



UNIVERSITY OF BIRMINGHAM

Software Workshop 2

Real Time Multi-User Quiz System

Benjamin Crispin,
Samuel Farmer,
Deedar Fatima,
Rowan Stringer,
Josh Wainwright

Group Osaka

Supervisor: Joe Gardiner

Date: March 2014

Contents

Contents	2
1 Protocol	3
2 Client	4
2.1 System Design	4
3 Server	5
3.1 System Design	5
4 Graphical User Interface	6
5 Database	8
5.1 Tables	8
5.2 Entity-Relationship Diagram	9
5.3 Normalization	10
6 Test Plan	11
6.1 Client Test Plan	11
6.2 Client Test Plan	12
6.3 Server Test Plan	14
7 Evaluation	16
7.1 Team Organisation	16
7.2 Evaluation of project work	16
7.3 Evaluation of project process	17
Class Diagram	18
Database Design	21

1 Protocol

The protocol for communicating between the different parts of the system is based around objects. There exists an object class that can be created for any of the several message types that could be needed to transfer information from the server to the client, or from the client to the server. Each of these objects implements the `Serializable` interface, allowing them to be converted to bytes and transferred across the socket connection using object streams.

The primary purpose of each of the objects used is described below.

AnswerResponse

To respond to a question, the Student selects the desired option. This information is passed to the server using an `AnswerResponse` object which simply holds the response and the timestamp to indicate when they made the selection.

DisplayQuestion

In order to signify that the allotted time for the current question has ended, and the next question should be displayed, the server sends a `DisplayQuestion` object to all of the clients and they should move on to the next question in the `Quiz` object and change the GUI accordingly.

LoginReply

Once a `loginRequest` has been received by the server, a `LoginReply` will be sent back. This gives the client the information about the requested login, most importantly if the login was successful, as well as the type of user that made the login, Student or Admin. This information is used to display the correct user interface.

LoginRequest

This is the first object that could be created. It is sent, by the client, to the server and contains the username of the Student that is attempting to login and the `java.lang.String` hash code of the inserted password. Though the security concerns of such a trivial system are non-existent, the password is never stored in plaintext.

Question

There exist several question objects in each `Quiz` object. They store the information required to present a Student with a question and the possible answers. Again, there is very little functionality as the question only serves as a wrapper for the question text and the possible answers that the user could respond with.

Quiz

This is the most important object. It has very little functionality, simply acting as a wrapper to hold and easily transfer several `Question` objects.

QuizRequest

When an Admin has successfully logged into the system, they have the option of starting one of the quizzes held in the database. This object is passed to the server and contains information about the quiz that the Admin has chosen to start.

Score

When the quiz has been completed, each of the clients will display the position of its user relative to the other Students. This object contains the score of all of the users so that each of the clients can work out where they are in the ranking.

StartQuiz

Once the user has successfully logged into the system, the next major event is the start of the quiz. This is signaled by an Admin user who is connected to the server, this information

must then be relayed to each of the connected clients. A `StartQuiz` object is sent to each of the clients, who, on receiving it, will display the first question to the user.

2 Client

The client exists to accept messages sent by the server, and present them in an order and a format that the user interface can present to the user, as well as accept the information entered by the user into the user interface and pass it to the server.

2.1 System Design

1. Login

When a client is started, a `QuizClient` object is created and starts the main loop. The initial stages set up the connection with the server and waits for the user to login. When the user enters their information, a `LoginRequest` object is created and pass to the server containing the username and password, to be checked against the contents of the database. The client then waits for a reponse from the server to indicate if the login was successfull or not. This comes in the form of a `LoginReply` object. If this says that the login was unsuccessful, the user is asked to re-input their details, otherwise, the display is changed and the options screen is shown.

2. Student/Admin

There exists separate functionality withing the client depending on if the user is a Student user, i.e. going to be answering questions, or an Admin, i.e. the teacher who starts and moderates the quiz. Distinguishing between these two is done by the server by checking the details associated with that user in the database. The `LoginReply` object contains this information and the client can then display the correct interface depending on the type of user that logged in.

3. Client Listens from Server

From here, the user can select the “Start Quiz” option to start the quiz. This causes the display to change to display the waiting screen and the client waits for information from the server.

From this point on, the client waits for any object to be sent from the server and acts according to what that object was. The possible objects that the client now expects to be able to distinguish between are:

- `Quiz`
- `StartQuiz`
- `DisplayQuestion`
- `Score`

Quiz

This object contains the information about the quiz itself, the number of questions, their contents and the duration that each question should be displayed for. It should only ever be recieved once by the client, at the start of the session, reducing the transfer of information over the connection.

StartQuiz

Once an Admin has logged in, they have control over the start of the quiz. When they decide to start the quiz, this object is sent to each of the listening clients and so the client will proceed to show the first question from the **Quiz** object.

DisplayQuestion

The first question is displayed as soon as the **StartQuiz** object is received. After this point in the quiz, the questions are changed when this object is received. The value contained verifies which question is to be displayed.

Score

After each question has been answered, the client can display a leader board showing the score of all the clients that have so far answered the current question along with the current client's position in this list. This object tells the client the relevant information for displaying the scores of the other clients.

4. Sent by Client

There are also a number of objects that the client can create and send to the server at different stages of the quiz:

- **LoginRequest**
- **QuizRequest**
- **AnswerResponse**

QuizRequest

This is sent by the client when an Admin is logged in in order to request a particular quiz from the server. Since the server can hold many quizzes, each with their own set of questions and answers, the Admin has the option to choose which of these to play when they log in.

AnswerResponse

This is the object that tells the server what answer the Student gave. It contains their response, so that it can be logged in the database, and the time it took for the Student to make their selection.

3 Server

The server exists to create a link between the database and the quiz program, as well creating connections with, and processing requests from, clients.

3.1 System Design

1. Initialising the server

When the server starts, a **QuizServer** object is created. The object creates a connection with the database by taking advantage of the inbuilt Java Database Connectivity (JDBC) functionality, which allows the server object to retrieve, and update, information contained in the database. In addition to this, a **ServerSocket** object is created, which waits to receive connections from **QuizClient** objects. When a connection from a client is received, a new **ClientThread** object is created.

2. Database Connectivity

The server interacts with the database through static methods contained in the `QuizJDBC` class. The class allows the server to establish a connection with the database through a `getConnection` method, which returns a `Connection` object.

A second method, `isUser`, is called by the server when it receives a `LoginRequest` object from a client thread. The method returns a `LoginReply` object containing the results of the query, which is sent to the client.

The final method, `getQuiz`, is called when the server receives a `QuizRequest` object from a client, and returns a `Quiz` object.

3. `ClientThread` Objects

When the server establishes a connection with a client, a `ClientThread` object is created, spawning a thread which allows interactions to occur between the server and client. The server then waits to receive a `LoginRequest` from the client, and returns a `LoginReply` object upon receiving it. Once the client has logged in, the server distinguishes between Student and Admin users.

4. Admin Clients

If a connection to an Admin has been made, the server waits for a `QuizRequest` object. Once received, the server retrieves the quiz in the database through the `QuizJDBC` class, then distributes this to all connected Student clients.

