

**פרויקט גמר הגנת הסייבר**

**נושא העבודה:**ניהול כיתת מחשבים

**שם מלא:** דניאל כהן

**תעודת זהות:**  217385277

**שם בית ספר ועיר:** קריית החינוך ע"ש עמוס דה-שליט, רחובות

**שם המנחה :** הילה גורן ברנע

**מועד הגשה :** מאי 2025

תוכן עניינים

**.1 מבוא ......................................................................................................2**

1.1 נושא העבודה......................................................................................2

1.2 מטרות מרכזיות...................................................................................2

1.3 רציונל................................................................................................2

1.4 קישור לחומר הנלמד.............................................................................3

**.2 תיאוריה................................................................................................. 4**

2.1 תיאוריה............................................................................................ 4

2.2 מוצרים קיימים....................................................................................6

**.3 תוצר סופי …......................................................................................... 6**

3.1 תיאור הפרויקט ................................................................................. 6

3.2 אלגוריתמים עיקריים ......................................................................... 11

3.3 דרישות ומגבלות מערכת..................................................................... 17

3.4 התייחסות לנושא אבטחה ....................................................................18

3.5 ממשק משתמש .................................................................................19

3.6 תרחישים עיקריים ............................................................................. 23

**.4 תהליך כתיבת הפרויקט 24 .........................................................................**

4.1 תהליך הפרויקט................................................................................ 24

4.2 אתגרים ואופציות שונות למימוש........................................................... 25

**.5 מרכיבי פתרון........................................................................................ 26**

5.1 תיחום הפרויקט................................................................................. 26

5.2 סביבת העבודה )טכנולוגיה.................................................................( 27

5.3 מבט טופולוגי 28.....................................................................................

5.4 מבנה נתונים..................................................................................... 29

5.5 מסד נתונים...................................................................................... 31

5.6 מבט מודולרי..................................................................................... 33

5.7 פירוט מודלים עיקריים........................................................................ 33

**.6 תסריטי בדיקה...................................................................................... 39**

6.1 דגשים בבדיקה................................................................................. 39

6.2 תסריטי בדיקה עיקריים...................................................................... 39

**.7 רפלקציה.............................................................................................. 40**

7.1 לוח זמנים מוערך לניהול הפרויקט 40 ........................................................

7.2 אתגרים ותרומה אישית...................................................................... 41

7.3 תובנות............................................................................................ 41

**.8 הוראות התקנה ותפעול.......................................................................... 42**

8.1 תצורה ודרישות קדם.......................................................................... 42

8.2 התקנה 42 ............................................................................................

**.9 ביבליוגרפיה......................................................................................... 44**

**.10 נספחים............................................................................................. 45**

1. מבוא
2. נושא העבודה

העבודה עוסקת בפיתוח מערכת לניהול כיתה מרחוק באמצעות שפת .Python המערכת נועדה לאפשר למורה לפקח על מחשבי התלמידים בזמן השיעור, לשלוח להם קבצים ומבחנים, לקבל מהם צילומי מסך, לחסום את המקלדת והעכבר במקרה הצורך, לשלוח הודעות ולוודא שהשיעור מתנהל בצורה ממוקדת וללא הסחות דעת. הפרויקט בנוי משלושה רכיבים מרכזיים: ממשק Admin גרפי שמיועד למורה, שרת מאובטח המתווך בין המורה לתלמידים, ואפליקציית לקוח שמותקנת אצל כל תלמיד. התקשורת בין הרכיבים מוצפנת באמצעות ,SSL והמערכת שומרת מידע חשוב של ציוני התלמידים במסד נתונים באמצעות .SQLAlchemy הפרויקט מדגים שילוב בין תכנות ממשקים, עבודה עם רשתות ואבטחת מידע, ומציע פתרון מעשי לשיפור חוויית הלמידה בכיתה דיגיטלית.

1. מטרות מרכזיות

המטרות היישומיות של הפרויקט הן:

• לאפשר שליטה מרחוק (remote desktop control) של המורה על מחשבי התלמידים במהלך השיעור.  
• לאפשר שליחה וקבלה של קבצים, הודעות ומבחנים בין המורה לתלמידים.  
• לספק למורה כלים לפיקוח בזמן אמת, כולל צפייה בצילומי מסך ממחשבי התלמידים.  
• לשפר את הסדר והקשב בכיתה באמצעות חסימת מקלדת ושליחת התראות לפי הצורך.

המטרות האישיות של העבודה הן:

• לפתח הבנה עמוקה בעבודה עם רשתות, פרוטוקולים ותקשורת מאובטח .(SSL)   
 • להעמיק את השליטה בשפת Python ובספריות מתקדמות כמו Tkinter ו.SQLAlchemy-  
 • לשפר יכולות בתכנון, בנייה ותחזוקה של מערכת מורכבת הכוללת ממשק משתמש, שרת ולקוח.

1. רציונל

בחרתי לעסוק בפיתוח מערכת לניהול כיתה מרחוק משום שמדובר בתחום שמשלב בין טכנולוגיה פרקטית לבין אתגרי תכנות מורכבים ומעניינים. ההשראה הראשונית הגיעה מכיתת המחשבים בבית הספר שלי, שבה ראיתי כיצד המורה משתמש במערכת ניהול כיתה כדי לפקח על פעילות התלמידים. ההתנסות הזו עוררה בי סקרנות להבין כיצד מערכות כאלה פועלות "מאחורי הקלעים" – איך הן מתקשרות עם מחשבים אחרים, כיצד הן מצליחות לשלוח פקודות, לצפות במסכים ולהגיב בזמן אמת.

התחום הזה מעניין אותי במיוחד כי הוא משלב בין כמה תחומים חשובים במדעי המחשב: ממשקי משתמש (GUI), תכנות רשת, אבטחת מידע, ומבני נתונים. הוא גם מציע פתרון ממשי לבעיה נפוצה – חוסר ריכוז של תלמידים בזמן שיעור מול מחשב. מעבר לכך, ראיתי בפרויקט הזה הזדמנות ליישם בפועל את הידע שצברתי בלימודים וליצור מערכת שלמה שמדמה מקרה אמיתי מהחיים, דבר שמרגש אותי מאוד כמתכנת צעיר.

1. קישור לחומר הנלמד

הפרויקט שלי מתקשר באופן ישיר לחומר הלימוד במגמת הסייבר, ובמיוחד לנושאים שנלמדו בכיתה י"א, כמו רשתות, תקשורת ופרוטוקולים. המערכת שפיתחתי מבוססת על תקשורת בין מחשבים ברשת מקומית באמצעות sockets בפרוטוקול ,TCP אשר פועל בשכבת התעבורה של מודל ה ,OSI -בנוסף, בניתי פרוטוקול יישומי פשוט מעל,TCP שמאפשר שליחה של פקודות כמו קבלת צילום מסך, חסימת מקלדת ושליחת קבצים – מה ששייך לשכבת היישום במודל ה.OSI -

מעבר לכך, נעשה בפרויקט שימוש ב- SSL לצורך הצפנה ואבטחת המידע המועבר בין רכיבי המערכת (Client ,Admin, Server) בהתאם לנושאים שנלמדו באבטחת מידע. כמו כן, נעשה שימוש ב־ SQLAlchemy לעבודה עם בסיס נתונים, מה שמאפשר שמירה ועדכון של נתוני תלמידים – בהתאם לחומר שנלמד על .SQL

בנוסף, בפרויקט משולב שימוש בתהליכונים (Threads), כך שכל תלמיד שמתחבר לשרת מקבל Thread ייעודי. הדבר מאפשר לשרת לבצע מספר פעולות במקביל – למשל, לשלוח קובץ לתלמיד אחד, לקבל צילום מסך מתלמיד שני ולהציג מבחן מתלמיד שלישי – מבלי שהמערכת "תיתקע". זהו יישום ישיר של עקרונות ריבוי משימות שנלמדו בכיתה י"ב במסגרת נושא מערכות הפעלה.

לסיכום, רוב רכיבי הפרויקט – התקשורת, אבטחת המידע, בסיסי הנתונים וריבוי התהליכים- נשענים על נושאים שנלמדו לעומק בכיתה י"א, והם מודגמים כאן בצורה מעשית וברורה.

1. תיאוריה
   1. תיאוריה

הפרויקט שפיתחתי עוסק בבניית מערכת שליטה ובקרה מרחוק על מחשבים ברשת מקומית, עם דגש על סביבות חינוכיות כמו כיתות מחשבים. המערכת נבנתה מתוך צורך ממשי שקיים במוסדות חינוך – לאפשר למורים שליטה אפקטיבית על מחשבי תלמידים, במטרה לשמור על סדר, למנוע הסחות דעת ולשפר את תהליך הלמידה בעידן דיגיטלי. את ההשראה הראשונית לפרויקט קיבלתי ממערכת ניהול הכיתה שנמצאת במעבדת המחשבים בבית הספר שלי, שדרכה נחשפתי ליתרונות של שליטה מרחוק (remote desktop control) על מחשבים – אך גם לחסרונות כמו חוסר גמישות והתאמה.

מהבחינה התיאורטית, הפרויקט נשען על מודלים מוכרים במדעי המחשב, בעיקר מודל Client-Server – מודל נפוץ בתחום התקשורת והרשתות. במודל זה, שרת מרכזי (Server) אחראי על ניהול הנתונים, קבלת הפקודות מהמנהל ( (Adminושליחתן ללקוחות (Clients), שהם מחשבים אחרים ברשת. כל קליינט מקבל פקודות מהשרת ומחזיר אליו מידע בהתאם לפקודה – לדוגמה, צילום מסך, סטטוס, או הודעת מערכת. בנוסף, נבנה ממשק ניהול גרפי (Admin GUI) המאפשר למשתמש לשלוט במערכת בצורה אינטואיטיבית.

מבחינת שכבות רשת, המערכת מתבססת על שכבת התחבורה (Transport Layer) של מודל ה־OSI, באמצעות פרוטוקול TCP/IP. זהו פרוטוקול אמין ומבוסס- חיבור שמאפשר שליחה מדויקת של נתונים בין השרת ללקוחות. מעליו, פותח פרוטוקול יישום (Application Protocol) פרטי, שנועד להגדיר את סוגי הפקודות – נעילה, שחרור, שליחת קובץ, הצגת הודעה ועוד – ואת הפורמט שבו הן מועברות. כל פקודה כוללת כותרת והגדרות תוכן, ונשלחת בתוך ערוץ TCP מאובטח.

בכדי להבטיח שהמערכת תוכל לנהל מספר מחשבים במקביל, נעשה שימוש בתהליכונים (Threads). כל קליינט שמתחבר לשרת מנוהל על ידי Thread נפרד, שמאזין לפקודות ומבצע אותן באופן עצמאי. שימוש זה ב־Multithreading מאפשר לשרת לבצע פעולות במקביל על קליינטים שונים – לדוגמה, לקבל צילום מסך מתלמיד אחד, לשלוח מבחן לאחר, ולנעול מקלדת לתלמיד שלישי – וכל זאת ללא הפרעות או השהיות. זהו יישום ישיר של נושאים שנלמדו במסגרת מערכות הפעלה, בעיקר בנושא ריבוי תהליכים וניהול משאבים.

היבט נוסף בתיאוריה של הפרויקט הוא אבטחת המידע. כל תקשורת בין השרת ללקוחות מוצפנת באמצעות SSL (Secure Sockets Layer) – פרוטוקול שמספק הצפנה ו־Authentication, במטרה למנוע יירוט של פקודות או התחזות של גורמים זדוניים. זהו יישום ישיר של עקרונות מתחום הסייבר ואבטחת מידע, שנלמדו במגמה. שימוש נוסף נעשה בספריית SQLAlchemy לעבודה עם בסיס נתונים, המשמש לאחסון מידע על מחשבים, סטטוסים, מבחנים שהוגשו, היסטוריית פעולות ועוד – בהתאם לנושאים שנלמדו ב־SQL בכיתה.

הפרויקט משתמש גם בספריות מתקדמות נוספות:

ctypes – לשליטה על מקלדת ועכבר (לצורך נעילה)

Pillow – לצילום מסך ושליחתו מהלקוח לשרת

Tkinter – ליצירת הממשק הגרפי למנהל

socket – לבניית ערוץ התקשורת בין רכיבי המערכת

os / shutil – לניהול קבצים ושליחתם בין מחשבים

במובן רחב יותר, תיאוריית הפרויקט מבוססת על תפיסת מערכות מבוזרות – מערכת שבה יש ריבוי מחשבים עצמאיים, שפועלים יחד לצורך ביצוע משימה כוללת. כל לקוח (Client) פועל באופן עצמאי, אך תלוי בהוראות מהשרת. המערכת כוללת מנגנוני ניטור, שליטה, בקרה ואינטראקציה דו־כיוונית, בדומה למערכות ניהול IT מקצועיות בארגונים.

לבסוף, חשוב לציין שהתיאוריה של הפרויקט אינה עומדת בפני עצמה – היא משתלבת עם הצרכים המעשיים שנולדו מהשטח. תלמידים היום עובדים מול מחשב, והמורה זקוק לכלים שיאפשרו לו לעקוב, להנחות ולפקח על מה שקורה במחשבים אלו. הפרויקט נותן לכך מענה אפקטיבי, מבוסס עקרונות מדעיים מוצקים, עם פוטנציאל אמיתי ליישום.

1. מוצרים קיימים

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | הפרויקט שלי | NetSupport School | LanSchool | Veyon |
| סוג רשת |  | |  | | --- | | רשת מקומית (LAN), TCP/IP |  |  | | --- | |  | | LAN / VPN | LAN | LAN |
| שפת פיתוח |  | Python (קוד פתוח) | C++ / .NET (קוד סגור) | C++ / Java (קוד סגור) | C++ (קוד פתוח) |
| פתיחת מבחנים אמריקאיים + בדיקה אוטומטית |  | כן – בנוי במערכת | לא | לא | לא |
| צפייה במסך בזמן אמת |  | כן – צילומי מסך מתעדכנים | כן – צפייה חיה | כן – צפייה חיה | כן – צפייה חיה |
| אבטחת תקשורת |  | כן – SSL | כן – הצפנה ואימות | כן – בקרת הרשאות | כן – TLS + מפתחות ציבוריים/פרטיים |
| עלות חודשית (בשקלים) |  | חינם | 27.5 ש"ח – 55 ש"ח | 16.5 ש"ח – 33 ש"ח | חינם |

1. תוצר סופי
   1. תיאור הפרויקט

הפרויקט שלי עוסק בפיתוח מערכת לניהול מרחוק של מחשבים בכיתת מחשבים בבית ספר, בדגש על שליטה וניטור יעילים על עמדות תלמידים. המערכת כוללת ממשק גרפי ידידותי למורה (Admin) דרכו ניתן לבצע המון פעולות על התלמידים (Clients). הפעולות אשר המורה (Admin) מבצעת עוברות אל השרת (Server) שמתקשר בין המורה אל התלמידים ומעביר בניהם הודעות, קבצים, ציונים ועוד....

הפרויקט מבוסס על תקשורת מאובטחת בפרוטוקול SSL וממומש בשפת Python תוך שימוש במודולים מתקדמים כמו socket, tkinter, ctypes, Pillow ו־sqlite3. המערכת נבנתה כמבנה שרת-לקוח (Client-Server) וממשק ניהול אשר עומד לצד המורה (.(Admin המערכת כוללת שלושה רכיבים עיקריים:

**שרת (Server):**

רכיב השרת הוא הלב הפועם של המערכת – הוא אחראי על ניהול כל התקשורת בין המורה (Admin) לבין מחשבי התלמידים (Clients), תוך שמירה על סדר, סנכרון, ואבטחת מידע.

**תפקידי השרת:**

**קבלת חיבורים מהקליינטים:**

השרת מאזין באופן קבוע לחיבורים נכנסים. כל מחשב תלמיד שמפעיל את תוכנת הלקוח שולח בקשת התחברות לשרת, אשר מזהה אותו, מאמת אותו, ומוסיף אותו לרשימת המחשבים המחוברים.

**תקשורת מוצפנת (SSL encrypted communication):**

התקשורת בין השרת לכל לקוח (וגם מול ה-Admin) מתבצעת על גבי חיבור SSL, אשר מבטיח שמידע רגיש כמו פקודות, קבצים או תמונות מסך לא ייחשפו לגורמים זדוניים ברשת.

**הפצת פקודות מהמורה ללקוחות:**

השרת משמש כמגשר – כאשר המורה שולח פקודה (למשל לנעול מקלדת, לשלוח מבחן, או לשלוח הודעה), השרת מנתב אותה ללקוח המתאים ומוודא ביצוע תקין. ניתן לנהל גם מספר לקוחות במקביל באמצעות תהליכונים (Threads).

**ניהול סטטוס בזמן אמת:**

השרת שומר רשימה מעודכנת של כל הקליינטים המחוברים – כולל האם הם פעילים, מתי התחברו, האם השיבו למבחן, מתי נשלחה אליהם הפקודה האחרונה, ועוד. כך המורה רואה את המצב העדכני של כל מחשב בכיתה.

**שמירת נתונים במסד נתונים:**

השרת מחובר ל־מסד נתונים (SQLAlchemy) שבו נשמרים פרטי התחברות, פעילות תלמידים, צילומי מסך, תשובות למבחנים וציון אוטומטי. ניתן להפיק ממנו נתונים לניתוח לאחר השיעור.

**ניהול תקלות ונפילות:**

השרת מתמודד עם תרחישים של ניתוק פתאומי של לקוח או קריסת תוכנה, ושולח דיווח ל-Admin. בנוסף, הוא מנסה לחדש את הקשר או לעדכן את סטטוס התלמיד בהתאם.

**ממשק מנהל (Admin GUI):**

ממשק המנהל הוא יישום גרפי המיועד לשימוש על ידי המורה או מנהל הכיתה, ומספק שליטה מלאה על מחשבי התלמידים המחוברים למערכת. ממשק זה מהווה את נקודת המפגש בין המשתמש האנושי (המורה) לבין השרת, ומאפשר ניהול מרכזי של כל הפעולות הפדגוגיות והטכניות הנדרשות במהלך השיעור. בממשק ישנם כפתורים אשר כל כפתור אחראי על פעולה אחרת.

**תפקידי ממשק המנהל:**

**שליטה מלאה על מחשבים בכיתה:**

נעילה ושחרור של מקלדת ועכבר בכל מחשב תלמיד באופן פרטני או קבוצתי.

שליחת הודעות פופ־אפ ברורות למסך התלמיד (למשל הערות או הוראות).

צפייה בזמן אמת בצילומי מסך של תלמידים, עם יכולת לעבור בין תלמידים בלחיצת כפתור.

**עבודה עם מבחנים:**

פתיחת מבחנים אמריקאיים בקובץ JSON או בפורמט מוגדר מראש, שליחתם לכלל התלמידים.

קבלת תשובות באופן אוטומטי מכל תלמיד, שמירתן במסד הנתונים, וביצוע בדיקה אוטומטית עם חיווי ציון מיידי.

אפשרות לייצא את התשובות והציונים לקובץ או להדפיסם.

**כלי ביקורת והבהרה:**

תכונה ייחודית המאפשרת למורה לצייר על צילום מסך של תלמיד (למשל בעיות בקוד או הסברים) ולשלוח אותו חזרה לתלמיד להבהרה.

פעולה זו מספקת משוב אישי ומיידי בצורה אינטראקטיבית.

**שליחת קבצים והודעות:**

שליחת קבצי טקסט, תמונות או מבחנים (.txt \ .png) לכלל התלמידים או לתלמיד מסוים.

שליחת הודעות לתלמיד\ים וקבלת הודעות מתלמידים תוך הצגתם על מסך המורה.

**ניהול סטטוס התלמידים:**

בממשק ישנו איזור מרובע (ListBox) אשר מציג את כל שמות התלמידים המחוברים ואת הIP שלהם (אם תלמיד מתנתק – הוא נעלם מה .(ListBox -

ישנה גם אופציה להוציא תלמיד ולסגור את התקשורת איתו.

**תכנון ונוחות עבודה:**

הממשק בנוי בצורה אינטואיטיבית ונגישה, ומפותח באמצעות ספריית Tkinter של Python.

פעולות מרובות מתבצעות במקביל בעזרת תהליכונים (Threads), כך שהממשק נשאר מהיר ואינו נתקע גם כאשר מתנהלות פעולות רבות.

**מסך תצוגה ((Canvas:**

בממשק המורה ישנו מסך תצוגה אשר עליו ניתן להציג:

- צילומי מסך מתלמידים

- סטטוס מבחנים בזמן אמת אשר מציג את כל התלמידים על המסך וניתן לראות בצבע ירוק- את התלמידים שנמצאים במבחן בזמן אמת, ובצבע אדום- תלמידים שלא נמצאים במבחן. (כמובן שישנן פעולות שלא ניתן לבצע על תלמיד שנמצא במבחן).

- ציוני תלמידים אשר המורה (Admin) מבקש מהשרת (Server) ומציג אותם על גבי מסך התצוגה.

**סיכום:**

ממשק המנהל מספק סט כלים עשיר ונגיש לניהול כיתה דיגיטלית בצורה חכמה, נוחה ומאובטחת. הוא מפשט את עבודת ההוראה בכיתה ממוחשבת, מגביר את השליטה של המורה על הנעשה בזמן אמת, ומשפר את יכולת הניטור, ההנחיה והבקרה באופן משמעותי.

**תלמיד (Client):**

התלמיד (Client) הוא יישום שמותקן מראש על כל עמדת מחשב תלמיד בכיתה. הוא פועל ברקע ומאזין באופן תמידי להוראות מהשרת המרכזי, באמצעות תקשורת מאובטחת ומוצפנת בפרוטוקול SSL, תוך שמירה על תגובתיות מהירה וביצועים גבוהים הודות לשימוש בתהליכונים (threads) לעבודה מקבילית.

**תפקידי הקליינט:**

**ביצוע פעולות לפי פקודות מהשרת:**

חסימה או שחרור של מקלדת ועכבר באופן מיידי לפי הוראת המורה.

שליחת צילומי מסך של התחנה לשרת, לפי דרישה ספציפית.

פתיחה אוטומטית של מבחנים אמריקאיים, הצגתם, איסוף תשובות, בדיקה אוטומטית וקבלת ציון מכך ושליחת הציון חזרה למורה.

קבלת הודעות טקסט או קבצים מהמורה והצגתן לתלמיד בפורמט ברור.

**פעולות אשר התלמיד יכול לבצע:**

שליחת הודעות למורה.

הורדת קבצים שהתקבלו מהמורה באמצעות כפתור.

לענות על מבחן אמריקאי שנפתח על ידי המורה.

**שליחת מידע לשרת:**

דיווח מיידי על התחברות /מנותק.

שליחת תוצאות מבחן, צילומי מסך, הודעות, אישורים או כל מידע נוסף לפי דרישת המורה.

**אבטחת תקשורת:**

כל התקשורת בין התלמיד לשרת עוברת הצפנה מבוססת SSL, במטרה להגן על פרטיות התלמידים ועל שלמות המידע.

המערכת נבנתה כך שתתמודד עם תקשורת בלתי יציבה – הקליינט יודע לנסות להתחבר מחדש לשרת במקרה של ניתוק.

**ביצועים והתאמה לריבוי תלמידים:**

הקליינט תוכנן לעבודה בסביבות מרובות משתמשים, עם תמיכה בהרבה פקודות במקביל.

השימוש בתהליכונים מאפשר לו לבצע פעולות כמו צילום מסך, קבלת פקודות ושליחת נתונים, במקביל, מבלי להפריע לחוויית המשתמש או לעכב את התגובה.

**סיכום:**

התלמיד ((Client מהווה את הרכיב המרכזי שמממש את הקשר בין עמדת התלמיד למורה. הוא שקט, קל במשקל, בטוח לשימוש ואינו דורש התערבות מצד התלמיד. שילוב של אבטחה, ביצועים והתאמה לסביבת למידה הופכים אותו לאידיאלי לניהול כיתה מודרנית וממוחשבת.

1. אלגוריתמים עיקריים

המערכת מבוססת על מספר מנגנונים מתקדמים המאפשרים פעולה רציפה, מאובטחת ויעילה מול מחשבים מרובים. תיאור מורחב של האלגוריתמים המרכזיים:

**1. הצפנת תקשורת (SSL):**

בפרויקט נעשה שימוש בפרוטוקול SSL (Secure Sockets Layer) כדי להצפין את כל התקשורת בין השרת לקליינטים. היישום עושה שימוש בתעודות דיגיטליות וספריית ssl של פייתון כדי להבטיח שמידע כגון מבחנים, צילומי מסך ופקודות נשלח בצורה מוצפנת שאינה נגישה לגורם חיצוני. ההצפנה מגינה על פרטיות התלמידים ומונעת התקפות כגון Man-in-the-Middle.

**שלבים:**

1. יצירת קונטקסט הצפנה:



* יצירת קונטקסט SSL בשרת שמבוסס על פרוטוקול TLS.
* הקונטקסט אחראי על הגדרת סוג ההצפנה ומנגנון האימות.

2. טעינת תעודת האבטחה והמפתח הפרטי:



* התעודה (server.pem) משמשת לאימות זהות השרת מול הלקוח (Admin).
* המפתח הפרטי (server.key) משמש לפענוח הנתונים שמוצפנים בצד הלקוח.
* הקבצים נוצרים מראש ע״י כלי כמו OpenSSL.

3. האזנה לחיבור מממשק הניהול :(Admin)



4. עטיפת חיבור ה-Admin:



* עטיפת החיבור לאחר ה־accept() של ממשק המורה.
* התקשורת עם ה-Admin מוצפנת מרגע זה.

5. עטיפת חיבורי התלמידים (Clients):



* כל תלמיד שמתחבר – החיבור שלו נעטף מיד עם SSL.
* כך נשמרת ההצפנה גם מהצד של הקליינטים.
* גם קליטת השם וגם שליחת תגובת "OK" נעשות על חיבור מוצפן.

**תוצאה:**

* אין אפשרות לתוקף לראות את שמות התלמידים, פקודות של המורה או צילומי מסך.
* גם אם התוקף יתחבר עם Wireshark לרשת – הוא יראה רק מידע מוצפן ולא קריא.
* בכך המערכת עומדת בסטנדרטים של אבטחת מידע.

**2. תקשורת מקבילית – Threads ו־Select:**

כדי לאפשר לשרת לנהל במקביל עשרות קליינטים ללא עיכוב, ולמורה ((Admin לקבל הודעות במקביל אשר מועברות דרך השרת, נעשה שימוש בתהליכונים (threading) לצד המודול select. בשלב האתחול, השרת מפעיל תהליכונים (Threads) נפרדים לקליטת חיבורים שונים – כגון חיבורי לקוחות, חיבורי admin לטובת שליחת פקודות, והודעות רקע.

