TEAM 39 ’DELHI’ Software Workshop Project Report

Group Members:
1880615 HIUKIT LEUNG
1663366 Kunal Singh
1958509 MAN KHI KIM
1935258 Nguyen Van Tiep
1859633 Oluwaseun Akindolire
1982178 Elizabeth Pickthall

Table of Content

1 Description of the system

The system involves a multiplayer quiz game with two players. In the beginning, player will be introduced to the registration screen that will require username and password. Users of the system will also be introduced to the registration page where they can create a new account. On the negative note, if the password or username is forgotten, the player will be able to retrieve the information. In that case, user will have to create a new account.

As the game starts, two players will be able to play against each other and will be presented with 5 multiple choice questions. Each question will have four possible answer of which only one will be correct. Questions will be shuffled randomly by the system, without the possibility of having the same question twice in a single game. For each question, players will have a limited amount of 10 seconds to answer the question. If a player runs out of time, the system will switch to the next question. If both players answer the question before the time ends, the next question will appear immediately with a timer being reset back to 10 seconds. For each correct answer to the question, fastest player to answer correctly will receive 100 points. If both players answer correctly, they will both receive 100 points. Players will also be able to keep track of both, their points and the points of their opponents when looking at the game interface. At the end of the game or when 10 questions are answered, the score will be calculated and the player with highest score will be declared as a winner. If both players have the same final score, there will be a draw without a chance of a tie-breaker. Additionally, the pop up window will show the name of the winner with a corresponding score. The final score will be stored in the leaderboard which tracks players' points that are being accumulated throughout each game.

1.1 Basic rules of the game

- Only multiple choice questions will be presented with four answers for each question
- Each question has only one correct answer
- A game will have 5 rounds, each round will have only one question
- Question will have a 10 seconds time limit to answer
- Each question is worth 100 points
- If both players answer the question correctly, each player would receive 100 points
- Inability to answer the question within the time limit would result in 0 points
- For an incorrect answer, player receives 0 points
- Disconnecting from the game would be automatically considered as a loss
- Player with highest points will be declared as a winner
- Winner can score 10 points in total score
- A draw will be declared if both players have the same final score

2 Requirement specifications

2.1 Functional Requirements

1. The system will have a login screen.

    1.1. New users will be able to go to a registration section of the interface to register an account.
    1.2. Existing players will be able to login with username and password.
    1.3. Correct combination of both password and username will users to log in.
    1.4. Incorrect username or password or combination of both will not be accepted by the sign in page.

2. The system will have a sign up screen.

    2.1. Users will be able to create a new account.

2.2. Username will be able to create only unique account that do not exist in a database.
2.3. Password can be same as in other accounts.
2.4. Successful registration will require password field and confirm password field to be identical.

3. The system will have a main menu screen.

3.1. Users will be able to see their name and the score on a page.
3.2. Users will have an option to go to either a leadership board screen, start a new game or log out of the system.
3.3. The log out option will direct the user back to the login page.

4. The system will have a leaderboard screen.

4.1. Users will be able to see their name and the score on a page.
4.2. The system will display a leaderboard in the form of a table with three columns that represent rank, username and a score.

5. The system will have a loading screen.

5.1. Players will have an ability to wait for other players to queue up for the game.
5.2. The system will transfer players to the game if both players joined the queue.
5.3. User will have an option to cancel the queue to return back to the main menu screen.

6. The system will have a game screen.

6.1. Users will be able to see their name, the score and a question timer.
6.2. Every player will have an ability to see opponent's name and a score.
6.3. A question will be displayed in the middle of interface and will be updated for upcoming questions and answer choices.
6.4. The system will reset a timer after each question.
6.5. The system will prevent the same question from repeating twice in the same game
6.6. The system will display 4 possible answers to the question.
6.7. Users will be able to select the right answer by clicking on it.

6.8. Points will be awarded accordingly if the system determines the right answer.

7. The system will have a winner screen.

7.1. The system will display user's name and a score.
7.2. The winner of the game will be mentioned by the system in a separate field.
7.3. Players will have an option to play again or to quit the game and return to the main menu screen.

2.2 Non-functional requirements

1. Efficiency

1.1. The game should start within the first five seconds after everyone connects to the client.
1.2. The leaderboard will be updated throughout the game.

2. Availability

2.1. Servers should be available 24/7 or depending on the hours of a school system.
2.2. The game should be available only when the game host/creator is online.

3. Internet

3.1. Internet access should be available to download and play the game.
3.2. The game will be not be playable unless all of the players are connected to the client via the network.

4. Server

4.1. Server should be able to send the information and retrieve it from the client within a short period of time.
4.2. Server must be able to handle all of the processes at any time once the game launches.

5. Client

    5.1. The client must be running at all times and receive messages from the server.
    5.2. The client must be ready to interact with a server and the interface at all times.

6. Reliability.

    6.1. The interaction between the client and a server should be running constantly or as long as the gui part is visualized to the user.

3 User Scenarios

Below are three examples of how different users can access different parts of the system throughout the different scenarios.

1. Simon decides to play a game.

   Simon is very competitive student and likes to test his knowledge, therefore he finds out about a new quiz game. He downloads the game and signs up. The game starts and he joins the queue. The opponent joins the queue and the game starts. Simon proceeds on winning the game. Finally, he is able to see his name on the winning screen at the end of the match.

2. John would like to brag about his score.

   John has been playing the game for a while. He would like to show his peers how well he is doing in the game, since many of his friends are also playing this game. He opens the leadership board and shows others that he is among the top players in the leaderboard.

3. Elisabeth suddenly runs out of battery.

   Elisabeth is playing the game out of boredom in the bus and she suddenly runs out of battery. She is worried about the match and is disappointed that she would probably lose the game. She is looking for the battery charge in the hurry so she can power up her mobile quickly and possible resume her game. However, in her

sad realization, 30 seconds has already passed and she wasn't awarded any points for these three questions. She ends up losing the game.

