

## חלק ראשון

### סעיף א

#### **א. דרך יצירת קובץ CSV עם הודעות שכבת היעשום**

בשלב הראשון, יצרנו קובץ הנתונים באמצעות בניית מלאכותית והוא מייצג את המידע שאוטו אנו רוצים להעביר ברשות. הקוד עושה שימוש בספרייה pandas כדי לטעון ולעבד מידע זה.

יצרנו קובץ CSV המכיל את ה הודעות שכבת היעשום. המבנה של הקובץ כולל עמודות (בגון timestamp, msg\_id, message) טענו את הקובץ באמצעות הפקודה pd.read\_csv(). בשלב זה מדמה את **שכבת היעשום (Application Layer)** במודל OSI זהו המידע הטהור אותו משתמש רוצה להעביר, לפני שנוסף לו בותרונות תקשורת בלבד.

#### **ב. תיאור והסבר של תהליכי אריזה של מנוגט(Encapsulation)**

הקוד בונה ידנית את הכותרות (Headers) בעזרת הספרייה struct ומשתמש במחלקה RawTcpTransport .

תהליכי האריזה (Encapsulation) בוצע באופן ידני בקוד כדי להמחיש כיצד נבנית חבילה מידע:

##### **1. שכבת התעבורה:(Transport - TCP)**

הfonקציה build\_tcp\_header בקוד מייצרת את בותרת-TCP. היא אורצת (באמצעות struct.pack) את פורט המקור ואת פורט היעד , מספרי הרצף (Flags), והדלים (Sequence Number). בדוגמה שלנו השתמשנו בדלים PSH, מה שמורה לצד השני לעבד את המידע מיידית.

##### **2. שכבת הרשת:(IP - Network)**

הfonקציה build\_ip\_header עוטפת את ה-TCP Segment שיצרנו. היא מוסיפה את כתובת-ה-IP של המקור והיעד (במקרה זה Localhost - 127.0.0.1) ומגדירה את הпрוטוקול ב-TCP

למעשה הקוד לוקח את ההודעה משכבה האפליקציה ועוטף אותה ב-header המתאים.ם.

## ג. תיאור והסבר של תהליך הלכידה (Capture)

1. ביעון שהקוד הוגדר לשלוח מכתובת 127.0.0.1 אל 127.0.0.1 בחרנו בתווך ה-. Loopback Wireshark להקשיב לממשק jupyter שקייבלנו.
2. צירפנו את קובץ ה CSV אל מחברת wireshark שקייבלנו.
3. הפעילנו את לכידת הפקודות ב-wireshark
4. הריצנו את הקוד שיצר הדמיה של שליחת פקודות.
5. לבסוף כיבינו את לכידת הפקודות וניתחנו את הלכידה.

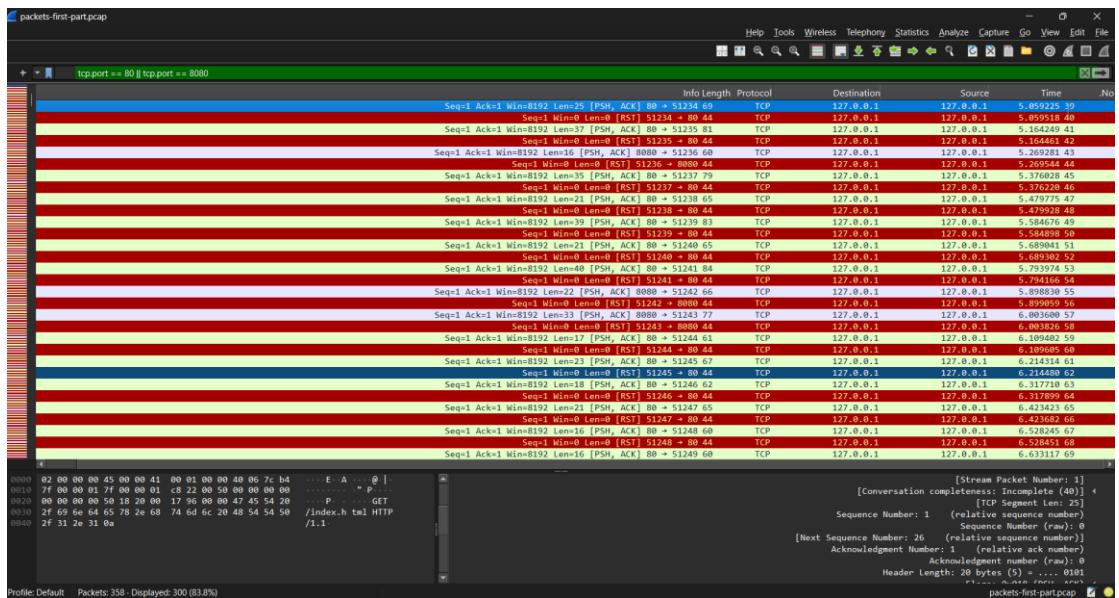
## ד. תיאור והסבר של תעבורת שנלקדה ב-Wireshark

