CLICKER

A TCP/IP based solution for authenticated, remote audience feedback.
Assignment 1 - EECS 3214

Thalammeherage Raj Perera
#2111429610
cse11011@cse.yorku.ca

Overview

The solution consists of two primary independent components. The “Server” and the “Client”. The speaker may set up the “Server” instance with a prespecified number of choices and the audience may use a “Client” instance each to connect to the speaker’s “Server” to send feedback.

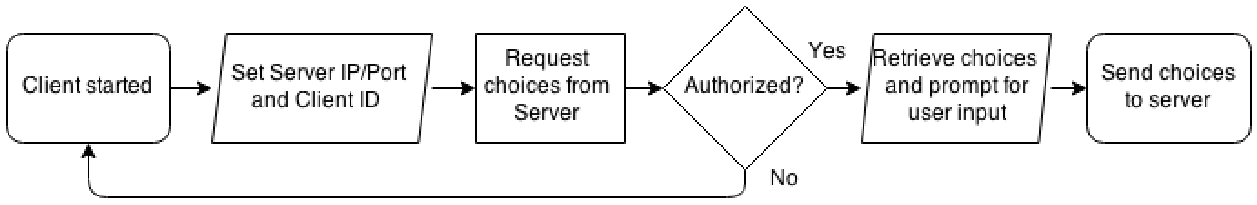
How It Works

The speaker running a “Server” process should let the audience know the IP address and the Port which the process is running.

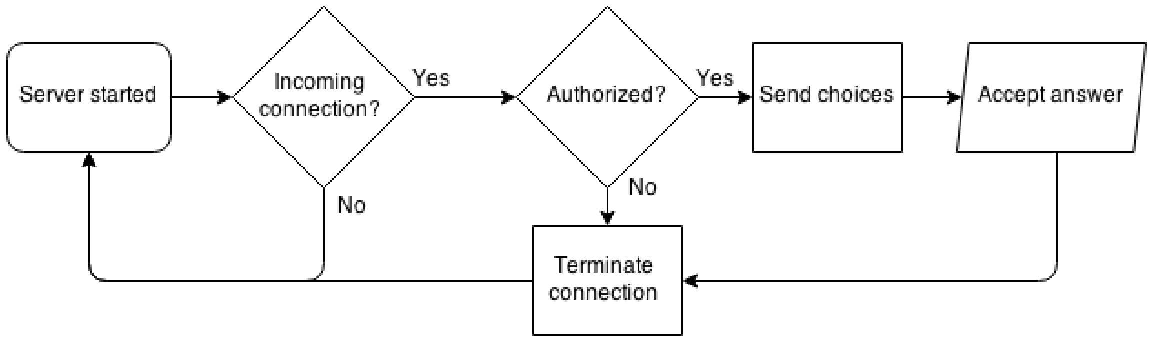
Then, each audience member should configure their “Client” to connect to the supplied IP/Port. The “Client” process must also be configured with a client ID that uniquely identifies each audience member (ie: a student number).

The “Server” instance will validate a feedback received from a “Client” by checking the client ID against a list of client IDs in a database. If the client ID is not known to the server, the server will allow the feedback to be provided and the offending “Client” will be notified.

Client


Client started
Set Server IP/Port and Client ID
Request choices from Server
Authorized?
Yes
No
Retrieve choices and prompt for user input
Send choices to server

Server



Architecture

The user interfacing elements of the components share many features in common. They both utilize an interactive top level command line interface (aka REPL - read, eval, print, loop) to manage user input and output. I chose this approach as it closely resembles the specification requirement of submitting ‘commands’ to the application.

The logic concerning communication between client and server are moved into a separate class and an API (Java) is exposed for manipulation. This modular approach allows scalability and portability via ‘pluggable’ user interfaces and also encapsulates the communication protocol to a centralized location. Furthermore, the separation of concerns allow graceful error handling and developer flexibility.

Communication

All communication between the components over TCP will be facilitated by serialized java objects. A serializable class that is present is in both components will act as the common medium. This class contain two variables, “action” and “value”. The parameter “action” refers to the specific intent of this request such as “Requesting authentication”, “Retrieve number of questions” or “Sending answers”. The parameter “value” simply refers to any corresponding data related to the “action”.

Server

Server process is required to be initialized with a set number of options and a TCP port to listen to. Upon initialization, it will utilize Java’s Thread class to fork itself out of main process and listen to incoming connections. Per each connection received, the server process will re-fork itself and handover the communication to the Server API class to negotiate protocol and data transfer (authentication, etc).

This approach allows the server to listen to new connections in a non-blocking way and handle many concurrent connections at a given time.

