

## פרויקט סיום – רשתות תקשורת :

### תוכן העניינים :

פרוטוקול DNS	עמודים 2-4 :
פרוטוקול DHCP	עמודים 5-7 :
שער האפליקציה:	עמודים 8-15 :
הוראות ההפעלה	עמודים 8-9 :
דיאגרמת מצבים, כולל איבוד חבילות , ובעיות השהיה	עמוד 10 :
תיאור הקובץ DB_manager.py	עמודים 11-12 :
מימוש האפליקציה על פי פרוטוקול TCP	עמודים 13-14 :
מימוש האפליקציה על פי פרוטוקול RUDP	עמוד 15 :
פתרון השאלות התיאורטיות	עמודים 16-17 :

## פרוטוקול DNS:

DNS (Domain Name System) - הינו פרוטוקול המאפשר גישה לאתרי

אינטרנט, ולמעבר מידע בתוך רשת האינטרנט בצורה נוחה ומתאימה למשתמש.

תפקידו של פרוטוקול זה הוא לספק למשתמשי רשת האינטרנט שמות הנוחים יותר לשימוש אנושי טבעי (כתובת URL), מאשר הכתובות האמיתיות (כתובות IP) דרכן יתבצע מעבר המידע בזמן הגלישה באינטרנט.

בני אדם זוכרים בקלות שמות, אך לא כתובות מספריות דוגמת כתובות IP. ה-DNS מגשר על הפער הזה על ידי ביצוע המרה בין הכתובת המותאמת על פי הפרוטוקול אותה זוכר המשתמש, לבין כתובת ה-IP האמיתית, בה המשתמש מיישם תקשורת עם היעד.

## תיאור הפרוטוקול על פי הרצת הקוד :

### DNS – SERVER:

```
server_address = ('localhost', 2001)
# Define DNS records
dns_records = {

    'www.youtube.com': '208.65.153.238',
    'www.google.com': '8.8.8.8',
    'www.example.com': '192.168.0.2',
    'mail.example.com': '192.168.0.3'
}

# Create UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# Bind the socket to the server address
sock.bind(server_address)
# Create a dictionary to store cached DNS records
cache = {}

print('DNS server listening ')
```

בתמונה לעיל מגדירים את הסרבר עם פורט רנדומאלי, אחר כך מגדירים את המאגר של הכתובות שהסרבר מספק, מייצרים UDP סוקט, ומתחילים להאזין ללקוחות. אחר כך מגדירים את ה cache של הסרבר.

```
while True:
    data, address = sock.recvfrom(4096)
    # Parse DNS query
    query = data.decode().strip()
    if query in cache:
        # Return cached DNS record if available
        response = cache[query]
    elif query in dns_records:
        # Build DNS response
        response = dns_records[query]
        # Cache the DNS record
        cache[query] = response
    else:
        # Return an error message if the DNS record is not found
        response = "DNS record not found"

    # Send the DNS response to the client
    sock.sendto(response.encode(), address)
```

בשאר הקוד, מקבלים כתובת מהקליינט דרך הסוקט, ממירים אותה לבייטים, ובודקים אם בתצורה המקורית שלה (של כתובת ה ip), היא נמצאת ב-cache. אם היא נמצאת שם אז מקבלים אותה, ומחזירים את כתובת ה-IP הדרושה לקליינט. אחרת אנו מחפשים במאגר הכתובות אחר כתובת ה ip הרלוונטית. אם מוצאים מוסיפים ל-cache ומחזירים את ה-IP הדרוש לקליינט. אחרת ה-DNS לא יכול להחזיר את ה-IP של הכתובת שהקליינט מחפש.

