**אפליקציית טריוויה**

**קורס עקרונות מתקדמים**

**מסמך עיצוב**



* תיאור כללי

פרויקט זה הוא משחק טריוויה בתצורת קליינט-סרבר.

הסרבר הוא שרת המשחק ויכתב ב-++C. ה-DB מבוסס על sqlite3. השרת ירוץ על מערכת הפעלה windows.

הקליינט נכתב ב-C# והוא גרפי ומיועד למערכת הפעלה windows.

התקשורת מבוססת על פרוטוקול אפליקטיבי ייחודי המתבסס על TCP.

* תיאור כללי של המשחק

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 ססמא ופרטים 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 אפשרויות. 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 ססמא ופרטים נוספים.

* תיאור כללי של השרת

השרת מבוסס TCP וקיים פרוטוקול אפליקטיבי ייחודי המוגדר בהמשך המסמך.

השרת הוא 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 השרת.

* תרשים זרימה כללי

מצורף קובץ נפרד המכיל את תרשים הזרימה הכללי של המערכת.

מטרת התרשים היא לתת תמונת-על לגבי המערכת ורצף ההתרחשויות בה.

* מחלקות – תיאור כללי

UML

מצורף קובץ נפרד המתאר את ה-UML

CLASSES

**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 הססמא 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 ססמא חוקית , האם יוזר חוקי.

**Helper**

תיאור כללי: מחלקה סטטית.

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

* מחלקות - תיאור מפורט
* **Question**

פונקציות עיקריות:

**בנאי –**

*פרמטרים*:

* + int id – מזהה שאלה
  + string question - שאלה
  + string correctAnswer – תשובה נכונה
  + string answer2 – תשובה שגויה
  + string answer3 – תשובה שגויה
  + string answer4 – תשובה שגויה

*ערך החזר*: אין

*פונקציונליות*: הבנאי מאתחל את המאפיינים של המחלקה. הסדר של 4 התשובות מוגרל באופן אקראי. יש לשמור את המיקום של התשובה הנכונה.

* **DataBase**

פונקציות עיקריות:

**בנאי**

*פרמטרים*: אין

*ערך החזר*: אין

*פונקציונליות*: פתיחת ה-DB. במידה ולא הצליח יש לזרוק Exception.

**D'tor**

*פרמטרים*: אין

*ערך החזר*: אין

*פונקציונליות*: סגירת ה-DB.

**isUserExists**

*פרמטרים*:

* + String username- שם היוזר

*ערך החזר*: בוליאני.

*פונקציונליות*: בדיקה האם קיים יוזר ב-DB.

**addNewUser**

*פרמטרים*:

* + String username- שם היוזר
  + String password – ססמא
  + String email - אימייל

*ערך החזר*: בוליאני. הצליח להוסיף או לא הצליח.

*פונקציונליות*: הכנסת רשומה חדשה לטבלת יוזרים

**isUserAndPassMatch**

*פרמטרים*:

* + String username- שם היוזר
  + String password – ססמא

*ערך החזר*: בוליאני. האם קיים יוזר עם ססמא תואמת.

*פונקציונליות*: בדיקה האם קיים יוזר עם ססמא תואמת במערכת.

**initQuestions**

*פרמטרים*:

* + Int questionsNo – מספר שאלות שיש לשלוף

*ערך החזר*: vector<Question\*> - רשימה של שאלות שנשלפו.

*פונקציונליות*: שולף כמות שאלות על פי הפרמטר בצורה רנדומלית (כלומר השאלות נבחרות באופן רנדומלי). מאתחל וקטור של אובייקטים מסוג מצביע לשאלה.

**insertNewGame**

*פרמטרים*: אין

*ערך החזר*: מספר שלם – המזהה של החדר שנוצר

*פונקציונליות*: יוצר רשומה חדשה בטבלת משחקים, כך שסטטוס משחק הוא 0 ושעת התחלה היא NOW.

**updateGameStatus**

*פרמטרים*: מזהה משחק

*ערך החזר*: בוליאני – הצליח לעדכן או לא.

*פונקציונליות*: מעדכן את סטטוס המשחק ל-1 ואת שעת סיום המשחק ל-NOW.

**addAnswerToPlayer**

*פרמטרים*:

* + int gameId- מזהה משחק
  + String username
  + Int questionId – מזהה השאלה
  + String answer – התשובה של היוזר
  + Bool isCorrect – האם התשובה נכונה או לא.
  + Int answerTime – זמן מענה לשאלה (בשניות).

*ערך החזר*: בוליאני. האם הצליח להוסיף או לא.

*פונקציונליות*: מוסיף רשומה חדשה בטבלת תשובות לשחקן.