לאחר מכן, בלולאה הראשית של השרת, מופעל select.select() שמזהה אילו sockets (clients או admin) שלחו מידע – כך שהשרת יכול להגיב מיידית, מבלי להמתין לחיבורים שאינם פעילים.

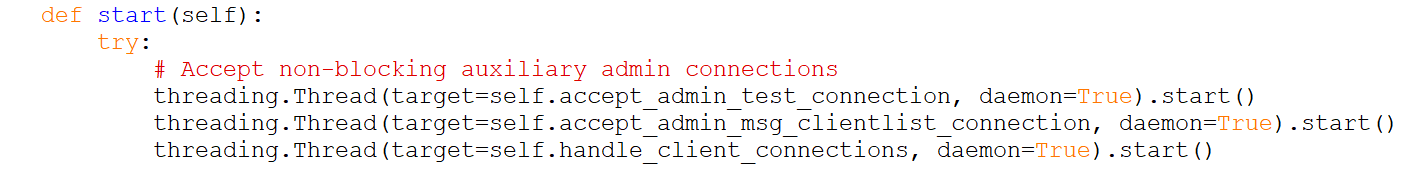
כל פעולה כמו קבלת פקודה, שליחת הודעה, או טיפול בתלמיד חדש – מתבצעת בתהליכון עצמאי, כדי לשמור על זמני תגובה מהירים ויכולת עיבוד מקבילי.

מודל זה מאפשר, לדוגמה, לקבל תגובות מלקוחות בזמן אמת – גם כאשר לקוחות רבים מחוברים בו זמנית.

**ניהול מקביליות בשרת – threading ו־select**:

**שימוש ב־threading להרצת פונקציות מקבילות**

כל חיבור שמתקבל מנוהל בתהליכון (Thread) נפרד:



כל Thread מאזין לסוג שונה של חיבורים (admin, מבחן, לקוחות):

* **accept\_admin\_test** – שליחת ציוני מבחנים מהתלמידים (Clients) למורה (Admin).
* **accept\_admin\_msg\_clientlist** – שליחת הודעות מהתלמידים (Clients) למורה

(Admin) וגם אם מתחבר\מתנתק תלמיד- השרת מודיע למורה לעשות refresh

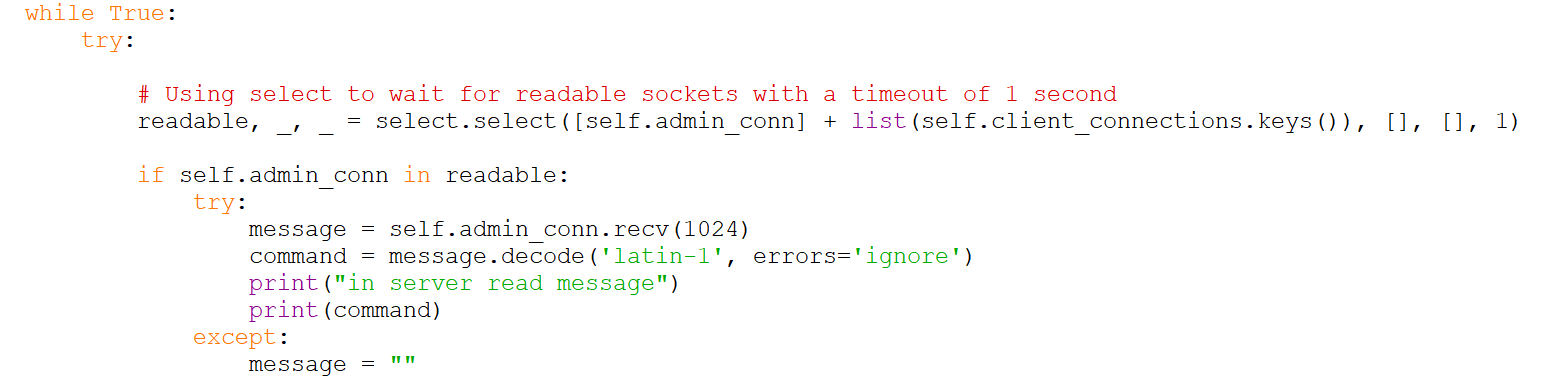
ל clientlist.

* **handle\_client\_connections**- מפעיל את הפעולה אשר מקבלת ומאשרת חיבורים מתלמידים.

זה מאפשר לשרת לטפל בכל חיבור בצורה נפרדת – בלי לחסום אחרים.

**שימוש ב־select.select() בלולאת השרת:**

מאפשר לשרת לבדוק אילו חיבורים שלחו מידע – מבלי לחסום את כל הפעילות.

****

* נמצא בתוך פעולת **main\_server\_loop** הפעולה המרכזית של קבלת הודעות וטיפולן בשרת.
* ברגע שמגיע מידע – השרת מגיב מיידית ומטפל בו.
* כל שאר החיבורים ממשיכים לרוץ במקביל, בלי להיתקע.

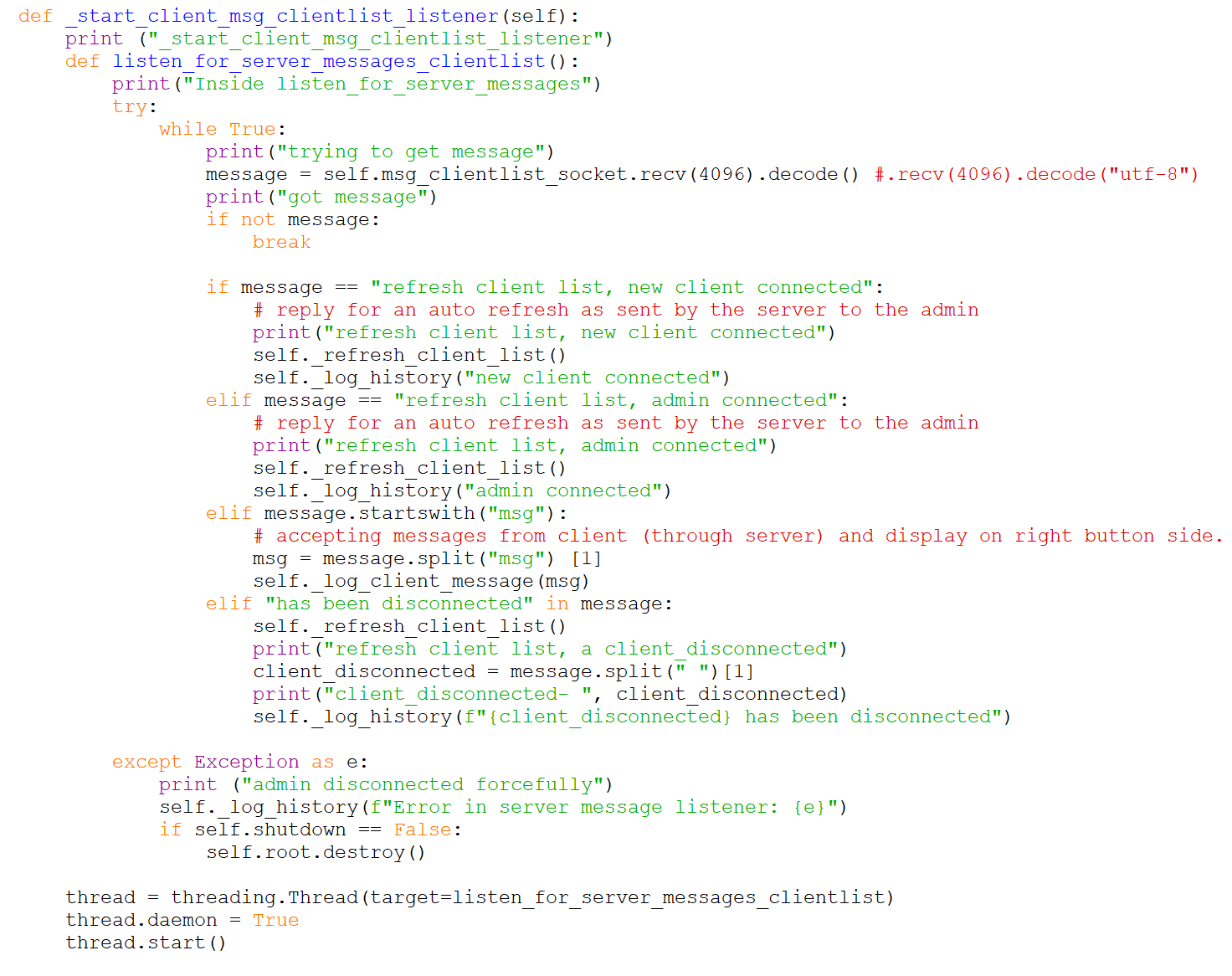
**תהליכונים בצד ה־Admin:**

**שימוש ב־threading להאזנה תמידית להודעות מהשרת**

**ה** **threadingבא לידי ביטוי בכמה מקומות בקוד המורה ((Admin**

1. הפונקציה **listen\_for\_server\_messages\_clientlist** מאזינה כל הזמן להודעות מהשרת דרך הSocket **msg\_clientlist\_socket**.

תהליכון נפרד (Thread) מריץ את הפונקציה הזו ברקע:



* ההאזנה מתבצעת ברקע – כלומר, לא חוסמת את הממשק הראשי.
* אפשר להמשיך ללחוץ כפתורים, לשלוח פקודות, ולהציג מידע – בזמן שהאדמין ממשיך לקבל עדכונים מהשרת.
* אם ההאזנה הייתה נעשית בתוך ה־main thread (ללא threading), הממשק היה קופא ומפסיק להגיב.

2. בסוף הפונקציה **\_send\_command** אשר אחראית על שליחת הפעולות הנלחצות בכפתורים אל השרת, ישנה שורה הפותחת תהליכון ((Thread אשר אחראי על ניהול התשובות\תגובות אשר מגיעות מהשרת באמצעות פעולה בשם **handle\_server\_reply**:



בנוסף אם התגובה היא אישור שאחד התלמידים התחיל לענות על מבחן:



נפתח Thread חדש אשר קורא לפעולה - **handle\_test\_answer:**



זאת מכיוון שלמבחנים אשר עונים עליהם התלמידים יש הגבלת זמן של דקה (60 שניות) אשר בזמן זה, במידה והמורה יבצע פעולות אחרות המערכת "תקרוס" מכיוון שהיא תחכה על קבלת תשובה מהמורה של ציון המבחן של התלמיד. לכן יש לפתוח Thread חדש אשר מחכה לתגובה ספציפית של ציון התלמיד.

**3. צילום מסך – בקשה, קבלה, שמירה והצגה:**

כאשר המורה (Admin) שולח פקודת צילום מסך, השרת מעביר את הפקודה " SCREENSHOT" לתלמיד הרלוונטי. התלמיד מצלם את המסך באמצעות ספריית PyAutoGUI, שומר את התמונה מקומית כקובץ PNG, ולאחר מכן שולח את התמונה לשרת במקטעים (chunks) בגודל 1024 בייט, כשההעברה תחומה בין הסימנים PIC\_START ו־PIC\_END.

השרת קולט את המידע הבינארי, מאחד אותו לקובץ אחד, ושומר אותו בתיקייה מקומית עם שם ייחודי לפי התאריך, הזמן המדויק ושם התלמיד. לאחר השמירה, השרת שולח הודעה לאדמין על הצלחת הפעולה.

הצגת צילום מסך התלמיד על מסך המורה מתבצעת באמצעות לחיצה על כפתור

\_request\_last\_file ולאחר מכן בחירת תלמיד. מתבצעת כניסה על הפונקציה \_request\_last\_file ובמידה ויש צילום מסך מהתלמיד – צילום המסך העדכני ביותר של אותו תלמיד מוצג על מסך המורה.

**4. ניהול ציונים ושליפתם (Database Management with SQLAlchemy)**

מערכת ניהול הציונים שומרת את הערכות התלמידים במסד נתונים מסוג SQLite תוך שימוש ב־SQLAlchemy לניהול נוח ובטוח של המידע. המורה ((Admin יכול להזין ציונים חדשים עבור תלמידים לפי ציון שם, והמערכת מתעדת את הציון עם חותמת שם. בנוסף, ניתן לשלוף את כל הציונים דרך ממשק הניהול, להציגם בצורה גרפית ברורה במסך המורה, ולבצע מעקב רציף אחר ביצועי התלמידים לאורך זמן.

**הזנת ציונים:**

כאשר האדמין שולח פקודה בפורמט GRADE <grade>:<client\_name\_or\_ip>> באמצעות הכפתור SetGrade"", השרת מבצע את השלבים הבאים:

* מזהה את התלמיד לפי השם או כתובת ה־IP (באמצעות מיפוי שנשמר בזיכרון).
* יוצר אובייקט חדש מסוג Student עם השם והציון.
* מוסיף את האובייקט למסד הנתונים (students.db) בעזרת SQLAlchemy ומבצע commit.
* נשלחת הודעה אל התלמיד שהושם לו ציון ואת הציון.

בנוסף, כאשר התלמיד מסיים מבחן – הציון נשלח לשרת ששולח למורה, והמורה שולח אוטומטית לשרת את הציון ומכניסו למסד הנתונים (students.db).

**שליפת ציונים:**

כאשר האדמין שולח את הפקודה GETGRADES, השרת:

* מפעיל שאילתת ORM שמחזירה את כל רשומות הציונים מטבלת students.
* יוצר מחרוזת עם כל הנתונים בפורמט ידידותי:



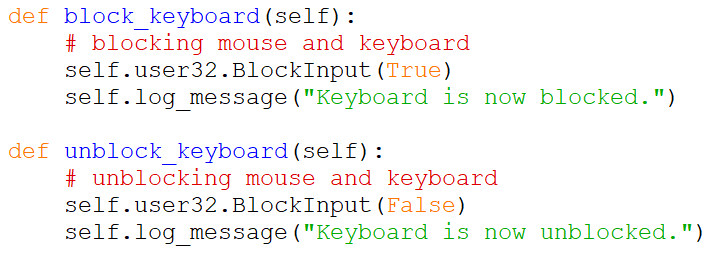
* שולח את המחרוזת ישירות לאדמין דרך הסוקט.

**תצוגה גרפית בצד האדמין:**

* המחרוזת שמתקבלת מוצגת בתוך canvas בממשק הגרפי של האדמין (Tkinter).
* גובה התצוגה מותאם אוטומטית לפי מספר השורות, עם תמיכה בגלילה — כדי לשמור על נראות נוחה וקריאה.

**5. נעילת והפעלת מקלדת ועכבר התלמיד\ים – Block/Unblock:**

הקליינט כולל מנגנון שמבוסס על ctypes שמאפשר לחסום את הקלט (מקלדת ועכבר) לפי הוראת השרת. פעולה זו מתבצעת בצורה מיידית ואינה דורשת את הסכמת המשתמש, ומתאימה במיוחד למצבים שבהם נדרש להחזיר את תשומת הלב של התלמיד. הפקודה מגיעה מהשרת, התלמיד ((Client מזהה את ההוראה ומבצע את החסימה (או השחרור) בתוך תהליך עצמאי, כך:



**6. ניהול משתמשים באמצעות שני מילונים (Dicts):**

השרת שומר על רשימת חיבורים עדכנית של כל הקליינטים בזמן אמת, כולל כתובת IP, שם תלמיד וחיבור פעיל. ניהול זה מתבצע באמצעות שני מילונים עיקריים, המאפשרים שליטה, בקרה ועדכון רציפים של מצב החיבור.

**מבני הנתונים:**

המערכת משתמשת בשני מילונים לניהול חכם של הלקוחות:

**client\_name\_ip\_mapping** .1 – קישור בין שם תלמיד לחיבור (socket):

* מאפשר שליחה ישירה לפקודות לפי שם.
* מונע כפילויות: אם אותו שם מנסה להתחבר פעמיים – נחסם.

**client\_ip\_name\_mapping** .2 – קישור בין IP לרשימת שמות תלמידים:

* תומך במספר תלמידים על אותו מחשב/רשת (IP אחד).

יחד, שני המבנים האלה שומרים את מצב החיבור וההרשאות של כל קליינט, ומאפשרים:

* בדיקת כפילויות.
* חיפושים לפי שם או IP.
* הסרה ועדכון קליינטים בצורה נקייה ומהירה.
* הרחבה עתידית (סטטוס, תעודת זהות, רמת הרשאה וכו').

**בעת התחברות קליינט:**

השרת מאזין ומקבל את החיבור החדש (accept()), תוך עטיפה ב־SSL, והתלמיד שולח את שמו עם ההתחברות.

השרת:

* מוודא שהשם לא בשימוש כבר, אם כן – שולח לתלמיד הודעה מתאימה ולא מכניס אותו למאגרי הנתונים.
* שולח אישור או הודעת שגיאה.
* שומר את הנתונים בכל אחד מהמילונים.
* שולח למורה הודעת "refresh client list" לעדכון ה־GUI.

**בעת ניתוק קליינט:**

התלמיד שולח פקודת shutting down> - <name>>.

השרת:

* מאתר את שם הקליינט.
* מסיר אותו מכל המילונים (client\_name\_ip\_mapping, client\_connections, client\_ip\_name\_mapping, client\_ips).
* סוגר את החיבור עם התלמיד.
* מודיע למורה: "Client <name> has been disconnected" מה שמודיע לו לעשות "refresh client list" לעדכון ה־GUI.

**7. העברת קבצים גדולים – שימוש במפרידים:**

המערכת תומכת בשליחת קבצים (טקסטים, מבחנים, תמונות) בין המורה לתלמידים. לשם כך נעשה שימוש בפרוטוקול תקשורת מותאם המבוסס על מפרידים ברורים בשליחת ההודעה מהמורה.

**מבנה ההודעה:**

<פקודת השליחה (SENDFILE או SENDFILE1)>

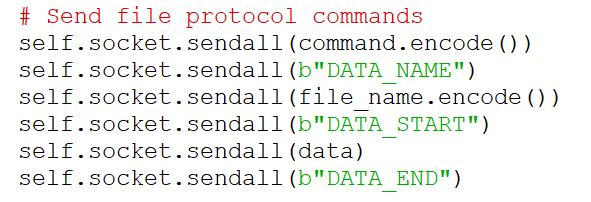
DATA\_NAME

<שם הקובץ>

DATA\_START

<תוכן הקובץ הבינארי>

DATA\_END



**אופן הפעולה:**

**בצד המורה:**

* נקרא הקובץ כ־binary.
* נשלחים בזה אחר זה: הפקודה, שם הקובץ, התוכן, וסימני התחלה/סיום.

**בצד השרת:**

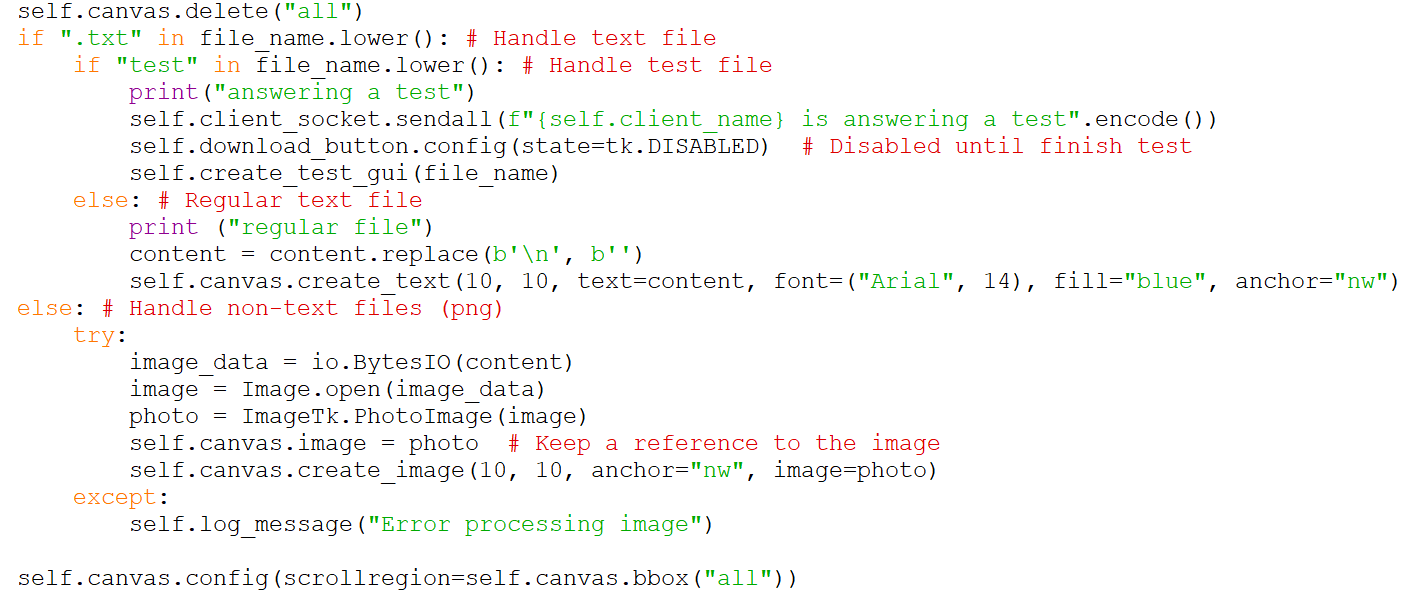
* השרת לא מפרש את הקובץ אלא רק מתווך את המידע לקליינט המתאים.

**בצד התלמיד:**

* המידע נאסף עד DATA\_END.
* שם הקובץ מופרד באמצעות DATA\_NAME.

הקובץ מפוענח ומוצג:

* .txt — נפתח כקובץ טקסט רגיל או כמבחן אינטראקטיבי.
* .png — מוצג כקובץ תמונה בתוך ה־GUI.



1. דרישות ומגבלות מערכת

**דרישות:**

1. יש להתקין Python בגירסה 3.10 ומעלה.

2. יש להתקין את הספריות הבאות:

* socket – לניהול תקשורת TCP בין שרת, אדמין וקליינטים.
* ssl – להצפנת החיבורים.
* tkinter – לממשק המשתמש הגרפי.
* Pillow – להצגת קבצי תמונה מסוג PNG.
* ctypes – לחסימת מקלדת (במערכת Windows בלבד).
* threading – לניהול חיבורים מרובים במקביל.
* hashlib – לחישוב checksum של קבצים.
* sqlalchemy – לעבודה עם מסד נתונים של תלמידים.

3. יש להפעיל את המערכת על רשת מקומית (LAN).

4. קבצים מועברים באמצעות פרוטוקול מותאם הכולל מפרידים ברורים לתחילת וסיום קובץ.

5. כל התקשורת מוצפנת (SSL encrypted communication) מקצה לקצה בין רכיבי המערכת.

**מגבלות:**

1. המערכת תומכת אך ורק בקבצים מסוג txt ו־png.

2. פועלת רק במערכות הפעלה Windows.

3. אינה תומכת בהפעלת שני תלמידים בעלי אותו שם.

4. פעולת צילום מסך זמינה רק לתלמיד בודד בכל פעם – אין תמיכה בפעולה קבוצתית.

5. מומלץ לא לעבור את כמות של כ־20 קליינטים בו זמנית – מעבר לכך עשויה להתרחש האטה בתגובה.

6. קבצים מסוג SENDFILE1 חייבים להישמר בתיקייה הראשית של הפרויקט לצורך שליחה תקינה.

7. המערכת עובדת רק על רשת מקומית (LAN).

1. התייחסות לנושא אבטחה

המערכת מתוכננת תוך דגש על שמירה על פרטיות, שלמות המידע ומניעת גישה לא מורשית, באמצעות כמה מנגנוני אבטחה מרכזיים:

1. **הצפנת מידע בתקשורת** – כל חיבור בין רכיבי המערכת (server, admin, client) מתבצע באמצעות פרוטוקול SSL, אשר מצפין את כל הנתונים העוברים ברשת ומגן עליהם מפני יירוט (man-in-the-middle).

2. **אימות זהות המשתמשים** – המערכת שומרת מידע על כל קליינט באמצעות מזהה ייחודי (IP או שם תלמיד), ומאפשרת שליחת פקודות רק לקליינטים מחוברים ומורשים.

3. **הגבלת גישה לפקודות** – רק ממשק האדמין מורשה לשלוח פקודות לשרת, מה שמונע מכל קליינט או גורם חיצוני להשתלט על המערכת או לבצע פעולות מזיקות.

4. **שמירה מבוקרת של קבצים** – קבצים שמתקבלים מהאדמין נשמרים בזיכרון בלבד (ולא נכתבים לדיסק), אלא אם המשתמש אישר זאת בלחיצת כפתור. כך נמנע סיכון של הדבקה מקבצים חשודים.

5. **ניהול פעולות בזמן אמת** – פעולות כמו חסימת מקלדת, פתיחת מבחן ושליחת הודעה נרשמות בלוג הפנימי של המערכת, המאפשר מעקב אחר היסטוריית הפעולות שבוצעו.

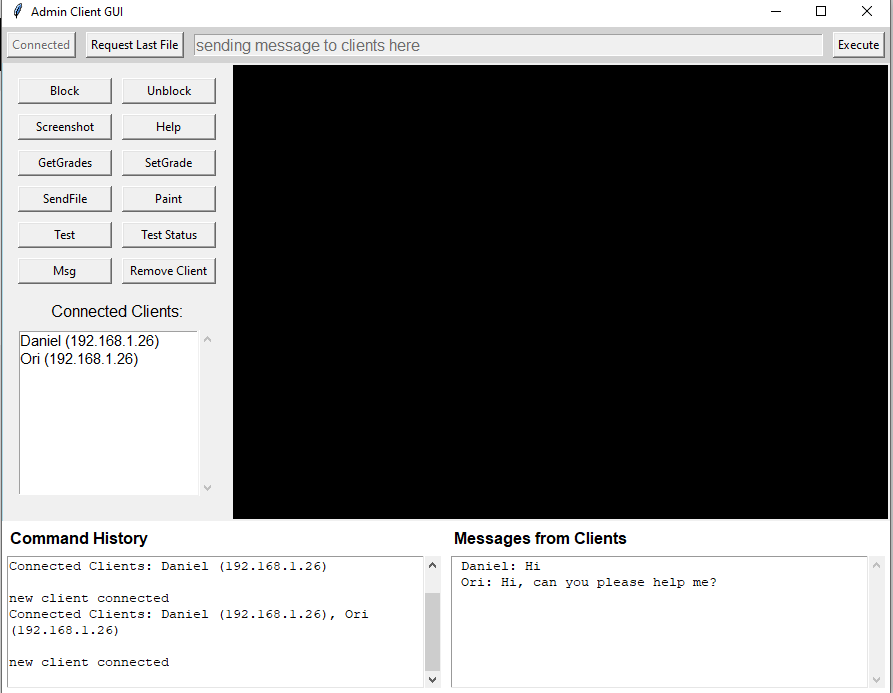
6. **הגנה מפני שליחת קבצים מזויפים** – כל קובץ מועבר עם חותמת (checksum) מסוג MD5, כדי לוודא שהקובץ שהתקבל זהה לקובץ שנשלח, ללא שיבוש או שינוי בדרך.

