תיק פרויקט – Idan Cloud

בית הספר: תיכון הנדסאים הרצליה

שם הפרויקט: IdanCloud

שם התלמיד: עידן חזאי

תעודת זהות: 330905159

שם המנחה: אופיר שביט

שם החלופה:

תאריך הגשה:

תוכן עניינים

- תוכן עניינים 2
- מבוא 5

ייזום: 5
תיאור ראשוני של המערכת: 5
הגדרת הלקוח: 5
יעדים ומטרות: 6
בעיות תועלת וחסכנות: 7
מערכות ותוכנות קיימות: 7
סקירת טכנולוגית: 9
תחומים בהם המערכת עוסקת: 10
תחומים בהם המערכת לא מטפלת: 11
פירוט תיאור המערכת (אפיון): 13
תיאור מפורט של המערכת: 13
פירוט יכולות לכל משתמש: 15
פירוט הבדיקות המתוכננות: 17
ניהול ותכנון לו"ז: 19
ניהול סיכונים ודרכי התמודדות: 21

- תיאור תחום הידע 23

פירוט מעמיק של יכולות המערכת: 23
יכולות בצד שרת: 23
יכולות בצד לקוח: 26

- מבנה/ארכיטקטורה של המערכת 29

תיאור הארכיטקטורה של המערכת: 29
תיאור החומרה: 29
שרטוט: 29
טכנולוגיה רלוונטית: 30
זרימת המידע במערכת: 31
תיאור האלגוריתמיקה בפרויקט: 32
ניסוח וניתוח של הבעיה האלגוריתמית: 32
אלגוריתמים קיימים: 33
סקירת הפתרון הנבחר: 34

תיאור סביבת הפיתוח: 36
כלי הפיתוח הנדרשים: 36
הסביבה והכלים הנדרשים לבדיקות: 36
פרוטוקול התקשורת: 37
תיאור מילולי של הפרוטוקול: 37
פירוט כלל ההודעות הזורמות במערכת: 37
מסכי המערכת: 42
תיאור כל מסך: 42
תרשים מסכים: 49
מבני הנתונים: 50
פירוט מבני הנתונים: 50
פירוט מאגרי המידע של המערכת: 50
מסד הנתונים: 54
סקירת חולשות ואיומים: 58
- מימוש הפרויקט 60
סקירת כל המודולים והמחלקות המרכיבים את המערכת והקשרים ביניהם: 60
מודולים מיובאים: 60
מודולים שפתחתי: 61
קטעי קוד ופיתוחים מיוחדים: 67
מסמך בדיקות מלא: 73
בדיקות שתוכננו בשלב האפיון: 73
בדיקות נוספות: 75
- מדריך למשתמש 78
קבצי המערכת: 78
התקנת המערכת: 80
הסביבה הנדרשת: 80
הכלים הנדרשים: 80
מיקומי קבצים: 80
נתונים התחלתיים: 80
רשת: 80
ארכיטקטורה נדרשת: 80
משתמשי המערכת: 81
אופן הפעלת המשתמש: 81

צילומי מסכי הפרויקט הרלוונטיים: 82

- **רפלקציה..... 85**

תהליך העבודה: 85

תהליך הלמידה: 87

כלים נלקחים להמשך: 89

תובנות מהתהליך: 91

ראייה לאחור: 92

איך הייתי משפר: 93

שאלות חקר עצמי: 94

תודות: 94

- **ביבליוגרפיה..... 95**
- **נספחים 96**

מבוא

ייזום:

תיאור ראשוני של המערכת:

הפרויקט שאני בחרתי לעשות הוא מערכת אחסון ענן, בדומה לGoogle Drive, Dropbox ו-OneDrive. מטרת הפרויקט היא לתת לכל אדם המעוניין להתחבר ולשמור קבצים בשרת, בצורה מהירה, נגישה ונוחה לשימוש.

