2016

צ'אט מאובטח

פרויטק 5 יח"ל בהנדסת תכנה

מאת איתמר רייף
(ת.ז. 206589764)
מנחה: שלומי אחנין

תוכן העניינים

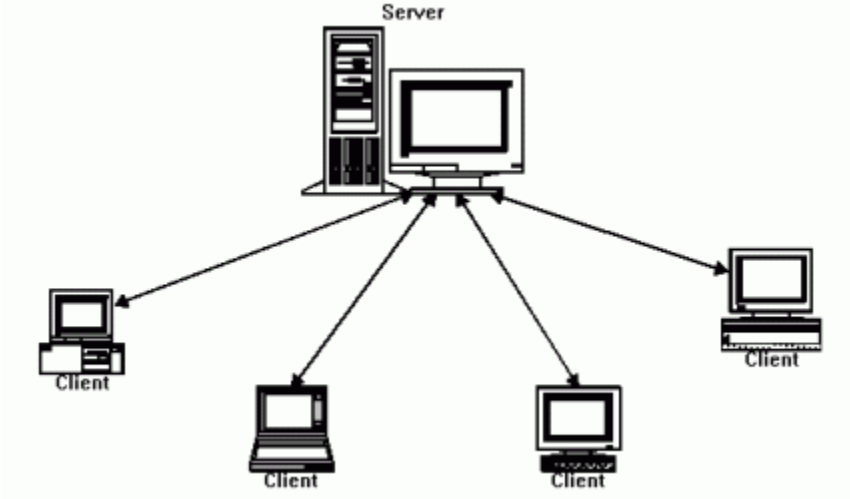
מבוא ומטרת הפרוייקט

בימינו, כולם משתמשים באינטרנט לכל מיני דברים – מרכישת ממתקים יפניים ועד למסחר בבורסה. כמעט כל תקשורת אינטרנטית מצריכה סוג מסוים של אבטחה – כדי להגן על האנשים הטובים מהאנשים הרעים שרוצים לפגוע בהם. קיימים סוגים שונים של אבטחה – הצפנה, ווידוי, זיהוי וכו'. בעבודתי התמקדתי בסוג מסוים הוא SSL.

בחרתי לעסוק בנושא הרשתות והאבטחה, נושא הסייבר, משום שבעולמנו המתקדם במהירות נוצרות עוד ועוד סכנות וצריך אנשים שידעו להגן עלינו מפניהן. זהו נושא שמרתק אותי ורציתי להתחיל ללמוד אותו כבר בבית הספר, לעומת לחכות עד האוניברסיטה.

מטרת הפרויקט היא ליישם תקשורת מאובטחת ע"י SSL על פי פרוטוקול ה-TCP/IP. מטרה זו מושלמת באמצעות תכנית צ'אט בין שני משתמשים. צ'אט הינו תכנית פשוטה המסיימת תקשורת בין לקוח לשרת ללקוח ומהווה דוגמא מושלמת למקרה בו יהיה צורך באבטחה – כדי לשמור על פרטיות המשוחחים ומניעת פגיעה בשרת ודרכי בכל הלקוחות (משתמשים).

בפרוייקט זה אדגים תכנית צ'אט המבוססת על פרוטוקול TCP/IP ומאובטחת באמצעות SSL. התקשורת המודגמת הינה בין לקוח לשרת ובין שרת ללקוח.



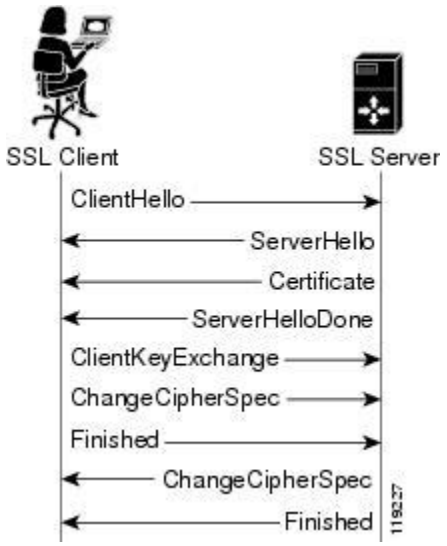
תקשורת בין שרת ללקוחות ובין לקוחות לשרת

מהו SSL?

כאשר אנו רוצים לתקשר באופן מבוטח אנו מיד חושבים כי הפתרון הסופי הינו הצפנה, כך למנוע מהמאזין לצוטט על ידי מניעתו מלהבין למה הוא מצוטט. אך קיים עוד קריטריון לתקשורת מבוטחת – זיהוי המוען. זיהוי כי המוען הוא מי שאתה חושב שהוא מבטיח שאתה מתקשר עם הצד השני בצורה יציבה וכי אינך מקבל או מוסר תקשורות לגורם שאינך מעוניין לתקשר איתו (לדוגמא, בשימוש בכרטיס אשראי באינטרנט – וידוי שהשרת אליו נשלחים פרטיך הוא השרת אליו אתה רוצה לשלוח אותם ולא גנב).

SSL הינו פרוטוקול תקשורת ואבטחה העונה על שני הקריטריונים – זיהוי והצפנה. אבטחת SSL מסתמכת על זיהוי שני הצדדים באמצעות תעודה – Certificate שאותה מחזיק השרת ואותה מאשר הלקוח לפני תחילת התקשורת. תהליך זה נקרא לחיצת היד – Handshake:

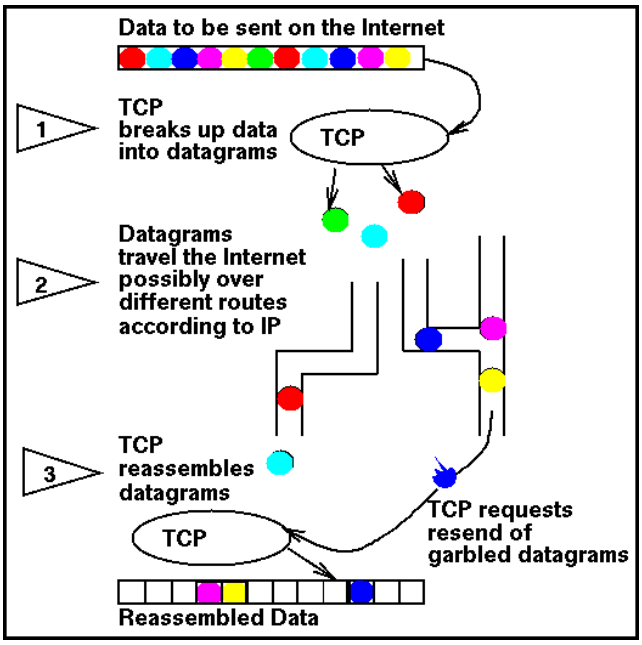
1. הלקוח שולח הודעת שלום – ClientHello.
2. השרת שולח הודעת שלום – ServerHello.
3. השרת משתף את תעודתו – Certificate.
4. השרת שולח הודעת שלום גמור – ServerHelloDone.
5. הלקוח שולח הודעת החלפת מפתחות הצפנה – ClientKeyExchange.
6. הלקוח שולח הודעת שינוי הצופן (ביסוס פרוטוקול הצפנת התקשורת) – ChangeChiperSpec.
7. הלקוח שולח הודעת סיום – Finished.
8. השרת שולח הודעת שינוי הצופן – ChangeCipherSpec.
9. השרת שולח הודעת סיום – Finished.



