

תרגול 13 MAGSHI-CHAT

רקע

אז במשך 12 שבועות:

הגדרנו רשימות

טענו מחסניות

בנינו מחלקות

ייצרנו אובייקטים

זרקנו exceptions

תכננו פרויקט גדול

לינקג'נו ספריות

טיילנו בין עצים מטומפליטים

שזרנו חוטים

ונעלנו מנעולים

עכשיו... הגענו לתרגיל האחרון, שמסכם כמעט כל מה שלמדנו בסמסטר!

מטרה

בתרגיל זה נממש **שרת הודעות** הפועל באופן דומה לשרתי אפליקציית Whatsapp אך פשוט בהרבה. נבנה שרת **המנהל מספר משתמשים** ומאפשר **שליחת הודעות** בין **משתמשים אשר מחוברים לשרת**. עבור כל משתמש השרת ישמור את היסטוריית השיחות עם כלל המשתמשים המחוברים באותה עת. הלקוח מסופק ע"י מגשימים, ועליכם/ן נדרש לכתוב את שרת ההודעות אתו הלקוחות יוכלו לתקשר.



אלה השלבים שנעבור:

הרצת Demo	בניית המערכת (חיבור לקוחות)	אכיפת הפרוטוקול ושליחת הודעות	בדיקות
<ul style="list-style-type: none"> נראה איך התוצר הסופי צריך להיראות שימוש ב-Wireshark 	<ul style="list-style-type: none"> יישום הרעיון של שרת MT שראינו בכיתה 	<ul style="list-style-type: none"> הרצה באמצעות thread-ים שימוש במנגנוני סנכרון 	<ul style="list-style-type: none"> נריץ טסטים כדי לראות שהכל עובד.

נתרגל מיומנויות חשובות:

- שימוש בת'רדים לצורך מקבול התוכנית.
- שימוש ב-sockets לתקשורת בין תהליכים.
- עבודה עם מבני נתונים.
- שימוש במנגנוני סנכרון.
- עבודה עם קבצים
- את התרגיל צריך להגיש ב-GIT: [לינק להוראות שימוש ב-GIT](#).
כדאי לקרוא גם [דגשים לתכנות נכון](#).



"PRACTICE MAKES PERFECT"

בהצלחה יא אלופות ואלופים!

שלב 1: הרצת Demo

בדיקות	אכיפת פרטוקול ההודעות	בניית המערכת (חיבור לקוחות)	הרצת Demo
--------	--------------------------	--------------------------------	-----------

הרצת קבצי ה-Demo של השרת והלקוח(ות)

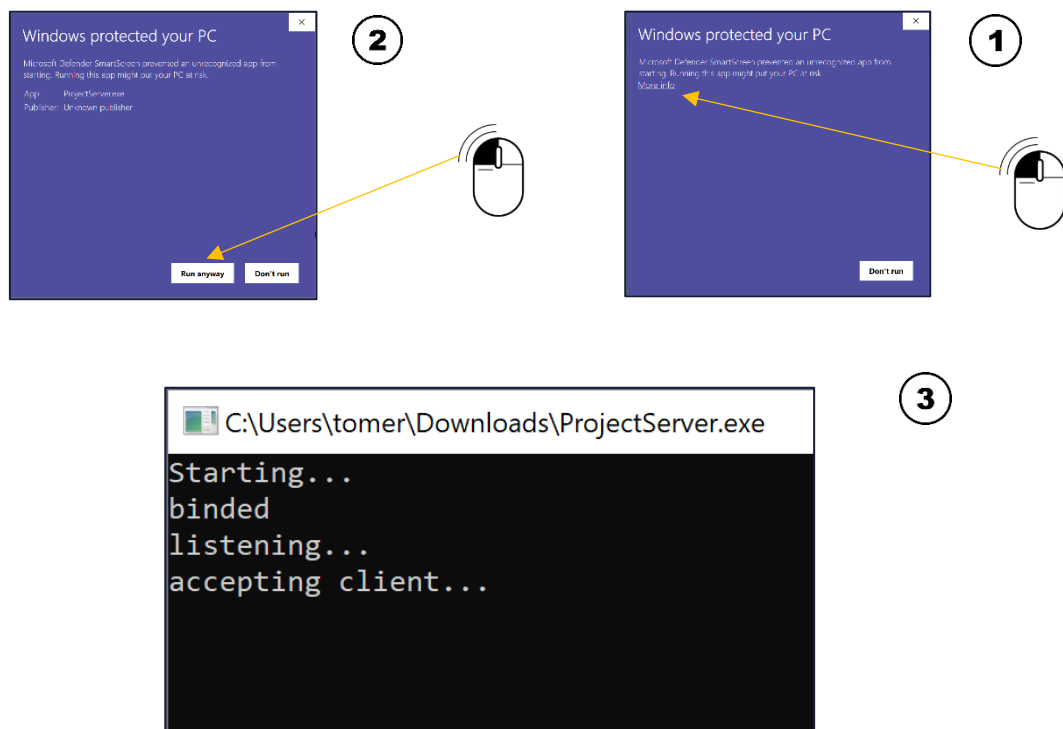
בתרגיל מורכב כמו זה, ובכלל בכל פרויקט שנעשה בהמשך חשוב שנבין מה המטרה שלנו, לאן אנחנו רוצים להגיע ואיך זה אמור להיראות בסוף. לא תמיד נוכל לדעת מראש איך הפרויקט ייראה, אבל במקרה הזה יש לנו הדגמה קטנה שאפשר להריץ.

בתיקיית התרגיל ישנה תת-תיקייה שנקראת Demo .

