

# מגישים

אביב יצחק-206718447

עומר שחר 209085570

GitHubLink

## Client

מחלקת Client מייצגת לקוח במערכת הרישום של RuppinRegistration. המחלקה אחראית לשמירת פרטי המשתמש, לאימות סיסמה ולזיהוי משתמשים קיימים במערכת. אובייקטים מסוג Client נשמרים ברשימת הלקוחות של השרת ומשמשים לרישום משתמשים חדשים, אימות משתמשים קיימים ועדכון פרטי משתמשים.

במחלקה קיימים מספר שדות:

- השדה userName מסוג String מייצג את שם המשתמש של הלקוח ומשמש כמזהה 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 ברופ"ן.
- השדה registered מסוג boolean מציין האם המשתמש כבר רשום במערכת, ומשמש להבחנה בין משתמש חדש למשתמש קיים.

למחלקה קיים בנאי המקבל כפרמטר שם משתמש בלבד. הבנאי משמש ליצירת אובייקט Client חדש בתחילת תהליך הרישום, כאשר שאר פרטי המשתמש נקבעים בהמשך תהליך העבודה מול הפרוטוקול.

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

בנוסף קיימות מתודות setter | getter לצורך עדכון ושליפת פרטי המשתמש, כולל שם משתמש, סיסמה, סטטוס אקדמי ומספר שנות לימוד.

מתודת equals ממומשת לצורך השוואה בין אובייקטים מסוג Client. שני לקוחות נחשבים זהים אם יש להם אותו שם משתמש. מימוש זה מאפשר שימוש במתודת contains על רשימת הלקוחות לצורך מניעת כפילויות בשמות משתמשים.

## ClientHandler

מחלקת ClientHandler אחראית לטיפול בלקוח בודד המחובר לשרת. המחלקה יורשת מהמחלקה Thread, וכל מופע שלה רץ ב־Thread נפרד, דבר המאפשר תמיכה בריבוי לקוחות במקביל. המחלקה כוללת שדות מרכזיים 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 ליצירת אובייקט ClientHandler עם חיבור ללקוח, ועם או בלי גישה לרשימת הלקוחות, בהתאם לפרוטוקול בו נעשה שימוש. המתודה המרכזית במחלקה היא run, אשר מנהלת את מחזור החיים של הטיפול בלקוח. במהלך הריצה נבחר פרוטוקול התקשורת בהתאם לפורט אליו מחובר הלקוח. לקוחות המתחברים לפורט 4444 מטופלים באמצעות הפרוטוקול KnockKnockProtocol. לקוחות המתחברים לפורט 4445 מטופלים באמצעות הפרוטוקול RuppinRegistrationProtocol. בסיום הטיפול בלקוח החיבור נסגר בצורה מבוקרת, תוך הקפדה על שחרור משאבים גם במקרה של שגיאה.

## RuppinRegistrationServer

מחלקת RuppinRegistrationServer אחראית להפעלת שרת הרישום של המערכת ולניהול חיבורי לקוחות לפרוטוקול הרישום. השרת מאזין לפורט 4445 ומאפשר חיבור של מספר לקוחות במקביל באמצעות יצירת Thread נפרד לכל לקוח. במחלקה מוגדרת רשימת לקוחות משותפת בשם clientList המכילה אובייקטים מסוג Client. הרשימה משמשת לשמירת מצב המשתמשים במערכת, לצורך אימות משתמשים קיימים, רישום משתמשים חדשים ומניעת כפילויות. הרשימה מוגדרת כרשימה מסונכרנת על מנת לאפשר גישה בטוחה בסביבת MultiThreading. במתודת main השרת נפתח ומאזין לפורט 4445. בכל פעם שלקוח חדש מתחבר, השרת יוצר אובייקט ClientHandler חדש, מעביר לו את ה־Socket של הלקוח ואת רשימת הלקוחות המשותפת, ומפעיל אותו ב־Thread נפרד. מבנה זה מאפשר לשרת להמשיך לקבל חיבורים חדשים מבלי להיחסם על ידי לקוח יחיד. בנוסף, המחלקה כוללת מתודה סטטית בשם saveToCSV האחראית לגיבוי נתוני הלקוחות לקובץ CSV. המתודה מקבלת את רשימת הלקוחות כפרמטר, מסנכרנת את הגישה אליה, ויוצרת קובץ גיבוי בשם הכולל תאריך ושעה. במהלך הגיבוי נכתבים פרטי כל המשתמשים לקובץ בפורמט טקסטואלי, כאשר כל שדה מופרד בפסיק. מתודה זו נקראת מתוך פרוטוקול הרישום לאחר הוספת משתמש חדש, כאשר מספר המשתמשים במערכת מתחלק ב־3, בהתאם לדרישות המטלה.