**את שאר הפונקציות יש לממש על פי שיקול דעתכם**

* **Validator**

פונקציות עיקריות:

**isPasswordValid <static>**

*פרמטרים*:

* + String password

*ערך החזר*: בוליאני, תקין או לא.

*פונקציונליות*: בודק האם ססמא חוקית. אלו החוקים:

* *חייבת להכיל לפחות 4 תווים*
* *לא יכולה להכיל רווחים*
* *חייבת להכיל לפחות ספרה אחת*
* *חייבת להכיל לפחות אות גדולה אחת (Upper case)*
* *חייבת להכיל לפחות אות קטנה אחת (Lower case)*

**isUsernameValid <static>**

*פרמטרים*:

* + String username

*ערך החזר*: בוליאני, תקין או לא.

*פונקציונליות*: בודק האם שם יוזר חוקי. אלו החוקים:

* *חייב להתחיל באות*
* *לא יכול להכיל רווחים*
* *לא יכול להיות ריק*
* **User**

פונקציות עיקריות:

**בנאי**

*פרמטרים*:

* + String username
  + SOCKET sock

*ערך החזר*: אין

*פונקציונליות*: מאתחל את המאפיינים על פי הפרמטרים. חדר ומשחק יפנו ל-nullptr.

**send**

*פרמטרים*:

* + String message – ההודעה שנשלחת

*ערך החזר*: אין

*פונקציונליות*: שולח את ההודעה ליוזר. ניעזרת ב-Helper::sendData

**setGame**

*פרמטרים*:

* + Game\* gm– המשחק

*ערך החזר*: אין

*פונקציונליות*: מפנה את ההצבעה לחדר ל-nullptr ואת ההצבעה של המשחק לפרמטר.

**clearGame**

*פרמטרים*: אין

*ערך החזר*: אין

*פונקציונליות*: מנתק את הקשר בין השחקן לחדר על ידי כך שההצבעה לאובייקט משחק מפנה ל-nullptr.

**createRoom**

*פרמטרים*:

* + Int roomId – מזהה חדר
  + String roomName – שם החדר
  + int maxUsers – כמות שחקנים מקסימלית
  + int questionsNo – מספר שאלות
  + int questionTime – זמן מענה מקסימלי לשאלה (בשניות)

*ערך החזר*: בוליאני. הצליח ליצור חדר או לא.

*פונקציונליות*: בדיקה האם היוזר משוייך לחדר.

אם כן, נשלחת לו הודעת כשלון על יצירת חדר (עפ"י הפרוטוקול - קוד הודעה 114).

אם לא, נוצר אובייקט חדר חדש והוא משויך ליוזר. נשלחת הודעת הצלחה על יצירת חדר (עפ"י הפרוטוקול - קוד הודעה 114).

**joinRoom**

*פרמטרים*:

* + Room\* newRoom– חדר

*ערך החזר*: בוליאני. הצליח להצטרף או לא.

*פונקציונליות*: אם ליוזר משויך חדר מחזיר FALSE. אם לא, מנסה לשייך את היוזר לחדר בעזרת פונקציה מתאימה של החדר (joinRoom).

**leaveRoom**

*פרמטרים*: אין

*ערך החזר*: אין

*פונקציונליות*: במידה והיוזר משויך לחדר מפעיל על החדר פונקציה מתאימה (leaveRoom). לאחר מכן יש לשנות את המאפיין שמצביע לחדר ל-nullptr.

**closeRoom**

*פרמטרים*: אין

*ערך החזר*: int - מספר החדר במידה והחדר נסגר בהצלחה, אם לא הצליח לסגור מחזיר 1-.

*פונקציונליות*: אם היוזר לא נמצא בחדר מחזיר 1-. אם נמצא מפעיל על החדר את הפונקציה המתאימה (closeRoom). אם הצליח מוחק את החדר מהזיכרון ומשנה את המאפיין של החדר כך שיצביע ל-nullptr.

**leaveGame**

*פרמטרים*: אין

*ערך החזר*: בוליאני. TRUE אם המשחק עדין פעיל. FALSE אם המשחק נגמר.

*פונקציונליות*: במידה והיוזר משויך למשחק מפעיל על המשחק פונקציה מתאימה (leaveGame). לאחר מכן יש לשנות את המאפיין שמצביע למשחק ל-nullptr.

**clearRoom**

*פרמטרים*: אין

*ערך החזר*: אין

*פונקציונליות*: משנה את המאפיין של החדר כך שיצביע ל-nullptr.

