



פרויקט במחשבים : שיתוף מסכים



בית ספר : מקיף עירוני ח' ראשון לציון

שם העבודה : Mirror Screen

שם התלמיד : שלו זלינגר

ת.ז. התלמיד : 325212652

שם המנחה : קובי שוצמן

תאריך ההגשה :

תוכן עניינים

4	מבוא
6	מבנה/ ארכיטקטורה של הפרויקט
6	הצגת הפיתרון המוצע והסיבות לבחירתו
7	האלגוריתמים
7	○ פורמט הממשק:
7	○ פורמט הלקוח:
8	○ פורמט השרת:
9	ארכיטקטורה של הפרויקט בפורמט של Top-Down level Design:
9	○ פורמט הממשק:
10	○ פורמט הלקוח:
11	○ פורמט השרת:
12	יחידות מחלקות הפרויקט (Components)
12	פעולות הממשק:
14	פעולות הלקוח:
15	פעולות השרת:
16	הקשרים בין היחידות השונות
17	תרשים use case
18	ארכיטקטורת הרשת
20	מדריך למשתמש
20	הוראות התקנה
21	תרשים מסכים
22	תפקיד כל מסך
22	המסך הראשי:
27	מדריך למפתח
27	הקובץ ראשון- server.py:
33	הקובץ השני- client.py :
39	הקובץ השלישי- Ui.py :
48	רפלקציה
50	ביבליוגרפיה

מבוא

הספר פרויקט מכיל את המבנה של הפרויקט שלי, את התהליך שלקח לי עד שהגעתי להחלטה של איזה פרויקט אני אבצע וכיצד בחרתי לבצע את הפרויקט. הספר יכול מבנה של הפרויקט שלי בכתב תוך כדי שילוב של תמונות ואמצעים להמחשה על מנת לקבל את ההבנה המיטבית של הפרויקט שלי עבור המשתמש. הספר פרויקט אף יכול מדריך למשתמש ומדריך למפתחים שירצו להכין פרויקט זהה. לסיום הספר יכול רפלקציה אישית שכוללת גם סיכום על כל התהליך שעברתי בכתבת הפרויקט וביבליוגרפיה של מקומות אשר בהם נעזרתי.

קצת אפרט קצת על הדרך שבה הלכתי עד שהגעתי להחלטה מה יהיה הנושא שעליו העסוק בפרויקט שלי. – **מדוע בחרתי בשיתוף מסכים כפרויקט גמר** תחילה חשבתי לעשות מאגר מוזיקה אך החלטתי שלא משום שהתהליך היה נראה לי ארוך מדי ומסובך לעומת הזמן הדל שיש לי. אך פתאום, לאחר שחיפשתי נושאים לעשות עליהם נמשך לי לעיין נושא בשם שיתוף מסכים. תחילה חשבתי זה יהיה פשוט וקל אך אחר כך במהלך התהליך עצמו גיליתי שזה אף הרבה יותר מסובך ממה שחשבתי.

תחילה לפני שאפרט על הקשיים שעברתי במהלך הפרויקט אני ארצה לפרט על המצב הקיים בשוק ועל תוכנות זהות.

כיום קיימות תוכנות רבות העוסקות בשיתוף מסכים בדרכים כאלה ואחרות בין אם זה תוכנות שהיו קיימות עוד לפני תקופת הקורונה כגון team viewer או any desk ואף תוכנות שהתפרסמו מאוד בעקבות תקופת הקורונה הכוללות גם אופציה לשיחה עם מצלמה בנוסף לשיתוף מסך כגון תוכנות כמו zoom או Google meet ואף התוכנה שבה אנו משתמשים בשיעורי המחשבים שלנו עם המורה cisco WebEx meetings.

הפרויקט שלי שונה מהתוכנות שיצאו לאחר תקופת הקורונה אלא הוא יותר זהה לתוכנות כמו team viewer שמטרתן לראות את המסך הנוכחי של המשתף ובכך ניתן לעזור לו בבעיות שונות במחשב.

הבעיה המרכזית שקיימת עם תוכנות אלו היא בדרך כלל העובדה שצריך לשלם על מנת להשתמש בהן או העובדה שניתן להסתכל על המסך של המשתף אך ורק אדם אחד, הפרויקט שלי מציע פתרון לבעיות אלו.

לאחר חודשיים מאז **בחירת נושא הפרויקט שלי** כאשר הקורונה הייתה בשיאה וכולם היו בבידוד החלתי לעבוד על הפרויקט שלי זאת במטרה לעזור לאנשים שצריכים עזרה במחשב בבעיות טכנולוגיות כאלו או אחרות או אפילו ללמוד באמצעות הפרויקט שלי. גם כיום לאחר סיום תקופת הקורונה או לפחות בתקופת הירגעות המצב אני יכול להשתמש בפרויקט על מנת בעצם לקיים בשיעורי מחשבים שיתוף שבו המורה מראה את המסך שלו והתלמידים יכולים לראות ולהקשיב לו תוך כדי.

אחת מהבעיות המרכזיות שאיתה נאלצתי להתמודד במהלך הפרויקט היא כיצד להעביר את התמונה הנוכחית על המסך מבלי לאבד חלק מהמידע. זאת משום שאת שיתוף המסכים אני מבצע

בפרוטוקול udp המאפשר העברה מהירה של המידע אך גורמת מדיי פעם לאיבוד המידע. לכן דבר זה גרם לכך שהגודל של התמונה היה נשלח ביחד עם החלקים הראשונים של התמונה.

על מנת להתמודד עם הבעיה שהייתה לי ראשית הייתי צריך להבין יותר את הנושא מה שאילץ אותי לקרוא במקומות שונים וללמוד כיצד המערכת פועלת ועל ידי כך אני אדע איך לבצע את השיתוף בדרך הטובה ביותר ללא תקיעות. לאחר מכאן ניסיתי לסדר את הבעיות שהיו לי גם באמצעות ידע קודם ואף באמצעות עזרה מחברים. לבסוף סיימתי את חלק התוכן ועבדתי על שלב הממשק הגרפי. לאחר מכאן שילבתי בין השתיים והוספתי פיצ'רים לפרויקט כגון צ'אט ושיחות אחרונות.

בנוסף לבעיה התכנית שהייתה היו לי גם כמה בעיות נפשיות. כאשר לא הייתי בטוח מה אני הולך לעשות בפרויקט וכיצד אני הולך לעשות אותו חששתי להתחיל את הפרויקט מה שמנע ממני להתקדם. אך ברגע שהתחלתי לתכנת ולכתוב את הפרויקט שלי קיבלתי מוטיבציה ורצון להמשיך. הידיעה שהזמן לא לטובתי כלומר זמן ההגשה הולך ומתקרב גם היא עזרה פעמים רבות מה שעודד אותי להמשיך ולהתקדם בפרויקט. גם העובדה שחברים שלי התקדמו בפרויקט ואני הייתי מאחור עודדה אותי פעמים רבות להמשיך עם הפרויקט.

מבנה / ארכיטקטורה של הפרויקט

הצגת הפיתרון המוצע והסיבות לבחירתו

הבעיה המרכזית שהייתה לי במהלך קידוד הקוד של הפרויקט הייתה שליחת התמונה הנוכחית של השרת ללקוח.

זאת משום שעל מנת לשלוח תמונה יש צורך לשלוח אותה בחלקים משום שתמונה זהו מסמך גדול ואחת מהבעיות בשליחת התמונה בחלקים היא שלא ידוע גודל התמונה. על מנת להתגבר על מכשול זה הייתי שולח את גודל התמונה בתחילת כל לולאה מה שהיה יוצר לי בעיה נוספת שגרמה לקח שהחלק הראשון של התמונה היה נשלח ביחד עם הגודל שלה. לכן מה שעשיתי היה להשתמש בפקודה split על המחרוזת מה שפיצל לי את החלק המספרי כלומר הגודל לחלק של התמונה. לאחר מכאן שלחתי את חלק התמונה למשתנה שבאמצעותו יצרתי את התמונה על המסך ולסיום נשלח כל המידע.

תחילה קצת הסתבכתי למצוא את הפתרון ואת הדרך היעילה ביותר לעשות זאת מבלי להיכנס לסיבוכים אחרים וקוד ארוך אך לאחר שחיפשתי פתרונות באינטרנט ולמדתי קצת יותר על מחרוזות ועל הדרך בה נשלח מידע ברשת הצלחתי להבין מה עליי לעשות.

לאחר שסיפרתי קצת על הבעיות שנתקלתי בהן במהלך הפרויקט אני ארצה בעצם לענות על השאלה בשביל מה צריך את הפרויקט שלי ולמה הוא שימושי?

אז הפרויקט שלי בא לענות על בעיה פשוטה מאוד והיא העברת מידע. כיום בחיים שלנו יש צורך לשתף מידע בין אנשים בדרכים שונות. דרך מרכזית שצצה בזמן האחרון היא דרך שיחות ברשתות החברתיות באמצעים שונים כגון זום או גוגל מיט זאת בעקבות וירוס הקורונה.

הפרויקט שלי אומנם שונה מאפליקציות אלו אך מטרתו זהה. באמצעות הפרויקט שלי הלקוח יכול לראות את המסך הנוכחי של הסרבר מה שיכול להועיל בין מורה ותלמיד כאמצעי להעברת מידע או לימוד אומנם ללא קול אך ניתן ככה לעזור גם בקשיים טכניים שאינם דורשים תקשורת בין ה2 אך גם לזה יש פתרון בפרויקט שלי וזה באמצעות חלון צ'אט שאתו ניתן לשוחח בין המורה לבין התלמיד.

הסיבה שבחרתי בפרויקט זה היא משום שכאשר החלה תקופת הקורונה הגעתי למסקנה שאנשים יצטרכו אמצעי תקשורת שאינו כולל תשלום ושארף קל למשתמש. וזאת הייתה מטרתי בפרויקט שלי. אני ניסיתי ליצור את הפרויקט שלי בצורה שהוא יהיה פשוט ביותר אפילו לאנשים שפחות מבינים בתחום עם ממשק שהוא יחסית פשוט להפעלה. אפילו הוספתי אמצעים כגון מציאת הip שלך בפרויקט לאנשים שלא מתמחים בתחום ולא יודעים כיצד לעשות זאת.

האלגוריתמים

○ פורמט הממשק:

1. מריץ את הממשק מכאן יש בחירה של כמה אופציות:
2. במידה ו**נבחרת האופציה של Instructions**.
 - 2.1 נפתח information box שמעניק מידע כיצד להשתמש בתוכנה.
 - 2.2 סוגרים את חלון המידע וחוזרים בחזרה למסך הראשי.
3. במידה ובחרים להיכנס בתור שרת ולוחצים על הכפתור Start
 - 3.1 נפתח הסרבר והוא מתחיל להריץ את התוכנה עד שמתחבר לקוח.
4. במידה ורוצים להיכנס בתור לקוח מכניסים את הכתובת ip שאליה רוצים להיכנס
 - 4.1 לוחצים go.
 - 4.2 במידה ואם הסרבר קיים אז מתחיל שיתוף המסך, במידה ואם לא הלקוח ממתין עד שהסרבר ירוץ.
5. כעת הסביר בפירוט על כל אחד מהפורמטים השונים (לקוח וסרבר) בנפרד.

○ פורמט הלקוח:

1. הכפתור go נלחץ :
- 1.1 במידה ולא הוכנסה כתובת ip קופצת הודעת שגיאה.
- 1.2 במידה ואם כן הוכנס כתובת ip זה מפעיל את הפונקציה check_server() שבודק אם הסרבר רץ כרגע או אם הכתובת ip נכונה.
 - 1.2.1 אם הכתובת ip שגויה או שהסרבר לא רץ כרגע זה מקפיץ הודעת שגיאה.
 - 1.2.2 במידה ואם הסרבר כן רץ זה מתחיל את השיתוף מסך.
2. הלקוח מפעיל את הפונקציה start_client() .
3. נוצר מסך בגודל של 1440x720 וה-client מתחיל את הלולאה.
4. מתבצעת בדיקה האם המשתמש רוצה להפסיק
 - 4.1 אם לא הלקוח מפסיק את השיתוף וסוגר את socketn
 - 4.2 אם כן :
 - 4.2.1 הלקוח מקבל את גודל התמונה הנוכחית

4.2.2 הלקוח מכניס את הגודל הכולל למשתנה ואם במידה נשלח גם חלק התחלתי מהתמונה הוא מכניס אותו לקובץ שיוצר את התמונה במחשב הלקוח.

4.2.3 הלקוח מתחיל לולאה של קבלת התמונה עד שמגיעה כל התמונה.

4.2.4 הלקוח מנסה להראות את התמונה על מסך pygamen שנוצר.

4.2.4.1 במידה והצליח התמונה מופיעה על המסך והלקוח חוזר בחזרה

ל4 ומבצע את הלולאה בשנית.

4.2.4.2 מדלג על התמונה הנוכחית וחוזר בחזרה ל4

○ פורמט השרת:

1. הכפתור start נלחץ

2. מופעלת הפונקציה run() על ידי servern.

3. servern מתחיל לרוץ ומחכה ללקוח:

3.1 ברגע שנכנס לקוח השרבר מתחיל thread שעובר על הפונקציה handle_client()

ומגדיל את מספר הלקוחות המחוברים באחד.

3.1.1 השרבר לוקח תמונה של המסך הנוכחי שלו ושומר אותה על המחשב שלו.

3.1.2 השרבר שולח את הגודל של התמונה ללקוח.

