**NUSmoney:** Personal Financial Management APP- Project DocumentationDone By: Erwin Lo, Yeo Theng Hee & Lin Zhenyao

1. Problem Background

We have been appointed as chief software architect for NUS Money app. We are responsible for the following:  
(a) Developing a database schema;  
(b) Populating the database with random data;  
(c) Documenting Create read Update Delete (CRUD) APIs to expose info from the database;  
(d) Implementing the APIs to expose the data from database onto listed URIs; and  
(e) Connecting front-end client with back-end service.

2. Program Objectives

To develop an app to enable users to have a good snapshot of their financial position **at any point in time.** To achieve this primary objective, we have provided to users the following functionalities:  
(a) A summary of their financial accounts, be it savings, current, fixed deposit or investment at banks or securities firms, and also CPF accounts.  
(b) the option to manually add accounts from financial institutions, that are not yet linked into our system;  
(c) the option to manually edit any account details that have been erroneously added previously;   
(d) the option to delete obsolete accounts; and  
(e) the option for the user to edit personal details (email and mobile number), as well as their password for logging into the app.

3. Database Structure

We have come up with the following tables and also populated the tables with random data, done with tools like Mockaroo.com and also manual editing. Database schema is imported using mySQL workbench, and ER diagram documented accordingly.

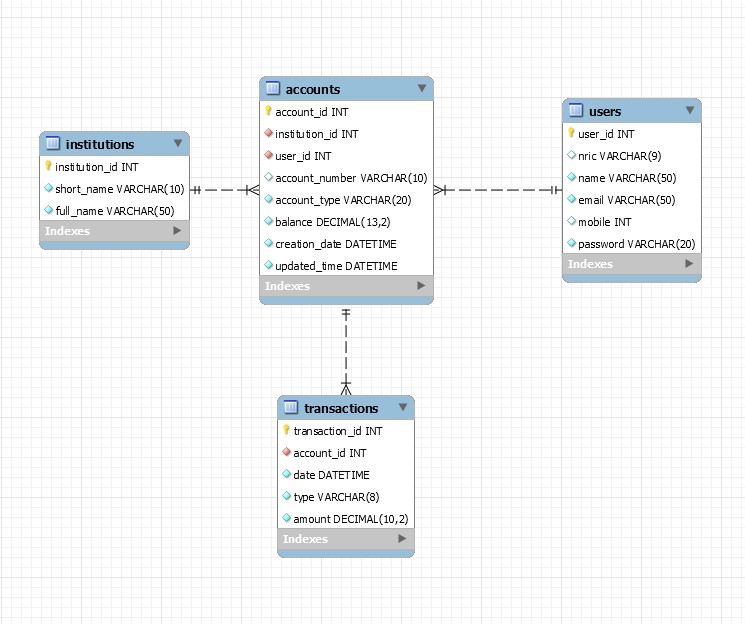


Table: Users  
Organisation: Indexed sequential  
Primary Key: User\_id

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Field Description** | **Field Type** | **Field Length** | **No of Dec** |
| User\_id | Auto incremented each time a new user is added | INT |  |  |
| NRIC | NRIC | VARCHAR | 9 | - |
| name | Name of user | VARCHAR | 50 | - |
| email | Email address | VARCHAR | 50 |  |
| mobile | Mobile number | INT | - | - |
| password | Password of user | Character | 20 | - |

Table: Accounts  
Organisation: Indexed sequential  
Primary Key: account\_id

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Field Description** | **Field Type** | **Field Length** | **No of Dec** |
| account\_id | Auto incremented each time a new account is added | INT | - | - |
| institution\_id | Identifier of financial insto | INT | - | - |
| user\_id | Owner of account | INT | - | - |
| account\_number | Account no | VARCHAR | 10 | - |
| account\_type | Account Type- Savings, Current, Investment, Fixed Deposit, Ordinary Account, Medisave Account and Special Account | VARCHAR | 20 | - |
| balance | Account balance | Decimal | 13 | 2 |
| creation\_date | Date of account creation | Datetime | - | - |
| updated\_time | Timestamp | Datetime | - | - |

Table: transactions  
Organisation: Indexed sequential  
Record Key: transaction\_id

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Field Description** | **Field Type** | **Field Length** | **No of Dec** |
| transaction\_id | Auto incremented each time a new transaction is added | INT | - | - |
| account\_id | To link back to Accounts | INT | - | - |
| date | Transaction Date | DateTime | - | - |
| type | Transaction Type | VARCHAR | 8 | - |
| Amount | Transaction amount | Decimal | 10 | 2 |

Table: institutions

Organisation: sequential

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Field Description** | **Field Type** | **Field Length** | **No of Dec** |
| institution\_id | Auto incremented each time a new bank is added | INT | - | - |
| short\_name | Bank name- short form | VARCHAR | 10 | - |
| full\_name | Full name of bank | VARCHAR | 50 | - |