Database

The API class for server end requires a backing data access object to perform student number lookups. For the purposes of this assignment, I have created a volatile mock data store that returns hard coded values on request.

Client

The client component simply consists of a UI layer that translates user input to provided API methods of the Client API class. Given an open socket connection and a client ID, the client API will establish protocol, authenticate and present the user with options to chose and return the feedback to server.

Possible Improvements

Thread Pooling

Currently, there is no limit imposed on the number of threads that can be spawned by a server process. While this may allow unlimited number of connections theoretically, it is greatly dependent on the availability of resources in the computer. By utilizing thread pooling, it is possible to reuse threads and utilize the resources more effectively in a situation where the server may have to handle many connections concurrently.

Real Database

Even though the data access layer is well defined in the application, it is not using a real database. The question data is not persistent and will reset every time the server is restarted or the choice count is changed. This could be addressed by using a permanent data storage mechanism like a SQL database or file system based storage.

Push questions

Clients have to connect to the server to retrieve the the question to answer. It would allow a better user experience if the clients could be notified in real time when the server starts and starts accepting answer requests. This could be achieved by either having a persistent TCP connection to the server or letting the client periodically poll the server for new connections.

Known Issues

Answer data resetting

As a result of using a hardcoded ephemeral backing store to perform client ID look ups and store answer data, the database will reset every time the number of questions changed in the server. This is an architectural limitation and will have to be remedied by utilizing a persistent medium for data storage.

Installation

1. Extract the submitted zip file to a folder, let that folder be called “appRoot”
   From “appRoot”, the file structure should be as follows.

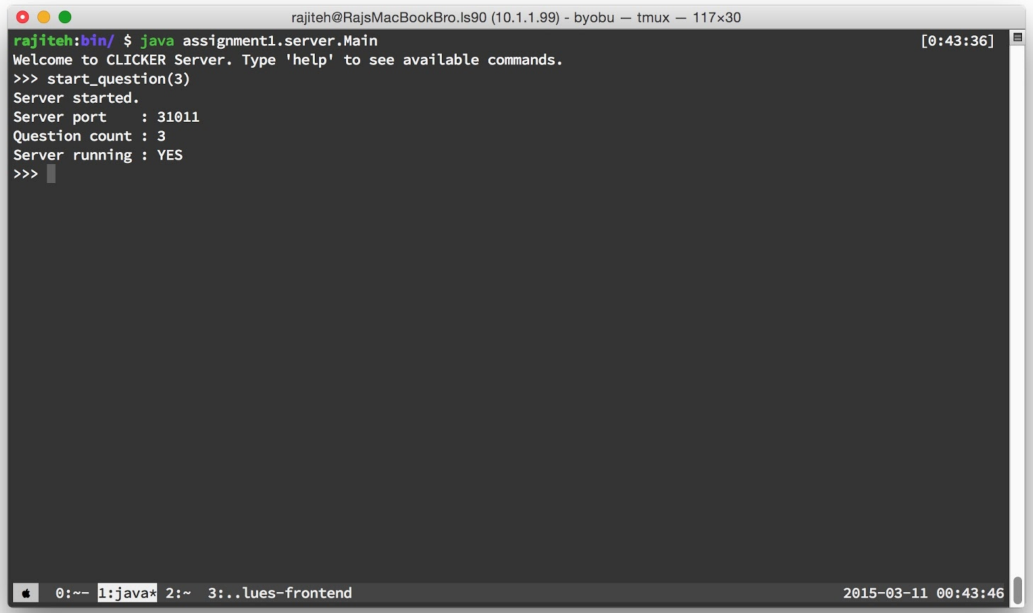
```
$> unzip assignment1.zip -d ~/appRoot
$> cd ~/appRoot
$> tree .
.
└── src
    └── assignment1
        ├── client
        │   ├── ClickerAPIClient.java
        │   ├── ClickerCLIClient.java
        │   ├── ClickerClient.java
        │   └── Main.java
        ├── server
        │   ├── ClickerAPIServer.java
        │   ├── ClickerCLIServer.java
        │   ├── ClickerDAO.java
        │   ├── ClickerDAOHardcoded.java
        │   ├── ClickerServer.java
        │   └── Main.java
        └── util
            ├── ClickerAPI.java
            ├── ClickerCLI.java
            └── ClickerRequest.java

5 directories, 13 files
```

2. In “appRoot”, create a new folder “bin”
   ```
   $> mkdir bin
   ```
3. Compile the source files to the bin folder
   ```
   $> javac src/**/*.java -d bin
   ```
4. `cd` in to the `bin` folder.
   ```
   $> cd  bin
   ```
5. To start,
   a. Server : **`$> java assignment1.server.Main [PortNumber]`**
      Replace [PortNumber] with a port number of choice, if not supplied it will default to 31011.
   b. Client: **`$> java assignment1.client.Main`**