* **Room**

מאפיינים עיקריים:

* + std::vector<User\*> \_users – היוזרים שמחוברים לחדר.
  + User\* \_admin – הפניה ליוזר שהוא האדמין (זה שפתח את החדר).
  + \_maxUsers – כמות יוזרים מקסמילית בחדר
  + \_questionTime – זמן מענה מקסימילי לשאלה (כשהחדר יהפוך למשחק)
  + \_questionNo – מספר שאלות מקסימלי במשחק (כשהחדר יהפוך למשחק)
  + \_name – שם החדר
  + \_id – מזהה החדר

פונקציות עיקריות:

**בנאי**

*פרמטרים*:

* + int id
  + User\* admin
  + string name
  + int maxUsers
  + int questionsNo
  + int questionTime

*ערך החזר*: אין

*פונקציונליות*: מאתחל את המאפיינים. יש לשים לב שיש לאתחל גם את שדה admin וגם להכניס את ה-admin כיוזר ברשימת היוזרים שמחוברים לחדר.

**getUsersAsString**

*פרמטרים*:

* + vector<User\*> usersList, - רשימת יוזרים
  + User\* excludeUser – יוזר כלשהו

*ערך החזר*: מחרוזת

*פונקציונליות*: הפונקציה נועדה לצרכי DEBUGING, ולכן אין חובה להשתמש בה.

הפונקציה מחזירה מחרוזת המייצגת את שמות היוזרים של הרשימה שהתקבלה כפרמטר, מלבד שם היוזר שמוצבע על ידי excludeUser.

**getUsersListMessage**

*פרמטרים*: אין

*ערך החזר*: מחרוזת המייצגת את הודעת "שליחת רשימת יוזרים בחדר" (108)

*פונקציונליות*: הפונקציה עוברת על רשימת היוזרים המחוברים לחדר ובונה הודעה מתאימה על פי הפרוטוקול האפליקטיבי ומחזירה מחרוזת מתאימה.

**sendMessage**

*פרמטרים*:

* + User\* excludeUser – יוזר כלשהו
  + string message – הודעה לשליחה

*ערך החזר*: אין

*פונקציונליות*: הפונקציה עוברת על כל היוזרים ושולחת לכולם, פרט ל-excludeUser, את ההודעה שהתקבלה כפרמטר. הפונקציה שולחת בעזרת הפונקציה send של המחלקה User.

אם ארעה חריגה במהלך הפונקציה יש לתפוס אותה ולא לגלגל אותה הלאה.

**sendMessage**

*פרמטרים*:

* + string message – הודעה לשליחה

*ערך החזר*: אין

*פונקציונליות*: מפעילה את הפונקציה sendMessage ומעבירה null לפרמטר excludeUser

**joinRoom**

*פרמטרים*:

* + User\* user – יוזר שמנסה להצטרף לחדר

*ערך החזר*: בוליאני. TRUE אם השחקן הצטרף, FALSE אם לא הצליח (כי לא היה מקום בחדר)

*פונקציונליות*: בדיקה האם החדר מלא. אם כן, שליחת כשלון ליוזר שמנסה להצטרף לחדר. אם לא מלא, יש להוסיף את היוזר לרשימת היוזרים של החדר ולשלוח לו הודעת הצלחה על הצטרפות. לאחר מכן יש לשלוח לכל היוזרים המחוברים (כולל זה שהתחבר כרגע) את רשימת היוזרים המחוברים המעודכנת (בעזרת הפונקציה sendMessage ו- getUsersListMessage)

.

**leaveRoom**

*פרמטרים*:

* + User\* user – השחקן שמבקש לעזוב

*ערך החזר*: אין

*פונקציונליות*: חיפוש היוזר שמבקש לעזוב ברשימת היוזרים שמחוברים לחדר. אם מצאנו, יש להסיר אותו מהרשימה.

הודעת הצלחת עזיבה ליוזר שעזב ושליחה לכל היוזרים המחוברים לחדר (בלי זה שהתנתק) את הרשימה המעודכנת של היוזרים המחוברים כרגע.

**closeRoom**

*פרמטרים*:

* + User\* user – השחקן שמבקש לסגור את החדר

*ערך החזר*: מספר. אם הצליח לסגור את החדר מחזיר את ה-ID שלו, אם לא מחזיר 1-.

*פונקציונליות*: בדיקה האם היוזר שמבקש לסגור הוא האדמין. אם לא, יש להחזיר 1-.

יש לשלוח הודעת סגירה לכל היוזרים שמחוברים לחדר.

עבור כל היוזרים שמחוברים , פרט לאדמין, יש להפעיל עליהם את הפונקציה clearRoom.

יש להחזיר את מזהה החדר.

* **Game**

מאפיינים עיקריים:

* + std::map<std::string, int> \_results – מיפוי של שם יוזר והניקוד שלו במשחק.
  + \_currentTurnAnswers – מספר השחקנים שהחזירו תשובה בסיבוב כלשהו.