בתרומות הבאות ניתן לראות את תעבורת הרשות שנוצרה על ידי סקריפט ה-`http_input11.py`.  
הסקריפט קרא את קובץ ה-CSV (02\_http\_input11.csv) וייצר עبور כל שורה  
פקטה ייחודית באמצעות Sockets Raw. ניתן להבחין כי תוכן ההודעות (Payload)  
זהה לטקסט המופיע בקובץ הקלט

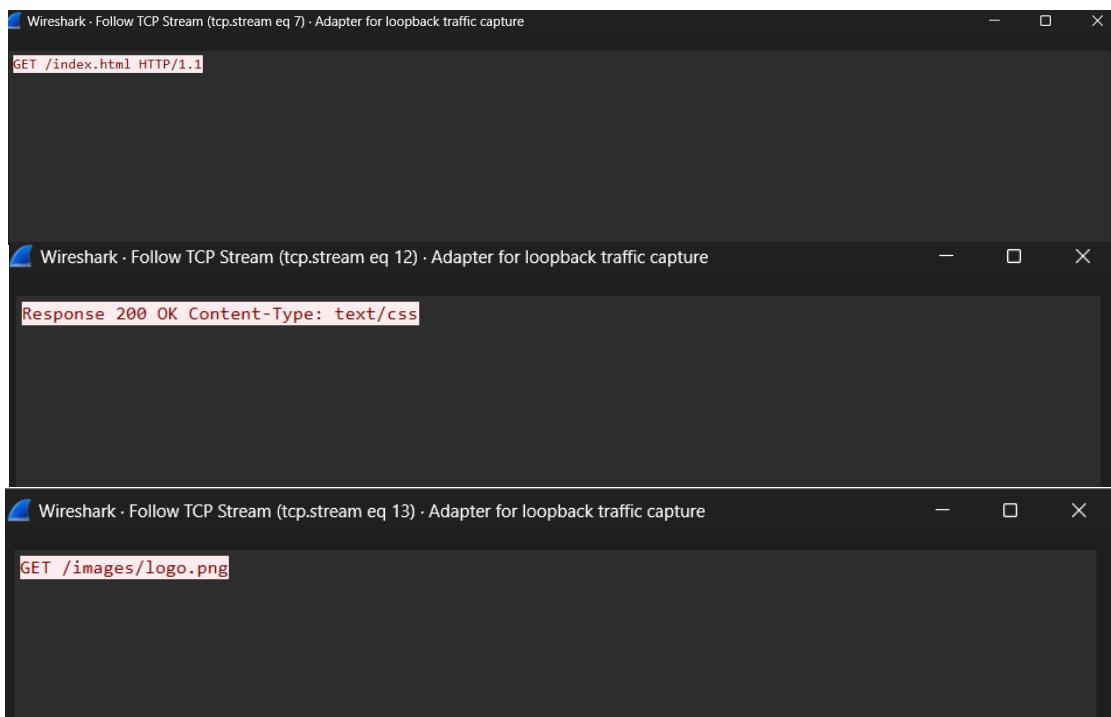
### תוכן הקובץ:

E	D	C	B	A
timestamp	message	dst_port	src_port	app_protocol
				1
				2
09/12/2025	GET /index.html HTTP/1.1	80	51234	HTTP
				3
09/12/2025	Response 200 OK Content-Length: 1024	80	51235	HTTP
				4
09/12/2025	POST /api/login	8080	51236	HTTP
				5
09/12/2025	Response 302 Found Location: /home	80	51237	HTTP
				6
09/12/2025	GET /styles/main.css	80	51238	HTTP
				7
09/12/2025	Response 200 OK Content-Type: text/css	80	51239	HTTP
				8
09/12/2025	GET /images/logo.png	80	51240	HTTP
				9
09/12/2025	Response 200 OK Content-Type: image/png	80	51241	HTTP
				10
09/12/2025	GET /user/profile/123	8080	51242	HTTP
				11
				12
				13
				14
				15
				16
				17
				18
				19
				20

## התעבורה הכוללת שנלכדה:



## תוכן הפקודות שלCDCN:



ניתן להבחן כי عمودת **Source Port** משתנה מפקטה לפקטה בהתאם לקובץ הקלט, בעוד פורט היעד משתנה בהתאם לסוג הבקשה.