5. Student Client

If a student user has logged in, their instance of `ClientThread` on the server will wait until the Quiz has been set to ready by an admin. Once a quiz has been started, the server sends the `Quiz` object to all connected student clients, and then waits to receive an `AnswerReponse` from each of the Student client threads. The `AnswerReposnse` objects contain the clients response to a question, and the time that it took for them to answer it. The server then calculates the clients score, and the results are stored in an `ArrayList`, which contains the results of all connected Student clients. This `ArrayList` of scores is sent to the connected clients every time it is updated, allowing users to see an ‘updating’ results table which contains all student clients scores.

6. Quiz Time

In order to carry out admin tasks on the server without disrupting the main thread which is listening for connecting clients, an inner class `QuizTime` which extends `Thread` was created. The thread loops through during the quiz session, checking certain conditions, to which it should respond with specific tasks. For example, it checks whether no clients are connected, in which case the quiz stops. It also checks whether the first question of the quiz has finished being asked, in which case further student clients should not be allowed to join the quiz.

4 Graphical User Interface

The Student and Admin users will interact with a graphical user interface to take and run the quiz. The model deals with the data so the GUI can just be a visual interaction and representation of the client model. Instead of having many pop ups for each of the GUI panels, we decided to have one frame per client in which its content panel changes depending on the current part of the quiz.

1. Model/View Separation

Originally we decided that the GUI would follow a model/view separation format where the model will be separate from the client. This, however, caused some problems so we decided to not have a separate model and use the client as the model instead. One of the problems we encountered was that the GUI would interact with the model and create an object that needs to be sent to the server. We found it tricky to find a way to send objects from the model to the client and realized that there was a lot of redundancy as objects had to go through the model and the client when really there only needed to be one.

2. GUI Panels

Each GUI panel used in the quiz has its own class. When a client is created, each of the GUI panels is instantiated and added to an array. To accomplish this we used inheritance. Each of the GUI panels extends the abstract class `MasterFrame`, and in turn, `MasterFrame` extends `JPanel`. Each of the GUI panels is added to the `guiElements'` `MasterFrame[]`. `MasterFrame` is an abstract class with one abstract method `resetDisplay()`. This method is used to reset the contents of the current panel to an updated version of the model. Whenever the content panel of the frame is changed, the `resetDisplay()` method is called to ensure correct information is displayed.

The overall GUI will display different panels depending on which part of the system is currently being used. The GUI panels are:

`LoginFrame`

The first panel users will log-in to (both Admin and Student). `LoginFrame` contains a username field, a password field and login button. When the login button is activated, it send the contents of the two fields to the client model to create a `LoginRequest` object using the `requestLogin()` method.

`StudentHomeFrame`

The panel a Student will see once they have logged in / received successful login reply object. Here they can push the Start quiz button which adds them to a pool of Students ready to start the quiz. This leads them to the `WaitingFrame` panel using the `requestWaitingScreen()` method.

`AdminHomeFrame`

The panel the Admin will see once they have logged in / received successful login reply object. Here they can see which Students have connected and joined the waiting pool to start the quiz. `AdminHomeFrame` has a `JComboBox` which lists the possible quizzes to run. This uses the `setCurrentQuizID()` method. The Admin can then push the start quiz button which sends the selected quiz to all the waiting Students using the `adminStart()` method.

`WaitingFrame`

The panel a Student will see when they have pushed start and are waiting for the Admin to start the quiz. Contains a message to the Student letting them know they are waiting for the Admin to start the quiz as well as a `JProgressBar` in intermediate state.

`QuestionFrame`

The panel which displays the question to the Students. This panel is only displayed for a limited time as each question has a short time limit. The question is displayed in a `JTextPane`, and each of the four possible multiple choice answers is displayed as a `JButton` beneath. To the side there is a countdown timer which visually displays the time to answer the question. As soon as the question is answered or the time has run out, the panel is changed to the `StudentResultsFrame` panel. When a button is pushed, the `setResponseNumber(i)` method is called. If a Student doesn't answer, then the response number is set to -1.

StudentResultsFrame

Once a Student has completed a question, they are displayed with this panel. **StudentResultsFrame** displays the last question with highlighted text for the correct answer and Students answer. It also displays the score received for the last question and current leader board for the quiz where each client is ranked by their total score so far. This panel is shown for a set amount of time before leading back to the **QuestionFrame** for a new question.

AdminResultsFrame

The **AdminResultsFrame** is displayed once the Admin has started the quiz. It displays the current question Students are completing and a live leader board showing the current scores of the Students. The leader board ranks the Students by their current total score from the completed questions so far.

FinalResultsFrame

This is similar to **StudentResultsFrame** but is the final panel the Student sees in the quiz. It shows the correct answers to the previous question and the score for the previous question. It also shows the final leader board rankings. Unlike **StudentResultsFrame**, there is no time limit so the Student can see the final results as long as they wish. There is a back button to return the Student to the **StudentHomeFrame** which uses the `returnHome()` method.

Other GUI components that we have created and used within the GUI panels are:

CountDownTimer

CountDownTimer is GUI component that counts down seconds. It shows a **JProgressBar** decrease as the time decreases. Also has a label beneath showing the seconds left to count down. Used in **QuestionFrame**.

LeaderBoard

A table consisting of each Student in the quiz, their position and score. This is updated whenever the `resetDisplay()` method is called in its parent panel (**MasterFrame**). Used in **StudentResultsFrame**, **AdminResultsFrame** and **FinalResultsFrame**.

5 Database

A database is an important part of any system which is designed to provide a mechanism for storing, managing and retrieving information. PostgreSQL, an object-relational database, was considered for the project. The two reasons behind choosing PostgreSQL were—availability on School's system and familiarity with the database model. The database designer considered business rules and processes during requirements analysis and came up with an initial draft of database model. The model was reviewed with the team and some modifications were done to accommodate the system requirements. The server used to create database was dbteach2. A new database, called osakagp, was created to store the data needed for Edify quiz.

5.1 Tables

The four tables created in osakagp DB are **quiz**, **questions**, **users** and **users_result**.

Users

The user login details are stored in the users table. The user details are inserted into this table when a new user or Admin registers. The login credentials entered by the users are

validated and the users are allowed to login only if the entered credentials exist in the users table.

Quiz

The quiz topics are stored in this table. The quiz could be on the following topics—Politics, Sports, History, Geography, Music, and Science and Technology. The Admin chooses the quiz topic from quiz table and fetches the topic-related questions from the questions table.

Questions

The questions table contains the questions which are answered in quiz. Question and possible answers are stored as rows in the questions table. The table also contains a separate column for quiz ID. The Admin uses the quiz ID to get the questions for the chosen quiz topic.

User_result

This table contains the quiz results for all the Students. It is loaded with quiz result once the quiz is completed. The user can see the result by quiz date, quiz topic, score and quiz status.