4. API Specification

|  |  |
| --- | --- |
| **API Name:** | **banks.js** |
| Description: | to show summary of bank accounts for the user |
| Method: | GET |
| URI: | /banks/:userID |
| Headers/ Parameters/ Body: | Include userID in the URI |
| (Positive) Response: | {  “account\_id”: 1,   “Institution\_id”: 2,   “short\_name”: “UOB”,  “account\_type”: “Savings”,  “account\_number”: “4721475626”,  “balance”: 9900 } |
| (Error) response: | “Id not found.” |

|  |  |
| --- | --- |
| **API Name:** | **investments.js** |
| Description: | to show summary of investment accounts for the user |
| Method: | GET |
| URI: | /investments/:userID |
| Headers/ Parameters/ Body: | Include userID in the URI |
| (Positive) Response: | {  “account\_id”: 65,   “Institution\_id”: 9,   “short\_name”: “UOBKH”,  “account\_type”: “Investment”,  “account\_number”: “4776812075”,  “balance”: 57262.7 } |
| (Error) response: | “Id not found.” |

|  |  |
| --- | --- |
| **API Name:** | **cpf.js** |
| Description: | to show summary of CPF accounts for the user |
| Method: | GET |
| URI: | /cpf/:userID |
| Headers/ Parameters/ Body: | Include userID in the URI |
| (Positive) Response: | {  “account\_id”: 119,   “Institution\_id”: 11,   “short\_name”: “CPF”,  “account\_type”: “Special Account”,  “account\_number”: “6403968528”,  “balance”: 68267.07 } |
| (Error) response: | “Id not found.” |

|  |  |
| --- | --- |
| **API Name:** | **transactions.js** |
| Description: | to show list of transactions for the user |
| Method: | GET |
| URI: | /transactions/:userID |
| Headers/ Parameters/ Body: | Include userID in the URI |
| (Positive) Response: | {  "date": "2020-12-19T00:12:00.000Z",  "short\_name": "Citi",  "account\_type": "Current",  "account\_number": "5761644686",  "type": "credit",  "amount": 667.33  } |
| (Error) response: | “Id not found.” |

|  |  |
| --- | --- |
| **API Name:** | **accounts.js** |
| Description: | to allow user to add a new account manually |
| Method: | POST |
| URI: | /accounts/:userID |
| Headers/ Parameters/ Body: | Body:  {  “insti\_id”: 1,  “ac\_number”: “024404226”,  “ac\_type”: “Savings”,  “balance”: 9000 } |
| (Positive) Response: | “Data added successfully” |
| (Error) response 1:  (Error) response 2:  (Error) response 3: | "Error! Institution is blank"  "Error! Account balance cannot be blank or negative"  "Error! Account type must be one of 'Savings', 'Current', 'Fixed Deposit', 'Investment'" |

|  |  |
| --- | --- |
| **API Name:** | **accounts.js** |
| Description: | to allow user to edit account details manually |
| Method: | PUT |
| URI: | /accounts/:userID |
| Headers/ Parameters/ Body: | Body:  {  “insti\_id”: 1,  “ac\_number”: “135793579”,  “ac\_type”: “Savings”,  “balance”: 9000 } |
| (Positive) Response: | “Data updated successfully” |
| (Error) response 1:  (Error) response 2:  (Error) response 3: | "Error! Account No is blank"  "Error! Account balance cannot be blank or negative"  "Error! Account balance needs to be numeric without any special characters" |

|  |  |
| --- | --- |
| **API Name:** | **accounts.js** |
| Description: | to allow user to delete obsolete accounts |
| Method: | DELETE |
| URI: | /accounts/:userID |
| Headers/ Parameters/ Body: | Include userID in the URI |
| (Positive) Response: | “You have successfully deleted account ID ” + <account\_number> |
| (Error) response: | 'Id not found.' |

|  |  |
| --- | --- |
| **API Name:** | **users.js** |
| Description: | to get the user’s personal details |
| Method: | GET |
| URI: | /users/:userID |
| Headers/ Parameters/ Body: | Include userID in the URI |
| (Positive) Response: | {  "name": "Victor Adelsberg",  "email": "vadelsberg1@blog.com",  "mobile": 85729105  } |
| (Error) response: | “Id not found.” |