בנוסף הנתונים נקראו מהקובץ כמחרוזות (**String**), עברו קידוד (**Encoding**) לבינארי, ושורשו לסוף ה-**TCP Header** כ-**Payload**.

## חלק שני

### סעיף א: הסבר כללי על המערכת ומבנה הקוד

המערכת היא אפליקציית צ'אט המבוססת על ארכיטקטורת (Client-Server) ומשתמשת ב프וטוקול TCP להעברת הודעות אמינה. השירות משתמש במתוך מרכז: כל ההודעות נשלחות מהלך לשרת, והשרת מנtab אותן לעד המתאים.

#### 1. צד השירות (server.py)

השירות בתוכן ב-`Python` תוך שימוש בספריות `socket` לתקשורת ו-`threading` לטיפול במקביליות.

##### **מבנה הנתונים והניהול:**

- השירות מחזיק מילון גלובלי בשם `clients`, הממחיה בין **שם המשתמש** (Key) לבין אובייקט ה-**Socket** הפעיל של (Value). מבנה זה מאפשר שיליפה חיבור של משתמש ספציפי לצורך שליחת הודעה פרטית ברגע.

##### **רכיבים מרכזיים בקוד השירות:**

- אתחול והאזנה (start\_server):** הפונקציה יוצרת Socket מסוג TCP (SOCK\_STREAM) ומבצעת קישור (Bind) לכתובת 0.0.0.0 ולפורט 42069. השירות נבנש למצוות האזנה (listen) וממתין לחיבורים נוספיםים בולאה אינסופית. ברגע שהלקוח מתחבר (accept), השירות ייצור עבورو **Thread** (תהליך) נפרד ו מעביר את הטיפול בו לפונקציה `handle_client`. שיטה זו מבטיחה שהשירות לא ייחסם ו יוכל לשירות לקוחות במקביל.

- **פרוטוקול ההתחברות (Handshake):** באשר לקוח מתחבר, השרת שולח לו הודעה בקשה "LOGIN\_REQUEST". הלקוח בתגובה שולח את שם המשתמש שלו. השרת רושם את המשתמש במילון-clients ומעדכן את כל שאר המחברים על ידי שליחת רשימת המשתמשים המעודכנת (USERS\_LIST).
- **ניתוב הודעות (handle\_client):** זהה הלוגיקה המרכזית הרצה עבור כל לקוח. הפונקציה מאזינה להודעות כניסה. המערכת עובדת לפי פרוטוקול טקסטואלי מוסכם:
  - הודעות פרטיות מגיעות בפורמט ."TargetUsername:MessageContent"
  - השרת נפרק את ההודעה, בודק אם משתמש היעד קיים במילון-clients (TargetUsername)
  - אם היעד קיים השרת שולח לו את ההודעה בפורמט: ."SenderName: MessageContent"
  - אם היעד אינו מחובר נשלחת הודעה שגיאה חזרה לשולח.
- **טיפול בניתוקים:** אם מתקבלת הודעה ריקה (המuida על סיגורת חיבור) או מתרחשת שגיאה, השרת מסיר את המשתמש מהמילון, סגור את ה-Socket ושולח עדכון לכל שאר המשתמשים שהליך התנתק.

## 2. צד הלקוח - (client.py) היבטי תקשורת

- למרות שהליך כולל משך גרפי, ליבת התקשרות מופרדת ומנווהת במקביל ל-gui
- **יצירת החיבור:** הלקוח יוצר Socket ומתחבר לבתוות ה-IP והפורט של השרת.
  - **האזנה להודעות (Threading):** מיד לאחר החיבור הלקוח מפעיל תהליך (Thread) נפרד המריץ את הפונקציה receive\_messages. תהליכון זה אחראי להמתין להודעות מהשרת מוביל "להקפיא" את המשך למשתמש.