## RuppinRegistrationProtocol

מחלקת RuppinRegistrationProtocol אחראית לניהול תהליך הרישום, האימות והעדכון של משתמשים במערכת. המחלקה מממשת פרוטוקול תקשורת מבוסס State Machine ומנהלת שיחה מובנית בין השרת ללקוח בהתאם לדרישות המטלה.

המחלקה מחזיקה שדות מצב פנימיים המייצגים את שלב השיחה הנוכחי. המשתנה state משמש לניהול תהליך רישום של משתמש חדש, והמשתנה stateExUser משמש לניהול תהליך של משתמש קיים.

בנוסף קיים משתנה בוליאני isExUser המשמש להבחנה בין תרחיש של משתמש חדש לתרחיש של משתמש קיים.

המחלקה מחזיקה הפניה לרשימת הלקוחות המשותפת של השרת, המתקבלת בבנאי. רשימה זו משמשת לאימות משתמשים קיימים, מניעת כפילויות, הוספת משתמשים חדשים ועדכון פרטי משתמשים. בנוסף קיימים שדות עזר לאחסון הלקוח הנוכחי בתהליך הרישום, וכן הפניה ללקוח קיים בעת תהליך עדכון.

המתודה המרכזית במחלקה היא processInput, אשר מקבלת קלט מהלקוח ומחזירה תשובה מתאימה בהתאם למצב הנוכחי של הפרוטוקול.

המתודה מנהלת את זרימת השיחה, כולל רישום משתמש חדש, בדיקת סיסמה, קביעת סטטוס אקדמי ומספר שנות לימוד. במקרה של משתמש קיים, הלוגיקה מועברת למתודה existingUser, האחראית לניהול תהליך הזדהות ועדכון פרטי משתמש. במהלך תהליך זה ניתן לעדכן שם משתמש, סיסמה ומספר שנות לימוד, בהתאם לבחירת המשתמש.

בעת סיום רישום של משתמש חדש, המשתמש מתווסף לרשימת הלקוחות בצורה מסונכרנת לצורך תמיכה בסביבת MultiThreading.

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

## **GeneralClient**

מחלקת GeneralClient משמשת כתוכנית הלקוח הכללית של המערכת, ומאפשרת למשתמש להתחבר לאחד משני השרתים הזמינים בהתאם לבחירתו. המחלקה מאפשרת תקשורת הן עם שרת KnockKnock והן עם שרת RuppinRegistration דרך ממשק אחיד בצד הלקוח.

בתחילת הריצה המשתמש מתבקש לבחור לאיזה שרת להתחבר על ידי הזנת מספר הפורט. המערכת מאפשרת בחירה בין פורט 4444, המשויך לפרוטוקול KnockKnockProtocol לבין פורט 4445, המשויך לפרוטוקול RuppinRegistrationProtocol. מתבצעת בדיקת תקינות לקלט המשתמש על מנת לוודא שהוזן פורט חוקי בלבד.

לאחר בחירת הפורט, הלקוח יוצר חיבור Socket לשרת המתאים ומאתחל ערוצי קלט ופלט לצורך תקשורת דו-כיוונית עם השרת. הלקוח מקבל הודעות מהשרת, מציג אותן למשתמש, ושולח חזרה קלט מהמשתמש בהתאם לזרימת השיחה של הפרוטוקול שנבחר.

הלקוח ממשיך את התקשורת עד לקבלת הודעת סיום מהשרת. במקרה של סיום רישום, יציאה יזומה או סיום שיחה, הלולאה נפסקת והחיבור נסגר בצורה מבוקרת.

