

מג'ישים

אביב יצחק-7184472067

עומר שחר 0855702090

GitHubLink

Client

מחלקה Client מייצגת לקוח במערכת הרישום של RuppinRegistration.

המחלקה אחראית לשימור פרטי המשתמש, לאיות סיסמה ולזיהוי משתמשים קיימים במערכת. אובייקטים מסוג Client נשמרים ברשימה הלקוחות של השרת ומשמשים לרישום משתמשים חדשים, איות משתמשים קיימים ועדכון פרטי משתמשים.

במחלקה קיימים מספר שדות:

- השדה userName מסוג String מייצג את שם המשתמש של הלוקו ומשמש כenzaה ייחודי במערכת.
- השדה password מסוג String שומר את סיסמת המשתמש.
- השדה academicStatus מסוג String מייצג את הסטטוס האקדמי של המשתמש.
- השדה yearsOfStd מסוג char מייצג את מספר שנות הלימוד של המשתמש ברפוי.
- השדה registered מסוג boolean מציין האם המשתמש כבר רשום במערכת, ומשמש להבנה בין משתמש חדש למשתמש קיים.

למחלקה קיימים בנאי המקבל כפרמטר שם משתמש בלבד.

הבניין משמש לייצור אובייקט Client חדש בתחלת תהליך הרישום, כאשר שאר פרטי המשתמש נקבעים בהמשך תהליך העבודה מול הprotoокול.

במחלקה ממומשת המתודה checkPassword האחראית לבדוק האם סיסמה עומדת בדרישות האבטחה שנקבעו במלטה. הבדיקה כוללת אורך מינימלי של תשעה תווים, קיום של לפחות אות גדולה אחת, אות קטנה אחת וספירה אחת. המתודה מחזירה ערך Boolean המציין האם הסיסמה תקינה.

בנוסף קיימות מתודות setter i getter-ליוצרים עבור פרטי המשתמש, כולל שם משתמש, סיסמה, סטטוס אקדמי ומספר שנות לימוד.

מתודת equals ממומשת לצורך השוואה בין אובייקטים מסוגים Client.

שני לקווחות נחשיים זהים אם יש להם אותו שם משתמש.

מיושן זה מאפשר שימוש במתודת contains על רשימת הלקווחות לצורך מניעת כפליות בשמות משתמשים.

ClientHandler

מחלקה ClientHandler אחראית לטיפול בלקוח בודד המחבר לשרת. המחלקה ירושת מהמחלקה Thread, וכל מופע שלה רץ ב-Thread נפרד, דבר המאפשר תמייה בריבוי ללקוחות במקביל. המחלקה כוללת שדות מרכזים לניהול החיבור והמצב. ה-*Socket* מייצג את החיבור הפיזי ללקוח, ורשימת הלקוחות מייצגת את מצב המשתמשים במערכת ומשותפת לכל ה-Threads של השרת.

למחלקה קיימים בנאים לייצרת אובייקט ClientHandler עם חיבור ללקוח, ועם או בלי גישה לרשימה הלקוחות, בהתאם לפרוטוקול בו נעשה שימוש.

המתודת המרכזית במחלקה היא `run`, אשר מנהלת את מחזור החיים של הטיפול בלקוח. במהלך הריצה נבחר פרוטוקול התקשרות בהתאם לפורט אליו מחובר הלקוח.

לקוחות המתחברים לפורט 4444 מטופלים באמצעות הפרוטוקול KnockKnockProtocol. لكוחות המתחברים לפורט 4445 מטופלים באמצעות הפרוטוקול RuppinRegistrationProtocol. בסיום הטיפול בלקוח החיבור נסגר بصورة מבוקרת, תוך הקפדה על שחרור משאבים גם במקרה של שגיאה.

RuppinRegistrationServer

מחלקה RuppinRegistrationServer אחראית להפעלת שרת הרישום של המערכת ולניהול חיבור ללקוחות לפרוטוקול הרישום.

