

**פרויקט גמר בתכנון ותכנות מערכות
בהתמחות הגנת סייבר**



שם התלמיד: איתמר דוד

תאריך הגשה: 18.6.2024

תוכן עניינים

2	תוכן עניינים
4	מבוא
4	למה בחרתי בשם Nimbus
4	האתגרים שהתמודדתי איתם בפרויקט
5	המניעים לפיתוח הפרויקט
5	הלקוחות במערכת
5	יעדים ומטרות
6	פתרונות קיימים
6	הפתרון שהפרויקט מציע
7	סקירת הטכנולוגיות בפרויקט
7	תיחום הפרויקט
7	תיאור המערכת
8	פירוט יכולות
8	פירוט בדיקות
9	תכנון ולוח זמנים
9	ניהול סיכונים
10	תיאור תחום הידע - פרק מילולי
10	יכולות התחברות
12	יכולות לקוח רגיל
14	יכולות מצד לקוח המנהל
15	ארכיטקטורה
15	תיאור הארכיטקטורה של המערכת המוצעת
19	תיאור הטכנולוגיה הרלוונטית
20	תיאור אלגוריתמים מרכזיים בפרויקט
22	תיאור סביבת הפיתוח
22	תיאור פרוטוקול התקשורת
26	תיאור מסכי המערכת
31	תיאור מבני נתונים
33	סקירת חולשות ואיומים (הפרק האהוב עליי 😊)
38	מימוש משתמש
38	מודולים מיובאים - סיפרות חיצוניות
38	המודולים שלי
39	המחלקות שלי
42	פונקציות עזר
46	קטעי קוד נבחרים
48	מסמך בדיקות
51	מדריך למשתמש
51	עץ קבצים

51.....	הסביבה הנדרשת
52.....	מדריך התקנת המערכת אצל לקוח
52.....	מיקומי קבצים ונתונים התחלתיים
52.....	שימוש במערכת- לקוח רגיל
53.....	שימוש במערכת- לקוח מנהל
54.....	סיכום אישי / רפלקציה
54.....	תיאור תהליך העבודה על הפרויקט
54.....	האתגרים, ההצלחות, הקשיים ודרכי הפתרון
54.....	תהליך הלמידה
54.....	יכולות חדשות שנלמדו
55.....	כלים להמשך
55.....	תובנות מהתהליך
55.....	כיצד ניתן לשפר את הפרויקט?
55.....	תודות
56.....	ביבליוגרפיה
56.....	מודולי הבינה המלכותית שנעזרתי בהם
56.....	תיעוד של הסיפוריות החיצוניות שהשמשתי בהם
56.....	ספר רשתות
56.....	Flaticon
57.....	נספחים
57.....	Protocol.py
59.....	Logger.py
60.....	Create_venv.py
60.....	Run_Nimbus.bat
61.....	Client.py
85.....	Client_Admin.py
91.....	Server.py
113.....	Server_Admin.py

מבוא

בעידן הדיגיטלי של ימינו, כמות המידע והנתונים האישיים שאנו צוברים גדלה בקצב מסחרר. אנו זקוקים לפתרונות חכמים לניהול ואחסון המידע שיאפשרו לנו גישה נוחה מכל מקום, אך גם יעניקו הגנה מקסימלית לפרטיות שלנו. הפרויקט שפיתחתי עונה בדיוק על הצרכים הללו. הוא מציע שירותי אחסון קבצים בענן המשלבים את הנוחות והזמינות עם הצפנה מתקדמת ואבטחה ברמה הגבוהה ביותר. עם ממשק אינטואיטיבי וידידותי למשתמש, תוכלו בקלות למחוק, להוריד, ולהעלות את הקבצים שלכם מכל מקום ובכל זמן. המערכת מושלמת לשימוש אישי או עסקי, ומספקת מגוון רחב של כלים לניהול יעיל של הנתונים תוך שמירה קפדנית על חשאיות וביטחון המידע. בין אם אתם צריכים לאחסן מסמכים, תמונות, סרטונים או כל סוג אחר של קבצים דיגיטליים.

פרויקט זה ששמו הוא "Nimbus" הוא מערכת לאחסון וניהול קבצים מרחוק. הפרויקט מאפשר למשתמשים להירשם ולהתחבר בעזרת חשבון אימייל ובכך לגשת ולנהל את הקבצים והתיקיות שלהם מכל מקום.

למה בחרתי בשם Nimbus

אז למה בעצם "Nimbus"? קודם כל, Nimbus הוא סוג של ענן בשמיים, וזה מושלם כי הפרויקט שלי עוסק בדיוק בזה – אחסון בענן. שם כמו Nimbus לא רק מתאר בצורה מדויקת את מהות השירות, אלא גם מעורר תחושה של קלילות, זרימה וחופש, בדיוק כמו העננים בשמיים.

כמו כן, יש לשם משמעות מיוחדת נוספת בישראל. זהו גם השם של הענן הממשלתי של ישראל, שנועד לספק שירותי מחשוב ענן למשרדי הממשלה וגופים ציבוריים.

האתגרים שהתמודדתי איתם בפרויקט

- שילוב טכנולוגיות וסביבות שלא עבדתי איתן לפני כמו שרת SMTP, הצפנת מידע באמצעות RSA ו AES.
- שמירה על קבצי המשתמשים בצורה מאובטחת במהלך האחסון בשרת.
- ניהול שגיאות, קריסות והצגת הודעות ברורות לשרת וללקוח.
- העברת תיקיות גדולות במלואן תוך שמירה על כל התוכן שבהן.
- שמירה על קוד נקי ומסודר לאורך כל תהליך הפיתוח.

המניעים לפיתוח הפרויקט

בחרתי ברעיון זה לפרויקט ממספר סיבות. ראשית, פרויקט של מערכת קבצים בענן מאפשר הרחבות ותוספות רבות שאינן תלויות זו בזו, כך שאוכל להמשיך ולהרחיב את הפרויקט גם לאחר הגשתו. שנית, הפרויקט כולל הרבה אלמנטים של אבטחה (כגון הצפנת סיסמאות, תעבורה, וקבצים) שמאוד מעניינים אותי. במהלך השנה חקרתי ולמדתי על התקפות שונות ועל דרכים להתגונן מפניהן. לבסוף, בחרתי ברעיון זה משום שמדובר במוצר קיים בגרסאות שונות של חברות כמו [Google Drive](#) ו-[Dropbox](#), מה שאיפשר לי לקבל השראה וללמוד על המתרחש מאחורי הקלעים ועל היקף ההשקעה במוצרים אלו.

הלקוחות במערכת

קהל היעד של המערכת הוא כל אדם (משתמש) שיש ברשותו מחשב עם [windows](#) וקבצים שאותם הוא רוצה לשמור במקום בטוח.

יעדים ומטרות

המטרות האישיות שלי לפרויקט:

- פיתוח מערכת מלאה מקצה לקצה עם משתמשים
- קביעת יעדים ברורים וריאליים לפיתוח הפרויקט.
- העמקת ההבנה שלי בנושאי אבטחת מידע, רשתות תקשורת ומערכות הפעלה.
- כתיבת קוד נקי, קריא ומתועד היטב שקל לתחזק ולהרחיב בעתיד.

מטרות המערכת שאני צופה להשיג בפרויקט:

- ממשק משתמש ידידותי, אינטואיטיבי ונוח לשימוש.
- אפשרות הרשמה והתחברות של משתמשים חדשים וקיימים.
- מנגנון שחזור סיסמה מאובטח למקרה שמשתמש ישכח את פרטי ההתחברות.
- תמיכה בהעלאה, הורדה ומחיקה של קבצים ותיקיות מהענן.
- הצפנת כל הנתונים הרגישים כולל סיסמאות, והקבצים עצמם.
- פיתוח ממשק ייעודי למנהל המערכת לבקרה ופיקוח על המשתמשים והפעילות.

פתרונות קיימים

Google Drive הוא שירות אחסון בענן של חברת גוגל המאפשר למשתמשים לשמור ולשתף קבצים בקלות. השירות מציע אפשרויות נרחבות לאחסון קבצים, כולל מסמכי טקסט, תמונות, סרטונים ועוד, ונגישות למשתמשים מכל מכשיר עם חיבור לאינטרנט. תוך כדי שימוש בגוגל דרייב, ניתן ליצור מסמכים חדשים, לערוך אותם בזמן אמת יחד עם משתמשים אחרים, ולשתף קבצים עם אנשים נוספים באמצעות שיתוף קישור או הזמנה לתיקייה מסוימת. בנוסף, גוגל דרייב מאפשר סנכרון קבצים בין מכשירים שונים וגיבוי קבצים אוטומטיים.



Dropbox הוא שירות אחסון בענן ושיתוף קבצים הנפוץ מאוד בעולם העסקי והאישי. הוא מאפשר למשתמשים לשמור את קבציהם באופן מאובטח ולגשת אליהם מכל מקום ומכל מכשיר עם חיבור לאינטרנט. באמצעות דרופבוקס, ניתן לשתף קבצים עם אנשים אחרים על ידי שיתוף קישור או הזמנתם לתיקייה מסוימת. השירות מצויד בתכונות נוספות כמו גיבוי אוטומטי של קבצים, סנכרון בין מכשירים שונים ואפשרות לשיתוף קבצים עם צוותים וקבוצות עבודה.



Microsoft OneDrive הוא שירות אחסון ושיתוף קבצים בענן תחת מערכת הענן של מיקרוסופט. השירות מאפשר למשתמשים לשמור קבצים בענן ולגשת אליהם מכל מקום ומכל מכשיר המחובר לאינטרנט. וואן דרייב משלב התאמה עם מוצרי מיקרוסופט אחרים כגון חבילת האופיס, מאפשר יצירת מסמכים חדשים במגוון פורמטים (כגון מסמכי Word, Excel ו-PowerPoint) ושמירתם ישירות ב-OneDrive. בנוסף, השירות מציע תכונות נוספות כמו סנכרון אוטומטי של קבצים בין מכשירים שונים, גיבוי קבצים בצורה אוטומטית, ואפשרות לשיתוף קבצים ותיקיות עם אנשים אחרים דרך שיתוף קישור או הזמנה לתיקייה מסוימת.



הפתרון שהפריקט מציע

הפריקט מציע מקום נגיש וזמין לאחסון וסידור קבצים בענן, כך שהגישה לקבצים אלה יכולה להתבצע מכל מחשב.

סקירת הטכנולוגיות בפרויקט

הפרויקט שלי בניגוד לשאר הפתרונות המרכזיים בשוק, אינו מציע עריכה של הקבצים אלא רק אחסונם בשרת. לדבר יש יתרונות וחסרונות, אך לדעתי היתרון המרכזי בכך הוא הפשטות של התוכנה מצד הלקוח, הוא לא צריך להסתבך עם כפתורים רבים אלא יכול לעלות את הקבצים שהוא רוצה לגבות בפשטות רבה, שזאת המטרה המרכזית של הפרויקט.

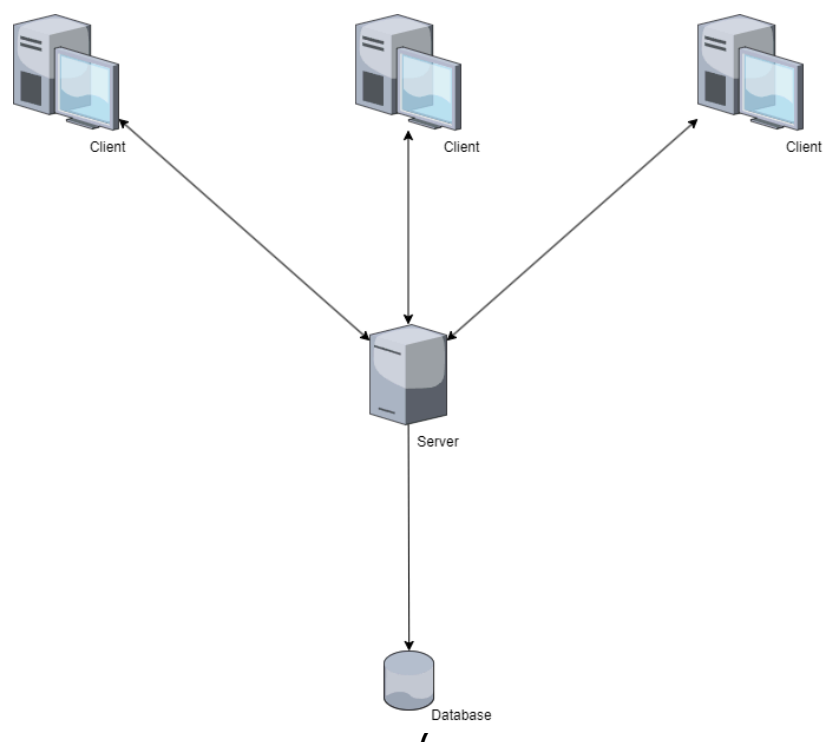
הטכנולוגיות בהן אני עושה שימוש הן חדשניות ונמצאות כיום בשימוש בשירותים הגדולים והפופולריים ביותר.

תיחום הפרויקט

הפרויקט עוסק במגוון תחומים ברשתות, מערכות הפעלה והגנת סייבר:

- יצירת קשר בין שרת ולקוחות
- שרת המטפל במספר לקוחות במקביל (multithreading programming)
- תקשורת ושליחת אימיילים עם שרת smtp של גוגל
- הצפנת מידע וחיבור מאובטח עם הצפנה סימטרית (AES) ואסימטרית (RSA)
- ממשק גרפי עם ספריית tkinter
- תקשורת עם מבנה נתונים באמצעות sqlite

תיאור המערכת



פירוט יכולות

יכולות התחברות-

- התחברות למשתמש קיים
- יצירת משתמש חדש
- שינוי סיסמא למשתמש קיים

יכולות לקוח רגיל-

- העלאת קובץ/תיקייה לשרת
- מחיקת קובץ/תיקייה מהשרת
- הורדת קובץ/תיקייה מהשרת
- יצירת תיקייה חדשה

יכולות לקוח מנהל-

- צפייה בלוגים
- צפייה בלקוחות קיימים
- מחיקת משתמש

פירוט בדיקות

- בדיקת העלאת קובץ/תיקייה לשרת ובדיקה על העלאת קובץ/תיקייה עם שם קיים. מטרת המערכת היא לשמור את הקבצים/תיקיות בצורה מוצפנת ומאובטחת ולאחר מכן שהלקוח יקבל את הקובץ בצורתו השלמה.
- ביצוע בדיקת אימות משתמש מתבצע באמצעות שליחת פרטי משתמש תקין ופרטי משתמש לא תקין לשרת, כמו גם בדיקת משתמש קיים ומשתמש שאינו קיים במאגר הנתונים. מטרת המערכת היא לאשר משתמשים תקינים הקיימים במערכת ולחסום משתמשים לא תקינים או שאינם קיימים במערכת.
- בדיקת הצפנת התעבורה מתבצעת באמצעות ניתוח התעבורה ובדיקת האפשרות לצפות בפקטות המועברות. הבדיקה כוללת ניסיון לגשת ולפענח את הנתונים המועברים ללא שימוש במפתחות ההצפנה המתאימים. מטרת המערכת היא לוודא שהמידע בתעבורה מוגן היטב על ידי הצפנה, כך שלא ניתן יהיה לחשוף את הנתונים המועברים לגורמים בלתי מורשים.

תכנון ולוח זמנים

תכנון השלבים העיקריים בעבודה במהלך השנה:

ספטמבר - בחירת פרויקט ותכנון ראשוני

הוכחת התכנות

אוקטובר - בחירת כלים מרכזיים בפרויקט (פרוטוקול, ממשק גרפי...)

נובמבר - תחילת לקוח ושרת ראשוניים, יצירת ממשק גרפי למסך הראשי

דצמבר - הוספת לקוחות עם מסד נתונים

ינואר - הוספת אבטחה בסיסית, אפשרויות חדשות ללקוחות (hash, מחיקת והורדת קבצים)

פבואר - הוספת לקוח מנהל, שיפור אבטחה והצפנת קבצים

אפריל ומאי - תיקוני באגים וטאצ'ים סופיים על הקוד עם שילוב עבודה על התיק פרויקט

ניהול סיכונים

חלק מאוד משמעותי בפרויקט שלי הוא הווידי שהקבצים יעברו בצורה תקינה במיוחד לאחר ההצפנה. אם לא הייתי מצליח לבצע את הפעולה הזאת בצורה מיטבית יכולתי לגרום לכך שהשרת מאבד את כל תוכן הקבצים של הלקוחות, הפעולה של שמירת הקבצים בצורה מוצפנת בעל סיכון רב מכיוון שהלקוחות מצפים לשירות מובטח שישמור על הקבצים שלהם בבטחה. עקב הסיכון הגדול בדקתי בקפידה את הקוד שלי בתהליך ווידאתי שהכל מבוצע במדויק ובלי בעיות.

תיאור תחום הידע - פרק מילולי

יכולות התחברות

שם יכולת- התחברות למשתמש קיים

מהות- אישור זהות המשתמש וגישה לחשבוננו האישי במערכת.

אוסף יכולות נדרשות:

- ממשק משתמש – מסך התחברות
- קליטת נתונים – אימייל וסיסמה
- בדיקת תקינות – פורמט האימייל והסיסמה
- שליחת האימייל והסיסמא לשרת - (פרוטוקול התעבורה הכולל הצפנה ופענוח)
- בדיקת זהות – אימות פרטי המשתמש מול בסיס הנתונים בשרת
- קבלת תשובה מהשרת
- פענוח – פענוח התשובה מהשרת
- הצגת התשובה למשתמש – אישור או דחיית ההתחברות

אובייקטים נחוצים: ממשק משתמש, הצפנה/פענוח, תקשורת, גישה למסד הנתונים.

שם יכולת- יצירת משתמש חדש במערכת

מהות- רישום משתמש חדש במערכת – קליטת פרטים אישיים נדרשים ואימות אימייל

אוסף יכולות נדרשות:

- ממשק משתמש – מסך התחברות
- קליטת נתונים – אימייל וסיסמה
- בדיקת תקינות – פורמט האימייל והסיסמא
- שליחת האימייל והסיסמא לשרת - (פרוטוקול התעבורה הכולל הצפנה ופענוח)
- בדיקת תקינות מול בסיס הנתונים בשרת (בדיקה שאין משתמש עם שם משתמש או אימייל זהים בבסיס הנתונים)
- יצירת קוד אימות בשרת - יצירת קוד ייחודי לשליחה למשתמש
- שליחת הקוד באימייל למשתמש
- קליטת קוד אימות – מסך להזנת קוד האימות על ידי המשתמש
- שליחת הקוד אימות לשרת - (פרוטוקול התעבורה הכולל הצפנה ופענוח)
- אימות הקוד בשרת
- קבלת תשובה מהשרת – קבלת סטטוס האימות

- פענוח והצגת תשובה למשתמש – פענוח תשובת השרת והצגת סטטוס האימות למשתמש

אובייקטים נחוצים: ממשק משתמש, הצפנה/פענוח, תקשורת, שירות דוא"ל, גישה למסד הנתונים.

שם יכולת- שינוי סיסמא למשתמש קיים

מהות- שחזור סיסמת משתמש במערכת באמצעות אימייל.

אוסף יכולות נדרשות:

- ממשק משתמש – מסך שחזור סיסמה
- קליטת נתונים – אימייל משתמש
- בדיקת תקינות פורמט האימייל
- שליחת האימייל לשרת - (פרוטוקול התעבורה הכולל הצפנה ופענוח)
- בדיקת תקינות מול בסיס הנתונים בשרת - בדיקה שהמשתמש קיים
- יצירת קוד אימות בשרת - יצירת קוד ייחודי לשליחה למשתמש
- שליחת הקוד באימייל למשתמש
- קליטת קוד אימות – מסך להזנת קוד האימות על ידי המשתמש
- שליחת הקוד אימות לשרת - (פרוטוקול התעבורה הכולל הצפנה ופענוח)
- אימות הקוד בשרת
- קליטת סיסמא חדשה - מסך להזנת סיסמא על ידי המשתמש
- בדיקת תקינות פורמט הסיסמא
- שליחת הסיסמא לשרת - (פרוטוקול התעבורה הכולל הצפנה ופענוח)
- אימות הקוד בשרת
- קבלת תשובה מהשרת – קבלת סטטוס האימות
- פענוח והצגת תשובה למשתמש – פענוח תשובת השרת והצגת סטטוס האימות למשתמש

אובייקטים נחוצים: ממשק משתמש, הצפנה/פענוח, תקשורת, שירות דוא"ל, גישה למסד הנתונים.

יכולות לקוח רגיל

שם יכולת- העלאת קובץ/תיקייה לשרת

מהות- העלאת קובץ או תיקייה מהמחשב המקומי לשרת המרכזי.

אוסף יכולות נדרשות:

- ממשק משתמש – מסך ראשי עם כפתור להעלאת קובץ/תיקייה
- בחירת קובץ/תיקייה – פתיחה של סייר הקבצים
- בדיקת תקינות – פורמט וגודל הקובץ/תיקייה
- שליחת הקבצים לשרת - (פרוטוקול התעבורה הכולל הצפנה ופענוח)
- בדיקת תקינות – קיום הרשאות מתאימות לשמירת קבצים
- שמירת נתונים – שמירת הקובץ/תיקייה בשרת
- קבלת תשובה מהשרת ופענוח
- הצגת התשובה למשתמש – אישור או דחיית ההעלאה

אובייקטים נחוצים: ממשק משתמש, הצפנה/פענוח, תקשורת, מערכת קבצים בשרת, גישה לסייר הקבצים בלקוח

שם יכולת- מחיקת קובץ/תיקייה לשרת

מהות- הסרת קובץ או תיקייה מהשרת המרכזי.

אוסף יכולות נדרשות:

- ממשק משתמש – מסך ראשי עם כפתור למחיקת קובץ/תיקייה
- בחירת קבצים/תיקיות במסך הראשי
- שליחת הבקשה לשרת - (פרוטוקול התעבורה הכולל הצפנה ופענוח)
- בדיקת תקינות – קיום הרשאות מתאימות למחיקה
- מחיקת נתונים – הסרת הקובץ/תיקייה מהשרת
- קבלת תשובה מהשרת ופענוח
- הצגת התשובה למשתמש – אישור או דחיית המחיקה

אובייקטים נחוצים: ממשק משתמש, הצפנה/פענוח, תקשורת, מערכת קבצים בשרת

שם יכולת- הורדת קובץ/תיקייה מהשרת

מהות- הורדת קובץ או תיקייה מהשרת למחשב המקומי.

אוסף יכולות נדרשות:

- ממשק משתמש – מסך ראשי עם כפתור להורדת קובץ/תיקייה
- בחירת קבצים/תיקיות במסך הראש
- שליחת הבקשה לשרת - (פרוטוקול התעבורה הכולל הצפנה ופענוח)
- בדיקת תקינות – קיום הרשאות מתאימות להורדה
- שליפת נתונים ופענוח – שליפת הקובץ/תיקייה מהשרת ופענוח תוכן הקובץ
- שליחת הקבצים ללקוח - (פרוטוקול התעבורה הכולל הצפנה ופענוח)
- שמירת הקבצים/תיקיות בתיקיית ההורדות במחשב המקומי

אובייקטים נחוצים: ממשק משתמש, הצפנה/פענוח, תקשורת, מערכת קבצים בשרת ובלקוח

שם יכולת- יצירת תיקייה חדשה

מהות- יצירת תיקייה חדשה בתיקיית הלקוח בצד השרת

אוסף יכולות נדרשות:

- ממשק משתמש – מסך ראשי עם כפתור ליצירת תיקייה
- קליטת שם בשביל התיקייה
- בדיקת תקינות – וידוא ששם התיקייה תקין
- שליחת הבקשה לשרת - (פרוטוקול התעבורה הכולל הצפנה ופענוח)
- בדיקת תקינות – קיום הרשאות מתאימות ובדיקה האם התיקייה קיימת
- יצירת התיקייה בשרת ושינוי שמה למקרה שצריך
- קבלת תשובה מהשרת ופענוח
- הצגת התשובה למשתמש – אישור או דחיית יצירת התיקייה

אובייקטים נחוצים: ממשק משתמש, הצפנה/פענוח, תקשורת, מערכת קבצים בשרת

יכולות מצד לקוח המנהל**שם יכולת- צפייה בלוגים**

מהות- הצגת רשומות הלוג של המערכת למנהל לצורך מעקב ובקרה.

אוסף יכולות נדרשות:

- ממשק משתמש – מסך ראשי הכולל הצגת הלוגים
- שאילת לוגים – שליפת רשומות הלוג מהשרת על ידי שליחת בקשה
- שליחת הבקשה לשרת - (פרוטוקול התעבורה הכולל הצפנה ופענוח)
- קבלת תשובה מהשרת והצגת הנתונים

אובייקטים נחוצים: ממשק משתמש, הצפנה/פענוח, תקשורת, מערכת קבצים בשרת, תיעוד מקרים בקובץ logs בשרת.

שם יכולת- צפייה בלקוחות קיימים במערכת

מהות- הצגת רשימת הלקוחות הקיימים במערכת למנהל לצורך ניהול ומעקב.

אוסף יכולות נדרשות:

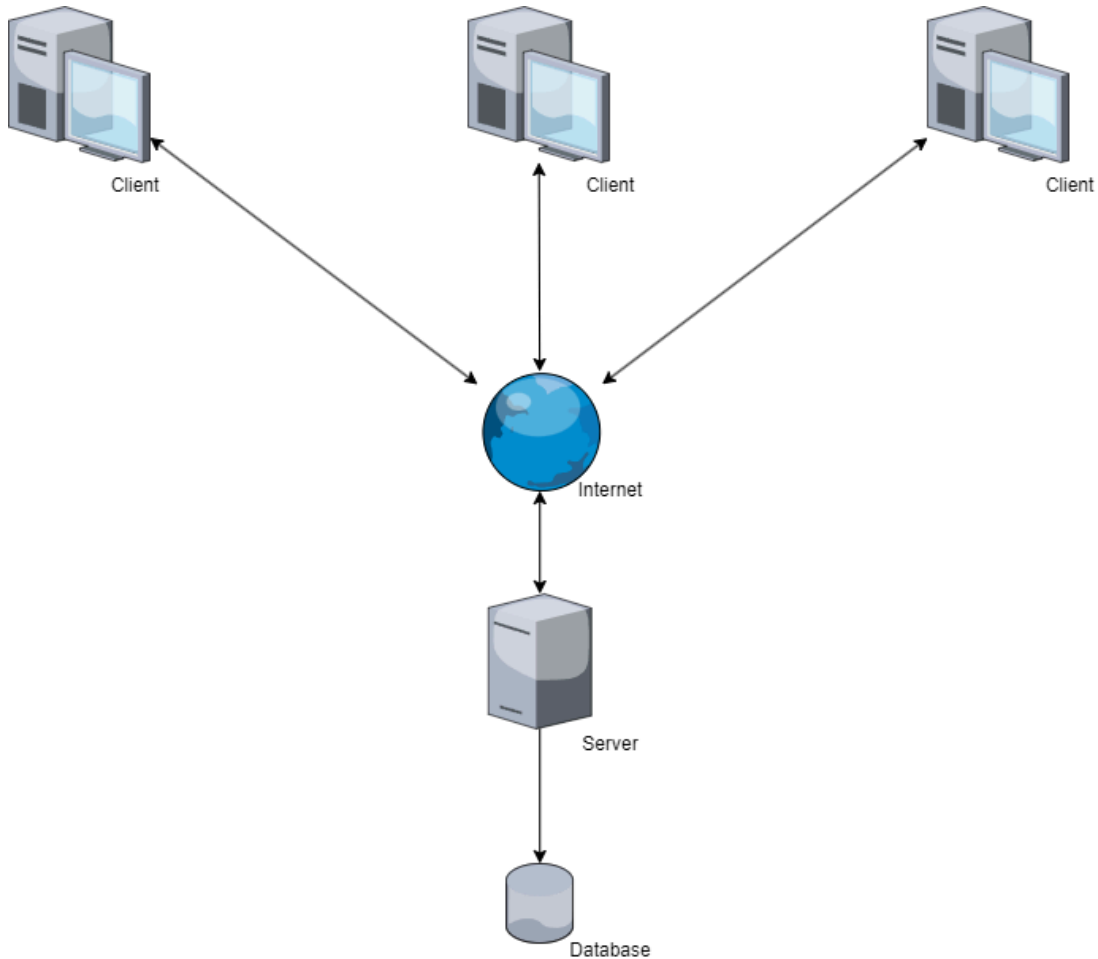
- ממשק משתמש – מסך ראשי הכולל הצגת הלקוחות
- שאילת לקוחות – שליפת המשתמשים מהשרת על ידי שליחת בקשה
- שליחת הבקשה לשרת - (פרוטוקול התעבורה הכולל הצפנה ופענוח)
- קבלת תשובה מהשרת והצגת הנתונים

אובייקטים נחוצים: ממשק משתמש, הצפנה/פענוח, תקשורת, מערכת קבצים בשרת, גישה למסד הנתונים.

ארכיטקטורה

תיאור הארכיטקטורה של המערכת המוצעת

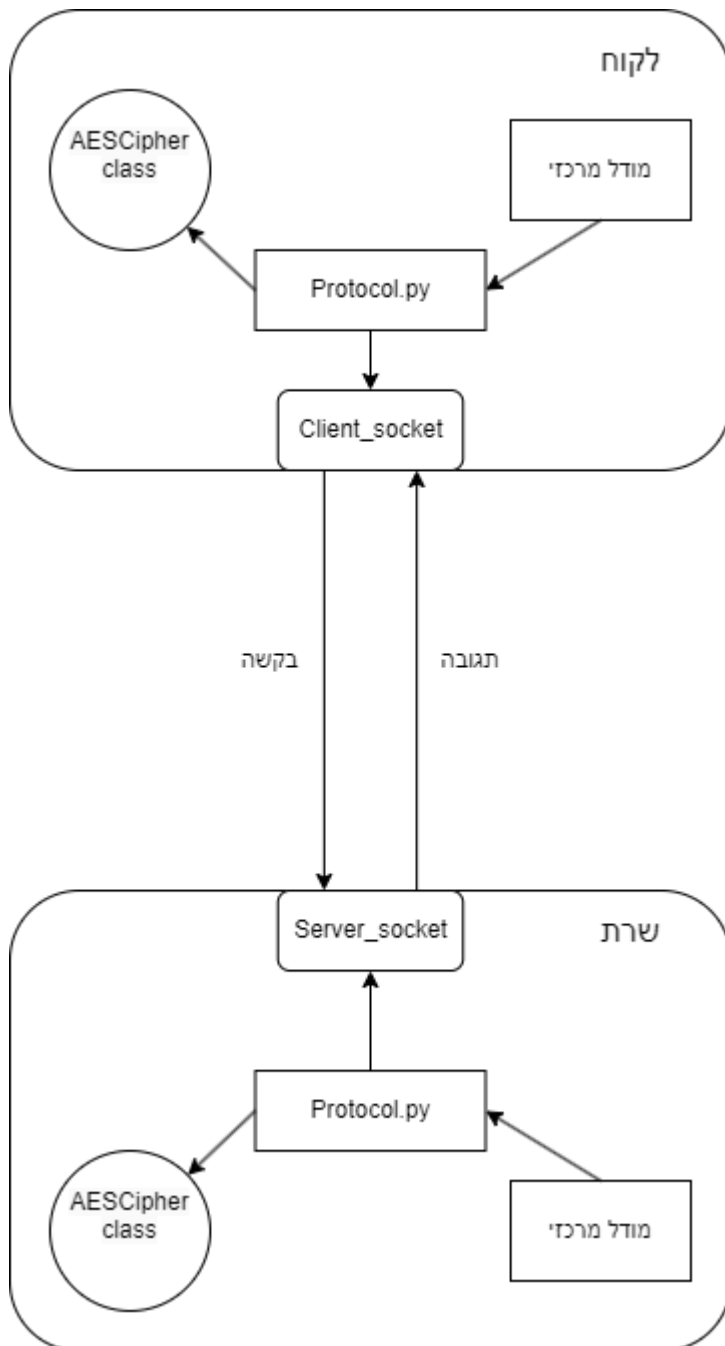
שרטוט כללי של המערכת-



בשרטוט מוצגים שלושה לקוחות המתחברים לשרת, אך בפועל אין מגבלה על מספר המשתמשים שיכולים להתחבר.