לחיצת היד של SSL

מהו TCP/IP?

TCP/IP הינו פרוטוקול תקשורת אינטרנטית בסיסי (יכול גם לשמש רשת פרטית). הפרוטוקול מורכב משני רבדים. רובד ה-TCP הוא הגבוה יותר, המחלק את ההודעה לחבילות קטנות המשודרות דרך האינטרנט ומתקבלות על ידי רובד TCP שמחבר אותן להודעה המקורית. רובד ה-IP (Internet Protocol) המטפל בכתובת שאליה כל חבילה מיועדת כדי שהחבילות יגיעו למקום הנכון. כל מחשב המחובר ברשת מקבל את החבילות ומעביר אותן לנמען הנכון שם הן מחוברות מחדש להודעה המקורית.



תהליך העברת התקשורת ב-TCP/IP

TCP/IP עושה שימוש במודל השרת/לקוח של תקשורת, לפיו מחשב הלקוח מבקש ומקבל שירות ממחשב השרת באותה הרשת. TCP/IP הינו פרוטוקול תקשורת מנקודה לנקודה, כלומר, קשר מנקודה אחת (לקוח או שרת) אל נקודה אחרת ברשת (לקוח אל שרת, שרת אל לקוח, שרת אל שרת).

הפרוטוקול מאפשר שימוש ותקשורת בין טווח רחב של מחשבים, בהגבלות טכניות ופיזיות מועטות, בשל צורת הפעולה שלו – הוא מתייחס לתקשורת כאל הרבה חבילות קטנות במקום כחבילה אחת גדולה ולכן יכול לשלוח תקשורות ממגוון רחב יותר (רוחב הפס מהווה הגבלה קטנה יותר על התקשורת).

הסבר האלגוריתם – איך הקוד עובד?

לפרוייקט שני מרכיבים עיקריים – השרת והפרוייקט. שני המרכיבים רצים במקביל ומתקשרים אחד עם השני.

פעולת השרת

בהפעלת התכנית טוען השרת את נתוני המשתמשים הרשומים מתוך קובץ ה-***users.dat*** (באמצעות ***FileStream***) לתוך מילון (dictionary) בשם Users. לאחר מכן השרת מחכה לקבלת תקשורת מלקוח, איתו הוא מבסס חיבור על thread חדש – זאת על מנת שיהיה מסוגל להתמודד עם כמה חיבורים במקביל (וכך לתקשר אם מספר לקוחות במקביל). החיבור נהפך להיות חיבור מאובטח SSL באמצעות התעודה בקובץ המצורף ***serverCert.pfx*** ונשלח ללקוח הודעות 'שלום' (הערך הקבוע `IM_Hello`).

עם קבלת הודעת 'שלום' בחזרה מקבל השרת מן הלקוח שם משתמש, סיסמא ומצב הרשמה (האם הנתונים המתקבלים הם עבור משתמש חדש או משתמש שכבר קיים). *במידה ולא התקבלה תשובה נסגר החיבור.

במידה והנתונים הם עבור משתמש חדש מוסיף השרת אותם ואת נתוני החיבור עם הלקוח למילון Users ומעדכן את הקובץ ***users.dat***. במידה והנתונים הם עבור משתמש קיים (והסיסמא תואמת את זו השמורה ב-Users) משנה השרת את נתוני החיבור המאוחסנים ב-Users לאלו של החיבור הנוכחי. במידה ואין בעיות שולח השרת הודעת אישור (הערך הקבוע IM_OK).

לאחר מכן נכנס השרת ללולאה המחכה לתקשורת של חבילה מהלקוח.

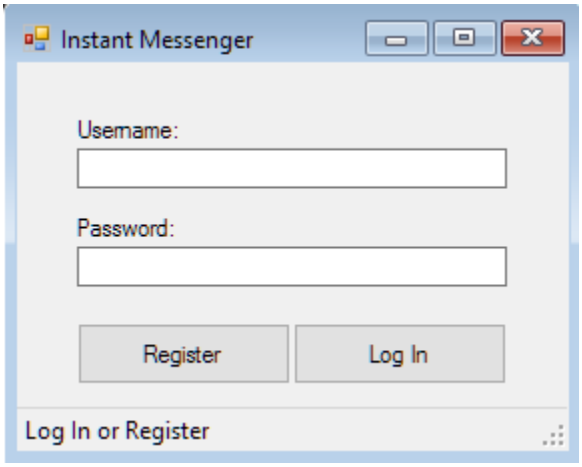
במידה והחבילה שהתקבלה היא הערך הקבוע IM_IsAvailable מקבל השרת עוד מחרוזת – שם המשתמש של המשתמש אותו בודק השרת. לאחר הבדיקה השרת מחזיר את הערך ***true*** במידה והמשתמש מחובר ו-***false*** במידה ואינו מחובר (או אינו רשום ב-Users).

כאשר החבילה המתקבלת היא הערך הקבוע IM_Send מקבל השרת עוד שתי מחרוזות מהלקוח – הראשונה מייצגת את שם הנמען והשנייה את ההודעה. השרת בודק האם הנמען מחובר (באמצעות Users). במידה וכן - ממשיך ומעביר את ההודעה לנמען.

כאשר נסגר החיבור עם הלקוח משנה השרת את הערך המייצג את מצבו של הלקוח ב-Users ללא מחובר.

פעולות הלקוח

החלון הראשון שהמשתמש רואה הוא חלון ה-Log In/Register.



חלון ה-Log In/Register

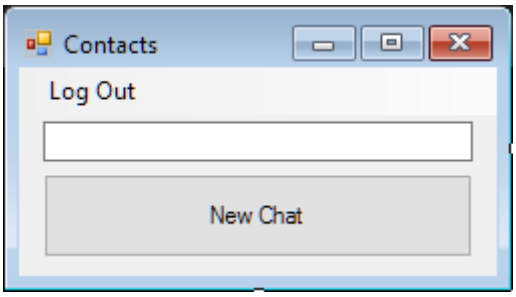
משתמש מכניס את שם המשתמש והסיסמא שלו ובוחר האם ברצונו להירשם כמשתמש חדש או להתחבר למשתמש קיים.

עם לחיצת אחד הכפתורים (***Log In/Register***) נפתח חיבור הלקוח עם השרת. לאחר שהלקוח מוודא שאין בעיות התעודה של השרת, וכי התעודה הונפקה עבור ***"Itamar Reif"*** שולח הלקוח לשרת הודעת 'שלום' (הערך הקבוע IM_Hello) ומחכה לתשובה.

לאחר קבלת תשובה שולח הלקוח לשרת את מצב ההרשמה (האם המשתמש מעוניין ליצור משתמש חדש או להתחבר למשתמש קיים), את שם המשתמש והסיסמא ומחכה לתשובה מהשרת. *במידה ולא התקבלה תשובה נסגר החיבור.