## DNS- Client:

```
import socket

server_address = ('localhost', 2001)

# Create UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

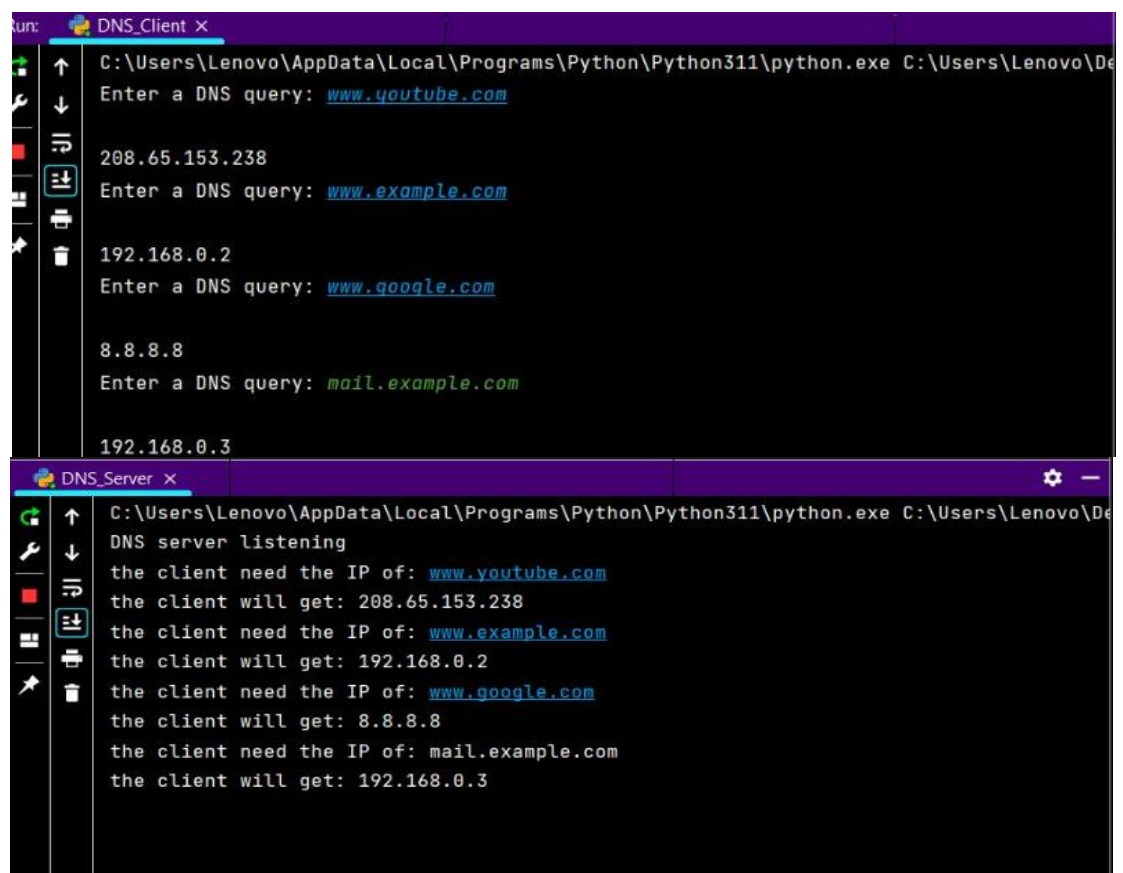
while True:
    # Get user input for DNS query
    query = input("Enter a DNS query: ")

    # Send DNS query to server
    sock.sendto(query.encode(), server_address)

    # Receive DNS response from server
    data, address = sock.recvfrom(4096)
    response = data.decode().strip()

    # Print DNS response
    print(response)
```

זה הקליינט שלנו שמתקשר עם הסרבר שעשינו ומבקש ממנו את כתובות ה ip הרלוונטיות עבור כתובות ה url אותן הוא מציג בפני הלקוח.



The screenshot shows two terminal windows. The top window, titled 'DNS\_Client X', shows the client's execution. It prompts the user to enter a DNS query, and for each query, it displays the received IP address. The bottom window, titled 'DNS\_Server X', shows the server's execution. It logs the received queries and the corresponding IP addresses it is returning to the client.

```
run: DNS_Client X
C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\python.exe C:\Users\Lenovo\De
Enter a DNS query: www.youtube.com
208.65.153.238
Enter a DNS query: www.example.com
192.168.0.2
Enter a DNS query: www.google.com
8.8.8.8
Enter a DNS query: mail.example.com
192.168.0.3

DNS_Server X
C:\Users\Lenovo\AppData\Local\Programs\Python\Python311\python.exe C:\Users\Lenovo\De
DNS server listening
the client need the IP of: www.youtube.com
the client will get: 208.65.153.238
the client need the IP of: www.example.com
the client will get: 192.168.0.2
the client need the IP of: www.google.com
the client will get: 8.8.8.8
the client need the IP of: mail.example.com
the client will get: 192.168.0.3
```

## פרוטוקול DHCP:

(DHCP(Dynamic Host Configuration Protocol):

זהו פרוטוקול רשת המשמש להקצאה אוטומטית של כתובות IP והגדרות אחרות של הרשת כגון subnet Mask, Default gateway ועוד.

DHCP מבטל את הצורך בחלוקה ידנית של כתובות IP, מה שהופך את ניהול הרשת ליעיל יותר ונוטה פחות לשגיאות.

שרת זה נולד למעשה כאשר מספר המשתמשים באינטרנט הלך וגדל, כך שלא ניתן יותר היה לספק עבורם כתובות IP באופן ידני.

כאשר משתמש מתחבר לרשת, הוא שולח בקשה לשרת DHCP עבור כתובת ה IP, המתאימה עבורו.

שרת ה-DHCP מקצה כתובת IP זמינה ושולח אותה בחזרה למכשיר יחד עם ההגדרות הרלוונטיות ביחס לאותה כתובת.

שימוש בכתובות ה IP המסופקות על ידי פרוטוקול זה הינן זמניות. ולכן למעשה משתמשי הרשת חייבים לחדש מעת לעת את הקצאת כתובת ה-IP שלהם משרת ה-DHCP.