כל לקוח זקוק למחשב עם מערכת הפעלה **Windows**, וכן למקלדת ועכבר, כדי להתחבר לשרת. התקשורת בין הלקוח לשרת מתבצעת באמצעות פרוטוקול ייעודי. השרת ניגש למסד הנתונים באמצעות **SQLite**.

פירוט תקשורת שרת - לקוח



הדיאגרמה מתארת את המערכת שבה הלקוח והשרת מתקשרים ביניהם בצורה מאובטחת באמצעות הצפנת AES.

בצד הלקוח, המודול המרכזי כולל `client.py` ו-`client_admin.py`, והם משתמשים ב-`Protocol.py` שאחראי על התקשורת. הפרוטוקול משתמש במחלקת `AESCipher` להצפנה והחיבור לשרת נעשה דרך `Client_socket`.

בצד השרת, המודול המרכזי כולל `Server.py` ו-`server_admin.py`, גם הם משתמשים ב-`Protocol.py` ומחלקת `AESCipher` להעברת התקשורת, הצפנתו ופענוח. החיבור ללקוח נעשה דרך `Server_socket`.

כאשר הלקוח שולח בקשה, היא מוצפנת ונשלחת לשרת דרך `Client_socket`. השרת מקבל את הבקשה דרך `Server_socket`, מפענח אותה, מעבד את הבקשה, ושולח תגובה מוצפנת חזרה ללקוח. הלקוח מקבל את התגובה המוצפנת, מפענח אותה ומציג את המידע. כך נשמרת התקשורת מאובטחת.

פירוט מודולים מרכזיים בשרת

הדיאגרמה מציגה את צד השרת בכלליות על ידי שימוש במודולים. מודול התקשורת-הוסבר בעמוד הקודם.

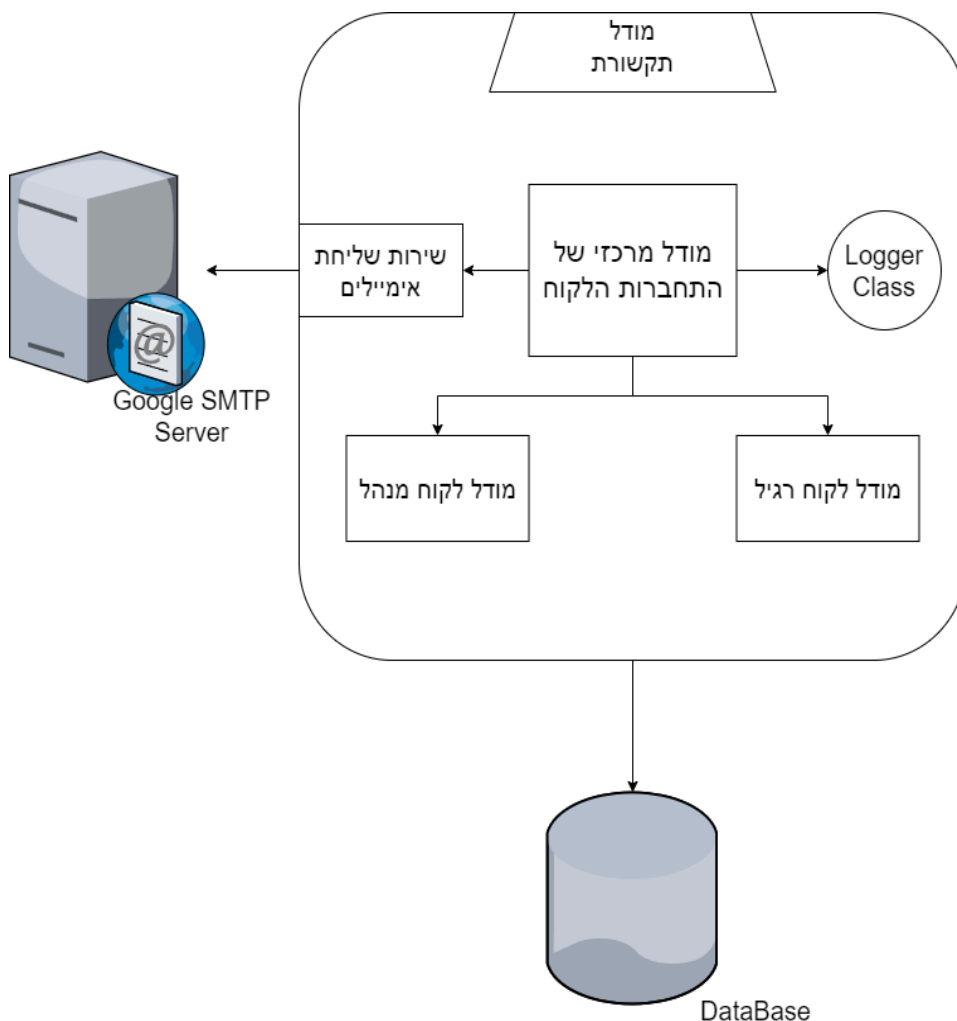
המודול המרכזי- כולל את הקובץ `server.py`, אחראי על ניהול התחברויות הלקוחות.

מודול לקוח רגיל- אחראי על ניהול הלקוח לאחר התחברותו וכולל את הקובץ `server.py`.

מודול לקוח מנהל- אחראי על ניהול לקוח מנהל לאחר התחברותו וכולל את הקובץ `server_admin`.

Logger Class- משתמש בקובץ `logger.py` לניהול ורישום פעולות המערכת.

בנוסף השרת עושה שימוש בשירות שליחת אימיילים דרך שרת `Google SMTP` ושומר את מידע הלקוחות במסד נתונים



פירוט מודולים מרכזיים בלקוח

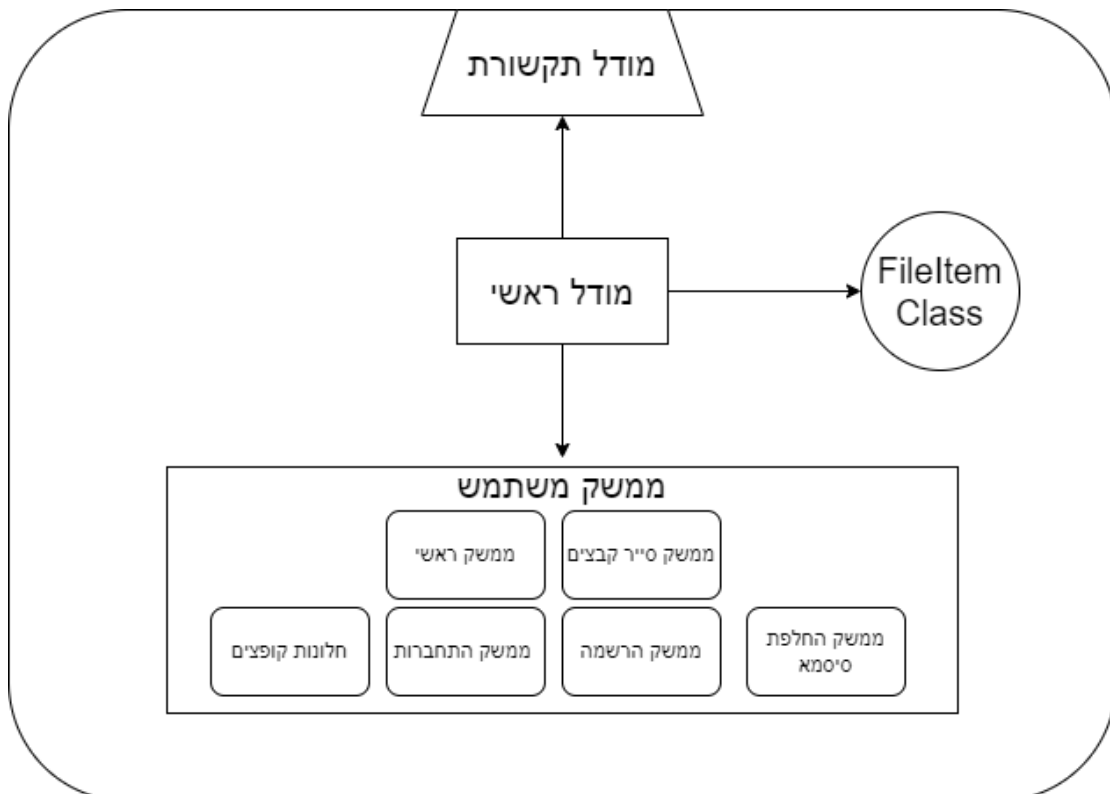
הדיאגרמה מציגה את צד הלקוח בכלליות על ידי שימוש במודולים.

מודול התקשורת-הוסבר מקודם.

המודול המרכזי- כולל את הקובץ `client.py`, `client_admin.py` אחראי על הניהול הכולל.

ממשק משתמש- אחראי על הממשק הגרפי של כל החלונות במהלך ההתחברות ולאחר מכן.

FileItem Class- נמצא בקובץ `Client.py` ומשמש על מנת לקצר את השמות קבצים הארוכים.



תיאור הטכנולוגיה הרלוונטית

- בחרתי להשתמש ב- **Python** בשביל לכתוב את הקוד **לשרת והלקוח** מכיוון שיש לי הכרות רבה עם השפה מהשנתיים האחרונות בבית הספר, בנוסף היא שפה עילית עם הרבה ספריות חיצוניות שעזרו לי בפרויקט ובמיוחד ברשתות. לכתובת הקוד השתמשתי בסביבת הפיתוח **Pycharm**, הסביבה מאוד נוחה לשימוש ואינואטיבית עם עזרים כמו ניפוי באגים
- **לבסיס הנתונים** השתמשתי ב- **SQLite Studio 3** בשל קלות השימוש והספרייה הזמינה לפיתוח. הממשק הנוח מאפשר יצירה, עריכה, מחיקה וניהול של טבלאות. בבסיס הנתונים עשיתי שימוש באלגוריתם **sha256** על מנת לגבב את סיסמאות המשתמשים וליצור שכבת אבטחה נוספת. האלגוריתם הוא פונקציה קריפטוגרפית חד כיוונית המפיק פלט באורך קבוע.
- **מערכת ההפעלה** שהשתמשתי בה היא **windows**. השתמשתי ב**תהליכונים** (**Threads**) בצד השרת בשביל לטפל במספר לקוחות במקביל, בנוסף השתמשתי בסייר הקבצים בצד הלקוח.
- **לתקשורת** בין הלקוח לשרת בחרתי להשתמש ב**סוקטים** בשיטת **TCP**, עקב היכולת לשלוח נתונים בצורה מהימנה ומאובטחת ברשת. כאשר משתמשים בתקשורת על פי פרוטוקול **TCP**, המידע מועבר בצורה יציבה ובטוחה, והמקבל יכול לוודא כי כל הנתונים התקבלו בהצלחה.
- השתמשתי ב**AES** ו**RSA** כדי להבטיח שהתעבורה בין הלקוח לשרת תישאר פרטית ובטוחה. השתמשתי ב**RSA** בהתחברות הראשונית על מנת להעביר את המפתח של ה**AES**. עשיתי שימוש בשני האלגוריתמים על מנת ליצור שכבת אבטחה נוספת וחזקה נגד התקפות.

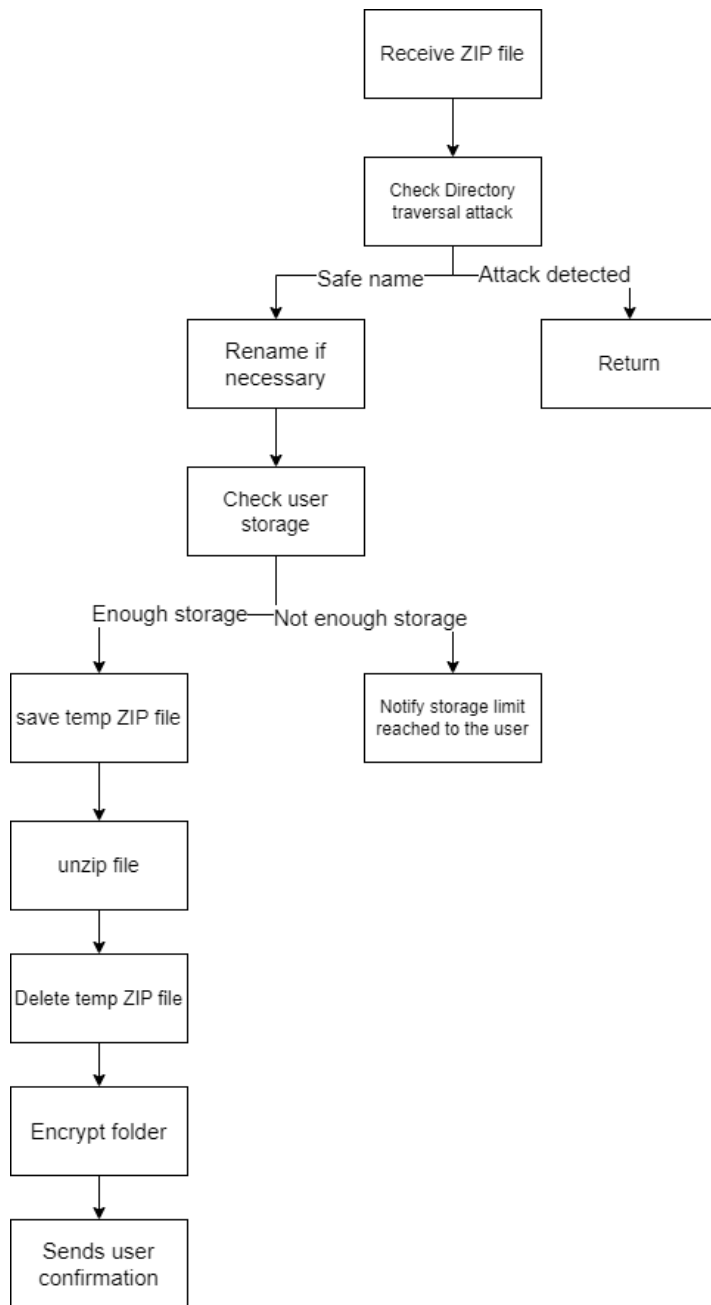
תיאור אלגוריתמים מרכזיים בפרויקט:

קבלת תיקייה (שרת)

הבעיה שהאלגוריתם פותר-

האלגוריתם נועד לפתור בעיה של קבלת תיקייה מהלקוח, הוא עושה זאת בעזרת קבצי ZIP. כאשר הוא מקבל תיקייה הוא צריך לבדוק את תקינותם, ניהול מקום האחסון הזמין למשתמשים, פריקת התוכן שלהם והצפנתו בצורה מאובטחת. בנוסף, יש להתמודד עם מתקפות אפשריות כמו Directory Traversal ולהבטיח שהקבצים מאוחסנים בשם ייחודי למקרה שקיימים תיקיות עם אותו שם.

הסבר הדיאגרמה-



האלגוריתם מתחיל בקבלת קובץ ZIP מהמשתמש, הכולל את שם הקובץ ואת תוכנו. תחילה מתבצעת בדיקה לזיהוי Directory Traversal Attack על ידי חיפוש ".." בשם הקובץ, כדי למנוע גישה לקבצים מחוץ לתיקייה המיועדת. אם זוהתה מתקפה, התהליך נפסק ומחזיר return. אם שם הקובץ בטוח, האלגוריתם ממשיך, מוסיף את שם המשתמש לנתיב התיקייה, ומוודא האם שם הקובץ כבר קיים. אם כן, שם הקובץ משתנה באופן ייחודי על ידי הוספת מספר סידורי כדי להבטיח ייחודיות.

לאחר מכן, האלגוריתם בודק אם יש מספיק מקום אחסון למשתמש על ידי חישוב גודל התיקייה של המשתמש. אם אין מספיק מקום, נשלחת הודעה למשתמש שהמגבלה הגיעה, והתהליך נפסק. אם יש מספיק מקום, הקובץ נשמר כקובץ זמני. לאחר מכן, תוכן הקובץ הזמני מועתק לתיקייה חדשה(על ידי הסרת הסיומת ".zip") באופן רקורסיבי (תיקייה בתוך תיקייה). הקובץ הזמני נמחק לאחר מכן.

בשלב הבא, הקבצים שבתוך התיקייה מוצפנים באמצעות מפתח ההצפנה שסופק. פונקציה זו מבצעת הצפנה על כל הקבצים בתיקייה באמצעות ספריית AES, ושומרת את וקטור ההצפנה (נוצר IV ייעודי לכל קובץ) בקובץ JSON לצורך שחזור עתידי. לבסוף, נשלחת למשתמש הודעת אישור שהקובץ התקבל והוצפן בהצלחה.

הורדת קבצים/תיקיות (לקוח)

הבעיה שהאלגוריתם פותר-

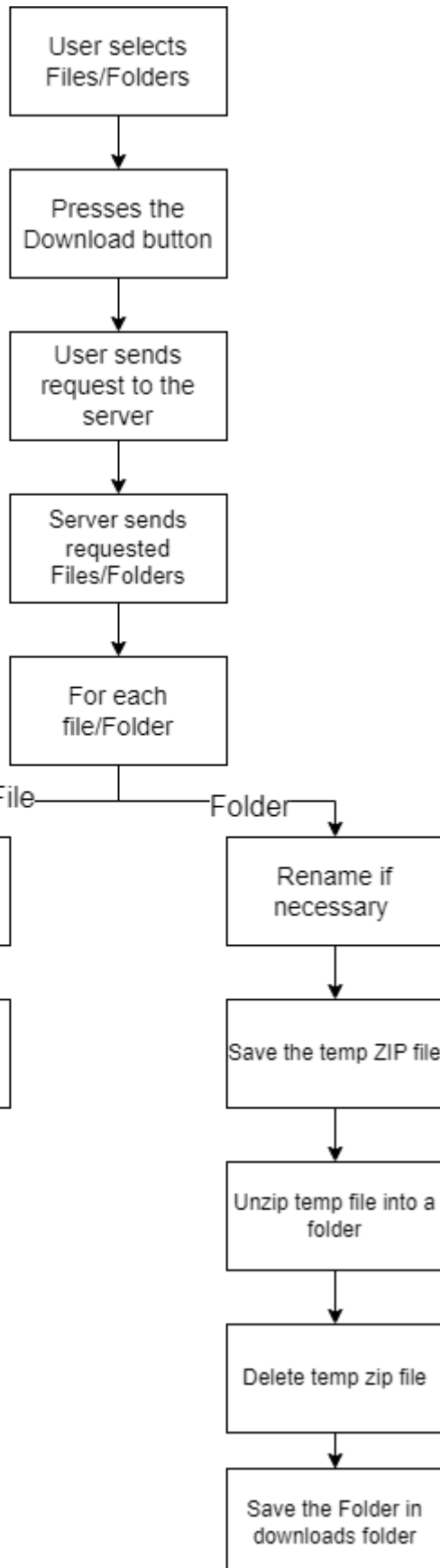
הבעיה שהאלגוריתם פותר היא הורדה מסודרת ויעילה של קבצים ותיקיות מהשרת לתיקיית ההורדות של המשתמש, תוך טיפול במצבים שבהם קיימים קבצים או תיקיות עם שמות זהים. המטרה היא לאפשר למשתמש לבחור קבצים ותיקיות מתוך ממשק, לשלוח בקשת הורדה לשרת, לקבל את הקבצים/תיקיות, לשמור אותם בתיקיית ההורדות עם טיפול נכון בשמות הכפולים ולפרוס תיקיות שהורדו כקובצי ZIP.

הסבר הדיאגרמה-

תחילה, המשתמש בוחר קבצים ותיקיות בממשק ומפעיל את תהליך ההורדה בלחיצת כפתור. הפונקציה יוצרת רשימת קבצים שנבחרו, מצרפת את הנתיב שלהם ושולחת בקשה לשרת עם רשימת הקבצים והנתיבים המלאים. השרת מקבל את הבקשה ושולח חזרה את הקבצים או התיקיות שנבחרו. האלגוריתם מקבל את התגובה מהשרת ומעבד כל קובץ או תיקיה שנשלחו.

כאשר מדובר בקובץ, האלגוריתם בודק אם קובץ עם אותו שם כבר קיים בתיקיית ההורדות. אם כן, שם הקובץ משתנה באופן ייחודי על ידי הוספת מספר סידורי כדי להבטיח ייחודיות ושומר את הקובץ בתיקיית ההורדות.

כאשר מדובר בתיקיה (הלקוח מקבל קובץ zip), האלגוריתם פועל באופן דומה לשינוי שם התיקיה אם היא כבר קיימת, שומר את הקובץ באופן זמני בתיקיית ההורדות, מעתיק את ה-ZIP לתוך תיקיה רגילה ולאחר מכן מוחק את קובץ ה-ZIP הזמני. בסיום התהליך, הפונקציה refresh מתבצעת כדי לעדכן את הממשק ולהציג את הקבצים והתיקיות שהורדו.

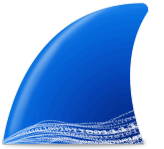


תיאור סביבת הפיתוח

בחרתי להשתמש ב-PyCharm לכתובת הפרויקט מכיוון שהוא מספק סביבת פיתוח משולבת (IDE) מתקדמת במיוחד לשפת Python. פייצ'ארם כולל עריכת קוד מתקדמת עם השלמת קוד אוטומטית ותיקון שגיאות בזמן אמת, מה שמאפשר לי לכתוב קוד בצורה מהירה ויעילה יותר. בנוסף, כלי ה-debugging החזק שלו מאפשר לי לעקוב אחר שגיאות ולבדוק את הביצועים של הקוד בצורה נוחה.



בחרתי להשתמש ב-Wireshark עבור הפרויקט מכיוון שהוא כלי שימושי להספנת תעבורה. התוכנה מאפשרת לי לבחון ולנתח את תעבורת הרשת בין הלקוח לשרת בצורה מעמיקה ובכך להבטיח שההצפנה תקינה. בנוסף בעזרת הכלי, אני יכול לזהות בעיות בתעבורה, לגלות תקלות ולנטר את הביצועים של הרשת בצורה מדויקת.



SQLite Studio 3 היא תוכנה לניהול מבני נתונים. באמצעות התוכנה ניתן לצפות בטבלאות, לערוך נתונים ולהריץ שאילתות בקלות. SQLite Studio 3 מספקת ממשק גרפי נוח שמקל על ניהול מסד הנתונים, ומאפשרת לבצע פעולות מורכבות בצורה אינטואיטיבית.



תיאור פרוטוקול התקשורת

בשביל החלפת הודעות בין השרת ללקוח השתמשתי בפרוטוקול ייעודי שיצרתי שכולל מספר שלבים. בתהליך השליחה, ההודעה מתחילה כמילון (dictionary) אשר נארז לפורמט קומפקטי באמצעות msgpack. לאחר מכן, ההודעה הארוזה מוצפנת באמצעות אלגוריתם AES במצב CBC, שבו נעשה שימוש במפתח (key) ובוקטור אתחול (iv) כדי להבטיח את סודיות המידע. האורך של ההודעה המוצפנת מצורף לראש ההודעה בפורמט קבוע של 10 ספרות, כדי לאפשר לצד המקבל לדעת את גודל ההודעה ולקרוא אותה במלואה.

בתחילת החיבור בין השרת ללקוח, מפתח ה-AES מועבר בצורה מאובטחת באמצעות אלגוריתם RSA. השימוש ב-RSA, שהוא אלגוריתם הצפנה אסימטרי, מאפשר להעביר את מפתח ה-AES בצורה מוצפנת ובטוחה. הלקוח מצפין את מפתח ה-AES עם המפתח הציבורי של השרת ושולח אותו לשרת. השרת, המשתמש במפתח הפרטי שלו, מפענח את המפתח שהתקבל.

כלל ההודעות העוברות בצורת מילון-* אם לא צויין אחרת הערך הוא string

רוב התגובות מהשרת נראות כך:

מפתח:	msg	command
ערך:	משתנה	response

התחברות מצד הלקוח:

command	email	username	password	מפתח:
login	קלט משתמש	קלט משתמש	קלט משתמש	ערך:

הרשמה מצד הלקוח:

command	email	username	password	מפתח:
signup	קלט משתמש	קלט משתמש	קלט משתמש	ערך:

שליחת קוד אימות מצד לקוח:

command	code	מפתח:
signup_code / forgot_password_code	קלט משתמש	ערך:

שינוי סיסמא מצד לקוח:

command	email	מפתח:
forgot_password	קלט משתמש	ערך:

יצירת תיקייה מצד לקוח:

command	folder_name	מפתח:
create_folder	קלט משתמש	ערך:

קבלת רשימת הקבצים הקיימים בתיקת הלקוח:

command	path	page	rows	cols	מפתח:
refresh	על פי התיקיה שהמשתמש מבקש	משתנה	משתנה	משתנה	ערך:

מחיקת קבצים/תיקיות מצד לקוח:

command	files	מפתח:
delete	קלט משתמש מיוצג על ידי רשימה	ערך:

הורדת קבצים/תיקיות מצד הלקוח:

command	files	מפתח:
download	קלט משתמש שמיוצג על ידי רשימה	ערך:

תגובת השרת להורדת קבצים:

command	files	מפתח:
response	רשימה של כל הקבצים. לכל קובץ יש: שם, סוג ותוכן.	ערך:

העלאת קובץ מצד הלקוח:

command	file_name	file_type	file_content	מפתח:
upload_file	נקבע על פי הקובץ ששלח	נקבע על פי סוג הקובץ	תוכן הקובץ (מיוצג בבייטים)	ערך:

העלאת תיקייה מצד הלקוח:

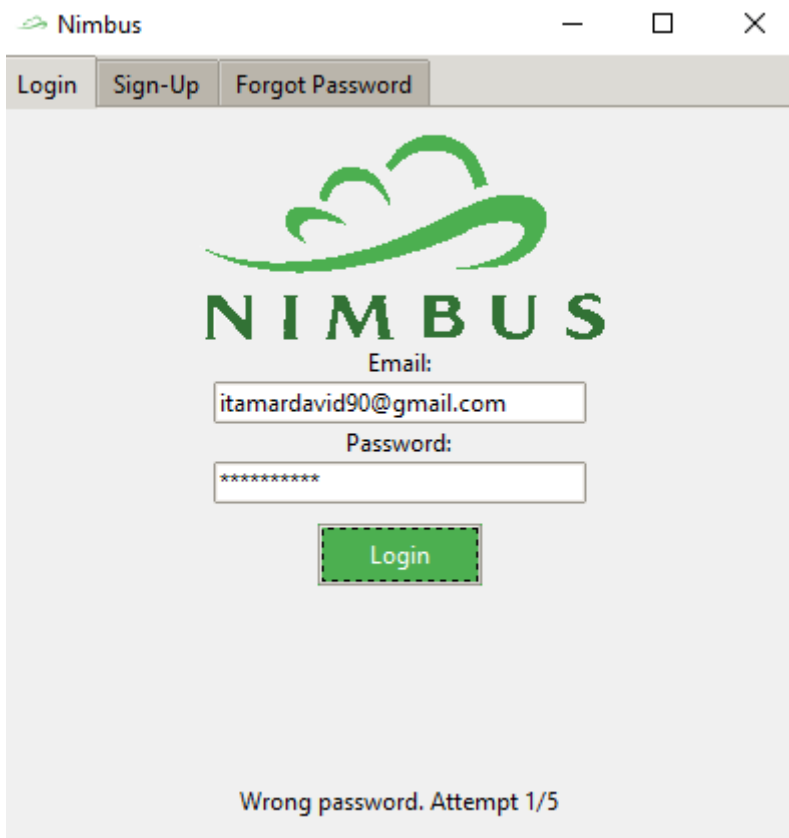
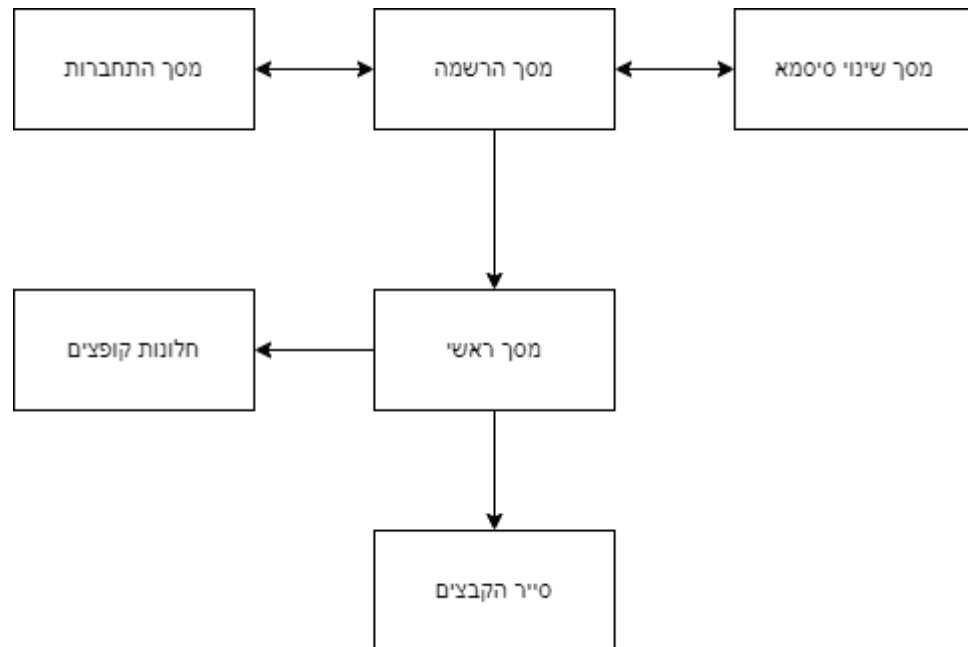
מפתח:	folder_content	folder_name	command
ערך:	תוכן הקובץ (מיוצג בבייטים)	נקבע על פי התיקייה ששלח	upload_folder

יציאת המשתמש מהתוכנה:

מפתח:	command
ערך:	exit

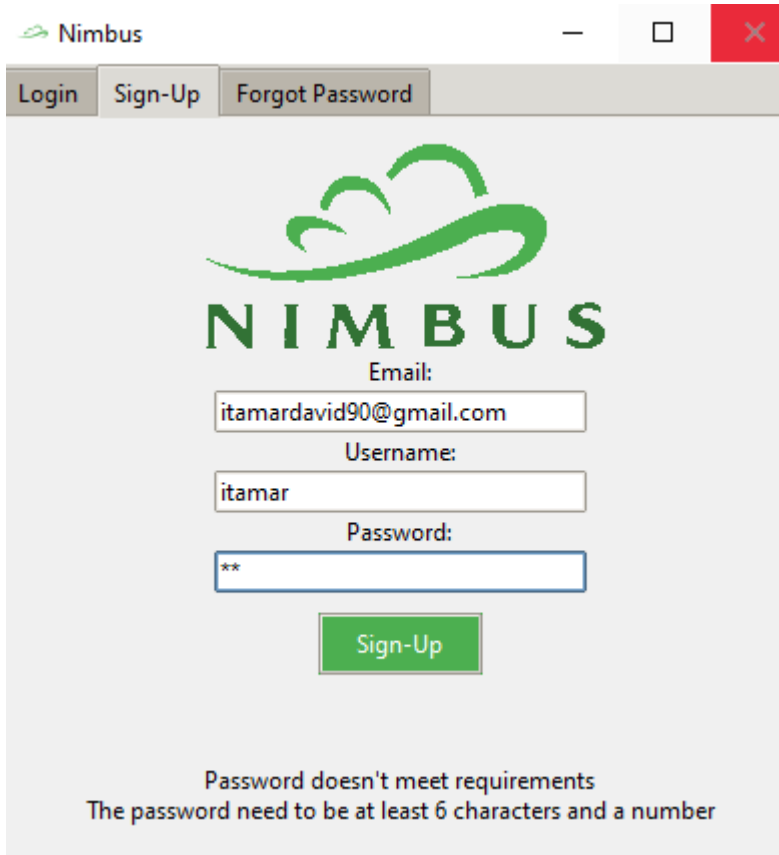
תיאור מסכי המערכת

תרשים מסכים



מסך ההתחברות- המסך הראשוני כאשר פותחים את התוכנה, משתמשים קיימים יכולים להזין את האימייל והסיסמא שלהם (הסיסמא מוסתרת על ידי כוכביות).