השרת מאזין לפורט 4445 ומאפשר חיבור של מספר ללקוחות במקביל באמצעות יצירה Thread נפרד לכל ללקוח. במחלקה מוגדרת רשימה ללקוחות משותפת בשם `clientList` המכילה אובייקטים מסוגרים מסוגים `ClientHandler` הרשימה משמשת לשימרת מצב המשתמשים במערכת, לצורך אימות משתמשים קיימים, רישום משתמשים חדשים ומינית כפליות.

הרשימה מוגדרת כרשימה מסונכנת על מנת לאפשר גישה בטוחה בסביבת `MultiThreading`.

במתודת `main` השרת נפתח ומאזין לפורט 4445. בכל פעם חדש מתחבר, השרת יוצר אובייקט ClientHandler חדש, מעביר לו את ה-*Socket* של הלקוח ואות רשימה הלקוחות המשותפת, ומפעיל אותו ב-*Thread* נפרד. מבנה זה מאפשר לשרת להמשיך לקבל חיבורים חדשים מבלי להיחסם על ידי ללקוח יחיד.

בנוסף, המחלקה כוללת מתודת סטטית בשם `saveToCSV` האחראית לגיבוי נתונים הלקוחות לקובץ CSV. המתודה מקבלת את רשימת הלקוחות כפרמטר, מסונכנת את הגישה אליה, ויוצרת קובץ גיבוי בשם הכלל `tar.gz` וועה. במהלך היגייני וכתיבים פרטי כל המשתמשים לקובץ בפורמט טקסטואלי, כאשר כל שדה מופרד בפסיק. מתודת זו נראית מתוך פרוטוקול הרישום לאחר הוספה משתמש חדש, כאשר מספר המשתמשים במערכת מתחלק ב-3, בהתאם לדרישות המطلוה.

RuppinRegistrationProtocol

מחלקה RuppinRegistrationProtocol אחראית לניהול תהליך הרישום, האימות והעדכן של משתמשים במערכת. המחלקה מימוש פרוטוקול תקשורת מבוסס State Machine ומנהלת שיכת מבנית בין השרת ללקוח בהתאם לדרישות המטלה.

מחלקה מחזיקה שדות מצב פנימיים המציגים את שלב השיכת הנוכחי. המשתנה stateExUser משמש לניהול תהליך של משתמש קיימ. המשתנה state משמש לניהול תהליך רישום של משתמש חדש, והמשנה zUser משמש לשיחת החדש.

בנוסף קיימ משתנה BolianIsExUser המשמש להבנה בין תרחיש של משתמש חדש לתרחיש של משתמש קיימ.

מחלקה מחזיקה הפניה לרשיית הליקות המשותפת של השרת, המתקבלת בבניאי. רשיימה זו משמשת לאימות משתמשים קיימים, מניעת כפליות, הוספה משתמשים חדשים ועודון פרטי משתמשים. בנוסף קיימים שדות עזר לאחסן הליקות הנוכחי בתהליך הרישום, וכן הפניה ללקוח קיים בעת תהליך עדכון.

המודה המרכזית במחלקה היא processInput אשר מקבל קלט מהלקוח ומחזירה תשובה מתאימה בהתאם למצב הנוכחי של הפרוטוקול.

המודה מנהלת את זרימת השיכת, כולל רישום משתמש חדש, בדיקת סיסמה, קביעת סטטוס אקדמי ומספר שנות לימוד.

במקרה של משתמש קיים, הלוגיקה מעברת למودה existingUser, בדיקת סיסמה, קביעת סטטוס אקדמי ומספר שנות לימוד. במהלך תהליך זה ניתן לעדכן שם משתמש, סיסמה ומספר שנות לימוד, בהתאם לבחירת המשתמש.

בעת סיום רישום של משתמש חדש, המשתמש מתווסף לרשיית הליקות בצורה מסונכנת לצורך תמייה בסביבה MultiThreading.

