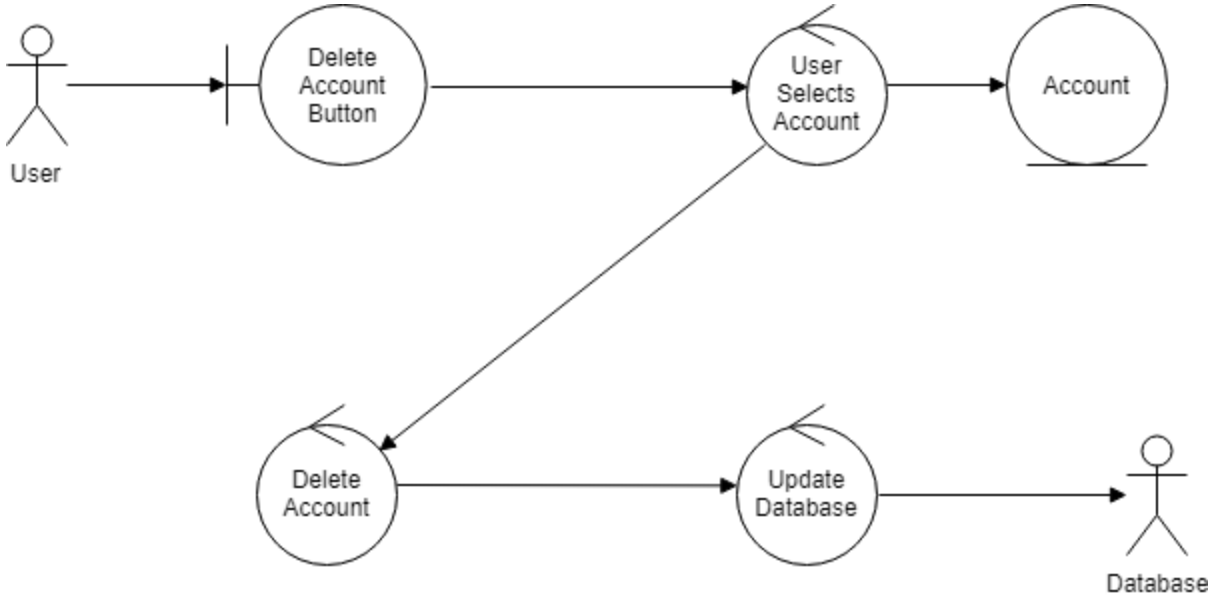
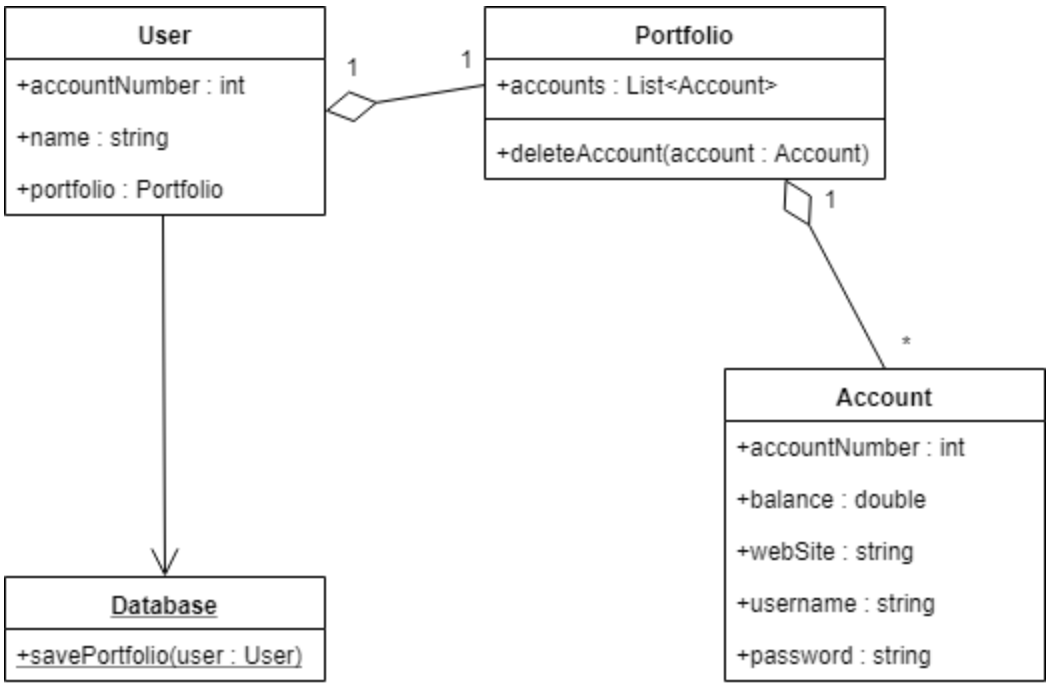
Use Case: Delete Account