פונקציות עיקריות:

**בנאי**

*פרמטרים*:

* + const vector<User\*>& players
  + int questionsNo
  + DataBase& db

*ערך החזר*: אין

*פונקציונליות*: מפעיל את הבנאי העתקה של DataBase עם הפרמטר שנשלח.

מנסה להכניס את המשחק ל-DB בעזרת הפונקציה DataBase::insertNewGame. האם המשחק לא הצליח להתווסף ל-DB נזרק Exception.

אם נכנס, מאתחל את השאלות בעזרת DataBase::initQuestions.

מאתחל את רשימת השחקנים (\_players), מאתחל את רשימת הניקוד של השחקנים (\_results), מגדיר עבור כל שחקן שהמשחק הנוכחי הוא המשחק שלו (User::setGame)

**D'tor**

*פרמטרים*: אין

*ערך החזר*: אין

*פונקציונליות*: מוחק מהזכרון את השאלות.

מרוקן את הוקטור של השאלות ואת הוקטור של השחקנים.

**sendQuestionToAllUsers**

*פרמטרים*: אין

*ערך החזר*: אין

*פונקציונליות*: בונה הודעה 118 (שליחת שאלה) על פי השאלה הנוכחית שקיימת ברשימת השאלות.

מאפס את השדה \_currentTurnAnswers ל-0.

עובר על כל השחקנים במשחק ושולח להם את ההודעה (User::send). במידה והיה Exception בעקבות השליחה יש לתפוס אותה ולהמשיך לשלוח לשאר השחקנים.

**handleFinishGame**

*פרמטרים*: אין

*ערך החזר*: אין

*פונקציונליות*: מעדכן את הרשומה של המשחק ב-DB (DataBase::updateGameStatus).

עבור השחקנים שנותרו במשחק שולח להם הודעת סיום משחק (121) ומאתחל את שדה המשחק שלהם ל-nullptr. במידה והיה Exception בעקבות השליחה יש לתפוס אותה ולהמשיך לשלוח לשאר השחקנים.

**sendFirstQuestion**

*פרמטרים*: אין

*ערך החזר*: אין

*פונקציונליות*: מפעיל את sendQuestionToAllUsers

**handleNextTurn**

*פרמטרים*: אין

*ערך החזר*: בוליאני. אם המשחק עדין פעיל מחזיר TRUE אם המשחק נגמר מחזיר FALSE.

*פונקציונליות*: בדיקה קיימים שחקנים בחדר. אם לא, קורא לפונקציה המטפלת בסיום משחק (handleFinishGame).

אם פעיל, בדיקה האם כל השחקנים ענו על התשובה בסיבוב הנוכחי. אם כן, בודקים האם זה היה הסיבוב האחרון (כלומר, השאלה האחרונה). אם זה הסיבוב האחרון יש לטפל בסיום משחק. אם זה לא הסיבוב האחרון יש לקדם את מונה השאלות ב-1 ולשלוח שאלה לכל היוזרים (sendQuestionToAllUsers).

אם עדין לא כל השחקנים ענו, לא עושים כלום.

**handleAnswerFromUser**

*פרמטרים*:

* + User\* user – שחקן
  + int answerNo – אינדקס התשובה שהשחקן בחר
  + int time – זמן שלקח לענות על השאלה (בשניות)

*ערך החזר*: בוליאני. True אם המשחק לא נגמר. False – אם המשחק נגמר.

*פונקציונליות*: מקדם את \_currentTurnAnswers ב-1. בודק האם השחקן ענה תשובה נכונה או לא. אם ענה נכון, מוסיף 1 לניקוד שלו (בכניסה המתאימה ב- \_results).

מוסיף רשומה ב-DB לגבי התשובה של השחקן (DataBase::addAnswerToPlayer). במידה והתשובה היא 5 (שחקן לא ענה בזמן על השאלה) הטקסט של התשובה שיכנס ל-DB צריך להיות ריק.

שולח הודעה ליוזר לגבי נכונות התשובה (הודעה 120).

מפעיל את handleNextTurn.

.

**leaveGame**

*פרמטרים*:

* + User\* currUser – השחקן שמבקש לעזוב

*ערך החזר*: בוליאני. TRUE אם המשחק עדין פעיל. FALSE אם המשחק נגמר.

*פונקציונליות*: מחפש את היוזר ברשימת השחקנים ואם נמצא מוציא אותו מהרשימה ומפעיל את הפונקציה handleNextTurn.

* **TriviaServer**

מאפיינים עיקריים:

* + std::map<SOCKET, User\*> \_connectedUsers– מייצג את רשימת היוזרים המחוברים. ממפה בין סוקט ליוזר.
  + std::map<int, Room\*> \_roomsList– מייצג את רשימת החדרים הפעילים. ממפה בין מזהה חדר לחדר.
  + queue<RecievedMessage\*> \_queRcvMessages – תור של הודעות שהגיעו מקליינטים ויש לטפל בהן.
  + static int \_roomIdSequence- מונה של מזהה החדר. כל פעם שנוצר חדר מקדם ב-1 את המונה.

