

## דוח מסכם

# פרויקט גמר בקורס רשתות

## תקשורת מחשבים

ניתוח תעבורת בפרוטוקול TCP/IP

קבוצה 5-61305

מגישות:

נעה סיאחצקי

עדי חדד

מעין לוי

**תוכן עניינים:**

**חלק 1 – אפיית נתונים ולכידת מנוט בעזרת Wireshark**

- ✓ 3 ..... ייצירת קובץ ה-CSV .....
- ✓ 3 ..... תיאור והסביר תהליך אפיית המנות (Encapsulation Process) .....
- ✓ 6 ..... ניתוח תעבורת Wireshark Analysis .....

**חלק 2 – כתיבת יישום רשות וניתוח תעבורת של אותו יישום**

- ✓ 10 ..... הסבר כללי על המערכת ומבנה הקוד .....
- ✓ 13 ..... הוראות התקנה והריצה .....
- ✓ 14 ..... דוגמאות קלט פלט וניתוח .....

**חלק 3 - תיאור שימוש במבנה מלאכותית**

- 21 ..... 22 ..... **חלק 4 – ייצרת וUI למשתמש – בונוס**

## **חלק 1: אരיזה ולכידת מנות (Capture & Encapsulation)**

בחלק זה של העבודה, נתמקד בישום המעני של תהליך התקשרות. נדגים כיצד הקוד קורא את ההודעות המאווחנות בקובץ CSV וממייר אותן לubyteות רשות חייה, ובמקביל נציג את הממצאים כפי שנקלטו ב-Wireshark לצורך אימונות וניתוח ה프וטוקול.

### ✓ **יצירת קובץ ה-CSV**

יצרנו את קובץ הקלט בעזרת AI, דוגמת תעבורת HTTP/Application. הקובץ מכיל את העמודות הבאות:

1. **id msg (מזהה הודעה):** מספר סידורי המאפשר מעקב אחר סדר הגעת החבילות ושלמותן ב-Wireshark.
2. **app\_protocol (פרוטוקול):** תוכית המציין את שם ה프וטוקול (כגון HTTP) שבו משתמשים בשכבה האפליקציה.
3. **app\_src (פורט מקור):** פורט הלוקה, המזון לתחום שדה ה-Source ברכבתת ה-TCP.
4. **app\_dst\_appt (פורטיעד):** פורט השרת (הוגדר כ-12345), רלוונטי לפינון ויזיהו התעבורה הרלוונטית ב-Wireshark.
5. **message (תוכן):** המידע עצמו. נארז בסגמנט ה-TCP ונitinן לקריאהقطפס בזמן הניתוה.
6. **timestamp (זמן):** קובע את מועד השליחה ומאפשר שליטה בקצב התעבורה בזמן היריצה.

### ✓ **תיאור והסביר תהליכי אrizת המנות (Encapsulation Process)**

תהליך האrizה בפרויקט מדמה את המתבצע במחסנית ה프וטוקולים (Protocol Stack) של מערכת הפעלה, תוך מעבר משכבה האפליקציה ועד ליצירת רצף בייטים המוכנים לשלוח:

1. **שכבות היישום (Application):** ההודעה הגלמית מה CSV-מקודדת למחרוזת בייטים (UTF-8) המהווה את המטען של החבילה (Payload).
2. **שכבת התעבורה (TCP - Transport):** **מבנה Header:** השתמשנו בפונקציה build\_tcp\_header כדי ליצור כוורת TCP תקנית בגודל 20 בתים. הכוורת כוללת את פורט המקור והיעד שהגדכנו ב-CSV. שדות Sequence Number ו-Source Port הוגרלו רנדומלית. **מנגנון ה-Csum:** לחישוב אמינות הנתונים, בנוינו "פואדו-כוורת" (Pseudo-header) הכוללת את כתובות ה-IP ופרוטוקול ה-TCP, וביצענו חישוב checksum על כל החבילה. זהו

שלב קרייטי המבטיח שהצד מקבל לא יפסול את החבילה כפגומה.

**שימוש בדגלים (Flags):** הגדרנו דגל 180x (PSH+ACK). דגל ה-Push מורה למערכת הפעלה ביעד להעביר את הנתונים מיד לישום מבלי להמתין למילוי ה-Buffer.

.3 **שכבה הרשת (Network - IP)**

הfonקציית build\_ip\_header אורצת את כתובת ה-TCP בתחום כתובת IP. CAN מוגדרים כתובות ה-IP.

.4 התוצאה הסופית:

הקוד לחבר את שלוש השכבות (IP + TCP + Application Data) לרוץ אחד המיצג Frame שלם שנייה להזירה לכרטיס הרשת.

```
[13] # Preview packet structure
src_ip = '127.0.0.1'
dst_ip = '127.0.0.1'
src_port = random.randint(1024, 65535)
dst_port = 12345
payload = b'Hello Packet (preview)'
pkt_preview = build_ip_header(src_ip, dst_ip, 20 + len(payload)) + build_tcp_header(src_ip, dst_ip, src_port, dst_port, payload) + payload
hexdump(pkt_preview)

[13] ✓ 0.0s
Python
```

0000 45 00 00 3e 21 e5 00 00 40 06 5a d3 7f 00 00 01 E..>!...@.Z.....
0010 7f 00 00 01 87 73 30 39 e5 33 ed f7 00 00 00 00 .....\$0.3. .....
0020 50 02 ff ff 13 1f 00 00 48 65 6c 6c 6f 20 50 61 P.....Hello Pa
0030 63 6b 65 74 20 28 70 72 65 76 69 65 77 29 ket (preview)

בצילום ניתן לראות את הביטים הראשונים (...00 45) המציגים את כוורתה, IP-ובסוף הרצף ניתן לראות בבירור את הטעסט המקודד ("Hello Packet") מה שמכיח אריזה מדוקט.  
(התמונה מהלכידה בwireshark צולמה בתאריך שונה ולכן הביטים אינם תואמים בשלמות. אך ניתן לראות כי החבילת הנרצה במצבה נכונה).

✓ פירוט והסבר על תהליכי הרצת מחברת Jupiter:

#### **1. עיינת הנתונים ותהליכי השילוח:**

בשלב הראשון, נען קובץ הנתונים (CSV) לתוכו טבלת נתונים בשם df\_messages. תהליך השילוח מתבצע באמצעות לולאת for על כל השורות בטבלה. בכל איטרציה נשלף תוכן הודעה (שדה message) ונשלח באמצעות אובייקט תעבורת (transport) שהוגדר מראש.

## 2. **תהליכי היפוי הרים (Encapsulation)**

האובייקט `transport` מוגדר כמופיע של המחלקה `RawTcpTransport`. מחלקת זו אחראית על לוגיקת השליחה ועל בניית המנות. בעת הפעלת הפונקציה `transport.send` מתבצע קרייה לפונקצייה `encapsulate`. מטרת פונקציה זו היא

"לאroud" את המידע משכבה היישום (הודעת הטקסט) ולהעבירו דרך שכבות התקשורת התחתונות עד ליצירת חבילת מלאה.

3. מתודולוגיית הלכידה(Wireshark Capture) פתחנו את Wireshark והזינו לממשק Loopback, מכיוון שהתקשרות מתחכעת בתוך המחשב המקומי, החבילות אינן יוצאות לכרטיס הרשות הפיזי. כדי לבדוק את המנות שלנו מכלל תשובות המחשב, השתמשנו בפילטר תצוגה: .tcp.port == 12345
4. הרצנו את הסקריפט (Jupyter Notebook) ששלח את המנות.
5. עם סיום הלכידה, ביצענו השוואה בין ה-Hex Dump של החבילת ב-Wireshark לבין שורת ההודעה המקורית ב-CSV. האיות הראה התאמה של 100%, מה שמכיח שההיליך האריזה והשליחה בוצע בהצלחה.