בתוכה ניתן למצוא שתי תיקיות עם קבצי EXE, אחד עבור התוכנה של **צד השרת** (אותה אנחנו נרצה לממש בתרגיל הזה), ואחת עבור התוכנה של **צד הלקוח**, שתתחבר לשרת שנבנה.

הורידו את התיקייה למחשב שלכם, והריצו את קובץ ה-EXE של צד השרת.

שימו , סיבוי מאוד גדול ש-Windows לא ייתן לנו מיד להריץ את הקובץ, צריך ללחוץ על "More info" ואז על "Run anyway"



השרת רץ! ומוכן לקבל לקוחות, בעת נריץ את התוכנה של צד הלקוח.

לפני שנריץ את קובץ ה-EXE של צד הלקוח, נשים לב שיש קובץ נוסף בתיקייה שנקרא **config.txt**. זה הקובץ שמגדיר פרמטרים לתוכנת הצד לקוח שלנו, אפשר לקבוע ערכים של פרמטרים שונים שרלבנטיים לריצה של התכנית:

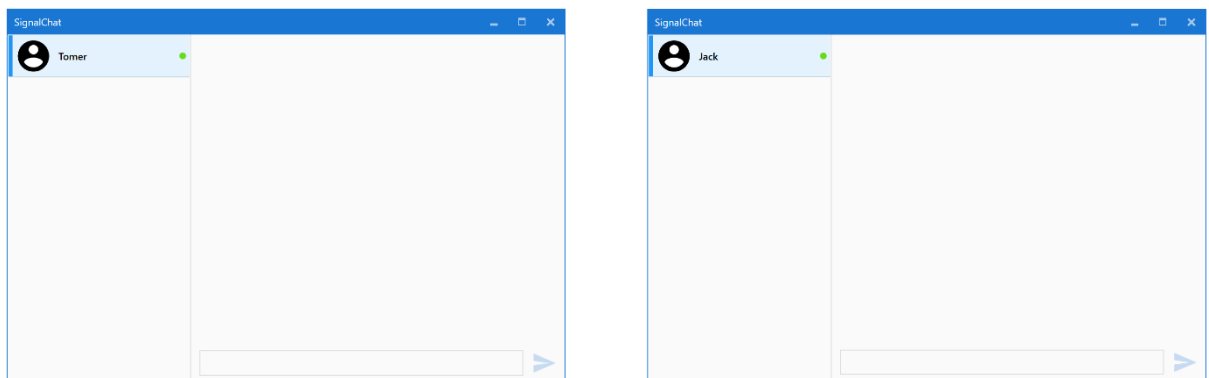
```
server_ip=127.0.0.1
port=8826
size=12
bold=false
underline=false
forecolor=Black
font=Thoma
backcolor=White
```

מה שחשוב להקפיד אצלנו זה שה-*server_ip* יהיה מוגדר על ה-**local host** (127.0.0.1) משום שגם הלקוח וגם השרת רצים על אותה מכונה. ובנוסף שה-**port** יהיה זהה לפורט של השרת שלנו. בנוסף אפשר להגדיר כל מיני פרמטרים גרפיים (גופן טקסט, צבע רקע וכו').

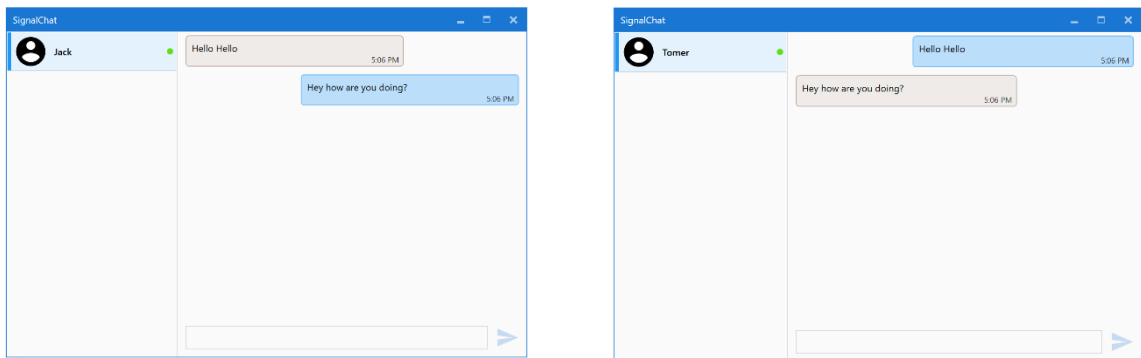
כדי להריץ את ההדגמה אפשר להשתמש בערכי ה-default שכבר רשומים בקובץ, נריץ את ה-EXE (כמו שהרצנו את התוכנה של השרת) ונוודא שהלקוח מצליח להתחבר לשרת אחרי שרשמנו את שם המשתמש ולחצנו "Login".