המוצר המוגמר צריך לבצע קודם כל את הבסיס – לתת למשתמש להעלות קבצים ולשמור אותם, וכן למחוק אותם לפי צורכו. בנוסף לבסיס, למוצר יהיה ממשק משתמש נוח לשימוש ולהבנה על ידי אנשים רבים, QOL פיצ'רים לנוחות המשתמש ועוד המון טכנולוגיות מאחורי הקלעים.

בחרתי לעשות את הפרויקט שלי מכיוון שנתקלתי בהמון שירותי אחסון בענן, אך לכל אחד היו חסרונות משלו שמאוד הפריעו לעבוד ולהשתמש בו. בחלק חסרים יכולות שימושיות כמו שיתוף קבצים, אחרים קשים מאוד לשימוש ורישום, חלק דורשים תשלום אפילו לתוכנית הבסיסית ולכן רציתי לבנות בעצמי תוכנה שיכולה לבצע את כל מה שחשוב לדעתי במקום אחד וגם ניתנת לעריכה ולהוספת אופציות על ידי.

לפני שאני אתחיל לכתוב את הקוד של הפרויקט יש כמה אתגרים שיכולים לצוץ. מערכת שיתוף הקבצים יכולה להיות טריקית מכיוון שכאשר מישהו משתף קובץ ולאחר מכן משנה את השם שלו, איך ניתן לעדכן זאת גם אצל האדם שאיתו שיתפו? עוד אתגר שאני צופה הוא יצירת תקשורת לא סינכרונית בין השרת ללקוח. כלומר, עד עכשיו בפרויקטים שעשיתי התקשורת בין השרת ללקוח הייתה סינכרונית – על כל הודעה שהלקוח היה שולח לשרת הוא היה מקבל תשובה כלשהי. בפרויקט הזה ישנן פעולות שיכולות לקחת הרבה זמן כמו לדוגמה העלאת קובץ מאוד גדול, ולכן אני לא רוצה שבזמן הזה הלקוח יהיה תקוע ותקשורת לא סינכרונית פותרת את הבעיה הזאת. תקשורת כזו היא יותר מסובכת שכן צריך לדעת לנהל את כל ההודעות בצד הלקוח.

בנוסף לאתגרים הטכניים, זהו הפרויקט הגדול ביותר שכתבתי, ולכן ניהול הקוד וחלוקתו לקבצים נפרדים ומחלקות יכול להיות מאתגר מאוד.

הגדרת הלקוח:

המערכת מיועדת לאנשים פרטיים וארגונים קטנים עד בינוניים הזקוקים לפתרון אמין ומאובטח לניהול קבצים דיגיטליים. באופן ספציפי, קהל היעד כולל: אנשי מקצוע שצריכים לאחסן ולגשת למסמכים

חשובים בצורה מאובטחת. סטודנטים ומחנכים המנהלים תיקים אקדמיים כגון מטלות, עבודות מחקר ומצגות. משתמשים כלליים שרוצים מקום בטוח ונוח לאחסון קבצים אישיים כמו תמונות, סרטונים ותוכן דיגיטלי אחר.

צוותים הדורשים כלים משותפים כדי לשתף ולנהל מסמכים בצורה מאובטחת. עסקים שזקוקים לפתרון פשוט אך מאובטח לארגון ואחסון נתונים רגישים כמו חוזים ודוחות.

עיצוב המערכת נותן עדיפות לנוחות השימוש עבור משתמשים בודדים תוך שמירה על ביטחון המידע הנדרש לשימוש מקצועיים. על ידי שימוש בממשק פשוט, אפשרויות גמישות לניהול קבצים ואבטחה חזקה, המערכת שואפת לענות על הצרכים המגוונים של בסיס המשתמשים שלה ביעילות.