פונקציות עיקריות:

**בנאי**

*פרמטרים*: אין

*ערך החזר*: אין

*פונקציונליות*: מפעיל את הבנאי של DataBase.

מייצר סוקט חדש. במידה ולא הצליח ליצור זורק Exception.

**D'tor**

*פרמטרים*: אין

*ערך החזר*: אין

*פונקציונליות*: מוחק מהזכרון את רשימת החדרים הקיימים

מוחק מהזכרון את רשימת היוזרים המחוברים.

סוגר את הסוקט.

**serve**

*פרמטרים*: אין

*ערך החזר*: אין

*פונקציונליות*: מפעיל את הפוקנציה bindAndListen. מייצר ת'רד חדש עבור טיפול בהודעות (clientHandler).

מתחיל לקבל לקוחות בלולאה אינסופית (קורא לפונקציה accept בתוך הלולאה)

**bindAndListen**

*פרמטרים*: אין

*ערך החזר*: אין

*פונקציונליות*: מבצע bind לסוקט ומתחיל להאזין (listen). אם נכשל ב-bind או ב-listen זורק exception.

**accept**

*פרמטרים*: אין

*ערך החזר*: אין

*פונקציונליות*: מקבל קליינט שהתחבר לשרת. יוצר עבורו ת'רד נפרד (מעביר כפרמטר את הסוקט של הקליינט).

**clientHandler**

*פרמטרים*:

* + SOCKET client\_socket– הסוקט של הקליינט שעבורו נוצר הת'רד

*ערך החזר*: אין

*פונקציונליות*: מקבל את קוד ההודעה שנשלח (הפונקציה Helper::getMessageTypeCode)

כל עוד קוד ההודעה שונה מסיום התקשורת או מ-0

בנה הודעה בעזרת buildRecieveMessage והוסף אותה לתור (בעזרת addRecieveMessage).

קבל את הקוד של ההודעה הבאה שנשלחה (Helper::getMessageTypeCode)

צור הודעת "סיום התקשרות" והוסף אותה לתור ההודעות.

במידה ואירעה חריגה כלשהי במהלך הפונקציה יש לתפוס אותה וליצור בה הודעת "סיום התקשרות" ולהוסיף אותה לתור ההודעות" ולתת לת'רד להסתיים.

**getRoomById**

*פרמטרים*:

* + int roomId – מזהה החדר

*ערך החזר*: Room\*, החדר או nullptr

*פונקציונליות*: מחפשת ומחזירה את החדר ברשימת החדרים.

**getUserByName**

*פרמטרים*:

* + string username

*ערך החזר*: User\*, היוזר או nullptr

*פונקציונליות*: מחפשת ומחזירה את היוזר ברשימת היוזרים המחוברים.

**getUserBySocket**

*פרמטרים*:

* + SOCKET client\_socket

*ערך החזר:* User\*, היוזר או nullptr

*פונקציונליות:* מחפשת ומחזירה את היוזר ברשימת היוזרים המחוברים.

**handleRecievedMessages**

*פרמטרים*: אין

*ערך החזר*: אין

*פונקציונליות*: הפונקציה מטפלת בהודעות שקיימות בתור ההודעות.

יש להשתמש ב-CV כדי לדעת מתי התור אינו ריק ומתי יש לטפל בהודעות (ה-CV יתעורר בת'רד של לקוח, כשהודעה מגיעה ונכנסת לתור).

כשיש הודעה בתור – ישלהוציא את ההודעה הראשונה ולאחר מכן לשחרר את המנעול על התור.

יש לאתחל את שדה היוזר של ההודעה לפי היוזר שחזר מהפונקציה getUserBySocket (כלומר מחפשים את היוזר ברשימת יוזרים על פי הסוקט של הקליינט שהכניס את ההודעה לתור).

בהתאם לקוד ההודעה קוראים לפונקצית הטיפול המתאימה.

במידה וקוד ההודעה הוא "סיום ההתקשרות" יש להפעיל את safeDeleteUser.

במידה וקוד ההודעה אינו ידוע יש להפעיל את safeDeleteUser.

במידה ואירעה חריגה כלשהי יש להפעיל את safeDeleteUser.

**safeDeleteUesr**

*פרמטרים*:

* + RecievedMessage\* msg – ההודעה שהתקבלה מהלקוח

*ערך החזר*: אין

*פונקציונליות*: הפונקציה מנתקת יוזר בצורה בטוחה.