3.1 Use Case Diagram

Preconditions

- User has to be registered (unique username and any password)
- User has successfully logged in
- 2 players are available

Postconditions

- Total points of the player is updated in the leadership board
- Player can play again or quit the game
- All or some questions have been answered during the game

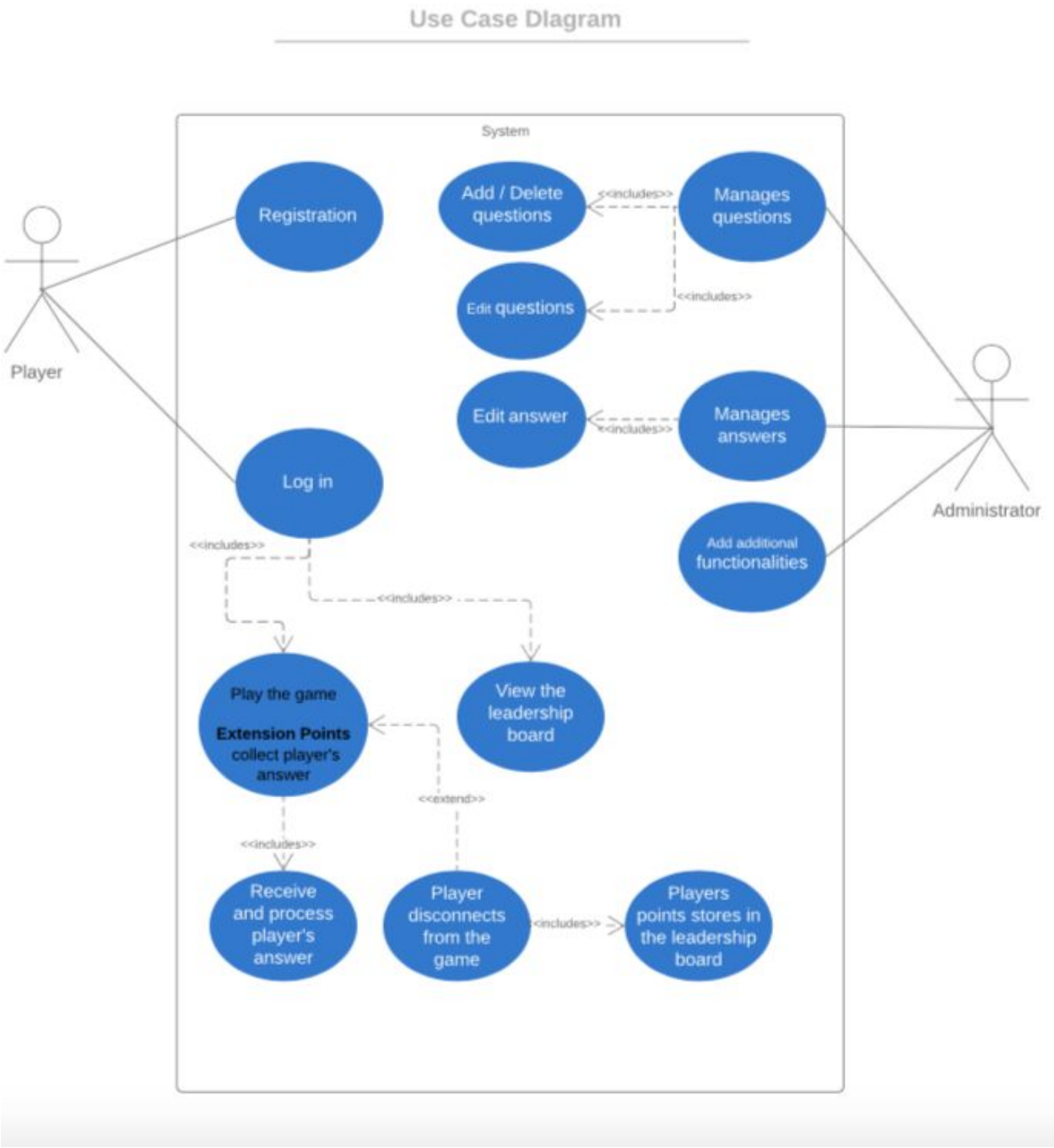


Figure 1: Use Case Diagram

4 System Design and Network

4.1 Architecture of the application

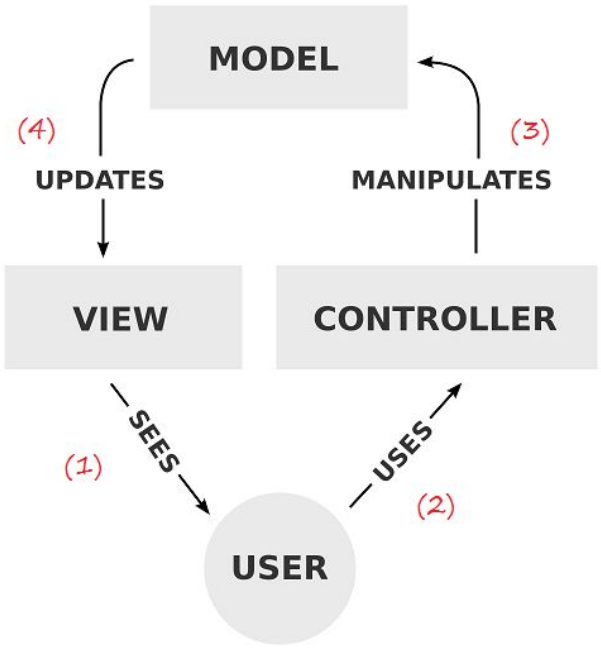


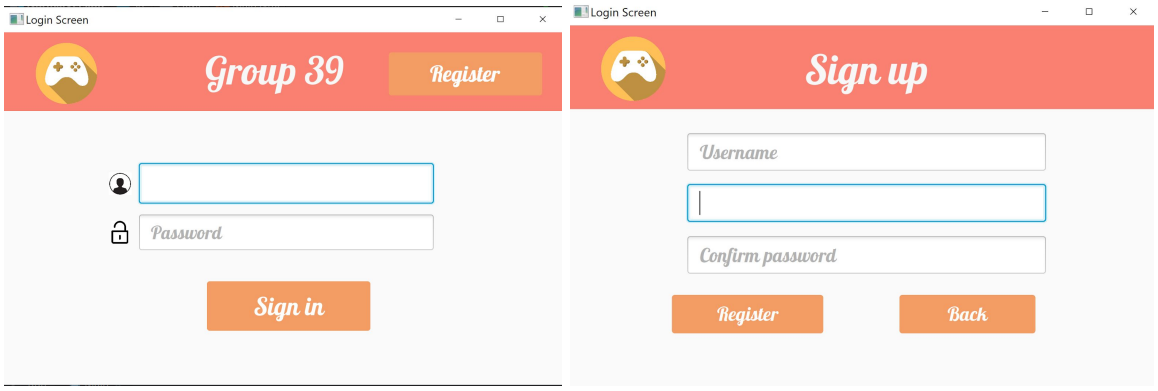
Figure 2: MVC(model-view-controller) model

When creating our software, we have decided to use a style similar to the model - view - controller architecture, also known as model - view - controller. In our case, the model here would be the server, the controller would be the client and the view is the graphical user interface that the user sees. For example, the user would input data or click on a button, which would then be sent to the client (the controller) that will then manipulate the data as it sees fit and outputs it to the server(the model). The server then uses this data to appropriately update the graphical user interface via the client, and this is an endless cycle during the duration of the program. We have used the server as the model in our program, as it manages all the data related to an instance of a game such as current scores, calculating high scores and as well as matching players together.

4.2 Graphical User Interface

4.2.1 Implementation and Management

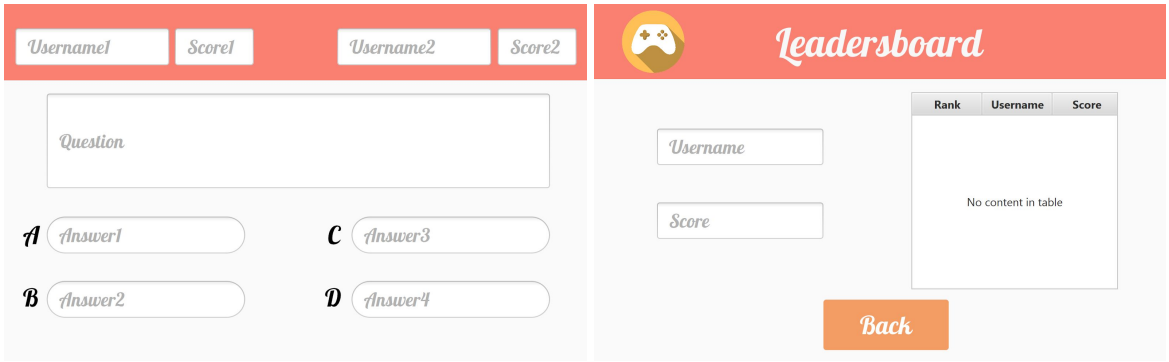
The graphical user interface or GUI is responsible for the visual part of the project. All of the visual effects were discussed beforehand to pick up the basic structure of the interface, as well as to decide on screens which are necessary for the final state of the game. The login screen and registration process of the game were decided early to create the GUI before trying to test whether the given fields work. The design, as well as the decision on fields which will be implemented, were discussed in a group to ensure that every step is vivid to each member of the group. Two screens were designed and constantly prototyped to the point where the final look was finalized and approved by every member of the group. An example of final design is shown below.