## ניתוח תבעורה (Wireshark Analysis)

לאחר הריצת הסקריפט הדורש, באמצעות מחברת הפיתון עם קלט DATA (כפי שתואר לעלה), אשר מגדים יצירה ושליחה של מנות - כתף, נציג ניתוחמנה שנעשתה באמצעות Wireshark. הניתוח יציג לפי סדר השכבות של המנה, החל מכותרות הпрוטוקולים, Headers, ועד לתוך המידע, payload.

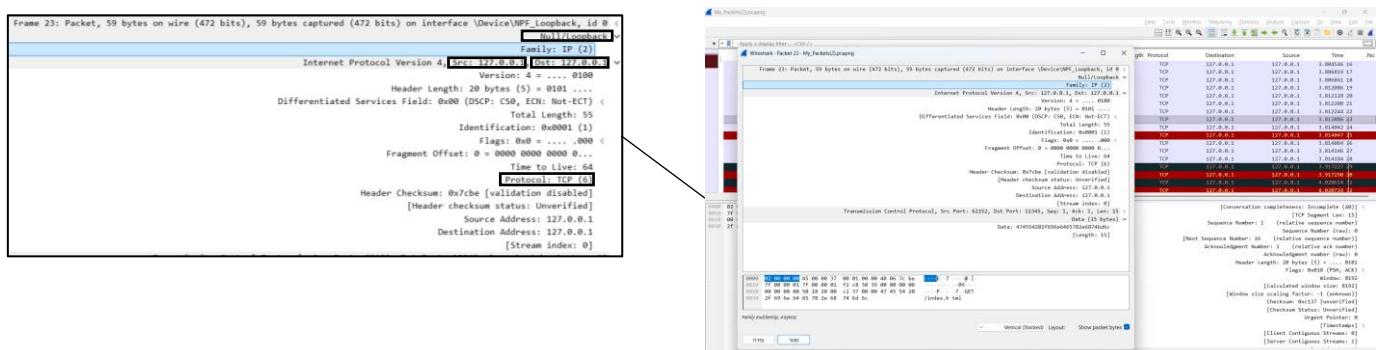
### 1. שכבת הרשות - Internet Protocol Version 4

בעקבות תחילת הרכimos הידני (Encapsulation) שבוצע בסקריפט, נרצה לאמת את הממצאים Wireshark.

צילומי המסך מטור האפליקציה נועדו להראות כי הגדרות שכבת הרשות בקוד תורגמו בהצלחה לכותרת IP תקינה, בדגש על כתובות המקור והיעד-127.0.0.1 ושדה הprotocal.

על מנת לבדוק את המנה הרלוונטיית מתוך כלל תעבורת הרשות, השתמשנו בסינון הבא:

- tcp.flags.push == 1 && tcp.flags.ack == 1 && tcp.port == 12345 •
- tcp.port == 12345 •
- tcp.flags.push == 1 •
- (tcp.flags.push == 1) , (tcp.flags.ack == 1) •
- tcp.flags.ack == 1 •



בצילום המסך ניתן לראות את כותרת IP כפוי שנלכדה בWireshark. ניתן לראות שתחליר הרכimos בסקריפט בוצע בהצלחה.

### 1. כתובות - Addressing

כתובות המקור (Src) וכתובות היעד (Dst) הן 127.0.0.1 (מודגש בתמונה המוגדלת). כתובות זו הינה כתובות המחשב המקומי (localhost) והיא מראה כי המנה הצלילה לעבור מטהלך אחד לשני בהתאם לכתובת המחשב דרך מסק הLoopback, כלומר מהמחשב המקומי לעצמו, ללא יציאה לרשות חיצונית, כפי שהוגדר בקוד הפיתון בפונקציה build\_ip\_header.

## 2. שדה הпротокול - Protocol

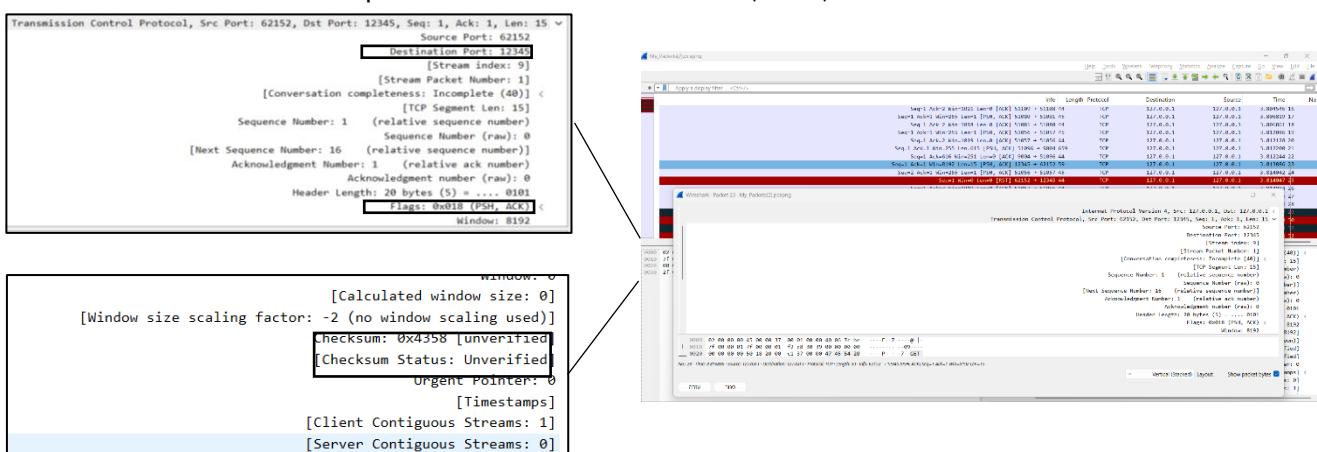
ניתן לראות כי המנה הועברה ב프וטוקול TCP-6 (מודגש בתמונה המוגדלת). נתון זה קריטי לתחילה היכטוס, הוא מסמן למקבל הודעה כי השכבה הבאה שיש לפענחו היא כותרת TCP.

## 3. פרוטוקול Loopback

בשורה העליונה של פירוט המנה ניתן לראות את פרוטוקול loopback (localhost 127.0.0.1) הייאינה משדרת את המנה לכרטיס הרשת הפיזי הפונקציה build\_ip\_header בקוד יוצרת מנת IP אך אינה יוצרת כותרת Ethernet.

## 2. שכבת התעבורה - Transmission Control Protocol

במשך למכצאי שכבת הרשות, שבה ראיינו ששכבה הпротокול הცבע על קיומה של שכבת TCP,icut נבחן את תוכן הכותרת עצמה. נרצה להראות כי הפרמטרים שהזנו לפונקציה build\_tcp\_header תואמים את הפלט שנלכד וכי הדגלים משקפים את סטטוס העברת המידע (Push) כפי שהוגדר בollowat שליחת בקוד.



בצלום המופיע ניתן לראות את כותרת TCP כפי שנלכדה ב-Wireshark. הצלום מראה שהלונגה בסקריפט יושמה בהצלחה:

### • Ports – פורטים

ניתן לראות שפורט היעד (Dst Port) הוא 12345 (מודגש בתמונה). ערך זה תואם בדיקן לשנתנה dst\_port = 12345dst\_port שהוגדר בקובץ הפיתון. פורט המקור (Src Port) הוא 62152 התואם את הפוקודה random.randint בקוד, שנועדה לדמות הקצתת פורט דינמית של לקוח אמיתי.

### • דגלים - Flags

בשדה הדגלים ניתן לראות את הדגלים ACK + PSH. (מודגש בתמונה).  
בחלקו החמישי של הסקרייפט, הרצינו פקודה עם הפרמטר flags=0x18. בኒיגוד לשלב  
יצירת הקשר (Handshake) שבו השתמשנו בדגל (0x02) SYN, כאן מדובר במנת הנושאת  
מידע.

שדה ה-ACK (Acknowledge) מציין שהמנת מכילה אישור קבלה ושדה ה-PSH  
(Push) מסמן למקבל כי יש להעביר את המידע (Payload) מיד לשכבה  
האפליקציה, וכי הקישור תקין.