יש להשיג את הסוקט של היוזר. לאחר מכן להפעיל את הפונקציה של יציאה מהמערכת (handleSignout). לאחר מכן יש לסגור את הסוקט של הקליינט.

במידה ואירעה חריגה במהלך הפונקציה יש לתפוס אותה ולא להעביר אותה הלאה.

**handleSignin (הודעה 200 בפרוטוקול)**

*פרמטרים*:

* + RecievedMessage\* msg – ההודעה שהתקבלה מהלקוח

*ערך החזר*: User\*, היוזר שנוצר או nullptr אם לא הצליח להתחבר

*פונקציונליות*: להודעה יש שני ערכים בהתאם לפרוטוקול – יוזר + ססמא. בודקים מול ה-DB האם היוזר והססמא תואמים (DataBase::isUserAndPassMatch). אם לא, שולחים הודעת כשלון מתאימה לקליינט. אם תואם, בודקים האם היוזר כבר מחובר (האם נמצא ברשימת היוזרים – getUserByName). אם היוזר מחובר שולחים הודעת כשלון מתאימה לקליינט.

אם לא מחובר, יוצרים יוזר חדש ומוסיפים אותו לרשימת היוזרים המחוברים. שולחים ליוזר הודעת התחברות מוצלחת.

**handleSignout(הודעה 201 בפרוטוקול)**

*פרמטרים*:

* + RecievedMessage\* msg – ההודעה שהתקבלה מהלקוח

*ערך החזר*: אין

*פונקציונליות*: בודקת האם להודעה משויך יוזר. אם משויך מוחקת מרשימת היוזרים המחוברים על פי הסוקט של היוזר.

מטפלת בסגירת חדר עבור היוזר (handleCloseRoom), מטפלת בעזיבת חדר (handleLeaveRoom), מטפלת בעזיבת משחק (handleLeaveGame).

**handleSignup (הודעה 203 בפרוטוקול)**

*פרמטרים*:

* + RecievedMessage\* msg – ההודעה שהתקבלה מהלקוח

*ערך החזר*: בוליאני, הצליח להירשם למערכת או לא.

*פונקציונליות*: משיגה מההודעה את הערכים שנשלחו (יוזר, ססמא, מייל). בודקת האם הססמא תקינה (Validator::isPasswordValid), אם לא שולחת הודעת כשלון מתאימה. אם תקין, בודקת האם היוזר תקין (Validator::isUsernameValid), אם לא שולחת הודעת כשלון מתאימה. אם תקין, בודקת האם היוזר כבר קיים במערכת (DataBase::isUserExists), אם קיים שולחת הודעת כשלון מתאימה. אם לא, מוסיפה את היוזר ל-DB (DataBase::addNewUser). אם הפעולה נכשלה שולחת הודעת כשלון מתאימה. אם הצליח שולחת הודעת הצלחה לקליינט.

**handleLeaveGame (הודעה 222 בפרוטוקול)**

*פרמטרים*:

* + RecievedMessage\* msg – ההודעה שהתקבלה מהלקוח

*ערך החזר*: אין

*פונקציונליות*: מפעילה את הפונקציה leaveGame על היוזר שממנו הגיעה ההודעה. אם הפעולה החזירה TRUE המשמעות היא שהמשחק נגמר ויש לשחרר את הזכרון של המשחק.

**handleStartGame (הודעה 217 בפרוטוקול)**

*פרמטרים*:

* + RecievedMessage\* msg – ההודעה שהתקבלה מהלקוח

*ערך החזר*: אין

*פונקציונליות*: מנסה ליצור משחק חדש בהתאם לחדר שמשויך ליוזר שממנו הגיעה ההודעה. אם אירעה Exception בעת יצירת משחק יש לשלוח הודעת כשלון ליוזר. אם הצליח ליצור משחק, יש למחוק את החדר שממנו נוצר המשחק מרשימת החדרים הקיימים. יש לשלוח את השאלה הראשונה (sendFirstQuestion על האובייקט של החדר)

**handlePlayerAnswer (הודעה 219 בפרוטוקול)**

*פרמטרים*:

* + RecievedMessage\* msg – ההודעה שהתקבלה מהלקוח

*ערך החזר*: אין

*פונקציונליות*: משיגה את המשחק על פי היוזר שממנו הגיעה ההודעה. אם המשחק שונה מ-null מפעילה את הפונקציה handleAnswerFromUser על אובייקט המשחק (עם הערכים הרלוונטיים שהגיעו בתוך ההודעה מהיוזר). אם הפונקציה החזירה FALSE, המשמעות היא שהמשחק נגמר ויש לשחרר את המשחק מהזכרון.