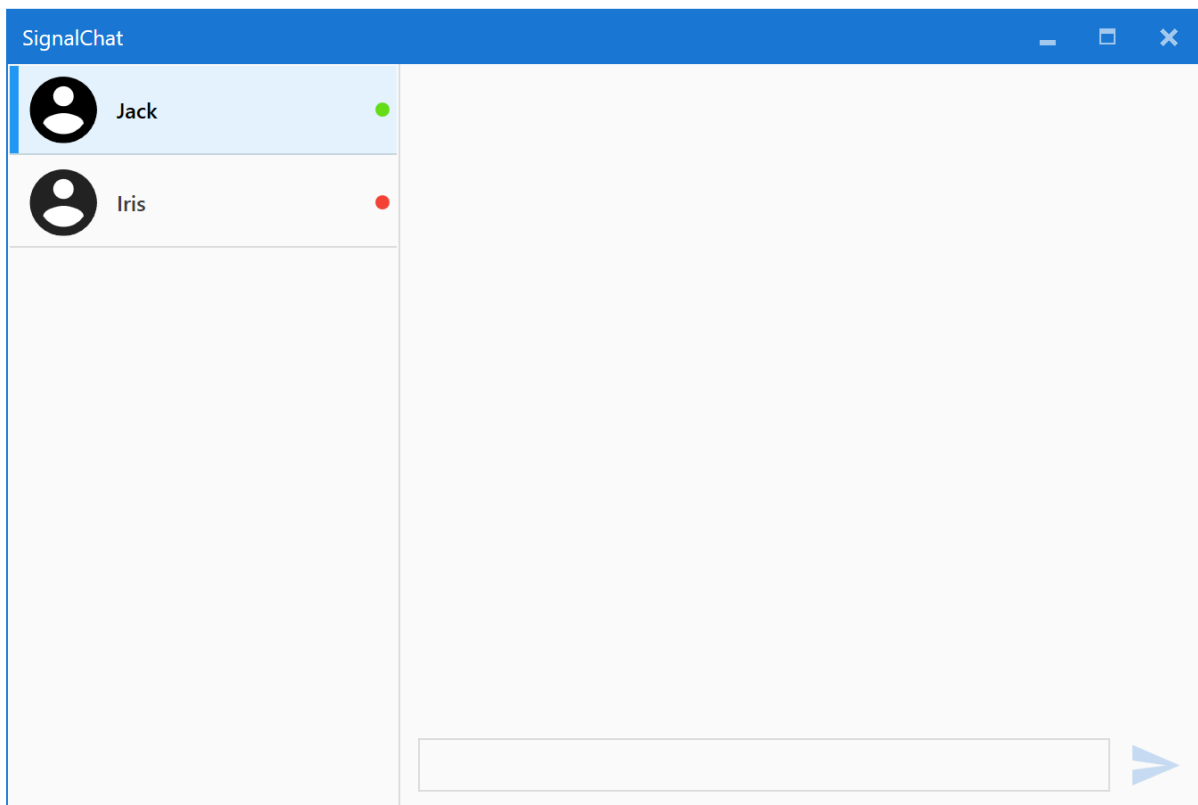
נריץ את ה-EXE פעם נוספת כדי לחבר לקוח נוסף, ונראה שהחלונות מתעדכנים



שני הלקוחות מחוברים לצ'אט, כל לקוח רואה את השני, ויכול לשלוח לו הודעות.



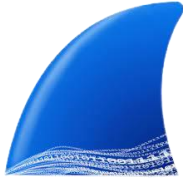
בעת לחיצה על משתמשים שונים, מתעדכן חלון ההודעות עבור משתמש זה. חלון ההודעות הנוכחי מתעדכן בlive כל **200ms**. עבור משתמשים מנותקים (צלמית אדומה) לא ניתן לשלוח הודעות, אך ניתן לצפות בתוכן הנוכחי של ההודעות.



בדוגמא הזו הייתה משתמשת בשם Iris שהתחברה לשרת, אחרי שהיא התנתקה כל שאר המשתמשים רואים צלמית אדומה ליד השם שלה.

שימוש ב-Wireshark כדי "להסניף" את התקשורת בין הלקוחות לשרת.

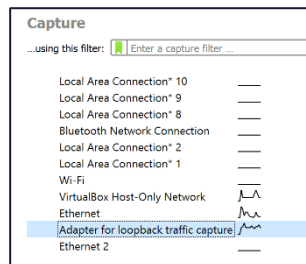
Wireshark היא תוכנה שכבר הכרתם/ בעבר, והיא יכולה מאוד לעזור לנו בתרגיל הזה. באמצעות Wireshark נוכל לוודא שההודעות בין השרת (שאנחנו נכתוב) לבין הלקוחות (שכבר בנו עבורנו) המוכנים נשלחו בצורה נכונה ובפורמט נכון.



מומלץ לעשות דוגמא קטנה, על הקבצים שכבר הרצנו.

במידה ו-Wireshark אינו מותקן על המחשב שלכם/ אפשר [להוריד מכאן](#).

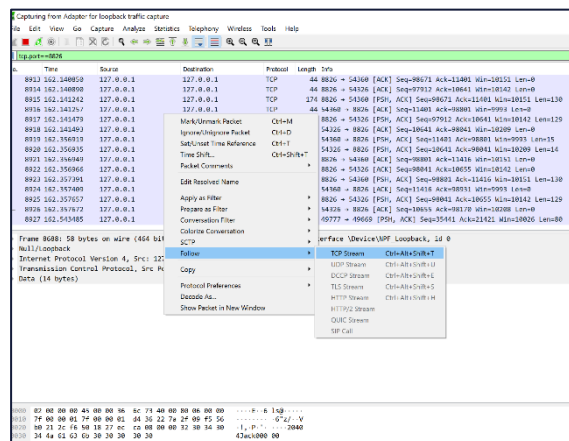
אחרי שסיימתם/ להתקין תריצו את התוכנה, ותבחרו בממשק בשם
"Adapter for loopback traffic capture"



עכשיו נתחיל לראות את הפקטות שנשלחו מתהליכים מקומיים במחשב שלנו. סיכוי לא קטן שחוץ מהתוכנה שהרצנו יש עוד תוכנות ש"מדברות" על ה-localhost, ולכן מומלץ לסנן לפי הפורט (8826).