As a matter of fact, both of the screens as well as the rest of the GUI were created in a program called Scene Builder. The program allowed for the clean CSS like code to implement the beautiful design and efficiently store multiple fields and buttons on each screen without directly interfering with the classes that were responsible for the server and a client. The decision to keep the style designed above was made in order to have a consistent look throughout the whole game. Soon enough, after the initial login screens were finalized, members responsible for GUI came to a conclusion on how subsequent screens will look alike and be connected among each other. In fact, all of the upcoming GUI screens were soon created and further discussed by the group on what is desired in each screen.

Additionally, three classes such as Main, MainController and a ButtonController were created to manage the screens' scene changing nature and to be able to assign specific functions/methods to the provided buttons. All classes are interconnected and responsible for the specific task in a GUI. The Main class was created to initiate the visual part of the system and to refer to a MainController through the Main.fxml file. Main.fxml file initially launches the first interface screen and passes control from Main to

the MainController when it first launches. The MainController class is responsible for initialization and visualization of all of the remaining screens. In this class, getters and instances of other screens were created in order to keep the code less confusing and not to allow each screen have its own class. For the same reason, ButtonController class was introduced to be able to handle changes in a scene and to assign buttons with specific functions. In fact, majority of the buttons include functions such as pop-ups and the ability to change the scene. The functions of the buttons depend on which purpose the button serves. Additionally, some buttons were left empty fielded in order to be able to create a connection with a client and a server and give them an ability to fetch information from the database and display it on the screen.



Furthermore, empty table such as the leaderboard table was also created. The leaderboards table was designed for a user to see the best players and compare the scores.