<u>יעדים ומטרות:</u>

המטרה העיקרית של המערכת היא לספק פתרון אחסון ענן אמין, מאובטח וידידותי למשתמש לניהול קבצים דיגיטליים. כדי להשיג זאת, המערכת מתמקדת ביעדים הבאים:

אחסון קבצים מאובטח – המערכת מוודאת שמשתמשים יכולים לאחסן בבטחה את הקבצים והתיקיות שלהם מבלי לדאוג לגבי גישה לא מורשית או אובדן נתונים.

שיתוף ושיתוף קבצים – מאפשר למשתמשים לשתף קבצים בקלות עם אחרים, מה שמקל על שיתוף פעולה וחילופי מידע יעילים.

ארגון וניהול – מספק כלים למשתמשים לארגן את הקבצים שלהם בתיקיות, לשנות שמות של קבצים ותיקיות ולתחזק מערכת קבצים מובנית.

ניהול חשבון – מאפשר למשתמשים ליצור חשבונות, להתחבר בצורה מאובטחת, לשנות את שמות המשתמש שלהם ולמחוק חשבונות בעת הצורך.

פרטיות נתונים והצפנה – מגן על נתוני משתמש על ידי הצפנת כל התקשורת בין הלקוח לשרת, הבטחת פרטיות ואבטחה במהלך העברת הנתונים.

חוויה ידידותית למשתמש – ממשק פשוט ואינטואיטיבי כך שמשתמשים יוכלו לבצע משימות ביעילות, ללא קשר למומחיות הטכנית שלהם.

מדרגיות – המערכת מתוכננת להתמודדות עם צמיחה, הכלה של יותר משתמשים וכמויות גדולות יותר של נתונים לאורך זמן.

על ידי עמידה ביעדים אלו, המערכת שואפת לספק פלטפורמת אחסון ענן רב-תכליתית ומהימנה לשימוש אישי ומקצועי.

בעיות תועלת וחסכנות:

ישנן כמה בעיות עיקריות שאותן אני שואף לפתור בפיתוח המערכת והמטרה הסופית היא ליצור מערכת שעובדת בצורה חלקה ללא חשש מאיבוד מידע או קריסה של התוכנה.

הבעיות העיקריות הינן:

1. ניהול המידע – המידע צריך להיות מנוהל בשרת באופן מאובטח שמאפשר רק לאנשים בעלי גישה לגשת לקבצים ולהבטיח לכל משתמש כי המידע שלו בטוח.
2. שיתוף מידע בצורה מאובטחת – כל משתמש צריך להיות בשליטה על הקבצים שלו ומי יכול לראות אותם ולערוך אותם. אין מצב בו קובץ של משתמש הגיע למשתמש אחר בצורה לא רצויה. הבטחון המיידעי הזה אינו יפגע ביכולות המשתמש.
3. שימוש פשוט – המערכת צריכה להיות פשוטה לשימוש ובעלת ממשק משתמש נוח ולא יותר מדי עמוס, וזה בזמן שהיא תהיה עשירה בפיצ'רים.

מערכות ותוכנות קיימות:

Google Drive הוא שירות אחסון ענן מבית גוגל שמאפשר למשתמשים לשמור, לשתף ולגשת לקבצים שלהם מכל מכשיר המחובר לאינטרנט. המערכת מציעה אינטגרציה חלקה עם שאר שירותי גוגל, כמו Google Docs, Sheets ו Slides,-מה שמאפשר עבודה משותפת על מסמכים בזמן אמת. המשתמשים יכולים להגדיר הרשאות גישה שונות לקבצים ולשתף אותם עם משתמשים אחרים באמצעות קישורים Google Drive .מספק שטח אחסון חינמי של 15 GBלכל משתמש, עם אפשרות לרכוש נפח אחסון נוסף דרך .Google One השירות כולל גם יכולות סנכרון עם המחשב המקומי, מה שמאפשר גיבוי אוטומטי של קבצים ותיקיות. מבחינת אבטחה, הקבצים מוצפנים בעת ההעברה ובעת האחסון, אך גוגל עדיין שומרת לעצמה גישה לנתונים לצורכי ניתוח ושיפור שירותים, מה שעשוי להוות חיסרון למשתמשים שמחפשים פרטיות מלאה.