## תיאור הפרוטוקול על פי הרצת הקוד :

בתמונות אלו ניתן לראות את הרצת הקוד אצל הסרבר.

```
import socket
import random

server_address = ('localhost', 67)

# Define DHCP message types
DHCP_DISCOVER = 1
DHCP_OFFER = 2
DHCP_REQUEST = 3
DHCP_ACK = 4

# Define IP address pool
ip_pool = ['192.168.0.' + str(i) for i in range(1, 255)]

# Create UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Bind the socket to the server address
sock.bind(server_address)

print('DHCP server listening on {}:{}'.format(*server_address))

while True:
    data, address = sock.recvfrom(4096)

    # Parse DHCP DISCOVER message
    msg_type, client_mac = data.decode().split(',')
    if int(msg_type) == DHCP_DISCOVER:
        # Generate a random IP address from the pool
        ip_address = random.choice(ip_pool)

        # Build DHCP OFFER message
        msg = ','.join([str(DHCP_OFFER), ip_address, client_mac])
        sock.sendto(msg.encode(), address)
        print('Sent DHCP OFFER message to client')

        # Wait for DHCP REQUEST message
        data, address = sock.recvfrom(4096)
        msg_type, ip_address, client_mac = data.decode().split(',')
        if int(msg_type) == DHCP_REQUEST:
            # Assign the requested IP address to the client
            if ip_address in ip_pool:
                ip_pool.remove(ip_address)

            # Build DHCP ACK message
            msg = ','.join([str(DHCP_ACK), ip_address, client_mac])
            sock.sendto(msg.encode(), address)
            print('Sent DHCP ACK message to client')
```

כפי שניתן לראות, קוד זה מיישם שרת DHCP בסיסי אשר מאזין בכתובות IP נתונות. השרת מאזין להודעות DHCP DISCOVER מהלקוח ומגיב בהודעת DHCP OFFER המכילה כתובת IP שנבחרה באקראי מהמאגר הנתון לשירותו. אם הלקוח שולח הודעת DHCP REQUEST כדי לבקש את כתובת ה-IP המוצעת, השרת שולח הודעת DHCP ACK כדי להקצות את הכתובת ללקוח ומסיר אותה ממאגר כתובות ה-IP. קוד זה משתמש בחיבור UDP כדי לשלוח ולקבל הודעות DHCP, ובו הגדרות קבועות עבור הלקוח. מאגר כתובות ה-IP מוגדר כרשימה של מחרוזות, וכתובת ה-IP נבחרת באופן אקראי.

## תיאור הרצת הקוד אצל הקליינט :

```
1 import socket
2
3 client_address = ('localhost', 68)
4 server_address = ('localhost', 67)
5
6 # Define DHCP message types
7 DHCP_DISCOVER = 1
8 DHCP_OFFER = 2
9 DHCP_REQUEST = 3
10 DHCP_ACK = 4
11
12 # Create UDP socket
13 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
14 sock.bind(client_address)
15
16 # Build DHCP DISCOVER message
17 msg = ','.join([str(DHCP_DISCOVER), '00:11:22:33:44:55'])
18 sock.sendto(msg.encode(), server_address)
19
```

```
while True:
    data, server = sock.recvfrom(4096)

    # Parse DHCP OFFER message
    msg_type, ip_address, server_mac = data.decode().split(',')
    if int(msg_type) == DHCP_OFFER:
        # Build DHCP REQUEST message
        msg = ','.join([str(DHCP_REQUEST), ip_address, '00:11:22:33:44:55'])
        sock.sendto(msg.encode(), server_address)

    # Parse DHCP ACK message
    data, server = sock.recvfrom(4096)
    msg_type, ip_address, server_mac = data.decode().split(',')
    if int(msg_type) == DHCP_ACK:
        print('Assigned IP address:', ip_address)
        break
```

בתמונות אלו ניתן לראות קליינט ל- DHCP SERVER המבקש כתובת IP משרת DHCP. הוא שולח הודעת DHCP DISCOVER לשרת, מקבל הודעת DHCP OFFER המכילה כתובת IP זמינה, ושולח הודעת DHCP REQUEST לאישור כתובת ה-IP שנבחרה, ומקבל הודעת DHCP ACK המאשרת את הקצאת כתובת ה-IP שנבחרה. והוא מחכה עד שמקבל בהצלחה הודעת DHCP ACK.

## שער האפליקציה :

### הוראות ההפעלה :

#### בודק יקר

לצורך הרצת הקוד עליך לוודא שמותקנת אצלך הגרסה העדכנית והמתאימה לסביבת העבודה שלך, של התוכנה SQLite, המאפשרת ליצור מסדי נתונים מסוג SQLite.

כמו כן אנו ממליצים לך להוריד גרסה מעודכנת של "DB Browser for SQLite" המאפשרת לך לעקוב אחר מסד הנתונים באופן גרפי, ברור ומובן .

לאחר מכן עליך להריץ את הקובץ DB\_manager.py. קובץ זה אמור ליצור אצלך בסביבת העבודה, את מסד הנתונים המבוקש המכיל בתוכו טבלת sql לצורך בדיקת המטלה. מסד נתונים זה נקרא "My\_Class.db", והטבלה הממומשת בתוכו נקראת "Students".

מסד נתונים זה אמור לייצג כיתה של סטודנטים, כאשר עמודות הטבלה הינן פרמטרים של :

(א) שם התלמיד.

(ב) מקצוע .

(ג) שנת לימודים.

(ד) ממוצע ציונים.