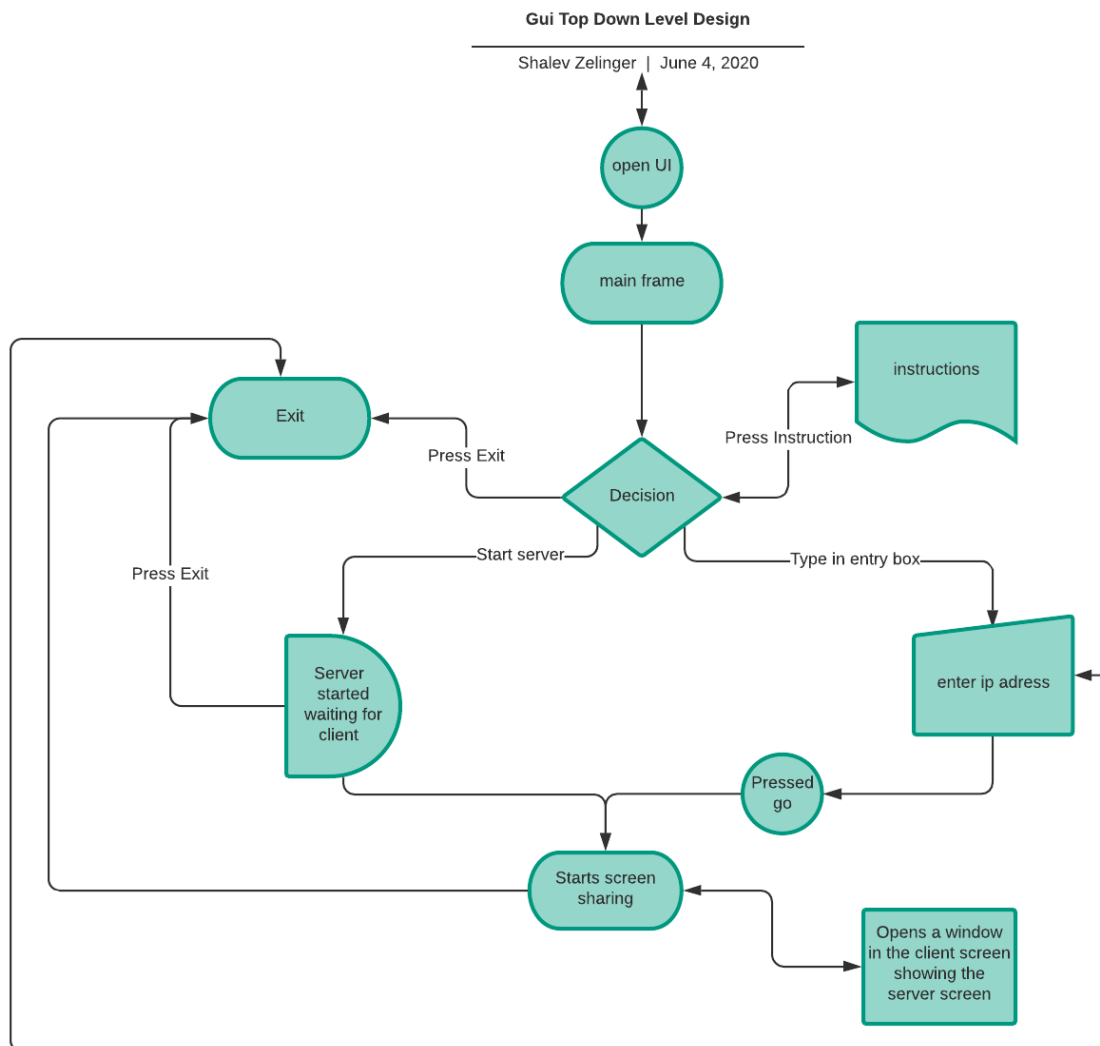
3.1.3 השרבר מתחיל לשלוח את התמונה בחלקים ללקוח עד שהכל נשלח.

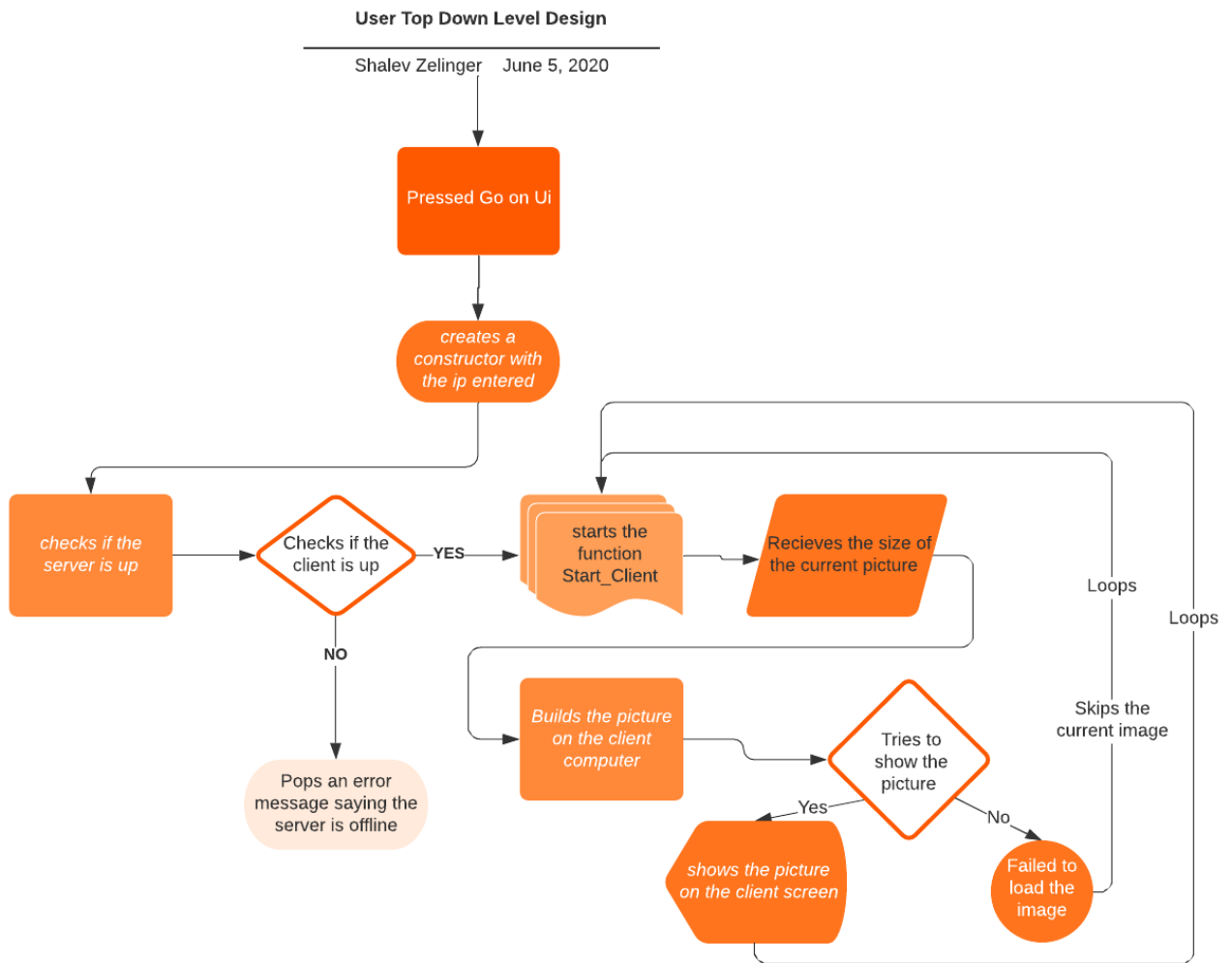
3.1.4 השרבר מסיים לשלוח את התמונה וחוזר בחזרה ל3.1.1.

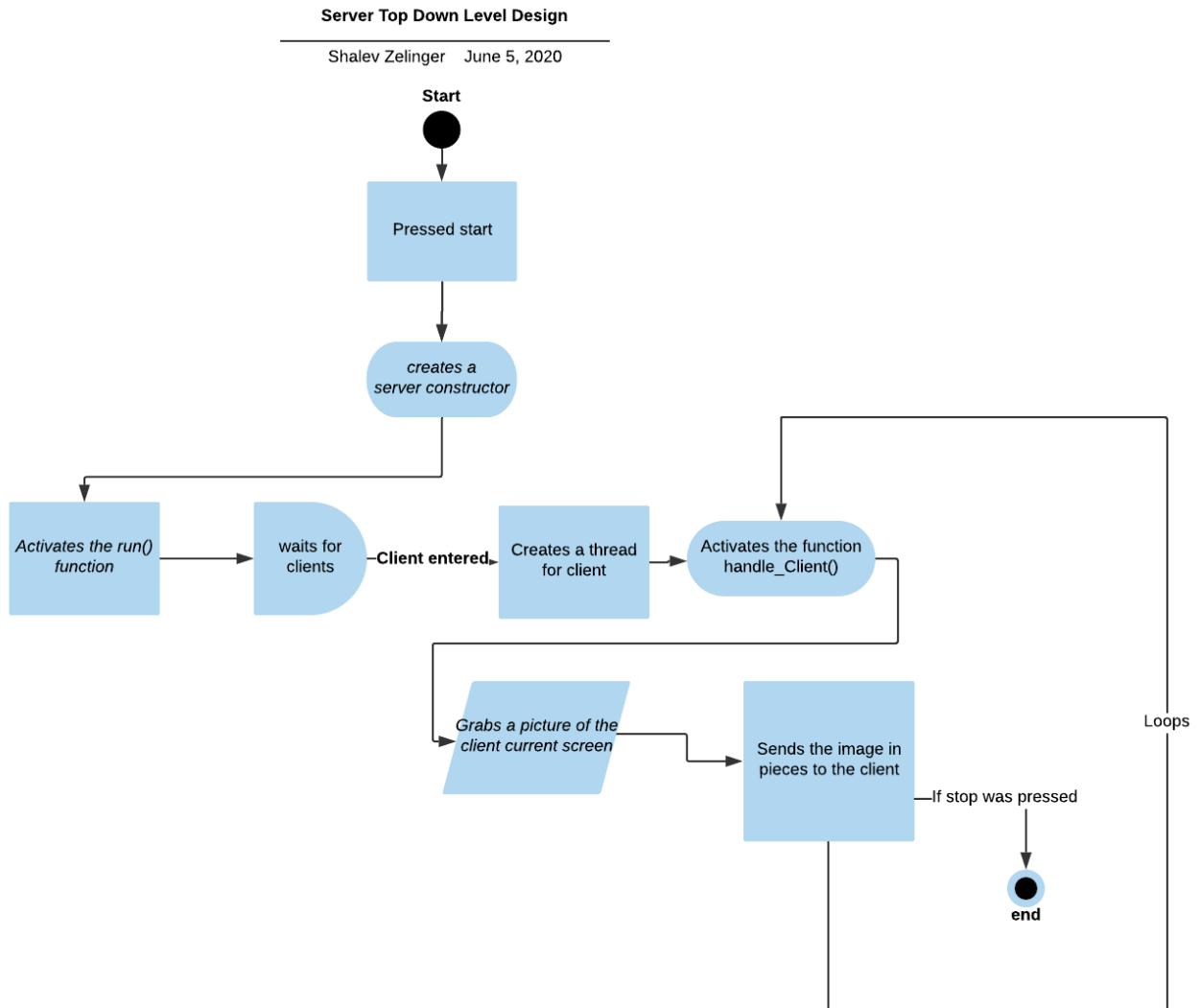
3.2 השרבר חוזר ל3 וממתין ללקוחות נוספים בעוד שהthread רץ ברקע.

אריכטקטורה של הפרויקט בפורמט של Top-Down level Design :

○ פורמט הממשק:



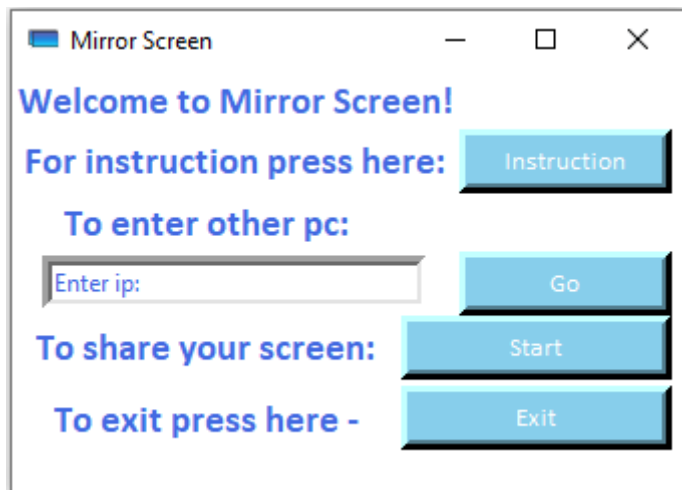




יחידות מחלקות הפרויקט (Components)

בחלק זה אני אפרט על כל הפונקציות המרכזיות שקיימות בפרויקט בכל החלקים השונים (הממשק, השרת והלקוח). אני לא אפרט על פונקציות של set/get ועוד פונקציות פשוטות כאלו או אחרות. למרות זאת לכל פונקציה קיים doestring המסביר מה היא עושה ומה היא מחזירה. הערה חשובה שחשוב לציין במהלך ההסברים אני מציין שרת ולקוח. הכוונה היא שהלקוח זה מי שרוצה לראות את המסך של החבר, והשרת זה מי שמשתף את המסך שלו. (צוין על מנת למנוע חוסר הבנות)

ועל מנת שההסברים על הכפתורים שעליהם אני מדבר במהלך ההסברים יהיו מובנים יותר אני אצרף לכאן תמונה של הממשק :



פעולות הממשק:

○ פונקציית `start_client(str,button)`:

תפקידה של פונקציה זאת היא להתחיל את הthread שמתחיל את הלקוח.
חשוב לי לציין לפני: למרות שיש שימוש כל פעם בלקוח אחד יש צורך להפעיל את זה threads על מנת שהממשק הגרפי יוכל להמשיך לרוץ ברקע.
 הפונקציה מקבלת אליה את הip שהכניס המשתמש בתיבה שקיימת בממשק ואת הכפתור שעליו לחצו כלומר את go. במידה והמשתמש לא הכניס ip הפונקציה מחזירה הודעת שגיאה של ip לא הוכנס. במידה והמשתמש הכניס ip אך הסרבר לא רץ או שהכתובת ip לא נכונה הממשק יקפיץ הודעת שגיאה נוספת (אך שונה מהראשונה).
 אם הכתובת ip נכונה והסרבר קיים התוכנית קוראת לפונקציה `client_activate()` שעליה אפרט בהמשך ומתחילה את השיתוף מסכים. בנוסף היא משנה את הכפתור go לstop.