1. ממשק משתמש

כפתור בקשת צילום מסך אחרון מתלמיד ספציפי

**ממשק משתמש מורה – Admin:**

כפתור סגירה

****

שורת כתיבת הודעות לתלמידי\ים

כפתור שליחת ההודעות

מסך בו מוצגים צילומי מסך\ טבלת הציונים\ סטטוס מבחנים

הודעות שהתקבלו מתלמידים

היסטוריה של אילו פעולות נעשו

תיבה שמראה אילו תלמידים מחוברים

כפתורים לפעולות

שניתן לעשות

כפתור חיבור לשרת

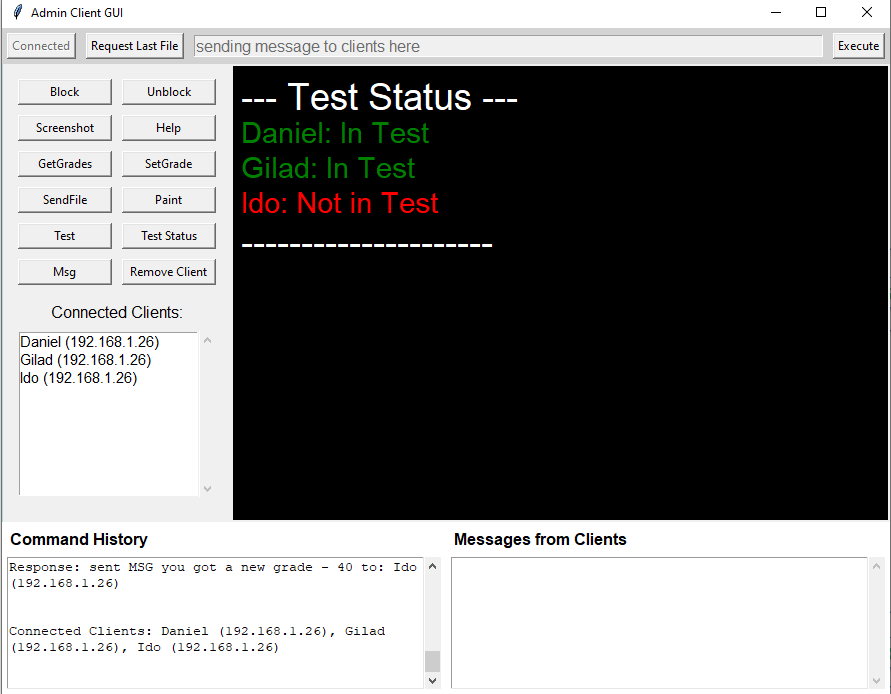
**תיאור:**

* ממשק ניהול מתקדם שמאפשר למורה לנהל את הכיתה בצורה מרחוק:
* שליחה וביצוע פקודות (כמו חסימה, פתיחת מבחן, שליחת קובץ ועוד).
* מעקב אחר רשימת תלמידים מחוברים.
* צפייה בהודעות שהתקבלו מהתלמידים.
* צפייה בהיסטוריית פקודות שבוצעו.
* מסך שחור מרכזי המשמש להצגת צילומי מסך מהתלמידים.

**מטרת החלון:**

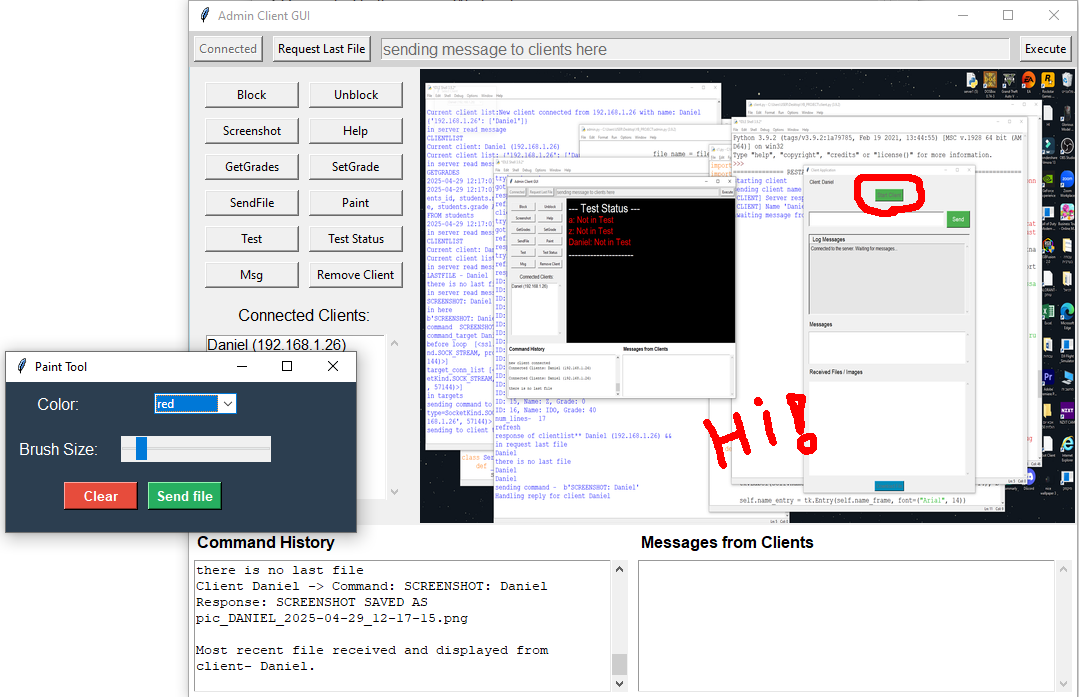
* שליטה מרחוק (remote desktop control) מלאה על מחשבי התלמידים.
* תקשורת עם תלמידים באופן אישי או קבוצתי.
* ניהול מבחנים, ציונים, תמיכה ועוד.

**דוגמאות לתרחישים נוספים בממשק המנהל:**



סטטוס מבחנים של כל אחד

מהתלמידים בזמן אמת



בחירת עובי

בחירת צבע

שליחת התמונה המעודכנת לאחר הפידבק לאותו תלמיד\תלמיד אחר\ כל תלמידי הכיתה

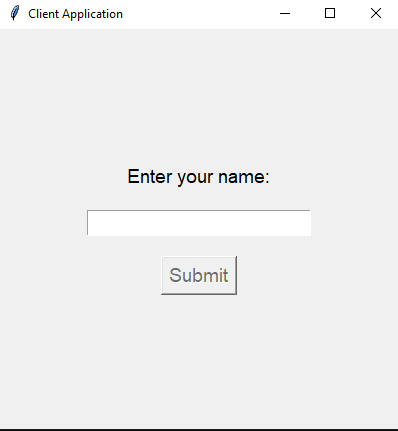
ניקוי המסך

כפתור הציור

כפתור בקשת המסך האחרון מתלמיד

דוגמה לציור

**ממשק משתמש תלמיד – Client:**

****

כפתור שליחת שם

מקום להכנסת שם התלמיד

**חלון התחברות לקוח – הזנת שם**

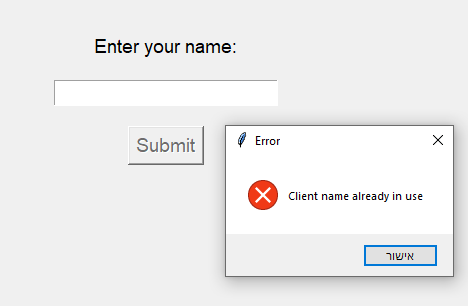
**תיאור:**

* החלון הראשון שמופיע למשתמש הקצה (תלמיד). כאן עליו להזין את שמו.

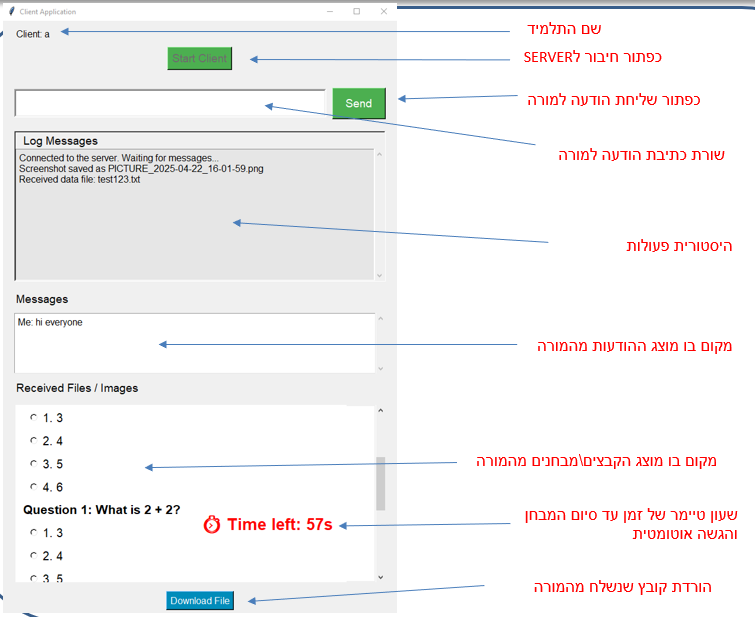
**מטרת החלון:**

* לזהות את המשתמש במערכת.
* לאפשר לשרת לשייך פעולות ומידע למשתמש הספציפי.
* ישנה גם בדיקה שנעשית האם השם תקין.

-------------------------------------------------------------------------------------------------------



**במידה והשם תפוס**



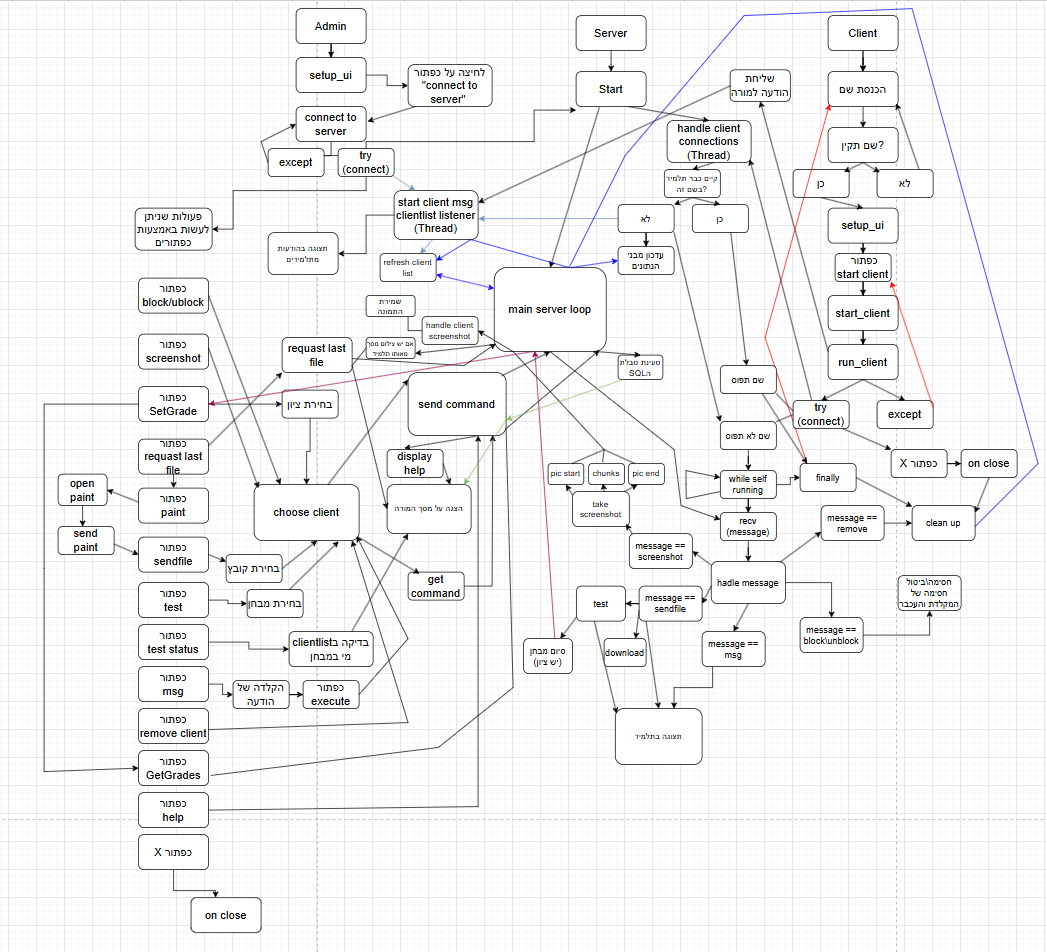
**ממשק תלמיד ((Client**

**תיאור:**

* לאחר הזדהות, התלמיד מועבר לממשק הראשי שבו הוא יכול:
* להתחבר לשרת (Start Client)
* לשלוח הודעות
* לצפות בהודעות היסטוריית המערכת (Log Messages)
* לראות הודעות מהמורה
* לראות קבצים שנשלחו מהמורה \ מבחנים שנפתחו על ידיו (כפי שניתן לראות בתמונה)
* להוריד קבצים שנשלחו אליו (כפתור Download File)

**מטרת החלון:**

* לאפשר תקשורת עם המורה.
* להציג קבצים שהתקבלו.
* לתעד התכתבויות ופעולות מהשרת.

1. תרחישים עיקריים

קישור לתרשים הזרימה - https://drive.google.com/file/d/1pans9KgtSr\_WEqJSWyDEw1fGoCiFAPVP/view?usp=sharing

1. תהליך כתיבת הפרויקט
   1. תהליך הפרויקט

במהלך העבודה על הפרויקט, עבדתי לפי שלבים ברורים – מהרעיון הראשוני ועד המימוש בפועל. שמתי דגש על מערכת שתהיה גם פרקטית וגם ייחודית. רוב הפעולות בתחילת הדרך בוצעו דרך ה־ , Command Line ורק בשלבים המאוחרים יותר עברתי לממשק גרפי עם כפתורים, לאחר שווידאתי שכל הפעולות פועלות באופן תקין.

**שלבי העבודה:**

1. **ניתוח צרכים והגדרת מטרות**  
   זיהיתי את הצורך במערכת שמאפשרת למורה לנהל את מחשבי התלמידים מרחוק.  
   מטרת הפרויקט הייתה לספק פתרון שבאמצעותו המורה יוכל לשלוט על הפעילות בכיתה – כמו חסימת מסכים, שליחת קבצים, העברת מבחנים, צילום מסך ועוד.
2. **תכנון ראשוני**  
   החלטתי לבנות מערכת מבוססת Client-Server בשפת Python.  
   בחרתי את הספריות המתאימות socket, ssl, tkinter, threading) ועוד) ותכננתי את מבנה הקוד והממשקים Admin) ו־Client).  
   בהתחלה לא חשבתי לשלב מבחנים או ציונים, אך בהמשך הבנתי שזה יכול להיות מרכיב שייחד את המערכת שלי, ולכן הוספתי את זה לקראת סוף הפיתוח.
3. **פיתוח דרך Command Line**  
   בשלב הראשון, עבדתי כמעט לחלוטין דרך שורת הפקודה. כל פעולה – שליחת פקודה, חסימה, פתיחה, שליחת קובץ או צילום מסך – בוצעה בקוד ולא דרך כפתורים.  
   זה עזר לי לבדוק שכל הפונקציונליות פועלת בצורה תקינה לפני שאני עובר לממשק משתמש.
4. **פיתוח צד השרת (Server)**   
   כתבתי את השרת כך שיוכל לנהל מספר חיבורים במקביל, באמצעות select ו־threading.  
   יצרתי מבנה נתונים שיאחסן מידע על כל לקוח, כתובת, סטטוס חיבור, ועוד.
5. **פיתוח צד הלקוח (Client)**   
   יצרתי GUI בסיסי עם Tkinter שמאפשר ללקוח) מחשב התלמיד (להתחבר לשרת, לקבל פקודות ולבצע אותן – כולל חסימת מקלדת, קבלת קבצים, צילום מסך והגשת מבחנים.
6. **הוספת תמיכה במבחנים וציונים**  
   כאמור, רעיון המבחנים והציונים הגיע לקראת הסוף. רציתי להוסיף אלמנט שבאמת ייחד את המערכת שלי מפרויקטים אחרים.  
   בניתי מערכת שמציגה מבחן לתלמיד, מאפשרת מענה, ומעבירה את הנתונים חזרה למורה, כולל מנגנון לבדיקה והצגת ציון.
7. **פיתוח ממשק Admin גרפי**  
   רק לאחר שהמערכת פעלה כמו שצריך דרך ה־Command Line, עברתי לממשק גרפי.  
   בניתי ממשק נוח למורה, שמכיל כפתורים לכל פעולה – שליחה, חסימה, פתיחת מסכים, הצגת צילום מסך, שליחת מבחנים, קבלת תוצאות ועוד.
8. **אבטחת מידע**  
   הוספתי תקשורת מוצפנת (SSL encrypted communication) על בסיס SSL, כך שכל המידע בין השרת ללקוחות עובר בצורה מאובטחת.  
   גם הגבתי לסיכונים פוטנציאליים כמו חסימה של קבצים מסוכנים – כך שניתן לשלוח רק קבצים בטוחים (.png ו־.txt).
9. **בדיקות ואינטגרציה**  
   ביצעתי בדיקות עם לקוחות מרובים, בדקתי חיבורים וניתוקים, העברות קבצים, ותפקוד תחת עומס.  
   ווידאתי שגם במקרה של תקלות – המערכת מתאוששת כראוי או מציגה הודעת שגיאה ברורה.
10. **שיפורים, עיצוב ותיעוד**  
    לאחר שכל המערכת פעלה, שיפרתי את העיצוב, עיגלתי כפתורים, הוספתי רקעים ותפריטים, הוספתי תיעוד מפורט לקוד והכנתי דוח סופי עם תרשימים, תמונות והסברים.
11. אתגרים ואופציות שונות למימוש

במהלך כתיבת הפרויקט נתקלתי במספר אתגרים טכנולוגיים ומעשיים שדרשו ממני לחפש פתרונות יצירתיים ולבחון כמה אופציות למימוש.

**1. חיבור מרובה לקוחות לשרת**

אחד האתגרים המרכזיים היה לאפשר למספר מחשבים (Clients) להתחבר לשרת בו זמנית, כך שמורה יוכל לשלוט על כל הכיתה בו־זמנית.

שקלתי להשתמש ב־threading, אך הבנתי שזה עלול להכביד על המערכת כשמדובר בעשרות חיבורים.

לבסוף החלטתי לשלב בין threading לבין select, כך שהשרת יוכל לנהל מאות חיבורים בצורה יעילה, תוך מעקב רציף אחרי כל לקוח.

**2. תקשורת בטוחה בין השרת ללקוחות**

ידעתי שמכיוון שהמערכת מעבירה מבחנים, קבצים, ואפילו צילומי מסך – חשוב שהכל יעבור בצורה מוצפנת.

לכן בחנתי מספר דרכים להצפין את התקשורת:

* להשתמש בפרוטוקול HTTPS דרך Flask
* לכתוב הצפנה ידנית בקוד
* להשתמש ב־SSL ברמת ה־socket

לבסוף בחרתי באופציה השלישית – שימוש ב־SSL של Python, שמספק הצפנה חזקה ומובנית, ויחד עם זאת מאפשר לי להישאר עם socket רגיל בלי לעבור לפרוטוקולים מורכבים יותר.

**3. חסימת מקלדת ותצוגה על מחשבי התלמידים**

רציתי לאפשר למורה "לנעול" את מחשב התלמיד – כלומר לחסום מקלדת ולהציג תמונה (למשל מבחן או הודעה).

במקום לבנות מערכת כבדה שמתלבשת על מערכת ההפעלה, בחרתי בגישה פשוטה יחסית בעזרת ספריות כמו ctypes (לחסימת מקשים) ו־Pillow (להצגת תמונה בממשק Tkinter על כל המסך).

**4. ממשק משתמש לעומת פיתוח דרך שורת הפקודה**

לאורך רוב הפיתוח עבדתי דרך ה־Command Line. היה לי חשוב קודם כל לוודא שכל הפעולות מתבצעות כמו שצריך – חסימות, שליחות, צילום מסך, מבחנים.

רק לקראת הסוף בניתי ממשק גרפי (GUI) נוח וברור – שמכיל כפתורים, תצוגות, רשימות תלמידים, ציונים וכו', למדתי להשתמש בgrid מה שעזר לי מאוד בסידור הכפתורים.

**5. ניהול מבחנים ובדיקת ציונים**

בחלקים הראשונים של הפרויקט בכלל לא תכננתי לכלול מבחנים. רק בהמשך עלה לי הרעיון להוסיף תכונה ייחודית שתבדיל את המערכת שלי – לתת למורה אפשרות לשלוח מבחן, לקבל תשובות, ולתת ציון.  
הוספתי תמיכה בשאלות אמריקאיות, שליחת מבחן, אפשרות לתלמידים לסמן תשובות, ותצוגה למורה עם אפשרות לבדוק ולתת ציון (כמובן שהבדיקה ומתן הציון הם אוטומטיים).   
מכיוון שלא רציתי ששליחת התשובות תפריע לחיבור הקיים בין השרת לתלמיד, ורציתי שהציון יתקבל בצורה אוטומטית עם סיום המבחן של התלמיד ושליחת התשובות, בניתי  **socket חדש** ייעודי רק להעברת תוצאות המבחן, שנפתח באופן זמני עבור כל תלמיד, וכך נשמרה היציבות של המערכת.

**6. ריענון אוטומטי לרשימת התלמידים (Client List)**

רציתי שרשימת התלמידים המחוברים תתעדכן אוטומטית אצל המורה בכל שינוי – חיבור/ניתוק.

בהתחלה האופציה לרענן את הרשימה הייתה רק כשהמורה לוחץ על כפתור ריענון, אבל בסוף יישמתי ריענון אוטומטי מחזורי – כאשר מתחבר/ מתנתק תלמיד או שהמורה מתחבר, השרת מקבל הודעה אותה הוא מעביר אל המורה, אשר מבקש את רשימת התלמידים המחוברים , והרשימה מתעדכנת בהתאם בממשק המורה.

**7. שליחת הודעות מתלמידים למורה**

עוד אתגר שעלה לקראת הסוף היה לאפשר לתלמידים לשלוח הודעה או שאלה למורה מתוך הממשק שלהם, כדי שגם הם יהיו אקטיביים.

מימשתי מערכת הודעות מבוססת פרוטוקול פשוט: כל הודעה שמגיעה עם תגית MSG מהתלמיד מתווכת ל־Admin, שמקבל אותה עם שם התלמיד, ותוכן ההודעה, ומיד מוצגת בממשק המורה.

1. מרכיבי פתרון
   1. תיחום הפרויקט

* **תקשורת –** המערכת שלי מבוססת על תקשורת בין שרת אחד לבין שני סוגי לקוחות – תלמידים ומורה (Admin).

השתמשתי בפרוטוקול TCP באמצעות מודול socket, ובנוסף עטפתי את החיבורים ב־SSL לצורך אבטחת מידע.

בחרתי גם לשלב את מודול select כדי לתמוך במספר חיבורים בו־זמנית ולנהל את כולם בצורה יעילה וחסכונית.

* **מערכות הפעלה –** הפרויקט נבנה בסביבת Windows, וכל הלקוחות (תלמידים ומורים) צפויים להפעיל את המערכת ממערכת זו.

בחרתי ב־Windows בעיקר כי רציתי להשתמש ביכולות מערכת כמו חסימת מקלדת (ctypes) והצגת תמונות במסך מלא, שדורשות גישה לרכיבי מערכת ברמה נמוכה.

* **תצוגה –**את הממשק הגרפי פיתחתי באמצעות ספריית Tkinter שמגיעה מובנית עם פייתון.

השתמשתי בכפתורים, טבלאות, תיבות טקסט וצילומי מסך כדי לבנות ממשק ידידותי ופשוט גם למורה וגם לתלמיד.

בהתחלה עבדתי בעיקר עם שורת פקודה (Command Line) כדי לבדוק שכל פעולה עובדת כמו שצריך, ורק לקראת הסוף בניתי את הממשק הגרפי המלא.

* **אבטחה –**בגלל שהמערכת מעבירה מבחנים, הודעות אישיות, צילומי מסך ואפשרות לשליטה – היה לי חשוב להבטיח שכל המידע יעבור בצורה מוצפנת.

השתמשתי ב־SSL כדי להצפין את כל התקשורת בין השרת ללקוחות, ומימשתי גם מנגנון לאימות זהות השרת באמצעות תעודה (certificate).

בנוסף, כל קובץ או מבחן שנשלח עובר בדיקה ונשמר במבנה ייעודי כדי למנוע קריסות או מתקפות.

* **מבנה נתונים –**השתמשתי במבני נתונים כמו מילונים (dict) כדי לנהל את רשימת התלמידים, הסטטוס שלהם, כתובות IP, נתוני מבחנים, ותוצאות ציונים.

בכל רגע נתון, המורה יכול לראות את כל התלמידים המחוברים והסטטוס של כל אחד – הכל מנוהל בזמן אמת בזיכרון על ידי השרת.

* **שימוש במסד נתונים -** בהמשך הפרויקט הוספתי גם מסד נתונים מסוג SQLite כדי לשמור את הציונים והתוצאות של כל מבחן לכל תלמיד.

בחרתי ב־SQLAlchemy כממשק ORM מעל SQLite – כך שכל התקשורת מול מסד הנתונים הייתה ברורה, נוחה וניתנת להרחבה.

זה גם מאפשר לשמור ולגשת למידע היסטורי על תלמידים – כמו מבחנים קודמים ותוצאות.

* **ארכיטקטורת קוד –** הפרויקט מחולק לשלושה חלקים עיקריים:

שרת (Server) – מנהל את כל התקשורת, מתווך בין הלקוחות, ומבצע פעולות.

ממשק מורה (Admin GUI) – מאפשר שליטה על הלקוחות, תצוגת תלמידים, שליחת מבחנים, קבלת ציונים ועוד.

ממשק תלמיד (Client GUI) – מציג לתלמיד מבחנים, הודעות, קבצים, וכן מאפשר לו לשלוח הודעות חזרה למורה.

כל חלק פועל כקובץ Python עצמאי, עם פונקציות מחולקות לפי אחריות – קלט/פלט, תקשורת, UI, ואבטחה.