בנוסף הם יכולים לבחור ליצור משתמש חדש או לשנות את סיסמתם. כאשר הלקוח לוחץ על כפתור ה"login" מודפסת הודעה מהשרת כמו "סיסמא לא נכונה" "אימייל לא קיים במערכת"...

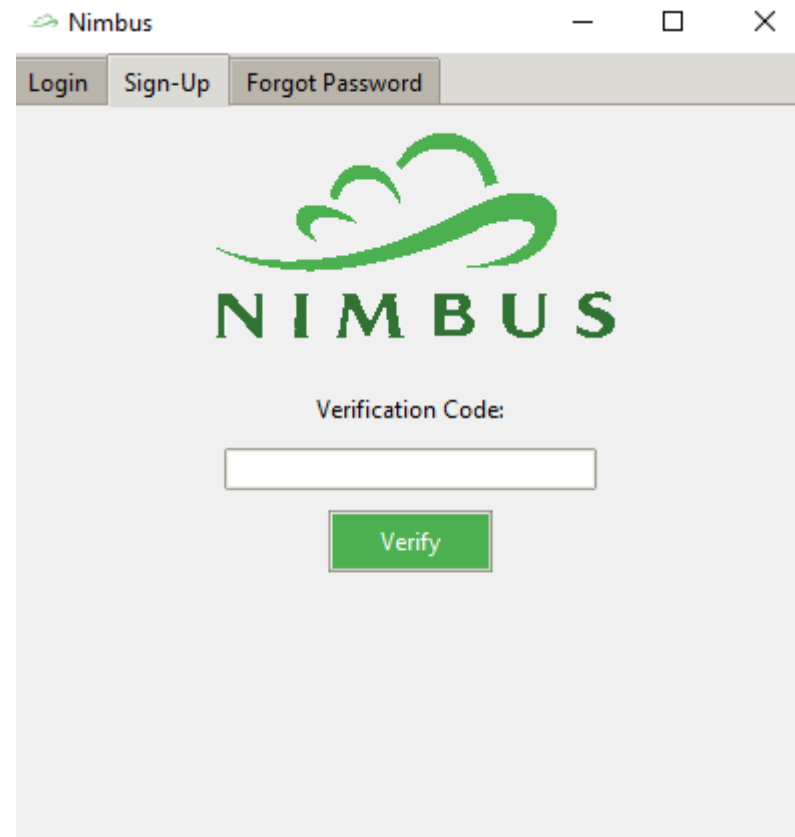


The image shows a web browser window with the Nimbus logo and navigation tabs: Login, Sign-Up, and Forgot Password. The Sign-Up form is active, showing fields for Email (itamardavid90@gmail.com), Username (itamar), and Password (**). A green Sign-Up button is below the fields. At the bottom, a message states: "Password doesn't meet requirements. The password need to be at least 6 characters and a number".

מסך ההרשמה-

משתמשים חדשים יכולים ליצור משתמש על ידי הזנת הפרטים האישיים שלהם. (גם כאן הסיסמא מוסתרת על ידי כוכביות)

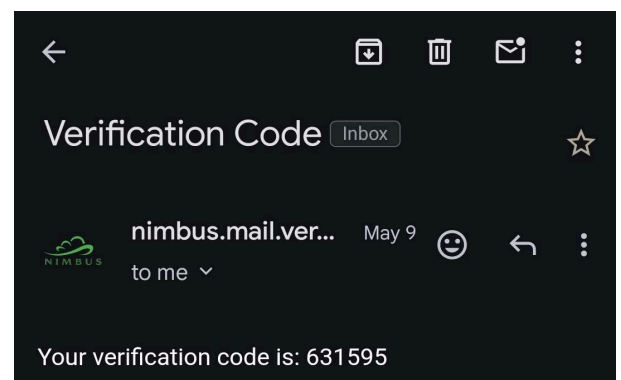
כאשר הלקוח לוחץ על כפתור ה"Sign-Up" מודפסת הודעה מהשרת כמו "סיסמא לא עומדת בתנאים" "אימייל קיים כבר במערכת"...



The image shows a web browser window with the Nimbus logo and navigation tabs: Login, Sign-Up, and Forgot Password. The Sign-Up form is active, showing a field for Verification Code. A green Verify button is below the field.

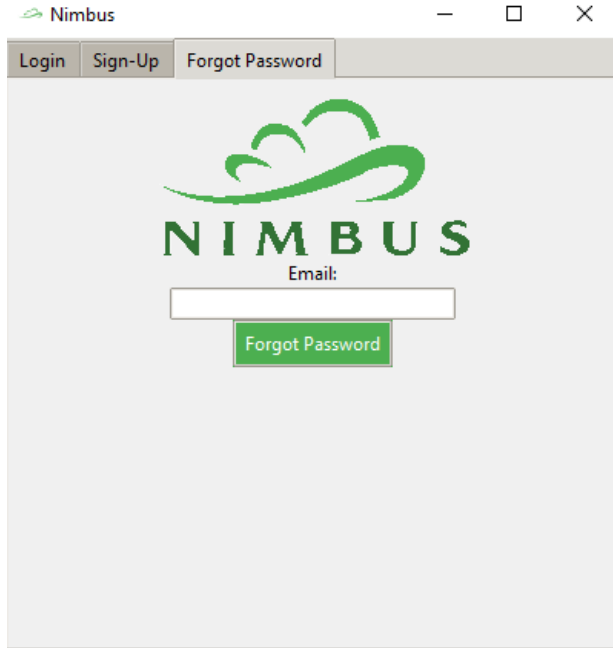
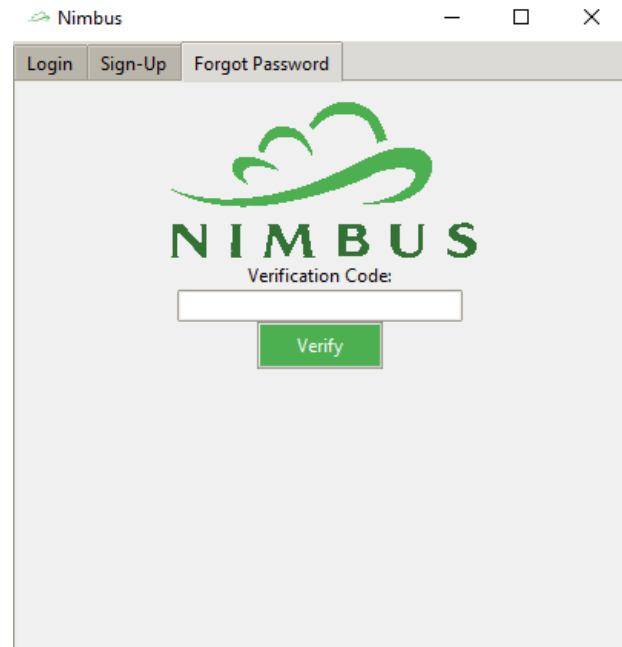
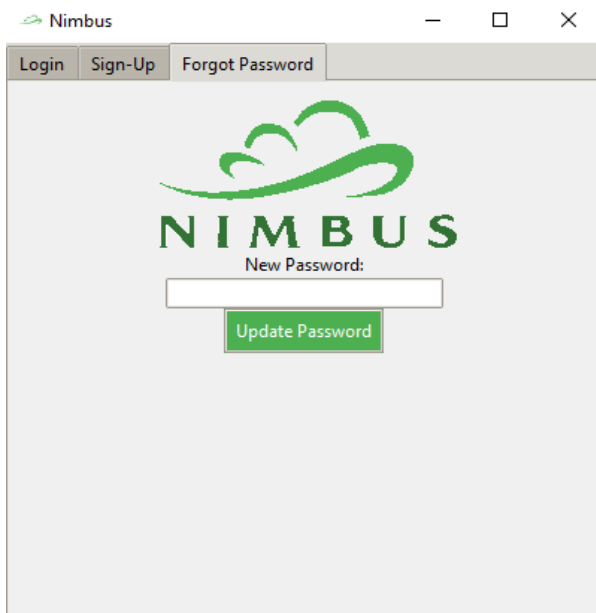
מסך ההרשמה - המשך

אחרי שהזינו את הפרטים האישיים עם סיסמא תקינה ואימייל תקין הם מועברים לחלון השני. המשתמש צריך לבדוק את האימייל שלו בשביל הקוד אימות שנשלח ולהזין אותו.



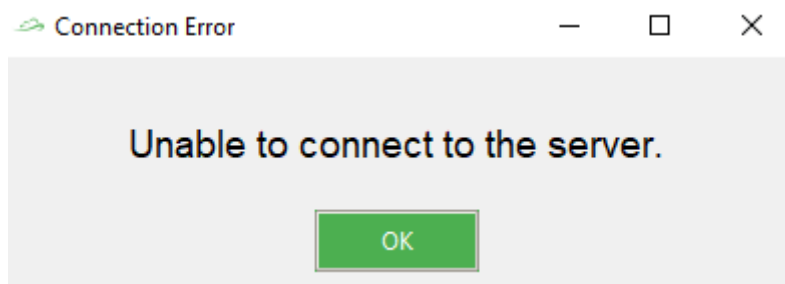
מסכי שכחתי סיסמא-

כאשר לקוח שכח את הסיסמא שלו הוא יכול לשנות את הסיסמא בעזרת החלון הזה. המשתמש מזין את האימייל שלו, לאחר מכן הוא מגיע לחלון אימות קוד ולאחר מכן הוא כותב סיסמא חדשה.

חלונות קופצים-

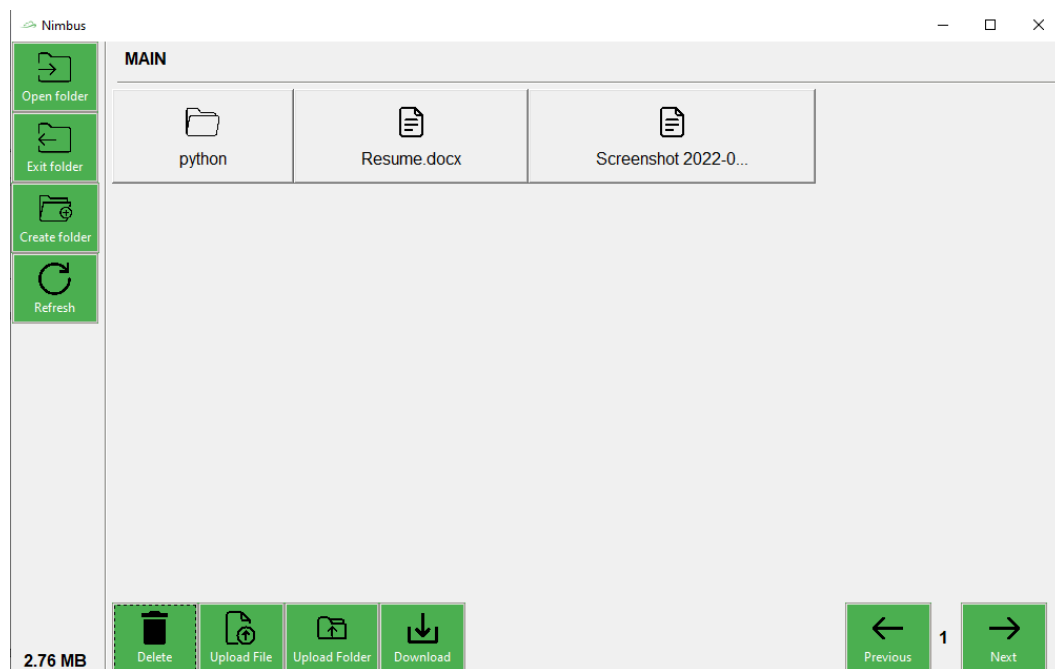
בכל שלב לקוח יכול להתקל בחלונות כאשר יש תקלה בצד השרת, לדוגמא כאשר השרת כבוי והלקוח מנסה להתחבר מוצג לו החלון הבא



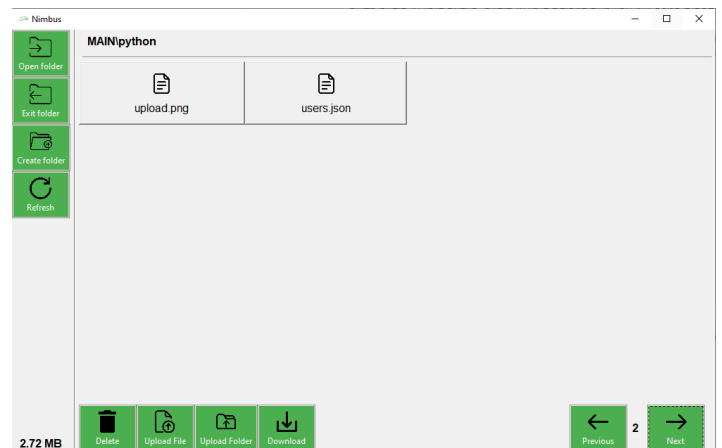
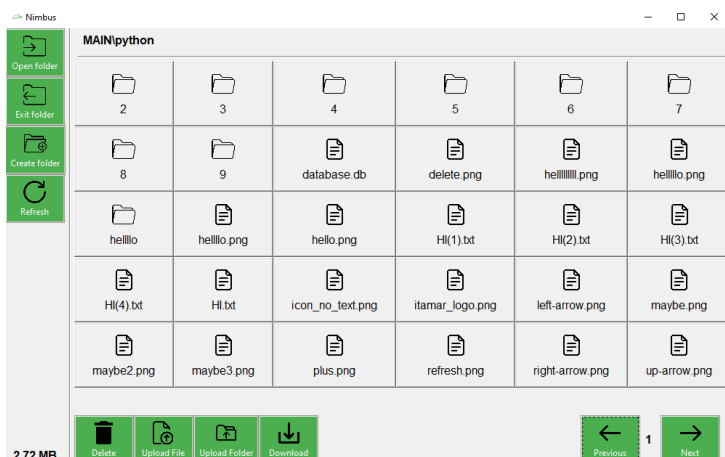
מסך ראשי לאחר התחברות-

במרכז המסך מוצגים הקבצים והתיקיות שהמשתמש שומר, כגון תיקייה בשם "python", קובץ מסמך בשם "Resume.docx" וקובץ תמונה בשם "Screenshot 2022-0..." (כאשר יש שם ארוך מידי הוא מקוצר למטרות הצגה). אזור זה משמש להצגת הקבצים והתיקיות השמורים בתיקייה הנוכחית. בצד שמאל ישנן אפשרויות כמו פתיחת תיקייה, יציאה מתיקייה נוכחית, יצירת תיקייה חדשה ורענון התצוגה.

בתחתית המסך ישנן אפשרויות למחיקת קובץ או תיקייה, העלאת קובץ או תיקייה חדשה והורדת קובץ או תיקייה נבחרת. בתחתית יש גם כפתורי ניווט בין עמודים, עם כפתורי "Previous" ו-"Next" למעבר בין עמודים ומספר העמוד הנוכחי מוצג באמצע. בפינה השמאלית התחתונה מוצג השטח הנוכחי של המשתמש, במקרה הזה 2.76 MB.

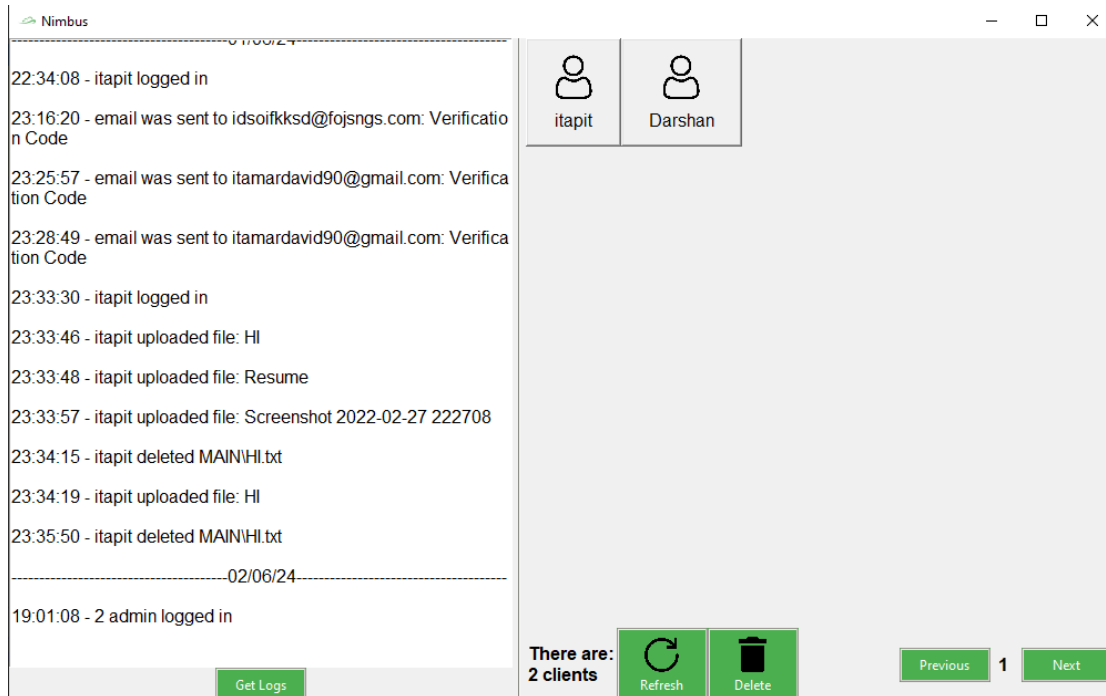


במקרה שבו ללקוח יש מספר רב של קבצים להציג והם לא נכנסים בעמוד הראשון הוא יכול לגלל לעמוד הבא,



מסך ראשי של המנהל- המנהל יכול לראות את הלקוחות ואת קובץ הלוגים.






יש לו אפשרות ללחוץ על המשתמשים ולשלוח בקשה לשרת למחוק אותם, בנוסף הוא יכול לגלל בקובץ הלוגים למעלה, גם למנהל יש מספר עמודים במקרה שיש לקוחות רבים.



The screenshot displays the Nimbus web application interface. On the left, a log window titled "Nimbus" shows a list of activities with timestamps and descriptions, such as "itapit logged in" and "email was sent to idsoifkksd@fojsngs.com: Verification Code". Below the log, a "Get Logs" button is visible. On the right, a panel shows a list of clients. At the top, there are two client cards for "itapit" and "Darshan". Below this, a message states "There are: 2 clients". At the bottom of the client list, there are buttons for "Refresh" (a circular arrow icon) and "Delete" (a trash can icon). To the right of these buttons, there are navigation controls: "Previous", a page number "1", and "Next".

תיאור מבני נתונים

בצד השרת אני מאחסן את נתוני המשתמשים כ- **DataBase** ויש לי טבלה אחת ששמה **Users**, יצרתי את הטבלה עם התוכנה **sqlite studio 3**. השרת ניגש לטבלה כאשר משתמש נמצא בשלבי ההתחברות לשרת.

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	Default value
1	username	TEXT								NULL
2	password	TEXT								NULL
3	email	TEXT								NULL
4	salt_password	BLOB								NULL
5	salt_key	BLOB								NULL
6	admin	INTEGER								NULL

username	password	email	salt_password	salt_key	admin
1 itapit	2ddd10cfcefe2ea74b2821fe204d7875792e8d998dfb81674b6ebbd9e96eb602	itamardavid90@gmail.com	~L?u?k*IJ?Q?M?8?ef?M?~	?Dr0c??P?Dy?<?φ?□??	0
2	441f1bae502614852aa1d0935cb9e142fc7bedc8d1d745d435a8296e143618c6	nimbus.mail.ver@gmail.com	??5?\.2s-r??	o/t3?^_06?K8□yy?n? X>??b??h	1
3 Darshan	21b7e335d19926d37a80e5bd6b8d348c13ead9e714e4b826e3ee2cb3726f8ba3	asaf.darsh@gmail.com	??bAEU?n)?□?6%--?y	??5?t	0

שדה	admin	salt_key	salt_password	email	password	username
סוג תוכן	int	blob	blob	text	text	text
תיאור	0- משתמש רגיל 1- מנהל	להצפנת salt קבצים	לסימא salt	אימייל המשתמש	סימט המשתמש שמור לאחר hash בצורת hexa	שם המשתמש של הלקוח
דוגמא	0	?Dr0c?? ?o??P?? y??<?φ? ???	L?u?*k*IJ? ??Q?M??8 ??ef?M? ~?	itamardavi d90@gmail .com	2ddd10cfcefe2e a74b2821fe204 d7875792e8d99 8dfb81674b6eb bd9e96eb602	itapit
אורך	אורך משוער: מספר אחד	אורך משוער: 32 בייטים	אורך משוער: 32 בייטים	בלתי מוגבל	בלתי מוגבל	בלתי מוגבל

הסבר נוסף על משתנים:

- כאשר משתמש חדש נוצר השרת מייצר לו salt_password ו salt_key ושומר בטבלה, הם לא משתנים בשום שלב.
- השרת משתמש ב salt_password בשביל לגבב את הסימא ולשמור אותה בצורה מאובטחת.
- השרת משתמש ב salt_key בשביל לייצר מפתח AES להצפנת הקבצים של הלקוח.
- Salt- ערך בייטים רנדומלי שמוגרל בשביל להבטיח רנדומליות נוספת לגיבוב הסימא עם שיטת sha256. אני אסביר על כך בפירוט בפרק חולשות ואיומים.
- האימייל, ושם המשתמש הם ייחודיים לכל לקוח ולא משתנים.
- סימט המשתמש משתנה במקרה שהלקוח משנה את סימטו.

לשרת יש תיקייה ראשית בשם STORED_DATA ובה הוא שומר את קבצי הלקוחות. עבור כל משתמש, הוא יוצר תיקייה בתוך התיקייה הראשית עם שם המשתמש הייחודי שלו, ובתוכה מאוחסנים קבצי הלקוח.

PC > Local Disk (C:) > Users > UserStud > Drive > STORED_DATA		
Name	Date modified	Type
asaf	02/06/2024 09:35	File folder
darshan	02/06/2024 09:35	File folder
itapit	02/06/2024 09:22	File folder

לכל לקוח בתוך התיקייה שלו יש קובץ json ששומר את ערכי ה-IV לכל קובץ מוצפן, אני ארחיב על כך יותר בהמשך של ארכיטקטורה בהצפנה.

בקובץ json יש לי מילון- המפתח הוא שם הקובץ והערך הוא ה-IV הייחודי לאותו קובץ

```
{
  "STORED_DATA\\itapit\\Resume.docx": "Lmtc4Ps54kWDMosw0JRR7w==",
  "STORED_DATA\\itapit\\Screenshot 2022-02-27 222708.png": "LRdvIbqkiX9QvPOqL2Oulg==",
  "STORED_DATA\\itapit\\python\\database.db": "PERI7jOEUiY9GEgiNs4Naw==",
  "STORED_DATA\\itapit\\python\\delete.png": "+YJDawXjUrqBEh5W1kY89g==",
  "STORED_DATA\\itapit\\python\\helllllllll.png": "n2YxoiRQikTTuqba9S1NBg==",
  "STORED_DATA\\itapit\\python\\hellllllo.png": "vmn8dhpLwFvmkbQsUT1G5g==",
  "STORED_DATA\\itapit\\python\\helllllo.png": "dqm7QCaCc7vqpKfdjwcuIQ==",
  "STORED_DATA\\itapit\\python\\hello.png": "yhSHdyzdf7xx1fccmGu3rw==",
  "STORED_DATA\\itapit\\python\\icon_no_text.png": "Thlyomn09OvR+GjahPszwA==",
  "STORED_DATA\\itapit\\python\\itamar_logo.png": "6DdnQNOx2VdUCFyykyBr4g==",
  "STORED_DATA\\itapit\\python\\left-arrow.png": "/37TLy9apglS89lx/5K5og==",
  "STORED_DATA\\itapit\\python\\maybe.png": "998U84vG3TUy3+S1dzhWgg==",
  "STORED_DATA\\itapit\\python\\maybe2.png": "sC3+7B76K3HrVgcZWfEvZg==",
  "STORED_DATA\\itapit\\python\\maybe3.png": "7oFLfquqWmluGao7LFXVRw==",
  "STORED_DATA\\itapit\\python\\plus.png": "N+rvjqxPscVtHgbCYS2INA==",
  "STORED_DATA\\itapit\\python\\refresh.png": "4IGVNETV8z5MUPT666hNdQ==",
  "STORED_DATA\\itapit\\python\\right-arrow.png": "m2Wxzaq4cIkh9JEW1SJBgQ==",
  "STORED_DATA\\itapit\\python\\up-arrow.png": "zG2rpnzIeWtaTLHYsh7rEw==",
  "STORED_DATA\\itapit\\python\\upload.png": "3PC14k8xuD6hEb58xmYF8A==",
  "STORED_DATA\\itapit\\python\\users.json": "XhPVy6KswW4gwBW1FqOp0Q==",
  "STORED_DATA\\itapit\\python\\9\\Screenshot 2022-04-03 225225.png": "EuiZM8uQNbX2L/suhOf9xu==",
  "STORED_DATA\\itapit\\python\\6\\my-2022-hypixel.png": "jcfvOX9VLc2oy+JhTGR0hA=="
}
```

שדה	ערך IV ייחודי	שם הקובץ
דוגמא:	Lmtc4Ps54kWDMosw0JRR7w"	"STORED_DATA\\itapit\\Resume.docx"
אורך:	אורך משוער: 16 בייטים	בלתי מוגבל

בנוסף אני שומר את פעולות המשתמשים וקריסות בצד השרת בקובץ logs לצורך תיעוד בתוספת השעה שהאירוע קרה, הקובץ דומה לקובץ Text (אין לו שדות שונים). כאשר לקוח מנהל מתחבר יש לו גישה לצפייה בקובץ logs

```
2024-05-10 14:33:48 - itapit logged in
2024-05-10 14:33:51 - itapit downloaded file: STORED_DATA\\itapit\\helllllo\\lo
2024-05-10 14:33:51 - error: itapit Error downloading file/folder: [WinErr
2024-05-10 14:34:47 - itapit logged in
2024-05-10 14:34:58 - itapit downloaded file: STORED_DATA\\itapit\\helllllo
```


סקירת חולשות ואיומים (הפרק האהוב עליי 🥰)

SQL Injection

כדי להגן על מסד הנתונים שלי מפני התקפות **SQL Injection**, אני משתמש בשאילתות מוכנים מראש (Prepared Statements) ובאופן ספציפי, אני מפריד בין השאילתה לבין הערכים המוזנים. במקום להכניס את הערכים ישירות לתוך השאילתה כטקסט, אני משתמש במשתנים (placeholders) בתוך השאילתה ומוסיף את הערכים בנפרד.

שיטה זו מונעת מהתוקפים להכניס קוד זדוני כי הערכים המוזנים לא מתפרשים כחלק מהקוד אלא כנתונים בלבד.

דוגמא מהקוד:

```
query = "SELECT admin FROM Users WHERE username = ?;"
cursor.execute(query, (username,))
```

תהליך ההתחברות

בתהליך ההתחברות יש לי מספר דברים שעשיתי על מנת להוסיף שכבות הגנה לפרטי המשתמשים.

כדי להבטיח שהסיסמאות באפליקציה שלי יהיו קשות לניחוש ויעמדו בתנאי האבטחה, קבעתי סטנדרט פשוט אך יעיל: כל סיסמא חייבת לכלול לפחות 6 תווים ולפחות ספרה אחת. תנאי זה נועד למנוע התקפות ניחוש בסיסי של סיסמאות, כך שלא ניתן יהיה לפרוץ לחשבונות עם סיסמאות קלות מדי.

Password doesn't meet requirements
The password need to be at least 6 characters and a number

כדי להגן על סיסמאות המשתמשים שלי, אני משתמש בטכניקה שנקראת **גיבוב (Hashing)** עם אלגוריתם **sha256**. גיבוב הוא תהליך שבו אני מעביר את הסיסמאות דרך פונקציה

מיוחדת שממירה אותן למחרוזת של תווים שאינה ניתנת לפענוח חזרה. בצורה זו, אפילו אם תוקפים מצליחים לגשת למסד הנתונים, הם לא יוכלו לראות את הסיסמאות המקוריות.

בכל פעם שמשתמש נרשם או משנה את הסיסמא שלו, אני מבצע את השלבים הבאים:

1. מקבל את הסיסמא מהמשתמש.
2. מעביר את הסיסמא דרך פונקציית גיבוב שמייצרת מחרוזת גיבוב ייחודית.
3. שומר את מחרוזת הגיבוב במסד הנתונים.

כדי לוודא שאפילו אם יש להם גישה למסד הנתונים הם לא יוכלו לגלות את הסיסמאות, אני גם משתמש בטכניקת "**מלח (Salt)**". מלח הוא ערך אקראי שנוסף לסיסמא לפני הגיבוב, מה שמבטיח שכל סיסמא תייצר מחרוזת גיבוב שונה אפילו אם שתי הסיסמאות זהות.

