בית חינוך שש שנתי
בן גוריון

# שם הפרויקט: "שליטת בוחן"

**מגיש: רון טקץ'**

**ת"ז: 330801853**

שם בית הספר: תיכון בן גוריון, פתח תקווה

שם מקצוע: תכנון ותכנות מערכות

שם חלופה: הגנת סייבר ומערכות הפעלה

מורה מנחה: לינה שמידט

מועד הגשה: 24.05.2025

# תוכן עניינים

שם פרויקט: "שליטת בוחן"
שם תלמיד: רון טקץ'- 330801853

# מבוא

## ייזום

### תיאור ראשוני של המערכת

בעידן של חוסר יציבות ביטחונית ומצבים כמו מגפות, הצורך בלמידה ובחינות מרחוק הולך וגובר. עם זאת, המעבר הזה יוצר אתגר מרכזי: כיצד ניתן להבטיח פיקוח אפקטיבי על תלמידים גם ללא נוכחות פיזית?

מתוך צורך זה פותחה מערכת "שליטת בוחן" – יישום דסקטופ ייעודי שמאפשר למורים / בוחנים (להלן "מורה") לפקח, לנהל ולסייע לתלמידים / נבחנים (להלן "תלמידים") במהלך בחינות בזמן אמת.

בחרתי בפרויקט זה מתוך הכרות אישית עם האתגרים של תקופת הקורונה, בה חוויתי את הקשיים שנוצרו בבחינות מרחוק – ובעיקר את תופעת ההעתקות, לה אני מתנגד בתוקף. מתוך תחושת אחריות ורצון לקדם סביבה לימודית הוגנת, פיתחתי מערכת שמאפשרת בקרה אפקטיבית.

**רציונל הפרויקט**

- הגברת השליטה והפיקוח של מורים על מהלך הבחינה.
- מניעת העתקות וחוסר מעקב בבחינות מרחוק.
- מתן סיוע אישי לתלמידים ללא צורך בגישה פיזית.

**אתגרים צפויים**

- שמירה על תקשורת יציבה ובזמן אמת בין המורה לתלמידים.
- הצגת מסכים מרובים במקביל תוך שמירה על ביצועים טובים.
- השתלטות חלקה ובטוחה על מחשבים מרוחקים.
- שמירה על פרטיות ואבטחת מידע.

### הגדרת הלקוח

המערכת מיועדת לבתי ספר, ארגוני הדרכה ובחינה, ומוסדות אקדמיים.

המערכת גם יכולה לשמש חברות או ארגונים אשר רוצים לעקוב אחר פעילותיהם של עובדים בעמדות המחשב של החברה.
כמו כן, המערכת עשויה לשמש חברות / משתמשים פרטיים אשר רוצים לתת תמיכה תכנית ללקחותיהם / לקרובי משפחתם.

### הגדרת יעדים / מטרות

- <u>חיזוק השליטה והפיקוח על מהלך הבחינה</u> – יצירת אפשרות לניטור בזמן אמת של מסכי תלמידים, כדי לאפשר למורה מעקב שוטף אחר פעולתם במהלך הבחינה.
- <u>מניעת העתקות בבחינות מרחוק</u> – מתן כלים להתערבות מיידית במקרים חשודים, תוך שמירה על אמינות הבחינה.
- <u>שיפור יכולת התגובה והסיוע האישי לתלמידים</u> – לאפשר למורה להתערב ולהשתלט על עמדת תלמיד באופן ממוקד בעת הצורך, כדי להדריך או לסייע באופן פרטני.
- <u>פישוט תהליך ההפעלה והשימוש במערכת</u> – פיתוח ממשק ניהולי מרכזי, נגיש וידידותי, המאפשר ניהול קל ויעיל של הבחינה מרחוק, ללא צורך בידע טכני מעמיק.
- <u>הנגשה והפעלה מהירה של המערכת</u> – להבטיח התקנה פשוטה והפעלה מיידית כדי לאפשר שימוש שוטף גם במסגרות חינוכיות שאינן טכניות במהותן.

שם פרויקט: "שליטת בוחן"
שם תלמיד: רון טקץ'- 330801853

## בעיות, תועלות וחסכונות

**בעיות**:

- חוסר אפשרות לעקוב אחר מספר רב של תלמידים בבחינות ממוחשבות מרחוק.
- חוסר יכולת התערבות ממוקדת ומהירה על ידי המורה במקרה הצורך.

**תועלות צפויות**:

- חיסכון בכוח אדם (פחות צורך במפקחים פיזיים).
- יכולת התערבות מיידית וממוקדת.
- רמת פיקוח גבוהה ויעילה.
- מניעת העתקות בבחינות ממחושבות.

**שירותי המערכת**:

- שידור מסכי התלמידים למורה.
- שליטה מרחוק על מחשב התלמיד על ידי המורה.
- חסימת העכבר והמקלדת של התלמיד על ידי המורה.
- גירוש התלמיד מהמערכת על ידי המורה.
- ממשק צפייה דינמי מרכזי עבור המורה.
- הצפנת תקשורת רגישה.

## סקירת פתרונות קיימים

קיימות מערכות כגון: **TeamViewer**, **AnyDesk** ופתרונות פיקוח כמו **Safe Exam Browser**.
אולם, לרוב מדובר בכלים כלליים שלא מותאמים ספציפית לסביבת מבחן מרובת תלמידים בו-זמנית, זאת מכיוון שאפשר להתחבר אצלם רק ללקוח אחד. הפתרון שלי מותאם באופן ייעודי לצורך ההתחברות למספר רב של לקוחות.

## סקירת טכנולוגית הפרויקט

המערכת תפותח בסביבת Desktop ותדרוש חיבור רשת פנימית/אינטרנט. מדובר בטכנולוגיות קיימות, אך המימוש הספציפי ידרוש אינטגרציה מאתגרת בין הצגת מסכים חיים מרובים לשליטה מרחוק – מבלי לפגוע בביצועים.

## תיחום הפרויקט

הפרויקט פותח בשפת Python תוך שימוש בספריות סטנדרטיות כמו:

- **socket**- לתקשורת רשת בין שרת ללקוח.
- **threading**- לניהול תהליכים מקבילים ולטיפול במספר חיבורים בו-זמנית.

המערכת מתמקדת בניהול תקשורת מבוססת TCP/IP בין מחשב של מורה (שרת) לבין מחשבים של תלמידים (לקוחות), בסביבת מערכת ההפעלה Windows בלבד.

**תחומים בהם הפרויקט עוסק:**

- ✓ ניהול חיבורי רשת.
- ✓ העברת מידע ברשת.

שם פרויקט: "שליטת בוחן"
שם תלמיד: רון טקץ'- 330801853

- ✓ טיפול באירועי קלט.
- ✓ ממשק משתמש גרפי.
- ✓ אבטחת מידע.

**תחומים שאינם מטופלים בפרויקט:**

- ✗ שמירה של נתוני משתמשים.
- ✗ ניתוח התנהגות או רישום פעולות של תלמידים.
- ✗ תמיכה במכשירים ניידים או במערכות הפעלה שאינן Windows.
- ✗ תמיכה בעבודה עם מספר מסכים באותו מחשב.
- ✗ תמיכה בריבוי שפות- הממשק והשליטה מרחוק באנגלית בלבד.
- ✗ תמיכה בגלגלת עכבר או סימון עם העכבר.
- ✗ תמיכה ברזולוציית מסך שונות (תומך רק ב- 1920x1080).

# פירוט תיאור המערכת (אפיון)

## תיאור מפורט יותר של המערכת

המערכת מורכבת מאפליקציית הפעלה מרוכזת שמאפשרת למשתמש לבחור באיזה תפקיד להפעיל את המערכת. לאחר הבחירה, המשתמש מזין כתובת IP ו-Port (בהתאם לתפקידו), ובמידה והקלט תקין, מופעל הממשק המתאים (שרת או לקוח). במקרה של ניתוק או שגיאה, האפליקציה מסוגלת להפעיל את עצמה מחדש אוטומטית עם הודעת שגיאה רלוונטית.

**ממשק השרת (מורה):**

- מציג מסך מרכזי ובו רשת של תצוגות חיות ממחשבי הלקוחות, אשר מוספים דינמית למסך כאשר הם מתחברים.
- בלחיצה על אחד ממסכי התלמידים, מכניס למצב "מסך מלא", אשר מוסיף 4 כפתורי שליטה למסך- "התחלת שליטה מרחוק", "התחלת חסימה" (חסימת מקלדת ועכבר של הלקוח), "הסרה" (ניתוק) ו-"יציאה ממסך מלא" (חזרה למצב צפייה).
- מציג על המסך מידע על ההתחברות לשרת (IP ו-Port) ואת מספר התלמידים אשר מחוברים למערכת.

**ממשק הלקוח (תלמיד):**

- צד התלמיד מינימלי, מבלי שיהיה צורך שהתלמיד יילמד איך להשתמש במערכת.
- מציג מסך קטן ועליו כתוב "מחובר לשרת, שיתוף מסך פועל".
- המחשב מתחיל לשדר את המסך לשרת באופן שוטף.
- הלקוח מאזין לפקודות מהשרת, ומבצע אותן בהתאם.

## פירוט היכולות שהיא תעניק לכל סוג משתמש במערכת

**יכולות אפליקציית הפעלה:**

- בחירת תפקיד מערכת.
- הפעלה מחדש עצמית והצגת סיבה.

**יכולות צד שרת:**

- שידורי מסך חיים.
- תקשורת מוצפנת.

- מסך מרכזי עם אוסף מסכי תצוגה מקדימה.
- מסך שליטה (״מסך מלא״).
- התחלת והפסקת שליטה מרחוק.
- התחלת והפסקת חסימת לקוח.
- ניתוק לקוח.

**יכולות הלקוח:**

- מסך מידע מינימלי.
- שידור מסך.
- ביצוע פעולות שליטה מרחוק וחסימת לקוח.
- התנתקות.
- תקשורת מוצפנת.

## בדיקות מתוכננות (קופסה שחורה)

| שם הבדיקה: | מטרת הבדיקה: | כיצד ייבדק: |
|---|---|---|
| אי קריסת לקוח ושרת | לבדוק שהלקוח והשרת אינם קורס מסיבה כלשהי. | אני אפעיל שרת, אתחבר אליו עם לקוח, ואשתמש בכל היכולות המוצאות על ידי השרת, כמו גם סגירה פתאומית של השרת או הלקוח. |
| בדיקת חסימת לקוח | לבדוק ש-״חסימת לקוח״ פועלת כפי שמצופה. | אני אפעיל שרת, אתחבר אליו עם לקוח, אפעיל ״חסימת לקוח״ דרך השרת, אבדוק שכל הכפתורים והמקשים אצל הלקוח חסומים על ידי לחיצתם במחשב הלקוח. |
| בדיקת שליטה מרחוק | לבדוק שה-״שליטה מרחוק״ עובדת כפי שמצופה, גם אם ״חסימת לקוח״ פועלת באותו הרגע. | אני אפעיל שרת, אתחבר אליו עם לקוח, אפעיל שליטה מרחוק, אנסה לפתוח קובץ ולכתוב בו, ואז אפעיל ״חסימת לקוח״ ואנסה לפתוח קובץ נוסף ולכתוב בו, ואבדוק שכל הכפתורים והמקשים אצל הלקוח חסומים על ידי לחיצתם במחשב הלקוח. |
| בדיקת התנתקות מסודרת | לבדוק שכאשר לקוח מתנתק מהשרת, השרת מטפל בניקוי המשאבים הקשורים ללקוח, והלקוח מתנתק בצורה מסודרת. | אני אוסיף בקוד הדפסות סטטוס ניתוק לקוח, אפעיל שרת, אתחבר אליו עם לקוח, אנתק את הלקוח דרך השרת, ואז אצפה בהודעות סטטוס המודפסות ב-״cmd״, כדי לראות שכל המשאבים נמחקו ונוקו מהלקוח. |
| בדיקת הפעלה מחודשת של אפליקציית הפעלה | לבדוק שכאשר לקוח או שרת מתנתק או נסגר על ידי המשתמש, אפליקציית הפעלה מופעלת מחדש, תוך ניקוי משאבים ישנים. | אני אוסיף בקוד הדפסות סטטוס סגירת לקוח ושרת, אני אפעיל שרת, אתחבר אליו עם לקוח, אנתק את הלקוח דרך השרת, אתחבר עם הלקוח שוב פעם, אסגור את הלקוח, אתחבר עם הלקוח שוב פעם, ואז אסגור את השרת. בכל הפעמים צריכה להיפתח אפליקציית ההפעלה, ואצפה בהודעות סטטוס סגירה המודפסות ב-״cmd״. |

שם פרויקט: "שליטת בוחן"
שם תלמיד: רון טקץ'- 330801853

## תכנון לו"ז פיתוח

| תכנון | זמן התחלה מתוכנן | זמן סיום מתוכנן | זמן התחלה בפועל | זמן סיום בפועל | הערות |
|---|---|---|---|---|---|
| יצירת ממשק גרפי לאפליקציית הפעלה | 23.2.25 | 25.2.25 | 23.2.25 | 25.2.25 | אין |
| יצירת מודול הצפנה | 28.2.25 | 28.2.25 | 28.2.25 | 28.2.25 | אין |
| יצירת שרת מבוסס תהליכונים ולקוח עם החלפת מפתחות הצפנה | 5.3.25 | 5.3.25 | 5.3.25 | 5.3.25 | אין |
| יצירת מודול צילום תמונת מסך | 6.3.25 | 6.3.25 | 6.3.25 | 6.3.25 | אין |
| יצירת תהליכון אצל הלקוח ששולח תמונות מסך דחוסות ומוצפנות | 10.3.25 | 11.3.25 | 10.3.25 | 11.3.25 | אין |
| יצירת תהליכון אצל השרת שמקבל תמונות מסך דחוסות ומוצפנות ומפענח ופורס אותם | 12.3.25 | 13.3.25 | 12.3.25 | 13.3.25 | אין |
| יצירת ממשק גרפי לשרת- יצירת מסך מרכזי, עדכון מסכים מקדימים והוספת מסכים מקדימים חדשים | 16.3.25 | 18.3.25 | 16.3.25 | 23.3.25 | לקח יותר זמן מהמצופה |
| יצירת סמל מידע | 19.3.25 | 19.3.25 | 19.3.25 | 19.3.25 | אין |
| יצירת פעולת סידור בממשק הגרפי של השרת | 22.3.25 | 23.2.25 | 23.3.25 | 24.3.25 | אין |
| יצירת תהליכון אצל השרת שמעדכן תמונות מתהליכון קבלת תמונות בממשק הגרפי | 24.3.25 | 24.3.25 | 24.3.25 | 24.3.25 | אין |
| הוספת אופציה לכניסה ל- "מסך מלא" | 28.3.25 | 28.3.25 | 28.3.25 | 28.3.25 | אין |
| הוספת כפתורי פעולה כשב- "מסך מלא" | 28.3.25 | 28.3.25 | 28.3.25 | 28.3.25 | אין |
| יצירת תהליכון אצל השרת שמצפין ושולח פקודות | 29.3.25 | 29.3.25 | 30.3.25 | 30.3.25 | אין |
| יצירת מודול שליטה מרחוק | 4.4.25 | 7.4.25 | 4.4.25 | 7.4.25 | אין |
| יצירת מודול חסימת לקוח | 4.4.25 | 7.4.25 | 4.4.25 | 7.4.25 | אין |
| יצירת פעולות ניתוק וניקוי משאבים | 25.4.25 | 26.4.25 | 25.4.25 | 26.4.25 | אין |
| הוספת תמיכה בקבלת, פענוח וביצוע פקודות אצל הלקוח | 27.4.25 | 30.4.25 | 27.4.25 | 30.4.25 | אין |
| יצירת רמזים צצים לכפתורי הפעולה וסמל המידע | 3.5.25 | 4.5.25 | 3.5.25 | 4.5.25 | אין |
| הוספת מגבלת תמונות בשנייה מלקוח ומחיקת תמונות ישנות בשרת | 5.5.25 | 5.5.25 | 5.5.25 | 5.5.25 | אין |
| הוספת פקודת השהיית שליחת תמונות | 7.5.25 | 7.5.25 | 7.5.25 | 7.5.25 | אין |
| הוספת פקודות שינוי גודל של תמונה | 8.5.25 | 10.5.25 | 8.5.25 | 10.5.25 | אין |
| ייעול קוד, הוספות קטנות, בדיקות ותיקונים אחרונים | 11.5.25 | 19.5.25 | 11.5.25 | 19.5.25 | אין |

## ניהול סיכונים

| הסיכון | הסבר הסיכון | דרכי התמודדות מתוכננים | דרכי התמודדות בפועל |
|---|---|---|---|
| העמסת יתר על השרת | לאור העובדה ששרת עושי להיות מחובר לעשרות לקוחות, ייתכן שייוצר עומס רב על השרת ועיכוב רב בהצגת שידורי המסך של הלקוחות. | • שימוש בחותמת זמן (כדי להתעלם מתמונות ישנות).<br>• דחיסת התמונות.<br>• שימוש בהצפנה סימטרית עבור שידורי המסך.<br>• הגבלת מספר תמונות שנשלח מלקוח בשנייה.<br>• הגבלת מספר תמונות בשנייה שעוברים עיבוד מכל לקוח בשנייה בשרת.<br>• בקשת תמונות בגודל נדרש בשרת מהלקוחות (כדי לחסוך בשינוי גודל תמונות). | • שימוש בחותמת זמן (כדי להתעלם מתמונות ישנות).<br>• דחיסת התמונות.<br>• שימוש בהצפנה סימטרית עבור שידורי המסך.<br>• הגבלת מספר תמונות שנשלח מלקוח בשנייה.<br>• הגבלת מספר תמונות בשנייה שעוברים עיבוד מכל לקוח בשנייה בשרת.<br>• בקשת תמונות בגודל נדרש בשרת מהלקוחות (כדי לחסוך בשינוי גודל תמונות). |
| תקיפות אדם בתווך | אדם יכול להתחזות כשרת / לקוח, והלקוח / השרת ישלחו לאדם זה את תמונות המסך שלו / לחיצות המקשים שלו (עלול להיות בהם מידע רגיש). האדם יעביר את המידע הזה הלאה כדי שלא יידעו שיש מישהו שמאזין. | • שימוש בהצפנה עבור המידע אשר עובר בתקשורת.<br>• שימוש באמצעי זיהוי כלשהו. | • שימוש בהצפנה עבור המידע אשר עובר בתקשורת. |

# תיאור תחום הידע – פרק מילולי

## פירוט יכולות אפליקציית הפעלה

❖ **בחירת תפקיד מערכת:**

- מהות: מסך מרוכז המאפשר למשתמש לבחור ולהפעיל את המערכת בתפקיד מסוים.
- אוסף פעולות נדרשות:
  - החלפת מסך לאחר בחירת תפקיד.
  - קבלת קלט IP ו-Port.
  - בדיקת תקינות קלט.
  - בדיקת יכולת התחברות לשרת.
  - בדיקת יכולת יצירת התחלת שרת.
  - הפעלת לקוח / שרת.
- אובייקטים נחוצים:
  - אובייקטים- אובייקט מסך, אובייקט השרת ואובייקט הלקוח.

❖ **הפעלה מחדש עצמית והצגת סיבה:**

- מהות: כאשר לקוח מתנתק / שרת נסגר, מפעיל מחדש את התוכנה ומציג את סיבת ההפעלה מחדש.
- אוסף פעולות נדרשות:
  - סגירת השרת / הלקוח.
  - הפעלה מחדש.
- אובייקטים נחוצים:
  - אובייקטים- אובייקט מסך, אובייקט השרת ואובייקט הלקוח.

## פירוט יכולות צד שרת

❖ **שידורי מסך חיים:**

- מהות: קבלת תמונות מסך מהתלמידים בזמן אמת ועדכונן בממשק הגרפי של השרת בצורה יעילה ומהירה.
- פעולות נדרשות:
  - קבלת חיבור חדש.
  - שליחת גודל תמונה דרוש (לייעל העברת תמונות).
  - פריסת תמונות דחוסות.
  - מנגנון "חניקת" לקוחות ("throttling").
  - מנגנון התעלמות מתמונות ישנות.
  - עדכון מסכים בממשק הגרפי.
- אובייקטים נחוצים:
  - אובייקטים- אובייקט "FrameConsumer", אובייקט המסך ואובייקט השרת.

❖ **תקשורת מוצפנת:**

- מהות: העברת מידע רגיש בדרך בטוחה.
- פעולות נדרשות:
  - יצירת מפתח אסימטרי וסימטרי.
  - החלפת מפתחות בין השרת והלקוח (מפתח אסימטרי מצפין את הסימטרי).
  - הצפנת מידע נשלח על ידי המפתח הסימטרי.
  - פענוח מידע מתקבל על ידי המפתח הסימטרי.

- אובייקטים נחוצים:
  - אובייקטים- אובייקט השרת.

❖ **מסך מרכזי עם רשת מסכי תצוגה מקדימה:**

- מהות: הצגת שידורי מסך החיים ברשת שמוסיפה לקוחות חדשים דינמית.
- פעולות נדרשות:
  - סידור מסכי הלקוחות כך שיהיו מרוכזים באמצע המסך.
  - התאמת גודל התמונה אם אינה בגודל המתאים.
  - הוספת מסכי לקוחות חדשים.
  - התראה כאשר אין לקוחות מחוברים.
  - יציאה מ-"מסך מלא" חזרה למסך המרכזי.
- אובייקטים נחוצים:
  - אובייקטים- אובייקט המסך ואובייקט השרת.

❖ **מסך שליטה ("מסך מלא"):**

- מהות: הצגת שידור מסך חי לקוח ספציפי.
- פעולות נדרשות:
  - הוספת כפתורי פעולה.
  - הצגת מצב "חסימה" עבור הלקוח.
  - מנגנון עצירת קבלת מידע משאר הלקוחות (יעילות).
  - הגדלת שידור מסך התלמיד ל"מסך מלא" (לקחת כמה שיותר מקום בממשק הגרפי).
- אובייקטים נחוצים:
  - אובייקטים- אובייקט המסך ואובייקט "CommandConsumer".

❖ **התחלת והפסקת שליטה מרחוק:**

- מהות: השתלטות מרחוק על מחשב הלקוח ב"מסך מלא".
- פעולות נדרשות:
  - התחלת האזנה להקשות כפתורי המקלדת ושליחתן ללקוח.
  - התחלת האזנה להקשות כפתורי העכבר ושליחתן ללקוח.
  - הפסקת האזנה להקשות כפתורי המקלדת.
  - הפסקת האזנה להקשות כפתורי העכבר.
  - שינוי מצב כפתור "שליטה מרחוק".
- אובייקטים נחוצים:
  - אובייקטים- אובייקט "CommandConsumer" ואובייקט המסך.

❖ **התחלת והפסקת חסימת לקוח:**

- מהות: חסימת המקלדת והעכבר של הלקוח ב"מסך מלא".
- פעולות נדרשות:
  - שליחת פקודת התחלת חסימה.
  - שליחת פקודת הפסקת חסימה.
  - שמירת מצב חסימה.
  - שינוי מצב כפתור "חסימה".
- אובייקטים נחוצים:
  - אובייקטים- אובייקט "CommandConsumer" ואובייקט המסך.

❖ **ניתוק לקוח:**

- מהות: ניתוק לקוח ב"מסך מלא".

- פעולות נדרשות:
  - מנגנון ניקוי משאבים שמטפלים בלקוח ומידע שקשור ללקוח (כולל מחיקה מהממשק הגרפי).
  - יציאה מ-"מסך מלא".
- אובייקטים נחוצים:
  - אובייקטים- אובייקט "CommandConsumer", אובייקט המסך ואובייקט שרת.

## פירוט יכולות צד לקוח

❖ **מסך מידע בסיסי:**

- מהות: הצגת מסך מינימלי אשר מאשר שעדיין מחובר לשרת.
- פעולות נדרשות:
  - יצירת המסך.
- אובייקטים נחוצים:
  - אובייקטים- אין.

❖ **שידור מסך:**

- מהות: שליחת צילום מסך כל כמה זמן, בגודל המבוקש על ידי השרת.
- פעולות נדרשות:
  - צילום מסך ושינוי גודלה לגודל המבוקש.
  - דחיסת תמונות.
  - שליחת הצילום לשרת.
  - מנגנון הגבלת מספר תמונות בשנייה (fps).
  - מנגנון הפסקת שליחת תמונות.
- אובייקטים נחוצים:
  - אובייקטים- אובייקט הלקוח.

❖ **ביצוע פעולות שליטה מרחוק וחסימת לקוח:**

- מהות: שליחת צילום מסך כל כמה זמן, בגודל המבוקש על ידי השרת.
- פעולות נדרשות:
  - לחיצה על מקשי המקלדת שהתקבלו.
  - לחיצה על מקשי העכבר שהתקבלו במיקום הנכון.
  - התחלת התעלמות מתנועות העכבר ומלחיצות ממנו ומהמקלדת של הלקוח.
  - הפסקת התעלמות מתנועות העכבר ומלחיצות ממנו ומהמקלדת של הלקוח.
- אובייקטים נחוצים:
  - אובייקטים- אובייקט "UserBlocker", אובייקט "InputController" ואובייקט הלקוח.

❖ **ניתוק לקוח:**

- מהות: ניתוק לקוח מהשרת אחרי קבלת פקודת התנתקות.
- פעולות נדרשות:
  - מנגנון ניקוי משאבים ומידע שקשורים לשרת.
  - סגירת הממשק הגרפי.
  - הפסקת חסימת לקוח.
- אובייקטים נחוצים:
  - אובייקטים- אובייקט "UserBlocker", אובייקט המסך ואובייקט הלקוח.

❖ **תקשורת מוצפנת:**

- מהות: העברת מידע רגיש בדרך בטוחה.
- פעולות נדרשות:
  - יצירת מפתח אסימטרי.
  - החלפת מפתחות בין השרת והלקוח.
  - הצפנת מידע נשלח על ידי המפתח הסימטרי.
  - פענוח מידע מתקבל על ידי המפתח הסימטרי.
- אובייקטים נחוצים:
  - אובייקטים- אובייקט הלקוח.

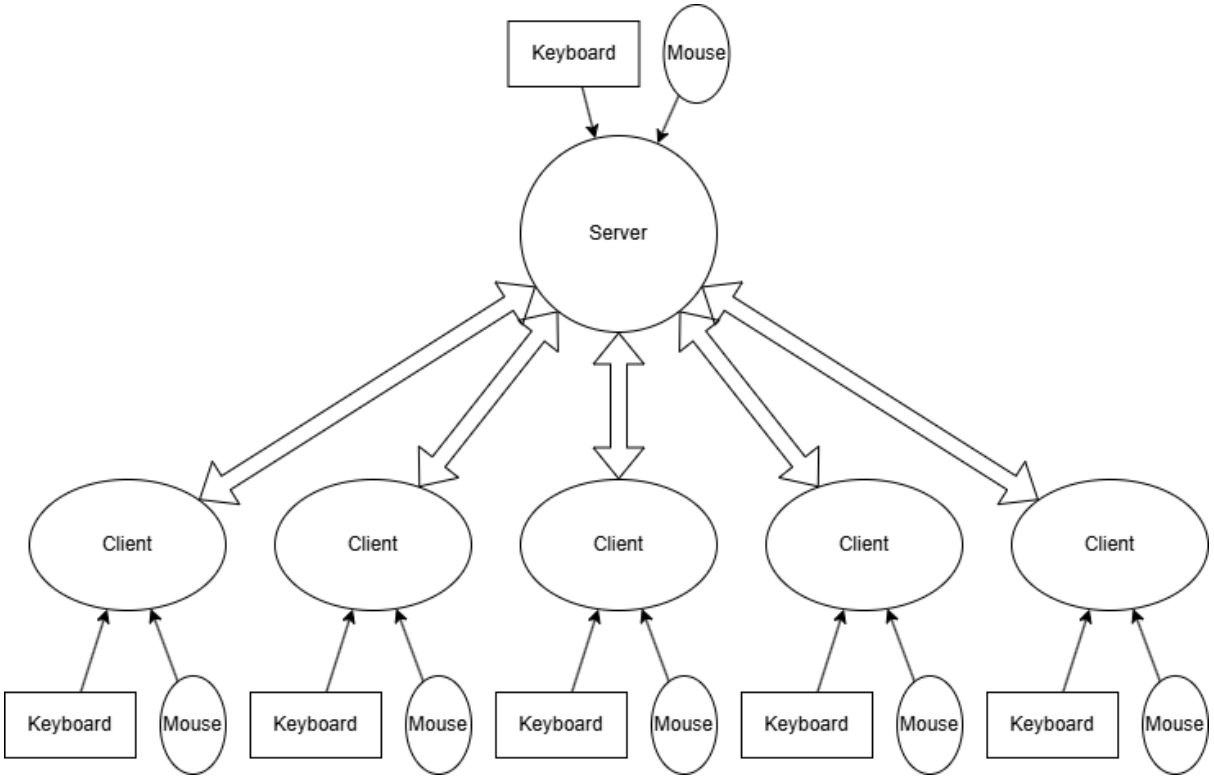
שם פרויקט: "שליטת בוחן"
שם תלמיד: רון טקץ'- 330801853

# ארכיטקטורה של הפרויקט

## תיאור הארכיטקטורה של המערכת המוצעת

החומרה היא מחשבים, אליהם מחוברים מקלדת ועכבר.
מספר מחשבי לקוח מחוברים למחשב שרת.