במידה והתשובה היא שלילית (חבילת error מתקבלת) מעביר הלקוח את ה-error המתאים לאירוע המתאים (OnRegisterFailed או OnLoginFailed בהתאמה למצב ההרשמה) והודעת error (בצורת Message Box) עולה למשתמש.

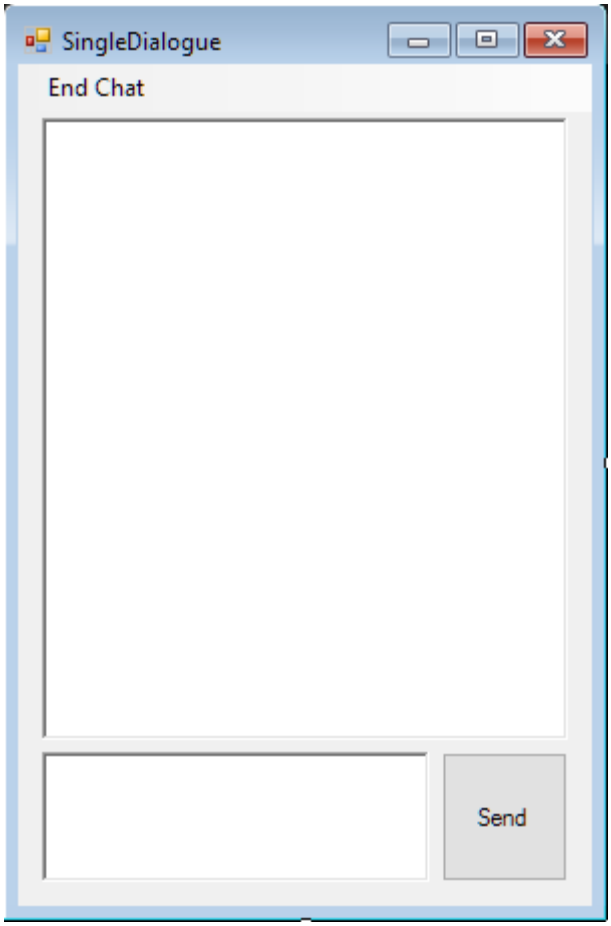
במידה והתשובה היא חיובית (הערך הקבוע IM_OK) ממשיך הלקוח ומפעיל את האירוע המתאים (OnRegisterOK או OnLoginOK בהתאמה למצב ההרשמה), זאת כדי לתקשר עם הממשק ולעדכן את חלונות המוצגים על המסך. האירועים הללו סוגרים את מסך ה- Log In/Register ופותחים את חלון ה-Contacts, בו המשתמש מקליד את המשתמש השני איתו רוצה לפתוח בשיחה. לאחר מכן ממשיך הלקוח ללולאה המחכה לקבלת חבילה מהשרת.



*חלון ה-Contacts**

במידה והחבילה המתקבלת היא הערך הקבוע IM_IsAvailable מקבל הלקוח עוד שני ערכים – מחרוזת וערך בוליאני מהשרת, אותם הוא מעביר לממשק באמצעות האירוע OnUserAvail, המעדכן את מצבו של חלון ה-SingleDialogue.

במידה והחבילה המתקבלת היא הערך הקבוע IM_Received מקבל הלקוח עוד שתי מחרוזות – שם השולח והודעה, אותם הוא מעביר לממשק באמצעות האירוע OnMessageRec, המשתמש לעדכון חלון ה-SingleDialogue.



*חלון ה-SingleDialogue***

*כאשר המשתמש לוחץ על כפתור ה-New Chat נפתח חלון SingleDialogue חדש על thread חדש. זאת כדי לאפשר למשתמש לשוחח עם מספר משתמשים שונים במקביל. במידה והמשתמש מנסה לשוחח עם עצמו או עם משתמש שהשיחה איתו כבר פתוחה יקבל הודעת error.

כאשר המשתמש לוחץ על כפתור ה-Log Out בחלון ה-Contacts מתנתק חיבור הלקוח אל השרת ונסגרת התוכנה (כולל כל חלונות ה-SingleDialogue הפתוחים).

**כאשר נפתח חלון ה-SingleDialogue מופעלת פונקציית ה-timer, המאפשר לתכנה לחזור על פעולה עבור כל "Tick" – במקרה זה השימוש הנעשה הוא רענון החיבור – כל עוד המשתמש השני אינו מחובר לא יהיה ניתן לשלוח הודעות, וכאשר יתחבר תעלה הודעה כי הוא מחובר.

כאשר לוחץ המשתמש על כפתור ה-Send שולח הלקוח לשרת חבילה המכילה את הערך הקבוע IM_Send (כדי לסמן לשרת שהמידע שהוא עומד לקבל הוא עבור שליחת הודעה) ועוד 2 מחרוזות – שם הנמען וההודעה.

כאשר לוחץ המשתמש על כפתור ה-End Chat נסגר.

בעיות במהלך הפרויקט ופתרונן

התחלת הפרוייקט והלימוד העצמי היו אתגר חדש שמעולם לא התמודדתי איתו. הידע הקודם שלי בנושא היה אפסי, והניסיון שלי בלימוד עצמי לא היה קיים. לאחר הרבה ניסוי וטעייה החלטתי לבסוף על רעיון סופי לפרוייקט, אותו יכולתי ללמוד ולהבין בעצמי.

תהליך הלימוד העצמי היה קשה משום שלא הייתה לי מסגרת או הדרכה לגבי מה אני צריך ללמוד. לאחר מחקר מעמיק באינטרנט ובפורומים כמו stackoverflow.com הצלחתי להרכיב לעצמי מסגרת אחרי עקבתי ואותה למדתי.

במהלך הלימודים העצמיים מצאתי ונעזרתי במספר פרוייקטים מוכנים ב-codeproject.com. למדתי לעומק את צורת הפעולה והמחלקות הבאות לידי שימוש ואת הדרכים השונות והרבות ליצירת פרוייקט צ'אט מאובטח.

כמו כן, סביבת העבודה של WinForms הייתה זרה לי. שעות של לימודים ומחקר בנושא הובילו אותי ללמוד על נושאים מרתקים כגון threads, delegates ו-events.

הלימוד העצמי לא היה פשוט ואיתו עלו אתגרים ותקלות רבים. דוגמא לכך היא מקרה בו רציתי ליצור מצב בו כפתור בטופס מסויים סוגר טפסים אחרים, אך נתקלתי בבעיה של ניהול ה-thread הפותח לעומת הסוגר. בסופו של דבר למדתי כי קיימת פונקציה הסוגרת את כל חלונות התוכנה והשתמשתי בה כדי להתגבר על אתגר זה.