5.2 Entity-Relationship Diagram

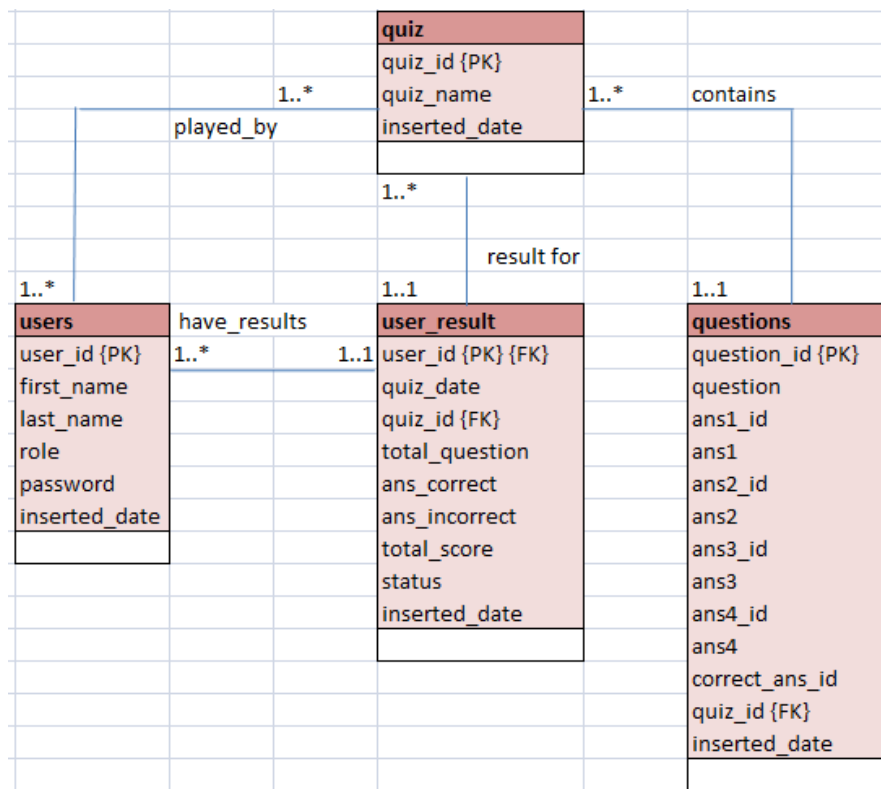


Figure 1: This is the entity-relationship (ER) diagram for the tables used in Edify quiz. The relationship shared among these tables is explained in Section 5.2.

5.2.1 One-to-One

- The table `questions` shares a one-to-one relationship with `quiz` table. Each question in the `questions` table can appear only in one quiz category.
- The table `user_result` shares a one-to-one relationship with `users` and `quiz` tables. The `user_result` table stores the result of all the quizzes played by the Students. Each row in the `user_result` table is linked to one quiz in the `quiz` table and one user in the `users` table.
- The `user_result` table shares one-to-one relationship with `users` table. Each row in the `user_result` table is linked to one user in the `users` table.

5.2.2 One-to-Many

- The `quiz` table shares one-to-many relationship with `questions` table. Each quiz topic has 10 questions in the `questions` table.
- The `quiz` table shares one-to-many relationship with `users_result` table. Each quiz topic in the `quiz` table is linked to multiple rows in the `users_result` table as quiz result is stored for multiple players and can be played on multiple days.
- The `quiz` table shares one-to-many relationship with `users` table. One quiz can be played by multiple users in the `users` table.
- The `users` table shares one-to-many relationship with `quiz` table. Each user in the `users` table can play one or more quiz in the `quiz` table.
- The `users` table shares one-to-many relationship with `user_result` table. Each user in the `users` table can have one or more quiz results in the `user_result` table.

5.3 Normalization

Data normalization was implemented to reduce and eliminate data redundancy.

First Normal Form

All the tables are in 1NF as they contain only atomic values and there are no repeating groups of data.

Second Normal Form

All tables are in 1NF and all of their non-key attributes are fully dependent on their primary keys. There is no partial dependency of any column on primary key.

Third Normal Form

All the tables are in 1NF and 2NF and all non-key attributes are fully functional dependent only on the primary key.

6 Test Plan

6.1 Client Test Plan

Test Date	Required Functionality	Modifications Made
17/03/14	The client is able to create Socket, through which it establishes a connection with the server	None? worked straight away.
20/03/14	The client is able create an Array containing the GUI frames	This was included, along with a method, which allows the Frame to be updated based on the current state of the client.
17/03/14	The client is able to send and receive objects from the server	None? this worked straight away, although exception handling was needed to include this functionality.
17/03/14	The client is able to send a LoginRequest to the server containing information that has been input by the user	The username and password are now stored as Strings in order to maintain consistancy across the system.
17/03/14	The client is able to receive a LoginRequest and differentiate between a student and admin	The LoginReply object contains a boolean, which is set to true for student users. This value is checked, and the correct HomeFrame is displayed based on the result.
17/03/14	The client remains on the LoginFrame until correct login details have been entered	The JDBC creates a LoginReply containing a boolean value which is set to true if the database contains the login details. This boolean is then chekced by the client.
21/03/14	The client is able to maintain a list of connected users	This was achieved by creating an ArrayList instance variable, which stores the username of the client, and their current score in an active Quiz. This is updated through the server, which sends an ArrayList to the client after all connected clients have completed a question.
17/03/14	An admin client is able to create and a send a QuizRequest object	Made sure that all Quizzes were complete in the database, so that the admin client could select from a full range of available quizzes.
18/03/14	A student client is able receive DisplayQuestion objects, which changes the GUI frame to QuestionFrame	This was completed by modifying the server, so that the 10 DisplayQuestions objects are sent to each connected client. As each DisplayObject is received, the client calls a method to udpate the frame to the QuestionFrame.

18/03/14	The client is able to send an AnswerReponse to the server containing the time taken to answer a question	This is handled in a method which returns an AnswerResponse object. The client obtains a time stamp upon receiving a DisplayQuestion, and calculates the time between this and a user selecting an answer. The AnswerResponse object that is created contains the selected answer and the time taken to respond.
22/03/14	The client is able to receive and display an ArrayList containing the scores of all student users	Whilst the client waits for the next DisplayQuestion to be received, it listens for an ArrayList which contains the scores of all connected clients. The ArrayList then passed to a LeaderBoard object.

6.2 Client Test Plan

Test Date	Required Functionality	Modifications Made
17/03/14	The LoginFrame is displayed when a client connects to the server	None? worked straight away.
17/03/14	The LoginFrame remains in view until the user has entered correct login details	None? worked straight away.
17/03/14	The frame is updated to either AdminHomeFrame or StudentHomeFrame depending on the user	This was implemented by the client switching to the relevant screen depending whether or not they were an admin or student client according to the LoginReply? isStudent was added as a field to LoginReply in order to provide this functionality.
17/03/14	A student user is able to click a 'Start Quiz' button which takes them to the WaitingFrame	None? worked straight away.
19/03/14	An admin user is able to see connected students on the AdminHomeFrame	This is a work in progress? while it is possible to add a list of connected students to this screen, since a record of connected students is not updated when a student leaves, it does not currently update when a student disconnects.