Capturing from Adapter for loopback traffic capture						
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help						
tcp.port==8826						
No.	Time	Source	Destination	Protocol	Length	Info
604	10.781041	127.0.0.1	127.0.0.1	TCP	173	8826 → 54326 [PSH, ACK] Seq=6580 Ack=729 Win=10181 Len=129
605	10.781058	127.0.0.1	127.0.0.1	TCP	44	54326 → 8826 [ACK] Seq=729 Ack=6709 Win=10054 Len=0
606	10.781285	127.0.0.1	127.0.0.1	TCP	174	8826 → 54360 [PSH, ACK] Seq=6631 Ack=781 Win=10192 Len=130

עכשיו נלחץ קליק ימני על אחת הפקטות ואז "Follow → TCP Stream"



נוכל לראות את ההודעות שנשלחו בין הלקוחות דרך השרת.
בשורה הראשונה ישנה ההודעה הראשונה, ולאחר מכן השרת ממשיך לעדכן את הלקוחות, ככה שתמיד יהיה לכל לקוח את היסטוריית השיחות והמשתמשים המחוברים.

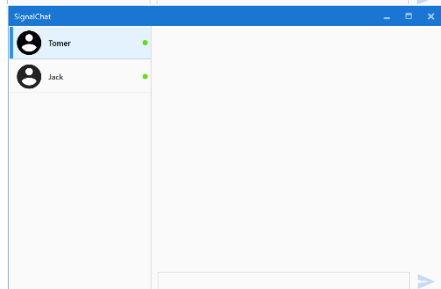
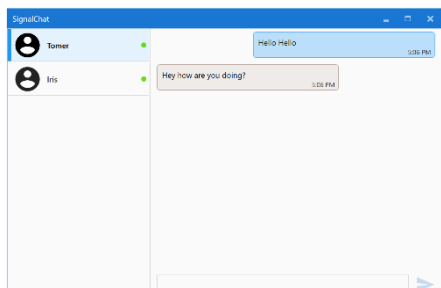
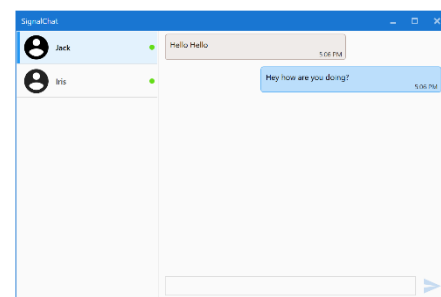
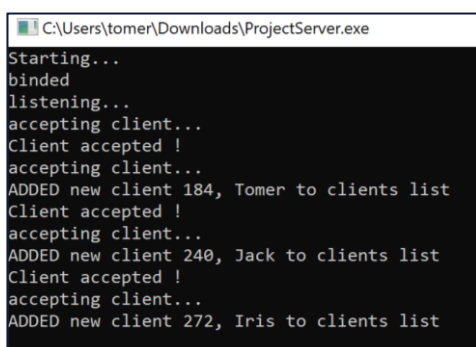


דוגמא נוספת,

נניח שהצטרפה משתמשת חדשה לתכנית, אם נעשה שוב TCP stream → follow נראה שהשרת בעת מעדכן את שאר הלקוחות על המשתמשת החדשה



וה-GUI (הממשק הגרפי) של כל לקוח מתעדכן בהתאם:



שלב 2: בניית המערכת


הרצת Demo	בניית המערכת (חיבור לקוחות)	אכיפת פרוטוקול ההודעות	בדיקות
-----------	--------------------------------	---------------------------	--------

בניית המערכת – שרת MT

השרת שנממש הינו שרת **multi-threaded** (אם כי לא ב-100%). השרת ישתמש **בתור הודעות** יחיד עבור כל הלקוחות, ידע להכניס הודעות לתור, ולהוציא הודעות מהתור ולטפל בהם. על השרת לשמור על ערוץ תקשורת (**socket**) עם כל לקוח שביקש להתחבר ולא התנתק, ולדעת לתת מענה לכל בקשת התחברות.

יש להגדיר thread לכל לקוח, שתפקידו יהיה לקבל הודעות מלקוח מסוים ולהכניס אותן לתור הודעות. ה-thread ייווצר לאחר שלקוח התחבר לשרת (כלומר עשינו **accept**).

נוסף על כך, יהיה thread נוסף אשר יטפל ב**הוצאת הודעות מהתור**. (חישבו היטב מתי ליצור אותו). לאחר ש-thread של לקוח כלשהו מכניס הודעה לתור הוא יידע (**notify**) את התור שמטפל בהודעות על כך שהתווספה הודעה חדשה בעזרת **std::condition_variable**.

שימו , קיים thread עבור כל משתמש שהתחבר, שמטרתו לקבל הודעות מהמשתמש ולהכניס אותם לתור גלובאלי של המערכת (משותף לכל המשתמשים). קיימת כאן בעיית סנכרון הקשורה לשימוש במבנה נתונים – תור הודעות. כמה thread-ים יצטרכו לגשת לתור ההודעות, חשבו על דרך בה בכל עת רק אחד מהthreads יוכל לגשת לתור.



רמז: כבר עשיתם/ן את זה בתרגיל הקודם...