שרטוט המציג את התיאור:

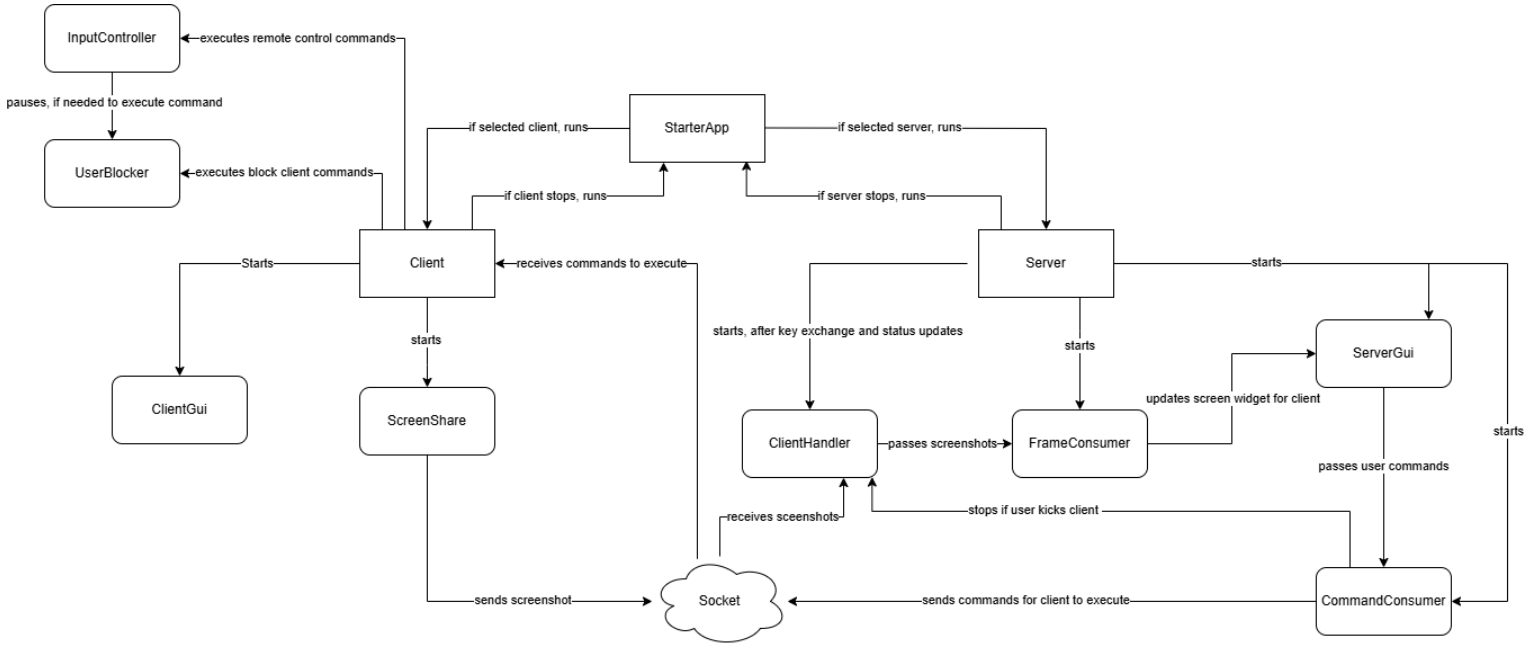



## תיאור הטכנולוגיה הרלוונטית

הטכנולוגיות הרלוונטיות הן:

- שימוש במחשבים- לשם הרצת קוד הלקוח וקוד השרת, ואשר מחוברים לעכבר ומקלדת.
- שימוש בשפת התכנות- "Python".
- שימוש במערכת ההפעלה - "Windows", תוך שימוש ב-"cmd" של מערכת הפעלה זאת.
- שימוש בסביבת העבודה - "PyCharm".

## תיאור זרימת מידע במערכת




## תיאור האלגוריתמים המרכזיים בפרויקט

❖ **אלגוריתם לקיחת ושליחת תמונות:**

- <u>ניסוח וניתוח של הבעיה האלגוריתמית:</u>
במסגרת הפרויקט, בצד הלקוח, נדרש לפתח מנגנון שיתוף מסך המאפשר ללקוח לשלוח תמונות מסך של שולחן העבודה שלו לשרת באופן שוטף ורציף, תוך עמידה בדרישות הבאות:
  - עיבוד והעברת תמונה בקצב סבירה.
  - דחיסת מידע להפחתת גודל התמונה לפני השליחה.
  - אבטחת המידע באמצעות הצפנה.
  - תמיכה בהשהייה ובחידוש השליחה לפי צורך.
  - תמיכה בשינוי דינמי של רזולוציית התמונה.
- <u>תיאור אלגוריתמים קיימים לפתרון הבעיה:</u>
ניתן לחלק את הפתרון למספר שלבים, שלכל אחד מהם קיימים פתרונות מקובלים:
  1. צילום מסך ושינוי רזולוציה
     - ספריות נפוצות: PIL.ImageGrab, mss, pyautogui.
     - שיקולים: מהירות לכידת המסך, תמיכה בפלטפורמות שונות, איכות.
  2. דחיסה על ידי פורמט תמונה
     - שימוש באלגוריתמים כגון: JPEG, PNG, ועוד.
     - JPEG מאפשר דחיסת תמונה באובדן חלקי תוך שמירה על איכות סבירה.
  3. דחיסה נוספת של המידע
     - שימוש ב-zlib לדחיסת נתוני התמונה ללא איבוד מידע.
     - אלטרנטיבות: zipfile, bz2, ועוד.
  4. הצפנת הנתונים
     - שימוש ב- AES (Advanced Encryption Standard) להצפנה סימטרית.
     - AES ו-SRTP הן דרכי הצפנות סטנדרטיים במערכות שיתוף מידע חי.
  5. שליחת המידע

- שליחה בפרוטוקול מבוסס TCP (יותר יציב ובטוח, יותר איטי) או UDP (פחות יציב ובטוח, יותר מהיר).
- שליחת גודל מידע, על מנת בדיקת קבלת תמונה מלאה.

- <u>הפנייה למקורות רלוונטיים:</u>
  - Python Imaging Library (PIL): https://pillow.readthedocs.io/
  - AES Encryption in Python (PyCryptodome): https://pycryptodome.readthedocs.io/
  - Zlib Compression: https://docs.python.org/3/library/zlib.html
  - Network Programming with Python Sockets: https://docs.python.org/3/library/socket.html

- <u>סקירת הפתרון הנבחר:</u>
  - <u>תיאור הפתרון:</u>

האלגוריתם בנוי כ-Thread עצמאי אשר מבצע את הפעולות הבאות במחזוריות:

1. צילום תמונת מסך עדכנית של המסך באמצעות PIL.ImageGrab.
2. שינוי גודל התמונה לרזולוציה מבוקשת (על ידי PIL).
3. דחיסת התמונות
   - שמירה בפורמט JPEG.
   - דחיסה נוספת עם zlib.
4. הצפנת מידע
   - שימוש במפתח AES ייחודי.
   - שימוש בוקטור אתחול אקראי.
5. שליחה המידע לשרת
   - שימוש בפרוטוקול מבוסס TCP
   - שימוש בפורמט מסודר: גודל תמונה, וקטור אתחול, מידע מוצפן.

האלגוריתם תומך בהשהיית שליחת תמונות, עצירה ושינוי רזולוציה בזמן ריצה.

  - <u>נימוק הבחירה:</u>
    - ✓ ***יעילות***: השימוש ב- JPEG ודחיסה נוספת מאפשר שידור מהיר תוך חיסכון ברוחב פס.
    - ✓ ***פשטות***: עבודה עם מודולים סטנדרטיים של Python.
    - ✓ ***אבטחה***: הצפנה סימטרית עם וקטור אתחול מבטיחה סודיות של התמונה.
    - ✓ ***יציבות ושליטה***: שימוש בפרוטוקול מבוסס TCP לשם תקשורת יציבה ובטוחה, ושימוש בשדה גודל מידע בשביל קבלה מדויקת של מידע.
    - ✓ ***שליטה דינמית***: אפשרות לעדכן את גודל התמונה תוך כדי ריצה תורמת לגמישות המערכת ומאפשרת הורדה מהעומס על השרת.
  - <u>שלילת פתרונות אלטרנטיביים:</u>
    - ✗ ***שימוש ב-PNG במקום JPEG***: PNG הוא פורמט ללא איבוד מידע, אך אינו יעיל לדחיסה של תמונות מסך בקצב מהיר, מה שמעלה את העומס על הרשת ועשוי ליצור עיכוב.
    - ✗ ***העברת תמונות ללא דחיסה***: גורמת לעומס רשת חריג, אינה ישימה בסביבה אמיתית.
    - ✗ ***שידור וידאו במקום תמונות***: פתרון אפשרי, אך דורש ספריות מתקדמות, מורכב יותר ודורש יותר כוח עיבוד בשני הצדדים.

- ***שימוש בפרוטוקול UDP*** : פתרון טוב עבור מהירות שליחת תמונות, אך הרבה יותר מורכב בליישום (סידור חבילות נתונים, קבלת חבילות ללא שחיתה, ועוד).

❖ **אלגוריתם קבלת ועדכון תמונות:**

- <u>ניסוח וניתוח של הבעיה האלגוריתמית:</u>
  במסגרת הפרויקט, בצד השרת, יש לפתח אלגרוריתם לקליטת זרמי תמונות מוצפנות ממספר לקוחות במקביל, פענוח ודחיסה חוזרת של התמונות, ועדכון בזמן אמת של ממשק המשתמש הגרפי, תוך עמידה בדרישות הבאות:
  - תמיכה בריבוי חיבורים (לקוחות) בו־זמנית.
  - שמירה על הביצועים.
  - מניעת עיכובים.
  - שמירה על תזמון קבוע לפי FPS מוגדר.
  - התאמת גודל תמונות להצגה.
  - הפרדת עדכון GUI מהלוגיקה ברקע.
- <u>תיאור אלגוריתמים קיימים לפתרון הבעיה:</u>
  ניהול חיבורים במקביל (Multithreading)
  - שימוש בתהליכונים לכל חיבור, אשר מטפלים בלקוח שמקושר להם.
  - אלטרנטיבה: שימוש ב-asyncio.

  האלגוריתם בנוי עבור Thread טיפול, אשר מבצע את הפעולות הבאות במחזוריות:
  1. הגבלת מספר עדכונים בשנייה (Frame Rate Control)
     - שמירה על מינימום מרווח בין עדכוני תמונות באמצעות ספריית time.
     - מבטיח שימוש יציב ברוחב הפס והמעבד.
  2. פענוח מידע (מתבסס על הבחירה באלגוריתם הקודם)
     - קבלת גודל תמונה, וקטור אתחול, מידע מוצפן.
     - פענוח עם AES ופריסה על ידי zlib.
  3. ניהול תור תמונות (Producer–Consumer)
     - תהליכון הטיפול מעביר דרך תור תמונות לעדכון את התמונה מהלקוח, ביחד עם חותמת זמן.
     - שימוש בתהליכון חדש, שמבצע את עדכון התמונות בתור תמונות לעדכון.
     - התהליכון החדש מתעלם מתמונות ישנות.
  4. עדכון GUI מבוקר
     - שימוש ב-tkinter לעדכון מסכים.
     - הבטחת עידכון רק עבור לקוחות רלוונטיים.
     - הוספת מסך עבור לקוח חדש.
     - בדיקת גודל התמונה – והתאמתם לשימוש בממשק הגרפי.
     - אלטרנטיבות: PyQt, PyGame, ועוד.
- <u>הפנייה למקורות רלוונטיים:</u>
  - Python threading module:
    https://docs.python.org/3/library/threading.html
  - AES encryption in Python (PyCryptodome):
    https://pycryptodome.readthedocs.io/
  - GUI with Tkinter:
    https://docs.python.org/3/library/tkinter.html
  - Producer–Consumer Pattern:
    https://en.wikipedia.org/wiki/Producer–consumer_problem
- <u>סקירת הפתרון הנבחר:</u>

- <u>תיאור הפתרון:</u>
המערכת מורכבת ממספר רכיבים תפקודיים:
1. יצירת תהליכון קריאת תור תמונות מסך לעדכון.
   - מוציא מהתור את התמונות.
   - בודק את "טריות" התמונה (חצי שנייה לכל היותר), אם "טרי ממשיך, אחרת מתעלם.
   - שולח את התמונה לעדכון במשק המשתמש, בהתאם לרלוטיות של התמונה במשק הגרפי.
   - אם התמונה לא רלוונטית, מתעלם ממנה.
2. יצירת תהליכון טיפול עבור כל לקוח:
   - קורא רציף מהחיבור (socket) עם הלקוח.
   - מפענח ופורס את התמונה.
   - מוסיף את התמונה לתור תמונות מסך לעדכון, עם חותמת זמן.
3. פונקציות עדכון תמונה במשק הגרפי:
   - מטפלות בעדכון מסכי הלקוחות.
   - אם אין מסך עבור הלקוח, יוצר מסך חדש ומסדר את המסך.
   - עושות Resize בהתאם לרזולוציה הרצויה במידת הצורך.
   - מעבירות פקודות לשליחת עדכון גודל ללקוח במידת הצורך.
- <u>נימוק הבחירה:</u>
  - ✓ ***ביצועים טובים במקביליות***: שימוש בתהליכונים מייעל את העבודה עם מספר חיבורים.
  - ✓ ***שליטה בזמן***: כל תמונה נבדק אם עומדת בדרישות זמן, מה שמונע עיכוב בשידור במשק הגרפי.
  - ✓ ***אבטחת המידע***: שימוש ב-AES עם וקטור אתחול מבטיח שכל תמונה מוצפנת בצורה בטוחה.
  - ✓ ***מניעת עומס במשק הגרפי***: הפרדה מלאה בין עיבוד נתונים לתצוגה.
- <u>שלילת פתרונות אלטרנטיביים:</u>
  - ✗ ***שימוש ב-asyncio***:
    עשוי להניב ביצועים טובים יותר במספר חיבורים גדול מאוד, אך קשה יותר למימוש.
  - ✗ ***שימוש בעדכון ישיר של הממשק הגרפי מתוך תהליכון הטיפול***:
    עלול לגרום לקריסות, בעיות תזמון, עומס יתר על הממשק ועומס על זמן הטיפול בתמונות חדשות.

❖ **אלגוריתם קבלת וביצוע פקודות שליטה מרחוק וחסימת לקוח:**

- <u>ניסוח וניתוח של הבעיה האלגוריתמית:</u>
במסגרת הפרויקט, יש צורך באלגוריתם שיאפשר למשתמש השרת לשלוט במחשב הלקוח ולחסום את פעילות העכבר והמקלדת של הלקוח, תוך עמידה בדרישות הבאות:
  - שימוש בממשק הגרפי לקבלת קלטי עכבר ומקלדת.
  - העברת פקודות מרחוק בצורה אמינה.
  - תמיכה בשילובי תווים (פעולות מיוחדות, כגון: "העתק" ו-"הדבק").
  - תמיכה במקשים מיוחדים במקלדת.
  - לאפשר פקודות שליטה מרחוק אצל הלקוח, גם אם חסימת לקוח פועלת.
  - שמירה על אבטחת מידע.
- <u>תיאור אלגוריתמים קיימים לפתרון הבעיה:</u>
  1. איתור קלט בממשק גרפי (מתבסס על הבחירה באלגוריתם הקודם)
     - שימוש באירועי tkinter לזיהוי לחיצות עכבר- <Button> והקשות מקשים- <KeyPress>, <KeyRelease>.

- מיפוי יחסי של מיקום לחיצת העכבר על מסך הלקוח המוגדל למיקום המדויק.

2. מניעת ריבוי פקודות מהירות (Debounce)
   - התעלמות מלחיצות מקשים חוזרות לתקופת זמן מסוימת.
   - פתרון נפוץ למניעת שליחת קלט כפול שלא במתכוון.
3. שליחת פקודות מרחוק מהשרת
   - הוספת פקודה לתור הפקודות (אשר בו מטפל תהליכון שליחת פקודות) תוך שימוש פורמט אחיד וברור.
   - פקודות נשלחות לפי כתובת הלקוח שעליו שולטים.
4. אבטחת מידע
   - שימוש בהצפנת AES עם וקטור אתחול, לשם אבטחת מידע רגיש.
5. קבלת פקודות על השרת
   - שימוש בתהליכון עבור האזנה לפקודות מהשרת.
   - מקבל פקודה מוצפנת, מפענח אותה, ומבצע אותה.
6. ביצוע פקודות לחיצת ושחרור המקשים והכפתורים.
   - מיפוי של מקשים מיוחדים.
   - שימוש בספריית pynput לשליטה בעכבר ובמקלדת.
   - אלטרנטביות: pyautogui, keyboard, ועוד.
   - מנגנון ביצוע לפי סוג הפקודה: לחיצת מקש, הרמת מקש, תזוזת עכבר ולחיצתו.
   - אם חסימת לקוח פועלת, מכבה אותה, מבצע את הפקודות, מדליק מחדש (תוך שימוש במנעולים כדי לא לגרום לשינוי סטטוס לא רצויים ותחרות בין התהליכונים).
7. חסימת לקוח
   - שימוש ב-pynput.Listener(suppress=True) לעכבר ולמקלדת.
   - השהיית החסימה לפני ביצוע קלט מרחוק, וחידושה לאחר מכן.

- <u>הפנייה למקורות רלוונטיים:</u>
  - Pynput – Keyboard & Mouse Control: https://pynput.readthedocs.io/
  - Tkinter Event Binding: https://docs.python.org/3/library/tkinter.html#bindings-and-events
  - Debouncing Key Events: https://en.wikipedia.org/wiki/Debounce
- <u>סקירת הפתרון הנבחר:</u>
  - <u>תיאור הפתרון:</u>

הפתרון מורכב ממספר רכיבים נפרדים העובדים בתיאום:

1. רכיב ממשק גרפי עבור איסוף קלט:
   - המשתמש לוחץ / מקליד בממשק הגרפי.
   - האירועים מנותחים – מיקום העכבר עובר מיפוי יחסי.
   - הפקודות מוזנות לתור הפקודות.
2. תהליכון שליחת פקודות:
   - מוציא פקודות מתור הפקודות.
   - מצפין אותם באזרת הצפנת AES.
   - שולח את הודעת הפקודה.
3. רכיב הלקוח לקבלת וזיהוי הפקודות:
   - הפקודות המוצפנות מתקבלות על ידי הלקוח, עוברות פענוח וזיהוי פקודה.

- הפקודה מבוצעת על ידי אובייקט שליטה מרחוק וחסימת לקוח.

4. אובייקט שליטה מרחוק:
   - שולט על העכבר והמקלדת.
   - מבצע את פקודות השליטה מרחוק.
   - אם חסימת לקוח פועלת, מכבה את החסימה, מבצע את הפקודה, ומדליק את החסימה (תוך שימוש במנעול).
5. אובייקט חסימת לקוח:
   - כאשר דלוק, מאזין למקלדת ולעכבר במחשב הלקוח ומונע מהם לעבוד.
   - כאשר כבוי, מאפשר שימוש חופשי בעכבר ובמקלדת.

- <u>נימוק הבחירה:</u>
  - ✓ ***דיוק ופשטות שליטה מרחוק***: מיפוי מיקום העכבר מאפשר שליטה מדויקת יחסית, נוחה ופשוטה.
  - ✓ ***פשטות ומודולריות***: הפרדת אחריות בין הממשק הגרפי לתהליכון ביצוע פקודות מאפשר מעקב פשוט וברור אחר תנועת תקשורת יוצאת, ועוזר בהפרדת אחריות בין הממשק הגרפי לשרת.
  - ✓ ***אמינות ויעילות***: שימוש ב-Debounce מונע שגיאות הקלדה כפולה וחוסך שליחת פקודות לא רלוונטיות.
  - ✓ ***אבטחה***: שימוש בהצפנה לשם אבטחת מידע בתקשורת.
- <u>שלילת פתרונות אלטרנטיביים:</u>
  - ✗ ***ביצוע פקודות ישירות ללא תור***:
    פתרון פחות גמיש – עלול לגרום לקונפליקטים, במיוחד בשליטה מרובת לקוחות או עומס.
  - ✗ ***שימוש בספריות pyautogui, keyboard***:
    פתרון אפשרי, אך דורש שימוש במספר ספריות שונות עבור החלפת pynput.
  - ✗ ***השארת חסימה תמידית***:
    עלולה לשבש את יכולת השליטה עצמה, ולכן יש להסירה זמנית בעת ביצוע פקודות.

❖ **<u>אלגוריתם סידור מסכים:</u>**

- <u>ניסוח וניתוח של הבעיה האלגוריתמית:</u>
  במסגרת הפרויקט, יש צורך באלגוריתם שיסדר את מסכי הלקוחות בתוך חלון ממשק המשתמש של השרת, תוך עמידה בדרישות הבאות:
  - סידור ללא חפיפות בין מסכים ותוויות קיימות.
  - סידור אסתטי של המסכים.
  - שמירה על יחס גובה/רוחב של המסכים (9:16).
- <u>תיאור אלגוריתמים קיימים לפתרון הבעיה:</u>
  - שימוש בסידור רשת
    1. חישוב דינמי של גודל הכפתורים
       - שמירה על יחס גובה/רוחב (Aspect Ratio) קבוע.
    2. חלוקת שטח המסך לרשת בגודל n על m
       - מימוש נפוץ בתצוגות עם פריטים מרובים (כגון גלריות).
  - שימוש במתמטיקה לסידור המסכים
    1. חישוב דינמי של גודל השטח הפנוי וגודל הכפתורים

- קביעה של רוחב וגובה בהתאם לגודל החלון ובהתחשב בגודל הרכיבים האחרים.
- שמירה על יחס גובה/רוחב (Aspect Ratio) קבוע.

2. מרכוז רכיבים בתוך השטח הפנוי
   - חישוב קיזוז אופקי ואנכי להצבה מדויקת.
   - מימוש מקובל באפליקציות גרפיות כדי לשמור על איזון עיצובי.
3. שיקולי ממשק גרפי נוספים
   - הימנעות מהצגת רכיבים מחוץ לגבולות המסך.

- <u>הפנייה למקורות רלוונטיים:</u>
  - Tkinter GUI Layout: https://tkdocs.com/tutorial/grid.html
  - Aspect Ratio & Responsive Design: https://en.wikipedia.org/wiki/Aspect_ratio_(image)
  - Python math module: https://docs.python.org/3/library/math.html
- <u>סקירת הפתרון הנבחר:</u>
  - <u>תיאור הפתרון:</u>

    האלגוריתם עובד בשלבים הבאים:

    1. בדיקת תנאי בסיס:
       - אם אין מסכים מוצגים, מוצגת הודעת "אין לקוחות."
       - אחרת, מוסרים רכיבים זמניים ומנקים את ממשק התצוגה.
    2. התאמת אזור מסך להצגה:
       - חישוב שטח פנוי בהתחשב במיקום וגודל של תווית המידע כדי למנוע חפיפה.
    3. קביעת כמות שורות ועמודות:
       - כמות מסכים בשורה (לפי שורש של N, N-מספר מסכים) ושורות בהתאמה.
       - טיפול מיוחד בשורה האחרונה אם אינה מלאה.
    4. חישוב גודל כל כפתור:
       - מבוסס על יחס גובה/רוחב של הסמכים שמוגדר מראש (16:9).
    5. מיקום כפתורים בצורה מרוכזת:
       - חישוב קיזוזים אופקיים ואנכיים.
       - הצבה מדויקת של כל כפתור בשורה ועמודה מתאימים.
  - <u>נימוק הבחירה:</u>
    - ✓ ***גמישות***: האלגוריתם מתאים עצמו לכל כמות לקוחות ומידות חלון.
    - ✓ ***שימור יחס תמונה***: שמירה על פרופורציה של המסכים.
    - ✓ ***עיצוב אחיד ונעים לעין***: מרכז את המסכים על המסך, נמנע מחפיפות.
    - ✓ ***התחשבות ברכיבי ממשק גרפי אחרים***: רכיבים קיימים (כגון תוויות) נלקחים בחשבון כדי שלא תהיה חפיפה בטעות.
  - <u>שלילת פתרונות אלטרנטיביים:</u>
    - ✗ ***סידור לינארי (טור או שורה אחת)***:

      מוביל לבזבוז מקום, ובנוסף כלל לא אסתטי ולא עומד בדרישות האלגוריתם.

שם פרויקט: "שליטת בוחן"
שם תלמיד: רון טקץ'- 330801853

## תיאור סביבת הפיתוח

- שפת פיתוח המערכת- "Python 3.13".
- סביבת פיתוח משולבת (IDE)- "PyCharm Community Edition".
- מערכת הפעלה עליה פותח- "Windows 11 pro".
- ספריות-
  - pynput
  - tkinter
  - zlib
  - queue
  - subprocess
  - sys
  - os
  - socket
  - ipaddress
  - threading
  - time
  - cryptography
  - io
  - PIL
  - math

## תיאור פרוטוקול התקשורת

**תיאור מילולי של פרוטוקול התקשורת:**

פרוטוקול התקשורת במערכת מבוסס על תקשורת TCP/IP לניהול כלל התקשורת במערכת.

למעט מספר הודעות החלפת מפתחות, כלל ההודעות בנויות במבנה: גודל ההודעה, וקטור האתחול (עבור ההצפנה), הודעה. בתוך ההודעה עשויים להיות תתי שדות, בהתאם לסוג ההודעה.

**פירוט כלל ההודעות הזורמות במערכת:**

❖ שם ההודעה: ClientPublicRSAKey
- נשלחת מ/אל: מהלקוח לשרת.
- מבנה השדות בהודעה:
  - גודל ההודעה: 4 בתים.
  - הודעה: המפתח הציבורי של הלקוח.

❖ שם ההודעה: ServerPublicRSAKey
- נשלחת מ/אל: מהשרת ללקוח.
- מבנה השדות בהודעה:
  - גודל ההודעה: 4 בתים.
  - הודעה: המפתח הציבורי של השרת.

❖ שם ההודעה: EncryptedAESKey
- נשלחת מ/אל: מהשרת ללקוח הנתון.
- מבנה השדות בהודעה:
  - גודל ההודעה: 4 בתים.
  - הודעה: המפתח הסימטרי של השרת.