## **הסברים נוספים (מה שסומן באדום)**

1. אופן בחירת פרוטוקול התקשורת בשרת



```

        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 clientSocket) {
        this.clientSocket=clientSocket;
    }
    public ClientHandler(Socket clientSocket, List<Client> clientList) {
        this.clientSocket=clientSocket;
        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 Ruppín?";
        }
        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) {
                        RuppínRegistrationServer.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

backup\_04012026\_2342.csv

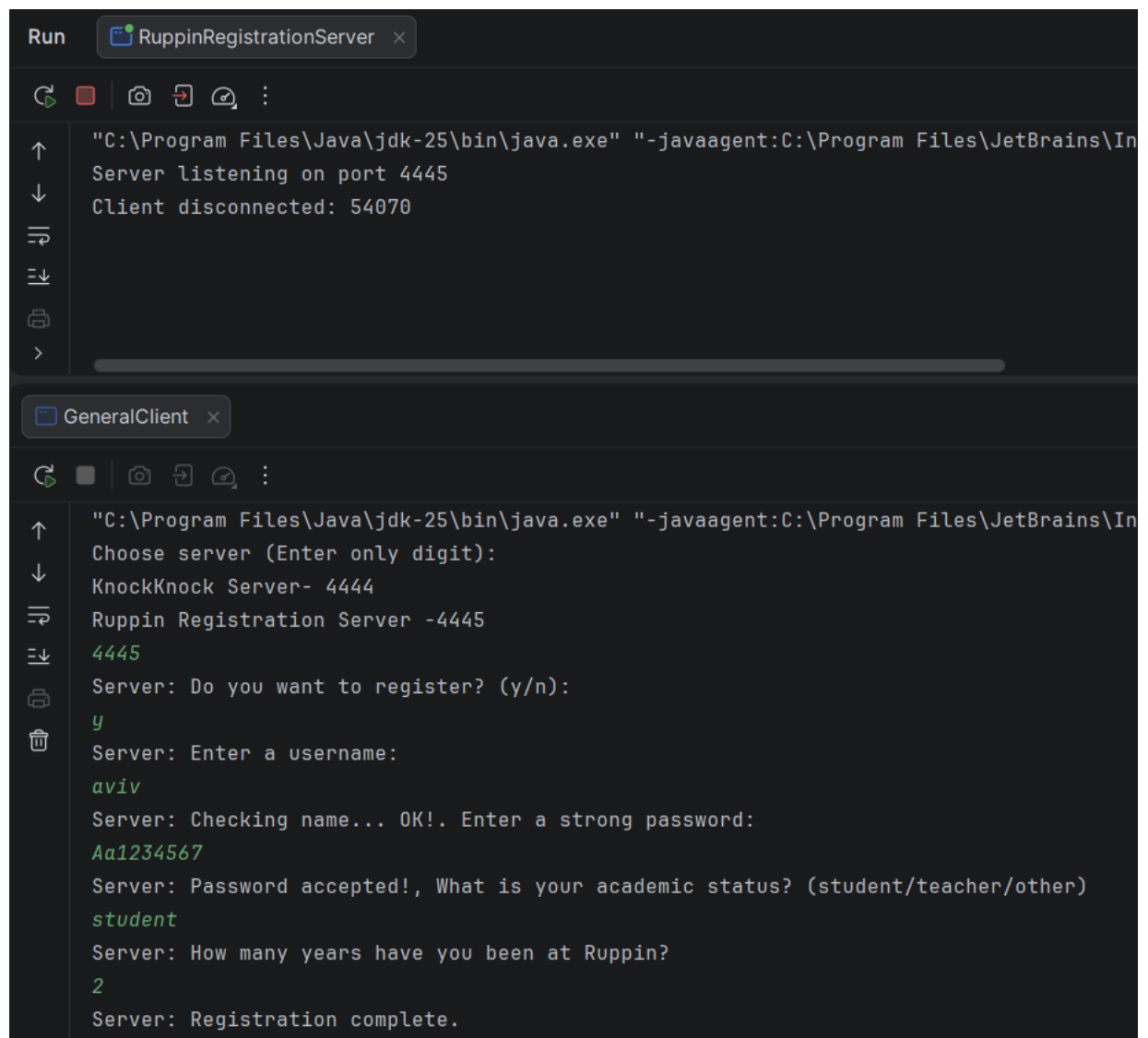
File Edit View

```

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



The image shows a screenshot of an IDE with two terminal windows. The top window, titled 'Run' and 'RuppinRegistrationServer', shows the server's output. The bottom window, titled 'GeneralClient', shows the client's interaction with the server.

```
Run RuppinRegistrationServer x
"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:\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
"\"C:\\Program Files\\Java\\jdk-25\\bin\\java.exe\" \"-javaagent:C:\\Program Files\\J
Server listening on port 4445
Client disconnected: 54070
Client disconnected: 54112

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

"\"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
Server listening on port 4445
Client disconnected: 54070
Client disconnected: 54112
Client disconnected: 54128

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
```