הוראות הפעלה

עם הספר מצורפים שתי תיקיות – תיקיית ה-Client ותיקיית ה-Server. כדי להפעיל את הפרוייקט יש להפעיל ראשית כל את קובץ ה-exe. של השרת (ולוודא כי תעודת ה-SSL, ***serverCert.pfx*** אכן נמצאת באותה תיקייה עם קובץ ה-exe.). לאחר הפעלת השרת יש להפעיל את תכנית הלקוח ולעקוב אחרי התכנה (להירשם/להתחבר בטופס ה- Log In/Register המתאים ולהקליד את הנמען איתו אתם מעוניינים לדבר בטופס ה-Contacts).

יש לציין כי חובה להריץ את השרת והלקוחות על אותו המחשב. זאת משום שכתובת ה-IP של השרת קבועה כ-"127.0.0.1" או "localhost" בגלל שאין ברשותי מחשב שאני יכול להגדיר כשרת ובו לפתוח את ה-ports הדרושים ב-router וכך ליצור שרת בעל גישה לאינטרנט. עם זאת, חשוב להדגיש כי אפשרות זו אינה רלוונטית אך ורק מבחינה לוגיסטית וכי במקרה אחר ואידיאלי יותר השרת היה בעל גישה לאינטרנט והיה ניתן לתקשר בין משתמשים שונים על מחשבים שונים ברשתות שונות (המחוברות לאינטרנט).

חלק I: שרת

המחלקה PROGRAM

מחלקה זו מהווה את פעולת תכנת השרת, התחלתה וסגירתה.

משתנים

Server	התהוות חיבור ה-TcpListener ע"פ ה-IP, Port הנתונים
Users	מילון המשתמשים הקיימים. נטען מתוך הקובץ ***users.dat*** ע"י הפעולה LoadUsers.
ServerCert	תעודת ה-SSL של השרת

פעולות

Program	הפעולה הבונה. מהווה את משתנה ה-Server ומפעיל אותו (Listen).
Listen	מחכה לקבלת TcpClient לחיבור ה-Server ומתחיל את הטיפול בו.
SaveUsers	שומר את נתוני המשתמשים החדשים לתוך ***users.dat***.
LoadUsers	טוען את נתוני המשתמשים מתוך ***users.dat*** למילון Users.

המחלקה USERINFO

מחלקה זו מהווה את צורת האחסון של נתוני המשתמשים במילון Users.

משתנים

Username	שם המשתמש
Password	סיסמא
LoggedIn	האם המשתמש מחובר כרגע?
Connection	נתוני החיבור והטיפול של השרת בלקוח

המחלקה CLIENT

מחלקה זו מהווה את החיבור של השרת אל הלקוח ואת הטיפול בו.

משתנים

prog	תכנית השרת שמפעיל את ה-Client הנוכחי
Client	נתוני לקוח ה-TCP

Ssl	חיבור ה-SSL המאובטח
Br	האובייקט הכותב מידע לחיבור ה-SSL
Bw	האובייקט הקורא מידע מחיבור ה-SSL
userInfo	נתוני המשתמש של הלקוח הנוכחי
IM_Hello	ערך קבוע המשמש כ-packet להודעת 'שלום'
IM_OK	ערך קבוע המשמש כ-packet לווידוי פעולה תקינה של תהליך ההתחברות/הרשמות של הלקוח
IM_Login	ערך קבוע המשמש כ-packet לסימון כי הפעולה הבאה היא התחברות משתמש קיים
IM_Register	ערך קבוע המשמש כ-packet לסימון כי הפעולה הבאה היא הירשמות משתמש חדש
IM_TooUsername	ערך קבוע המשמש כ-packet להודעת error כי שם המשתמש ארוך מדי
IM_TooPassword	ערך קבוע המשמש כ-packet להודעת error כי הסיסמא ארוכה מדי
IM_Exists	ערך קבוע המשמש כ-packet לסימון כי המשתמש כבר קיים במערכת
IM_NoExists	ערך קבוע המשמש כ-packet לסימון כי המשתמש אינו קיים במערכת
IM_WrongPass	ערך קבוע המשמש כ-packet להודעת error כי הסיסמא שגויה
IM_IsAvailable	ערך קבוע המשמש כ-packet לסימון כי המשתמש זמין
IM_Send	ערך קבוע המשמש כ-packet לסימון כי הפעולה הבאה היא שליחת הודעה
IM_Received	ערך קבוע המשמש כ-packet לסימון כי ההודעה התקבלה

פעולות

SetupConn	אתחול החיבור בין השרת ללקוח, מצד השרת ופעולת ההתחברות של המשתמש.
CloseConn	סגירת החיבור בין השרת ללקוח, מצד השרת ופעולת ההתנתקות של המשתמש.
Receiver	אתחול הלולאה המקבלת חבילות מהלקוח והטיפול בהן

המחלקה LOGREGFORM

מחלקה זו מתארת את הטופס LogRegForm ואת התנהגותו.

משתנים

im	החיבור בין המשתמש (הלקוח) לשרת
usernameInput	שם המשתמש שנשלח לשרת
passwordInput	הסיסמא שנשלחת לשרת

פעולות

loginButton_Click	טיפול באירוע הלחיצה על כפתור ה-Login. שולח לשרת את פרטי ההתחברות ומתחיל את החיבור.
OnIm_LoginOK	טיפול באירוע ה-LoginOK. סוגר את חלון ה-Log In/Register ופותח את חלון ה-Contacts.
OnIm_LoginFailed	טיפול באירוע ה-LoginFailed. מעלה הודעת error מתאימה ומאפס את החלון. כמו כן, מאתחל את החיבור בין הלקוח לשרת.
registerButton_Click	טיפול בכפתור ה-Register. שולח לשרת את פרטי ההתחברות ומתחיל את החיבור.
OnIm_RegisterOK	טיפול באירוע ה-RegisterOK. סוגר את חלון ה-Log In/Register ופותח את חלון ה-Contacts.
OnIm_RegisterFailed	טיפול באירוע ה-RegisterFailed. מעלה הודעת error מתאימה ומאפס את החלון. כמו כן, מאתחל את החיבור בין הלקוח לשרת.

המחלקה CONTACTS

מחלקה זו מתארת את הטופס Contacts ואת התנהגותו.

משתנים

im	החיבור בין הלקוח לשרת
singleChats	מילון המאחסן את השיחות הפתוחות של המשתמש

פעולות

chatButton_Click	טיפול באירוע הלחיצה על הכפתור Chat. בודק האם אפשר לפתוח שיחה חדשה עם שם המשתמש המסופק ומעלה הודעת error או פותח שיחה חדשה, בהתאמה.
logout_Click	מנתק את החיבור בין הלקוח לשרת וסוגר את התכנה.

המחלקה SINGLEDIALOGUE

מחלקה זו מתארת את הטופס SingleDialogue ואת התנהגותו.