במידה והמשתמש לחץ על הכפתור stop הפונקציה מפסיקה את שיתוף המסכים על ידי קריאה לפונקציה stop_client() שגם עליה ארחיב כאשר הסביר על הפונקציות של הלקוח. בנוסף היא מחזירה את הכפתור ל go.

○ פונקציית client_activate():

תפקידה של פונקציה זאת היא ליצור thread שמתחיל את שיתוף המסכים. זאת על ידי קריאה לפונקציה start_client שנמצאת בתוך המחלקה של client. היא לא מקבלת משתנים והיא לא מחזירה פלט כלשהו.

○ פונקציית start_server(button):

תפקידה של פונקציה זאתי הוא ליצור את threadn שיתחיל את השרת. כמו שהוסבר למעלה יש חשיבות להתחיל את השרבר כמו גם את הלקוח באמצעות thread על מנת שלא יקרוס הממשק הגרפי. הפונקציה מקבלת כקלט את הכפתור start זאת על מנת לשנות אותו לאחר שהכפתור נלחץ לstop במידה ואם המשתמש רוצה לכבות את השרת. במידה ואם נלחץ הכפתור stop הפונקציה מכבה את השרת ואת כל החיבורים הקיימים.

○ פונקציית server_activate():

תפקידה של פונקציה זאתי היא ליצור את threadn שמפעיל את השרת.

○ פונקציית find_ip():

תפקידה של פונקציה זאתי הוא להחזיר את כתובת ה ip הנוכחית של השרת.

○ פונקציית exit_program():

תפקידה של פונקציה זאתי היא לסגור את כל התוכנית. במידה ואם הלקוח או השרת עובדים הפעולה הבאה תסגור גם אותם.

○ פונקציית instruction():

פעולה פשוטה היוצרת information box שמסביר על הפרויקט וכיצד להשתמש בו.

פעולות הלקוח:

הערה חשובה לפני שאפרט על פעולות הלקוח והשרת. על מנת שאוכל לבצע עצירה של השרת או הלקוח ועל מנת שהפרויקט יעבוד חלק יותר וללא שגיאות נאלצתי ליצור את הלקוח והשרת בצורת מחלקות. אך בכל מקרה תמיד ניתן להמיר את הלקוח והשרת לפונקציות רגילות שמופעלות בנפרד אך שוב כפי שצוין השילוב עם הממשק הגרפי עלול להיות קשה יותר. אין קשר בין מחלקת הלקוח לבין מחלקת השרת מבחינת הורשה ותכנות מונחה עצמים.

○ פעולה בונה () init :

תפקידה של פעולה זאת הוא ליצור עצם מסוג client. המחלקת של הלקוח מכילה את התכונות הבאות: האם לעצור, חיבור socket, כתובת ip, ופורט. את התכונות היא מגדירה כברירת מחדל ולא מקבלת אף תכונה מהלקוח.

○ קיימות שתי פונקציות של () get_should_stop ו() set_connection:

פעולות get וset פשוטות לחלוטין.

○ פונקציית () stop_client:

תפקידה של פונקציה זאת הוא לעצור את חיבור הלקוח. פעולה זאת מבצעת זאת על ידי שינוי התכונה should_stop של הלקוח.

○ פונקציית () check_server:

תפקידה של פונקציה זאת הוא לבדוק האם השרת רץ כרגע. אם הסרבר רץ הפונקציה מחזירה True ומתחברת אליו, ואם השרת לא מחובר הפונקציה מחזירה False.

○ פונקציית () start_client:

תפקידה של פונקציה הוא להראות את המסך של השרת.

הפונקציה יוצרת מסך בגודל של 1440x720 ובודקת האם הלקוח רוצה להמשיך את השיתוף. במידה וכן הפונקציה מקבלת מהשרת את גודל התמונה הנוכחית ולאחר מכאן מתחילה לולאה שבה היא מקבלת את כל התמונה עד שכל הגודל הועבר. לאחר מכאן הפונקציה מנסה להציג את התמונה על המסך, במידה והצליחה היא מציגה את התמונה על המסך וחוזרת לתחילת הלולאה. במידה ולא הצליחה בעקבות שגיאות תעבורה כאלו או אחרות (שעליהם אפרט בהמשך) הפונקציה חוזרת לתחילת הלולאה ומתעלמת מהתמונה האחרונה. במידה והלקוח החליט שהוא רוצה לסיים את התוכנית, משתנה התכונה should_stop מה שעוצר את הלולאה ולאחר מכאן מסיים את הקישור של הלקוח עם השרת.

פעולות השרת:

○ פעולה בונה () init :

יוצרת שרת עם התכונות הבאות: `should_stop` (בודקת האם צריך להמשיך את השרת או לעצור כי המשתמש ביקש), `server_socket`, רשימת `threads`, כמות ה-`threads`, הפורט של השרת (נקבע 5000) ומיקום שמירת התמונה הנוכחית.

○ פעולות בונות ומאתחלות:

פעולות בסיסיות של `set` ו-`get`.

○ פונקציית () handle_client :

פעולה זאת שולחת את התמונה הנוכחית של המסך ללקוחות המחוברים. פעולה זאת מבצעת זאת על ידי לולאה שמפסיקה אך ורק אם הלקוח התנתק. במהלך הלולאה השרת שולח ללקוח תחילה את גודל התמונה ולאחר מכן את התמונה עצמה בחלקים עד שכל התמונה נשלחה.

○ פונקציית () run :

פונקציה זאת מאתחלת את השרת ומחכה ללקוחות. הפונקציה תפסיק אך ורק אם המשתמש החליט לסגור את השרת על ידי לחיצה על כפתור `stop`. כאשר לקוח מתחבר לשרת פונקציה זו יוצרת `thread` חדש ומתחילה את הפונקציה `handle_client`. בנוסף היא מגדילה את התכונה של מספר הלקוחות (מספר ה-`threads`).

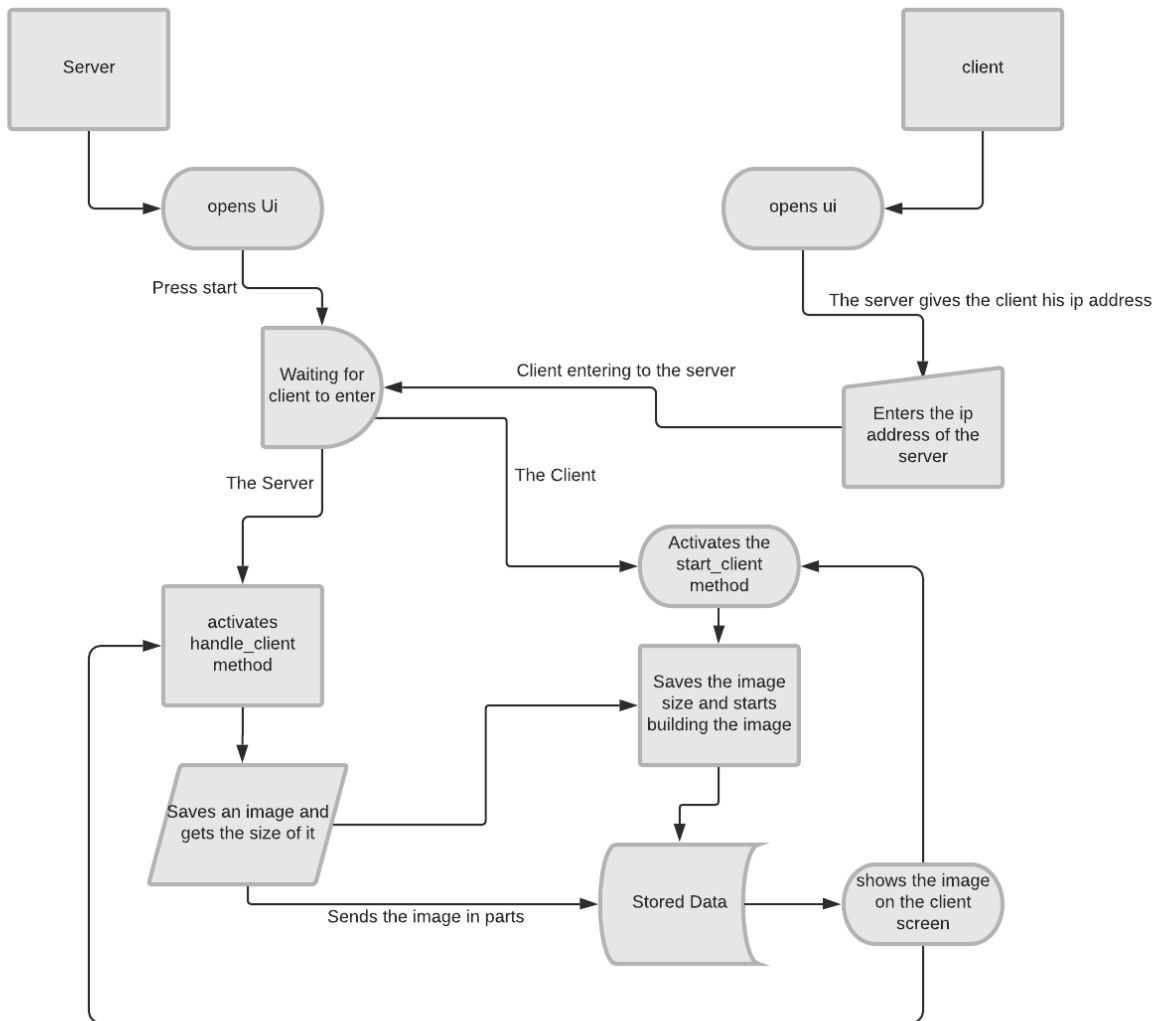
○ פונקציית () stop :

פונקציה זאת מפסיקה את פעולת השרת. פונקציה זאת מבצעת זאת על ידי חיבור לקוח חדש לשרת ושינוי `should_stop` על מנת שהשרת יפסיק את הלולאה של `run`. לאחר מכן היא מסיימת את כל ה-`threads` הקיימים ומאפסת את מספר ה-`threads`.

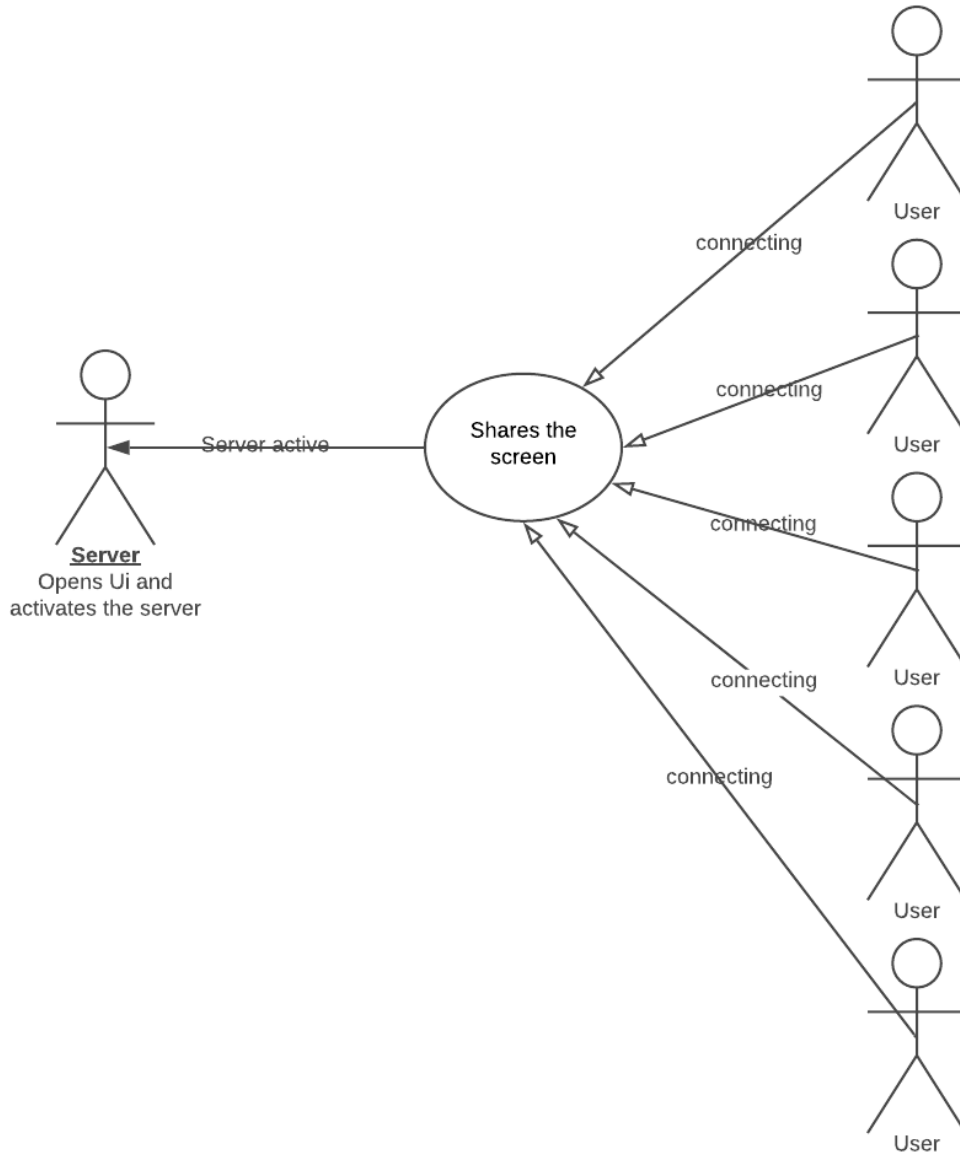
הקשרים בין היחידות השונות

Data flowchart

Shalev Zelinger | June 6, 2020



תרחיש use case



○ יכול להיות יותר מלקוח חדש

ארכיטקטורת הרשת

פרוטוקול (Protocol, תקן) הינו סט מוגדר של חוקים, הקובע כללים ברורים כיצד צריכה להיראות התקשורת בין הצדדים השונים. למעשה, פרוטוקול מכתוב לשני הצדדים בשיחה כיצד תראה התקשורת ביניהם. ובאמצעות צורה זו הם יכולים לדבר ולהבין האחד את השני.