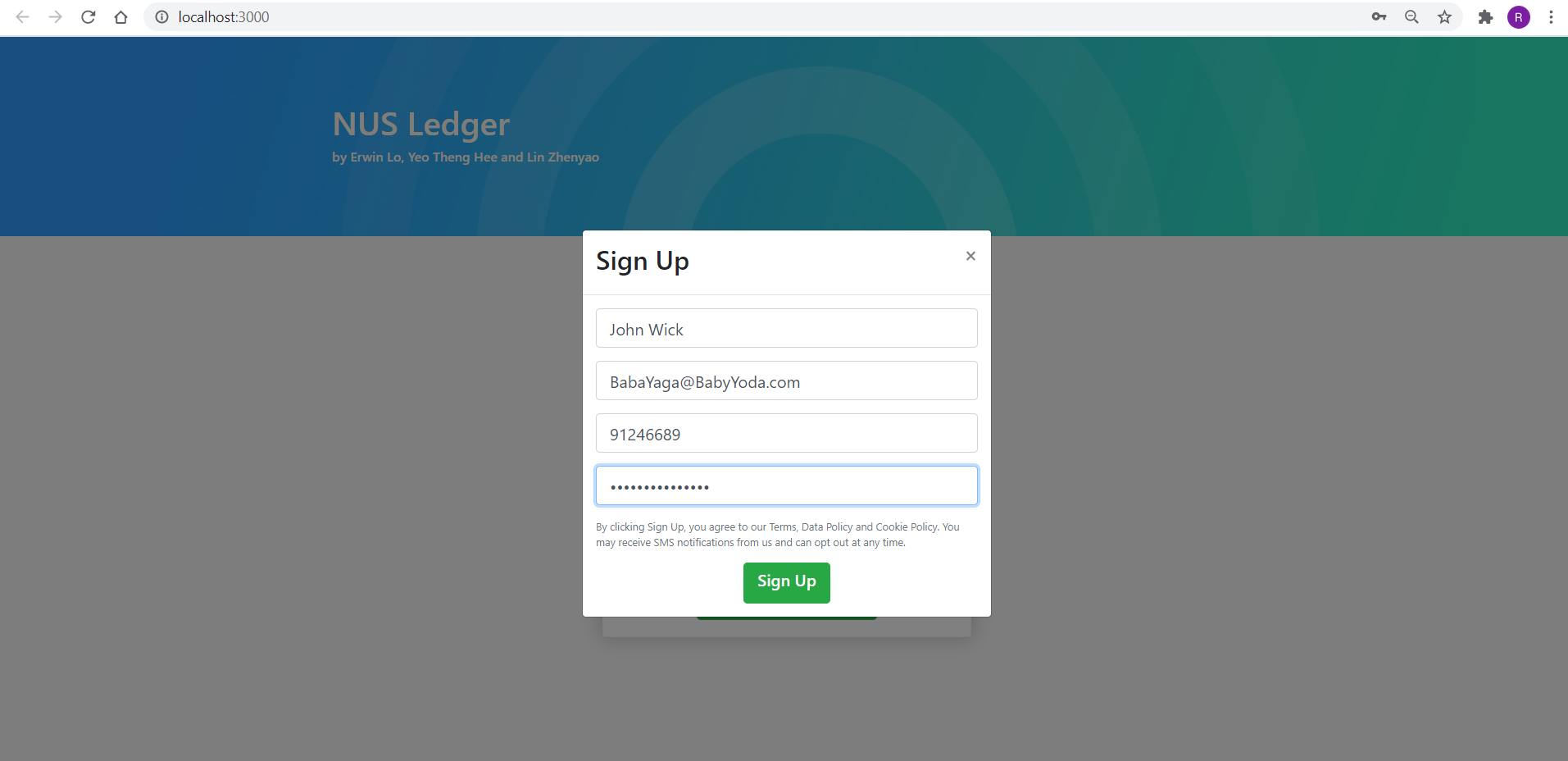
|  |  |
| --- | --- |
| **API Name:** | **email.js** |
| Description: | to allow user to edit his or her email address |
| Method: | PUT |
| URI: | /email/:userID |
| Headers/ Parameters/ Body: | Body:  {  “email”: “mysite@ourearth.com” } |
| (Positive) Response: | “Email updated successfully” |
| (Error) response: | 'Error! Your email id seems to be invalid.' |

|  |  |
| --- | --- |
| **API Name:** | **mobile.js** |
| Description: | to allow user to edit his or her mobile number |
| Method: | PUT |
| URI: | /mobile/:userID |
| Headers/ Parameters/ Body: | Body:  {  “mobile”:96969696 } |
| (Positive) Response: | “Mobile number updated successfully” |
| (Error) response: | 'Error! Mobile number needs to be an 8-digit number.' |

|  |  |
| --- | --- |
| **API Name:** | **password.js** |
| Description: | to allow user to edit his or her password |
| Method: | PUT |
| URI: | /password/:userID |
| Headers/ Parameters/ Body: | Body:  {  “oldPassword”: “xxxxxx”,  “newPassword”: “yyyyyy”,  } |
| (Positive) Response: | “Password updated successfully” |
| (Error) response 1: (Error) response 2:  (Error) response 3: | 'Error! Old Password is blank.'  'Error ocurred while sending request.'  'Error! old password does not match record' |

