

עבודה חקר בבניית משחק טריווית שחמט
בסביבת העבודה Visual Studio ובSQLite3.
כתוב בשפת C++ ושפת C#. משלב טכנולוגיות
של שרת לקוח ושמירת נתונים בעזרת
DataBase.

עבודת חקר מדעי המחשב

משחק טריווית שחמט

מנואל נמירובסקי

פרטי התלמיד:

שם התלמיד: מנואל נמירובסקי

תעודת זהות: 211338108

תאריך לידה: 8.4.2000

כתובת: החסידה 3ב', נתניה

טלפון התלמיד: 054-255-5687

פרטים כלליים:

שם בית הספר: הכפר הירוק ע"ש לוי אשכול

טלפון בית הספר: 03-6455666

מספר יחידות לימוד: 5 יחידות

תחום העבודה (מקצוע): מדעי המחשב

פרטי המנחה:

שם המנחה: יהודה אור

תעודת זהות: 023098007

תואר אקדמי: מהנדס מבנים. מוסמך הטכניון בהנדסה, הסבה אקדמאית למחשבים מטעם המדינה מאז שנת 2000. תעודה הסמכה מטעם Microsoft.

מקום העבודה: מכון וייצמן, מכללת ג'והן ברייס, הכפר הירוק, תיכון עירוני ד', ת"א.

טלפון: 050-734-4457

אי-מייל: yooda@gmail.com

תוכן עניינים

- 3.....מבוא על משחק השחמט ומשחק הטריז
- 4-13.....מבוא לתקשורת שרת – לקוח
 - מושגים והסברים כלליים
 - סוגי לקוחות
 - העברת מידע בין שרת ללקוח
- 14-16.....היכרות עם מבנה בסיס נתונים
- 17-19.....היכרות עם SQLite3
- 20-29.....פרויקט מעשי - כתיבת משחק "טריז שחמט"
 - מבוא – מבנה הפרויקט
 - שלבי בניית הפרויקט
 - הצגת שימוש במוצר
- 30.....סיכום ומסקנות
- 31.....ביבליוגרפיה
- 32-98.....נספחים
 - נספח א' – טבלאות הDB של המשחק
 - נספח ב' – טבלת UML לשרת
 - נספח ג' – טבלת FLOW לפרויקט
 - נספח ד' – קוד לקוח
 - נספח ה' – קוד שרת

מבוא על משחק השחמט ומשחק הטריוויה

שחמט הוא ענף ספורט ומשחק לוח אסטרטגי המיועד לשני שחקנים. זהו אחד מהמשחקים השכיחים והמורכבים ביותר הקיימים בתרבות האנושית. המשחק מקובל ברחבי העולם כתחביב וכספורט תחרותי כאחד. אדם העוסק במשחק שחמט באופן מקצועי נקרא שחמטאי.

לוח השחמט מחולק ל-64 משבצות, בשמונה שורות הממוספרות מ-1 עד 8 ושמונה עמודות המסומנות א' עד ח', או A עד H (משמאל לימין), הצבועות בצבעי שחור ולבן לסירוגין (או, בכלליות, צבע בהיר וצבע כהה) - כל ארבע הצלעות של משבצת לבנה גובלות במשבצת שחורה ולהפך. מניחים את הלוח כך שהמשבצת הימנית ביותר בשורה הקרובה לכל שחקן היא לבנה.

טריוויה הוא מונח המשמש לציון פריטי מידע, לרוב לא חשובים במיוחד. לפעמים מציין המונח ידע כללי. מקור המילה באנגלית, כגזירה לאחר של שם התואר "טריוויאלי". מוצאה של המילה "טריוויאלי" מן השם הלטיני "טריוויום" ("שלוש דרכים"), הקבוצה הראשונה והפחות חשובה מבין שבע האומנויות החופשיות - דקדוק, רטוריקה ולוגיקה (ארבע האחרות, אריתמטיקה, גאומטריה, מוזיקה ואסטרונומיה נקראו קוואדריוויום).

הרעיון ליצור משחקים למחשב הובא מתי שאנשים יכלו לקנות לעצמם מחשבים ביתיים (Home Computers), בשנות ה-70. האנשים שמכרו את המחשבים רצו ליצור חוויה מסוימת לקונים שבה הם יוכלו להרגיש תחושת "התמכרות" למחשב. אחד הדרכים הייתה לשלב דברים שהיו קיימים ללא המחשב (כמו משחקי קופסא, משחקי קלפים וכדומה), ולשלב אותם לתוך המחשב, ובכך ליצור המשך שימוש רציף במחשב.

מבוא לתקשורת שרת – לקוח

מושגים נפוצים:

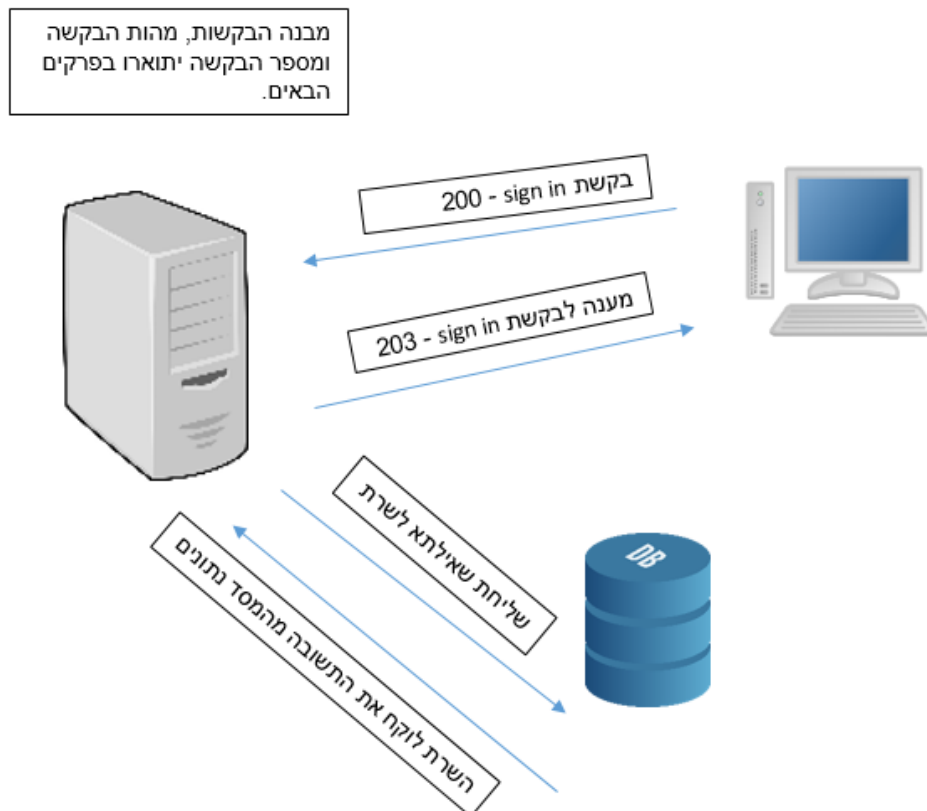
- נתונים: פרטי מידע הבסיסיים.
 - תקשורת נתונים: תהליכים ומערכות להעברת נתונים בין מחשבים או מכשירים אלקטרוניים אחרים, ללא העברה פיזית של אמצעי לאחסון נתונים, אלא באמצעות תווך כלשהו אשר משמש לתקשורת.
 - רשת מחשבים: מערכת המיועדת לתקשורת בין שני מחשבים או יותר. המחשבים מתקשרים דרך ערוצי תקשורת (ערוץ תקשורת יכול להיות גם אלחוטי) לפי פרוטוקולי תקשורת.
 - Session: החלפת מידע אינטראקטיבית חצי-קבועה, הידועה גם כ"דיאלוג", "שיחה" או "פגישה", בין שני מכשירים מתקשרים או יותר, או בין מחשב למשתמש קצה.
 - שרת: מחשב ותוכנה המותקנת בו המחברים לרשת מחשבים ותפקידם לספק שירותים שונים למחשבים ברשת. דוגמה לשרתים: שרת קבצים, שרת דואר.
- מודל שרת לקוח היא ארכיטקטורת תוכנה לחישוב מבוזר, אשר מגדירה את היחס בין תוכנות משתפות פעולה. המודל מחלק את המשימות או עומס העבודה בין ספק השירות או המשאבים - השרת, לבין מבקש השירות - הלקוח. שרת-לקוח היא אחת מתצורות ההתקשרות הנפוצות ברשתות מחשבים. השרת הוא תוכנה פסיבית, המאזינה לרשת ומחכה לקבל בקשות. הלקוח לעומתו בדרך כלל מהווה את ממשק המשתמש - הוא מופעל על ידי המשתמש ופונה לשרת כאשר הוא זקוק למידע או שירותים ממנו.
- בדרך כלל תוכנות השרת והלקוח רצות על גבי מחשבים שונים והתקשורת ביניהן מתבצעת על גבי רשת מחשבים, אך בעבודה זו התקשורת בין השרת לבין הלקוח תתבצע על אותה רשת פנימית, (LAN – Local area network), רשת הנמצאת על איזו גיאוגרפי מוגבל, והיא תתבצע על אותו מחשב.
- נהוג להפריד את הלקוחות ללקוח "רזה" (thin client), כזה שמסוגל רק לבצע את ההתחברות לשרת, והשרת הוא זה שמנהל את כל התקשורת עם המשתמש, ולקוח "עשיר" (rich client), כזה שמבצע את כל האינטרקציה עם המשתמש בעצמו, ופונה לשרת רק על מנת לאחזר נתונים. בעבר היה שימוש נפוץ במסופים, שהיוו לקוחות "רזים", התחברו למחשב מרכזי (MainFrame) וביצעו את כל הפעולות עליו, אך עם ההתקדמות המהירה של המחשבים האישיים הם נדחקו

הצידה בשנות התשעים ופינו את מקומם ללקוחות "עשירים", המקטינים את תעבורת הרשת ואת העומס על השרת. התפתחות האינטרנט והגידול ברוחב הפס החזירו את הלקוח הרזה למרכז התמונה.

בפריקט זה, השתמשתי בטכנולוגיית הלקוח "העשיר", מכיוון שהמשתמש לא יכול להתנהל מול השרת, שמעביר את המידע לבסיס נתונים. אם ללקוח הייתה האפשרות להתערב באלגוריתמי השרת, הלקוח היה יכול להכניס מידע שקרי ואף למחוק מידע. הלקוח בעבודה מציג את המשחק לשחקן, מעביר את המידע שהשחקן מכניס לשרת, והשרת מעביר את המידע לבסיס נתונים.

התקשורת מתוארת בצורה הבאה: הלקוח שולח בקשה לפי הפרוטוקול (יוצג בהמשך), השרת מפענח את הבקשה, בודק את הבקשה מול בסיס הנתונים (DataBase), ואחרי הבדיקה, מחזיק את התשובה ללקוח.

בדוגמא הבאה, ניתן לראות את התקשורת בין השרת לבין הלקוח כאשר הלקוח מנסה להתחבר.



כדי שהשרת והלקוח יתקשרו ביניהם בצורה נקייה ובלי הפרעות, לשרת וללקוח צריך להיות פרוטוקול תקשורת שמובן לשני הצדדים. פרוטוקול הוא בעצם "השפה" שבה מתקשרים השרת

והלוקוח. הלוקוח מעביר בעזרת הפרוטוקול בקשות (Requests) לשרת, והשרת מחזיר ללוקוח תשובות (Responses).

הפרוטוקול שבניתי לעבודה מוצג בטבלה הבאה:

- בכל ההודעות יש סוגריים מרובעים. זהו אינו חלק מהפרוטוקול, ורק לשם נוחות הם נוספו.

- אפור – הודעה מהלוקוח לשרת
לבן – הודעה מהשרת ללוקוח

מספר הודעה	מוען - נמען	מהות הבקשה	מבנה הבקשה והסברים
200	קליינט-סרבר	בקשת sign in	<p><u>תיאור מילולי:</u></p> <p>קוד הודעה = 3 בתים (כמחרוזת)</p> <p>מספר התווים של היוזר - שני בתים (כמחרוזת)</p> <p>שם משתמש</p> <p>מספר התווים של הסיסמה - שני בתים (כמחרוזת)</p> <p>סיסמה</p> <p><u>מבנה ההודעה:</u></p> <p>[200##username##pass]</p>
201	קליינט-סרבר	בקשת sign out	[201]
102	סרבר-קליינט	מענה לבקשת ההתחברות- sign in	<p>[1020] //success</p> <p>[1021] //Wrong Details</p> <p>[1022] //User is already connected</p>
203	קליינט-סרבר	בקשת הירשמות- sign up	<p>קוד הודעה = 3 בתים</p> <p>מספר התווים של היוזר - 2 בתים (כמחרוזת)</p> <p>שם משתמש</p> <p>מספר התווים של הסיסמה - 2 בתים (כמחרוזת)</p> <p>סיסמה</p>

			מספר התווים של המייל - 2 בתים (כמחרוזת) סיסמה [203##username##pass##Email]
104	סרבר- קליינט	מענה לבקשת הירשמות - sign up	[1040] // success [1041] // Pass illegal [1042] // Username is already exists [1043] // Username is illegal [1044] // Other
205	קליינט- סרבר	בקשת רשימת החדרים הקיימים	[205]
106	סרבר- קליינט	שליחת רשימת החדרים	קוד הודעה = 3 בתים (כמחרוזת) מספר החדרים - 4 בתים (כמחרוזת) [אם אין חדרים אז המספר = 0] <u>רשימת חדרים [במידה ויש חדרים...]:</u> מזהה חדר - 4 בתים גודל שם החדר - 2 בתים שם החדר [106 numberOfRooms roomId ## roomName roomId ## roomName...] (ההודעה האמיתית בלי רווחים. הרווחים כאן רק כדי להבהיר את חלקי ההודעה)
207	קליינט- סרבר	בקשת היוזרים של החדר.	קוד הודעה = 3 בתים (כמחרוזת) מזהה חדר - 4 בתים [207roomId]
108	סרבר- קליינט	שליחת רשימת היוזרים של חדר	קוד הודעה = 3 בתים (כמחרוזת) מספר שחקנים נוכחי בחדר - בית אחד

			<p>גודל שם יוזר - 2 בתים</p> <p>שם יוזר</p> <p>[108 numberOfUsers ## username ## username ...]</p> <p>אם החדר לא קיים (המשחק התחילוהמנהל סגר את החדר) - מספר המשתמשים יהיה 0 ולא ישלחו שמות יוזרים.</p> <p>בעצם ההודעה תהיה כזאת:</p> <p>[1080]</p>
209	קליינט-סרבר	בקשת הצטרפות לחדר קיים	<p>קוד הודעה = 3 בתים (כמחרוזת)</p> <p>מזהה חדר - 4 בתים</p> <p>[209roomId]</p>
110	סרבר-קליינט	מענה לבקשת הצטרפות לחדר קיים	<p>קוד הודעה = 3 בתים (כמחרוזת)</p> <p>קוד הצלחה\כישלון- בית אחד (0\1\2, כתו)</p> <p>במקרה של הצלחה:</p> <p>מספר השאלות - שני בתים</p> <p>זמן שאלה - שני בתים</p> <p>[1100 questionsNumber questionTimeInSec]</p> <p>//success</p> <ul style="list-style-type: none"> אחרי הודעת הצלחה השרת שולח ליוזר שהתחבר ולכל היוזרים המחוברים לחדר הודעת 108 (רשימת היוזרים של החדר) <p>[1101] // failed - room is full</p> <p>[1102] // failed - room not exist or other reason</p>
211	קליינט-סרבר	בקשת עזיבת חדר	[211]
112	סרבר-קליינט	מענה לבקשת עזיבת חדר	<p>[1120] // success</p> <ul style="list-style-type: none"> במקרה שהחדר כבר לא קיים (כי המשחק התחיל או שהאדמין סגר את החדר) - השרת כבר שלח לקליינט הודעת "סגירת חדר" - 116 או הודעת

			"התחלת משחק" - 119 (ואז לא תשלח גם הודעת 112).
213	קליינט-סרבר	בקשה ליצירת חדר חדש	קוד הודעה = 3 בתים גודל שם החדר - 2 בתים שם החדר מספר שחקנים - 1 בית (9-1 , יישלח כתו). מספר השאלות - 2 בתים זמן מענה לשאלה בשניות - 2 בתים [213##roomName playersNumber questionsNumber questionTimeInSec]
114	קליינט-סרבר	מענה לבקשת יצירת חדר חדש	[1140] //success [1141] //fail
215	קליינט-סרבר	בקשה לסגירת חדר	[215]
116	סרבר-קליינט	תגובה לבקשת סגירת חדר.	[116] יישלח לכל המשתמשים ששייכים לחדר כולל האדמין
217	קליינט-סרבר	התחלת משחק	[217] הודעה זו תשלח רק ע"י האדמין. במידה והצליח השרת ישלח לכל מי שמחובר לחדר הודעת 118 כמפורט בהמשך
118	סרבר-קליינט	שליחת שאלה עם ארבע תשובות אפשריות	קוד הודעה - 3 בתים גודל שאלה - 3 בתים שאלה [גודל תשובה - 3 בתים תשובה]- עבור כל תשובה [118 ### question ###answer1###answer2###answer3###answer4] במקרה של כישלון יישלח(נשלח רק לאדמין):

			[1180] כלומר ניתן לזהות שגיאה כשאורך השאלה הוא 0.
219	קליינט- סרבר	הלקוח שולח את תשובותו	קוד הודעה - 3 בתים (כמחרוזת) מספר התשובה - בית 1 (כתו) זמן תשובה - 2 בתים (הזמן שלקח לשחקן לענות) [219 answerNumber TimeInSeconds] במידה והשחקן לא ענה בזמן על השאלה מספר התשובה יהיה 5.
120	סרבר- קליינט	התייחסות השרת לנכונות השאלה	נכון/לא נכון :תו אחד - 01 [120 lastAnswerIndication]
121	סרבר- קליינט	הודעת סיום משחק	קוד הודעה - 3 בתים (כמחרוזת) מספר יוזרים - בית אחד (כתו) [גודל שם משתמש - 2 בתים שם משתמש] תוצאה -(מספר התשובות הנכונות) 2 בתים [121 usersNumber ## userName score ## userName score...]
222	קליינט- סרבר	הודעת עזיבת משחק	[222]
223	קליינט- סרבר	בקשת best scores	[223]
124	סרבר קליינט	מענה על בקשת best scores	קוד הודעה- 3 בתים [גודל שם משתמש - 2 בתים שם משתמש מספר תשובות נכונות - 6 בתים] נשלחים 3 יוזרים עם התוצאות שלהם.

			<p>במידה ויש פחות משלושה יוזרים שרשומים במערכת השרת ישלח עבור משתמש לא קיים את הפרטים הבאים : גודל שם משתמש=0 תוצאה=0</p> <p>[124 ## userName highestScore ## userName highestScore ## userName highestScore]</p>
225	קליינט- סרבר	בקשת "מצב אישי"	[225]
126	סרבר- קליינט	מענה לבקשת "מצב אישי"	<p>מספר המשחקים - 4 בתים</p> <p>מספר התשובות הנכונות - 6 בתים</p> <p>מספר התשובות השגויות - 6 בתים</p> <p>זמן מענה ממוצע לשאלה - 4 בתים (2 בתים ראשונים מייצגים את החלק השלם, 2 בתים אחרי את החלק העשירוני. לדוגמא : 0354 מציין שהזמן הוא 3.54 שניות)</p> <p>[126 numberOfGames numberOfRightAns numOfWrongAns avgTimeForAns]</p> <p>במידה ליוזר אין משחקים ותשובות יש להחזיר את ההודעה הבאה (0 עבור כל אחד מהשדות) :</p> <p>[1260000]</p>
299	קליינט - סרבר	יציאה מהאפליקציה.	[299]

כל הודעה בטבלה שולחת אחריה פרטים שיעזרו לשרת וללקוח לתקשר ביניהם ולהעביר מידע. לדוגמא, בשליחת בקשת ההתחברות (בקשה מספר 200), חוץ ממספר הבקשה, צריך לשלוח גם את מספר התווים של שם המשתמש וגם את מספר התווים של הסיסמא.