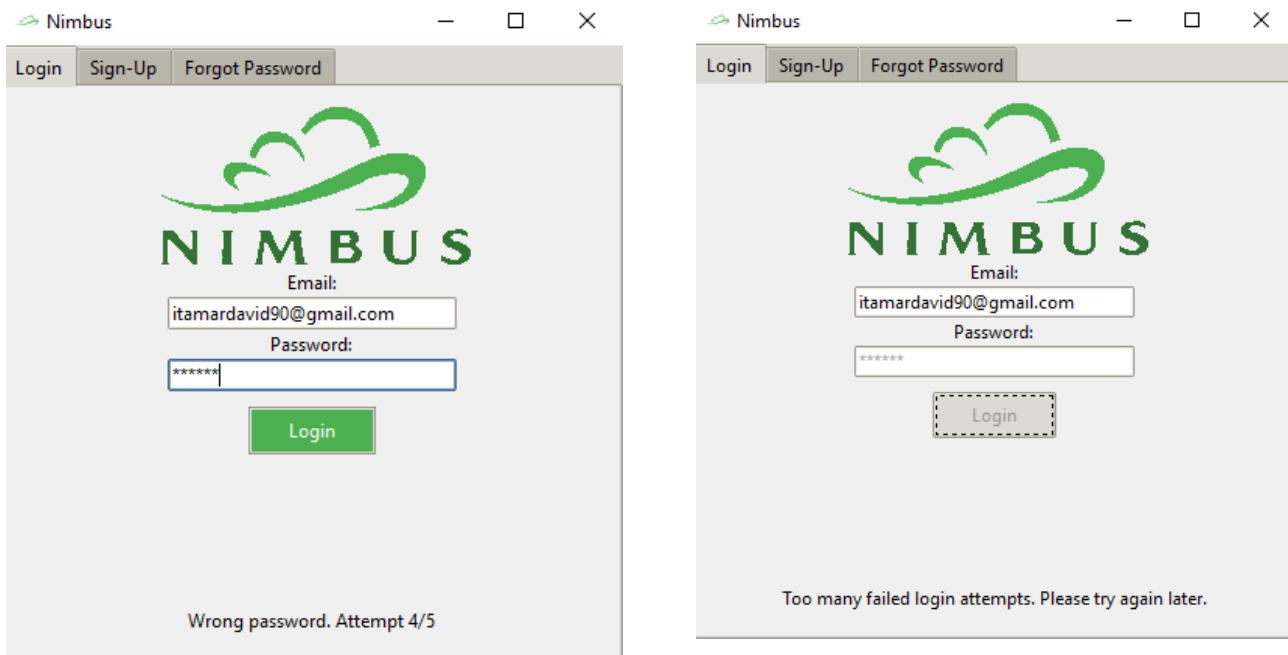
Rainbow table attack

rainbow table attack היא שיטה לפיצוח סיסמאות מוצפנות באמצעות טבלה גיבוב גדולה המכילה ערכים מקודדים מראש של סיסמאות פוטנציאליות וגבוביהן (hashes). הטבלאות הללו מאפשרות לתוקפים להפוך את תהליך ניחוש הסיסמאות ליעיל ומהיר יותר על ידי ביצוע חיפושים בטבלה במקום חישוב הגבוב של כל סיסמה אפשרית בזמן אמת. כאשר תוקף משיג את מסד הנתונים מהשרת, הוא יכול להשוות את הערכים בטבלה ל-Rainbow Table ולשחזר את הסיסמאות המקוריות במהירות רבה יותר מאשר בניחוש ברוטלי סטנדרטי.

הוספת הרנדומליות של המלח לגיבוב הסיסמאות מגן מהתקפה זאת במקרה שתוקף משיג גישה למסד הנתונים.

Brute Force בניחוש הסיסמא

כדי להגן על השרת שלי מפני התקפות Brute Force בזמן התחברות, אני מוודא להגביל את מספר הניסיונות השגויים של כל משתמש. כאשר לקוח מנסה להתחבר, אני סופר את מספר הניסיונות השגויים שלו. אם הוא חורג ממספר הניסיונות המותר, אני חוסם אותו משליחות נוספות של התחברות. (אין לי חסימה לפי כתובת IP)



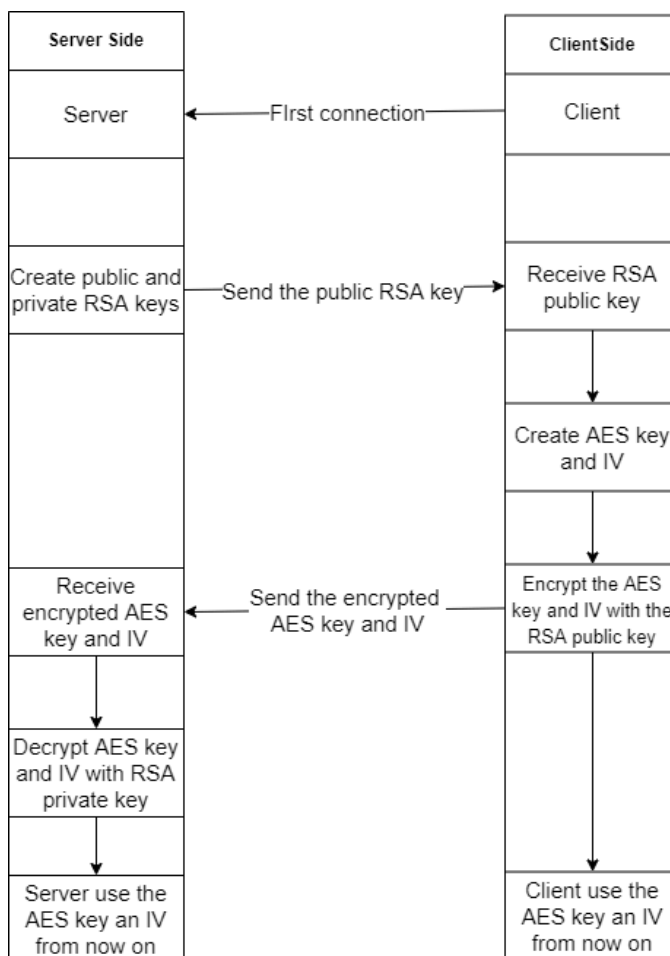
Directory traversal attack

כדי להגן על השרת שלי מ**directory traversal attack** כאשר לקוח מעלה, מוחק, או מוריד קבצים, אני דואג לבצע בדיקה קפדנית על שם הקובץ שהוא מבקש לגשת אליו. הבדיקה שלי כוללת נרמול הנתיב כדי להסיר כל רכיב כמו ".." שמצביע על תיקייה הורה, כך שהנתיב המבוקש תמיד יישאר בתוך התחום המותר עבור המשתמש. בנוסף, אני בודק שהנתיב מתחיל בתיקיית הבסיס המיועדת למשתמש ואין בו מרכיבים שעשויים להוביל אותו מחוץ לתחום הזה. אני גם מוודא שהקבצים ניגשים עם שמות תקינים וללא סימנים או תווים מיוחדים שעלולים להוות סכנה. כך אני מונע מהמשתמש גישה לא מורשית לקבצים שמחוץ לתיקייה שהוקצתה לו ושומר על אבטחת השרת.

הצפנת התעבורה

החיבור הראשוני בין הלקוח לשרת כולל סדרה של צעדים שמבטיחים יצירת תקשורת מאובטחת באמצעות מפתחות הצפנה. תהליך זה מתחיל ביצירת זוג מפתחות RSA (מפתח ציבורי ומפתח פרטי) בשרת. המפתח הציבורי נשלח ללקוח, שמקבל אותו ויוצר מפתח AES ווקטור אתחול (IV). הלקוח מצפין את מפתח ה-AES וה-IV עם המפתח הציבורי של RSA שקיבל מהשרת, ושולח אותם חזרה לשרת. השרת מקבל את המפתחות המוצפנים, מפענח אותם באמצעות המפתח הפרטי של RSA, וכך מקבל את מפתח ה-AES וה-IV בצורה בטוחה. תהליך זה מבטיח שהמפתח של AES שנוצר בלקוח מועבר בצורה מאובטחת לשרת, ובכך נמנע ממי שמנסה להאזין (man in the middle) או להפריע לתעבורה ללכוד את מפתח ה-AES.

לאחר שהשרת והלקוח מחליפים בהצלחה את המפתחות וה-IV בצורה מאובטחת, הם מתחילים להשתמש במפתח AES וב-IV להצפנה וסימטריה של התקשורת ביניהם. הצפנה סימטרית זו מאפשרת העברת מידע בצורה מהירה ויעילה, כשהמפתח הסימטרי ידוע רק לשרת וללקוח. שימוש ב-AES להצפנת התקשורת מגביר את הביטחון מאחר והנתונים המועברים מוצפנים בצורה חזקה, ובכך נמנעת גישה של צדדים שלישיים לתוכן התקשורת. התהליך הכולל מבטיח שתקשורת המידע מתבצעת בצורה מאובטחת, מוגנת מפני התקפות man in the middle והאזנות שונות, ובכך שומרת על פרטיות ושלמות הנתונים המועברים בין הלקוח לשרת.



הצפנת קבצים

ניסוח התהליך-

בעת כניסת או רישום המשתמש, השרת מקבל את סיסמת המשתמש ושולף את ה-salt_password ממסד הנתונים. השרת יוצר מפתח AES על ידי Hashing של הסיסמה וה-salt_password. בעת העלאת קובץ, השרת מייצר IV אקראי עבור כל קובץ מוצפן, מצפין את תוכן הקובץ באמצעות מפתח AES וה-IV, ושומר את ה-IV בקובץ JSON נפרד. בעת הורדת קובץ, השרת שולף את ה-IV מקובץ ה-JSON, מפענח את תוכן הקובץ באמצעות מפתח AES וה-IV, ושולח את הקובץ המפוענח למשתמש.

ההגנה שההצפנה מספקת-

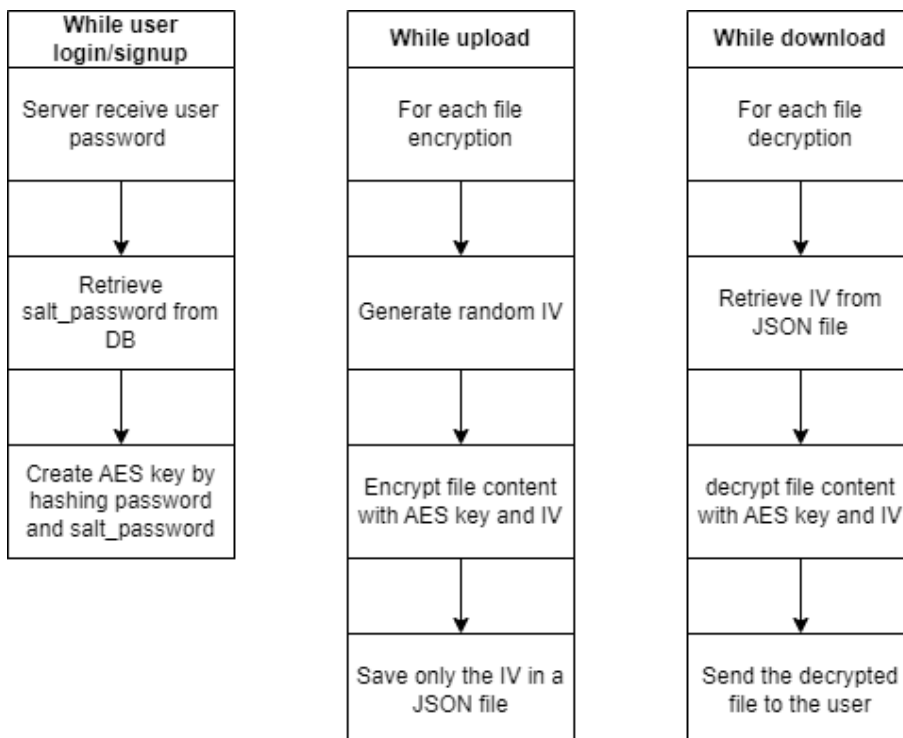
הצפנת הקבצים באמצעות AES מספקת רמת אבטחה גבוהה בכך שהיא הופכת את תוכן הקבצים לבלתי קריא ללא המפתח המתאים שמופק מסיסמת המשתמש. הצפנת הקבצים מגנה מתקיפות פוטנציאליות שבהם יגנבו את קבצי המשתמשים. השימוש ב-IV אקראי עבור כל קובץ מונע התקפות מסוג known-plaintext* ומוסיף שכבת אבטחה נוספת.

known-plaintext-

התקפת Known Plaintext מתרחשת כאשר לתוקף יש גישה למידע מוצפן (ciphertext) ולגרסה המפוענחת (plaintext) של חלק מהמחרוזת המוצפנת. עם מידע זה, התוקף יכול לנסות לפענח חלקים נוספים של המידע המוצפן, או אפילו את כל המידע, באמצעות ניתוח הקשר בין ה-ciphertext ל-plaintext הידועים.

החולשות שעדיין קיימות-

הצפנת הקבצים אינה מגינה מפני כל סוגי המתקפות, מכיוון שהמפתח מיוצר על ידי סיסמת המשתמש. במקרה פוטנציאלי שבו תוקף משיג גישה לשרת לאורך זמן הוא יכול להשיג את סיסמאות המשתמשים ומהם לייצר את המפתח. בנוסף אם תוקף יצליח לגנוב גם את הקבצים המוצפנים וגם את קובץ ה-JSON IV, הוא יוכל להשתמש בהם יחד כדי לנסות לפענח את הקבצים.



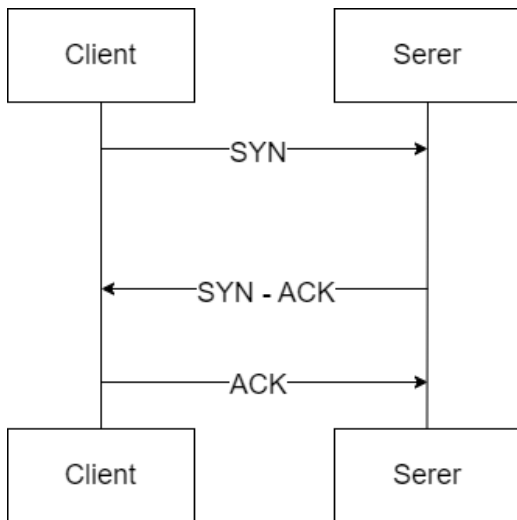
שכבת התעבורה

TCP - לחיצת יד משולשת

לחיצת היד המשולשת (Three-Way Handshake) של פרוטוקול TCP הוא תהליך ההתחברות הראשוני בין הלקוח לשרת, המבטיח תקשורת אמינה. התהליך מתחיל כאשר הלקוח שולח לשרת הודעת SYN כדי להתחיל את החיבור. השרת משיב בהודעת SYN-ACK המאשרת את קבלת הודעת ה-SYN של הלקוח וגם מסנכרנת את השרת עם הלקוח. לבסוף, הלקוח מאשר את הודעת ה-SYN-ACK על ידי שליחת הודעת ACK חזרה לשרת. לאחר שלושת השלבים הללו, שני הצדדים יכולים להתחיל לשלוח נתונים זה לזה.

היתרונות של לחיצת היד המשולשת הם רבים. ראשית,

התהליך מבטיח שהשרת והלקוח מוכנים להתחיל בתקשורת, ומאפשר סנכרון נכון של מספרי הסדר בין שני הצדדים. זה חשוב כדי להבטיח שכל חבילה נמסרת ומעובדת בסדר הנכון, מה שמביא לאמינות גבוהה יותר של העברת הנתונים.



DOS/ DDOS

התקפת DDoS (התקפת מניעת שירות מבוזרת) או DoS (התקפת מניעת שירות) היא סוג של מתקפת סייבר שמטרתה להפסיק את הפעילות התקינה של שירות מקוון על ידי הצפת השרת או הרשת בבקשות יתרות. כתוצאה מכך, המשאבים הזמינים נצרכים במהירות, והשירות אינו יכול לספק מענה לבקשות לגיטימיות של משתמשים.

התקפה זו נעשית בדרך כלל על ידי מספר רב של מחשבים או מכשירים שנפרצו ונשלטים על ידי התוקף, מה שמקשה על זיהוי המקור האמיתי של המתקפה. בפרויקט שלי אין הגנה למתקפה מהסוג הזה שגורם לחולשה רצינית ביותר, לאחר הגשת הפרויקט אני מתכוון להתייחס לבעיות אבטחה נוספות ולשפר את ההגנה שלי מ DOS/DDOS.

גישה לא רצויה למשתמש המנהל

אם תוקף משיג גישה למשתמש מנהל, מדובר בחולשת אבטחה משמעותית שעלולה לגרום לנזקים חמורים. גישה מנהל מאפשרת לתוקף למחוק משתמשים ממסד הנתונים, וגישה למידע רגיש.

מימוש משתמש

מודולים מיובאים - סיפריות חיצוניות

- **msgpack** - ספרייה ההופכת מילון לרצף ביטים כך שאפשר לשלוח דרך הסוקטים את ההודעות.
- **PyCryptoDome** - ספרייה עם פעולות צפנה ופענוח לייצור ושימוש מפתחות AES ו RSA.
- **socket** - ספרייה המאפשרת יצירה וניהול של חיבורי רשת באמצעות סוקטים.
- **Tkinter** - ספרייה לבניית ממשקי משתמש גרפיים (GUI).
- **Tkinter ttk** - הרחבה של Tkinter המספקת ווידג'טים מעוצבים נוספים.
- **sys** - מאפשרת גישה למשתנים ופעולות של מערכת ההפעלה.
- **os** - מספקת פונקציות לממשק עם מערכת ההפעלה, כמו גישה לקבצים ותיקיות.
- **zipfile** - לעבודה עם קבצים מכווצים בפורמט ZIP.
- **datetime** - ספרייה לטיפול בתאריכים ושעות.
- **logging** - ליצירה וניהול של רישומי יומן (logs).
- **hashlib** - מאפשרת יצירת ערכים מוצפנים (hash) לאבטחת נתונים.
- **base64** - להצפנה ופענוח של נתונים בפורמט Base64.
- **ssl** - ליצירת חיבורי רשת מאובטחים באמצעות פרוטוקול SSL/TLS.
- **smtplib** - לשליחת דואר אלקטרוני באמצעות פרוטוקול SMTP.
- **email** - לבניית ועיבוד הודעות דואר אלקטרוני.
- **sqlite3** - לעבודה עם מסדי נתונים מבוססי SQLite.
- **shutil** - לביצוע פעולות מערכת קבצים מתקדמות כמו העתקה והעברה של קבצים.
- **threading** - לניהול וביצוע של תהליכונים (threads) במקביל.
- **random** - ליצירת מספרים אקראיים ושימוש באלגוריתמים של אקראיות.
- **json** - לעבודה עם נתונים בפורמט JSON.

המודולים שלי

- Client.py
- client_admin.py
- create_venv.py
- Run_Nimbus.bat
- protocol.py
- Server.py
- server_admin.py
- logger.py

המחלקות שלי

FileItem מחלקת

מודול המחלקה נמצא ב `Client.py` וממומש לכל קובץ שמוצג בחלון הלקוח. מטרתו העיקרית היא לקצר שמות של קבצים ארוכים בשביל הצגה יותר נוחה בממשק בזמן שהוא עדיין שומר את השם המקורי ולא דורס אותו. הוא כולל שלושה משתנים, פונקציה סטטית אחת ופונקציות החזרת משתנים:

```
class FileItem:
    def __init__(self, name, file_type, max_display_length=20):
        self.full_name = name
        self.file_type = file_type
        self.display_name = FileItem.truncate_name(name, max_display_length)

    @staticmethod
    def truncate_name(name, max_length):
        if len(name) > max_length:
            return f"{name[:max_length-3]}..."
        else:
            return name

    def __str__(self):
        return self.display_name

    def __repr__(self):
        return self.full_name

    def get_full_name(self):
        return self.full_name
```

`self.fullname` - שומר את השם המקורי של הקובץ

`self.filetype` - שומר את סוג הקובץ

`self.displayname` - השם המקוצר להצגה

`def truncate_name` - אחראי על קיצור

`(def __str__(self)` מחזיר את השם המקוצר

`(def __repr__(self)` מחזיר את השם המקורי

`def get_full_name` מחזיר את השם המקורי

כאשר הלקוח מקבל את רשימת הקבצים שיש לו מהשרת הוא מממש את המחלקה ויוצר אובייקט לכל אחד מהקבצים, לאחר מכן כאשר הוא מעדכן את הכפתור של כל אחד מהקבצים הוא מגדיר שם את ה `string` בתור האובייקט עם השם המקוצר. לאחר מכן כאשר הלקוח רוצה להשתמש בקובץ (הורדה,מחיקה). הוא ניגש בכפתור הקובץ לאובייקט ושולף את השם המלא כך שהשם המקורי לא נדרס ואין שימוש במשתנים גלובליים רבים.

מחלקת Logger

מודול המחלקה נמצא בקובץ `logger.py` ולא ממומש כאובייקט אלא השרת משתמש בפונקציות שלו לכל אורך ההרצה של השרת.

מטרתו העיקרית של המחלקה היא תיעוד כל התהליכים הקורים בשרת, קריסות, יצירת והתחברות משתמשים חדשים, הורדות וגם יש מספר מתקפות שאני מזהה ומוסיף תיעוד אליהם ל-`logger`. לכל מקרה כזה השרת מצרף את השעה ואת שם הלקוח שגרם למקרה.

המחלקה כוללת 9 פונקציות סטטיות ועושה שימוש בספרייה החיצונית `logging`

```
import logging

class Logger:
    @staticmethod
    def log_upload(username, file_name):
        logging.basicConfig(filename='user_actions.log', level=logging.INFO, format='%(asctime)s - %(message)s', datefmt='%Y-%m-%d %H:%M:%S')
        logging.info(f'{username} uploaded file: {file_name}')

    @staticmethod
    def log_delete(username, file_or_folder_name):
        logging.basicConfig(filename='user_actions.log', level=logging.INFO, format='%(asctime)s - %(message)s', datefmt='%Y-%m-%d %H:%M:%S')
        logging.info(f'{username} deleted {file_or_folder_name}')

    @staticmethod
    def log_login(username):
        logging.basicConfig(filename='user_actions.log', level=logging.INFO, format='%(asctime)s - %(message)s', datefmt='%Y-%m-%d %H:%M:%S')
        logging.info(f'{username} logged in')

    @staticmethod
    def log_admin_login(username):
        logging.basicConfig(filename='user_actions.log', level=logging.INFO, format='%(asctime)s - %(message)s', datefmt='%Y-%m-%d %H:%M:%S')
        logging.info(f'{username} admin logged in')

    @staticmethod
    def log_signup(username):
        logging.basicConfig(filename='user_actions.log', level=logging.INFO, format='%(asctime)s - %(message)s', datefmt='%Y-%m-%d %H:%M:%S')
        logging.info(f'{username} signed up')

    @staticmethod
    def log_download(username, file_name):
        logging.basicConfig(filename='user_actions.log', level=logging.INFO, format='%(asctime)s - %(message)s', datefmt='%Y-%m-%d %H:%M:%S')
        logging.info(f'{username} downloaded file: {file_name}')

    @staticmethod
    def log_error(string):
        logging.basicConfig(filename='user_actions.log', level=logging.INFO, format='%(asctime)s - %(message)s', datefmt='%Y-%m-%d %H:%M:%S')
        logging.info(f'error: {string}')

    @staticmethod
    def log_email_sent(email, string):
```

הפונקציות מאוד דומות אז אסביר בכלליות; כאשר יש מקרה בצד השרת הוא קורא לפונקציה מהקלאס עם צירוף שם המשתמש ואת המשתנים הקשורים למקרה. לדוגמא:

לקוח: itamar העלה קובץ : `example.txt`
ולאחר מכן ההודעה תכתב בקובץ `user_actions.log` בתור:
`itamar uploaded file: example.txt - 00:27:31 2024-04-13`

לאחר מכן ללקוח המנהל יש גישה לראות את תוכן הקובץ

מחלקת AESCipher

מודול המחלקה נמצא בקובץ `protocol.py` וממומש כאובייקט בחיבור הראשוני של לקוח.

מטרתו העיקרית של המחלקה היא ליצור אובייקט שישמור את מפתח ה AES ואת ה IV כאובייקט אחד כך שאוכל להעביר אותו בשרת בצורה יעילה יותר (משתנה אחד פחות!)

המחלקה כוללת שני משתנים ושתי פונקציות:

```
class AESCipher:
    def __init__(self, key, iv):
        self.key = key
        self.iv = iv

    def encrypt(self, message):
        aes_cipher = AES.new(self.key, AES.MODE_CBC, self.iv)
        padded_plaintext = pad(message, AES.block_size)
        encrypted_message = aes_cipher.encrypt(padded_plaintext)
        return encrypted_message

    def decrypt(self, encrypted_message):
        aes_cipher = AES.new(self.key, AES.MODE_CBC, self.iv)
        decrypted_data = aes_cipher.decrypt(encrypted_message)
        plaintext = unpad(decrypted_data, AES.block_size)
        return plaintext
```

`self.key` - שומר את מפתח ה AES

`self.iv` - שומר את הוקטור ההתחלתי (IV)

פונקציית `encrypt` - מקבלת רצף ביטים ומצפינה אותם בעזרת הצפנת ה AES, השימוש במחלקה זאת קורת כחלק מהפרוטוקול.

פונקציית `decrypt` - פונקציה הפוכה לקודמת, מקבל ביטים מוצפנים ומפענחת אותם בעזרת המפתח וה IV

פונקציות עזר

- הפונקציה קיימת בצד השרת והלקוח
הפונקציה קוראת את תוכן הקובץ במצב בינארי ומחזירה את התוכן כערך מסוג .bytes

```
def read_file_content(file_path):
    """
    Reads the content of a file in binary mode.
    :param file_path: The path to the file to be read.
    :return: (bytes) The content of the file.
    """
    try:
        with open(file_path, 'rb') as file:
            return file.read()
    except FileNotFoundError:
        print(f"Error: File '{file_path}' not found.")
    except Exception as e:
        print(f"Error: {e}")
```

- הפונקציה קיימת בצד בשרת והלקוח
הפונקציה ממירה ערך בינארי לפורמט קריא ומחזירה את הגודל עם סיומת מתאימה.

```
def print_bytes(bytes_value):
    """
    Converts a byte value into a human-readable format
    :param bytes_value:
    :return: string that contain the size and suffix
    """
    suffixes = ['B', 'KB', 'MB', 'GB']
    suffix_index = 0

    while bytes_value >= 1024 and suffix_index < len(suffixes) - 1:
        bytes_value /= 1024.0
        suffix_index += 1

    print(f"{bytes_value:.2f} {suffixes[suffix_index]}")
```

- הפונקציה קיימת בצד השרת
הפונקציה מחשבת את הגודל הכולל של כל הקבצים בתיקיה מסוימת ומחזירה את סכ"ה הגודל.

```
def get_folder_size(folder_path):
    """
    get the sum size for all the files within a folder
    :param folder_path:
    :return:
    """
    # Convert folder_path to a string
    folder_path = str(folder_path)
    total_size = 0
    # Walk through all files and subdirectories in the specified folder
    for dir_path, dir_names, filenames in os.walk(folder_path):
        for filename in filenames:
            # Get the full path of the file
            file_path = os.path.join(dir_path, filename)
            # Ensure file_path is a string
            file_path = str(file_path)
            # Add the size of the file to the total size
            total_size += os.path.getsize(file_path)
    return total_size
```

- הפונקציה קיימת בצד השרת
הפונקציה בודקת אם קובץ או תיקיה קיימים עבור משתמש מסוים בתיקיית ההעלאות.

```
def check_file_exists(username, filename):
    """
    Checks if a file or directory exists for a given user in the upload
    folder.
    :param username: (str) The username associated with the folder.
    :param filename: (str) The name of the file or directory to check.
    :return: (bool) True if the file or directory exists, False otherwise.
    """
    user_folder = os.path.join(UPLOAD_FOLDER, username)
    file_path = os.path.join(user_folder, filename)
    return os.path.isfile(file_path) or os.path.isdir(file_path)
```

- הפונקציה קיימת בצד השרת
הפונקציה מוודאת שהסיסמא במינימום סטנדרטי של אבטחה ומחזירה משתנה בוליאני

```
def is_valid_password(password):
    """
    check if the password is within the minimum standard
    :param password:
    :return:
    """
    # Check if password has at least 6 characters and a number
    if len(password) < 6:
        return False
    has_number = any(char.isdigit() for char in password)
    return has_number
```

- הפונקציה קיימת בצד השרת
הפונקציה מוודאת שהאימייל בפורמט הגיוני ומחזירה משתנה בוליאני

```
def is_valid_email(email):
    """
    Validates an email address by checking its structure.
    :param email: (str) The email address to validate.
    :return: (bool) True if the email address is valid, False otherwise.
    """
    if '@' not in email or '.' not in email:
        return False
    parts = email.split('@')
    if len(parts) != 2:
        return False
    local_part, domain_part = parts
    if not local_part or not domain_part:
        return False
    if ' ' in local_part or ' ' in domain_part:
        return False
    domain_parts = domain_part.split('.')
    if len(domain_parts) < 2:
        return False
    for part in domain_parts:
        if not part:
            return False
    return True
```

- הפונקציה קיימת בצד השרת
הפונקציה שולחת אימייל אימות עם קוד לכתובת האימייל הנתונה. היא משתמשת בשרת SMTP של Gmail ומשתמשת ב-SSL לתקשורת מאובטחת.

```
def send_verification_email(email, code):
    """
    Sends a verification email with a provided code to a specified email
    address.
    :param email: (str) The recipient's email address.
    :param code: (int) The verification code to be included in the email.
    :return:
    """

    sender = 'nimbus.mail.ver@gmail.com'
    sender_password = "*" # I removed the password for the email
    receiver = email
    subject = 'Verification Code'
    body = f'Your verification code is: {code}'

    em = EmailMessage()
    em["From"] = sender
    em["To"] = receiver
    em["Subject"] = subject
    em.set_content(body)
    context = ssl.create_default_context()

    with smtplib.SMTP_SSL("smtp.gmail.com", 465, context=context) as smtp:
        smtp.login(sender, sender_password)
        smtp.sendmail(sender, receiver, em.as_string())
        print("email sent")
        Logger.log_email_sent(email, subject)
```

קטעי קוד נבחרים

קטע קוד שאהבתי מאוד שלא צירפתי לתוצר הסופי הוא המימוש האישי שלי ליצירת מפתחות RSA- אני יודע שמימוש אישי לא יעיל ולכן בפרויקט השתמשתי בספריות חיצוניות.

למרות זאת מאוד נהנתי מהלמידה של המתמטיקה וההצפנה האסימטרית.

```
import msgpack
import random
import math

def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True

def generate_random_prime(min_val, max_val):
    while True:
        # Generate a random integer within the specified range
        num = random.randint(min_val, max_val)
        # Check if the number is prime
        if is_prime(num):
            return num

def mod_inverse(e, phi):
    for d in range(3, phi):
        if (d * e) % phi == 1:
            return d

def rsa_encrypt(msg, public_key, n):
    msg_encoded = [ord(ch) for ch in msg]
    msg_encrypted = [pow(ch, public_key, n) for ch in msg_encoded]
    return msg_encrypted

def rsa_decrypt(msg, private_key, n):
    msg_a = [pow(ch, private_key, n) for ch in msg]
    msg_b = "".join(chr(ch) for ch in msg_a)
    return msg_b

def rsa_get_keys():
    p, q = generate_random_prime(1000, 5000), generate_random_prime(1000, 5000)
    while p == q:
        q = generate_random_prime(1000, 5000)

    n = p * q
    phi_n = (p-1) * (q-1)
    public_key = random.randint(3, phi_n-1)

    while math.gcd(public_key, phi_n) != 1:
        public_key = random.randint(3, phi_n-1)

    private_key = mod_inverse(public_key, phi_n)
    print(f"public_key:{public_key}")
    print(f"private_key :{private_key}")
    return public_key, private_key, n
```

הורדת קבצים/תיקיות (לקוח)

צירפתי את הקוד של הפונקציה האחראית על האלגוריתם, למרות שהפונקציה ארוכה ואני יכול לפצל אותה למספר חלקים, הקוד שלי נשאר נקי וקריא עם חילוק שלבים ואני מאוד גאה בו.

```
def download():
    """
    Downloads selected files from the server to the local machine's Downloads folder.
    Global Variables:
    - client_path (str): Path to the client directory within the UI
    This function performs the following steps:
    1. Retrieves the list of selected files based on the user's selection.
    2. Constructs the file paths the selected files.
    3. Sends a download request to the server with the list of selected files.
    4. Receives the files from the server, handling potential responses indicating
        that no files were selected.
    5. Saves the received files to the Downloads folder, ensuring unique filenames
        to avoid conflicts.
    6. If a folder is downloaded, it is saved as a zip file, extracted, and the zip
        file is then removed.
    :return:
    """
    global client_path
    # construct a list with the file paths
    files_list_items = [filename_map.get(button1) for button1 in selected_buttons]
    files_list = [item.get_full_name() for item in files_list_items]
    print(f"files list: {files_list}")
    files_with_path = [os.path.join(client_path, filename) for filename in files_list]
    # send the request to the server
    download_dict = {"command": "download", "files": files_with_path}
    client_socket.sendall(protocol.send_message_aes(download_dict, aes_cipher))
    response_dict = protocol.get_message_aes(client_socket, aes_cipher)

    downloads_folder = os.path.expanduser("~") + os.sep + "Downloads"
    files = response_dict.get("files")
    print(files)
    if files == "no files selected":
        return
    for file_info in files:
        file_name = file_info["file_name"]
        file_type = file_info["file_type"]
        file_content = file_info["file_content"]
        file_path = os.path.join(downloads_folder, f"{file_name}{file_type}")
        new_file_name = file_name
        counter = 1
        while os.path.exists(file_path):
            new_file_name = f"{file_name}({counter})"
            file_path = os.path.join(downloads_folder, f"{new_file_name}{file_type}")
            counter += 1
        file_name = new_file_name

        if file_type == "folder":
            # Handle the case when the user downloads a folder
            folder_name = os.path.splitext(file_name)[0] # Remove the .zip extension
            unzipped_folder_path = os.path.join(downloads_folder, folder_name)
            # Added a loop to handle folder name conflicts
            counter = 1
            while os.path.exists(unzipped_folder_path):
                new_folder_name = f"{folder_name}({counter})"
                unzipped_folder_path = os.path.join(downloads_folder, new_folder_name)
                counter += 1
            zip_file_path = os.path.join(downloads_folder, file_name)
            with open(zip_file_path, "wb") as f:
                f.write(file_content)
            print(f"Folder saved: {zip_file_path}")
            # Unzip the folder
            with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
                for member in zip_ref.infolist():
                    extracted_path = zip_ref.extract(member, unzipped_folder_path)
                    if member.is_dir():
                        os.makedirs(extracted_path, exist_ok=True)
            print(f"Folder extracted: {unzipped_folder_path}")
            # Remove the zip file
            os.remove(zip_file_path)
        else:
            # Handle the case when the user downloads a file
            filepath = os.path.join(downloads_folder, file_name + file_type)
            with open(filepath, "wb") as f:
                f.write(file_content)
            print(f"File saved: {filepath}")
    refresh()
```

בדיקה ראשונה

```
"movie_script.txt"
```

הקובץ לאחר הצפנה:

[illegible]

48

בדיקה שניה

אימות המשתמש על פי תיאור הבדיקה במבוא.