כאשר מספר המשתמשים במערכת מתחלק ב-3, הפרוטוקול מפעיל את מודד הגיבוי בשרת לצורך שמירת נתוני המשתמשים לקובץ CSV.

GeneralClient

מחלקה GeneralClient מושמת כתוכנית הליקו הכללית של המערכת, ומאפשרת למשתמש להתחבר לאחד משל הרשתים הזמינים בהתאם לבחירתו.

המחלקה מאפשרת תקשורת הן עם שרת KnockKnock והן עם שרת RuppinRegistration דרך ממשק אחד לצד הליקו.

בתחילת הריצה המשתמש מבקש לחבר לאיזה שרת להתחבר על ידי הזנת מספר הפורט.

המערכת מאפשרת בחירה בין פורט 4444, המשויך לפרוטוקול KnockKnockProtocol לבין פורט 4445, המשויך לפרוטוקול RuppinRegistrationProtocol. מטבחת בדיקת תקינות קלט המשתמש על מנת לוודא שהוזן פורט חוקי בלבד.

לאחר בחירת הפורט, הליקו יוצר חיבור Socket לשרת המתאים ומאתחל ערכיו קלט ופלט לצורך תקשורת דו-כיוונית עם השרת.

הליקו מקבל הודעות מהשרת, מציג אותן למשתמש, ושולח חזרה קלט מהמשתמש בהתאם לזרימת השיכת של הפרוטוקול שנבחר.

הליקו ממשיך את התקשרות עד לקבלת הודעה סיום מהשרת.

במקרה של סיום רישום, יצאה יזומה או סיום שיכת, הלווא נפסקת והחיבור נסגר בצורה מבוקרת.

הסברים נוספים (מה שסומן באדום)

1. אופן בחירת פרוטוקול התקשרות בשרת

במסגרת משימה 2 נוספה תמייה בפרוטוקול תקשות RuppinRegistrationProtocol הפעיל מול שרת המאזין לפורט 4445.

השרת מקבל חיבור לקו ויציר עבورو אובייקט ClientHandler אשר רץ ב-Thread נפרד.

במחלקה ClientHandler מתבצעת בחירת ה프וטוקול על פי הפורט המקורי של החיבור.

אם החיבור בוצע לפורט 4444 מופעל KnockKnockProtocol ואם החיבור בוצע לפורט 4445 מופעל RuppinRegistrationProtocol.

לאחר בחירת הפרוטוקול, ניהול השיחה מתבצע באמצעות קריאות למתחודת `processInquiry` של הפרוטוקול המתאים.

2. סיום חיבור הליקות וניתוק השירות

לאחר שליחת הודעה האחורונה ללקוח, השרת מנתק את החיבור בצורה יזומה.

במחלקה ClientHandler, במהלך הריצה של המתחודה, חנו מתבצעת בדיקה על הודעות הסיום שМОוחזרות מהפרוטוקול.

כאשר מתקבלת הודעה המעדיה על סיום תהליך הרישום או העדכון, כגון `complete` או `bye bye`,
השלואה נפסקת והטיפול בלקוח מסתיים.

לאחר היציאה מהלולאה, החיבור נסגר באמצעות סגירת ה- `Socket` בלוק, `finally` דבר המבטיח שחרור משאבים תקין גם במקרה של שגיאה.

באופן זה השרת ממשיר לפעול ולקלבל חיבורים חדשים, בעוד החיבור ללקוח שסיים את התהליך נסגר בצורה מבוקרת.