Screenshots

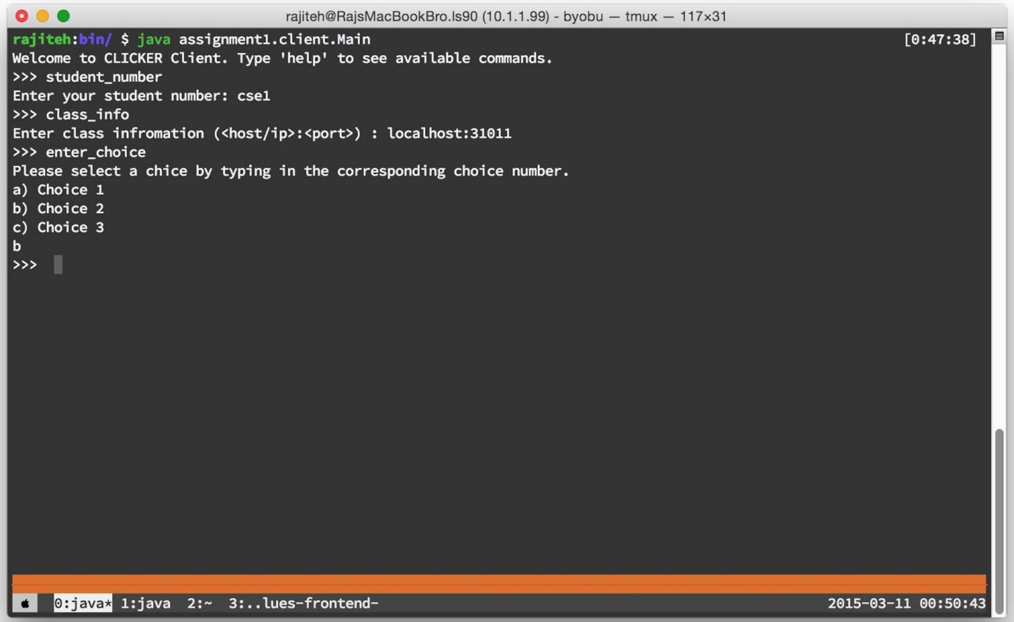
Server Process



```
rajiteh@RajsMacBookBro.ls90 (10.1.1.99) - byobu — tmux — 117×30
rajiteh:bin/ $ java assignment1.server.Main                                                              [0:43:36]
Welcome to CLICKER Server. Type 'help' to see available commands.
>>> start_question(3)
Server started.
Server port    : 31011
Question count : 3
Server running : YES
>>>
```

```
0:~- 1:java* 2:~  3:..lues-frontend                                                          2015-03-11 00:43:46
```

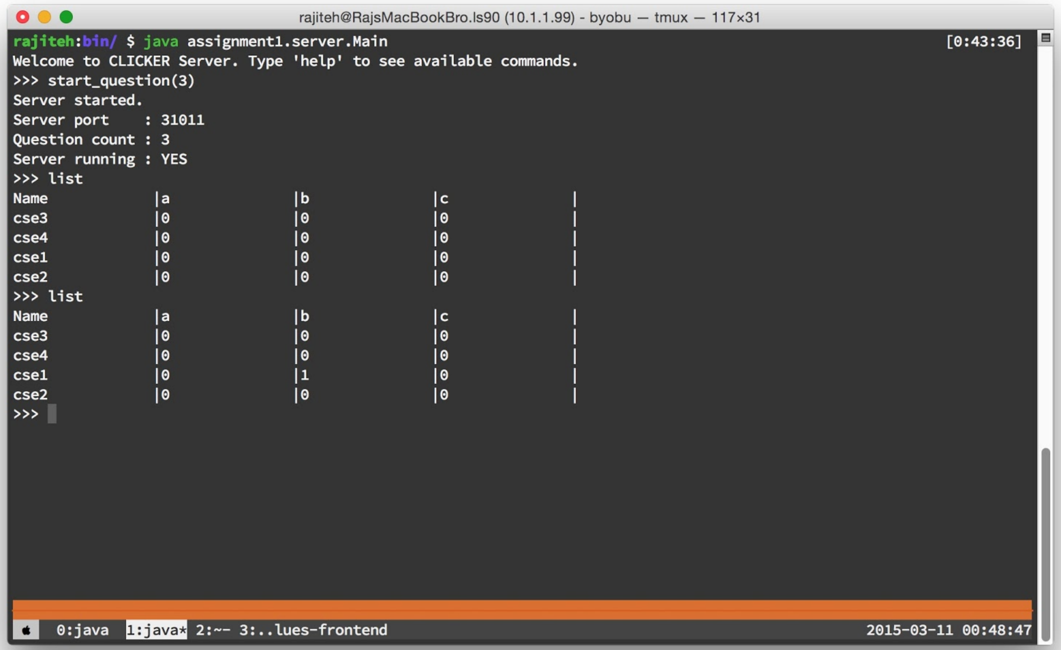
Client process



```
rajiteh@RajsMacBookBro.ls90 (10.1.1.99) - byobu — tmux — 117×31
rajiteh:bin/ $ java assignment1.client.Main                                                              [0:47:38]
Welcome to CLICKER Client. Type 'help' to see available commands.
>>> student_number
Enter your student number: cse1
>>> class_info
Enter class infromation (<host/ip>:<port>) : localhost:31011
>>> enter_choice
Please select a chice by typing in the corresponding choice number.
a) Choice 1
b) Choice 2
c) Choice 3
b
>>>
```

```
0:java* 1:java  2:~  3:..lues-frontend-                                                     2015-03-11 00:50:43
```