רשימת הקודים לתקשורת עם השרת נתונה בקוד מקור של הלקוח:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace trivia
{
    public class Codes
    {
        public const string SIGN_IN = "200";
        public const string SIGN_OUT = "201";
        public const string SIGN_IN_ANS = "102";
        public const string SIGN_UP = "203";
        public const string SIGN_UP_ANS = "104";
        public const string GET_ROOMS = "205";
        public const string GET_ROOMS_ANS = "106";
        public const string GET_USERS = "207";
        public const string GET_USERS_ANS = "108";
        public const string JOIN_ROOM = "209";
        public const string JOIN_ROOM_ANS = "110";
        public const string LEAVE_ROOM = "211";
        public const string LEAVE_ROOM_ANS = "112";
        public const string CREATE_ROOM = "213";
        public const string CREATE_ROOM_ANS = "114";
        public const string CLOSE_ROOM = "215";
        public const string CLOSE_ROOM_ANS = "116";
        public const string START_GAME = "217";
        public const string SERVER_QUESTION = "118";
        public const string SEND_ANSWER = "219";
        public const string SEND_ANSWER_ANS = "120";
        public const string GAME_OVER = "121";
        public const string LEAVE_GAME = "222";
        public const string GET_BEST_SCORES = "223";
        public const string BEST_SCORES_ANS = "124";
        public const string GET_MY_STATUS = "225";
        public const string MY_STATUS_ANS = "126";
        public const string QUIT = "229";

        public const int CODE_LEN = 3;
        public const int USERLEN = 2;
        public const int PASSLEN = 2;
        public const int EMAILLEN = 2;
        public const int ROOM_NAME_LEN = 2;
        public const int ROOM_ID_LEN = 4;

        public const int PLAYERS_LEN = 1;
        public const int QUESTIONS_NUM_LEN = 2;
        public const int TIME_PER_QUESTION_LEN = 2;
    }
}
```

כל מספר הודעה שהלקוח צריך לשלוח
לשרת מצוין בקובץ הזה, על מנת נוחות
בפונקציות שבהן אני שולח את
ההודעות.

רשימת הקודים לתקשורת עם הלקוח נתונה בקוד מקור של השרת:

```
#pragma once

// SERVER ---> CLIENT
// SIGN IN
#define SRV_MSG_SIGN_IN_SUCCESS "1020"
#define SRV_MSG_SIGN_IN_WRONG_DET "1021"
#define SRV_MSG_SIGN_IN_ALREADY_CONNECTED "1022"

// SIGN UP
#define SRV_MSG_SIGN_UP_SUCCESS "1040"
#define SRV_MSG_SIGN_UP_PASS_ILLEGAL "1041"
#define SRV_MSG_SIGN_UP_USER_EXIST "1042"
#define SRV_MSG_SIGN_UP_USERNAME_ILLEGAL "1043"
#define SRV_MSG_SIGN_UP_FAILED "1044"

// ROOM
#define SRV_MSG_JOIN_ROOM_SUCCESS "1100"
#define SRV_MSG_JOIN_ROOM_FULL "1101"
#define SRV_MSG_JOIN_ROOM_NOT_EXIST "1102"
#define SRV_MSG_LEAVE_ROOM_SUCCESS "1120"
#define SRV_MSG_CREATE_ROOM_SUCCESS "1140"
#define SRV_MSG_CREATE_ROOM_FAILED "1141"
#define SRV_MSG_CLOSE_ROOM "116"
#define SRV_MSG_USERS_IN_ROOM "108"

// LOBY
#define SRV_MSG_ROOM_LIST "106"

// Game
#define SRV_MSG_GAME_QUESTION "118"
#define SRV_MSG_GAME_ANSWER_IND "120"
#define SRV_MSG_GAME_FINISH "121"

// REPORTS
#define SRV_MSG_REP_BEST_SCORES 124
#define SRV_MSG_REP_PERSONAL_STATUS 126

// CLIENT ---> SERVER

//CONNECTION
#define CLT_END_CONNECTION 299

// SIGN
#define CLT_MSG_SIGN_IN 200
#define CLT_MSG_SIGN_OUT 201
#define CLT_MSG_SIGN_UP 203

// LOBY
#define CLT_MSG_GET_ROOMS 205
#define CLT_MSG_GET_USERS_IN_ROOM 207

// ROOM
#define CLT_MSG_CREATE_ROOM 213
#define CLT_MSG_CLOSE_ROOM 215
#define CLT_MSG_LEAVE_ROOM 211
#define CLT_MSG_JOIN_ROOM 209
```

פקודת define, מגדירה שם שמור שמיצג מחרוזת מסוימת, ניתן לשמור מילה מסוימת בתור ערך בקוד בשני דרכים: בשימוש הפקודה Define, ובשימוש const.

העדפתי להשתמש בdefine מכיוון ששימוש בdefine יוצר ערך מסוים מבלי שהוא נשמר בזיכרון. למעשה, define לוקח את המילה השמורה, ומציג את המילה בתור הערך השמור עוד לפני שהקוד מתקמפל. Define לוקח את המילה ומשנה אותה מבלי לשמור אותה בזיכרון.

Const הוא בעצם משתנה שמור בזיכרון כמו משתנה רגיל, אך את המשתנה הזה לא ניתן לשנות. משתנה const מקבל אזור שמור בזיכרון להבדיל ממשתנה define, וצריך להעביר אותו ממקום למקום במהלך ריצת התכנית.

היכרות עם מבנה בסיס נתונים

בסיס נתונים הוא אמצעי המשמש לאחסון מסודר של נתונים במחשב, לשם אחזורם ועיבודם. בסיס נתונים מאוחסן באמצעי אחסון נתונים, בדרך כלל על גבי דיסק קשיח, המאפשר גישה ישירה לנתונים. הגישה לבסיס הנתונים נעשית באמצעות תוכנה ייעודית - מערכת לניהול בסיס נתונים (DBMS - Database Management System). בסיס הנתונים בנוי לפי מודל לאחסון הנתונים, כמו מגנונים פנימיים למיון ולחיפוש.

ישנם מספר מודלים לבסיסי נתונים:

רלציוני - משמש לאחסון נתונים במחשב עבור עיבוד ושחזור

רשת - מציג את בסיס הנתונים כרשת של צמתים וערוצים, כאשר כל הצמתים מייצגים רשומות נתונים, והערוצים מייצגים יחסים בין הרשומות.

היררכי - ניתן לגשת לנתונים מהשורש ודרך ענפי העץ, מלמעלה למטה. כל רשומה יכולה לשמש "אב" לרשומה או רשומות ברמה שמתחתיה (בנים). באותה צורה לכל בן ייתכנו בנים משלו. כך נוצר קשר N:1 בין הרשומות. לכל אב יכולים להיות מספר בנים אך לכל בן, אב אחד בלבד.

טבלאי - בסיס הנתונים בנוי מטבלאות, כאשר כל טבלה מכילה מידע על ישות מסוימת (לדוגמה, לקוחות במערכת בנקאית). ולכל רשומה בטבלה יש שדה ID שמזהה באופן ייחודי את הרשומה.

מונחה עצמים - הוא מסד מידע המאפשר שמירה, סידור ונגישות אל המידע על פי תכונות העצמים שהמידע מייצג.

NoSQL - קטגוריה חדשה יחסית של בסיסי נתונים, אשר נותנים פתרון אחסון וגישה למידע שאינו ממודל במבנה טבלאי יחסי אשר נפוץ בבסיסי נתונים יחסיים.

המודלים הללו מבטאים סוגי קשרים שונים בין הנתונים השונים.

המודל שאני משתמש בו לכתיבת הפרויקט שלי הוא DB רלציוני.

הDB עצמו אחראי לעדכן את הקבצים המתאימים – הוא יהיה אחראי על שמירת הנתונים הפיזית. כמשתמש של התוכנה, אני ארצה כמה שפחות להתעסק בשמירת נתונים ואצפה שהתוכנה שאיתה אני עובד, תדע להסתדר עם המידע שאני מכניס לה לבד.

במידת הצורך שבה הבעלים של התוכנה רוצה לתת גישה לאנשים אחרים להכניס מידע לDB, ניתן לתת יוזרים עם סיסמאות שתהיה להם גישה. לכל יוזר יש אפשרות לגשת למידע שונה או לבצע פעולות שונות.

אחת הסיבות שבגללן רציתי לעבוד עם DB, היא מכיוון שכך לא אצטרך להתעסק עם המון קבצים מיותרים, וכל המידע יהיה באיזור אחד שתהיה לי אליו גישה בצורה מאוד נוחה. בנוסף, מספר משתמשים יכולים להשתמש בנתונים בו זמנית כאשר הם נמצאים על שרת. בDB יש מנגנונים מובנים שמתעסקים בגישה לנתונים ומייעלים אותה מאשר שימוש בקבצים, ואם הייתי מממש את המשחק בקובץ, הפרויקט היה לוקח הרבה יותר זמן, שאותו ניצלתי להוספת פיצ'רים. עם זאת, לרוב DB שמוצאים היום בשוק, יש אפשרות להתאוששות במידה והDB קרס, מה שיוצר סביבת עבודה נוחה יותר.

כאשר ישנה התעסקות בDB, ניתן לשמור סוגים שונים של נתונים כגון מספרים, מחרוזות, תמונות, קבצים וכו'. ניתן גם לוודא אמינות של הנתונים בעזרת הDB. לדוגמא, לא יכולים להיות שני אנשים עם אותו מספר תעודת זהות.

הDB מחולק לטבלאות שמסודרות לפי הנתונים המבוקשים:

• t_users

תיאור: טבלה המכילה את היוזרים שנרשמו למערכת שדות:

- **username** – מפתח ראשי, טקסט, not null
- password – טקסט, not null
- email – טקסט, not null

• t_games

תיאור: טבלה המכילה את נתוני המשחקים שדות:

- **game_id** – מפתח ראשי, מספר שלם, not null, autoincrement
- status – מספר שלם (0 – המשחק התחיל, 1 – המשחק הסתיים), not null
- start_time – datetime, זמן התחלת המשחק, not null
- end_time – datetime, זמן סיום המשחק. יכול להיות NULL. מתעדכן כשמסתיים המשחק.

• t_questions

תיאור: טבלה המכילה את השאלות והתשובות לכל שאלה שדות:

- **question_id** – מפתח ראשי, מספר שלם, not null, autoincrement
- **question** – טקסט, not null
- **correct_ans** – טקסט, not null, תשובה נכונה
- **ans2** – טקסט, not null, תשובה שגויה 2
- **ans3** – טקסט, not null, תשובה שגויה 3
- **ans4** – טקסט, not null, תשובה שגויה 4

• t_players_answers

תיאור: טבלה השומרת מידע עבור תשובה של שחקן. שדות:

- **game_id** – PK, מספר שלם, not null, FK לשדה מתאים בטבלת משחקים.
- **username** – PK, טקסט, not null, FK לשדה מתאים בטבלת יוזרים.
- **question_id** – PK, מספר שלם, not null, FK לשדה מתאים בטבלת שאלות.
- **player_answer** – טקסט, not null, התשובה של השחקן.
- **is_correct** – מספר שלם, not null, 0 – תשובה שגויה, 1 – תשובה נכונה.
- **answer_time** – מספר שלם, not null, הזמן בשניות שלקח לשחקן לענות על השאלה.

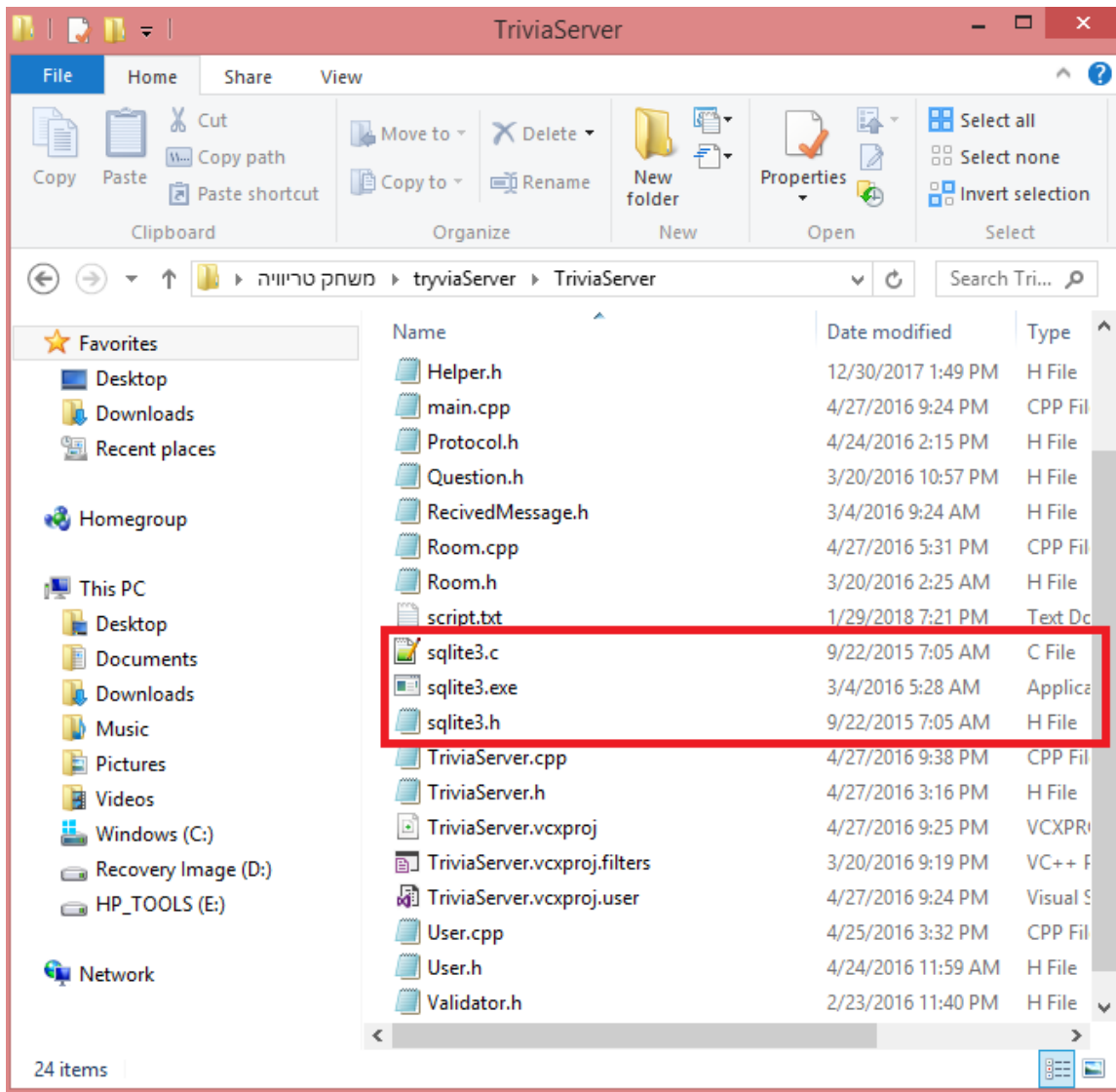
מפתח ראשי – game_id, username, question_id

הקוד שמריץ ובונה את הDB נמצא בתוך קובץ שנקרא script.txt ומורץ בצד השרת.

בקובץ זה ישנם פקודות שבונות את הDB לפני שהמשחק מתחיל לרוץ, ובכך מאפשרות לשרת להתחבר לDB ולשנות/להוסיף דברים.

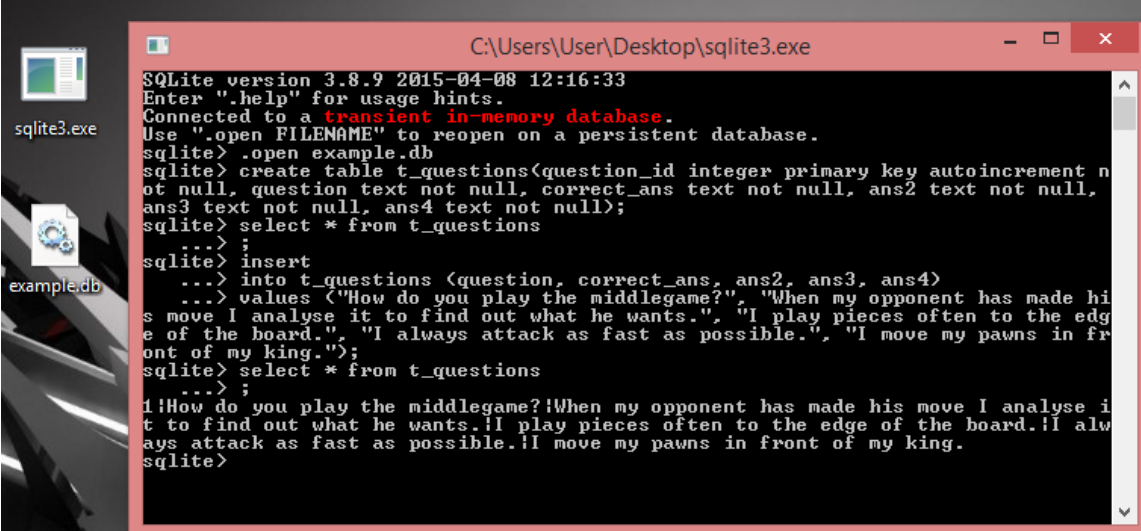
היכרות עם SQLite3

כדי לעבוד עם sqlite3 כל מה שצריך בפרויקט הוא את קובץ ההרצה ואת קובץ ההוראות של התוכנה (שכתוב בשפת C).



לאחר שקובץ ההרצה נמצא בתכנית, ניתן ללחוץ עליו ויפתח לנו CMD שמריץ את המערכת של sqlite3.

להלן כמה דוגמאות לקוד של sqlite3:



```
SQLite version 3.8.9 2015-04-08 12:16:33
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open example.db
sqlite> create table t_questions(question_id integer primary key autoincrement not null, question text not null, correct_ans text not null, ans2 text not null, ans3 text not null, ans4 text not null);
sqlite> select * from t_questions
...>
sqlite> insert
...> into t_questions (question, correct_ans, ans2, ans3, ans4)
...> values ("How do you play the middlegame?", "When my opponent has made his move I analyse it to find out what he wants.", "I play pieces often to the edge of the board.", "I always attack as fast as possible.", "I move my pawns in front of my king.");
sqlite> select * from t_questions
...>
1|How do you play the middlegame?!When my opponent has made his move I analyse it to find out what he wants.!I play pieces often to the edge of the board.!I always attack as fast as possible.!I move my pawns in front of my king.
sqlite>
```

open example.db – יצירת DB ריק חדש

create table t_questions(question_id integer primary key autoincrement not null, question text not null, correct_ans text not null, ans2 text not null, ans3 text not null, ans4 text not null;

Select * from t_questions – פעולה זו מדפיסה את כל הערכים הנמצאים בטבלה. מכיוון שרק יצרנו את הטבלה, הטבלה ריקה ולכן פעולה זו לא תדפיס לנו כלום

insert מכניסה איברים לתוך הטבלה לפי הערכים המוכנסים לטבלה מראש. ניתן לראות שהמחרוזת לתשובת השאלה תמיד תופיע לאחר שליחת השאלה עצמה, אך כאשר אני מושך את המידע מהDB, אני מבצע את פעולה rand שתציג לשחקן את התשובות האפשריות באופן רנדומלי.

כדי להכניס/לעדכן/למחוק נתונים בטבלה, קיימות 3 פקודות אשר נקראות פקודות DML (Data Manipulation Language):

insert-

Update-

Delete-

דוגמאות להכנסת רשומה לטבלה:

```
insert into students(id, name, age) values(339, "erez", 32); insert into students(id, age)
values;(23.4 ,444)
```

דוגמאות לעדכון רשומות בטבלה:

```
update students set id=4; update students set name="GAL", id=430;
```

דוגמאות למחיקת רשומות בטבלה:

```
delete from students;
```

פקודת alter מאפשרת שינויים במבנה הטבלה דוגמאות:

הוספת עמודה לטבלה

```
alter table students add column grade double ;
```

שינוי שם טבלה

```
alter table students rename to myStudents ;
```

כדי למחוק טבלה נשתמש בפקודה

```
drop table: drop table myStudents;
```

SQL (Structured Query Language) היא שפת שאילתות המאפשרת לאחזר נתונים מטבלאות

ניתן לשלוף בצורות שונות ומורכבות

שליפת כל המידע מטבלה :

```
Select * from students;
```

אפשרויות נוספות:

```
select id, name from students;
```

```
select id as "STUDENT ID", name as "STUDENT NAME" from students;
```

**** כל הטבלאות של המשחק מופיעות בנספח א' המצורף.**

פרויקט מעשי: כתיבת משחק "טריוויה שחמט"

מבוא – מבנה הפרויקט

הגדרות:

יוזר – משתמש שקיים במאגרי המערכת

חדר – חדר שמשתמשים יכולים להיכנס אליו ולאחר מכן נהפך למשחק טריוויה ממשי. כל יוזר יכול להיות משויך אך ורק לחדר אחד בו זמנית. חדר אינו נשמר ב-DB.

אדמין – יוצר החדר. הוא מגדיר את ההגדרות הדרושות לחדר (שם חדר, מספר שחקנים מקסימלי, זמן מירבי לתשובה, כמות שאלות במשחק). הוא זה שמחליט האם להפוך את החדר למשחק או לסגור את החדר ולא להתחיל משחק. עם תחילת משחק האדמין הופך לשחקן רגיל ואין משמעות לזה שהוא יצר את החדר.

משחק – משחק טריוויה ממשי שבו השחקנים מקבלים שאלות ועונים תשובות. המשחק יכול להכיל גם שחקן אחד. משחק נשמר ב-DB.

שחקן – משתמש שנמצא במהלך משחק. יכול לעזוב את המשחק בכל רגע. יוזר שהוא שחקן לא יכול להיות משויך לחדר אחר או למשחק אחר.

המשחק מבוסס על כך שניתן ליצור משתמשים חדשים ולהגדיר להם 1000 פרטים נוספים. כל יוזר יכול ליצור חדר ולהגדיר לו כפרמטרים את שם החדר, מספר השחקנים, מספר השאלות, זמן מקסימלי לתשובה. זהו **האדמין**.

יוזרים אחרים יכולים להצטרף לחדר במידה ויש מקום, ובמידה והם אינם נמצאים בחדר אחר. האדמין מחליט מתי להתחיל את המשחק (לא חייב שהחדר יהיה מלא) או האם לסגור את החדר ולא להתחיל את המשחק.

כאשר משחק מתחיל כל השחקנים מקבלים שאלה עם 4 תשובות אפשריות. כל שחקן בוחר תשובה והשרת מעדכן אותו האם התשובה נכונה או לא. רק לאחר שכל השחקנים ענו על השאלה בסיבוב, מתקדמים לשאלה הבאה. במידה והמשחק נגמר – כלומר, מכסת השאלות תמה וכל השחקנים ענו על השאלה האחרונה, כל שחקן מקבל סיכום של התשובות הנכונות של כל השחקנים שהשתתפו במשחק (גם אם עזבו באמצע).

הניקוד מתבצע כך שעל שאלה נכונה מקבלים נקודה אחת. המנצח הוא זה שענה על מספר התשובות המירביות במהלך המשחק. שחקנים אינם מקבלים הודעות עזיבה של שחקנים אחרים – כך שיתכן מצב בו שחקן משחק סולו בלי שהוא מודע (המצב תקין). משחק סולו אפשרי הוא אם האדמין שפתח חדר מתחיל לשחק כשרק הוא בחדר.

לכל יוזר קיימת אפשרות לראות את 3 התוצאות הטובות ביותר הקיימות במערכת (high-scores). בנוסף, כל יוזר יכול לראות את פרטים על ההישגים שלו – כמות משחקים, מספר תשובות נכונות, מספר תשובות שגויות, זמן ממוצע לתשובה.

המשחק מבוסס על כך שניתן להירשם כיוזר חדש למערכת ולהגדיר סממא ופרטים נוספים.

בעבודה זו הושקעו מאמצים רבים ללמידת מושגים מורכבים במדעי המחשב. נעזרתי בהמון חומרים על מנת לחקור לעומק נושאים מורכבים כגון תקשורת שרת לקוח, כתיבת פרויקט בסביבת העבודה Visual Studio, שנחשבת לאחת מסביבות העבודה הנפוצות ביותר בקרב מתכנתים, לימוד עצמי של שפת C++, על מנת ליצור שרת מאובטח, לימוד עצמי של שפת C# כדי להכין GUI (Graphical user interface) מושקע ומתקדם וכדי ליצור לקוח בסיסי שיתקשר עם השרת.

שלבי בניית הפרויקט

ראשית, כתבתי את התקשורת בין השרת ובין הלקוח בשתי השפות, בשתי פרויקטים נפרדים. לאחר שבוצע חיבור בין השרת לבין הלקוח, התקדמתי עם כתיבת הלקוח והתחלתי לכתוב את פרוטוקול התקשורת בין הלקוח לבין השרת, ומשם המשכתי לכתיבת הבקשות לשרת, כאשר כל בקשה מלוות בקוד הGUI שלה. לדוגמא, לפני שרשמתי את הבקשה לכניסה המשתמש, יצרתי את קוד הGUI של הכניסה ושל התפריט, ולאחר מכן כתבתי את הבקשה לכניסה (sign in) של המשתמש.

אני מאמין ששיטת עבודה זו, היא שיטה נורא נקייה ששומרת על קוד יפה, ועוזרת לי להתקדם בצורה הכי נוחה והכי מהירה מבלי להיתקל במצבים מסוימים שבהם אני צריך לחשוב על חלקים קודמים שעשיתי כבר, ולשנות אותם כדי שהקוד החדש יעבוד.

משהו נוסף שנורא עזר לי לשמור על סדר בעבודה, הוא הכנת טבלת UML (Unified Modeling Language) מודרת עם כל האובייקטים שצריך לכתוב וכל החלקים שצריכים להיות בצד השרת, וטבלת FLOW שתתאר לי את התהליך של הפרויקט (ניתן לראות בנספח ב' – ג' את הטבלאות).

מכיוון שהעבודה נכתבה בשני פרויקטים שונים, חשוב מאוד שכל התהליך של התאמת הקודים ותיאום בין השרת ללקוח יהיה כמה שיותר קל כדי להימנע משגיאות מיותרות, ולכן השתמשתי בטבלאות UML וFLOW.

לאחר חיבור השרת והלקוח ויצירת פרוטוקול אפליקטיבי ייחודי המתבסס על TCP, הפכתי את השרת לשרת Multi-Threaded אשר מנהל תור הודעות. כל קליינט מקבל ת'רד שמאזין להודעות ממנו. כאשר מגיעה הודעה היא נכנסת לתור ההודעות.

קיים ת'רד נוסף בשרת אשר עובר על תור ההודעות ומטפל בהן.

אחת הסיבות להחלטה לעבוד בשיטה זו היא כדי להימנע מבעיות סנכרון ותכנון וקידוד פשוטים יותר בצד של השרת.

מחלקות – תיאור כללי

TriviaServer

תיאור כללי: המחלקה העיקרית. מאזינה ומקבלת לקוחות. מקבלת הודעות מקליינטים ומטפלת בהם. מחזיקה את רשימת היוזרים המחוברים, רשימת החדרים הפעילים.

Room

תיאור כללי: מייצגת חדר. מכילה את ההגדרות של החדר ופרטים כגון היוזרים הרשומים לחדר, האדמין של החדר. מאפשרת ביצוע פעולות על חדר כגון הצטרפות לחדר, סגירת חדר, שליחת הודעות ליוזרים הרשומים לחדר ועוד.

User

תיאור כללי: מייצגת יוזר. מכילה את הפרטים עבור יוזר. מאפשרת ביצוע פעולות על יוזר כגון הצטרפות לחדר, יצירת חדר, עזיבת חדר, עזיבת משחק וכו'.

Protocol

תיאור כללי: אוסף של קבועים המגדירים את ההודעות בפרוטוקול האפליקטיבי.

Game

תיאור כללי: מחלקה המייצגת משחק פעיל. מכילה את רשימת השחקנים, מאפיינים של המשחק כמו מספר שאלות, מספר השחקנים שכבר ענו על השאלה הנוכחית וכו'. מאפשרת ביצוע פעולות כגון שליחת שאלה לכל השחקנים, הודעות על עזיבת משחק, טיפול בסיום משחק, עדכונים מול ה-DB ועוד.

RecievedMessage

תיאור כללי: מחלקה המייצגת הודעה שנכנסת לתור ההודעות שבהן השרת צריך לטפל. מכילה את היוזר שממנו הגיעה ההודעה, הסוקט, קוד ההודעה, פרטים נוספים בהתאם לסוג ההודעה.

DataBase

תיאור כללי: מחלקה המאפשרת גישה ל-DB. קיימות בה פעולות כגון האם יוזר קיים ב-DB, האם הסמא תואמת ליוזר, הכנסת משחק חדש, שליפת ההיי-סקור ועוד.

Question

תיאור כללי: מייצגת שאלה במשחק. מכילה את השאלה ואת התשובות. דואגת לסדר באופן אקראי את התשובות.

Validator

תיאור כללי: מחלקה סטטית. מכילה פעולות הקשורות בתקינות נתונים כגון האם סמא חוקית, האם יוזר חוקי.

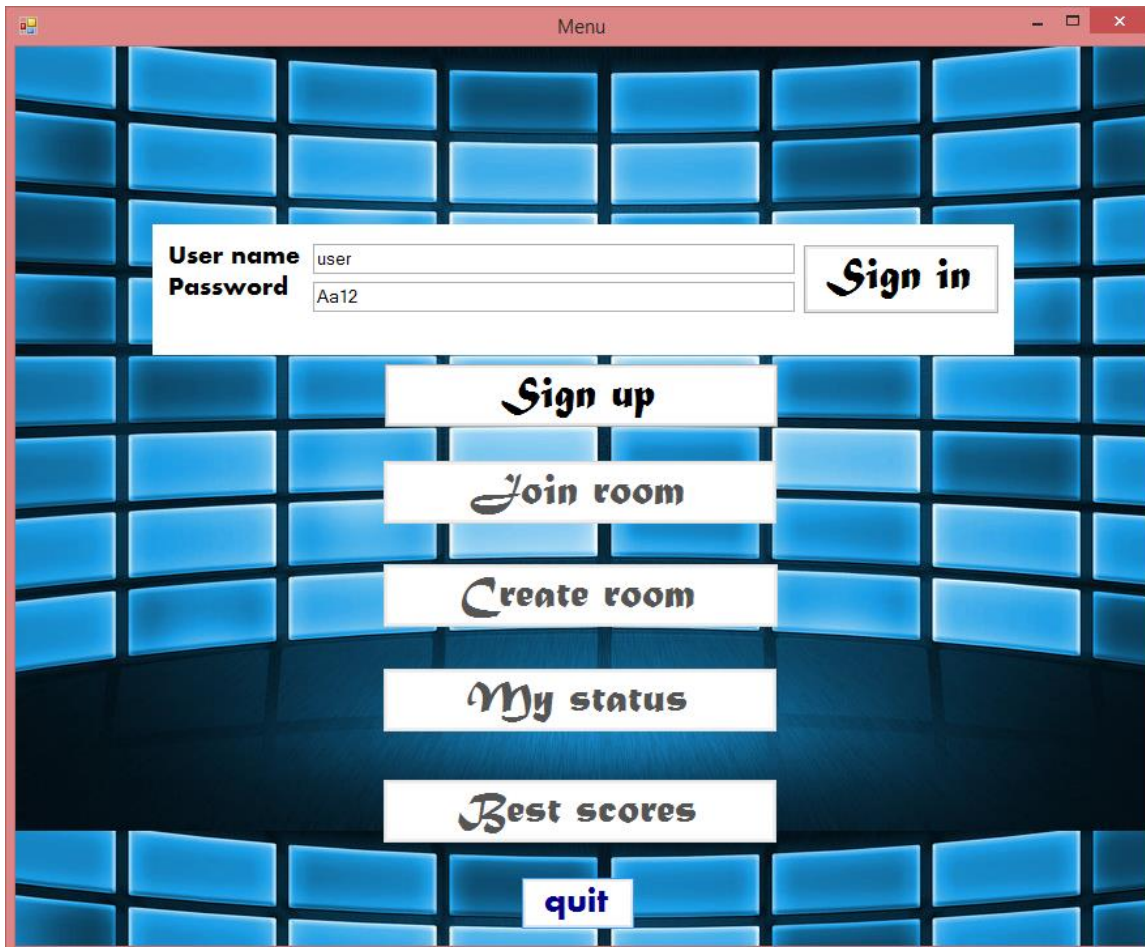
Helper

תיאור כללי: מחלקה סטטית. מחלקה המכילה פונקציות סטטיות אשר מאפשרות הדפסת הודעות DEBUG, קבלה ושליחת מידע מ-SOCKET ועוד.

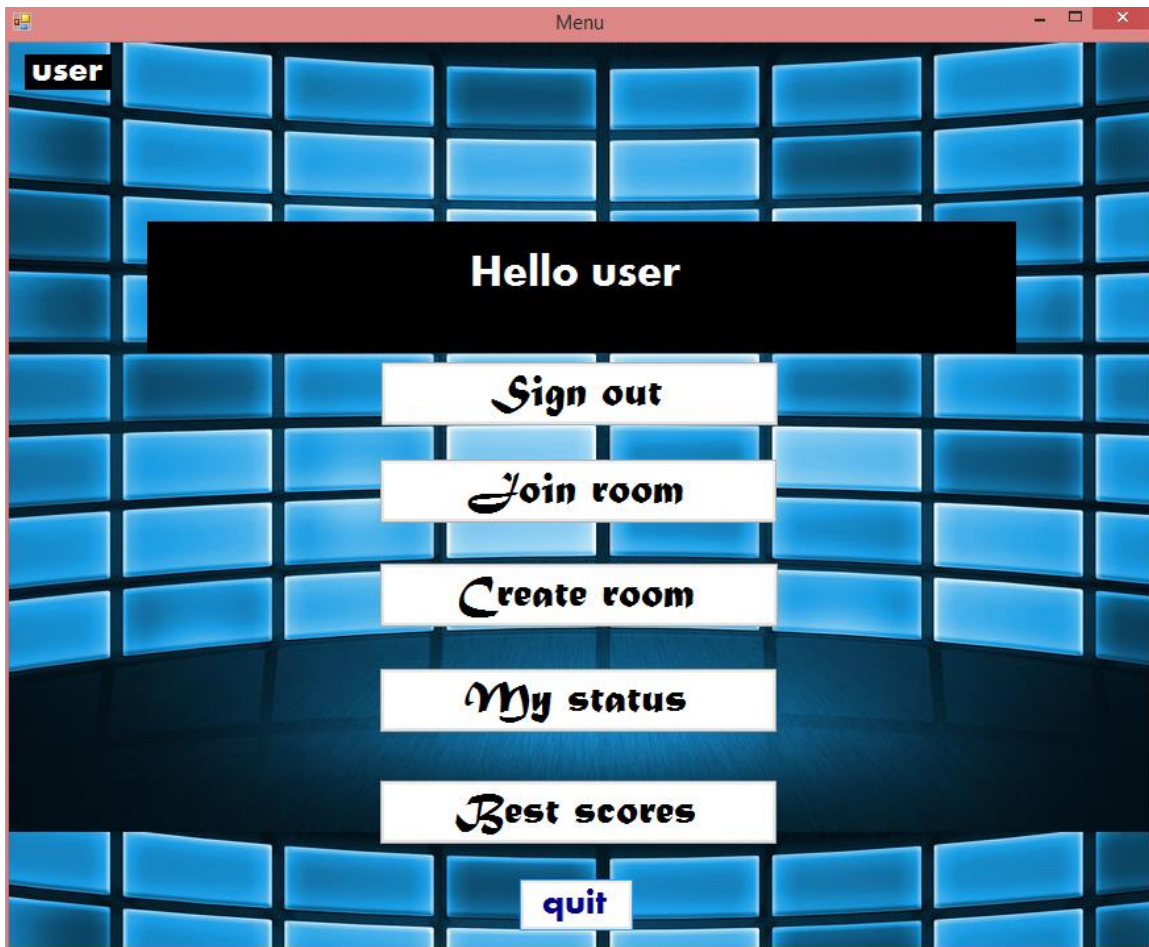
הצגת שימוש במוצר

קודם כל אנו זקוקים להריץ את השרת ברקע.

לאחר שהרצנו את השרת אפשר להריץ את הלקוח ויופיע לנו התמונה הבאה:



כדי להתחבר, אני צריך ללחוץ על כפתור ה sign in (שישלח הודעה לשרת ויבקש בקשה התחברות לפי הפרוטוקול)



לאחר שהתחברנו, אפשר לראות שניתן ללחוץ על כל הכפתורים.

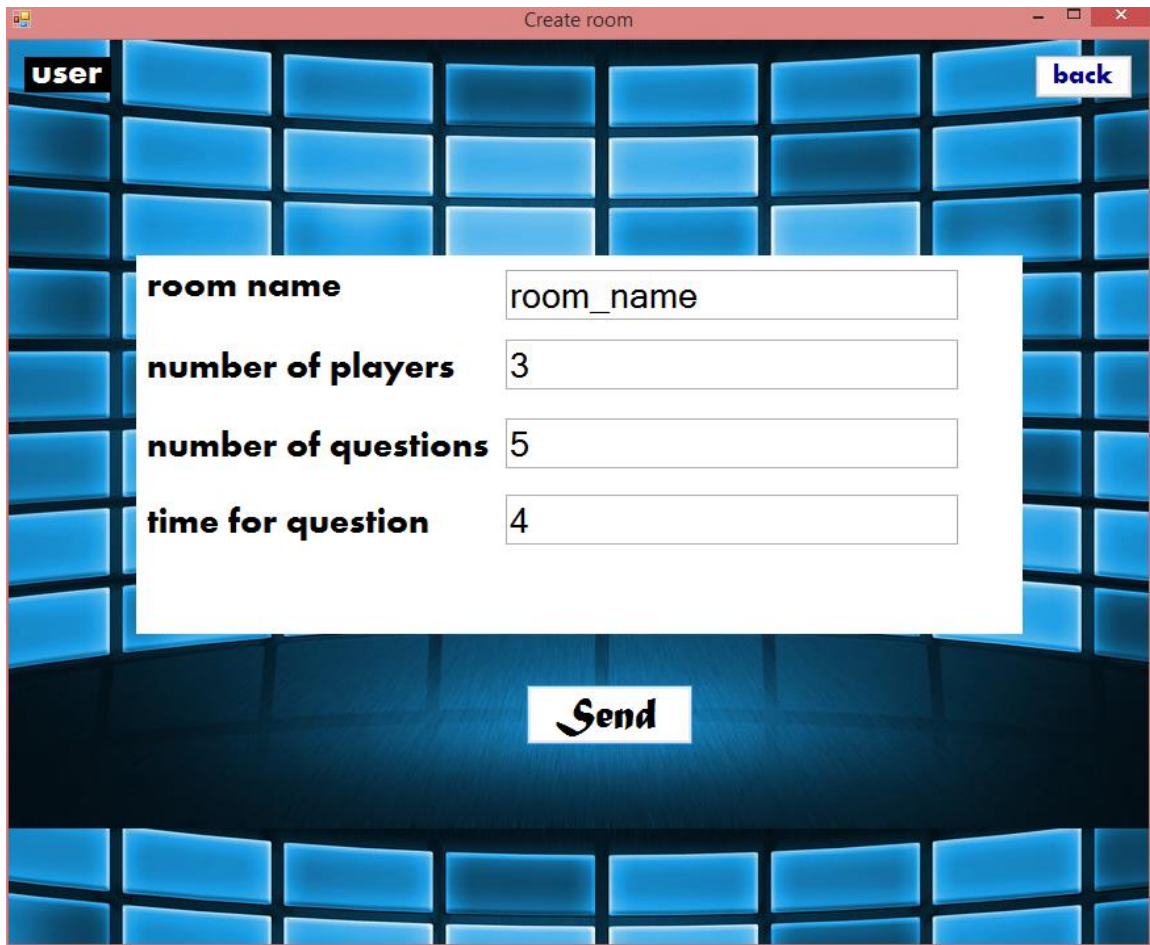
לאחר ההתחברות, קוד השרת יראה כך:

```

C:\Users\User\Desktop\טריויה\משחק\tryviaServer\Debug\TriviaServer.exe
Starting...
binded
listening (Port = 8820)...
accepting client...
Client accepted. Client socket = 224
accepting client...
-----
handleRecievedMessages: msgCode = 200, client_socket: 224
Adding new user to connected users list: socket = 224, username = user
Message sent to user: user, msg: 1020
SEND: User signed in successfully: username = user, socket = 224
-----

```

הצעד הבא שלנו לעבר הטריזיה יהיה ליצור חדר!



user

back

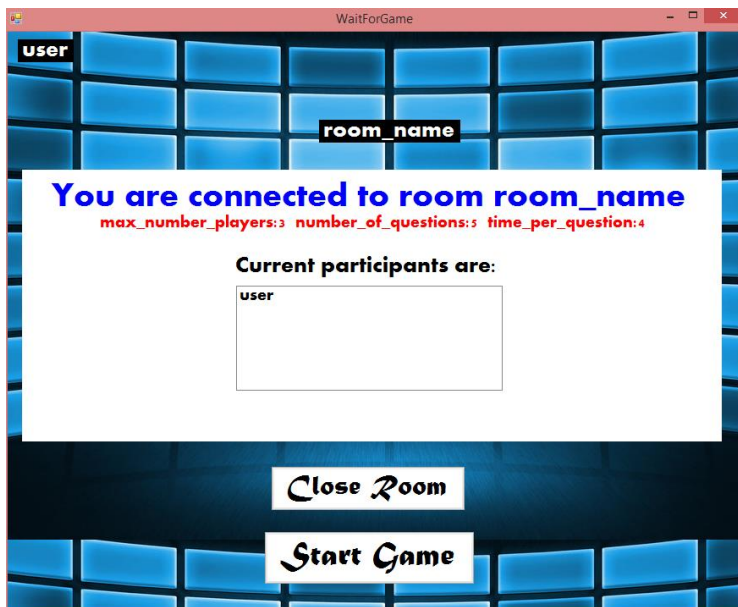
room name

number of players

number of questions

time for question

Send



user

room_name

You are connected to room room_name
max_number_players:3 number_of_questions:5 time_per_question:4

Current participants are:

user

Close Room

Start Game

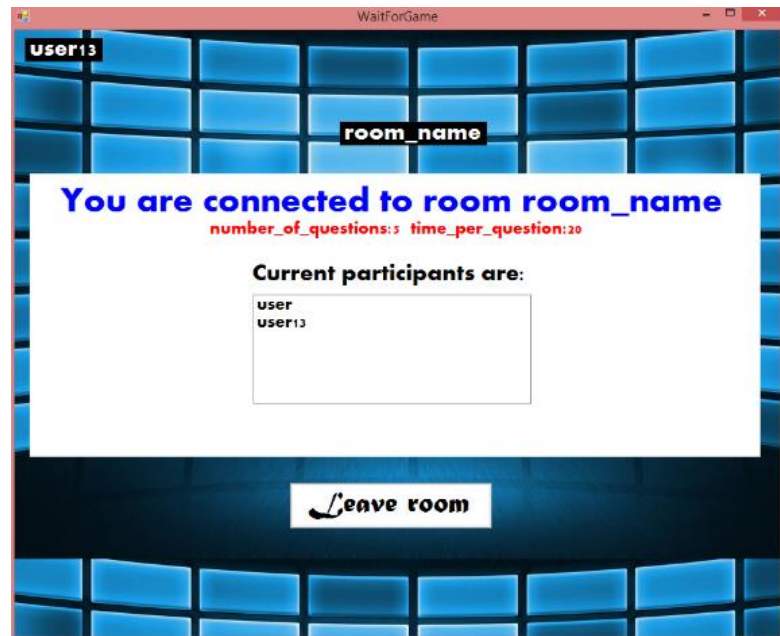
מכאן ניתן לראות שללקוח ישנה
השפעה על שם החדר, מספר
השחקנים שיכולים להיכנס לחדר,
מספר השאלות והזמן לכל שאלה.
לאחר יצירת החדר, הלקוח (user)
יהיה האדמין של החדר

כעת, אחבר לקוח נוסף למשחק, (בשם user13), ואכנס לרשימת החדרים שמופיעה במשחק:

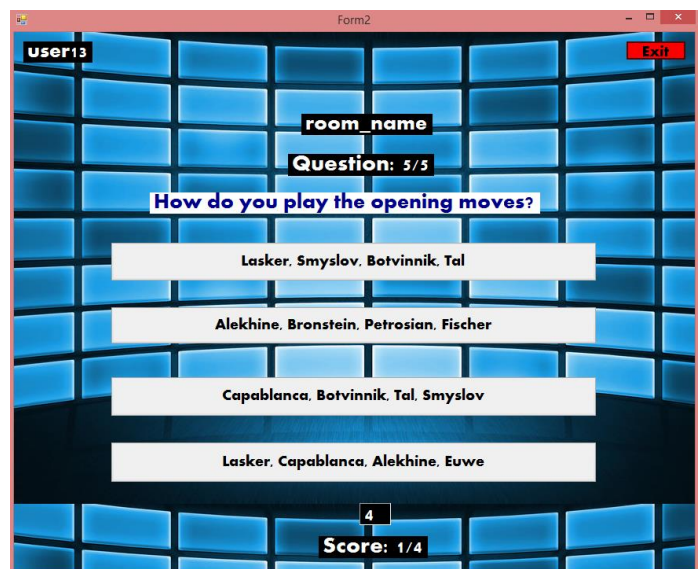
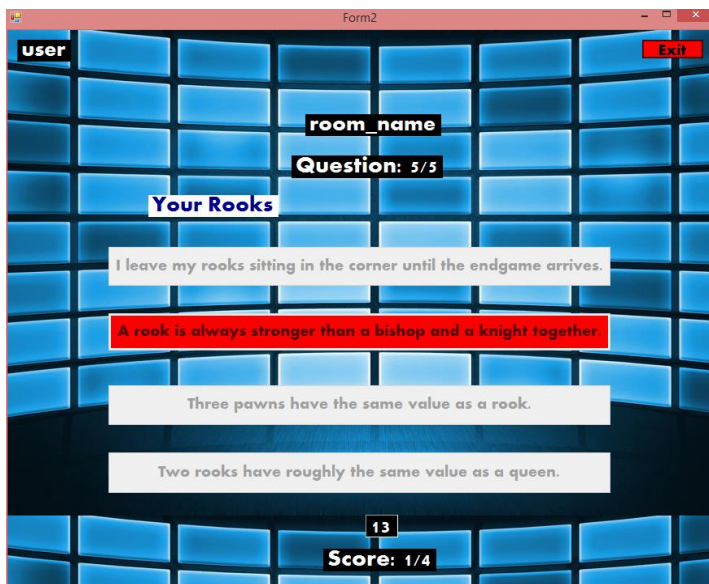
The screenshot shows a game interface window titled "Form4". The background is a blue brick wall. In the top left corner, there is a black box with the text "user13". In the top right corner, there is a white button with blue text that says "back". In the center of the window, there is a large red rectangle. Inside this rectangle, the text "Rooms list:" is displayed above a white input field with the placeholder text "room_name". Below this, the text "Selected room players:" is displayed above another white input field with the placeholder text "user". At the bottom of the red rectangle, there is a white button with blue text that says "refresh". Below the red rectangle, there is a white button with black text that says "Join" in a stylized font.

ניתן להבחין שהשחקן user13 יכול לראות ברשימת החדרים הפנויים למשחק את החדר שעשינו ואם הוא ילחץ על החדר (כמו שעשיתי בתמונה), שמות האנשים הנמצאים בחדר יופיעו.

כעת, לאחר שכל השחקנים מחוברים, השחקן user יכול להתחיל את המשחק ונוכל לשחק "טריווית שחמט"!



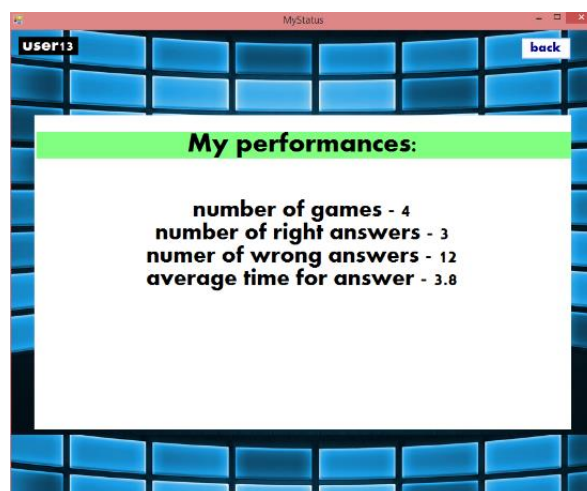
לאחר שהשחקן user לחץ על תחילת המשחק, שני השחקנים נכנסים למשחק על זמן



לבסוף, לכל אחד מהשחנים נשלחת הודעה לגבי הניקוד שהוא צבר במהלך המשחק, לאחר ששחקן מאשר את ההודעה, הוא חוזר לתפריט.



לכל שחקן יש אפשרות לראות את התוצאות האישיות שלו



בנוסף לכך, כל שחקן יכול לראות את התוצאות של שלושת השחקנים הכי טובים



סיכום ומסקנות

עבודה זו הייתה נורא מהנה ונורא חווייתית מבחינתי. הצלחתי ללמוד דברים לבד, ליישם אותם וליצור משחק מהנה עם המון טכנולוגיות שמשולבות בתוכו. הצלחתי לשפר את יכולתי ולשפר את רמת התכנות שלי בצורה גבוהה. בנוסף לכך, בחלק מהנושאים, הצלחתי להגיע למצב שבו אני יכול ללמוד חומר לבד, ללא עזרת גורמים חיצוניים.

העבודה מראה מה למדתי במהלך השנה, ולאחר שסיימתי אותה אני יכול להיות בטוח בעצמי שעשיתי עבודה טובה ושהקודים כתובים ברמה גבוהה.

לאחר שהראתי את הפרויקט לכמה מחבריי, ולמשפחתי, הרגשתי גאווה עצומה שהצלחתי להגיע להישגים כאלו גדולים.

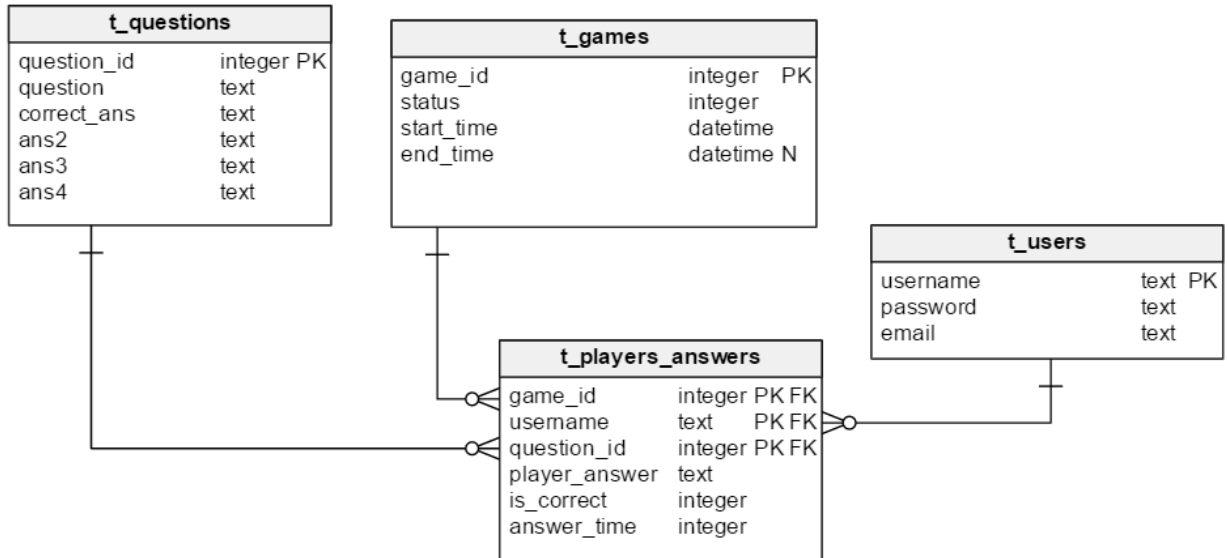
אני רוצה להודות למנטור שלי, יהודה אור, שעזר ותמך וליווה אותי בכל הפרויקט מתחילתו ועד סופו, ונתן לי מנסיונו האישי.

ביבליוגרפיה

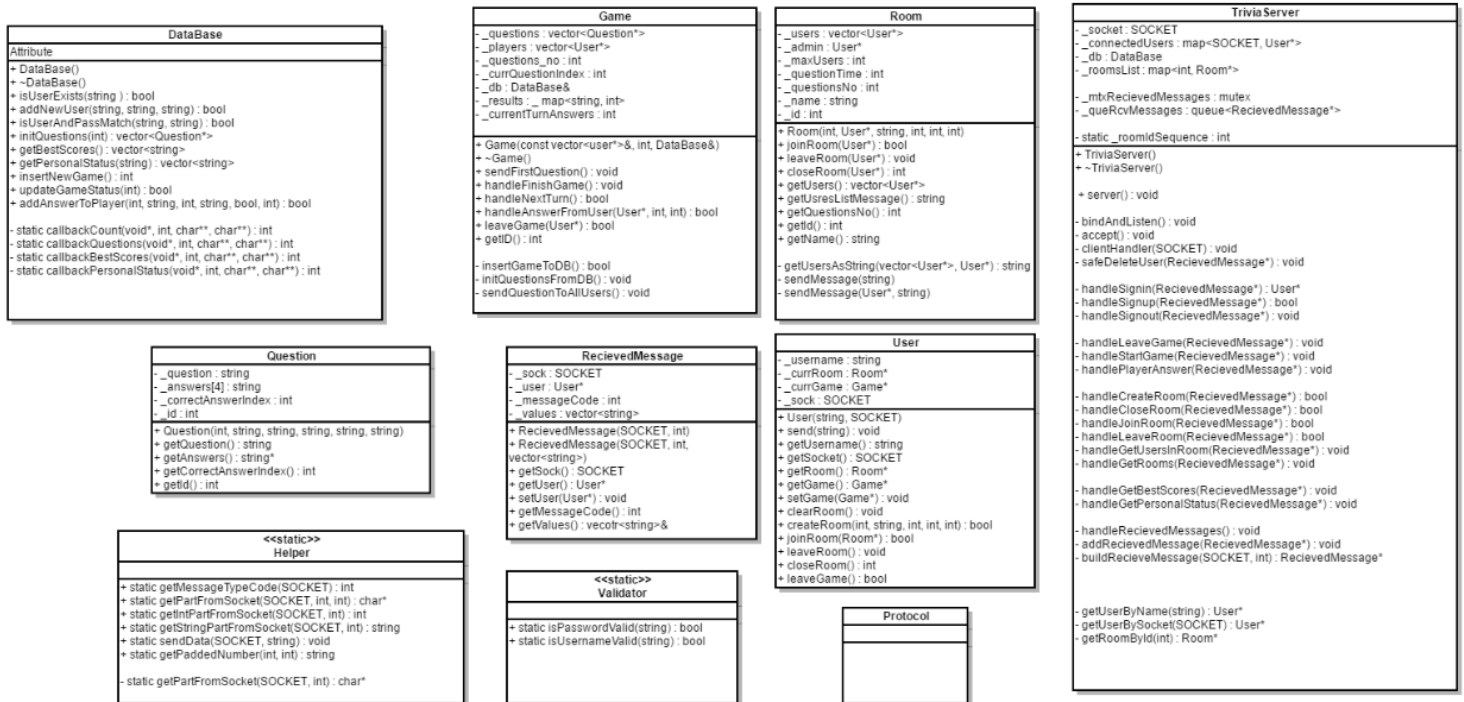
- [illegible]

נספחים

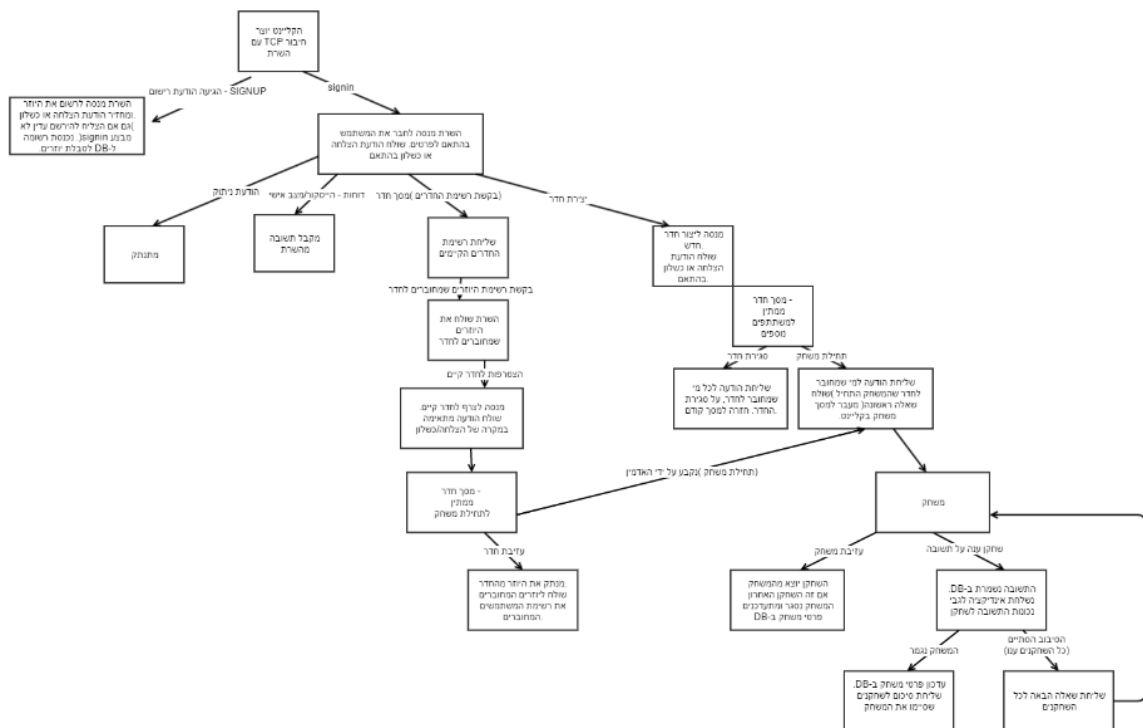
נספח א' – טבלאות הDB של המשחק



נספח ב' – טבלת UML לשרת



נספח ג' – טבלת FLOW



נספח ד' – קוד הלקוח

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using System.Net;
using System.Net.Sockets;

using System.Windows.Forms;

using System.Diagnostics;

namespace trivya
{
    public class Client : Codes
    {
        static IPAddress ipAddress;
        static IPEndPoint remoteEP;
        static Socket client;

        public int numberOfQuestions=0;
        public int timePerQuestion=0;
        public int numberOfPlayers=0;

        int sumQuestions = 0;
        int sumTime = 0;
        int sumRight = 0;
        int sumWrong = 0;

        public string myName = "";
        public string myRoom = "";

        //Setting the connection with the server
        private void setConnection()
        {
            //object that reads the file that keeps all of the configuration
            System.IO.StreamReader sr = new System.IO.StreamReader("config.txt");
            //after opening the file, we need to split the data and get the
            //ip address and the port
            string ip = sr.ReadLine().Split('=')[1];
            int port = Int32.Parse(sr.ReadLine().Split('=')[1]);
            bool encrypt = Convert.ToBoolean(sr.ReadLine().Split('=')[1]);
            ipAddress = System.Net.IPAddress.Parse(ip);
            remoteEP = new IPEndPoint(ipAddress, port);
            client = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
            DialogResult result = DialogResult.OK;
        }
    }
}
```

```

        while (true)
        {
            try
            {
                //setting the connection
                client.Connect(remoteEP);
                break;
            }
            catch
            {
                //if we found an error, we pr
                string message = "You do not have a server running\n\n Ok - try again. \n Cancel - close
client. ";

                string caption = "Error Detected";
                MessageBoxButtons buttons = MessageBoxButtons.OKCancel;
                result = MessageBox.Show(message, caption, buttons);
                if (result == DialogResult.Cancel)
                {
                    System.Environment.Exit(1);
                    break;
                }
                else
                {
                    System.Threading.Thread.Sleep(1000);
                }
            }
        }
    }

    //The constractor needs to set the connection
    public Client()
    {
        setConnection();
    }

    //Adding the padding of the messages to the message that we will send to the server.
    //gets: input - source message that we want to add
    //numOfChars - num of charechters to copy from the input string.
    //dstOffset - num of charechters to copy from the input string.
    //res - the destination that we will input the string to.
    //returns: void.
    public void addToMsg(string input, int numOfChars, int dstOffset, ref byte[] res)
    {
        string len = input.Length.ToString();
        if (input.Length < 10)
            len = "0" + len;
        System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(len), 0, res, dstOffset, numOfChars);
        System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(input), 0, res, dstOffset + numOfChars,
input.Length);
        return;
    }
}

```

```

//Creating client sign in request for the server.
//Gets: two strings, user name and password of the registration.
//returns: answer from the server, if the user managed to get in.
public string signIn(string userName, string pass)
{
    byte[] msg = new byte[CODE_LEN + USERLEN + userName.Length + PASSLEN + pass.Length];
    System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(SIGN_IN), 0, msg, 0, CODE_LEN);

    //send to the addToMsg pointer to the msg for adding the padding
    //of the message.
    addToMsg(userName, USERLEN, CODE_LEN, ref msg);
    addToMsg(pass, PASSLEN, CODE_LEN + USERLEN + userName.Length, ref msg);

    //The client sends the name and the password of the user and wait for the server's response.
    //Meanwhile, the server checks if the user exist in the DB.
    client.Send(msg);

    //Getting answer back from the server.
    byte[] ans = new byte[4];
    int bytesRec = client.Receive(ans);

    //converting the code from the server to string.
    string code = System.Text.Encoding.Default.GetString(ans, 0, 3);
    //Check if the code is not equal to what we should get from the server.
    if (!code.Equals(SIGN_IN_ANS, StringComparison.Ordinal))
        return "ERROR";

    //now we convert the response to chars and we read the messages according the language
    //between the server and the client.
    char res = Convert.ToChar(ans[3]);
    if (res == '0')
    {
        myName = userName;
        return "Success";
    }
    else if (res == '1')
        return "Wrong details";
    else if (res == '2')
        return "User is already connected";
    else
        return "ERROR";
}

```

```

//Creating client sign up request for the server.
//Gets: three strings, user name and password and email of the registration.
//returns: answer from the server, if the user managed to get in.
public string signUp(string userName, string pass, string email)
{
    //Creating message in the size of the sign up message.
    byte[] msg = new byte[CODE_LEN + USERLEN + userName.Length + PASSLEN + pass.Length + EMAILLEN +
email.Length];
    System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(SIGN_UP), 0, msg, 0, CODE_LEN);

    //as we made before, adding the user name, pass and email to the message for checking with the
server.
    addToMsg(userName, USERLEN, CODE_LEN, ref msg);
    addToMsg(pass, PASSLEN, CODE_LEN + USERLEN + userName.Length, ref msg);
    addToMsg(email, EMAILLEN, CODE_LEN + USERLEN + userName.Length + PASSLEN + pass.Length, ref msg);
    client.Send(msg);

    //making all of the process with taking out the characters
    byte[] ans = new byte[4];
    int bytesRec = client.Receive(ans);

    string code = System.Text.Encoding.Default.GetString(ans, 0, 3);
    if (!code.Equals(SIGN_UP_ANS, StringComparison.Ordinal))
        return "ERROR";

    //checking answers from the server to make sure everything is ok.
    char res = Convert.ToChar(ans[3]);
    if (res == '0')
        return "Success";
    else if (res == '1')
        return "Pass Illegal";
    else if (res == '2')
        return "Username is already exists";
    else if (res == '3')
        return "Username is illegal";
    else if (res == '4')
        return "Other";
    else
        return "ERROR";
}

//Close the program.
public void signOut()
{
    byte[] msg = new byte[CODE_LEN];
    System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(SIGN_OUT), 0, msg, 0, CODE_LEN);
    client.Send(msg);
}