• **עיבוד הודעות נכונות: הלקוח מפענח את המידע המגיע מהשרת וմבדיל בין**

**סוגי הודעות:**

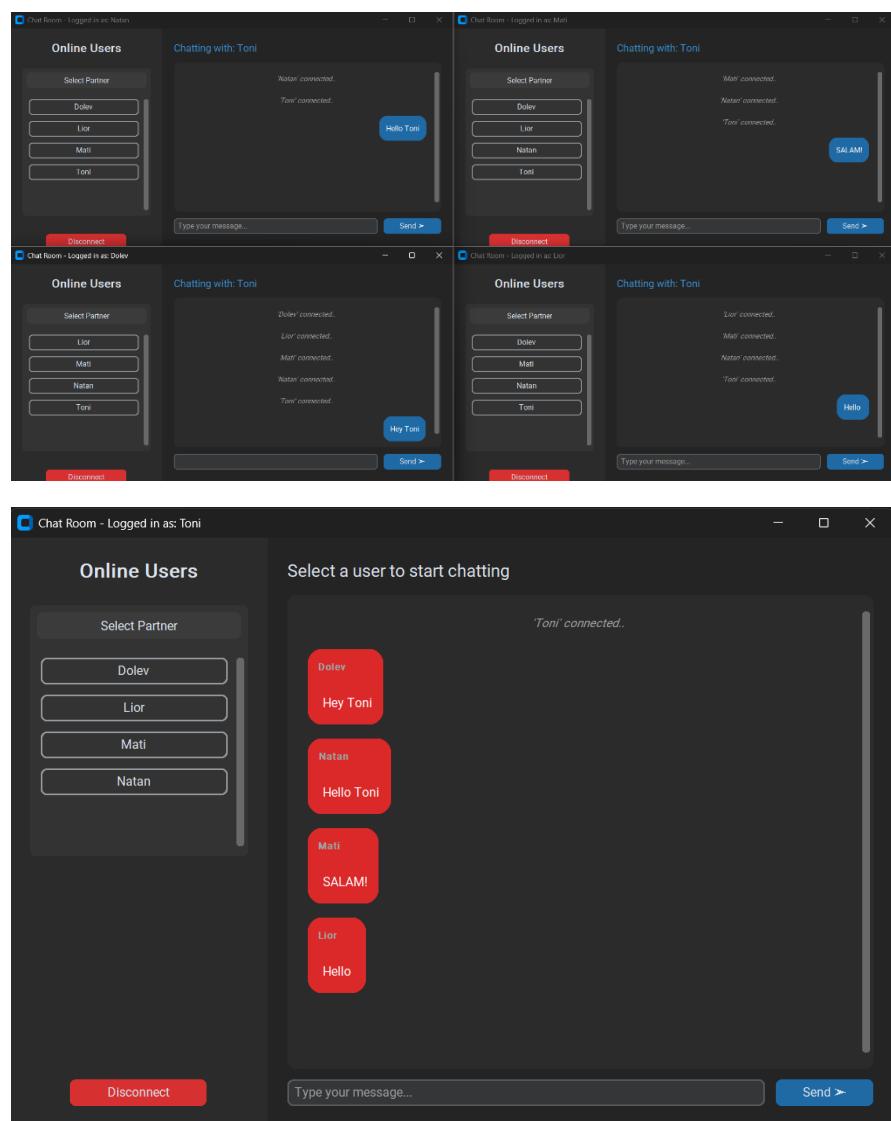
- . LOGIN\_REQUEST: ○
- . USERS\_LIST: ○
- . הודעות צ'אט רגילים מוצגות בחלון השיחה. ○