Use Case Realization:



Conceptual Class Model:

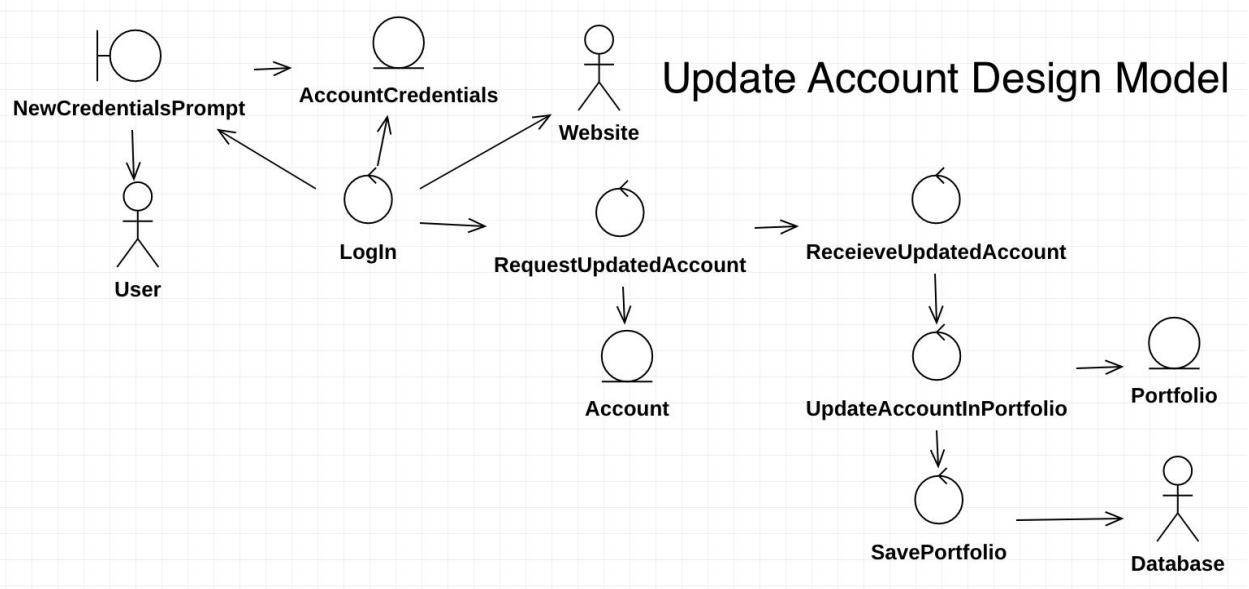


Basic Flow: The User will click on the Delete Account button, which will then ask the user to select an account from the list of accounts in the Portfolio class. The User class will then call deleteAccount() in the Portfolio class which will delete the account from the list of accounts.

Once this is completed, the User will call savePortfolio() in the Database class to save the updated portfolio to the database.

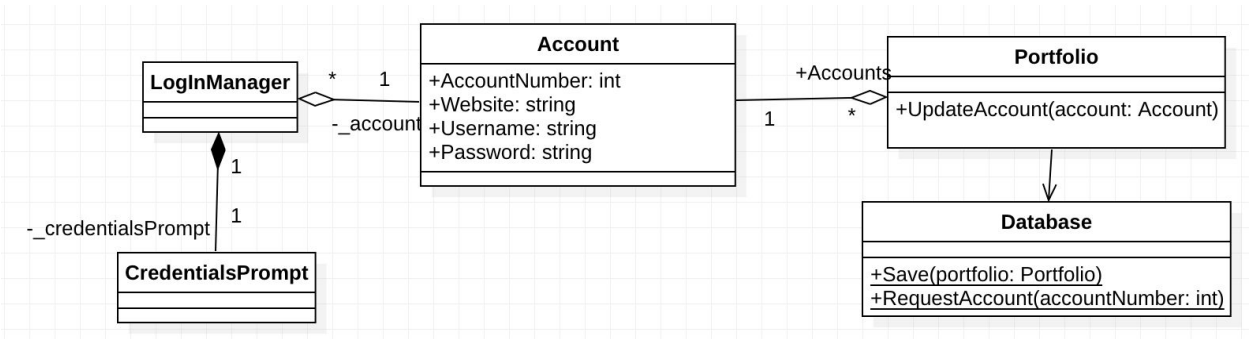
Design Decisions Made: A Database class was created to accommodate the querying of the database from the application. The Asset class was not included in this class model because it was never referenced in this particular use case. Also, the accounts attribute of the Portfolio object was chosen to be of type List because it needs to be able to grow and shrink as the user adds and removes accounts. The savePortfolio() method in the Database class was declared static because there is only ever one database connection at any given time, and so individual classes need not have individual references to database objects. The instances of the Account class were omitted because the type of account being deleted is unimportant.