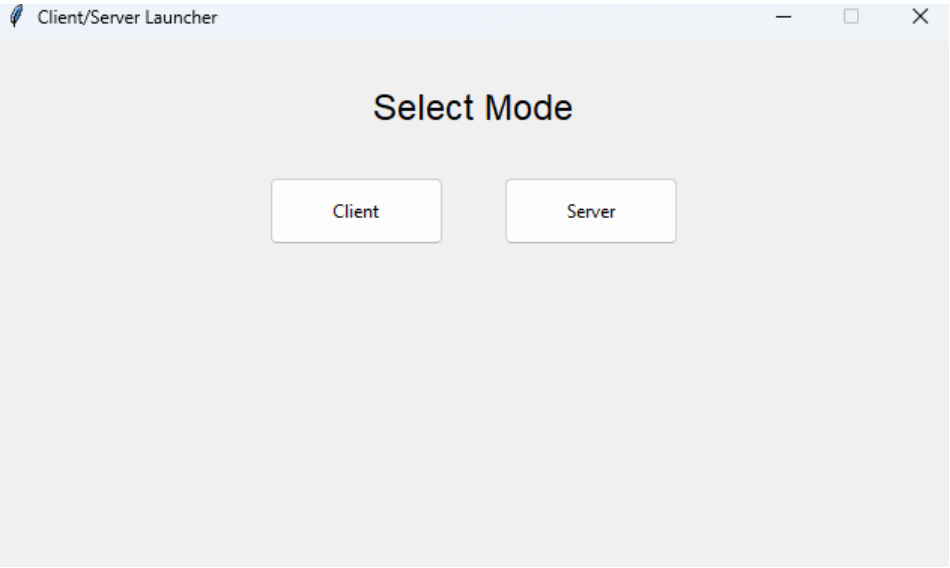
- שם ההודעה: Screenshot
  - נשלחת מ/אל: מהלקוח לשרת.
  - מבנה השדות בהודעה:
    - גודל ההודעה: 8 בתים.
    - וקטור אתחול: וקטור האתחול עבור ההודעה המוצפנת (16 בתים).
    - הודעה: המידע הדחוס של תמונת המסך.
- שם ההודעה: PauseAllExcept
  - נשלחת מ/אל: מהשרת לכל הלקוחות למעט הלקוח הנתון.
  - מבנה השדות בהודעה:
    - גודל ההודעה: 4 בתים.
    - וקטור אתחול: וקטור האתחול עבור ההודעה המוצפנת (16 בתים).
    - הודעה: פקודת הפסקת שליחת צילומי מסך- "pause".
- שם ההודעה: UnpauseAllExcept
  - נשלחת מ/אל: מהשרת לכל הלקוחות למעט הלקוח הנתון.
  - מבנה השדות בהודעה:
    - גודל ההודעה: 4 בתים.
    - וקטור אתחול: וקטור האתחול עבור ההודעה המוצפנת (16 בתים).
    - הודעה: פקודת המשך שליחת צילומי מסך- "unpause".
- שם ההודעה: ResizeAllExcept
  - נשלחת מ/אל: מהשרת לכל הלקוחות למעט הלקוח הנתון.
  - מבנה השדות בהודעה:
    - גודל ההודעה: 4 בתים.
    - וקטור אתחול: וקטור האתחול עבור ההודעה המוצפנת (16 בתים).
    - הודעה: מחולקת לשדות לפי " :"
      1. פקודה: שינוי גודל תמונות המסך שנשלחות על ידי הלקוחות- "resize".
      2. ערך-רוחב: רוחב שינוי הגודל.
      3. ערך-אורך: אורך שינוי הגודל.
- שם ההודעה: MouseInput
  - נשלחת מ/אל: מהשרת ללקוח הנתון.
  - מבנה השדות בהודעה:
    - גודל ההודעה: 4 בתים.
    - וקטור אתחול: וקטור האתחול עבור ההודעה המוצפנת (16 בתים).
    - הודעה: מחולקת לשדות לפי " :"
      1. פקודה: תשתמש בעכבר- "button".
      2. מספר כפתור: איזה כפתור של העכבר ללחוץ- "1/2/3".
      3. ערך-x: ערך ה-x של הלחיצה.
      4. ערך-y: ערך ה-y של הלחיצה.

- ❖ שם ההודעה: ButtonDown
  - נשלחת מ/אל: מהשרת ללקוח הנתון.
  - מבנה השדות בהודעה:
    - גודל ההודעה: 4 בתים.
    - וקטור אתחול: וקטור האתחול עבור ההודעה המוצפנת (16 בתים).
    - הודעה: מחולקת לשדות לפי " ":
      1. פקודה: תלחץ על מקש מקלדת- "key_down".
      2. ערך-מקש: המקש שיש ללחוץ.
- ❖ שם ההודעה: ButtonUp
  - נשלחת מ/אל: מהשרת ללקוח הנתון.
  - מבנה השדות בהודעה:
    - גודל ההודעה: 4 בתים.
    - וקטור אתחול: וקטור האתחול עבור ההודעה המוצפנת (16 בתים).
    - הודעה: מחולקת לשדות לפי " ":
      1. פקודה: תשחרר מקש מקלדת- "key_up".
      2. ערך-מקש: המקש שיש לשחרר.
- ❖ שם ההודעה: BlockUser
  - נשלחת מ/אל: מהשרת ללקוח הנתון.
  - מבנה השדות בהודעה:
    - גודל ההודעה: 4 בתים.
    - וקטור אתחול: וקטור האתחול עבור ההודעה המוצפנת (16 בתים).
    - הודעה: פקודת חסימת עכבר ומקלדת של הלקוח- "block"
- ❖ שם ההודעה: UnblockUser
  - נשלחת מ/אל: מהשרת ללקוח הנתון.
  - מבנה השדות בהודעה:
    - גודל ההודעה: 4 בתים.
    - וקטור אתחול: וקטור האתחול עבור ההודעה המוצפנת (16 בתים)
    - הודעה: פקודת ביטול חסימת עכבר ומקלדת של הלקוח- "unblock"
- ❖ שם ההודעה: KickUser
  - נשלחת מ/אל: מהשרת ללקוח הנתון.
  - מבנה השדות בהודעה:
    - גודל ההודעה: 4 בתים.
    - וקטור אתחול: וקטור האתחול עבור ההודעה המוצפנת (16 בתים)
    - הודעה: פקודת ניתוק הלקוח- "kick"

## תיאור מסכי המערכת

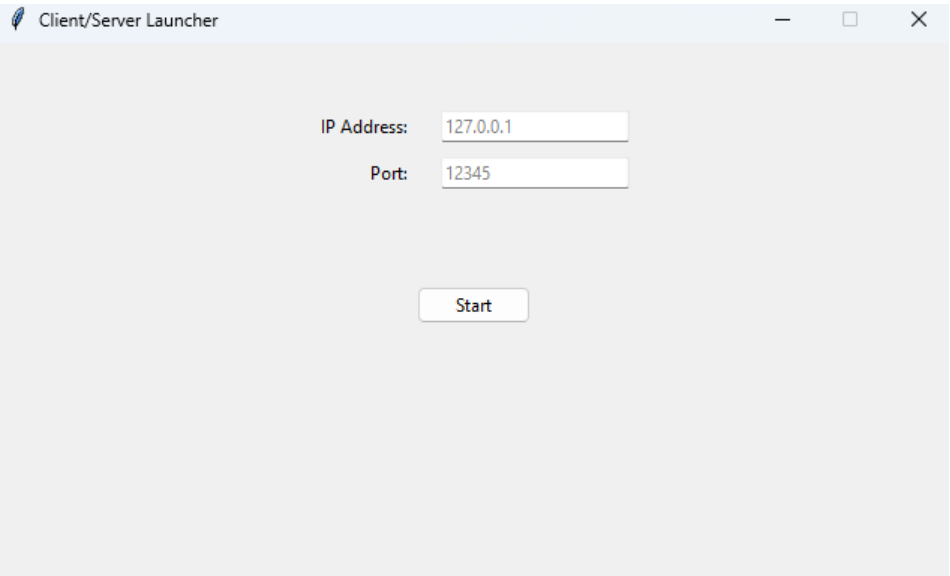
### ❖ מסך בחירת תפקיד

- תיאור המסך:
  - מסך התחלתי עם שני כפתורים לבחירת בתור איזה תפקיד להתחבר.
  - כאשר מתרחשת הפעלה מחדש (לאחר סגירת הלקוח או השרת מכל סיבה שהיא), מציג בנוסף באותיות אדומות סיבת הפעלה מחדש.
- תמונת מסך:



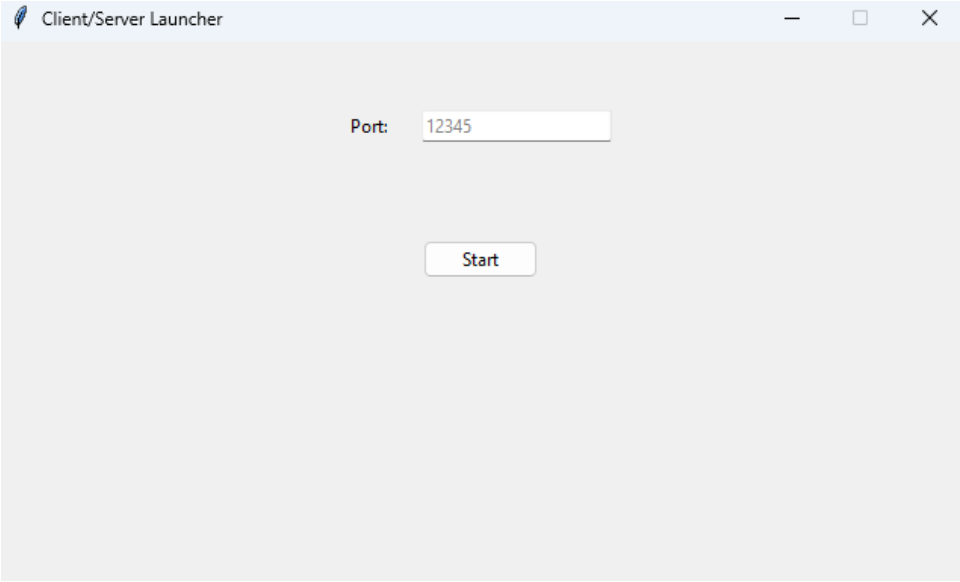

### ❖ מסך קלט מתלמיד

- תיאור המסך:
  - מסך עם שני שדות קלט IP ו-Port, אשר כברירת מחדל מכיל את הכתובת 127.0.0.1:12345 .
  - קיים במסך כפתור התחלת המערכת בתור התפקיד הנבחר מקודם ולפי הכתובת הנתונה.
  - כאשר הקלט אינו תקין, או שאי אפשר להתחבר לכתובת הנתונה, מציג באותיות אדומות הודעת שגיאה עם הסבר.
- תמונת מסך:



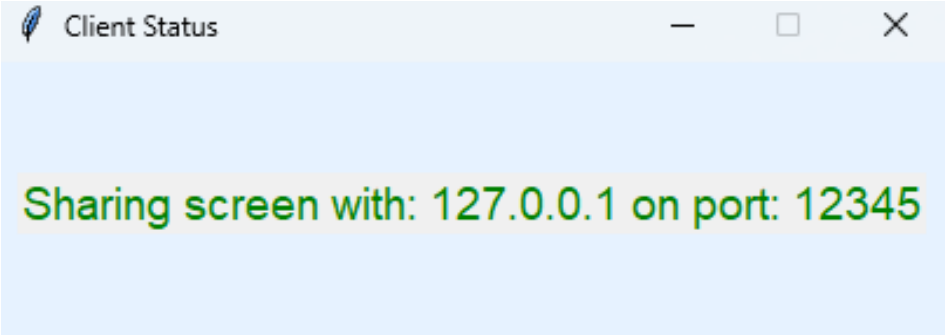
### ❖ מסך קלט מהמורה

- תיאור המסך:
  - מסך עם שדה קלט Port, אשר כברירת מחדל מכיל 12345 .
  - קיים במסך כפתור התחלת המערכת בתור התפקיד הנבחר מקודם ולפי ה-Port הנתון.
  - כאשר הקלט אינו תקין, או שאי אפשר להקים שרת ב-Port הנתון, מציג באותיות אדומות הודעת שגיאה עם הסבר.
- תמונת מסך:



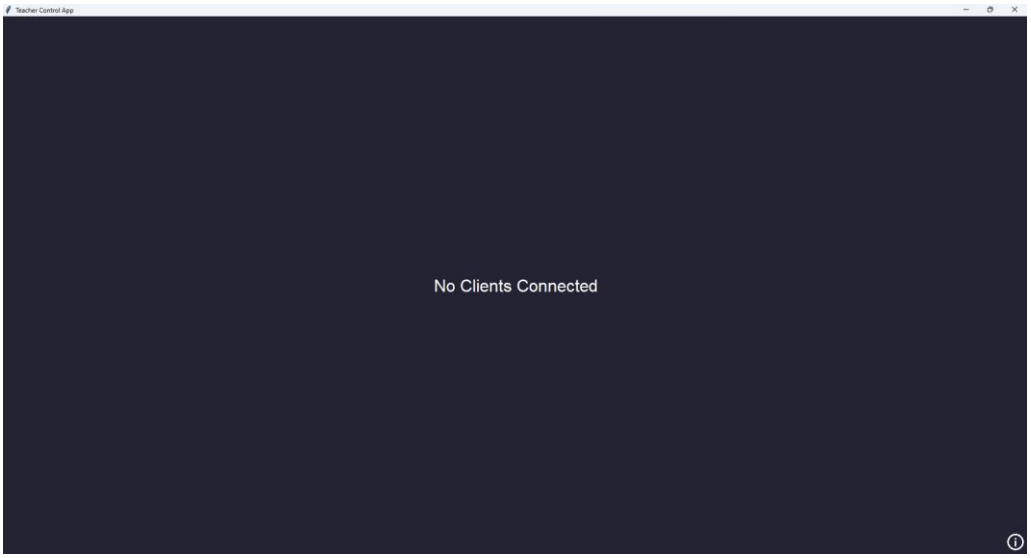

### ❖ מסך סטטוס תלמיד

- תיאור המסך:
  - מסך מינימלי אשר מראה שמחובר לשרת בכתובת אשר ניתנה ב-"**מסך קלט מתלמיד**".
- תמונת מסך:



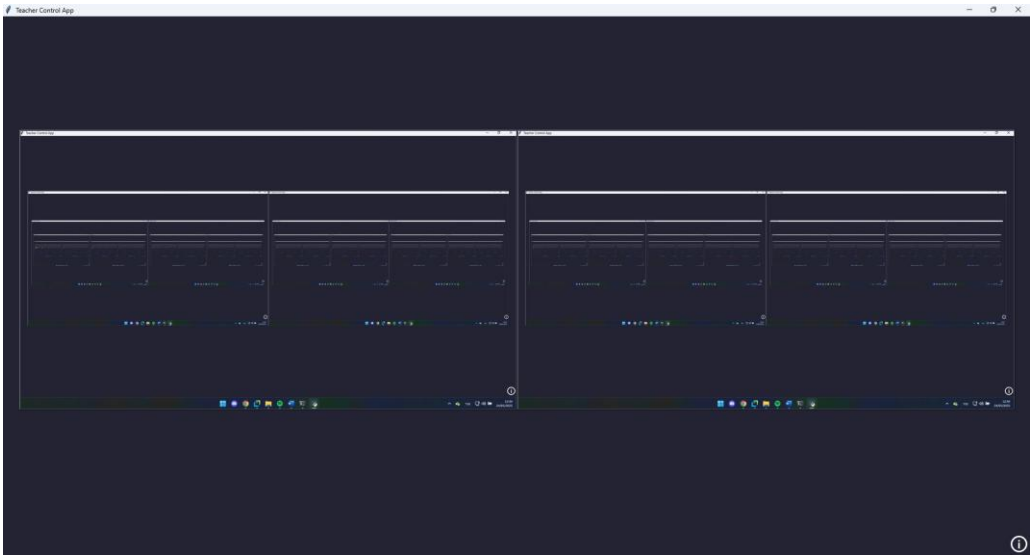
❖ **מסך רשת מסכי תצוגה מקדימה (ללא תלמידים)**

- תיאור המסך:
  - מציג הודעת "אין לקוחות מחוברים".
  - בפינה הימנית למטה יש אייקון מידע. כאשר מרחפים עם העכבר מעל האייקון, מציג את כתובת ההתחברות של השרת (מה שהלקוח צריך לכתוב ב-"**מסך קלט מתלמיד**" ובנוסף, מציג גם את מספר הלקוחות המחוברים לשרת.
- תמונת מסך:




❖ **מסך רשת מסכי תצוגה מקדימה (עם תלמידים)**

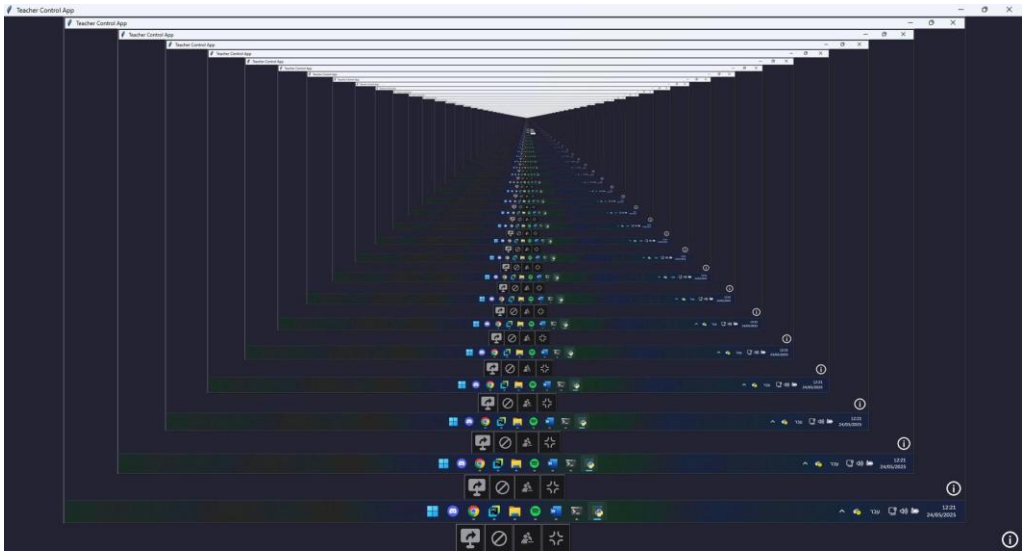
- תיאור המסך:
  - מציג רשת מסכי תצוגה מקדימה לקוחות.
  - כל מסך הוא כפתור, שכאשר נלחץ, פותח את המסך ב-"**מסך מלא**".
  - בפינה הימנית למטה יש אייקון מידע. כאשר מרחפים עם העכבר מעל האייקון, מציג את כתובת ההתחברות של השרת (מה שהלקוח צריך לכתוב ב-"**מסך קלט מתלמיד**" ובנוסף, מציג גם את מספר הלקוחות המחוברים לשרת.
- תמונת מסך:



### ❖ מסך שליטה (״מסך מלא״)

- תיאור המסך:
  - מציג ״מסך מלא״ של לקוח מסויים.
  - למטה באמצע יש 4 כפתורים (הסבר מפורט, משמאל לימין):
    1. כפתור הפעלת ״שליטה מרחוק״- לאחר שנלחץ, כאשר לוחצים על ה״מסך המלא״ עם העכבר, פעולת העכבר מתרחשת גם אצל הלקוח. כמו כן, לאחר שנלחץ, כל מקש מקלדת שנלחץ על ידי המשתמש, מרחש גם אצל הלקוח. כאשר ״שליטה מרחוק״ דלוקה, הכפתור משנה את תמונתו ואת הרמז הצץ (tooltip) שלו. על מנת לכבות ״שליטה מרחוק״ יש ללחוץ על הכפתור שוב, או ללחוץ על כפתור ״יציאה ממסך מלא״ או כפתור ״ניתוק לקוח״.
    2. כפתור הפעלת ״חסימת לקוח״- לאחר שנלחץ, העכבר והמקלדת של הלקוח נחסם. כאשר ״חסימת לקוח״ דלוקה, הכפתור משנה את תמונתו ואת הרמז הצץ (tooltip) שלו. על מנת לכבות ״חסימת לקוח״ יש ללחוץ על הכפתור שוב או כפתור ״ניתוק לקוח״ (כפתור ״יציאה ממסך מלא״ לא יכבה את ״חסימת לקוח״).
    3. כפתור ״ניתוק לקוח״- מנתק את הלקוח, מכבה את ״שליטה מרחוק״ ואת ״חסימת לקוח״. כמו כן, יוצא מהמסך המלא.
    4. כפתור ״יציאה ממסך מלא״- מכבה את ״שליטה מרחוק״ ויוצא מהמסך המלא.
  - בפינה הימנית למטה יש אייקון מידע. כאשר מרחפים עם העכבר מעל האייקון, מציג את כתובת ההתחברות של השרת (מה שהלקוח צריך לכתוב ב-״**מסך קלט מתלמיד**״ ובנוסף, מציג גם את מספר הלקוחות המחוברים לשרת.

- תמונת מסך:



## תיאור מבני הנתונים

- תורים (Queue):
  - ❖ frame_queue:
    - תור תמונות מסך לעדכון על ידי השרת.
      מבנה: tuple[tuple[str, int], bytes, float].
    - ערכים:
      1. סדורה המכילה כתובת לקוח (מבנה tuple[str, int]).
      2. מידע של תמונת מסך (מבנה bytes).
      3. חותמת זמן (מבנה float).
  - ❖ command_queue:
    - תור פקודות לביצוע של השרת .
      מבנה: tuple[tuple[str, int], str].
    - ערכים:
      1. סדורה המכילה כתובת לקוח (מבנה tuple[str, int]).
      2. פקודה (מבנה str).
- מילונים (Dictionary):
  - ❖ rsa_keys:
    - מפתח- סדורה המכילה כתובת לקוח (מבנה tuple[str, int]).
    - ערך- אובייקט מפתח RSA ציבורי.
  - ❖ client_sockets_and_threads:
    - מפתח- סדורה המכילה כתובת לקוח (מבנה tuple[str, int]).
    - ערך- סדורה המכילה חיבור ללקוח ואובייקט הטיפול בו (מבנה tuple[socket.socket, ClientHandler]).
  - ❖ Buttons:
    - מפתח- סדורה המכילה כתובת לקוח (מבנה tuple[str, int]).
    - ערך- כפתור (מבנה tk.Button).
  - ❖ fullscreen_buttons:
    - מפתח- שם הכפתור (מבנה str).
    - ערך- כפתור (מבנה tk.Button).
  - ❖ block_states:
    - מפתח- סדורה המכילה כתובת לקוח (מבנה tuple[str, int]).
    - ערך- בוליאני (מבנה bool).
  - ❖ last_key_press_time:
    - מפתח- מקש מקלדת (מבנה str).
    - ערך- חותמת זמן (מבנה float).
- רשימות (List):
  - ❖ images: רשימת תמונות כדי שלא יעברו "איסוף זבל" (מסוג ImageTk.PhotoImage).
- קבוצה (Set):
  - ❖ kicked_addresses: קבוצת סדורות המכילות כתובות של לקוחות מנותקים (מבנה tuple[str, int]).
  - ❖ pressed_keys: קבוצת מקשי מקלדת שלחוצים כרגע (מבנה str).

## סקירת חולשות ואיומים

- **שכבת האפלקציה:**
  - איום: התחברות של לקוחות לא רצויים לפגישות.
    פתרון: אין.
  - איום: ציטוט למידע במהלך העברתו (MITM).
    פתרון: שימוש בהצפנה מקצה לקצה: RSA לחילוף מפתחות AES להצפנה סימטרית של הודעות.
  - איום: הצפת השרת בבקשות שמונעות גישה מהמשתמשים החוקיים (DoS / DDoS).
    פתרון: אין.

- **שכבת התעבורה:**
  - איום: SYN flood (DoS על שלב פתיחת הקשר).
    פתרון: אין.
  - איום: שימוש במפתחות חלשים.
    פתרון: שימוש במפתחות RSA של 2048 ביט.

- **מערכת ההפעלה:**
  - איום: הזרקת קוד זדוני בתמונת המסך שנשלחים על ידי הלקוח.
    פתרון: שימוש בספריות לטיפול תמונות בטוחות ומעודכנות.

שם פרויקט: "שליטת בוחן"
שם תלמיד: רון טקץ'- 330801853

# מימוש הפרויקט

## חלק א'

**מודולים/ מלחקות מיובאים:**

- threading- מאפשר יצירת וניהול תהליכונים (threads) להרצת משימות במקביל.
- socket- מאפשר תקשורת בין שרת ללקוחות באמצעות TCP.
- ipaddress- משמש לבדיקת כתובות ip.
- queue- מבנה נתונים לתורים, משמש להעברת מידע בין תהליכונים.
- zlib- משמש לדחיסה ופריסה של מידע.
- time- משמש למדידת זמן, ביצוע השהיות וחישוב תזמונים בין תמונות מסך.
- cryptography- משמש להצפנה.
- subprocess- משמש להפעלת תתי תהליכים.
- sys- משמש להרצת תהליכים.
- os- משמש ליצירת מידע אקראי (למפתחות ו-וקטורי אתחולי) ולמציאת מיקום קבצים.
- PIL- צילום מסך של המסך הראשי.
- io- שמירת תמונה בזיכרון כנתוני בתים.
- pynput- שליטה והאזנה לעכבר ולמקלדת.
- tkinter- ממשק משתמש גרפי ליצירת GUI.
- math- משמש למירכוז וסידור מסכי הלקוחות במסך רשת מסכי תצוגה מקדימה.

**מודולים/מחלקות שאני פיתחתי:**

- מודול: Encryption
  - תפקיד המודול: מספקת פונקציות להצפנה ופענוח של נתונים באמצעות RSA ו-AES.
  - פעולות המודול:
    - generate_rsa_keys()
      טענת כניסה: ללא.
      טענת יציאה: מחזירה זוג מפתחות RSA (פרטי, ציבורי).
    - generate_aes_key()
      טענת כניסה: ללא.
      טענת יציאה: מחזירה מפתח AES באורך 32 בתים.
    - rsa_encrypt()
      טענת כניסה: מפתח RSA ציבורי (public_key) וטקסט/בתים להצפנה (text).
      טענת יציאה: מחזירה את הטקסט, מוצפן עם RSA.
    - rsa_decrypt()
      טענת כניסה: מפתח RSA פרטי (private_key) וטקסט מוצפן (ciphertext).
      טענת יציאה: מחזירה את הטקסט המפוענח.
    - serialize_public_key()
      טענת כניסה: מפתח RSA ציבורי (public_key).
      טענת יציאה: מחזירה את המפתח בפורמט PEM (מחרוזת בתים).
    - deserialize_public_key()
      טענת כניסה: מחרוזת בתים (בפורמט PEM).
      טענת יציאה: מחזירה את המפתח RSA הציבורי.

- encrypt_aes()
  טענת כניסה: מפתח AES (key) ומחרוזת/בתים להצפנה (text).
  טענת יציאה: מחזירה וקטור אתחול ואת הטקסט המוצפן עם הוספת ריפוד.
- decrypt_aes(key, iv, ciphertext)
  טענת כניסה: מפתח AES (key), וקטור אתחול (iv) וטקסט מוצפן (ciphertext).
  טענת יציאה: מחזירה את הטקסט המפוענח ללא הריפוד.

❖ מחלקה: UserBlocker

- תפקיד המחלקה: חוסמת קלט מהמקלדת ומהעכבר על ידי שימוש במאזינים (Listener) של pynput.
- תכונות המחלקה:
  - keyboard_listener- אובייקט שמאזין לקלט מהמקלדת וחוסם אותו.
  - mouse_listener- אובייקט שמאזין לקלט מהעכבר וחוסם אותו.
- פעולות המחלקה:
  - __init__()
    טענת כניסה: ללא.
    טענת יציאה: מאתחל את המחלקה.
  - start_blocking()
    טענת כניסה: ללא.
    טענת יציאה: מתחיל לחסום קלט מהמקלדת והעכבר.
  - stop_blocking()
    טענת כניסה: ללא.
    טענת יציאה: מפסיק את החסימה של הקלט (אם פועל).

❖ מחלקה: InputController

- תפקיד המחלקה: שולט על פעולות עכבר ומקלדת – מקבל פקודות ומבצע אותן, תוך כדי טיפול בחסימה לפי הצורך.
- תכונות המחלקה:
  - block_event- דגל שמציין אם הקלט חסום.
  - block_event_lock- מנגנון נעילה שמונע תחרות על block_event.
  - user_blocker- מופע של UserBlocker שאחראי לחסימת הקלט.
  - mouse- אובייקט לביצוע פעולות שליטה בעכבר.
  - keyboard- אובייקט לביצוע פעולות שליטה במקלדת.
- פעולות המחלקה:
  - __init__()
    טענת כניסה: block_event, block_event_lock, user_blocker.
    טענת יציאה: מאתחל את המחלקה ואת התכונות שלה.
  - set_mouse_pos()
    טענת כניסה: מיקום ערך רוחב (x) ומיקום ערך גובה (y).
    טענת יציאה: מעדכן את מיקום הסמן של העכבר.
  - handle_command()
    טענת כניסה: מחרוזת פקודה (command).
    טענת יציאה: מבצע פעולה בהתאם לסוג הפקודה- לחיצה בעכבר, הקשה או שחרור מקש.
  - _press_key()
    טענת כניסה: מחרוזת מקש (key)
    טענת יציאה: לוחץ מקש במקלדת.

- _release_key()
  טענת כניסה: מחרוזת מקש (key)
  טענת יציאה: משחרר את המקש שנלחץ.

❖ פעולה: take_screenshot()

- תפקיד הפונקצייה: מבצע צילום מסך, משנה את הגודל, שומר את התמונה בפורמט JPEG באיכות נתונה, ומחזיר את הנתונים בבייטים.
- טענת כניסה: רוחב תמונה- ברירת מחדל: 1920 (x_size), גובה תמונה- ברירת מחדל: 1080 (y_size) ואיכות- ברירת מחדל: 75 (quality).
- טענת יציאה: מחזירה את התמונה בבתים.