אני אשלח לשרת מספר מקרים ואדווח על התוצאות:
בעת ההתחברות-

- אימייל נכון וסיסמא נכונה -> התחברות בהצלחה
- אימייל נכון עם סיסמא לא נכונה -> קבלת הודעת שגיאה
- אימייל לא קיים עם סיסמא שגויה -> קבלת הודעת שגיאה
- שדות אימייל וסיסמא ריקים -> קבלת הודעת שגיאה

בעת ההרשמה

- אימייל הגיוני, שם משתמש ייחודי וסיסמא שעומדת בסטנדרט -> מעבר לשלב הבא
- פורמט אימייל שגוי -> קבלת הודעת שגיאה
- אימייל שקיים במערכת -> קבלת הודעת שגיאה
- שם משתמש שקיים במערכת -> קבלת הודעת שגיאה
- סיסמא שלא עומדת בסטנדרט -> קבלת הודעת שגיאה
- שילובים אחרים של פרמטרים שגויים -> קבלת הודעת שגיאה

שלב שני של ההתחברות

- שליחת קוד תקין על פי האימייל שקיבלתי -> יצירת משתמש בהצלחה
- שליחת קוד לא תקין -> קבלת שגיאה

בעת שינוי סיסמא

- אימייל שקיים במערכת -> מעבר לשלב הבא
- אימייל לא קיים במערכת -> קבלת הודעת שגיאה
- פורמט אימייל שגוי -> קבלת הודעת שגיאה

השלב השני של שינוי הסיסמא

- שליחת קוד תקין על פי האימייל שקיבלתי -> מעבר לשלב הבא
- שליחת קוד לא תקין -> קבלת שגיאה

השלב השני של שינוי הסיסמא

- סיסמא חדשה שעומדת בסטנדרט -> שינוי סיסמא בהצלחה
- סיסמא חלשה שלא עומדת בסטנדרט -> קבלת שגיאה

לאחר ביצוע הבדיקות ווידאתי באופן מקיף על כל בעיות שיכולות להווצר בעת ההתחברות, הבדיקות עוזרות מאוד בלאתר באגים פוטנציאליים וחשיבה על מקרי קצה.

בדיקה שלישית

מטרת הבדיקה היא לוודא שהתעבורה באמת עוברת בצורה מוצפנת על פי הפרוטוקול ולא יהיה אפשר לגשת לתוכן הפקטות על ידי הספנה מגורם שלישי. גם כאן תיאור הבדיקה הוא לפי הבדיקות במבוא

לצורך הבדיקה השתמשתי בתוכנה **wireshark** שהסברתי עליה בתיאור סביבת העבודה, בעזרת התוכנה אני אסתכל על התעבורה בין הלקוח לשרת ואסניף את הפקטות.

מכיוון שבתחילת החיבור כאשר הלקוח והשרת מחליפים את מפתח ה-AES התעבורה ביניהם לא מוצפנת אסתכל על פקטת שליחת מפתח public RSA.

0000	02 00 00 00 45 00 01 fc	51 40 40 00 80 06 00 00E... Q@@....
0010	7f 00 00 01 7f 00 00 01	30 39 e4 d5 1c 47 33 48 09...G3H
0020	43 fd 12 e0 50 18 27 f9	43 3d 00 00 30 30 30 30	C...P... C...0000
0030	30 30 30 34 35 38 81 a3	6b 65 79 c5 01 c2 2d 2d	000458.. key....
0040	2d 2d 2d 42 45 47 49 4e	20 50 55 42 4c 49 43 20	---BEGIN PUBLIC
0050	4b 45 59 2d 2d 2d 2d 2d	0a 4d 49 49 42 49 6a 41	KEY----- MIIBIJA
0060	4e 42 67 6b 71 68 6b 69	47 39 77 30 42 41 51 45	NBqkqhki G9w0BAQE
0070	46 41 41 4f 43 41 51 38	41 4d 49 49 42 43 67 4b	FAAOCAQ8 AMIIBGK
0080	43 41 51 45 41 70 78 58	56 4c 39 50 46 37 37 56	CAQEApxX VL9PF77V
0090	55 65 4e 66 4d 63 5a 6e	6b 0a 65 43 35 47 55 34	UeNfMcZn k·eC5GU4
00a0	78 6a 42 42 30 7a 64 32	4f 44 75 71 63 49 4d 36	xjBB0zd2 ODuqcIM6
00b0	47 34 41 32 67 77 42 73	49 54 5a 45 7a 6b 56 44	G4A2gwBs ITZEzkVD
00c0	4e 45 49 61 68 4d 33 78	70 2f 75 4a 6f 53 59 6b	NEIahM3x p/uJoSYk
00d0	6c 56 51 6f 6a 4a 77 35	64 6b 0a 4d 39 2f 70 33	lVQoJw5 dk·M9/p3
00e0	4a 37 66 48 51 43 6b 49	54 42 42 5a 73 79 53 73	J7fHQCKI TBBZsySs
00f0	76 71 51 4f 4a 75 36 36	54 78 42 56 61 6c 75 57	vqQ0Ju66 TxBVaUw
0100	45 36 31 59 76 36 44 66	33 4d 6f 33 75 2f 63 49	E61Yv6Df 3Mo3u/cI
0110	74 59 49 57 76 71 56 33	7a 36 71 0a 75 63 46 42	tYIWvqV3 z6q·ucFB
0120	59 67 57 49 4b 54 37 55	72 65 71 6b 78 31 76 6b	YgWIKT7U reqkxlvk
0130	38 55 4d 47 64 38 78 74	42 54 45 38 75 6f 49 57	8UMGd8xt BTE8uoIW
0140	31 78 6d 43 71 78 74 39	4f 69 70 68 56 50 43 5a	1xmCqxt9 OiphVPCZ
0150	33 6f 41 70 30 35 73 69	74 68 38 55 0a 47 32 58	3oAp05si th8U·G2X
0160	37 79 37 31 6c 58 39 47	79 42 58 73 64 65 79 52	7y71lX9G yBXsdeyR
0170	59 79 71 55 77 74 65 6e	50 7a 7a 4e 39 45 36 4c	YyqUwten PzzN9E6L
0180	67 33 34 35 2b 52 63 50	73 69 75 6e 67 59 45 54	g345+RcP siungYET

וכמובן על פי התמונה בקלות אפשרות לגלות את ההודעה מהספנה של גורם שלישי.

ההצפנה של התעבורה קורת רק לאחר ההודעה הראשונית אז אסתכל על פקטה נוספת לאחר החיבור שמוצפנת.

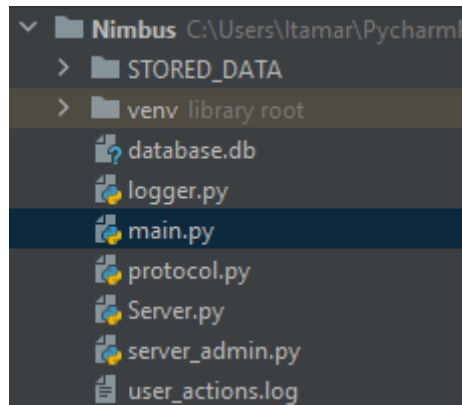
0000	02 00 00 00 45 00 00 62	52 02 40 00 80 06 00 00E..b R·@....
0010	7f 00 00 01 7f 00 00 01	30 39 e4 d5 1c 47 35 1c 09...G5·
0020	43 fd 15 46 50 18 27 f7	7c 88 00 00 30 30 30 30	C·FP·'· ...0000
0030	30 30 30 30 34 38 05 12	45 a5 e7 bd 04 92 a2 19	000048· E·.....
0040	f6 2b 5c 44 b1 d4 97 ff	47 47 47 27 0b a7 54 b2	·+D·... GGG'·T·
0050	d4 7e 39 ac c1 57 10 ee	83 d6 94 f6 f9 be b2 30	·~9·W·0
0060	e2 6a 77 8b f5 10		·jw·...

הבדיקה עבדה! ההצפנה עובדת מצויין ואין אפשרות לפענח את ההודעה המקורית על ידי הסנפת התקשורת

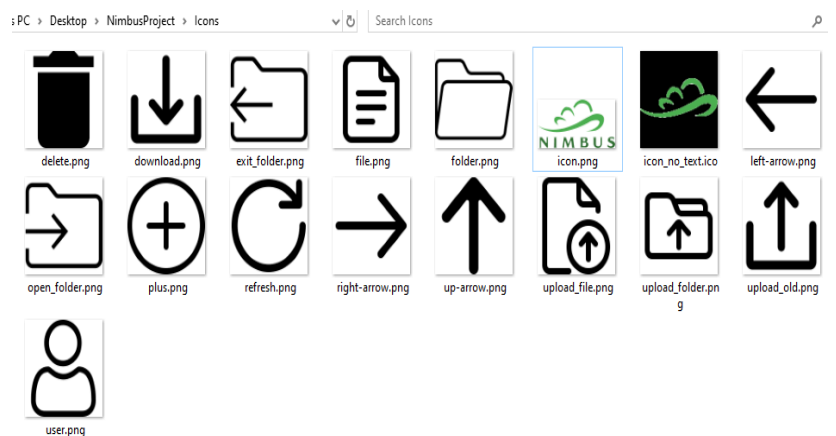
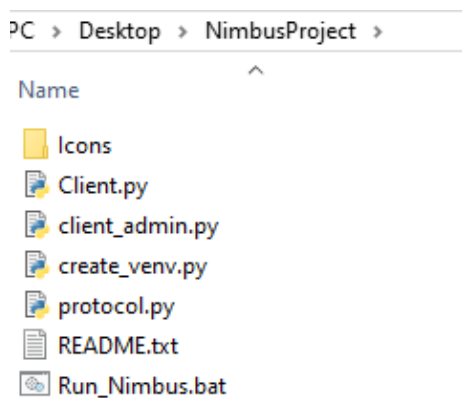
מדריך למשתמש

עץ קבצים

שרת-



לקוח-



הסביבה הנדרשת

דרישות חומרה: מחשב

דרישות תוכנה:

- מערכת הפעלה windows
- כ- 25Mb פנוי
- חיבור לאינטרנט
- שפת פייטון מותקן

דרישות נוספות: חשבון אימייל

מדריך התקנת המערכת אצל לקוח

על מנת להתקין ולהריץ את התוכנה הלקוח צריך להוריד את קובץ zip של Nimbus.zip, לעשות unzip ולחיצה כפול על Run_Nimbus.bat

כתבתי סקריפט קצר בשפת batch שמתקין את כל הדברים הנחוצים בשביל הקוד (virtual enviroment) ואת הסיפריות החיצוניות שצריך ומפעיל את הקוד בשביל ליצור חוויה קלה למשתמש, אין צורך בטרמינל או לפתוח את הקוד בשביל להריץ אותו.

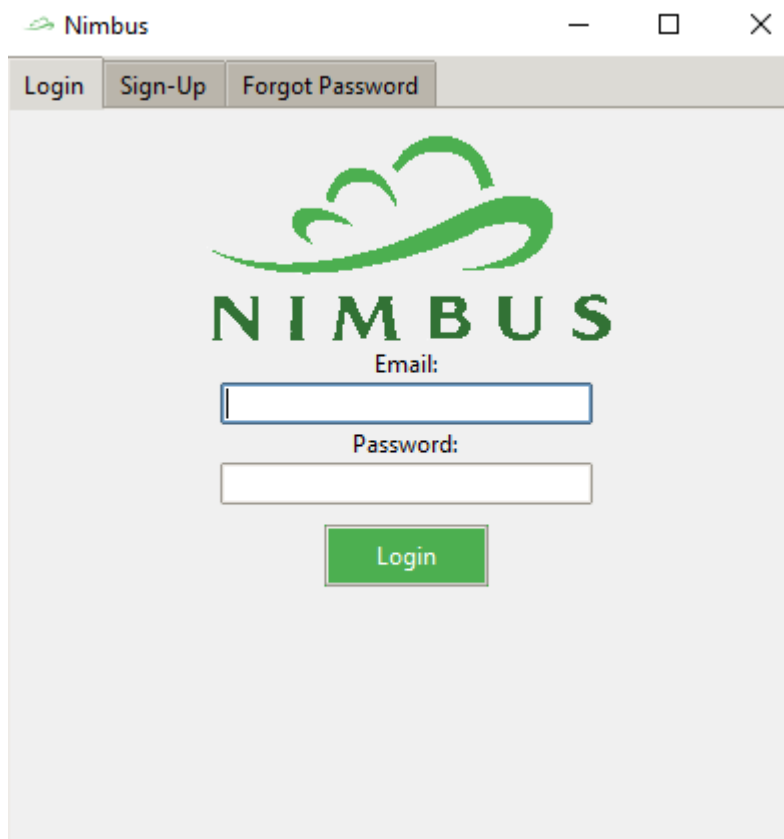
מיקומי קבצים ונתונים התחלתיים

כל הקבצים שהלקוח צריך כבר מסודרים במיקום שלהם בתיקיית הלקוח.

אין נתונים התחלתיים שמשתמש צריך, כאשר יש לקוח חדש הוא יכול ליצור משתמש בעזרת חלון ההצטרפות.

שימוש במערכת- לקוח רגיל

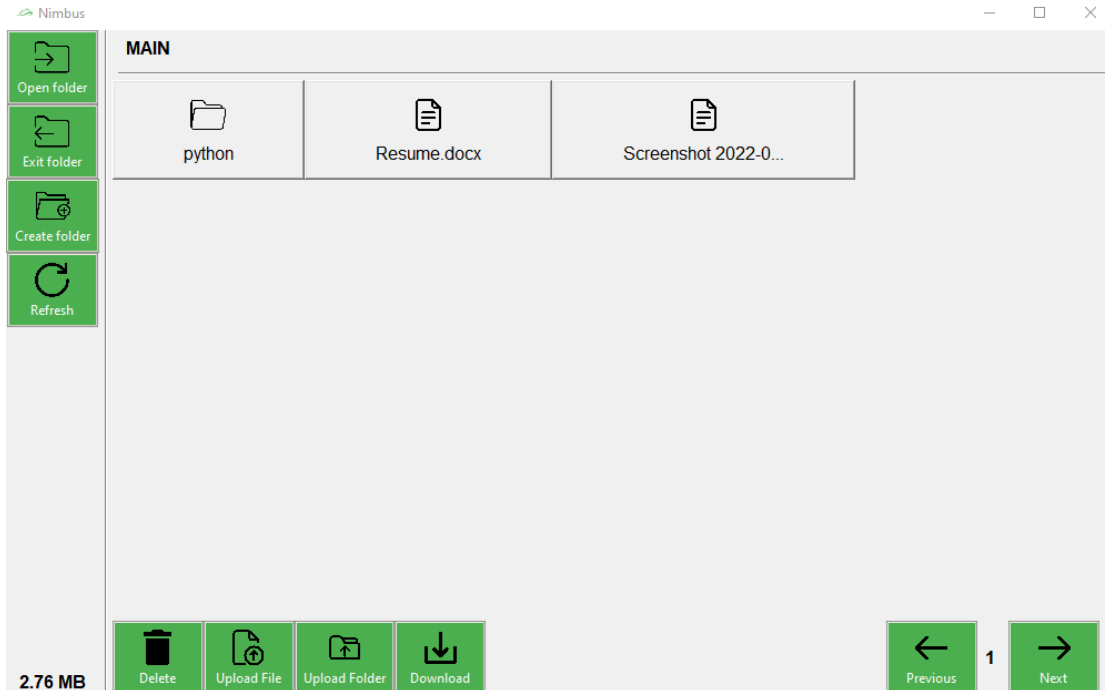
כאשר לקוח פותח את התוכנה יש לו אפשרות ליצור משתמש/ להתחבר למשתמש קיים/ לשנות סיסמא למשתמש קיים.



The screenshot shows a web application window titled "Nimbus". At the top, there are three tabs: "Login", "Sign-Up", and "Forgot Password". The "Login" tab is selected. The main content area features the Nimbus logo (a green cloud-like shape above the word "NIMBUS" in green capital letters). Below the logo, there are two input fields: "Email:" and "Password:". Below these fields is a green "Login" button. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

לאחר ההתחברות הוא יכול, לעלות קבצים/תיקית. לצפות בקבצים שהוא שמר, להכנס לתוך תיקיות, ליצור תיקייה חדשה, למחוק קובץ ועוד.

בנוסף בתחתית המסך משמאל רשום לו את כמות האחסון של הקבצים שלו בשרת.(לכל לקוח יש מגבלה של 1GB)

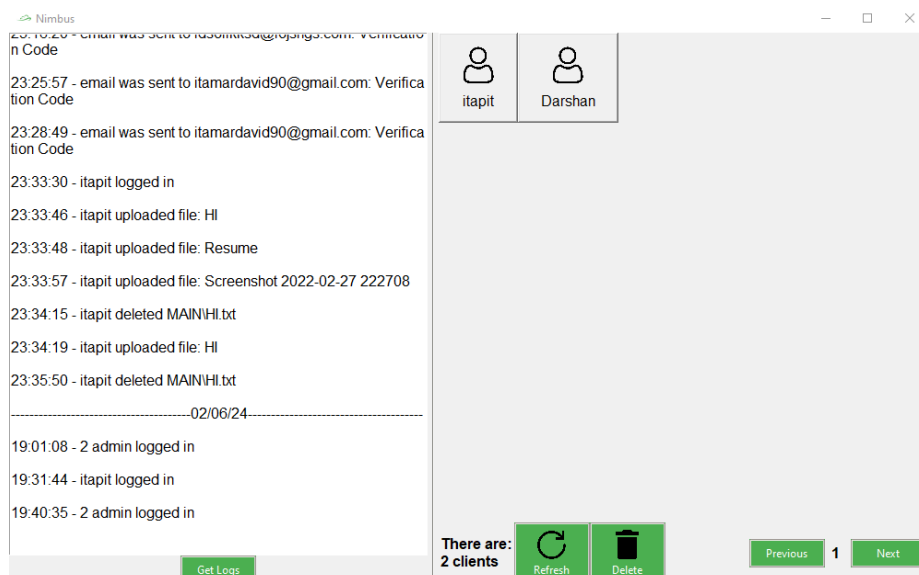


שימוש במערכת- לקוח מנהל

כאשר מנהל פותח את התוכנה יש לו אפשרות להתחבר למשתמש קיים/ לשנות סיסמא.

רק עם גישה לשרת יש אפשרות ליצור משתמש של מנהל לכן הוא לא יכול ליצור משתמש חדש.

לאחר ההתחברות הוא יכול, להסתכל על קובץ ה logs ולהסתכל על האירועים שקרו בשרת, בנוסף הוא יכול לראות את המשתמשים הקיימים בשרת ואפילו במקרה שיש משתמש זדוני הוא יכול למחוק אותו מהDB ואת הקבצים המשייכים לו.



סיכום אישי / רפלקציה

תיאור תהליך העבודה על הפרויקט

את תהליך העבודה שלי על הקוד נהגתי לעשות ב"מרתונים", בכל פעם הגדרתי לעצמי מטרה ברורה והתמקדתי בה לאורך מספר ימים פנויים שהיו לי, השיטה הזאת עזרה לי מאוד בשביל שלא אשכח חלקים מהקוד וגם אקבל הפסקה בין כל פונקציה חדש שהוספתי.

אפשר לרכז את שלבי העבודה שלי לשלושה חלקים מרכזיים, בתחילת הפרויקט השקעתי זמן רב בשביל לתכנן ארכיטקטורה יעילה ומסודרת לקוד שבהמשך חסך לי הרבה זמן. לאחר מכן היה לי מאוד קל להוסיף פונקציות ופיצ'רים חדשים והמתקדתי בזאת רבות ביחד עם ממשק המשתמש. בשלב השלישי המתקדתי בתיקון באגים ושיפור האבטחה בפרויקט שלי.

האתגרים, ההצלחות, הקשיים ודרכי הפתרון

כאשר נתקלתי בבאגים וקשיים בקוד נעזרתי במספר כלים חיצוניים; ב- debugger של pycharm, עזרה מחברים ולעיתים שימוש ב wireshark

היו לי הרבה אתגרים לאורך העבודה על הפרויקט, לאחר זמן מה שעבדתי על הפרויקט שמתי לב שכאשר נתקלתי באתגר משמעותי, היה לי מועיל במיוחד לחשוב על האתגר דווקא כשלא הייתי מול המסך. גיליתי שזמן הליכה ברגל או ביצוע משימות בבית אפשרו לי לחשוב על פתרונות יצירתיים ואפילו לזהות בעיות חדשות שעליי לתקן.

תהליך הלמידה

חלק חשוב בתהליך הלמידה שלי היה אבטחת המידע, במהלך הפרויקט למדתי על מתקפות שונות ודרכי התגוננות, השתמשתי בעיקר ביוטיוב ותיעוד באינטרנט על המתקפות השונות, כאשר נתקלתי במתקפה חדשה חקרתי אליה ולאחר מכן חשבתי על איך אני יכול להטמיע הגנה להתקפה בפרויקט שלי.

עזר נוסף שהשתמשתי בשביל ללמוד מידע חדש ובעיקר סיפריות חדשות הוא מודולים של בינה מלכותית כמו claude - i chat gpt, השימוש בבינה המלכותית עוזרת מאוד בשביל ללמוד מידע חדש אך לא מספיק מכיוון שהם יכולים לטעות.

יכולות חדשות שנלמדו

למדתי הרבה על התקפות שונות ואפשרויות להגן מהם ומאוד נהנתי מהשלב הזה, אני מאוד מתעניין באבטחת מידע ונהנתי ללמוד איך להגן מסוגי התקפות שונות, לחקור על כל אחת מהן ולמצוא דרכים לחזק את שכבת ההגנה של המערכת.

בנוסף למידע החדש שלמדתי על אבטחת מידע רכשתי מיומנויות חדשות נוספות, למדתי הרבה על יצירת חווית משתמש שלמה ואינטואטיבית- יצרתי לוגו לפרויקט, בחרתי שם מתאים, בחירת פלטת צבעים (גם לתיק), הוספת אייקונים מתאימים לכפתורים וכמובן כתיבת הקוד של ממשק המשתמש.

בשנים האחרונות בתיכון, למדתי את החשיבות של כתיבת קוד נקי; השנה, הבנתי עד כמה חשוב לתת שמות מתאימים למשתנים ולפונקציות בפרויקט שלי. מכיוון כשמדובר במערכת קצה-לקצה עם מספר רב של משתנים, ההתייחסות לשמות המשתנים נהיית הרבה יותר חשובה.

כלים להמשך

הכלי הכי חשוב שלמדתי הוא היכולת ללמוד מידע חדש ולדעת ליישם אותו בקוד בצורה אפקטיבית, התהליך של מחקר לפני יישום יעזור לי בהמשך בלמנוע בעיות בפרויקטים עתידיים.

בנוסף, התמצאות במודולי בינה מלאכותית מתקדמים כמו Claude ו-ChatGPT היא מיומנות חשובה בעולם התוכנה המשתנה במהירות. היכולת להסתגל לשינויים וללמוד טכנולוגיות חדשות באופן מתמיד חשובה לי ולעומת זאת גם חשוב לדעת לאזן ולקבל מידע ממקורות שונים ואמינים.

תובנות מהתהליך

במהלך השנה היה לי מאוד כיף ומלמד לעבוד על הפרויקט, במשך שלושת השנים שלי בתיכון למדתי הרבה דברים שונים בהקשר של מדעי המחשב (שפות תכנות, אלגוריתמים שונים, כתיבת תיעוד לקוד, רשתות, תקשורת מחשבים ועוד) ואני מרגיש שכל חלק עזר לי בתכנון הפרויקט, כתיבת הקוד ותיק הפרויקט.

בתחילת השנה חששתי מאוד מהממשק הגרפי של הלקוח (frontend) פחדתי שלא אצליח לעצב ממשק יפה שירצה אותי ועכשיו לאחר סיום הפרויקט אני יכול להגיד שגם בממשק יש לי ניסיון והצלחתי במשימה שלי.

כיצד ניתן לשפר את הפרויקט?

בראייה לאחור אם היה לי יותר זמן הייתי מוסיף אפשרות של שיתוף קבצים בין לקוחות, בתחילת השנה רציתי להוסיף את האפשרות הזאת אבל במקום זאת התמקדתי בהצפנת הקבצים ושיפור האבטחה. הייתי שמח להוסיף את האפשרות הזאת לאחר הגשת הפרויקט מכיוון שהיא נראית לי מאוד מגניבה עם הרבה אפשרויות ללמוד.

בנוסף לכך הייתי מוסיף certificate לשרת וללקוח, לא הספקתי להבין איך אפשר ליישם תהליך כזה אבל הייתי רוצה להוסיף זאת; בנוסף הייתי מוסיף עוד חלקי הגנה כמו blacklist של כתובות IP.

ביבליוגרפיה

מודולי הבינה המלכותית שנעזרתי בהם

OpenAI. (2023). ChatGPT (GPT-4). [Chat GPT](#)

Anthropic. (2023). Claude. [Claude](#)

תיעוד של הסיפריות החיצוניות שהשמשתי בהם

Aviramha. (2024). Git - [msgpack](#)

Legrandin PyCryptoDome.(2024). [PyCryptoDome](#)

ספר רשתות

מאוד עוזר בלמידת וכאשר הייתי צריך להזכר מחדש על פרוטוקולים שונים-

עומר רוזנבוים, ברק גונן, תומר גביש, מתן זינגר, רמי עמר, שלומי, בוטנרו, שלומי הוד
(2020) - [ספר רשתות](#)

Flaticon

השתמשתי באתר הבא בשביל אייקונים לכפתורים בצד המשתמש-

Flaticon (2024). Flaticon - [icons](#).