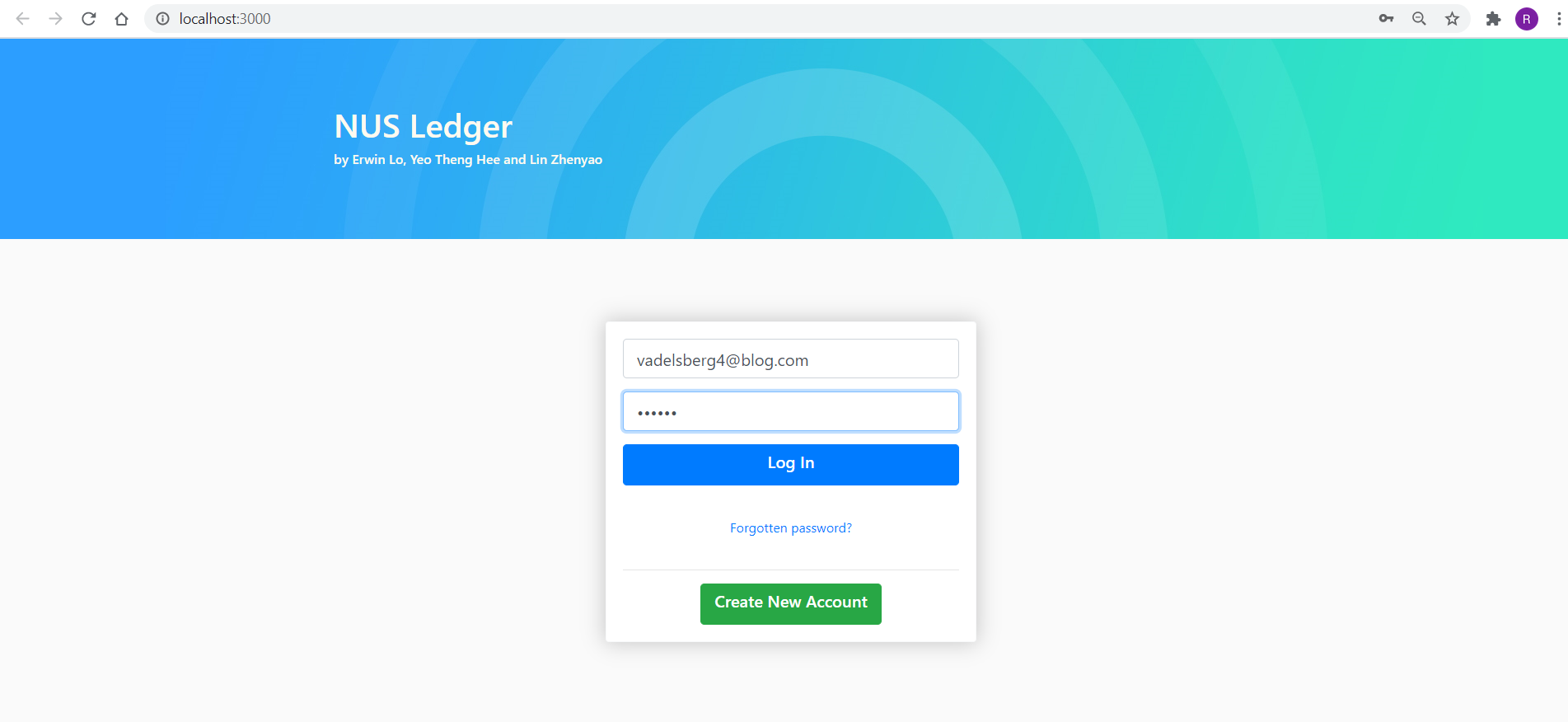
|  |  |
| --- | --- |
| **API Name:** | **users.js** |
| Description: | to allow user to be deleted |
| Method: | POST |
| URI: | /users |
| Headers/ Parameters/ Body: | Body:  {  “name”: “Evelyn Lau”,  “email”: “mysite@ourearth.com”,  “mobile”:96969696  “password”: “abcdefg” } |
| (Positive) Response: | “User saved successfully.” |
| (Error) response 1:  (Error) response 2:  (Error) response 3: | 'Error! Name is blank.'  'Error! Invalid Email format.'  'Error! Mobile Number is blank.' |

5. Program Flow/ Interface

Upon execution of npm start on the backend and frontend in visual studio, we arrive at the login page where users can create an account or log in if they are existing users. The following screenshot is the form for registering as new user.  
****

**Fig 5.1- Login Page (Sign Up form)**

For a thorough example, we will use an existing user account login to display the information in the database.