אחד מהפרוטוקולים הנפוצים שקיים הוא פרוטוקול HTTP המשמש להעברת דפי האינטרנט אליהם אנחנו גולשים בדפדפן. הוא קובע כיצד ידבר הדפדפן עם השרת המרוחק, ובאיזו צורה יציג חזרה למשתמש את הדף שהגיש לו השרת. עמוד 85 להרחיב

פרוטוקול נוסף ונפוץ שקיים הוא פרוטוקול DNS (Domain Name Server) שקיים בשכבת האפליקציה. תפקידו העיקרי הינו לתרגם שמות דומיינים (כגון www.google.com או www.facebook.com) לבין כתובת ה IP הרלוונטית. למעשה, ניתן לחשוב על DNS כמעין "ספר טלפונים" – כמו שספר הטלפונים מתרגם בין שם של אדם או עסק (שאותו יותר קל לבני אדם לזכור) לבין מספר טלפון, כך ה DNS-שמאפשר לאתר כתובת IP באמצעות שם של אתר.

עד עכשיו הבאתי דוגמאות לפרוטוקולים בשכבת האפליקציה. כעת גם הביא דוגמאות לפרוטוקולים בשכבת התעבורה.

בשכבת התעבורה, פרוטוקולים יכולים להיות מבוססי קישור (Connection Oriented) או לא מבוססי קישור (Connection Less).

פרוטוקול TCP (Transmission Control Protocol) הוא דוגמה לפרוטוקול מבוסס קישור. כאשר משתמשים בפרוטוקול זה ראשית יש צורך ליצור קישור עם התוכנה המרוחקת וברגע שמתחברים כל חבילה שנשלחת תהיה חלק מאותו קישור.

פרוטוקולים מבוססי תקשורת מבטיחים אמינות בשליחת מידע. כלומר, מבטיחים שכל המידע שנשלח יגיע אל המקבל, וכן שהוא יגיע בסדר שבו הוא נשלח. עם זאת, לפרוטוקולים מבוססים קישור יש תקורה (Overhead) גבוהה יחסית. כלומר, יש מידע רב שנשלח ברשת בנוסף על המידע שרצינו להעביר. באם נרצה להעביר את המסר "שלום לכם" באמצעות פרוטוקול מבוסס קישור, עלינו להרים את הקישור לפני שליחת ההודעה, לסיים את הקישור בסיום, ולהשתמש במנגנונים שונים כדי להבטיח שהמסר אכן הגיע אל היעד. פעולות אלו לוקחות זמן ומשאבים, ולכן העברת המסר "שלום לכם" תהיה איטית יותר מאשר שליחת המסר מבלי הרמת הקישור.

פרוטוקול UDP (User Datagram Protocol) הוא דוגמה לפרוטוקול שאינו מבוסס קישור. כשאנו, בתור מפתחי שכבת האפליקציה, משתמשים ב-UDP בכדי לשלוח חבילה, אין הבטחה שהחבילה תגיע ליעדה. כמו כן, אין הבטחה שהחבילות יגיעו בסדר הנכון. אי לכך, אין גם צורך בהרמה וסגירה של קישור. באם מתכנת בשכבת האפליקציה רוצה לשלוח חבילה מעל פרוטוקול UDP, הוא פשוט שולח את החבילה.

קיים גם פרוטוקול עבור שכבת הרשת שהוא נקרא IP (Internet Protocol) שבאמצעותו אני מעבירים מידע בין אישיות שונות כאשר לכל ישות יש כתובת IP שונה.

פרוטוקול נוסף שקיים הוא פרוטוקול Ethernet, בו משתמשים כרטיסי רשת מסוג Ethernet, הכוונה היא לכרטיס רשת המתחבר באופן קווי.

ופרוטוקול אחרון שעליו אפרט הוא פרוטוקול ARP (Address Resolution Protocol). פרוטוקול זה ממפה בין כתובות לוגיות של שכבת הרשת לכתובות פיזיות של שכבת הקו.

הפרויקט שלי בנוי מכמה פרוטוקולים מרכזיים שצוינו למעלה. הפרויקט שלי מתבסס על העובדה ששיתוף המסכים בין המחשבים המחוברים צריך להיות מהיר ככל האפשר. לכן הפרוטוקול המרכזי שמתאר את הפרויקט שלי הוא פרוטוקול UDP. פרוטוקול זה אומנם גורם לחלק מהמידע הנשלח ללכת לאיבוד בדרכו ללקוח אך הוא אינו דורש חיבור ספציפי בין השרת ללקוח כך שכל אחד יכול להתחבר מבלי בעיה והוא מאפשר את השליחה המהירה ביותר של המידע מבלי לבצע בדיקות מיותרות.

במידה והייתי משתמש בפרוטוקול TCP השיתוף מסך היה מתבצע באופן איטי יותר אך מדויק יותר כך שיותר תמונות היו מועברות באופן מושלם אך מה שהיה מהווה בעיה מבחינת מהירות השליחה.

בנוסף לכך הפרויקט שלי מתבצע בהתאם לפרוטוקולים IP ו-Ethernet זאת משום שעל מנת לבצע את שיתוף המסכים הלקוח מכניס את הכתובת IP של השרת שהריץ את התוכנה ובאמצעות שליחת הכתובת IP הלקוח יודע לאן להתחבר והשיתוף מתחיל. יתרה מזאת מטרתי בפרויקט היא להפיץ את השיתוף בין חברי כיתה או בין מורה לתלמידים לכן במרבית הזמן האנשים יהיו באותו אזור ואף חשוב יותר באותו חיבור רשת. כלומר, למרות שהם יהיו במחשבים שונים המזהה רשת שלהם (שתי המספרים הראשונים של הכתובת IP) תהיה זהה. ישנה חשיבות לקח ששתי המחשבים יהיו באותו חיבור Ethernet זאת משום שבמידה ולא היו, חומת ההגנה של הרשת הייתה חוסמת את הפורט והיו צריכים להיעשות שינויים במחשב.

מדריך למשתמש

הוראות התקנה

אם ברשותך Pycharm קיים נא לקרוא את מה שרשום פה מתחת ולדלג לאיפה שיש כוכבית.

במידה ויש למשתמש Pycharm על המחשב:

ראשית במידה ואם יש במחשב הנוכחי Pycharm אין צורך לבצע את ההתקנה המלאה שאפרט כרגע אלא יש צורך להשתמש בpython 2.7 לכן במידה ואם Pycharm נמצא על Python 3.6 יש צורך להיכנס לאתר הרשמי של python להוריד את interpreter של python 2.7 (exe file) ולאחר מכאן בPycharm יש צורך ללחוץ על settings לאחר מכאן project interpreter ואז לשנות את interpreter מהinterpreter הנוכחי של interpreter 2.7.

במידה ולמשתמש אין Pycharm על המחשב:

ההורדה שאני ממליץ אומנם נוחה לי וידידותית לי אך באותו אופן ניתן תמיד להוריד את זה בכל דרך אפשרית כל עוד יש Pycharm עובד interpreter של 2.7.

אני אישית ממליץ על ההורדה של גבהים. הורדה זו נוחה וידידותית מאוד למשתמש בזמן שהיא מורידה כל מה שצריך בשביל שיהיה ניתן להתחיל לעבוד.

שלב 1: נכנסים לספר של גבהים של פיתון (חשוב מאוד פיתון לא רשתות)
קישור לספר: https://data.cyber.org.il/python/python_book.pdf

שלב 2: לוחצים על הקישור הראשון של התקנה סביבת העבודה בעמוד 8 ומורידים את הקובץ ששם.

שלב 3: לוחצים על התקן ועוקבים אחר הוראות ההפעלה. יש לבחור את המקום הורדה הנוח ביותר לך (המשתמש) כך שתזכור איכן הורדת את זה ותוכל להגיע לשם בעתיד.

שלב 4: לאחר סיום ההורדה יש להפעיל את Start שירד ולאחר מכאן ללחוץ על Pycharm.

ברגע זה הסתיימה ההורדה של סביבת העבודה עצמה וכעת אסביר מה השמות שצריך להעניק לכל קובץ שמשתמשים בו ואיזה קבצים צריך להוסיף לשם.

* חלק זה גם כאלה שיש להם Pycharm על המחשב צריכים לבצע

שמות הקבצים:

ראשית יש צורך לפתוח תיקייה חדשה של פרויקט (השם לא משנה) אך חשוב מאוד לזכור איכן נפתחה התיקייה על מנת לדעת לאן לייבא קבצים אחר כך.

שנית יש צורך לפתוח 3 קבצי פיתון שונים. אחד קוראים לו: server.py, השני קוראים לו: client.py, והשלישי קוראים לו: Ui (לזה ספציפית לא משנה איך תקראו).

כל אחד מהקבצים הבאים מכיל קוד שונה וב-Ui מבצעים import לשני הקבצים האחרים.

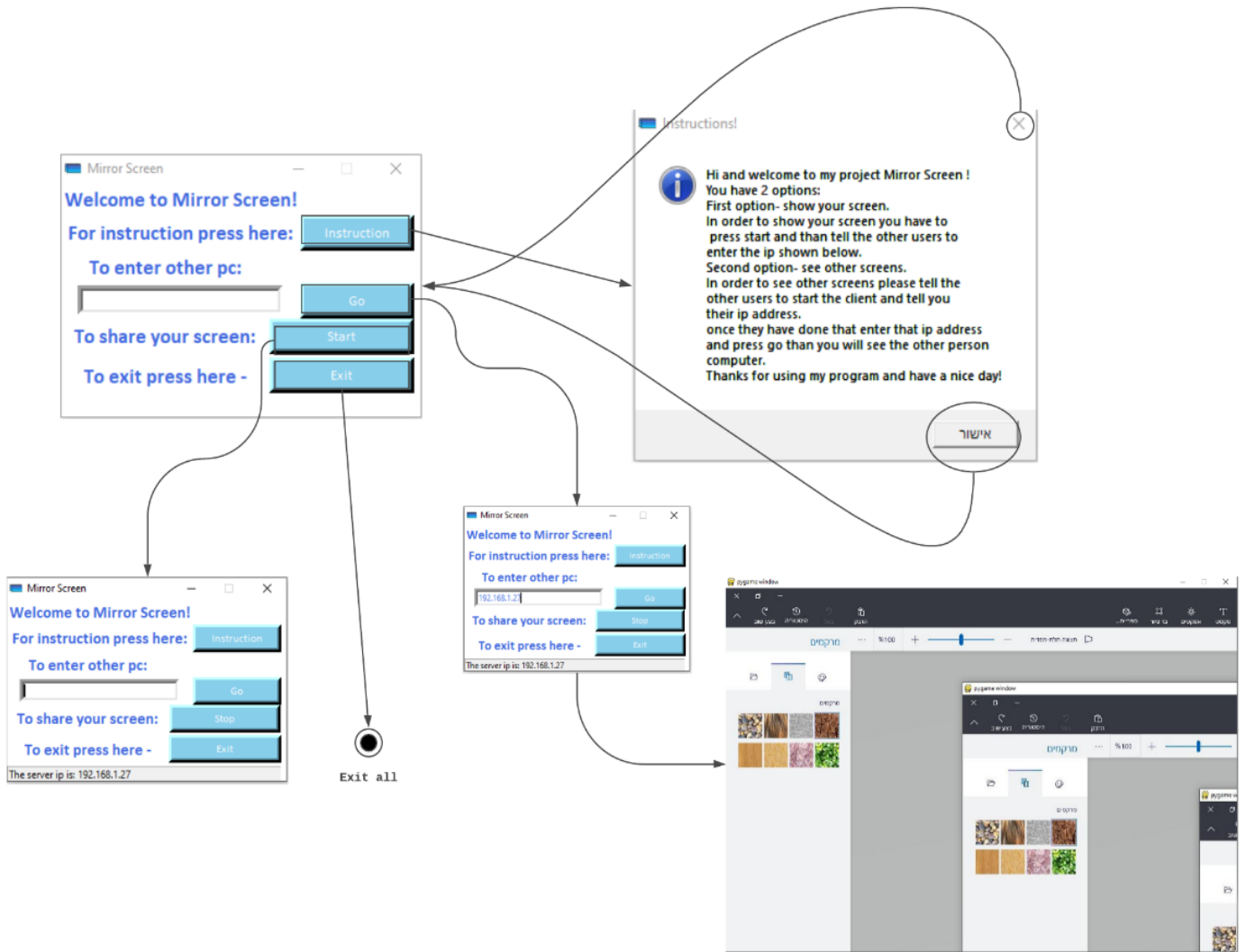
בהמשך אפרט על מה כל קובץ מכיל.