**סעיף ב- הוראות התקנה והרצה :**

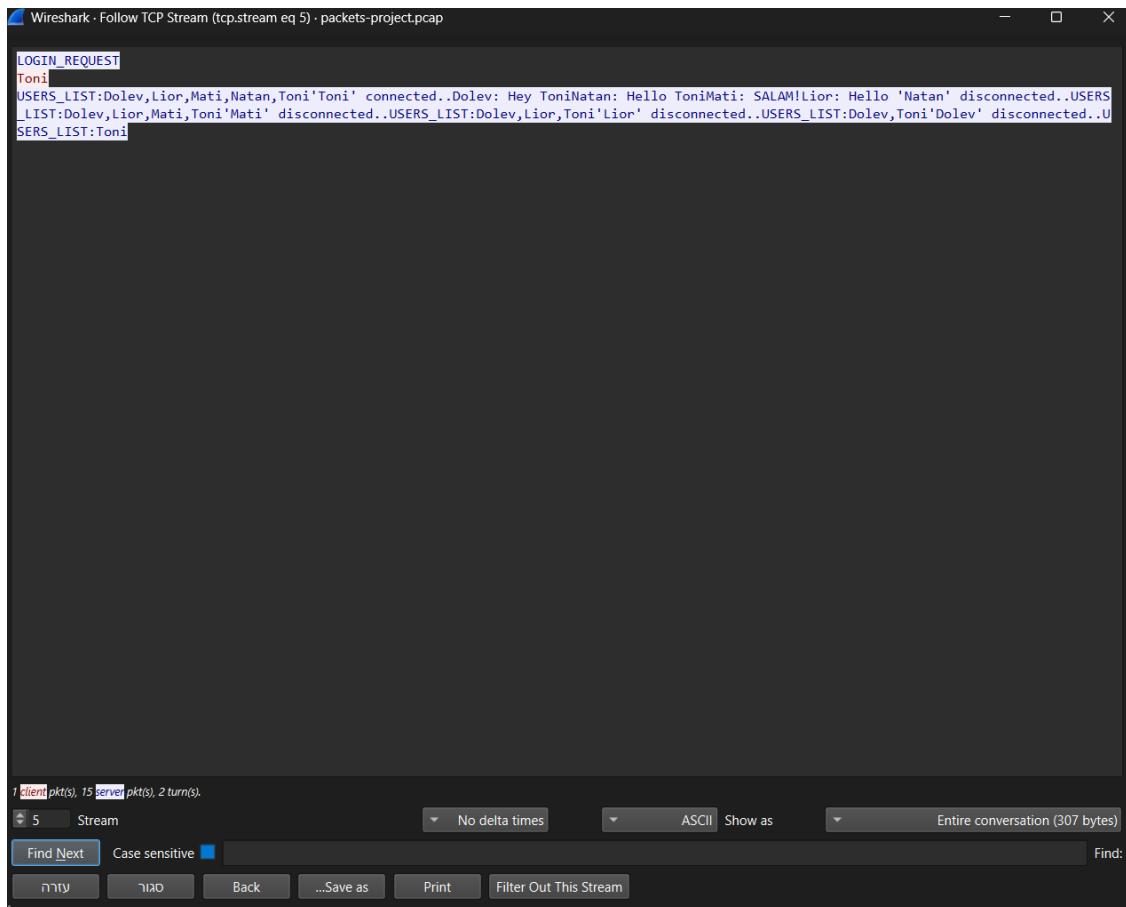
כל ההוראות ההרצה וההתקנה הנדרשות מפורטות בקובץ readme המצורף.

**סעיף ג+: דוגמאות פקודות מהצ'אט**

בתמונה אלו אפשר לראות את השיחה המתועדת בין חמישה למשתמשים שונים ובנוסף מוצג בהגשת הפרויקט קובץ PCAP מה-Wireshark.



## ביתוח ראשוני



בתמונה הנ"ל אפשר לראות את כל ההיסטוריה של הלוקו "טוני" במהלך כל שהותו בצד (התחברות, התנטקות והודעות) ונפרט על כך עבשיו:

- שורה ראשונה והשנייה רואים את התחברות של טוני אל הצד.
- בשורה השלישית והרביעית רואים התנתקות של שאר הלוקוחות וההודעות שנשלחו אל טוני
- לבסוף רואים את התנטקות של הלוקו טוני מהצד.