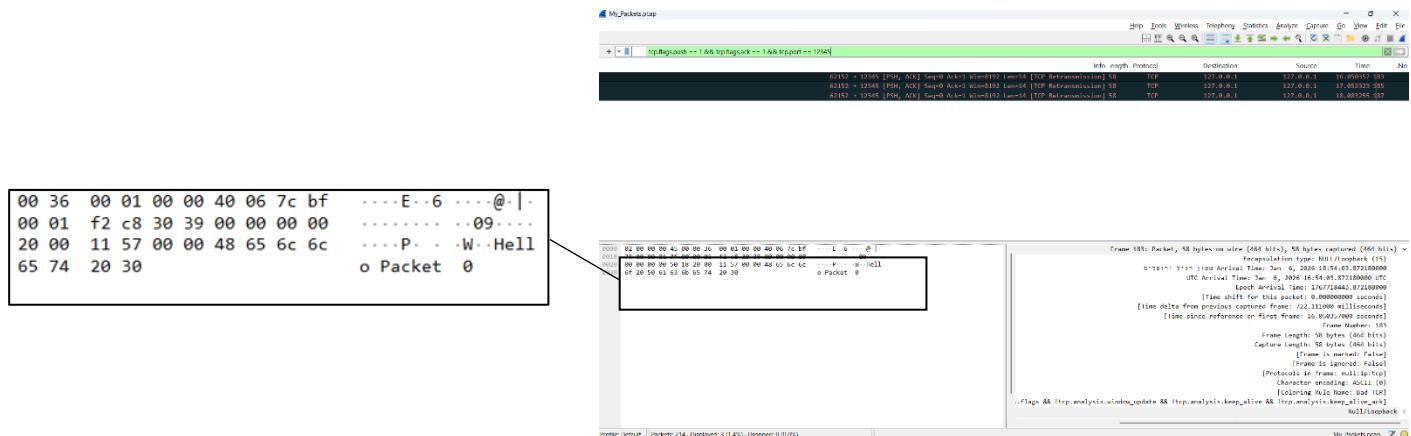
### • סיכום ביקורת – checksum

ניתן לראות את הערך 0x4358 עם הסטטוס Unverified. הסטטוס "לא מאומת" נובע  
מכך שההעבורה נלכדה בLoopback ומערכת הפעלה מבצעת אופטימיזציה ולכן לא  
תמיד מחשבת אותו בזמן הלכידה.

הפונקציה checksum בקוד הפיתון ביצעה את החישוב המתמטי (סכום ואינורסיה של  
16 ביט) כדי למלא שדה זה, כנדרש ב프וטוקול TCP לשמרה על שלמות המידע.

## 3. תוכן המידע – Payload

לאחר שווידאננו כי כותרות ה-IP וה-TCP נבנו ותפקדו כראוי, נרצה לוודא שגם השכבה הפנימית  
מתפקדת כראוי. מטרת צילום המסר הבא היא להוכיח כי המחרוזת שנשלחה מתוך קוד הפיתון  
עבירה את תהליך הרכום, שודרה ברשות, ופוענחה בהצלחה בצד מקבל ללא שיבושים.



בחולונית המידע הגולמי (Packet Bytes) בחתית המסר, ניתן לזהות בבירור את  
הטקסט "Hello" .. בצד שמאל ניתן לראות את הייצוג הקסדיימלי (Hex) של  
תווים אלו.

- תוכן זה תואם את הפקודה `message.encode()` במחברת ההרצה.  
המחרוזת (`String`) הפכה לרצף של בתים (`Bytes`) לפני השילוח ל-`Socket`.  
היכולת של Wireshark להציג את הטקסט בחזרה מוכיחה כי הקידוד והשידור בוצעו  
בצורה תקינה.  
העובדת שהטקסט מופיע בדיק כפי שנשלח, ללא תווים חסרים או מושבעים,  
מהוות הוכחה סופית לכך שכל השכבות הקודומות (TCP, IP) תפקדו כהלאה ושמרו  
על שלמות המנה לאורך המעבר ב-`Loopback`.
- יש לציין, בניסוי זה, לא התבצע שידור פיזי (כבלים או אלחוט), שכן כאמור, כל התעבורה נוהלה  
באופן וירטואלי בתחום המחשב(`Loopback Interface`) ולכן, אין שום ייצוג לשכבה הפיזית.

## חלק 2 – ייצור יישום וניתוח תעבורה:

### הסבר כללי על המערכת ומבנה הקוד:

המערכת היא אפליקציית מסרים מיידיים (Instant Messaging) הפעלתה במודל Client-Server על גבי פרוטוקול התעבורה TCP. המערכת מאפשרת למספר משתמשים להתחבר בו-זמנית, לzechות זה את זה ולנהל שיחות פרטיות בזמן אמת דרך מסך גרפי (GUI). בנוסף, המערכת יודעת לנטר התנטקויות פתאומיות של משתמשים (התנטקויות רשות, ניתוק באמצע שיחה, סגירת צ'אט וכו') להתריע על כך למשתמש ולשרת, ולתפעל את הממשק בהתאם.

## חלק א' - ארכיטקטורת המערכת

- **שכבה התעבורה (Transport):** שימוש ב-Socket-Streams SOCK\_STREAM להבטחת העברת נתונים אמינה, מסודרת ולא שגיאות.
- **שכבה היישום (Application):** המידע נארז בפורמט JSON המאפשר העברה מובנית של נתונים הכוללים את שם השולח, היעד ותוכן הודעה.
- **מקביליות (Concurrency):** השירות מממש מנגן Multi-threading המאפשר לו לנצל "שיחה" נפרדת עם כל לקוח מבלי לחסום חיבורים חדשים.

## חלק ב' - מבנה הקוד

### 1. צד השירות - Server

השרות מהוות את רכיב הליבה של המערכת ואחראי על ניהול הקישוריות, אימוט המשתמשים וניהול ההודעות. כלל תכונות הרשות, ובפרט הודעות על חיבור משתמשים חדשים וניתוק משתמשים קיימים, עוברות דרך השירות. השירות מנהל באופן ריכוזי את האירועים הללו כדי לשמר על סנכרון מלא של רשימת המשתמשים הזמינים לשיחה.

#### **פונקציות מרכזיות ותהליכי:**

- () **start\_server** - פונקציה זו יוצרת את ה-Socket-הראשי ומאזינה לחיבורים נוספים. עבור כל בקשה התחברות, השירות יוצר Thread חדש המריץ את הפונקציה.
- () **handle\_client(conn, addr)** - מנהלת את ה Session-מול ללקוח ספציפי. הפונקציה רושמת את המשתמש במילון הגלובלי, (clients) הממפה בין שם משתמש ל-Socket-שלו, ומנתבת את הלוגיקה לשלבים הבאים: בחירת חבר ושיחת הצ'אט עצמה. בסיום, היא דואגת לניקוי המחשבים ומחיקת המשתמש מהרשימה.

- הפונקציה בודקת האם החבר המבוקש קיים וזמן ברישימת הלקוחות המחברים. אם החבר לא נמצא, היא מנהלת דיווח מול הלקוח המאפשר לו לנסות שוב או להתנתק.
- (chat\_friend(conn, friend\_name, name) - פונקציה להודעות מהלקוח ומנתבת אותן ליעדן. היא כוללת טיפול בפקודות מערכתיות כגון "list" להציג חברים (וטיפול במקרה קצה כמו התנתקות מתאימות של הצד השני לשיחה).
- () Create\_JsonMsg - פונקציית עזר המיישמת את פרוטוקול האפליקציה. כל הודעה נארצת לאובייקט JSON המכיל שדות מוגדרים (שלוח, נמען, תוכן), מה שמאפשר סטנדרטיזציה ומונע שגיאות פענוות.