3. הסבר כולל על מחלקה Client

הסביר לעיל

קוד המחלקות

```
import java.net.*;
import java.io.*;

public class KnockKnockServer {
    public static void main(String[] args) throws IOException {

        ServerSocket serverSocket = null;
        try {
            serverSocket = new ServerSocket(4444);
            System.out.println("Server listening on port 4444");

            while (true) {
                Socket clientSocket = null;
                try {
                    clientSocket = serverSocket.accept();
                }
```

```
ClientHandler clientHandler = new ClientHandler(clientSocket);
clientHandler.start();

} catch (IOException e) {
    System.err.println("Accept failed.");
    System.exit(1);
}

}
}
catch (IOException e)
{
    System.err.println("Could not listen on port: 4444.");
    System.exit(1);
}

finally{
    if(serverSocket != null){
        try{
            serverSocket.close();

        } catch (IOException e){
            System.out.println("Error closing server socket" + e.getMessage());
        }
    }
}

}
}
```

```
import java.io.*;
import java.net.*;

public class KnockKnockClient{
    public static void main(String[] args) throws IOException {

        Socket kkSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;
```

```

try{
    kkSocket = new Socket("127.0.0.1", 4444);
    out = new PrintWriter(kkSocket.getOutputStream(), true);
    in = new BufferedReader(new InputStreamReader(kkSocket.getInputStream()));
} catch (UnknownHostException e){
    System.err.println("Don't know about host: your host.");
    System.exit(1);
} catch (IOException e){
    System.err.println("Couldn't get I/O for the connection to: your host.");
    System.exit(1);
}

BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
String fromServer;
String fromUser;

while ((fromServer = in.readLine()) != null) {
    System.out.println("Server: " + fromServer);
    if (fromServer.equals("Bye."))
        break;

    fromUser = stdIn.readLine();
    if (fromUser != null) {
        System.out.println("Client: " + fromUser);
        out.println(fromUser);
    }
}

out.close();
in.close();
stdIn.close();
kkSocket.close();

}
}

```

```

import java.net.*;
import java.io.*;

public class KnockKnockProtocol {
    private static final int WAITING = 0;
    private static final int SENTKNOCKKNOCK = 1;
    private static final int SENTCLUE = 2;

```

```

private static final int ANOTHER = 3;

private static final int NUMJOKES = 5;

private int state = WAITING;
private int currentJoke = 0;

private String[] clues = { "Turnup", "Little Old Lady", "Atch", "Who", "Who" };
private String[] answers = { "Turnup the heat, it's cold in here!", "I didn't know you could yodel!", "Bless you!",
    "Is there an owl in here?", "Is there an echo in here?" };

public String processInput(String theInput) {
    String theOutput = null;

    if (state == WAITING) {
        theOutput = "Knock! Knock!";
        state = SENTKNOCKKNOCK;
    }
    else if (state == SENTKNOCKKNOCK) {
        if (theInput.equalsIgnoreCase("Who's there?")) {
            theOutput = clues[currentJoke];
            state = SENTCLUE;
        } else {
            theOutput = "You're supposed to say \"Who's there?\"! " + "Try again. Knock! Knock!";
        }
    }
    else if (state == SENTCLUE) {
        if (theInput.equalsIgnoreCase(clues[currentJoke] + " who?")) {
            theOutput = answers[currentJoke] + " Want another? (y/n)";
            state = ANOTHER;
        } else {
            theOutput = "You're supposed to say \" " + clues[currentJoke] + " who?\"! Try again. Knock! Knock!";
            state = SENTKNOCKKNOCK;
        }
    }
    else if (state == ANOTHER) {
        if (theInput.equalsIgnoreCase("y")) {
            theOutput = "Knock! Knock!";
            if (currentJoke == (NUMJOKES - 1))
                currentJoke = 0;
            else
                currentJoke++;
            state = SENTKNOCKKNOCK;
        } else {
            theOutput = "Bye.";
        }
    }
}