נספחים

Protocol.py

```
import msgpack
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad

ZFILL_LENGTH = 10

class AESCipher:
    def __init__(self, key, iv):
        self.key = key
        self.iv = iv

    def encrypt(self, message):
        aes_cipher = AES.new(self.key, AES.MODE_CBC, self.iv)
        padded_plaintext = pad(message, AES.block_size)
        encrypted_message = aes_cipher.encrypt(padded_plaintext)
        return encrypted_message

    def decrypt(self, encrypted_message):
        aes_cipher = AES.new(self.key, AES.MODE_CBC, self.iv)
        decrypted_data = aes_cipher.decrypt(encrypted_message)
        plaintext = unpad(decrypted_data, AES.block_size)
        return plaintext

def send_message_aes(dict1, aes_cipher):
    """
    encode the given msg by the protocol standard
    :param dict1: the msg to encode
    :param aes_cipher:
    :return: the msg after encoding
    """
    print(dict1)
    message = b''
    packed_message = msgpack.packb(dict1)
    encrypted_message = aes_cipher.encrypt(packed_message)
    message += str(len(encrypted_message)).zfill(ZFILL_LENGTH).encode() +
    encrypted_message
    return message

def get_message_aes(my_socket, aes_cipher):
    """
    receive a msg from the server/client and decode it by the protocol
    standard
    :param my_socket:
    :param aes_cipher:
    :return:
    """
```

```

"""
exit1 = False
length = ""
while not exit1:
    length = recvall(my_socket, ZFILL_LENGTH)
    if length is None:
        exit1 = False
    else:
        exit1 = True
length = int(length.decode())
encrypted_message = recvall(my_socket, length)
decrypted_message = aes_cipher.decrypt(encrypted_message)
print(msgpack.unpackb(decrypted_message))
return msgpack.unpackb(decrypted_message)

def send_message(dict1):
    """
    encode the given msg without the encryption
    only used for the first connection
    :param dict1:
    :return:
    """
    message = b''
    packed_message = msgpack.packb(dict1)
    message += str(len(packed_message)).zfill(ZFILL_LENGTH).encode() +
packed_message
    return message

def get_message(my_socket):
    """
    receive a msg from the server/client and decode it without the
    encryption
    only used for the first connection
    :param my_socket:
    :return: the msg after decoding
    """
    exit1 = False
    length = ""
    while not exit1:
        length = recvall(my_socket, ZFILL_LENGTH)
        if length is None:
            exit1 = False
        else:
            exit1 = True
    length = int(length.decode())
    print()
    return msgpack.unpackb(recvall(my_socket, length))

def recvall(sock, size):
    """
    used to ensure the socket receive the complete packet by using its
    packet size
    :param sock:
    :param size:
    :return: msg

```

```

"""
data = b''
while len(data) < size:
    packet = sock.recv(size - len(data))
    if not packet:
        return None # Connection closed prematurely
    data += packet
return data

```

Logger.py

```

import logging

class Logger:
    """
    multiple functions to log the server actions into a file
    """
    @staticmethod
    def log_upload(username, file_name):
        logging.basicConfig(filename='user_actions.log',
            level=logging.INFO, format='%(asctime)s - %(message)s', datefmt='%Y-%m-%d
            %H:%M:%S')
        logging.info(f"{username} uploaded file: {file_name}")

    @staticmethod
    def log_delete(username, file_or_folder_name):
        logging.basicConfig(filename='user_actions.log',
            level=logging.INFO, format='%(asctime)s - %(message)s', datefmt='%Y-%m-%d
            %H:%M:%S')
        logging.info(f"{username} deleted {file_or_folder_name}")

    @staticmethod
    def log_login(username):
        logging.basicConfig(filename='user_actions.log',
            level=logging.INFO, format='%(asctime)s - %(message)s', datefmt='%Y-%m-%d
            %H:%M:%S')
        logging.info(f"{username} logged in")

    @staticmethod
    def log_admin_login(username):
        logging.basicConfig(filename='user_actions.log',
            level=logging.INFO, format='%(asctime)s - %(message)s', datefmt='%Y-%m-%d
            %H:%M:%S')
        logging.info(f"{username} admin logged in")

    @staticmethod
    def log_signup(username):
        logging.basicConfig(filename='user_actions.log',
            level=logging.INFO, format='%(asctime)s - %(message)s', datefmt='%Y-%m-%d
            %H:%M:%S')
        logging.info(f"{username} signed up")

    @staticmethod
    def log_download(username, file_name):
        logging.basicConfig(filename='user_actions.log',
            level=logging.INFO, format='%(asctime)s - %(message)s', datefmt='%Y-%m-%d
            %H:%M:%S')

```

```

        logging.info(f"{username} downloaded file: {file_name}")

    @staticmethod
    def log_error(string):
        logging.basicConfig(filename='user_actions.log',
            level=logging.INFO, format='%(asctime)s - %(message)s', datefmt='%Y-%m-%d
            %H:%M:%S')
        logging.info(f"error: {string}")

    @staticmethod
    def log_email_sent(email, string):
        logging.basicConfig(filename='user_actions.log',
            level=logging.INFO, format='%(asctime)s - %(message)s', datefmt='%Y-%m-%d
            %H:%M:%S')
        logging.info(f"email was sent to {email}: {string}")

    @staticmethod
    def log_account_deletion(username, username_admin):
        logging.basicConfig(filename='user_actions.log',
            level=logging.INFO, format='%(asctime)s - %(message)s', datefmt='%Y-%m-%d
            %H:%M:%S')
        logging.info(f"Admin {username_admin} deleted user:{username}")

```

Create_venv.py

```

import os
import venv

# Set the name of the virtual environment directory
venv_dir = "myenv"

# Create the virtual environment
venv.create(venv_dir, with_pip=True)

# Get the path to the Python executable in the virtual environment
python_executable = os.path.join(venv_dir, "Scripts", "python.exe")

# Install the required packages using pip
packages = ["msgpack", "pycryptodome"]
for package in packages:
    os.system(f'"{python_executable}" -m pip install {package}')

```

Run_Nimbus.bat

```

@echo off

REM Set the path to the virtual environment directory relative to the
current directory
set VENV_DIR=%~dp0myenv

REM Check if the virtual environment exists
if not exist "%VENV_DIR%" (

```

```

        echo Creating virtual environment and downloading packages...
        REM Create the virtual environment and install packages
        python create_venv.py
        echo Done!
    )

    REM Activate the virtual environment
    call "%ENV_DIR%\Scripts\activate.bat"

    REM Run the Client.py file using the Python interpreter from the virtual
    environment
    start "" pythonw Client.py

    REM Deactivate the virtual environment
    call myenv\Scripts\deactivate

```

Client.py

```

import socket
import os
import sys
import tkinter as tk
import tkinter.ttk as ttk
from tkinter import filedialog
import client_admin
import protocol
import zipfile
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from Crypto.Random import get_random_bytes

SERVER_HOST = '127.0.0.1'
SERVER_PORT = 12345
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
selected_buttons = set()
client_path = "MAIN"
current_page = 1
total_pages = 1
client_connected = False
admin_connected = False
MAXIMUS_SIZE = 1073741824

class FileItem:
    def __init__(self, name, file_type, max_display_length=20):
        self.full_name = name
        self.file_type = file_type
        self.display_name = FileItem.truncate_name(name,
max_display_length)

    @staticmethod
    def truncate_name(name, max_length):
        if len(name) > max_length:
            return f"{name[:max_length-3]}..."
        else:
            return name

```

```

def __str__(self):
    return self.display_name

def __repr__(self):
    return self.full_name

def get_full_name(self):
    return self.full_name

def read_file_content(file_path):
    """
    Read the content from a given path
    :param file_path:
    :return: file content in bytes form
    """
    try:
        with open(file_path, 'rb') as file:
            return file.read()
    except FileNotFoundError:
        print(f"Error: File '{file_path}' not found.")
    except Exception as e:
        print(f"Error: {e}")

def print_bytes(bytes_value):
    """
    Converts a byte value into a human-readable format
    :param bytes_value:
    :return: string that contain the size and suffix
    """
    suffixes = ['B', 'KB', 'MB', 'GB']
    suffix_index = 0

    while bytes_value >= 1024 and suffix_index < len(suffixes) - 1:
        bytes_value /= 1024.0
        suffix_index += 1

    return f"{bytes_value:.2f} {suffixes[suffix_index]}"

def file_explorer_ui(aes_cipher):
    """
    The main function for the homepage of a regular user after connection
    to his user
    The function is responsible for rendering the UI and starting the
    functions based on the user buttons presses
    The user leave this part of function only when he closes the program
    :param aes_cipher:
    :return: nothing
    """
    client_path = "MAIN"
    current_page = 1

    def open_folder():
        """
        responsible for the button "open_folder"

```

```

    when a user selects a folder that he want to "get into" and see
    the files inside the folder
    changes the global var client_path
    :return:
    """

    global client_path
    # Get the selected buttons
    selected_buttons_list = list(selected_buttons)
    # Check if only one item is selected
    if len(selected_buttons_list) != 1:
        print("Please select only one item.")
        return

    # Check if the selected item is a folder
    selected_item = selected_buttons_list[0]
    if selected_item.cget("image") != str(subsampled_icon_folder):
        print("Please select a folder, not a file.")
        return

    # Get the folder name
    folder_name = filename_map.get(selected_item).get_full_name()
    # Update the client path
    client_path = os.path.join(client_path, folder_name)
    print(f"Opening folder: {client_path}")
    refresh()

def exit_folder():
    """
    responsible for the button "exit_folder"
    when a user wants to exit to the parent directory
    changes the global var client_path
    :return:
    """

    global client_path
    # Get the parent directory of the current path
    if client_path != "MAIN":
        client_path = os.path.dirname(client_path)
        print(f"Exiting to: {client_path}")
        refresh()

def create_folder():
    """
    responsible for the button "create_folder"
    when a user wants to create a new function he can name it and send
    the request to the server.
    :return:
    """

    global client_path
    folder_name = ask_string_popup("Enter folder name:")
    if folder_name:
        create_folder_dict = {"command": "create_folder",
                              "folder_name": os.path.join(client_path[5:], folder_name)}

    client_socket.sendall(protocol.send_message_aes(create_folder_dict,
                                                    aes_cipher))
    response_dict = protocol.get_message_aes(client_socket,
                                              aes_cipher)
    msg = response_dict.get("msg")

```

```

        if msg == "Folder created successfully":
            refresh()
        else:
            show_popup(msg)

    def refresh():
        """
        responsible for the button "refresh"
        this function is called upon with many other functions because his
        responsible for refreshing
        request the files and folders to display from the server with the
        global var client_path.
        :return:
        """

        global current_page, total_pages, client_path
        print("refresh")
        refresh_dict = {"command": "refresh", "path": client_path, "page":
current_page, "rows": rows, "cols": cols}
        path_label.config(text=client_path)
        client_socket.sendall(protocol.send_message_aes(refresh_dict,
aes_cipher))
        refresh_response_dict = protocol.get_message_aes(client_socket,
aes_cipher)

        # Get the file_size, files, and total_pages from the response
        file_size = refresh_response_dict.get("file_size")
        files = refresh_response_dict.get("files")
        total_pages = refresh_response_dict.get("total_pages")
        folder_size_label.config(text=print_bytes(file_size))

        # Clear the buttons array
        for current_row in buttons:
            for button1 in current_row:
                button1.grid_remove()

        # Iterate over the files and update the buttons
        current_row = 0
        current_col = 0
        for file_name, file_type in files.items():
            file_item = FileItem(file_name, file_type)
            if current_row < rows and current_col < cols:
                icon_image = subsampled_icon_file if file_type == "file"
            else:
                icon_image = subsampled_icon_folder

            buttons[current_row][current_col].config(text=str(file_item),
            image=icon_image, font=("Arial", 12), compound="top",
            relief="raised",
            command=lambda r=current_row, c=current_col: button_click(r, c))
            buttons[current_row][current_col].grid()
            filename_map[buttons[current_row][current_col]] =
file_item
            current_col += 1
            if current_col == cols:
                current_col = 0
                current_row += 1
        selected_buttons.clear()

    def delete():

```



```

"""
responsible for the button "Delete"
confirm if the user wants to delete the files
create a list of paths of the files that he wants to delete
calls the function delete_files with the list
:return:
"""

global client_path
files_list_items = [filename_map.get(button1) for button1 in
selected_buttons]
files_list = [item.get_full_name() for item in files_list_items]
print(f"files list: {files_list}")

files_with_path = [os.path.join(client_path, filename) for
filename in files_list]
if len(files_list) > 5 or any(item.file_type == "folder" for item
in files_list_items):
    amount_of_items = len(files_list)
    confirmation_text = f"Are you sure you want to delete
{amount_of_items} items?"
    confirmed = show_confirmation_popup(confirmation_text)
    if confirmed:
        delete_files(files_with_path)
    else:
        delete_files(files_with_path)

def delete_files(files_with_path):
    """
    send a request to the server to delete list of files and folders
:param files_with_path: list of the files_path the user wants to
delete
:return:
"""
    delete_dict = {"command": "delete", "files": files_with_path}
    client_socket.sendall(protocol.send_message_aes(delete_dict,
aes_cipher))
    response_dict = protocol.get_message_aes(client_socket,
aes_cipher)
    refresh()

def download():
    """
    Downloads selected files from the server to the local machine's
Downloads folder.
Global Variables:
    - client_path (str): Path to the client directory within the
UI
    This function performs the following steps:
    1. Retrieves the list of selected files based on the user's
selection.
    2. Constructs the file paths the selected files.
    3. Sends a download request to the server with the list of
selected files.
    4. Receives the files from the server, handling potential
responses indicating
        that no files were selected.
    5. Saves the received files to the Downloads folder, ensuring
unique filenames

```

```

        to avoid conflicts.
        6. If a folder is downloaded, it is saved as a zip file,
        extracted, and the zip
        file is then removed.
        :return:
        """
        global client_path
        # construct a list with the file paths
        files_list_items = [filename_map.get(button1) for button1 in
selected_buttons]
        files_list = [item.get_full_name() for item in files_list_items]
        print(f"files list: {files_list}")
        files_with_path = [os.path.join(client_path, filename) for
filename in files_list]
        # send the request to the server
        download_dict = {"command": "download", "files": files_with_path}
        client_socket.sendall(protocol.send_message_aes(download_dict,
aes_cipher))
        response_dict = protocol.get_message_aes(client_socket,
aes_cipher)

        downloads_folder = os.path.expanduser("~") + os.sep + "Downloads"
        files = response_dict.get("files")
        print(files)
        if files == "no files selected":
            return
        for file_info in files:
            file_name = file_info["file_name"]
            file_type = file_info["file_type"]
            file_content = file_info["file_Content"]

            file_path = os.path.join(downloads_folder,
f"{file_name}{file_type}")
            new_file_name = file_name
            counter = 1
            while os.path.exists(file_path):
                new_file_name = f"{file_name}({counter})"
                file_path = os.path.join(downloads_folder,
f"{new_file_name}{file_type}")
                counter += 1
            file_name = new_file_name

            if file_type == "folder":
                # Handle the case when the user downloads a folder
                folder_name = os.path.splitext(file_name)[0] # Remove the
.zip extension
                unzipped_folder_path = os.path.join(downloads_folder,
folder_name)
                # Added a loop to handle folder name conflicts
                counter = 1
                while os.path.exists(unzipped_folder_path):
                    new_folder_name = f"{folder_name}({counter})"
                    unzipped_folder_path = os.path.join(downloads_folder,
new_folder_name)
                    counter += 1
                zip_file_path = os.path.join(downloads_folder, file_name)
                with open(zip_file_path, "wb") as f:
                    f.write(file_content)

```

```

        print(f"Folder saved: {zip_file_path}")

        # Unzip the folder
        with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
            for member in zip_ref.infolist():
                extracted_path = zip_ref.extract(member,
unzipped_folder_path)
                if member.is_dir():
                    os.makedirs(extracted_path, exist_ok=True)
                print(f"Folder extracted: {unzipped_folder_path}")
            # Remove the zip file
            os.remove(zip_file_path)
        else:
            # Handle the case when the user downloads a file
            filepath = os.path.join(downloads_folder, file_name +
file_type)
            with open(filepath, "wb") as f:
                f.write(file_content)
            print(f"File saved: {filepath}")
        refresh()

def upload_file():
    """
    responsible for the button "upload_file"
    open up the file explorer of the user and lets him select a file
    send a request of the server of the upload_file
    :return:
    """
    global client_path
    print("upload file")
    try:
        file_path =
filedialog.askopenfilename(initialdir=os.path.expanduser("~") + os.sep +
"Desktop",
                                title="Open file",
                                filetypes=[
                                    ("Common File
Types",
".*.txt;*.docx;*.doc;*.xlsx;*.xls;*.pptx;*.ppt;*.pdf;*.jpg;*.jpeg;*.png;*.
gif;*.mp4;*.avi;*.mov;*.mp3;*.m4a;*.zip;*.rar;*.7z;*.gz;*.tar;"),
                                    ("All Files",
".*.*")
                                ])
        if file_path:
            file_name, file_type =
os.path.splitext(os.path.basename(file_path))
            file_name = os.path.join(client_path, file_name)
            file_content = read_file_content(file_path)
            if len(file_content) < MAXIMUS_SIZE:
                upload_dict = {"command": "upload_file", "file_name":
file_name, "file_type": file_type, "file_Content": file_content}
            client_socket.sendall(protocol.send_message_aes(upload_dict, aes_cipher))
            refresh()
        else:
            print("file too large")
    
```

```

        show_popup("the file is too large")
    else:
        print("No file selected")
except Exception as e:
    print(f"Error uploading file/folder: {e}")
    show_popup(f"Error uploading file, {str(e)}")

def upload_folder():
    """
    responsible for the button "upload_folder"
    open up the file explorer of the user and lets him select a folder
    the client zip the folder and send it as a zip file
    send a request of the server of the upload_folder
    :return:
    """
    global client_path
    print("upload folder")
    try:
        folder_path =
filedialog.askdirectory(initialdir=os.path.expanduser("~") + os.sep +
"Desktop",
                                title="Open file")
        if folder_path:
            zip_file_name = os.path.basename(folder_path) + ".zip"

            # Create a zip file containing the existing folder
            with zipfile.ZipFile(zip_file_name, 'w') as zip_file:
                # Iterate over the files and directories in the folder
                for root1, dirs, files in os.walk(folder_path):
                    # Add empty directories to the zip file
                    for dir_name in dirs:
                        dir_path = os.path.join(root1, dir_name)
                        zip_file.write(dir_path,
os.path.relpath(dir_path, folder_path))

                    # Add files to the zip file
                    for file in files:
                        file_path = os.path.join(root1, file)
                        zip_file.write(file_path,
os.path.relpath(file_path, folder_path))

            # Read the contents of the zip file
            with open(zip_file_name, 'rb') as file:
                file_content = file.read()

            if len(file_content) < MAXIMUS_SIZE:
                temp_zip_file_name = os.path.join(client_path,
zip_file_name)

                upload_dict = {"command": "upload_folder",
"folder_name": temp_zip_file_name, "folder_content": file_content}
client_socket.sendall(protocol.send_message_aes(upload_dict, aes_cipher))
                refresh()
            else:
                print("folder too large")
                show_popup("folder too large")

```

```

        # Remove the temporary zip file
        os.remove(zip_file_name)
    else:
        print("No folder selected")
except Exception as e:
    print(f"Error uploading file/folder: {e}")
    show_popup(f"Error uploading folder, {str(e)}")

def show_popup(string1):
    """
    responsible for a popup msg that can appear in different
    situations
    :param string1: that can be many kinds of strings, it will render
    this string in the popup
    :return:
    """
    popup = tk.Toplevel(root)
    popup.title("Nimbus")
    popup.iconbitmap('icons//Icon_no_text.ico')

    popup_frame = ttk.Frame(popup, padding=10)
    popup_frame.pack(fill="both", expand=True)

    label = ttk.Label(popup_frame, text=string1)
    label.pack(pady=20)

    ok_button = ttk.Button(popup_frame, text="OK",
    command=popup.destroy)
    ok_button.pack()

    # Center the popup window on the screen
    popup.update_idletasks() # Update geometry information
    window_width = 400 # Increase the width to 400 pixels
    window_height = popup.winfo_height()
    screen_width = popup.winfo_screenwidth()
    screen_height = popup.winfo_screenheight()
    x = (screen_width - window_width) // 2
    y = (screen_height - window_height) // 2
    popup.geometry(f"{window_width}x{window_height}+{x}+{y}")

def show_confirmation_popup(confirmation_text):
    """
    responsible for a popup msg that can appear in different
    situations
    it renders the msg with buttons of confirmation "yes" and "no"
    :param confirmation_text: the string to render in the popup
    :return: a bool that represent what the user pressed
    """
    popup = tk.Toplevel(root)
    popup.title("Confirmation")
    popup.iconbitmap('icons//Icon_no_text.ico')

    popup_frame = ttk.Frame(popup, padding=10)
    popup_frame.pack(fill="both", expand=True)

    label = ttk.Label(popup_frame, text=confirmation_text)

```

```

label.pack(pady=20)

result = None

def confirm_yes():
    nonlocal result
    result = True
    popup.destroy()

def confirm_cancel():
    nonlocal result
    result = False
    popup.destroy()

yes_button = ttk.Button(popup_frame, text="Yes",
command=confirm_yes)
yes_button.pack(side="left", padx=10)

cancel_button = ttk.Button(popup_frame, text="Cancel",
command=confirm_cancel)
cancel_button.pack(side="right", padx=10)

# Center the popup window on the screen
popup.update_idletasks() # Update geometry information
window_width = 400
window_height = popup.winfo_height()
screen_width = popup.winfo_screenwidth()
screen_height = popup.winfo_screenheight()
x = (screen_width - window_width) // 2
y = (screen_height - window_height) // 2
popup.geometry(f"{window_width}x{window_height}+{x}+{y}")

popup.wait_window(popup)
return result if result is not None else False

def ask_string_popup(prompt):
    """
    responsible for a popup msg that can appear in different
    situations
    it renders the msg with an input area of a string
    :param prompt: the string to render in the popup
    :return: the string that the user wrote
    """
    popup = tk.Toplevel(root)
    popup.title("Nimbus")
    popup.iconbitmap('icons//Icon_no_text.ico')

    popup_frame = ttk.Frame(popup, padding=10)
    popup_frame.pack(fill="both", expand=True)

    label = ttk.Label(popup_frame, text=prompt)
    label.pack(pady=10)

    entry = ttk.Entry(popup_frame)
    entry.pack(pady=5)

    def on_ok(arg=None):

```

```

        popup.result = entry.get()
        popup.destroy()

    popup.bind('<Return>', on_ok)

    ok_button = ttk.Button(popup_frame, text="OK", command=on_ok)
    ok_button.pack(pady=10)

    # Center the popup window on the screen
    popup.update_idletasks() # Update geometry information
    window_width = 300
    window_height = popup.winfo_height()
    screen_width = popup.winfo_screenwidth()
    screen_height = popup.winfo_screenheight()
    x = (screen_width - window_width) // 2
    y = (screen_height - window_height) // 2
    popup.geometry(f"{window_width}x{window_height}+{x}+{y}")

    popup.result = None
    popup.wait_window(popup)
    return popup.result if popup.result is not None else ""

def previous_page():
    """
    responsible for the button "previous_page"
    incase the user have to many files to display he can move pages,
    this allows him to move backwards
    :return:
    """
    global current_page
    if current_page > 1:
        current_page -= 1
        page_label.config(text=current_page)
        refresh()

def next_page():
    """
    responsible for the button "next_page"
    incase the user have to many files to display he can move to the
    next page
    :return:
    """
    global current_page, total_pages
    if current_page < total_pages:
        current_page += 1
        page_label.config(text=current_page)
        refresh()

root = tk.Tk()
root.title("Nimbus")
root.iconbitmap('icons//Icon_no_text.ico')

def on_closing():
    """
    responsible for the closure of the program, when a user exit the
    program this function is called
    sends a msg to the server that the client disconnected and end the

```

```

program
    :return:
    """
    print("closed")
    closing_connection_dict = {"command": "exit"}

    client_socket.sendall(protocol.send_message_aes(closing_connection_dict,
    aes_cipher))
    root.destroy()
    sys.exit()

    root.protocol("WM_DELETE_WINDOW", on_closing)

    # Apply ttk style for a modern look
    style = ttk.Style()
    style.theme_use('clam')

    # Define custom colors
    background_color = "#f0f0f0"
    button_color = "#4CAF50"
    button_text_color = "#FFFFFF"

    # Configure style elements
    style.configure('TFrame', background=background_color)
    style.configure('TButton', background=button_color,
    foreground=button_text_color)
    style.configure('TLabel', background=background_color,
    font=('TkDefaultFont', 12, 'bold'))
    style.configure('TCanvas', background=background_color)
    style.configure('TSeparator', thickness=2)

    # Set initial window size
    root.geometry("1000x600") # Width x Height

    # Create frames with styled background
    left_frame = ttk.Frame(root) # Use ttk.Frame for styling
    left_frame.pack(side="left", fill="y") # Fill vertically

    right_frame = ttk.Frame(root)
    right_frame.pack(side="right", fill="both", expand=True)

    # Create thicker styled line between frames
    line = ttk.Separator(root, orient="vertical") # Use ttk.Separator for
styling
    line.pack(side="left", fill="y", padx=5)

    icon_image_open_folder = tk.PhotoImage(file="icons//open_folder.png")
    subsampled_icon_open_folder = icon_image_open_folder.subsample(16, 16)
# Reduce size

    icon_image_exit_folder = tk.PhotoImage(file="icons//exit_folder.png")
    subsampled_icon_exit_folder = icon_image_exit_folder.subsample(16, 16)
# Reduce size

    # Create styled widgets in left frame using ttk
    open_folder_button = ttk.Button(left_frame, text="Open folder",
image=subsampled_icon_open_folder, compound="top", command=open_folder)
    open_folder_button.pack(side="top")

```



```

exit_folder_button = ttk.Button(left_frame, text="Exit folder",
image=subsampled_icon_exit_folder, compound="top", command=exit_folder)
exit_folder_button.pack(side="top")

icon_image_create_folder =
tk.PhotoImage(file="icons//create_folder.png")
subsampled_icon_image_create_folder =
icon_image_create_folder.subsample(16, 16) # Reduce size

create_folder_button = ttk.Button(left_frame, text="Create folder",
image=subsampled_icon_image_create_folder, compound="top",
command=create_folder)
create_folder_button.pack(side="top")

icon_image_refresh = tk.PhotoImage(file="icons//refresh.png")
subsampled_icon_refresh = icon_image_refresh.subsample(16, 16) #
Reduce size

refresh_button = ttk.Button(left_frame, text="Refresh",
image=subsampled_icon_refresh, compound="top", command=refresh)
refresh_button.pack(side="top")

folder_size_label = ttk.Label(left_frame, text="")
folder_size_label.pack(side="bottom")

# Load icon image
icon_image_delete = tk.PhotoImage(file="icons//delete.png")
subsampled_icon_delete = icon_image_delete.subsample(16, 16) # Reduce
size

icon_image_upload_file = tk.PhotoImage(file="icons//upload_file.png")
subsampled_icon_upload_file = icon_image_upload_file.subsample(16, 16)
# Reduce size

icon_image_upload_folder =
tk.PhotoImage(file="icons//upload_folder.png")
subsampled_icon_upload_folder = icon_image_upload_folder.subsample(16,
16) # Reduce size

icon_image_download = tk.PhotoImage(file="icons//download.png")
subsampled_icon_download = icon_image_download.subsample(16, 16) #
Reduce size

# Create styled text label in right frame with bold font
path_label = ttk.Label(right_frame, text=client_path)
path_label.pack(anchor="n", fill="x", padx=10, pady=5) # Anchor to
top, fill horizontally, and add padding

# Create thicker styled horizontal line under the text label
line_horizontal = ttk.Separator(right_frame, orient="horizontal") #
Use ttk.Separator for styling
line_horizontal.pack(anchor="s", fill="x", padx=5, pady=5)

# Create buttons frame in right frame
buttons_frame = ttk.Frame(right_frame)
buttons_frame.pack(side="bottom", fill="x")

```

```

# Create styled buttons in buttons frame using ttk
delete_button = ttk.Button(buttons_frame, text="Delete",
image=subsampled_icon_delete, compound="top",
                           command=delete)
delete_button.pack(side="left")

upload_button_file = ttk.Button(buttons_frame, text="Upload File",
image=subsampled_icon_upload_file, compound="top",
                           command=upload_file)
upload_button_file.pack(side="left")

upload_button_folder = ttk.Button(buttons_frame, text="Upload Folder",
image=subsampled_icon_upload_folder, compound="top",
                           command=upload_folder)
upload_button_folder.pack(side="left")

download_button = ttk.Button(buttons_frame, text="Download",
image=subsampled_icon_download, compound="top", command=download)
download_button.pack(side="left")

# Create a new frame for the "Next" and "Previous" buttons
nav_buttons_frame = ttk.Frame(buttons_frame)
nav_buttons_frame.pack(side="right", padx=10) # Add some padding to
the right

icon_image_left = tk.PhotoImage(file="icons//left-arrow.png")
subsampled_icon_left = icon_image_left.subsample(16, 16) # Reduce
size

icon_image_right = tk.PhotoImage(file="icons//right-arrow.png")
subsampled_icon_right = icon_image_right.subsample(16, 16) # Reduce
size

# Create the "Next" and "Previous" buttons
next_button = ttk.Button(nav_buttons_frame, text="Next",
image=subsampled_icon_right, compound="top",
                           command=next_page)
next_button.pack(side="right", padx=5) # Add some padding between the
buttons

page_label = ttk.Label(nav_buttons_frame, text=current_page)
page_label.pack(side="right", padx=5)

previous_button = ttk.Button(nav_buttons_frame, text="Previous",
image=subsampled_icon_left, compound="top",
                           command=previous_page)
previous_button.pack(side="right")

files_frame = ttk.Frame(right_frame)
files_frame.pack(fill="x")

icon_image_file = tk.PhotoImage(file="icons//file.png")
subsampled_icon_file = icon_image_file.subsample(16, 16) # Reduce
size

icon_image_folder = tk.PhotoImage(file="icons//folder.png")
subsampled_icon_folder = icon_image_folder.subsample(16, 16) # Reduce

```

```

size

rows = 5
cols = 6
buttons = []
filename_map = {}

def button_click(row_index1, col_index1):
    """
    responsible to update the global var of the selected buttons
    when the user press any of the files/folder buttons it calls this
    function
    :param row_index1: the row index of the button that he selected
    :param col_index1: the colum index of the button that he selected
    :return:
    """
    button1 = buttons[row_index1][col_index1]
    if button1 in selected_buttons:
        # Button is already selected, so unselect it
        selected_buttons.remove(button1)
        button1.config(relief="raised") # Change the button's
        appearance to unselected state
    else:
        # Button is not selected, so select it
        selected_buttons.add(button1) # For set
        # selected_buttons.append(button) # For list
        button1.config(relief="sunken") # Change the button's
        appearance to selected state

    print(f"Selected buttons: {[filename_map.get(button1) for button1
in selected_buttons]}")

for row_index in range(rows):
    row = []
    for col_index in range(cols):
        button = tk.Button(files_frame, text=f"Button
{row_index},{col_index}", font=("Arial", 12), image=subsampled_icon_file,
                           compound="top", padx=20, pady=10,
command=lambda r=row_index, c=col_index: button_click(r, c))
        button.grid(row=row_index, column=col_index, sticky="nsew")
        row.append(button)
    buttons.append(row)

# Make all buttons expand and fill the frame
for row_index in range(rows):
    files_frame.grid_rowconfigure(row_index, weight=1)
for col_index in range(cols):
    files_frame.grid_columnconfigure(col_index, weight=1)

refresh()
root.mainloop()

def login_signup(aes_cipher):
    """
    The main function for the login/signup/forgot_password
    The function is responsible for rendering the UI and starting the

```

```

functions based on the user buttons presses
    The user leave this part of function only when he logs in
    successfully.
    param aes_cipher: the aes_cipher object that responsible for the
    encryption and decryption with the protocol.py
    :return: a confirmation to the main function if a normal user logged
    in or and admin
    """
    global client_connected, admin_connected
    root = tk.Tk()
    root.withdraw() # Hide the main window

    def on_closing():
        """
        responsible for the closure of the program, when a user exit the
        program this function is called
        sends a msg to the server that the client disconnected and end the
        program
        :return:
        """
        print("closed")
        closing_connection_dict = {"command": "exit"}

    client_socket.sendall(protocol.send_message_aes(closing_connection_dict,
    aes_cipher))
    root.destroy()
    sys.exit()

    dialog = tk.Toplevel(root)
    dialog.protocol("WM_DELETE_WINDOW", on_closing)
    dialog.title("Nimbus")
    dialog.geometry("400x400")

    dialog.iconbitmap('icons//Icon_no_text.ico')

    # Create style for modern look (optional)
    style = ttk.Style(dialog)
    style.theme_use('clam')

    # Custom colors
    background_color = "#f0f0f0"
    button_color = "#4CAF50"
    button_text_color = "#FFFFFF"

    # Configure style elements
    style.configure('TButton', background=button_color,
    foreground=button_text_color)
    style.configure('TFrame', background=background_color)
    style.configure('TLabel', background=background_color)

    # Create a notebook to switch between login and signup frames
    notebook = ttk.Notebook(dialog)
    notebook.pack(fill=tk.BOTH, expand=True)

    # Login frame
    login_frame = ttk.Frame(notebook, padding=10)
    notebook.add(login_frame, text="Login")

```

```

icon_image = tk.PhotoImage(file="icons//icon.png")
subsamped_icon = icon_image.subsample(6, 6) # Reduce size

icon_image_label = ttk.Label(login_frame, image=subsamped_icon)
icon_image_label.pack()

# Login email label and entry
login_email_label = ttk.Label(login_frame, text="Email:")
login_email_label.pack()

login_email_entry = ttk.Entry(login_frame, width=30)
login_email_entry.pack()

# Login password label and entry
login_password_label = ttk.Label(login_frame, text="Password:")
login_password_label.pack()

login_password_entry = ttk.Entry(login_frame, width=30, show="*")
login_password_entry.pack()

def login(arg=None):
    """
    responsible for sending the server a request to login
    rendering the response from the server in the UI
    :param arg: the input that the client enter: email,password
    :return: a confirmation to the previous function if a normal user
    logged in or and admin
    """
    global client_connected, admin_connected
    print("Login clicked")
    email_entry = login_email_entry.get()
    password_entry = login_password_entry.get()
    print(email_entry)
    print(password_entry)
    login_dict = {"command": "login", "email": email_entry,
"username": "", "password": password_entry}
    client_socket.sendall(protocol.send_message_aes(login_dict,
aes_cipher))
    response_dict = protocol.get_message_aes(client_socket,
aes_cipher)
    msg = response_dict.get("msg")
    print(msg)
    if msg == "Login successful":
        message_label_login.config(text=msg)
        client_connected = True
        root.destroy()
    elif msg == "Admin connected":
        admin_connected = True
        root.destroy()
    elif msg is not None and msg.startswith("Wrong password"):
        message_label_login.config(text=msg)
    elif msg == "Too many failed login attempts. Please try again
later.":
        message_label_login.config(text=msg)
        login_password_entry.config(state="disabled")
        login_button.config(state="disabled")
        login_frame.unbind('<Return>') # Disable the Enter key