```

```

//sending the server request for getting all of the rooms
//send string of the msg code according to the protocol.
public string[] getRooms()
{
    string[] res;
    byte[] msg = new byte[CODE_LEN];
    System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(GET_ROOMS), 0, msg, 0, CODE_LEN);
    client.Send(msg);

    byte[] ans = new byte[200000];
    int bytesRec = client.Receive(ans);
    string code = System.Text.Encoding.Default.GetString(ans, 0, 3);
    if (!code.Equals(GET_ROOMS_ANS, StringComparison.Ordinal))
    {
        res = new string[1];
        res[0] = code;
        return res;
    }
    int numOfRooms = Int32.Parse(System.Text.Encoding.Default.GetString(ans, 3, 4));
    res = new string[numOfRooms * 2];
    int nameLen, offset = 0;
    for (int i = 0; i < numOfRooms * 2; i += 2)
    {
        res[i] = System.Text.Encoding.Default.GetString(ans, 7 + offset, 4);
        nameLen = Int32.Parse(System.Text.Encoding.Default.GetString(ans, 7 + offset + 4, 2));
        res[i + 1] = System.Text.Encoding.Default.GetString(ans, 7 + offset + 4 + 2, nameLen);
        offset += 4 + 2 + nameLen;
    }
    return res;
}

//Sends request for getting the users in the room (207)
public void askForUsers(string roomId)
{
    byte[] msg = new byte[CODE_LEN + ROOM_ID_LEN];
    System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(GET_USERS), 0, msg, 0, CODE_LEN);
    while (roomId.Length < ROOM_ID_LEN)
    {
        roomId = "0" + roomId;
    }
    System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(roomId), 0, msg, 3, ROOM_ID_LEN);
    client.Send(msg);
}

```

```

// when we call this function we may get the first question.
// it happens if the admin started his game
public string[] recieveUsers()
{
    string[] res;
    byte[] ans = new byte[10000];

    //try getting updates with timeout
    client.ReceiveTimeout = 100;
    try
    {
        int bytesRec = client.Receive(ans);
    }
    catch
    {
        client.ReceiveTimeout = -1;
        res = new string[1];
        res[0] = "error";
        return res;
    }
    client.ReceiveTimeout = -1;

    string code = System.Text.Encoding.Default.GetString(ans, 0, 3);
    int offset;

    //first question recieved from server
    if (code.Equals(SERVER_QUESTION))
    //add check for fail if (Int32.Parse(System.Text.Encoding.Default.GetString(ans, 3 , 3))==0)
    {
        int len;
        offset = 3;
        res = new string[6];
        res[0] = code;
        for (int i = 1; i < 6; i++)
        {
            len = Int32.Parse(System.Text.Encoding.Default.GetString(ans, offset, 3));
            res[i] = System.Text.Encoding.Default.GetString(ans, offset + 3, len);
            offset += 3 + len;
        }
        return res;
    }
}

```



```

//admin closed room
if (code.Equals(CLOSE_ROOM_ANS))
{
    res = new string[1];
    res[0] = CLOSE_ROOM_ANS;
    return res;
}
int numOfUsers = Int32.Parse(System.Text.Encoding.Default.GetString(ans, 3, 1));
res = new string[numOfUsers + 1];
res[0] = code;
int nameLen;
offset = 0;
for (int i = 1; i < numOfUsers + 1; i++)
{
    nameLen = Int32.Parse(System.Text.Encoding.Default.GetString(ans, 4 + offset, 2));
    res[i] = System.Text.Encoding.Default.GetString(ans, 4 + offset + 2, nameLen);
    offset += 2 + nameLen;
}
return res;
}

//ask for the users list
public string[] getUsers(string roomId)
{
    askForUsers(roomId);
    return recieveUsers();
}

//sends name of a user and ID of room and put the user inside the room
public string joinRoom(string roomId, string roomName)
{
    myRoom = roomName;
    byte[] msg = new byte[CODE_LEN + ROOM_ID_LEN];
    System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(JOIN_ROOM), 0, msg, 0, CODE_LEN);
    while (roomId.Length < ROOM_ID_LEN)
    {
        roomId = "0" + roomId;
    }
    System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(roomId), 0, msg, 3, ROOM_ID_LEN);
    client.Send(msg);

    byte[] ans = new byte[8];
    int bytesRec = client.Receive(ans);
    string code = System.Text.Encoding.Default.GetString(ans, 0, 3);
    if (!code.Equals(JOIN_ROOM_ANS))
    {
        return "ERROR";
    }
    int status = Int32.Parse(System.Text.Encoding.Default.GetString(ans, 3, 1));
    if (status == 2)
        return "ROOM NOT EXIST OR OTHER ERROR";
    else if (status == 1)
        return "ROOM IS FULL";
    else if (status == 0)
    {
        numberOfQuestions = Int32.Parse(System.Text.Encoding.Default.GetString(ans, 4, 2));
        if (numberOfQuestions < 1 || numberOfQuestions > 99)

```

```

        return "wrong number of questions recieved from server";
        timePerQuestion = Int32.Parse(System.Text.Encoding.Default.GetString(ans, 6, 2));
        if (timePerQuestion < 1 || timePerQuestion > 99)
            return "wrong time recieved recieved from server";
        return "SUCCESS";
    }
    return "UNKNOWN ERROR";
}

```

//takes the room ID and sends to the server for leaving the room

```

public string leaveRoom(string roomId)
{
    byte[] msg = new byte[CODE_LEN];
    System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(LEAVE_ROOM), 0, msg, 0, CODE_LEN);
    client.Send(msg);
    byte[] ans = new byte[4];
    int bytesRec = client.Receive(ans);
    string code = System.Text.Encoding.Default.GetString(ans, 0, 3);
    int status = Int32.Parse(System.Text.Encoding.Default.GetString(ans, 3, 1));
    if (!code.Equals(LEAVE_ROOM_ANS) || status != 0)
    {
        return "error: got message code: " + code;
    }
    return "success";
}

```

//convert number to string

```

public string numToString(int num, int requestLen)
{
    string strNum = num.ToString();
    while (strNum.Length < requestLen)
    {
        strNum = "0" + strNum;
    }
    return strNum;
}

```

//create room and sends the name of the room, the number of players that
 //can get into the room and the number of questions, and the time for a question
 //gets in return if the room has been created or not.

```

public string createRoom(string roomName, int playersNumber, int questionsNumber, int questionTime)
{
    numberOfQuestions = questionsNumber;
    timePerQuestion = questionTime;
    numberOfPlayers = playersNumber;
    myRoom = roomName;

```

```

        byte[] msg = new byte[CODE_LEN + ROOM_NAME_LEN + roomName.Length + PLAYERS_LEN +
QUESTIONS_NUM_LEN + TIME_PER_QUESTION_LEN];
        System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(CREATE_ROOM), 0, msg, 0, CODE_LEN);
        addToMsg(roomName, ROOM_NAME_LEN, CODE_LEN, ref msg);
        System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(numToString(playersNumber, 1)), 0, msg, CODE_LEN +
ROOM_NAME_LEN + roomName.Length, 1);
        System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(numToString(questionsNumber, 2)), 0, msg, CODE_LEN
+ ROOM_NAME_LEN + roomName.Length + 1, 2);
        System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(numToString(timePerQuestion, 2)), 0, msg, CODE_LEN
+ ROOM_NAME_LEN + roomName.Length + 3, 2);

```

```

    client.Send(msg);

    byte[] ans = new byte[4];
    int bytesRec = client.Receive(ans);
    string code = System.Text.Encoding.Default.GetString(ans, 0, 3);
    if (!code.Equals(CREATE_ROOM_ANS))
    {
        return "error: got message code: " + code;
    }
    int status = Int32.Parse(System.Text.Encoding.Default.GetString(ans, 3, 1));
    if (status == 0)
        return "SUCCESS";
    else
    {
        return "create room failed, code msg: " + code + status.ToString();
    }
}

//sends msg to close specific room.
public void sendCloseRoom()
{
    byte[] msg = new byte[CODE_LEN];
    System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(CLOSE_ROOM), 0, msg, 0, CODE_LEN);
    client.Send(msg);
}

//gets room to close
public string recvCloseRoom()
{
    byte[] ans = new byte[3];
    int bytesRec = client.Receive(ans);
    string code = System.Text.Encoding.Default.GetString(ans, 0, 3);
    if (!code.Equals(CLOSE_ROOM_ANS, StringComparison.Ordinal))
    {
        return "close error";
    }
    else
    {
        return "success";
    }
}

//starting the game with the msg from the protocol
public void startGame()
{
    byte[] msg = new byte[CODE_LEN];
    System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(START_GAME), 0, msg, 0, CODE_LEN);
    client.Send(msg);
}

```

```

//get question from the DB
public string[] getQuestion()
{
    string[] res;
    byte[] ans = new byte[10000];
    int bytesRec = client.Receive(ans);
    string code = System.Text.Encoding.Default.GetString(ans, 0, 3);

    if (code.Equals(SERVER_QUESTION))
    //add check for fail if (Int32.Parse(System.Text.Encoding.Default.GetString(ans, 3, 3))==0)
    {
        int len, offset = 3;
        res = new string[5];
        for (int i = 0; i < 5; i++)
        {
            len = Int32.Parse(System.Text.Encoding.Default.GetString(ans, offset, 3));
            res[i] = System.Text.Encoding.Default.GetString(ans, offset + 3, len);
            offset += 3 + len;
        }
    }
    else if (code.Equals(GAME_OVER))
    {
        int len, offset = 4;
        int usersNumber = Int32.Parse(System.Text.Encoding.Default.GetString(ans, 3, 1));
        res = new string[usersNumber * 2];
        for (int i = 0; i < usersNumber*2; i += 2)
        {
            len = Int32.Parse(System.Text.Encoding.Default.GetString(ans, offset, 2));
            res[i] = System.Text.Encoding.Default.GetString(ans, offset + 2, len); //username
            res[i + 1] = System.Text.Encoding.Default.GetString(ans, offset + 2 + len, 2); //score
            offset += 2 + len + 2;
        }
    }
    else
    {
        res = new string[1];
        res[0] = "error";
    }

    return res;
}

//send answer according to the protocol msg
public void sendAnswer(char answer, int time)
{
    byte[] msg = new byte[CODE_LEN + 1 + 2];
    System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(SEND_ANSWER), 0, msg, 0, CODE_LEN);
    msg[CODE_LEN] = Convert.ToByte(answer);

    System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(numToString(time, 2)), 0, msg, CODE_LEN + 1, 2);
    client.Send(msg);
    sumQuestions++;
    sumTime += time;
}

```

```

//sends response to the question from the server.
public string[] sendAnswerResponse()
{
    string[] res;
    byte[] ans = new byte[10000];
    int bytesRec = client.Receive(ans);
    string code = System.Text.Encoding.Default.GetString(ans, 0, 3);
    if (code.Equals(SEND_ANSWER_ANS, StringComparison.Ordinal))
    {
        res = new string[1];
        res[0] = System.Text.Encoding.Default.GetString(ans, 3, 1);
        if (res[0].Equals("1"))
            sumRight++;
        else if (res[0].Equals("0"))
            sumWrong++;
    }
    else
    {
        res = new string[1];
        res[0] = "error";
    }

    return res;
}

//Leaving specific game
public void LeaveGame()
{
    byte[] msg = new byte[CODE_LEN];
    System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(LEAVE_GAME), 0, msg, 0, CODE_LEN);
    client.Send(msg);
}

//getting the best scores from the DB
public string[] geBestScores()
{
    byte[] msg = new byte[CODE_LEN];
    System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(GET_BEST_SCORES), 0, msg, 0, CODE_LEN);
    client.Send(msg);

    string[] res;
    byte[] ans = new byte[1000];
    int bytesRec = client.Receive(ans);
    string code = System.Text.Encoding.Default.GetString(ans, 0, 3);
    if (!code.Equals(BEST_SCORES_ANS))
    {
        res = new string[1];
        res[0] = "error";
    }
    else
    {
        int nameLen, offset = 3;
        res = new string[3 * 2];
    }
}

```

```

        for (int i = 0; i < 6; i += 2)
        {
            nameLen = Int32.Parse(System.Text.Encoding.Default.GetString(ans, offset, 2));
            res[i] = System.Text.Encoding.Default.GetString(ans, offset + 2, nameLen); //username
            res[i + 1] = System.Text.Encoding.Default.GetString(ans, offset + 2 + nameLen, 6);
//score

            offset += 2 + nameLen + 6;
        }
    }

    return res;
}

//ask for the specific status of the user that got entered to the game
public double[] getMyStatus()
{
    byte[] msg = new byte[CODE_LEN];
    System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(GET_MY_STATUS), 0, msg, 0, CODE_LEN);
    client.Send(msg);

    double[] res;
    byte[] ans = new byte[3 + 4 + 6 + 6 + 4];
    int bytesRec = client.Receive(ans);
    string code = System.Text.Encoding.Default.GetString(ans, 0, 3);
    if (!code.Equals(MY_STATUS_ANS, StringComparison.Ordinal))
    {
        res = new double[1];
        res[0] = -1;
    }
    else
    {
        res = new double[4];
        res[0] = Int32.Parse(System.Text.Encoding.Default.GetString(ans, 3, 4)); //numberOfGames
        res[1] = Int32.Parse(System.Text.Encoding.Default.GetString(ans, 7, 6));
//numberOfRightAns
        res[2] = Int32.Parse(System.Text.Encoding.Default.GetString(ans, 13, 6)); //numOfWrongAns
        res[3] = Int32.Parse(System.Text.Encoding.Default.GetString(ans, 19, 4))/100.0;
//avgTimeForAns
    }

    return res;
}

//getting out of the app (sends to the server so the server will know that the app closed)
public void quitApp(string request)
{
    byte[] msg = new byte[CODE_LEN];
    System.Buffer.BlockCopy(Encoding.ASCII.GetBytes(QUIT), 0, msg, 0, CODE_LEN);
    client.Send(msg);
    client.Close();
    Application.Exit();
    Environment.Exit(0);
}
}
}

```

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace trivya
{
    public partial class CreateRoom : Form
    {
        Client client;
        public CreateRoom(Client cl)
        {
            client = cl;
            InitializeComponent();
            username.Text = client.myName;
            this.StartPosition = FormStartPosition.CenterScreen;
        }

        private void back_Click(object sender, EventArgs e)
        {
            this.Owner.Show();
            this.Close();
        }

        private void send_Click(object sender, EventArgs e)
        {
            if (roomName.Text.Length > 99 || roomName.Text.Length < 1)
            {
                errorLabel.Visible = true;
                errorLabel.Text = "wrong room name";
                return;
            }
            else if (Int32.Parse(numberOfPlayers.Text) > 9 || Int32.Parse(numberOfPlayers.Text) < 1)
            {
                errorLabel.Visible = true;
                errorLabel.Text = "wrong number of players";
                return;
            }
            else if (Int32.Parse(numberOfQuestions.Text) > 99 || Int32.Parse(numberOfQuestions.Text) < 1)
            {
                errorLabel.Visible = true;
                errorLabel.Text = "wrong number of questions";
                return;
            }
            else if (Int32.Parse(time.Text) > 99 || Int32.Parse(time.Text) < 1)
            {
                errorLabel.Visible = true;
                errorLabel.Text = "wrong time";
                return;
            }
        }
    }
}

```

```

        string res = client.createRoom(roomName.Text, Int32.Parse(numberOfPlayers.Text),
Int32.Parse(numberOfQuestions.Text), Int32.Parse(time.Text));
        if (res.Equals("SUCCESS"))
        {
            WaitForGame W = new WaitForGame(client, "", roomName.Text);
            W.Owner = this.Owner;
            W.Show();
            this.Close();
        }
        else
        {
            errorLabel.Visible = true;
            errorLabel.Text = res;
        }
    }

}

}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

using System.Threading;

namespace trivya
{
    public partial class CustomMessageBox : Form
    {
        public CustomMessageBox()
        {
            InitializeComponent();
        }

        static CustomMessageBox msgBox;

        public static void Show(string msg, string caption)
        {
            msgBox = new CustomMessageBox();
            msgBox.StartPosition = FormStartPosition.CenterScreen;
            msgBox.label1.Text = msg;
            msgBox.label1.TextAlign = ContentAlignment.MiddleCenter;
            msgBox.AutoSize = true;
            msgBox.Text = caption;
            msgBox.button1.Dock = DockStyle.Bottom;

            msgBox.tableLayoutPanel.Left = (msgBox.ClientSize.Width - msgBox.tableLayoutPanel.Width) / 2;
            msgBox.tableLayoutPanel.Top = (msgBox.ClientSize.Height - msgBox.tableLayoutPanel.Height) / 2;
        }
    }
}

```



```

        msgBox.FormBorderStyle = System.Windows.Forms.FormBorderStyle.Fixed3D;

        msgBox.ShowDialog();
        msgBox.BringToFront();

    }

    private void button1_Click(object sender, EventArgs e)
    {
        msgBox.Close();
    }
}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

using System.Diagnostics;

namespace trivya
{
    public partial class Game : Form
    {
        int counter;
        Client client;
        int time;
        int rightAnswers = 0;
        int qNumber;
        Stopwatch sw = new Stopwatch();

        System.Windows.Forms.Timer countDownTimer = new System.Windows.Forms.Timer();
        private void timer1_Tick(object sender, EventArgs e)
        {
            counter--;
            timeLeft.Text = counter.ToString();
            if (counter == 0)
                countDownTimer.Stop();
        }

        static bool exitFlag = false;

        //this timer is for creating a short delay in order to show the user whether his answer was right
        System.Windows.Forms.Timer myTimer = new System.Windows.Forms.Timer();
        private void myTimerEventProcessor(Object myObject, EventArgs myEventArgs)
        {
            myTimer.Stop();
            exitFlag = true;
        }
    }
}

```

```

}

System.Windows.Forms.Timer timeForQuestion = new System.Windows.Forms.Timer();
private void timeEventProcessor(Object myObject, EventArgs myEventArgs)
{
    timeForQuestion.Stop();
    sw.Stop();

    client.sendAnswer('5', 10);
    string[] response = client.sendAnswerResponse();
    ans1.BackColor = Color.Red;
    ans2.BackColor = Color.Red;
    ans3.BackColor = Color.Red;
    ans4.BackColor = Color.Red;
    make_gui_pause();
    get_question_or_gameOver();

}

public Game( Client cl, string[] q)
{
    client = cl;
    InitializeComponent();
    username.Text = client.myName;
    roomName.Text = client.myRoom;

    this.StartPosition = FormStartPosition.CenterScreen;
    string[] question;
    time = client.timePerQuestion;
    qNumber = 1;

    counter = time;

    countdownTimer.Tick += new EventHandler(timer1_Tick);
    countdownTimer.Interval = 1000;
    countdownTimer.Start();
    timeLeft.Text = counter.ToString();

    if (q == null)
    {
        client.startGame();
        question = client.getQuestion();
    }
    else
    {
        question = q;
    }

    UpdateQuestionAndAnswer(question);
    myTimer.Tick += new EventHandler(myTimerEventProcessor);
    myTimer.Interval = 500;

    timeForQuestion.Tick += new EventHandler(timeEventProcessor);
}

```

```

        timeForQuestion.Interval = client.timePerQuestion*1000;

        timeForQuestion.Start();
        sw.Start();
    }

    private void UpdateQuestionAndAnswer(string[] question)
    {
        labQues.Text = question[0];
        ans1.Text = question[1];
        ans2.Text = question[2];
        ans3.Text = question[3];
        ans4.Text = question[4];

        ans1.BackColor = SystemColors.Control;
        ans2.BackColor = SystemColors.Control;
        ans3.BackColor = SystemColors.Control;
        ans4.BackColor = SystemColors.Control;

        questionNumber.Text = "Question: " + qNumber.ToString() + "/" + client.numberOfQuestions;
        score.Text = "Score: " + rightAnswers.ToString() + "/" + (qNumber - 1).ToString();
        qNumber++;
    }

    private void exit_Click(object sender, EventArgs e)
    {
        timeForQuestion.Stop();
        countdownTimer.Stop();
        sw.Stop();
        client.LeaveGame();
        this.Owner.Show();
        this.Close();
    }

    void get_question_or_gameOver()
    {
        string[] ans = client.getQuestion();
        if (qNumber <= client.numberOfQuestions)
        {
            UpdateQuestionAndAnswer(ans);
            counter = client.timePerQuestion;
            timeLeft.Text = counter.ToString();
            countdownTimer.Start();
            sw.Restart();
            timeForQuestion.Start();
        }
        else //game over
        {
            string summary = "";
            for (int i = 0; i < ans.Length; i += 2)
            {
                summary += "username: " + ans[i] + "      score: " + Int32.Parse(ans[i + 1]).ToString() +
"\n";
            }
        }
    }

```

```

        CustomMessageBox.Show(summary, "Scores");

        this.Owner.Show();
        this.Close();
    }

}

void make_gui_pause()
{
    exitFlag = false;

    ans1.Enabled = false;
    ans2.Enabled = false;
    ans3.Enabled = false;
    ans4.Enabled = false;
    myTimer.Start();

    while (exitFlag == false)
    {
        Application.DoEvents();
    }
    ans1.Enabled = true;
    ans2.Enabled = true;
    ans3.Enabled = true;
    ans4.Enabled = true;
}

private void ans_Click(object sender, EventArgs e)
{
    timeForQuestion.Stop();
    countdownTimer.Stop();
    sw.Stop();