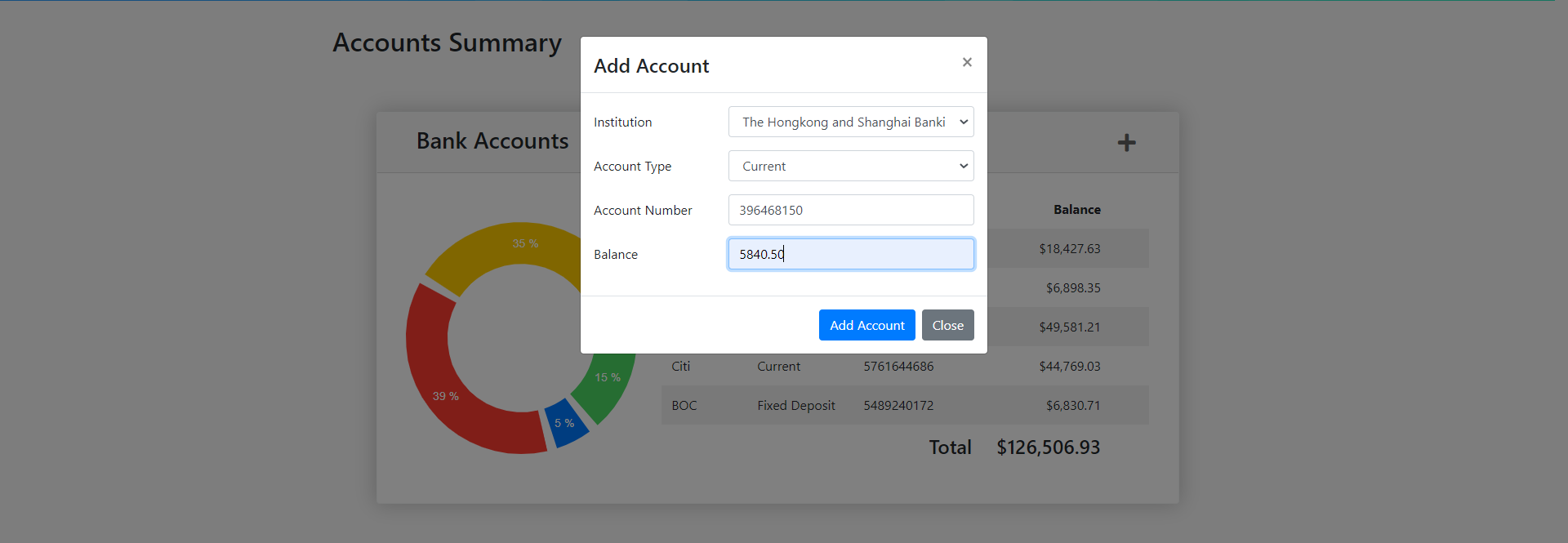


**Fig 5.2- Login Page (for existing user)**

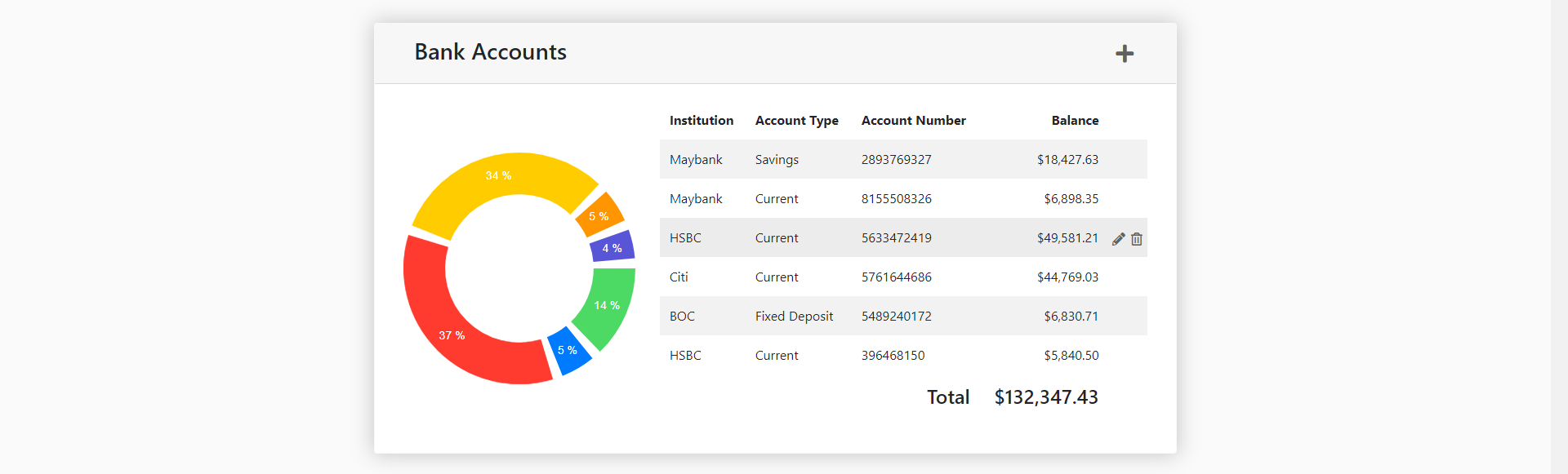
After login to the account, the user arrives at the main page where he/she can check on the bank accounts he/she has across the institutions, and if he hovers over any pie segment, a tooltip appears showing the bank, account type and balance for that segment.

|  |  |
| --- | --- |
| **Fig 5.3- Main Page (Bank Accounts)** | **Fig 5.4- Tooltip feature** |

The user can add account to the list of accounts which he/she wants to keep track of by pressing on the + sign and fill up the form. After successful addition, newly included account appears on the last line of the accounts list. Each line has an edit or delete button when user hover the mouse over the line that he/she wants to amend or remove.