משתנים

im	החיבור בין הלקוח לשרת
sendTo	הנמען בשיחה הנוכחית
contacts	טופס ה-Contacts שפתח את השיחה הנוכחית
lastAvail	מסמן האם הנמען היה זמין בפעם האחרונה שהנושא נבדק

פעולות

sendButton_Click	מטפל באירוע הלחיצה של הכפתור Send. שולח את ההודעה שהוקלדה.
SD_Load	פעולה הפועלת בטעינת הטופס. מאתחלת את ה-Timer ורושמת את ה-EventHandlers לאירועים המתאימים.
OnIm_UserAvail	מטפל באירוע של בדיקת זמינות של משתמש. קובע האם יהיה אפשר להקליד הודעה (כאשר הנמען לא זמין אי אפשר להקליד הודעה) ומודיע כאשר מצב הזמינות של הנמען משתנה.
OnIm_MsgRec	מטפל באירוע קבלת הודעה. מראה את ההודעה שהתקבלה על המסך.
Timer_Tick	פעולה החוזרת על פרק זמן קבוע של 0.1 שניות

SD_FormClosing	פעולה הפועלת עם סגירת הטופס. מנתקת את הפעולות שנרשמו לאירועים בטעינת הטופס ומוחקת את הטופס מרשימת הטפסים ב-Contacts.
endChat_Click	סוגר את הטופס.

המחלקה IM_CLIENT

מחלקה זו מתארת כיצד הלקוח מתקשר עם השרת.

משתנים

Conn	האם הלקוח מחובר/מתחבר
Logged	האם המשתמש זמין/מחובר
User	שם משתמש
Pass	סיסמא
Reg	מצב הרשמה
Client	נתוני לקוח ה-TCP
Ssl	חיבור ה-SSL המאובטח
Br	האובייקט הכותב מידע לחיבור ה-SSL
Bw	האובייקט הקורא מידע מחיבור ה-SSL
IM_Hello	ערך קבוע המשמש כ-packet להודעת 'שלום'
IM_OK	ערך קבוע המשמש כ-packet לווידוי פעולה תקינה של תהליך ההתחברות/הרשמות של הלקוח
IM_Login	ערך קבוע המשמש כ-packet לסימון כי הפעולה הבאה היא התחברות משתמש קיים
IM_Register	ערך קבוע המשמש כ-packet לסימון כי הפעולה הבאה היא הירשמות משתמש חדש
IM_TooUsername	ערך קבוע המשמש כ-packet להודעת error כי שם המשתמש ארוך מדי
IM_TooPassword	ערך קבוע המשמש כ-packet להודעת error כי הסיסמא ארוכה מדי
IM_Exists	ערך קבוע המשמש כ-packet לסימון כי המשתמש כבר קיים במערכת
IM_NoExists	ערך קבוע המשמש כ-packet לסימון כי המשתמש אינו קיים במערכת

IM_WrongPass	ערך קבוע המשמש כ-packet להודעת error כי הסיסמא שגויה
IM_IsAvailable	ערך קבוע המשמש כ-packet לסימון כי המשתמש זמין
IM_Send	ערך קבוע המשמש כ-packet לסימון כי הפעולה הבאה היא שליחת הודעה
IM_Received	ערך קבוע המשמש כ-packet לסימון כי ההודעה התקבלה

פעולות

connect	מתחיל את חיבור הלקוח אל השרת על thread חדש.
Disconnect	הלקוח מתנתק מהשרת.
IsAvailable	הלקוח שואל את השרת האם המשתמש זמין?
SendMessage	הלקוח שולח לשרת הודעה שצריכה להשלח למשתמש אחר.
SetupConn	אתחול החיבור בין השרת ללקוח, מצד הלקוח ופעולת ההתחברות של המשתמש.
CloseConn	סגירת החיבור בין השרת ללקוח, מצד הלקוח ופעולת ההתנתקות של המשתמש.
Receiver	אתחול הלולאה המקבלת חבילות מהשרת והטיפול בהן
ValidateCert	אישור כי תעודת ה-SSL שהתקבלה מהשרת בלחיצת היד אכן תקינה.

המחלקות IMERROREVENTARGS, IMAVAILEVENTARGS, IMRECEIVEDEVENTARGS

מחלקות אלו מתארות את המידע המועבר עם אירועים שונים ומספקות מסגרות לגישה למידע זה.

ביבליוגרפיה

www.stackoverflow.com

www.codeproject.com

www.reddit.com/r/learnprogramming

www.msdn.microsoft.com

www.wikipedia.org

נספח ו: קוד התכנה

חלק ו: שרת

המחלקה PROGRAM

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Sockets;
using System.Net;
using System.IO;
using System.Security.Cryptography.X509Certificates;
using System.Runtime.Serialization.Formatters.Binary;

namespace IM_Server
{
    class Program
    {
        static void Main(string[] args)
        {
            Program p = new Program();
            Console.WriteLine();
            Console.WriteLine("Press enter to close program.");
            Console.ReadLine();
        }

        public IPAddress ip = IPAddress.Parse("127.0.0.1"); // Currently set
to localhost
        public int port = 2000;
        public bool running = true;
        public TcpListener server;
        public Dictionary<string, UserInfo> Users;
        public X509Certificate2 serverCert = new
X509Certificate2("serverCert.pfx", "2486");

        public Program()
        {
            Console.Title = "InstantMessenger Server";
            Console.WriteLine("----- InstantMessenger Server -----");
            Users = new Dictionary<string, UserInfo>();
            LoadUsers();
            Console.WriteLine("[{0}] Starting server...", DateTime.Now);

            server = new TcpListener(ip, port);
            server.Start();
            Console.WriteLine("[{0}] Server is running properly!",
DateTime.Now);

            Listen();
        }

        void Listen()
        {
            while (running)
```

```
            {
                TcpClient tcpClient = server.AcceptTcpClient();
                Client client = new Client(this, tcpClient);
            }
        }

        string usersFileName = Environment.CurrentDirectory + "\\users.dat";
        public void SaveUsers() // Save users data to file
        {
            try
            {
                Console.WriteLine("[{0}] Saving users...", DateTime.Now);
                BinaryFormatter bf = new BinaryFormatter();
                FileStream file = new FileStream(usersFileName,
FileMode.Create, FileAccess.Write);
                bf.Serialize(file, Users.Values.ToArray()); // Conevert Users
to array and serialize to file
                file.Close();
                Console.WriteLine("[{0}] Users saved!", DateTime.Now);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
            }
        }
        public void LoadUsers() // Load user data
        {
            try
            {
                Console.WriteLine("[{0}] Loading users...", DateTime.Now);
                BinaryFormatter bf = new BinaryFormatter();
                FileStream file = new FileStream(usersFileName,
FileMode.Open, FileAccess.Read);
                UserInfo[] infos = (UserInfo[])bf.Deserialize(file); //
Deserialize info from file into UserInfo array
                file.Close();

                Users = infos.ToDictionary((UserInfo u) => u.Username,
(UserInfo u) => u); // Converts the "infos" array to a dictionary using
<UserInfo>u's Username property as key and "u" as value
            }
            catch { }
        }

    }
}
```

```
using System;
using System.Text;
using System.Threading;
using System.Net.Sockets;
using System.Net.Security;
using System.IO;
using System.Security.Authentication;


namespace IM_Server
{
    class Client
    {
        public Client(Program p, TcpClient c)
        {
            prog = p;
            client = c;
            (new Thread(new ThreadStart(SetupConn))).Start(); // Start new
connection with TcpClient c on a new thread
        }