    Button button = (Button)sender;

    time = (int)sw.ElapsedMilliseconds/1000;

    client.sendAnswer(button.Name[3], time);
    string[] response = client.sendAnswerResponse();
    if (response[0].Equals("1"))
    {
        rightAnswers++;
        button.BackColor = Color.GreenYellow;
    }
    else
    {
        button.BackColor = Color.Red;
    }
    make_gui_pause();

    get_question_or_gameOver();
}
}
}

```

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace trivya
{
    public partial class JoinRoom : Form
    {
        Client client;
        string[] roomsIdAndName;
        string[] users;

        public JoinRoom(Client cl)
        {
            client = cl;
            InitializeComponent();
            username.Text = client.myName;

            updateRoomsList();
            if (roomsIdAndName.Length == 0)
            {
                available.Visible = true;
                join.Enabled = false;
            }

            this.StartPosition = FormStartPosition.CenterScreen;
        }

        public void updateRoomsList()
        {
            playersHeader.Visible = false;
            playersList.Visible = false;
            roomsIdAndName = client.getRooms();
            if (roomsIdAndName.Length > 0)
                available.Visible = false;
            else
                available.Visible = true;
            roomsList.Items.Clear();
            for (int i = 0; i < roomsIdAndName.Length; i += 2)
            {
                roomsList.Items.Add(roomsIdAndName[i + 1]);
            }
        }
    }
}

```

```

//first item in users will be the code , so we can know what message the user got
public bool updateUsersList()
{
    if (roomsList.SelectedIndex < 0) //wrong click
        return false;
    users = client.getUsers(roomsIdAndName[roomsList.SelectedIndex * 2]);
    playersList.Items.Clear();
    string code = users[0];
    for (int i = 1; i < users.Length; i++)
    {
        playersList.Items.Add(users[i]);
    }
    return true;
}

private void roomsList_SelectedIndexChanged(object sender, EventArgs e)
{
    if (updateUsersList())
    {
        join.Enabled = true;
        playersHeader.Visible = true;
        playersList.Visible = true;
    }
}

private void join_Click(object sender, EventArgs e)
{
    string res = client.joinRoom(roomsIdAndName[roomsList.SelectedIndex * 2],
roomsIdAndName[roomsList.SelectedIndex * 2 + 1]);
    if (res.Equals("SUCCESS"))
    {
        WaitForGame W = new WaitForGame(client, roomsIdAndName[roomsList.SelectedIndex * 2],
roomsIdAndName[roomsList.SelectedIndex * 2+1]);
        W.Owner = this.Owner;
        W.Show();
        this.Close();
    }
    else
    {
        errorLabel.Text = res;
        errorLabel.Visible = true;
    }
}

private void back_Click(object sender, EventArgs e)
{
    this.Owner.Show();
    this.Close();
}

private void refresh_Click(object sender, EventArgs e)
{
    updateRoomsList();
    join.Enabled = false;
}
}
}

```

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace trivya
{
    public partial class Menu : Form
    {
        public static Client client;

        public Menu()
        {
            client = new Client();
            InitializeComponent();
            this.StartPosition = FormStartPosition.CenterScreen;
        }

        private void signIn_Click(object sender, EventArgs e)
        {
            if (String.IsNullOrEmpty(signInUser.Text) || String.IsNullOrEmpty(signInPass.Text))
            {
                labErrSignIn.Text = "Please fill all fields";
                labErrSignIn.Visible = true;
                return;
            }

            string res = client.signIn(signInUser.Text, signInPass.Text);

            if (String.Compare(res, "Success") != 0)
            {
                labErrSignIn.Visible = true;
                labErrSignIn.Text = res;
                return;
            }

            //sign in text and lables
            signIn.Visible = false;
            labPassSignIn.Visible = false;
            labUserSignIn.Visible = false;
            signInUser.Visible=false;
            signInPass.Visible = false;
            labErrSignIn.Visible = false;
            panel.BackColor = System.Drawing.Color.Black;

            labHello.Visible = true;
            labHello.Text = "Hello " + signInUser.Text;

            signUp.Text = "Sign out";
        }
    }
}

```

```

        myStatus.Enabled = true;
        joinRoom.Enabled = true;
        createRoom.Enabled = true;
        scores.Enabled = true;
        username.Visible = true;
        username.Text = client.myName;
    }
    private void signUp_Click(object sender, EventArgs e)
    {
        if (String.Compare(signUp.Text, "Sign out") == 0)
        {
            client.signOut();

            labHello.Visible = false;

            //sign in text and lables
            //sign in text and lables
            signIn.Visible = true;
            labPassSignIn.Visible = true;
            labUserSignIn.Visible = true;
            signInUser.Visible = true;
            signInPass.Visible = true;

            panel.BackColor = System.Drawing.Color.White;
            signInUser.Text = "";
            signInPass.Text = "";

            signUp.Text = "Sign up";

            myStatus.Enabled = false;
            joinRoom.Enabled = false;
            createRoom.Enabled = false;
            scores.Enabled = false;
            username.Visible = false;
            return;
        }

        SignUp signUpForm = new SignUp(client);
        signUpForm.Owner = this;
        this.Hide();
        signUpForm.Show();
    }
    private void quit_Click(object sender, EventArgs e)
    {
        Application.Exit();
    }
    private void createRoom_Click(object sender, EventArgs e)
    {
        this.Hide();
    }

```



```

        CreateRoom createRoomForm = new CreateRoom(client);
        createRoomForm.Owner = this;
        createRoomForm.Show();
    }
    private void joinRoom_Click(object sender, EventArgs e)
    {
        this.Hide();
        JoinRoom joinRoomForm = new JoinRoom(client);
        joinRoomForm.Owner = this;
        joinRoomForm.Show();
    }
    private void scores_Click(object sender, EventArgs e)
    {
        this.Hide();
        Scores sc = new Scores(client);
        sc.Owner = this;
        sc.Show();
    }
    private void myStatus_Click(object sender, EventArgs e)
    {
        this.Hide();
        MyStatus sc = new MyStatus(client);
        sc.Owner = this;
        sc.Show();
    }
}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace trivya
{
    public partial class MyStatus : Form
    {
        Client client;
        public MyStatus(Client cl)
        {
            InitializeComponent();
            this.StartPosition = FormStartPosition.CenterScreen;
            client = cl;
            username.Text = client.myName;

            double[] myStatus = client.getMyStatus();
            string[] strings = { "number of games", "number of right answers", "number of wrong answers",
"average time for answer" };
            userData.Text = "";
            for (int i = 0; i < 4; i++)
            {
                userData.Text += strings[i] + " - " + myStatus[i].ToString() + "\n";
            }
        }
    }
}

```

```

    }
    private void back_Click(object sender, EventArgs e)
    {
        this.Owner.Show();
        this.Close();
    }
}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace trivya
{
    public partial class Scores : Form
    {
        Client client;
        public Scores(Client cl)
        {
            InitializeComponent();
            this.StartPosition = FormStartPosition.CenterScreen;
            client = cl;
            username.Text = client.myName;
            string[] best = client.geBestScores();
            bestScores.Text = "";
            for (int i = 0; i < 6; i += 2)
            {
                bestScores.Text += best[i] + " - " + Int32.Parse(best[i + 1]).ToString() + "\n\n";
            }
        }

        private void back_Click(object sender, EventArgs e)
        {
            this.Owner.Show();
            this.Close();
        }
    }
}

```

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace trivya
{
    public partial class SignUp : Form
    {
        Client client;
        public SignUp(Client cl)
        {
            client = cl;
            InitializeComponent();
            this.StartPosition = FormStartPosition.CenterScreen;
        }
        private void ok_Click(object sender, EventArgs e)
        {
            if (String.IsNullOrEmpty(User.Text) || String.IsNullOrEmpty(Pass.Text) ||
String.IsNullOrEmpty(Email.Text))
            {
                labResSignUp.Text = "Please fill all fields";
                labResSignUp.Visible = true;
            }
            else
            {
                labResSignUp.Text = client.signUp(User.Text, Pass.Text, Email.Text);
                labResSignUp.Visible = true;
                if (String.Compare(labResSignUp.Text, "Success") == 0)
                {
                    this.Owner.Show();
                    this.Close();
                }
            }
            return;
        }
        private void back_Click(object sender, EventArgs e)
        {
            this.Owner.Show();
            this.Close();
        }
    }
}

```

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

using System.Threading;

namespace trivia
{
    public partial class WaitForGame : Form
    {
        Client client;
        string roomID;

        string[] users;

        bool adminStartedGame = false;
        bool exitRoom = false;
        Thread roomListUpdateThread;

        //first item in users will be the code , so we can know what message the user got
        public void updateUsersList()
        {
            users = client.receiveUsers();

            if (users.Length != 1 && !users[0].Equals(Codes.SERVER_QUESTION))
            {
                if (playersList.InvokeRequired)
                    Invoke((MethodInvoker)delegate { playersList.Items.Clear(); });
                else
                    playersList.Items.Clear();
                string code = users[0];
                for (int i = 1; i < users.Length; i++)
                {
                    if (playersList.InvokeRequired)
                        Invoke((MethodInvoker)delegate { playersList.Items.Add(users[i]); });
                    else
                        playersList.Items.Add(users[i]);
                }
            }
        }

        public void listenToUpdates()
        {
            //while (running && (users[0].Equals("error"))) || (users.Length != 1 && !clientClosedRoom))
            while (!users[0].Equals(Codes.SERVER_QUESTION) && !users[0].Equals(Codes.CLOSE_ROOM_ANS) &&
!adminStartedGame && !exitRoom)
            {
                updateUsersList();
            }
            if (users[0].Equals(Codes.SERVER_QUESTION))//if room was closed or user left the room we will just

```

```

        {
            Invoke((MethodInvoker)delegate { string[] question = new string[5]; for (int i = 0; i < 5;
i++) { question[i] = users[i + 1]; } Game g = new Game(client, question); g.Owner = this.Owner; g.Show(); });
            Invoke((MethodInvoker)delegate { this.Close(); });
            return;
        }
        else if (users[0].Equals(Codes.CLOSE_ROOM_ANS))
        {
            Invoke((MethodInvoker)delegate { err.Visible = true; err.Text = "The admin closed the room
press OK to continue"; });
            leaveRoom.Text = "OK";

            return;
        }
    }

    public WaitForGame(Client cl, string roomId, string roomName)
    {
        roomId = roomId;
        client = cl;
        InitializeComponent();
        username.Text = client.myName;
        this.roomName.Text = client.myRoom;
        if (client.numberOfPlayers > 0)
            roomDetails.Text = "max_number_players:" + client.numberOfPlayers.ToString() + "
number_of_questions:" + client.numberOfQuestions.ToString() + " time_per_question:" +
client.timePerQuestion.ToString();
        else
            roomDetails.Text = "number_of_questions:" + client.numberOfQuestions.ToString() + "
time_per_question:" + client.timePerQuestion.ToString();
        this.StartPosition = FormStartPosition.CenterScreen;
        if (roomId.Equals(""))
        {
            leaveRoom.Text = "Close Room";
            startGame.Visible = true;
            playersList.Items.Add(client.myName);
            // doing this only because the thread checks out users
            users = new string[2];
            users[0] = Codes.GET_USERS_ANS;
            users[1] = client.myName;
        }
        else
        {
            client.askForUsers(roomId);
            updateUserList();
        }

        header.Text = "You are connected to room " + roomName;
        roomListUpdateThread = new Thread(listenToUpdates);
        roomListUpdateThread.Start();
    }

    private void startGame_Click(object sender, EventArgs e)
    {
        adminStartedGame = true;
        System.Threading.Thread.Sleep(200);
        Game g = new Game(client, null);
    }

```

```

        g.Owner = this.Owner;
        g.Show();
        this.Close();
    }

    private void leaveRoom_Click(object sender, EventArgs e)
    {
        exitRoom = true;
        //if room was not closed by admin we do this
        if (!"OK".Equals(leaveRoom.Text))
        {
            System.Threading.Thread.Sleep(200);
            if (roomID.Equals(""))
            {
                client.sendCloseRoom();
                string res = client.recvCloseRoom();
            }
            else
            {
                client.leaveRoom(roomID);
            }
        }
        this.Owner.Show();
        this.Close();
    }
}
}
}

```

נספח ה' – קוד השרת

```
#pragma once

#include <string>
#include <vector>
#include "Helper.h"
#include "sqlite3.h"
#include "Question.h"
using namespace std;

class DataBase
{
private:
    sqlite3* _sqlite;

    static int callbackCount(void* notUsed, int argc, char** argv, char** azCol)
    {
        int* res = (int*)notUsed;
        *res = atoi(argv[0]);

        return 0;
    }

    static int callbackQuestions(void* notUsed, int argc, char** argv, char** azCol)
    {
        vector<Question*>* res = (vector<Question*>*)notUsed;

        Question* currQuest;

        currQuest = new Question(atoi(argv[0]), argv[1], argv[2], argv[3], argv[4], argv[5]);
        res->push_back(currQuest);

        return 0;
    }

    static int callbackBestScores(void* notUsed, int argc, char** argv, char** azCol)
    {
        vector<std::string*> res = (vector<std::string*>*)notUsed;

        std::string username = argv[0];
        int count = atoi(argv[1]);

        std::string s = Helper::getPaddedNumber(username.length(), 2) + username +
        Helper::getPaddedNumber(count, 6);
        res->push_back(s);

        return 0;
    }
}
```

```

static int callbackPersonalStatus(void* notUsed, int argc, char** argv, char** azCol)
{
    vector<std::string>* res = (vector<std::string>*)notUsed;

    // if user has no data
    string s1 = argv[0];
    if (s1 == "0")
    {
        res->push_back("0"); // total games
        res->push_back("0"); // correct answers
        res->push_back("0"); // wrongs answers
        res->push_back("0"); // average answer time
    }
    else
    {
        res->push_back(argv[0]); // total games
        res->push_back(argv[1]); // correct answers
        res->push_back(argv[2]); // wrongs answers

        string avg = argv[3];

        char buffer[50];
        sprintf(buffer, "%.2f", atof(avg.c_str()));

        int i = atoi(buffer);
        string s = "";
        if (i < 10)
        {
            s = "0";
            s += buffer[0];
            s += buffer[2];
            s += buffer[3];
        }
        else
        {
            s = buffer[0];
            s += buffer[1];
            s += buffer[3];
            s += buffer[4];
        }
        res->push_back(s); // average answer time
    }
    return 0;
}

public:
DataBase()
{
    sqlite3_open("trivia.db", &_sqlite);

    // should check if succeeded
}

~DataBase()
{
    sqlite3_close(_sqlite);
    _sqlite = nullptr;
}

```



```

}

bool isUserExists(string username)
{
    std::string sql = "select count(*) from t_users where username = \'' + username + '\''";
    int res = 0;

    char *zErrMsg = 0;
    int m = sqlite3_exec(_sqlite, sql.c_str(), callbackCount, &res, &zErrMsg);

    if (m != SQLITE_OK)
    {
        TRACE("DB::isUserExists failed: err_msg=%s, sql=%s", zErrMsg, sql.c_str());
        sqlite3_free(zErrMsg);
        return false;
    }
    return res != 0;
}

bool addNewUser(string username, string password, string email)
{
    std::string sql = "insert into t_users (username, password, email) values('\' + username + '\', \' + password + '\', \' + email + '\')";
    char *zErrMsg = 0;
    int m = sqlite3_exec(_sqlite, sql.c_str(), nullptr, nullptr, &zErrMsg);

    if (m != SQLITE_OK)
    {
        TRACE("DB::addNewUser failed: err_msg=%s, sql=%s", zErrMsg, sql.c_str());
        sqlite3_free(zErrMsg);
        return false;
    }

    return true;
}

bool isUserAndPassMatch(string username, string password)
{
    std::string sql = "select count(*) from t_users where username = \'' + username + '\'' and password = \'' + password + '\''";
    int res = 0;

    char *zErrMsg = 0;
    int m = sqlite3_exec(_sqlite, sql.c_str(), callbackCount, &res, &zErrMsg);

    if (m != SQLITE_OK)
    {
        TRACE("DB::isUserAndPassMatch failed: err_msg=%s, sql=%s", zErrMsg, sql.c_str());
        sqlite3_free(zErrMsg);
        return false;
    }

    return res == 1;
}

```

```

vector<Question*> initQuestions(int questionsNo)
{
    std::string sql = "select question_id, question, correct_ans, ans2, ans3, ans4 from
t_questions order by random() limit " + to_string(questionsNo);

    vector<Question*> res;

    char *zErrMsg = 0;
    int m = sqlite3_exec(_sqlite, sql.c_str(), callbackQuestions, &res, &zErrMsg);

    if (m != SQLITE_OK)
    {
        TRACE("DB::initQuestions failed: err_msg=%s, sql=%s", zErrMsg, sql.c_str());
        sqlite3_free(zErrMsg);
    }

    return res;
}

vector<string> getBestScores()
{
    std::string sql = "select username, sum(is_correct) as ans from t_players_answers group by
username order by ans DESC limit 3";

    vector<std::string> res;

    char *zErrMsg = 0;
    int m = sqlite3_exec(_sqlite, sql.c_str(), callbackBestScores, &res, &zErrMsg);

    if (m != SQLITE_OK)
    {
        TRACE("DB::getBestScores failed: err_msg=%s, sql=%s", zErrMsg, sql.c_str());
        sqlite3_free(zErrMsg);
    }

    return res;
}

vector<string> getPersonalStatus(string username)
{
    std::string sql = "select count(distinct game_id), sum(is_correct) as correct, count(*) -
sum(is_correct), avg(answer_time) from t_players_answers where username = '" + username + "'";

    vector<std::string> res;

    char *zErrMsg = 0;
    int m = sqlite3_exec(_sqlite, sql.c_str(), callbackPersonalStatus, &res, &zErrMsg);

    if (m != SQLITE_OK)
    {
        TRACE("DB::getPersonalStatus failed: err_msg=%s, sql=%s", zErrMsg, sql.c_str());
        sqlite3_free(zErrMsg);
    }

    return res;
}

```

```

int insertNewGame()
{
    std::string sql = "insert into t_games (status, start_time) values(0, DATETIME('\now\'))";
    char *zErrMsg = 0;
    int m = sqlite3_exec(_sqlite, sql.c_str(), nullptr, nullptr, &zErrMsg);

    if (m != SQLITE_OK)
    {
        TRACE("DB::insertNewGame failed: err_msg=%s, sql=%s", zErrMsg, sql.c_str());
        sqlite3_free(zErrMsg);
        return -1;
    }

    sql = "select last_insert_rowid()";
    int res = 0;

    m = sqlite3_exec(_sqlite, sql.c_str(), callbackCount, &res, &zErrMsg);

    if (m != SQLITE_OK)
    {
        TRACE("DB::insertNewGame failed: err_msg=%s, sql=%s", zErrMsg, sql.c_str());
        sqlite3_free(zErrMsg);
        return -1;
    }

    return res;
}

bool updateGameStatus(int id)
{
    std::string sql = "update t_games set status=1, end_time=DATETIME('\now\') where game_id = " +
to_string(id);
    char *zErrMsg = 0;
    int m = sqlite3_exec(_sqlite, sql.c_str(), nullptr, nullptr, &zErrMsg);

    if (m != SQLITE_OK)
    {
        TRACE("DB::updateGameStatus failed: err_msg=%s, sql=%s", zErrMsg, sql.c_str());
        sqlite3_free(zErrMsg);
        return false;
    }

    return true;
}

bool addAnswerToPlayer(int gameId, string username, int questionId, string answer, bool isCorrect, int
answerTime)
{
    std::string sql = "insert into t_players_answers (game_id, username, question_id, player_answer,
is_correct, answer_time) values(" + to_string(gameId) + ", '\"" + username + "\", " + to_string(questionId) +
", '\"" + answer + "\", " + to_string(isCorrect ? 1 : 0) + ", " + to_string(answerTime) +
")";

    char *zErrMsg = 0;
    int m = sqlite3_exec(_sqlite, sql.c_str(), nullptr, nullptr, &zErrMsg);

    if (m != SQLITE_OK)
    {

```

```

        TRACE("DB::addAnswerToPlayer failed:  err_msg=%s, sql=%s", zErrMsg, sql.c_str());
        sqlite3_free(zErrMsg);
        return false;
    }

    return true;
}

};

#pragma once

#include <vector>
#include "User.h"
#include <map>
#include "Question.h"
#include <string>
#include "DataBase.h"

class User;

class Game
{
private:
    std::vector<Question*> _questions;
    std::vector<User*> _players;
    int _questionsNo;
    int _currQuestionIndex;
    int _id;
    DataBase& _db;
    std::map<std::string, int> _results;
    int _currentTurnAnswers; // the number of player that answer to the question in the current turn (so
we know then to move to the next question)

    bool insertGameToDB();
    void initQuestionsFromDB();
    void sendQuestionToAllUsers();

public:
    // throw exception if didnt success
    Game(const std::vector<User*>& players, int questionsNo, DataBase& db);

    void sendFirstQuestion();
    void handleFinishGame();
    bool handleNextTurn();
    bool handleAnswerFromUser(User* user, int answerIndex, int time);
    bool leaveGame(User*);
    int getID();
    ~Game();
};

```