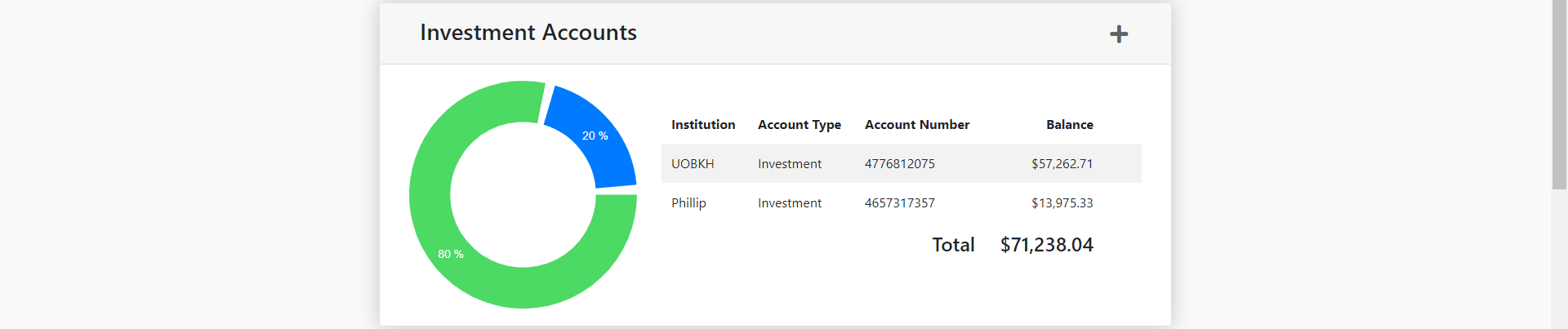


**Fig 5.5- Add Account window**



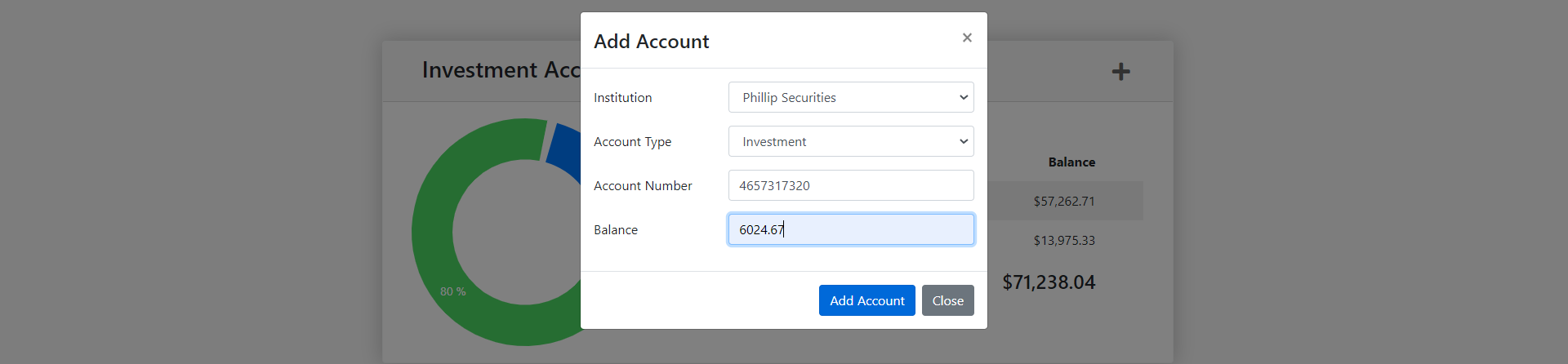
**Fig 5.6- Bank account successfully added**

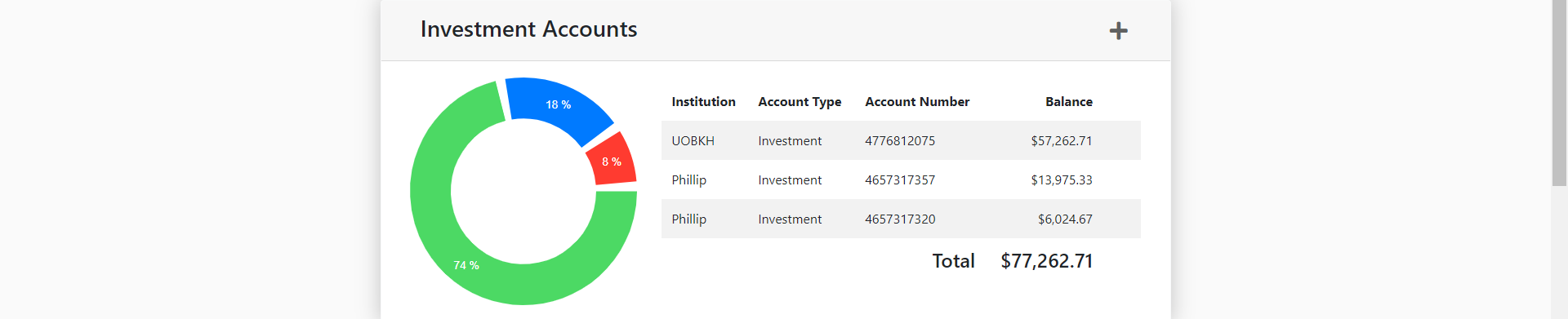
Next is the investment accounts list where users can keep track of the amount and percentage of investments, they have with the investment institutions.