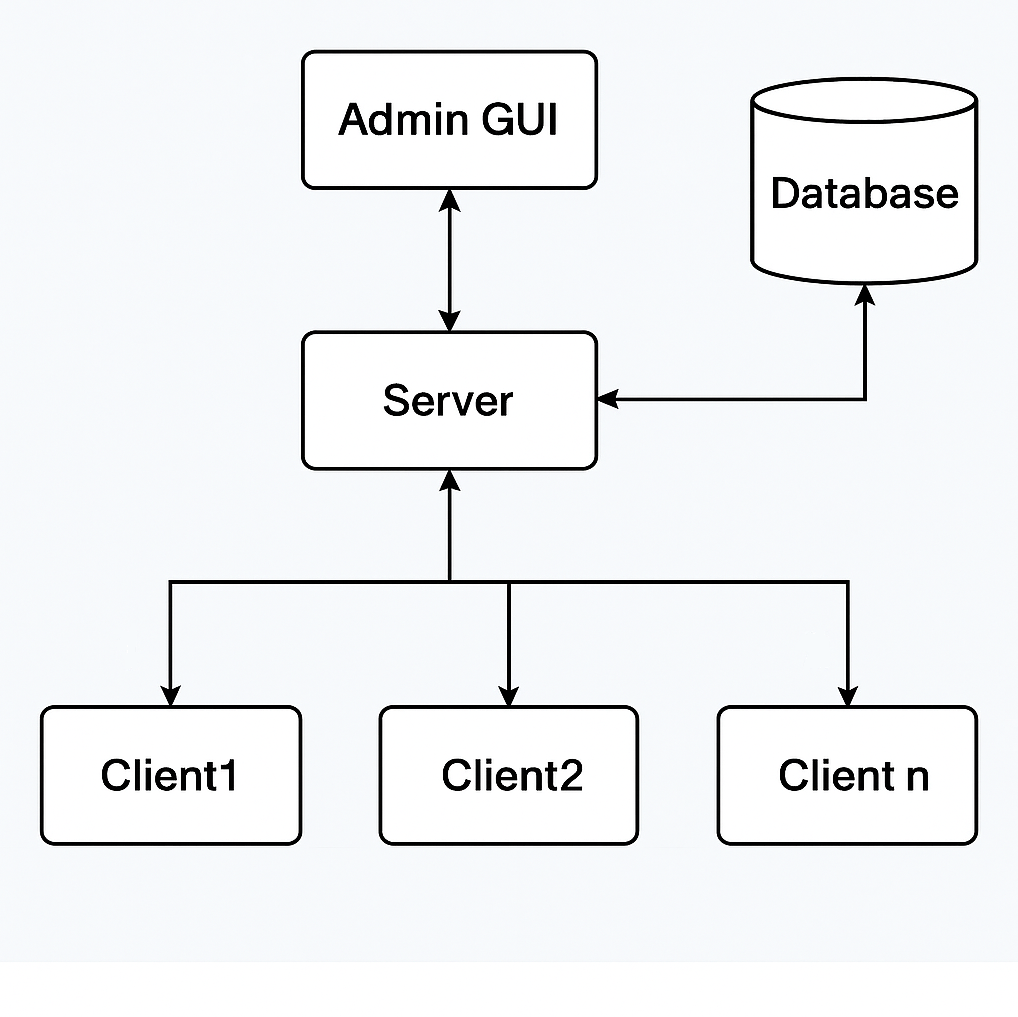
* **תיעוד –**במהלך הפיתוח הקפדתי לתעד את כל השלבים, כולל תיעוד פנימי בקוד עם הערות, מחלקות, פונקציות ופרוטוקולים.  
  בנוסף, תיעדתי את מבנה הפרויקט, תרשימי זרימה לתרחישים העיקריים, הסבר על כל פעולה ושורה חשובה וכל רכיב בתוכנה – כדי שכל אדם אחר שירצה להשתמש בפרויקט, יוכל להבין ולתחזק אותו בקלות.

1. סביבת העבודה (טכנולוגיה)

* שפות התכנות: Python- גם עבור התקשורת וגם עבור הממשק הגרפי.
* סביבת פיתוח:**-**IDLE סביבת הפיתוח המובנית של פייתון.

1. מבט טופולוגי

המערכת בנויה משרת מרכזי (Server), ממשק ניהול (Admin) ומספר לקוחות (Clients).



ה-Clients מתחברים לשרת המרכזי ומתקשרים מולו באופן קבוע.

כל תלמיד מקבל מהשרת פקודות, קבצים, מבחנים והודעות, ושולח בחזרה תשובות, תמונות מסך והודעות יזומות.

ה-Admin שולט במערכת דרך ממשק גרפי – שולח פקודות ללקוחות, צופה בתמונות מסך ומנהל ציונים דרך גישה למסד הנתונים המשותף עם השרת.

לצורך תפעול מבחנים נפתח socket ייעודי, וה-Client List מתעדכן אוטומטית בעזרת ריפרש אוטומטי כאשר יש שינוי בסטטוס של אחד מהתלמידים (על אותו Socketעם הודעות מהתלמידים).

1. מבנה נתונים

**Server:**

**client\_connections .1**

**סוג:** מילון (dictionary)

**מבנה:** המפתח הוא socket (האובייקט של חיבור הלקוח), והערך הוא מילון עם פרטים על הלקוח - כתובת IP ושם.

**תפקיד:** לשמור את כל החיבורים הפעילים ללקוחות בצורה שתאפשר לך למצוא פרטים של הלקוח (IP ושם) מתוך ה-socket שלו.

**client\_ips .2**

**סוג:** רשימה (list)

**מבנה:** רשימת כתובות IP (מחרוזות) של הלקוחות.

**תפקיד:** לאגור את כל כתובות ה־IP של הלקוחות שהתחברו לשרת, כולל כפילויות אם כמה משתמשים הגיעו מאותו IP.

**3. client\_ip\_name\_mapping**

**סוג:** מילון (dictionary)

**מבנה:** המפתח = כתובת IP (str), הערך = רשימת שמות לקוחות המחוברים מאותה הכתובת.

**תפקיד:** מאפשר חיפוש שמות לפי כתובת IP – לדוגמה, לראות אילו לקוחות מחוברים מאותו מחשב.

**4. client\_name\_ip\_mapping**

**סוג:** מילון (dictionary)

**מבנה**: המפתח = שם הלקוח ובסוגריים ה IP(str), הערך = אובייקט socket שלו.

**תפקיד:** מיפוי מהיר משם הלקוח ל־socket כדי לשלוח אליו הודעות או לפקח עליו.

**5. client\_files**

**סוג:** מילון (dict)

מבנה: המפתח = שם לקוח (באותיות גדולות), הערך = רשימה של זוגות (זמן, שם\_קובץ).

**תפקיד:** מאפשר לשמור את תמונות המסך או קבצים האחרונים שנשמרו עבור כל לקוח, על מנת לשלוח למורה לפי בקשה ולמצוא את התמונה העדכנית ביותר של אותו תלמיד.

**Admin:**

**1. connected\_clients**

**סוג:** רשימה (list)

**תוכן:** שמות הלקוחות המחוברים

**תפקיד**: משמש להצגת לקוחות פעילים לבחירה ושליחה של פקודות / קבצים, ולהציגם בclientlist.

**2. test\_status**

**סוג:** מילון (dict)

**מבנה:**מפתח = שם לקוח (str)**,** ערך = True/False (אם הלקוח כרגע במבחן)**.**

**תפקיד:** מונע שליחה של קבצים לתלמידים שנמצאים בעיצומו של מבחן, עוזר להצגת test status.

**3. self.selected\_file\_info**

**סוג:** מילון (dict)

**מבנה:** מפתח = פקודת "SENDFILE {file\_name}:{client\_name}", ערך = זוג (file\_path, file\_name)

**תפקיד:** שמירת נתיבי קבצים שנבחרו לשליחה, כדי שיהיה ניתן לשלוח אותם כשיגיע רגע הביצוע.

**Client:**

**1. questions (ב-load\_questions\_from\_content)**

**סוג:**

רשימה (list) של מילונים (dict).

**מבנה כל איבר ברשימה:**

כל איבר הוא מילון שמייצג שאלה אחת עם המפתחות:

* "question": מחרוזת עם טקסט השאלה (לדוגמה: "Question 1: What is 2+2?").
* "options": רשימה של מחרוזות, כל אחת היא אפשרות תשובה לשאלה.
* "answer": מספר שלם (int) שמייצג את התשובה הנכונה (לדוגמה: 2).

**תפקיד:**

הפונקציה load\_questions\_from\_content מקבלת מחרוזת (תוכן של מבחן) ומפרקת אותה למבנה הנתונים questions.

מבנה זה מאפשר לייצג את כל השאלות, האפשרויות והתשובות בצורה נוחה לעיבוד.

**שימוש:**

הפונקציה מחזירה את הרשימה questions, והיא משמשת אחר כך ב-create\_test\_gui להצגת השאלון בממשק המשתמש.

**2. selected\_answers (ב-create\_test\_gui)**

**סוג:**

רשימה (list) של משתני Tkinter מסוג tk.IntVar().

**תפקיד:**

כל IntVar ברשימה מייצג את התשובה שנבחרה עבור שאלה מסוימת במבחן.

המשתנה שומר ערך מספרי (0 אומר שלא נבחרה תשובה, ו-1,2,... מציין את האופציה שנבחרה).

**שימוש:**

ב-create\_test\_gui נוצרת רשימת selected\_answers, ומתווסף אליה IntVar עבור כל שאלה.

המשתנים מחוברים לכפתורי רדיו (RadioButtons) בממשק, והם מאפשרים מעקב דינמי אחרי בחירת התשובות.

בנוסף, הם משמשים לבדיקה האם כל השאלות נענו כדי להפעיל את כפתור ההגשה.

1. מסד נתונים

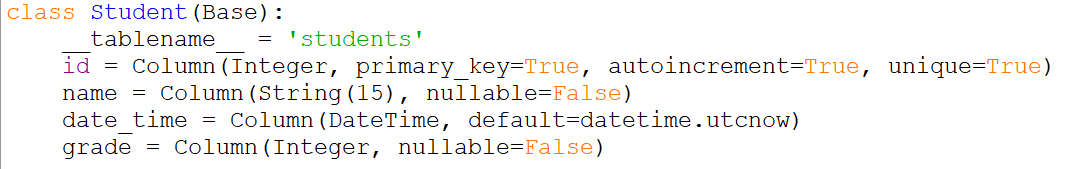
במערכת ניהול הכיתה, מסד הנתונים משמש לאחסון מידע חיוני אודות ציוני התלמידים. השימוש במסד הנתונים מתבצע בצד השרת, והוא כולל שמירת ציונים לאחר מבחנים או בעת עדכון ידני של המורה, וכן שליפת כלל הציונים לצפייה מיידית.

**טכנולוגיה**

השרת משתמש במסד נתונים מסוג SQLite, שמוגדר בקובץ מקומי בשם students.db. לצורך ניהול נוח ובטוח יותר של מסד הנתונים, נעשה שימוש ב־SQLAlchemy — ספריית ORM בפייתון המאפשרת להגדיר טבלאות ומודלים של נתונים באופן מונחה עצמים, במקום כתיבת שאילתות SQL גולמיות.

**הגדרת הטבלה**

השרת כולל טבלה אחת בשם students, המוגדרת כך:

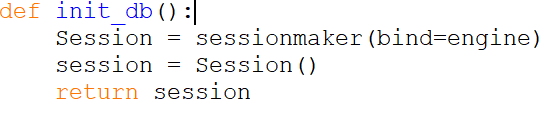


שדות הטבלה:

* **id:** מזהה ייחודי לכל תלמיד (נוצר אוטומטית).
* **name:** שם התלמיד.
* **grade:** הציון שקיבל.

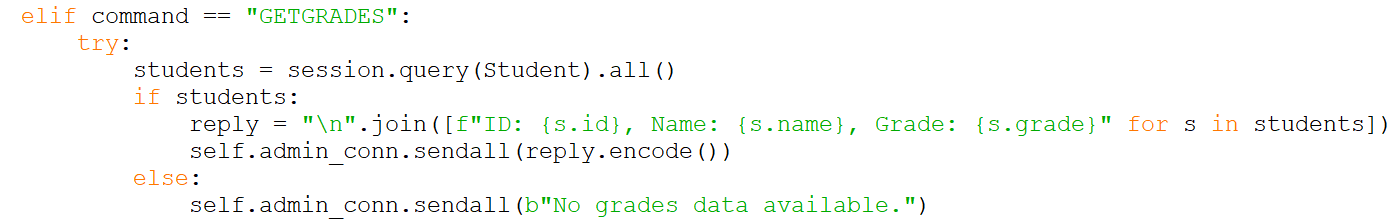
**יצירת חיבור למסד**

הפונקציה init\_db() אחראית על יצירת Session מול מסד הנתונים:



**שליפת ציונים – GETGRADES**

כשהמורה שולח את הפקודה GETGRADES, השרת שולף את כל הציונים הקיימים ושולח אותם חזרה, כך (main\_server\_loop):



**דרכים להוספת ציונים**

במערכת קיימות שתי דרכים להוספת ציונים למסד הנתונים, ושתיהן משתמשות באותו מנגנון בשרת.

**1. הוספה ידנית דרך ממשק הניהול**

בממשק הניהול (Admin), למורה מוצג כפתור המאפשר להזין ציון לתלמיד, לאחר הלחיצה על הכפתור המורה מזין ציון בין 0-100 ובוחר תלמיד אשר קיבל רת הציון. כאשר כפתור השליחה נלחץ, נשלחת לשרת פקודת GRADE בפורמט הבא:

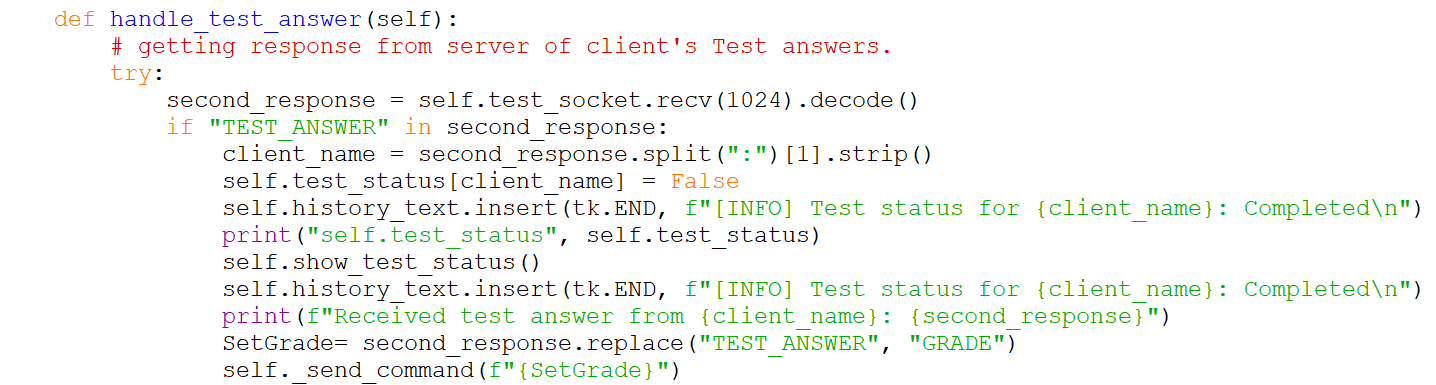


השרת מזהה את הציון והשם (או ממיר את כתובת ה-IP לשם במידת הצורך), יוצר מופע חדש של תלמיד עם הציון, ושומר אותו במסד הנתונים. בנוסף, נשלחת לתלמיד הודעה אישית המציינת את הציון שקיבל, לדוגמה:



**2. הוספה אוטומטית לאחר סיום מבחן**

כאשר תלמיד מסיים מבחן, קוד התלמיד (client) מחשב את הציון באופן אוטומטי ושולח אותו לשרת בפורמט של TEST\_ANSWER <grade> : <client\_name>. כלומר, גם כאן, הפקודה דומה.

השרת מבצע שליחה אל המורה. המורה מחליף בצורה אוטומטית את המילה TEST\_ANSWER"" ב"grade" ושולח באופן אוטומטי דרך send\_command:  בכך נשמרת אחידות בין שתי הדרכים, והציונים נשמרים במסד הנתונים באופן עקבי ואוטומטי.

1. מבט מודולרי

המערכת מחולקת לשלושה רכיבים עיקריים: שרת, מורה (Admin) ותלמיד (Client). כל רכיב ממומש כקובץ נפרד, המכיל מחלקה אחת בלבד המרכזת את כל הלוגיקה הרלוונטית לאותו רכיב:

**()Server –** המחלקה היחידה בקוד השרת. מרכזת את כל הפונקציונליות של צד השרת: יצירת חיבורים מאובטחים עם לקוחות, ניהול תקשורת במקביל באמצעות threads ו-select, הפצת פקודות, תיעוד נתונים, וטיפול בשינויים בזמן אמת.

**()AdminClient –** המחלקה היחידה בקוד המורה. אחראית על ממשק המשתמש הגרפי של המורה, שליחת פקודות לשרת, קבלת נתונים, והצגתם בצורה נוחה תוך שליטה מלאה במחשבי התלמידים.

**ClientApp() –** המחלקה היחידה בקוד התלמיד. מופקדת על קבלת הוראות מהשרת, ביצוען באופן אוטומטי (כגון חסימת מקלדת, הצגת מבחן), ושמירה על תקשורת תקינה ויציבה מול השרת.

גישה זו מבטיחה הפרדה ברורה בין רכיבי המערכת, תחזוקה נוחה יותר, ושיפור הקריאות של הקוד.

1. פירוט מודלים עיקריים

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | | Input\Output | Function | Class |
|  | ---SERVER--- | |  |  |
| הפעולה שמאתחלת את השרת. היא שומרת את פרטי ההגדרות, מגדירה את ההקשרים,(SSL) יוצרת את כל הSockets הדרושים לתקשורת עם המורה והתלמידים, ומבצעת bind + listen לports השונים. | | |  | | --- | |  |   Input:   * server\_ip- כתובת ה-IP של השרת * - admin\_port port לתקשורת עם המורה * -admin\_port\_test port נפרד לשליחת ציוני מבחן עם המורה * client\_port - port לתקשורת עם תלמידים * client msg clientlist port port נוסף להעברת הודעות או רשימות תלמידים   output:  אין (הפעולה היא פעולה בונה – constructor, אין לה פלט ישיר). | \_\_init\_\_ | SERVER() |
| שתי פונקציות בשרת שמאזינות לפורטים ייעודיים ומקבלות חיבורים מהמורה – אחת לבדיקות מבחן, והשנייה לשליחת הודעות ורשימות תלמידים. | | None | Accept admin test  connection\ accept admin msg clientlist connection |  |
| מפעילה את שרת השליטה המרכזי. מאזינה לחיבור מהמורה ומפעילה שלוש Threads מקבילות: קבלת ציונים, ניהול הודעות, וחיבור לקוחות.  לאחר מכן מפעילה את הלולאה הראשית. | | None | start |  |
| מאזינה ברקע לחיבורים חדשים מתלמידים.  עם  חיבור חדש, מאמתת את שם התלמיד,  שולחת אישור/שגיאה, ומעדכנת את רשימות המעקב של החיבורים והמיפויים. | | Input: חיבורים נכנסים דרך self.client\_sock (לא ישירות מהמשתמש).  :Output אין פלט ישיר. נשלחים מסרים לממשק המורה לעדכון הרשימה, ונשמרים נתונים פנימיים בשרת ,IP) שם, socket וכו'). | Handle client connections |  |
| מקבלת נתוני צילום מסך מהקליינט, שומרת את התמונה כקובץ עם שם הכולל את שם התלמיד וזמן ההגעה, ומעדכנת את המורה. | | :input client\_conn (ה-socket של התלמיד), data (התחלה של תמונה עם "PIC\_START"), client\_info (מילון עם מידע כמו שם התלמיד).  :output קובץ תמונה מקומי (פורמט PNG), הודעה לממשק המורה עם שם הקובץ או שגיאה במקרה הצורך. | Handle client screenshot |  |
| שולחת למורה את קובץ צילום המסך האחרון שנשמר מהתלמיד המבוקש, אם קיים. | | :input client\_name (מחרוזת עם שם התלמיד).  Output: שליחה של קובץ PNG דרך חיבור ה־socket של המורה, או הודעת שגיאה. | Send last file |  |
| בודקת האם חיבור socket פתוח. | | :input conn (אובייקט socket או SSL socket).  Output: bool (True אם החיבור פעיל, אחרת False). | Is socket open |  |
| מאתר sockets של תלמידים לפי שם, כתובת IP, או המילה "all" (כדי לפעול על כולם). | | :input client\_name\_or\_ip (מחרוזת – שם תלמיד, IP או המילה "all").  Output: רשימת sockets (או None אם לא נמצא). | Get target ip |  |
| לולאה ראשית של השרת המאזינה באופן רציף לנתונים מ-admin ומהלקוחות. מטפלת בפקודות admin, שולחת פקודות/נתונים ללקוחות ומקבלת מהם הודעות שונות. מנהלת מצב לקוחות מחוברים, מעדכנת מסד נתונים עם ציונים, מטפלת בקבלת קבצים/צילום מסך ומגיבה בהתאם. בלולאה יש טיפול שגיאות למניעת קריסות והסרת לקוחות שניתקו. בסיום סוגרת את כל החיבורים. | | :input אין.  Output: עקרונית אין, אבל:   * שליחה של תגובות ופקודות דרך socket ל-admin ו-clients. * עדכון רשימות לקוחות, מסד נתונים (למשל ציונים). * סגירת חיבורים במקרה של ניתוקים או סיום הלולאה. * הדפסות. | Main server loop |  |
|  | | ---ADMIN--- |  |  |
| מאתחלת את האדמין עם פרטי חיבור, מגדירה משתנים פנימיים ובונה את הממשק. | | :input   * **admin\_ip** :כתובת ה-IP של האדמין. * **admin\_port:** פורט החיבור הראשי של האדמין. * **admin\_test\_port:** פורט מיועד לקבלת תוצאות ממבחנים של תלמידים. * **:Client msg clientlist port** פורט עבור קבלת הודעות ורשימת לקוחות.   Output: אין. | \_\_init\_\_ | AdminClient() |
| שתי פונקציות שעובדות יחד, Read help קוראת תוכן מקובץ בשם help ומחזירה את התוכן שלו וdisplay help מקבלת את תוכן זה ומציגה אותו על מסך המורה. | | Output(read help): מחזירה תוכן קובץ txt בשם help, אם לא קיים מחזירה הודעה מתאימה. | Read help\ display help |  |
| יוצרת חיבור מוצפן (SSL) בין המורה לשרת, כולל חיבורים נוספים לבדיקה ושליחת הודעות. | | None | Connect to server |  |
| מופעלת כאשר המשתמש סוגר את חלון האדמין. שואלת לאישור סגירה, סוגרת סוקטים פעילים ומפסיקה את הממשק הגרפי (GUI). | | None | On close |  |
| בונה את ממשק המשתמש הגרפי (GUI) של admin בעזרת Tkinter — כולל חלונות לפקודות, רשימת לקוחות, היסטוריה, הודעות, כפתורים, וcanvas. | | None | Setup ui |  |
| מפעילה גלילה אנכית בCanvas באמצעות סרגל גלילה וגלגלת העכבר. | | :input width – גובה האזור הנגלל (scrollregion), הגובה של מה שאנו רוצים להציג על מסך זה.  Output: אין. | Enable screenshot scroll |  |
| מבטלת את הגלילה בקנבס של צילום המסך, מסירה את הסקרולבר ומכבה את גלגלת העכבר. | | None | Disable screenshot scroll |  |
| הפונקציה יוצרת תהליך רקע (Thread) שמאזין להודעות מהשרת בערוץ msg\_clientlist\_socket. בהתאם לסוג ההודעה המתקבלת (חיבור חדש, הודעת טקסט, ניתוק לקוח), הפונקציה מפעילה פעולות מתאימות כמו רענון רשימת לקוחות, הצגת הודעות או רישום היסטוריה. אם מתרחש ניתוק מהשרת באופן לא יזום, הפונקציה סוגרת את הממשק הגרפי. | | None | Start client msg clientlist listener |  |
| הפונקציות מאפשרות לאדמין לצייר עם צבע וגודל מברשת שונים על הcanvas בתוך ממשק Tkinter. ניתן לנקות את הציור במלואו או לשלוח את התמונה המעודכנת ללקוח נבחר דרך הרשת. השליחה כוללת צילום מסך של אזור ה־canvas, שמירת התמונה ושליחתה רק אם הלקוח אינו במצב מבחן. כל פעולה מטופלת עם טיפול בשגיאות למניעת קריסות. | | Input(paint): event- מיקום העכבר.  Output: אין. | Open paint, paint, clear canvas, send paint |  |
| הפונקציה מציגה על הcanvas את סטטוס המבחנים של כל הלקוחות המחוברים.  תלמידים בצבע ירוק – בתוך מבחן, בצבע אדום- לא בתוך מבחן. | | Input: refresh (בוליאני, אופציונלי): אם True, הפונקציה תרענן קודם כל את רשימת הלקוחות לפני הצגת הסטטוס. ברירת המחדל היא False.  :Output אין. | Show test status |  |
| הפונקציה פותחת חלון לבחירת לקוח מחוברים או "ALL". אם אין לקוחות, מציגה אזהרה. מחזירה את הבחירה או None אם בוטל. | | Input: allow\_all (ברירת מחדל: True)- האם להציג אפשרות "ALL" לבחור את כל הלקוחות.  :Output שם הלקוח שנבחר (מחרוזת), "ALL" או None אם לא נבחר כלום. | Choose client |  |
| הפונקציה מקבלת תגובה מהשרת עבור לקוח ספציפי או עבור כל הלקוחות. היא קוראת את התגובה מה-socket, מעדכנת את מצב הממשק בהתאם לתגובה (כגון הפעלת כפתור קובץ אחרון, שליחת בקשה לקבל ציונים), רושמת את ההיסטוריה, ומעדכנת סטטוס מבחן אם לקוח עונה על מבחן. במקרה של מבחן, מתחילה טיפול נוסף ברקע לקבלת תוצאות. במידה ויש שגיאה בתקשורת, היא מדווחת עליה. | | Input:   * command — מחרוזת הפקודה שנשלחה לשרת. * client\_name — שם הלקוח שממנו מצפים לתגובה (או "all" לכל הלקוחות).   :Output אין. | Handle server reply |  |
| הפונקציה מקבלת טקסט מקלט הפקודה (Entry), ובודקת אם המשתמש במצב של שליחת הודעה (msg\_mode). אם כן, היא מבקשת לבחור לקוח לשליחת ההודעה אליו, שולחת את ההודעה ומעדכנת את מצב הממשק. אם לא במצב הודעה, שולחת את הפקודה הרגילה לשרת. בכל מקרה, מנקה את שדה הקלט ומטפלת בשגיאות אפשריות. | | None | Get command |  |
| הפונקציה שולחת פקודות לשרת דרך socket פתוח ומטפלת במקרים מיוחדים:  אם אין חיבור פעיל, מציגה הודעת שגיאה.  בפקודת HELP מציגה עזרה מקומית.  בפקודת CLIENTLIST מעדכנת רשימת לקוחות.  בפקודת GETGRADES שולחת בקשה ומציגה ציונים על הקנבס.  בפקודות שליחת קובץ קוראת את הקובץ, מחשבת בדיקת שלמות (checksum) ושולחת את התוכן בפרוטוקול מיוחד.  בכל מקרה אחר שולחת את הפקודה לשרת ומפעילה באופן נפרד טיפול בתגובה בשרשור רקע.  הפונקציה דואגת גם ללוג שגיאות. | | Input: מחרוזת command שמייצגת פקודה לשליחה לשרת (כגון פקודות לשליחת קובץ, בקשת רשימת לקוחות, שליחת הודעה, בקשת ציונים וכו').  :Output אין (שולחת את הפקודה לשרת). | Send command |  |
| מציג הודעות לוג בחלון הטקסט של ההיסטוריה. | | Input: מחרוזת טקסט (message) לתיעוד בלוג.  :Output אין (מציג את ההודעה בלוג). | Log history |  |
| מציג הודעות שנשלחו מלקוחות בחלון ייעודי להודעות. | | Input: מחרוזת טקסט (message) להודעות מלקוחות.  :Output אין (מציג את ההודעה בחלונית הודעות). | Log client message |  |
| שולח בקשה לשרת לקבלת רשימת לקוחות מחוברים, ומעדכן את ה-Listbox עם השמות; גם מסיר מסטטוס הבדיקה לקוחות שאינם מחוברים יותר. | | None | Refresh client list |  |
| מבקש מהשרת את הקובץ האחרון של לקוח מסוים, מוריד את הנתונים, מאמת את תקינותם, ומציג את התמונה בממשק. אם אין קובץ או בעיה, מדווח על כך בלוג. | | None | Request last file |  |
|  | | ---CLIENT--- |  |  |
| אתחול ממשק הלקוח (client GUI) עם הגדרות בסיסיות של חלון, משתנים ראשוניים, וקריאה ליצירת מסך שם. | | Input: אובייקט master של Tkinter (חלון ראשי).  :Output אין. | \_\_init\_\_ | ClientApp() |
| יוצר מסך בו המשתמש מוזמן להזין את שמו. כולל שדה טקסט, כפתור אישור שמופעל רק אם הוזן שם כלשהו, ואירועים לבדיקת תקינות הקלט. | | None | Create name screen |  |
| בודק את תקינות השם (ללא נקודות, סוגריים, המילה "all", או אותיות עבריות). אם השם לא תקין – מציג הודעת שגיאה ומחזיר למסך הזנת שם. אם תקין – עובר למסך ממשק המשתמש הראשי. | | Input: שם שהוזן בשדה הטקסט.  :Output אין. | Set client name |  |
| יוצרת את ממשק המשתמש הראשי עבור לקוח ה-Tkinter לאחר שהמשתמש הזין את שמו.  מגדירה את חלון התוכנה, אזורי קלט ופלט (הודעות, לוג, קבצים), כפתורי שליטה, סרגלי גלילה, ומכינה משתני רשת ראשוניים. בנוסף, מפעילה לולאת בדיקה לתור ההודעות. | | None | Setup ui |  |
| פונקציה שמצלמת צילום מסך (Screenshot), שומרת אותו כקובץ PNG עם שם מבוסס זמן נוכחי, ואז שולחת את קובץ התמונה לשרת בחתיכות (Chunks) דרך סוקט.  במקרה של שגיאות, היא מדווחת אותן בלוג המערכת. | | None | Take screenshot |  |
| הפונקציה מקבלת הודעה מהשרת ומבצעת פעולות בהתאם לסוג ההודעה (פקודה). היא מזהה האם מדובר בהוראות חסימת מקלדת, פתיחה, קבלת קבצים, הודעות טקסט, בקשה לצילום מסך או פקודות ניהול אחרות, ומגיבה בהתאם דרך עדכון ה-GUI או פעולות פנימיות. | | Input: message (bytes): הודעה המגיעה מהשרת, בפורמט של מחרוזת בייטים עם מזהה לפקודה (כמו BLOCK, UNBLOCK, DATA\_NAME וכו').  Output: אין. | Handle message |  |
| מתבצעת בלחיצת כפתור – מפעילה תהליך נפרד (thread) שמריץ את הפונקציה run\_client, ואחראית להתחברות לשרת. | | None | Start client |  |
| האחראית לחיבור לשרת.  בודקת אם שם המשתמש פנוי, ואם כן מאזינה להודעות מהשרת ומעבירה אותן לטיפול.  במקרה של שגיאה או כיבוי – מבצעת ניקוי (cleanup). | | None | Run client |  |
| מנקה ומסיימת את פעילות הלקוח (client) כשהמשתמש בוחר לצאת בכפתור X, כשהמורה מנתק את התלמיד או כשמתבצע סיום אוטומטי של התוכנית. שולחת הודעת סיום לשרת, מכבה תהליכים, ומסוגרת את חלון ה־Tkinter. | | Input: confirm (ברירת מחדל: True) – אם יש לבקש מהמשתמש אישור לפני הסיום (לדוגמה בלחיצה על כפתור X).  Output: אין. | Clean up |  |