Server with updated answers

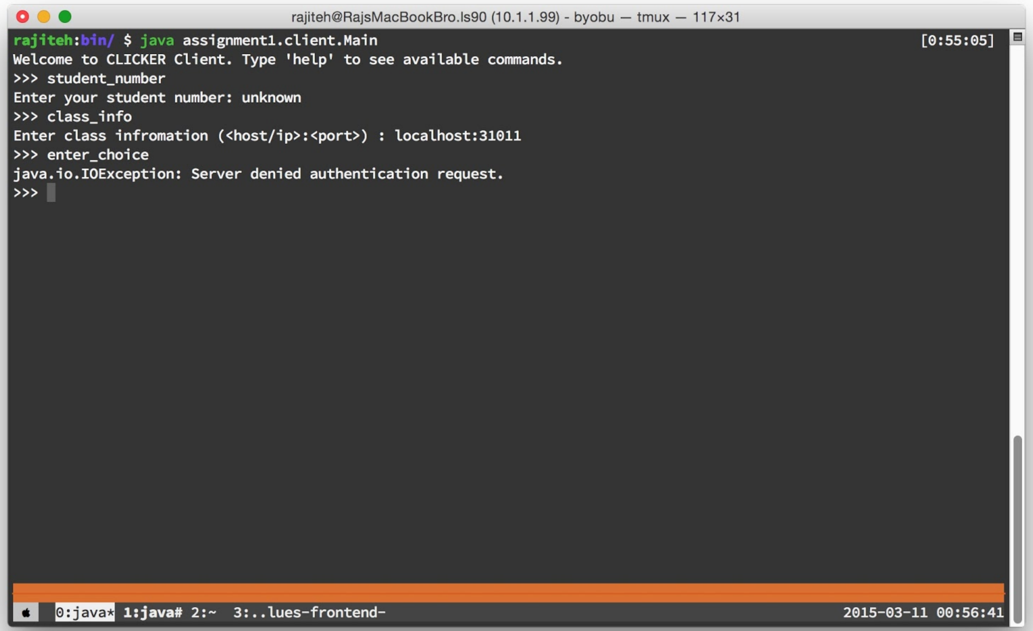


rajiteh@RajsMacBookBro.ls90 (10.1.1.99) - byobu — tmux — 117×31

```
rajiteh:bin/ $ java assignment1.server.Main                                                                   [0:43:36]
Welcome to CLICKER Server. Type 'help' to see available commands.
>>> start_question(3)
Server started.
Server port    : 31011
Question count : 3
Server running : YES
>>> list
Name            |a              |b              |c              |
cse3            |0              |0              |0              |
cse4            |0              |0              |0              |
cse1            |0              |0              |0              |
cse2            |0              |0              |0              |
>>> list
Name            |a              |b              |c              |
cse3            |0              |0              |0              |
cse4            |0              |0              |0              |
cse1            |0              |1              |0              |
cse2            |0              |0              |0              |
>>>
```

```
   0:java  1:java* 2:~- 3:..lues-frontend                                                       2015-03-11 00:48:47
```

Client with rejected authentication



rajiteh@RajsMacBookBro.ls90 (10.1.1.99) - byobu — tmux — 117×31

```
rajiteh:bin/ $ java assignment1.client.Main                                                                  [0:55:05]
Welcome to CLICKER Client. Type 'help' to see available commands.
>>> student_number
Enter your student number: unknown
>>> class_info
Enter class infromation (<host/ip>:<port>) : localhost:31011
>>> enter_choice
java.io.IOException: Server denied authentication request.
>>>
```

```
   0:java* 1:java# 2:~  3:..lues-frontend-                                                      2015-03-11 00:56:41
```