4.3 Client

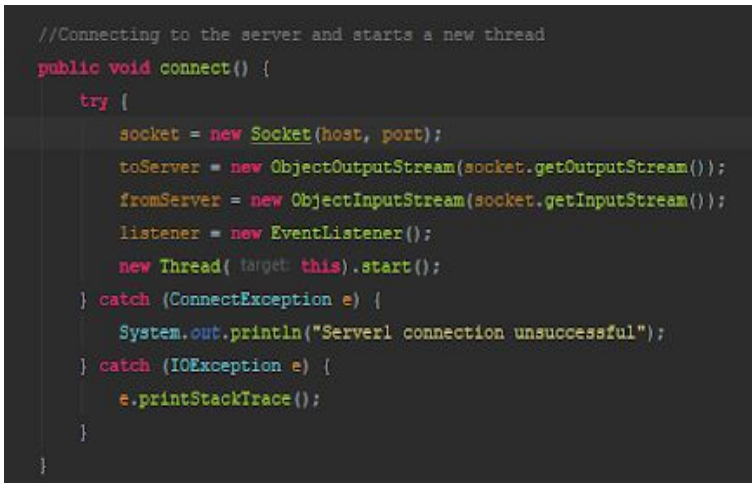
4.3.1 Implementation and Management

In order to allow the game to be multiplayer, we require a client and a server. The server acts as a central hub for all the information regarding the instance of the game the players are currently playing. The client serves two purposes, one of which is to relay data to the server from the game. For example, the user would input their username and password, and the client would take these two pieces of information and send it to the server to be confirmed. The second purpose of the client is to relay information from the server to the game, in this case, the server would confirm whether or not the username or password is correct, and the client would then take this message and pass it on to the game.

To start creating the client, a constructor was first developed. It took in a parameter called String host, which represented the server's IP address. It also took an integer port which represented the port in the server that had an open socket connection and was able to accept the client. A "getInstance" method was also defined to allow the game to create an instance of the client which it can continuously refer back to in its methods, without having to consistently create a new client.

```
//constructor
private Client(String host, int port) {
    this.host = host;
    this.port = port;
}

public static Client getInstance() {
    if (instance == null) {
        instance = new Client( host: "localhost",  port: 1255);
    }
    return instance;
}
```



```
//Connecting to the server and starts a new thread
public void connect() {
    try {
        socket = new Socket(host, port);
        toServer = new ObjectOutputStream(socket.getOutputStream());
        fromServer = new ObjectInputStream(socket.getInputStream());
        listener = new EventListener();
        new Thread( target: this).start();
    } catch (ConnectException e) {
        System.out.println("Server1 connection unsuccessful");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Creating the connection was the next step in the process. To start with, all the connection statements had to be kept in a try and catch block to be able to catch any errors that may occur during connection. Firstly, the socket that would send the output stream of data, and also receive the stream of data was assigned using the host and the port of the server. After that, variables for the corresponding data stream were assigned. Variable "toServer" was used for data being sent to the server, and variable "fromServer" was assigned for data being received from the server. Additionally, an event listener object was created and assigned to the variable listener (The class Event listener will be discussed later in depth). A new thread was also created to start running the following code.

```java
@Override
public void run() {
    try {
        running = true;
        while (running) //Listens to data from the server and collects it
            try {
                do {
                    Object data = fromServer.readObject();
                    listener.received(data);
                    System.out.println(data);
                } while (!playerActive.equals("Quit"));
                close();
            } catch (ClassNotFoundException e) { //If the method is unable to get the data
                e.printStackTrace();
            } catch (SocketException e) { //If there's a problem with the socket connection while the Client is running
                close();
            }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

The objective of this block of code is to continuously check for any new data received from the server, and to interpret it using the Event listener class, as shown in the statement “listener.received(data)”. However, as soon as the player quits, the “close” function is called, closing all the connections with the server, as shown below:

The running boolean is also set to false so that after the program is closed, the connections are not made again. The client will also send a message to the server saying “Quit” to let the server know that the player has quit and that it should close its port. Finally, after the message is sent, the output, input and socket are closed.

```java
//Ends the connection when a player logs off
public void close() {
    try {
        running = false;
        write( object: "Quit");
        fromServer.close();
        toServer.close();
        socket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

As previously mentioned, the client sends messages to the server, and to do this it needs to “write” to the server by sending an object as a packet. The method to the right utilizes the output stream to do so, and uses a try and catch method to catch an IO exception.

```java
//Send data to the server
public void write(Object object) {
    try {
        toServer.writeObject(object);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

An event listener class was also created to help aid the client program to interpret messages received from the server. This class only had one method, named “received”

which takes in an object packet as its only parameter and compares the strings that are expected from the server.

```
public class EventListener extends ButtonController {



    public void received(Object packet) throws IOException {


        //Checks the packet received from the server and what to do with it
        if (packet.equals("Password is correct")) { // If the password is correct
            Client.getInstance().setPasswordCorrect(true);


        }
        if (((String) packet).equals("Password is incorrect")) { //If the password is incorrect
            Client.getInstance().setPasswordCorrect(false);
        }
```

An example of a string we expect from the server would be a “Password is correct” message. The server would send this message after it has received the username and password entered by the user and it would validate their identity. The method then sets the boolean that checks whether the password is correct to true, using its setter function so that the game can progress onto the next screen. On the other hand, if the password is incorrect then the server would send a similar message and set the boolean to false, prompting the game to show the error page and informing users that their password is incorrect.

To bridge the connection between the client and the graphical user interface, the methods to send data to the server were created in the client, so they could be called in between screens whenever specific buttons were pressed. These methods allow specific data to be sent to the server, in the example shown to the right, “request Login” takes in two parameters, and

```
public void requestLogin(String username, String password) {
    String account = "LogIn---" + username + "---" + password;
    try {
        write(account);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void requestRegister(String username, String password) {
            String account = "SignUp---" + username + "---" + password;
            try {
                write(account);
            } catch (Exception e) {
                e.printStackTrace();
    }
}
```

4.4 Server

4.4.1 Implementation and Management

The goal of our system design is to enable multiple users to connect to the server and issue certain instructions to and from the client . Moreover, the server is build to follow the instructions , obtain the data from the database and send the message to the client.

The main functions in a server were designed and implemented with the certain specifications in mind. First, the TCP protocol was used to connect to the client using socket. Second, we used the threadPool to handle multiple clients. Third, after client connected to the server, the server follows the do-while loop structure to keep listening to the client and execute operations in the database according to the instructions it hears. Another class is using the threads to handle game's waiting queues and the running process of the game.