```

```

binding
    else:
        message_label_login.config(text=msg)

    login_frame.bind('<Return>', login)
    login_password_entry.bind('<Return>', login)
    login_email_entry.bind('<Return>', login)

    # Login button
    login_button = ttk.Button(login_frame, text="Login", command=login)
    login_button.pack(pady=10)

    # Login message label
    login_message_label = ttk.Label(login_frame, text="")
    login_message_label.pack()

    # Signup frame
    signup_frame = ttk.Frame(notebook, padding=10)
    notebook.add(signup_frame, text="Sign-Up")

    icon_image_label_signup = ttk.Label(signup_frame,
image=subsampled_icon)
    icon_image_label_signup.pack()

    # Signup email label and entry
    signup_email_label = ttk.Label(signup_frame, text="Email:")
    signup_email_label.pack()

    signup_email_entry = ttk.Entry(signup_frame, width=30)
    signup_email_entry.pack()

    # Signup username label and entry
    signup_username_label = ttk.Label(signup_frame, text="Username:")
    signup_username_label.pack()

    signup_username_entry = ttk.Entry(signup_frame, width=30)
    signup_username_entry.pack()

    # Signup password label and entry
    signup_password_label = ttk.Label(signup_frame, text="Password:")
    signup_password_label.pack()

    signup_password_entry = ttk.Entry(signup_frame, width=30, show="*")
    signup_password_entry.pack()

    def signup(arg=None):
        """
        responsible for sending the server the first request to signup
        rendering the response from the server in the UI
        if its successful its calls the next function of the signup :
        verify_code
        :param arg:
        :return: a confirmation to the previous function if a user signup
        successfully
        """
        global client_connected
        print("Signup clicked")
        email_entry = signup_email_entry.get()

```

```

username_entry = signup_username_entry.get()
password_entry = signup_password_entry.get()
signup_dict = {"command": "signup", "email": email_entry,
"username": username_entry, "password": password_entry}
client_socket.sendall(protocol.send_message_aes(signup_dict,
aes_cipher))
response_dict = protocol.get_message_aes(client_socket,
aes_cipher)
response = response_dict.get("msg")
if response == "Enter code":
    # Save the original layout of the widgets
    original_layout = {
        "email_label": signup_email_label.pack_info(),
        "email_entry": signup_email_entry.pack_info(),
        "username_label": signup_username_label.pack_info(),
        "username_entry": signup_username_entry.pack_info(),
        "password_label": signup_password_label.pack_info(),
        "password_entry": signup_password_entry.pack_info(),
        "signup_button": signup_button.pack_info()
    }

    # Hide the email, username, and password entries and labels
    signup_email_label.pack_forget()
    signup_email_entry.pack_forget()
    signup_username_label.pack_forget()
    signup_username_entry.pack_forget()
    signup_password_label.pack_forget()
    signup_password_entry.pack_forget()
    signup_button.pack_forget()
    message_label_signup.config(text="")

    # Create a new label and entry for the verification code
    verification_code_label = ttk.Label(signup_frame,
text="Verification Code:")
    verification_code_label.pack(side="top", padx=5, pady=5)

    verification_code_entry = ttk.Entry(signup_frame, width=30)
    verification_code_entry.pack(side="top", padx=5, pady=5)

    def verify_code(arg1=None):
        """
        the second part of the signup,
        rendering the response from the server in the UI
        send a request to the server with the input code the
client wrote
        :param arg1:
        :return: a confirmation to the previous function if a user
signup successfully
        """
        global client_connected
        code_entry = verification_code_entry.get()
        verification_dict = {"command": "signup_code", "code":
code_entry}

        client_socket.sendall(protocol.send_message_aes(verification_dict,
aes_cipher))

        verification_response_dict =

```

```

protocol.get_message_aes(client_socket, aes_cipher)
    msg = verification_response_dict.get("msg")
    print(msg)
    if msg == "Account created":
        message_label_signup.config(text=msg)
        client_connected = True
        root.destroy()
    elif msg is not None and msg.startswith("Invalid code"):
        message_label_signup.config(text=msg)
    elif msg == "Maximum attempts reached. Please try again
later.":
        message_label_signup.config(text=msg)
        # Clear the verification code entry
        verification_code_entry.delete(0, tk.END)
        # Disable the Verify button
        verify_button.config(state="disabled")
        # Unbind the Enter key event from the signup frame and
verification code entry
        signup_frame.unbind('<Return>')
        verification_code_entry.unbind('<Return>')
    else:
        message_label_signup.config(text=msg)
        # Restore the original layout of the widgets

signup_email_label.pack(**original_layout["email_label"])
signup_email_entry.pack(**original_layout["email_entry"])
signup_username_label.pack(**original_layout["username_label"])
signup_username_entry.pack(**original_layout["username_entry"])
signup_password_label.pack(**original_layout["password_label"])
signup_password_entry.pack(**original_layout["password_entry"])
signup_button.pack(**original_layout["signup_button"])
# Remove the verification code label and entry
verification_code_label.pack_forget()
verification_code_entry.pack_forget()
verify_button.pack_forget()

verify_button = ttk.Button(signup_frame, text="Verify",
command=verify_code)
verify_button.pack(side="top", padx=5, pady=5)

signup_frame.bind('<Return>', verify_code)
verification_code_entry.bind('<Return>', verify_code)
else:
    message_label_signup.config(text=response)

signup_frame.bind('<Return>', signup)
signup_password_entry.bind('<Return>', signup)
signup_email_entry.bind('<Return>', signup)
signup_username_entry.bind('<Return>', signup)

# Signup button
signup_button = ttk.Button(signup_frame, text="Sign-Up",
command=signup)

```



```

signup_button.pack(pady=10)

# Signup message label
signup_message_label = ttk.Label(signup_frame, text="")
signup_message_label.pack()

# signup respond from server label
message_label_signup = tk.Label(signup_frame, text="")
message_label_signup.pack(side="bottom", padx=5, pady=5)

# signup respond from server label
message_label_login = tk.Label(login_frame, text="")
message_label_login.pack(side="bottom", padx=5, pady=5)

forgot_password_frame = ttk.Frame(notebook, padding=10)
notebook.add(forgot_password_frame, text="Forgot Password")

icon_image_label_forgot_password = ttk.Label(forgot_password_frame,
image=subsampled_icon)
icon_image_label_forgot_password.pack()

# Forgot Password email label and entry
forgot_password_email_label = ttk.Label(forgot_password_frame,
text="Email:")
forgot_password_email_label.pack()

forgot_password_email_entry = ttk.Entry(forgot_password_frame,
width=30)
forgot_password_email_entry.pack()

def forgot_password(arg=None):
    """
    responsible for changing a user password
    rendering the response from the server in the UI
    send a request to the server with the email the client wrote
    if its successful its calls the next function : verify_code
    :param arg:
    :return:
    """
    email = forgot_password_email_entry.get()
    forgot_password_dict = {"command": "forgot_password", "email":
email}

client_socket.sendall(protocol.send_message_aes(forgot_password_dict,
aes_cipher))
response = protocol.get_message_aes(client_socket, aes_cipher)
msg = response.get("msg")

if msg == "Enter code":
    # Hide the email entry and label
    forgot_password_email_label.pack_forget()
    forgot_password_email_entry.pack_forget()
    forgot_password_button.pack_forget()
    message_label_forgot_password.config(text="")
    # Create a new label and entry for the verification code
    verification_code_label = ttk.Label(forgot_password_frame,
text="Verification Code:")
    verification_code_label.pack()

```

```

verification_code_entry = ttk.Entry(forgot_password_frame,
width=30)
verification_code_entry.pack()

def verify_code(arg1=None):
    """
    the second part of the forgot_password
    rendering the response from the server in the UI
    send a request to the server with the input code the
client wrote
    if its successful its calls the next function :
update_password
:param arg1:
:return:
    """
    code = verification_code_entry.get()
    forgot_pass_response_dict = {"command":
"forgot_password_code", "code": code}

client_socket.sendall(protocol.send_message_aes(forgot_pass_response_dict
, aes_cipher))
    response1 = protocol.get_message_aes(client_socket,
aes_cipher)
    msg1 = response1.get("msg")

    if msg1 == "Code verified":
        # Hide the verification code label and entry
        verification_code_label.pack_forget()
        verification_code_entry.pack_forget()
        verify_button.pack_forget()
        message_label_forgot_password.config(text="")

        # Create new labels and entries for the new password
        new_password_label = ttk.Label(forgot_password_frame,
text="New Password:")
        new_password_label.pack()

        new_password_entry = ttk.Entry(forgot_password_frame,
width=30, show="*")
        new_password_entry.pack()

    def update_password(arg2=None):
        """
        the third part of the forgot_password
        rendering the response from the server in the UI
        send a request to the server with the new password
the client wrote
        :param arg2:
        :return:
        """
        new_password = new_password_entry.get()
        update_pass_dict = {"command":
"forgot_password_new", "password": new_password}

        client_socket.sendall(protocol.send_message_aes(update_pass_dict,
aes_cipher))
        update_pass_response_dict =

```

```

protocol.get_message_aes(client_socket, aes_cipher)
    msg2 = update_pass_response_dict.get("msg")
    message_label_forgot_password.config(text=msg2)

    update_password_button =
ttk.Button(forgot_password_frame, text="Update Password",
command=update_password)
    update_password_button.pack()

    forgot_password_frame.bind('<Return>',
update_password)
    new_password_entry.bind('<Return>', update_password)
    elif msg1 is not None and msg1.startswith("Invalid code"):

        message_label_forgot_password.config(text=msg1)
    elif msg1 == "Maximum attempts reached. Please try again
later.":

        message_label_forgot_password.config(text=msg1)
        # Clear the verification code entry
        verification_code_entry.delete(0, tk.END)
        # Disable the Verify button
        verify_button.config(state="disabled")
    else:
        message_label_forgot_password.config(text=msg)

    verify_button = ttk.Button(forgot_password_frame,
text="Verify", command=verify_code)
    verify_button.pack()

    forgot_password_frame.bind('<Return>', verify_code)
    verification_code_entry.bind('<Return>', verify_code)
    else:
        message_label_forgot_password.config(text=msg)

    forgot_password_button = ttk.Button(forgot_password_frame,
text="Forgot Password", command=forgot_password)
    forgot_password_button.pack()

    message_label_forgot_password = ttk.Label(forgot_password_frame,
text="")
    message_label_forgot_password.pack(side="bottom")

    forgot_password_frame.bind('<Return>', forgot_password)
    forgot_password_email_entry.bind('<Return>', forgot_password)

root.wait_window(dialog)
if client_connected:
    return "success"
elif admin_connected:
    return "admin_connected"
else:
    return "exit"

def show_connection_error_popup():

```

```

"""
    responsible for a popup msg incase the client can't connect to the
    server
    :return:
    """

    root = tk.Tk()
    root.withdraw() # Hide the main window

    popup = tk.Toplevel(root)
    popup.title("Connection Error")
    popup.iconbitmap('icons//Icon_no_text.ico')

    # Apply ttk style for a modern look
    style = ttk.Style(popup)
    style.theme_use('clam')

    # Custom colors
    background_color = "#f0f0f0"
    button_color = "#4CAF50"
    button_text_color = "#FFFFFF"

    # Configure style elements
    style.configure('TFrame', background=background_color)
    style.configure('TButton', background=button_color,
foreground=button_text_color)
    style.configure('TLabel', background=background_color)

    popup_frame = ttk.Frame(popup, padding=10)
    popup_frame.pack(fill="both", expand=True)

    label = ttk.Label(popup_frame, text="Unable to connect to the
server.", font=("Arial", 14))
    label.pack(pady=20)

    def on_closing():
        root.destroy()

    ok_button = ttk.Button(popup_frame, text="OK", command=on_closing)
    ok_button.pack()
    popup.protocol("WM_DELETE_WINDOW", on_closing)
    # Center the popup window on the screen
    popup.update_idletasks() # Update geometry information
    window_width = 400 # Increase the width to 400 pixels
    window_height = popup.winfo_height()
    screen_width = popup.winfo_screenwidth()
    screen_height = popup.winfo_screenheight()
    x = (screen_width - window_width) // 2
    y = (screen_height - window_height) // 2
    popup.geometry(f"{window_width}x{window_height}+{x}+{y}")

    # Run the Tkinter event loop for the popup
    popup.mainloop()

def first_connection():
    """
    Establishes the first connection between the client and the server
    by securely exchanging encryption keys.

```

```

: return:
"""
# Generate a random 32-byte encryption key for AES
encryption_key = get_random_bytes(32)
# Generate a random 16-byte initialization vector (IV) for AES
iv = get_random_bytes(16)
# Create an AES cipher object using the generated encryption key and
IV
aes_cipher = protocol.AESCipher(encryption_key, iv)
# Extract the public RSA key from the server
keys_dict = protocol.get_message(client_socket)
public_key = keys_dict["key"]
cipher = PKCS1_OAEP.new(RSA.import_key(public_key))
# Encrypt the AES encryption key and iv with the RSA public key
encrypted_aes_key = cipher.encrypt(encryption_key)
encrypted_iv = cipher.encrypt(iv)
# Send the dictionary back to the client
keys_response_dict = {"aes_key": encrypted_aes_key, "iv":
encrypted_iv}
client_socket.sendall(protocol.send_message(keys_response_dict))
return aes_cipher

def main():
"""
the first and main function of the client
: return:
"""
try:
    client_socket.connect((SERVER_HOST, SERVER_PORT))
    aes_cipher = first_connection()
    result = login_signup(aes_cipher)
    if result == "success":
        file_explorer_ui(aes_cipher)
    if result == "admin_connected":
        client_admin.admin_ui(client_socket, aes_cipher)
    else:
        print("Exiting the application.")
        client_socket.close()
        sys.exit()
except socket.error as e:
    print(f"Error connecting to the server: {e}")
    show_connection_error_popup()
    sys.exit()

if __name__ == '__main__':
    main()

```

Client_Admin.py

```

import tkinter as tk
import tkinter.ttk as ttk
import protocol
import sys
import datetime

```

```

selected_clients = set()
current_page = 1
total_pages = 1

def admin_ui(client_socket, aes_cipher):
    """
    the main function for the admin client
    responsible for the UI and other functions.
    param client_socket:
    :param aes_cipher:
    :return:
    """
    global current_page

    def refresh_clients():
        """
        send a request to the server to receive and display the users
        :return:
        """
        global current_page, total_pages
        print("refresh_clients")
        refresh_dict = {"command": "refresh_clients", "page":
current_page, "rows": rows, "cols": cols}
        client_socket.sendall(protocol.send_message_aes(refresh_dict,
aes_cipher))
        response = protocol.get_message_aes(client_socket, aes_cipher)

        users = response.get("users")
        total_pages = response.get("total_pages")
        total_users = response.get("total_users")

        # Clear the buttons array
        for row1 in buttons:
            for button1 in row1:
                button1.grid_remove()

        # Iterate over the users and update the buttons
        row1 = 0
        col = 0
        for username, user_type in users.items():
            if row1 < rows and col < cols:
                buttons[row1][col].config(text=username, font=("Arial",
12), compound="top",
                                relief="raised", command=lambda
r=row1, c=col: button_click(r, c))
                buttons[row1][col].grid()
                col += 1
            if col == cols:
                col = 0
                row1 += 1
        selected_clients.clear()
        users_amount_label.config(text=f"There are:\n{total_users}
clients")

    def get_logs():
        """
        send a request to the server to receive and display the logs

```

```

: return:
"""
try:
    logs_dict = {"command": "get_logs"}
    client_socket.sendall(protocol.send_message_aes(logs_dict,
aes_cipher))
    logs_response_dict = protocol.get_message_aes(client_socket,
aes_cipher)
    logs = logs_response_dict.get("logs")

    logs_text.delete('1.0', tk.END) # Clear the existing text
    current_date = None
    for log in logs:
        log_timestamp, log_message = log.split(" - ", 1)
        log_datetime = datetime.datetime.strptime(log_timestamp,
'%Y-%m-%d %H:%M:%S')
        log_date = log_datetime.date()
        if current_date != log_date:
            current_date = log_date
            logs_text.insert(tk.END,
f"{log_date.strftime('%d/%m/%y').center(85, '-')}}\n\n")
            logs_text.insert(tk.END,
f"{log_datetime.strftime('%H:%M:%S')} - {log_message}\n")
            logs_text.see(tk.END)
    except Exception as e:
        print(f"Error getting logs: {e}")
        logs_text.insert(tk.END, "Error retrieving logs.")

def delete():
    """
    send a request to the server to delete a list of clients
    : return:
    """
    global current_page
    print("delete")
    try:
        users_list = [button1.cget('text') for button1 in
selected_clients]
        delete_dict = {"command": "delete_users", "users": users_list}
        print(delete_dict)
        client_socket.sendall(protocol.send_message_aes(delete_dict,
aes_cipher))
        response = protocol.get_message_aes(client_socket, aes_cipher)
        print(response)
        refresh_clients()
    except Exception as e:
        print(f"Error deleting users: {e}")
        show_popup(f"Error deleting users, {str(e)}")

def previous_page():
    """
    responsible for the button "previous_page"
    incase the user have to many users to display he can move pages,
    this allows him to move backwards
    : return:
    """
    global current_page
    if current_page > 1:

```

```

        current_page -= 1
        page_label.config(text=current_page)
        refresh_clients()

def next_page():
    """
    responsible for the button "next_page"
    incase the user have to many users to display he can move to the
next page
    :return:
    """
    global current_page, total_pages
    if current_page < total_pages:
        current_page += 1
        page_label.config(text=current_page)
        refresh_clients()

def show_popup(string1):
    """
    responsible for a popup msg that can appear in different
situations
    :param string1: that can be many kinds of strings, it will render
this string in the popup
    :return:
    """
    popup = tk.Toplevel(root)
    popup.title("Error")
    popup.iconbitmap('icons//Icon_no_text.ico')

    popup_frame = ttk.Frame(popup, padding=10)
    popup_frame.pack(fill="both", expand=True)

    label = ttk.Label(popup_frame, text="string1")
    label.pack(pady=20)

    ok_button = ttk.Button(popup_frame, text="OK",
command=popup.destroy)
    ok_button.pack()

    # Center the popup window on the screen
    popup.update_idletasks() # Update geometry information
    window_width = 400 # Increase the width to 400 pixels
    window_height = popup.winfo_height()
    screen_width = popup.winfo_screenwidth()
    screen_height = popup.winfo_screenheight()
    x = (screen_width - window_width) // 2
    y = (screen_height - window_height) // 2
    popup.geometry(f"{window_width}x{window_height}+{x}+{y}")

root = tk.Tk()
root.title("Nimbus")
root.iconbitmap('icons//Icon_no_text.ico')

def on_closing():
    """
    responsible for the closure of the program, when a user exit the
program this function is called
    sends a msg to the server that the client disconnected and end the

```



```

program
    :return:
    """
    print("closed")
    closing_connection_dict = {"command": "exit"}

    client_socket.sendall(protocol.send_message_aes(closing_connection_dict,
    aes_cipher))
    root.destroy()
    sys.exit()

    root.protocol("WM_DELETE_WINDOW", on_closing)

    style = ttk.Style()
    style.theme_use('clam')

    # Define custom colors
    background_color = "#f0f0f0"
    button_color = "#4CAF50"
    button_text_color = "#FFFFFF"

    # Configure style elements
    style.configure('TFrame', background=background_color)
    style.configure('TButton', background=button_color,
    foreground=button_text_color)
    style.configure('TLabel', background=background_color,
    font=('TkDefaultFont', 12, 'bold'))
    style.configure('TCanvas', background=background_color)
    style.configure('TSeparator', thickness=2)

    # Set initial window size
    root.geometry("1000x600") # Width x Height

    left_frame = ttk.Frame(root) # Use ttk.Frame for styling
    left_frame.pack(side="left", fill="y") # Fill vertically

    right_frame = ttk.Frame(root)
    right_frame.pack(side="right", fill="both", expand=True)

    commands_frame = ttk.Frame(right_frame)
    commands_frame.pack(side="bottom", fill="x")

    nav_frame = ttk.Frame(commands_frame)
    nav_frame.pack(side="right")

    clients_frame = ttk.Frame(right_frame)
    clients_frame.pack(fill="both")

    line = ttk.Separator(root, orient="vertical") # Use ttk.Separator for
styling
    line.pack(side="left", fill="y", padx=5)

    rows = 5
    cols = 6
    buttons = []

    def button_click(row1, col):
        """

```

```

        responsible to update the global var of the selected buttons
        when the user press any of the users buttons it calls this
function
    :return:
    """
    button1 = buttons[row1][col]
    if button1 in selected_clients:
        # Button is already selected, so unselect it
        selected_clients.remove(button1)
        button1.config(relief="raised") # Change the button's
appearance to unselected state
    else:
        # Button is not selected, so select it
        selected_clients.add(button1) # For set
        # selected_buttons.append(button) # For list
        button1.config(relief="sunken") # Change the button's
appearance to selected state

    print(f"Selected buttons: {[button1.cget('text') for button1 in
selected_clients]}")

    icon_image_account = tk.PhotoImage(file="icons//user.png")
    subsampled_icon_account = icon_image_account.subsample(13, 13) #
Reduce size

    for i in range(rows):
        row = []
        for j in range(cols):
            button = tk.Button(clients_frame, text=f"Button {i},{j}",
image=subsampled_icon_account, font=("Arial", 12),
                                compound="top", padx=20, pady=10)
            button.grid(row=i, column=j, sticky="nsew")
            row.append(button)
        buttons.append(row)

    # Make all buttons expand and fill the frame
    for i in range(rows):
        right_frame.grid_rowconfigure(i, weight=1)
    for j in range(cols):
        right_frame.grid_columnconfigure(j, weight=1)

    logs_text = tk.Text(left_frame, height=10, width=50, font=("Arial",
12))
    logs_text.pack(side="top", fill="both", expand=True)

    get_logs_button = ttk.Button(left_frame, text="Get Logs",
command=get_logs)
    get_logs_button.pack(side="top")

    users_amount_label = ttk.Label(commands_frame, text="There are:\n5
clients")
    users_amount_label.pack(side="left")

    icon_image_refresh = tk.PhotoImage(file="icons//refresh.png")
    subsampled_icon_refresh = icon_image_refresh.subsample(16, 16) #
Reduce size

    refresh_button = ttk.Button(commands_frame, text="Refresh",

```

```

image=subsampled_icon_refresh, compound="top", command=refresh_clients)
refresh_button.pack(side="left")

icon_image_delete = tk.PhotoImage(file="icons//delete.png")
subsampled_icon_delete = icon_image_delete.subsample(16, 16) # Reduce
size

delete_button = ttk.Button(commands_frame, text="Delete",
image=subsampled_icon_delete, compound="top",
command=delete)
delete_button.pack(side="left")

next_button = ttk.Button(nav_frame, text="Next", compound="top",
command=next_page)
next_button.pack(side="right", padx=5) # Add some padding between the
buttons

page_label = ttk.Label(nav_frame, text=current_page)
page_label.pack(side="right", padx=5)

previous_button = ttk.Button(nav_frame, text="Previous",
compound="top",
command=previous_page)
previous_button.pack(side="right")

refresh_clients()
get_logs()
root.mainloop()

```

Server.py

```

import socket
import ssl
import threading
import os
import sqlite3
import protocol
import hashlib
import zipfile
import shutil
import random
import smtplib
import json
import base64
from email.message import EmailMessage
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from logger import Logger
import server_admin

SERVER_HOST = '127.0.0.1'
SERVER_PORT = 12345
UPLOAD_FOLDER = 'STORED_DATA' # Update the upload folder path
MAXIMUS_SIZE_PER_CLIENT = 1073741824

```

```
def send_verification_email(email, code):
    """
    send an email to the user with a generated code
    :param email:
    :param code:
    :return:
    """
    sender = 'nimbus.mail.ver@gmail.com'
    sender_password = "eomh jcbn nrzl kvwl"
    receiver = email
    subject = 'Verification Code'
    body = f'Your verification code is: {code}'

    em = EmailMessage()
    em["From"] = sender
    em["To"] = receiver
    em["Subject"] = subject
    em.set_content(body)
    context = ssl.create_default_context()

    with smtplib.SMTP_SSL("smtp.gmail.com", 465, context=context) as smtp:
        smtp.login(sender, sender_password)
        smtp.sendmail(sender, receiver, em.as_string())
        print("email sent")
        Logger.log_email_sent(email, subject)

def read_file_content(file_path):
    """
    Read the content from a given path
    :param file_path:
    :return: file content in bytes form
    """
    try:
        with open(file_path, 'rb') as file:
            return file.read()
    except FileNotFoundError:
        print(f"Error: File '{file_path}' not found.")
        Logger.log_error("File '{file_path}' not found.")
    except Exception as e:
        print(f"Error: {e}")
        Logger.log_error(e)

def print_bytes(bytes_value):
    """
    Converts a byte value into a human-readable format
    :param bytes_value:
    :return: string that contain the size and suffix
    """
    suffixes = ['B', 'KB', 'MB', 'GB']
    suffix_index = 0

    while bytes_value >= 1024 and suffix_index < len(suffixes) - 1:
        bytes_value /= 1024.0
        suffix_index += 1
```

```

print(f"{bytes_value:.2f} {suffixes[suffix_index]}")

def get_folder_size(folder_path):
    """
    get the sum size for all the files within a folder
    :param folder_path:
    :return:
    """
    # Convert folder_path to a string
    folder_path = str(folder_path)
    total_size = 0
    # Walk through all files and subdirectories in the specified folder
    for dir_path, dir_names, filenames in os.walk(folder_path):
        for filename in filenames:
            # Get the full path of the file
            file_path = os.path.join(dir_path, filename)
            # Ensure file_path is a string
            file_path = str(file_path)
            # Add the size of the file to the total size
            total_size += os.path.getsize(file_path)
    return total_size

def check_file_exists(username, filename):
    """
    check if a given path to a file exist
    :param username:
    :param filename:
    :return: bool of the file existence
    """
    user_folder = os.path.join(UPLOAD_FOLDER, username)
    file_path = os.path.join(user_folder, filename)
    return os.path.isfile(file_path) or os.path.isdir(file_path)

def create_folder_by_user(create_folder_dict, username, client_socket,
aes_cipher):
    """
    responsible for creating a new folder when a user request it
    :param create_folder_dict:
    :param username:
    :param client_socket:
    :param aes_cipher:
    :return:
    """
    try:
        folder_name = create_folder_dict.get("folder_name")
        user_folder = os.path.join(UPLOAD_FOLDER, str(username))
        new_folder_path = os.path.join(user_folder, folder_name)

        if ".." in folder_name: # prevent Directory traversal attack
            response_dict = {"command": "response", "msg": "Invalid folder
name"}
            client_socket.sendall(protocol.send_message_aes(response_dict,
aes_cipher))
            Logger.log_error(f"{username} Attempted directory traversal:

```

```
{folder_name}")
    return

    if not os.path.exists(new_folder_path):
        os.makedirs(new_folder_path)
        response_dict = {"command": "response", "msg": "Folder created successfully"}
    else:
        response_dict = {"command": "response", "msg": "Folder already exists"}
    except Exception as e:
        response_dict = {"command": "response", "msg": f"Error creating folder: {str(e)}"}
        Logger.log_error(f"{username} Error creating folder: {str(e)}")

    client_socket.sendall(protocol.send_message_aes(response_dict, aes_cipher))

def receive_file(files_dict, username, aes_cipher, encryption_key):
    """
    responsible for receiving a new file when a user request it
    :param files_dict:
    :param username:
    :param aes_cipher:
    :param encryption_key:
    :return:
    """
    try:
        file_name = files_dict.get("file_name")[5:]
        file_type = files_dict.get("file_type")
        file_content = files_dict.get("file_Content")

        if ".." in file_name or ".." in file_type: # prevent Directory traversal attack
            return

        # Convert username to string for path construction
        username_str = str(username)

        # Combine file_name and file_type for proper path construction
        complete_file_name = f"{file_name}{file_type}" # Include file type with dot
        file_path = os.path.join(UPLOAD_FOLDER, username_str, complete_file_name)
        counter = 1
        base_name, extension = os.path.splitext(complete_file_name)
        while os.path.exists(file_path):
            new_file_name = f"{base_name}({counter}){extension}"
            file_path = os.path.join(UPLOAD_FOLDER, username_str, new_file_name)
            counter += 1

        folder_size = get_folder_size(f"{UPLOAD_FOLDER}/{username}")
        if folder_size + len(file_content) > MAXIMUS_SIZE_PER_CLIENT:
            print("User folder size reached its max storage size of 1GB")
        else:
            with open(file_path, 'wb') as file:
```

```

        file.write(file_content)

    encrypt_file(username, file_path, encryption_key)
    Logger.log_upload(username, file_name)
except Exception as e:
    print(f"Error receiving file: {e}")
    Logger.log_error(f"{username} error receiving file: {e}")

def receive_folder(folders_dict, username, aes_cipher, encryption_key):
    """
    Receives, processes, and stores an uploaded folder from a user.
    :param folders_dict: (dict) A dictionary containing folder information
    with keys "folder_name" and "folder_content".
    :param username: (str) The username of the user uploading the folder.
    :param aes_cipher: (aes_cipher object)
    :param encryption_key: (AES key) The encryption key used to encrypt
    the folder.
    :return: None
    """
    try:
        folder_name = folders_dict.get("folder_name")[:-4]
        folder_name = folder_name[5:]
        folder_content = folders_dict.get("folder_content")
        if ".." in folder_name: # prevent Directory traversal attack
            return

        # Convert username to string for path construction
        username_str = str(username)
        # Create a temporary file path to save the ZIP file
        temp_folder_path = os.path.join(UPLOAD_FOLDER, username_str,
        folder_name)
        counter = 1
        base_name, extension = os.path.splitext(folder_name)
        # Added a loop to handle folder name conflicts
        while os.path.exists(temp_folder_path):
            new_folder_name = f"{base_name}({counter}){extension}"
            temp_folder_path = os.path.join(UPLOAD_FOLDER, username_str,
            new_folder_name)
            counter += 1
        final_folder_path = temp_folder_path
        temp_folder_path = temp_folder_path + ".zip"
        folder_size_user = get_folder_size(f"{UPLOAD_FOLDER}/{username}")
        if folder_size_user + len(folder_content) >
        MAXIMUS_SIZE_PER_CLIENT:
            print("User folder size reached its max storage size of 1GB")
        else:
            # Save the ZIP file temporarily
            with open(temp_folder_path, 'wb') as file:
                file.write(folder_content)

            # Create a path for the unzipped folder (remove the .zip
            extension)
            unzipped_folder_path = temp_folder_path[:-4]
            # Unzip the file to the specified folder
            print(f"temp_folder_path: {temp_folder_path}")
            print(f"unzipped_folder_path: {unzipped_folder_path}")
            with zipfile.ZipFile(temp_folder_path, 'r') as zip_ref:
                for member in zip_ref.infolist():

```

```

        extracted_path = zip_ref.extract(member,
unzipped_folder_path)
        if member.is_dir():
            os.makedirs(extracted_path, exist_ok=True)

        # Remove the temporary ZIP file
        os.remove(temp_folder_path)

        encrypt_folder(username, final_folder_path, encryption_key)
    except Exception as e:
        print(f"Error receiving folder: {e}")
        Logger.log_error(f"{username} error receiving folder: {e}")

def refresh(refresh_dict, client_socket, username, aes_cipher):
    """
    responsible for sending the files withing a folder when a user request
    it
    :param refresh_dict:
    :param client_socket:
    :param username:
    :param aes_cipher:
    :return:
    """
    path = refresh_dict.get("path")[4:]
    if path != "":
        path = f"//{path}"
    user_folder = f"{UPLOAD_FOLDER}/{username}{path}"
    files_dict = {} # Create an empty dictionary for files
    files_list = []
    for content_name in os.listdir(user_folder):
        content_path = os.path.join(user_folder, content_name)
        is_file = os.path.isfile(content_path) # Check if it's a file
        files_list.append((content_name, "file" if is_file else "folder"))

    if ("encryptions_iv.json", "file") in files_list:
        files_list.remove(("encryptions_iv.json", "file")) # hide the
        json file from the user

    # Calculate the total number of pages
    rows = refresh_dict.get("rows")
    cols = refresh_dict.get("cols")
    items_per_page = rows * cols
    total_pages = (len(files_list) + items_per_page - 1) // items_per_page

    # Get the files for the current page
    page = refresh_dict.get("page", 1)
    start_index = (page - 1) * items_per_page
    end_index = start_index + items_per_page
    for file_name, file_type in files_list[start_index:end_index]:
        files_dict[file_name] = file_type

    folder_size = get_folder_size(user_folder)

    response_dict = {"command": "response", "file_size": folder_size,
"files": files_dict, "total_pages": total_pages}
    client_socket.sendall(protocol.send_message_aes(response_dict,
aes_cipher))