דרישות נוספות

- בתכנית יש $n + 2$ thread-ים
 - **Connector thread** – שמקבל לקוחות (**accept**) בלולאה אינסופית, בכל פעם שהוא מקבל לקוח הוא מעביר את ה-socket שלו ל-thread חדש כדי שיטפל בלקוח.
 - **main thread** (הת'רד הראשי בתכנית) שמטפל בהודעות שנכנסות לתור ההודעות.
 - **Client thread * n** - מקבלים הודעות מכל לקוח ומכניסים אותן לתור ההודעות.
- חישבו היטב אילו **משאבים משותפים** יש לנעול. רשימת המשתמשים המחוברים לא אמורה להיות משאב משותף מבחינת הthread-ים משום שרק ה-thread הראשי רשאי לשנות אותה, לעומת תור ההודעות שהוא משאב משותף (!)
- במידה ויש חריגה ב-Client thread יש להתייחס לזה כאילו הוא התנתק מה-chat.
- במידה ושליחת ההודעה עבור client נכשלה או בכל חריגה אחרת בתקשורת מול לקוח, יש להתייחס לכך כאילו הלקוח התנתק מה-chat.

👉 הערה חשובה - Helper !

בשביל שלא תצטרכו להתעסק בדברים בסיסיים, נכתבה עבורכם/ן מחלקת **Helper**. במחלקה תוכלו למצוא פונקציות שימושיות (מאוד) שיחסכו ממכם/ן את הצורך לכתוב דברים מחדש. בנוסף, חלק מהפונקציות של המחלקה כתובות בצורה שמבטיחה קריאה נכונה מה-socket ועשוי לחסוך זמן רב.



בבקשה תעברו על המחלקה לפני תחילת המימוש של התרגיל

👉 הערה חשובה #2 – להשתמש בשרת מהשיעור !

מומלץ (מאוד!) להשתמש ב-main ובקבצי הקוד של השרת שהודגם בכיתה (Server.cpp, Server.h). אל תכתבו את כל החלק של ה-bind & listen בעצמכם/ן, תתחילו בקוד שהמדריך/ה הציג/ה בכיתה ושנו אותו ככה שיתמוך בחיבור של כמה משתמשים במקביל. אחרי שהצלחתם/ן לחבר כמה לקוחות במקביל הוסיפו מתודות בשביל המשך התרגיל. זכרו לכלול את הקבצים של ה-WSAInitializer וגם את ה-WSAInitializer.cpp, הוסיפו אותם לפרויקט ה-VS, בלעדיהם אחרי כל שגיאה הסוקט לא יצליח להתחבר שוב לאותו הפורט.

לבסוף, זכרו לעבור על דגשי הפיתוח שהיו בשקופיות האחרונות

אחרי שראינו **שכמה לקוחות יכולים להתחבר לשרת שלנו** (מבלי לשלוח הודעות עדיין),

רק אז מומלץ **לעבור לשלב הבא**

הערות נוספות

- העזרו בפתרון של תרגיל 12, זה יעזור לכם/ן להיזכר בחלק של ה-Condition Variable.
- בקשו מהמדריך/ה לפתוח את הפתרון במידה והוא לא ב-NEO.
- הוסיפו מחלקות במידת הצורך.
- מומלץ לשרטט את המערכת לפני שמתחילים לעבוד.
- נהלו את השגיאות של התכנית באמצעות exceptions, זה הדרך שבה מתנהלים בתעשייה ומומלץ להתנסות בה כמה שיותר.
- בסוף התרגיל יינתן לינק ל-repo עם בדיקות שאפשר להריץ כדי לבדוק את התקינות של השרת שלכם/ן, השתמשו בו.
- למשקיעניות והמשקיענים שבינינו, הקוד של צד הלקוח נכתב בשפת C# וזמין בתיקיית התרגיל. במידה ותרצו לשנות אותו ולא רק להריץ את ה-EXE, אז התיקייה זמינה ב-drive ועומדת



לרשותכם/ן. מומלץ לשנות את העיצוב המכוער של מגשימים

שלב 3: העברת הודעות בין לקוחות

בדיקות	אכיפת פרוטוקול ההודעות	בניית המערכת	הרצת Demo
--------	---------------------------	--------------	-----------

אחרי שבנינו את המערכת ודאגנו שהשרת שלנו יכול לשרת לקוחות במקביל, הגיע הזמן לאפשר שליחת הודעות בין הלקוחות

שמירת הודעות בין שני משתמשים ע"י השרת - Chat

קבצי שיחות

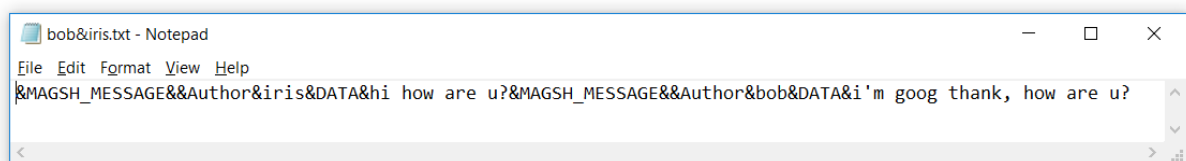
ההודעות בין שני משתמשים יישמרו **בקבצי טקסט** ע"י השרת בפורמט שייפורט בהמשך. עבור כל שני משתמשים שמעוניינים לתקשר ביניהם (כלומר שנשלחה לפחות הודעה אחת ביניהם), השרת ייצור קובץ שייקרא כך: נניח ששני שמות המשתמשים הם alice ו-bob, הקובץ ייקרא כך: *alice&bob.txt* ובו **יישמר כל תוכן השיחה** בין alice ו-bob. השם הראשון ייקבע לפי מיון אלפביתי של השמות. כלומר השם alice יופיע ראשון בשם הקובץ.
רמז: פונקציה sort של המחלקה `string`: `std::`.