(ה) התואר אליו הסטודנט לומד, כך שהאפשרויות הן: Ph.D, M.A, B.A.

מכיוון שבשפת sql לא ניתן ליצור במסד נתונים אחד, את אותה הטבלה בעלת אותו השם פעמיים, וקובץ זה משתמש במחרוזות בשפת sql לצורך הפעלתו, לכן הינך אמור להריץ אותו פעם אחת בלבד בתחילת הבדיקה. אם תנסה להריץ אותו יותר מפעם אחת, אתה תקבל שגיאה.

מימוש האפליקציה מבוסס על קשר בין client ל server . כאשר הלקוח מקבל רשימת שאליות משרת האפליקציה, והמשתמש מקיש את מספר השאלית המבוקשת, לצורך קבלת תשובה מהשרת.

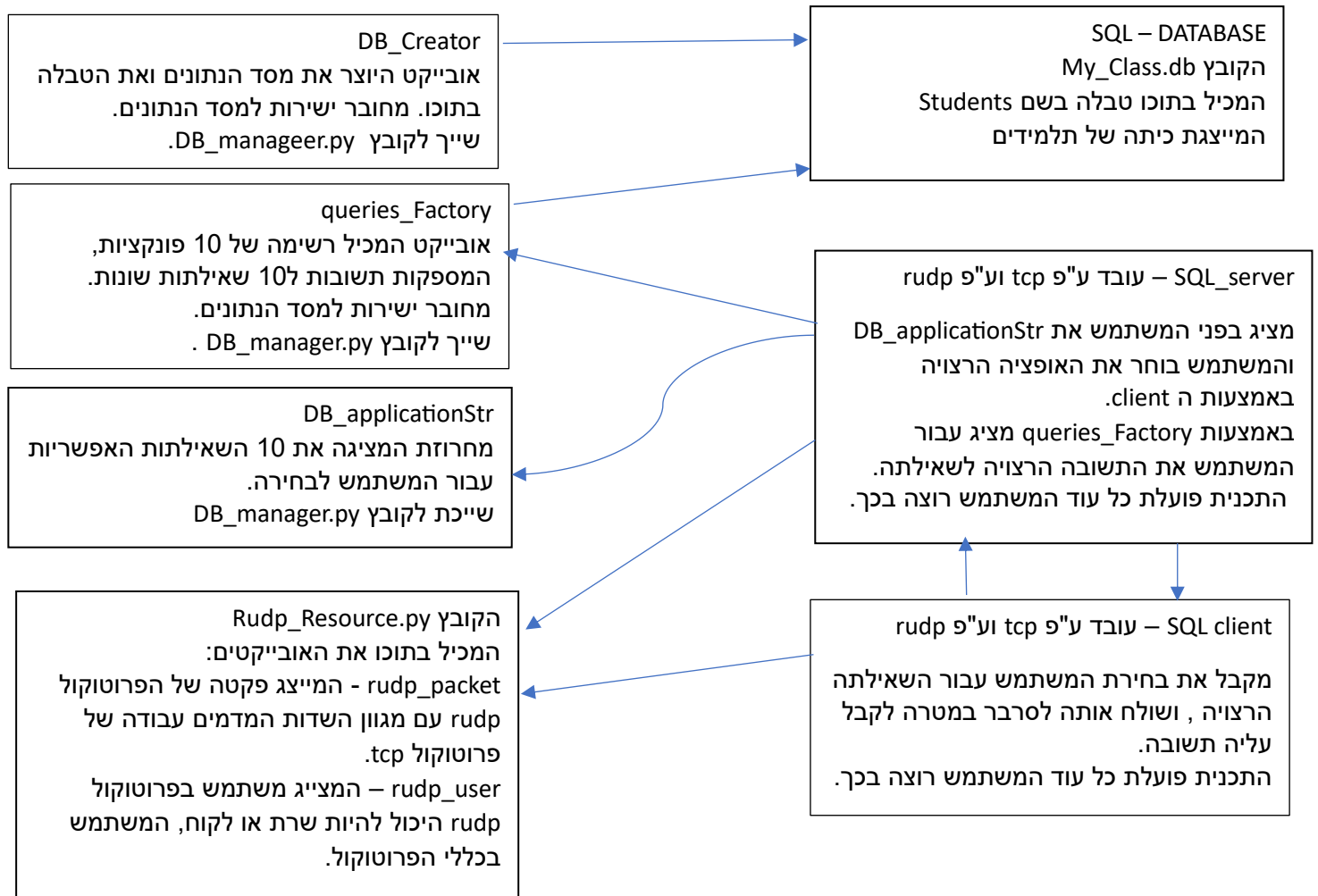
כמו כן, בסיום ההרצה של כל סבב מסבבי התוכנית, המשתמש מחליט האם הוא רוצה להמשיך לסבב נוסף, ושולח זאת לשרת דרך הלקוח. סבב - זוהי מערכת של בחירת שאלית מצד המשתמש, שליחתה אל השרת, וקבלת תשובה עליה.

שים לב שלצורך הבדיקה, תחילה עליך להריץ את הקובץ של השרת, ולאחר מכן את הקובץ של הלקוח.