**Fig 5.7- Main Page (Investment Accounts)**

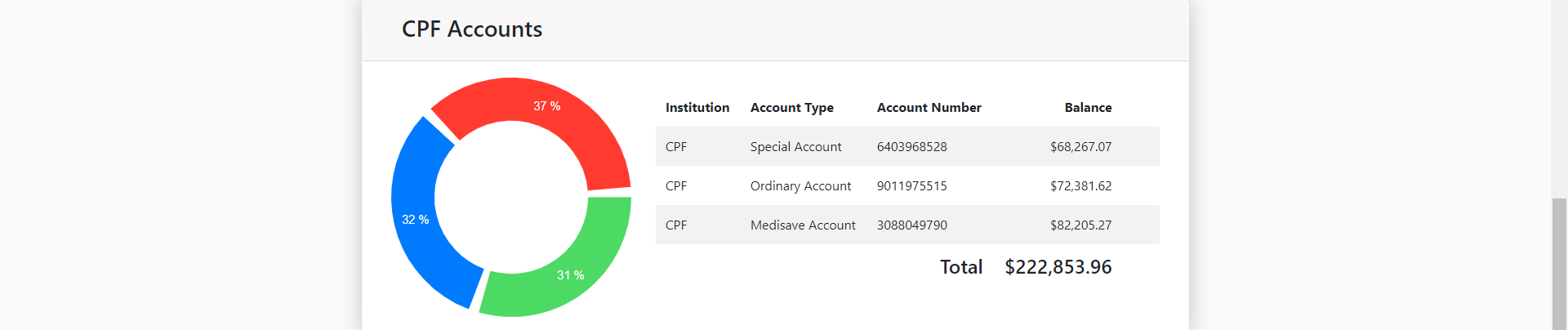
Similarly, the investment list also has the add account function for users to manually insert any investment to the list. After successful addition, the pie chart has a new portion, and the sum of investment amount is different from the initial value.

  
**Fig 5.8- Add Account window**



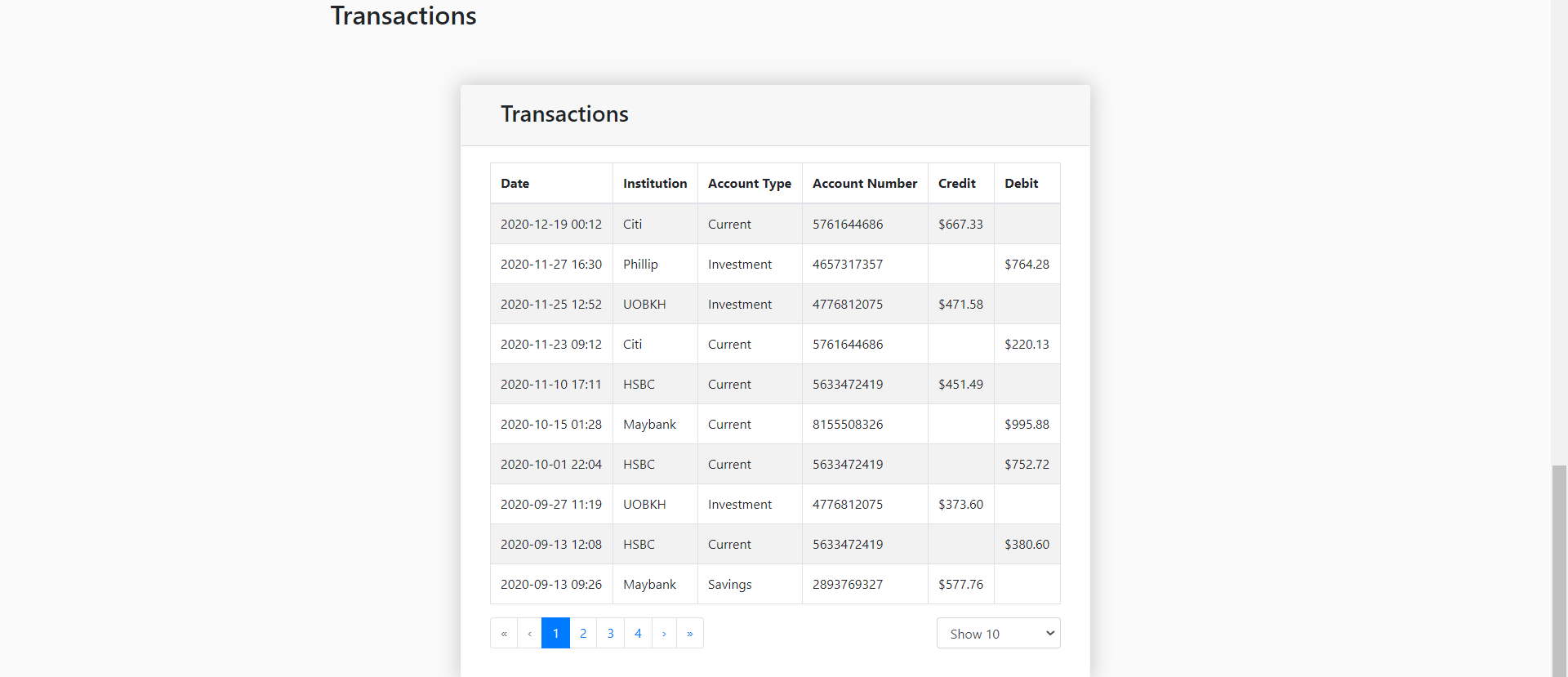
**Fig 5.9- Investment account successfully added**