## ניתוח שבבות הרשות

```
Frame 419: Packet, 59 bytes on wire (472 bits), 59 bytes captured (472 bits) ▾
  Encapsulation type: NULL/Loopback (15)
  Arrival Time: Jan 6, 2026 15:46:49.165011000
    UTC Arrival Time: Jan 6, 2026 13:46:49.165011000 UTC
    Epoch Arrival Time: 1767707209.165011000
      [Time shift for this packet: 0.000000000 seconds]
      [Time delta from previous captured frame: 23.000 microseconds]
      [Time delta from previous displayed frame: 23.000 microseconds]
      [Time since reference or first frame: 3 minutes, 17.420057000 seconds]
        Frame Number: 419
        Frame Length: 59 bytes (472 bits)
        Capture Length: 59 bytes (472 bits)
          [Frame is marked: False]
          [Frame is ignored: False]
        [Protocols in frame: null:ip:tcp:data]
        Character encoding: ASCII (0)
          [Coloring Rule Name: TCP]
          [Coloring Rule String: tcp]
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1 ▾
  Version: 4 = .... 0100
  Header Length: 20 bytes (5) = 0101 ....
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) ▾
    Total Length: 55
    Identification: 0xf0c1 (61633)
    Flags: 0x2, Don't fragment = .... .010 ▾
    Fragment Offset: 0 = 0000 0000 0000 0... ▾
      Time to Live: 128
      Protocol: TCP (6)
    Header Checksum: 0x0000 [validation disabled]
      [Header checksum status: Unverified]
    Source Address: 127.0.0.1
    Destination Address: 127.0.0.1
      [Stream index: 2]
Transmission Control Protocol, Src Port: 42069, Dst Port: 62458, Seq: 289, Ack: 5, Len: 15 ▾
  Source Port: 42069
  Destination Port: 62458
    [Stream index: 5]
    [Stream Packet Number: 34]
    [Conversation completeness: Complete, WITH_DATA (47)] ▾
      [TCP Segment Len: 15]
      Sequence Number: 289 (relative sequence number)
      Sequence Number (raw): 1076949061
      Next Sequence Number: 304 (relative sequence number)
      Acknowledgment Number: 5 (relative ack number)
      Acknowledgment number (raw): 3708362092
      Header Length: 20 bytes (5) = .... 0101
      Flags: 0x018 (PSH, ACK) ▾
        Window: 255
        [Calculated window size: 65280]
        [Window size scaling factor: 256]
        Checksum: 0x4d1a [unverified]
          [Checksum Status: Unverified]
          Urgent Pointer: 0
            [Timestamps] ▾
            [SEQ/ACK analysis] ▾
          [Client Contiguous Streams: 1]
          [Server Contiguous Streams: 1]
          TCP payload (15 bytes)
            Data (15 bytes) ▾
            Data: 55534552535f4c4953543a546f6e69
              [Length: 15]
```

**הסביר כלל על השבבות ומה אפשר להסיק מהתמונה**

שבבת האפליקציה:

מופיעות השורה: (Data 15 bytes) זהה התוכן המקורי שמעבר לברשת.  
ניתן לראות את המידע ב-Data גולמי, ובאשר נמיר את הקוד הhexadecimal לאנגלית  
נקבל את הודעה: "Toni\_LIST:USERS".

### **شبכת התעבורה (Transport - TCP)**

מופיע בשורה: **Transmission Control Protocol** (TCP) כאן מנוילת "השיכחה" בין התוכנות. אנו רואים שהשלוח הוא פורט 42069 והמקבל הוא פורט 62458. הדגמים PSH, ACK, זולקים, מה שאומר שהמידע נדחף לאפליקציה מיד ולא עיבוביים. אורך המידע הטו (ללא בותרות) הוא 15 בתים.

## شبב הרשת (Network - IP)

(מופיע בשורה: Internet Protocol Version 4) שכבה זו אחראית על הכתובות והניתוב. הפרטים החשובים כאן הם כתובת המקור (Source) וכתובת היעד (Destination), ששתיהן מופיעות ב-[127.0.0.1](#) (localhost). זה מאשר שהשלוח והמקבל הם אותו מחשב. בנוסף, השכבה מצינית שהפרוטוקול המוביל בתוכה הוא [TCP](#).

## **שימוש במבנה מלאכותית**

בפרויקט זה נעזרנו במבנה מלאכותית באופן הבא:

### **1. ייצרת קובץ csv לשאלת 1.**

הפרומפט שבו השתמשנו: "אשmach שתעדור לי ליצור קובץ csv שمدמה הודעות משבבת האפליקציה." (בנוסף צירפנו את הוראות ייצרת קובץ ה-csv)

### **2. הכוונה ליצירת ה-UI עבור הצ'אט.**

הפרומפט שבו השתמשנו: "אני רוצה לבנות UI לפרויקט. אשmach שתכזון אותי שלב שלב איך לבנות אותו ובאיזה ספריות מומלץ להשתמש"

### **3. הבנת קובץ readme (בעיקר בהוראות התקנה).**

הפרומפט שבו השתמשנו: "אשmach שתעדור לי לנסה מחדש את קובץ ה-readme שיצרתי בצורה ברורה יותר"