	17/03/14	An admin user is able select and start a Quiz from their home frame	Originally when the admin pressed the start button, the GUI found the currently selected item in the quiz choice ComboBox and then sent this to the server, although this was unreliable. In order to fix this issue, the currently selected quiz was maintained as a variable on the client. This variable was initialised on the creation of the ComboBox to the first element in the list, and updated each time it was changed. Then, when the admin clicked start, it was this variable that was sent to the server as part of the QuizRequest.
	17/03/14	A DisplayQuestion screen contains a countdown timer displaying how long the user has left to commit an answer	None? worked straight away.
	17/03/14	Clicking on an AnswerButton in the DisplayQuestion screen creates an AnswerResponse object	None? worked straight away.
	19/03/14	Clicking on an AnswerButton takes the user to a ResultsFrame	This did not work straight away, but was resolved by fixing an issue between the server and the client (model)? the client was not receiving the AllScores object, therefore did not update the display to show the results.
13	23/03/14	The ResultsFrame displays a LeaderBoard containing all connected student users scores	This displayed only the latest iteration of the AllScores object, meaning that while it did display results for all users scores (they are added to the ArrayList with a score of 0 when they begin the quiz), this list did not update. See below for further improvement to this functionality.
	23/03/14	The LeaderBoard on the ResultsFrame updates as clients answer questions	Once the server sent the client the allScores data for each client updating to all clients, this functionality was possible. However, the LeaderBoard was initially reconstructed with each iteration of data, which caused unreliable behaviour in which it did not display the results for certain clients. This was fixed by adding a model to the Table.
	23/03/14	An admin user is able to see an updating LeaderBoard based on student reponses	This is a work in progress? the admin users must both be sent the AllScores objects and maintain their connection while the quiz is in progress for this to be possible, however this is not currently the case meaning that further work has to be done to be able to provide this functionality.

23/03/14	After quiz completion a FinalResultsFrame is displayed	Currently the standard results frame is displayed after the completion of the quiz, however the FinalResultsFrame class has been coded—meaning that it just needs to be integrated to provide this functionality. Tied to this functionality is resetting the quiz to the initial state to allow for multiple quizzes to be set sequentially, however this is a larger task that is a work in progress.
23/03/14	The final scores are displayed after the last question.	The final LeaderBoard is already displayed on the standard StudentResultsFrame for the last question, and therefore this functionality is already provided? no modifications required.
23/03/14	The FinalResultsFrame contains a button which takes users back to their HomeFrame	The FinalResultsFrame does contain this button, however the frame has not yet been implemented meaning that this is not currently functional.

6.3 Server Test Plan

Test Date	Required Functionality	Modifications Made
17/03/14	The server creates a ServerSocket object on port 9010	None? worked straight away.
17/03/14	The server accepts connections from a client and initializes a Client-Thread	This worked without a problem, however this functionality was altered so that the new client thread was initialised as a member of an ArrayList, in order to support the distribution of objects to all connected clients.
17/03/14	The server allows objects to be sent and received with the client through ObjectOutput and ObjectOutputStreams	None? worked straight away (had to incorporate exception handling to provide this functionality).
17/03/14	The server receives a LoginRequest object from client and queries the database with its contents	Changed username and password in LoginRequest to Strings, in order to maintain their data types throughout the system (GUI, client, server & database).
17/03/14	The server returns a LoginReply to a client containing the correct information	Altered the default values of IsStudent to true and loginSuccessful to false, in order to improve system security.
17/03/14	The server is able to distinguish between Admin and Student users	This is now checked using a JDBC PreparedStatement? assuming that the client is a student (the lowest level of privileges), unless they can prove otherwise (by having an admin account in the database).

18/03/14	The server is able to create a Quiz object, which is then sent to all connected clients	This is now done by retrieve the quiz for the admin client thread, storing it on the server as the current quiz, then distributing it to each of the student clients connected when it is set to be ready.
21/03/14	The server is able to send 10 questions to connected clients	Quizzes were updated to each have 10 questions in the database in order to facilitate this functionality.
23/03/14	The server is able to synchronize Questions so that clients receive them at the same time	Since the last student client to answer the question (by running out of time or otherwise) increments the next question to be displayed, it was necessary to implement a check where student clients ensured that they were not answering the same as the previous question on each loop. Student clients are allowed to bypass this condition on the first question since there is no comparison to the previous question to be made.
17/03/14	The server is able to receive AnswerResponse objects and calculate Student scores	This functionality worked, however did not support the updating of the leaderboard.
22/03/14	The server is able to keep a running track of all connected Student Scores	This functionality was implemented by synchronising updates of the AllScores object, which is static ArrayList of Score object stored on the server.
22/03/14	The server is able to send each iteration of the AllScore object updates to all student clients.	On updating their score in the AllScores object, a client calls a Server class method that sends the AllScores object to all connected StudentClient threads. The clients are added to an ArrayList of StudentClients as soon as they log in, and it possible to distinguish that they are students clients.
19/03/01	The server is able to handle clients disconnecting during a quiz	Failed? in order to provide this functionality it is necessary to maintain a list of currently connected clients that is updated as clients disconnect. This issue is a work in progress.

7 Evaluation

7.1 Team Organisation

The group consists of 5 students. Since there were a number of distinct sections to the project, the different components were distributed among the members of the team with one person in a position of responsibility for that component and another to provide assistance. The allocations were as follows.

Component	Responsible	Assisting
GUI	Benjamin Crispin	Deedar Fatima
Client	Josh Wainwright	Rowan Stringer
Server	Rowan Stringer	Benjamin Crispin
Database	Deedar Fatima	Sam Farmer
Protocol	Sam Farmer	Josh Wainwright

These roles were followed closely during the initial stages of development and through the first round of testing. As the program became more complete and bug fixes were required, the roles were shared more evenly through the team. This has the advantage that all members have a full understanding of all aspects of the project having worked on all parts at some time.

7.2 Evaluation of project work

1. Correctness and reliability

We set out to create an interactive live educational quiz and accomplished a reliable working project. We have had some coding problems throughout the project but worked as team to come up with solutions. One of the major problems we have encountered was updating the student results leader board. Whenever a student answers, their score is added to an array list in the server and this should be distributed to every other student so that their results screen is accurate. The problem when implementing was that the first student to answer would not receive the updated score array list as other students answered.

This was corrected by having the students constantly listening for new score array lists and insured an up to date leader board. This also lead onto other problems such as students not being synchronised on the same question. The cause of this was as each student requested a new question, the question counter was incremented and students skipped questions. This was solved by having a counter so that the question would only be incremented once all students had requested the new question.

2. Performance

There were issues with performance with a live quiz that we had to be concerned with. After the users have logged in and the quiz has started, the questions must be synchronised so that each student is answering the same question. This is accomplished through the server sending the question to each client, rather than the client being individually responsible for the current question. The questions are reliably sent to all students at the same time so that quiz is synchronised. When a student answers, they are directed to the results screen where they need to see an updated version of the leader board. As mentioned in **Correctness and reliability** above, a solution for this was found and works well.

3. Usability

The system works well and is designed so that users find it easy and intuitive to use. We decided on a standardised quiz format where each quiz has 10 questions, and each question has 4 multiple choice answers. The design of the GUI is intuitive so that users don't have

to learn how to use the system. A user should immediately recognise how to use the system and not have to rely on previous knowledge to answer quizzes. Originally the GUI for the questions had text boxes displaying the multiple choice answers with buttons next to them. We gave the system to fellow students in the lab to test and found that they tended to want to push the multiple choice text boxes rather than the buttons. Therefore, we decided to scrap the text boxes and put the multiple choice answers directly as the button text.

4. Substantiates

Much the originally functionality that we set out for we have accomplished. The users can successfully log in and are displayed the right screen depending on if they are a student or admin. The admin can register students, and students can view their profile. The admin can select a quiz and this is sent to the student. Students can take the quiz and see a live leader board based on the ranking through out the quiz. The admin can also see a leader board of the students live results. Extra functionality we would wish to include would be for the admin to be able to create quizzes and for these to be stored in the database and to be run for students.