```

```
        state = WAITING;
    }
}
return theOutput;
}
}
```

```
public class Client {
    private String userName;
    private String password;
    private String academicStatus;
    private char yearsOfStd;
    private boolean registered;

    public Client(String userName) {
        this.userName=userName;
    }

    //checking the password requirement
    public boolean checkPassword(String password) {
        char [] passwordChar=password.toCharArray();
        boolean isUpper =false;
        boolean isLower=false;
        boolean isDigit=false;

        if (passwordChar.length < 9) {
            return false;
        }
        for (char i:passwordChar ) {
            if (Character.isUpperCase(i)) {
                isUpper=true;
            } else if (Character.isLowerCase(i)) {
                isLower=true;
            } else if (Character.isDigit(i)) {
                isDigit=true;
            }
        }
        return isUpper && isLower && isDigit;
    }
}
```

```

}

public void setPassword(String password) {
    this.password=password;

}

public String getPassword() {
    return this.password;
}

public void setUserName(String userName) {
    this.userName=userName;
}

public String getUserName() {
    return userName;
}

public void setAcademicStatus(String academicStatus) {
    this.academicStatus=academicStatus;
}

public String getAcademicStatus() {
    return academicStatus;
}

public void setYearsOfStd(char yearsOfStd){
    this.yearsOfStd=yearsOfStd;
}

public char getYearsOfStd() {
    return yearsOfStd;
}

public boolean equals(Object obj){
    if (obj instanceof Client){
        return this.userName.equals(((Client)obj).userName);
    }
    return false;
}

}

```

```

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;

```

```

import java.net.Socket;
import java.util.List;

public class ClientHandler extends Thread {
    private final Socket clientSocket;
    private List<Client> clientList;

    public ClientHandler(Socket clinetSocket) {
        this.clientSocket=clinetSocket;
    }
    public ClientHandler(Socket clinetSocket, List<Client> clientList) {
        this.clientSocket=clinetSocket;
        this.clientList=clientList;
    }

    public void run() {
        try (PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));) {
            String inputLine, outputLine;
            //talk with KnockKnockProtocol
            if (clientSocket.getLocalPort() == 4444) {
                KnockKnockProtocol kkp = new KnockKnockProtocol();
                outputLine = kkp.processInput(null);
                out.println(outputLine);

                while ((inputLine = in.readLine()) != null) {
                    if (inputLine.equals("q")) break;
                    outputLine = kkp.processInput(inputLine);
                    out.println(outputLine);
                }
                //talk with RuppinRegistrationProtocol
            } else {
                RuppinRegistrationProtocol rrp=new RuppinRegistrationProtocol(clientList);
                outputLine=rrp.processInput(null);
                out.println(outputLine);

                while ((inputLine = in.readLine()) != null) {
                    if (inputLine.equals("q")) break;
                    outputLine = rrp.processInput(inputLine);
                    out.println(outputLine);
                }
            }
            //break from the loop and continue to close the clientSocket
        }
    }
}

```

```

        //after the registration completed
        if(outputLine.equals("Registration complete.") || outputLine.equals("Thank you, bye bye")
        || outputLine.contains("Thanks, Your information has been updated")) break;

    }

}

} catch (IOException e) {
    System.out.println("Error handling client" + e.getMessage());
}
finally{
    try{
        clientSocket.close();
        System.out.println("Client disconnected: " + clientSocket.getPort());
    } catch (IOException e) {
        System.out.println("Error closing socket: " + clientSocket.getPort() + "\n" + e.getMessage());
    }
}

}
}

```

```

import java.io.*;
import java.net.*;
import java.util.Scanner;

public class GeneralClient {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        int chosenPort;

        System.out.println("Choose server (Enter only digit):" + "\n" + "KnockKnock Server- 4444 "
        + "\n" +"Ruppin Registration Server -4445");

        while (true) {
            if (scanner.hasNextInt()) {