האפליקציה ממומשת בשני פרוטוקולים שונים : tcp , udp , כאשר במימוש  
האפליקציה על פי פרוטוקול udp , נעשו מספר שינויים לצורך דימוי פרוטוקול  
udp אמין, המתנהג בהתאם לצורת ההתנהגות המובנית מראש של הפרוטוקול  
tcp , ובכך ליצור מודל חדש לפרוטוקול תקשורת, המממש בתוכו גם את  
המהירות של הפרוטוקול udp , לצד האמינות הגבוהה הבאה לידי ביטוי  
באמצעות הפרוטוקול tcp .

## דיאגרמת מצבים:



## תשובה על איבוד חבילות:

המערכת שלנו מתגברת על אובדן חבילות בפרוטוקול `rudp` באמצעות שימוש בטכניקות הקיימות בפרוטוקול `tcp`. אנחנו מבצעים שליחת `ack` עבור כל חבילה שהתקבלה, וכאשר עובר זמן קבלת `ack` עבור החבילה, אנו שולחים אותה מחדש. כמו כן, קיים אצלנו מנגנון עבור שליחת "3 dup\_ACK messages" עבור חבילה אשר התקבלה שלא כסדר, וזה גורם לשולח להבין שאבדה לו חבילה. מצב זה תורם לכך שקצב זרימת הנתונים בין השרת ללקוח בהתבססות על `rudp`, יהיה מהיר, אמין ומשמעותי.

## תשובה על בעיות "latency":

המנגנון של שליחת "3 dup\_ACK messages" המוזכר לעיל מסייע אף הוא במניעת עיכובים ברשת, שכן מנגנון זה גורם לשליחה חוזרת ומהירה של פקטות אבודות ומצריך מאיתנו פחות זמן השהיה בפעילות הרשת שבין הלקוח לשרת. כמו כן לכל משתמש בפרוטוקול `rudp`, הצמדנו מגוון עזרים המאפשרים לו לעקוב אחר סדרי שליחת וקבלת ההודעות עבורו, כולל שימוש בחלון בקרת עומס עבור פקטות יוצאות ופקטות נכנסות, שימוש בפונקציית המחלקת לסגמנטים קצרים יותר הודעות ארוכות מדי. מצב זה מסייע במניעת עומס תעבורה, ומקל על התעבורה ברשת להיות מהירה יותר.

## הקובץ DB\_manager.py:

קובץ זה מכיל בתוכו 2 מחלקות שונות, וכן מחרוזת ארוכה האמורה להישלח כצעד ראשון עבור הלקוח, כחלק מאופן השימוש באפליקציה. קובץ זה משתמש בספרייה sqlite3 בשפת python.

### שתי המחלקות השונות הן:

**(1) Creator DB –** מחלקה זו מכילה בתוכה שני משתנים שונים:  
**connector –** משתנה זה מאותחל על ידי הפונקציה "connect" מהספרייה sqlite3, ומאפשר ליצור חיבור עם מסד הנתונים הממומש בשפת SQL.  
**(2) cursor –** משתנה זה אמור להביא לידי ביטוי מגוון יכולות הקשורות למסד הנתונים כגון: יצירת טבלאות, אתחול טבלאות, מימוש שאילתות, ועוד..  
 הוא מממש זאת באמצעות הפונקציה "execute" מהספרייה sqlite3, המקבלת כפרמטר מחרוזת בשפת SQL.  
 באמצעות משתנים אלו, "Creator\_DB" מממש את הפונקציות "create\_table" ו "add\_objects", המאתחלות טבלה ומוסיפות לה נתונים רלוונטיים לצורך ביצוע השאילתות.

### **queries\_Factory (2)**

בדומה ל "Creator\_DB", גם מחלקה זו מכילה בתוכה 2 משתנים מסוג "connector" ו "cursor", וזאת לצורך מימוש 10 פונקציות המממשות 10 שאילתות שונות.  
 שרת האפליקציה אמור להשתמש במחלקה "queries\_Factory", ובפונקציות המובנות בתוכה, לצורך מתן תשובות אל הלקוח.

כמו כן קובץ זה מכיל בתוכו את המחרוזת "DB\_applicationStr", המציגה למשתמש רשימה של 10 שאילתות שונות למימוש, כאשר מתוכן המשתמש אמור לבחור אחת בכל סבב מסבבי הרצת התוכנית.  
 זוהי המחרוזת הראשונה הנשלחת מהשרת אל הלקוח לצורך מימוש האפליקציה.