7.3 Evaluation of project process

1. Management of team work

Initially the project was split into five sections with each member of the group volunteering to be responsible for a section. In addition, each member was also assigned a section to assist on. Rather than appointing a team leader to distribute the work amongst the group, the tasks to be completed were discussed during bi-weekly team meetings.

2. Time keeping and scheduling

Group meetings were held twice weekly; one early in the week to discuss and distribute the tasks that were to be completed for the following week, and a second meeting later in the week to discuss the current progress and any problems that were being had. Each task that was to be completed was assigned a deadline. Throughout the duration of the project, these deadlines were adhered to by all group members.

3. Subdivision of work

During the initial stages of the project, each member was assigned specific tasks that were to be completed by a certain deadline. By adhering to these deadlines, the code for each of the five portions were completed within the first two weeks. The remainder of the time was then spent on integrating the code and fixing the bugs that inhibited the system from meeting the software specification. During this time, a list of tasks to be completed was produced and group members would systematically work through this list, either individually or in smaller subgroups consisting typically of two or three members. This required effective communication between each of the group members, which was achieved through regular submissions to subversion and whole group emails.

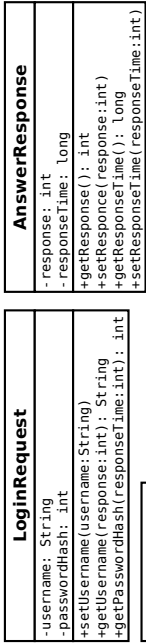
4. Integration

After the first few weeks of the project we decided to integrate the code each team member had written. This took a while to complete and we realised that some modifications had to be made. One of the modifications was changing the client/model/view to model/view separation as stated in Section 4. We spent a day integrating different parts of the project where all group members were present. This had the advantage of all group members knowing about all parts of the system and made the next few weeks of working and communication flow much more smoothly.

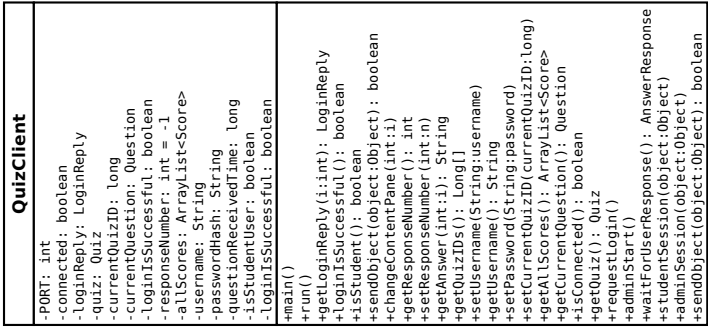
References

- [1] Elliottte Harold. *Java Network Programming*. O'Reilly Media Inc., 2013.
- [2] Harvey Deitel and Paul Deitel. *Java: How to Program*. Pearson, 2002.
- [3] Scott Oaks and Henry Wong. *Java Threads*. O'Reilly Media Inc., 1999.
- [4] Xingchen Chu Rajkumar Buyya, S Thamarai Selvi. *Object-Oriented Programming with Java*. Tata McGraw Hill, 2009.
- [5] Kenneth L. Calvert and Michael J. Donahoo. *TCP/IP Sockets in Java*. Morgan Kauffman, 2001.

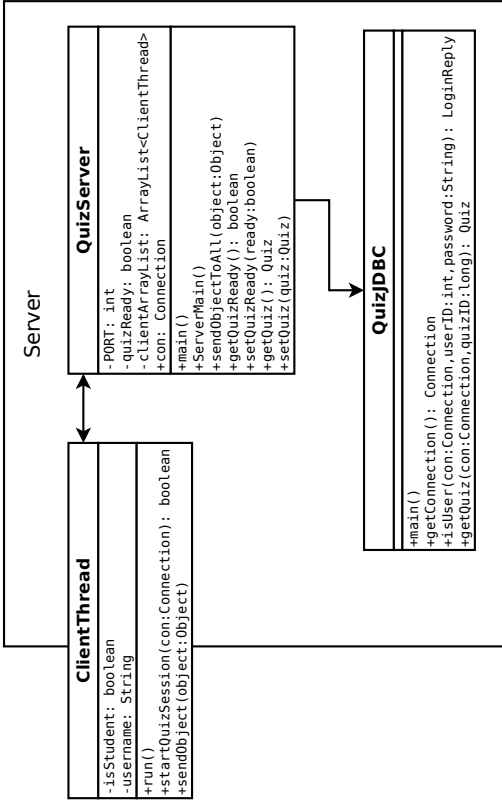
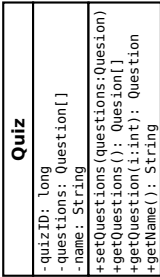
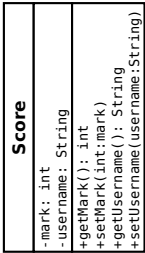
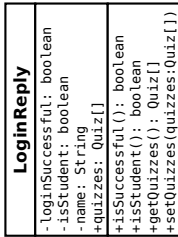
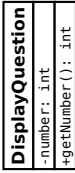
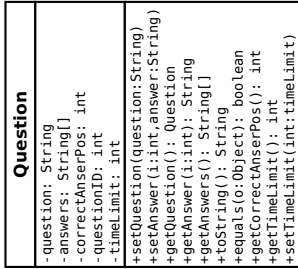
"Client → Server" Protocol



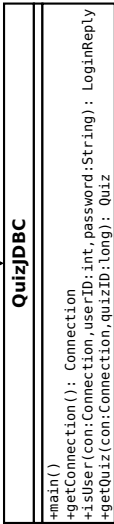
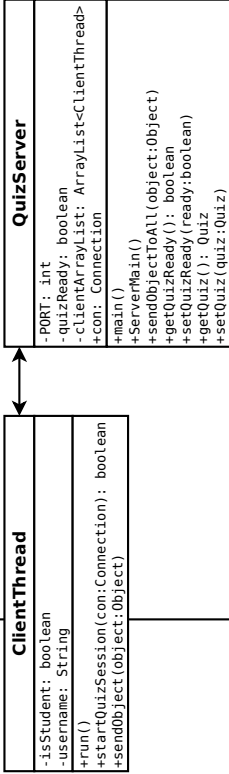
Client