```

#include "Game.h"
Game::Game(const vector<User*>& players, int questionsNo, DataBase& db) : _db(db)
{
    _currQuestionIndex = 0;
    _questionsNo = questionsNo;

    if (!insertGameToDB())
    {
        throw exception("Insert new game into db failed");
    }

    initQuestionsFromDB();

    int i;
    for (i = 0; i < players.size(); ++i)
    {
        _players.push_back(players[i]);
        pair<string, int> p(players[i]->getUsername(), 0);
        _results.insert(p);

        _players[i]->setGame(this);
    }
}

void Game::sendQuestionToAllUsers()
{
    vector<User*>::iterator it;
    User* currUser;

    string qst = _questions[_currQuestionIndex]->getQuestion();
    string msg = SRV_MSG_GAME_QUESTION + Helper::getPaddedNumber(qst.length(), 3) + qst;

    string* ans = _questions[_currQuestionIndex]->getAnswers();
    msg += Helper::getPaddedNumber(ans[0].length(), 3) + ans[0];
    msg += Helper::getPaddedNumber(ans[1].length(), 3) + ans[1];
    msg += Helper::getPaddedNumber(ans[2].length(), 3) + ans[2];
    msg += Helper::getPaddedNumber(ans[3].length(), 3) + ans[3];

    _currentTurnAnswers = 0;

    TRACE("sending question to all players (%d/%d)", _currQuestionIndex + 1, _questions.size());
    for (it = _players.begin(); it != _players.end(); it++)
    {
        currUser = *it;

        // send question to player
        try
        {
            currUser->send(msg);
        }
        catch (...)
        {
            TRACE("Sending Error (Game::sendQuestionToAllUsers): cant send message to %s", currUser-
>getUsername().c_str());
        }
    }
}

```

```

    }
}
}
void Game::handleFinishGame()
{
    _db.updateGameStatus(this->_id);

    // no players left
    if (_players.size() == 0)
        return;

    string msg = SRV_MSG_GAME_FINISH + Helper::getPaddedNumber(_results.size(), 1);

    map<string, int>::iterator it;
    for (it = _results.begin(); it != _results.end(); it++)
    {
        msg += Helper::getPaddedNumber(it->first.length(), 2) + it->first + Helper::getPaddedNumber(it->second, 2);
    }

    TRACE("Send finish game to all players");
    for (int i = 0; i < _players.size(); ++i)
    {
        try
        {
            _players[i]->setGame(nullptr);
            _players[i]->send(msg);
        }
        catch (...)
        {
            TRACE("Sending Error (Game::handleFinishGame): cant send message to %s", _players[i]->getUsername().c_str());
        }
    }
}

bool Game::insertGameToDB()
{
    // insert record to DB
    _id = _db.insertNewGame();

    return _id != -1;
}

void Game::initQuestionsFromDB()
{
    _questions = _db.initQuestions(_questionsNo);
}

```

```

// return true if game still on
// false - if game ended
bool Game::leaveGame(User* currUser)
{
    vector<User*>::iterator it= std::find(_players.begin(), _players.end(), currUser);
    if (it != _players.end())
    {
        _players.erase(it);

        // if game finished
        return handleNextTurn();
    }

    return true;
}

int Game::getID()
{
    return _id;
}

bool Game::handleNextTurn()
{
    if (_players.size() == 0)
    {
        handleFinishGame();
        return false;
    }

    if (_currentTurnAnswers == _players.size())
    {
        if (_questions.size() - 1 == _currQuestionIndex)
        {
            // Game Over
            handleFinishGame();
            return false;
        }
        else
        {
            // Next question
            _currQuestionIndex++;
            sendQuestionToAllUsers();
        }
    }

    return true;
}

// return true if game still on
// false - if game ended
bool Game::handleAnswerFromUser(User* user, int answerNo, int time)
{
    _currentTurnAnswers++;

    string result = "0";
    bool res = false;
    int correctAns = _questions[_currQuestionIndex]->getCorrectAnswerIndex();

    TRACE("AnswerNo=%d, time=%d, CorrectAnswer=%d", answerNo, time, correctAns + 1);
}

```

```

    if (answerNo - 1 == correctAns)
    {
        result = "1";
        res = true;
        _results[user->getUsername()] ++;
    }

    string ans = "";

    // if answer=5 it means the user didnt answer within the time
    if (answerNo <= 4)
        ans = _questions[_currQuestionIndex]->getAnswers()[answerNo - 1];

    _db.addAnswerToPlayer(_id, user->getUsername(), _questions[_currQuestionIndex]->getId(), ans , res,
time);

    // send indication to user
    TRACE("Sending answer indiction to user.");
    TRACE("CurrentPlayersNo=%d, CurrentQuestionIndex=%d, TotalQuestionsNo=%d, CurrentTurnAnswres=%d",
_players.size(), _currQuestionIndex, _questions.size(), _currentTurnAnswers);
    user->send(SRV_MSG_GAME_ANSWER_IND + result);

    return handleNextTurn();
}

void Game::sendFirstQuestion()
{
    sendQuestionToAllUsers();
}

Game::~~Game()
{
    vector<Question*>::iterator it;

    for (it = _questions.begin(); it != _questions.end(); it++)
    {
        delete *it;
    }

    _questions.clear();
    _players.clear();
}

```



```

#pragma once

#include <vector>
#include <string>
#include <WinSock2.h>

using namespace std;

class Helper
{
public:

    static int getMessageTypeCode(SOCKET sc);
    static char* getPartFromSocket(SOCKET sc, int bytesNum, int flags);
    static int getIntPartFromSocket(SOCKET sc, int bytesNum);
    static string getStringPartFromSocket(SOCKET sc, int bytesNum);
    static void sendData(SOCKET sc, std::string message);
    static string getPaddedNumber(int num, int digits);

private:
    static char* Helper::getPartFromSocket(SOCKET sc, int bytesNum);
};

#ifdef _DEBUG // vs add this define in debug mode
#include <stdio.h>
// Q: why do we need traces ?
// A: traces are a nice and easy way to detect bugs without even debugging
// or to understand what happened in case we miss the bug in the first time
#define TRACE(msg, ...) printf(msg "\n", __VA_ARGS__);
// for convenient reasons we did the traces in stdout
// at general we would do this in the error stream like that
// #define TRACE(msg, ...) fprintf(stderr, msg "\n", __VA_ARGS__);

#else // we want nothing to be printed in release version
#define TRACE(msg, ...) printf(msg "\n", __VA_ARGS__);
#define TRACE(msg, ...) // do nothing
#endif

```

```

#include "Helper.h"

#include <string>
#include <iomanip>
#include <sstream>

using namespace std;

// recieves the type code of the message from socket (first byte)
// and returns the code. if no message found in the socket returns 0 (which means the client disconnected)
int Helper::getMessageTypeCode(SOCKET sc)
{
    char* s = getPartFromSocket(sc, 3);
    std::string msg(s);

    if (msg == "")
        return 0;

    int res = std::atoi(s);
    delete s;
    return res;
}

// send data to socket
// this is private function
void Helper::sendData(SOCKET sc, std::string message)
{
    const char* data = message.c_str();

    if (send(sc, data, message.size(), 0) == INVALID_SOCKET)
    {
        throw std::exception("Error while sending message to client");
    }
}

int Helper::getIntPartFromSocket(SOCKET sc, int bytesNum)
{
    char* s = getPartFromSocket(sc, bytesNum, 0);
    return atoi(s);
}

string Helper::getStringPartFromSocket(SOCKET sc, int bytesNum)
{
    char* s = getPartFromSocket(sc, bytesNum, 0);
    string res(s);
    return res;
}

// recieve data from socket according byteSize
// this is private function
char* Helper::getPartFromSocket(SOCKET sc, int bytesNum)
{
    return getPartFromSocket(sc, bytesNum, 0);
}

```

```

char* Helper::getPartFromSocket(SOCKET sc, int bytesNum, int flags)
{
    if (bytesNum == 0)
    {
        return "";
    }

    char* data = new char[bytesNum + 1];
    int res = recv(sc, data, bytesNum, flags);

    if (res == INVALID_SOCKET)
    {
        std::string s = "Error while recieving from socket: ";
        s += std::to_string(sc);
        throw std::exception(s.c_str());
    }

    data[bytesNum] = 0;
    return data;
}

string Helper::getPaddedNumber(int num, int digits)
{
    std::ostringstream ostr;
    ostr << std::setw(digits) << std::setfill('0') << num;
    return ostr.str();
}

#pragma once

#include <string>
#include <time.h>

class Question
{
private:
    std::string _question;
    std::string _answers[4];
    int _correctAnswerIndex;
    int _id;

public:
    Question(int id, std::string question, std::string correctAnswer, std::string answer2, std::string
answer3, std::string answer4)
    {

        _id = id;
        _question = question;

        _correctAnswerIndex = rand() % 4;
        _answers[_correctAnswerIndex] = correctAnswer;

        // rand the second answer index
        int firstIndex = rand() % 4;
        while (firstIndex == _correctAnswerIndex)

```

```

        {
            firstIndex = rand() % 4;
        }
        _answers[firstIndex] = answer2;

        // rand the third answer index
        int secondIndex = rand() % 4;
        while (secondIndex == _correctAnswerIndex || secondIndex == firstIndex)
        {
            secondIndex = rand() % 4;
        }
        _answers[secondIndex] = answer3;

        int thirdIndex = 6 - _correctAnswerIndex - firstIndex - secondIndex;
        _answers[thirdIndex] = answer4;
    }

    std::string getQuestion()
    {
        return _question;
    }

    std::string* getAnswers()
    {
        return _answers;
    }

    int getCorrectAnswerIndex()
    {
        return _correctAnswerIndex;
    }

    int getId()
    {
        return _id;
    }
};

#pragma once

#include <string>
#include <Windows.h>
#include "User.h"
using namespace std;

class RecievedMessage
{
private:
    SOCKET _sock;
    User* _user;

    int _messageCode;
    // string _messageContent;
    vector<string> _values;

```

```

public:
    RecievedMessage(SOCKET sock, int messageCode)
    {
        _sock = sock;
        _messageCode = messageCode;
        _user = nullptr;
    }

    RecievedMessage(SOCKET sock, int messageCode, vector<string> values) : RecievedMessage(sock,
messageCode)
    {
        _values = values;
    }

    SOCKET getSock() { return _sock; }
    User* getUser() { return _user; }
    void setUser(User* user) { _user = user; }
    int getMessageCode() { return _messageCode; }
    //string getMessageContent() { return _messageContent; }
    vector<string>& getValues()
    {
        return _values;
    }

};

#pragma once
#include <vector>
#include "User.h"
#include "Protocol.h"
#include "Helper.h"
#include <iomanip>
#include <mutex>
class User;
class Room
{
private:
    std::vector<User*> _users;
    User* _admin;
    int _maxUsers;
    int _questionTime;
    int _questionsNo;
    string _name;
    int _id;
    string getUsersAsString(vector<User*> usersList, User* excludeUser);
    void sendMessage(string message);
    void sendMessage(User* excludeUser, string message);
public:
    Room(int id, User* admin, string name, int maxUsers, int questionsNo, int quuestionTime);
    bool joinRoom(User* user);
    void leaveRoom(User* user);
    int closeRoom(User* user);
    vector<User*> getUsers();
    string getUsersListMessage();
    int getQuestionsNo();
    int getId();
    string getName();
};

```

```

define _CRT_SECURE_NO_WARNINGS
#include "Room.h"
#include <iostream>

using namespace std;

Room::Room(int id, User* admin, string name, int maxUsers, int questionsNo, int questionTime)
{
    _id = id;
    _name = name;
    _maxUsers = maxUsers;
    _questionsNo = questionsNo;
    _questionTime = questionTime;
    _admin = admin;

    _users.push_back(admin);
}

string Room::getUsersAsString(vector<User*> usersList, User* excludeUser)
{
    vector<User*>::iterator it;
    User* tmp = nullptr;
    string res = "";
    for (it = usersList.begin(); it != usersList.end(); it++)
    {
        tmp = *it;

        if (tmp != excludeUser)
            res += "[" + tmp->getUsername() + "]";
    }
    return res;
}

string Room::getUsersListMessage()
{
    User* tmp;
    vector<User*>::iterator it;
    string res = "";
    string currUsername;

    for (it = _users.begin(); it != _users.end(); it++)
    {
        tmp = *it;

        currUsername = tmp->getUsername();
        res += Helper::getPaddedNumber(currUsername.size(), 2) + currUsername;
    }

    string msg = SRV_MSG_USERS_IN_ROOM + Helper::getPaddedNumber(_users.size(), 1) + res;

    return msg;
}

```

```

void Room::sendMessage(string message)
{
    sendMessage(nullptr, message);
}

void Room::sendMessage(User* excludeUser, string message)
{
    vector<User*>::iterator it;
    User* tmp = nullptr;

    try
    {
        TRACE("SEND: Room sending message to %d users (%s): message = %s", _users.size(),
getUsersAsString(_users, excludeUser).c_str(), message.c_str());

        for (it = _users.begin(); it != _users.end(); it++)
        {
            tmp = *it;

            if (tmp != excludeUser)
            {
                tmp->send(message);
            }
        }

        catch (exception e)
        {
            // if send failed not throw the exception because it not sure that the error came from the same
            user thread
            // (becuase it is spread message)
            std::cout << "Sending Error (Room::sendMessage): cant send message to " << tmp->getUsername() <<
std::endl;
        }
    }

    string Room::getName()
    {
        return _name;
    }

    int Room::getId()
    {
        return _id;
    }

    bool Room::joinRoom(User* user)
    {
        // room is full
        if (_users.size() == _maxUsers)
        {
            user->send(SRV_MSG_JOIN_ROOM_FULL);
            return false;
        }
    }

```

```

        _users.push_back(user);

        // send success message to the new user
        string m = SRV_MSG_JOIN_ROOM_SUCCESS + Helper::getPaddedNumber(this->_questionsNo, 2) +
Helper::getPaddedNumber(this->_questionTime, 2);
        user->send(m);

        // send the users list message to all the users in the room
        sendMessage(getUsersListMessage());

        return true;
}

void Room::leaveRoom(User* user)
{
    bool flag = false;
    vector<User*>::iterator it;

    for (it = _users.begin(); it != _users.end() && !flag; it++)
    {
        if (*it == user)
            flag = true;
    }

    if (flag)
    {
        --it;
        _users.erase(it);
    }

    TRACE("User has left room: roomId=%d, roomName=%s, user=%s", _id, _name.c_str(), user-
>getName().c_str());

    // send success message to the user that left
    user->send(SRV_MSG_LEAVE_ROOM_SUCCESS);

    // send the users list message to all the users in the room
    sendMessage(getUsersListMessage());
}

// return the id of the room
// if failed return -1
int Room::closeRoom(User* user)
{
    // only admin can close the room
    if (_admin != user)
        return -1;

    // send closing message to all the users in the room
    sendMessage(SRV_MSG_CLOSE_ROOM);

    // set the room of the users (except the admin) to be null
    for (int i = 0; i < _users.size(); ++i)
    {
        if (_users[i] != _admin)
            _users[i]->clearRoom();
    }
}

```



```

    }

    _users.clear();

    return _id;
}

vector<User*> Room::getUsers()
{
    return _users;
}

int Room::getQuestionsNo()
{
    return _questionsNo;
}

#pragma once

#include <string>
#include "Room.h"
#include "Game.h"
#include "Helper.h"
#include <Windows.h>

using namespace std;

class Room;
class Game;

class User
{
private:
    string _username;
    Room* _currRoom;
    Game* _currGame;
    SOCKET _sock;

public:
    User(string username, SOCKET sock);
    void send(string message);

    string getUsername();
    SOCKET getSocket();
    Room* getRoom();
    Game* getGame();
    void setGame(Game*);
    void clearRoom();

    bool createRoom(int roomId, string roomName, int maxUsers, int questionsNo, int questionTime);
    bool joinRoom(Room* newRoom);
    void leaveRoom();
    int closeRoom();

    bool leaveGame();
};

```

```

#include "User.h"

User::User(string username, SOCKET sock) : _username(username), _sock(sock)
{
    _currRoom = nullptr;
    _currGame = nullptr;
}

void User::send(string message)
{
    Helper::sendData(_sock, message);
    TRACE("Message sent to user: %s, msg: %s", _username.c_str(), message.c_str());
}

string User::getUsername()
{
    return _username;
}

SOCKET User::getSocket()
{
    return _sock;
}

Room* User::getRoom()
{
    return _currRoom;
}

Game* User::getGame()
{
    return _currGame;
}

void User::setGame(Game* gm)
{
    _currRoom = nullptr;
    _currGame = gm;
}

bool User::createRoom(int roomId, string roomName, int maxUsers, int questionsNo, int questionTime)
{
    // cant create room if the user in room
    if (_currRoom)
    {
        this->send(SRV_MSG_CREATE_ROOM_FAILED);
        return false;
    }

    _currRoom = new Room(roomId, this, roomName, maxUsers, questionsNo, questionTime);
    this->send(SRV_MSG_CREATE_ROOM_SUCCESS);

    return true;
}

```

```

bool User::joinRoom(Room* newRoom)
{
    // cant join room if the user in room
    if (_currRoom)
        return false;

    bool res = newRoom->joinRoom(this);

    if (res)
        _currRoom = newRoom;

    return res;
}

bool User::leaveGame()
{
    if (!_currGame)
        return false;

    TRACE("User left game: userName=%s, gameId=%d", _username.c_str(), _currGame->getID());
    bool res = _currGame->leaveGame(this);
    this->_currGame = nullptr;

    return res;
}

void User::leaveRoom()
{
    // if not in room
    if (!_currRoom)
        return;

    _currRoom->leaveRoom(this);
    _currRoom = nullptr;
}

// return the id of the room
// if failed return -1
int User::closeRoom()
{
    // if not in room
    if (!_currRoom)
        return -1;

    int res = _currRoom->closeRoom(this);
    if (res != -1)
    {
        delete _currRoom;
        _currRoom = nullptr;
    }

    return res;
}

```

```

void User::clearRoom()
{
    _currRoom = nullptr;
}

#pragma once

#include <thread>
#include <map>
#include <mutex>
#include <WinSock2.h>
#include <Windows.h>
#include <condition_variable>
#include "User.h"
#include "DataBase.h"

#include <queue>
#include "RecievedMessage.h"

using namespace std;

class TriviaServer
{
public:
    TriviaServer();
    ~TriviaServer();
    void serve();

private:
    void bindAndListen();
    void accept();
    void clientHandler(SOCKET client_socket);
    void safeDeleteUser(RecievedMessage*);

    SOCKET _socket;

    User* handleSignin(RecievedMessage*);
    bool handleSignup(RecievedMessage* msg);
    void handleSignout(RecievedMessage* msg);

    void handleLeaveGame(RecievedMessage*);
    void handleStartGame(RecievedMessage* msg);
    void handlePlayerAnswer(RecievedMessage* msg);
    bool handleCreateRoom(RecievedMessage* );
    bool handleCloseRoom(RecievedMessage* );
    bool handleJoinRoom(RecievedMessage* );
    bool handleLeaveRoom(RecievedMessage*);
    void handleGetUsersInRoom(RecievedMessage* );
    void handleGetRooms(RecievedMessage* );

```

```

void handleGetBestScores(RecievedMessage*);
void handleGetPersonalStatus(RecievedMessage*);

map<SOCKET, User*> _connectedUsers;
DataBase _db;

static int _roomIdSequence ;
map<int, Room*> _roomsList;

User* getUserByName(string userName);
User* getUserBySocket(SOCKET client_socket);
Room* getRoomById(int roomId);
void handleRecievedMessages();
void addRecievedMessage(RecievedMessage* msg);
RecievedMessage* buildRecieveMessage(SOCKET userSock, int msgCode);

mutex _mtxRecievedMessages;
queue<RecievedMessage*> _queRcvMessages;

std::condition_variable cvQueue;
};

#include "TriviaServer.h"
#include "User.h"
#include "Protocol.h"
#include "Validator.h"
#include "Game.h"
#include <exception>
#include <iostream>
#include <string>

// using static const instead of macros
static const unsigned short PORT = 8820;
static const unsigned int IFACE = 0;

int TriviaServer::_roomIdSequence = 1;

TriviaServer::TriviaServer() : _db()
{
    WSADATA wsa_data = {};
    if (WSAStartup(MAKEWORD(2, 2), &wsa_data) != 0)
        throw std::exception("WSAStartup Failed");

    // notice that we step out to the global namespace
    // for the resolution of the function socket
    _socket = ::socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (_socket == INVALID_SOCKET)
        throw std::exception(__FUNCTION__ " - socket");
}

```

```

TriviaServer::~TriviaServer()
{
    TRACE(__FUNCTION__ " closing server");

    try
    {
        for (map<int, Room*>::iterator it = _roomsList.begin(); it != _roomsList.end(); it++)
        {
            delete it->second;
        }
        _roomsList.clear();

        // delete the users
        for (map<SOCKET, User*>::iterator it = _connectedUsers.begin(); it != _connectedUsers.end();
it++)
        {
            // free the memory of the user
            delete it->second;
        }
        _connectedUsers.clear();