## להלן תיאור הרצת הקובץ "DB\_manager.py" על פי התוכנה DB Browser for SQLite :

DB Browser for SQLite - C:\Users\nerya\PycharmProjects\Ex\_Final\My\_Class.db

File Edit View Tools Help

New Database Open Database Write Changes Revert Changes Open Project Save Project Attach Database Close Database

Database Structure Browse Data Edit Pragma Execute SQL

Table: Students Filter in any column

	Name	Major	Year	Average	Degree_Level
	Filter	Filter	Filter	Filter	Filter
1	Maria	Computer Science	year of study: 2	80.0	Bachelor
2	Mor	Psychology	year of study: 3	79.0	Bachelor
3	Mario	Psychology	year of study: 2	90.0	Master
4	Luigi	Economy	year of study: 2	67.97	Bachelor
5	Sasha	Economy	year of study: 3	88.0	Bachelor
6	Nathan	Computer Science	year of study: 1	77.0	Bachelor
7	Shimi	Psychology	year of study: 3	95.0	Bachelor
8	Sharif	Computer Science	year of study: 2	89.0	Master
9	Sandra	Economy	year of study: 1	90.0	Bachelor
10	Sakura	Computer Science	year of study: 1	92.0	Bachelor
11	Ido	Economy	year of study: 1	70.0	Bachelor
12	Ori	Psychology	year of study: 4	82.0	Bachelor
13	Alice	Economy	year of study: 2	69.0	Master
14	James	Computer Science	year of study: 1	86.0	Master
15	Dave	Computer Science	year of study: 1	81.0	Bachelor

Go to: 1

1 - 15 of 15

Edit Database Cell

Mode: Text

Type of data currently in cell: Text / Numeric  
1 character(s)

Apply

DB Schema

Name	Type	Schema
Tables (1)		
Students		CREATE TABLE Students (Na
Indices (0)		
Views (0)		
Triggers (0)		

SQL Log Plot DB Schema Remote

UTF-8

20:38 08/03/2023 עבר

מעונ... 81%

הקלד כאן כדי לחפש

בתמונה זו ניתן לראות כיצד יצרנו את מסד הנתונים "My\_Class.db" וכן את הטבלה "Students", ומילאנו אותה ב15 אובייקטים שונים, על פי הפרמטרים הנתונים.

## מימוש האפליקציה על פי פרוטוקול tcp:

מימוש האפליקציה באמצעות פרוטוקול tcp, בא לידי ביטוי באמצעות דו שיח בין השרת ללקוח.

### בשלב א':

השרת שולח ללקוח את המחרוזת "DB\_applicationStr", המיובאת מהקובץ "DB\_manager.py".

מחרוזת זו מסבירה למשתמש על מסד הנתונים, ומציגה בפניו רשימה של 10 שאלות, כאשר מתוכן הוא אמור לבחור אחת.

לאחר שהמשתמש בוחר את מספר השאלתה הרצויה, השרת משתמש בפונקציה "navigator\_queries", המקבלת כקלט את מספר השאלתה הנבחרת על ידי המשתמש, ומחזירה לו את התשובה לשאלתה זו באמצעות האובייקט "queries" המממש את המחלקה "qeries\_Factory", המיובאת אף היא מהקובץ "DB\_manager.py".  
על תשובה זו שולח הלקוח לשרת את המילה "accept".

### בשלב ב':

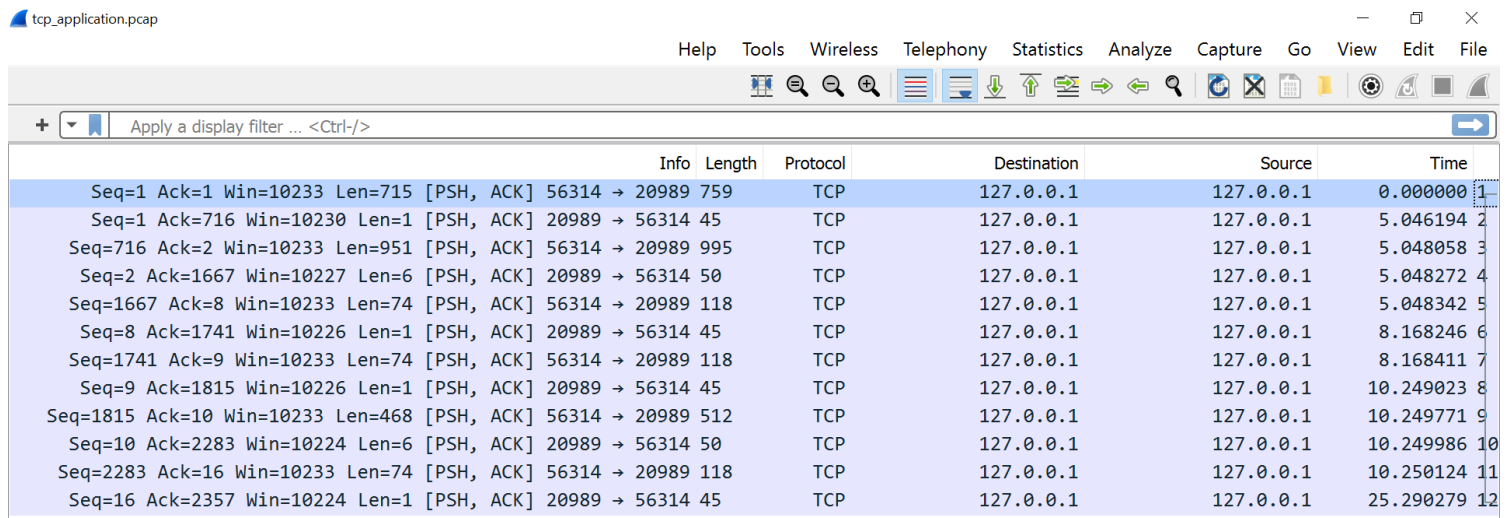
השרת שואל את הלקוח האם הוא מעוניין להמשיך לסבב נוסף בהפעלת האפליקציה.