**handleCreateRoom (הודעה 213 בפרוטוקול)**

*פרמטרים*:

* + RecievedMessage\* msg – ההודעה שהתקבלה מהלקוח

*ערך החזר*: בוליאני, האם הצליח ליצור חדר או לא

*פונקציונליות*: משיגה את היוזר מההודעה. אם לא משויך יוזר להודעה לא עושה כלום.

משיגה את הערכים המצורפים להודעה. מקדמת ב-1 את מונה החדרים. מפעילה על היוזר את הפעולה createRoom. אם הפעולה הצליחה מוסיפה את החדר שנוצר לרשימת החדרים.

**handleCloseRoom (הודעה 215 בפרוטוקול)**

*פרמטרים*:

* + RecievedMessage\* msg – ההודעה שהתקבלה מהלקוח

*ערך החזר*: בוליאני, האם הצליח לסגור את החדר או לא

*פונקציונליות*: משיגה את החדר של היוזר שמשויך להודעה. אם לא משוייך לחדר מחזירה FALSE. אם משויך, מפעילה על היוזר את הפונקציה closeRoom. אם הפונקציה הצליחה (ערך החזר שונה מ- 1-) מוחקת את החדר מרשימת החדרים.

**handleJoinRoom (הודעה 209 בפרוטוקול)**

*פרמטרים*:

* + RecievedMessage\* msg – ההודעה שהתקבלה מהלקוח

*ערך החזר*: בוליאני, האם הצליח להצטרף או לא.

*פונקציונליות*: משיג את היוזר שמשויך להודעה. אם אין יוזר מחזיר FALSE. משיג את מזהה החדר מהערכים המצורפים להודעה. משיג את החדר על פי מזהה ההודעה (getRoomById). אם לא מצא את החדר, שולח הודעת כשלון מתאימה ליוזר. אם מצא, מפעיל את הפונקציה joinRoom על היוזר.

**handleLeaveRoom (הודעה 211 בפרוטוקול)**

*פרמטרים*:

* + RecievedMessage\* msg – ההודעה שהתקבלה מהלקוח

*ערך החזר*: בוליאני, האם הצליח לעזוב או לא.

*פונקציונליות*: משיג את היוזר שמשויך להודעה. אם אין יוזר מחזיר FALSE. משיג את החדר אליו היוזר משויך. אם לא משויך מחזיר FALSE. מפעיל את leaveRoom על היוזר.

**handleGetUsersInRoom (הודעה 207 בפרוטוקול)**

*פרמטרים*:

* + RecievedMessage\* msg – ההודעה שהתקבלה מהלקוח

*ערך החזר*: אין

*פונקציונליות*: משיג את היוזר שמשויך להודעה. משיג את מזהה החדר מהערכים המצורפים להודעה. משיג את החדר על פי מזהה ההודעה (getRoomById). אם לא מצא את החדר, שולח הודעת כשלון מתאימה ליוזר. אם מצא, מפעיל את הפונקציה getUsersListMessage על החדר ושולח ליוזר את מה שחזר מהפעולה.

**handleGetRooms (הודעה 205 בפרוטוקול)**

*פרמטרים*:

* + RecievedMessage\* msg – ההודעה שהתקבלה מהלקוח

*ערך החזר*: אין

*פונקציונליות*: עובר על רשימת החדרים ובונה הודעה מתאימה לפרוטוקול. שולח את ההודעה ליוזר שממנו הגיעה ההודעה שהתקבלה.

**handleGetBestScores (הודעה 223 בפרוטוקול)**

*פרמטרים*:

* + RecievedMessage\* msg – ההודעה שהתקבלה מהלקוח

*ערך החזר*: אין

*פונקציונליות*: קורא לפונקציה DataBase::getBestScores ומייצר הודעה בהתאם למידע שחזר מהפונקציה ובהתאם לפרוטוקול. את ההודעה שולח ליוזר.

**handleGetPersonalStatus (הודעה 225 בפרוטוקול)**

*פרמטרים*:

* + RecievedMessage\* msg – ההודעה שהתקבלה מהלקוח

*ערך החזר*: אין

*פונקציונליות*: קורא לפונקציה DataBase::getPersonalStatus ומייצר הודעה בהתאם למידע שחזר מהפונקציה ובהתאם לפרוטוקול. את ההודעה שולח ליוזר.

**addRecievedMessage (הודעה 225 בפרוטוקול)**

*פרמטרים*:

* + RecievedMessage\* msg – ההודעה שהתקבלה מהלקוח

*ערך החזר*: אין

*פונקציונליות*: נועל את הMUTEX שמוגדר כמנעול של תור ההודעות (רצוי בעזרת lock\_guard) ומכניס את ההודעה לתור. משחרר את המנעול ומעורר את ה-CV כך שהת'רד שאחראי על טיפול בהודעות יפעל.