❖ פעולה: recv_all()

- תפקיד הפונקצייה: מבצע קבלה מלאה של הודעה.
- טענת כניסה: חיבור אל הלקוח (sock) ומספר בתים לצפות (num_bytes).
- טענת יציאה: מחזירה את המידע שהתקבל (בבתים).

❖ מחלקה: StarterApp

- תפקיד המחלקה: ממשק ראשי שמאפשר למשתמש לבחור אם להפעיל את התוכנה כלקוח או כשרת, ולהזין כתובת IP ו-Port.
- תכונות המחלקה:
  - root- אובייקט שורש של tkinter.
  - reason- סיבה להפעלה מחדש (אם קיימת).
  - client_switch- מציין אם נבחר מצב לקוח (True) או שרת (False).
  - רכיבי ממשק גרפי: client_button, server_button, input_frame, start_button, ip_entry, port_entry- שדות קלט וכפתורים.
- פעולות המחלקה:
  - __init__()
    טענת כניסה: אובייקט שורש (root) וסיבה- ברירת מחדל: ללא (reason).
    טענת יציאה: יוצר, מאתחל תכונות ומכין את הממשק.
  - create_widgets()
    טענת כניסה: ללא.
    טענת יציאה: יוצר את כפתורי הבחירה Client / Server.
  - clear_initial_buttons()
    טענת כניסה: ללא.
    טענת יציאה: מסתיר את כפתורי הבחירה.
  - client_mode()
    טענת כניסה: ללא.
    טענת יציאה: בוחר מצב לקוח ומציג שדות קלט.
  - server_mode()
    טענת כניסה: ללא.
    טענת יציאה: בוחר מצב שרת ומציג שדות קלט.
  - show_input_fields()
    טענת כניסה: ללא.
    טענת יציאה: מציג שדות IP ו-Port בהתאם למצב.
  - add_placeholder()
    טענת כניסה: שדה (entry) וערך ברירת מחדל (placeholder).
    טענת יציאה: מאפשר ערך זמני שנעלם בלחיצה.
  - is_valid_ip()

טענת כניסה: מחרוזת כתובת IP (ip).
טענת יציאה: מחזיר True אם כתובת תקינהת אחרת False.

- is_port_available()
  טענת כניסה: מספר Port (port).
  טענת יציאה: מחזיר True אם ה-Port פנוי, אחרת False.
- can_connect_to_server()
  טענת כניסה: מחרוזת כתובת IP (ip) ומספר Port (port).
  טענת יציאה: מחזיר True אם ניתן להתחבר לשרת, אחרת False.
- start_pressed()
  טענת כניסה: ללא.
  טענת יציאה: מאמת קלט ומריץ את מצב הלקוח או השרת.
- run()
  טענת כניסה: ללא.
  טענת יציאה: מתחיל את הלולאה של tkinter.

❖ פעולה: run_client_mode()

- תפקיד הפונקצייה: מריץ את מצב הלקוח: יוצר ממשק גרפי, מחבר את הלקוח לשרת ומאזין לניתוק.
- טענת כניסה: מחרוזת כתובת IP (ip) ומספר Port (port).
- טענת יציאה: מפעיל מחדש את התוכנה אם יש סגירה או ניתוק.

❖ פעולה: run_server_mode()

- תפקיד הפונקצייה: מריץ את מצב השרת: יוצר ממשק גרפי מפעיל תור פקודות ותור תמונות מסך לעדכון ומאזין לחיבורים.
- טענת כניסה: מחרוזת כתובת IP (ip) ומספר Port (port).
- טענת יציאה: מפעיל מחדש את התוכנה אם יש סגירה.

❖ פעולה: main()

- תפקיד הפונקצייה: טוען את חלון הבחירה הראשוני, ומפעיל אותו.
- טענת כניסה: ללא.
- טענת יציאה: יוצא מהפעולה כשהתוכנית נעצרת.

❖ מחלקה: Server

- תפקיד המחלקה: מנהל את שרת ה-TCP כולל קבלת חיבורים מלקוחות, ניהול מפתחות הצפנה, שליחת פקודות, קבלת תמונות מסך ועוד.
- תכונות המחלקה:
  - ip- כתובת של השרת.
  - port- פורט של השרת.
  - app- הממשק הגרפי של השרת.
  - server- אובייקט socket שמאזין לחיבורים נכנסים.
  - close_callback- פונקציה שתיקרא בעת ניתוק.
  - public_key- מפתח RSA ציבורי של השרת להצפנה אסימטרית.
  - private_key- מפתח RSA פרטי של השרת להצפנה אסימטרית.
  - aes_key- מפתח AES להצפנה סימטרית של נתונים.
  - rsa_keys- מפתחות ציבוריים של לקוחות.
  - client_sockets_and_threads- מיפוי של כתובות לקוחות לחיבור אל הלקוח והתהליכון שמטפל בלקוח.
  - command_queue- תור לפקודות.
  - frame_queue- תור לתמונות מסך לעדכון.

- pause_event- דגל להפסקת שליחת נתונים זמנית.
- stop_event- דגל לסיום הפעולה.
- command_sender- תהליכון ששולח פקודות ללקוחות.
- frame_updater- תהליכון שמעדכן את המסך בהתאם לתמונות המסך מהלקוחות.

- פעולות המחלקה:
  - __init__()
    טענת כניסה: כתובת ה-IP של השרת (ip), ה-Port של השרת (port), הממשק הגרפי של השרת (app), תור פקודות (command_queue), תור תמונות מסך לעדכון (frame_queue), פונקציית חזרה בסגירה- ברירת מחדל: ללא (close_callback).
    טענת יציאה: מאתחל את שרת ה-TCP ואת כל המשתנים הדרושים לניהול לקוחות, הצפנה ותהליכונים.
  - run()
    טענת כניסה: ללא
    טענת יציאה: מאזין לחיבורים, מבצע החלפת מפתחות, מפעיל תהליכוני טיפול פקודות וטיפול בעדכון תמונות מסך, יוצר תהליכונים לכל לקוח חדש.
  - stop()
    טענת כניסה: סיבת סגירת השרת (stop_reason).
    טענת יציאה: עוצר את פעילות השרת, סוגר חיבורים, מפסיק תהליכונים ומפעיל קריאת חזרה אם סופקה.
  - key_exchange()
    טענת כניסה: ללא.
    טענת יציאה: מאזין לחיבור חדש, מבצע החלפת מפתחות הצפנה בין השרת ללקוח ומחזיר סדורת כתובת לקוח אם הצליח, אחרת (None, None).

❖ מחלקה: ClientHandler

- תפקיד המחלקה: מטפל בחיבור עם לקוח יחיד – מקבל ממנו תמונות מסך, מפענח ומעביר לתור.
- תכונות המחלקה:
  - client_socket- החיבור אל הלקוח.
  - client_address- סדורת כתובת הלקוח.
  - frame_queue- תור לתמונות מסך לעדכון.
  - aes_key- מפתח AES להצפנה סימטרית של נתונים.
  - app- הממשק הגרפי של השרת.
  - stop_event- דגל להפסקת פעילות.
  - min_frame_interval- מרווח זמן מינימלי בין קבלת תמונות מסך.
- פעולות המחלקה:
  - __init__()
    טענת כניסה: החיבור אל הלקוח (client_socket), סדורת כתובת הלקוח (client_address), תור לתמונות מסך לעדכון (frame_queue), מפתח הצפנה AES (aes_key) וממשק גרפי של השרת (app).
    טענת יציאה: מאתחל תהליכון שמטפל בתקשורת עם לקוח יחיד, כולל הצפנת הנתונים ושמירתם בתור.
  - run()
    טענת כניסה: ללא.
    טענת יציאה: מאזין לזרם נתונים מהלקוח, מפענח, מפרק ומעביר את תמונות המסך לתור לעיבוד.

- stop()
  טענת כניסה: ללא.
  טענת יציאה: מסמן לתהליכון להפסיק לפעול.

❖ מחלקה: FrameConsumer

- תפקיד המחלקה: מקבל תמונות מסך מהתור ומעדכן את ממשק השרת בתמונה לפי כתובת הלקוח.
- תכונות המחלקה:
  - app- ממשק הגרפי של השרת.
  - frame_queue- תור שבו נשמרים תמונות מסך וחותמת זמן.
  - max_frame_age- מגבלת זמן לתמונות מסך ישנים.
  - stop_event- דגל להפסקת פעילות.
- פעולות המחלקה:
  - __init__()
    טענת כניסה: ממשק גרפי של השרת (app) ותור תמונות מסך לעדכון (frame_queue).
    טענת יציאה: מאתחל תהליכון שמקבל תמונות מסך מהתור ומעדכן את המסך בהתאם.
  - run()
    טענת כניסה: ללא.
    טענת יציאה: מאזין לתור תמונות המסך לעדכון, בודק אם התמונה עדכנית, ומציג אותו בממשק המשתמש. תמונות מסך ישנות נזרקות.
  - stop()
    טענת כניסה: ללא.
    טענת יציאה: מסמן לתהליכון להפסיק לפעול.

❖ מחלקה: CommandConsumer

- תפקיד המחלקה: מקבל פקודות מתור הפקודות ושולח ללקוחות.
- תכונות המחלקה:
  - clients_dict- מיפוי של כתובות לקוחות אל התהליכון והחיבור האחראיים על הלקוח.
  - clients_lock- מנגנון סנכרון לשימוש בטוח במבנה הנתונים.
  - command_queue- תור לפקודות שמגיעות לשרת.
  - pause_event- דגל שמייצג האם השהיית שיתופי מסך מלקוחות פועל.
  - stop_event- דגל להפסקת פעילות.
  - app- ממשק הגרפי של האפליקציה.
- פעולות המחלקה:
  - __init__()
    טענת כניסה: מילון של כתובות לקוחות והתהליכונים והחיבורים שלהם (clients_dict), מנעול גישה למילון (client_lock), תור פקודות (command_queue), דגל השהיית שליחת שיתופי מסך (pause_event) וממשק גרפי של השרת (app).
    טענת יציאה: מאתחל תהליכון שמאזין לפקודות ושולח אותן ללקוחות בצורה מאובטחת.
  - run()
    טענת כניסה: ללא.
    טענת יציאה: שואב פקודות מהתור, מצפין אותן, ושולח ללקוח/לקוחות הרלוונטיים. מטפל בשגיאות ומנקה לקוחות ניתוקים.

- stop()
  טענת כניסה: ללא.
  טענת יציאה: מסמן לתהליכון להפסיק לפעול.

❖ מחלקה: ServerGui

- תפקיד המחלקה: מספקת ממשק גרפי לניהול ושליטה על לקוחות (תלמידים) המחוברים לשרת. מאפשרת צפייה בשיתוף המסך שלהם, הפעלת פקודות כמו חסימה, השתלטות, העפה, ועוד.
- תכונות המחלקה:
  - ip- כתובת ה-IP המקומית של המחשב המארח.
  - port- ה-Port שבו השרת פועל.
  - root- אובייקט שורש של tkinter.
  - command_queue- תור לשליחת פקודות ללקוחות.
  - frame_queue- תור המכיל תמונות מסך לעדכון משיתופי מסך של הלקוחות.
  - buttons- מיפוי של כתובות IP לכפתורים המציגים את שיתוף המסך של הלקוח.
  - images- רשימה של תמונות תצוגה מקדימה עבור הלקוחות.
  - fullscreen_widget- ווידג'ט תצוגה במצב מסך מלא של לקוח בודד.
  - fullscreen_address- כתובת ה-IP של הלקוח במצב מסך מלא.
  - control_switch- בוליאני המייצג אם מצב "שליטה" פעיל.
  - block_states- מיפוי של מצב חסימה עבור כל לקוח.
  - kicked_addresses- כתובות של לקוחות שנותקו.
  - kicked_lock- מנגנון סנכרון לטיפול בכתובות שנותקו.
  - pressed_keys- סט של מקשים שנלחצו במצב שליטה.
  - pressed_keys_lock- מנעול לסנכרון גישה למקשים.
  - last_key_press_time- מיפוי של מתי נלחץ מקש.
  - shutdown_event- דגל שמודיע על סגירת האפליקציה.
  - no_clients_label- רכיב הודעה על חוסר לקוחות.
  - info_label- רכיב תצוגה למידע כללי.
  - fullscreen_buttons- מיפוי של כפתורי שליטה (חסימה, ניתוק, שליטה, יציאה).
- פעולות המחלקה:
  - __init__()
    טענת כניסה: ה-Port שעליו פועל השרת (port), אובייקט שורש (root), תור פקודות (command_queue) ותור תמונות מסך לעדכון (frame_queue).
    טענת יציאה: מאתחל את ממשק המשתמש, כולל פריסת מסכים, תוויות, לחצנים, אירועים ואתחול משתנים.
  - run()
    טענת כניסה: ללא.
    טענת יציאה: מפעיל את לולאת tkinter לצורך ריצת הממשק הגרפי.
  - organize_screens()
    טענת כניסה: ללא.
    טענת יציאה: מארגן את כפתורי הלקוחות על מסך "רשת מסכי תצוגה מקדימה" לפי מספרם וגודל המסך, ממרכז אותם ומונע חפיפות עם כפתורים אחרים.
  - show_no_clients_message()
    טענת כניסה: ללא.
    טענת יציאה: מציג הודעה כאשר אין לקוחות מחוברים לממשק.

- _add_screen_on_main_thread()
  טענת כניסה: כתובת לקוח (address) ותמונה מקדימה (preview_photo).
  טענת יציאה: מוסיף את הלקוח לרשימת מסכי הלקוחות ומעדכן את תצוגת הממשק אם לא במסך מלא.
- update_screen()
  טענת כניסה: כתובת לקוח (address) ובתים של תמונת מסך (image_bytes).
  טענת יציאה: ממיר את התמונה ומטפל בעידכונה.
- _update_screen_on_main_thread()
  טענת כניסה: כתובת לקוח (address) ותמונות מעודכנות ( preview_photo, fullscreen_photo).
  טענת יציאה: מעדכן את הכפתור או את המסך המלא של הלקוח בממשק הגרפי.
- toggle_fullscreen()
  טענת כניסה: כתובת לקוח (address) ותמונה בגודל מלא (image_bytes).
  טענת יציאה: נכנס למצב מסך מלא עבור לקוח מסוים.
- fullscreen_buttons_create()
  טענת כניסה: ללא.
  טענת יציאה: יוצר ומכין את כפתורי הפעולה עבור מצב מסך מלא (שליטה, חסימה, יציאה וכו').
- fullscreen_buttons_show()
  טענת כניסה: ללא.
  טענת יציאה: מציג את כל כפתורי הפעולה במסך מלא.
- fullscreen_buttons_hide()
  טענת כניסה: ללא.
  טענת יציאה: מסתיר את כפתורי הפעולה במסך מלא.
- exit_fullscreen()
  טענת כניסה: ללא.
  טענת יציאה: יוצא ממצב מסך מלא חזרה למצב רשת מסכי תצוגה מקדימה ומחביא את כפתורי הפעולה.
- handle_mouse_click()
  טענת כניסה: אירוע לחיצת עכבר (event).
  טענת יציאה: מתרגם קואורדינטות למסך ומעביר פקודת לחיצה ללקוח במסך המלא.
- on_key_press()
  טענת כניסה: אירוע לחיצה על מקש (event).
  טענת יציאה: מעביר פקודת לחיצת מקש ללקוח במסך המלא, תוך שימוש במנגנון דיבאונס.
- on_key_release()
  טענת כניסה: אירוע שחרור מקש (event).
  טענת יציאה: מעביר פקודת שחרור מקש ללקוח במסך המלא ומנקה את מצב המקש.
- control_start()
  טענת כניסה: ללא.
  טענת יציאה: מפעיל שליטה מלאה (מקלדת ועכבר) על הלקוח במסך מלא.
- control_stop()
  טענת כניסה: ללא.
  טענת יציאה: מבטל את מצב השליטה על הלקוח במסך מלא.
- update_control_button()
  טענת כניסה: שם הפעולה (name).

טענת יציאה: מעדכן את הסמל ואת הרמז הצץ של הכפתור לפעולת השליטה בהתאם למצב הנוכחי.

- block()
  טענת כניסה: ללא.
  טענת יציאה: שולח פקודת התחלת חסימה ללקוח.
- unblock()
  טענת כניסה: ללא.
  טענת יציאה: שולח פקודת הפסקת חסימה ללקוח.
- update_block_button()
  טענת כניסה: שם הפעולה (name).
  טענת יציאה: מעדכן את הסמל ואת הרמז הצץ של הכפתור לפעולת החסימה בהתאם למצב הנוכחי.
- kick()
  טענת כניסה: כתובת לקוח- ברירת מחדל: ללא (address).
  טענת יציאה: שולח פקודת ניתוק ללקוח ומפעיל פעולת ניקוי.
- cleanup()
  טענת כניסה: כתובת לקוח (target).
  טענת יציאה: מנקה את כל המשאבים הקשורים ללקוח (כפתורים, תמונות, מצבים, תצוגות וכו').
- on_close()
  טענת כניסה: אין.
  טענת יציאה: מסמן על סגירה בטוחה ומסיים את הממשק.
- allow_address()
  טענת כניסה: כתובת לקוח (address).
  טענת יציאה: מסיר לקוח מרשימת החסומים (אם נותק).
- remove_frames_for_address()
  טענת כניסה: כתובת לקוח (address).
  טענת יציאה: מנקה את תור תמונות מסך לעדכון מכל התמונות שמקורן בלקוח הנתון.
- add_info_button()
  טענת כניסה: אין.
  טענת יציאה: יוצר אייקון מידע עם כתובת ה-IP ה-Port ומספר חיבורים.
- get_info_text()
  טענת כניסה: אין.
  טענת יציאה: מחזיר מחרוזת מידע עדכנית על מצב השרת (IP, Port ומספר חיבורים).

❖ מחלקה: ToolTip

- תפקיד המחלקה: מציגה רמז צץ (tooltip) כאשר העכבר מרחף מעל ווידג'ט.
- תכונות המחלקה:
  - widget- הווידג'ט שעליו נוסף ה-tooltip.
  - text- הטקסט שמוצג ב-tooltip.
  - tooltip- חלונית שמציגה את הטקסט.
- פעולות המחלקה:
  - __init__()
    טענת כניסה: ווידג'ט (widget) וטקסט לתצוגה (text).
    טענת יציאה: מחבר את ה-tooltip לווידג'ט עם מאזינים לאירועי <Enter> ו־<Leave>.

- update_text()
  טענת כניסה: המחרוזת החדשה (new_text).
  טענת יציאה: מעדכן את הטקסט ומסתיר את ה-tooltip אם הוא פתוח.
- show_tooltip()
  טענת כניסה: אירוע- ברירת מחדל: ללא (event).
  טענת יציאה: יוצר tooltip חדש מעל הווידג'ט וממקם אותו במרכז רוחב הווידג'ט.
- hide_tooltip()
  טענת כניסה: אירוע- ברירת מחדל: ללא (event).
  טענת יציאה: הורס את חלון ה-tooltip אם קיים.

❖ מחלקה: InfoButtonToolTip (יורש מ-ToolTip)

- תפקיד המחלקה: tooltip מותאם להצגת מידע כללי (IP, פורט, מספר חיבורים).
- פעולות ייחודיות:
  - show_tooltip()
    טענת כניסה: אירוע- ברירת מחדל: ללא (event).
    טענת יציאה: מציג את ה-tooltip בצד שמאל־למעלה של הווידג'ט.
- שאר התכונות והפעולות: כמו במחלקת האב ToolTip.

❖ מחלקה: Client

- תפקיד המחלקה: מנהלת את תקשורת הלקוח מול השרת – חיבור, החלפת מפתחות, קבלת פקודות, שליחת שיתוף מסך וביצוע פעולות מקומיות.
- תכונות המחלקה:
  - client- חיבור TCP לתקשורת עם השרת.
  - address- כתובת ה-IP של השרת.
  - port- ה-Port של השרת.
  - disconnect_callback- פונקציה שתיקרא בעת ניתוק.
  - _disconnected- דגל למניעת ניתוק כפול.
  - _lock -מנעול למניעת ניתוק כפול.
  - private_key- מפתח RSA פרטי של הלקוח להצפנה אסימטרית.
  - public_key- מפתח RSA ציבורי של הלקוח להצפנה אסימטרית.
  - server_rsa_key- מפתח RSA ציבורי של השרת להצפנה אסימטרית.
  - server_aes_key- מפתח AES של השרת להצפנה סימטרית של נתונים.
  - pause_event- דגל להפסקה זמנית של שיתוף המסך.
  - screen_share- מופע של מחלקת שיתוף מסך.
  - img_size- גודל התמונה לשיתוף.
  - block_event_lock- מנעול לחסימת משתמש.
  - block_event- דגל לחסימת משתמש.
  - blocker- אובייקט לחסימת המשתמש.
  - input_controller- אובייקט לטיפול בהזנות מקשי מקלדת ועכבר.
  - stop_event- דגל לעצירת הפעולה.
- פעולות המחלקה:
  - __init__()
    טענת כניסה: כתובת IP של השרת (address), ה-Port של השרת (port) ופונקציה שתיקרא בעת ניתוק- ברירת מחדל: ללא (disconnect_callback).
    טענת יציאה: מאתחל את כל רכיבי הלקוח (תקשורת, הצפנה, חסימה, שליטה וכו').

- run()
  טענת כניסה: ללא.
  טענת יציאה: מתחבר לשרת, מבצע החלפת מפתחות, מקשיב לפקודות מהשרת, מטפל בהן (שליטה, חסימה, שינוי גודל, בעיטה וכו') ומפעיל שיתוף מסך.
- key_exchange()
  טענת כניסה: ללא.
  טענת יציאה: מבצע החלפת מפתחות עם השרת לצורך תקשורת מוצפנת.
- stop(stop_reason)
  טענת כניסה: מחרוזת סיבת עצירה (stop_reason).
  טענת יציאה: עוצר את תהליך הלקוח בצורה בטוחה, כולל סגירת שיתוף מסך וניתוק החיבור.

❖ מחלקה: ScreenShare

- תפקיד המחלקה: תהליך שמצלם את מסך המשתמש, מכווץ, מצפין ושולח לשרת תמונות באופן רציף.
- תכונות המחלקה:
  - client- חיבור TCP לתקשורת עם השרת.
  - aes_key- מפתח AES של השרת להצפנה סימטרית של נתונים.
  - img_size- גודל התמונה לשיתוף.
  - stop_event- דגל לעצירת הפעולה.
  - pause_event- דגל להפסקה זמנית של שיתוף המסך.
  - _lock- מנעול לשינוי גודל התמונה.
- פעולות המחלקה:
  - __init__()
    טענת כניסה: החיבור אל השרת (client), מפתח ה-AES להצפנה (aes_key) ודגל להפסקה זמנית של שיתוף המסך (pause_event).
    טענת יציאה: מאתחל את תהליך שיתוף המסך.
  - run()
    טענת כניסה: ללא.
    טענת יציאה: רץ בלולאה- עושה צילום מסך, מכווץ, מצפין ושולח לשרת תוך שליטה בקצב (FPS) והשהיה לפי הצורך.
  - update_size()
    טענת כניסה: גודל תמונה חדש (new_size).
    טענת יציאה: מעדכן את הגודל שבו תתבצע צילום המסך.
  - stop()
    טענת כניסה: ללא.
    טענת יציאה: מסמן לתהליך להפסיק לפעול.

❖ מחלקה: ClientGui

- תפקיד המחלקה: מציגה ממשק גרפי בצד הלקוח שמודיע למשתמש שהמסך משותף עם כתובת השרת.
- תכונות המחלקה:
  - root- אובייקט שורש של tkinter.
  - address- כתובת הלקוח.
  - label- תווית המציגה למשתמש את כתובת השרת שאליו משותף המסך.
- פעולות המחלקה:
  - __init__()
    טענת כניסה: אובייקט שורש (root) וסדורת כתובת הלקוח (address).

טענת יציאה : יוצר ממשק גרפי בסיסי עם תווית המציגה את כתובת השרת, מקבע גודל, רקע ואירוע סגירה.

- update_geometry()
  טענת כניסה : ללא.
  טענת יציאה : מחשב את רוחב התווית ומעדכן את גודל החלון בהתאם, עם רווח מסוים.
- on_close()
  טענת כניסה : ללא.
  טענת יציאה : לא סוגר את הממשק, ובכך את הלקוח.
- run()
  טענת כניסה : ללא.
  טענת יציאה : מריץ את הלולאה הראשית של המשק הגרפי.