iCloud הוא שירות אחסון הענן של אפל, המיועד בעיקר למשתמשי מכשירי Apple כמו iPhone, iPad .Mac-השירות מספק סנכרון אוטומטי של תמונות, מסמכים, אנשי קשר, תזכורות ונתונים נוספים

בין כל המכשירים של המשתמש, מה שמאפשר חוויית שימוש חלקה לכל מי שמשתמש באקוסיסטם של אפל iCloud .משולב עמוק בתוך מערכת ההפעלה של אפל ומאפשר גם גיבוי מלא של מכשירים, כך שבמקרה של אובדן או תקלה ניתן לשחזר את כל הנתונים בקלות. השירות מציע 5 GBשל אחסון חינמי, עם אפשרות לרכוש שטח נוסף דרך מנוי .+iCloud מבחינת אבטחה, אפל משתמשת בהצפנה מקצה לקצה עבור מידע רגיש כמו סיסמאות ונתוני בריאות, אך קבצים רגילים מאוחסנים עם הצפנה סטנדרטית, מה שאומר שלאפל יש יכולת לגשת אליהם במקרה הצורך. היתרון המרכזי של iCloud הוא הסנכרון האוטומטי והעמוק עם מוצרי אפל, אך החיסרון הוא התאימות המוגבלת למערכות הפעלה שאינן של אפל, מה שהופך אותו לפחות נוח למשתמשי Windows אוAndroid.

Nextcloud היא פלטפורמת אחסון קבצים בענן בקוד פתוח, המספקת למשתמשים אפשרות להקים שרת אחסון פרטי ולשלוט באופן מלא על הנתונים שלהם. המערכת תומכת בסנכרון קבצים בין מכשירים, שיתוף קבצים עם משתמשים אחרים וניהול הרשאות מתקדם. היא כוללת גם תמיכה באימות דו-שלבי והצפנת קבצים כדי לשמור על אבטחת המידע. אחד היתרונות המרכזיים של Nextcloud הוא היכולת להרחיב את המערכת עם תוספים שמאפשרים אינטגרציה עם כלי עבודה נוספים כמו מסמכים, לוחות שנה ושיחות וידאו.

ownCloud דומה ל Nextcloud-אך ממוקד בעיקר בארגונים הדורשים פתרון אחסון ענן פרטי עם שליטה מלאה על הנתונים. המערכת מאפשרת אחסון קבצים, סנכרון בין מכשירים ושיתוף מידע בצורה מאובטחת. היא מתאפיינת בשימוש בפרוטוקולי אבטחה מתקדמים ותומכת באימות דו-שלבי ובהצפנת נתונים מקצה לקצה. ל ownCloud-יש גרסה קהילתית חינמית לצד גרסה מסחרית המיועדת לארגונים הזקוקים לתמיכה מקצועית ואפשרויות התאמה אישית רחבות.

Seafile הוא פתרון אחסון ענן המתמקד בביצועים גבוהים ובסנכרון מהיר של קבצים. המערכת פועלת על עיקרון של אחסון מבוסס בלוקים, מה שמאפשר לה להקטין את זמן ההעלאה וההורדה על ידי סנכרון רק של החלקים שהשתנו בקובץ במקום להעלות את הקובץ כולו מחדש Seafile .כולל תמיכה בגישה מבוססת דפדפן, אפליקציות שולחניות ומובייל, וניהול הרשאות פרטני לכל קובץ או תיקייה. הוא מתאים בעיקר לארגונים המחפשים פתרון יעיל ומהיר לניהול מסמכים ונתונים.