קבצים שיש להוסיף:

יש צורך להוריד את הקובץ של האייקון של הפרויקט על מנת שיהיה ניתן להפעיל אותו. יש צורך להוריד אותו על התיקייה שעל הפרויקט שאותה. חשוב מאוד לקרוא לו "mirrorImage.ico"

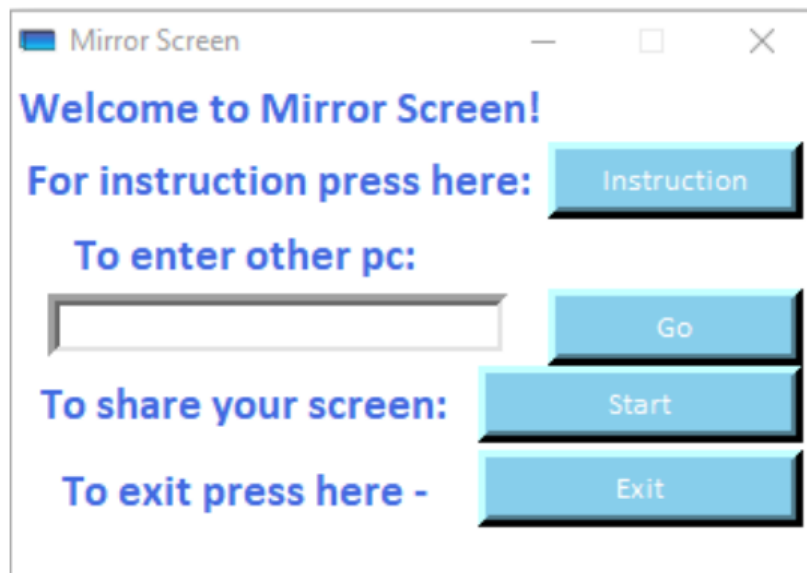
עד לכאן פירטתי על השלבים להורדה ועל קבצים שצריך להוסיף כעת אפרט על כל מסך ומה הוא עושה.

תרשים מסכים



תפקיד כל מסך

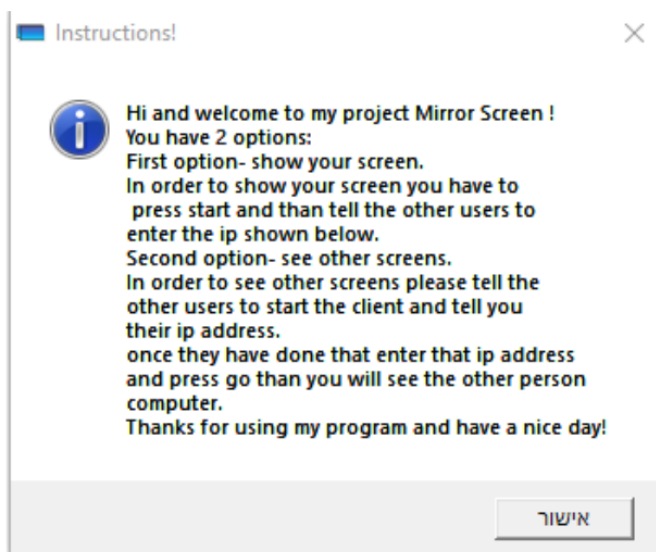
המסך הראשי:



הסבר: המסך הראשי משמש כמערכת ניווט לשאר המסכים בפרויקט.

במסך הראשי ניתן לראות כמה אופציות שונות, כעת אסביר על כל אופציה ומה היא עושה:

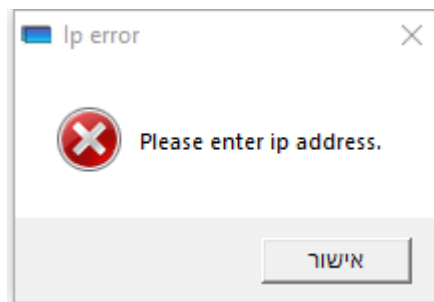
כפתור instruction: לחיצה על הכפתור פותחת חלונית מידע שמסבירה כיצד להשתמש בפרויקט:



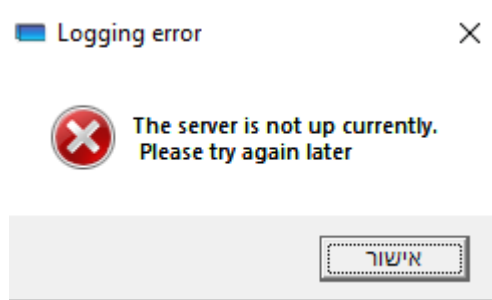
לחיצה על הכפתור אישור או איקס מחזירה אותך למסך הקודם.

כפתור הבא שאסביר עליו הוא כפתור ה go :

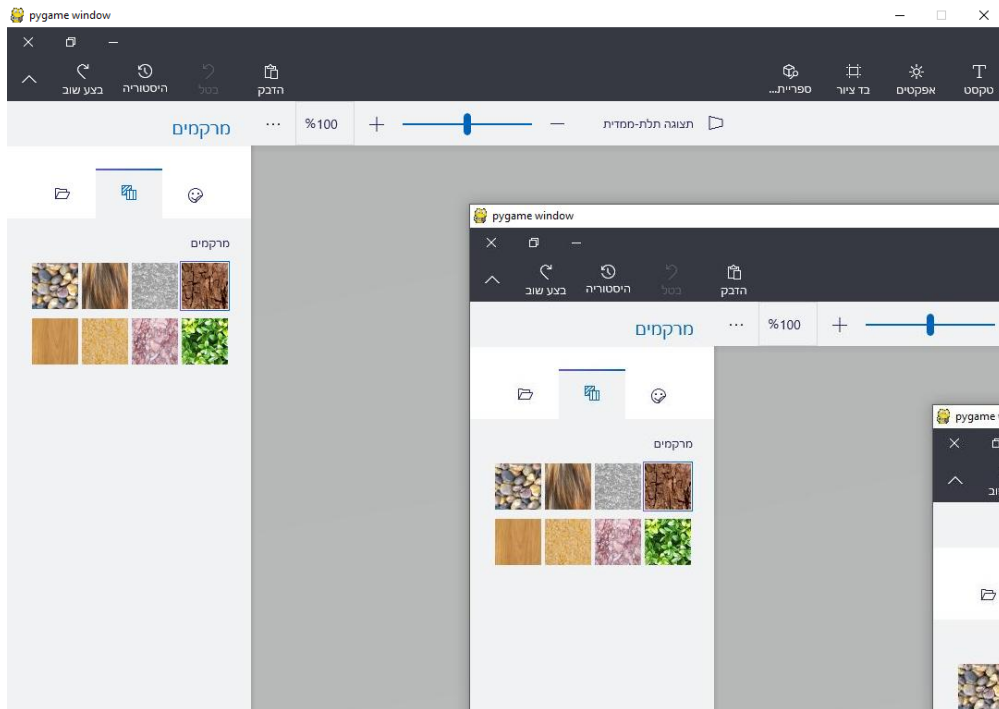
כפתור זה מטרתו להיכנס לשרת ובעצם ישר אחריו יופעל שיתוף המסכים. אך, יש צורך להכניס ראשית את הכתובת IP של השרת ורק לאחר מכן ללחוץ על go. במידה ונלחץ go לפני שהוכנסה כתובת IP קופצת ההודעה הבאה :



במידה ואם הוכנסה כתובת IP אך הכתובת אינה נכונה קופצת הודעת השגיאה הבאה :

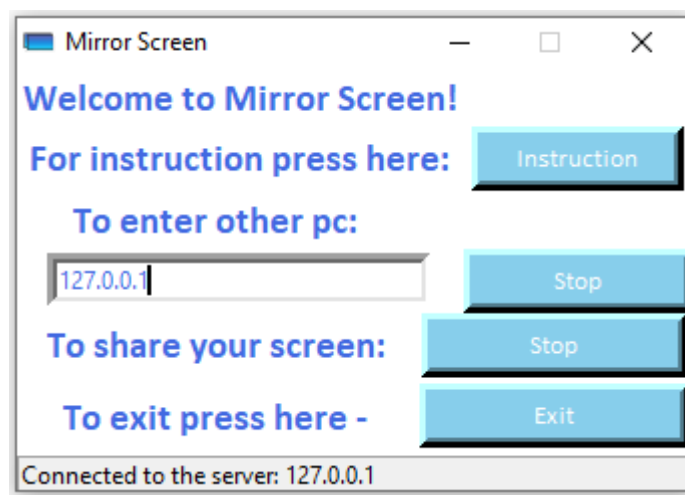


במידה ואם הוכנסה כתובת IP והשרת עובד שיתוף המסכים יחל לפעול:
(הדוגמה כאן היא כאשר מריצים על מחשב אחד אך ניתן להריץ על יותר ממחשב אחד ועל

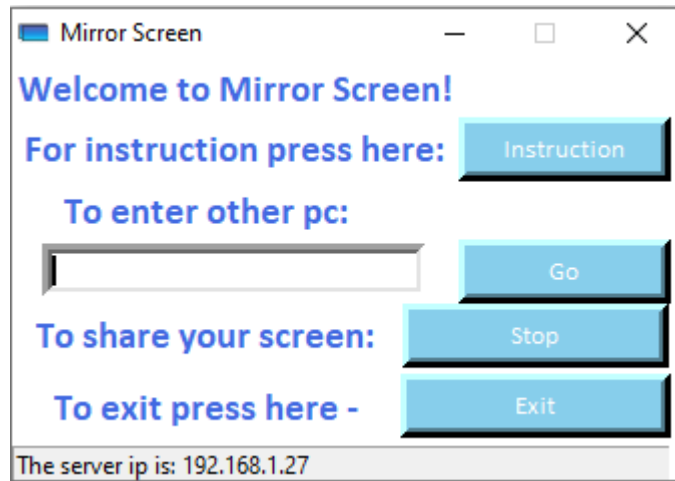


מחשבים באותו שרת אינטרנט)

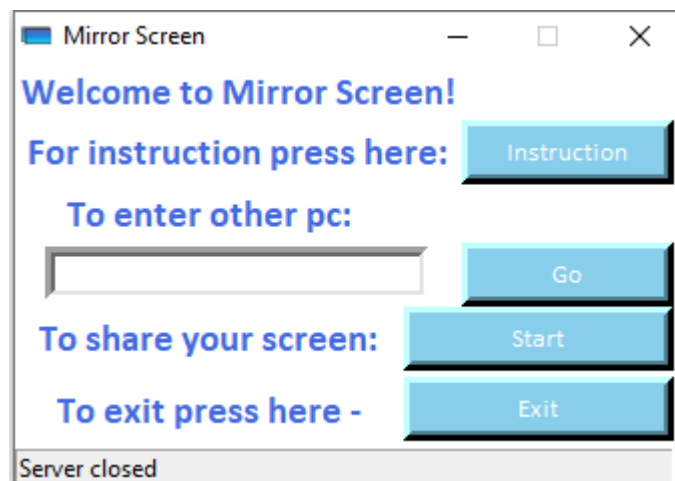
המסך שניתן לראות למעלה הוא בעצם שיתוף המסכים שמתרחש בין השרת ללקוחות השונים.
לסיום אפשרות אחרונה שנותרת עם הכפתור go היא כעת לאחר שהשיתוף החל ניתן לעצור את
השיתוף על ידי לחיצה על הכפתור בשנית כלומר לחיצה על Stop :



הכפתור Start : הכפתור Start מתחיל את השרת ובכך מאפשר כניסה של לקוחות. בלחיצה על
הכפתור Start status bar למטה משתנה וכך ניתן לדעת מהי כתובת ה-IP הנוכחית.



במידה ונלחץ הכפתור Go בשנית כלומר לוחצים על הכפתור Stop השרת מפסיק את פעילותו וסוגר את כל הלקוחות המחוברים.



כפתור אחרון שנותר הוא Exit: כפתור זה סוגר את כל הדברים הפתוחים. כלומר במידה ואם השרת פתוח הוא יסגור את השרת. אם הלקוח מחובר אז הוא יסגור גם את הלקוח בנוסף לשרת ולסיום הוא יסגור את התוכנית.

חשוב לציין שסגירת התוכנית בלחיצה על כפתור האיקס של toolbar תוביל לאותה תוצאה.

הודעות שונות שקופצות במהלך התוכנית:

את הודעות השגיאה הצגתי כבר במהלך ההסברים כעת אציג את הודעות שמופיעות בstatus bar כלומר בשורת טקסט שנמצאת בתחתית של הממשק.

הודעה ראשונה- הפעלת השרת

```
The server ip is: 192.168.1.27
```

הודעה שניה - סגירת השרת

```
Server closed
```

הודעה שלישית - פתיחת הלקוח (כאשר לא הוכנסה כתובת IP)

```
Ip address was not given
```

הודעה רביעית - פתיחת הלקוח (כאשר לא השרת לא עובד)

```
Connecting to the server...
```

```
Failed to log-in to the server.
```

הודעה חמישית - חיבור לקוח

```
Connected to the server: 127.0.0.1
```

הודעה שישית - סגירת הלקוח

```
Client stopped connection
```

מדריך למפתח

על מנת להסביר כיצד לפתח את הפרויקט אני אסביר על כל קובץ בנפרד ולסיום אני אסביר על התוכנית שמשלבת את כולם כלומר את קובץ הממשק. כל הקבצים ממוקמים בתיקייה שמכילה את הפרויקט.

הקובץ ראשון - server.py:

הסבר על תוכן הקובץ:

באמצעות קובץ זה המשתמש מפעיל את השרת ששולח ללקוחות את התמונה הנוכחית של המסך שלו. הקוד מפעיל את השרת ולאחר מכאן מחכה ללקוחות. כאשר מתחבר לקוח הוא מתחיל את הפעולה וחוזר להמתין ללקוח הבא.

