Student Bank

1Junwei Dai, 2Yeqian Liu, 3Xinyue Yuan, 4Zhifei Li

Software Engineering Systems, Northeastern University

{dai.junw, liu.yeq, yuan.xinyu, li.zhife}@northeastern.edu

# **Abstrac**

The purpose of the banking system is to provide an easy way for people to save their money, and transfer money.Our system faces not only clients but also admins. admin can manage the client’s account. Users can transfer money to others.

Our application has 6 screens. And we used Scene Builder to edit our FXML file.We used SQLite as our database and it contains 5 core tables. Transaction, Account management is our core functions. We use Agile development to work together.

The outcome of our work is excellent, we get a fully functional system. and can correctly and accurately make a deposit and transfer money to others.

***Bank System, SQLite, JavaFX, Userfriendly UI***

# **I. Problem Description**

A student bank system is a type of financial institution that is specifically designed to serve the financial needs of students. This can include various services, such as savings and checking accounts, loans, and credit cards. The goal of a student bank is to provide financial products and services that are tailored to the unique needs of students, such as low-interest rates on loans and flexible repayment options. These institutions may also offer educational resources and support to help students make informed financial decisions and manage their money effectively.

The system allows students to open a bank account, deposit and withdraw funds and access other financial services such as loans and credit cards. The system may also provide features such as mobile and online banking, budgeting tools, and educational resources to help students manage their money. The main goal of a student bank system is to provide financial support and resources to students to help them achieve their educational and financial goals.

This system typically offers a range of financial products and services, including checking and savings accounts, credit cards, and loans. These products and services are often tailored to the unique needs and financial circumstances of students, with features such as low fees, flexible repayment options, and the ability to establish a credit history. The goal of a student bank system is to provide students with the tools and resources they need to manage their finances effectively, while also helping them to build a solid foundation for their financial future.

# **II. Analysis (Related Work)**

A bank system is a financial institution that provides services such as accepting deposits, providing loans, and offering financial products such as investment options and credit cards. The performance and stability of a bank system are crucial for the overall health of an economy.

To analyze a bank system, we look at several key factors, including the bank's financial health, compliance with regulatory requirements, and ability to meet its customers' needs.

The financial health of a bank is typically measured by its capital adequacy, which is the amount of capital the bank holds in relation to its total assets. This is important because a bank with high capital adequacy is better able to withstand financial shocks and continue to provide services to its customers.

Another important factor to consider is the bank's compliance with regulatory requirements, such as the guidelines set by the national banking authority. A bank that is not in compliance with these requirements may be at risk of fines or other penalties, which can affect its financial health and reputation.

Additionally, it is important to assess the bank's ability to meet the needs of its customers. This includes factors such as the availability of different financial products and services, the ease of accessing accounts and conducting transactions, and the quality of customer service.

Examining a range of factors to evaluate the institution's financial health, compliance with regulations, and ability to meet the needs of its customers