"Server → Client" Protocol



Server



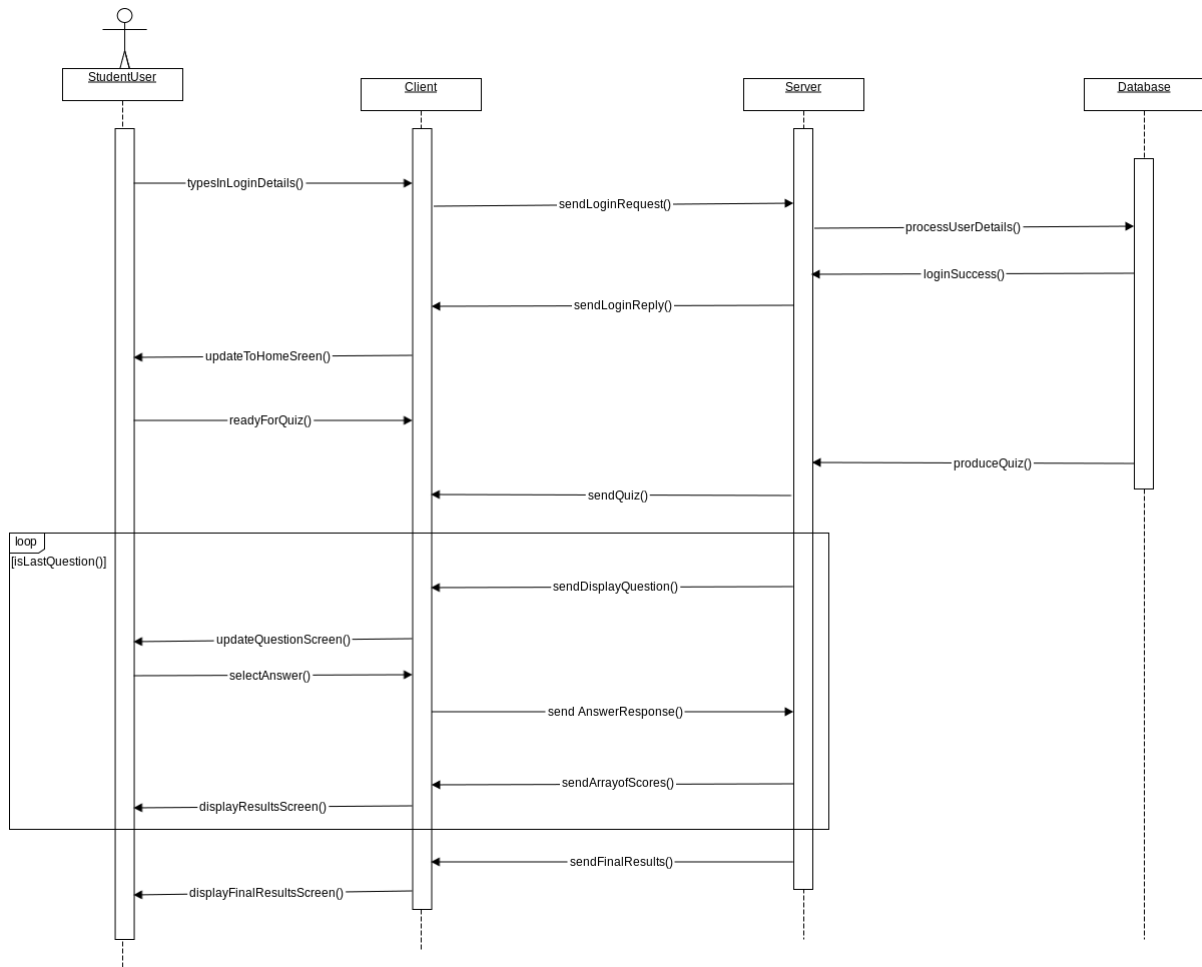


Figure 2: Student scenario diagram.

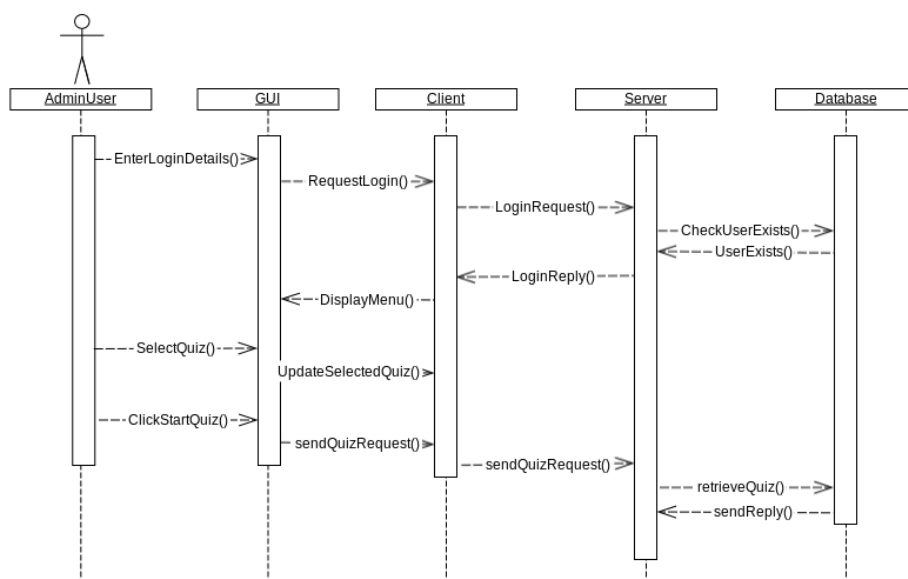


Figure 3: Admin sequence diagram.

Table name	users			
Attribute	Description	Type	Nullability	Example of values
user_id	Unique ID of admin/student	BIG INT	NOT NULL	Between 1 and 9223372036854775807
first_name	First name of admin/student	VARCHAR (20)	NULL	Mary
last_name	Last name of admin/student	VARCHAR (20)	NULL	Ande
role	Role of user	VARCHAR (20)	NULL	admin or student
password	Password entered by admin/student to access the tool	VARCHAR (10)	NULL	
inserted_date	Timestamp of the transaction	TIMESTAMP	NOT NULL	DEFAULT is the current timestamp.
Primary Key	user_id			
Foreign Key				
Index	user_id			

Table name	quiz			
Attribute	Description	Type	Nullability	Example of values
quiz_id	Unique ID of quiz	BIG INT	NOT NULL	Between 1 and 9223372036854775807
quiz_name	Topic of quiz	VARCHAR (40)	NULL	Politics, Sports
inserted_date	Timestamp of the transaction	TIMESTAMP	NOT NULL	DEFAULT is the current timestamp.
Primary Key	quiz_id			
Foreign Key				
Index	quiz_id			

Table name	questions			
Attribute	Description	Type	Nullability	Example of values
question_id	Unique ID of question	BIG INT	NOT NULL	Between 1 and 9223372036854775807
question	The question to be answered by students	VARCHAR (200)	NULL	In which country is the Albert canal?
ans1_id	ID of first possible answer	INT	NOT NULL	DEFAULT is 1
ans1	First possible answer	VARCHAR (40)	NULL	Spain
ans2_id	ID of second possible answer	INT	NOT NULL	DEFAULT is 2
ans2	Second possible answer	VARCHAR (40)	NULL	Belgium
ans3_id	ID of third possible answer	INT	NOT NULL	DEFAULT is 3
ans3	Third possible answer	VARCHAR (40)	NULL	Canada
ans4_id	ID of fourth possible answer	INT	NOT NULL	DEFAULT is 4
ans4	Fourth possible answer	VARCHAR (40)	NULL	Portugal
correct_ans_id	The ID of correct answer	INT	NOT NULL	2
quiz_id	The ID of quiz	BIG INT	NOT NULL	4
inserted_date	Timestamp of the transaction	TIMESTAMP	NOT NULL	DEFAULT is the current timestamp.
Primary Key	question_id			
Foreign Key	quiz_id			
Index	question_id, quiz_id			