פורמט קבצי שיחה

כל הודעה שתתקבל ע"י משתמש אחד לשני, תשמר במבנה הנ"ל שישורשר לקובץ.

`&MAGSH_MESSAGE&&Author<author_username>&DATA<message_data>`

יש להחליף את התגית `<author_username>` בשם משתמש אשר רשם את ההודעה. ואת התגית `<message_data>` בתוכן ההודעה עצמה. כל הודעה חדשה תשמר במבנה הזה, ותשורשר לקובץ הקיים, **במידה ולא קיים תיצור קובץ חדש**, לפי הגדרת שמות קבצי שיחה.
דוגמא לקובץ שיחה בין bob לבין iris:



קבצי שיחות

בהמשך יפורט פרוטוקול התקשורת בין הלקוח לשרת, כאשר נאמר שהשרת יישלח את **כל** תוכן הצ'אט בין שני המשתמשים ללקוח, הכוונה היא לשליחת תוכן קובץ השיחה המתאים. יש לקרוא את כל תוכן הקובץ ולשלוח אותו למשתמש הרלוונטי.

פרוטוקול תקשורת לקוח-שרת

הקדמה

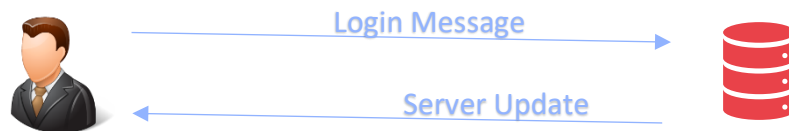
פרוטוקול התקשורת בין השרת שלנו ללקוחות הוא **פרוטוקול טקסטואלי** במלואו, הסיבה לכך היא נוחות פיתוח ונוחות דיבוג.

כל שדה הוא שדה טקסט, לדוגמא, כאשר מצוין לשלוח אורך של משתנה מסוים (נגיד אורך של שם משתמש), אז יישלח המספר הטקסטואלי שלו, כלומר בהסנפה נראה '2' ולא הייצוג הבינארי של הספרה 0x2. בנוסף לכך, במידה ומצוין אורך של שדה מסוים, לדוגמא השדה 'אורך של שם משתמש' אורכו 2 בתים אז צריך לעשות שימוש ב-2 הבתים, **במידת הצורך יש לעשות padding של '0' מצד שמאל**.

טיפ: מצורף קובץ packet capture לדוגמא (client_server_comm.pcap) של תעבורת לקוח-שרת.

התחברות

נגדיר את פרוטוקול **ההתחברות** של לקוח לשרת ההודעות. הלקוח יישלח לשרת **פקטת התחברות (login packet)** שתכיל את **השם משתמש** שלו, לאחר מכן השרת יישלח ללקוח **הודעת עדכון של השרת (server update message)**



Login Message

3 bytes	2 bytes	
200	Len(username)	Username

נסביר איך נראית פקטת התחברות לשרת. 3 הבתים הראשונים הם קוד התחברות – הוא תמיד יהיה **200**. לאחר מכן 2 הבתים הבאים הם **האורך** של השם משתמש (שם המשתמש הוא השדה השלישי בפקטה).

דוגמא

אם שם המשתמש הוא "bob" אז השדה השני יהיה '03' – padding של 0 מצד שמאל על מנת שיתאים לפרוטוקול. השדה השלישי הוא השם משתמש עצמו והוא בעל אורך משתנה – כאמור השדה השני בפקטה קובע את אורכו של השדה השלישי. **דוגמא לפקטה כזאת: 20003bob**.

ServerUpdateMessage

מדובר בפקטת שרת גנרית ולה יש מספר מטרות. בשלב ההתחברות, המטרה שלה היא לאשר התחברות תקינה, ולשלוח את רשימת המשתמשים שכרגע מחוברים לשרת, בשלב שליחת ההודעות, המטרה של פקטה זו היא לעדכן על משתמשים חדשים שהתחברו, וכאלה שהתנתקו, ולעדכן על הודעות חדשות שנשלחו למשתמש. לכן, ישנם שדות רבים, ולהם שימושים שונים עפ"י השלב שבו נמצא המשתמש.

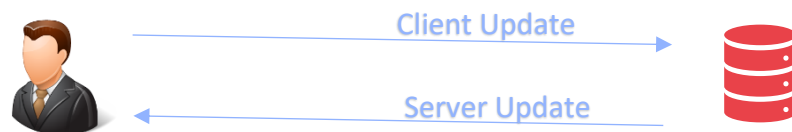
3b	5b		2b		5b	
101	Len_chat content	chat_content	len_par_user_name	par_username	len_all_usernames	all_usernames

נפרוט את השדות וערכם בשלב ההתחברות:

- **קוד הודעה** – השדה הראשון הוא קוד הודעת עדכון של השרת – תמיד יהיה 101 ואורכו 3 בתים
- **Len_chat_content** – בשלב ההתחברות יהיה 0, כלומר: 00000 כיוון שאורכו צריך להיות 5 בתים
- **Chat_content** – בשלב ההתחברות שדה זה יהיה ריק, אורכו נקבע לפי השדה len_chat_content ולכן צריך להיות בעל אורך 0 == ריק, (כלומר לא יופיע בפקטה)
- **Len_per_user_name** – בשלב ההתחברות יהיה 0, כלומר 00 כי אורכו צריך להיות 2 בתים.
- **Par_user_name** – בשלב ההתחברות יהיה ריק. (כלומר לא יופיע בפקטה)
- **Len_all_usernames** – אורך של כל השמות משתמשים משורשרים אחד אחרי השני. הסימן & מפריד בין שמות משתמשים – אורך השדה יהיה תמיד 5 בתים.
- **All_username** – כל שמות המשתמשים שמחוברים כעת, עם הסימן & כסימן מפריד. לדוגמא: שמות המשתמשים שמחוברים כעת הם alice ו-Bob. לכן השדה יראה כך: alice&bob, ערך השדה הקודם len_all_usernames – 00009.