```

```

chosenPort = scanner.nextInt();
if (chosenPort == 4444 || chosenPort == 4445) {
    break;
}
}
else
{
    scanner.next();
}
System.out.println("Invalid input!" + "\n" + "Enter 4444 or 4445: ");
}

try (Socket socket = new Socket("127.0.0.1", chosenPort);
    PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in)))
){

String fromTheServer;
String fromTheUser;

while ((fromTheServer = in.readLine()) != null) {
    System.out.println("Server: " + fromTheServer);

    if (fromTheServer.equals("Registration complete.") ||
        fromTheServer.equals("Thank you, bye bye") ||
        fromTheServer.contains("Thanks, Your information has been updated")||
        fromTheServer.equalsIgnoreCase("Bye.")) {
        break;
    }

    fromTheUser = userInput.readLine();
    if (fromTheUser != null){
        out.println(fromTheUser);
    }
}

} catch (IOException e){
    System.out.println("Connection error: " + e.getMessage());
}
}
}

```

```

import java.util.List;

public class RuppinRegistrationProtocol {
    private int state=1;
    private int stateExUser=1;
    private boolean isExUser =false;

    List<Client> clientList;
    Client client;
    Client temp;
    Client refToRealClient;

    public RuppinRegistrationProtocol(List<Client> clientList) {
        this.clientList=clientList;
    }

    public String processInput(String theInput){
        //String theOutput=null;
        char yearOfStd;

        if (isExUser) {
            return existingUser(theInput);
        }

        switch (state){
            case 1:
                state++;
                return "Do you want to register? (y/n): ";
            case 2:
                if (theInput.equalsIgnoreCase("y")){
                    state++;//equal 3
                    return "Enter a username: ";
                } else if (theInput.equalsIgnoreCase("n")){
                    isExUser=true;
                    return existingUser(theInput); //go to the second session scenario
                } else {
                    return "invalid selection!, Do you want to register? (y/n): ";
                }
            case(3):
                client = new Client(theInput);
                if (clientList.contains(client)){
                    return "Checking name... Name not OK. Username exists. Choose a different name: ";
                } else {

```

```

        state++; //equal 4
        return "Checking name... OK!. Enter a strong password: ";

    }

    case (4):
        if (client.checkPassword(theInput)) {
            client.setPassword(theInput);
            state++; //equal 5
            return "Password accepted!, What is your academic status? (student/teacher/other)";
        }
        else
            return "Password does not meet the requirements!, try again";

    case(5):
        if (theInput.equalsIgnoreCase("student") || theInput.equalsIgnoreCase("teacher")
            || theInput.equalsIgnoreCase("other")){
            client.setAcademicStatus(theInput);
            state++; //equal 6
            return "How many years have you been at Ruppin?";
        }
        else
            return "Invalid status";

    case (6):
        yearOfStd = theInput.charAt(0);
        if (theInput.length() == 1 && Character.isDigit(yearOfStd)) {
            state = 1;
            client.setYearsOfStd(yearOfStd);
            //lock that block, prevent multi client insertion to the list
            synchronized (clientList) {
                if (clientList.contains(client)) {
                    state = 3; //go back to the user picking name stage
                    return "Username was taken during registration. Please choose a different username: ";
                } else {
                    clientList.add(client); // after we're done configure the client fields we can add him to the list
                    if (clientList.size() % 3 == 0){
                        RuppinRegistrationServer.saveToCSV(clientList);

                    }
                    return "Registration complete.";
                }
            }
        }
    }

} else {