Table name	user_result			
Attribute	Description	Type	Nullability	Example of values
user_id	ID of user	BIG INT	NOT NULL	Between 1 and 9223372036854775807
quiz_date	Date on which quiz is played	TIMESTAMP	NOT NULL	DEFAULT is Current Timestamp
quiz_id	ID of quiz played by the student	BIG INT	NOT NULL	Between 1 and 9223372036854775807
total_question	Number of questions displayed in a quiz	INT	NULL	10
ans_correct	Number of questions answered correctly before any other student	INT	NULL	Between 0 and 10
ans_incorrect	Number of questions answered incorrectly	INT	NULL	Between 0 and 10
total_score	Number of questions answered correctly before any other student	INT	NULL	Between 0 and 10
status	If a student won or lost the quiz	VARCHAR(10)	NULL	WON or LOST
inserted_date	Timestamp of the transaction	TIMESTAMP	NOT NULL	DEFAULT is the current timestamp.
Primary Key	user_id			
Foreign Key	user_id, quiz_id			
Index	user_id, quiz_id			

Minutes of the
Team Osaka Project Meeting
UG04, Computer Science
March 04, 2014; 11:00-12:00
Minutes Taker: Rowan Stringer

Attendance

1. Rowan Stringer
2. Benjamin Crispin
3. Sam Farmer
4. Josh Wainwright
5. Deedar Fatima

Minutes

1. We need to hand in a more detailed written specification.
2. Get a paragraph on each section and splice together – add paragraph to Subversion as .txt file, Sam will splice together.
3. We need an object diagram.
4. We need a class diagram.
5. Deedar is responsible for JDBC (database connection) and Prepared Statements.
6. JDBC will be carried out in a separate server thread.
7. 4 responses possible for questions – so arrays will be used instead of ArrayLists.
8. Share ERD + document on databases on Subversion.
9. Short meeting for Thursday will be held at 14:30-15:00, like last week.
10. Sam will help out with JDBC
11. Deedar will form the SQL for the Prepared Statements.
12. Rowan is responsible for passing the quiz object to the client (clarifications):
 - Server notified with a question object
 - Client carries out timing
 - Server listens for results object (including username, answer, time)
 - Tell client to end connection their end, close connection, close thread.

Action Summary			
Action Specifics	Responsible	Deadline	Status
Hand in a more detailed written specification	Sam		Open
Get a paragraph on each section and splice together	All		Open
Create object diagram	-		Open
Create class diagram	-		Open
Give system JDBC functionality	Deedar, Sam		Open
Create PreparedStatements	Deedar		Open
Share ERD + document on databases on Subversion.	Deedar		Open

*** End of Minutes ***

Tutorial Notes

Team Osaka Project Meeting

245, Computer Science

March 06, 2014; 16:00-16:30

Notes Taker: Rowan Stringer

Attendance

1. Rowan Stringer
2. Benjamin Crispin
3. Sam Farmer
4. Josh Wainwright
5. Deedar Fatima
6. Joe Gardiner (Tutor)

Notes

- Protocols must be documented
 - Testing must be planned
 - Using JUnit
 - Manual tests where JUnit is inappropriate
 - Testing must be logged
 - Recording whether test cases were successful / failed
 - Recording what bugs were fixed
 - Working system by next Wednesday.
 - 2 weeks until report submission.
 - Need 1 week for testing / bug fixing.
-

Work Status

- Deedar:
 - Completed Actions:
 - Database is working
 - Upcoming Actions
 - JUnit test cases
 - JDBC / Prepared Statements
- Sam:
 - Completed Actions:
 - Server connections
 - Server searching through database
 - Upcoming Actions
 - Server obtaining quiz object
 - Aside note: On query, pass connection and close / leave open based on result of condition
 - JUnit testing should be done by only one person

- Josh:
 - Completed Actions:
 - Client talks to server
 - Client passes and receives objects
 - Upcoming Actions
 - Document protocol
 - Communication with GUI (ensuring right information is available)
 - Ben:
 - Completed Actions:
 - Some GUI screens are sorted
 - Upcoming Actions
 - All GUI screens will be done by Saturday (08/03/2014)
 - Communication with connected Client
 - Rowan:
 - Completed Actions:
 - Client side server code written to pass quiz data obtained from the server-database connection.
 - Upcoming Actions
 - Enforcing synchronisation between connected clients during quiz mode.
 - Passing on client result information to the server-database connection to be stored in the database.
-

- SVN repository currently in use was setup on Josh's account as project SVN repository was unavailable at the start of the project:
 - Ideally the project should be migrated to the new project repository
 - Logs should be exported as .txt files
 - Actual files should be copied not the SVN bit
- Command line will be supported (vs Eclipse plugins)
- Joe will provide support regarding connections to the database through Eclipse (check project setup)
- For connecting to the database from outside the school, use university VPN connection.
- Extra testing (MSc only):
 - Detailed test plan (document) is required, that is formal and comprehensive.
 - Tests must be carried out and results logged.
- Format of presentation:
 - 10 minute presentation / 20 minute demonstration / 10 minutes deciding mark
- See Joe Tuesday to check in with an update of project
- Add own work to Subversion, and provide log messages when committing (can do this by writing note in TortoiseSVN/ writing note in line above in cmd line while committing/ can use -n flag to write note in cmd line)

Action Summary			
Action Specifics	Responsible	Deadline	Status
Protocols must be documented.	-	13/03/2014	Open
Test plan must be documented	-	?	Open
Testing according to the plan must be logged	-	?	Open
System must be working	All	12/03/2014	Open
Report must be submitted	All	(Two weeks)	Open
Migrate project SVN directory to one provided	-	?	Open
Presentation must be prepared	All	?	Open
See Joe Tuesday to check in with update of project	-	11/03/2014	Open

*** End of Notes ***

Tutorial Notes

Team Osaka Project Meeting

245, Computer Science

March 13, 2014; 16:00-16:30

Notes Taker: Rowan Stringer

Attendance

1. Rowan Stringer
2. Benjamin Crispin
3. Sam Farmer
4. Josh Wainwright
5. Deedar Fatima
6. Joe Gardiner (Tutor)

Notes

- Wait / Notify vs while loop
- Could do a read while in the while loop condition to make it wait
- Answers should update on the server
- GUI needs some work – “pretty”
- Colour change on admin when students answer
 - Countdown screen (big)
 - Control screen + who has answered
- Students see updating leaderboard
- Report due by Monday + prepare presentation
- Reallocate work
 - If database done, give that person new work
 - If connections done, get that person to work on the GUI
- Comment code
- Stability is important
 - Shows good testing has been implemented
 - A crash during the presentation can impact heavily on marks
- Documentation
 - Design:
 - Write what you did
 - Technologies you used
 - How it works
 - Testing:
 - JUnit
 - Manual
- Joe is absent Tuesday until Friday next week
- Short meeting with Joe Monday next week

*** End of Notes ***

**Minutes of the
Team Osaka Project Meeting
UG04, Computer Science**

February 21, 2014; 10:00-11:00 (approved on February 22)

Minutes Taker: Rowan Stringer

Attendance

1. Rowan Stringer
2. Benjamin Crispin
3. Sam Farmer
4. Josh Wainwright

Agenda

1. Extra team member
2. Project documentation
3. Scoping out project and idea
4. Research
5. Allocations
6. Selection of Software Engineering Paradigm
7. Project Outline
8. Next Meeting
9. Any Other Business
10. Time of Meeting End

Minutes

- 1. Extra team member:**
 - Notify Deedar about meeting.
 - Send Deedar the minutes.
 - Add Deedar to WhatsApp group.
- 2. Project documentation:**
 - Write down project proposal down as soon as possible to avoid losing a week. Deadline of Monday.
 - Use latex for final deliverables (to give ability to use version control like subversion), if possible.
 - Team will use UML where appropriate.
- 3. Eliciting project requirements:**
 - Creating a *use case diagram* as a team to determine broad system requirements.
 - Brief system description: the system is a live educational quiz in which students compete in real time to answer questions correctly. It will implement a client-server model with connected computers. There will be two user classes, henceforth referred to as Admin and Student. The system will implement sockets and threads (which avoid races/deadlocks) where appropriate and will have a graphical user interface (GUI).