כעת אציג את הקוד בחלקים ואסביר על כל חלק מה תפקידו ואף על משתנים חשובים אסביר מה הם עושים.

הסבר על הקוד:

תחילה יש צורך לייבא את הספריות הבאות:

```
import socket
```

```
import os
```

```
import threading
```

```
from PIL import ImageGrab
```

socket – על מנת לבצע את הקישור בין השרת ללקוח. באמצעות socket ניתן להשתמש בפקודות שמעבירות מידע בין השרת ללקוח או מקבלות מידע.

os- במחלקה זאת אני עושה שימוש פעם אחת כל פעם ששולחים תמונה חדשה. באמצעות מחלקה זאת אני יכול להשתמש במתודה os.path.getsize(filename) שמאפשרת לי לקבל את הגודל של הקובץ שהכנסתי בתוך הסוגריים.

Threading- באמצעות threading אני יכול להפעיל כמה לקוחות במקביל.

ImageGrab- באמצעות ספרייה זו אני יכול לבצע צילום של התמונה הנוכחית.

לאחר מכאן הגדרתי את הקבועים הבאים:

```
PORT = 5000
```

```
BUFFER_SIZE = 1024
```

```
LISTEN = 5
```

ב-PORט אני משתמש כדי ליצור את השרת.

ב-SIZE_BUFFER אני משתמש כאשר אני רוצה לשמור חלקים של התמונה שישלחו לאחר מכאן ללקוח.

ב-LISTEN אני משתמש על מנת לקבוע את מספר הלקוחות שיכולים להתחבר.

כעת לפני שאסביר על הפעולה החיצונית שקיימת בקובץ אסביר על המחלקה ShareServer:

```
class ShareServer:
```

```
    """ creates a class which activates the server and deactivates it at will. """
```

```
    def __init__(self):
```

```
        self.should_stop = True
```

```
        self.server_socket = socket.socket()
```

```
        self.threads = []
```

```
        self.port = PORT
```

הפעולה הראשונה במחלקה הפעולה הבונה. למחלקה 4 תכונות שהן האם צריך לעצור, socket של השרת, רשימת threads שמכילה את הלקוחות המחוברים ותכונה שמכילה את הפורט השרת.

כעת אעבור על הפעולות הבסיסיות של המחלקה (פעולות אלו מכילות כבר docstring שמסביר היטב מה הן עושות לכן לא אפרט יותר מדי):

```
    def is_running(self):
```

```
        """ Returns whither the server is running or not """
```

```
        return not self.should_stop
```

הפעולה מחזירה האם השרת רץ או לא. אם self.should_stop = false משמע שהשרת רץ ולכן הפעולה מחזירה True. אם ה-self.should_stop = false הפעולה תחזיר שהשרת מכובה.

```

def stop(self):
    """ stops the server. """
    self.should_stop = True # stops sharing the screen
    socket.socket(socket.AF_INET, socket.SOCK_STREAM).connect(("127.0.0.1",
self.port)) # connects to the server
        # so it can close it
    self.server_socket.close()
    for i in self.threads: # Closes all the existing threads
        i.join()
    self.threads = []

```

הפעולה הבאה מפסיקה את פעולת השרת. הפעולה משנה את ערך התכונה של האם צריך לעצור ל True מה שאומר שהסרבר יעצור לאחר מכאן היא מחברת לקוח שמטרתו להמשיך את הלולאה שנמצאת ב run מה שמאפשר את סגירת השרת.

לאחר מכאן הפעולה סוגרת את כל ה threads הקיימים על ידי הפעולה Join ומאפסת את מערך ה threads.

כעת אסביר על הפעולה run שמפעילה את השרת ואחראית על חיבור לקוחות :

```
def run(self):
    """ opens the server and waiting for clients to connect. """
    self.should_stop = False
    server_socket = socket.socket()
    server_socket.bind(("0.0.0.0", self.port))
    # CR: Set 5 as const in this file
    server_socket.listen(LISTEN)
    while self.is_running():
        c, addr = server_socket.accept()
        thread = threading.Thread(target=handle_client,
            args=("Thread"+str(len(self.threads)),c))
        self.threads.append(thread)
        self.threads[len(self.threads)-1].start()
    server_socket.close()
```

הפעולה run היא אחת הפעולות החשובות והמרכזיות שקיימות בקובץ ולכן אסביר עליה בקפידות. תחילה הפעולה משנה את ערך התכונה של האם לעצור ל False מה שאומר שהשרת מתחיל לפעול. לאחר מכאן היא יוצרת את השרת על ידי הפעולה bind ומאפשרת האזנה לעד 5 לקוחות (ערכו של LISTEN).

כעת היא בודקת האם השרת עובד ומתחילה את הלולאה :

דבר ראשון שהיא עושה זה בודקת האם השרת פועל. אם כן, הפעולה ממשיכה ומחכה לחיבור לקוחות, עד שלא מתחבר לקוח הפעולה לא ממשיכה את הפעולה.

כאשר מתחבר לקוח הפעולה יוצרת Thread חדש שמכיל socket עם השרת ושם ייחודי עבור כל Thread ומבצע את הפעולה handle_client שעליה אפרט בהמשך.

לאחר מכאן הפעולה מכניסה את ה Threadn לרשימת threadsn (ומגדילה את גודלה) ואז מתחילה את Threadn ושולחת אותו לבצע את הפעולה.

בשלב זה מתקשרת הפעולה stop. במידה ואם המשתמש החליט לעצור את השרת בדרך כזאת או אחרת הפעולה מדמה התחברות לקוח על מנת שהיא תוכל להמשיך את הלולאה ולהגיע לתנאי של whilen שכעת השתנה והוא כבר לא מאפשר המשך של הלולאה.

לסיום הפעולה סוגרת את socketn וכך היא מכבה את השרת.

כעת אסביר על הפעולה handle_client :

```

def handle_client(name, sock):
    """ Sends the current screen to the client non-stop. creating the screen sharing. """
    try:
        img_place = 'server'+str(name)+' .jpg'

        while True:

            img = ImageGrab.grab() # gets a picture of the current screen and saves it in the
server computer

            img.save(img_place) # saves it in the project directory

            filename = img_place

            sock.send(str(os.path.getsize(filename)) + 'NewFile') # sending the client the
size of the image

            with open(filename, 'rb') as f: # opens the file as binary object.

                bytes_to_send = f.read(BUFFER_SIZE) # gets buffer size part of the image

                sock.send(bytes_to_send) # sends the first part

                while bytes_to_send != "": # sends all the left parts.

                    bytes_to_send = f.read(BUFFER_SIZE)

                    sock.send(bytes_to_send)

        except Exception as e:

            pass

    sock.close()

```

פעולה זו היא הפעולה המרכזית של הקובץ שבאמצעותה הלקוח מקבל את התמונה הנוכחית. הקוד נמצא תחת try except על מנת שבמידה ואם הלקוח החליט להתנתק הפעולה תפסיק ולא תקרוס.

בתחילת הפעולה הפעולה יוצרת את המיקום שבו תשמר התמונה ואת שמה. המיקום יהיה בתיקיית הפרויקט ושמה ייקבע לפי כמות הthreads הקיימים.

לאחר מכאן הפעולה מתחילה לולאה אינסופית ששולחת את התמונה הנוכחית שכעת אסביר כיצד היא עושה זאת.

בשלב הראשון התמונה מצלמת את המסך הנוכחי ושומרת אותו בתיקיית הפרויקט. לאחר מכאן היא שולחת ללקוח את הגודל של התמונה כדי שיידע מתי הוא צריך לעצור את הלולאה. לאחר מכאן היא מבצעת את הקריאה של הקובץ ושולחת כל חלק ממנו בנפרד.

בתחילת כל לולאה הפעולה קוראת חלק של הקובץ בגודל של `BUFFER_SIZE` (כלומר 1024) ולאחר מכאן היא שולחת את המידע שקראה כך היא מבצעת עד שמסתיימת הלולאה ובסיום היא סוגרת את הקובץ.

כך הלולאה מתרחשת עד שהמשתמש החליט לסגור את החיבור. במידה ואם החיבור נסגר השרת סוגר את ה `socket` בינו לבין הלקוח.

הקובץ השני - client.py :

הסבר על תוכן הקובץ :

באמצעות קובץ זה הלקוח מתחבר לשרת ומקבל את התמונה הנוכחית. למשתמש יש את האמצעות לעצור את חיבור הלקוח בכל רגע ורגע בזמן הנתון. במידה ועצר את התחברות הלקוח הוא תמיד יכול להתחבר בשנית.

כעת אציג את הקוד בחלקים ואסביר על כל חלק מה תפקידו ואף על משתנים חשובים אסביר מה הם עושים.

הסבר על הקוד :

תחילה יש צורך לייבא את הספרייה הבאות :

```
import socket
```

```
import pygame as pg
```

socket – על מנת לבצע את הקישור בין הלקוח לשרת. באמצעות socket ניתן להשתמש בפקודות שמקבלות מידע מהשרת.

Pygame - באמצעות הספרייה pygame ניתן לראות את התמונה על מסך בגודל של 1440x720.

לאחר מכאן הגדרתי את הקבועים הבאים :

```
PORT = 5000
```

```
BUFFER_SIZE = 1024
```

```
FILENAME = r'client.jpg'
```

```
SIZE_CONTENT_SEPERATOR = 'NewFile'
```

PORT אני משתמש כדי להתחבר לשרת.

ב- SIZE_BUFFER אני משתמש כאשר אני רוצה לקבל חלקים של התמונה מהשרת.

ב- FILENAME אני משתמש כדי לדעת איכן לשמור את התמונה ובאיזה שם כלומר לשמור אותה בתיקיית הפרויקט.

ב- SIZE_CONTENT_SEPERATOR אני משתמש כאשר אני רוצה לוודא שנשלח רק הגודל של התמונה ולא נשלח גם חלק מהתמונה.

כעת אסביר על המחלקה Client :

class Client:

```
def __init__(self, ip):
    """ Creates a client object """
    self.should_stop = True
    self.client_socket = socket.socket()
    self.ip = ip
    self.port = PORT
```

המחלקה client מכילה 4 תכונות. התכונה הראשונה היא האם צריך לעצור את הלקוח, התכונה השנייה היא חיבור socket של הלקוח, התכונה השלישית היא כתובת ה-IP אליה הוא מתחבר והתכונה הרביעית היא הפורט שאליו הלקוח מתחבר.

כעת אסביר על פעולות המחלקה. כמו בשרת רוב פעולות המחלקה בעלות docstring שמסביר את הפעולה באופן מצוין אך עדיין אפרט בכמה מילים כדי להסביר את הפעולות באופן טוב יותר (או בעברית):

```
def is_running(self):
    """ Returns whether the client is running or not """
    return not self.should_stop
```

הפעולה is_running בודקת האם הלקוח מחוברת לשרת ורץ. אם self.should_stop = false משמע שהלקוח רץ ולכן הפעולה מחזירה True. אם ה- self.should_stop = false הפעולה תחזיר שהלקוח מכובה.

```
def stop_client(self):
```

```
    """ Stops the client """
```

```
    self.should_stop = True
```

הפעולה stop_client משנה את ערכו של should_stop כך שהפונקציה תעצור את חיבור הלקוח לשרת.

```
def check_server(self):
```

```
    """ Checks if the server is online or offline. """
```

```
    try:
```

```
        self.client_socket.connect((self.ip, PORT)) # tries to connect so it can check if
the server is up or not
```

```
        self.should_stop = False
```

```
        return True
```

```
    except socket.error as e:
```

```
        pass
```

```
    return False
```

הפעולה הבאה מדמה חיבור של הלקוח לשרת על מנת לבדוק האם הוא מחובר או לא. הפעולה מנסה להתחבר לשרת. אם היא הצליחה היא משנה את ערכו של should_stop ומחזירה True. במידה והשרת אינו זמין או כתובת ה ip אינה נכונה הלקוח מפסיק את ניסיון ההתחברות ומחזיר False.

```

def start_client(self):
    """ starts showing the screen share """
    pg.init() # builds the pygame screen
    display_screen = pg.display.set_mode((1440, 720), 0)
    clock = pg.time.Clock() # sets a clock
    while not self.should_stop: # runs the loop until the user stops it
        for event in pg.event.get(): # checks if the exit button on the ui or on the toolbar
            was pressed
                if event.type == pg.QUIT:
                    self.should_stop = True
                    continue
                data = self.client_socket.recv(BUFFER_SIZE) # receives the size of the image
                temp = data.split(SIZE_CONTENT_SEPERATOR)
                if temp[0].isdigit(): # makes sure that the file size is long and doesnt contain part
                    of the image
                        file_size = long(temp[0])
                    else:
                        continue
                with open(FILENAME, 'wb') as f: # opens the file
                    total_rcv = 0
                    if len(temp) > 1: # makes sure part of the image was not sent with the size
                        f.write(temp[1]) # if it does it inputs it into the file
                        total_rcv += len(temp[1])
                    while total_rcv < file_size: # builds the image on the client computer
                        diff = file_size - total_rcv
                        if diff < BUFFER_SIZE: # checks if this is the last receive or not
                            data = self.client_socket.recv(diff)

```

```

else:
    data = self.client_socket.recv(BUFFER_SIZE)
    total_recv += len(data)
    f.write(data)
try:
    show_img = pg.image.load(FILENAME) # shows the image on the screen
    display_screen.blit(show_img, (0, 0))
    pg.display.flip()
    # print 'success !'
except Exception as e:
    print "Error occurred - msg: " + e.message
    clock.tick(60)
pg.quit() # closes pygame so it can be opened again when restarted
self.client_socket.close() # close connection to the socket

```

הפעולה `start_client` היא הפעולה החשובה ביותר שקיימת בלקוח ותפקידה להציג את תמונת השרת על מסך הלקוח. לכן אסביר עליה בקפידות.

