and a UserLoan. The UserBalance has an attribute "Amount" which shows the amount of money user has and contains a function called "Simplify Balance" which rearranges balance among a web of users after a loan has been added to the system. The UserLoan has an attribute "Amount" shows the amount of money user owe.

Work Overview

Shanglin Ye:
Balance Page – The balances for the outstanding and owed balances are currently display the amount before any simplification and the next steps is to implement that logic. The following features have not been started on yet and will be worked on for in the following order: Ability for users to pay off debts for all users, view history of loans, pay off debt for single user and ability to manually enter debt payment.

Zijian Wang:
Login Page – Login and Sign Up components are mostly complete. The last feature that needs to be implemented is the account recovery feature.

UI – A preliminary design for the application has been built. As the application becomes more complete, the design will evolve along with the system.

Shuangyou Huang:
Money Loan Page – Users can add loan payment records for another user and the next step is to implement sending a notification so that the other user must accept the loan payment. Additionally, users are not able to view all their loans, which is the next item to be implemented.

Haoqing Tian:
Search Bar/Contact List – Currently, the search bar will only be able to find the other user if you enter their full ID. This needs to be changed to a dynamic list that finds users as the search input updates. The other user's contact page is not yet populated with information and currently only allows you to send a friend request. Lastly, the ability to other people to accept your friend request and the ability to remove someone from your friends list must be implemented.

All:
The Firebase Server is used by all modules and all group members must work together to maintain and update the database. Additionally, there are two modules that have not been worked on yet: The Wallet Module and the Settings Tab. Any group member can take on those modules as they complete their current work.