## חלק ב'

❖ **אלגוריתם לקיחת ושליחת תמונות (קטעי קוד רלוונטיים):**

```
def take_screenshot(x_size=1920, y_size=1080, quality=75):
    """
    Capture a screenshot of the screen resized to given dimensions
and compressed as JPEG.

    :param x_size: Width of the resized screenshot.
    :param y_size: Height of the resized screenshot.
    :param quality: JPEG quality (0-100).
    :return: JPEG image bytes.
    """
    # Capture the entire screen
    screenshot = ImageGrab.grab()

    # Resize screenshot to target resolution
    screenshot = screenshot.resize((x_size, y_size))

    # Save screenshot to in-memory byte stream in JPEG format with
specified quality
    byte_io = BytesIO()
    screenshot.save(byte_io, format='JPEG', quality=quality)
    byte_io.seek(0)

    # Return the raw bytes of the JPEG image
    return byte_io.getvalue()


class ScreenShare(threading.Thread):
    """
    The ScreenShare class handles periodic screen capturing,
compression,
    encryption, and transmission to the server.
    """

    def __init__(self, client, aes_key, pause_event):
        """
        Initialize the screen sharing thread.

        :param client: Socket used for sending screen data.
        :type client: socket.socket
        :param aes_key: AES key for encrypting screen data.
```

```
        :type aes_key: bytes
        :param pause_event: Event to pause/resume screen sharing.
        :type pause_event: threading.Event
        """
        super().__init__()
        self.client = client
        self.aes_key = aes_key
        self.img_size = (1920, 1080)
        self.stop_event = threading.Event()
        self.pause_event = pause_event
        self._lock = threading.Lock()

    def run(self):
        """
        Continuously capture, compress, encrypt, and send screen
frames to the server.
        """
        while not self.stop_event.is_set():
            if self.pause_event.is_set():
                time.sleep(0.5)
                continue

            try:
                print("[Screen Share] Taking screenshot...")
                start = time.perf_counter()

                with self._lock:
                    width, height = self.img_size

                # Capture screen image
                image_bytes = take_screenshot(width, height,
quality=75)
                if not image_bytes:
                    print("[Screen Share] Screenshot failed or
returned empty")
                    time.sleep(0.5)
                    continue

                # Compress image data
                compressed_bytes = zlib.compress(image_bytes,
level=6)
                if not compressed_bytes:
                    print("[Screen Share] Compression failed")
                    time.sleep(0.5)
                    continue

                # Encrypt the compressed data
                iv, encrypted = Encryption.encrypt_aes(self.aes_key,
compressed_bytes)
                if not encrypted:
                    print("[Screen Share] Encryption failed")
                    time.sleep(0.5)
                    continue

                # Send data to server
                self.client.sendall(len(encrypted).to_bytes(8,
'big'))
                self.client.sendall(iv)
                self.client.sendall(encrypted)

                print(f"[Screen Share] Frame sent |
```

```
raw={len(image_bytes)} | compressed={len(compressed_bytes)} |
encrypted={len(encrypted)}")

                # Maintain frame rate
                elapsed = time.perf_counter() - start
                time.sleep(max(0, (1 / FPS) - elapsed))

            except Exception as e:
                print(f"[Screen Share] Error: {e}")
                time.sleep(1)

    def update_size(self, new_size):
        """
        Update the screen capture resolution.

        :param new_size: New resolution as (width, height).
        :type new_size: tuple
        """
        with self._lock:
            print(f"[Screen Share] Updating image size to:
{new_size}")
            self.img_size = new_size
```

❖ **אלגוריתם קבלת ועדכון תמונות (קטעי קוד רלוונטיים):**

```
def recv_all(sock, num_bytes):
    """
    Receive an exact number of bytes from a socket, handling partial
receives.

    :param sock: Socket object to receive from.
    :param num_bytes: Number of bytes to receive.
    :return: Received bytes.
    :raises ConnectionError: If the connection is lost before all bytes are
received.
    """
    data = b''
    while len(data) < num_bytes:
        # Receive the remaining number of bytes or MAX_CHUNK_SIZE,
whichever is smaller
        packet = sock.recv(min(MAX_CHUNK_SIZE, num_bytes - len(data)))
        if not packet:
            # Connection lost unexpectedly
            raise ConnectionError("Connection lost while receiving data")
        data += packet
    return data


def run(self):
    """
    Main loop that consumes frames and triggers screen updates if frames
are fresh.
    """
    while not self.stop_event.is_set():
        try:
            # Wait for a new frame from any client, timeout after 1 second
            client_address, frame, timestamp =
self.frame_queue.get(timeout=1)
            # Only update screen if frame is fresh enough
            if time.time() - timestamp <= self.max_frame_age:
                # Update screen only if no fullscreen widget or fullscreen
```

```
client matches frame client
                if not self.app.fullscreen_widget or
self.app.fullscreen_address == client_address:
                    self.app.update_screen(client_address, frame)
            else:
                print(f"[FrameConsumer] Dropped stale frame from
{client_address}")
        except queue.Empty:
            continue  # No frame available, continue loop


def run(self):
    """
    Main loop that receives encrypted and compressed image frames from the
client,
    decrypts and decompresses them, and puts them in the frame queue.
    """
    try:
        while not self.stop_event.is_set():
            current_time = time.time()
            elapsed = current_time - self.last_frame_time

            # Sleep if frames are arriving too fast (enforce FPS limit)
            if elapsed < self.min_frame_interval:
                time.sleep(self.min_frame_interval - elapsed)

            # Receive size of incoming frame (8 bytes)
            image_size = int.from_bytes(recv_all(self.client_socket, 8),
byteorder='big')
            if image_size == 0:
                raise ConnectionError("Received empty frame size")

            # Receive AES IV and encrypted image bytes
            iv = recv_all(self.client_socket, 16)
            encrypted_img = recv_all(self.client_socket, image_size)

            # Decrypt and decompress the frame image data
            frame = Encryption.decrypt_aes(self.aes_key, iv, encrypted_img)
            frame = zlib.decompress(frame)

            timestamp = time.time()
            # Put the frame into the shared queue with client address and
timestamp
            self.frame_queue.put((self.client_address, frame, timestamp))
            self.last_frame_time = timestamp


def _add_screen_on_main_thread(self, address, preview_photo):
    """
    Add a new client screen preview button to the GUI on the main thread.

    :param address: Client's network address.
    :type address: str
    :param preview_photo: Tkinter PhotoImage preview of the client's
screen.
    :type preview_photo: ImageTk.PhotoImage
    """
    print(f"[GUI] Added a new screen")
    button = tk.Button(self.root, image=preview_photo,
background="#232333",
                       command=lambda: self.toggle_fullscreen(address,
```

```
b""))
    button.image = preview_photo
    button.image_id = id(preview_photo)  # Track image to avoid redundant
updates
    self.buttons[address] = button
    self.images.append(preview_photo)  # Keep a reference to avoid garbage
collection

    info = self.get_info_text()
    self.info_label.tooltip.update_text(info)  # Update tooltip info

    # Organize screens and notify clients of new resize if no fullscreen is
active
    if not self.fullscreen_widget:
        self.organize_screens()
        self.command_queue.put((None,
f"resize:{self.button_size[0]}:{self.button_size[1]}"))
def update_screen(self, address, image_bytes):
    """
    Receive updated screen image bytes from a client and schedule GUI
update.

    :param address: Client's network address.
    :type address: str
    :param image_bytes: Raw image data bytes of the client's screen.
    :type image_bytes: bytes
    """
    if self.shutdown_event.is_set():
        return  # Ignore updates if server is shutting down

    if address in self.kicked_addresses:
        print(f"[GUI] Ignoring frame update from kicked / disconnected
client {address}")
        return

    print(f"[GUI] Updating image for {address}")
    try:
        image = Image.open(BytesIO(image_bytes))

        preview_photo = None
        fullscreen_photo = None

        if self.fullscreen_address == address:
            # Resize fullscreen image if needed
            if image.size != self.fullscreen_size:
                fullscreen_image = image.resize(self.fullscreen_size)
                self.command_queue.put((address,
f"resize:{self.fullscreen_size[0]}:{self.fullscreen_size[1]}"))
            else:
                fullscreen_image = image
            fullscreen_photo = ImageTk.PhotoImage(fullscreen_image)
        else:
            # Resize preview image if needed
            if image.size != self.button_size:
                preview_image = image.resize(self.button_size)
                self.command_queue.put((address,
f"resize:{self.button_size[0]}:{self.button_size[1]}"))
            else:
                preview_image = image
            preview_photo = ImageTk.PhotoImage(preview_image)
```

```
        # Schedule GUI update on the main thread
        self.root.after(0, lambda:
self._update_screen_on_main_thread(address, preview_photo,
fullscreen_photo))
    except Exception as e:
        print(f"[GUI] Error processing image from {address}: {e}")
def _update_screen_on_main_thread(self, address, preview_photo,
fullscreen_photo):
    """
    Update the GUI components for a client's screen on the main thread.

    :param address: Client's network address.
    :type address: str
    :param preview_photo: Tkinter PhotoImage for the preview button, or
None.
    :type preview_photo: ImageTk.PhotoImage or None
    :param fullscreen_photo: Tkinter PhotoImage for fullscreen display, or
None.
    :type fullscreen_photo: ImageTk.PhotoImage or None
    """
    if self.shutdown_event.is_set():
        return  # Prevent updates after shutdown

    if address in self.kicked_addresses:
        print(f"[GUI] Skipping update for kicked / disconnected client
{address}")
        return

    if fullscreen_photo is not None:
        # Update fullscreen widget if active and address matches
        if self.fullscreen_widget is not None and self.fullscreen_address
== address:
            if getattr(self.fullscreen_widget, "image_id", None) !=
id(fullscreen_photo):
                self.fullscreen_widget.config(image=fullscreen_photo)
                self.fullscreen_widget.image = fullscreen_photo
                self.fullscreen_widget.image_id = id(fullscreen_photo)
        return  # No button update needed in fullscreen mode

    if preview_photo is not None:
        if address in self.buttons:
            button = self.buttons[address]

            if getattr(button, "image_id", None) != id(preview_photo):
                button.config(image=preview_photo)
                button.image = preview_photo
                button.image_id = id(preview_photo)
        else:
            # Add new preview button if not found
            self._add_screen_on_main_thread(address, preview_photo)
```

❖ **אלגוריתם קבלת וביצוע פקודות שליטה מרחוק וחסימת לקוח (קטעי קוד רלוונטיים):**

```
def recv_all(sock, num_bytes):
    """
    Receive an exact number of bytes from a socket, handling partial
receives.
    :param sock: Socket object to receive from.
    :param num_bytes: Number of bytes to receive.
    :return: Received bytes.
    :raises ConnectionError: If the connection is lost before all bytes are
```

```
received.
    """
    data = b''
    while len(data) < num_bytes:
        # Receive the remaining number of bytes or MAX_CHUNK_SIZE,
whichever is smaller
        packet = sock.recv(min(MAX_CHUNK_SIZE, num_bytes - len(data)))
        if not packet:
            # Connection lost unexpectedly
            raise ConnectionError("Connection lost while receiving data")
        data += packet
    return data


def run(self):
    """
    Run the client thread to connect to the server, handle commands,
    and manage screen sharing.
    """
    stop_reason = "client crashed"
    try:
        self.client.connect((self.address, self.port))
        print(f"[Client] Connected to server at
{self.address}:{self.port}")
        self.key_exchange()

        self.screen_share = ScreenShare(self.client, self.server_aes_key,
self.pause_event)
        self.screen_share.start()

        while not self.stop_event.is_set():
            try:
                # Receive command length and actual command (AES encrypted)
                cmd_len_bytes = recv_all(self.client, 4)
                cmd_len = int.from_bytes(cmd_len_bytes, 'big')

                iv = recv_all(self.client, 16)
                encrypted_cmd = recv_all(self.client, cmd_len)
                command = Encryption.decrypt_aes(self.server_aes_key, iv,
encrypted_cmd).decode()

                # Handle input-related commands
                if command.startswith(("key_down", "key_up", "button")):
                    self.input_controller.handle_command(command)

                # Handle resize command
                elif command.startswith("resize"):
                    try:
                        _, width, height = command.split(":")
                        width, height = int(width), int(height)
                        print(f"[Client] Resizing to: {width}x{height}")
                        self.img_size = (width, height)
                        if self.screen_share:
                            self.screen_share.update_size(self.img_size)
                    except ValueError as e:
                        print(f"[Client] Invalid resize command: {command}
({e})")

                # Handle session termination command
                elif command == "kick":
                    stop_reason = "kicked from session"
```

```
                    print("[Client] Received kick command.")
                    break

                # Block/unblock user input
                if command == "block":
                    with self.block_event_lock:
                        self.block_event.set()
                    self.blocker.start_blocking()
                    print("Blocking: True")
                elif command == "unblock":
                    with self.block_event_lock:
                        self.block_event.clear()
                    self.blocker.stop_blocking()
                    print("Blocking: False")

                # Pause/unpause screen sharing
                elif command == "pause":
                    self.pause_event.set()
                    print("Paused: True")
                elif command == "unpause":
                    self.pause_event.clear()
                    print("Paused: False")


class UserBlocker:
    """
    Class to block all user input (keyboard and mouse) by suppressing
events.
    """

    def __init__(self):
        """
        Initialize UserBlocker with no active listeners.
        """
        self.keyboard_listener = None
        self.mouse_listener = None

    def start_blocking(self):
        """
        Start blocking user input by suppressing keyboard and mouse events.
        This creates and starts listeners that prevent any input from
reaching other apps.
        """
        if not self.keyboard_listener or not self.mouse_listener:
            # Suppress=True means events are blocked/suppressed system-wide
            self.keyboard_listener = keyboard.Listener(suppress=True)
            self.mouse_listener = mouse.Listener(suppress=True)
            self.keyboard_listener.start()
            self.mouse_listener.start()

    def stop_blocking(self):
        """
        Stop blocking user input by stopping the keyboard and mouse
listeners.
        """
        if self.keyboard_listener and self.keyboard_listener.running:
            self.keyboard_listener.stop()
        if self.mouse_listener and self.mouse_listener.running:
            self.mouse_listener.stop()
        self.keyboard_listener = None
        self.mouse_listener = None
```

```
class InputController:
    """
    Handles injecting mouse and keyboard inputs programmatically.
    """

    def __init__(self, block_event, block_event_lock, user_blocker):
        """
        Initialize the input controller.

        :param block_event: threading.Event used to track if input blocking
is active.
        :param block_event_lock: threading.Lock protecting access to
block_event.
        :param user_blocker: Instance of UserBlocker to start/stop blocking
when needed.
        """
        self.block_event = block_event
        self.block_event_lock = block_event_lock
        self.user_blocker = user_blocker
        self.mouse = mouse.Controller()
        self.keyboard = keyboard.Controller()

    def set_mouse_pos(self, x_pos, y_pos):
        """
        Set the mouse cursor position on screen.

        :param x_pos: X coordinate.
        :param y_pos: Y coordinate.
        """
        self.mouse.position = (x_pos, y_pos)

    def handle_command(self, command):
        """
        Handle an input command string and execute mouse or keyboard
actions.

        :param command: Command string, e.g. "button:1:100:200",
"key_down:Shift_L".
        """
        try:
            with self.block_event_lock:
                was_blocking = self.block_event.is_set()
                if was_blocking:
                    # Temporarily stop blocking so injected inputs are
accepted by the system
                    self.user_blocker.stop_blocking()

                sliced_command = command.split(":")
                if sliced_command[0] == "button":
                    # Mouse button click command:
button:<button_num>:<x>:<y>
                    button_num = sliced_command[1]
                    x = int(sliced_command[2])
                    y = int(sliced_command[3])
                    button = MOUSE_BUTTON_MAP.get(button_num)
                    if button:
                        self.set_mouse_pos(x, y)
                        self.mouse.click(button)
                    else:
```

```
                        print(f"[InputController] Unknown mouse button: {button_num}")

                elif sliced_command[0] == "key_down":
                    # Key press command
                    self._press_key(sliced_command[1])

                elif sliced_command[0] == "key_up":
                    # Key release command
                    self._release_key(sliced_command[1])

                if was_blocking:
                    # Restart blocking after input injection to continue suppressing real user input
                    self.user_blocker.start_blocking()

        except Exception as e:
            print(f"[InputController] Error handling command '{command}': {e}")

    def _press_key(self, key):
        """
        Press a keyboard key.

        :param key: String key name to press.
        """
        if key in KEYBOARD_SPECIAL_KEYS_MAP:
            pynput_key = KEYBOARD_SPECIAL_KEYS_MAP[key]
            self.keyboard.press(pynput_key)
        else:
            try:
                self.keyboard.press(key)
            except ValueError:
                print(f"[InputController] Invalid key press: {key}")

    def _release_key(self, key):
        """
        Release a keyboard key.

        :param key: String key name to release.
        """
        if key in KEYBOARD_SPECIAL_KEYS_MAP:
            pynput_key = KEYBOARD_SPECIAL_KEYS_MAP[key]
            self.keyboard.release(pynput_key)
        else:
            try:
                self.keyboard.release(key)
            except ValueError:
                print(f"[InputController] Invalid key release: {key}")
```

❖ **אלגוריתם סידור מסכים (קטעי קוד רלוונטיים):**

```
def organize_screens(self):
    """
    Organize and place client screen buttons dynamically based on the number of clients
    and available screen space, avoiding overlap with the info icon.
    """
    total_buttons = len(self.buttons)
    print(f"[GUI] Organizing {total_buttons} screens")
```

```
    if total_buttons == 0:
        self.show_no_clients_message()
        return

    if self.no_clients_label:
        self.no_clients_label.destroy()
        self.no_clients_label = None

    for button in self.buttons.values():
        button.place_forget()

    # Determine available space
    usable_width = self.screen_width
    usable_height = self.screen_height
    margin = 20

    if self.info_label:
        info_x = self.info_label.winfo_x()
        info_y = self.info_label.winfo_y()
        info_width = self.info_label.winfo_width()
        info_height = self.info_label.winfo_height()

        if info_y + info_height + margin > self.screen_height -
BUTTON_SIZE:
            usable_height = info_y - margin

        if info_x + info_width + margin > self.screen_width -
BUTTON_SIZE:
            usable_width = info_x - margin

    # Layout computation
    num_buttons_per_row = math.ceil(math.sqrt(total_buttons))
    number_of_rows = math.ceil(total_buttons / num_buttons_per_row)
    num_buttons_in_last_row = total_buttons % num_buttons_per_row or
num_buttons_per_row

    if num_buttons_per_row == number_of_rows:
        button_height = math.floor(usable_height / number_of_rows)
        button_width = math.floor(RATIO * button_height)
    else:
        button_width = math.floor(usable_width / num_buttons_per_row)
        button_height = math.floor(1 / RATIO * button_width)

    current_row = 0
    current_column = 0
    self.button_size = (button_width, button_height)

    for address, button in self.buttons.items():
        if current_row == number_of_rows - 1:
            row_buttons = num_buttons_in_last_row
        else:
            row_buttons = num_buttons_per_row

        x_offset = (self.screen_width - (button_width * row_buttons))
// 2
        y_offset = (usable_height - (button_height * number_of_rows))
// 2

        x_position = x_offset + (current_column * button_width)
        y_position = y_offset + (current_row * button_height)
```

```
        button.place(x=x_position, y=y_position, width=button_width,
height=button_height)

        current_column += 1
        if current_column >= num_buttons_per_row:
            current_column = 0
            current_row += 1
```

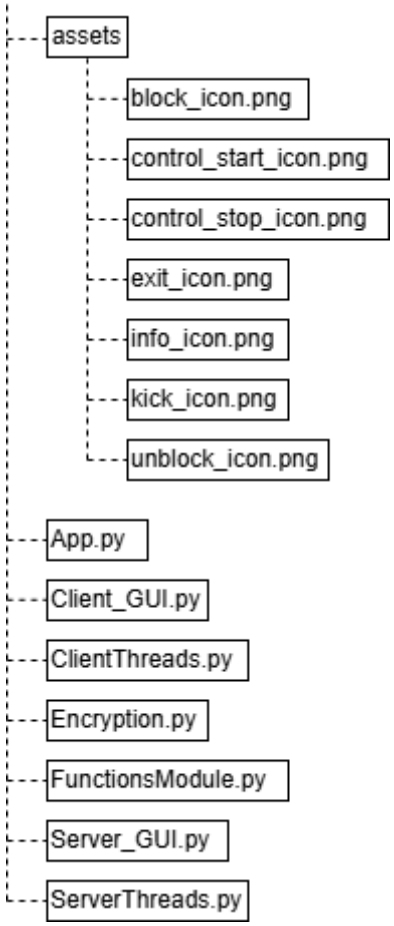
## חלק ג'

| שם הבדיקה: | מטרת הבדיקה: | מה בוצע בפועל: | תוצאות הבדיקה: | פתרון: |
|---|---|---|---|---|
| אי קריסת לקוח ושרת | לבדוק שהלקוח והשרת אינם קורס מסיבה כלשהי. | הפעלתי שרת, התחברתי אליו עם לקוח, והשתמשתי בכל היכולות שסופקו על ידי השרת, כולל סגירה פתאומית של השרת או הלקוח. | הכל עבד כמצופה. | אין צורך |
| בדיקת חסימת לקוח | לבדוק ש-"חסימת לקוח" פועלת כפי שמצופה. | הפעלתי שרת, התחברתי אליו עם לקוח, הפעלתי "חסימת לקוח" דרך השרת, ובדקתי שכל הכפתורים והמקשים אצל הלקוח היו חסומים על ידי לחיצתם | הכל עבד כמצופה. | אין צורך |
| בדיקת שליטה מרחוק וחסימת לקוח | לבדוק שה-"שליטה מרחוק" עובדת כפי שמצופה, גם אם קיימת "חסימת לקוח" פועלת. | הפעלתי שרת, התחברתי אליו עם לקוח, הפעלתי שליטה מרחוק, ניסיתי לפתוח קובץ ולכתוב בו, לאחר מכן הפעלתי "חסימת לקוח" וניסיתי לפתוח קובץ נוסף ולכתוב בו, ובדקתי שכל הכפתורים והמקשים היו חסומים. אצל הלקוח חסומים על ידי לחיצתם. | הכל עבד כמצופה. | אין צורך |
| בדיקת התנתקות מסודרת | לבדוק שכאשר לקוח מתנתק מהשרת, השרת מטפל בניקוי המשאבים הקשורים ללקוח, והלקוח מתנתק בצורה מסודרת. | הוספתי בקוד הדפסות סטטוס ניתוק לקוח, הפעלתי שרת, התחברתי אליו עם לקוח, ניתקתי את הלקוח דרך השרת, ולאחר מכן צפיתי בהודעות הסטטוס שהודפסו ב-"cmd" כדי לוודא שכל המשאבים נמחקו ונוקו מהלקוח. | הכל עבד כמצופה. | אין צורך |
| בדיקת פעולת הפצנה | לבדוק שההצפנה באמת עובדת. | הוספתי בקוד הדפסת הודעות לאחר הצפנה, הפעלתי שרת, התחברתי אליו עם לקוח, הפעלתי את כל הפקודות של השרת, ולאחר מכן צפיתי בהודעות שהודפסו ב-"cmd" כדי לוודא שכל | הכל עבד כמצופה. | אין צורך |

| | | | | |
|---|---|---|---|---|
| | | הפקודות ושיתוף המסך מהלקוח מוצפנים. | | |
| בדיקת הפעלה מחודשת של אפליקציית הפעלה | לבדוק שכאשר לקוח או שרת מתנתק או נסגר על ידי המשתמש, אפליקציית הפעלה מופעלת מחדש, תוך ניקוי משאביישנים. | הוספתי בקוד הדפסות סטטוס סגירת לקוח ושרת, הפעלתי שרת, התחברתי אליו עם לקוח, ניתקתי את הלקוח דרך השרת, התחברתי שוב עם הלקוח, סגרתי את הלקוח, התחברתי שוב פעם נוספת עם הלקוח, ולאחר מכן סגרתי את השרת. בכל אחת מהפעמים נפתחה אפליקציית ההפעלה, וצפיתי בהודעות סטטוס הסגירה שהודפסו ב-"cmd". | לא הציג את סיבות ההפעלה מחדש הנכונות. | הייתה טעות במיקום קביעת סיבת ההפעלה מחדש. |

# מדריך למשתמש

## פירוט כלל קבצי המערכת- עץ קבצים

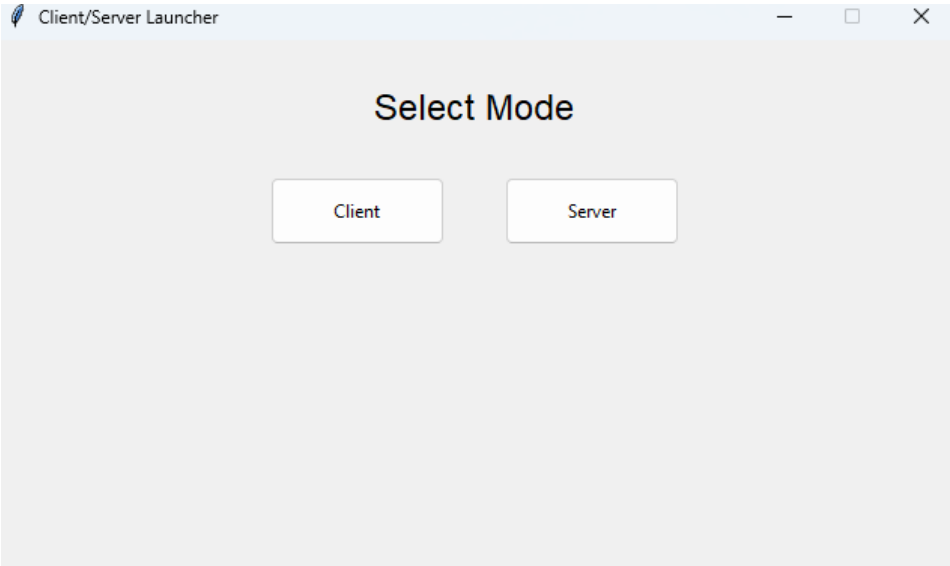



## התקנת המערכת

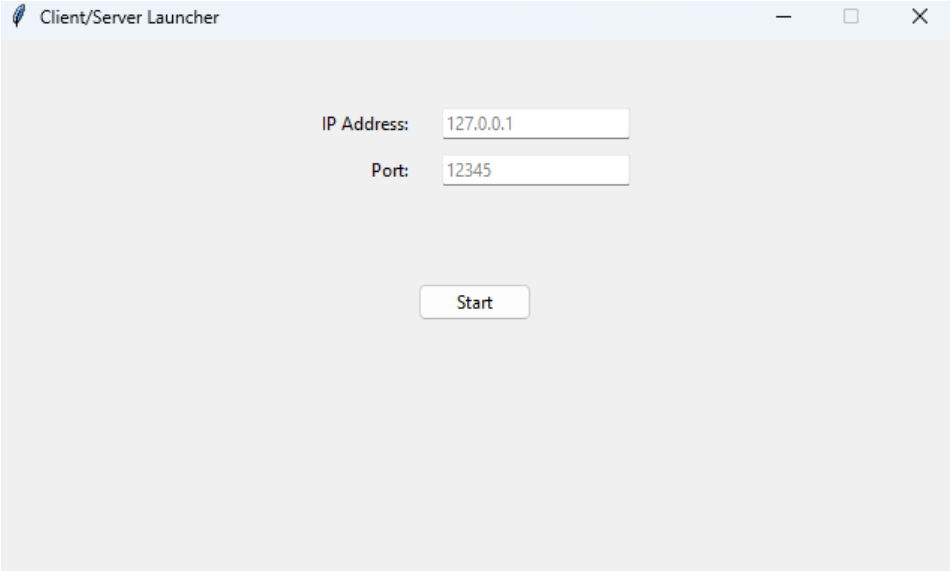
כדי להשתמש במערכת, יש להוריד "Python 3.13" ואת הספריות "PIL" (אשר מורד בתור "pillow"), "cryptography" ו-"pynput". יש לסדר את הקבצים כפי שמתואר בעץ הקבצים. יש להיות מחובר עם עכבר ומקלדת, ובנוסף מחובר לרשת מקומית.

## מדריך לתלמיד:

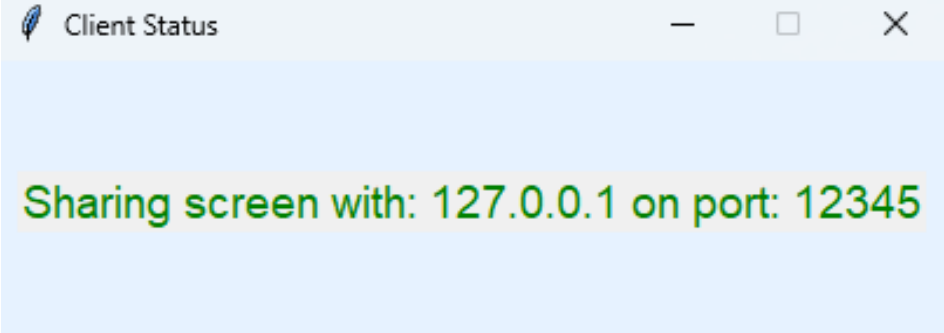
על התלמיד להפעיל את "App.py" (עדיף לפתוח כ-"pyw.", כדי שלא יהיה ניתן לראות את ה-"cmd" של "Windows"), מה שיוביל לפתיחת החלון עם המסך הבא:




על התלמיד ללחוץ על הכפתור "Client", מה שיוביל אותו למסך הבא:



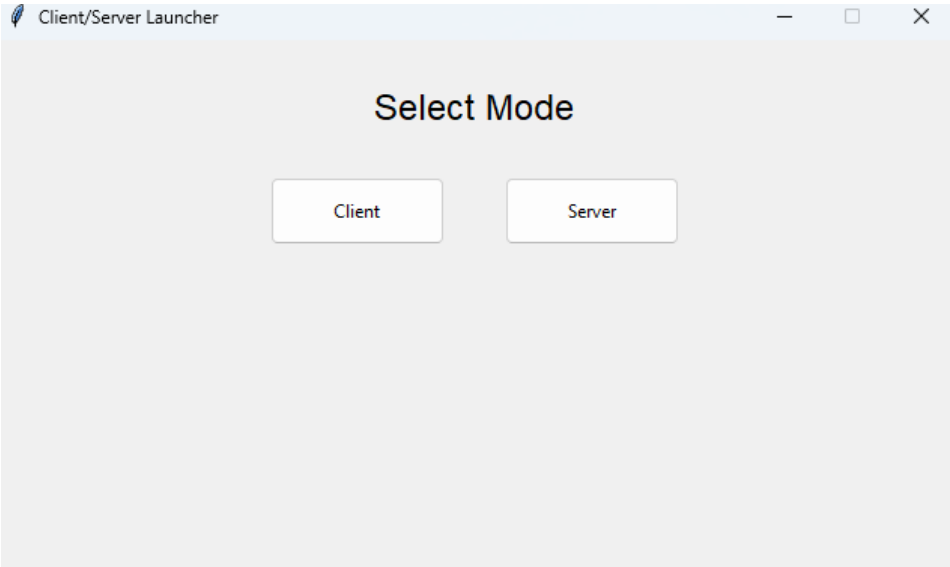

על התלמיד להזין את כתובת ה-IP ואת ה-Port של שרת המורה אליו רוצה להתחבר.
התחברות מוצלחת תוביל לסגירת החלון הנוכחי ופתיחת החלון עם המסך הבא:



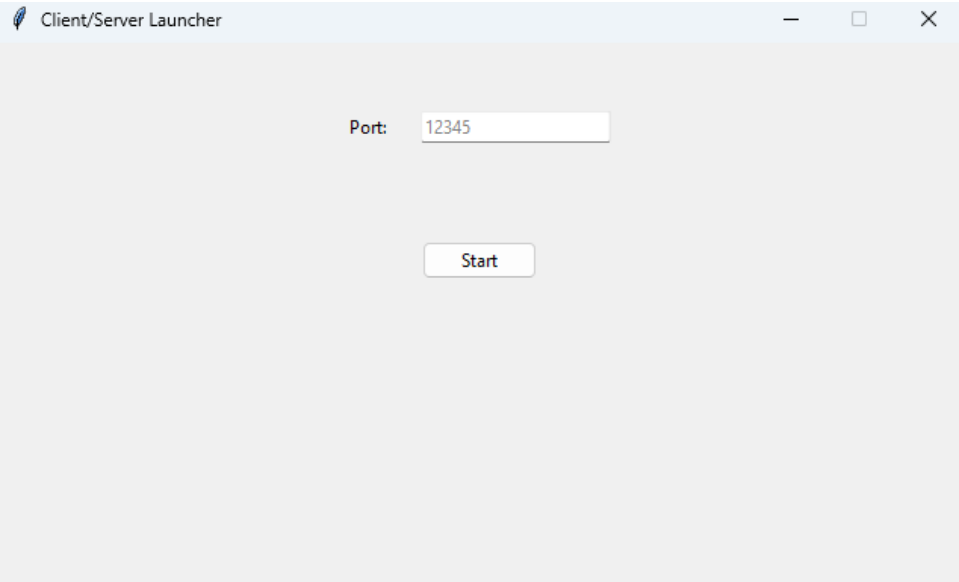
לאחר הופעת מסך זה, יכול התלמיד להתחיל במבחנו / משימותיו.
במקרה והתלמיד הזין כתובת לא תקינה, או שהשרת לא יכל לחבר אותו, יקבל התלמיד הודעת שגיאה, ויהיה עליו להזין כתובת IP ו-Port של שרת מורה אחר.
במידה והתלמיד נותק משרת המורה, או שהשרת נסגר, ייסגר החלון, וייפתח המסך הראשון ביחד עם סיבת ההפעלה מחדש של התוכנית ("נותק" / "הפגישה נסגרה").