1. תסריטי בדיקה
   1. דגשים בבדיקה

במהלך פיתוח הפרויקט השתמשתי בשיטה שנקראת testing unit ובדקתי כל

אלגוריתם בנפרד. הכנסתי פעולות מהמורה דרך כתיבה בCommand line ורק כאשר ראיתי שעובד החלטתי להעביר לכפתורים.

דוגמאות לבדיקות שעשיתי:

* תקינות התקשורת בין רכיבי המערכת (SSL, Socket).
* יכולת זיהוי וחיבור של לקוחות חדשים בזמן אמת.
* שליחה וקבלה של פקודות מסוגים שונים (חסימת מקלדת, שליחת מבחן, קבלת צילום מסך וכו').
* טיפול בהתנתקות לקוחות והתחברות מחדש.
* תגובה תקינה לקלטים חריגים או שגויים (לדוגמה: קובץ פגום, חיבור לקוח לא מוכר).
* עדכון ממשק המשתמש בהתאם למצב הרשת בפועל.
* שמירה על יציבות המערכת תחת עומס (מספר רב של תלמידים פעילים במקביל).

1. תסריטי בדיקה עיקריים

(הכנתי לעצמי לקראת הבדיקות הסופיות של הפרוייקט)

1. העלאת ADMIN ללא שרת, ניסיון התחברות וקבלת ERROR מסודר. שים לב לכך ב CLIENT זה עף בתסריט הזה שאין שרת. יש לחשוב אם פותרים.

2. העלאת admin ושרת, connect, קבלת חיווי חיבור ב admin. הצגת HELP וקבלת תוצאה נוחה לקריאה כולל כל הפקודות. בדיקה ש HELP לא עף כאשר אין קובץ HELP. בדיקה שבחלונית clients לא מופיע אף לקוח. כפתור CONNECT DISABLED.

3. ניסיון ביצוע פעולות ללא client, כגון send file ALL או block ALL – ווידוא חסימה על ידי ה admin, בהודעה מתאימה.

4. העלאת client(במחשב שונה מה admin). בחירת שם לא חוקי, מניעה. העלאה ללא שם (חסימה?). בחירת שם נכון. ביצוע חיבור. קבלת חיווי חיבור ב client. הופעת הלקוח החדש באופן אוטומטי בחלונית clients ב admin. (לחילופין ביצוע Refresh ב admin וקבלת המשתמש המחובר ברשימה.)

5. העלאת clientנוסף באותו מחשב (אך שונה מה admin). בחירת שם תפוס. מניעה. בחירת שם טוב. כפתור EXIT DISABLED. ביצוע חיבור START. קבלת חיווי חיבור ב client. הופעת הלקוח החדש באופן אוטומטי בחלונית clients ב admin, כפתור START בלקוח DISABLED. EXIT ENABLED.

6. יציאה מה client על ידי EXIT. וידוא יציאה מסודרת ללא תעופה, עדכון אוטומטי של הלקוחות ב ADMIN.

7. העלאה מחודשת של ה client, בחירת אותו שם ואישור, עדכון אוטומטי של הלקוחות ב ADMIN.

8. ביצוע block ו unblock ליחיד ולכולם. בדיקה שה block אכן חוסם עכבר ומקלדת, וה unblock משמיש אותם. הודעות ב LOG של ADMIN על הפעולות כולל שעת ביצוע וחיווי.

9. ביצוע request last file בהיעדר קובץ בשרת, הודעת חיווי בנושא. לחיצת SCREENSHOT ובחירת אחד הלקוחות. קבלת אישור בלקוח וב ADMIN. חזרה על request last file, אפשרות בחירה של הלקוח שאין לו SCREENSHOT וקבלת חיווי על היעדר הצלחה. בקשת SCFEENSHOT גם ממנו והצגתו ב ADMIN.

10. ציור PAINT על ה SCREENSHOT ושליחתו ללקוח הספציפי ממנו הוא התקבל – האם זה מתבצע אוטומטית?

11. בקשת GETGRADES והצגת רשימה ריקה. לבדוק שלא עף כאשר חסר DB, או שה DB ריק.

12. בקשת SENDFILE ל ALL לקובץ טקסט פשוט. קבלת פידבק שהקובץ נשלח ללקוחות, הצגת הקובץ בשני הלקוחות. חזרה על הפעולה עם קובץ תמונה.

13. שליחת הודעה (אחת ארוכה 50 מילים) ללקוח, וידוא הצגת ההודעה כולה אצל הלקוח וחיווי הצלחה ב ADMIN. חזרה על הפעולה לגבי ALL.

14. שליחת מבחן קצר ללקוח אחד. קבלת פידבק על תחילת המבחן והפעלת TIMER. מענה לקוח 100 וקבלת חיווי ל ADMIN. שליחת ציון אוטומטי ללקוח מה ADMIN ותצוגה.

15. שליחת מבחן קצר ללקוח השני. קבלת פידבק על תחילת המבחן והפעלת TIMER. אין מענה וה TIMER הסתיים, לקוח לא ענה וקיבל 0. וקבלת חיווי ל ADMIN. שליחת ציון אוטומטי ללקוח מה ADMIN ותצוגה.

16. בקשת GETGRADES והצגת הציון של שני הלקוחות שנבחנו.

17. שליחת מבחן רגיל לכל הלקוחות. קבלת פידבק על תחילת המבחן מכל לקוח בנפרד והפעלת TIMER. מענה לקוח 50, 100 וקבלת חיווי ל ADMIN. שליחת ציון אוטומטי ללקוח מה ADMIN ותצוגה. בקשת GETGRADES והצגת הציון של שני הלקוחות שנבחנו (פעמיים).

18. לבצע כאשר ALL מבצע גם TIMER שנגמר ואחד שעונה. נסה להגדיר את משך ה TIMER הנכון למענה 60 שניות או פחות. אפשר גם לבצע במהלך ההמתנה הודעות ללקוח SEND MSG לכולם או SCREENSHOT למישה ספציפי, ולראות אם ההודעות MSG מוצגות אצל הלקוח ועדיין המבחן נשלח בהצלחה ל ADMIN.

19. יציאה מהשרת. קבלת FEEDBACK ב ADMIN ובלקוחות ללא נפילה. גם אם לא עובד ב לקוחות, בסדר, רק צריך לדעת.

20. העלאת client ולחיצת start ללא שרת, העלאת admin ללא שרת ולחיצת connect, קבלת הודעת שגיאה ללא קריסה.

21. כפילויות, מבחנים.

1. רפלקציה
   1. לוח זמנים מוערך לניהול הפרויקט:

**סוף ספטמבר\תחילת אוקטובר-** נסגרתי על רעיון הפרוייקט והתחלתי לתכנן.

**נובמבר-** פיתחתי תקשורת ראשונית בין השרת למורה לתלמיד, הוספתי פעולות חסימה וביטול חסימה.

**דצמבר\תחילת ינואר-** הוספתי אופציה לצילום מסך והצגתו על מסך המורה, שיפרתי את התצוגה והתקשורת.

**ינואר\פברואר-** הוספתי שליחת הודעות וקבצים מהמורה אל התלמיד, כפתור הורדת קבצים בתלמיד, אופציה לציור עם צילום מסך של תלמיד ושליחה חזרה.....

**מרץ\אפריל-** הוספתי מסד נתונים, הצפנה, ציונים, מבחנים, ניתוק כל אחד מרכיבי הפרוייקט, הודעות מהתלמיד, ריפרש אוטומטי ל……..clientlist

**סוף אפריל-** בדיקות ומקרי קצה.

1. אתגרים ותרומה אישית

* באמצע השנה עברתי ניתוח אשר מנע ממני לעבוד על הפרוייקט לתקופה מסויימת אך בסופו של דבר זה רק גרם לי עוד יותר לרצות לעבוד עליו ולשפר אותו וזו הייתה נקודת המפנה שלי השנה.
* האתגר המרכזי היה סנכרון תקשורת מאובטחת בין שלושה רכיבים (שרת, אדמין, קליינט) תוך שימוש ב-SSL.
* אתגר נוסף היה תכנון מערכת נוחה לשימוש שתאפשר למורה לנהל את התלמידים בצורה שוטפת בזמן אמת.
* תרמתי באופן מלא לתכנות צד השרת, ממשק ה-Admin וה-client, תוך הקפדה על ארכיטקטורה מודולרית, חווית משתמש, ותיעוד איכותי.
* איך לחכות לתוצאות מבחנים מתלמידים מבלי שהמורה "יקרוס".
* טיפלתי בבאגים שעלו בתסריטי הבדיקה ויישמתי פתרונות יזומים לבעיות שלא נצפו מראש.

1. תובנות

* גיליתי כמה חשוב לתכנן היטב מבנה קוד ואופן עבודה של כל רכיב בפרויקט עם תקשורת רשתית.
* למדתי על הצורך באבטחה גם בפרויקטים פנימיים – השימוש ב-SSL, טיפול נכון בשמות משתמשים, והתייחסות לתקינות מידע.
* העבודה עם Tkinter ו-Sockets חיזקה לי את ההבנה בתקשורת בזמן אמת ובפיתוח ממשקים יעילים ונגישים.
* החוויה חיזקה בי את התשוקה לעולמות הסייבר, האבטחה והתקשורת, וגרמה לי להבין שזה התחום שבו אני רוצה להעמיק ולהתמקצע בעתיד.
* למדתי הרבה מעבר למה שציפיתי ללמוד בפרוייקט והוא חידש לי המון.

1. הוראות התקנה ותפעול
   1. תצורה ודרישות קדם

* מערכת הפעלה: Windows 10 / 11 (גרסת 64 ביט מומלצת)
* גרסת פייתון: Python 3.10 ומעלה (לוודא התקנה מלאה כולל pip)
* נדרש חיבור לרשת מקומית (LAN) בין כל הרכיבים (שרת, אדמין, לקוחות)
* התקנת ספריות צד שלישי באמצעות pip (רשימה מלאה בהמשך)
* הרצת כל רכיב (שרת, אדמין, לקוח) מתבצעת על מחשב נפרד או באותה הרשת
* יש לאפשר גישה דרך חומת אש לפורטים הנדרשים

1. התקנה

**קבצים נדרשים (יש לשמור את הקבצים בתיקיות המתאימות בכל מחשב):**

server.py – קוד השרת

admin.py – קוד המורה

client.py – קוד התלמיד

students.db – מסד נתונים

server.key – מפתח פרטי

server.pem – תעודת SSL

help.txt – קובץ עזרה והסבר

קבצי מבחנים בפורמט: test.txt

**התקנת ספריות פייתון (יש להריץ ב-CMD עם הרשאות מנהל):**

****

* **Pyautogui**
* **Pillow**
* **Sqlalchemy**

**הרצה**

יש לפתוח את ה־CMD בכל מחשב במצב מנהל:

* הקש על כפתור Start → הקלד cmd → לחץ קליק ימני → Run as Administrator
* עבור לתיקיה בה נמצאים הקבצים באמצעות הפקודה cd

**סדר הפעלה מומלץ:**

**הפעלת השרת:**

cd path\to\folder

python server.py

**הפעלת המורה:**

cd path\to\folder

python admin\_client.py

**הפעלת התלמיד:**

cd path\to\folder

python client\_app.py

**חובה להריץ כל קובץ במצב Admin – אחרת חסימות מקלדת ועכבר לא יעבדו כראוי.**

1. ביבליוגרפיה

במהלך כתיבת הפרויקט הסתמכתי על מספר מקורות מידע:

• https://stackoverflow.com – פתרון בעיות קוד וטעויות נפוצות.

• https://www.geeksforgeeks.org – הסברים ודוגמאות לשימוש בספריות כמו socket, threading, Tkinter ועוד.

• https://chat.openai.com – קבלת עזרה בניסוח, תכנון ופתרון בעיות לוגיות בפרויקט.

• https://docs.python.org/3 – תיעוד רשמי של שפת Python והמודולים הסטנדרטיים.

• https://docs.sqlalchemy.org – תיעוד רשמי של ספריית SQLAlchemy בה השתמשתי לניהול מסד הנתונים של ציונים.

• https://pillow.readthedocs.io – תיעוד של ספריית Pillow (PIL) לעיבוד תמונה, בשימוש בצילומי מסך ושליחת תמונות.

• https://docs.python.org/3/library/tkinter.html – תיעוד רשמי של Tkinter, כלי לבניית ממשק המשתמש הגרפי.

• https://pyautogui.readthedocs.io – תיעוד ספריית pyautogui בה השתמשתי לשליטה בעכבר ובמקלדת (חסימה ושחרור).

• https://realpython.com/python-sockets/ – מדריכים על תקשורת Socket עם דוגמאות ותרגולים.

• https://slproweb.com/products/Win32OpenSSL.html – הורדת ספריית OpenSSL לגרסה תואמת של Windows עבור יצירת SSL certificates.

• https://www.ssl.com/how-to/create-a-self-signed-certificate-using-openssl/ – יצירת קבצי server.key ו-server.pem לשרת מאובטח.

1. נספחים

## admin.py

# admin code  
  
import socket  
import tkinter as tk  
from tkinter import ttk, messagebox, Canvas, simpledialog, filedialog  
from PIL import ImageGrab  
from PIL import Image, ImageTk, ImageOps  
from io import BytesIO  
import hashlib  
import ssl  
import os  
import time  
import threading  
import queue  
import sys  
  
"""  
AdminClient is the main class for the admin interface in the classroom management system.  
It sets up the GUI, handles server communication via SSL, and manages connected clients.   
"""  
   
class AdminClient:  
 def \_\_init\_\_(self, admin\_ip, admin\_port, admin\_test\_port, client\_msg\_clientlist\_port):  
 self.admin\_ip = admin\_ip  
 self.admin\_port = admin\_port  
 self.admin\_test\_port = admin\_test\_port  
 self.client\_msg\_clientlist\_port = client\_msg\_clientlist\_port  
  
 # Socket variables for secure and plain connections  
 self.socket = None  
 self.test\_socket = None  
 self.msg\_clientlist\_socket = None  
  
 # Client tracking  
 self.connected\_clients = []  
 self.test\_status = {} # Tracks if client is currently taking a test  
 self.selected\_file\_info = {}  
 self.msg\_mode = False  
 self.shutdown = False  
  
 # Build the user interface   
 self.\_setup\_ui()  
  
 """  
 The function allows the display of an external file help.txt  
 in the log history and display it on the canvas, that contains explanations all functions.  
 """  
 def \_read\_help(self):  
 try:  
 with open("help.txt", 'r') as help\_file:  
 content = help\_file.read()  
 return content  
 except FileNotFoundError:  
 return "Error: help.txt file not found."  
 except Exception as e:  
 return f"An error occurred while reading help.txt: {e}"  
  
  
 def display\_help(self):  
 self.enable\_screenshot\_scroll(3500) # Enables scrolling if necessary  
 content = self.\_read\_help() # Read the help content from help.txt  
 self.screenshot\_canvas.delete("all") # Clear previous content on the canvas  
 self.screenshot\_canvas.create\_text(  
 10, 10,  
 text=content,  
 font=("Arial", 20),  
 fill="white",  
 anchor="nw",  
 width=580) # <-- This is the important part! Set width to wrap text)  
  
  
  
 def \_connect\_to\_server(self):  
 try:  
 try:  
 # sock is the admin sock before applying ecryption SSL -> self.socket  
 sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.test\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.msg\_clientlist\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 ssl\_context = ssl.create\_default\_context()  
 ssl\_context.check\_hostname = False # Disable hostname verification  
 ssl\_context.verify\_mode = ssl.CERT\_NONE # Don't require a trusted CA  
   
 # this is the main admin socket after encryption.  
 self.socket = ssl\_context.wrap\_socket(sock, server\_hostname=self.admin\_ip)  
  
 print ("admin requested connection, ", self.admin\_port)  
 self.socket.connect((self.admin\_ip, self.admin\_port))  
 print ("admin was approved connection")  
 except:  
 print("cannot connect to server")  
 messagebox.showwarning("connection Error", "cannot connect to server.")  
 return  
 try:  
 self.test\_socket.connect((self.admin\_ip, self.admin\_test\_port))  
 self.\_log\_history("Connected to server as test admin.")  
 except Exception as e:  
 self.\_log\_history(f"Failed to connect to test port: {e}")  
 try:   
 self.msg\_clientlist\_socket.connect((self.admin\_ip, self.client\_msg\_clientlist\_port))  
 self.\_log\_history("Connected to server as msg \ clientlist admin.")  
 except Exception as e:  
 self.\_log\_history(f"Failed to connect to msg/clientlist port: {e}")  
   
  
 # Disable the button and change text  
 self.connect\_button.config(text="Connected", state=tk.DISABLED)  
  
 # Start background listener after successful connection  
 self.\_start\_client\_msg\_clientlist\_listener()  
  
   
 except Exception as e:  
 messagebox.showerror("Connection Error", f"Failed to connect to server: {e}")  
  
 def on\_close(self):  
 """  
 Triggered when the window is closed. Closes sockets and shuts down the GUI.  
 """  
 if messagebox.askokcancel("Quit", "Are you sure you want to quit?"):  
 self.shutdown = True  
 # Optional: clean socket resources  
 if self.socket:  
 try:  
 self.socket.close()  
 except:  
 pass  
 if self.test\_socket:  
 try:  
 self.test\_socket.close()  
 except:  
 pass  
 if self.msg\_clientlist\_socket:  
 try:  
 self.msg\_clientlist\_socket.close()  
 except:  
 pass  
  
 print("Closing application. \n")  
 self.root.destroy()  
  
 def \_setup\_ui(self):  
 """  
 Sets up the Tkinter GUI layout and components for the admin client.  
 Includes error handling to catch and report issues during UI creation.  
 """  
 # Initialize main window  
 self.root = tk.Tk()  
 self.root.title("Admin Client GUI")  
   
 # Window size & resizing settings  
 self.root.geometry("800x600") # Default size  
 self.root.minsize(888, 666) # Minimum allowed size  
 self.root.maxsize(888, 666) # Maximum allowed size  
 self.root.resizable(True, True) # Allow resizing  
   
 # Set background color  
 self.root.config(bg="#f0f0f0")  
  
 # Bind the close event to the on\_close method  
 self.root.protocol("WM\_DELETE\_WINDOW", self.on\_close)  
  
 # Frames for different sections  
 command\_frame = tk.Frame(self.root, bg="lightgray", width=200, height=150)  
 command\_frame.pack(side=tk.TOP, fill=tk.X)  
  
 history\_frame = tk.Frame(self.root, bg="white", width=200, height=150)  
 history\_frame.pack(side=tk.BOTTOM, fill=tk.X)  
  
 client\_list\_frame = tk.Frame(self.root, bg="lightblue", width=200)  
 client\_list\_frame.pack(side=tk.LEFT, fill=tk.Y)  
   
 screenshot\_frame = tk.Frame(self.root, bg="black")  
 screenshot\_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)  
  
 # Command Section  
 self.connect\_button = tk.Button(command\_frame, text="Connect to Server", command=self.\_connect\_to\_server)  
 self.connect\_button.pack(side=tk.LEFT, padx=5, pady=5)  
  
 execute\_button = tk.Button(command\_frame, text="Execute", command=self.\_get\_command)  
 execute\_button.pack(side=tk.RIGHT, padx=5, pady=5)  
  
 self.command\_entry = tk.Entry(command\_frame, font=("Arial", 12))  
 self.command\_entry.insert(0, "sending message to clients here")  
   
 self.command\_entry.config(state="disabled")  
 self.command\_entry.pack(side=tk.RIGHT, padx=5, pady=5, fill=tk.X, expand=True)  
 self.command\_entry.bind("<Return>", lambda event: self.\_get\_command())  
  
 # "Request Last File" button  
 self.last\_file\_button = tk.Button(command\_frame, text="Request Last File", command=self.\_request\_last\_file)  
 self.last\_file\_button.pack(side=tk.LEFT, padx=5, pady=5)  
  
 # History - split into two parts  
 # History - split into two parts with equal width using grid  
 history\_frame.grid\_rowconfigure(0, weight=1)  
 history\_frame.grid\_columnconfigure(0, weight=1)  
 history\_frame.grid\_columnconfigure(1, weight=1)  
  
 # Left Frame  
 left\_history\_frame = tk.Frame(history\_frame, bg="white")  
 left\_history\_frame.grid(row=0, column=0, sticky="nsew")  
  
 left\_title = tk.Label(left\_history\_frame, text="Command History", font=("Arial", 12, "bold"), bg="white")  
 left\_title.pack(side=tk.TOP, anchor="w", padx=5, pady=(5, 0))  
  
 left\_text\_container = tk.Frame(left\_history\_frame, bg="white")  
 left\_text\_container.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)  
  
 # Left Text + Scrollbar using grid  
 self.history\_text = tk.Text(left\_text\_container, wrap=tk.WORD, bg="white", state="disabled", height=8)  
 self.history\_text.grid(row=0, column=0, sticky="nsew")  
  
 history\_scrollbar = tk.Scrollbar(left\_text\_container, orient=tk.VERTICAL, command=self.history\_text.yview)  
 history\_scrollbar.grid(row=0, column=1, sticky="ns")  
  
 left\_text\_container.grid\_rowconfigure(0, weight=1)  
 left\_text\_container.grid\_columnconfigure(0, weight=1)  
  
 self.history\_text.config(yscrollcommand=history\_scrollbar.set)  
  