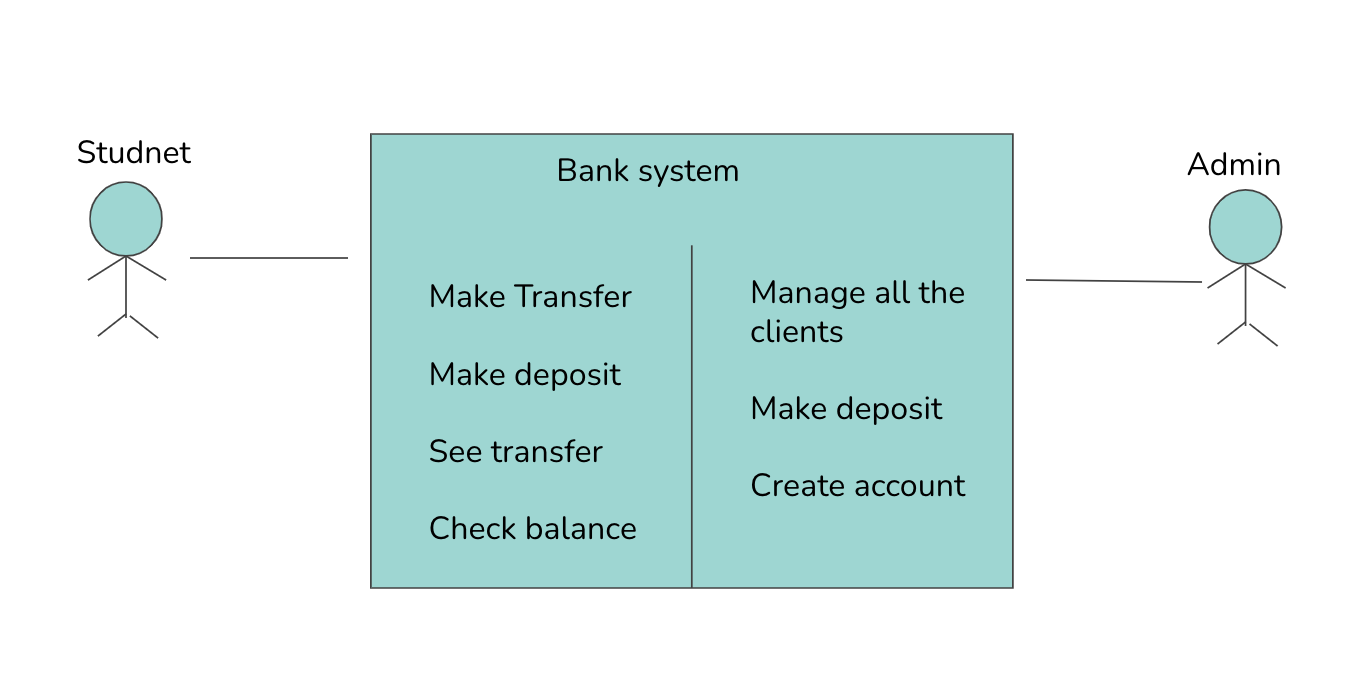


Figure 1. Domain Model

# **III. System Design**

A. *Algorithms*

Only administrators can add client accounts. Each user can only have one saving account and a checking account.

Administrators can transfer money to the client's saving accounts. The total amount between users is relatively stable.

Only saving accounts support transfers between different users. The saving account and checking account of the same user can transfer money to each other.

B. *MVC structure*

**Views** represent user interfaces, and with the complexity and scale of applications, the processing of interfaces becomes challenging. An application may have many different views, and the MVC design pattern limits the processing of views to the collection and processing of data on the view and the user's request and does not include the processing of business processes on the view. The processing of business process is handled by the model. In this project, the user's view only accepts data from the model and displays it to the user, as well as passes input data and requests from the user interface to the control and model.

**Model**: It is the processing of business processes/states and the formulation of business rules. The processing of the business process is a black-box operation for other layers, and the model accepts the data requested by the view and returns the final processing result. The design of the business model is arguably the most important core of MVC. It further divides the model from the perspective of application technology implementation in order to take advantage of existing components, but it cannot be used as a framework for application design models. It simply tells you that you can take advantage of certain technical components by designing according to this model, which reduces technical difficulties. For a developer, it is possible to focus on the design of the business model.

**Control** can be understood as receiving requests from users, matching models with views, and jointly completing user requests. The role of dividing the control layer is also obvious, it clearly tells you that it is a distributor, what model to choose, what view to choose, and what kind of user request to complete. The control layer does not do any data processing. For example, if the user clicks a connection, the control layer accepts the request and does not process the business information, it only passes the user's information to the model, tells the model what to do, and selects the view that meets the requirements to return to the user. Therefore, a model may correspond to multiple views, and a view may correspond to multiple models.The separation of models, views, and controllers allows a model to have multiple display views. If a user changes the model's data through the controller of a view, all other views that depend on that data should reflect those changes. Therefore, whenever any data change occurs, the controller notifies all views of the change, resulting in an update to the display. This is actually a model change-propagation mechanism. The relationship between models, views, and controllers and their main functions.

