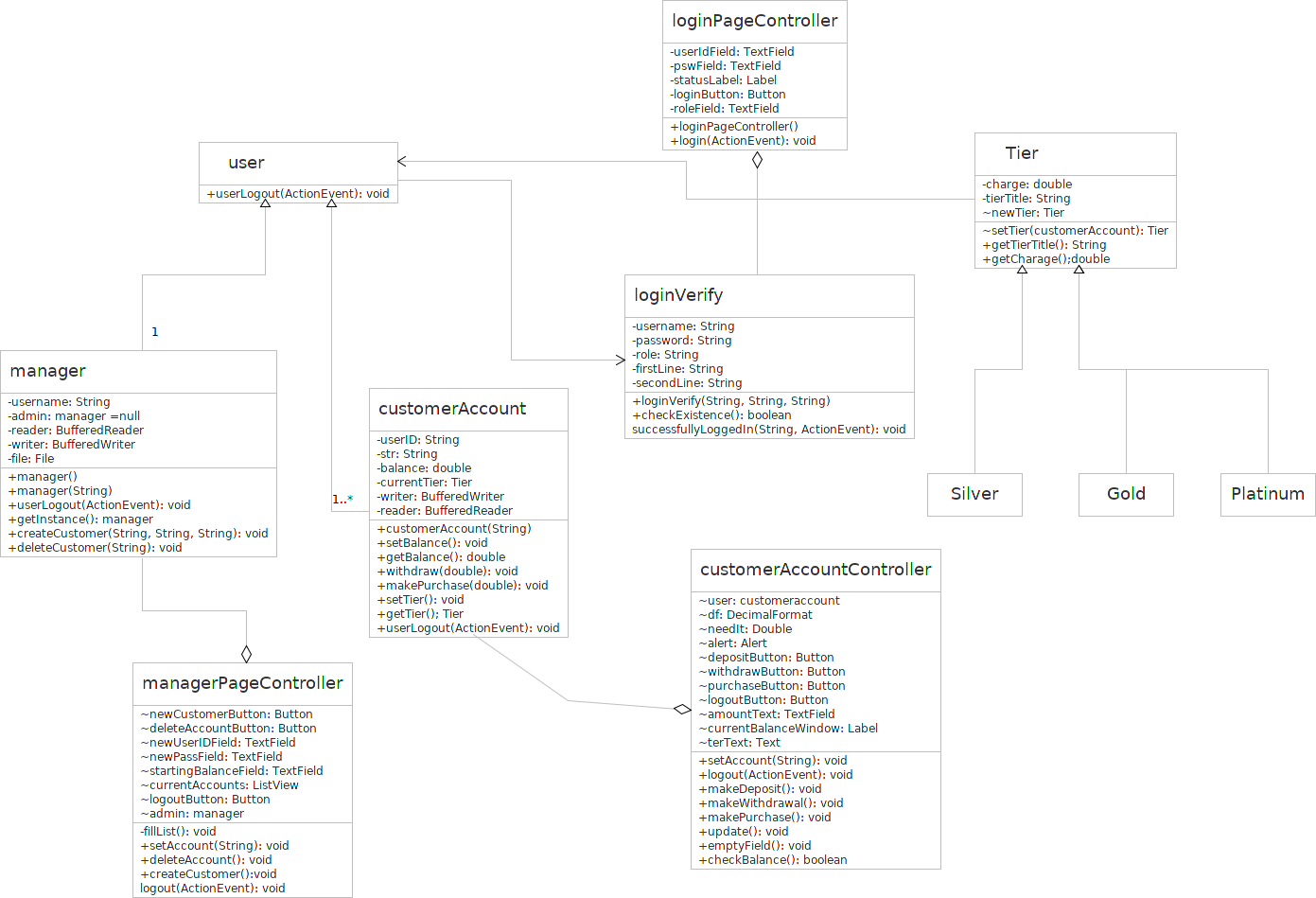
COE528

Major Project Report

# Banking App

The banking app created for the major project can be considered a success, as it followed and met all the requirements. There were small minor challenges that were faced when coding certain implementations. Some of the challenges were dealt with, however there were a handful of others that were deemed minor and left for future improvements.



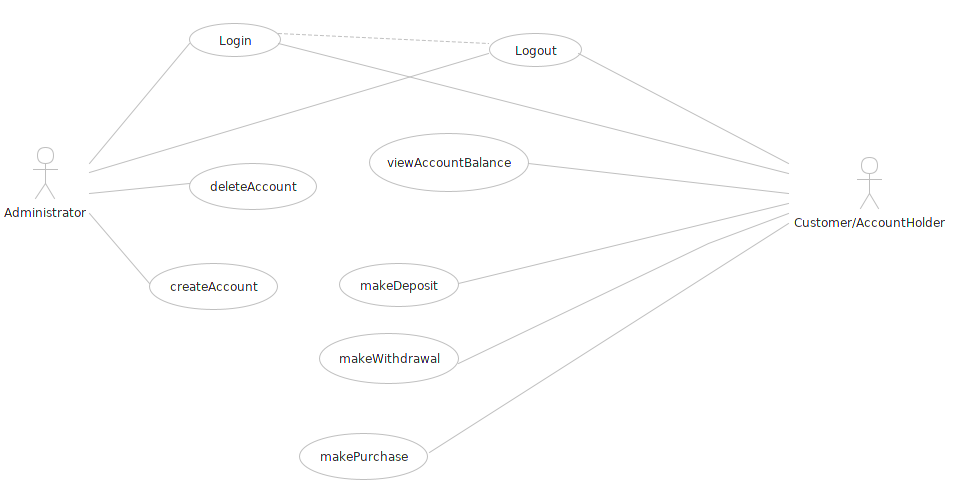
**FIgure 1:** *Class Diagram for the Banking App*