 # Right Frame  
 right\_msg\_frame = tk.Frame(history\_frame, bg="white")  
 right\_msg\_frame.grid(row=0, column=1, sticky="nsew")  
  
 right\_title = tk.Label(right\_msg\_frame, text="Messages from Clients", font=("Arial", 12, "bold"), bg="white")  
 right\_title.pack(side=tk.TOP, anchor="w", padx=5, pady=(5, 0))  
  
 right\_text\_container = tk.Frame(right\_msg\_frame, bg="white")  
 right\_text\_container.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)  
  
 # Right Text + Scrollbar using grid  
 self.client\_msg\_text = tk.Text(right\_text\_container, wrap=tk.WORD, bg="white", state="disabled", height=8)  
 self.client\_msg\_text.grid(row=0, column=0, sticky="nsew")  
  
 msg\_scrollbar = tk.Scrollbar(right\_text\_container, orient=tk.VERTICAL, command=self.client\_msg\_text.yview)  
 msg\_scrollbar.grid(row=0, column=1, sticky="ns")  
  
 right\_text\_container.grid\_rowconfigure(0, weight=1)  
 right\_text\_container.grid\_columnconfigure(0, weight=1)  
  
 self.client\_msg\_text.config(yscrollcommand=msg\_scrollbar.set)  
  
  
 # Client List Section (Moves text down)  
 client\_bottom\_frame = tk.Frame(client\_list\_frame, bg="lightblue")  
 client\_bottom\_frame.pack(side=tk.BOTTOM, fill=tk.X, pady=(10, 5))  
  
 # Create a frame for buttons  
 client\_bottom\_frame = tk.Frame(self.root)  
 client\_bottom\_frame.pack(padx=10, pady=10)  
  
 # Define buttons and their commands in the admin left side grid  
 buttons = [  
 ("Block", self.block\_client),  
 ("Unblock", self.unblock\_client),  
 ("Screenshot", self.screenshot),  
 ("Help", self.help),  
 ("GetGrades", self.GetGrades),  
 ("SetGrade", self.Grade),  
 ("SendFile", self.SendFile),  
 ("Paint", self.\_open\_paint),  
 ("Test", self.Test),  
 ("Test Status", lambda: self.show\_test\_status(refresh=True)),  
 ("Msg", self.Msg),  
 ("Remove Client", self.Remove\_Client)  
 ]  
  
 # Place buttons in a grid  
 for i, (text, command) in enumerate(buttons):  
 btn = tk.Button(client\_bottom\_frame, text=text, command=command, width=12)  
 btn.grid(row=i // 2, column=i % 2, padx=5, pady=5, sticky="nsew")  
  
 # Configure grid to expand properly  
 for i in range(2):   
 client\_bottom\_frame.columnconfigure(i, weight=1)  
 for i in range(3):   
 client\_bottom\_frame.rowconfigure(i, weight=1)  
  
 # The label  
 tk.Label(client\_bottom\_frame, text="Connected Clients:", font=("Arial", 12)).grid(row=6, column=0, columnspan=2, pady=(10, 2))  
  
 # Frame for Listbox & Scrollbar  
 listbox\_frame = tk.Frame(client\_bottom\_frame)  
 listbox\_frame.grid(row=7, column=0, columnspan=2, sticky="nsew", padx=5, pady=(5, 2))  
  
 # Listbox with Scrollbar  
 self.client\_listbox = tk.Listbox(listbox\_frame, font=("Arial", 11), height=9)   
 self.client\_listbox.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)  
  
 scrollbar = tk.Scrollbar(listbox\_frame, orient=tk.VERTICAL, command=self.client\_listbox.yview)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
  
 self.client\_listbox.config(yscrollcommand=scrollbar.set)  
  
   
 # Screenshot Section  
 self.screenshot\_canvas = tk.Canvas(screenshot\_frame, bg="black")  
 self.screenshot\_canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)  
  
  
 self.root.mainloop()  
  
  
 def enable\_screenshot\_scroll(self, width):  
 """Enable scrollbar and mouse wheel scrolling on screenshot canvas."""  
 try:  
 self.screenshot\_canvas.bind("<MouseWheel>", self.\_on\_mousewheel)  
  
 # Check if scrollbar exists, if not, create it  
 if not hasattr(self, 'screenshot\_scrollbar') or not self.screenshot\_scrollbar.winfo\_exists():  
 self.screenshot\_scrollbar = tk.Scrollbar(self.screenshot\_canvas.master, orient=tk.VERTICAL, command=self.screenshot\_canvas.yview)  
 self.screenshot\_scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
 else:  
 if not self.screenshot\_scrollbar.winfo\_ismapped():  
 self.screenshot\_scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
  
 self.screenshot\_canvas.config(yscrollcommand=self.screenshot\_scrollbar.set)  
 self.screenshot\_canvas.config(scrollregion=(0, 0, 1000, width)) # Set a big scrollable area  
 except Exception as e:  
 print(f"An unexpected error occurred: {e}")  
  
 def disable\_screenshot\_scroll(self):  
 """Disable scrollbar and mouse wheel scrolling on screenshot canvas."""  
 try:  
 self.screenshot\_canvas.unbind("<MouseWheel>")  
  
 if hasattr(self, 'screenshot\_scrollbar') and self.screenshot\_scrollbar.winfo\_exists():  
 self.screenshot\_scrollbar.pack\_forget()  
  
 self.screenshot\_canvas.config(yscrollcommand=None)  
  
 self.screenshot\_canvas.yview\_moveto(0)  
 except Exception as e:  
 print(f"An unexpected error occurred: {e}")  
  
  
 def \_start\_client\_msg\_clientlist\_listener(self):  
 print ("\_start\_client\_msg\_clientlist\_listener")  
 def listen\_for\_server\_messages\_clientlist():  
 print("Inside listen\_for\_server\_messages")  
 try:  
 while True:  
 print("trying to get message")  
 message = self.msg\_clientlist\_socket.recv(4096).decode() #.recv(4096).decode("utf-8")  
 print("got message")  
 if not message:  
 break  
  
 if message == "refresh client list, new client connected":  
 # reply for an auto refresh as sent by the server to the admin  
 print("refresh client list, new client connected")  
 self.\_refresh\_client\_list()  
 self.\_log\_history("new client connected")  
 elif message == "refresh client list, admin connected":  
 # reply for an auto refresh as sent by the server to the admin  
 print("refresh client list, admin connected")  
 self.\_refresh\_client\_list()  
 self.\_log\_history("admin connected")  
 elif message.startswith("msg"):  
 # accepting messages from client (through server) and display on right button side.  
 msg = message.split("msg") [1]  
 self.\_log\_client\_message(msg)  
 elif "has been disconnected" in message:  
 self.\_refresh\_client\_list()  
 print("refresh client list, a client\_disconnected")  
 client\_disconnected = message.split(" ")[1]  
 print("client\_disconnected- ", client\_disconnected)  
 self.\_log\_history(f"{client\_disconnected} has been disconnected")  
  
 except Exception as e:  
 print ("admin disconnected forcefully")  
 self.\_log\_history(f"Error in server message listener: {e}")  
 if self.shutdown == False:  
 self.root.destroy()  
  
 thread = threading.Thread(target=listen\_for\_server\_messages\_clientlist)  
 thread.daemon = True  
 thread.start()  
  
   
  
 """  
 paint on a last file defs:  
 """  
  
 # open the painter  
 def \_open\_paint(self):   
 try:  
 if not hasattr(self, "screenshot\_canvas") or not hasattr(self.screenshot\_canvas, "image"):  
 messagebox.showwarning("Paint Error", "No image available to paint on.")  
 return  
  
 # Create a pop-up paint tool window  
 self.paint\_window = tk.Toplevel(self.root)  
 self.paint\_window.title("Paint Tool")  
 self.paint\_window.geometry("350x150")  
 self.paint\_window.configure(bg="#2C3E50")  
  
 # UI elements  
 tk.Label(self.paint\_window, text="Color:", fg="white", bg="#2C3E50", font=("Arial", 12)).grid(row=0, column=0, padx=10, pady=10)  
 self.color\_var = tk.StringVar(value="red")  
 colors = ["red", "blue", "green", "black", "yellow", "purple", "orange"]  
 self.color\_menu = ttk.Combobox(self.paint\_window, textvariable=self.color\_var, values=colors, state="readonly", width=10)  
 self.color\_menu.grid(row=0, column=1, padx=10, pady=10)  
  
 tk.Label(self.paint\_window, text="Brush Size:", fg="white", bg="#2C3E50", font=("Arial", 12)).grid(row=1, column=0, padx=10, pady=10)  
 self.brush\_size = tk.IntVar(value=3)  
 self.brush\_size\_slider = ttk.Scale(self.paint\_window, from\_=1, to=20, variable=self.brush\_size, orient=tk.HORIZONTAL, length=150)  
 self.brush\_size\_slider.grid(row=1, column=1, padx=10, pady=10)  
  
 button\_frame = tk.Frame(self.paint\_window, bg="#2C3E50")  
 button\_frame.grid(row=2, column=0, columnspan=2, pady=10)  
  
 tk.Button(button\_frame, text="Clear", command=self.\_clear\_canvas, bg="#E74C3C", fg="white", font=("Arial", 10, "bold"), width=8).pack(side=tk.LEFT, padx=5)  
 tk.Button(button\_frame, text="Send file", command=self.\_send\_paint, bg="#27AE60", fg="white", font=("Arial", 10, "bold"), width=8).pack(side=tk.LEFT, padx=5)  
  
 self.screenshot\_canvas.bind("<B1-Motion>", self.\_paint)  
  
 except Exception as e:  
 print(f"Error while opening paint tool: {e}")  
 messagebox.showerror("Paint Tool Error", f"An unexpected error occurred: {e}")   
  
  
 # paint  
 def \_paint(self, event):  
 try:  
 x, y = event.x, event.y  
 r = int(self.brush\_size.get())  
 color = self.color\_var.get()  
 self.screenshot\_canvas.create\_oval(x - r, y - r, x + r, y + r, fill=color, outline=color)  
 except Exception as e:  
 print(f"Error while painting: {e}")  
 messagebox.showerror("Paint Error", f"An error occurred while painting: {e}")  
  
 # clearing the canvas  
 def \_clear\_canvas(self):  
 self.screenshot\_canvas.delete("all")  
  
 # sending the updated painted image  
 def \_send\_paint(self):  
 try:  
   
 file\_path = "painted\_image.png"  
  
 self.paint\_window.iconify()  
 self.paint\_window.destroy()  
 time.sleep(0.1)  
 # Get canvas position  
 x = self.screenshot\_canvas.winfo\_rootx()  
 y = self.screenshot\_canvas.winfo\_rooty()  
 width = x + self.screenshot\_canvas.winfo\_width()  
 height = y + self.screenshot\_canvas.winfo\_height()  
  
 # Capture only the canvas area  
 image = ImageGrab.grab(bbox=(x, y, width, height))  
  
 # Save as PNG (optimized)  
 image.save(file\_path, "PNG", optimize=True, compress\_level=9)  
  
 # Send to client  
 client\_ip = self.\_choose\_client()  
 if self.test\_status.get(client\_ip, False):  
 messagebox.showinfo("error", f"{client\_ip} is currently taking a test - cannot send paint to him.")  
 return  
 elif client\_ip:  
 print(client\_ip)  
 print(type(client\_ip))  
 #self.\_send\_image\_to\_server(file\_path, client\_ip)  
 self.\_send\_command("SENDFILE1 painted\_image.png: "+client\_ip)  
   
 except Exception as e:  
 print(f"Error while sending painted image: {e}")  
 messagebox.showerror("Send Paint Error", f"An error occurred while sending the painted image: {e}")   
   
 # defs for different functions:  
   
 def block\_client(self):  
 try:  
 client\_name = self.\_choose\_client()  
 print (client\_name)  
 if client\_name:  
 command = f"BLOCK: {client\_name}"  
 self.\_send\_command(command)   
 except Exception as e:  
 print(f"Error while blocking client: {e}")  
 messagebox.showerror("Block Client Error", f"An error occurred while blocking the client: {e}")  
  
  
 def unblock\_client(self):  
 try:  
 client\_name = self.\_choose\_client()  
 print(client\_name)  
 if client\_name:  
 command = f"UNBLOCK: {client\_name}"  
 self.\_send\_command(command)  
 except Exception as e:  
 print(f"Error while unblocking client: {e}")  
 messagebox.showerror("UnBlock Client Error", f"An error occurred while blocking the client: {e}")  
  
  
 def screenshot(self):  
 try:  
 client\_name = self.\_choose\_client(allow\_all=False)  
 print(client\_name)  
 if client\_name:  
 # Disable the last file button  
 self.last\_file\_button.config(state=tk.DISABLED)  
  
 command = f"SCREENSHOT: {client\_name}"  
 self.\_send\_command(command)  
 except Exception as e:  
 print(f"Error while taking screenshot: {e}")  
 messagebox.showerror("Screenshot Error", f"An error occurred while taking the screenshot: {e}")  
  
  
 def help(self):  
 self.\_send\_command("HELP")  
  
 def GetGrades(self):  
 self.\_send\_command("GETGRADES")  
  
 def Grade(self):  
 try:  
 client\_name = self.\_choose\_client(allow\_all=False)  
 if client\_name:  
 if self.test\_status.get(client\_name, False):  
 messagebox.showinfo("error", f"{client\_name} is currently taking a test - cannot set grade for him.")  
 return  
 root = tk.Tk()  
 root.withdraw() # Hide the root window  
   
 grade = simpledialog.askinteger("Enter the grade", "Enter a grade between 0-100:", minvalue=0, maxvalue=100)  
   
 if grade is not None:  
 self.\_send\_command(f"GRADE {grade}: {client\_name}")  
  
   
 else:  
 messagebox.showinfo("Info", "No grade entered.")  
 except Exception as e:  
 print(f"Error while setting grade: {e}")  
 messagebox.showerror("Grade Error", f"An error occurred while setting the grade: {e}")  
  
  
 def SendFile(self):  
 try:  
 client\_name = self.\_choose\_client()  
 if self.test\_status.get(client\_name, False):  
 messagebox.showinfo("error", f"{client\_name} is currently taking a test - cannot send file to him.")  
 return  
 elif client\_name:  
 print("client\_name ", client\_name)  
 # Open File Explorer to choose a file  
 file\_path = filedialog.askopenfilename(filetypes=[("Text and PNG files", "\*.txt;\*.png")])  
  
 if file\_path:  
 file\_name = os.path.basename(file\_path)  
 command = f"SENDFILE {file\_name}: {client\_name}"  
 self.selected\_file\_info[command] = (file\_path, file\_name) # Save full path for later use  
 print(f"Selected file: {file\_name}")  
 self.\_send\_command(command)  
 except Exception as e:  
 print(f"Error while sending file: {e}")  
 messagebox.showerror("Send File Error", f"An error occurred while sending the file: {e}")  
  
  
  
 def Test(self):  
 try:  
 client\_name = self.\_choose\_client()  
 if client\_name:  
 if client\_name.lower() == "all":  
 for client in self.connected\_clients:  
 client\_check = client.split("(")[0].strip()  
 if self.test\_status.get(client\_check, False):  
 print (f"{client\_check} in test")  
 messagebox.showinfo("error", f"{client\_check} is currently taking a test - cannot set another test for him.")  
 return  
 if self.test\_status.get(client\_name, False):  
 messagebox.showinfo("error", f"{client\_name} is currently taking a test - cannot create a new test for him.")  
 return  
 elif client\_name:  
 while True:  
 file\_path = filedialog.askopenfilename(filetypes=[("Text files", "\*.txt")])  
  
 if not file\_path:  
 return # User cancelled  
  
 file\_name = file\_path.split("/")[-1] # Get file name  
  
 if "TEST" in file\_name.upper():  
 print(f"Selected file: {file\_name}")  
 self.\_send\_command(f"SENDFILE1 {file\_name}: {client\_name}")  
 break  
 else:  
 messagebox.showwarning("Invalid File", "Please select a .txt file that contains the word 'TEST' in its name.")  
 except Exception as e:  
 print(f"Error while assigning test: {e}")  
 messagebox.showerror("Test Error", f"An error occurred while assigning the test: {e}")  
   
  
 def show\_test\_status(self, refresh=False):  
 try:  
 if not hasattr(self, "screenshot\_canvas"):  
 return  
  
 if refresh:  
 print("refresh")  
 self.\_refresh\_client\_list()  
 self.disable\_screenshot\_scroll()  
  
 self.screenshot\_canvas.delete("all")  
 y = 10  
 self.screenshot\_canvas.create\_text(  
 10, y, text="--- Test Status ---", font=("Arial", 28), fill="white", anchor="nw")  
 y += 40  
  
 for client, status in self.test\_status.items():  
 color = "green" if status else "red"  
 status\_text = "In Test" if status else "Not in Test"  
 self.screenshot\_canvas.create\_text(  
 10, y, text=f"{client}: {status\_text}", font=("Arial", 22), fill=color, anchor="nw")  
 y += 35  
  
 self.screenshot\_canvas.create\_text(  
 10, y, text="---------------------", font=("Arial", 28), fill="white", anchor="nw")  
 except Exception as e:  
 print(f"Error while showing test status: {e}")  
 messagebox.showerror("Test Status Error", f"An error occurred while showing the test status: {e}")  
  
  
 def Msg(self):  
 try:  
 self.command\_entry.config(state="normal")  
 self.command\_entry.delete(0, tk.END)  
 self.msg\_mode = True  
 self.command\_entry.focus\_set()  
 except Exception as e:  
 print(f"Error while enabling message mode: {e}")  
 messagebox.showerror("Message Mode Error", f"An error occurred while enabling message mode: {e}")  
   
  
 def Remove\_Client(self):  
 try:  
 client\_name = self.\_choose\_client(allow\_all=False)   
 if self.test\_status.get(client\_name, False):  
 messagebox.showinfo("error", f"{client\_name} is currently taking a test - cannot remove him.")  
 return  
 print (client\_name)  
 if client\_name:  
 command = f"REMOVE: {client\_name}"  
 self.\_send\_command(command)  
 except Exception as e:  
 print(f"Error while removing client: {e}")  
 messagebox.showerror("Remove Client Error", f"An error occurred while removing the client: {e}")  
   
   
 # choose client for functions  
   
 def \_choose\_client(self, allow\_all=True):  
 try:  
 if not self.connected\_clients:  
 self.\_refresh\_client\_list()  
 if not self.connected\_clients:  
 messagebox.showwarning("Client Selection", "No connected clients available.")  
 return None  
  
 # Only add "ALL" if allowed  
 client\_options = ["ALL"] + self.connected\_clients if allow\_all else self.connected\_clients  
  
 client\_selection = tk.Toplevel(self.root)  
 client\_selection.title("Select Client")  
 client\_selection.geometry("250x120")  
  
 selected\_client = tk.StringVar()  
 user\_confirmed = tk.BooleanVar(value=False) # Track if user confirmed  
  
 client\_dropdown = ttk.Combobox(  
 client\_selection, textvariable=selected\_client,  
 values=client\_options, state="readonly"  
 )  
 client\_dropdown.pack(pady=10)  
 client\_dropdown.current(0)  
  
 def confirm\_selection():  
 user\_confirmed.set(True)  
 client\_selection.destroy()  
  
 tk.Button(client\_selection, text="Select", command=confirm\_selection).pack(pady=5)  
  
 client\_selection.protocol("WM\_DELETE\_WINDOW", client\_selection.destroy) # Handle 'X' close  
 client\_selection.wait\_window()  
  
 if not user\_confirmed.get():  
 return None # User closed the window with 'X'  
  
 selected\_client\_name = selected\_client.get()  
 if selected\_client\_name and selected\_client\_name != "ALL":  
 selected\_client\_name = selected\_client\_name.split("(")[0].strip()  
  
 print(selected\_client\_name)  
 return selected\_client\_name  
 except Exception as e:  
 print(f"Error while choosing client: {e}")  
 messagebox.showerror("Choose Client Error", f"An error occurred while choosing the client: {e}")  
 return None  
  
  
   
 def handle\_server\_reply(self, command, client\_name=None):  
 # handling reply from server  
 try:  
 if client\_name.lower() == "all":  
 a = self.connected\_clients  
 print ("sendall in handle server reply")  
 else:  
 a= [client\_name]  
 for client in a:  
 print(f"Handling reply for client {client\_name}\n")  
 try:  
 response = self.socket.recv(1024).decode()  
 except Exception as e:  
 print(f"Error getting response: {e}")  
 self.\_log\_history(f"Error getting response: {e}")  
  
   
 if response:   
 if response.startswith("SCREENSHOT SAVED AS"):  
 # Enable the last file button  
 self.last\_file\_button.config(state=tk.NORMAL)  
 elif "sent MSG you got a new grade" in response:  
 self.\_send\_command("GETGRADES")  
   
 self.\_log\_history(f"Client {client\_name} -> Command: {command}\nResponse: {response}\n")  
  
 if "is answering a test" in response:  
 c\_name = response.split ("is") [0].strip()  
 self.test\_status[c\_name] = True  
 self.history\_text.insert(tk.END, f"[INFO] Test status for {c\_name}: Started\n")  
 print("self.test\_status", self.test\_status)  
 self.show\_test\_status()  
 # create a new socket on another port  
 print(f"{client\_name} is answering a test... waiting up to 60 seconds for results")  
 threading.Thread(target=self.handle\_test\_answer, args=(), daemon=True).start()   
   
 except Exception as e:  
 print(f"Error getting reply from server: {e}")  
  
   
 def handle\_test\_answer(self):  
 # getting response from server of client's Test answers.  
 try:  
 second\_response = self.test\_socket.recv(1024).decode()  
 if "TEST\_ANSWER" in second\_response:  
 client\_name = second\_response.split(":")[1].strip()  
 self.test\_status[client\_name] = False  
 self.history\_text.insert(tk.END, f"[INFO] Test status for {client\_name}: Completed\n")  
 print("self.test\_status", self.test\_status)  
 self.show\_test\_status()  
 self.history\_text.insert(tk.END, f"[INFO] Test status for {client\_name}: Completed\n")  
 print(f"Received test answer from {client\_name}: {second\_response}")  
 SetGrade= second\_response.replace("TEST\_ANSWER", "GRADE")  
 self.\_send\_command(f"{SetGrade}")  
 self.\_log\_history(f"Client {client\_name} -> Test Answer: {second\_response}\n")  
 except Exception as e:  
 print(f"Error handling test answer: {e}")  
  
 def \_get\_command(self):  
 # getting command from the command and sending it to client\s.  
 try:  
 if self.msg\_mode:  
 message = self.command\_entry.get().strip()  
 if not message:  
 messagebox.showwarning("Empty Message", "Please type a message before sending.")  
 return  
   
 client\_name = self.\_choose\_client()  
 if not client\_name:  
 return # User cancelled  
  
 self.\_send\_command(f"MSG {message}: {client\_name}")  
 self.command\_entry.delete(0, tk.END)  
 self.command\_entry.config(state="disabled")  
 self.msg\_mode = False  
 else:  
 # Normal command behavior  
 command = self.command\_entry.get().strip()  
 if command:  
 self.\_send\_command(command)  
 self.command\_entry.delete(0, tk.END)  
 except Exception as e:  
 print(f"Error in \_get\_command: {e}")  
 self.\_log\_history(f"Command entry error: {e}")  
   
 def \_send\_command(self, command):  
 #this def is sending commands to the server.  
   
 try:  
 # Ensure socket is open and connected  
 if not self.socket or self.socket.\_closed:  
 messagebox.showerror("Error", "No active connection. Please connect to the server.")  
 return  
  
 # Handle Help locally without sending to server  
 elif command == "HELP":  
 print ("in help, ", command)  
 content = self.\_read\_help()  
 self.display\_help()  
 self.\_log\_history(content)  
 return  
  
 # Request client list  
 elif command == "CLIENTLIST":  
 self.\_refresh\_client\_list()  
  
 # Request grade list from server  
 elif command == "GETGRADES":  
 self.socket.sendall(command.encode())  
 response = self.socket.recv(4096).decode()  
 print(response)  
 if response != "Database error occurred. Please try again later.":  
 response = "current grades:\n" + response  
 # Estimate the height needed based on number of lines  
 num\_lines = response.count('\n') + 1  
 print ("num\_lines- ",num\_lines)  
 line\_height = 30 # Approximate pixel height per line with font size 20  
 total\_height = num\_lines \* line\_height + 200 # Some extra padding  
  
 self.enable\_screenshot\_scroll(total\_height)  
  
 self.screenshot\_canvas.delete("all")  
 self.screenshot\_canvas.create\_text(10, 10, text=response, font=("Arial", 20), fill="white", anchor="nw")  
  
 else:  
 # Handle file transfer commands  
 if "SENDFILE" in command:  
 if "SENDFILE1" in command: #handles sendfile from the same directory of the project  
 print ("command ", command)  
 file\_name = command.split(':')[0].replace('SENDFILE1','').strip()  
 ip = command.split(':')[1] # need to also check for errors  
 with open(file\_name, "rb") as f:  
 data = f.read()  
 else: #handles sendfile from other directorys  
 print("command ", command)  
 file\_info = self.selected\_file\_info.get(command)  
 if not file\_info:  
 self.\_log\_history("Error: File path info missing.")  
 return  
  
 file\_path, file\_name = file\_info  
 ip = command.split(':')[1].strip()  
 with open(file\_path, "rb") as f:  
 data = f.read()  
  
 checksum = hashlib.md5(data).hexdigest()  
 print(f"Sending file of size {len(data)} bytes with checksum {checksum}")  
  
 # Send file protocol commands  
 self.socket.sendall(command.encode())  
 self.socket.sendall(b"DATA\_NAME") # fix: take the daniel.txt from the command  
 self.socket.sendall(file\_name.encode())  
 self.socket.sendall(b"DATA\_START")   
 self.socket.sendall(data)  
 self.socket.sendall(b"DATA\_END")  
  
 print(f"Sent file {file\_name} to server.")  
   
   
 else: # including SCREENSHOT: supports BLOCK: ip, UNBLOCK: ip, MSG, GRADE, Remove, xxxx: IP  
 # add IMG xxxx: IP  
 print ("sending command - ",command.encode())  
 self.socket.sendall(command.encode())  
  
 client\_target = command.split(":")[1].strip()  
   
 threading.Thread(target=self.handle\_server\_reply, args=(command, client\_target), daemon=True).start()  
  