שליחת הודעות ועדכון הלקוח

שלב זה קורה לאחר שהלקוח יתחבר לשרת, ומטרתו לעדכן את השרת עם הודעות חדשות, קבלת רשימת משתמשים עדכנית, ושליחת הודעות.



הלקוח יוזם בקשת **ClientUpdateMessage**, והשרת עונה לו ב- **ServerUpdateMessage**.

ClientUpdateMessage

3b	2b		5b	
204	Len_second_user	second_username	len_new_message	new_message

ההודעה זו משמשת עבור שליחת הודעות חדשות למשתמשים, וגם עבור בקשת הודעת עדכון מהשרת (ServerUpdateMessage).

נפרוט את השדות בפקטה:

- **קוד הודעה** – עבור הודעת ClientUpdateMessage קוד ההודעה הוא 204 ואורכו הוא 3 בתים
- **Len_second_user** – מספר תווים בשם המשתמש אליו נרצה לשלוח הודעה. אורך השדה הוא 2 בתים, כאמור עם padding של אפסים משמאל במידת הצורך.
- **Second_username** – שם המשתמש אליו נרצה לשלוח הודעה, אורך משתנה, נקבע לפי השדה len_second_user
- **Len_new_message** – אורך של ההודעה החדשה אותה נרצה לשלוח, אורך השדה הוא 5 בתים, עם padding של אפסים משמאל במקרה הצורך.
- **New_message** – אורך משתנה, נקבע עפ"י השדה len_new_message.

השרת יענה על הודעה ClientUpdateMessage עם הפקטה ServerUpdateMessage. בסעיף ההתחברות קיים פירוט על פורמט הפקטה, כעת היא תשלח עם השדות הבאים, תוכן השדות תלוי בהודעות שהגיע מ client - ClientUpdateMessage:

- **קוד הודעה** – השדה הראשון הוא קוד הודעת עדכון של השרת – תמיד יהיה 101 ואורכו 3 בתים
- **Len_chat_content** ו- **chat_content** – תוכן הצאט (תוכן הקובץ) בין המשתמש ששלח את הבקשה, ולבין שם המשתמש שמופיע בשדה **second_username**. אם השדה ריק (באורך 0), או שלא קיים משתמש כזה במערכת, תוכן צ'אט ריק בגודל 0.
- **Len_per_user_name** ו- **par_user_name** – יכול **second_username** במידה וקיים, אחרת שדה בגודל 0.
- **Len_all_usernames** – אורך של כל השמות משתמשים משורשרים אחד אחרי השני. הסימן & מפריד בין שמות משתמשים – אורך השדה יהיה תמיד 5 בתים.
- **All_username** – כל שמות המשתמשים שמחוברים כעת, עם הסימן & כסימן מפריד. לדוגמא: שמות המשתמשים שמחוברים כעת הם alice ו-Bob. לכן השדה יראה כך: alice&bob, ערך השדה הקודם len_all_usernames – 00009.

התחברות

ההתנקות היא סגירת האפליקציה כלומר סגירת ה TCP stream של הלקוח. לא קיימת הודעה מיוחדת לשם כך, אלא סגירת ה socket של המשתמש. על השרת לזהות את התנתקות המשתמש ולעדכן את רשימת המשתמשים המחוברים כעת בצד השרת. כאמור על השרת לשמור את רשימת המשתמשים המחוברים בכל עת.

שלב 4: בדיקות

הרצת Demo	בניית המערכת	אכיפת פרוטוקול ההודעות	בדיקות
-----------	--------------	---------------------------	--------

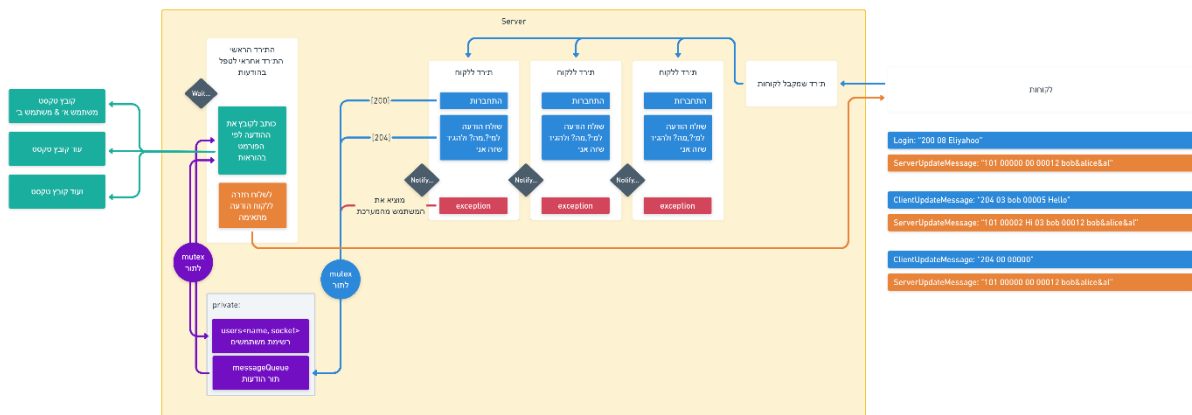
בדיקת השרת

כפיילוט נכתב פרוייקט בפיייתון אשר בודק את קוד השרת שכתבתם, הפרוייקט עלה לתיקיייה של התרגיל, וניתן למצוא אותו [בקישור הבא](#)