The first thing that the server should do is to connect to our database . Therefore,we use the Interface of JDBC to connect our PostgreSQL database.

Firstly, the functions was implemented, using the socket for the TCP protocol to transmit the message in the transport layer. Ports in a range from 1024 to 49151 were chosen at will. Ports are divided into two broad categories called fixed port and dynamic port (0~1023). Since these ports are tightly tied to some services, we often scan these ports to determine whether the other party such as TCP (FTP), (HTTP), (NetBIOS), UDP 7 (Echo), (TFTP) has turned on these services. These ports are not pinned to a service, which dynamically assigns them to each process, and two assignments of the same process are likely to be assigned to different ports.

To add, the rational implementation of thread pools can bring lots of benefits for the systems. First of all, it reduces the resource consumption caused by the thread creation. Secondly, it improves response speed. When a task arrives, the task can be executed immediately without waiting for the thread to be created. The last but not least, it improves the manageability of threads. Therefore, using thread pools for unified allocation, tuning, and monitoring is essential in handling every need of the client. The picture below shows the initialization of thread pools as well as the function for in-game waiting queues.

```java
/**
 * Constructor of Server
 * <p>
 *     Generate a ThreadPool that can handle clients and a waiting queue
 * </p>
 * @param port The server port
 */
public Server(int port) {
    this.serverPort = port;
    this.threadPool = Executors.newFixedThreadPool( nThreads: 100);
    this.wq = new WaitingQueue();
}
```

To continue, ClientHandler class controls a panel in a server to handle each client. Whenever a instance of this class is generated, it automatically opens the ObjectInputStream and OutputStream. It facilitates the management of code, and has a corresponding close method to shut it down. In the network system, we transmit the meaningful string between a client and a server. After receiving the message from client, server splits the string into a string array by using a "---" indicator. Furthermore, the an index[0] of string array represents the instructions. For example, if the login message received from client is "Login---username---password". The server executes the specified method according to the specified instructions. The message loops until the quit() command is received and the quit method is executed. In that case, the ClientHandler disconnects from the client and closes itself to all the streams.

```java
String[] message;
do {
    String ms = getListener().listen();
    System.out.println(ms);//Display the message in Server
    message = ms.split( regex: "---");
    System.out.println(message[0]);//Display the Command in Server
    switch (message[0]) {

        case "LogIn":
            logIn(message);
            break;
        case "SignUp":
            signIn(message);
            break;
```

```java
} while (!message[0].equals("Quit")) ;
quit();
```

Moving to another class, a WaitingQueue class acts as a waiting queue for a game. This queue is running under the principle of “FIFO” (First in First out) based on the property of queue in data structure. In the other words, the player who is in front of the

queue starts the game first. Once number of players in the queue is equal or larger than 2, additional players are removed from the queue. The game is then executed by a “new Thread” function.

```
@Override
public void run() {
    while (true) {
        if (this.queue.size() >= 2) {
            ClientHandler ch1 = this.queue.remove();
            ClientHandler ch2 = this.queue.remove();
            System.out.println("Successfully starting the game");
            Game game = new Game(ch1, ch2);
            ch1.write( object: "GameMatch");
            ch2.write( object: "GameMatch");
            ch1.run();
            ch2.run();
            new Thread(game).start();
        }
    }
```

4.5 Implementation of a game (server related)

4.5.1 Implementation and Management

The implementation of a quiz game is primarily in a server. This mainly includes the function of sending the quiz questions to the players and recording the results of the game between two players.

The architecture of a quiz game consists of two classes, the first is a class called “Game” and the other is a class called “Question”. In the constructor of a “Game” class, two ClientHandler functions were used to be able to send and receive the specified information to the two players. In addition, when you set up each game object, you will get five objects (question and four answers) from the database, and then use the ArrayList <Questions> to store these. It can largely avoid connecting the database multiple times and result in delays. After both player answer the question within the time, ClientHandler automatically sends the next question until they finish all of the questions required. From the start to the end of the game, the game results are transmitted to the server based on the scores of the two players, and the winner's score is updated.

```
public Question(String question, String[] options, String correctAnswer){
    this.question = question;
    this.options = options;
    this.correctAnswer = correctAnswer;
}
```

```
public static ArrayList<Question> getQuestionAndAnswer() {
    ArrayList<Question> questions = new ArrayList<~>( initialCapacity: 5);
    try (PreparedStatement selectStatement = connection.prepareStatement(
            sql: "select * from question_bank order by random() limit 4;")) {
        ResultSet resultSet = selectStatement.executeQuery();
        while (resultSet.next()) {
            String question = resultSet.getString( columnIndex: 2);
            String option1 = resultSet.getString( columnIndex: 3);
            String option2 = resultSet.getString( columnIndex: 4);
            String option3 = resultSet.getString( columnIndex: 5);
            String option4 = resultSet.getString( columnIndex: 6);

            String[] options = new String[4];
            options[0] = option1;
            options[1] = option2;
            options[2] = option3;
            options[3] = option4;
            String correctAns = resultSet.getString( columnIndex: 6);

            System.out.println(question);
            System.out.println(option1);
            System.out.println(option2);
            System.out.println(option3);
            System.out.println(option4);
            System.out.println(correctAns);
            questions.add(new Question(question,options,correctAns));

            //return questions;
        }
        return questions;
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}
```

Using the "Question" class to store the information can make the game structure more clear. A “Question” class has three constructors named question, options and correctAnswer. Since server simultaneously sends questions and options with the correct answer to the client, the client is able to check the answers without accessing to the server when players submit their answers.

```
public Question(String question, String[] options, String correctAnswer){
    this.question = question;
    this.options = options;
    this.correctAnswer = correctAnswer;
}
```

4.6 Database

Implementation and Management

In this project, jdbc which is specified as the **PostgreSQL 42.2.5** was implemented to connect and execute the query with the database. In order to manage tables and modify data more efficiently, we took advantage of **IntelliJ Datagrip** to act as a visual data management for us.