  
   
 except Exception as e:  
 self.\_log\_history(f"Error sending command: {e}\n")  
   
  
 def \_log\_history(self, message):  
 # displays log history information in the bottom-left corner of the screen  
 try:  
 self.history\_text.configure(state="normal")  
 self.history\_text.insert(tk.END, message + "\n")  
 self.history\_text.configure(state="disabled")  
 self.history\_text.see(tk.END)  
 except Exception as e:  
 if self.shutdown == False:  
 print(f"Error logging history: {e}")  
   
  
 def \_log\_client\_message(self, message):  
 # displays log messages from clients in the bottom-right corner of the screen  
 try:  
 self.client\_msg\_text.configure(state="normal")  
 self.client\_msg\_text.insert(tk.END, message + "\n")  
 self.client\_msg\_text.configure(state="disabled")  
 self.client\_msg\_text.see(tk.END)  
 except Exception as e:  
 if self.shutdown == False:  
 print(f"Error logging client's messages: {e}")  
  
 def \_refresh\_client\_list(self):  
 # refrsh the client list by requasting from the server, and display it in a listbox at the left side  
  
 if not self.socket:  
 self.\_log\_history("Cannot refresh client list: Not connected to server.")  
 return  
  
 try:  
  
 # Send the CLIENTLIST request  
 self.socket.sendall(b"CLIENTLIST")  
  
 try:  
 # Try to receive the response from the server  
 response = self.socket.recv(1024).decode()  
 print ("response of clientlist\*\*",response,"&&")  
   
 # Check if there are no clients connected  
 if not response:  
 self.\_log\_history("No clients connected to the server.")  
 self.client\_listbox.delete(0, tk.END)  
 return  
  
 self.client\_listbox.delete(0, tk.END)  
 self.connected\_clients = response.split(", ") if response != "empty" else []  
  
 # Remove clients from test\_status if they're not in connected\_clients  
 for client in list(self.test\_status.keys()):  
 if client not in [c.split("(")[0].strip() for c in self.connected\_clients]:  
 del self.test\_status[client]  
  
   
 if self.connected\_clients == []:  
 self.\_log\_history("No clients connected.")  
 return  
  
  
 for client in self.connected\_clients:  
 self.client\_listbox.insert(tk.END, client)  
 c = client.split("(")[0].strip()  
 if c not in self.test\_status:  
 self.test\_status[c] = False  
  
 client\_list = ", ".join(self.connected\_clients)  
 self.\_log\_history(f"Connected Clients: {client\_list}\n")  
  
  
 except:  
 self.\_log\_history("Error: No clients are connected.")  
  
 except Exception as e:  
 self.\_log\_history(f"Error refreshing client list: {e}\n")  
  
  
  
  
 def \_request\_last\_file(self):  
 # requasting last file for specific client from the server  
   
 print ("in request last file")  
 if not self.socket:  
 messagebox.showwarning("Connection Error", "You must connect to the server first.")  
 return  
  
 try:  
 self.disable\_screenshot\_scroll()  
 client\_name = self.\_choose\_client(allow\_all=False)  
 if client\_name:  
 self.socket.sendall(f"LASTFILE - {client\_name}".encode())  
 chunk = self.socket.recv(1024)  
 if chunk.decode().startswith("Error: No files found for client"):  
 print ("no last file from client - {client\_name}")  
 self.\_log\_history(f"no last file from client - {client\_name}")  
 return  
 elif "Error: No files found on server" not in chunk.decode():  
 # Start collecting file data  
 data = chunk.replace(b"FILE\_START", b"")  
   
 while True:  
 chunk = self.socket.recv(4096)  
 if b"FILE\_END" in chunk:  
 data += chunk.replace(b"FILE\_END", b"")  
 break  
 data += chunk  
  
 checksum = hashlib.md5(data).hexdigest()  
 print(f"Received file of size {len(data)} bytes with checksum {checksum}")  
  
 # Validate if the received data is not empty  
 if not data:  
 self.\_log\_history("Received empty data for the most recent file.")  
 return  
  
 # Attempt to open the image  
 try:  
 # Open and resize image for canvas  
 image = Image.open(BytesIO(data))  
 print ("before calc canvas\_size")  
 canvas\_width = self.screenshot\_canvas.winfo\_width()  
 canvas\_height = self.screenshot\_canvas.winfo\_height()  
 image = image.resize((canvas\_width, canvas\_height), Image.Resampling.LANCZOS)  
  
 photo = ImageTk.PhotoImage(image)  
 self.screenshot\_canvas.create\_image(0, 0, image=photo, anchor=tk.NW)  
 self.screenshot\_canvas.image = photo  
 self.\_log\_history(f"Most recent file received and displayed from client- {client\_name}.")  
 except Exception as e:  
 self.\_log\_history(f"Error displaying the image: {e}")  
 else:  
 print("there is no last file")  
 self.\_log\_history(f"there is no last file")  
  
 except Exception as e:  
 self.\_log\_history(f"Error receiving the most recent file: {e}")  
  
  
 def \_on\_mousewheel(self, event):  
 # This method allows the admin to scroll through the canvas display area  
 self.screenshot\_canvas.yview\_scroll(int(-1\*(event.delta/120)), "units")  
  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 try:  
 admin\_ip = "192.168.1.26"   
 admin\_port = 5000  
 admin\_test\_port = 5002  
 client\_msg\_clientlist\_port = 5003  
 AdminClient(admin\_ip, admin\_port, admin\_test\_port, client\_msg\_clientlist\_port)  
 except Exception as e:  
 print(f"Fatal error starting AdminClient: {e}")

## server.py

# server code  
  
import socket  
import hashlib  
import threading  
from datetime import datetime  
import os  
import select  
import ssl  
import re  
from sqlalchemy.orm import sessionmaker, DeclarativeBase  
from sqlalchemy import create\_engine, Column, Integer, String, DateTime  
from sqlalchemy.ext.declarative import declarative\_base  
  
  
# --- Database Setup ---  
# Set up a SQLite database to log student test submissions  
engine = create\_engine("sqlite:///students.db", echo=True)  
class Base(DeclarativeBase):  
 pass #Base = declarative\_base()  
  
# Table to store grades  
class Student(Base):  
 \_\_tablename\_\_ = 'students'  
 id = Column(Integer, primary\_key=True, autoincrement=True, unique=True)  
 name = Column(String(15), nullable=False)  
 date\_time = Column(DateTime, default=datetime.utcnow)  
 grade = Column(Integer, nullable=False)  
  
  
# Initialize DB session  
def init\_db():  
 # Create the table  
 #Base.metadata.create\_all(engine)  
  
 # Create a session  
 Session = sessionmaker(bind=engine)  
 session = Session()  
 return session  
  
  
# --- Server Class ---  
class Server:   
 def \_\_init\_\_(self, server\_ip, admin\_port, admin\_port\_test, client\_port, client\_msg\_clientlist\_port):  
 self.start\_time = datetime.now()  
  
 # Configuration  
 self.server\_ip = server\_ip  
 self.admin\_port = admin\_port  
 self.admin\_port\_test = admin\_port\_test  
 self.client\_msg\_clientlist\_port = client\_msg\_clientlist\_port  
 self.client\_port = client\_port  
 print ("sock with admin ", self.admin\_port)  
 print ("test sock with admin ", self.admin\_port\_test)  
 print ("msg / clientlist - client sock with admin ", client\_msg\_clientlist\_port)  
  
 # SSL context for secure communication  
 self.ssl\_context = ssl.SSLContext(ssl.PROTOCOL\_TLS\_SERVER)  
 self.ssl\_context.load\_cert\_chain(certfile="server.pem", keyfile="server.key")  
  
 # Initialize server sockets  
 self.admin\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)   
 self.admin\_test\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)   
 self.client\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.client\_msg\_clientlist\_sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
  
 self.admin\_conn = None  
 self.Admin\_Connected = False  
 self.client\_connections = {} # socket -> {"ip": ..., "name": ...}  
 self.client\_ips = []  
 self.client\_ip\_name\_mapping = {} # {IP: [names]}  
 self.client\_name\_ip\_mapping = {} # {name: IP}  
  
 try:  
 # Bind sockets to ports  
 self.admin\_sock.bind((self.server\_ip, self.admin\_port))  
 self.admin\_test\_sock.bind((self.server\_ip, self.admin\_port\_test))  
 self.client\_sock.bind((self.server\_ip, self.client\_port))  
 self.client\_msg\_clientlist\_sock.bind((self.server\_ip, self.client\_msg\_clientlist\_port))  
  
 # Listen for incoming connections  
 self.admin\_sock.listen(1)  
 self.admin\_test\_sock.listen(1)  
 self.client\_sock.listen(5)  
 self.client\_msg\_clientlist\_sock.listen(1)  
 except Exception as e:  
 print(f"[ERROR] Failed to bind server sockets: {e}")  
  
 print(f"Server is listening for admin on port {self.admin\_port}")  
 print(f"Server is listening for clients on port {self.client\_port}")  
  
 def accept\_admin\_test\_connection(self):  
 try:  
 print ("attempting test connection with admin\n")  
 self.admin\_conn\_test, \_ = self.admin\_test\_sock.accept()   
 print("Admin test connected ", self.admin\_conn\_test)  
 except Exception as e:  
 print(f"[ERROR] Admin test connection failed: {e}")  
  
 def accept\_admin\_msg\_clientlist\_connection(self):  
 try:  
 print ("attempting msg / clientlist connection with admin\n")  
 self.admin\_conn\_msg\_clientlist, \_ = self.client\_msg\_clientlist\_sock.accept()   
 print("Admin msg / clientlist connected ", self.admin\_conn\_msg\_clientlist)  
 except Exception as e:  
 print(f"[ERROR] Admin msg/clientlist connection failed: {e}")  
  
 def start(self):  
 try:  
 # Accept non-blocking auxiliary admin connections  
 threading.Thread(target=self.accept\_admin\_test\_connection, daemon=True).start()  
 threading.Thread(target=self.accept\_admin\_msg\_clientlist\_connection, daemon=True).start()  
 threading.Thread(target=self.handle\_client\_connections, daemon=True).start()  
  
 print("Waiting for admin connection...")   
 admin\_conn, \_ = self.admin\_sock.accept()  
 print("Admin connected ", admin\_conn)  
   
   
 if (admin\_conn is None):  
 print("Error: Admin connection failed.")  
 return  
  
 self.admin\_conn = self.ssl\_context.wrap\_socket(admin\_conn, server\_side=True)  
 print(f"Admin connection fileno after SSL: {self.admin\_conn.fileno()}") # Debugging line  
 print("Admin client connected successfully with SSL.")  
  
 self.Admin\_Connected = True  
 self.main\_server\_loop()  
  
 except Exception as e:  
 print(f"Error in start(): {e}")  
 raise  
  
 def handle\_client\_connections(self):  
 while True:  
 try:  
 conn\_decrypt, addr = self.client\_sock.accept()  
 print("client connection received")  
 conn = self.ssl\_context.wrap\_socket(conn\_decrypt, server\_side=True)  
  
 client\_ip = addr[0]  
  
 # Receive client's name (first message after connection)  
 client\_name = conn.recv(1024).decode().strip()  
 print(f"[SERVER] Received client name: {client\_name}")  
  
 # Check if the client name is already connected  
 if client\_name in self.client\_name\_ip\_mapping and self.client\_name\_ip\_mapping[client\_name] != conn:  
 conn.sendall("error- Client name already in use".encode())  
 conn.close()  
 print(f"Rejected duplicate client name: {client\_name}")  
 continue  
 else:  
 conn.sendall("OK".encode())  
 print("refresh client list, new client connected")  
 msg = "refresh client list, new client connected"  
 try:  
 self.admin\_conn\_msg\_clientlist.sendall(msg.encode())  
 except Exception as e:  
 print(f"Failed to send message to admin: {e}")  
  
  
  
 # Store the client connection with IP and name  
 self.client\_connections[conn] = {"ip": client\_ip, "name": client\_name}  
 print("init", self.client\_connections.keys())  
  
 self.client\_ips.append(client\_ip)  
 if client\_ip not in self.client\_ip\_name\_mapping:  
 self.client\_ip\_name\_mapping[client\_ip] = [] # Create a list if IP is new  
  
 self.client\_ip\_name\_mapping[client\_ip].append(client\_name) # Store name under IP  
 self.client\_name\_ip\_mapping[client\_name] = conn # Store name-to-socket mapping  
  
 print("Updated IP-Name mapping:", self.client\_ip\_name\_mapping)  
 print("Updated Name-IP mapping:", self.client\_name\_ip\_mapping)  
 print(f"New client connected from {client\_ip} with name: {client\_name}")  
  
 except Exception as e:  
 print(f"Error handling client connections: {e}")  
 break  
  
   
 def handle\_client\_screenshot(self, client\_conn, data, client\_info):  
 """Receive and save a screenshot from a client with their name in the filename."""  
 print("received client screenshot")  
 data = data.replace(b"PIC\_START", b"")  
  
 try:  
 while True:  
 chunk = client\_conn.recv(4096)  
 print("received chunk", len(chunk))  
 if b"PIC\_END" in chunk:  
 data += chunk.replace(b"PIC\_END", b"")  
 break  
 data += chunk  
  
 # Format filename using client name and current timestamp  
 current\_time = datetime.now().strftime('%Y-%m-%d\_%H-%M-%S')  
  
 client\_name = client\_info['name'].upper().replace(" ", "\_")  
 filename = f"pic\_{client\_name}\_{current\_time}.png"  
  
 with open(filename, "wb") as f:  
 f.write(data)  
  
 print(f"Screenshot saved as {filename}")  
 self.admin\_conn.sendall(f"SCREENSHOT SAVED AS {filename}".encode())  
  
 except Exception as e:  
 print(f"[ERROR] Error saving screenshot: {e}")  
 try:  
 self.admin\_conn.sendall(b"Error: SCREENSHOT SAVING FAILED")  
 except:  
 print("[ERROR] Failed to notify admin about screenshot error.")  
  
   
 def send\_last\_file(self, client\_name):  
 """Send the most recent file to the admin."""  
 try:  
 files = [f for f in os.listdir('.') if f.startswith('pic\_') and f.endswith('.png')]  
 if not files:  
 self.admin\_conn.sendall(b"Error: No files found on server.")  
 print("there is no last file")  
 return  
 client\_files = {}  
 for f in files:  
 match = re.match(r'pic\_(\w+)\_(\d{4}-\d{2}-\d{2}\_\d{2}-\d{2}-\d{2})\.png', f)  
 if match:  
 name = match.group(1).upper()  
 time\_str = match.group(2)  
 file\_time = datetime.strptime(time\_str, "%Y-%m-%d\_%H-%M-%S")  
 print ("file\_time- ", file\_time)  
 if file\_time >= self.start\_time:  
 if name not in client\_files:  
 client\_files[name] = []  
 client\_files[name].append((file\_time, f))  
 print ("client\_files-", client\_files)  
 client\_name = client\_name.upper()  
 if client\_name not in client\_files:  
 self.admin\_conn.sendall(f"Error: No files found for client '{client\_name}'.".encode())  
 print(f"No files found for client: {client\_name}")  
 return  
  
 # Find the latest file  
 last\_file = max(client\_files[client\_name], key=lambda x: x[0])[1]  
  
 print("Sending the most recent screenshot file to the admin.")  
 with open(last\_file, "rb") as f:  
 data = f.read()  
  
 checksum = hashlib.md5(data).hexdigest()  
 print(f"Sending file of size {len(data)} bytes with checksum {checksum}")  
  
 self.admin\_conn.sendall(b"FILE\_START")  
 self.admin\_conn.sendall(data)  
 self.admin\_conn.sendall(b"FILE\_END")  
  
 print(f"Sent file {last\_file} to admin.")  
  
 except Exception as e:  
 print(f"[ERROR] Error sending last file: {e}")  
 try:  
 self.admin\_conn.sendall(b"Error: Failed to send the last file.")  
 except:  
 print("[ERROR] Failed to notify admin about file send failure.")  
  
 def is\_socket\_open(conn):  
 """  
 Check whether a socket connection is open.  
 """  
 try:  
 # Peek at incoming data without removing it from the buffer  
 return bool(conn.recv(1, socket.MSG\_PEEK))  
 except (socket.error, ssl.SSLError):  
 return False  
  
 def get\_target\_ip(self, client\_name\_or\_ip):  
 """  
 Return a list of sockets matching the given client name or IP.  
 - If "all", return all client sockets.  
 - If name, return socket(s) by name.  
 - If IP, return socket(s) by IP.  
 """  
 # Check if the target is "ALL"   
 if client\_name\_or\_ip.lower() == "all":  
 return list(self.client\_connections.keys()) # Return all connected client sockets  
   
 # Normalize name dictionary for case-insensitive lookups  
 normalized\_name\_ip\_mapping = {name.lower(): ip for name, ip in self.client\_name\_ip\_mapping.items()}   
 client\_name\_or\_ip = client\_name\_or\_ip.lower() # Normalize input for case-insensitive matching  
  
 # Direct lookup by name  
 if client\_name\_or\_ip in normalized\_name\_ip\_mapping:  
 return [normalized\_name\_ip\_mapping[client\_name\_or\_ip]] # Return as list for consistency  
   
 # Direct lookup by IP  
 elif client\_name\_or\_ip in self.client\_ip\_name\_mapping:  
 return [conn for conn, data in self.client\_connections.items() if data["ip"] == client\_name\_or\_ip]  
   
 return None # Not found  
  
 def main\_server\_loop(self):  
 if self.Admin\_Connected:  
 print("refresh client list, admin connected")  
 msg = "refresh client list, admin connected"  
 try:  
 self.admin\_conn\_msg\_clientlist.sendall(msg.encode())  
 except:  
 print("cannot refresh clientlist")  
 self.Admin\_Connected = False  
   
 while True:  
 try:  
  
 # Using select to wait for readable sockets with a timeout of 1 second  
 readable, \_, \_ = select.select([self.admin\_conn] + list(self.client\_connections.keys()), [], [], 1)  
  
 if self.admin\_conn in readable:  
 try:  
 message = self.admin\_conn.recv(1024)  
 command = message.decode('latin-1', errors='ignore')  
 print("in server read message")  
 print(command)  
 except:  
 message = ""  
  
 if not message or (message == "bye"):  
 print("Admin disconnected")  
 break  
  
 # Command handling  
 if command == "CLIENTLIST":  
 client\_list = ", ".join([f"{data['name']} ({data['ip']})" for data in self.client\_connections.values()])  
 print("Current client:", client\_list) # Print list  
 print("Current client list:", self.client\_ip\_name\_mapping) # Print the dictionary  
 if client\_list == "": client\_list = "empty"  
 try:  
 self.admin\_conn.sendall(client\_list.encode())  
 except Exception as e:  
 print(f"Error sending client list to admin: {e}")  
 self.admin\_conn.sendall(b"Error sending client list.")  
  
 elif command == "GETGRADES":  
 try:  
 students = session.query(Student).all()  
 if students:  
 reply = "\n".join([f"ID: {s.id}, Name: {s.name}, Grade: {s.grade}" for s in students])  
 self.admin\_conn.sendall(reply.encode())  
 else:  
 self.admin\_conn.sendall(b"No grades data available.")  
 except Exception as e:  
 print(f"Database error in GETGRADES: {e}")  
 self.admin\_conn.sendall(b"Database error occurred. Please try again later.")  
  
 elif command.startswith ("LASTFILE"):  
 client\_name = command.split("-")[1].strip()  
 self.send\_last\_file(client\_name)  
  
 else: # Handling commands like SENDFILE, SCREENSHOT, BLOCK, UNBLOCK, MSGxxx, GRADExxx, REMOVE  
 print("in here")  
 print(message)  
  
 cmd = command.split(":")[0].strip() # Extract command  
 command\_parts = command.split(":")  
 if len(command\_parts) > 1:  
 command\_target = command\_parts[1].strip().split("DATA\_NAME")[0] # Extract target (name or IP)  
 else:  
 print(f"Invalid command format: {command}") # Debug message  
 self.admin\_conn.sendall(b"Error: Invalid command format")  
 continue # Skip processing this command  
 print("command ", cmd)  
 print("command\_target", command\_target)  
  
 # Convert name or IP to a list of target connections  
 target\_conn\_list = self.get\_target\_ip(command\_target)  
  
 if not target\_conn\_list:  
 print(f"Error: Target '{command\_target}' not found.")  
 self.admin\_conn.sendall(f"Error: Target '{command\_target}' not found.".encode())  
 continue # Skip processing if the target is invalid  
  
 # Ensure target\_conn\_list is always a list for iteration  
 if not isinstance(target\_conn\_list, list):  
 target\_conn\_list = [target\_conn\_list]  
  
 # Handle data extraction if it's a file command  
 command\_data = message.split(b"DATA\_NAME")  
 data = b"DATA\_NAME" + command\_data[1] if b'DATA\_NAME' in message else b''  
  
 # If "ALL" is specified, send to all clients  
 if command\_target.lower() == "all" or command\_target == "\*":  
 print("Sending command to all clients")  
 target\_conn\_list = list(self.client\_connections.keys())  
  
  
 # Sending command to selected clients  
 print("before loop ", target\_conn\_list)  
 reply = ""  
 print ("target\_conn\_list", target\_conn\_list)  
 for target in target\_conn\_list:  
 print("in targets")  
 if target:  
 print("sending command to client ", target, ":", cmd)  
 if "SENDFILE" in cmd:   
 file\_name= cmd.split(':')[0].replace('SENDFILE','').strip()  
 print (file\_name)  
 print("server in SENDFILE")  
 while b"DATA\_END" not in data:  
 chunk = self.admin\_conn.recv(4096)  
 data += chunk  
  
 print("sending to client target ", data)  
 target.sendall(data)  
 else:  
 if "GRADE" in cmd:  
 print("GRADE in command")  
 gr = cmd.replace("GRADE", "").strip()  
 print(gr)  
 client\_name\_or\_ip = command.split(":")[1].strip()  
 print(client\_name\_or\_ip)  
  
 # Check if the provided identifier is an IP (contains ".")  
 if "." in client\_name\_or\_ip:  
 if client\_name\_or\_ip in self.client\_ip\_name\_mapping and self.client\_ip\_name\_mapping[client\_name\_or\_ip]:  
 client\_name = self.client\_ip\_name\_mapping[client\_name\_or\_ip][0]  
 else:  
 error\_msg = f"Error: No name found for IP {client\_name\_or\_ip}"  
 print(error\_msg)  
 self.admin\_conn.sendall(error\_msg.encode())  
 return # skip  
 else:  
 client\_name = client\_name\_or\_ip  
  
 client\_name = client\_name.upper()  
  
 # Insert into the database  
 new\_student = Student(name=client\_name, grade=gr)  
 session.add(new\_student)  
 session.commit()  
  
 cmd = re.sub(r"GRADE (\d+)", r"MSG you got a new grade - \1", cmd)  
 print("Got a new grade - must update DB", cmd)  
  
 print(f"New entry added to the database: {client\_name} - {gr}")  
   
 print("sending to client target - simple msg, ", command)  
 target.sendall(cmd.encode())  
  
 if cmd != "SCREENSHOT":  
 if not("SENDFILE" in cmd and "test" in file\_name):  
 print("sending automatic reply to client")  
 reply += f"sent {cmd} to: {self.client\_connections[target]['name']} ({self.client\_connections[target]['ip']})\n"  
 else:  
 reply += f"Error: Client with target '{command\_target}' not found\n"  
 if reply:  
 print("server sending reply to admin, ", reply)  
 self.admin\_conn.sendall(reply.encode())  
 print("server sent msg to admin, end")  
  
 # Receiving messages from clients  
 for conn in list(self.client\_connections.keys()):  
 if conn in readable:  
 try:  
 print ("waiting for data from client")  
 data = conn.recv(1024)  
 print("received data from client")  
 msg = data.decode('latin-1', errors='ignore')  
   
 if msg.startswith ("msg"): # message from client to admin.  
 print("message from client to admin")  
 self.admin\_conn\_msg\_clientlist.sendall(msg.encode())  
 elif msg.startswith("PIC\_START"): #client screenshot.  
 print("in PIC\_START")  
 self.handle\_client\_screenshot(conn, data, self.client\_connections[conn])  
 elif "is answering a test" in msg: # confirmation that the client is starting a test  
 print ("sending answering a test")  
 self.admin\_conn.sendall(msg.encode())  
 elif msg.startswith("TEST\_ANSWER"): # sending the client's test grade to the admin  
 print ("sending test answers over second socket")  
 # sending reply with the second socket (test)  
 self.admin\_conn\_test.sendall(msg.encode())  
 elif msg.startswith("shutting down"): #client shutting down  
 print("message from client to admin - shutting down")  
  
 """  
 removing the client from the lists and data bases, clientlist....  
 """  
   
 # Extract the client name from the message  
 client\_name\_to\_remove = msg.split("-")[1].strip()  
  
 # Check if the client name exists in the mappings  
 if client\_name\_to\_remove in self.client\_name\_ip\_mapping:  
 # Get the connection object associated with the client  
 conn\_to\_remove = self.client\_name\_ip\_mapping[client\_name\_to\_remove]  
  
 # Get the client IP from the connection info  
 client\_ip\_to\_remove = self.client\_connections[conn\_to\_remove]["ip"]  
  
 # Remove the client from all mappings  
 del self.client\_name\_ip\_mapping[client\_name\_to\_remove]  
 del self.client\_connections[conn\_to\_remove]  
  
 if client\_ip\_to\_remove in self.client\_ips:  
 self.client\_ips.remove(client\_ip\_to\_remove)  
  
 if client\_ip\_to\_remove in self.client\_ip\_name\_mapping:  
 self.client\_ip\_name\_mapping[client\_ip\_to\_remove].remove(client\_name\_to\_remove)  
  
 if not self.client\_ip\_name\_mapping[client\_ip\_to\_remove]:  
 del self.client\_ip\_name\_mapping[client\_ip\_to\_remove]  
  
 # Close the connection to the client  
 conn\_to\_remove.close()  
 print(f"Client {client\_name\_to\_remove} removed and connection closed.")  
  
 # Notify the admin that the client disconnected  
 self.admin\_conn\_msg\_clientlist.sendall(f"Client {client\_name\_to\_remove} has been disconnected.".encode())  
  
 else:  
 print(f"Error: Client {client\_name\_to\_remove} not found in mappings.")  
  