בנוסף, הפרויקט נמצא בgit: <https://gitlab.com/tomeriq/magshimimex13tests.git> - הבדיקות מדמות תקשורת שרת-לקוח, ובדקות שכל הלוגיקה ממומשת נכון בצד שרת. אין חובה להשתמש בו, והוא נועד ככלי עזר על מנת לבדוק שהמימוש נכון. התחילו מלקרוא את הקובץ README

תיאור מערכת

הנה תיאור נוסף של המערכת שיכול לעזור להבין איך הדברים מתחברים ביחד (את התמונה ניתן למצוא בתיקיית התרגיל)




בהצלחה!

נספחים

הגשה ב-GIT

- את הפרויקט יש לנהל ב-Git, לפתוח repository חדש בתוך קבוצת ה-gitlab שלנו ושל המדריך/ה, ולהגיש לינק לפרויקט ב-NEO (אפשר לעשות comment עם הלינק או להגיש מסמך txt עם הלינק בפנים).
- יש להעלות ל-repository את כל הקבצים הרלבנטיים לתרגיל (קבצי txt, מסמכים, ומשאבים אחרים שבהם השתמשנו).
- חשוב להעלות את פרויקט ה-Visual Studio השלם ולהתעלם מקבצים לא נחוצים ([הנחיות במסמך הבא](#)), במידה ולא הועלה הפרויקט השלם, אין להעלות את שאר הקבצים שיוצר Visual Studio – הם רבים מאוד, הם לא מכילים מידע נחוץ להרצת הפרויקט אצל המדריך, ורק יוצרים בלגן.
- הבחירה אילו קבצים להעלות ל-repository נעשית באמצעות הפקודות add ו-rm. אופציה נוספת (מומלצת) היא להוסיף קובץ .gitignore. אשר יתעלם מהקבצים הלא נחוצים. במידה ותרצו תוכלו להיעזר ב[סרטוני עזר בנושא GIT](#).
- כסיימתם/ן, בדקו שניתן להריץ את הפרויקט בקלות – בצעו Clone אל תיקייה במחשב אשר שונה מזו שעבדתם/ן, ותראו שהפרויקט נפתח ע"י לחיצה על קובץ ה-sln ויכול לרוץ בלי בעיה

 Ex13Server.sln

דגשים:

- את הפרויקט יש לפתוח בקבוצת ה-gitlab שאליה משותף/ת המדריך/ה כ-Maintainer.
- יש לוודא שכל הקבצים הרלבנטיים נוספו ל-repository (באמצעות הפקודה add), במידת הצורך ניתן להוריד קבצים מיותרים (באמצעות הפקודה rm)
- יש לבצע commit עבור כל סעיף, ובנקודות שבהן הוספנו שינויים חשובים (לפי הדגשים שהועברו בכיתה).
- עבור כל commit, זכרו לכתוב הודעה קצרה ואינפורמטיבית, שאפשר יהיה להבין מה היה השינוי בקוד.
- יש לדחוף את הקוד (באמצעות הפקודה push) ל-repository בסיום העבודה שלנו, חשוב שבסיום העבודה שלנו, ובמידה ונפנה למדריך/ה, ב-repository יהיה הקוד המעודכן ביותר.
- במידה ושכחנו או שאנחנו לא בטוחים איך מעלים קובץ, או מתעלמים מקבצים, כדאי לצפות בסרטוני ההדרכה בנושאי GIT. ניתן לגשת לסרטונים בלשונית ה-resources שבכיתה ה-NEO.
- בסיום העבודה יש להגיש לכיתה ה-NEO קישור ל-repository.

כללי

1. יש לבדוק שכל המטלות מתקמפלות ורצות ב-VS2022. מטלה שלא תעבור קומפילציה אצל הבודק לא תיבדק **והניקוד שלה יהיה 0** 😞
2. יש לבדוק שהקוד שכתבתם עובד. יש להריץ בדיקות שלכם ולוודא שהקוד ברמה טובה.
3. כאשר אתם מתבקשים לממש פונקציה, ממשו בדיוק את הנדרש. אין להוסיף הדפסות וכדו'. אם הוספתם תוך כדי הבדיקות שלכם הדפסות, אנא דאגו להוריד אותם לפני ההגשה.
4. להזכירכם! העבודה היא עצמית, ואין לעשות אותה ביחד.
5. על כל שאלה או בעיה יש לפנות למדריך, לפחות 36 שעות לפני מועד ההגשה.

דגשים לתכנות נכון

- כדאי לקמפל כל מספר שורות קוד ולא לחכות לסוף! הרבה יותר קל לתקן כאשר אין הרבה שגיאות קומפילציה. בנוסף קל יותר להבין מאיפה השגיאות נובעות.
- כדאי לכתוב פונקציה ולבדוק אותה לפני שאתם ממשיכים לפונקציה הבאה. כלומר, כתבו תכנית ראשית שמשתמשת בפונקציה ובודקת האם היא עובדת כראוי. חישבו על מקרי קצה ונסו לראות מה קורה.
- בכל פעם שאתם מתקנים משהו, זכרו שיכול להיות שפגעתם במשהו אחר. לכן עליכם לבדוק שוב מהתחלה.
- חשפו החוצה רק את הממשק המינימלי הדרוש (minimal API), הגדירו את שדות המחלקה כפרטיים, וכמה שפחות מתודות כציבוריות.