        Program prog;
        public TcpClient client;
        public NetworkStream netStream;  // Raw-data stream of connection.
        public SslStream ssl;            // Encrypts connection using SSL.
        public BinaryReader br;          // Read simple data
        public BinaryWriter bw;          // Write simple data
        UserInfo userInfo;

        void SetupConn()  // Set up connection
        {
            Console.WriteLine("[{0}] New connection!", DateTime.Now);
            netStream = client.GetStream();
            ssl = new SslStream(netStream, false);
            ssl.AuthenticateAsServer(prog.serverCert, false,
SslProtocols.Tls, true);
            Console.WriteLine("[{0}] Connection authenticated!",
DateTime.Now);
            br = new BinaryReader(ssl, Encoding.UTF8);
            bw = new BinaryWriter(ssl, Encoding.UTF8);

            bw.Write(IM_Hello); // Send confirmation to client
            bw.Flush();

            int hello = br.ReadInt32();
            if (hello == IM_Hello) // Confirmation from client valid
            {
                byte logMode = br.ReadByte();
                string user = br.ReadString();
                string pass = br.ReadString();
                if (user.Length < 10 || user.Length < 3) // User isnt too
long
                {
                    if (pass.Length < 20 || pass.Length < 3) // Password isnt
too long
```

```
                    {
                        if (logMode == IM_Register) // User registers
                        {
                            if (!prog.Users.ContainsKey(user)) // User
doesn't exist in user collection
                            {
                                // Add user to user collection and associates
it with current (this) connection
                                userInfo = new UserInfo(user, pass, this);
                                prog.Users.Add(user, userInfo);
                                bw.Write(IM_OK);
                                bw.Flush();
                                Console.WriteLine("[{0}] ({1}) Registered new
user", DateTime.Now, user);
                                prog.SaveUsers(); // Update Users collection
                                Receiver();
                            }
                            else
                                bw.Write(IM_Exists);
                        }
                        else if (logMode == IM_Login) // User logs in
                        {
                            if (prog.Users.TryGetValue(user, out userInfo))
// User exists in user collection
                            {
                                if (pass == userInfo.Password) // Correct
password
                                {
                                    if (userInfo.LoggedIn) // Disconnect
whoever is logged in on this user
                                        userInfo.Connection.CloseConn();

                                    // Associate connection to the logged-in
user
                                    userInfo.Connection = this;
                                    bw.Write(IM_OK);
                                    bw.Flush();
                                    Receiver();
                                }
                                else
                                    bw.Write(IM_WrongPass);
                            }
                            else
                                bw.Write(IM_NoExists);
                        }
                    }
                    else
                        bw.Write(IM_TooPassword);
                }
                else
                    bw.Write(IM_TooUsername);

            }

            CloseConn();
        }
        void CloseConn() // Close connection
```

```
        {
            try
            {
                userInfo.LoggedIn = false;
                br.Close();
                bw.Close();
                ssl.Close();
                client.Close();
                Console.WriteLine("[{0}] End of connection!", DateTime.Now);
            }
            catch { }
        }

        void Receiver() // Receive all incoming packets loop
        {
            Console.WriteLine("[{0}] ({1}) User logged in", DateTime.Now,
userInfo.Username);
            userInfo.LoggedIn = true;
            try
            {
                while (client.Connected) // While connected
                {
                    byte type = br.ReadByte(); // Get incoming packet type

                    if (type==IM_IsAvailable) // Check if 'who' is available
                    {
                        string who = br.ReadString();
                        bw.Write(IM_IsAvailable);
                        bw.Write(who);

                        UserInfo info;
                        if (prog.Users.TryGetValue(who, out info)) // If
'who' is registered, check if logged in
                        {
                            if (info.LoggedIn)
                                bw.Write(true); // Available
                            else
                                bw.Write(false); // Unavailable
                        }
                        else
                            bw.Write(false); // 'who' Is not registered -
unavailable
                        bw.Flush();
                    }
                    else if (type==IM_Send) // A message is sent to another
user
                    {
                        string to = br.ReadString();
                        string msg = br.ReadString();

                        UserInfo recipient;
                        if (prog.Users.TryGetValue(to, out recipient)) //
Does 'recipient' exist?
                        {
                            if (recipient.LoggedIn) // Is 'recipient' logged
in?
                            {
```

```
                                recipient.Connection.bw.Write(IM_Received);
// Confirm message is received

recipient.Connection.bw.Write(userInfo.Username); // From who the message was
sent
                                recipient.Connection.bw.Write(msg); // The
message
                                recipient.Connection.bw.Flush();
                                Console.WriteLine("[{0}] ({1} -> {2}) Message
sent!", DateTime.Now, userInfo.Username, recipient.Username);

                            }
                        }
                    }
                }
            }
            catch (IOException){ } // AKA no input = closed connection

            userInfo.LoggedIn = false;
            Console.WriteLine("[{0}] ({1}) User logged out", DateTime.Now,
userInfo.Username);
        }

        public const int IM_Hello = 2012;      // Hello
        public const byte IM_OK = 0;           // OK
        public const byte IM_Login = 1;        // Login
        public const byte IM_Register = 2;     // Register
        public const byte IM_TooUsername = 3;  // Too long username
        public const byte IM_TooPassword = 4;  // Too long password
        public const byte IM_Exists = 5;       // Already exists
        public const byte IM_NoExists = 6;     // Doesn't exists
        public const byte IM_WrongPass = 7;    // Wrong password
        public const byte IM_IsAvailable = 8;  // Is user available?
        public const byte IM_Send = 9;        // Send message
        public const byte IM_Received = 10;    // Message received
    }
}
```

המחלקה USERINFO

```
using System;

namespace IM_Server
{
    [Serializable]
    class UserInfo
    {
        public string Username;
        public string Password;
        [NonSerialized] public bool LoggedIn;       // Is logged in and
connected?
        [NonSerialized] public Client Connection;   // Connection info

        public UserInfo(string user, string pass)
        {
            this.Username = user;
            this.Password = pass;
            this.LoggedIn = false;
        }

        public UserInfo(string user, string pass, Client conn)
        {
            this.Username = user;
            this.Password = pass;
            this.LoggedIn = true;
            this.Connection = conn;
        }
    }
}
```

המחלקה LOGREGFORM

```
using System;
using System.Windows.Forms;

namespace IM_Client
{
    public partial class LogRegForm : Form
    {
        IM_Client im;

        public LogRegForm()
        {
            InitializeComponent();

            this.im = new IM_Client();

            im.LoginOK += OnIm_LoginOK;
            im.RegisterOK += OnIm_RegisterOK;
            im.LoginFailed += OnIm_LoginFailed;
            im.RegisterFailed += OnIm_RegisterFailed;
        }

        private string usernameInput;
        private string passwordInput;

        private void loginButton_Click(object sender, EventArgs e)
        {
            if (usernameTextBox.Text != null && passwordTextBox.Text != null)
            {
                this.usernameInput = usernameTextBox.Text;
                this.passwordInput = passwordTextBox.Text;

                im.Login(usernameInput, passwordInput);
                status.Text = "Logging In...";
            }
        }
        private void OnIm_LoginOK(object sender, EventArgs e)
        {
            this.BeginInvoke(new MethodInvoker(delegate
            {
                status.Text = "Logged In!";
                registerButton.Enabled = false;
                loginButton.Enabled = false;
                usernameTextBox.Enabled = false;
                passwordTextBox.Enabled = false;