### IMPLEMENTATION MANAGEMENT FUNCTIONS ARE MADE IN A MEDIUM OF DICTIONARY TYPE.

Implementation of management functions is done in a Hash Table structure (Dictionary type).  
The structure uses the client's name as the key and the socket connection as the value.  
This allows for quick access to the client's socket connection.

clients[name] = conn

The connection is stored under the client's name and is translated to memory internally.  
The connection is stored in memory under the client's name.

In the implementation of the management functions, it is necessary to obtain the connection to the client.  
The connection is obtained by looking up the client's name in the clients dictionary.  
The connection is obtained by looking up the client's name in the clients dictionary.

## 2. Client Side

The client uses the same connection as the server. It is used to support two parallel operations: testing and connection to the server.

### FUNCTIONS FOR CENTRAL SERVERS AND CONNECTIONS:

- (start\_client - The function is the first to implement the creation of the Socket and the connection to the server.

מפעילה את תהליך האזנה ברקע. הולאה הראשית בפונקציה זו אחראית אך ורק על קבלת קלט מהמקלדת ושליחתו לשרת.

- **receive\_messages(sock)** - פונקציה זו ריצה ב-thread Daemon (Daemon) ואחראית על האזנה לתחבורה מהשרת. היא מבצעת קריאה חוסמת (recv), מפענחת את הודעות ה-JSON ומציגה אותן לפ██ן. הדרה זו קריטית כדי שהודעות מחברים יופיעו מיד כשהן מגיעות, ללא תלות באם המשתמש מקליד כרגע או לא.

- **send\_message()** - שולחת את הקלט מהמשתמש לשרת לאחר קידוד ל-UTF-8.

- **friend\_connection(client\_socket)** - מטפלת בשלב הספציפי של בחירת החבר לשיחה מול השרת לפני תחילת הצ'אט.

**מנגנון סנסרין** - הקוד משתמש ב-Event threading כדי לסייע בין ה-thread המאזין ל-thread הכותב. כאשר השרת שולח שאלת המצריכה תשובה ספציפית (כמו "enter yes" לניסיון חזרה), תהליך האזנה מסמן את הדגל, והולאה הראשית יודעת להתייחס לקלט הבא כתשובה למערכת ולא כהודעת צ'אט רגילה.

### 3. סיכום האינטגרציה

המערכת מדגימה יישום מלא של תקשורת נתונים, כאשר השרת מחזיק את תמונה המציב העדכנית ומנתב תעבורה. הליקות פועלם בקצבאות בצורה אסינכרונית. השימוש ב-JSON עוטף את המידע הגלמי ומאפשר לשני הצדדים ("Server" ו-"Client") לדבר באותה שפה, לזהות ניטוקים ולנהל שגיאות בצורה אלגנטית.

## הוראות התקינה והרצתה:

כדי להרייך את המערכת בהצלחה, יש לוודא שביבת הפיתוח כוללת 3 Python ומעלה.

### **1. דרישות מוקדמות:**

אין צורך בהתקנת ספריות חיצונית, המערכת מתחבשת על ספריות מובנות של Python-socket, threading, json, tkinter.

### **2. שלבי ההרצה:**

כדי לדמות סביבת רשות אמיתית עם מספר משתמשים, יש לבצע את השלבים הבאים בסדר זה:

#### **✓ הפעלת השרת:**

- פותח טרמינל והרץ את פקודת הפעלה: python server.py.
- הרשת ידפיס הודעה המאשר שהיא מצויה בכתובת 0.0.0.0 ופורט 10000.

#### **✓ הפעלת לקוח א' :**

- פותח טרמינל חדש והרץ: python client.py.
- במסך הכניסה, הzin כתובת IP (למשל 127.0.0.1), פорт 10000 ושם משתמש.

#### **✓ הפעלת לקוח ב':**

- פותח טרמינל נוסף והרץ שוב את python client.py.
- הzin שם משתמש שונה כדי לאפשר לשרת למפות את המשתמשים בצורה נכונה.

#### **✓ תחילת הצ'אט:**

- לאחר החיבור, הרשת יבקש ממך להזין את שם החבר אותו תרצה לדבר.
- הzin את השם של הלוקה השני ותחל בהתקבות.

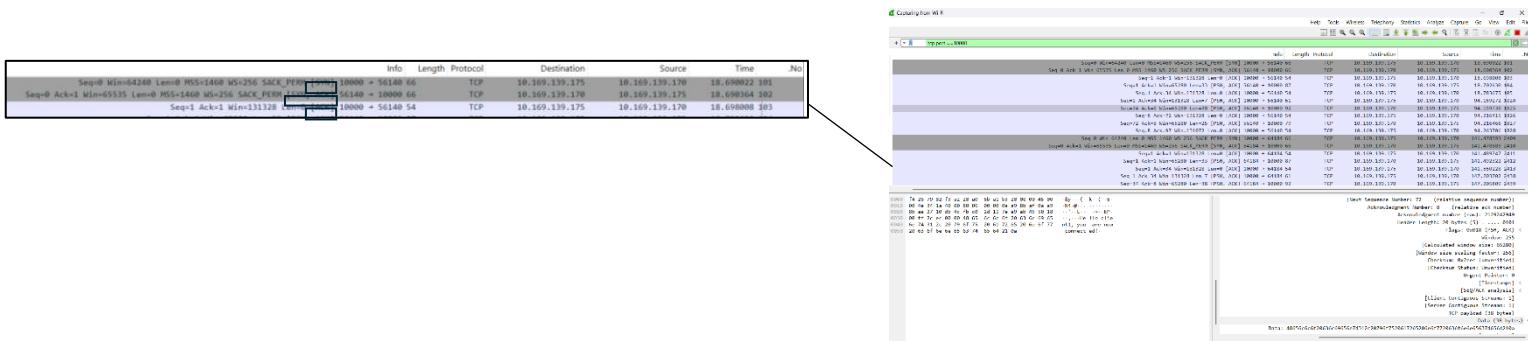
#### **✓ פקודות צ'אט**

- לאחר ההתקבות, ניתן להשתמש בפקודות המינוחות הבאות:
- רשימה: הקולד פקודה זו כדי להציג רשימה בזמן אמת של כל המשתמשים המוחברים כעת.
- יציאה: הקולד פקודה זו כדי להתנתק מהשרת.

## דוגמאות קלט פלט וניתוח ב-Wireshark

בחלק זה הרצינו את השרת, הרצנו שני לקוחות, ביצענו שיחה ביןיהם ולכדנו ב-k-keep-alive את השלבים הבאים:

### [SYN] -> [SYN, ACK] -> [ACK]



בצילום המספר ניתן לראות את תהליך יצירת החיבור בין הלקוח (בעל IP 10.169.139.170) לשרת (בעל IP 10.169.139.175) בפrotocol TCP.

התהליך מורכב משלוש מנות (Packets) עוקבות: חיבור

#### **Client → Server: [SYN]**

הלקוח יוזם את ההתקשרות ושולח בקשה סנכרון לשרת בפורט 10000, חיבור זה מסמן את הרצון לפתח ערוץ תקשורת.

#### **Server → Client: [SYN, ACK]**

השרת מקבל את הבקשת, מאשר אותה באמצעות דגל ה-ACK (Acknowledgment), ושולח בקשה סנכרון משלו (SYN) חוזרת ללקוח.