C. *Flow Chart*

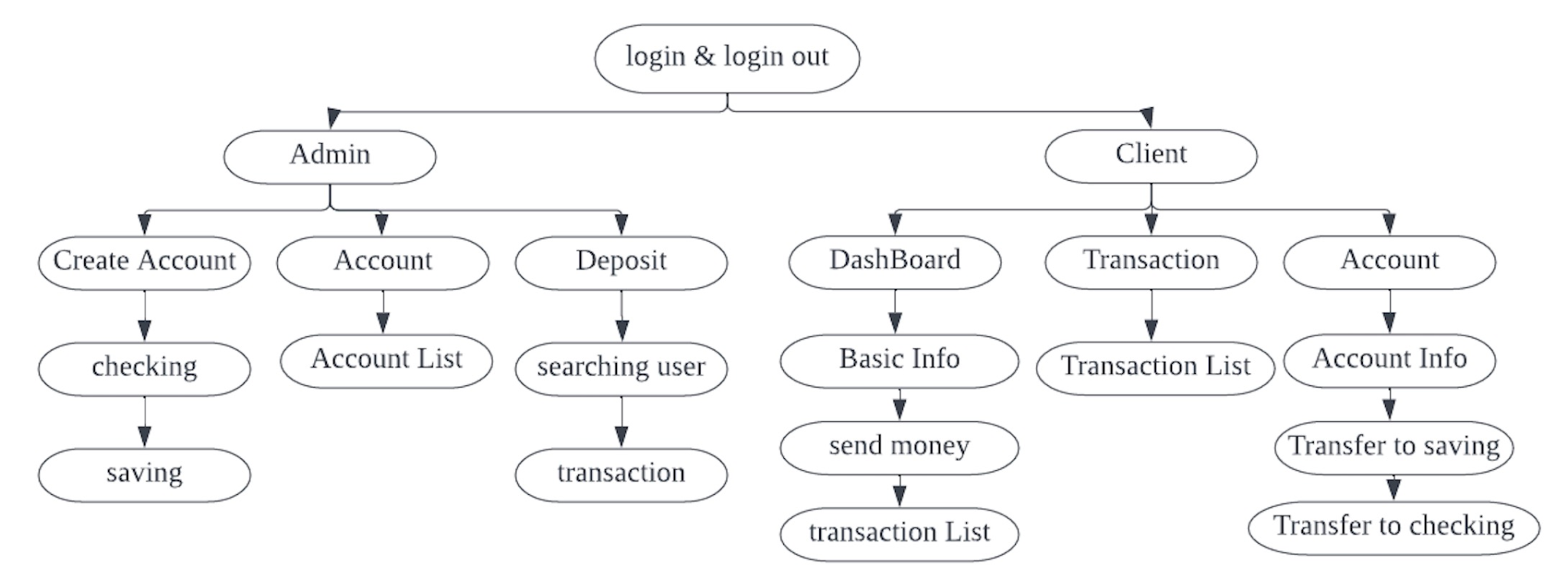


Figure 2. Flow Chart

# **IV. Implementation**

A.*Login*

The most important part of login is to evaluate the Admin/Client login credentials, from the input we get the pass 2 parameter: PayAddress and Password, and we pass it to the model layer, in the model layer, the date base driver use select \*from clients to get clients’ data. If they are matched, we have a login successful flag to proceed with the login. And also we can track the LoginAccountType when it is changed: when the user chooses admin it will be the username, when the user chooses client it will be Payee Address.When the user wants to log out, it will get to the stage, close the clients’ window, and return to the login window.

B.*create client*

In the controller layer we use fx\_id to get the info from input， add a listener to the check box, clicking the button is bound with the method in controller createClient.We add a listener to Address\_box, calling the createPayeeAddress() to generate the character and last Id+1, where we back to the model layer, since we can only get the last client’s Id from the database, so we plus one to it, then it will be this client’s Id.

We add a listener to Checking/Saving ch\_box, in createClient() we have to create Checking Account Flag to check whether the user selects the checking box.After the user finishes this page, we go back to the model layer, database Driver use “executeUpdate (SQL:” INSERT INFO”+..)to update the database, so we can insert the client’s information into the database.

C. *See all the clients*

Client pages: Admin can see all the client detail on this page. All the information is listed together. We separated our project into three layers. Controller, Models, Views. MVC is a software architectural pattern commonly used for developing user interfaces that divide the related program logic into three interconnected elements.Retrieve the data from the database.First, we go to the controller and initialize the page. we have a method called initClientList(); Then we go to the model layer. This is where we call the database driver to actually get the list.The database driver used this query “SELECT \* FROM Clients”. which interact with our database and retrieve all the data.if we got all the data then we go back to the controller layer to set Items. then we display all the lists.

D. *Admin makes deposit*

Deposit page: On this page, We need to implement search and add a deposit to the Account.we add a listener in the search button.On search action, we need to get the information that makes the separate transaction cell view through SceneBuilder. When initializing this page, ListView controls will use setItems() method to call getAllTransactions() which will retrieve data from the database. When a client clicks the bell icon, it will new a scene object and present a transaction message. Also when a client on the Dashboard page clicks the Send Money button, this ListView will reload data from the database, and the transactions page will refresh.

E. *Transactions*

This page will present the client’s transaction details and transaction message.First, we define Transaction Class and make the separate transaction cell view through SceneBuilder.When initializing this page, ListView controls will use setItems() method to call getAllTransactions() which will retrieve data from the database.When a client clicks the bell icon, it will new a scene object and present a transaction message. Also when a client on the Dashboard page clicks the Send Money button, this ListView will reload data from the database, and the transactions page will refresh.

F. *Accounts*

This page will present the client’s account details. It includes Checking Account, Savings Account, and Transfer functions between Savings and Checking.When initializing this page, account details bound with fx: id will retrieve the data from the savings and checking database.The client could input money value in the TextField and click the button. After clicking the button, it will call methods to update the balance data in the database. Then use setBalance() method to reset the balance number and show the latest data.

G. *See saving balance*

From the card, users can check their saving account and checking account in the dashboard. First, we go to the controller and initialize the page. we have a method called bindData(). Then we go to the model layer. This is where we call the databaseDriver to actually get all the data. The database driver used this query “SELECT \* FROM SavingAccounts”. which interact with our database and retrieve all the data. We use the same way to get the data in the CheckingAccounts DB. If we got all the data then we go back to the controller layer to setItems . then we display the data.

We can also get the income and expenses on this page. Use accountSummary() method to bind data in the current controller. We get all user transactions and add the num to get the result. The database used this query“SELECT \* FROM Transactions” to retrieve data.

H. *Send money to other*

Users can send money to others by pAddress. We have a method called onSendMoney() in current Controller. Invoked the searchClient() method in databaseDriver directly and used UPDATE SavingAccounts SET statement to update both current user and payee. Then we go to the model layer. This is where we call the databaseDriver to actually get all the data. Invoked the UpdateTransections() and updateExpense() methods to keep the data correct.

Set the limitation as 4. The databaseDriver used this query “SELECT \* FROM Transactions”. which interact with our database and retrieve all the data. If we got all the data then we go back to the controller layer to setItems . then we display 4 pieces of data.

I. *Database*

SQLite is an in-process library that implements a serverless and zero-configuration SQL database engine. The code for SQLite is in the public domain and is free for use for any purpose, commercial or private.

SQLite requires no configuration, which means no installation or management. Also, it is very small and lightweight, less than 400KiB when fully configured, and less than 250KiB when optional function configuration is omitted.