                Contacts c = new Contacts(im);
                c.Show();
                this.Hide();
            }));
        }
        private void OnIm_LoginFailed(object sender, IMErrorEventArgs e)
        {
```

```
            this.BeginInvoke(new MethodInvoker(delegate
            {

                MessageBox.Show(String.Format("Login Failed! Error Code {0}",
e.Error.ToString()));
                usernameTextBox.Text = "";
                passwordTextBox.Text = "";
                this.status.Text = "Log In or Register";
                im.Disconnect();
                im = new IM_Client();
                im.LoginOK += OnIm_LoginOK;
                im.RegisterOK += OnIm_RegisterOK;
                im.LoginFailed += OnIm_LoginFailed;
                im.RegisterFailed += OnIm_RegisterFailed;
            }));
        }

        private void registerButton_Click(object sender, EventArgs e)
        {
            if (usernameTextBox.Text != null && passwordTextBox.Text != null)
            {
                this.usernameInput = usernameTextBox.Text;
                this.passwordInput = passwordTextBox.Text;

                im.Register(usernameInput, passwordInput);
                status.Text = "Registering...";
            }

        }
        private void OnIm_RegisterOK(object sender, EventArgs e)
        {
            this.BeginInvoke(new MethodInvoker(delegate
            {
                status.Text = "Registered and Logged In!";
                registerButton.Enabled = false;
                loginButton.Enabled = false;
                usernameTextBox.Enabled = false;
                passwordTextBox.Enabled = false;