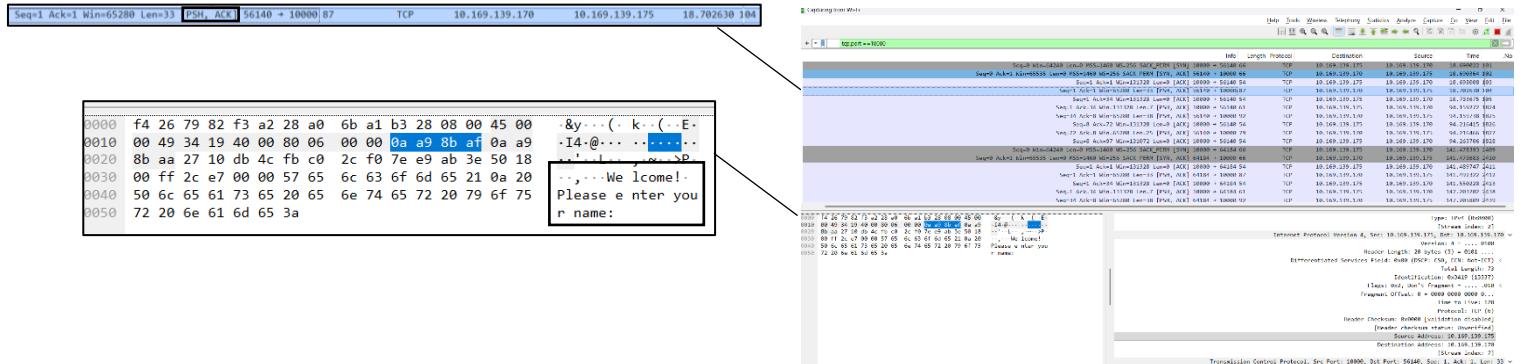
#### **Client → Server: [ACK]**

הלקוח מקבל את אישור השרת ושולח אישור סופי (ACK), עכשו ה-socket פתוח בשני הצדדים מעבר למצב (ESTABLISHED).

מערכת הפעלה מוכנה להעברת הנתונים של אפליקציית הצל'אט.

תהליך זה מבטיח שהצדדים מטונכרים ומוכנים לקלוט מידע ללא איבוד נתונים.

## העברת מידע : (Data Transfer - PSH, ACK)



בתמונה אפשר לראות חבילה מפורט 10000 (השרת) לפורט 56140 (הלקוח).  
56140 הוא פורט זמני שהמערכת של הלקוח בחרה כshaow התחבר.  
אורק המידע (Len) הוא 33 בתים.  
החבילה מכילה מידע מסווג בדגל **[PSH,ACK]**

1. **הראשוני חייבת לכלול דגל ACK (Acknowledgment).** ב-TCP, כל חבילה שנשלחת אחרי שלב לחיצת היד תקינה והוא חלק מהשיטה הקיימת.
2. **PSH (Push):** דגל זה מונחה את פרוטוקול ה-TCP בצד מקבל לא להמתין למילוי ה Buffer אלא להעביר את המידע מיד לשכבות האפליקציה, חינוי באפליקציות זמן אמת כמו צ'אט כדי למנוע השהיות.
3. **תוכן החבילה (Payload) -** בחלק התיכון של המסר ניתן לראות את המידע עצמו כפי ."welcome! Please enter your name:"

התחברות של 6 לקוחות:צד שרת: ✓

```

Server is listening on 0.0.0.0:10000
Client connected from: ('10.169.139.170', 58338)
Added client1 to client list
Client connected from: ('10.169.139.170', 58341)
Added client2 to client list
Client connected from: ('10.169.139.170', 62182)
Added client3 to client list
Client connected from: ('10.169.139.170', 55955)
Added client4 to client list
Client connected from: ('10.169.139.170', 55957)
Added client5 to client list
Client connected from: ('10.169.139.170', 55959)
Added client6 to client list
  
```

בטרמינל של השרת נראה הת לחברות של 6 לקוחות עד client6 אחד אחרי השני. השרת פועל במודול **Multi-threading** כדי לאפשר למספר לקוחות להתחבר ולנהל שיחות במקביל. ניהול המשתמשים המחוברים לשרת מתבצע באמצעות מבנה נתונים מסוג מילון, מבנה זה משמש למיפוי בין זהות המשתמש לבווערץ התקשורת שלו.

**הקצתה Threads:** עבור כל לקוח שמתחבר, השרת יוצר תהליך ייעודי ונפרד. גישה זו מביאה שהשרות הראשי נשאר פנוי לקבלת לקוחות נוספים, בעוד שהתהליכונים מטפלים בתעבורה המידע מול לקוחות הקיימים.

**דיהוי הלקוחות:** למרות שכל הלקוחות מגעים אותה כתובת IP (10.169.139.170), המערכת מבדילה ביניהם באמצעות פורט המקור הייחודי המוקצה לכל חיבור (לדוגמה: 58341, 58338).

**ניהול המשתמשים:** המפתחות הם שמות המשתמשים, כאשר כל שם הוא ייחודי. הערכים הם אובייקטי `socket` מסוג `conn`, המיצגים את חיבור התקשורת הפיזי בין השירות ללקוח.

**תחילת CONNECTION בין שני ל��וחות וניתוק אחד הל��וחות:**

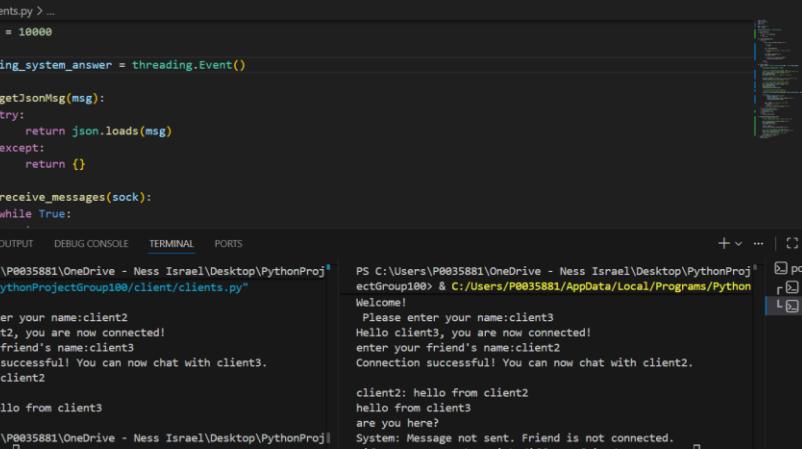
צד שרת: ✓

ונכל לראות ש-client3-client2-i רצים לדבר אחד השני, נפתח צ'אט בינם.

```
client2 wants to talk to client3
client3 wants to talk to client2

View Go Run Terminal Help ← → C Search
Welcome Server.py 9 X
C:\Users\adha>cd Desktop>HIT>Computer Networks>Project>Server.py>connect_friend
1
2 Server module for the chat application.
3 Handles client connections and messaging.
4 ...
5 import socket
6 import threading
7
8 HOST = "0.0.0.0"
9 PORT = 10000
10
11 clients = []
12
13 def connect_friend(conn, name):
14     """Handles the logic for connecting two friends."""
PROBLEMS DEBUG CONSOLE OUTPUT PORTS TERMINAL
PS C:\Users\adha> & C:/Users/adha/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/adha/Desktop/HIT/Computer Networks/Project/Server.py" > connect_friend
Client connected from: ('10.169.139.170', 58338)
Added client1 to client list
Client connected from: ('10.169.139.170', 58341)
Added client2 to client list
Client connected from: ('10.169.139.170', 52182)
Added client3 to client list
Client connected from: ('10.169.139.170', 55995)
Added client4 to client list
Client connected from: ('10.169.139.170', 55997)
Added clients to client list
Client connected from: ('10.169.139.170', 55999)
Added client5 to client list
Client connected from: ('10.169.139.170', 58340)
Added client6 to client list
client2 wants to talk to client1
client1 wants to talk to client2
client1 wants to talk to client5
client2 wants to talk to client3
client3 wants to talk to client1
Client disconnected from ('10.169.139.170', 58338)
Removed client2 from client list
Socket closed for ('10.169.139.170', 58341)
```

צד לקוח: ✓



```
client > clients.py > ...
7 PORT = 10000
8
9 waiting_system_answer = threading.Event()
10
11 def getJsonMsg(msg):
12     try:
13         return json.loads(msg)
14     except:
15         return {}
16
17 def receive_messages(sock):
18     while True:
...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + x | x
PS C:\Users\P0035881\OneDrive - Ness Israel\Desktop\PythonProj1\Desktop\PythonProjectGroup100> client\clients.py"
Welcome!
Please enter your name:client2
Hello client2, you are now connected!
enter your friend's name:client3
Connection successful! You can now chat with client3.
hello from client2

client3: hello from client3
exit
PS C:\Users\P0035881\OneDrive - Ness Israel\Desktop\PythonProj1\Desktop\PythonProjectGroup100> ]
```

PS C:\Users\P0035881\OneDrive - Ness Israel\Desktop\PythonProj1\Desktop\PythonProjectGroup100> & C:/Users/P0035881/AppData/Local/Programs/Python>Welcome!
Please enter your name:client3
Hello client3, you are now connected!
enter your friend's name:client2
Connection successful! You can now chat with client2.