ראשית בתחילת הפעולה הלקוח מגדיר את המסך ובונה אותו. הוא בונה מסך בגודל של `.1440x720`.

לאחר מכאן מתחילה הלולאה להעלות התמונה על המסך.

בתחילת כל הלולאה לפני שהיא מתחילה זה בודק האם המשתמש רוצה להפסיק את החיבור. במידה ואם הלקוח רוצה להמשיך את הצגת המסך של השרת הלולאה ממשיכה.

לאחר תנאי הכניסה ללולאה נעשית בדיקה האם הלקוח לחץ על כפתור האיקס של `toolbar` על מנת לבדוק אם הוא סגר את התוכנית. אם כן זה משנה את ערכו של `should_stop` ומדלג על הלולאה האחרונה.

לאחר שסיים עם הבדיקות נעשה השלב הראשון על מנת להראות את תמונת השרת על המסך. ראשית הלקוח מקבל מהשרת את הגודל של התמונה. לאחר מכאן הוא עושה לזה `split` זאת על מנת להימנע ממצב שבו הגודל של התמונה התמזג עם חלק מהתמונה עצמה.

במידה ואם חלק מהתמונה אכן נשלח ביחד עם הגודל קיים תנאי שבודק האם עורך הרשימה שנוצרה גדול מאחד ואם כן הוא מכניס את גודל התמונה למשתנה אחד ואת שאר התמונה עצמה

כותב בfile שיצר על המחשב שלו. במידה ואם לא נשלח הגודל עם התמונה עצמה הוא ישר מכניס את הגודל לתמונה וממשיך הלאה להמשך שליחת התמונה.

כעת נעשית לולאה שבודקת כל פעם האם כל התמונה נשלחה או לא כלומר האם הגודל של כל מה שנשלח עד עכשיו שווה או גדול מהגודל של התמונה שנשלח בתחילת הלולאה הראשית.

במידה ואם לא נשלח הלולאה מבצעת recv של התמונה מהשרת וכותבת את התמונה בfile נפרד שנמצא על מחשב הלקוח.

כאשר כל התמונה נשלחה הלקוח מנסה להציג את התמונה באמצעות Pygame. במידה ואם הקובץ לא ניתן להצגה השיתוף ממשיך ומדלגים על התמונה הנוכחית.

במידה ואם המשתמש החליט להפסיק את הצגת מסך השרת בשלב כזה או אחר הלולאה מפסיקה ולאחר מכאן סוגרים את pygame ואת חיבור הsocket לשרת.

הקובץ השלישי - Ui.py :

הסבר על תוכן הקובץ :

הקובץ Ui הוא בעצם הקובץ העיקרי של הפרויקט והוא הקובץ אשר אותו מריצים על מנת להפעיל את הפרויקט.

קובץ זה מכיל את כל הקבצים הקודמים שעליהם דיברתי ולכן יכול גם אזכורים וקריאות לפעולות שלהם.

באמצעות קובץ זה ניתן לראות כפתורים והסברים שונים שעליהם אסביר בהמשך.

על מנת להסביר כיצד פועלת התוכנית אני אסביר על כל כפתור ואל המקום שאליו הוא שולח את המשתמש כך שכל כפתור מפעיל פעולה שונה. באמצעות זה אוכל להסביר את הפיתוח של הממשק הגרפי בדרך הטובה ביותר.

הסבר על הקוד :

ראשית אפרט על הספריות שצריך לייבא לקוד :

```
from Tkinter import *
from server import *
from client import *
from threading import *
import tkMessageBox
import socket
```

כעת אסביר על כל ספרייה ומה היא מאפשרת

Tkinter- הספרייה הראשית שבאמצעותה יש ממשק גרפי. הספרייה מכילה אופציה לפתח מסך שמכיל כפתורים, תיבות טקסט ותיבות הכנסת טקסט שונות.

Server- באמצעות פעולה זו ניתן לייבא את מחלקת השרת ואת כל הפעולות שנמצאות בקובץ הזה.

Client- באמצעות פעולה זו ניתן לייבא את מחלקת הלקוח ואת כל הפעולות שנמצאות בקובץ הזה.

Threading- באמצעות ספרייה זאת ניתן להפעיל את השרת או הלקוח ולהפעיל את הממשק פועל ועובד. אסביר זאת בהמשך כאשר אבצע שימוש בthreada.

tkMessageBox- באמצעות ספרייה זאת ניתן להקפיץ הודעות שונות כגון הודעות מידע או הודעות שגיאה.

Socket - באמצעות ספרייה זאת הצלחתי למצוא את כתובת IP של השרת.

כעת אפרט על המשתנים הגלובליים שהגדרתי לפני שאפרט על הממשק עצמו:

```
root = Tk()
```

```
share_server = ShareServer() # The server object
```

```
share_client = Client("") # The client object
```

```
status_bar = Label(root, text="", bd=1, relief=SUNKEN, anchor=W) # The status bar
```

ראשית אסביר על האובייקט root. Root זה בעצם המסך הראשי של הממשק. באמצעות root אני יכול להציג את הפריטים השונים שהוספתי למסך אחד ספציפי.

Share_server הוא אובייקט של המחלקה שייבאתי מ server. באמצעות עצם זה אני יכול לבצע פעולות שונות שקשורות לשרת כגון להפעיל אותו לסגור אותו ולחבר לקוחות שונים.

Share_client הוא אובייקט של המחלקה שייבאתי מ client. באמצעות עצם זה אני יכול לבצע פעולות שונות שקשורות ללקוח כגון להפעיל את החיבור לשרת, בדיקה אם השרת זמין לחיבור ולסגור את הלקוח אם המשתמש ירצה בזאת.

status_bar שנמצא בחלק התחתון של הפרויקט שמראה לי מה הפעולה האחרונה שהתרחשה ונותן לי מאין אינדיקציה לכך שדברים אכן מתרחשים בפרויקט.

כעת אסביר על הדברים הראשיים שהגדרתי בתחילת הפרויקט כגון אייקון וכותרת:

```
def main_interface():
```

```
    """ Creates the main interface and from that navigates you to different functions. """
```

```
    # Define Default screen:
```

```
    # Make the gui window 335x200 and place it 300 pixels from the top
```

```
    # and left of the screen.
```

```
    root.geometry('{ }x{ }+{ }+{ }'.format(335, 210, 300, 300))
```

```
    root.title('Mirror Screen')
```

```
    root.iconbitmap('mirrorImage.ico')
```

```
    root.configure(background='white')
```

```
    root.resizable(0, 0)
```

```
    root.pack_propagate(0)
```

```
    root.protocol("WM_DELETE_WINDOW", on_closing)
```


הפעולה הראשונה geometry יוצרת מסך שגודלו 335x210 וממקמת אותו 300 פיקסלים מהחלק העליון של המסך.

Title יוצרת את הכותרת של חלון הממשק והכותרת של הפרויקט שלי היא : Mirror Screen.

Iconbitmap הוא הפריט הבודד היחידי שצריך לייבא לפרויקט על מנת שלא יהיו בעיות והוא בעצם אייקון קטן שמופיע בצד שמאל למעלה של הממשק על toolbar. שזהו בעצם, אייקון קטן שהכנתי בעצמי.

פעולה נוספת שקיימת שם היא root.configure שבעצם משנה את הגדרות מסך הרקע של הממשק ובפעולה זאת שיניתי את זה ללבן.

Root.resizable, Root.pack_propagate הן שתי פעולות שביחד מטרתן למנוע מהמשתמש לשנות את גודל המסך הנוכחי של הממשק.

Root.protocol זאת בעצם פעולה שמאפשרת לבצע בדיקה האם הלקוח לחץ על כפתור היציאה של הממשק הגרפי זאת על מנת שאני אוכל לבצע סגירה מוחלטת של כל הדברים הרצים בקוד.

במידה ואם הלקוח לחץ על איקס בעקבות הפעולה הזאת זה מפעיל את הפעולה on_closing :

```
def on_closing():
```

```
    """ This function makes sure everything is being closed when pressing exit. """
```

```
    if tkMessageBox.askokcancel("Quit", "Do you want to quit?"):
```

```
        exit_program() # exit everything if the client wants to exit. (including active
server)
```

פעולה זאת סוגרת את כל הדברים הרצים בתוכנית במידה ואם הלקוח החליט לסגור את הממשק ולא באמצעות כפתור הexit.

הפעולה exit_program :

```
def exit_program():
```

```
    """ Exit the program (closing everything) """
```

```
    global share_server
```

```
    global share_client
```

```
    global root
```

```
    if share_client.is_running(): # If the client is up the function will stop it.
```

```
        share_client.stop_client()
```

```
    if share_server.is_running(): # If the server is up the function will stop it.
```

```
        share_server.stop()
```

```
root.destroy()
```

```
root.quit()
```

הפעולה הבאה מבצעת בדיקה האם השרת והלקוח אכן רצים ברקע ואם במידה והם כן רצים היא סוגרת אותם ולאחר מכאן סוגרת את הממשק הגרפי ואת התוכנית לחלוטין.

כעת אסביר על הlabels השונים שקיימים בפרויקט שלי:

```
main_label = Label(root, text='Welcome to Mirror Screen!', font='Calibri 14 bold',
,fg='royal blue
```

```
bg='white') # Welcome label
```

```
info_label = Label(root, text='For instruction press here: ', font='Calibri 14 bold',
,fg='royal blue
```

```
.bg='white') # Go to instruction label
```

```
,label1 = Label(root, text='To enter other pc: ', font='Calibri 14 bold', fg='royal blue
```

```
bg='white') # Enter other pc label
```

```
,label2 = Label(root, text='To share your screen: ', font='Calibri 14 bold
```

```
fg='royal blue', bg='white') # Share your screen label
```

```
exit_label = Label(root, text='To exit press here -', font='Calibri 14 bold', fg='royal
,blue
```

```
bg='white') # Exit label
```

```
main_label.grid(row=0, column=0, columnspan=2)
```

```
info_label.grid(row=1, column=0, columnspan=2)
```

```
label1.grid(row=2, column=0)
```

```
label2.grid(row=4, column=0)
```

```
exit_label.grid(row=6, column=0, pady=5)
```

כעת אפרט על כל label מה הוא עושה ואיכן הוא ממוקם.

label1 main_label הוא בעצם ההודעה הראשית שמוצגת בראש הממשק והיא Welcome to Mirror Screen.

label2 info_label הוא label שצמוד לכפתור information על מנת שהמשתמש יידע על מה ללחוץ כדי להגיע להוראות הפרויקט וכיצד להשתמש בו. כתוב בlabel:

For instruction press here:

הוא label1 label2 שמעל תיבת הטקסט שאליה מכניסים את כתובת ה ip של השרת

ולאחר מכאן מתחברים אליו. רשום בו: To enter other pc:

הוא label2 label1 שנמצא ליד כפתור startn שמפעיל את השרת. רשום בו:

To share your screen:

והוא exit_label label1 נמצא ליד הכפתור exit ורשום בו: - To exit press here.

כעת אסביר על כל כפתור ועל הפעולה שהוא מפעיל:

```
button_go = Button(root, text='Go', padx=38, borderwidth=5, fg='white',  
font='Calibri 10',
```

```
bg='sky blue', command=lambda: start_client(e1.get(), button_go)) # The  
client button
```

```
button_go.grid(row=3, column=2, columnspan=1)
```

הכפתור Go נמצא ליד תיבת הטקסט שאליה מכניסים את כתובת ה ip ולכן אציג גם את תיבת הטקסט לפני שאפרט על פעולת ה start_client.

```
e1 = Entry(root, borderwidth=5, width=30, fg='royal blue') # Entering ip address
```

```
e1.grid(row=3, column=0, columnspan=2)
```

הפעולה הבאה יוצרת תיבת טקסט בגדלים הנתונים ובצבע הנתון. היא ממקמת אותו מתחת label1 label2 כלומר to enter other pc.

הכפתור קורא לפעולה start_client אך הוא שולח גם פרמטרים לכן יש צורך להוסיף את המתודה lambda. הפרמטרים שהכפתור שולח הוא את המידע שהוכנס לתיבת הטקסט ואת הכפתור עצמו.

כעת אסביר על הפעולה :start_client()

```

def start_client(ip, button):
    """ Starts the client request """
    global share_client
    if ip == "" and button["text"] == "Go": # Checks if ip was entered
        tkMessageBox.showerror("Ip error", "Please enter ip address.")
        status_bar.configure(text="Ip address was not given ")
        return

    if button["text"] == "Go": # checks if the button is Go or Stop
        share_client = Client(ip)
        status_bar.configure(text="Connecting to the server...")
        if share_client.check_server(): # if the server is up the client will connect to the
server
            thread_client = Thread(target=client_activate, args=(button,))
            thread_client.start()
            button.configure(text="Stop")
            status_bar.configure(text="Connected to the server: "+ip)
        else: # if the server is offline or the ip is incorrect it will show an error message
            tkMessageBox.showerror("Logging error", "The server is not up currently.\n
Please try again later")
            status_bar.configure(text="Failed to log-in to the server.")
    elif button["text"] == "Stop": # stops the client if the button is Stop
        share_client.stop_client()
        button.configure(text="Go")
        status_bar.configure(text="Client stopped connection")

```

הפעולה start_client היא הפעולה שבאמצעותה המשתמש רואה את המסך של משתמש אחר. ראשית הפעולה מגדירה את global share_client זאת על מנת שהפעולה תוכל לבצע שינויים בעצם הזה.