We use DB Browser for SQLite as a visualization tool to show data. Here are the related tutorials of SQLITE：https://www.knowledgedict.com/tutorial/sqlite-intro.html

We have 6 collections in DB: Admins, CheckingAccounts, Clients, SavingsAccounts, Transactions, and sqlite\_sequence.

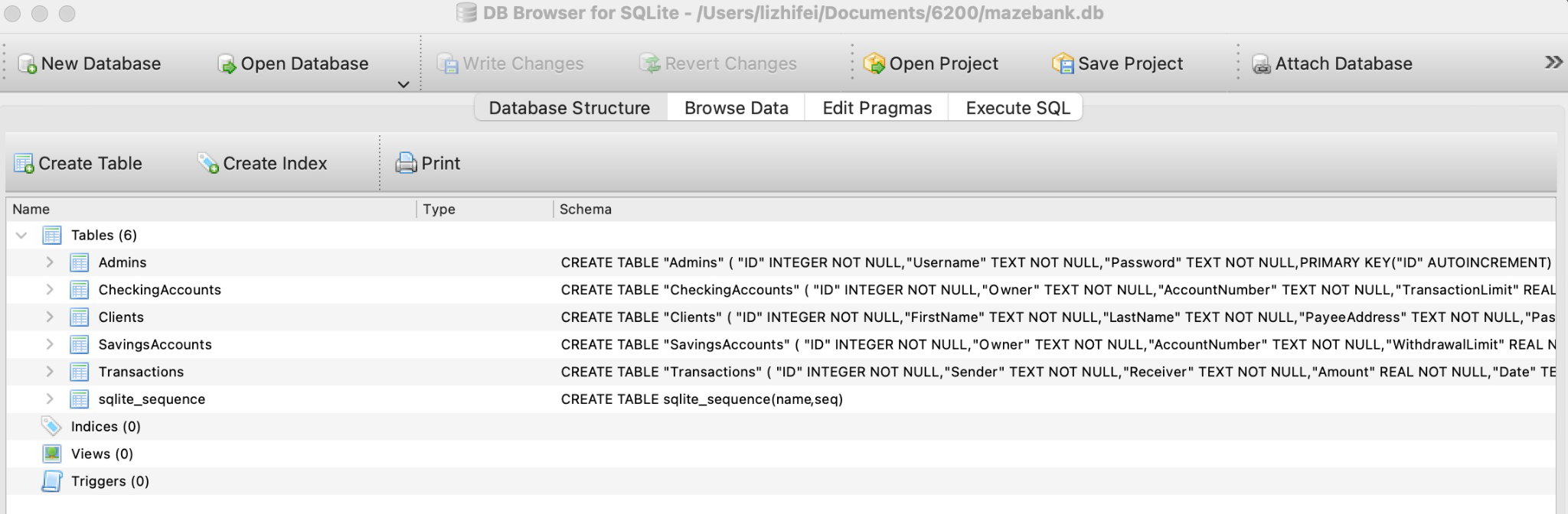


Figure 3. Database Tables

# **V. Evaluation**

* Screenshots
* Login

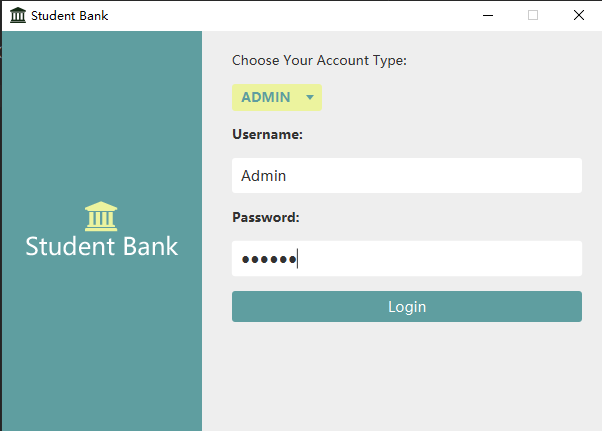


Figure 4. Login Page

* Create New Client

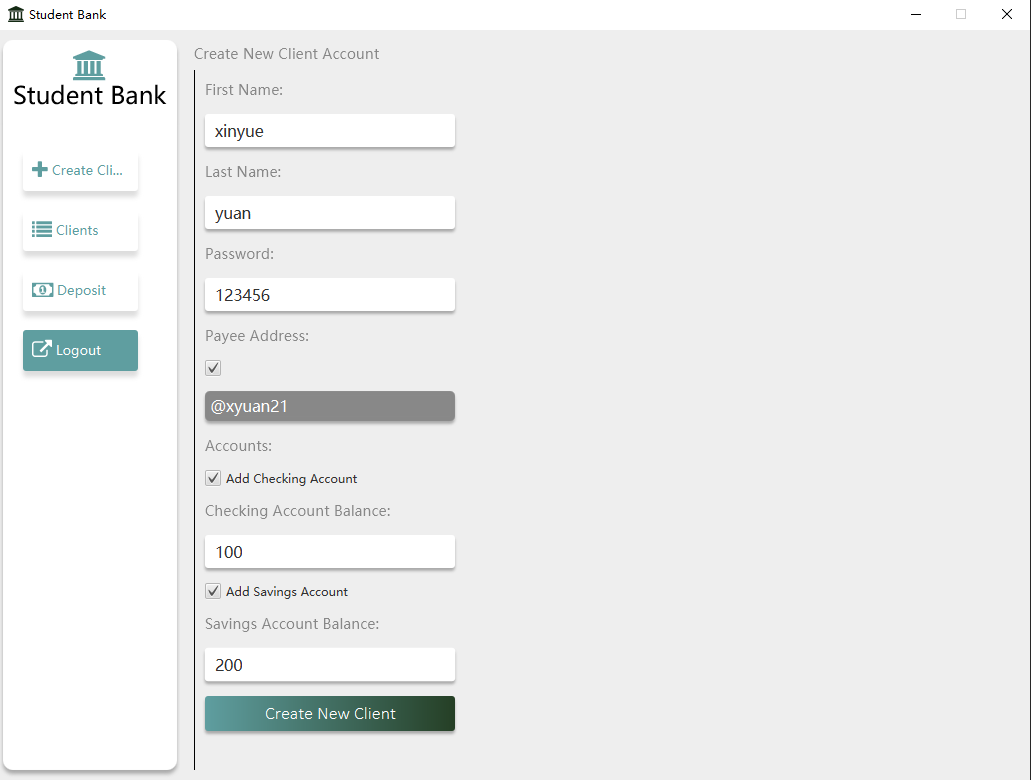


Figure 5. Create New Client Page