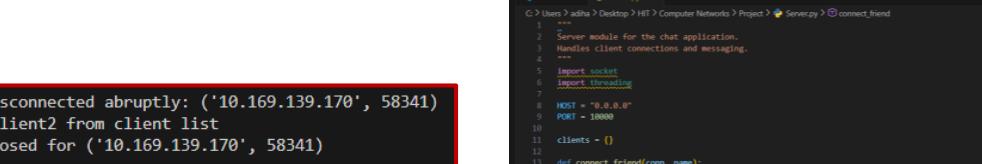
client2: hello from client2
hello from client3
are you here?
System: Message not sent. Friend is not connected.
If you want to chat with different friend enter yes: ]

נראה את הצלאת בין הלקחות:

client2 שלוח שלום ואחריו client3 שלוח שלום.

client2 כותב exit ו יצא מהשינה.

client3 שולח הודעה are you here? לא נשלחת לclient2 כי הוא כבר מנותק.  
client3 מקבל הודעה מערכת שההודהה שלו לא נשלחה כי החבר שלו לא מחובר, ובמידה והוא  
רוצה לדבר עם חבר אחר שיכתוב yes.



```
Client disconnected abruptly: ('10.169.139.170', 58341)
Removed client2 from client list
Socket closed for ('10.169.139.170', 58341)

Client disconnected abruptly: ('10.169.139.170', 58341)
Removed client2 from client list
Socket closed for ('10.169.139.170', 58341)

Client disconnected abruptly: ('10.169.139.170', 58341)
Removed client2 from client list
Socket closed for ('10.169.139.170', 58341)

Client disconnected abruptly: ('10.169.139.170', 58341)
Removed client2 from client list
Socket closed for ('10.169.139.170', 58341)
```

בשים socket נסגר, הזיכרון במחשב וה포רט משוחררים לטובת משתמשים אחרים. ברגע שהלקוח סגור את החלון בפותאומיות הרשות מוחק את המשתמש מהmillion כדי למנוע שימושו למשתמש שאינו זמין יותר.

## :Wireshark

tcp.port == 10000	Length	Protocol	Destination	Source	Time	No.
<b>Info</b>						
10000 + 62182 [PSH, ACK] Seq=151 Ack=15 Win=65280 Len=29	83	TCP	10.169.139.170	10.169.139.175	496.040667	13999
10000 + 58341 [ACK] Seq=151 Ack=33 Win=65280 Len=0	54	TCP	10.169.139.170	10.169.139.175	496.087664	14000
62182 + 10000 [ACK] Seq=15 Ack=180 Win=65280 Len=0	54	TCP	10.169.139.175	10.169.139.170	496.092282	14001
62182 + 10000 [PSH, ACK] Seq=15 Ack=180 Win=65280 Len=18	72	TCP	10.169.139.175	10.169.139.170	513.762089	14098
10000 + 58341 [PSH, ACK] Seq=151 Ack=33 Win=65280 Len=29	83	TCP	10.169.139.170	10.169.139.175	513.762530	14099
10000 + 62182 [ACK] Seq=180 Ack=33 Win=65280 Len=0	54	TCP	10.169.139.170	10.169.139.175	513.805307	14100
58341 + 10000 [ACK] Seq=33 Ack=180 Win=65280 Len=0	54	TCP	10.169.139.175	10.169.139.170	513.850622	14102
<b>58341 + 10000 [RST, ACK] Seq=33 Ack=180 Win=0 Len=0</b>	54	TCP	10.169.139.175	10.169.139.170	653.945167	16248
62182 + 10000 [PSH, ACK] Seq=33 Ack=180 Win=65280 Len=19	73	TCP	10.169.139.175	10.169.139.170	856.493138	20741
10000 + 62182 [PSH, ACK] Seq=18 Ack=52 Win=65280 Len=104	158	TCP	10.169.139.175	10.169.139.175	856.493338	20742
62182 + 10000 [ACK] Seq=52 Ack=284 Win=65280 Len=0	54	TCP	10.169.139.175	10.169.139.170	856.544713	20744
62182 + 10000 [PSH, ACK] Seq=52 Ack=284 Win=65280 Len=3 [TCP PDU reassembled in 2113..	57	TCP	10.169.139.175	10.169.139.170	886.380865	20888
10000 + 62182 [PSH, ACK] Seq=284 Ack=55 Win=65280 Len=25	79	TCP	10.169.139.170	10.169.139.175	886.381106	20889
62182 + 10000 [ACK] Seq=55 Ack=309 Win=65280 Len=0	54	TCP	10.169.139.175	10.169.139.170	886.432611	20890
<b>re (1264 bits), 158 bytes captured (1264 bits) on interface \Device\NPF_{3D5F4A4C-F</b>						
<b>:#0:6:b1:b3:28, Dst: Intel_82:f3:a2 (4:26:79:b3:f3:a2)</b>						
<b>.169.139.175, Dst: 10.169.139.170</b>						
<b>ort: 10000, Dst Port: 62182, Seq: 100, Ack: 52, Len: 104</b>						
<b>0000  f4 26 79 82 f3 28 a8 6b al b3 28 00 45 00 .&amp;y-(.-k-(.-E</b>						
<b>0001  00 34 7a 48 00 00 00 00 0a 89 af 0a 0f .-4@-.-.-----</b>						
<b>0020  8b aa 27 18 fe ec cb 3b 0c cc 32 fd 4e 50 18 .-2-NP-</b>						
<b>0030  ff ff 2d 00 00 53 79 73 73 65 6d 3a 20 4d 65 .-.Sy stem: Me</b>						
<b>0040  73 73 61 65 20 66 67 74 20 73 65 66 74 26 message n't sent.</b>						
<b>0050  46 72 69 65 66 20 69 73 20 66 67 74 20 63 Friend i s not co</b>						
<b>0060  66 65 63 74 65 64 2e 8a 20 69 66 20 79 67 65 nected . if you</b>						
<b>0070  20 77 61 66 74 20 74 26 63 68 61 74 20 77 69 want to chat wi</b>						
<b>0080  74 68 20 64 69 66 72 65 66 74 20 66 72 69 65 th diff'rent fri</b>						
<b>0090  64 20 65 66 74 65 72 20 79 65 73 3a 20 end yes:</b>						

השרת שלח הודעה ל-**client3** שהתנתק בפתאומיות.  
השרת זיהה ש-**client2** לא נמצא במילון המחברים (clients).  
השרת שלח הודעה מערכת ל-**client3** שסבירה את השגיאה ומציעה למשתמש להקליד yes כדי  
לבחור חבר חדש.

58341 -> 10000 [RST, ACK] Seq=33 Ack=180 Win=0 Len=0 54 TCP 10.169.139.175 10.169.139.170 653.945107 16248

**דגל RST (Reset)** – מעיד על סיום פתאומי/מיידי של חיבור-הTCP מצד הלקוח, ולא על סגירה "משמעותית" ותקינה. הטרמינל מצד הלקוח נסגר בלי לבצע את תהליך הסגירה הסטנדרטי באמצעות לחיצת היד (handshake) של FIN. כתוצאה לכך, מערכת הפעלה שלחה חבילת Reset כדי לשחרר מיד את משאבי הלקוח מצד השרת.