Use Case: Update Account



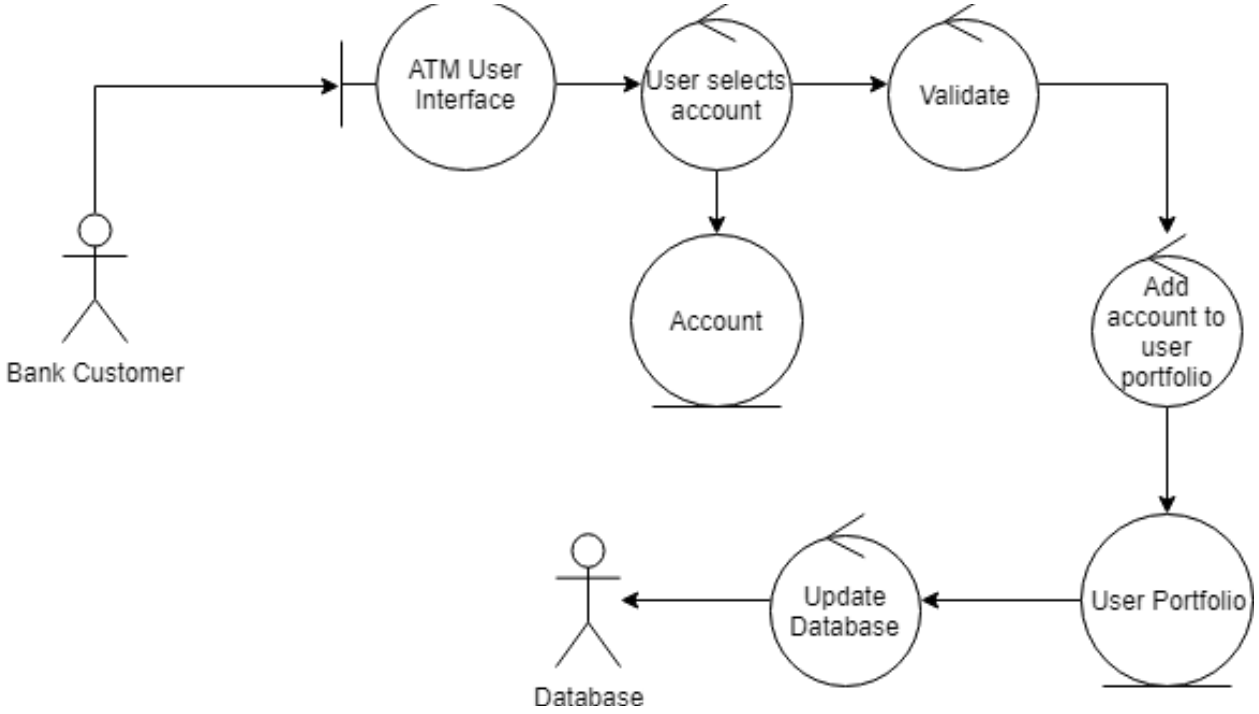
Design Model/Flow: The system starts off by attempting to login to the website using some account credentials. If the credentials are invalid, the *user* is prompted to enter in new account credentials (this is the reason the user actor was added to the design model). After logging in, the system requests updated information for the account (now, it may get this from the website or the database, I am unsure of which it does), accessing the current Account to send the request. Then, Once the updated account is received, we update the account in the current portfolio and save that portfolio to the database.