4. Research:

- The team should research other existing systems in order that the project solution can improve on and differ from currently existing systems.
- It is important to keep log of bibliography references during the project as this will be a useful resource at the end of the project.

5. Allocations:

- Rowan (rjs305@bham.ac.uk) will act as the team email liaison with the tutor.

6. Selection of Software Engineering Paradigm:

- Due to the project multiple streams running concurrently, and unclear initial requirements, an iterative model (such as an Iterative Waterfall Model) will be adopted to reduce the amount of planning required and allow changing requirements to be met in a natural progression.

7. Project Outline (in bullet point form):

- A real-time educational team quiz system
- The system allows Students to logs into a user account
- The system allows logged in Students to compete in a many on many in a multiple choice / simple answer quiz
- The system gives a set number of questions per quiz (as per the admin)
- The system ensures synchronicity between all clients and the server
- The system allows users to log on with past question results
- The system displays a leader board and some feedback about the question to the Student
- The system displays a basic statistical information during the quiz to the Admin
- The system displays a summary report of statistical information at the end of the quiz to the Admin
- The system will allow the Admin to add or remove questions
- The system will log information about the Student's quiz session

8. Next Meeting:

- Next meeting will be arranged via WhatsApp group in response to the Joe's feedback.

9. Any Other Business: – NIL.**10. Time of Meeting End:**

- The meeting was adjourned at 11:00.

Action Summary				
ID	Action Specifics	Responsible	Deadline	Status
1	Send a project outline (bullet points) to Joe	Josh	24/02/2014	Open
2	Notify Deedar about meeting, send her the minutes, add her to WhatsApp group	Rowan	21/02/2014	Open
3	Arrange next meeting via WhatsApp group in response to the Joe's feedback	Rowan	27/02/2014	Open
4	For the next meeting, have a bibliography reference about sockets and/or threads	All	Next Meeting	Open

*** End of Minutes ***

Minutes of the
Team Osaka Project Meeting
Student Common Room, Computer Science
February 24, 2014; 14:30-14:55 (approved on February 24)
Minutes Taker: Rowan Stringer

Attendance

1. Rowan Stringer
2. Benjamin Crispin
3. Sam Farmer
4. Josh Wainwright
5. Deedar Fatima
6. Joe Gardiner (Tutor)

Agenda

1. Present an informal project outline to tutor
2. Any Other Business
3. Time of Meeting End

Minutes

1. Present an informal project outline to tutor (outcomes)

- An outline time schedule must be created for the project
- Animations in GUI should be considered in design, such as:
 - i. a digital countdown to each question start
 - ii. a leader board that that updates in real time
- Either threads or a push-data model can be considered in order to provide 'real-time' feedback functionality
- Students should receive more points for faster answers
- The server should be designed to be robust in the circumstances of client connections dropping out, i.e. session should continue
- Division of labour:
 - i. JDBC and socket programming should done by all group members to some extent
 - ii. No more than 2 people per task.

•

2. Any Other Business:

- Next Meeting to be arranged for tomorrow.

3. Time of Meeting End:

- The meeting was adjourned at 14:55.

Action Summary				
ID	Action Specifics	Responsible	Deadline	Status
5	Time Plan (which includes testing) must be created	All	27/02/2014	Open
6	Some sketches for GUI must be created (this task could be split over modules of project with couple sketches each)	All	27/02/2014	Open
7	Arrange next meeting for 25/02/2014	Rowan	25/02/2014	Open

*** End of Minutes ***

**Minutes of the
Team Osaka Project Meeting
UG04, Computer Science**

February 27, 2014; 14:30-15:00 (approved on February 24)

Minutes Taker: Rowan Stringer

Attendance

1. Rowan Stringer
2. Benjamin Crispin
3. Sam Farmer
4. Josh Wainwright
5. Deedar Fatima

Agenda

1. Discuss GUI Sketches (that have been made)
2. Any Other Business
3. Time of Meeting End

Minutes

1. Discuss GUI Sketches

- Creating a list of GUI screens required for sketching:
 - i. Login Screen – same for student and admin – Ben
 - ii. Create quiz for admin – Ben
 - iii. Home Screen for Admin - Deedar
 - iv. Home Screen for Student - Deedar -> Keep frames to a minimum unless simple functions. Need a lobby for connected clients to wait in before quiz starts. Begin with harder user coding – need to know question IDs to delete them
 - v. Question Screen – 4 multiple choice options, with countdown timer, with an example question – Rowan -> just word question and answer to begin with. Randomise answers.
 - vi. Immediate results (Student) - Sam
 - vii. Immediate results (Admin) – Sam
 - viii. Final results (Student/Admin) – Sam

2. Any Other Business:

- Tutorial meeting in rm 245(?)
- How everyone is getting on with subversion.
- Next Meeting to be arranged via doodle poll for Friday.

3. Time of Meeting End:

- The meeting was adjourned at 14:53.

*** End of Minutes ***

Tutorial Notes

Team Osaka Project Meeting

245, Computer Science

February 27, 2014; 16:30-17:00

Notes Taker: Rowan Stringer

Attendance

1. Rowan Stringer
2. Benjamin Crispin
3. Sam Farmer
4. Josh Wainwright
5. Deedar Fatima
6. Joe Gardiner (Tutor)

Notes

Division of Labour should be (for creating and testing):

- Server – Control, most complex, need two people
- Database / JDBC
- Client – local state, connect with GUI
- GUI

Project is given a mark, and then a division of work document explains how team worked together.

A report of install instructions, troubleshooting document is required.

Want low latency in system.

Need to rework time plan if GUI will be created from the start, with a week to test.

When client logs in, something (like a string) will be required to facilitate the connection.

Server acknowledges, and can send back information.

Need a protocol which says what functionality component must provide – what situations will occur, and how component should act.

Need to plan test cases – not necessarily JUnit testing. Some will be set up a system, and check system behaviour. (E.g. might not want closing one window to close all Java windows). What happens if the client dies, server dies.

Run tests a few times to check for undefined behaviour.

Demonstrations will be on lab machines.

Project will be handed in with 3 paper copies of the report and subversion link – might be easier for us to give Uday and Joe access to our subversion rather than movement across to another subversion account.

No office hour on Monday – and Joe is out of contact all day Friday (this week).

Action Summary				
ID	Action Specifics	Responsible	Deadline	Status
19	Written specification needs to be written for next week.	-	06/02/2014	Open

*** End of Notes ***