Mac Barrett

Senior Capstone – Spring 2021

Trivia Game – Full Documentation

SERVER DOCUMENTATION -----------------------------------------------------------------

public class server ------------------------------------------------------------------

variables:

 static CopyOnWriteArrayList<ServerThread> al ------- List of people in the game

 static CopyOnWriteArrayList<ServerThread> ST\_List -- List of all clients connected

 static CopyOnWriteArrayList<Queue<String>> ql ------ List of queue’s w/questions

 static CopyOnWriteArrayList<User> userList --------- List of registered users

functions:

static void connect\_WebSocket\_to\_ServerThread

Input: server socket, int id

Output: N/A

Desc: waits on port for new connections and then handles the WebSocket handshake for a new connection. Then it passes that connection to a new serverthread.

private static void initialize\_userlist

Input: N/A

Output: N/A

Desc: Loads the list of users from file into userList

class ServerThread extends Thread ----------------------------------------------------

functions:

ServerThread() constructor

Input: Socket, String username, AL<User>, AL<ServerThread> ST\_List, AL<ServerThread> al, AL<queue<String>>

Output: N/A

Desc: ServerThread constructor assigns the shared ArrayLists and things that make the game run.

Main Menu

Input: socket input stream "in", socket output stream "out"

Output: N/A

Desc: Main\_Menu is set up to handle messages recieved from the client and handle the other events that stem from the Main Menu

Pre Game Lobby

Input: socket input stream "in", socket output stream "out"

Output: N/A

Desc: Pre\_Game\_Lobby waits on user input to begin the real game and then waits for all of the players that have joined the waiting lobby to do the same

Initialize Questions

Input: N/A

Output: N/A

Desc: Reads the questions from file and then randomizes their order into "ql" which is deloaded at the beginning

Game\_Play\_Loop

Desc: Handles all the facets of the core gameplay loop. Acts as a driver for all of the other gameplay related functions

Get Input

Input: socket input stream "in", int "timeout": represents seconds that the timeout should be active for

Output: decoded message as string "in\_data"

Desc: GET\_INPUT reads the data from the Websocket for "timeout" seconds and calls the decode function. If the timer runs out the data is set with a dummy string

Send Output

Input: socket output stream "out", string "out\_message": the string to be encoded & sent out

Output: N/A

Desc: SEND\_OUTPUT simply takes the string, encodes it, and writes the message to the socket output stream. This has been made a public function so that a certain thread may tell another to send a message if need be

Select Category

Input: socket input and output streams "in" & "out"

Output: void

Desc: SELECT\_CATEGORY sends data to the category selector which then selects the category to pull a question from. If the category's queue has data left in it, it pops the top of the queue and outputs the data to the other ServerThreads.

Handle Selector Switching

Input: none

Output: void

Desc: Handle\_Selector\_Switching declares the category selector for the next round. Highest score from this round selects the next question's category and all edge cases are accounted for

Allocate Points

Input: websocket output stream "out", string "in\_data"

Output: int that contains the amount of points earned this round

Desc: Compares the client's answer (from in\_data) to the correct answer and allocates points according to the number attached to the answer if they are correct

Compile Scoreboard

Input: none

Output: string "temp" that contains strings to be sent to the client

Desc: Compiles the scores from all ServerThreads and writes them to a string

Check for Winner

Input: int "target\_points" to end the game, string [] "q" which contains the correct answer

Output: string "temp" that contains strings to be sent to the client

Desc: Determines if the game should end based on each ServerThread's point totals and appends the appropriate opcode

Write Users to File

Input: N/A

Output: N/A

Desc: Writes the contents of the userList file to users.txt, Called after each game by the selector thread only.

shuffle

Input: str

Output: str

Desc: splits and shuffles the incoming string and then ensures the question is back at the top of the array, then puts the string back together

decode

Input: inputstream in

Output: string

Desc: decodes the message from the websocket dataframe into a usable string

encode

Input: string

Output: byte[]

Desc: encodes the message from a string into a websocket dataframe message. Found it here by Haribabu Pasupathy -> https://stackoverflow.com/questions/8125507/how-can-i-send-and-receive-websocket-messages-on-the-server-side

class User ---------------------------------------------------------------------------

variables:

 String password ------- duh

 String username ------- duh

 int points ------------ total points accumulated by all games

 int games ------------- count of games joined

 int wins -------------- number of wins this user has

 int correct\_answers --- count of right answers

 int wrong\_answers ----- count of wrong answers

functions:

public String dump()

desc: returns a string that is written to file in a way that the server can read it with initialize userlist

public String Leaderboard\_Display()

desc: returns a string that is designed for use by the client’s scoreboard display logic

public Boolean validate(String validateString)

input: validateString, which is the username + password of the data received

output: Boolean

desc: returns true if validateString == this User’s username + password, else false

CLIENT DOCUMENTATION -----------------------------------------------------------------

function FORM\_SEND(data, formname)

input: String data, String formname

output: N/A

desc: sends the data string to client and hides the form that corresponds to formname string. Performs certain actions depending on formname such as stopping timers & intervals

function QUIT\_BUTTON()

DEPRECATED DON’T WORRY ABOUT THIS

sock.onmessage = function(event)

input: String event

output: N/A

desc: This is the event listener that is called everytime a message comes in over the WebSocket. The first thing it does is preform a string.split(“,”) on event.data to get instructions[]. It then hides all of the forms in case any are open, and then performs a switch on the first element in the instructions array to get the form to display. Depending on the case, does different stuff, but will typically use the rest of the instructions array to populate a screen which the user will interact with.

window.onload = function()

input: N/A

output: N/A

desc: This is called when the page first loads before the connection to the client is even established. All it does is hide all of the forms so you don’t do anything to break the game before data can be sent in properly and then setup some event listeners on the Login Submit Button, the New Question Button, and the Quit button, since they needed some extra fields in the “data” string field before being sent to the FORM\_SEND function.