Group 10 Iteration 1 Report

**Release 1:**

For this release, a server was used to implement the communications between a mobile client and a database that were implemented.  The server consists of a virtual machine with the capability of socket communications to send and receive data.  Based on this information, login services and communications were implemented on the server side to handle incoming data.  A testbench for the server was written and used to test and further implement the sockets that would be used for server communications.  Also, a listener was implemented and tested to make sure the server would receive and respond to incoming communications.  The User Stories which reflect these are portions of Client and server along with Account Creation. The person-hours spent working on this iteration total 109, which far exceed estimates found in Requirements Specification which estimated work on these aspect to take 36 hours total.  The estimates made were far too conservative for our level of knowledge and best practices in larger scale projects.

The mobile client in this release consists of a simple UI developed using the iOS compatible language Swift.  The ability to create a new account was added with the ability to login and logout also.   The data entered into the user interface will be sent by way of the server communications in order to be stored in the database.  The mobile client was developed using Xcode, which also contained a simulator in which to test the mobile application on iOS.  This feature allowed for the location of specific problems in the code, which lead to faster debugging.  The mobile application would allow for the user to log in to an existing account, or to create a new account for use if necessary.  An API called TCPIPSocket helped us implement and send the necessary communications to handle logins on a socket connection in the mobile client.  The mobile client also allows the user to switch between these 2 options in case they selected the wrong option.  Once the server and the database were up and running, the mobile client was simulated with the proper sockets connections open and messages were successfully passed between the client and server.  After verifying that the communications were functioning properly between the client server, the client create account and login methods were tested in the interface.  The only issue encountered was after logging to an existing account, the mobile client would not redirect the user to the home page.  The create account feature seemed to work as designed.

The Database is setup to listen for the communications from the Server and perform the operations requested.  Upon attempting to log in, the database will check that the username and password entered match the ones stored, and that the user has an account that has been previously created.  The Database will also listen for the creation of a new account, in which case it takes the username, password, and email fields entered into the mobile client and stores them.  A testbench was created with an example name, password, and email.  This test class was written and run in order to pass test information through the Database’s handler methods  to make sure they were all functioning properly.  A testbench was also created to make sure the Login methods in the Database handler were implemented correctly, which attempted a series of logins onto the database.  The Database methods were released and then re-released several times as the testbenches revealed weaknesses in the system until 2-way communications could be confirmed.  Unfortunately, the Database methods consisted of several bugs that prevented from fully implementing it correctly.  Learning more about SQL, of which most of the group was unfamiliar, and package import errors proved insurmountable for this deliverable.

A smattering of Extreme Programming practices were used in the first release of this mobile application.  The 3 tasks that implemented were incrementally planned out in a spreadsheet, with a list of features and parts that needed completion listed.  All of these separate parts were uploaded to the GitHub repository, where the code was released in small chunks.  These pieces were constantly evaluated and refactored, making changes that were necessary and others simply for optimization.  Since the mobile client required Swift for implementation on iOS, pair programming was used as this language was unfamiliar to the team.  Using pair programming on an unfamiliar language added another layer of checking to help avoid syntax errors and similar situations that would hinder progress.  This extreme programming technique was also used to learn how to use SQL functions to access the database, as none of us have had previous experience with SQL.

Test Cases:

Server Communications:

Test Cases used:  Strings of characters, Strings of special characters and digits, whitespaces, and simply a newline character.

Unused test case: A null string or string not terminated by \n.  \n indicates end of transmission for a TCP/IP stream, and without this, the transmission will fail to function correctly, and cannot be safely handled by the receiving system due to the inability to have any foreknowledge of the contents.

Expected output was achieved from these test cases as indicated below:

idiscordi@bluebox-nix:~/workspace/setestbench/src$ java ServCommTestBench

establishing connection

sending: hello world

Server received your message as: hello world

got back comms

sending: 63217863178267dgsajhabdshm/sahuisadg78sa08ycanjk%#$@%(\*(SD\*

Server received your message as: 63217863178267dgsajhabdshm/sahuisadg78sa08ycanjk%#$@%(\*(SD\*

got back comms

sending: testing this out

Server received your message as: testing this out

got back comms

sending:

Server received your message as:

got back comms

sending: you can send a blank

Server received your message as: you can send a blank

got back comms

sending:

Server received your message as:

got back comms

sending: or simply a newline char

Server received your message as: or simply a newline char

got back comms

idiscordi@bluebox-nix:~/workspace/setestbench/src$

te