        ::closesocket(_socket);
        WSACleanup();
    }
    catch (...) {}
}

void TriviaServer::serve()
{
    bindAndListen();

    // create new thread for handling message
    std::thread tr(&TriviaServer::handleRecievedMessages, this);
    tr.detach();

    while (true)
    {
        // the main thread is only accepting clients
        // and add then to the list of handlers
        TRACE("accepting client...");
        accept();
    }
}

// listen to connecting requests from clients
// accept them, and create thread for each client
void TriviaServer::bindAndListen()
{
    struct sockaddr_in sa = { 0 };
    sa.sin_port = htons(PORT);
    sa.sin_family = AF_INET;
    sa.sin_addr.s_addr = IFACE;
    // again stepping out to the global namespace
    if (::bind(_socket, (struct sockaddr*)&sa, sizeof(sa)) == SOCKET_ERROR)
        throw std::exception(__FUNCTION__ " - bind");
    TRACE("binded");
}

```

```

    if (::listen(_socket, SOMAXCONN) == SOCKET_ERROR)
        throw std::exception(__FUNCTION__ " - listen");
    TRACE("listening (Port = %d)...", PORT);
}

void TriviaServer::accept()
{
    SOCKET client_socket = ::accept(_socket, NULL, NULL);
    if (client_socket == INVALID_SOCKET)
        throw std::exception(__FUNCTION__);

    TRACE("Client accepted. Client socket = %d", client_socket);
    // create new thread for client and detach from it
    std::thread tr(&TriviaServer::clientHandler, this, client_socket);
    tr.detach();
}

User* TriviaServer::getUserByName(string userName)
{
    map<SOCKET, User*>::iterator userIter;;

    for (userIter = _connectedUsers.begin(); userIter != _connectedUsers.end(); userIter++)
    {
        if (userIter->second->getUsername() == userName)
            return userIter->second;
    }

    return nullptr;
}

User* TriviaServer::getUserBySocket(SOCKET client_socket)
{
    map<SOCKET, User*>::iterator userIter = _connectedUsers.find(client_socket);

    if (userIter == _connectedUsers.end())
    {
        return nullptr;
    }

    return userIter->second;
}

// Msg Code: 200
// if connected successfully return user object
// if not return null
User* TriviaServer::handleSignin(RecievedMessage* msg)
{
    SOCKET client_socket = msg->getSock();

    // get details
    vector<string> vals = msg->getValues();
    string userName = vals[0];
    string passWord = vals[1];

    // check if the username and password are correct
    bool res = _db.isUserAndPassMatch(userName, passWord);
    if (!res)

```

```

{
    Helper::sendData(client_socket, SRV_MSG_SIGN_IN_WRONG_DET);
    TRACE("SEND: User try sign in with wrong details: username = %s, socket = %d", userName.c_str(),
client_socket);
    return nullptr;
}

// if user already connected
User* currUser = getUserByName(userName);
if (currUser)
{
    Helper::sendData(client_socket, SRV_MSG_SIGN_IN_ALREADY_CONNECTED);
    TRACE("SEND: User already connected: username = %s, socket = %d", userName.c_str(), currUser-
>getSocket());
    return nullptr;
}

// success
// create user and add it to conntected users list
currUser = new User(userName, client_socket);
pair<SOCKET, User*> p(client_socket, currUser);

TRACE("Adding new user to connected users list: socket = %d, username = %s", client_socket,
userName.c_str());
_connectedUsers.insert(p);

// send success message
currUser->send(SRV_MSG_SIGN_IN_SUCCESS);
TRACE("SEND: User signed in successfully: username = %s, socket = %d", userName.c_str(), client_socket);

return (currUser);
}
// Msg Code: 203
bool TriviaServer::handleSignup(RecievedMessage* msg)
{
    SOCKET client_socket = msg->getSock();

    // get details
    vector<string> vals = msg->getValues();
    string userName = vals[0];

    string passWord = vals[1];
    string email = vals[2];

    // check if password is legal
    if (!Validator::isPasswordValid(passWord))
    {
        Helper::sendData(client_socket, SRV_MSG_SIGN_UP_PASS_ILLEGAL);
        TRACE("SEND: User signup failed. Invalid password: username = %s, socket = %d", userName.c_str(),
client_socket);
        return false;
    }
}

```



```

    if (!Validator::isUsernameValid(userName)) // check if username is legal
    {
        Helper::sendData(client_socket, SRV_MSG_SIGN_UP_USERNAME_ILLEGAL);
        TRACE("SEND: User signup failed. Invalid username: username = %s, socket = %d", userName.c_str(),
client_socket);
        return false;
    }

    if (_db.isUserExists(userName)) // check if user already exists
    {
        Helper::sendData(client_socket, SRV_MSG_SIGN_UP_USER_EXIST);
        TRACE("SEND: User signup failed. User already exist: username = %s, socket = %d", userName.c_str(),
client_socket);
        return false;
    }

    // try to insert the user to DB
    bool res = _db.addNewUser(userName, password, email);

    // if didnt success to add user in DB
    if (!res)
    {
        Helper::sendData(client_socket, SRV_MSG_SIGN_UP_FAILED);
        TRACE("SEND: User signup failed. Failed insert record to DB: username = %s, socket = %d",
userName.c_str(), client_socket);
        return false;
    }

    Helper::sendData(client_socket, SRV_MSG_SIGN_UP_SUCCESS);
    TRACE("SEND: User signed up successfully: username = %s, socket = %d", userName.c_str(), client_socket);

    return (true);
}

bool TriviaServer::handleCreateRoom(RecievedMessage* msg)
{
    User* currentUser = msg->getUser();

    if (!currentUser)
        return false;

    // get details
    vector<string> vals = msg->getValues();
    string roomName = vals[0];
    int playersNo = atoi(vals[1].c_str());
    int QuestionsNo = atoi(vals[2].c_str());
    int AnswerTime = atoi(vals[3].c_str());

    int roomId = _roomIdSequence++;
    bool res = currentUser->createRoom(roomId, roomName, playersNo, QuestionsNo, AnswerTime);

    if (res)
    {
        pair<int, Room*> p(roomId, currentUser->getRoom());
        _roomsList.insert(p);
    }
}

```

```

        TRACE("Room was created: roomId=%d, roomName=%s, adminName=%s, playersNo=%d,
QuestionsNo=%d, AnswerTime=%d",
            roomId, roomName.c_str(), currentUser-> getUsername().c_str(), playersNo,
QuestionsNo, AnswerTime);
    }

    return res;
}

Room* TriviaServer::getRoomById(int roomId)
{
    map<int, Room*>::iterator it = _roomsList.find(roomId);

    if (it == _roomsList.end())
    {
        return nullptr;
    }

    return it->second;
}

bool TriviaServer::handleJoinRoom(RecievedMessage* msg)
{
    User* currentUser = msg->getUser();
    if (!currentUser)
        return false;

    int roomId = atoi(msg->getValues()[0].c_str());
    Room* currRoom = getRoomById(roomId);

    // didnt find the room
    if (!currRoom)
    {
        Helper::sendData(msg->getSock(), SRV_MSG_JOIN_ROOM_NOT_EXIST);
        TRACE("User failed to join room. Room doesnt exist. username = %s, roomId = %d",
currentUser->getUsername().c_str(), roomId);
        return false;
    }

    bool res = currentUser->joinRoom(currRoom);

    if (res)
    {
        TRACE("User joined to room: roomId=%d, roomName=%s, userName=%s", roomId, currRoom-
>getName().c_str(), currentUser->getUsername().c_str());
    }
    return res;
}

void TriviaServer::handlePlayerAnswer(RecievedMessage* msg)
{
    vector<string> vals = msg->getValues();
    int ansNo = atoi(vals[0].c_str());
    int time = atoi(vals[1].c_str());

```

```

Game* g = msg->getUser()->getGame();

if (!g)
{
    TRACE("handlePlayerAnswer: User has no room");
    return;
}

// if game ended
if (!g->handleAnswerFromUser(msg->getUser(), ansNo, time))
{
    delete g;
    TRACE("handlePlayerAnswer: Game was ended and released from memory");
}
}

void TriviaServer::handleStartGame(RecievedMessage* msg)
{
    TRACE("Starting Game");

    Room* rm = msg->getUser()->getRoom();

    Game* gm = nullptr;
    try
    {
        gm = new Game(rm->getUsers(), rm->getQuestionsNo(), this->_db);
    }
    catch (...)
    {
        // send fail message to admin
        msg->getUser()->send(SRV_MSG_GAME_QUESTION + Helper::getPaddedNumber(0, 1));
        TRACE("Game failed to start");
        delete gm;
        return;
    }

    // remove the room and clear its users
    int roomId = rm->getId();
    _roomsList.erase(roomId);
    TRACE("Room was removed from rooms list: roomId=%d, roomName=%s", roomId, rm->getName().c_str());
    gm->sendFirstQuestion();
    TRACE("Game was started");
}

void TriviaServer::handleLeaveGame(RecievedMessage* msg)
{
    User* currUser = msg->getUser();
    Game* g = currUser->getGame();
    // if game ended release the memory
    if (!currUser->leaveGame())
    {
        delete g;
        TRACE("handleLeaveGame: Game was ended and released from memory");
    }
}

```

```

bool TriviaServer::handleCloseRoom(RecievedMessage* msg)
{
    User* currentUser = msg->getUser();
    Room* rm = currentUser->getRoom();
    if (!rm)
    {
        return false;
    }

    string roomName = rm->getName();

    int roomId = currentUser->closeRoom();

    if (roomId != -1)
    {
        _roomsList.erase(roomId);
        TRACE("Room was closed: roomId=%d, roomName=%s, closingUser=%s", roomId, roomName.c_str(),
currentUser->getUsername().c_str());
    }

    return roomId != -1;
}

bool TriviaServer::handleLeaveRoom(RecievedMessage* msg)
{
    User* currentUser = msg->getUser();
    if (!currentUser)
        return false;

    Room* rm = currentUser->getRoom();

    if (!rm)
        return false;

    int roomId = rm->getId();
    string roomName = rm->getName();

    currentUser->leaveRoom();
    TRACE("User left room: roomId=%d, roomName=%s, userName=%s", roomId, roomName.c_str(), currentUser-
>getUsername().c_str());

    return true;
}

void TriviaServer::handleGetUsersInRoom(RecievedMessage* msg)
{
    User* currentUser = msg->getUser();

    int roomId = atoi(msg->getValues()[0].c_str());

    Room* rm = getRoomById(roomId);

    TRACE("handleGetUsersInRoom: Get users in room id = %d", roomId);

    // if didnt find the room
    if (!rm)

```

```

{
    // sould send 0 users
    currentUser->send(SRV_MSG_USERS_IN_ROOM + Helper::getPaddedNumber(0, 1));
    return;
}

// room was found
string usersMsg = rm->getUsersListMessage();

// send message
currentUser->send(usersMsg);
}

void TriviaServer::handleGetRooms(RecievedMessage* msg)
{
    map<int, Room*>::iterator it;

    string roomId;
    string roomNameSize;
    string roomName;
    string content = SRV_MSG_ROOM_LIST + Helper::getPaddedNumber(_roomsList.size(), 4);
    for (it = _roomsList.begin(); it != _roomsList.end(); it++)
    {
        content += Helper::getPaddedNumber(it->second->getId(), 4);
        roomName = it->second->getName();
        content += Helper::getPaddedNumber(roomName.length(), 2) + roomName;
    }

    msg->getUser()->send(content);
}

void TriviaServer::handleGetBestScores(RecievedMessage* msg)
{
    vector<string> s = _db.getBestScores();

    string sRes = "";

    for (int i = 0; i < s.size(); i++)
    {
        sRes += s[i];
    }
    // creates the details in case there aren't 3 users
    for (int i = 0; i < 3 - s.size(); i++)
    {
        sRes += "00";
    }

    string res = std::to_string(SRV_MSG_REP_BEST_SCORES) + sRes;

    msg->getUser()->send(res);
}

```

```

void TriviaServer::handleGetPersonalStatus(RecievedMessage* msg)
{
    vector<string> s = _db.getPersonalStatus(msg->getUser()->getUsername());

    int games = atoi(s[0].c_str());
    int correct = atoi(s[1].c_str());
    int wrong = atoi(s[2].c_str());

    string sRes = Helper::getPaddedNumber(games, 4) + Helper::getPaddedNumber(correct, 6) +
    Helper::getPaddedNumber(wrong, 6) + s[3];

    string res = std::to_string(SRV_MSG_REP_PERSONAL_STATUS) + sRes;

    msg->getUser()->send(res);
}

void TriviaServer::addRecievedMessage(RecievedMessage* msg)
{
    unique_lock<mutex> lck(_mtxRecievedMessages);

    //RecievedMessage* msg = new RecievedMessage(sc, messageCode);
    _queRcvMessages.push(msg);
    lck.unlock();
    cvQueue.notify_all();
}

RecievedMessage* TriviaServer::buildRecieveMessage(SOCKET client_socket, int msgCode)
{
    RecievedMessage* msg = nullptr;
    vector<string> values;
    if (msgCode == CLT_MSG_SIGN_IN)
    {
        int userSize = Helper::getIntPartFromSocket(client_socket, 2);
        string userName = Helper::getStringPartFromSocket(client_socket, userSize);
        int passSize = Helper::getIntPartFromSocket(client_socket, 2);
        string passWord = Helper::getStringPartFromSocket(client_socket, passSize);
        values.push_back(userName);
        values.push_back(passWord);
    }

    else if (msgCode == CLT_MSG_SIGN_UP)
    {
        int userSize = Helper::getIntPartFromSocket(client_socket, 2);
        string userName = Helper::getStringPartFromSocket(client_socket, userSize);
        int passSize = Helper::getIntPartFromSocket(client_socket, 2);
        string passWord = Helper::getStringPartFromSocket(client_socket, passSize);
        int emailSize = Helper::getIntPartFromSocket(client_socket, 2);
        string email = Helper::getStringPartFromSocket(client_socket, emailSize);
        values.push_back(userName);
        values.push_back(passWord);
        values.push_back(email);
    }
}

```

```

else if (msgCode == CLT_MSG_CREATE_ROOM)
{
    int roomNameSize = Helper::getIntPartFromSocket(client_socket, 2);
    string roomName = Helper::getStringPartFromSocket(client_socket, roomNameSize);
    string playersNo = Helper::getStringPartFromSocket(client_socket, 1);
    string QuestionsNo = Helper::getStringPartFromSocket(client_socket, 2);
    string AnswerTime = Helper::getStringPartFromSocket(client_socket, 2);
    values.push_back(roomName);
    values.push_back(playersNo);
    values.push_back(QuestionsNo);
    values.push_back(AnswerTime);
}
else if (msgCode == CLT_MSG_CLOSE_ROOM || msgCode == CLT_MSG_LEAVE_ROOM || msgCode == CLT_END_CONNECTION
|| msgCode == CLT_MSG_GET_ROOMS)
{
    // do nothing becuase there arent another data in the recieved message (according protocol)
}
else if (msgCode == CLT_MSG_GET_USERS_IN_ROOM || msgCode == CLT_MSG_JOIN_ROOM)
{
    string roomId = Helper::getStringPartFromSocket(client_socket, 4);
    values.push_back(roomId);
}
else if (msgCode == CLT_MSG_ANSWER)
{
    string answerNo = Helper::getStringPartFromSocket(client_socket, 1);
    string answerTime = Helper::getStringPartFromSocket(client_socket, 2);

    values.push_back(answerNo);
    values.push_back(answerTime);
}

msg = new RecievedMessage(client_socket, msgCode, values);
return msg;
}

void TriviaServer::clientHandler(SOCKET client_socket)
{
    User* currentUser = nullptr;
    RecievedMessage* currRcvMsg = nullptr;

    try
    {
        // get the message code
        int msgCode = Helper::getMessageTypeCode(client_socket);

        // loop will ended when there is an exception or the client sent message
        // which means to terminate the connction
        // the excpetion might occure also when the client has disconnected
        while (msgCode != CLT_END_CONNECTION && msgCode != 0)

            //getMessageContent(msgCode);
            currRcvMsg = buildRecieveMessage(client_socket, msgCode);
            addRecievedMessage(currRcvMsg);
            msgCode = Helper::getMessageTypeCode(client_socket);
    }
}

```

```

currRcvMsg = buildRecieveMessage(client_socket, CLT_END_CONNECTION);
addRecievedMessage(currRcvMsg);

}
catch (const std::exception& e)
{
    std::cout << "Exception was catch in function clientHandler. socket=" << client_socket << ", what="
<< e.what() << std::endl;
    currRcvMsg = buildRecieveMessage(client_socket, CLT_END_CONNECTION);
    addRecievedMessage(currRcvMsg);
}
}

void TriviaServer::handleRecievedMessages()
{
    RecievedMessage* currRcvMessage = nullptr;
    int msgCode;
    User* currUser = nullptr;
    SOCKET client_socket = 0;
    while (true)
    {
//        while (!_queRcvMessages.empty())
        {
            try
            {
                unique_lock<mutex> lck(_mtxRecievedMessages);

                if (_queRcvMessages.empty())
                    cvQueue.wait(lck);

                // in case the queue is empty...
                if (_queRcvMessages.empty())
                    continue;

                currRcvMessage = _queRcvMessages.front();
                _queRcvMessages.pop();
                lck.unlock();

                client_socket = currRcvMessage->getSock();
                currUser = getUserBySocket(client_socket);
                currRcvMessage->setUser(currUser);
                msgCode = currRcvMessage->getMessageCode();

                TRACE("-----");
                TRACE("handleRecievedMessages: msgCode = %d, client_socket: %d", msgCode,
client_socket);

                if (msgCode == CLT_MSG_SIGN_IN) // Sign in
                {
                    handleSignin(currRcvMessage);
                }
                else if (msgCode == CLT_MSG_SIGN_UP) // Sign up
                {
                    handleSignup(currRcvMessage);
                }
                else if (msgCode == CLT_MSG_SIGN_OUT)

```



```

{
    handleSignout(currRcvMessage);
}
else if (msgCode == CLT_MSG_CREATE_ROOM)
{
    handleCreateRoom(currRcvMessage);
}
else if (msgCode == CLT_MSG_CLOSE_ROOM)
{
    handleCloseRoom(currRcvMessage);
}
else if (msgCode == CLT_MSG_JOIN_ROOM)
{
    handleJoinRoom(currRcvMessage);
}
else if (msgCode == CLT_MSG_LEAVE_ROOM)
{
    handleLeaveRoom(currRcvMessage);
}
else if (msgCode == CLT_MSG_GET_ROOMS)
{
    handleGetRooms(currRcvMessage);
}
else if (msgCode == CLT_MSG_GET_USERS_IN_ROOM)
{
    handleGetUsersInRoom(currRcvMessage);
}
else if (msgCode == CLT_MSG_START_GAME)
{
    handleStartGame(currRcvMessage);
}
else if (msgCode == CLT_MSG_ANSWER)
{
    handlePlayerAnswer(currRcvMessage);
}
else if (msgCode == CLT_MSG_LEAVE_GAME)
{
    handleLeaveGame(currRcvMessage);
}
else if (msgCode == CLT_END_CONNECTION)
{
    // close the connection with the user
    safeDeleteUser(currRcvMessage);
}
else if (msgCode == CLT_MSG_REP_BEST_SCORES)
{
    handleGetBestScores(currRcvMessage);
}
else if (msgCode == CLT_MSG_REP_PERSONAL_STATUS)
{
    handleGetPersonalStatus(currRcvMessage);
}
}

```

```

        else // unknown message
        {
            // close the connection with the user
            safeDeleteUser(currRcvMessage);
        }
    }
    catch (const std::exception& e)
    {
        std::cout << "Exception was catch in function handleRecievedMessages. socket=" <<
client_socket << ", what=" << e.what() << std::endl;
        safeDeleteUser(currRcvMessage);
    }

    TRACE("-----");
}
}

void TriviaServer::handleSignout(RecievedMessage* msg)
{
    User* currUser = msg->getUser();

    if (!currUser)
        return;
    SOCKET id = msg->getSock();

    // remove user from the queue
    TRACE("Erase user from connected users list: username = %s, socket = %d", currUser-
>getUsername().c_str(), id)
    _connectedUsers.erase(id);

    // in case the user is admin of room or in room
    handleCloseRoom(msg);
    handleLeaveRoom(msg);
    handleLeaveGame(msg);
    TRACE("User had signedout: username = %s, socket = %d", currUser->getUsername().c_str(), id)
}

// remove the user from connected users list
void TriviaServer::safeDeleteUser(RecievedMessage* msg)
{
    try
    {
        SOCKET id = msg->getSock();

        handleSignout(msg);

        TRACE("Closing socket = %d", id)
        closesocket(id);
        TRACE("User has disconnected: socket = %d", id);
    }
    catch (...) {}
}

```

```

#pragma comment (lib, "ws2_32.lib")
#include "TriviaServer.h"
#include <iostream>
#include <sstream>

#include "Helper.h"

void main()
{
    try
    {
        TRACE("Starting...");
        srand(time(NULL));

        TriviaServer md_server;
        md_server.serve();
    }
    catch (const std::exception& e)
    {
        std::cout << "Exception was catch in function: " << e.what() << std::endl;
    }
    catch (...)
    {
        std::cout << "Unknown exception in main !" << std::endl;
    }
}

```