Update Account Class Model



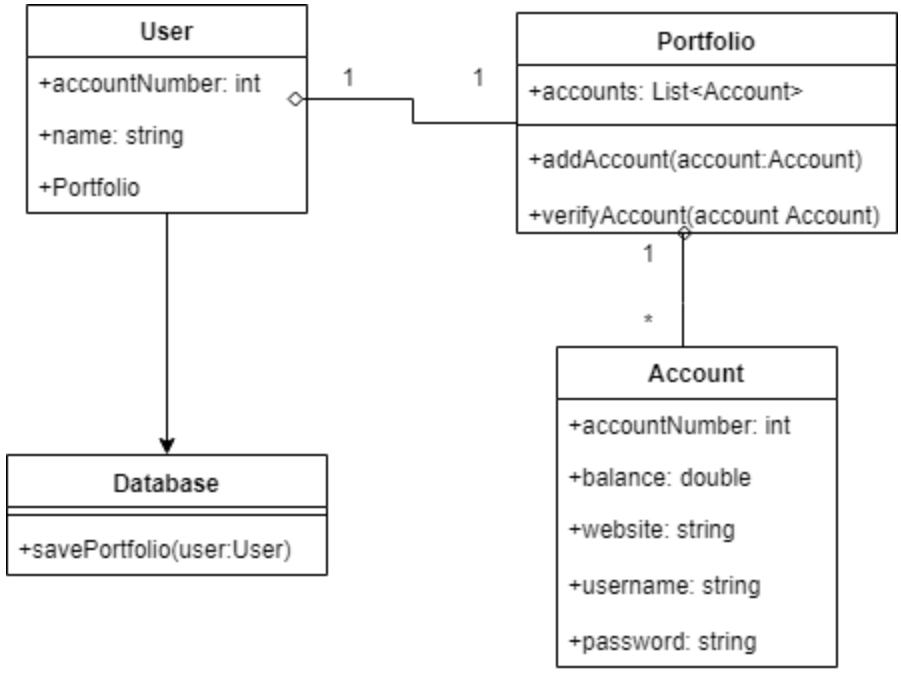
Class Model Design Choices: Add account design model decisions: I added a LoginManager because the system needs a way to attempt a login on the website using the given Account's credentials. Also, it needs to be able to request new credentials from the user if the current ones are invalid. A Portfolio contains a list of accounts that can individually be updated from the update account operation. The Database was added so that things may be added and retrieved from the database. The Save and Request Account operations were added as static operations because there is only ever 1 database in the lifetime of the application. There was no need to add anything for receiving the requested account as it is assumed, that even if RequestAccount were an async operation, the program would be able to await a return or simply pause until receiving it. The LogInManager and CredentialsPrompt dont have any attributes or operations added because they are added in addition to the given domain model and are not necessary to show the class model for this use case. The LogInManager would probably have a Login() method though.

Use Case: Add Account



Design model reasoning: The diagram starts with the bank customer actor as it is the initiator for this interaction. The diagram then flows to the ATM user interface where the bank customer selects the account type and is prompted for the account information. From the user selects account control, it is connected to an account entity as that is when it becomes relevant. Then the diagram flows towards a validate algorithm where it validates the account information. Then after validating to flows towards another algorithm that adds the account to the user portfolio, which then interacts with the entity: user portfolio. Finally the diagram flows to the update database where it then flows the database actor itself.

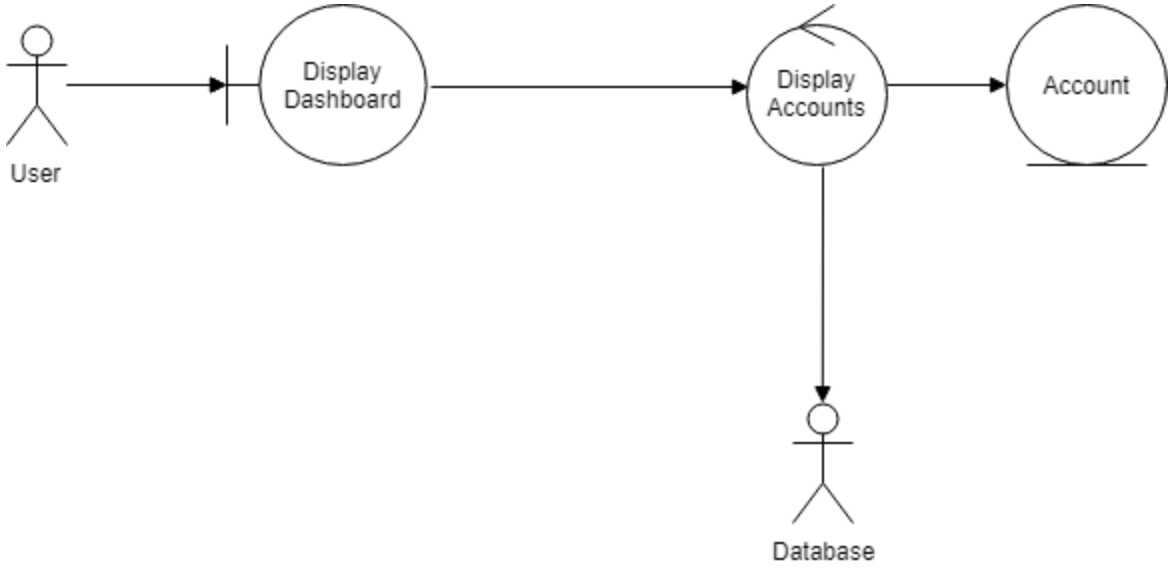
Add account class model:



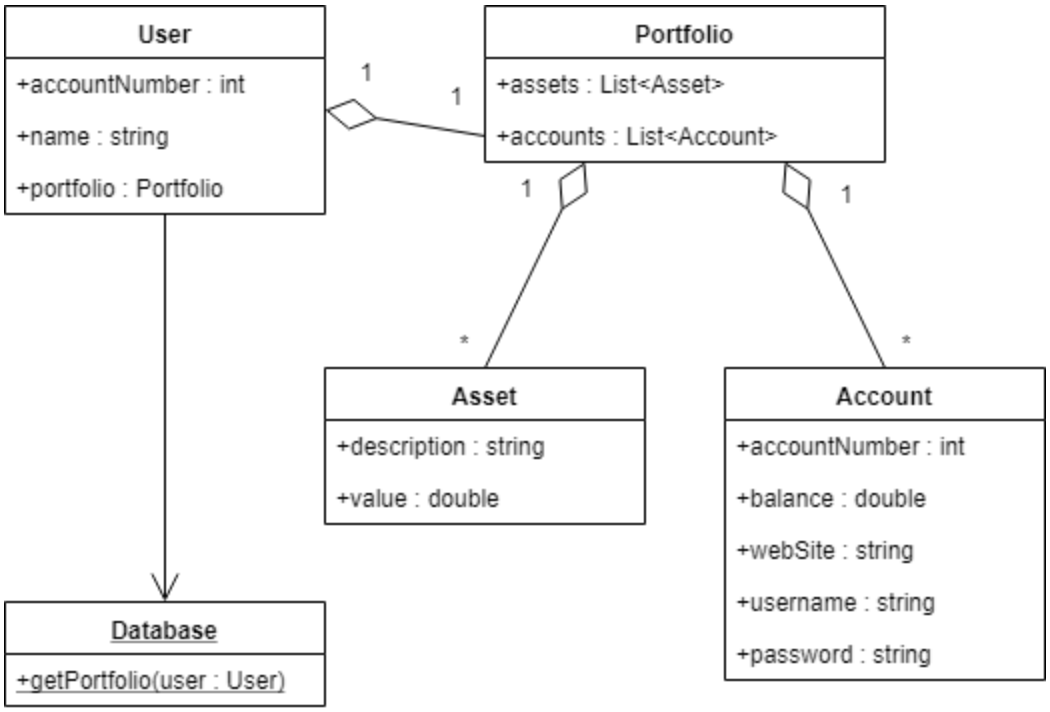
Class model reasoning: For the class model for add account, it follows much the same pattern as the delete account class model. However where it differs is firstly instead of an deleteAccount method it is a addAccount method. Beyond that it also has contains a verifyAccount method that takes in an account and verifies via the list of accounts in the Portfolio class.

Use Case: Display Dashboard

Use Case Realization:



Conceptual Class Model:



Basic Flow: The user will go to the dashboard. The User class will then call getPortfolio() in the Database class, which will get the portfolio data from the database. The User class will then iterate over the assets and accounts in the Portfolio class and build a list of all of the items and display them to the user.

Design Decisions Made: The Asset class was included in the class model because when the dashboard is displayed, the assets will be displayed alongside the accounts. The asset attribute of the Portfolio class was chosen to be type of List because more than one asset can be owned by the user.