החלטה זו תלויה בבחירת המשתמש, האמור להקליד "Y" במידה והוא מעוניין להמשיך. אחרת, עליו להקליד "N", ובכך למעשה לשלוח הודעת יציאה אל השרת.

כאש המשתמש בוחר להמשיך, הוא יאלץ לבחור את מספר השאלתה הרצויה עבור הסבב החדש.

שלב זה ממומש באמצעות לולאת while אינסופית הממשיכה או נגמרת בהתאם לרצון המשתמש.

## תיאור הרצת האפליקציה על פי פרוטוקול tcp בהתאם ל-Wireshark:



	Info	Length	Protocol	Destination	Source	Time
1	Seq=1 Ack=1 Win=10233 Len=715 [PSH, ACK]	56314 → 20989 759	TCP	127.0.0.1	127.0.0.1	0.000000
2	Seq=1 Ack=716 Win=10230 Len=1 [PSH, ACK]	20989 → 56314 45	TCP	127.0.0.1	127.0.0.1	5.046194
3	Seq=716 Ack=2 Win=10233 Len=951 [PSH, ACK]	56314 → 20989 995	TCP	127.0.0.1	127.0.0.1	5.048058
4	Seq=2 Ack=1667 Win=10227 Len=6 [PSH, ACK]	20989 → 56314 50	TCP	127.0.0.1	127.0.0.1	5.048272
5	Seq=1667 Ack=8 Win=10233 Len=74 [PSH, ACK]	56314 → 20989 118	TCP	127.0.0.1	127.0.0.1	5.048342
6	Seq=8 Ack=1741 Win=10226 Len=1 [PSH, ACK]	20989 → 56314 45	TCP	127.0.0.1	127.0.0.1	8.168246
7	Seq=1741 Ack=9 Win=10233 Len=74 [PSH, ACK]	56314 → 20989 118	TCP	127.0.0.1	127.0.0.1	8.168411
8	Seq=9 Ack=1815 Win=10226 Len=1 [PSH, ACK]	20989 → 56314 45	TCP	127.0.0.1	127.0.0.1	10.249023
9	Seq=1815 Ack=10 Win=10233 Len=468 [PSH, ACK]	56314 → 20989 512	TCP	127.0.0.1	127.0.0.1	10.249771
10	Seq=10 Ack=2283 Win=10224 Len=6 [PSH, ACK]	20989 → 56314 50	TCP	127.0.0.1	127.0.0.1	10.249986
11	Seq=2283 Ack=16 Win=10233 Len=74 [PSH, ACK]	56314 → 20989 118	TCP	127.0.0.1	127.0.0.1	10.250124
12	Seq=16 Ack=2357 Win=10224 Len=1 [PSH, ACK]	20989 → 56314 45	TCP	127.0.0.1	127.0.0.1	25.290279

בהקלטה זו ניתן לראות תיאור של הרצת הקוד בהתאם לפרוטוקול tcp ע"פ Wireshark.

נחלק הקלטה זו ל 3 שלבים:

שלב א':

- ניתן לראות כיצד השרת שולח ללקוח את רשימת השאילתות – הקלטה מס' 1.
- השרת בוחר את אחת האופציות – הקלטה מס' 2.
- הלקוח מקבל עליה תשובה – הקלטה מס' 3.
- הלקוח משיב באמירה "accept" – הקלטה מספר 4.

שלב ב':

- השרת שואל את הלקוח האם ברצונו להמשיך בהרצת התוכנית – הקלטה 5.
- הלקוח משיב בחיוב – הקלטה 6.
- השרת שואל את הלקוח, איזו שאילתה ירצה לממש עכשיו – הקלטה 7.
- הלקוח בוחר את אחת האופציות – הקלטה 8.
- הלקוח מקבל תשובה – הקלטה 9.
- הלקוח משיב "accept" – הקלטה 10.

שלב ג':

- השרת שואל את הלקוח האם ירצה להמשיך בהרצת התוכנית – הקלטה 11.
- הלקוח מסרב – הקלטה 12.