## שיחות עם משתמש אחר:

```
if you want to chat with diffrent friend enter yes:yes  
enter your friend's name:client3  
Connection successful! You can now chat with client3.
```

```
client > clients.py > ...
client > PORT = 10000
client > waiting_system_answer = threading.Event()
client >
client > def getJsonMsg(msg):
client >     try:
client >         return json.loads(msg)
client >     except:
client >         return {}
client >
client > def receive_messages(sock):
client >     while True:
client >         try:
client >             msg = sock.recv(1024).decode('utf-8')
client >         except:
client >             print("Connection closed by server")
client >             break
client >
client >         if msg == "Client connected":
client >             print("Client connected")
client >         elif msg == "Client disconnected":
client >             print("Client disconnected")
client >         else:
client >             print("Received message:", msg)
client >
client >             if msg == "Enter your name:client1":
client >                 name = input("Please enter your name:client1")
client >                 print("Hello", name, ", you are now connected!")
client >             elif msg == "Enter your friend's name:client2":
client >                 friend_name = input("Enter your friend's name:client2")
client >             elif msg == "Error: User client2 is not connected. You cannot send messages. If you want to chat with different friend enter yes:yes":
client >                 print("Error: User client2 is not connected. You cannot send messages.")
client >                 if input("If you want to chat with different friend enter yes:yes") == "yes":
client >                     friend_name = input("Enter your friend's name:client3")
client >             elif msg == "Connection successful! You can now chat with client3.":
client >                 print("Connection successful! You can now chat with client3.")


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

השרת לא מוצא אותו ברשימה אז הוא שלוח הודעה שגיאה ל-client1 ומחכה שהמשתמש יקליד "yes" כדי לנסוח שוב.

כאשר מקבל "yes", הולאה True מתחילה מחדש, ומאפשרת למשתמש להזין "client3", שנמצא בהצלחה במילון — וכך נוצר חיבור הצ'אט.

אפשרה בכל המשתמשים המחברים:

```
list
System: Connected users: client1, client3
```

```
clients.py M
client > clients.py > ...
7   PORT = 10000
8
9   waiting_system_answer = threading.Event()
10
11  def getJsonMsg(msg):
12      try:
13          return json.loads(msg)
14      except:
15          return {}
16
17  def receive_messages(sock):
18      while True:
19          try:
20              msg = sock.recv(1024).decode('utf-8')
21
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\P0035881\OneDrive - Ness Israel\Desktop\PythonProjectGroup100>
on14\python.exe "c:/Users/P0035881/OneDrive - Ness Israel/Desktop/PythonP
Welcome!
Please enter your name:client1
Hello client1, you are now connected!
enter your friend's name:client2
Error: User 'client2' is not connected. You cannot send messages.
if you want to chat with diffrent friend enter yes:yes
enter your friend's name:client3
Connection successful! You can now chat with client3.
list
System: Connected users: client1, client3
```

ונכל לראות שהמשתמש client1 שולח את המילה list, השרת מקבל אותה בתוך הלולאה של chat\_friend לפני העברת הודעה לחבר.

השרת מזזהה שהקלט הוא פקודה באמצעות הבדיקה: if data\_d.lower() == ("list"). אם התנאי מתקיים, הקוד ניגש למילון הגלובלי clients, שמשמש כ"זיכרון" של השרת לכל החיבורים הפעילים () מכיל את כל שמות המשתמשים).

הפקודה (...).join(", ") לוקחת את המפתחות של המילון ומחברת אותם לחרוזת אחת (למשל ("client1, client3") כדי שייהיה קרייא.

לאחר מכן השרת עוטף את הרשימה בהודעת System בפורמט JSON ושולח אותה חוזה וקלקוק שביבקש (conn.sendall).

ההודעה list לא תישלח לחבר בצוות הפקודה continue שגורמת לוולאה להתחיל מחדש.

### **חלק 3 - תיאור שימוש במבנה מלאכותית**

במהלך העבודה על הפרויקט, נעזרנו בא (Gemini, Chat GPT) על מנת לבדוק את נכונות הקוד, לשפר את הקוד ולהשוו עלי מקרי קצה נוספים שיכולים לקרות במערכת מסווג זה, על מנת להגיע לפרויקט ברמה גבוהה ומקצועית.

נעזרנו במבנה המלאכותית בכמה שלבים:

- יצירת נתוני הקלט(CSV) :**

ב חלק א' של הפרויקט, נעזרנו ב AI כדי ליצור טבלה המדמה תעבורת HTTP/Application TCP/IP טכנית. AI יצר עבורנו קובץ CSV עם נתונים פיקטיביים, בו השתמשנו במהלך החלק הראשון של העבודה.

- זיהוי ופתרון שגיאות( Debugging ) :**

במהלך כתיבת הקוד נתקלנו בעוויות טכניות, כמו חסימת השירות נחסם (Blocking) ולא הצליח לקבל ליקוח חדש בזמן שדיבר עם לקוח קיים. בעזרתו של AI הבנו איך להטעין מנגנון של Multi-threading שבו כל לקוח מקבל תהליכי נפרד בשרת, מה שפתר את הבעיה ואפשר עבודה במקביל.

- בנייה הלוגיקה של השירות והלקוח :**

בעזרת תשאול AI, הבנו אילו מקרי קצה עלולים לקרות ולהשוו על איות המערכת. בהתאם, הוספנו טיפולים ייעודיים בקוד שיעודים לתמוך במקרים אלו כפי שפורט קודם לכך.

- עיצוב הממשק הגרפי (GUI) :**

השתמשנו בכלים כדי לעצב ממשק למשתמש.

#### **דוגמאות לפרויקטים:**

- lezirat.netonim :**

"צורך לי קובץ CSV עם 10 שורות של הודעות המדומות תעבורת אפליקציה. תכלול את העמודות הבאות: עמודה של מזהה הودעה, שם הפרטוקול, פורט מקור ופורט יעד, תוכן ההודעה וזמן שליחה".

- lptorun.shgiasot :**

"בניית שירות TCP בפייתון, כשאני מנתח לחבר לקוח שני הוא לא מצליח להתחבר עד שהלקוח הריאון מתנתך. איך אני משתמש ב threading-CDI שהשרת יטפל בכמה אנשים במקביל מבלי להיתקע?"

- logika.sh.servet.mkeri.kaza :**

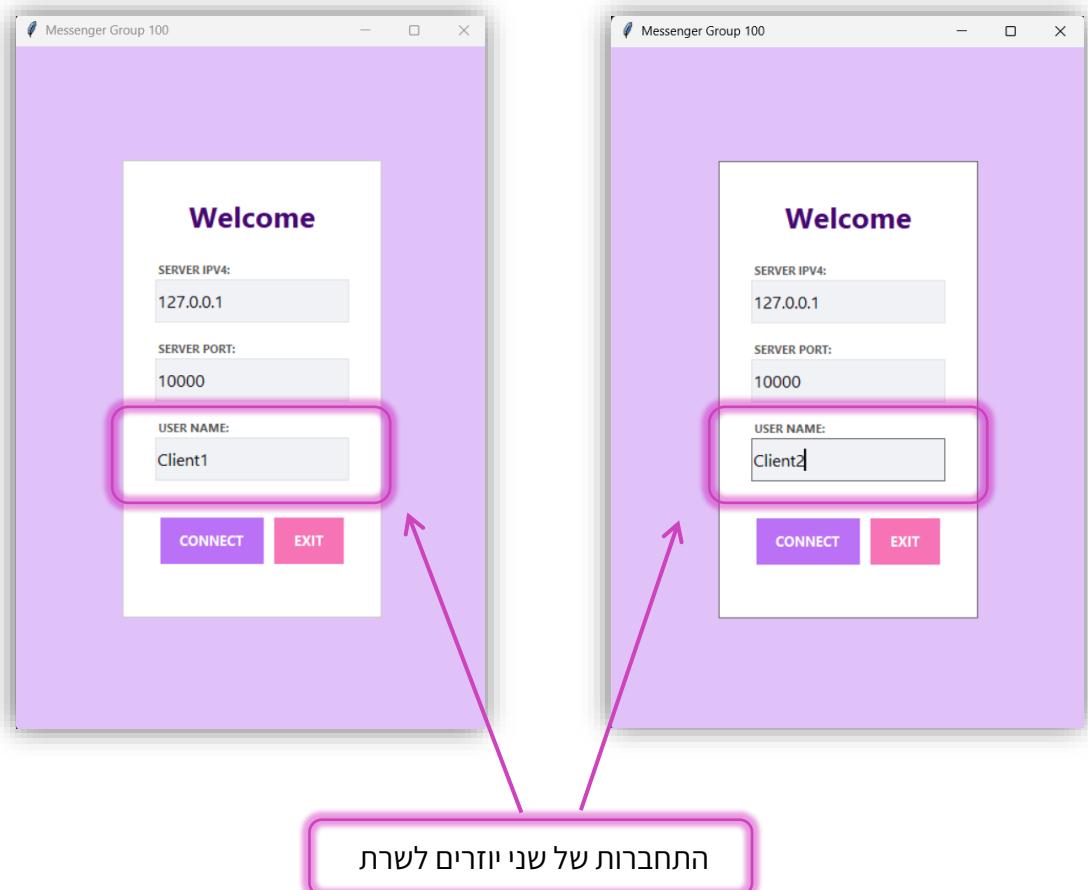
"מימשנו קוד התומך בהעברת מידע בתוצאות השירות ללקוח. באילו מקרים פתאומיים עשויים המשתמשים להיתקל שכדי להסייע לתמיכה בקוד שלנו?"

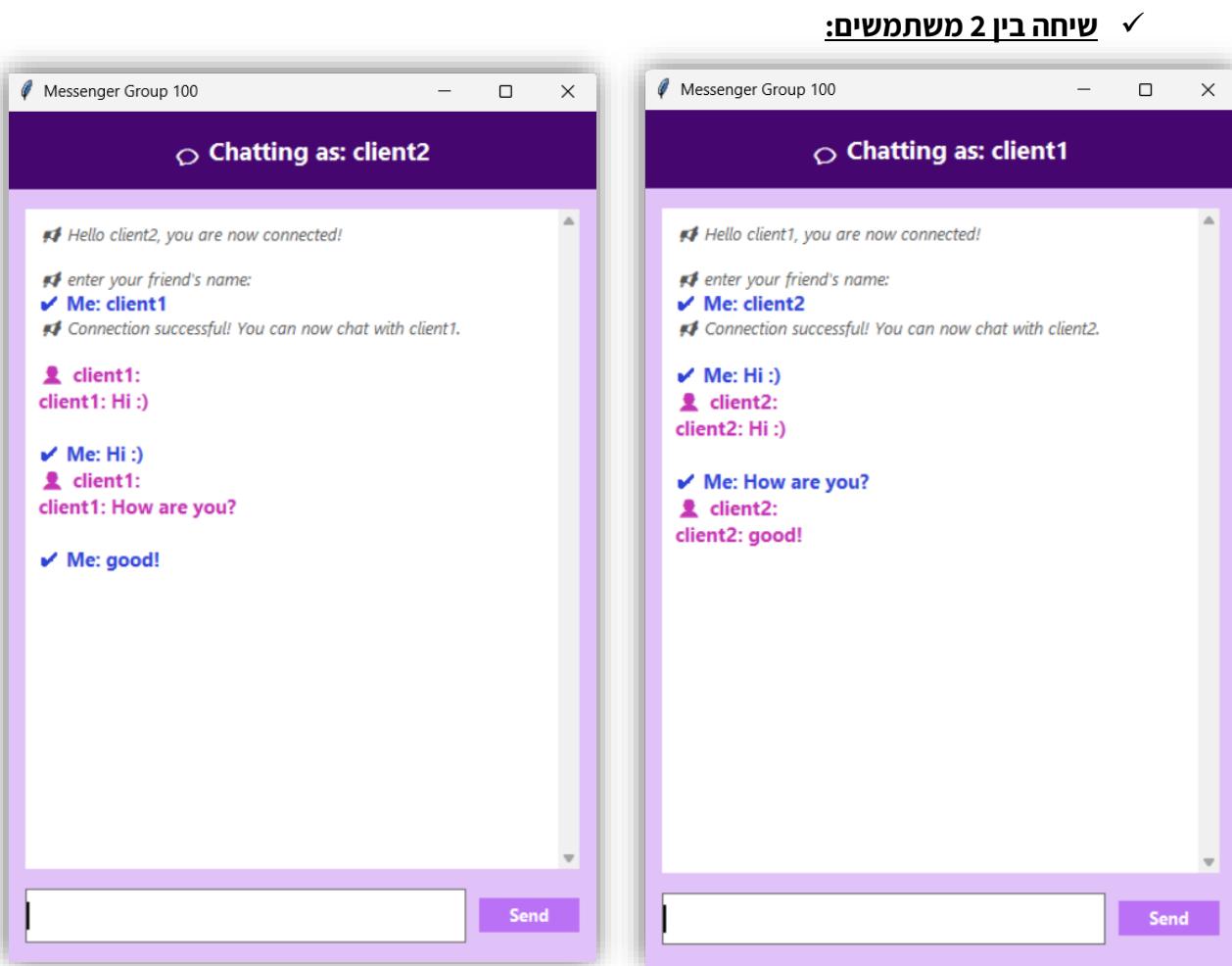
## פרק 4 – יצרת וGU למשתמש - בונוס

בחלק זה ייצרנו ממשק למשתמש שייהנו נוח ויזואלי.  
נעזרנו בבינה מלאכותית על מנת ליצור את המשapk הנכון ביותר אשר לא מסביר את המשתמש ובתוכו יש הוראות ברורות לתפעול והסבירים במידה וישנים ניתוקים.

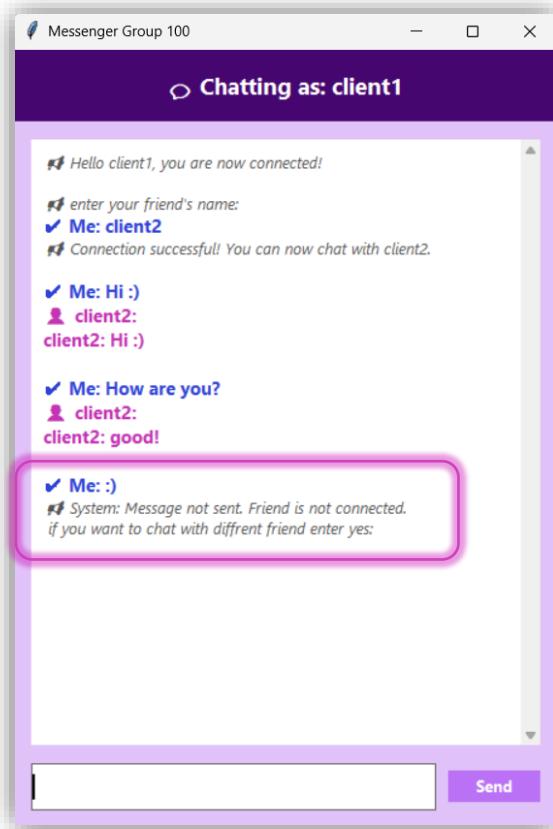
### ✓ מסך כניסה:

- כולל 3 פרמטרים למילוי ע"י המשתמש ותלויים במס' המחשבים בשימוש:  
 1. SERVER IPV4 – כתובת IP פנימית (במידה וישנם 2 מחשבים, מכינים את כתובת IP שמריץ את השירות).  
 2. SERVER PORT – 10000 (מתאים לשרת)  
 3. USER NAME – שם המשתמש לזיהוי בשרת.





✓ הודעת שגיאה במצב של ניתוק:



**-זריקת שגיאה מהשרת-**

המשתמש השני הtentתק ולכך  
ההודעה לא נשלחה.

במידה והמשתמש רוצה לפנות  
למשתמש אחר הוא יצטרך לכתוב  
את שמו והחיבור יבוצע.