  
   
 else:   
 print(f"{self.client\_connections[conn]['name']} disconnected")  
 self.client\_ips.remove(self.client\_connections[conn]["ip"])  
 del self.client\_connections[conn]  
 conn.close()  
 continue  
  
 except (ConnectionResetError, socket.error) as e:  
 print(f"Client Connection error with {self.client\_connections[conn]['name']}: {e}")  
 self.client\_ips.remove(self.client\_connections[conn]["ip"])  
 del self.client\_connections[conn]  
 conn.close()  
 continue  
  
 except Exception as e:  
 print(f"Error in main server loop: {e}")  
 raise  
 break  
  
 # Clean up connections  
 for conn in self.client\_connections:  
 conn.close()  
  
 try:  
 self.admin\_conn.close()  
 self.admin\_socket.close()  
 except:  
 pass  
   
 print ("server shutting down")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 try:  
 session = init\_db()  
 server\_ip = "192.168.1.26"   
 admin\_port = 5000 # main port for admin to server communication  
 admin\_port\_test = 5002 # port for providing client's answers to test  
 client\_port = 5001  
 client\_msg\_clientlist\_port = 5003 # port for providing client's messages to admin, and updates in the clientlist  
  
 server = Server(server\_ip, admin\_port, admin\_port\_test, client\_port, client\_msg\_clientlist\_port)  
 server.start()  
 except Exception as e:  
 print(f"Failed to start server: {e}")

## client.py

# client code  
  
import socket  
import ctypes   
import pyautogui  
import time  
from datetime import datetime  
import tkinter as tk  
from tkinter import messagebox  
import threading  
import queue  
from PIL import Image, ImageTk  
import io  
import ssl  
  
class ClientApp:  
 def \_\_init\_\_(self, master):  
 self.root = root  
 self.master = master  
 self.master.title("Client Application")  
 self.master.geometry("600x1200")  
 self.master.config(bg="#f0f0f0")  
 self.master.resizable(width=True, height=True)  
 self.master.minsize(width=333, height=333) #width=666, height=666  
 self.master.maxsize(width=400, height=400) #width=999, height=999  
   
 self.connected = False  
 self.shutdown = False  
 self.name\_accepted = True  
 self.client\_name = None  
 self.timer\_active = False # Timer is initially inactive  
 self.test\_timer\_id = None  
 self.time\_left = 60 # Default time for countdown  
 self.create\_name\_screen()  
  
 def create\_name\_screen(self):  
 # name screen for entering client name  
 try:  
 self.name\_frame = tk.Frame(self.master, bg="#f0f0f0")  
 self.name\_frame.pack(expand=True)  
  
 tk.Label(self.name\_frame, text="Enter your name:", font=("Arial", 14), bg="#f0f0f0").pack(pady=10)  
   
 self.name\_entry = tk.Entry(self.name\_frame, font=("Arial", 14))  
 self.name\_entry.pack(pady=10)  
 self.name\_entry.bind("<KeyRelease>", self.validate\_name)  
   
 # Bind Enter key to self.name\_entry  
 self.name\_entry.bind("<Return>", lambda event: self.submit\_button.invoke())  
  
 self.submit\_button = tk.Button(self.name\_frame, text="Submit", font=("Arial", 14), command=self.set\_client\_name, state=tk.DISABLED)  
 self.submit\_button.pack(pady=10)  
 except Exception as e:  
 messagebox.showerror("UI Error", f"Could not create name screen: {str(e)}")  
  
  
 def validate\_name(self, event):  
 if self.name\_entry.get().strip():  
 self.submit\_button.config(state=tk.NORMAL)  
 else:  
 self.submit\_button.config(state=tk.DISABLED)  
 def contains\_hebrew(self, string):  
 # function that check if the client name contains hebrew letters.  
 for char in string:  
 if '\u0590' <= char <= '\u05FF': # Check if character is in the Hebrew Unicode range  
 return True  
 return False  
  
  
 def set\_client\_name(self):  
 # function that sets the client name and check for invalid names  
 self.client\_name = self.name\_entry.get().strip()  
   
 if "." in self.client\_name :  
 self.master.after(0, lambda: messagebox.showerror("Error", "You cannot set a name with dots"))  
 self.name\_frame.destroy()  
 self.create\_name\_screen()  
 return  
 elif "(" in self.client\_name :  
 self.master.after(0, lambda: messagebox.showerror("Error", "You cannot set a name with '(' "))  
 self.name\_frame.destroy()  
 self.create\_name\_screen()  
 return  
 elif "all" in self.client\_name.lower():  
 self.master.after(0, lambda: messagebox.showerror("Error", "You cannot set a name with the word- all"))  
 self.name\_frame.destroy()  
 self.create\_name\_screen()  
 return  
 elif self.contains\_hebrew(self.client\_name.lower()):  
 self.master.after(0, lambda: messagebox.showerror("Error", "You cannot set a name with Hebrew letters"))  
 self.name\_frame.destroy()  
 self.create\_name\_screen()  
 return  
 self.name\_frame.destroy()  
 self.setup\_ui()  
  
   
 def setup\_ui(self):  
 # Adjust window limits  
 self.master.minsize(width=666, height=999)  
 self.master.maxsize(width=666, height=999)  
  
 self.message\_queue = queue.Queue()  
  
 # Client Title Section  
 self.top\_frame = tk.Frame(self.master, bg="#f0f0f0")  
 self.top\_frame.pack(pady=10, padx=20, fill=tk.X)  
 tk.Label(self.top\_frame, text=f"Client: {self.client\_name}", anchor="w", justify="left", bg="#f0f0f0", font=("Arial", 12), fg="black").pack(fill="both", expand=True)  
  
 self.start\_button = tk.Button(self.top\_frame, text="Start Client", command=self.start\_client, font=("Arial", 14), bg="#4CAF50", fg="white", relief="raised", bd=2)  
 self.start\_button.pack(pady=10)  
  
 # Message Send Section  
 self.send\_frame = tk.Frame(self.master, bg="#f0f0f0")  
 self.send\_frame.pack(fill=tk.X, padx=20, pady=10)  
  
 self.message\_entry = tk.Entry(self.send\_frame, font=("Arial", 16), bd=3, relief="sunken")  
 self.message\_entry.pack(side=tk.LEFT, fill=tk.X, expand=True, padx=(0, 10), ipady=8) # ipady makes it taller  
 self.message\_entry.bind("<Return>", self.send\_message)  
  
 self.send\_button = tk.Button(self.send\_frame, text="Send", font=("Arial", 14), bg="#4CAF50", fg="white",  
 command=self.send\_message, padx=15, pady=8)  
 self.send\_button.pack(side=tk.RIGHT)  
  
 # Log Section  
 self.log\_frame = tk.Frame(self.master, bg="#f0f0f0", bd=2, relief="sunken")  
 self.log\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=10)  
  
 tk.Label(self.log\_frame, text="Log Messages", font=("Arial", 14), bg="#f0f0f0", anchor="w").pack(fill="x", padx=10)  
  
 self.log\_scrollbar = tk.Scrollbar(self.log\_frame, orient="vertical")  
 self.log\_scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
  
 self.log\_display = tk.Text(self.log\_frame, height=10, wrap=tk.WORD, font=("Arial", 12), bg="#e6e6e6", fg="black", padx=5, pady=5, state=tk.DISABLED)  
 self.log\_display.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)  
 self.log\_display.config(yscrollcommand=self.log\_scrollbar.set)  
 self.log\_scrollbar.config(command=self.log\_display.yview)  
  
 # Message Display Section  
 tk.Label(self.master, text="Messages", font=("Arial", 14), bg="#f0f0f0", anchor="w").pack(fill="x", padx=20, pady=5)  
  
 # Create a frame for the message\_display and scrollbar  
 self.message\_frame = tk.Frame(self.master)  
 self.message\_frame.pack(fill=tk.X, padx=20, pady=5)  
  
 # Add Scrollbar for the message\_display section  
 self.message\_scrollbar = tk.Scrollbar(self.message\_frame, orient="vertical")  
 self.message\_scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
  
 # Create the message\_display Text widget  
 self.message\_display = tk.Text(self.message\_frame, height=5, wrap=tk.WORD, font=("Arial", 12), bg="white", fg="black", padx=5, pady=5, state=tk.DISABLED)  
 self.message\_display.pack(side=tk.LEFT, fill=tk.X, expand=True)  
 self.message\_display.config(yscrollcommand=self.message\_scrollbar.set)  
 self.message\_scrollbar.config(command=self.message\_display.yview)  
  
  
 # File Display Section (Canvas)  
 tk.Label(self.master, text="Received Files / Images", font=("Arial", 14), bg="#f0f0f0", anchor="w").pack(fill="x", padx=20, pady=5)  
  
 self.canvas\_frame = tk.Frame(self.master, bg="#f0f0f0")  
 self.canvas\_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=10)  
  
 self.canvas\_scrollbar = tk.Scrollbar(self.canvas\_frame, orient="vertical")  
 self.canvas\_scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
  
 self.canvas = tk.Canvas(self.canvas\_frame, bg="white", yscrollcommand=self.canvas\_scrollbar.set)  
 self.canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)  
 self.canvas\_scrollbar.config(command=self.canvas.yview)  
  
 # Bind mouse wheel to canvas for scrolling  
 self.canvas.bind\_all("<MouseWheel>", self.on\_mouse\_wheel\_canvas)  
 # Download Button  
 self.download\_button = tk.Button(self.master, text="Download File", font=("Arial", 12),  
 bg="#008CBA", fg="white", command=self.download\_file)  
 self.download\_button.pack(pady=5)  
 self.download\_button.config(state=tk.DISABLED) # Disabled until a file is received  
  
  
 # Networking Setup  
 self.server\_ip = "192.168.1.26"  
 self.client\_port = 5001  
 self.client\_socket = None  
 self.running = False  
 self.user32 = ctypes.WinDLL('user32', use\_last\_error=True)  
 self.master.after(100, self.process\_queue)  
  
 def send\_message(self, event=None):  
 # function to send messages to admin  
 message = self.message\_entry.get().strip()  
 if self.running:  
 if message:  
 if self.contains\_hebrew(message):  
 self.master.after(0, lambda: messagebox.showerror("Error", "You cannot send message with Hebrew letters"))  
 return  
 self.message\_display.config(state=tk.NORMAL)  
 self.message\_display.insert(tk.END, f"Me: {message}\n")  
 self.message\_display.config(state=tk.DISABLED)  
 self.message\_display.see(tk.END)  
  
 self.client\_socket.sendall(f"msg {self.client\_name}: {message}".encode())  
 self.message\_entry.delete(0, tk.END)  
 else:  
 messagebox.showinfo("Empty Message", "Please type a message before sending.")  
 else:  
 messagebox.showinfo("Connection Error", "You must start the client first")  
  
 def log\_message(self, message):  
 # log history messages  
 self.message\_queue.put(message)  
   
 def widget\_alive(self, widget):  
 try:  
 return widget.winfo\_exists()  
 except tk.TclError:  
 return False  
  
 def process\_queue(self):  
 try:  
 if self.widget\_alive(self.master):  
 # Check if the log display widget exists and is still active  
 if self.widget\_alive(getattr(self, "log\_display", None)):  
 # Drain the message queue  
 while not self.message\_queue.empty():  
 message = self.message\_queue.get\_nowait()  
 self.log\_display.config(state=tk.NORMAL)  
 self.log\_display.insert(tk.END, message + "\n")  
 self.log\_display.yview(tk.END)  
 self.log\_display.config(state=tk.DISABLED)  
 self.master.after(100, self.process\_queue)  
 else:  
 print("Master window closed, stopping process\_queue.")  
 except Exception as e:  
 print(f"Error in process\_queue: {e}")  
  
  
  
 def block\_keyboard(self):  
 # blocking mouse and keyboard  
 self.user32.BlockInput(True)  
 self.log\_message("Keyboard is now blocked.")  
   
 def unblock\_keyboard(self):  
 # unblocking mouse and keyboard  
 self.user32.BlockInput(False)  
 self.log\_message("Keyboard is now unblocked.")  
  
 def take\_screenshot(self):  
 try:  
 # Generate a filename based on the current date and time  
 current\_time = datetime.now().strftime('%Y-%m-%d\_%H-%M-%S')  
 screenshot\_filename = f"PICTURE\_{current\_time}.png"  
 # Take a screenshot using pyautogui and save it locally  
 screenshot = pyautogui.screenshot()  
 screenshot.save(screenshot\_filename)  
 self.log\_message(f"Screenshot saved as {screenshot\_filename}")  
  
 try:  
 print ("sending screenshot to server")  
 self.client\_socket.sendall(b"PIC\_START")  
  
 # Open the screenshot file in binary mode and read it in chunks  
 with open(screenshot\_filename, "rb") as f:   
 num = 0  
 while True:  
 num+=1  
 chunk = f.read(1024)  
 if not chunk:  
 break # Stop when the file is fully read  
  
 self.client\_socket.sendall(chunk) # Send chunk  
 # print ("sent chunk ",num)  
   
 self.client\_socket.sendall(b"PIC\_END")  
 print("Screenshot sent successfully.")  
  
 except Exception as e:  
 self.log\_message(f"Failed to send screenshot: {e}")  
 except Exception as e:  
 self.log\_message(f"Error taking screenshot: {e}")  
  
 def load\_questions\_from\_content(self, content):  
 # load questions from test  
 try:  
 # Step 1: Decode the content from bytes to a string, and split into lines  
 lines = content.decode().splitlines()  
  
 questions = []  
 current\_question = None  
 # Step 2: Loop through each line to build question blocks  
 for line in lines:  
 line = line.strip()  
 if line.startswith("Question"):  
 if current\_question:  
 questions.append(current\_question)  
 current\_question = {"question": line, "options": [], "answer": None}  
 elif line.startswith("Answer:"):  
 current\_question["answer"] = int(line.split(":")[1].strip())  
 elif line and current\_question:  
 current\_question["options"].append(line)  
 # Step 3: After loop ends, save the last question if it exists  
 if current\_question:  
 questions.append(current\_question)  
  
 return questions  
 except Exception as e:  
 self.log\_message(f"Failed to load questions: {e}")  
 return [] # Return an empty list on failure  
  
  
 def check\_answers(self, selected\_answers, questions):  
 # checking test answer  
 try:  
 print ("check\_answers")  
 score = 0  
 for i, var in enumerate(selected\_answers):  
 if var.get () == questions[i]["answer"]:  
 score += 1  
 # Calculate percentage grade  
 s = int((score / len(questions)) \* 100)  
 # Send the score to the server  
 self.client\_socket.sendall(f"TEST\_ANSWER {s}: {self.client\_name}".encode())  
 print ("sent")  
 #self.client\_socket.sendall(b"TEST\_ANSWER 2")  
 messagebox.showinfo("Results", f"You got {score}/{len(questions)} correct!")  
 except Exception as e:  
 self.log\_message(f"Error checking answers: {e}")  
 messagebox.showerror("Error", f"An error occurred while checking answers: {e}")  
  
 def create\_test\_gui(self, filename):  
 # creating the test  
 try:  
 # Step 1: Load questions from the received file content  
 questions = self.load\_questions\_from\_content(self.received\_file\_content)  
  
 # Step 2: Setup canvas for test display  
 self.canvas.delete("all")  
 scrollable\_frame = tk.Frame(self.canvas, bg="#ffffff")  
 window\_id = self.canvas.create\_window((0, 0), window=scrollable\_frame, anchor="nw")  
  
 def update\_scroll\_region(event=None):  
 self.canvas.configure(scrollregion=self.canvas.bbox("all"))  
   
 scrollable\_frame.bind("<Configure>", update\_scroll\_region)  
  
 def on\_canvas\_resize(event):  
 canvas\_width = event.width  
 self.canvas.itemconfig(window\_id, width=canvas\_width)  
  
 self.canvas.bind("<Configure>", on\_canvas\_resize)  
  
 # Step 3: Initialize selected answers and submit button  
 selected\_answers = []  
 submit\_button = None  
  
 def check\_all\_answered(\*args):  
 # Check if all questions have been answered  
 all\_answered = all(var.get() != 0 for var in selected\_answers)  
 if all\_answered:  
 submit\_button.config(state=tk.NORMAL)  
  
 # Step 4: Reset timer and start countdown  
 self.timer\_active = True  
 self.time\_left = 60  
  
 # Countdown  
 def countdown(time\_left):  
 if not self.timer\_active:  
 return  
 if time\_left <= 0:  
 self.timer\_active = False  
 self.submit\_and\_disable(submit\_button, selected\_answers, questions)  
 messagebox.showinfo("Time's up", "Your test has been submitted.")  
 else:  
 timer\_label.config(text=f"⏱ Time left: {time\_left}s")  
 self.test\_timer\_id = self.canvas.after(1000, countdown, time\_left - 1)  
  
 # RED and BIG countdown timer  
 timer\_label = tk.Label(scrollable\_frame, text=f"⏱ Time left: {self.time\_left}s",  
 font=("Arial", 20, "bold"), fg="red", bg="white")  
 timer\_label.pack(side="right", padx=20, pady=10)  
  
 countdown(self.time\_left)  
  
 # Step 5: Loop through questions and display them  
 for i, q in enumerate(questions):  
 tk.Label(scrollable\_frame, text=q["question"], font=("Arial", 16, "bold"),  
 bg="white", wraplength=700, justify="left", anchor="w", pady=5).pack(anchor="w", padx=10)  
  
 var = tk.IntVar()  
 var.trace\_add("write", check\_all\_answered)  
 selected\_answers.append(var)  
  
 for idx, option in enumerate(q["options"]):  
 tk.Radiobutton(scrollable\_frame, text=option, variable=var, value=idx + 1,  
 bg="white", anchor="w", justify="left", font=("Arial", 15)).pack(anchor="w", padx=20, pady=2)  
 # Step 6: Submit button (disabled until all questions are answered)  
 submit\_button = tk.Button(scrollable\_frame, text="Submit",  
 command=lambda: self.submit\_and\_disable(submit\_button, selected\_answers, questions),  
 bg="#28a745", activebackground="#218838", fg="white",  
 font=("Arial", 14, "bold"), padx=10, pady=6,  
 state=tk.DISABLED)  
 submit\_button.pack(pady=20)  
 except Exception as e:  
 self.log\_message(f"Error creating test GUI: {e}")  
 messagebox.showerror("Error", f"An error occurred while creating the test interface: {e}")  
  
  
  
 def submit\_and\_disable(self, button, selected\_answers, questions):  
 try:  
 self.timer\_active = False # Stop the timer immediately  
 # Check answers and disable the submit button  
 self.check\_answers(selected\_answers, questions)  
 button.config(state=tk.DISABLED)  
 # Clear the canvas after submission  
 self.canvas.delete("all")  
 except Exception as e:  
 self.log\_message(f"Error submitting answers: {e}")  
 messagebox.showerror("Error", f"An error occurred while submitting the answers: {e}")  
  
  
 def handle\_message(self, message):  
 # handle reccived messages from server  
 try:  
 # Handle block/unblock requests  
 if message.startswith(b"BLOCK"):  
 self.log\_message("Received block request")  
 self.block\_keyboard()  
  
 elif message.startswith(b"UNBLOCK"):  
 self.unblock\_keyboard()  
  
 # Handle removal request - cleanup  
 elif message.startswith(b"REMOVE"):  
 self.cleanup(confirm=False)  
  
 # message from admin  
 elif message.startswith(b"MSG"):   
 msg = message.decode()[3:].strip() # Remove "MSG" prefix and strip extra spaces  
 self.log\_message(f"Received message: {msg}")  
 self.display\_message(f"admin: {msg}") # Display the message in the Text widget  
  
 # Handle file data transfer (e.g., .txt files or png image data)  
 elif message.startswith(b"DATA\_NAME"):  
 while b"DATA\_END" not in message:  
 chunk = self.client\_socket.recv(1024)  
 message += chunk  
 message = message.replace(b'DATA\_NAME', b'').replace(b'DATA\_END', b'')  
 msg = message.split(b'DATA\_START')  
 file\_name = msg[0].strip().decode()  
 content = msg[1].strip()  
  
 self.log\_message(f"Received data file: {file\_name}")  
   
 self.received\_filename = file\_name  
 self.received\_file\_content = content # Store content for downloading  
 self.download\_button.config(state=tk.NORMAL) # Enable download button  
   
 self.canvas.delete("all")  
 if ".txt" in file\_name.lower(): # Handle text file  
 if "test" in file\_name.lower(): # Handle test file  
 print("answering a test")  
 self.client\_socket.sendall(f"{self.client\_name} is answering a test".encode())  
 self.download\_button.config(state=tk.DISABLED) # Disabled until finish test  
 self.create\_test\_gui(file\_name)  
 else: # Regular text file  
 print ("regular file")  
 content = content.replace(b'\n', b'')  
 self.canvas.create\_text(10, 10, text=content, font=("Arial", 14), fill="blue", anchor="nw")  
 else: # Handle non-text files (png)  
 try:  
 image\_data = io.BytesIO(content)  
 image = Image.open(image\_data)  
 photo = ImageTk.PhotoImage(image)  
 self.canvas.image = photo # Keep a reference to the image  
 self.canvas.create\_image(10, 10, anchor="nw", image=photo)  
 except:  
 self.log\_message("Error processing image")  
   
 self.canvas.config(scrollregion=self.canvas.bbox("all"))  
  
 # Handle screenshot requests  
 elif message.startswith(b"SCREENSHOT"):  
 self.take\_screenshot()  
 else:  
 self.log\_message(f"Unknown command: {message}") # Log unknown message types  
   
 except Exception as e:  
 self.log\_message(f"Error handling message: {e}") # Log any errors that occur during message handling  
  
 def download\_file(self):  
 # download txt or png files  
 try:  
 if hasattr(self, "received\_filename") and hasattr(self, "received\_file\_content"):  
 try:  
 # Attempt to save the file content to disk  
 with open(self.received\_filename, "wb") as f:  
 f.write(self.received\_file\_content)  
 self.log\_message(f"File downloaded: {self.received\_filename}")  
 except Exception as e:  
 self.log\_message(f"Error saving file: {e}")  
 else:  
 # Handle the case where no file content is available  
 self.log\_message("No file to download.")  
   
 except Exception as e:  
 # Catch any exceptions during the file-saving process and log the error  
 self.log\_message(f"Error saving file: {e}")  
  
  
 def display\_message(self, msg):  
 # displaying messages  
 try:  
 self.message\_display.config(state=tk.NORMAL)  
 self.message\_display.insert(tk.END, msg + "\n")  
 self.message\_display.yview(tk.END)  
 self.message\_display.config(state=tk.DISABLED)  
 except Exception as e:  
 self.log\_message(f"Error displaying message: {e}") # Log any errors that occur  
  
 def start\_client(self):  
 print ("starting client")  
 self.start\_button.config(state=tk.DISABLED) # Disable the start button  
 if not self.running:  
 self.running = True  
 threading.Thread(target=self.run\_client, daemon=True).start()  
   
  
 def run\_client(self):  
 # this function is running the client, checking with the server if the name is already exists, and listening for messages  
 while True and self.running: # Keep retrying until we successfully connect  
 try:  
 sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
  
 ssl\_context = ssl.create\_default\_context()  
 ssl\_context.check\_hostname = False # Disable hostname verification  
 ssl\_context.verify\_mode = ssl.CERT\_NONE # Don't require a trusted CA  
  
 self.client\_socket = ssl\_context.wrap\_socket(sock, server\_hostname=self.server\_ip)  
 try:  
 self.client\_socket.connect((self.server\_ip, self.client\_port))  
 self.connected = True  
 self.log\_message("Connected to the server. Waiting for messages...")  
 except ConnectionRefusedError as e:  
 if self.connected == False:  
 self.running = False  
 # Handle the case where the connection is refused  
 print(f"Connection refused: {e}")  
 self.log\_message("Connection refused. Server might not be running.")  
 self.start\_button.config(state=tk.NORMAL)  
 return # Exit the loop or retry if you want  
  
 # Send client name to server  
 name\_message = f"{self.client\_name}".encode()  
 print ("sending client name to server ",name\_message,"&&")  
 self.client\_socket.sendall(name\_message)  
  
  
 # Receive server response  
 response = self.client\_socket.recv(1024).decode().strip()  
 print(f"[CLIENT] Server responded with: '{response}'") # Debug log  
  
 if response.lower() == "ok":  
 print(f"[CLIENT] Name '{self.client\_name}' accepted! Proceeding to main chat...")  
 self.name\_accepted = True  
 else:  
 print(f"[CLIENT] Name '{self.client\_name}' was rejected. Re-entering name selection...")  
 self.log\_message("Error: Client name already in use. Restarting name entry...")  
  
 # Close socket before restarting  
 self.client\_socket.close()  
 self.client\_socket = None  
   
 # Destroy all widgets inside master, but not master itself  
 for widget in self.master.winfo\_children():  
 widget.destroy()  
  
 # Show error  
 self.master.after(0, lambda: messagebox.showerror("Error", "Client name already in use"))  
  
 # Go back to the name selection screen  
 self.create\_name\_screen()  
  
 #self.running = False  
 self.name\_accepted = False  
 return  
  
 # Handle messages normally after successful connection  
 while self.running:  
 if self.shutdown: # <-- Check if shutting down  
 break  
 print ("awaiting message from server")  
 message = self.client\_socket.recv(1024)  
 if not message:  
 print ("server ended connection")  
 break  
 self.handle\_message(message)  
  
  
 except Exception as e:  
 print ("error in client")  
 self.log\_message(f"Error on client: {e}")  
 self.client\_socket = None # Ensure socket is reset on failure  
   
 finally:  
 if self.name\_accepted == True:  
 if self.client\_socket: # Only clean up if a valid connection was made  
 if self.running:  
 self.cleanup(confirm=False) #confirm=True  
 else:  
 print("name didnt accepted")  
 break  
  
  
 def cleanup(self, confirm=True):  
 # cleanning up the client and closing  
 print("in cleanup")  
 if confirm:  
 if not messagebox.askokcancel("Quit", "Are you sure you want to exit?"):  
 return  
 try:  
 msg = f"shutting down - {self.client\_name}"  
 print("msg- ", msg)   
 self.client\_socket.sendall(msg.encode())  
 print("Cleaning up before exit...")  
 self.running = False  
 self.shutdown = True # <-- Set shutdown flag  
 try:  
 if self.client\_socket:  
 #self.client\_socket.close()  
 self.log\_message("Disconnected from the server.")  
 except Exception as e:  
 print("Socket close error:", e)  
  
 print("cleaned up successfully")  
  
 #if confirm:  
 self.root.destroy()  
 except:  
 print ("not connected to server, clean up")  
 self.root.destroy()  
  
  
 def on\_mouse\_wheel\_canvas(self, event):  
 # allow mouse wheel on canvas  
 if event.delta > 0:  
 self.canvas.yview\_scroll(-1, "units")  
 else:  
 self.canvas.yview\_scroll(1, "units")  
  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 root = tk.Tk()  
 app = ClientApp(root)  
  
  
 def on\_close():  
 app.cleanup(confirm=True) # shows messagebox on "X" button  
  
   
 root.protocol("WM\_DELETE\_WINDOW", on\_close)  
 root.mainloop()