## מדריך למורה:

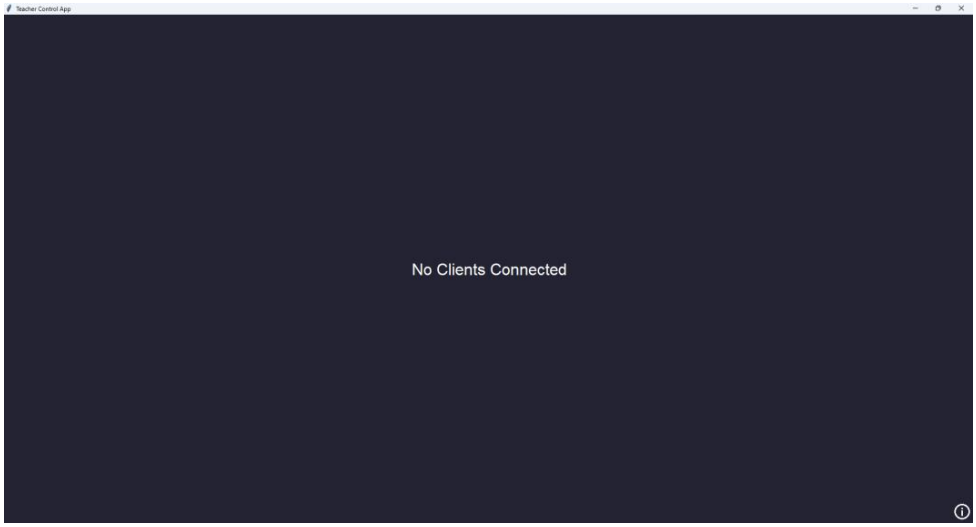
על המורה להפעיל את "App.py" (עדיף לפתוח כ-"pyw."), כדי שלא יהיה ניתן לראות את ה-"cmd" של "Windows"), מה שיוביל לפתיחת החלון עם המסך הבא:



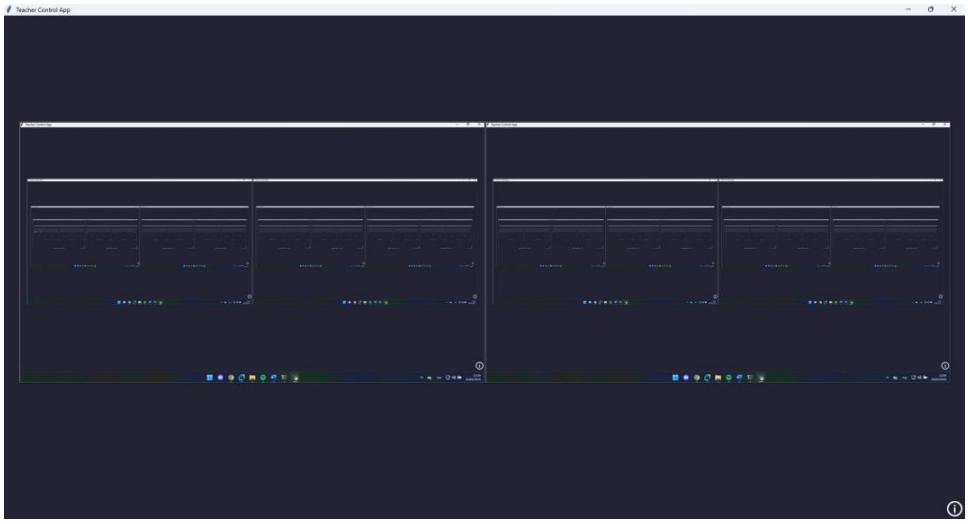

על המורה ללחוץ על הכפתור "Server", מה שיוביל אותו למסך הבא:



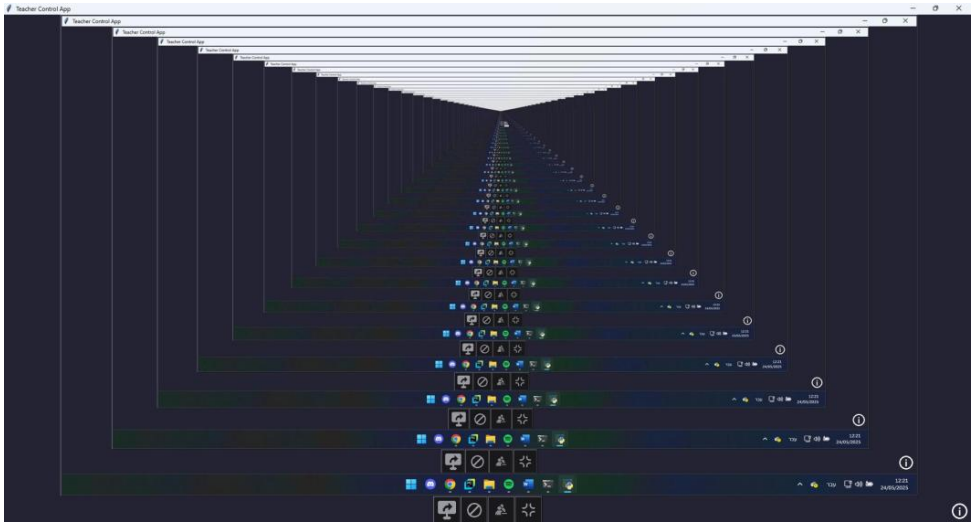

על המורה להזין את ה-Port אליו יאזין השרת שלו. במידה וה-Port לא פנוי, תופיע הודעת שגיאה ויהיה על המורה להזין Port אחר. הקמת שרת מוצלחת תוביל לסגירת החלון הנוכחי ופתיחת החלון עם המסך הבא:




על המורה לרחף עם עכברו מעל סמל המידע מימין למטה אשר יופיע במיקום זה בכל עת, ודרכו יוכל לראות את כתובת ה-IP וה-Port של השרת שלו. כמו כן, יוכל המורה לראות את מספר התלמידים המחוברים אליו. עליו יהיה ליידע את התלמידים מה היא כתובת ה-IP וה-Port של השרת, כך שיוכלו להתחבר, לאחר חיבור של מספר תלמידים (במקרה הבא- 2 תלמידים), ייראה מסכו כך (במקרה הבא שני לקוחות מחוברים מהמחשב עליו נמצא השרת):




לחיצה על אחד המסכים תוביל לפתיחת המסך הבא:



במסך למעלה, יש 4 כפתורים חדשים למטה באמצע. הכפתורים (משמאל לימין) עושים את הדברים הבאים:

- הפעלת השליטה מרחוק, דבר שיגרום לכל לחיצה על מקשי המקלדת והעכבר במסך התלמיד (יש ללחוץ בתוך מסך התלמיד המוגדל) לקרות גם אצל התלמיד. כדי לכבות את השליטה מרחוק יש ללחוץ על הכפתור שוב פעם.
- הפעלת חסימת התלמיד, דבר שיגרום לחסימת העכבר והמקלדת של התלמיד. כדי לכבות את חסימת התלמיד, יש ללחוץ על הכפתור שוב פעם.
  הפעלת חסימת התלמיד, לא תחסום את השליטה מרחוק (יהיה ניתן עדיין לשלוט מרחוק, אפילו שהתלמיד לא יכול להשתמש בעכבר ובמקלדת שלו).
- ניתוק התלמיד ויציאה ממסך מלא חזרה למסך הקודם.
- יציאה ממסך מלא חזרה למסך הקודם.

במידה והמורה סגר את השרת, כל התלמידים ינותקו, ייסגר החלון, וייפתח המסך הראשון ביחד עם סיבת ההפעלה מחדש של התוכנית ("הפגישה נסגרה").

שם פרויקט: "שליטת בוחן"
שם תלמיד: רון טקץ'- 330801853

# סיכום אישי / רפלקציה

אחרי כל הזמן הזה של כתיבת הקוד ותיק הפרויקט, להגיע לנקודה הזאת, נקודה שבה צריך פשוט לשבת ולכתוב על מה שעברתי, זו תחושה קצת מוזרה. הפרויקט הזה הוא הפרויקט הכי ארוך ומשמעותי שעשיתי בבית הספר, ועכשיו, כשאני מסתכל על כולו ומבין שאני בשלבי הסיום, אני מרגיש בעיקר גאווה.

התחלתי לחשוב על הפרויקט הזה בתחילת פברואר, וכבר בסוף החודש התחלתי לכתוב את הקוד עבורו. מההתחלה הבנתי שתכנון נכון ועמידה בלו״ז יעזרו לי להצליח, אז חילקתי את העבודה לשלבים, וסימנתי ביומן מה צריך לעשות ומתי. הדבר לא רק עזר לי להתקדם בקצב מסודר, אלא גם אפשר לי לקחת הפסקות ולהתכונן למבחנים אחרים, מה שהיה חשוב לא פחות.

כמובן, לא הכל הלך חלק. היו גם קשיים, בהם: שמירה על מוטיבציה, שמירה על הרצון לעשות את הפרויקט שלי על הנושא שעשיתי ולא להחליף למשחק כלשהו פשוט יותר. כמובן שהיו גם שגיאות ובעיות בזמן כתיבת הקוד, שלקחו שעות על גבי שעות למצוא ולתקן. כל פעם שהיה משהו שלא עבד, ניסיתי להבין מה השתבש- קראתי מדריכים, צפיתי בסרטונים, וחיפשתי פתרונות בפורומים. למדתי המון דברים על תחומים שמעולם לא נגעתי בהם, מאבטחת מידע ועד ממשקים גרפיים.

מה שלמדתי בפרויקט הזה, זה הרבה מעבר לכתיבת קוד, למדתי דרכים חדשות לפתירת בעיות לבד, איך לתכנן מערכת שלמה, וטכניקות שונות לאיתור תקלות ושגיאות. למדתי איך ליצור מערכת לוגים שבאמת עוזרת לאתר בעיות, ואיך להשתמש בכלים הקיימים בסביבת הפיתוח ובשפת התכנות כדי לאתר תקלות.

אם הייתי עושה את זה שוב פעם, כנראה הייתי משקיע יותר זמן בשלב התכנון. בזמן יישום הפרויקט הבנתי שהשלב הזה מהווה את החלק הכי חשוב בקביעת הקושי שיהיה במהלך היישום.
כמו כן, אם היה לי יותר זמן לעבוד על הפרויקט, הייתי כנראה מוסיף יותר יכולות לשרת וללקוח (יכולות שהיו, בין היתר, מסייעות להגיע ליעדים שהוצבו בתחילת הפרויקט), כגון: תמיכה בשפות נוספות, תמיכה במערכות הפעלה שונות, תמיכה במצלמה, תמיכה מלאה בעכבר, מערכת הזדהות, לשפר את הממשק הגרפי, ועוד הרבה.
בנוסף, אם הייתי מכיר יותר לעומק שפה ״מהירה״ יותר, למשל ״c++״, הייתי מיישם את כל הפרויקט בשפה הזאת, כדי לקבל ביצועים טובים יותר מהתוכנית שלי.

לסיום, אני רוצה לומר תודה.
קודם כל להורים שלי, שתמכו בי לאורך כל הפרויקט, עודדו, ונתנו לי את השקט שהייתי צריך כדי להתרכז.
לאחותי, שתמיד נתנה מילה טובה ברגע הנכון, ותמיד ניסתה לעזור לי כשניתקלתי בבעיה כלשהי.
תודה ענקית גם לאיתי זוקין, חבר יקר שעזר לי לחשוב, לתכנן ולדייק את הרעיון כבר מהשלבים הראשונים. בלעדיו, הפרויקט הזה לא היה נראה כמו שהוא עכשיו.

# ביבליוגרפיה

1. Python Software Foundation. (n.d.). *pynput 1.7.6 documentation*. Read the Docs. Retrieved May 24, 2025, from https://pynput.readthedocs.io/en/latest/
2. Wikipedia contributors. (n.d.). *JPEG*. Wikipedia. Retrieved May 24, 2025, from https://en.wikipedia.org/wiki/JPEG
3. Wikipedia contributors. (n.d.). *RSA (cryptosystem)*. Wikipedia. Retrieved May 24, 2025, from https://en.wikipedia.org/wiki/RSA_cryptosystem
4. Wikipedia contributors. (n.d.). *Advanced Encryption Standard*. Wikipedia. Retrieved May 24, 2025, from https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
5. Python Cryptographic Authority. (n.d.). *cryptography 42.0.4 documentation*. Retrieved May 24, 2025, from https://cryptography.io/en/latest/
6. Python Software Foundation. (n.d.). *zlib — Compression compatible with gzip*. Python 3.12.3 documentation. Retrieved May 24, 2025, from https://docs.python.org/3/library/zlib.html
7. Python Software Foundation. (n.d.). *time — Time access and conversions*. Python 3.12.3 documentation. Retrieved May 24, 2025, from https://docs.python.org/3/library/time.html
8. Python Software Foundation. (n.d.). *socket — Low-level networking interface*. Python 3.12.3 documentation. Retrieved May 24, 2025, from https://docs.python.org/3/library/socket.html
9. Python Software Foundation. (n.d.). *io — Core tools for working with streams*. Python 3.12.3 documentation. Retrieved May 24, 2025, from https://docs.python.org/3/library/io.html
10. Python Imaging Library Contributors. (n.d.). *Pillow (PIL Fork) documentation*. Read the Docs. Retrieved May 24, 2025, from https://pillow.readthedocs.io/en/stable/
11. Python Software Foundation. (n.d.). *tkinter — Python interface to Tcl/Tk*. Python 3.12.3 documentation. Retrieved May 24, 2025, from https://docs.python.org/3/library/tkinter.html
12. Python Software Foundation. (n.d.). *os — Miscellaneous operating system interfaces*. Python 3.12.3 documentation. Retrieved May 24, 2025, from https://docs.python.org/3/library/os.html
13. Python Software Foundation. (n.d.). *queue — A synchronized queue class*. Python 3.12.3 documentation. Retrieved May 24, 2025, from https://docs.python.org/3/library/queue.html