The following screenshot shows the breakdown of the user’s superannuation fund where the app is part of government initiatives and the data can be retrieved by the user for convenience of understanding his/her financial standings.



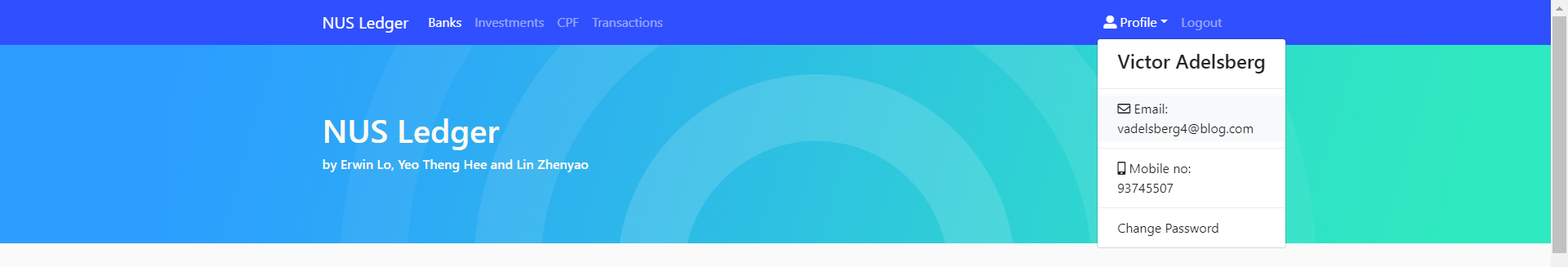
**Fig 5.10- Main Page (CPF Accounts)**

This screenshot is the list of transactions for different accounts within the list. If we want to view a long list of transactions, we can use the drop-down list to increase the number of transactions shown.



**Fig 5.11- Main Page (Transactions)**

The user can click on the headers in the tool bar to manoeuvre on the main page and look up the specific information needed. The profile tab allows the user to update their personal information should there be changes in them.



**Fig 5.12- Main Page (Tool Bar)**

The following forms for amending information are tested with validation codes.

|  |  |
| --- | --- |
|  |  |
|  |  |

**Fig 5.13- Forms for Amending User Details**

6. Program Enhancement  
  
1. Backend API security can be implemented. Ideally, when the backend API is accessed, they should send authorization tokens so that only authorized users can view and make changes to the database.

2. We have a frontend for users to reset their passwords. The program can be enhanced by implementing API calls for handling password reset requests. Once the backend server receives the requests, it will automatically send email with a reset link.

3. Cookies can be implemented to store login sessions for users.

4. Login process may be enhanced by using third party authentication & authorization providers.

5. SGFindex API can be implemented so that users can get their live account balance across participating financial institutions.

7. Our Team Experience and Contribution statement  
  
Our team of three (Erwin Lo, Yeo Theng Hee, and Lin Zhenyao) has worked very well together. Amongst us, Erwin is most technically competent, hence took up the most challenging task of doing up a fully-functional frontend with React.js, and creating the overall architecture of our project. He also kindly led the team to embrace the use of Github as a collaborative development platform, which facilitated our code sharing. Theng Hee brought along his prior experiences in database programming in the realm of desktop application development, to value-add to database design, a few API designs and program documentation. Zhenyao demonstrated very keen enthusiasm to pick up the ropes along the way, and assisted with numerous tasks along the way, e.g. database creation, idea generation, and some code writing. Our team has found the learning experience fruitful and beneficial for our personal and career development.