As seen in the above in Figure 1, the Banking App utilizes various classes. The controller classes are used to aid their class counterparts. They take in the information and pass it to their counterparts methods. They are in charge of ensuring that the information being sent to the counterparts methods matches the parameters. If they do not, an error window should appear informing the user that a field is missing, or another form of error has occurred on the user’s end. The loginVerify class checks to ensure that with the given parameters passed by loginController, match with the user account files. If not an error appears that alerts the user and notifies them to speak to a manager. Once logged in, if the user were an admin, the admin window will appear. The admin window is controlled by the managerPageController. The window is sectioned off into 2 portions. The left hand side would show the current accounts once the update accounts button is pressed and is used to delete accounts. Deleting accounts was simply implementing a ListView, that once clicked would delete the account. The right hand side of the window is used to create new customer accounts. The admin enters a username, and password and has to add a minimum of one hundred dollars. If any fields are missing or the initial deposit is insufficient an error window appears. If everything is properly entered, the controller sends the method to the manager class which creates the class.

Had the user been a customer, the customerAccountController class loads the customerAccount class, and fills in the data on the window. They have the option to make transactions. As the transactions occur, the left hand side shows the current account balance, and tier. Whereas the right-hand side shows the possible transactions and has a TextField box for the user to enter an amount. The customerAccount class has the methods: makeDeposit(), makeWithdrawal(), makePurchase(). Everytime these methods are called, the transaction is recorded into the customer file, as well as setting the customer’s Tier using the setTier(). This in effect changes the Tier using the State Pattern. Writing every transaction to the customer’s file was implemented to allow for further development of the application.

The Tier class seen to the right of Figure 1, is an abstract class that is in charge of setting the tier of the User class using the balance of the user account. It is the superclass of the Silver, Gold, and Platinum classes. The tiered classes (Silver, Gold, and Platinum) each have their own “charge” value that is of type double. They inherit methods such as getTierTitle, which is used by the customerAccountController to display the Tier of the customer’s account. The Tier class uses the owner’s balance to decide which Tier is set for the customerAccount. As any transaction is carried out by the customerAccount (makeDeposit, makeWithdrawal and makePurchase), the setTier() method of the Tier class is called, which changes the tier, as the balance changes. Thus as any transaction occurs the Tier of the customerAccount is changed instantaneously. For purchases the customerAccount class calls the getCharge() of the Tier for the account, when calculating the balance. Once more the setTier() method is called, which updates the Tier after the transaction.

Figure 2, depicted below, shows the use cases for the banking app. As seen, there are two users, an administrator and customers/account holders. The uses for both of them are very different. They only share two uses, which are login and logout. As the administrator is not a client of the bank, they do not need to make any transactions, instead their actions involve deleting and creating new accounts. The customer, on the other hand, does not have these capabilities. Instead the customer makes transactions such as makeDeposit, makeWithdrawal, and makePurchase. They can also view their current account balance.



**Figure 2:** *Use Case Diagram for Banking App*

As seen and described the app functioned as it should. However as mentioned earlier, there were some challenges that were left for future improvements. One of them is that the customer account balance is currently allowed to go into a negative balance. However once in a negative balance, the user is only allowed to make deposits and not purchases or withdrawals. This acts as a kind of overdraft. However at the moment there is no limit to how low the account is allowed to go. A future improvement to the code would limit the overdraft amount. Another is with the manager window. Once logged in, the admin must click a “current accounts” button that loads the list with accounts. An improvement would be to load the list immediately once the account is logged in. Another case with the admin, once an account has been created, the file is not updated into the ListView. Immediate update to the ListView would be recommended going forward. However even with the list of current accounts not updated, if the admin logs out and the user logs into the newly created account, it will work fine.

Completely unrelated to the requirements of the project, the current files for the users, record all the transactions that occur. Once the user logs in, all the transactions are summed to get the current balance. Using this method, improvement to the performance can be implemented. Also as all the transactions are recorded a future upgrade to the app, would list further information including transaction type, time and date. With this information, a search function can be implemented for the user to see the amount deposited, or purchases and withdrawals made within a timeframe.

As stated earlier, the overall banking app implements all the requirements stated. Small details were left unhandled but do not affect the overall use of the application. Further details were implemented when designing certain aspects for future improvements. Thus the Banking App can be considered a success and fully functioning application.