# נספחים

```
#App.py
import tkinter as tk
from tkinter import ttk
import queue
import subprocess
import sys
import os
import socket
import ipaddress

from ClientThreads import Client
from Client_GUI import ClientGui
from ServerThreads import Server
from Server_GUI import ServerGui

# Define default IP addresses and port
DEFAULT_PORT = '12345'
DEFAULT_CLIENT_IP = '127.0.0.1'
DEFAULT_SERVER_IP = '0.0.0.0'


class StarterApp:
    """
    A GUI application that lets the user choose between Client or Server
mode,
    enter configuration details (IP and Port), and launch the corresponding
mode.
    """

    def __init__(self, root, reason=None):
        """
        Initialize the starter application GUI.

        :param root: The root tkinter window.
        :type root: tk.Tk
        :param reason: Reason for relaunching the app (e.g.,
disconnection).
        :type reason: str, optional
        """
        self.root = root
        self.root.title("Client/Server Launcher")
        self.root.geometry("640x360")
        self.root.configure(bg="#f0f0f0")
        self.root.resizable(False, False)

        self.reason = reason
        self.client_switch = None

        # GUI elements initialized as None
        self.client_button = None
        self.server_button = None
        self.input_frame = None
        self.start_button = None
        self.title_label = None
        self.button_frame = None
        self.ip_label = None
        self.ip_entry = None
        self.port_label = None
        self.port_entry = None
```

```
        self.info_label = None

        self.create_widgets()

    def create_widgets(self):
        """
        Create the initial GUI widgets for selecting client/server mode.
        """
        if self.reason:
            reason_label = ttk.Label(
                self.root,
                text=f"Previous session ended: {self.reason}",
                foreground="red",
                font=("Helvetica", 10)
            )
            reason_label.pack(pady=(10, 0))

        self.title_label = ttk.Label(self.root, text="Select Mode",
font=("Helvetica", 18))
        self.title_label.pack(pady=30)

        self.button_frame = ttk.Frame(self.root)
        self.button_frame.pack()

        self.client_button = ttk.Button(self.button_frame, text="Client",
command=self.client_mode)
        self.client_button.grid(row=0, column=0, padx=20, ipadx=20,
ipady=10)

        self.server_button = ttk.Button(self.button_frame, text="Server",
command=self.server_mode)
        self.server_button.grid(row=0, column=1, padx=20, ipadx=20,
ipady=10)

        self.input_frame = ttk.Frame(self.root)
        self.start_button = ttk.Button(self.root, text="Start",
command=self.start_pressed)

    def clear_initial_buttons(self):
        """
        Remove the initial mode selection buttons from the UI.
        """
        self.client_button.grid_forget()
        self.server_button.grid_forget()
        self.button_frame.pack_forget()
        self.title_label.pack_forget()

    def client_mode(self):
        """
        Configure the interface for client mode.
        """
        self.clear_initial_buttons()
        self.client_switch = True
        self.show_input_fields()

    def server_mode(self):
        """
        Configure the interface for server mode.
        """
        self.clear_initial_buttons()
        self.client_switch = False
```

```
        self.show_input_fields()

    def show_input_fields(self):
        """
        Show input fields for IP and Port based on selected mode.
        """
        self.input_frame.pack(pady=40)
        row = 0

        if self.client_switch:
            self.ip_label = ttk.Label(self.input_frame, text="IP Address:")
            self.ip_label.grid(row=row, column=0, sticky='e', padx=10,
pady=5)
            self.ip_entry = ttk.Entry(self.input_frame, foreground="gray")
            self.ip_entry.grid(row=row, column=1, padx=10, pady=5)
            self.add_placeholder(self.ip_entry, DEFAULT_CLIENT_IP)
            row += 1

        self.port_label = ttk.Label(self.input_frame, text="Port:")
        self.port_label.grid(row=row, column=0, sticky='e', padx=10,
pady=5)
        self.port_entry = ttk.Entry(self.input_frame, foreground="gray")
        self.port_entry.grid(row=row, column=1, padx=10, pady=5)
        self.add_placeholder(self.port_entry, DEFAULT_PORT)

        self.info_label = ttk.Label(self.input_frame, text="",
foreground="red")
        self.info_label.grid(row=row + 1, column=0, columnspan=2, pady=10)
        self.info_label.grid_remove()

        self.start_button.pack(pady=20)

    def add_placeholder(self, entry, placeholder):
        """
        Add placeholder behavior to an entry field.

        :param entry: Entry widget.
        :type entry: tk.Entry
        :param placeholder: Placeholder text.
        :type placeholder: str
        """
        entry.insert(0, placeholder)
        entry.config(foreground="gray")

        def on_focus_in(event):
            if entry.get() == placeholder:
                entry.delete(0, tk.END)
                entry.config(foreground="black")

        def on_focus_out(event):
            if entry.get() == "":
                entry.insert(0, placeholder)
                entry.config(foreground="gray")

        entry.bind("<FocusIn>", on_focus_in)
        entry.bind("<FocusOut>", on_focus_out)

    def is_valid_ip(self, ip):
        """
        Check if the provided IP address is valid.
```

```
        :param ip: IP address string.
        :type ip: str
        :return: True if valid, False otherwise.
        :rtype: bool
        """
        try:
            ipaddress.ip_address(ip)
            return True
        except ValueError:
            return False

    def is_port_available(self, port):
        """
        Check if the given port is available on localhost.

        :param port: Port number.
        :type port: int
        :return: True if available, False if in use.
        :rtype: bool
        """
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            return s.connect_ex(('localhost', port)) != 0

    def can_connect_to_server(self, ip, port):
        """
        Check if a connection can be made to a server.

        :param ip: IP address.
        :type ip: str
        :param port: Port number.
        :type port: int
        :return: True if connection is possible, False otherwise.
        :rtype: bool
        """
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
            sock.settimeout(2)
            return sock.connect_ex((ip, port)) == 0

    def start_pressed(self):
        """
        Validate user input and launch the appropriate mode
(client/server).
        """
        port_str = self.port_entry.get()
        ip_val = self.ip_entry.get() if self.ip_entry else
DEFAULT_SERVER_IP

        try:
            port = int(port_str)
            if not (1024 <= port <= 65535):
                raise ValueError
        except ValueError:
            self.info_label.config(text="Port must be a number between 1024
and 65535.")
            self.info_label.grid()
            return

        if self.client_switch and not self.is_valid_ip(ip_val):
            self.info_label.config(text="Invalid IP address.")
            self.info_label.grid()
            return
```

```
        if self.client_switch:
            if not self.can_connect_to_server(ip_val, port):
                self.info_label.config(text="Cannot connect to server at
given IP/Port.")
                self.info_label.grid()
                return
        else:
            if not self.is_port_available(port):
                self.info_label.config(text="Port is already in use. Server
may already be running.")
                self.info_label.grid()
                return

        self.root.destroy()

        if self.client_switch:
            run_client_mode(ip_val, port)
        else:
            run_server_mode(ip_val, port)

    def run(self):
        """
        Run the main tkinter loop.
        """
        self.root.mainloop()


def run_client_mode(ip, port):
    """
    Start the client application.

    :param ip: IP address to connect to.
    :type ip: str
    :param port: Port to connect to.
    :type port: int
    """
    def relaunch(reason=None):
        print(f"[Run Client] Relaunching StarterApp due to: {reason}")
        subprocess.Popen([sys.executable, os.path.abspath(__file__), reason
or "disconnected"])
        sys.exit(0)

    def on_disconnect(reason):
        print(f"[Run Client] Client disconnect reason: {reason}")
        try:
            root.after(0, root.destroy)
        except Exception as e:
            print("[Run Client] Error during root destruction:", e)
        relaunch(reason)

    root = tk.Tk()
    status_window = ClientGui(root, (ip, port))
    client = Client(ip, port, disconnect_callback=on_disconnect)
    client.start()

    try:
        status_window.run()
    finally:
        print("[Run Client] Stopping client")
        client.stop("disconnected from server")
```

```
def run_server_mode(ip, port):
    """
    Start the server application.

    :param ip: IP address to bind to.
    :type ip: str
    :param port: Port to bind to.
    :type port: int
    """
    def relaunch(reason=None):
        print(f"[Run Server] Relaunching StarterApp due to: {reason}")
        subprocess.Popen([sys.executable, os.path.abspath(__file__), reason
or "session stopped"])
        os._exit(0)

    def on_server_close(reason):
        print(f"[Run Server] Server close reason: {reason}")
        try:
            root.after(0, root.destroy)
        except Exception as e:
            print("[Run Server] Error during root destruction:", e)
        relaunch(reason)

    root = tk.Tk()
    command_queue = queue.Queue()
    frame_queue = queue.Queue()

    app = ServerGui(port, root, command_queue, frame_queue)
    server = Server(ip, port, app, command_queue, frame_queue,
close_callback=on_server_close)
    server.start()

    try:
        app.run()
    finally:
        print("[Run Server] Stopping server")
        app.shutdown_event.set()
        server.stop("session closed")
        server.join()


def main():
    """
    Entry point for the application. Launches the starter GUI.
    If a reason is passed as a command-line argument, show it.
    """
    reason = None
    if len(sys.argv) > 1:
        reason = sys.argv[1]
    root = tk.Tk()
    app = StarterApp(root, reason)
    app.run()

if __name__ == "__main__":
    main()

#Client_GUI.py
from tkinter import ttk
```

```
class ClientGui:
    """
    A simple GUI application using tkinter to display the status of a
client
    sharing its screen with a server at a specific IP address and port.
    """

    def __init__(self, root, address):
        """
        Initialize the GUI.

        :param root: The main tkinter root window.
        :type root: tk.Tk
        :param address: A tuple containing IP address and port number.
        :type address: tuple
        """
        self.root = root
        self.address = address

        # Configure the main window appearance
        self.root.title("Client Status")
        self.root.configure(bg="#e6f2ff")
        self.root.resizable(False, False)

        # Create and pack a label that shows connection status
        self.label = ttk.Label(
            self.root,
            text=f"Sharing screen with: {self.address[0]} on port:
{self.address[1]}",
            font=("Helvetica", 14),
            foreground="green"
        )
        self.label.pack(expand=True, pady=30)

        # Schedule a geometry update after 100 milliseconds
        self.root.after(100, self.update_geometry)

        # Set up window close protocol
        self.root.wm_protocol("WM_DELETE_WINDOW", self.on_close)

    def update_geometry(self):
        """
        Dynamically adjusts the window width based on the label size
        to ensure proper spacing and visibility.
        """
        self.root.update()
        label_width = self.label.winfo_width()
        print("Label width:", label_width)  # Debug print for label width
        self.root.geometry(f"{label_width + 20}x120")  # Adjust window size

    def on_close(self):
        """
        Handler for window close event.
        Currently does nothing—to block client closing window.
        """
        pass

    def run(self):
        """
        Start the main GUI loop.
```

```
        """
        self.root.mainloop()

#ClientThreads.py
import socket
import Encryption
import threading
import time
import zlib

from FunctionsModule import InputController, UserBlocker, take_screenshot,
recv_all

# Frames per second for screen sharing
FPS = 10


class Client(threading.Thread):
    """
    The Client class handles connection to a remote server, input commands,
screen sharing,
    encryption key exchange, and user interaction blocking.
    """

    def __init__(self, address, port, disconnect_callback=None):
        """
        Initialize the Client thread and prepare for server connection.

        :param address: IP address of the server.
        :type address: str
        :param port: Port number of the server.
        :type port: int
        :param disconnect_callback: Callback executed on disconnection.
        :type disconnect_callback: callable, optional
        """
        super().__init__()
        self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.address = address
        self.port = port
        self.disconnect_callback = disconnect_callback
        self._disconnected = False
        self._lock = threading.Lock()

        # RSA encryption setup
        self.private_key, self.public_key = Encryption.generate_rsa_keys()
        self.server_rsa_key = None
        self.server_aes_key = None

        # Threading and control mechanisms
        self.pause_event = threading.Event()
        self.screen_share = None
        self.img_size = (1920, 1080)
        self.block_event = threading.Event()
        self.block_event_lock = threading.Lock()
        self.blocker = UserBlocker()
        self.input_controller = InputController(self.block_event,
self.block_event_lock, self.blocker)
        self.stop_event = threading.Event()

    def run(self):
        """
```

```
        Run the client thread to connect to the server, handle commands,
        and manage screen sharing.
        """
        stop_reason = "client crashed"
        try:
            self.client.connect((self.address, self.port))
            print(f"[Client] Connected to server at
{self.address}:{self.port}")
            self.key_exchange()

            self.screen_share = ScreenShare(self.client,
self.server_aes_key, self.pause_event)
            self.screen_share.start()

            while not self.stop_event.is_set():
                try:
                    # Receive command length and actual command (AES
encrypted)
                    cmd_len_bytes = recv_all(self.client, 4)
                    cmd_len = int.from_bytes(cmd_len_bytes, 'big')

                    iv = recv_all(self.client, 16)
                    encrypted_cmd = recv_all(self.client, cmd_len)
                    command = Encryption.decrypt_aes(self.server_aes_key,
iv, encrypted_cmd).decode()

                    # Handle input-related commands
                    if command.startswith(("key_down", "key_up",
"button")):
                        self.input_controller.handle_command(command)

                    # Handle resize command
                    elif command.startswith("resize"):
                        try:
                            _, width, height = command.split(":")
                            width, height = int(width), int(height)
                            print(f"[Client] Resizing to:
{width}x{height}")
                            self.img_size = (width, height)
                            if self.screen_share:

self.screen_share.update_size(self.img_size)
                        except ValueError as e:
                            print(f"[Client] Invalid resize command:
{command} ({e})")

                    # Handle session termination command
                    elif command == "kick":
                        stop_reason = "kicked from session"
                        print("[Client] Received kick command.")
                        break

                    # Block/unblock user input
                    if command == "block":
                        with self.block_event_lock:
                            self.block_event.set()
                        self.blocker.start_blocking()
                        print("Blocking: True")
                    elif command == "unblock":
                        with self.block_event_lock:
                            self.block_event.clear()
```

```
                        self.blocker.stop_blocking()
                        print("Blocking: False")

                    # Pause/unpause screen sharing
                    elif command == "pause":
                        self.pause_event.set()
                        print("Paused: True")
                    elif command == "unpause":
                        self.pause_event.clear()
                        print("Paused: False")

                except (ConnectionResetError, ConnectionError) as e:
                    print(f"[Client] Error: {e}")
                    stop_reason = "session closed"
                    break
                except Exception as e:
                    print(f"[Client] Error: {e}")
                    break

        finally:
            self.stop(stop_reason)

    def key_exchange(self):
        """
        Perform RSA public key exchange with the server and receive AES
session key.
        """
        # Send our public key to the server
        pub_key_bytes = Encryption.serialize_public_key(self.public_key)
        self.client.send(len(pub_key_bytes).to_bytes(4, 'big'))
        self.client.send(pub_key_bytes)

        # Receive server's public key
        server_key_len = int.from_bytes(recv_all(self.client, 4), 'big')
        server_key_bytes = recv_all(self.client, server_key_len)
        self.server_rsa_key =
Encryption.deserialize_public_key(server_key_bytes)

        # Receive AES session key encrypted with our public RSA key
        aes_key_len = int.from_bytes(recv_all(self.client, 4), 'big')
        encrypted_aes_key = recv_all(self.client, aes_key_len)
        self.server_aes_key = Encryption.rsa_decrypt(self.private_key,
encrypted_aes_key)

    def stop(self, stop_reason):
        """
        Stop the client gracefully, terminate screen sharing, and clean up.

        :param stop_reason: Reason for stopping the client (used in
callback).
        :type stop_reason: str
        """
        with self._lock:
            if self._disconnected:
                return
            self._disconnected = True

            self.stop_event.set()
            if self.screen_share:
                self.screen_share.stop()
            if self.blocker:
```

```
                self.blocker.stop_blocking()
            self.client.close()
            if self.disconnect_callback:
                self.disconnect_callback(stop_reason)


class ScreenShare(threading.Thread):
    """
    The ScreenShare class handles periodic screen capturing, compression,
    encryption, and transmission to the server.
    """

    def __init__(self, client, aes_key, pause_event):
        """
        Initialize the screen sharing thread.

        :param client: Socket used for sending screen data.
        :type client: socket.socket
        :param aes_key: AES key for encrypting screen data.
        :type aes_key: bytes
        :param pause_event: Event to pause/resume screen sharing.
        :type pause_event: threading.Event
        """
        super().__init__()
        self.client = client
        self.aes_key = aes_key
        self.img_size = (1920, 1080)
        self.stop_event = threading.Event()
        self.pause_event = pause_event
        self._lock = threading.Lock()

    def run(self):
        """
        Continuously capture, compress, encrypt, and send screen frames to
the server.
        """
        while not self.stop_event.is_set():
            if self.pause_event.is_set():
                time.sleep(0.5)
                continue

            try:
                print("[Screen Share] Taking screenshot...")
                start = time.perf_counter()

                with self._lock:
                    width, height = self.img_size

                # Capture screen image
                image_bytes = take_screenshot(width, height, quality=75)
                if not image_bytes:
                    print("[Screen Share] Screenshot failed or returned
empty")
                    time.sleep(0.5)
                    continue

                # Compress image data
                compressed_bytes = zlib.compress(image_bytes, level=6)
                if not compressed_bytes:
                    print("[Screen Share] Compression failed")
                    time.sleep(0.5)
```

```
                    continue

                # Encrypt the compressed data
                iv, encrypted = Encryption.encrypt_aes(self.aes_key,
compressed_bytes)
                if not encrypted:
                    print("[Screen Share] Encryption failed")
                    time.sleep(0.5)
                    continue

                # Send data to server
                self.client.sendall(len(encrypted).to_bytes(8, 'big'))
                self.client.sendall(iv)
                self.client.sendall(encrypted)

                print(f"[Screen Share] Frame sent | raw={len(image_bytes)}
| compressed={len(compressed_bytes)} | encrypted={len(encrypted)}")

                # Maintain frame rate
                elapsed = time.perf_counter() - start
                time.sleep(max(0, (1 / FPS) - elapsed))

            except Exception as e:
                print(f"[Screen Share] Error: {e}")
                time.sleep(1)

    def update_size(self, new_size):
        """
        Update the screen capture resolution.

        :param new_size: New resolution as (width, height).
        :type new_size: tuple
        """
        with self._lock:
            print(f"[Screen Share] Updating image size to: {new_size}")
            self.img_size = new_size

    def stop(self):
        """
        Stop the screen sharing thread.
        """
        self.stop_event.set()

#Encryption.py
from cryptography.hazmat.primitives.asymmetric import rsa, padding as
asy_pad
from cryptography.hazmat.primitives import serialization, hashes, padding
as sy_pad
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms,
modes
from cryptography.hazmat.backends import default_backend
import os


def generate_rsa_keys():
    """
    Generate an RSA private-public key pair.

    :return: Tuple containing (private_key, public_key).
    :rtype: (rsa.RSAPrivateKey, rsa.RSAPublicKey)
    """
```

```
    # Generate private RSA key with public exponent 65537 and 2048-bit key
size
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
    )
    # Derive the corresponding public key from private key
    public_key = private_key.public_key()
    return private_key, public_key


def generate_aes_key():
    """
    Generate a random 256-bit AES key.

    :return: AES key bytes.
    :rtype: bytes
    """
    # os.urandom generates cryptographically secure random bytes
    return os.urandom(32)  # 32 bytes = 256 bits


def rsa_encrypt(public_key, text):
    """
    Encrypt text using RSA public key with OAEP padding.

    :param public_key: RSA public key for encryption.
    :type public_key: rsa.RSAPublicKey
    :param text: Plaintext to encrypt (str or bytes).
    :type text: str or bytes
    :return: Encrypted ciphertext bytes.
    :rtype: bytes
    """
    # Convert text to bytes if necessary
    if type(text) is not bytes:
        text = text.encode()

    # Encrypt using OAEP padding with SHA-256 as hash function
    ciphertext = public_key.encrypt(
        text,
        asy_pad.OAEP(
            mgf=asy_pad.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return ciphertext


def rsa_decrypt(private_key, ciphertext):
    """
    Decrypt ciphertext using RSA private key with OAEP padding.

    :param private_key: RSA private key for decryption.
    :type private_key: rsa.RSAPrivateKey
    :param ciphertext: Encrypted bytes to decrypt.
    :type ciphertext: bytes
    :return: Decrypted plaintext bytes.
    :rtype: bytes
    """
    # Decrypt ciphertext using OAEP padding with SHA-256 as hash function
```

```
    decrypted_message = private_key.decrypt(
        ciphertext,
        asy_pad.OAEP(
            mgf=asy_pad.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return decrypted_message


def serialize_public_key(public_key):
    """
    Serialize RSA public key to PEM format bytes.

    :param public_key: RSA public key to serialize.
    :type public_key: rsa.RSAPublicKey
    :return: PEM-encoded public key bytes.
    :rtype: bytes
    """
    pem = public_key.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
    )
    return pem


def deserialize_public_key(pem):
    """
    Deserialize PEM bytes to RSA public key.

    :param pem: PEM-encoded public key bytes.
    :type pem: bytes
    :return: RSA public key object.
    :rtype: rsa.RSAPublicKey
    """
    return serialization.load_pem_public_key(pem)


def encrypt_aes(key, text):
    """
    Encrypt text using AES-CBC with PKCS7 padding.

    :param key: AES key bytes (256-bit).
    :type key: bytes
    :param text: Plaintext to encrypt (str or bytes).
    :type text: str or bytes
    :return: Tuple (iv, ciphertext) where iv is the initialization vector.
    :rtype: (bytes, bytes)
    """
    # Convert text to bytes if necessary
    if type(text) is not bytes:
        text = text.encode()

    # Generate a random 16-byte IV for AES CBC mode
    iv = os.urandom(16)

    # Pad the plaintext to be multiple of block size (128 bits for AES)
    padder = sy_pad.PKCS7(128).padder()
    padded_data = padder.update(text) + padder.finalize()
```

```
    # Create AES cipher object in CBC mode
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv),
backend=default_backend())
    encryptor = cipher.encryptor()

    # Encrypt the padded plaintext
    ciphertext = encryptor.update(padded_data) + encryptor.finalize()
    return iv, ciphertext


def decrypt_aes(key, iv, ciphertext):
    """
    Decrypt AES-CBC encrypted ciphertext with PKCS7 padding removal.

    :param key: AES key bytes (256-bit).
    :type key: bytes
    :param iv: Initialization vector used during encryption.
    :type iv: bytes
    :param ciphertext: Encrypted ciphertext bytes.
    :type ciphertext: bytes
    :return: Decrypted plaintext bytes.
    :rtype: bytes
    """
    # Create AES cipher object in CBC mode with provided IV
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv),
backend=default_backend())
    decryptor = cipher.decryptor()

    # Decrypt ciphertext to padded plaintext
    padded_plaintext = decryptor.update(ciphertext) + decryptor.finalize()

    # Remove PKCS7 padding from plaintext
    unpadder = sy_pad.PKCS7(128).unpadder()
    plaintext = unpadder.update(padded_plaintext) + unpadder.finalize()
    return plaintext

#FunctionsModule.py
from PIL import ImageGrab
from io import BytesIO
from pynput import mouse, keyboard


# Maximum chunk size for socket receiving to avoid too large reads at once
MAX_CHUNK_SIZE = 4096

# Mapping string mouse button IDs to pynput mouse.Button enums
MOUSE_BUTTON_MAP = {
    "1": mouse.Button.left,
    "2": mouse.Button.middle,
    "3": mouse.Button.right
}

# Mapping string key names to pynput keyboard.Key special keys or
characters
KEYBOARD_SPECIAL_KEYS_MAP = {
    'Shift_L': keyboard.Key.shift,
    'Shift_R': keyboard.Key.shift_r,
    'Control_L': keyboard.Key.ctrl,
    'Control_R': keyboard.Key.ctrl_r,
    'Alt_L': keyboard.Key.alt,
    'Alt_R': keyboard.Key.alt_r,
```

```
    'BackSpace': keyboard.Key.backspace,
    'Tab': keyboard.Key.tab,
    'Return': keyboard.Key.enter,
    'Escape': keyboard.Key.esc,
    'space': keyboard.Key.space,
    'Left': keyboard.Key.left,
    'Right': keyboard.Key.right,
    'Up': keyboard.Key.up,
    'Down': keyboard.Key.down,
    'Delete': keyboard.Key.delete,
    'Caps_Lock': keyboard.Key.caps_lock,
    'Insert': keyboard.Key.insert,
    'Home': keyboard.Key.home,
    'End': keyboard.Key.end,
    'Page_Up': keyboard.Key.page_up,
    'Page_Down': keyboard.Key.page_down,
    'Num_Lock': keyboard.Key.num_lock,
    'Scroll_Lock': keyboard.Key.scroll_lock,
    'Pause': keyboard.Key.pause,
    'Print': keyboard.Key.print_screen,
    'Menu': keyboard.Key.menu,
    'F1': keyboard.Key.f1,
    'F2': keyboard.Key.f2,
    'F3': keyboard.Key.f3,
    'F4': keyboard.Key.f4,
    'F5': keyboard.Key.f5,
    'F6': keyboard.Key.f6,
    'F7': keyboard.Key.f7,
    'F8': keyboard.Key.f8,
    'F9': keyboard.Key.f9,
    'F10': keyboard.Key.f10,
    'F11': keyboard.Key.f11,
    'F12': keyboard.Key.f12,
    'period': '.',
    'comma': ',',
    'slash': '/',
    'backslash': '\\',
    'bracketleft': '[',
    'bracketright': ']',
    'semicolon': ';',
    'apostrophe': "'",
    'grave': '`',
    'minus': '-',
    'equal': '=',
    'plus': '+',
    'underscore': '_',
    'colon': ':',
    'quotedbl': '"',
    'bar': '|',
    'asciitilde': '~',
    'less': '<',
    'greater': '>',
    'question': '?',
    'exclam': '!',
    'at': '@',
    'numbersign': '#',
    'dollar': '$',
    'percent': '%',
    'asciicircum': '^',
    'ampersand': '&',
    'asterisk': '*',
```

```
    'parenleft': '(',
    'parenright': ')',
}


class UserBlocker:
    """
    Class to block all user input (keyboard and mouse) by suppressing
events.
    """

    def __init__(self):
        """
        Initialize UserBlocker with no active listeners.
        """
        self.keyboard_listener = None
        self.mouse_listener = None

    def start_blocking(self):
        """
        Start blocking user input by suppressing keyboard and mouse events.
        This creates and starts listeners that prevent any input from
reaching other apps.
        """
        if not self.keyboard_listener or not self.mouse_listener:
            # Suppress=True means events are blocked/suppressed system-wide
            self.keyboard_listener = keyboard.Listener(suppress=True)
            self.mouse_listener = mouse.Listener(suppress=True)
            self.keyboard_listener.start()
            self.mouse_listener.start()

    def stop_blocking(self):
        """
        Stop blocking user input by stopping the keyboard and mouse
listeners.
        """
        if self.keyboard_listener and self.keyboard_listener.running:
            self.keyboard_listener.stop()
        if self.mouse_listener and self.mouse_listener.running:
            self.mouse_listener.stop()
        self.keyboard_listener = None
        self.mouse_listener = None


class InputController:
    """
    Handles injecting mouse and keyboard inputs programmatically.
    """

    def __init__(self, block_event, block_event_lock, user_blocker):
        """
        Initialize the input controller.

        :param block_event: threading.Event used to track if input blocking
is active.
        :param block_event_lock: threading.Lock protecting access to
block_event.
        :param user_blocker: Instance of UserBlocker to start/stop blocking
when needed.
        """
        self.block_event = block_event
```

```
        self.block_event_lock = block_event_lock
        self.user_blocker = user_blocker
        self.mouse = mouse.Controller()
        self.keyboard = keyboard.Controller()

    def set_mouse_pos(self, x_pos, y_pos):
        """
        Set the mouse cursor position on screen.

        :param x_pos: X coordinate.
        :param y_pos: Y coordinate.
        """
        self.mouse.position = (x_pos, y_pos)

    def handle_command(self, command):
        """
        Handle an input command string and execute mouse or keyboard
actions.

        :param command: Command string, e.g. "button:1:100:200",
"key_down:Shift_L".
        """
        try:
            with self.block_event_lock:
                was_blocking = self.block_event.is_set()
                if was_blocking:
                    # Temporarily stop blocking so injected inputs are
accepted by the system
                    self.user_blocker.stop_blocking()

                sliced_command = command.split(":")
                if sliced_command[0] == "button":
                    # Mouse button click command:
button:<button_num>:<x>:<y>
                    button_num = sliced_command[1]
                    x = int(sliced_command[2])
                    y = int(sliced_command[3])
                    button = MOUSE_BUTTON_MAP.get(button_num)
                    if button:
                        self.set_mouse_pos(x, y)
                        self.mouse.click(button)
                    else:
                        print(f"[InputController] Unknown mouse button:
{button_num}")

                elif sliced_command[0] == "key_down":
                    # Key press command
                    self._press_key(sliced_command[1])

                elif sliced_command[0] == "key_up":
                    # Key release command
                    self._release_key(sliced_command[1])

                if was_blocking:
                    # Restart blocking after input injection to continue
suppressing real user input
                    self.user_blocker.start_blocking()

        except Exception as e:
            print(f"[InputController] Error handling command '{command}':
{e}")
```

```
    def _press_key(self, key):
        """
        Press a keyboard key.

        :param key: String key name to press.
        """
        if key in KEYBOARD_SPECIAL_KEYS_MAP:
            pynput_key = KEYBOARD_SPECIAL_KEYS_MAP[key]
            self.keyboard.press(pynput_key)
        else:
            try:
                self.keyboard.press(key)
            except ValueError:
                print(f"[InputController] Invalid key press: {key}")

    def _release_key(self, key):
        """
        Release a keyboard key.

        :param key: String key name to release.
        """
        if key in KEYBOARD_SPECIAL_KEYS_MAP:
            pynput_key = KEYBOARD_SPECIAL_KEYS_MAP[key]
            self.keyboard.release(pynput_key)
        else:
            try:
                self.keyboard.release(key)
            except ValueError:
                print(f"[InputController] Invalid key release: {key}")


def take_screenshot(x_size=1920, y_size=1080, quality=75):
    """
    Capture a screenshot of the screen resized to given dimensions and
compressed as JPEG.

    :param x_size: Width of the resized screenshot.
    :param y_size: Height of the resized screenshot.
    :param quality: JPEG quality (0-100).
    :return: JPEG image bytes.
    """
    # Capture the entire screen
    screenshot = ImageGrab.grab()

    # Resize screenshot to target resolution
    screenshot = screenshot.resize((x_size, y_size))

    # Save screenshot to in-memory byte stream in JPEG format with
specified quality
    byte_io = BytesIO()
    screenshot.save(byte_io, format='JPEG', quality=quality)
    byte_io.seek(0)

    # Return the raw bytes of the JPEG image
    return byte_io.getvalue()


def recv_all(sock, num_bytes):
    """
    Receive an exact number of bytes from a socket, handling partial
```

```
receives.

    :param sock: Socket object to receive from.
    :param num_bytes: Number of bytes to receive.
    :return: Received bytes.
    :raises ConnectionError: If the connection is lost before all bytes are
received.
    """
    data = b''
    while len(data) < num_bytes:
        # Receive the remaining number of bytes or MAX_CHUNK_SIZE,
whichever is smaller
        packet = sock.recv(min(MAX_CHUNK_SIZE, num_bytes - len(data)))
        if not packet:
            # Connection lost unexpectedly
            raise ConnectionError("Connection lost while receiving data")
        data += packet
    return data

#Server_GUI.py
import tkinter as tk
from PIL import ImageTk, Image
from io import BytesIO
import math
import os
import queue
import threading
import time
import socket


# Directory where this script is located, used for loading assets reliably
SCRIPT_DIR = os.path.dirname(os.path.abspath(__file__))

# Names of buttons shown in fullscreen mode
FULLSCREEN_BUTTON_NAMES = ["control start", "block", "kick"]

# Aspect ratio for screen previews (width / height)
RATIO = 16 / 9

# Size of control buttons in fullscreen mode (width and height)
BUTTON_SIZE = 54

# Height reserved for taskbar or UI elements at bottom
TASKBAR_HEIGHT = 72

# Tooltip texts associated with each button name
TOOLTIPS = {
    "control start": "Take control of the screen",
    "control stop": "Stop control of the screen",
    "block": "Block the student screen",
    "unblock": "Unblock the student screen",
    "kick": "Kick the student",
    "exit": "Exit fullscreen mode"
}

# File paths for button icons
ICON_FILES = {
    "info": os.path.join(SCRIPT_DIR, "assets", "info_icon.png"),
    "control start": os.path.join(SCRIPT_DIR, "assets",
"control_start_icon.png"),
```

```
    "control_stop": os.path.join(SCRIPT_DIR, "assets",
"control_stop_icon.png"),
    "block": os.path.join(SCRIPT_DIR, "assets", "block_icon.png"),
    "unblock": os.path.join(SCRIPT_DIR, "assets", "unblock_icon.png"),
    "kick": os.path.join(SCRIPT_DIR, "assets", "kick_icon.png"),
    "exit": os.path.join(SCRIPT_DIR, "assets", "exit_icon.png")
}


class ToolTip:
    """Tooltip class to show helper text when hovering over a widget."""

    def __init__(self, widget, text):
        self.widget = widget
        self.text = text
        self.tooltip = None
        self.widget.tooltip_instance = self  # allow external
refresh/update

        # Bind events to show/hide tooltip on mouse enter/leave
        self.widget.bind("<Enter>", self.show_tooltip)
        self.widget.bind("<Leave>", self.hide_tooltip)

    def update_text(self, new_text):
        """Update the tooltip text and hide it if currently shown."""
        self.text = new_text
        if self.tooltip:
            self.hide_tooltip()

    def show_tooltip(self, event=None):
        """Display the tooltip near the widget, centered above it."""
        if self.tooltip or not self.text:
            return  # Tooltip already shown or no text to display

        # Create a new top-level window for the tooltip
        self.tooltip = tk.Toplevel(self.widget)
        self.tooltip.overrideredirect(True)  # Remove window decorations

        # Create a label with the tooltip text and style it
        label = tk.Label(self.tooltip, text=self.text,
foreground="#FFFFFF", background="#232333",
                         relief="solid", borderwidth=1, font=("Arial", 10))
        label.pack()

        self.tooltip.update_idletasks()  # Make sure geometry is updated

        # Get the absolute position and size of the widget
        widget_x = self.widget.winfo_rootx()
        widget_y = self.widget.winfo_rooty()
        widget_width = self.widget.winfo_width()

        tooltip_width = self.tooltip.winfo_width()
        tooltip_height = self.tooltip.winfo_height()

        # Calculate tooltip position (centered horizontally above the
widget)
        x = widget_x + (widget_width // 2) - (tooltip_width // 2)
        y = widget_y - tooltip_height - 5  # 5 pixels above widget

        self.tooltip.geometry(f"+{x}+{y}")
        self.tooltip.update()
```

```
    def hide_tooltip(self, event=None):
        """Destroy the tooltip window if it exists."""
        if self.tooltip:
            self.tooltip.destroy()
            self.tooltip = None


class InfoButtonToolTip(ToolTip):
    """Special tooltip class for the info button, positioned
differently."""

    def show_tooltip(self, event=None):
        """Display tooltip to the left and above the info button."""
        if self.tooltip or not self.text:
            return

        self.tooltip = tk.Toplevel(self.widget)
        self.tooltip.overrideredirect(True)

        label = tk.Label(self.tooltip, text=self.text,
foreground="#FFFFFF", background="#232333",
                         relief="solid", borderwidth=1, font=("Arial", 10))
        label.pack()

        self.tooltip.update_idletasks()

        widget_x = self.widget.winfo_rootx()
        widget_y = self.widget.winfo_rooty()

        tooltip_width = self.tooltip.winfo_width()
        tooltip_height = self.tooltip.winfo_height()

        # Position tooltip to the left and above the info icon
        x = widget_x - tooltip_width
        y = widget_y - tooltip_height

        self.tooltip.geometry(f"+{x}+{y}")
        self.tooltip.update()


class ServerGui:
    """Main GUI class for teacher control app."""

    def __init__(self, port, root, command_queue, frame_queue):
        """
        Initialize the ServerGui instance.

        :param port: Port number the server listens on.
        :type port: int
        :param root: Tkinter root window.
        :type root: tkinter.Tk
        :param command_queue: Queue for sending commands to clients.
        :type command_queue: queue.Queue
        :param frame_queue: Queue for receiving frames/screenshots from
clients.
        :type frame_queue: queue.Queue
        """

        def get_local_ip():
            """
```

```
            Get local IP address for display; fallback to error message if
unavailable.

            :return: Local IP address or error message.
            :rtype: str
            """
            try:
                s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
                s.connect(("8.8.8.8", 80))  # Use Google's DNS server for
IP discovery
                ip = s.getsockname()[0]
                s.close()
                return ip
            except Exception as e:
                return f"Error: {e}"

        self.ip = get_local_ip()
        self.port = port
        self.root = root
        self.command_queue = command_queue