```

```

        return "invalid input, try again";
    }

}

// fallback – should not happen
return "Invalid input. Please try again.";
}

private String existingUser(String theInput) {
    switch (stateExUser) {
        case (1):
            stateExUser++; //equal 2
            return "Username: ";
        case (2):
            temp = new Client(theInput);
            if (clientList.contains(temp)) {
                stateExUser++; //equal 3
                return "Password: ";
            } else {
                return "Incorrect Username: ";
            }
        case (3):
            refToRealClient = clientList.get(clientList.indexOf(temp));
            if (theInput.equals(refToRealClient.getPassword())) {
                stateExUser++; //equal 4
                return "Welcome back " + refToRealClient.getUserName() + ", Last time you defined yourself as " +
refToRealClient.getAcademicStatus()
                + " for " + refToRealClient.getYearsOfStd() + " years, Do you want to update your information?
(yes/no) ";
            } else {
                return "Incorrect Password: ";
            }
        case (4):
            if (theInput.equals("no")) {
                return "Thank you, bye bye";
            } else if (theInput.equals("yes")) {
                stateExUser++; //equal 5
                return "Do you want to change your username? (yes/no)";
            }
        } else {
            return "Invalid input, Do you want to update your information? (yes/no)";
        }
    }
}
```

```

    }

case (5):
    if (theInput.equals("yes")) {
        stateExUser++; // equal 6
        return "Enter new username: ";

    } else if (theInput.equals("no")) {
        stateExUser = 7;
        return "Do you want to change your password? (yes/no)";

    } else {
        return "Invalid input!. Do you want to change your username? (yes/no)";
    }

case (6):
    refToRealClient.setUserName(theInput);
    stateExUser++; // equal 7
    return "Username updated successfully. Do you want to change your password? (yes/no)";

case (7):
    if (theInput.equals("yes")){
        stateExUser++; // equal 8
        return "Enter new password:";

    } else if (theInput.equals("no")){
        stateExUser = 9;
        return "Do you want to update your years of study? (yes/no)";

    } else {
        return "Invalid input!. Do you want to change your password? (yes/no)";
    }

case (8):
    if (refToRealClient.checkPassword(theInput)){
        refToRealClient.setPassword(theInput);
        stateExUser++; // equal 9
        return "Password updated successfully. Do you want to update your years of study? (yes/no)";

    } else {
        return "Password does not meet the requirements, try again";
    }

case (9):
    if (theInput.equals("yes")){
        stateExUser++; // equal 10

```

```

        return "Enter number of years:";
    } else if (theInput.equals("no")) {
        return "Thanks, Your information has been updated.";
    } else {
        return "Invalid input!. Do you want to update your years of study? (yes/no)";
    }

    case (10):
        if (theInput.length() == 1 && Character.isDigit(theInput.charAt(0))) {
            refToRealClient.setYearsOfStd(theInput.charAt(0));
            return "Years of study updated successfully. Thanks, Your information has been updated.";
        } else {
            return "Invalid input, Enter number of years";
        }

    }

    // fallback – should not happen
    return "Invalid input. Please try again.";
}

}

```

```

import
java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class RuppinRegistrationServer {
    public static void main(String[] args) throws IOException {

        //lock the list for safe run of parallel client
        List<Client> clientList = Collections.synchronizedList(new ArrayList<>());

```

```

ServerSocket serverSocket = null;
try{
    serverSocket = new ServerSocket(4445);
    System.out.println("Server listening on port 4445");

    while (true) {
        Socket clientSocket = null;
        try{
            clientSocket = serverSocket.accept();
            ClientHandler clientHandler = new ClientHandler(clientSocket,clientList);
            clientHandler.start();

        } catch (IOException e){
            System.err.println("Accept failed.");
            System.exit(1);
        }

    }
} catch (IOException e)
{
    System.err.println("Could not listen on port: 4445.");
    System.exit(1);
}

finally{
    if(serverSocket != null){
        try{
            serverSocket.close();

        } catch (IOException e){
            System.out.println("Error closing server socket" + e.getMessage());
        }
    }
}

}

public static void saveToCSV(List<Client> clientList) {
    synchronized (clientList){

        String date = LocalDateTime.now()
            .format(DateTimeFormatter.ofPattern("ddMMyyyy_HHmm"));
    }
}

```

```

String fileName = "backup_" + date + ".csv";

try (PrintWriter writer = new PrintWriter(new FileWriter(fileName))) {

    for (Client i : clientList) {
        writer.println(
            i.getUserName() + "," +
            i.getPassword() + "," +
            i.getAcademicStatus() + "," +
            i.getYearsOfStd()
        );
    }

    System.out.println("Backup created: " + fileName);

} catch (IOException e) {
    System.err.println("Backup error: " + e.getMessage());
}
}

}

}