כעת הפעולה מבצעת 3 בדיקות שונות.

בדיקה ראשונה שהפעולה מבצעת היא בדיקה האם קיים ערך כלשהו בתיבת הטקסט ששלחו ושהכפתור הוא Go.

אם לא קיים ערך הפעולה תקפיץ הודעת שגיאה ותשנה את ערכו של הסטטוס בר ל Ip address was not given. והיא תחזור לממשק הראשי.

במידה ואם הוכנס ip הפעולה עוברת לבדיקה השנייה שבודקת האם הטקסט על הכפתור הוא Go.

במידה ואם כן הפעולה יוצרת לקוח שכתובת ip שלו היא מה שהוכנס לתיבת הטקסט. לאחר מכאן היא משנה את הסטטוס בר ל Connecting to the server... כעת היא מבצעת בדיקה האם הסרבר קיים ורץ. אם לא היא מקפיצה הודעת שגיאה ומשנה את הסטטוס בר ל Failed to log-in to the server.

במידה ואם השרת כן עובד והכתובת IP נכונה הפעולה יוצרת thread שמתחבר שמפעיל את הפונקציה client_activate ומשנה את הכפתור ל Go Stop ואת הסטטוס בר ל Connected to the server: וכתובת IP של השרת.

במידה ואם התנאי של האם הכפתור הוא Go יוצא False זה מגיע ל else שבודק האם הכפתור הוא Stop. במידה ואם כן זה מפסיק את ריצת הלקוח משנה את הכפתור בחזרה ל Go ורושם בסטטוס בר Client stopped connection.

כעת אסביר על הפעולה client_activate(): 454545

```
def client_activate(button):
```

```
    """ Activates the screen sharing """
```

```
    global share_client
```

```
    share_client.start_client() # activate the server
```

```
    button.configure(text="Go") # if the client stopped it will turn back to Go.
```

```
    status_bar.configure(text="Client stopped connection")
```

הפעולה start_client מתחילה את הלקוח. במידה ואם הלקוח סיים את הפעילות שלו על ידי לחיצה על הכפתור איקס שבר toolbar אז הפונקציה מחזירה את הכפתור להיות Go ואת הסטטוס בר ל Client stopped connection והפעולה מפסיקה את פעולתה.

כעת אסביר על הכפתור start ועל הפעולה שהוא מפעיל.

```
    button_start = Button(root, text='Start', padx=47, borderwidth=5, fg='white',
font='Calibri 10',
```

```
        bg='sky blue', command=lambda: start_server(button_start, button_go))
```

```
    # The server button
```

```
    button_start.grid(row=4, column=1, columnspan=2)
```

הכפתור start הוא כפתור המפעיל את הפונקציה start_server שלפי השם מפעילה את השרת משמע שמשתמש אחר יוכל לראות את המסך הנוכחי של המשתמש.

```

def start_server(button, button_go):
    """ Sends a request to activate the server on the current computer """

    if button["text"] == "Start": # if the button is start it will activate the server
        server_thread = Thread(target=server_activate)
        server_thread.start()
        button.configure(text="Stop")
    else:
        if share_client.is_running(): # If the client is up the function will stop it.
            share_client.stop_client()
            button_go.configure(text="Go")
        share_server.stop() # stops the server
        button.configure(text="Start")
        status_bar.configure(text="Server closed ")

הפונקציה start_server היא הפונקציה שמפעילה ומכבה את השרת בהתאם לבקשת המשתמש.
היא מקבלת שתי כפתורים הראשון button הוא הכפתור של השרת והכפתור button_go
לפונקציה שתי תנאים מרכזיים.

תחילה הפונקציה בודקת האם הכפתור הוא start במידה ואם כן הפונקציה יוצרת thread
שמפעיל את הפונקציה server_activate שבאמצעות פונקציה זה מפעילים את השרת.
במידה והכפתור הוא לא start כלומר הוא stop אז הפונקציה בודקת האם קיימים לקוחות
מחוברים. אם כן, היא מכבה אותם ומשנה את הכפתור של הלקוח בחזרה לGo. אם לא היא
מכבה את השרת ומחזירה את הכפתור של השרת לStart ומשנה את הסטטוס בר ל Server
.closed

כעת אסביר על הפונקציה server_activate():

```

```

def server_activate():
    """ Activates the server in the current computer. """

    ip = find_ip()
    try:
        status_bar.configure(text="Server started on ip: "+ip)
        share_server.run()
    except Exception as e:
        status_bar.configure(text="Error occurred: "+str(e))
        status_bar.configure(text="Server is already up on ip: "+ip)

הפונקציה ראשית מוצאת את כתובת Ip של השרת ולאחר מכן מנסה להפעיל את השרת.
במידה והפונקציה לא מצליחה להפעיל את השרת, זה אומר שהשרת כבר פעיל במקום אחר ולכן
הסטטוס בר משתנה ל- הסרבר כבר רץ בכתובת IP – (הכתובת).

```

כעת אסביר על שתי הכפתורים האחרונים שנשארו כפתור הinstruction :

```
button_instruction = Button(root, text='Instruction', padx=15, borderwidth=5,
fg='white', font='Calibri 10',
```

```
bg='sky blue', command=instructions) # The instruction button
```

כפתור זה קורא לפונקציה instruction שתפקידה להציג הודעה קופצת ששם כתוב הסברים כיצד להשתמש בפרויקט.

```
def instructions():
```

```
    """ Creates the instruction box. """
```

```
    tkMessageBox.showinfo("Instructions!", "Hi and welcome to my project Mirror
Screen !\n")
```

```
        "You have 2 options:\n"
```

```
        "First option- show your screen. \n"
```

```
        "In order to show your screen you have to\n "
```

```
        "press start and than tell the other users to \n"
```

```
        "enter the ip shown below.\n"
```

```
        "Second option- see other screens. \n"
```

```
        "In order to see other screens please tell the\n"
```

```
        "other users to start the client and tell you \n"
```

```
        "their ip address. \n"
```

```
        "once they have done that enter that ip address \n"
```

```
        "and press go than you will see the other person\n"
```

```
        "computer.\n"
```

```
        "Thanks for using my program and have a nice day!")
```

הפונקציה מקפיצה הודעת מידע שבה רשום את מה שכתוב בין הגרשיים בשורות למעלה.

ובנוסף קיים הכפתור `exit` שסוגר את כל הדברים שרצים על המחשב.

הכפתור קורא לפעולה `exit_program` שעליה כבר הסברתי למעלה בעמוד 40-41.

חשוב לציין עוד פקודה אחת חשובה שבאמצעותה הממשק רץ והיא :

```
root.mainloop()
```

רפלקציה

לפני הכל, לפני שאתחיל לפרט על תהליך כתיבת הפרויקט ועל הדרך שעברתי עם עצמי עד שהגעתי לספר פרויקט ולפרויקט מוכנים לגמרי, אני רוצה לומר שכתבת ספר הפרויקט הזה והפרויקט עצמו הייתה חוויה שמעוררת כל כך הרבה רגשות ותחושות נגדיים ממש כאילו התחללה סערת רגשות בתוכי.

הכתיבה של ספר הפרויקט הייתה תהליך ארוך ואף מדי פעם קצת מייגע. היו לי כישלונות רבים במהלך הדרך בין אם זה קשיים בהרצת מספר לקוחות רבים (כלומר יכולתי להתחלה להריץ רק לקוח אחד) ובין אם זה קריסות של הממשק כאשר שילבתי את הקוד ואת הממשק ביחד. אך מצד שני לכתיבת הפרויקט היה גם פן מלמד וחיובי שהשפיע עליי. התחושה הזאת שאחרי כל הזמן שמשו נכשל לי פעם אחר פעם, פתאום עובד והצלחתי לסדר הכל הייתה תחושה של כיף וסיפוק שגרם לי אפילו לצעוק בבית יש בשעה שתיים בלילה כשההורים ישנו.

הפרויקט אף לימד אותי להסתדר יותר בעזרת עצמי וללמוד יותר באמצעות האינטרנט. מה שהולך לעזור לי רבות במהלך תקופת שירותי בצבא משום שהתפקיד שאליו התקבלתי בחייל התקשוב הוא תפקיד שידרוש ממני לשבת שעות רבות מול מחשב בסובלנות וללמוד לבד כמה שיותר.

אני אף ירצה להוסיף ולהגיד שבתחילת השנה אני אף שקלתי לפרוש מהמגמה בחששות שאני אולי לא אצליח לעשות את הפרויקט ולא אצליח לסיים אותו אך כעת אחרי שעברתי את כל הדרך הזאת אני ממש גאה בעצמי שהצלחתי לעשות משהו שאני יצרתי אותו בעצמי ועם עזרה קטנה מחבריי שתמכו ועודדו אותי להמשיך.

במהלך כתיבת הפרויקט היו קשיים רבים. רוב הקשיים שאני עברתי היו אם להיות כנה קשיים מנטליים. העובדה שאני לא מצליח משהו או העובדה שתאריך ההגשה הולך ומתקרב ואני לא רואה שאני לקראת סיום הדאגה אותי ממש. אך כמובן שלא ישבתי ללא מעש והתלוננתי על כך שאני לא מספיק. אני ישבתי ימים רצופים שעות על גבי שעות עד שסיימתי כל מה שתכננתי לסיים באותו יום ובכל יום הקצבתי לי לעשות משהו אחר. כך התקדמתי עד שהגעתי למצב שהיה לי ממש עובד וקוד עובד וכל מה שנותר לי היה לשלב אותם ולגרום לזה לעבוד.

אם הייתי מתחיל היום את כתיבת הפרויקט הייתי מנסה מההתחלה להיות קצת יותר מסודר, לעשות חקר יותר יסודי ולתכנן לוח זמנים מפורט. באמצעות כך הייתי מגיע למצב שאפילו חודש לפני תאריך ההגשה היה לי כבר פרויקט מוכן וכל מה שהיה נותר לי זה ספר פרויקט.

יתרה מזאת אני חושב שאם היו מבצעים אתנו את הפרויקט בחלקים במהלך השנה ומבצעים מעקב שבועי זה היה הופך את העבודה שלי ליעילה יותר משום שכך אני הייתי יודע אם אני עושה את מה שאני עושה נכון או שמע אני צריך לשנות דברים מסוימים.

ולסיום אני רוצה להודות לקובי שעזר לי במהלך הפרויקט בדברים שונים ודאג לבצע אתנו מעקב פעם בכמה זמן מה שעודד אותי להתקדם עם הספר ועם הפרויקט עצמו ואני חושב שהפרויקט

הזה עזר לי להתפתח בין אם זה מבחינת אחריות וניהול זמן ובין אם מבחינה חברתית של לעזור לחבריי שמתקשים בדברים מסוימים שאני אולי יכול לעזור.

ביבליוגרפיה

Low-level networking — socket. (אין תאריך). Python Software Foundation 2001-2020
GitHub: <https://docs.python.org/3/library/socket.html> אוחזר מתוך *interface*

Tkinter Course - Create Graphic User. (November 2019 19). freeCodeCamp.org
YouTube: אוחזר מתוך *Interfaces in Python Tutorial*
[t=9069s&https://www.youtube.com/watch?v=YXPYB4XeYLA](https://www.youtube.com/watch?v=YXPYB4XeYLA)

PEP 257 -- Docstring Conventions. (2001 May 29). Guido, R' v &, Goodger, D
/Python (TM): <https://www.python.org/dev/peps/pep-0257> מתוך

How do I handle the window close event in Tkinter. (September 2008 8). Gregory, M
Stack Overflow: אוחזר מתוך
<https://stackoverflow.com/questions/111155/how-do-i-handle-the-window-close-event-in-tkinter>

LucidChart: אוחזר מתוך (אין תאריך). Lucid Software Inc
[https://www.lucidchart.com/pages/landing?utm_source=google
utm_medium=km_C&utm_campaign=en_OL_desktop_branded_x_exact_lucidchart&m=cpc
km_CPC&km_CPC_AdGroupID=67487228588&PC_CampaignId=1551524812
=km_CPC_ExtensionID&km_CPC_MatchType=e&_Keyword=lucid%20chart
&k&](https://www.lucidchart.com/pages/landing?utm_source=google&utm_medium=km_C&utm_campaign=en_OL_desktop_branded_x_exact_lucidchart&m=cpc_km_CPC&km_CPC_AdGroupID=67487228588&PC_CampaignId=1551524812=&km_CPC_ExtensionID&km_CPC_MatchType=e&_Keyword=lucid%20chart&k&)

Nitratine: אוחזר מתוך (May 2018 3). *Python Threading Basics*
[/https://nitratine.net/blog/post/python-threading-basics](https://nitratine.net/blog/post/python-threading-basics)

Python Tutorials: אוחזר מתוך *TkInter message box*. (אין תאריך). pythonspot
[/https://pythonspot.com/tk-message-box](https://pythonspot.com/tk-message-box)

/lamed-oti: <http://www.lamed-oti.com/school/rl> אוחזר מתוך (אין תאריך). jshutzman,

Stack Overflow: אוחזר מתוך (אין תאריך). Stack Exchange Inc
[/https://stackoverflow.com](https://stackoverflow.com)

Creating a Table of Contents in Microsoft Word. (September 2019 9). Students, T' f
YouTube: אוחזר מתוך <https://www.youtube.com/watch?v=0cN-JX6HP7c>

MS Word - Header and Footer for Slides. (January 2018 15). Tutorials Point (India) Ltd
YouTube: אוחזר מתוך <https://www.youtube.com/watch?v=54ugHfkXfvU>

PEP 8 -- Style Guide for Python Code. (July 2001 5). Rossum, G' v &, Warsaw, B', Coghlan, N
Python: אוחזר מתוך <https://legacy.python.org/dev/peps/pep-0008/#constant>