                Contacts c = new Contacts(im);
                c.Show();
                this.Hide();
            }));
        }
        private void OnIm_RegisterFailed(object sender, IMErrorEventArgs e)
        {
            this.BeginInvoke(new MethodInvoker(delegate
            {
                MessageBox.Show("Register Failed! Error Code {0}",
e.Error.ToString());
                usernameTextBox.Text = "";
                passwordTextBox.Text = "";
                this.status.Text = "Log In or Register";
                im = new IM_Client();
            }));
        }
    }}
```

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace IM_Client
{
    public partial class Contacts : Form
    {
        IM_Client im;
        public Dictionary<string, SingleDialogue> singleChats = new
Dictionary<string, SingleDialogue>();

        public Contacts(IM_Client IM)
        {
            InitializeComponent();
            this.im = IM;
            this.Text = im.Username;
            this.ControlBox = false;
        }

        private void chatButton_Click(object sender, EventArgs e)
        {
            if (chatTextBox.Text == im.Username)
                MessageBox.Show("You cannnot talk to yourself.");
            else if (!singleChats.ContainsKey(chatTextBox.Text))
            {
                SingleDialogue sd = new SingleDialogue(im, chatTextBox.Text,
this);
                singleChats.Add(chatTextBox.Text, sd);
                chatTextBox.Text = "";
                sd.Show();
            }
            else
                MessageBox.Show(String.Format("Chat with {0} is already
open.", chatTextBox.Text));
        }

        private void logOut_Click(object sender, EventArgs e)
        {
            im.Disconnect();
            Application.Exit();
        }
    }
}
```

```
using System;
using System.Windows.Forms;

namespace IM_Client
{
    public partial class SingleDialogue : Form
    {
        IM_Client im;
        public string sendTo;
        Contacts contacts;

        public SingleDialogue(IM_Client IM, string sendTO, Contacts c)
        {
            InitializeComponent();

            this.contacts = c;
            this.im = IM;
            this.sendTo = sendTO;
        }

        private void sendButton_Click(object sender, EventArgs e)
        {
            im.SendMessage(sendTo, this.sendTextBox.Text);
            this.talkTextBox.Text += String.Format("{0}: {1}\r\n",
im.Username, this.sendTextBox.Text);
            this.sendTextBox.Text = "";
        }

        private void SingleDialogue_Load(object sender, EventArgs e)
        {
            this.Text = sendTo;
            availHandler = new
EventHandler<IMAvailEventArgs>(OnIm_UserAvailable);
            receivedHandler = new
EventHandler<IMReceivedEventArgs>(OnIm_MessageReceived);
            this.talkTextBox.ReadOnly = true;
            this.sendTextBox.Enabled = false;
            im.UserAvailable += availHandler;
            im.MessageReceived += receivedHandler;

            this.timer.Start();
            im.IsAvailable(sendTo);
        }

        EventHandler<IMAvailEventArgs> availHandler;
        EventHandler<IMReceivedEventArgs> receivedHandler;

        private bool lastAvail = false;
        private void OnIm_UserAvailable(object sender, IMAvailEventArgs e)
        {
            if (e.Username == sendTo)
                if (e.IsAvailable != lastAvail)
                    this.BeginInvoke(new MethodInvoker(delegate
                    {
                        lastAvail = e.IsAvailable;
```

```
                        this.sendTextBox.Enabled = true;
                        string avail = (lastAvail ? "Available" :
"Unavailable");
                        this.sendTextBox.Enabled = lastAvail;
                        this.talkTextBox.Text += string.Format("[{0} is
{1}]\r\n", sendTo, avail);
                    }));
        }

        private void OnIm_MessageReceived(object sender, IMReceivedEventArgs
e)
        {
            this.BeginInvoke(new MethodInvoker(delegate
            {
                if (e.From == sendTo)
                    this.talkTextBox.Text += String.Format("{0}: {1}\r\n",
e.From, e.Message);
            }));
        }

        private void timer_Tick(object sender, EventArgs e)
        {
            im.IsAvailable(sendTo);
        }

        private void SingleDialogue_FormClosing(object sender,
FormClosingEventArgs e)
        {
            im.UserAvailable -= availHandler;
            im.MessageReceived -= receivedHandler;
            contacts.singleChats.Remove(sendTo);
        }

        private void endChatToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            this.Close();
        }
    }
}
```

```
using System;
using System.Text;
using System.Threading;
using System.Net.Sockets;
using System.Net.Security;
using System.IO;
using System.Security.Cryptography.X509Certificates;
using System.Windows.Forms;

namespace IM_Client
{
    public class IM_Client
    {
        public Thread TcpThread;    // Connection Thread
        public bool Conn = false;   // Is connected/connecting?
        private bool Logged = false; // Is logged in?
        private string User;         // Username
        private string Pass;         // Password
        public bool Reg;            // Register Mode

        public string Server { get { return "localhost"; } }
        public int Port { get { return 2000; } }

        public bool IsLoggedIn { get { return Logged; } }
        public string Username { get { return User; } }
        public string Password { get { return Pass; } }

        // Start connection thread, login, register
        void connect(string username, string password, bool register)
        {
            if (!Conn)
            {
                Conn = true;
                User = username;
                Pass = password;
                Reg = register;
                TcpThread = new Thread(new ThreadStart(SetupConn)); // New
thread for new connection
                TcpThread.Start();
            }
        }

        public void Login(string username, string password)
        {
            connect(username, password, false);
        }
        public void Register(string username, string password)
        {
            connect(username, password, true);
        }
        public void Disconnect()
        {
            if (Conn)
                CloseConn();
        }
```

```
        public void IsAvailable(string user)
        {
            if (Conn)
            {
                bw.Write(IM_IsAvailable);
                bw.Write(user);
                bw.Flush();
            }
        } // At launch of chat instance (OnTalkButton_Click
        public void SendMessage(string to, string msg)
        {
            bw.Write(IM_Send);
            bw.Write(to);
            bw.Write(msg);
            bw.Flush();
        }

        public TcpClient client;
        public NetworkStream netStream;  // Raw-data stream of connection.
        public SslStream ssl;            // SSL connection
        public BinaryReader br;          // Read simple data
        public BinaryWriter bw;          // Write simple data

        void SetupConn()  // Setup connection
        {
            client = new TcpClient(Server, Port);
            netStream = client.GetStream();
            ssl = new SslStream(netStream, false, new
RemoteCertificateValidationCallback(ValidateCert));
            ssl.AuthenticateAsClient("Itamar Reif");
            br = new BinaryReader(ssl, Encoding.UTF8);
            bw = new BinaryWriter(ssl, Encoding.UTF8);

            int hello = br.ReadInt32();
            if (hello == IM_Hello) // Receive confirmation from server
            {
                bw.Write(IM_Hello);
                bw.Flush(); // Send back confirmation to server

                bw.Write(Reg ? IM_Register : IM_Login); // Register or log in
                bw.Write(Username);
                bw.Write(Password);
                bw.Flush();

                byte ans = br.ReadByte();
                if (ans == IM_OK) // Login/Register OK
                {
                    if (Reg)
                    {
                        OnRegisterOK();
                        Receiver(); // Packet receiving loop
                    }
                    else
                    {
                        OnLoginOK();
                        Receiver(); // Packet receiving loop
                    }
```

```
                }
                else // Login/Register FAIL
                {
                    IMErrorEventArgs error = new
IMErrorEventArgs((IMError)ans);
                    if (Reg)
                        OnRegisterFailed(error);
                    else
                        OnLoginFailed(error);
                }
            }

            CloseConn();
        }

        void CloseConn() // Close connection.
        {
            br.Close();
            bw.Close();
            netStream.Close();
            ssl.Close();
            OnDisconnected();
            client.Close();
        }

        void Receiver() // Packet receiving loop
        {
            Logged = true;

            try
            {
                while (client.Connected)
                {
                    byte type = br.ReadByte();
                    if (type == IM_IsAvailable)
                    {
                        string user = br.ReadString();
                        bool isAvail = br.ReadBoolean();
                        OnUserAvail(new IMAvailEventArgs(user, isAvail));
                    }
                    else if (type == IM_Received)
                    {
                        string from = br.ReadString();
                        string msg = br.ReadString();
                        OnMessageRec(new IMReceivedEventArgs(from, msg));
                    }
                }
            }
            catch (IOException) { } // AKA no input = closed connection

            Logged = false;
        }

        // Events
        public event EventHandler LoginOK;
        public event EventHandler RegisterOK;
        public event EventHandler<IMErrorEventArgs> LoginFailed;
```

```
        public event EventHandler<IMErrorEventArgs> RegisterFailed;
        public event EventHandler Disconnected;
        public event EventHandler<IMAvailEventArgs> UserAvailable;
        public event EventHandler<IMReceivedEventArgs> MessageReceived;

        protected virtual void OnLoginOK()
        {
            if (LoginOK != null)
                LoginOK(this, EventArgs.Empty);
        }
        protected virtual void OnRegisterOK()
        {
            if (RegisterOK != null)
                RegisterOK(this, EventArgs.Empty);
        }
        protected virtual void OnLoginFailed(IMErrorEventArgs e)
        {
            if (LoginFailed != null)
                LoginFailed(this, e);
        }
        protected virtual void OnRegisterFailed(IMErrorEventArgs e)
        {
            if (RegisterFailed != null)
                RegisterFailed(this, e);
        }
        protected virtual void OnDisconnected()
        {
            if (Disconnected != null)
                Disconnected(this, EventArgs.Empty);
        }
        protected virtual void OnUserAvail(IMAvailEventArgs e)
        {
            if (UserAvailable != null)
                UserAvailable(this, e);
        }
        protected virtual void OnMessageRec(IMReceivedEventArgs e)
        {
            if (MessageReceived != null)
                MessageReceived(this, e);
        }

        public const int IM_Hello = 2012;       // Hello
        public const byte IM_OK = 0;            // OK
        public const byte IM_Login = 1;         // Login
        public const byte IM_Register = 2;      // Register
        public const byte IM_TooUsername = 3;   // Too long username
        public const byte IM_TooPassword = 4;   // Too long password
        public const byte IM_Exists = 5;        // Already exists
        public const byte IM_NoExists = 6;      // Doesn't exists
        public const byte IM_WrongPass = 7;     // Wrong password
        public const byte IM_IsAvailable = 8;   // Is user available?
        public const byte IM_Send = 9;         // Send message
        public const byte IM_Received = 10;     // Message received

        public static bool ValidateCert(object sender, X509Certificate
certificate, X509Chain chain, SslPolicyErrors sslPolicyErrors)
        {
```

```
            if (sslPolicyErrors == SslPolicyErrors.None ||
sslPolicyErrors==SslPolicyErrors.RemoteCertificateChainErrors) // Assume cert
is not trusted by local machine (personal)
                return true;
            else
            {
                MessageBox.Show(string.Format("SSL Error:{0}",
sslPolicyErrors.ToString()));
                return false;
            }
            // return true;
        }
    }
}
```

המחלקות IMERROREVENTARGS, IMAVAILEVENTARGS, IMRECEIVEDEVENTARGS

```
namespace IM_Client
{
    using System;

    public enum IMError : byte
    {
        TooUserName = IM_Client.IM_TooUsername,
        TooPassword = IM_Client.IM_TooPassword,
        Exists = IM_Client.IM_Exists,
        NoExists = IM_Client.IM_NoExists,
        WrongPassword = IM_Client.IM_WrongPass
    }

    public class IMErrorEventArgs : EventArgs
    {
        private IMError err;

        public IMErrorEventArgs(IMError error)
        {
            this.err = error;
        }

        public IMError Error { get { return err; } }
    }

    public class IMAvailEventArgs : EventArgs
    {
        private string user;
        private bool avail;

        public IMAvailEventArgs(string User, bool Avail)
        {
            this.user = User;
            this.avail = Avail;
        }

        public string Username { get { return user; } }
        public bool IsAvailable { get { return avail; } }
    }

    public class IMReceivedEventArgs : EventArgs
    {
        private string user;
        private string msg;

        public IMReceivedEventArgs(string User, string Msg)
        {
            this.user = User;
            this.msg = Msg;
        }

        public string From { get { return user; } }
        public string Message { get { return msg; } }
    }
}
```