* Admin - View Clients List

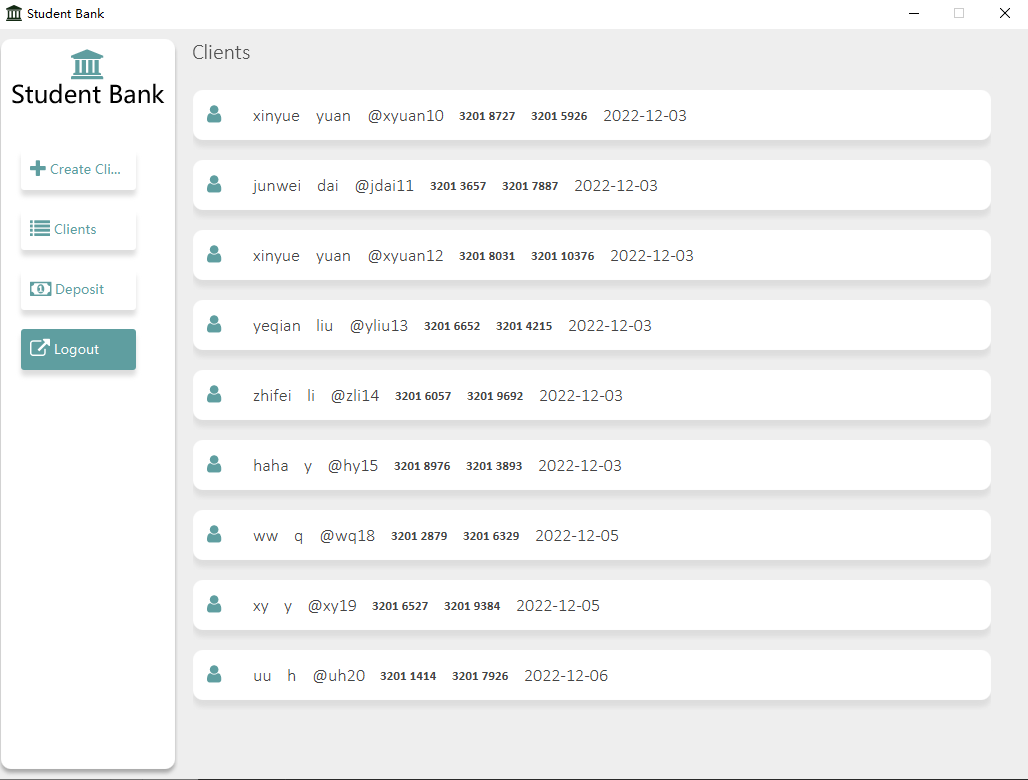


Figure 6. Clients List Page

* Admin - Deposit

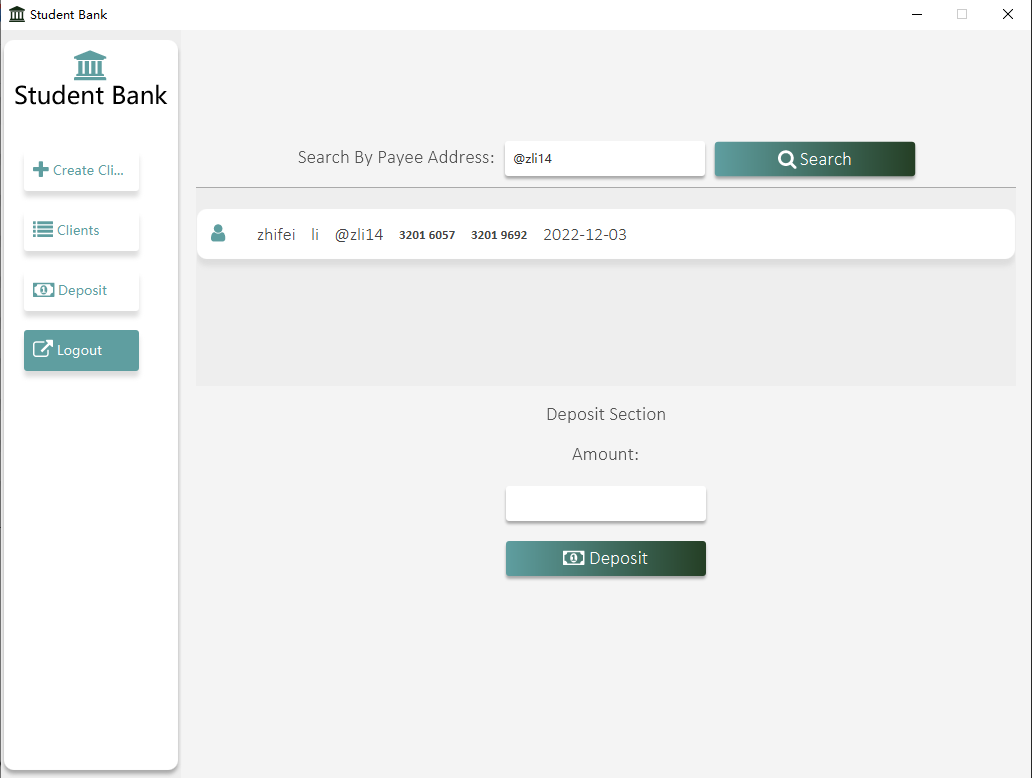


Figure 7. Deposit Page

* Client - Dashboard

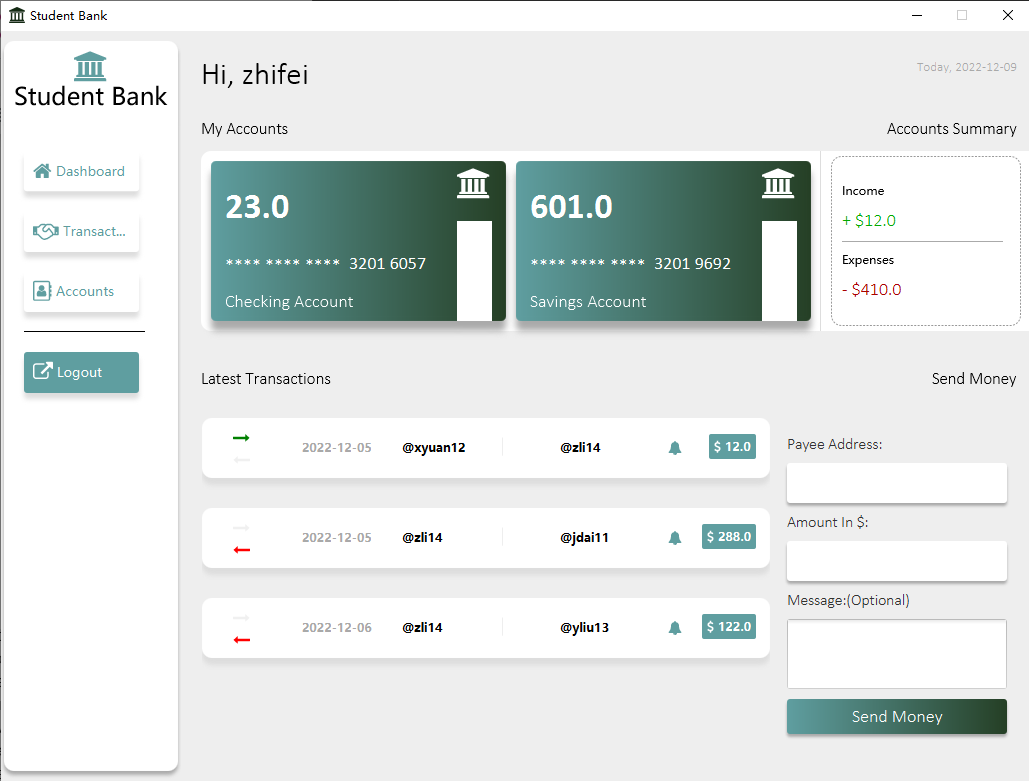


Figure 8. Client Dashboard Page

* Client - Transactions

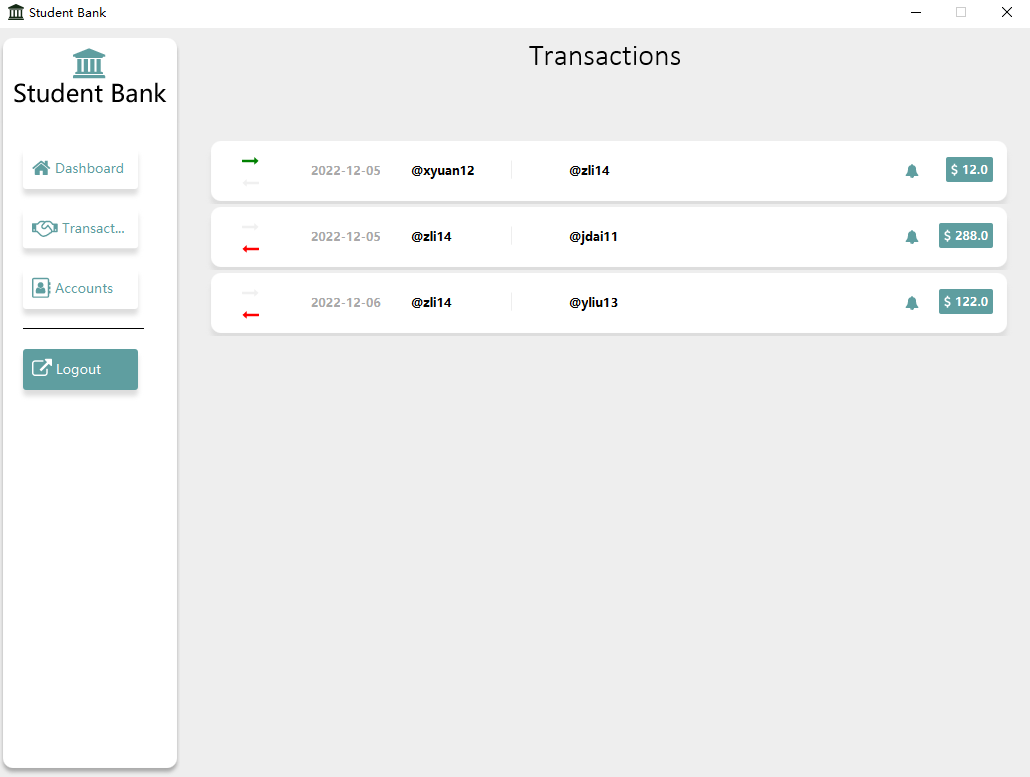


Figure 9. Transactions Page

* Client - Accounts

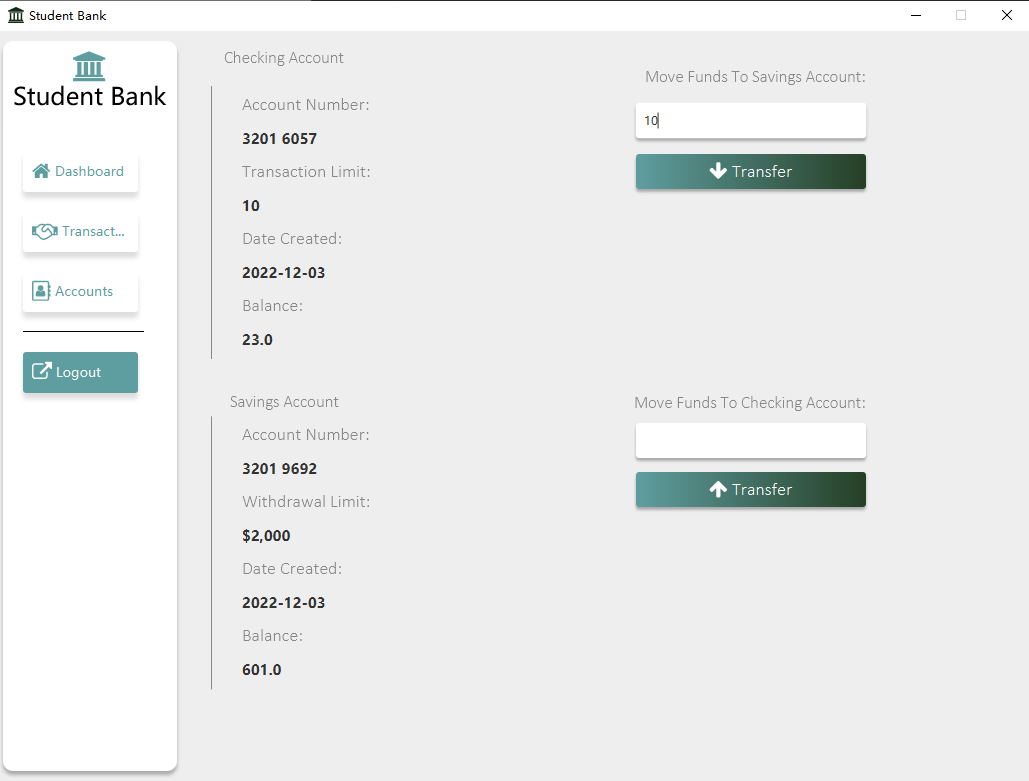


Figure 10. Client Account Page

* Comparison

Compared with popular bank applications like Bank of America and Chase. Here are our application’s same and different parts:

Same: Both of our application has both user and admin login screens. We both check the legitimacy of usernames and passwords. Both of our user modules have the functions of checking bank card information, showing balance, transferring money to other clients, displaying transactions, and transferring money between savings and checking cards. Both of our administrator modules have the functions of adding new users, displaying all users, and user deposits.

Difference: The popular bank applications would also show the consumption records and client profile page.

* User study

We invited 10 users to use our app, 80% of users think our app is very interesting and useful, and 30% of users think our app can be improved.

# **VII. Discussion (Reflection)**

* Functions

Although our application realizes the core functions of the bank, there are still some functions that we can continue to improve in the future, such as the user's profile page, customer service help page, and display consumption record page.