4.6.1 Principle of design

Our database design is designed according to the principles of **ER model**. Each table is tailor-made according to the practical needs of application. The verbs in the middle of each table represent the functionality of our application. In some tables, username in player table acts as foreign keys, which in turn makes it easy for us to find other relevant data about the player, such as score and times of victory, and so on.

The table below shown that when we need to use in the database:

No.	Functional Requirements	Parameters in attract
1	Login in for the exist player	username,password
2	Sign up for the new player	username,password
	When signup succeeds, the player's game record is automatically generated	Username, score=0;
3	Viewing the Leaderboard	username,score
4	Get the question from the question bank	Question,4 options ,correctAnswer
5	Upload Game Results	Winner’s username,his score

Table 1: Requirements and its parameters

In accordance with the above conditions, four tables were built in the database to manage the related data. These tables, which shown below, are player, game, player_score and question_bank tables.

Database Table	The Description of a Table
game	Contains information about the game
player	Contains player's unique information
player_score	Contains information about score of players
question_bank	Contains information on questions and answers for the quiz game.

Table 2: Database table and its description

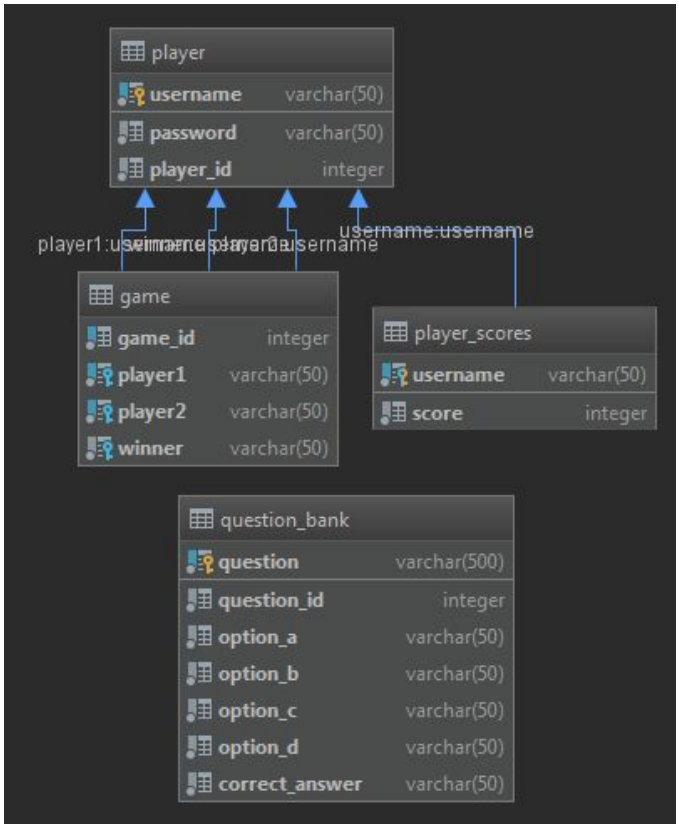


Figure 3:The Parameter structure of tables

4.6.2 Entity Relationship Diagram

The entity relationship diagram is demonstrated below to show the relationship among the tables and its values.