**buildRecieveMessage (הודעה 225 בפרוטוקול)**

*פרמטרים*:

* + SOCKET client\_socket – הסוקט של הקליינט
  + Int msgCode – קוד ההודעה

*ערך החזר*: אין

*פונקציונליות*: הפונקציה אחראית ליצור הודעה בהתאם לקוד ההודעה ולמידע הנוסף שהתקבל. כאשר יש מידע נלווה להודעה מעבר לקוד יש להכניס אותו כערכים לוקטור ולהעביר אותו כפרמטר לבנאי של RecievedMessage.

* עיצוב מפורט של ה-DB

**ERD**



**טבלאות**

* **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, autoincrement, not null
* 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, autoincrement, not null
* 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.

* פרוטוקול אפליקטיבי

בכל ההודעות יש סוגריים מרובעים. זה אינו חלק מהפרוטוקול, ורק לשם נוחות בוורד הוספנו אותם..

אפור – הודעה מהקליינט לסרבר

לבן – הודעה מהסרבר לקליינט

|  |  |  |  |
| --- | --- | --- | --- |
| מספר הודעה | מוען - נמען | מהות הבקשה | מבנה הבקשה והסברים |
| 200 | קליינט-סרבר | בקשת sign in | תיאור מילולי:  קוד הודעה = 3 בתים (כמחרוזת)  מספר התווים של היוזר - שני בתים (כמחרוזת)  שם משתמש  מספר התווים של הסיסמה - שני בתים (כמחרוזת)  סיסמה  מבנה ההודעה:  [200##username##pass] |
| 201 | קליינט-סרבר | בקשת sign out | [201] |
| 102 | סרבר-קליינט | מענה לבקשת ההתחברות-  sign in | [1020] //success  [1021] //Wrong Details  [1022] //User is already connected |
| 203 | קליינט-סרבר | בקשת הירשמות-sign up | קוד הודעה = 3 בתים  מספר התווים של היוזר - 2 בתים (כמחרוזת)  שם משתמש  מספר התווים של הסיסמה - 2 בתים (כמחרוזת)  סיסמה  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]  רשימת חדרים [במידה ויש חדרים..]:  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  גודל שם יוזר - 2 בתים  שם יוזר  [108 numberOfUsers ## username ## username …]  אם החדר לא קיים (המשחק התחיל\המנהל סגר את החדר) - מספר המשתמשים יהיה 0 ולא ישלחו שמות יוזרים.  בעצם ההודעה תהיה כזאת:  [1080] |
| 209 | קליינט-סרבר | בקשת הצטרפות לחדר קיים | קוד הודעה = 3 בתים (כמחרוזת)  מזהה חדר - 4 בתים  [209roomID] |
| 110 | סרבר-קליינט | מענה לבקשת הצטרפות לחדר קיים | קוד הודעה = 3 בתים (כמחרוזת)  קוד הצלחה\כישלון- בית אחד (2\1\0, כתו)  במקרה של הצלחה:  מספר השאלות - שני בתים  זמן שאלה - שני בתים  [1100 questionsNumber questionTimeInSec] //success   * אחרי הודעת הצלחה השרת שולח ליוזר שהתחבר ולכל היוזרים המחוברים לחדר הודעת 108 (רשימת היוזרים של החדר)   [1101] // failed - room is full  [1102] // failed - room not exist or other reason |
| 211 | קליינט-סרבר | בקשת עזיבת חדר | [211] |
| 112 | סרבר-קליינט | מענה לבקשת עזיבת חדר | [1120] // success   * 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 (1-9 , 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 - 1\0  [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.  במידה ויש פחות משלושה יוזרים שרשומים במערכת השרת ישלח עבור משתמש לא קיים את הפרטים הבאים: גודל שם משתמש=0 תוצאה=0  [124 ## userName highestScore ## userName highestScore ## userName highestScore] |
| 225 | קליינט-סרבר | בקשת "מצב אישי" | [225] |
| 126 | סרבר-קליינט | מענה לבקשת "מצב אישי" | מספר המשחקים - 4 בתים  מספר התשובות הנכונות - 6 בתים  מספר התשובות השגויות - 6 בתים  זמן מענה ממוצע לשאלה - 4 בתים (2 בתים ראשונים מייצגים את החלק השלם, 2 בתים אחרי את החלק העשרוני. לדוגמא: 0354 מציין שהזמן הוא 3.54 שניות)  [126 numberOfGames numberOfRightAns numerOfWrongAns avgTimeForAns]  במידה ליוזר אין משחקים ותשובות יש להחזיר את ההודעה הבאה (0 עבור כל אחד מהשדות):  [1260000] |
| 299 | קליינט - סרבר | יציאה מהאפליקציה. | [299] |