## מימוש האפליקציה על פי rudp:

לצורך מימוש זה השתמשנו ב 3 קבצי פייתון שונים:  
 1) Rudp\_Resource – המכיל בתוכו את המחלקות:

- rudp\_user – המייצג משתמש בפרוטוקול rudp אשר יכול להיות שרת או לקוח באותה מידה. לכל משתמש ישנם שדות פרטיים המאפשרים לו לממש את הפרוטוקול rudp לפי צורת ההתנהגות של פרוטוקול tcp. בין אמצעים אלו ניתן לראות:
  - (a) last seq, last acknowledge – משתנים מסוג int אשר תפקידם הוא לעקוב אחרי איבוד פקטות ואחר הרצף הסדיר של ההודעות הנשלח בחיבור שבין השרת ללקוח.
  - (b) packets RecvList, packets SendList, acks list - רשימות של הפקטות שנשלחו, של הפקטות שהתקבלו, וכן של הודעות ה ack שהתקבלו עבור כל משתמש.
  - (c) slide window – למעשה לכל משתמש ישנם 2 חלונות כאלו, הן עבור פקטות המתקבלות, והן עבור פקטות נשלחות. תפקיד פקטות אלו הוא למנוע הצפה של מידע ברשת האינטרנט, כך שלא תיווצר מעמסה על תשתיות הרשת. אנו הגדרנו אותם בגודל קבוע של 4, כך שלא ניתן לשלוח יותר מ-4 פקטות בבת אחת.

- rudp\_packet – המייצג פקטת rudp אשר בדומה לפקטת tcp מכילה בתוכה שדות המיוצגים בפקטת tcp כגון seq\_number, ack\_number דגלים מיוחדים ועוד, המאפשרים מעקב אחר הפקטה באופן המאפשר ביצועים ברמת פרוטוקול tcp ביחס לשימוש בפרוטוקול udp רגיל.

2) rudp\_SQL\_client המייצג לקוח של שרת האפליקציה בפרוטוקול זה.

3) rudp\_SQL\_server המייצג שרת אפליקציה בפרוטוקול זה.

שני קבצים אלו של השרת והלקוח מתקשרים ביניהם בצורה זהה לדרך שבה המימוש של האפליקציה בפרוטוקול tcp עובד, אלא שלאור השימוש שלהם בקובץ Rudp\_Resource, הם מציגים ביצועים טובים יותר ברמת פרוטוקול udp המתנהג בהתאם לצורת ההתנהגות של פרוטוקול tcp במצבים של flow control ושל congestion control, וכן לצורך מתן שירותי אמינות טוב יותר.

## פתרון השאלות:

(1)

תעבורה: TCP (פרוטוקול בקרת תעבורה) הוא פרוטוקול תקשורת המספק חיבור בין שרת ללקוח אחד, ואילו QUIC (המספק חיבורי אינטרנט מהירים מבוססי פרוטוקול UDP) הוא פרוטוקול תעבורה היכול לספק שירותי אספקת נתונים למגוון מקורות שונים ברשת.

חיבור: פרוטוקול TCP משתמש בתהליך לחיצת ידיים נפרד, כדי לייצר חיבור רציף בין הלקוח והשרת ברשת אותם הוא מחבר, בעוד QUIC נועד לפעול על גבי פרוטוקול UDP, ולכן אינו מחזיק בחיבור מסונכרן בין שרת ללקוח.

בקרת עומס: TCP מסתמך על אלגוריתם לבקרת עומס על פי אובדן פקטות. לעומת זאת, QUIC משתמש במנגנון בקרת עומס מבוסס RTT.

אבטחה: TCP אינו מספק תכונות אבטחה מובנות כלשהן ומסתמך על פרוטוקולים נוספים כגון SSL/TLS. לעומת זאת, QUIC מספק הצפנה מקצה לקצה, מה שהופך אותו לפרוטוקול מאובטח יותר מ-TCP.

(2)

### תגובה לעומס:

Cubic מחליטה האם להקטין או להגדיל את גודל חלון העומס ביחס להודעות הנכנסות והיוצאות של משתמשי האלגוריתם, בהתאם לזיהוי אובדן פקטות ברשת. Vegas מצד שני, מחליטה זאת על פי זיהוי עיכובים ברשת, כאשר היא שמה לב לכך שזמן ה RTT של כל פקטה הולך וגדל.

### תגובה לאובדן פקטות:

Cubic עשויה להקטין בצורה דרסטית את גודל חלון העומס במקרה של אובדן פקטות. Vegas לעומת זאת מורידה את גודל חלון העומס שלה בצורה די קבועה, ולא גדולה במיוחד.

(3)

BGP (Border Gateway Protocol) הוא פרוטוקול ניתוב המשמש להעברת מידע ברשת האינטרנט.

בניגוד ל-OSPF (Open Shortest Path First), שהוא פרוטוקול העברת מידע המבוסס על מציאת מסלולים קצרים ביותר, BGP הינו פרוטוקול מתוחכם יותר המחפש את המסלול הטוב ביותר להעברת מידע, שהוא לא בהכרח חייב להיות



הקצר ביותר, אלא בחירתו תלויה למעשה במגוון גורמים כגון: איכות הקשר מנקודה לנקודה, עומסי הרשת במקומות שונים ועוד..

(5)

פרוטוקול הכתובות (ARP) משמש למיפוי כתובת ה mac של המכשיר לכתובת ה-IP שלו ברשת מקומית.

(1) בעוד DNS משמש למציאת שמות דומיין לכתובות IP ברשת האינטרנט, ARP משמש לתקשורת בין מכשירים שונים באותה הרשת.

(2) ARP עובד בשכבת הרשת, שהיא השכבה השלישית במודל השכבות הנלמד בקורס, בעוד DNS עובד בשכבת האפליקציה.