150108

Web Apps Coursework Report

- (5%) How your design fits with the 3-tier architectural model and the model-view-controller software pattern

My project follows the 3 tier Architectual model set out by the labs over the term, the presentation layer which the user directly interacts with ,coded with xhtml, building elements within the body of the pages such as forms for entering information and commandbuttons/input/output elements which interact and pull data from the next layer. Due to using xhtml we can code in java to pull data and create methods to fill these xhtml pages through JSF's. These beans' methods can be directly referenced from the xhtml. My xhtmls pages are separated in three part the main index which links to login and registration and then user and admin pages, only accessible if the user has been authenticated and is allowed to view those pages.

The next layer is the logic layer, the JSF beans call methods from EJB's to get information, these EJB's are server-side classes which can directly connect to the stored information and carry out the main calculations and methods that should be done server side, then sending the results to the user sides JSF's. I have two main EJB classes one for user services and one for admin, both EJB security protected. I also have a third EJB which hold setup for the database when first run, if an admin doesn't exist it creates one per the assessment instructions. The user EJB holds methods for creating an account and transferring money while the admin holds methods for getting vast data from the database.

The third layer is the database itself; I have three databases all set up as entity classes within the project, SystemUser which holds each user's information and balance, SystemUsersGroup which holds their access level and transactions, this holds a record of every transaction between users.

- (5%) How your design could be extended so that your server is not a single point of failure

To stop the server being a single point of failure more information could be cached client-side. Instead of having to refer to the server for every request, previously retrieved data could be saved in the users' cache meaning the ability to still view previous transaction could remain while navigating back and forth pages without having to contact the server reliving congestion.

Multiple servers could be set up, checking each other to remain constant and allowing room for one to go down while the other takeover, or in cases where the server has a fatal error and the database could corrupt, keep a backup, non-user accessible server which holds a backup of all the data.

- (5%) How your system would deal with concurrent users accessing functionality and data

The moving of money is kept within a single transaction, if one user moves money another user should not have their transaction attempted until the first users is complete. This should provide no opportunity for money to be 'missing' if someone else executes something at the same time. Each user's transaction will occur one at a time and until the database is fully updated no other transactions should occur.

However, on the other side a large number of users would block up the system and if all requesting and updating balances it could cause an excessive wait time, needing timeouts to try and clear the system.