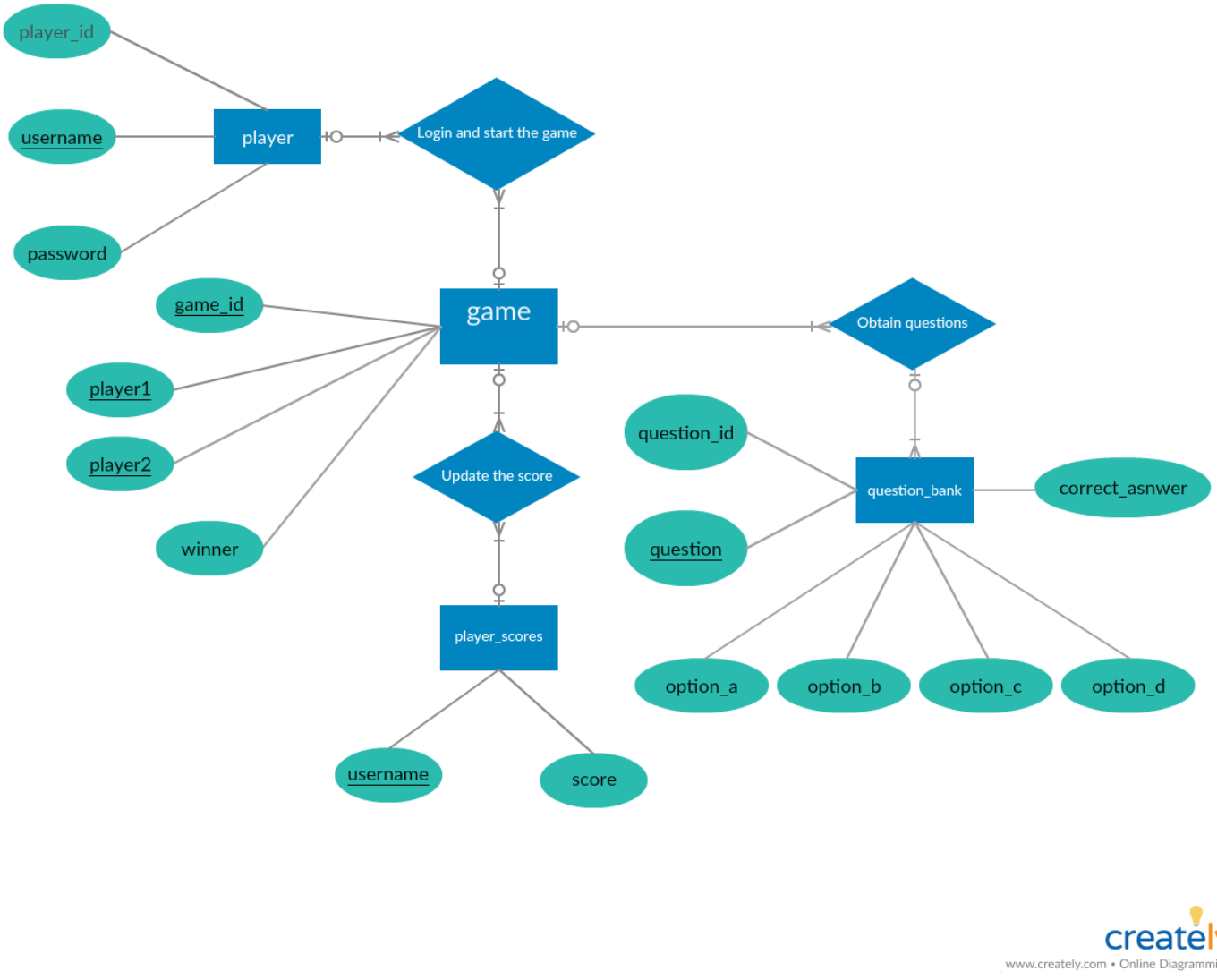


Figure 4: ER-model for the quiz game

5 Test Plan

The purpose of the test plan is to verify the functionality of the quiz game application and provide milestones to give us a reference to the process. Among these test case, we test every possible situation. Through the test of these conditions, we can reduce bugs of the program. In the following tests, we include most of the circumstances caused by the operation or system of the user.

No.	Property tested	Expected result	Observed result
1	Login: correct username + correct password	successful login	successful login
2	Login: incorrect username + correct password	unsuccessful login	unsuccessful login
3	Login: correct username + incorrect password	unsuccessful login	unsuccessful login
4	Login: incorrect username + incorrect password	unsuccessful login	unsuccessful login
5	Signup: unique username + unique password	successful signup	successful signup
6	Signup: unique username + password that is already in database	successful signup	successful signup
7	Signup: username that is already in database + password	unsuccessful sign up	unsuccessful sign up
8	Signup:(username + password) that is already in database + password	unsuccessful sign up	unsuccessful sign up
9	Display the username and relative score after login	successful	
10	Display the players who are top 5 players in total scores in the screen of leaderboard	successful	
11	Player successful add into the waiting queue	successful	
12	Player successful remove from the waiting queue	successful	
13	When the size of the waiting queue is equal or larger than 2, the game will start for those two players	successful	
14	Game: timer is synchronized for both players	both players have the same remaining time to answer	

15	Game: timer is reset after each question	both players have 10 seconds at the beginning of the next question	
16	Game: one player answers question faster then the other player, faster player has to wait	faster player will not be displayed new question unless the second player has answered ot the time limit has run out	
17	Game: both players answer the question before time limit	next question appear immediately after their submission	
18	Game: one player answers question correctly	player gains 100 points	
19	Game: both players answer question correctly	each player gain 100 points	
20	Game: one player answers question incorrectly	player gains 0 points	
21	Game: player does not answer within 10 second time limit	player gains 0 points	
22	Game: player does not answer within 10 second time limit	next question appear after time limit	
23	Game: same question will not appear in the game	the same question will not appear in the rest of the questions	
24	Game: one player disconnects	the other player still plays until all questions are finished	
25	Game: one player disconnects	the awarded points for answered questions are allocated to the leadership board	

26	Game: one player disconnects	disconnected player is loser	
27	Game: one player disconnects	questions for the disconnected player still run but due to time limit, the players receives 0 points	
28	Game: both players disconnects	points from both players are allocated to the leadership board	
29	Game: both players disconnects	questions for both disconnected players runs and both of them will receive 0 points due to time limit	
30	Game: comparing players points	player with highest points is winner	
31	Game: comparing players points	2 players with equal points => draw situation	
32	When the game finished,one of the players win the game,server upload the game result with “winner” to the game table.	successful	
33	When the game finished,the game result is draw.,server upload the game result without “winner” to the game table.	successful	

Table 3: Test Cases

6 Evaluation

Until now, our software has implemented login and signup functions, but some features have not been implemented. This is due to the mismanagement of in the beginning of the project. During the stage of project planning, we used to have two versions of the game that we tried to combine into a harder version, but the technical requirements were too high, causing group to give up on the harder version and chose a simpler version instead. Therefore, in the future, we would assess the risk and personnel abilities that will make it possible for members to choose a difficulty that can be achieved.

Secondly, although each person had a clear division of labor, communication was a big problem. Miscommunication resulted in inability to combine separate tasks into a single working program. Furthermore, due to the slow individual progress, the overall project wasn't able to continue further.

In the future, we realized that breaking the task even smaller is helpful. Even if there's no connection between the tasks and some of the tasks are not completed, others can continue to work without compromising the efficiency of the whole team.

Our programming consisting of client and server encountered lots of problems in the beginning. Group members also tried to ask classmates and tutors for help. We've been revisiting it all times, under their guidance. We found that some of the problems are due to the lack of overall planning of the basic system. In fact, in the project, we should have first planned the architecture of the application, and then proceeded on to coding, so as to reduce systematic errors.

Finally, we found the limitations of this project. Since at first everyone had their own division of labor, such as database, GUI, client and server, team members were able to focus only on their own parts. However, once integration problem occured, members were not able to understand and solve the problem. We became aware that in the early stage of project, after a certain degree of understanding, team members should have constantly work together in understanding each others' code. This would allow other teammates to assist even if the problem was in the specific section. In addition, this will greatly assist us in controlling the entire project progress with more details.

7 Team Organization Report

Members of a group: *Man Khi Kim, Hiu Kit Leung, Kunal Singh, Oluwaseun Akindollre, Nguyen Van Tiep, Lizzie Pickthall.*

7.1 Responsibilities and Duties

Groups below were assigned according to their preferences as well as the difficulty of the task.

