Eighth Assignment Report

Sara Akbarzadeh 401222155

ClientMain

The main method:

Establishes a socket connection with the server.

Sets up input and output streams for communication with the server.

Sends a request to the server to show the initial menu.

Reads the response from the server in a loop until the response is null.

Processing the server response:

The response from the server is parsed as a JSON object.

The JSON object is examined to determine the type of command received from the server.

Based on the command, different actions are taken. The provided code only includes handling for "Login" and "SignUp" commands. Other cases are missing from the code.

Handling "Login" and "SignUp" commands:

If the status in the response is "true," the client displays a menu to the user.

The user enters a command (1, 2, 3, or 4) to perform various actions.

If the command is 4, the client logs out and the execution of the program ends.

Otherwise, the client sends a request to the server based on the user's command and the received JSON response.

ServerMain

It listens for incoming client connections, creates a ClientHandler thread for each client, and handles the communication with clients.

Here's a breakdown of the code:

The main method:

Creates an instance of ServerMain with a specified port number (2345).

Initializes a ServerSocket object to listen for incoming client connections on the specified port number.

Maintains a list of ClientHandler objects to handle communication with individual clients.

The start method:

Prints a message indicating that the server has started.

Accepts incoming client connections in a loop using serverSocket.accept().

For each new client connection, it creates a new ClientHandler object, adds it to the list of clients, and starts the thread.

The ClientHandler class (inner class of ServerMain):

Represents a thread that handles communication with an individual client.

Initializes input and output streams for the client socket.

Overrides the run method to listen for client messages in a loop.

Parses the received message as a JSON object and extracts the command.

Based on the command, different actions are taken. The code provided includes handling for "Login" and "SignUp" commands but lacks implementation for the "List of available games" command.

If an exception occurs during communication with the client, the socket is closed, and the client is removed from the list of clients.

Request

The Request class:

Represents a request object with a command, user information, and JSON data.

Provides getter and setter methods for accessing and modifying the command, user, and JSON data.

Includes several static methods for generating different types of requests.

The showMenu method:

Prompts the user to enter a command (1 for login, 2 for sign up) and reads the input.

Based on the command entered, it prompts for additional information (username, password, and birth date) for login or sign up.

Creates a JSON object with the command and the entered information and returns it.

The showUserPage method:

Takes a JSON string and a command as input.

Based on the command, it performs different actions related to user interaction.

For command 1, it queries the database for a list of available games and prints the information of each game.

For command 2, it prompts the user to enter the title of a specific game, queries the database for that game's information, and prints it.

For command 3, it calls the downloadFile method to download a game file.

Finally, it converts the JSON string to a JSON object and returns it.

The downloadFile method:

Prompts the user to enter the title of a game.

Queries the database for the game's information, including the file path.

Copies the file from the source path to a destination path using Files.copy().

Prints a success message and calls the Downloaded method of the Response class.

The toString method:

Takes a ResultSet object as input and prints the information of a game (title, developer, genre, price, and reviews).

Response

The Response class:

Represents a response object with a file, JSON object, JSON string, and message.

Provides getter and setter methods for accessing and modifying the file, JSON object, JSON string, and message.

The Login method:

Takes a JSON object containing the username and password as input.

Queries the database for accounts and checks if the username and password match any existing account.

If a match is found, it sets the "status" field of the input JSON object to "true" and returns it.

If no match is found, it sets the "status" field to "false" and returns the JSON object.

The SignUp method:

Takes a JSON object containing the username, password, and birthday as input.

Queries the database for accounts and checks if the username already exists.

If the username exists, it sets the "status" field of the input JSON object to "false" and returns it.

If the username doesn't exist, it sets the "status" field to "true" and inserts a new account record into the database.

Returns the JSON object.

The Downloaded method:

Takes a JSON string and a game ID as input.

Converts the JSON string to a JSON object.

Inserts a new record into the "Downloads" table in the database, representing the download action by the user for the specified game.

DataBase

The DataBase class:

Establishes a connection to the PostgreSQL database using the JDBC driver.

Initializes a statement object to execute SQL queries.

Provides getter and setter methods for accessing and modifying the statement and connection objects.

The constructor:

Loads the PostgreSQL driver class using Class.forName.

Creates a connection to the specified database using the JDBC URL, username, and password.

Initializes the statement object with the connection.

The query method:

Takes an SQL query string as input.

Executes the query using the statement object.

Returns a ResultSet object containing the results of the query.

<u>Bonus</u>

It includes two JavaFX applications: ClientApp and ServerApp, which represent a basic client-server communication setup.

Here's a breakdown of each class:

ClientApp:

The start method sets up the client application's UI, including a text area for logging messages, a text field for entering commands, and a send button to send commands to the server.

The connectToServer method is responsible for establishing a connection with the server in a separate thread. It creates a socket, initializes input and output streams, and starts listening for server responses.

The disconnectFromServer method closes the client socket when the application is closed.

The sendCommand method is triggered when the user presses the send button or presses Enter in the command text field. It sends the entered command to the server via the output stream and logs the sent command.

The listenForResponses method reads server responses from the input stream in a loop. It logs the received responses and performs specific actions based on the received command, such as handling login and signup responses.

The log method updates the UI by appending log messages to the text area using Platform.runLater.

ServerApp:

The start method sets up the server application's UI, including a text area for logging messages and buttons to start and stop the server.

The startServer method is triggered when the user clicks the Start Server button. It attempts to create a client socket and output stream to connect to the server.

The stopServer method is triggered when the user clicks the Stop Server button. It closes the client socket.

The log method updates the UI by appending log messages to the text area using Platform.runLater.

Both classes use JavaFX for UI components and threading to perform network operations in separate threads, allowing for a responsive user interface.