* UI

We can better display our application by setting the CSS style sheet.

* DataBase

We can choose more efficient database management software like MySQL, optimize the table structure, and pay more attention to data security.

* Framework

We can use a more efficient framework to unify our development style and improve development efficiency, such as SpringBoot.

# **VIII. Conclusions and Future Work**

*Conclusion:* During this project, it has been determined that the front-end is completed using Scene Builder, we first proposed to use the MVC architecture to complete the back-end work, and in the implementation process, we have a clearer understanding of the advantages of JavaFX, and we have a deeper understanding of the framework structure of MVC. For example, in the design of the Controller layer, we introduced inheritance, interfaces, and other important knowledge learned in this course so that we have a more thorough understanding of the basic concepts of the Java language.

In our experiments we found various advantages of JavaFX: To develop client applications with rich features, programmers used to rely on various libraries to add features such as media, UI controls, Web, 2D, and 3D. JavaFX includes all these features in a single library.

In addition to these, developers can also access existing features of Java libraries, it provides a rich set of graphics and media APIs and leverages modern graphics processing units with hardware-accelerated graphics. We can easily drag elements, and JavaFX also provides interfaces that developers can use to combine graphical animations and UI controls.We use Scene Builder as a scene generator when integrating this application into the IDE, and we have access to a drag-and-drop design interface for developing FXML applications.

In the backend we use MVC structure, the MVC design pattern tells us that the application model is extracted according to certain rules, and the level of extraction is very important, which is also the basis for judging whether the developer is a good designer. Abstraction and concrete cannot be separated too far apart, or too close.

MVC does not provide a design method for models but only tells you that you should organize and manage these models in order to facilitate model refactoring and improve reuse. To use the analogy of object programming, MVC defines a top-level class and tells its subclasses that you can only do this, but there is no limit to what you can do. This is very important for programming developers.

*Future work:*  Although this project can be put into use easily, it still has some places to improve.

* The main function of this project is the transfer between users and has realized the operation of user addition, modification, query, etc., and then prepare to add the administrator to delete the user account.
* Use data analysis and model building,, mobile banking App operation precipitated a large number of user data, in addition to asset stratification, banks can also further segment users according to users' age, behavior, scene preferences, habits, and other dimensions. This requires banks to have strong enough data analysis and user model-building capabilities, which is the basis for user stratification. We are ready to add financial information to classify each transfer of users so that users have a deeper understanding of their spending income.

# **IX. Job Assignment**

In this project we mainly divide the project into four modules, client and administrator login, the administrator creation account, client main interface, administrator to set the client's savings amount, client main interface, client transfer interface (including the saving and checking account transfer of my account, and transfer with others), client transfer record interface, and customer account balance.

Each member of the group is responsible for the editing of their own part FXML files, the adjustment of CSS format, and the implementation of the underlying logic and user interaction at the controller, model, and view layers.

* Yeqian Liu: determine the type of user is client or admin, achieve login and design admins’ page,the admin has access to create the clients with Payee Address and password, saving account, and checking account.
* Junwei Dai: admin shows the clients list and basic info, searches users, and makes a deposit to clients, fixes data-grabbing bugs. designed the deposit page and main clients page, and fix search bugs.
* Xinyue Yuan: Implemented the client account display page, savings and checking bank card information display page, the transfer function between savings and checking accounts, and the transactions page to show all records.
* Zhifei Li: Implement a dashboard page that shows basic information about the user checking account and saving account, and total inbound and outbound amounts. Transfer to other saving accounts and display the latest 4 transfers.

##### **References**

1. Shu-Qiang Huang, Huan-ming Zhang, ” Research on Improved MVC Design Pattern Based on Struts and XSL”, in Information Science and Engineering ISISE 08 International Symposium on, 2008, vol. 1 PP. 451 – 455
2. Juanjuan Yan; Bo Chen; Xiu-e Gao, "Le Wang; Research of Structure Integration Based on Struts and Hibernate" , in 2009 WRI World Congress on Computer Science and Information Engineering,2009, vol. 7, PP. 530-534
3. Wojciechowski, J.; Sakowicz, B.; Dura, K.; Napieralski, A., "MVC model, struts framework and file upload issues in web applications based on J2EE platform", in Proceedings of the International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science 2004,
4. Wang Ning; Li Liming; Wang Yanzhang; Wang Yi-bing; Wang Jing, "Research on the Web Information System Development Platform Based on MVC Design Pattern", in IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, 2008, Vol 3, pp. 203-206
5. Sheard P. The main bank system and corporate monitoring and control in Japan[J]. Journal of Economic Behavior & Organization, 1989, 11(3): 399-422.
6. Aoki M. Monitoring Characteristics of the Main Bank System: An Analytical and[J]. The Japanese main bank system: Its relevance for developing and transforming economies, 1995: 109.
7. Hayashi F. The main bank system and corporate investment: An empirical reassessment[J]. 1997.
8. Wang Ning; Li Liming; Wang Yanzhang; Wang Yi-bing; Wang Jing,"Research on the Web Information System Development Platform Based on MVC Design Pattern",in IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, 2008 , Vol 3,pp. 203-206 .