TASK	**INDIVIDUAL/SUBTEAM**
Programming of the Graphic User Interface (G.U.I.)	MAN KHI KIM Nguyen Van Tiep
Programming of Clients and Servers	Kunal Singh(Client) HIUKIT LEUNG(Server)
Creation of Database; programming connection to the Server	HIU KIT LEUNG, Lizzie Pickthall, Oluwaseun Akindollre
Team Secretary/Primary Tutor Communication	MAN KHI KIM
Programming of Game Overall coding process	HIU KIT LEUNG, Man Khi Kim, Nguyen Van Tiep, Oluwaseun Akindollre, Lizzie Pickthall.
Writing a report	HIU KIT LEUNG, Man Khi Kim, Nguyen Van Tiep, Oluwaseun Akindollre, Lizzie Pickthall.
Brainstorming Process	HIU KIT LEUNG, Man Khi Kim, Nguyen Van Tiep, Oluwaseun Akindollre, Lizzie Pickthall.
Meeting Minutes	2400 minutes

8 Project Diary

Hours below do not include a time in a tutorial session.

Date: February 26th, 2019

Duration of the meeting: 3 hours.

State & Accomplishments: Two additional members were added to the group. Group was introduced to a new idea by two new members. An idea of a new quiz game was created. Basic functionalities of the game were discussed. Report is assigned as the work to do at home after the meeting.

Date: March 1st, 2019.

Duration of the meeting: 2 hours.

State & Accomplishments: Each member was assigned a specific task until the next meeting. Specific members were assigned to the GUI, database, client, and a server. Report is still being edited at this point.

Date: March 11th, 2019.

Duration of the meeting: 2 hours.

State & Accomplishments: Members were told to proceed on writing the introduction and description of the game in a report. Pages of the GUI interface were discussed in detail. Everything from transition of pages to the design, as well as the required fields were determined at a meeting.

Date: March 14th, 2019.

Duration of the meeting: 2 hours.

State & Accomplishments: Code for the server was finished during the meeting. The client was still having some issues. The GUI interface was created, but not connected to the buttons and functions were assigned. Information to be included in a database was finalized during the meeting.

Date: March 15th, 2019.

Duration of the meeting: 9 hours.

State & Accomplishments: Server wasn't interracting properly with a client. GUI was completely finished including all of the functions in each screen, but there was no connection established between the GUI and the client. Database was also finished and tables that were populated according to the previous discussion. The team spend a great amount of time to work and improve the errors and address the bugs.

Date: March 16th, 2019.

Duration of the meeting: 7 hours.

State & Accomplishments: Client was completely finished and the server connection was working properly with a client. The sign in part of the gui wasn't working in the beginning of the meeting, but the team managed to fix the problem at the end. Minor mistakes in code and bugs were perfected further during the whole meeting. The report was also written during the meeting.

Date: March 17th, 2019.

Duration of the meeting: 7 hours.

State & Accomplishments: The registration feature and the connection to database were fixed. Now, each member was able to run the code properly and create an account for the game using the server and the database. GUI was improved further with pop-ups and certain exceptions to the errors. At this point server, client, database, and GUI are all interacting properly with each other.

Date: March 19th, 2019.

Duration of the meeting: 5 hours.

State & Accomplishments: The game screen was worked on and the team was trying to figure out how to put the message from the database to display on a screen. The GUI was simplified to a smaller amount of classes that take more functions but are easier to control. The report was completely finished by end of the day.

Date: March 21th, 2019.

Duration of the meeting: 3 hours.

State & Accomplishments: The game has a working prototype at this point. Minor bugs and problems were fixed further. The testing part of the code takes place during the meeting.

9 Contribution of each member

Members of a group: *Man Khi Kim, Hiu Kit Leung, Kunal Singh, Oluwaseun Akindollre, Nguyen Van Tiep, Lizzie Pickthall.*

Every team member verbally agreed on the percentages below due to the inability of signing the digital document.

Man Khi Kim - 22%

Hiu Kit Leung - 26%

Kunal Singh - 18%

Oluwaseun Akindollre - 11%

Lizzie Pickthall - 9%

10 Bibliography

1. Fu,Yong-gui (2018) The credit game on network supplier and customer based on big data. Electronic Commerce Research, Vol.18 (3), p.605-628

2.HENNING, MICHI(2009) API Design Matters. Communications of the ACM, Vol.52 (5), p.46-57

3.McBride, Matthew R(2007)THE SOFTWARE ARCHITECT.Communications of the ACM, Vol.50 (5), p.75-82

4.Kumar, B.(1980)Computer System Design Using a Hierarchical Approach to Performance Evaluation.Communications of the ACM, Vol.23 (9), p.511-522

5.ERIK WILDE(2008)Document Design Matters.Communications of the ACM, 2008, Vol.51 (10), p.43-50

6.Janson, Marius A.(1958)Prototyping For Systems Development: A Critical Appraisal.MIS Quarterly, Vol.9 (4), p.305-317

7.Malone, Ian(1996)Getting real about client-server design.
Business Communications Review, 1996, Vol.26 (1), p.31-35

8.Hyeon-Gyu Cho(2004)A client-based logging technique using backward analysis of log in client/server environment.Journal of Systems & Software, 2004, Vol.72 (3), p.455-467

9.Azuma, Kazuhiro(2005)Design, implementation and evaluation of resource management system for Internet servers.Journal of High Speed Networks, 2005, Vol.14 (4), p.301-317

10.Poulter, Dale(1998)3-Tier Client/Server at Work.Internet Research, 01 December 1998, Vol.8(5)

11.McBride, Matthew R.(2007)THE SOFTWARE ARCHITECT.Communications of the ACM, 2007, Vol.50 (5), p.75-82

12.Reeves, Leah(2004).Guidelines for multimodal user interface design
Communications of the ACM, Vol.47(1), pp.57-59