```

צילומי מסך

Name	Date modified	Type	Size
.idea	05-Jan-26 12:38 AM	File folder	
out	22-Dec-25 8:55 PM	File folder	
src	05-Jan-26 12:38 AM	File folder	
.gitignore	22-Dec-25 7:41 PM	Git Ignore Source ...	1 KB
Assignment3.iml	22-Dec-25 7:41 PM	IML File	1 KB
backup_04012026_2338.csv	04-Jan-26 11:38 PM	Excel.CSV	1 KB
backup_04012026_2342.csv	04-Jan-26 11:42 PM	Excel.CSV	1 KB

File	Edit	View
avivi,Aa1234567,other,7 aviva,Aa1234567,student,6 omer,A12345a6789,teacher,2 mike,Bb1234567,other,3 yossi,Cc123c456c7,teacher,1 moshe,123V123v123,other,7		

user 1 register

```
Run RuppinRegistrationServer x
C ⚡ | 📸 ⏪ ⏴ ⏵ :  
↑ "C:\Program Files\Java\jdk-25\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\In  
↓ Server listening on port 4445  
Client disconnected: 54070  
= ⌂  
= ⌄  
= ⌆  
= ⌇  
= ⌈  
= ⌉  
GeneralClient x
C ⚡ | 📸 ⏪ ⏴ ⏵ :  
↑ "C:\Program Files\Java\jdk-25\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\In  
↓ Choose server (Enter only digit):  
KnockKnock Server- 4444  
Ruppin Registration Server -4445  
= ⌂  
= ⌄  
= ⌆  
= ⌇  
= ⌈  
= ⌉  
4445  
Server: Do you want to register? (y/n):  
y  
Server: Enter a username:  
aviv  
Server: Checking name... OK!. Enter a strong password:  
Aa1234567  
Server: Password accepted!, What is your academic status? (student/teacher/other)  
student  
Server: How many years have you been at Ruppin?  
2  
Server: Registration complete.
```

User 2 register

```
Run RuppinRegistrationServer x

GeneralClient x

Choose server (Enter only digit):
↑ KnockKnock Server- 4444
↓ Ruppin Registration Server -4445
4445
	Server: Do you want to register? (y/n):
y
Server: Enter a username:
omer
Server: Checking name... OK!. Enter a strong password:
C147258Aa
Server: Password accepted!, What is your academic status? (student/teacher
teacher
Server: How many years have you been at Ruppin?
5
Server: Registration complete.
```

User 1 update

```
Run RuppinRegistrationServer x
Run GeneralClient x

↑ "C:\Program Files\Java\jdk-25\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2025.3.1\lib\idea_rt.jar=60961" -Dfile.encoding=UTF-8
↓ Server listening on port 4445
→ Client disconnected: 54070
→ Client disconnected: 54112
→ Client disconnected: 54128

GeneralClient x
Run GeneralClient x

↑ KnockKnock Server- 4444
↓ Ruppin Registration Server -4445
→ 4445
→ Server: Do you want to register? (y/n):
→ n
→ Server: Username:
→ aviv
→ Server: Password:
→ Aa1234567
→ Server: Welcome back aviv, Last time you defined yourself as student for 2 years, Do you want to update your information? (yes/no)
→ yes
→ Server: Do you want to change your username? (yes/no)
→ yes
→ Server: Enter new username:
→ av
→ Server: Username updated successfully. Do you want to change your password? (yes/no)
→ no
→ Server: Do you want to update your years of study? (yes/no)
→ yes
→ Server: Enter number of years:
→ 3
→ Server: Years of study updated successfully. Thanks, Your information has been updated.

Process finished with exit code 0
```