```



```
def delete(delete_dict, username, client_socket, aes_cipher):
    """
    responsible for deleting a file/folder when a user request it
    :param delete_dict:
    :param username:
    :param client_socket:
    :param aes_cipher:
    :return:
    """
    try:
        file_names = delete_dict.get("files")
        for filename in file_names:
            if filename == "encryptions_iv.json": # Prevent deletion of
the "encryptions_iv.json" file
                continue
            if not check_file_exists(username, filename[5:]): # prevent
Directory traversal attack
                continue

            filepath = os.path.join(UPLOAD_FOLDER, str(username),
filename[5:])

            if os.path.isfile(filepath):
                os.remove(filepath)
                print(f"Deleted file: {filename}")
                Logger.log_delete(username, filename)
                delete_from_json(username, filepath)

            elif os.path.isdir(filepath):
                try:
                    shutil.rmtree(filepath)
                    print(f"Deleted folder: {filename}")
                    Logger.log_delete(username, filename)

                    # Remove the IVs of all files in the deleted folder
from the JSON file
                    iv_file_path = os.path.join(UPLOAD_FOLDER,
str(username), "encryptions_iv.json")
                    with open(iv_file_path, 'r') as iv_file:
                        iv_dict = json.load(iv_file)

                    # Iterate over the keys (file paths) in the IV
dictionary
                    for key in list(iv_dict.keys()):
                        # Check if the key starts with the folder path and
has an additional path separator
                        if key.startswith(filepath + os.path.sep):
                            delete_from_json(username, key)

                except PermissionError as e:
                    print(f"Error deleting folder '{filename}': {e}")
                    Logger.log_error(f"{username} PermissionError:
{filename}")
                else:
                    print(f"File not found: {filename}")
                    Logger.log_error(f"{username} File not found: {filename}")
```

```

        delete_response_dict = {"command": "response", "msg": "deleted"}

client_socket.sendall(protocol.send_message_aes(delete_response_dict,
aes_cipher))
    except Exception as e:
        print(f"Error deleting file/folder: {e}")
        Logger.log_error(f"{username} Error deleting file/folder: {e}")

def download(download_dict, username, client_socket, aes_cipher,
encryption_key):
    """
    responsible for sending a folder/file when a user request it
    :param download_dict:
    :param username:
    :param client_socket:
    :param aes_cipher:
    :param encryption_key:
    :return:
    """
    try:
        file_names = download_dict.get("files")
        if not file_names:
            response_dict = {"command": "response", "files": "no files
selected"}
            client_socket.sendall(protocol.send_message_aes(response_dict,
aes_cipher))
            return

        file_data = []
        for file in file_names:
            if not check_file_exists(username, file[5:]): # prevent
Directory traversal attack
                continue

            file_path = os.path.join(UPLOAD_FOLDER, str(username),
file[5:])
            if os.path.isfile(file_path):

                decrypt_file(username, file_path, encryption_key) #
decrypt the file before reading

                file_name, file_type =
os.path.splitext(os.path.basename(file_path))
                file_content = read_file_content(file_path)
                if len(file_content) < MAXIMUS_SIZE_PER_CLIENT:
                    file_data.append({"file_name": file_name, "file_type":
file_type, "file_Content": file_content})
                    Logger.log_download(username, file)
                else:
                    print("file too large")
                    Logger.log_error(f"{username} File too large:
{file_path}")

                encrypt_file(username, file_path, encryption_key)

            elif os.path.isdir(file_path):

```

```

        decrypt_folder(username, file_path, encryption_key)

        # Handle the case when the user wants to download a folder
        zip_file_name = os.path.basename(file_path) + ".zip"
        temp_zip_file_path = os.path.join(UPLOAD_FOLDER,
str(username), zip_file_name)

        # Create a zip file containing the existing folder
        # Create a zip file containing the folder
        with zipfile.ZipFile(temp_zip_file_path, 'w') as zip_file:
            for root, dirs, files in os.walk(file_path):
                # Add empty directories to the zip file
                for dir_name in dirs:
                    dir_path_in_zip = os.path.join(root, dir_name)
                    zip_file.write(dir_path_in_zip,
os.path.relpath(dir_path_in_zip, file_path))

                    # Add files to the zip file
                for file1 in files:
                    file_path_in_zip = os.path.join(root, file1)
                    zip_file.write(file_path_in_zip,
os.path.relpath(file_path_in_zip, file_path))

            # Read the contents of the zip file
            with open(temp_zip_file_path, 'rb') as file2:
                file_content = file2.read()

            if len(file_content) < MAXIMUS_SIZE_PER_CLIENT:
                file_data.append({"file_name": zip_file_name,
"file_type": "folder", "file_Content": file_content})
                Logger.log_download(username, file_path)

            else:
                print("folder too large")
                Logger.log_error(f"{username} Folder too large:
{file_path}")

            # Remove the temporary zip file
            os.remove(temp_zip_file_path)
            encrypt_folder(username, file_path, encryption_key)
        else:
            print(f"File not found: {file}")

        response_dict = {"command": "response", "files": file_data}
        client_socket.sendall(protocol.send_message_aes(response_dict,
aes_cipher))
    except Exception as e:
        print(f"Error downloading file/folder: {e}")
        Logger.log_error(f"{username} Error downloading file/folder: {e}")

def login_signup(client_socket, aes_cipher):
    """
    the main function for the login/signup/forgot_pass part
    contains most of the login and calls the function
    :param client_socket:
    :param aes_cipher:
    :return:

```

```

"""
max_attempts = 5
login_attempts = 0
signup_attempts = 0
forgot_password_attempts = 0
verification_code_create = ""
verification_code_forgot = ""
is_verified_forgot = False
account_data_signup_dict = {}
account_data_forgot_dict = {}

while True:
    print("waiting for msg")
    login_or_create_dict = protocol.get_message_aes(client_socket,
aes_cipher)
    command = login_or_create_dict.get("command")
    print(command)

    if command == "login":
        if login_attempts < max_attempts:
            response, username, is_admin, file_encryption_key =
login(login_or_create_dict)
            if response == "Login successful" or response == "Admin
connected":
                response_dict = {"command": "response", "msg":
response}
client_socket.sendall(protocol.send_message_aes(response_dict,
aes_cipher))
                return username, is_admin, file_encryption_key
            elif response == "Wrong password":
                login_attempts += 1
                response_dict = {"command": "response", "msg": f"Wrong
password. Attempt {login_attempts}/{max_attempts}"}
client_socket.sendall(protocol.send_message_aes(response_dict,
aes_cipher))
            elif response == "User doesn't exist":
                response_dict = {"command": "response", "msg":
response}
client_socket.sendall(protocol.send_message_aes(response_dict,
aes_cipher))
            else:
                response_dict = {"command": "response", "msg": "Too many
failed login attempts. Please try again later."}
client_socket.sendall(protocol.send_message_aes(response_dict,
aes_cipher))

        elif command == "signup":
            if signup_attempts < max_attempts:
                account_data_signup_dict = login_or_create_dict.copy()
                response, verification_code_create =
signup(login_or_create_dict)
                if response == "Code sent":
                    response_dict = {"command": "response", "msg": "Enter
code"}

```

```

client_socket.sendall(protocol.send_message_aes(response_dict,
aes_cipher))
        else:
            response_dict = {"command": "response", "msg":
response}

client_socket.sendall(protocol.send_message_aes(response_dict,
aes_cipher))

        elif command == "signup_code":
            if signup_attempts < max_attempts:
                response, username, file_encryption_key =
signup_verify_code(login_or_create_dict, verification_code_create,
account_data_signup_dict)
                if response == "Account created":
                    print(f"hello: {username}")
                    response_dict = {"command": "response", "msg":
"Account created"}

client_socket.sendall(protocol.send_message_aes(response_dict,
aes_cipher))

                    Logger.log_signup(username)
                    return username, False, file_encryption_key
                    elif response == "Invalid code":
                        signup_attempts += 1
                        response_dict = {"command": "response", "msg":
f"Invalid code. Attempt {signup_attempts}/{max_attempts}"}

client_socket.sendall(protocol.send_message_aes(response_dict,
aes_cipher))
                        else:
                            response_dict = {"command": "response", "msg": "Maximum
attempts reached. Please try again later."}

client_socket.sendall(protocol.send_message_aes(response_dict,
aes_cipher))

                            elif command == "forgot_password":
                                account_data_forgot_dict = login_or_create_dict.copy()
                                response, verification_code_forgot =
forgot_password(login_or_create_dict)
                                if response == "Code sent":
                                    response_dict = {"command": "response", "msg": "Enter
code"}

client_socket.sendall(protocol.send_message_aes(response_dict,
aes_cipher))
                                    else:
                                        response_dict = {"command": "response", "msg": response}

client_socket.sendall(protocol.send_message_aes(response_dict,
aes_cipher))

                                        elif command == "forgot_password_code":
                                            if forgot_password_attempts < max_attempts:
                                                entered_code = login_or_create_dict.get("code")

```

```

        if verification_code_forgot == entered_code:
            is_verified_forgot = True
            response_dict = {"command": "response", "msg": "Code
verified"}

client_socket.sendall(protocol.send_message_aes(response_dict,
aes_cipher))

        else:
            forgot_password_attempts += 1
            response_dict = {"command": "response",
                            "msg": f"Invalid code. Attempt
{forgot_password_attempts}/{max_attempts}"}

client_socket.sendall(protocol.send_message_aes(response_dict,
aes_cipher))

        else:
            response_dict = {"command": "response", "msg": "Maximum
attempts reached. Please try again later."}

client_socket.sendall(protocol.send_message_aes(response_dict,
aes_cipher))

        elif command == "forgot_password_new":
            if is_verified_forgot:
                response = update_password(login_or_create_dict,
account_data_forgot_dict)
                response_dict = {"command": "response", "msg": response}

client_socket.sendall(protocol.send_message_aes(response_dict,
aes_cipher))

            else:
                print("error while changing password")

        elif command == "exit":
            print("Client requested exit")
            return "exit", None

        else:
            print("Invalid command received")
            response_dict = {"command": "response", "msg": "Invalid
command received"}
            client_socket.sendall(protocol.send_message_aes(response_dict,
aes_cipher))

def login(login_dict):
    """
    responsible for the login of a user
    :param login_dict:
    :return:
    """
    email = login_dict.get("email")
    password = login_dict.get("password")
    users_db = sqlite3.connect("database.db")
    cursor = users_db.cursor()

    query = "SELECT username, password, salt_key, salt_password FROM Users
WHERE email = ?"
    cursor.execute(query, (email,))

```

```

user_data = cursor.fetchall()

if len(user_data) == 0:
    cursor.close()
    users_db.close()
    print("User doesn't exist")
    return "User doesn't exist", None, False, None

username, hashed_pass_db, salt_key, salt_password = user_data[0]

hashed_pass_bytes = password_hashing(password, salt_password)
hashed_pass = hashed_pass_bytes.hex()

if hashed_pass == hashed_pass_db:
    print("Login successful")

    query = "SELECT admin FROM Users WHERE username = ?;"
    cursor.execute(query, (username,))
    is_admin = cursor.fetchone()

    cursor.close()
    users_db.close()

    if is_admin[0] == 1:
        Logger.log_admin_login(username)
        return "Admin connected", username, True, None
    else:
        if not os.path.exists(f"{UPLOAD_FOLDER}/{username}"):
            create_folder(username)
            create_json(username)
            Logger.log_login(username)
            file_encryption_key = password_hashing(password, salt_key)
            return "Login successful", username, False,
file_encryption_key
        else:
            return "Wrong password", None, False, None

def signup(signup_dict):
    """
    responsible for the first part of the signup
    :param signup_dict:
    :return:
    """
    email = signup_dict.get("email")
    username = signup_dict.get("username")
    password = signup_dict.get("password")

    # Check if password meets requirements
    if not is_valid_password(password):
        return "Password doesn't meet requirements\nThe password need to
be at least 6 characters and a number", None

    if not is_valid_email(email):
        return "Email address is not valid. Please enter a valid email
address.", None

    # Connect to the database

```

```

try:
    with sqlite3.connect("database.db") as users_db:
        cursor = users_db.cursor()

        # Check if email already exists
        query = "SELECT 1 FROM Users WHERE (email = ? OR username = ?)
LIMIT 1;"
        cursor.execute(query, (email, username))
        if cursor.fetchone():
            return "User already exists", None

        verification_code = '{:06d}'.format(random.randint(100000,
999999))

        # Send the verification email
        send_verification_email(email, verification_code)
        return "Code sent", verification_code

except sqlite3.Error as e:
    # dle database errors
    Logger.log_error(f"{username} signup error: {e}")
    return f"Database error: {str(e)}", None

def signup_verify_code(verification_dict, verification_code,
account_data_dict):
    """
    responsible for the second part of the signup
    the verification code part
    :param verification_dict:
    :param verification_code:
    :param account_data_dict:
    :return:
    """
    email = account_data_dict.get("email")
    username = account_data_dict.get("username")
    password = account_data_dict.get("password")
    entered_code = verification_dict.get("code")
    try:
        with sqlite3.connect("database.db") as users_db:
            cursor = users_db.cursor()

            if verification_code == entered_code:
                # Generate salt and hash password
                salt_password = os.urandom(32)
                salt_key = os.urandom(32)

                hashed_pass_bytes = password_hashing(password,
salt_password)
                hashed_pass = hashed_pass_bytes.hex()
                # Insert new user into the database
                query = "INSERT INTO Users (username, password, email,
salt_password, salt_key, admin) VALUES (?, ?, ?, ?, ?, ?)"
                cursor.execute(query, (username, hashed_pass, email,
sqlite3.Binary(salt_password), sqlite3.Binary(salt_key), 0))
                users_db.commit()
                create_folder(username)
                create_json(username)
                file_encryption_key = password_hashing(password, salt_key)

```



```

        return "Account created", username, file_encryption_key
    else:
        return "Invalid code", username, None

except sqlite3.Error as e:
    # Handle database errors
    Logger.log_error(f"{username} signup_verify_code error: {e}")
    return f"Database error: {str(e)}"

def forgot_password(forgot_pass_dict):
    """
    responsible for the first part of changing a user password
    :param forgot_pass_dict:
    :return:
    """
    email = forgot_pass_dict.get("email")

    try:
        with sqlite3.connect("database.db") as users_db:
            cursor = users_db.cursor()
            query = "SELECT 1 FROM Users WHERE email = ? LIMIT 1;"
            cursor.execute(query, (email,))
            if not cursor.fetchone():
                return "Email not found", None
    except sqlite3.Error as e:
        Logger.log_error(f"{email} forgot_password error: {e}")
        return f"Database error: {str(e)}", None

    verification_code = '{:06d}'.format(random.randint(100000, 999999))
    send_verification_email(email, verification_code)
    return "Code sent", verification_code

def update_password(update_pass_dict, account_data_dict):
    """
    responsible for the second part of changing a user password
    :param update_pass_dict:
    :param account_data_dict:
    :return:
    """
    email = account_data_dict.get("email")
    password = update_pass_dict.get("password")

    if not is_valid_password(password):
        return "Password doesn't meet requirements\nThe password need to be at least 6 characters and a number"

    try:
        with sqlite3.connect("database.db") as users_db:
            cursor = users_db.cursor()
            salt = os.urandom(32)

            hashed_pass_bytes = password_hashing(password, salt)
            hashed_pass = hashed_pass_bytes.hex()
            query = "UPDATE Users SET password = ?, salt_password = ?"
            WHERE email = ?"
            cursor.execute(query, (hashed_pass, sqlite3.Binary(salt),

```

```
email))
    users_db.commit()
    return "Password updated successfully"
except sqlite3.Error as e:
    Logger.log_error(f"{email} update_password error: {e}")
    return f"Database error: {str(e)}"

def is_valid_password(password):
    """
    check if the password is within the minimum standard
    :param password:
    :return:
    """
    # Check if password has at least 6 characters and a number
    if len(password) < 6:
        return False
    has_number = any(char.isdigit() for char in password)
    return has_number

def is_valid_email(email):
    """
    check if the given string is a valid email
    :param email:
    :return:
    """
    if '@' not in email or '.' not in email:
        return False
    parts = email.split('@')
    if len(parts) != 2:
        return False
    local_part, domain_part = parts
    if not local_part or not domain_part:
        return False
    if ' ' in local_part or ' ' in domain_part:
        return False
    domain_parts = domain_part.split('.')
    if len(domain_parts) < 2:
        return False
    for part in domain_parts:
        if not part:
            return False
    return True

def create_folder(username):
    """
    create a folder within the stored_data folder for a given username
    :param username:
    :return:
    """
    try:
        # Create a new directory
        os.mkdir(F"{UPLOAD_FOLDER}/{username}")
        print(f"Folder '{username}' created successfully.")
    except OSError as error:
        Logger.log_error(f"{username} Creation of the folder failed")
```

```

        print(f"Creation of the folder '{username}' failed: {error}")

def create_json(username):
    """
    create the json file for each user that saves the iv
    :param username:
    :return:
    """
    try:
        user_folder = os.path.join(UPLOAD_FOLDER, username)
        json_file_path = os.path.join(user_folder, "encryptions_iv.json")

        # Check if the JSON file already exists
        if not os.path.exists(json_file_path):
            # Create an empty dictionary to store the JSON data
            json_data = {}

            # Write the empty dictionary to the JSON file
            with open(json_file_path, "w") as json_file:
                json.dump(json_data, json_file)

            print(f"JSON file 'encryptions_iv.json' created successfully for user '{username}'.")
        else:
            print(f"JSON file 'encryptions_iv.json' already exists for user '{username}'.")
        except OSError as error:
            Logger.log_error(f"{username} Creation of the JSON file failed")
            print(f"Creation of the JSON file 'encryptions_iv.json' failed for user '{username}': {error}")

def add_to_json(username, key, value):
    """
    add to the json file the given key and value
    :param username:
    :param key:
    :param value:
    :return:
    """
    try:
        iv_file_path = os.path.join(UPLOAD_FOLDER, str(username), "encryptions_iv.json")

        # Load the existing IV dictionary from the JSON file
        with open(iv_file_path, 'r') as iv_file:
            iv_dict = json.load(iv_file)

        # Add the new key-value pair to the IV dictionary
        iv_dict[key] = value
        print(f"Added {key}: {value} to the JSON file.")

        # Write the updated IV dictionary back to the JSON file
        with open(iv_file_path, 'w') as iv_file:
            json.dump(iv_dict, iv_file)

    except FileNotFoundError:

```

```

        print(f"JSON file not found for user '{username}'.")
    except Exception as e:
        print(f"Error adding to JSON file: {e}")

def delete_from_json(username, filename):
    """
    delete from the json file the given filename
    :param username:
    :param filename:
    :return:
    """
    try:
        iv_file_path = os.path.join(UPLOAD_FOLDER, str(username),
                                     "encryptions_iv.json")

        # Load the existing IV dictionary from the JSON file
        with open(iv_file_path, 'r') as iv_file:
            iv_dict = json.load(iv_file)

        # Check if the filename exists as a key in the IV dictionary
        if filename in iv_dict:
            del iv_dict[filename]
            print(f"Deleted {filename} from the JSON file.")

            # Write the updated IV dictionary back to the JSON file
            with open(iv_file_path, 'w') as iv_file:
                json.dump(iv_dict, iv_file)
        else:
            print(f"{filename} not found in the JSON file.")

    except FileNotFoundError:
        print(f"JSON file not found for user '{username}'.")
    except Exception as e:
        print(f"Error deleting from JSON file: {e}")

def password_hashing(plaintext_password, salt):
    """
    hash a given text with the given salt
    :param plaintext_password:
    :param salt:
    :return:
    """
    # Derive the encryption key using PBKDF2 with the plaintext password
    # and salt_encryption
    encryption_key = hashlib.pbkdf2_hmac('sha256',
                                           plaintext_password.encode(), salt, 100000)
    return encryption_key

def encrypt_file(username, file_path, encryption_key):
    """
    encrypt the content of a given file with a given AES key
    :param username:
    :param file_path:
    :param encryption_key:
    :return:
    """

```

```

"""
try:
    # Read the file content
    with open(file_path, 'rb') as file:
        file_content = file.read()
    # Generate a random IV
    iv = os.urandom(16)
    # Create an AES cipher object
    cipher = AES.new(encryption_key, AES.MODE_CBC, iv)
    # Encrypt the file content
    encrypted_content = cipher.encrypt(pad(file_content,
AES.block_size))
    # Write the encrypted content back to the file
    with open(file_path, 'wb') as file:
        file.write(encrypted_content)
    add_to_json(username, file_path,
base64.b64encode(iv).decode('utf-8'))
    print(f"File '{file_path}' encrypted successfully.")
except Exception as e:
    print(f"Error encrypting file: {e}")

def encrypt_folder(username, folder_path, encryption_key):
    """
    encrypt each file within a folder using a stack and the encrypt_file
    function
    :param username:
    :param folder_path:
    :param encryption_key:
    :return:
    """
    # Create a stack to store directories to traverse
    stack = [folder_path]
    # Iterate until the stack is empty
    while stack:
        # Get the current directory from the stack
        current_dir = stack.pop()
        # Get the list of files and directories in the current directory
        items = os.listdir(current_dir)
        # Iterate over each item in the current directory
        for item in items:
            # Get the full path of the item
            item_path = os.path.join(current_dir, item)
            # Check if the item is a file
            if os.path.isfile(item_path):
                # Print the file name
                print(item_path)
                encrypt_file(username, item_path, encryption_key) #
encrypt the file
            # Check if the item is a directory
            elif os.path.isdir(item_path):
                # Add the subdirectory to the stack for traversal
                stack.append(item_path)

def decrypt_file(username, file_path, encryption_key):
    """
    decrypt the content of a given file with a given AES key

```

```

:param username:
:param file_path:
:param encryption_key:
:return:
"""
try:
    # Read the encrypted file content
    with open(file_path, 'rb') as file:
        encrypted_content = file.read()

    # Load the IV dictionary from the JSON file
    iv_file_path = os.path.join(UPLOAD_FOLDER, username,
"encryptions_iv.json")
    with open(iv_file_path, 'r') as iv_file:
        iv_dict = json.load(iv_file)

    # Get the IV for the current file from the dictionary
    iv_base64 = iv_dict.get(file_path)

    if iv_base64:
        iv = base64.b64decode(iv_base64)

        # Create an AES cipher object
        cipher = AES.new(encryption_key, AES.MODE_CBC, iv)

        # Decrypt the file content
        decrypted_content = unpad(cipher.decrypt(encrypted_content),
AES.block_size)

        # Write the decrypted content back to the file
        with open(file_path, 'wb') as file:
            file.write(decrypted_content)

        print(f"File '{file_path}' decrypted successfully.")
    else:
        print(f"IV not found for file '{file_path}'.")
except Exception as e:
    print(f"Error decrypting file: {e}")

def decrypt_folder(username, folder_path, encryption_key):
    """
    decrypt each file within a folder using a stack and the decrypt_file
    function
    :param username:
    :param folder_path:
    :param encryption_key:
    :return:
    """
    # Create a stack to store directories to traverse

    stack = [folder_path]

    # Iterate until the stack is empty
    while stack:
        # Get the current directory from the stack
        current_dir = stack.pop()
        # Get the list of files and directories in the current directory

```

```

items = os.listdir(current_dir)
# Iterate over each item in the current directory
for item in items:
    # Get the full path of the item
    item_path = os.path.join(current_dir, item)
    # Check if the item is a file
    if os.path.isfile(item_path):
        # Print the file name
        print(item_path)
        decrypt_file(username, item_path, encryption_key) #
encrypt the file

    # Check if the item is a directory
    elif os.path.isdir(item_path):
        # Add the subdirectory to the stack for traversal
        stack.append(item_path)

def first_connection(client_socket):
    """
    responsible for the exchange of the AES key and IV at the start of
    each connection
    :return: aes_cipher object
    """
    key = RSA.generate(2048) # generate a pair of keys
    public_key = key.public_key().export_key()
    private_key = key.export_key()

    keys_dict = {"key": public_key}
    client_socket.sendall(protocol.send_message(keys_dict))
    keys_response_dict = protocol.get_message(client_socket)
    encrypted_aes_key = keys_response_dict.get("aes_key")
    encrypted_iv = keys_response_dict.get("iv")

    cipher = PKCS1_OAEP.new(RSA.import_key(private_key))
    aes_key = cipher.decrypt(encrypted_aes_key)
    iv = cipher.decrypt(encrypted_iv)

    # print(f"Decrypted aes key: {aes_key}")
    # print(f"Decrypted iv: {iv}")
    aes_cipher = protocol.AESCipher(aes_key, iv)
    return aes_cipher

def handle_client(client_socket, client_address):
    """
    responsible for handling each user, from the start of connection to
    the end
    :param client_socket:
    :param client_address:
    :return:
    """
    print(f"Accepted connection from {client_address}")
    aes_cipher = first_connection(client_socket)

    username, is_admin, file_encryption_key = login_signup(client_socket,
aes_cipher)
    if is_admin:

```

```

        handle_admin(client_socket, username, aes_cipher)
        return
    if username == "exit":
        print("Client requested exit")
        client_socket.close()
        return
    print(f"hello: {username}")
    while True:
        command_dict = protocol.get_message_aes(client_socket, aes_cipher)
        command = command_dict.get("command")
        print(command)
        if command == "exit":
            print("Client requested exit")
            client_socket.close()
            return
        elif command == "upload_file":
            receive_file(command_dict, username, aes_cipher,
file_encryption_key)
        elif command == "upload_folder":
            receive_folder(command_dict, username, aes_cipher,
file_encryption_key)
        elif command == "refresh":
            refresh(command_dict, client_socket, username, aes_cipher)
        elif command == "download":
            download(command_dict, username, client_socket, aes_cipher,
file_encryption_key)
        elif command == "delete":
            delete(command_dict, username, client_socket, aes_cipher)
        elif command == "create_folder":
            create_folder_by_user(command_dict, username, client_socket,
aes_cipher)

def handle_admin(client_socket, username, aes_cipher):
    """
    responsible for handling admin user, after the connection until the
    end
    :param client_socket:
    :param username:
    :param aes_cipher:
    :return:
    """
    print("welcome admin")
    while True:
        command_dict = protocol.get_message_aes(client_socket, aes_cipher)
        command = command_dict.get("command")
        print(command)
        if command == "exit":
            print("Client requested exit")
            client_socket.close()
            return
        elif command == "refresh_clients":
            server_admin.refresh_clients(client_socket, aes_cipher,
command_dict)
        elif command == "delete_users":
            server_admin.delete_user(client_socket, aes_cipher,
command_dict, username)
        elif command == "get_logs":

```



```

server_admin.send_logs(client_socket, aes_cipher)

def main():
    """
    the first and main function of the server
    for each client connection it creates a new thread
    :return:
    """
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((SERVER_HOST, SERVER_PORT))
    server_socket.listen()

    print(f"Server listening on {SERVER_HOST}:{SERVER_PORT}")

    while True:
        client_socket, client_address = server_socket.accept()

        # Start a new thread for each client
        client_thread = threading.Thread(target=handle_client,
args=(client_socket, client_address))
        client_thread.start()

if __name__ == '__main__':
    main()

```

Server_Admin.py

```

import sqlite3
import protocol
import os
import shutil
from logger import Logger

UPLOAD_FOLDER = 'STORED_DATA'

def refresh_clients(client_socket, aes_cipher, refresh_dict):
    """
    responsible for sending the clients list in the DB
    :param client_socket:
    :param aes_cipher:
    :param refresh_dict:
    :return:
    """
    print("refresh clients")
    try:
        with sqlite3.connect("database.db") as users_db:
            cursor = users_db.cursor()
            query = "SELECT username, admin FROM Users WHERE admin = 0"
            cursor.execute(query)
            users = cursor.fetchall()

            # Calculate the total number of pages

```

```

        rows = refresh_dict.get("rows")
        cols = refresh_dict.get("cols")
        items_per_page = rows * cols
        total_pages = (len(users) + items_per_page - 1) //
items_per_page

        # Get the users for the current page
        page = refresh_dict.get("page", 1)
        start_index = (page - 1) * items_per_page
        end_index = start_index + items_per_page
        users_dict = {username: "user" for username, is_admin in
users[start_index:end_index] if not is_admin}

        total_users = len(users)

        refresh_response_dict = {"command": "response", "users":
users_dict, "total_pages": total_pages, "total_users": total_users}

client_socket.sendall(protocol.send_message_aes(refresh_response_dict,
aes_cipher))

    except sqlite3.Error as e:
        Logger.log_error(f"refresh_clients error: {e}")
        print(f"Database error: {str(e)}")

def send_logs(client_socket, aes_cipher):
    """
    responsible for sending the logs to the admin
    :param client_socket:
    :param aes_cipher:
    :return:
    """
    try:
        with open('user_actions.log', 'r') as log_file:
            log_entries = log_file.readlines()

        logs_dict = {"command": "response", "logs": log_entries}
        client_socket.sendall(protocol.send_message_aes(logs_dict,
aes_cipher))

    except FileNotFoundError:
        print("Log file not found.")
        logs_dict = {"command": "response", "logs": ["No logs
available."]}
        Logger.log_error(f"send_logs error: FileNotFoundError ")
        client_socket.sendall(protocol.send_message_aes(logs_dict,
aes_cipher))
    except Exception as e:
        print(f"Error reading log file: {e}")
        logs_dict = {"command": "response", "logs": ["Error reading log
file."]}
        Logger.log_error(f"send_logs error: {e} ")
        client_socket.sendall(protocol.send_message_aes(logs_dict,
aes_cipher))

def delete_user(client_socket, aes_cipher, delete_user_dict,

```

```

username_admin):
    """
    responsible for deleting users when an admin request it
    :param client_socket:
    :param aes_cipher:
    :param delete_user_dict:
    :param username_admin:
    :return:
    """
    print("delete user")
    users_to_delete = delete_user_dict.get("users")

    try:
        with sqlite3.connect("database.db") as users_db:
            cursor = users_db.cursor()

            for username in users_to_delete:
                # Delete the user from the database
                query = "DELETE FROM Users WHERE username = ?"
                cursor.execute(query, (username,))

                # Delete the user's folder
                user_folder_path = os.path.join(UPLOAD_FOLDER, username)
                if os.path.exists(user_folder_path):
                    shutil.rmtree(user_folder_path)
                    print(f"Deleted folder: {username}")
                else:
                    print(f"Folder not found: {username}")

            users_db.commit()
            Logger.log_account_deletion(username, username_admin)
            delete_user__response_dict = {"command": "response", "msg":
"Users deleted"}

        client_socket.sendall(protocol.send_message_aes(delete_user__response_dic
t, aes_cipher))

    except sqlite3.Error as e:
        print(f"Database error: {str(e)}")
        delete_user__response_dict = {"command": "response", "msg":
f"Error deleting users: {str(e)}"}
        Logger.log_error(f"delete_user error: {e} ")

    client_socket.sendall(protocol.send_message_aes(delete_user__response_dic
t, aes_cipher))

```