        self.frame_queue = frame_queue

        # Maintain a set of kicked client addresses for blocking
        self.kicked_addresses = set()
        self.kicked_lock = threading.Lock()

        # Graceful shutdown management
        self.root.protocol("WM_DELETE_WINDOW", self.on_close)
        self.shutdown_event = threading.Event()

        # Window setup
        self.root.title("Teacher Control App")

self.root.geometry(f"{self.root.winfo_screenwidth()}x{self.root.winfo_scree
nheight()}")
        self.root.configure(bg='#232333')

        # Screen dimensions and button sizing
        self.screen_width = self.root.winfo_screenwidth()
        self.screen_height = self.root.winfo_screenheight() -
TASKBAR_HEIGHT
        self.fullscreen_size = (
            math.floor(RATIO * (self.screen_height - BUTTON_SIZE)),
            self.screen_height - BUTTON_SIZE - 5
        )
        self.button_size = self.fullscreen_size

        # Track GUI elements and states
        self.buttons = {}
        self.images = []
        self.fullscreen_buttons = self.fullscreen_buttons_create()
        self.fullscreen_widget = None
        self.fullscreen_address = ''
        self.control_switch = False
        self.block_states = {}

        # Key handling and debouncing
        self.last_key_press_time = {}
        self.pressed_keys = set()
        self.pressed_keys_lock = threading.Lock()
```

```
        # Bind keyboard events
        self.root.bind("<KeyPress>", self.on_key_press)
        self.root.bind("<KeyRelease>", self.on_key_release)

        # Display when no clients are connected
        self.no_clients_label = None
        self.show_no_clients_message()

        # IP and port info label
        self.info_label = None
        self.add_info_button()

    def add_info_button(self):
        """
        Add an information icon to the bottom-right corner of the GUI,
        which displays the server's IP, port, and connected clients.
        """
        try:
            info_icon_image = Image.open(ICON_FILES["info"])
        except (FileNotFoundError, KeyError) as e:
            print("Error loading info icon:", e)
            return

        info_icon = ImageTk.PhotoImage(info_icon_image)
        self.info_label = tk.Label(self.root, image=info_icon,
background="#232333")
        self.info_label.image = info_icon  # Prevent garbage collection

        x = self.screen_width - self.info_label.image.width() - 10
        y = self.screen_height - self.info_label.image.height() - 10
        self.info_label.place(x=x, y=y)

        info = self.get_info_text()
        self.info_label.tooltip = InfoButtonToolTip(self.info_label, info)

    def get_info_text(self):
        """
        Generate the tooltip text for the info icon.

        :return: Tooltip text with IP, port, and client count.
        :rtype: str
        """
        return f"IP: {self.ip}\nPort: {self.port}\nConnections:
{len(self.buttons)}"

    def organize_screens(self):
        """
        Organize and place client screen buttons dynamically based on the
number of clients
        and available screen space, avoiding overlap with the info icon.
        """
        total_buttons = len(self.buttons)
        print(f"[GUI] Organizing {total_buttons} screens")

        if total_buttons == 0:
            self.show_no_clients_message()
            return

        if self.no_clients_label:
            self.no_clients_label.destroy()
            self.no_clients_label = None
```

```
        for button in self.buttons.values():
            button.place_forget()

        # Determine available space
        usable_width = self.screen_width
        usable_height = self.screen_height
        margin = 20

        if self.info_label:
            info_x = self.info_label.winfo_x()
            info_y = self.info_label.winfo_y()
            info_width = self.info_label.winfo_width()
            info_height = self.info_label.winfo_height()

            if info_y + info_height + margin > self.screen_height -
BUTTON_SIZE:
                usable_height = info_y - margin

            if info_x + info_width + margin > self.screen_width -
BUTTON_SIZE:
                usable_width = info_x - margin

        # Layout computation
        num_buttons_per_row = math.ceil(math.sqrt(total_buttons))
        number_of_rows = math.ceil(total_buttons / num_buttons_per_row)
        num_buttons_in_last_row = total_buttons % num_buttons_per_row or
num_buttons_per_row

        if num_buttons_per_row == number_of_rows:
            button_height = math.floor(usable_height / number_of_rows)
            button_width = math.floor(RATIO * button_height)
        else:
            button_width = math.floor(usable_width / num_buttons_per_row)
            button_height = math.floor(1 / RATIO * button_width)

        current_row = 0
        current_column = 0
        self.button_size = (button_width, button_height)

        for address, button in self.buttons.items():
            if current_row == number_of_rows - 1:
                row_buttons = num_buttons_in_last_row
            else:
                row_buttons = num_buttons_per_row

            x_offset = (self.screen_width - (button_width * row_buttons))
// 2
            y_offset = (usable_height - (button_height * number_of_rows))
// 2

            x_position = x_offset + (current_column * button_width)
            y_position = y_offset + (current_row * button_height)

            button.place(x=x_position, y=y_position, width=button_width,
height=button_height)

            current_column += 1
            if current_column >= num_buttons_per_row:
                current_column = 0
                current_row += 1
```

```
    def show_no_clients_message(self):
        """
        Display a message in the center of the screen indicating that no
clients are connected.
        """
        if not self.no_clients_label:
            self.no_clients_label = tk.Label(
                self.root,
                text="No Clients Connected",
                font=("Arial", 24),
                fg="white",
                bg="#232333"
            )
            self.no_clients_label.place(relx=0.5, rely=0.5,
anchor="center")

    def _add_screen_on_main_thread(self, address, preview_photo):
        """
        Add a new client screen preview button to the GUI on the main
thread.

        :param address: Client's network address.
        :type address: str
        :param preview_photo: Tkinter PhotoImage preview of the client's
screen.
        :type preview_photo: ImageTk.PhotoImage
        """
        print(f"[GUI] Added a new screen")
        button = tk.Button(self.root, image=preview_photo,
background="#232333",
                           command=lambda: self.toggle_fullscreen(address,
b""))
        button.image = preview_photo
        button.image_id = id(preview_photo)  # Track image to avoid
redundant updates
        self.buttons[address] = button
        self.images.append(preview_photo)  # Keep a reference to avoid
garbage collection

        info = self.get_info_text()
        self.info_label.tooltip.update_text(info)  # Update tooltip info

        # Organize screens and notify clients of new resize if no
fullscreen is active
        if not self.fullscreen_widget:
            self.organize_screens()
            self.command_queue.put((None,
f"resize:{self.button_size[0]}:{self.button_size[1]}"))

    def update_screen(self, address, image_bytes):
        """
        Receive updated screen image bytes from a client and schedule GUI
update.

        :param address: Client's network address.
        :type address: str
        :param image_bytes: Raw image data bytes of the client's screen.
        :type image_bytes: bytes
        """
        if self.shutdown_event.is_set():
```

```
            return  # Ignore updates if server is shutting down

        if address in self.kicked_addresses:
            print(f"[GUI] Ignoring frame update from kicked / disconnected
client {address}")
            return

        print(f"[GUI] Updating image for {address}")
        try:
            image = Image.open(BytesIO(image_bytes))

            preview_photo = None
            fullscreen_photo = None

            if self.fullscreen_address == address:
                # Resize fullscreen image if needed
                if image.size != self.fullscreen_size:
                    fullscreen_image = image.resize(self.fullscreen_size)
                    self.command_queue.put((address,
f"resize:{self.fullscreen_size[0]}:{self.fullscreen_size[1]}"))
                else:
                    fullscreen_image = image
                fullscreen_photo = ImageTk.PhotoImage(fullscreen_image)
            else:
                # Resize preview image if needed
                if image.size != self.button_size:
                    preview_image = image.resize(self.button_size)
                    self.command_queue.put((address,
f"resize:{self.button_size[0]}:{self.button_size[1]}"))
                else:
                    preview_image = image
                preview_photo = ImageTk.PhotoImage(preview_image)

            # Schedule GUI update on the main thread
            self.root.after(0, lambda:
self._update_screen_on_main_thread(address, preview_photo,
fullscreen_photo))
        except Exception as e:
            print(f"[GUI] Error processing image from {address}: {e}")

    def _update_screen_on_main_thread(self, address, preview_photo,
fullscreen_photo):
        """
        Update the GUI components for a client's screen on the main thread.

        :param address: Client's network address.
        :type address: str
        :param preview_photo: Tkinter PhotoImage for the preview button, or
None.
        :type preview_photo: ImageTk.PhotoImage or None
        :param fullscreen_photo: Tkinter PhotoImage for fullscreen display,
or None.
        :type fullscreen_photo: ImageTk.PhotoImage or None
        """
        if self.shutdown_event.is_set():
            return  # Prevent updates after shutdown

        if address in self.kicked_addresses:
            print(f"[GUI] Skipping update for kicked / disconnected client
{address}")
            return
```

```
        if fullscreen_photo is not None:
            # Update fullscreen widget if active and address matches
            if self.fullscreen_widget is not None and
self.fullscreen_address == address:
                if getattr(self.fullscreen_widget, "image_id", None) !=
id(fullscreen_photo):
                    self.fullscreen_widget.config(image=fullscreen_photo)
                    self.fullscreen_widget.image = fullscreen_photo
                    self.fullscreen_widget.image_id = id(fullscreen_photo)
            return  # No button update needed in fullscreen mode

        if preview_photo is not None:
            if address in self.buttons:
                button = self.buttons[address]

                if getattr(button, "image_id", None) != id(preview_photo):
                    button.config(image=preview_photo)
                    button.image = preview_photo
                    button.image_id = id(preview_photo)
            else:
                # Add new preview button if not found
                self._add_screen_on_main_thread(address, preview_photo)

    def toggle_fullscreen(self, address, image_bytes):
        """
        Toggle fullscreen preview mode for a selected client screen.

        :param address: Client's network address to display fullscreen.
        :type address: str
        :param image_bytes: Optional raw image data for fullscreen (used if
no cached button image).
        :type image_bytes: bytes
        """
        self.fullscreen_address = address

        # Pause all other clients to focus on fullscreen client
        self.command_queue.put((self.fullscreen_address, "pause"))

        # Hide all preview buttons while in fullscreen
        for btn in self.buttons.values():
            btn.place_forget()

        # Attempt to use cached preview image, else load from bytes
        button = self.buttons.get(address)
        if button and hasattr(button, "image"):
            photo = button.image
        else:
            try:
                image = Image.open(BytesIO(image_bytes))
                photo = ImageTk.PhotoImage(image)
            except Exception as e:
                print(f"[GUI] Failed to load image for {address}: {e}")
                return

        # Create and display fullscreen widget centered on screen
        self.fullscreen_widget = tk.Button(self.root, image=photo,
background="#232333")
        self.fullscreen_widget.image = photo
        self.fullscreen_widget.image_id = id(photo)
        self.fullscreen_widget.bind("<Button>", self.handle_mouse_click)
```

```
        self.fullscreen_widget.place(
            x=(self.screen_width - self.fullscreen_size[0]) // 2,
            y=0
        )

        self.fullscreen_buttons_show()  # Show fullscreen control buttons

        # Restore block/unblock state for this client
        is_blocked = self.block_states.get(address, False)
        self.update_block_button("unblock" if is_blocked else "block")

        # Reset control switch and update control button UI
        self.control_switch = False
        self.update_control_button("control start")

    def fullscreen_buttons_create(self):
        """
        Create fullscreen control buttons (pause, block, exit, etc.)
centered below the fullscreen image.

        Buttons are initially disabled and hidden; stored in a dict by
name.

        :return: Dictionary mapping button names to Tkinter Button objects.
        :rtype: dict[str, tk.Button]
        """
        buttons = {}
        total_buttons = len(FULLSCREEN_BUTTON_NAMES) + 1  # Extra one for
"exit" button
        offset = (self.screen_width - total_buttons * BUTTON_SIZE) // 2  #
Center buttons horizontally

        for i, name in enumerate(FULLSCREEN_BUTTON_NAMES + ["exit"]):
            x = offset + i * BUTTON_SIZE
            y = self.fullscreen_size[1] + 6  # Place just below fullscreen
image

            # Map button name to method or exit_fullscreen
            command = self.exit_fullscreen if name == "exit" else
getattr(self, name.replace(" ", "_"))

            # Load and resize icon image for button
            icon_image = Image.open(ICON_FILES[name]).resize((BUTTON_SIZE,
BUTTON_SIZE))
            icon = ImageTk.PhotoImage(icon_image)

            # Create button, initially disabled and hidden
            button = tk.Button(self.root, image=icon, background="#232333",
state=tk.DISABLED, command=command)
            button.image = icon  # Keep reference to prevent GC
            button.tooltip = ToolTip(button, TOOLTIPS[name])  # Attach
tooltip

            button.place(x=x, y=y)
            # Save placement info for later show/hide
            button.place_info = {"x": x, "y": y, "width": BUTTON_SIZE,
"height": BUTTON_SIZE}
            button.place_forget()  # Hide initially
            buttons[name] = button

        return buttons
```

```
    def fullscreen_buttons_show(self):
        """Show and enable all fullscreen control buttons."""
        for button in self.fullscreen_buttons.values():
            info = button.place_info
            button.place(**info)
            button.lift()  # Bring on top
            button.config(state=tk.NORMAL)

    def fullscreen_buttons_hide(self):
        """Hide and disable all fullscreen control buttons."""
        for button in self.fullscreen_buttons.values():
            button.place_forget()
            button.config(state=tk.DISABLED)

    def exit_fullscreen(self):
        """Exit fullscreen mode, restore buttons, and unpause the
client."""
        if self.fullscreen_widget:
            self.fullscreen_widget.destroy()
            self.fullscreen_widget = None

        # Notify client to unpause
        self.command_queue.put((self.fullscreen_address, "unpause"))

        self.fullscreen_address = None  # Clear fullscreen state

        self.control_switch = False
        self.fullscreen_buttons_hide()
        self.organize_screens()  # Show previews again

    def handle_mouse_click(self, event):
        """
        Handle mouse click events on fullscreen widget by sending scaled
coordinates to client.

        :param event: Tkinter mouse event.
        """
        if self.fullscreen_widget and self.fullscreen_address and
self.control_switch:
            x, y = event.x, event.y
            # Scale click coordinates from fullscreen size to client's
1920x1080 resolution
            scaled_x = int(x * 1920 / self.fullscreen_size[0])
            scaled_y = int(y * 1080 / self.fullscreen_size[1])
            scaled_x = min(scaled_x, 1920)
            scaled_y = min(scaled_y, 1080)
            self.command_queue.put((self.fullscreen_address,
f"button:{event.num}:{scaled_x}:{scaled_y}"))

    def on_key_press(self, event):
        """
        Handle key press events when controlling a fullscreen client.

        Implements debounce to prevent flooding, sends "key_down" commands.

        :param event: Tkinter keyboard event.
        """
        if self.fullscreen_widget and self.fullscreen_address and
self.control_switch:
            key = event.keysym
```

```
            now = time.time()

            with self.pressed_keys_lock:
                last_time = self.last_key_press_time.get(key, 0)
                if now - last_time < 0.03:  # 30 ms debounce
                    return
                self.last_key_press_time[key] = now

                if key not in self.pressed_keys:
                    try:

self.command_queue.put_nowait((self.fullscreen_address, f"key_down:{key}"))
                        self.pressed_keys.add(key)
                    except queue.Full:
                        print(f"[KeyPress] Queue full, couldn't send
key_down:{key}")

    def on_key_release(self, event):
        """
        Handle key release events when controlling a fullscreen client.

        Sends "key_up" commands and cleans up tracking.

        :param event: Tkinter keyboard event.
        """
        if self.fullscreen_widget and self.fullscreen_address and
self.control_switch:
            key = event.keysym
            with self.pressed_keys_lock:
                if key in self.pressed_keys:
                    try:

self.command_queue.put_nowait((self.fullscreen_address, f"key_up:{key}"))
                    except queue.Full:
                        print(f"[KeyRelease] Queue full, couldn't send
key_up:{key}")
                    self.pressed_keys.discard(key)
                    self.last_key_press_time.pop(key, None)

    def control_start(self):
        """Enable control mode for the fullscreen client."""
        if self.fullscreen_address:
            self.control_switch = True
            self.update_control_button("control stop")

    def control_stop(self):
        """Disable control mode for the fullscreen client."""
        if self.fullscreen_address:
            self.control_switch = False
            self.update_control_button("control start")

    def update_control_button(self, name):
        """
        Update the control button's icon and command.

        :param name: Button state name (e.g., "control start" or "control
stop").
        """
        btn = self.fullscreen_buttons["control start"]
        new_icon =
ImageTk.PhotoImage(Image.open(ICON_FILES[name]).resize((BUTTON_SIZE,
```

```
BUTTON_SIZE)))
        btn.config(image=new_icon, command=getattr(self, name.replace(" ",
"_")))
        btn.image = new_icon
        btn.tooltip.update_text(TOOLTIPS[name])

    def block(self):
        """
        Block input/control from the fullscreen client if not localhost.

        Updates button and internal block state.
        """
        if self.fullscreen_address and "127.0.0.1" not in
self.fullscreen_address:
            self.command_queue.put((self.fullscreen_address, "block"))
            self.block_states[self.fullscreen_address] = True
            self.update_block_button("unblock")

    def unblock(self):
        """
        Unblock input/control from the fullscreen client.

        Updates button and internal block state.
        """
        if self.fullscreen_address:
            self.command_queue.put((self.fullscreen_address, "unblock"))
            self.block_states[self.fullscreen_address] = False
            self.update_block_button("block")

    def update_block_button(self, name):
        """
        Update the block/unblock button icon and command.

        :param name: Button state name ("block" or "unblock").
        """
        btn = self.fullscreen_buttons["block"]
        new_icon =
ImageTk.PhotoImage(Image.open(ICON_FILES[name]).resize((BUTTON_SIZE,
BUTTON_SIZE)))
        btn.config(image=new_icon, command=getattr(self, name))
        btn.image = new_icon
        btn.tooltip.update_text(TOOLTIPS[name])

    def kick(self, address=None):
        """
        Kick a client, disconnecting it and cleaning up resources.

        :param address: Optional address to kick; defaults to fullscreen
client.
        """
        target = address or self.fullscreen_address
        if not target:
            print("[kick] No address to kick.")
            return

        self.command_queue.put((target, "kick"))

        with self.kicked_lock:
            self.kicked_addresses.add(target)

        self.cleanup(target)
```

```
    def cleanup(self, target):
        """
        Clean up resources related to a kicked or disconnected client.

        Removes buttons, block state, queued frames, and updates GUI.

        :param target: Client address to clean up.
        """
        if target in self.buttons:
            self.buttons[target].destroy()
            del self.buttons[target]

        if target in self.block_states:
            del self.block_states[target]

        info = self.get_info_text()
        self.info_label.tooltip.update_text(info)

        self.remove_frames_for_address(target)

        if self.fullscreen_address == target:
            self.exit_fullscreen()
        else:
            self.organize_screens()

    def on_close(self):
        """Handle application close event by signaling shutdown and
destroying the GUI."""
        self.shutdown_event.set()  # Notify other threads
        self.root.after(0, self.root.destroy)  # Safely exit Tkinter main
loop

    def allow_address(self, address):
        """Allow a previously kicked client address to reconnect."""
        with self.kicked_lock:
            if address in self.kicked_addresses:
                self.kicked_addresses.remove(address)

    def remove_frames_for_address(self, address):
        """
        Remove all queued frames associated with a specific client address.

        Prevents processing frames from kicked clients.

        :param address: Client address whose frames to remove.
        """
        temp_list = []

        # Drain queue, keep only frames not from the given address
        while not self.frame_queue.empty():
            try:
                item = self.frame_queue.get_nowait()
                if item[0] != address:
                    temp_list.append(item)
            except queue.Empty:
                break

        # Put valid frames back in the queue
        for item in temp_list:
            self.frame_queue.put(item)
```

```
    def run(self):
        """Start the Tkinter GUI event loop."""
        self.root.mainloop()

#ServerThreads.py
import threading
import Encryption
import socket
import queue
import zlib
import time
from FunctionsModule import recv_all  # Helper to receive fixed amount of
bytes from socket

FPS = 10  # Frames per second for limiting frame handling rate


class Server(threading.Thread):
    """
    TCP Server for handling encrypted client connections, receiving image
frames, and dispatching commands.

    :param ip: IP address to bind the server.
    :param port: Port number to listen on.
    :param app: Reference to the application object for GUI and state
interactions.
    :param command_queue: Queue for sending commands to clients.
    :param frame_queue: Queue for handling incoming frames from clients.
    :param close_callback: Optional callback to be called when the server
stops.
    """
    def __init__(self, ip, port, app, command_queue, frame_queue,
close_callback=None):
        super().__init__()
        self.ip = ip
        self.port = port
        self.app = app
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  #
TCP socket

        self.close_callback = close_callback  # Optional callback when
server closes

        # Generate RSA key pair for secure key exchange
        self.private_key, self.public_key = Encryption.generate_rsa_keys()
        # Generate AES symmetric key for encrypting communication after
exchange
        self.aes_key = Encryption.generate_aes_key()

        self.rsa_keys = {}  # Store clients' RSA public keys after key
exchange
        self.rsa_keys_lock = threading.Lock()  # Lock for thread-safe
access to rsa_keys

        self.client_sockets_and_threads = {}  # Map client address ->
(socket, handler thread)
        self.clients_lock = threading.Lock()  # Lock for thread-safe client
dict access

        self.command_queue = command_queue  # Queue of commands to send to
```

```
clients
        self.frame_queue = frame_queue  # Queue of received frames from
clients

        self.pause_event = threading.Event()  # Used to pause/unpause
clients
        self.condition = threading.Condition()  # Condition variable (not
shown used here)
        self.stop_event = threading.Event()  # Signal to stop the server
and threads

        # Thread that sends commands from the queue to clients
        self.command_sender =
CommandConsumer(self.client_sockets_and_threads, self.clients_lock,
self.command_queue,
                                                self.pause_event, self.app)
        # Thread that consumes frames from clients and updates the app UI
        self.frame_updater = FrameConsumer(self.app, self.frame_queue)

    def run(self):
        """
        Main server loop that listens for incoming connections, handles key
exchange, and starts handler threads.
        """
        stop_reason = "session closed"  # Default stop reason
        try:
            # Bind and listen on given IP and port
            self.server.bind((self.ip, self.port))
            self.server.listen(100)  # Allow backlog of 100 connections
            print(f"[Server] TCP Server listening on port: {self.port}")

            # Start the command sender and frame updater threads
            self.command_sender.start()
            self.frame_updater.start()

            # Main loop to accept new clients and handle key exchange
            while not self.stop_event.is_set():
                client_socket, client_address = self.key_exchange()
                if client_socket:
                    print(f"[Server] Accepted connection from
{client_address}")

                    # Notify client of UI button size for resizing input
areas
                    button_width, button_height = self.app.button_size
                    self.command_queue.put((client_address,
f"resize:{button_width}:{button_height}"))

                    # If paused, send pause command to the new client
immediately
                    if self.pause_event.is_set():
                        try:
                            iv, encrypted_command =
Encryption.encrypt_aes(self.aes_key, b"pause")

client_socket.send(len(encrypted_command).to_bytes(4, 'big'))
                            client_socket.send(iv)
                            client_socket.send(encrypted_command)
                            print("[Server] Pause sent to new client")
                        except Exception as e:
                            print(f"[Server] Failed to send pause to new
```

```
client {client_address}: {e}")

                    # Create and start a client handler thread to receive
frames from this client
                    handler = ClientHandler(client_socket, client_address,
self.frame_queue, self.aes_key, self.app)
                    with self.clients_lock:
                        self.client_sockets_and_threads[client_address] =
(client_socket, handler)
                    handler.start()

        except Exception as e:
            stop_reason = "server crashed"
            print(f"[Server] Unexpected exception: {e}")
        finally:
            self.stop(stop_reason)

    def stop(self, stop_reason):
        """
        Stops the server and all running threads and connections.

        :param stop_reason: A message indicating the reason for stopping
the server.
        """
        # Signal all threads and connections to stop
        self.stop_event.set()

        # Close all client connections and stop their threads
        with self.clients_lock:
            for sock, handler in self.client_sockets_and_threads.values():
                handler.stop()
                sock.close()

        # Stop the command sender and frame updater threads
        self.command_sender.stop()
        self.frame_updater.stop()

        # Close the server socket
        self.server.close()

        # Call the optional close callback (e.g., for UI updates)
        if self.close_callback:
            self.close_callback(stop_reason)

    def key_exchange(self):
        """
        Performs RSA key exchange with a new client and sends the AES key.

        :returns: Tuple of (client_socket, client_address) if successful,
else (None, None).
        """
        # Perform RSA key exchange with a newly connecting client
        if self.stop_event.is_set():
            return None, None

        try:
            client_socket, client_address = self.server.accept()

            if self.stop_event.is_set():
                client_socket.close()
                return None, None
```

```
            # Set short timeout for key exchange phase
            client_socket.settimeout(5)
            try:
                # Receive length and then client's public key bytes
                client_key_len = int.from_bytes(recv_all(client_socket, 4),
'big')
                client_key_bytes = recv_all(client_socket, client_key_len)
                # Deserialize client RSA public key
                client_key =
Encryption.deserialize_public_key(client_key_bytes)
            except socket.timeout:
                client_socket.close()
                return None, None

            # Send server's public key to client
            server_key_bytes =
Encryption.serialize_public_key(self.public_key)
            client_socket.send(len(server_key_bytes).to_bytes(4, 'big'))
            client_socket.send(server_key_bytes)

            # Encrypt AES key with client's RSA public key and send
            encrypted_aes_key = Encryption.rsa_encrypt(client_key,
self.aes_key)
            client_socket.send(len(encrypted_aes_key).to_bytes(4, 'big'))
            client_socket.send(encrypted_aes_key)

            client_socket.settimeout(None)  # Remove timeout for normal
operation

            # Save the client's public key for future reference
            with self.rsa_keys_lock:
                self.rsa_keys[client_address] = client_key

            # Allow the app to accept input/frames from this address
            self.app.allow_address(client_address)

            return client_socket, client_address
        except Exception as e:
            print(f"[Server] Key exchange error: {e}")
            return None, None


class ClientHandler(threading.Thread):
    """
    Handles receiving frames from an individual client over a secure
channel.

    :param client_socket: The socket connected to the client.
    :param client_address: Tuple representing the client's IP and port.
    :param frame_queue: Queue where the received frames are placed.
    :param aes_key: AES key for decrypting received data.
    :param app: Reference to the application for state and UI updates.
    """
    def __init__(self, client_socket, client_address, frame_queue, aes_key,
app):
        super().__init__()
        self.client_socket = client_socket
        self.client_address = client_address
        self.frame_queue = frame_queue
        self.aes_key = aes_key
```

```
        self.app = app
        self.stop_event = threading.Event()
        self.min_frame_interval = 1.0 / FPS  # Enforce max FPS
        self.last_frame_time = 0  # Timestamp of last received frame

    def run(self):
        """
        Main loop that receives encrypted and compressed image frames from
the client,
        decrypts and decompresses them, and puts them in the frame queue.
        """
        try:
            while not self.stop_event.is_set():
                current_time = time.time()
                elapsed = current_time - self.last_frame_time

                # Sleep if frames are arriving too fast (enforce FPS limit)
                if elapsed < self.min_frame_interval:
                    time.sleep(self.min_frame_interval - elapsed)

                # Receive size of incoming frame (8 bytes)
                image_size = int.from_bytes(recv_all(self.client_socket,
8), byteorder='big')
                if image_size == 0:
                    raise ConnectionError("Received empty frame size")

                # Receive AES IV and encrypted image bytes
                iv = recv_all(self.client_socket, 16)
                encrypted_img = recv_all(self.client_socket, image_size)

                # Decrypt and decompress the frame image data
                frame = Encryption.decrypt_aes(self.aes_key, iv,
encrypted_img)
                frame = zlib.decompress(frame)

                timestamp = time.time()
                # Put the frame into the shared queue with client address
and timestamp
                self.frame_queue.put((self.client_address, frame,
timestamp))
                self.last_frame_time = timestamp

        except Exception as e:
            print(f"[ClientHandler] {self.client_address} error: {e}")
            if not self.app.shutdown_event.is_set():
                try:
                    # Request app cleanup of this client from the GUI
thread
                    self.app.root.after(0, lambda:
self.app.cleanup(self.client_address))
                except RuntimeError as tk_err:
                    print(f"[ClientHandler] GUI already closed for
{self.client_address}: {tk_err}")
        finally:
            try:
                self.client_socket.close()
            except OSError as e:
                print(f"[ClientHandler] Error closing socket for
{self.client_address}: {e}")

    def stop(self):
```

```
        """
        Signals the thread to stop receiving frames.
        """
        # Signal to stop receiving frames and close the thread
        self.stop_event.set()


class FrameConsumer(threading.Thread):
    """
    Consumes image frames from the queue and updates the UI accordingly.

    :param app: Reference to the application for UI updates.
    :param frame_queue: Queue containing incoming frames from clients.
    """
    def __init__(self, app, frame_queue):
        super().__init__()
        self.app = app
        self.frame_queue = frame_queue
        self.max_frame_age = 0.5  # Discard frames older than 0.5 seconds
        self.stop_event = threading.Event()

    def run(self):
        """
        Main loop that consumes frames and triggers screen updates if
frames are fresh.
        """
        while not self.stop_event.is_set():
            try:
                # Wait for a new frame from any client, timeout after 1
second
                client_address, frame, timestamp =
self.frame_queue.get(timeout=1)
                # Only update screen if frame is fresh enough
                if time.time() - timestamp <= self.max_frame_age:
                    # Update screen only if no fullscreen widget or
fullscreen client matches frame client
                    if not self.app.fullscreen_widget or
self.app.fullscreen_address == client_address:
                        self.app.update_screen(client_address, frame)
                else:
                    print(f"[FrameConsumer] Dropped stale frame from
{client_address}")
            except queue.Empty:
                continue  # No frame available, continue loop

    def stop(self):
        """
        Signals the frame consumer to stop processing frames.
        """
        # Signal to stop processing frames
        self.stop_event.set()


class CommandConsumer(threading.Thread):
    """
    Sends encrypted commands to one or more clients based on the queue
input.

    :param clients_dict: Dictionary mapping client addresses to (socket,
handler thread).
    :param clients_lock: Lock for thread-safe access to the client
```

```
dictionary.
    :param command_queue: Queue containing commands to be sent.
    :param pause_event: Event signaling whether the system is paused.
    :param app: Reference to the application for callbacks and UI handling.
    """

    def __init__(self, clients_dict, clients_lock, command_queue,
pause_event, app):
        super().__init__()
        self.app = app
        self.clients_dict = clients_dict  # Dictionary of client sockets
and handlers
        self.clients_lock = clients_lock
        self.command_queue = command_queue  # Queue of commands to send to
clients
        self.pause_event = pause_event
        self.stop_event = threading.Event()

    def run(self):
        """
        Main loop that listens for commands in the queue and dispatches
them to appropriate clients.
        Supports pause, unpause, resize, and other custom commands.
        """
        while not self.stop_event.is_set():
            try:
                # Wait for next command, timeout after 1 second
                client_address, cmd = self.command_queue.get(timeout=1)
                print(f"[CommandConsumer] Command: {cmd}")

                if cmd in ("pause", "unpause"):
                    # Handle global pause/unpause commands
                    if cmd == "pause":
                        self.pause_event.set()
                    else:
                        self.pause_event.clear()

                    with self.clients_lock:
                        # Send pause/unpause command to all clients except
sender
                        for addr, (sock, thread) in
list(self.clients_dict.items()):
                            if addr != client_address:
                                try:
                                    iv, encrypted_cmd =
Encryption.encrypt_aes(thread.aes_key, cmd.encode())

sock.send(len(encrypted_cmd).to_bytes(4, 'big'))
                                    sock.send(iv)
                                    sock.send(encrypted_cmd)
                                except Exception as e:
                                    print(f"[CommandConsumer] Failed to
send command to {addr}: {e}")
                                    thread.stop()
                                    del self.clients_dict[addr]
                                    self.app.root.after(0, lambda:
self.app.cleanup(addr))
                    continue

                elif cmd.startswith("resize"):
                    # Handle resize commands either for a specific client
```

```
or all
                    with self.clients_lock:
                        targets = []
                        if client_address is None:
                            # Broadcast to all clients
                            targets = list(self.clients_dict.items())
                        elif client_address in self.clients_dict:
                            # Send only to specified client
                            targets = [(client_address,
self.clients_dict[client_address])]

                        for addr, (sock, thread) in targets:
                            try:
                                iv, encrypted_cmd =
Encryption.encrypt_aes(thread.aes_key, cmd.encode())
                                sock.send(len(encrypted_cmd).to_bytes(4,
'big'))
                                sock.send(iv)
                                sock.send(encrypted_cmd)
                            except Exception as e:
                                print(f"[CommandConsumer] Failed to send
resize to {addr}: {e}")
                                thread.stop()
                                del self.clients_dict[addr]
                                self.app.root.after(0, lambda:
self.app.cleanup(addr))
                    continue

                else:
                    # Handle other commands directed at specific client
                    with self.clients_lock:
                        if client_address in self.clients_dict:
                            sock, thread =
self.clients_dict[client_address]
                            aes_key = thread.aes_key
                            try:
                                iv, encrypted_cmd =
Encryption.encrypt_aes(aes_key, cmd.encode())
                                sock.send(len(encrypted_cmd).to_bytes(4,
'big'))
                                sock.send(iv)
                                sock.send(encrypted_cmd)

                                # If the command is "kick", stop the client
handler and remove client
                                if cmd == "kick":
                                    thread.stop()
                                    del self.clients_dict[client_address]

                            except Exception as e:
                                print(f"[CommandConsumer] Unexpected error
with {client_address}: {e}")
                                thread.stop()
                                del self.clients_dict[client_address]
                                self.app.root.after(0, lambda:
self.app.kick(client_address))
            except queue.Empty:
                continue  # No command available, continue loop

    def stop(self):
        """
```

```
        Signals the command consumer to stop processing commands.
        """
        # Signal to stop processing commands
        self.stop_event.set()
```