Pydio היא מערכת אחסון קבצים בענן שמתמקדת באבטחה ובהתאמה אישית לארגונים. המערכת מאפשרת יצירת סביבת עבודה שיתופית עם אפשרויות מתקדמות לשליטה בהרשאות וגישה מאובטחת לקבצים Pydio .מציעה תכונות כמו הצפנה מובנית, מעקב אחר פעילות המשתמשים ויכולת התאמה

אישית של הממשק כדי שיתאים לצרכים ארגוניים ספציפיים. אחד היתרונות הבולטים שלה הוא היכולת לנהל קבצים על פני מספר שרתים שונים וליצור מדיניות גיבוי מותאמת.

Syncthing היא מערכת קוד פתוח המתמקדת בסנכרון קבצים מבוזר ללא שימוש בשרת מרכזי. בניגוד לפתרונות אחסון ענן אחרים Syncthing ,מאפשרת למשתמשים לסנכרן קבצים ישירות בין מכשירים שונים ללא צורך בהעלאה לענן. המערכת משתמשת בפרוטוקולי הצפנה מתקדמים כדי להגן על המידע המועבר, והיא מתאימה במיוחד למי שרוצה לשמור על פרטיות הנתונים שלו מבלי להסתמך על שירותי ענן חיצוניים.

כל אחת מהמערכות האלו מספקת פתרון דומה לשלך אך עם דגשים שונים, בין אם זה שליטה ארגונית, פרטיות, ביצועים או סנכרון ישיר בין מכשירים.

סקירת טכנולוגית:

הפרויקט שלי אינו עושה בשימוש בטכנולוגיה חדשה ולא מוכרת, בעזרת שימוש בשפת תכנות פייתון (שהיא מהחדשניות בעולם ובעלת המון אופציות שימושיות) וניהול מסד נתונים בSQL, המערכת שלי תוכל לבצע את המטרה שלה, שכן קיימת כבר אך בעלת יתרונות על המתחרים, בצורה הכי יעילה מבחינת משאבים וכן נוחה למשתמש.

ישנם כמה סייגים שחשוב לציין במערכת שלי. ראשית המערכת נועדה לרוץ על שרת אחד, שכן כאשר המון משתמשים ירצו להתחבר שרת נוסף יצטרך להיפתח ולו לא יהיה את אותו מסד הנתונים. בנוסף לכך, נדרש שרת חזק מאוד ובעל אחסון רב מאוד בשביל שהמון משתמשים יוכלו לאחסן את המידע שלהם בו. עוד בעיה שנוצרת בשימוש במערכת שלי היא שהמערכת אינה כוללת Data Redundancy, כלומר אם קורה משהו לאחד מהכוננים – הקבצים שבו יאבדו ולא יהיה ניתן לשחזרם.

תיחום הפרויקט: הפרויקט שלי עוסק במגוון תחומים טכנולוגיים, תוך התמקדות בממשקים בין רשתות, מערכות הפעלה, וניהול נתונים. להלן תחומי העיסוק המרכזיים של המערכת, לצד התחומים שאינם מכוסים.

תחומים בהם המערכת עוסקת:

1. **רשתות תקשורת -**

- **אבטחת תעבורה**: הצפנת נתונים בתעבורה באמצעות הצפנות (AES ,RSA), מניעת גישה לא מורשית תוך שימוש במנגנוני אימות.
- **ניהול חיבורים בענן:** יצירת חיבור מאובטח בין לקוחות (Clients) לשרתים בענן, תמיכה בגישה ממכשירים שונים עם כתובת IP דינמית או סטטית.
- **ניהול שיתוף ברשת**: הפעלת מנגנונים לשיתוף קבצים מאובטח בין משתמשים ברשת, הגדרת הרשאות מבוססות תפקידים עבור משתמשים שונים.

2. **מערכות הפעלה -**

- **אינטגרציה בין מערכות הפעלה**: המערכת מאפשרת גישה מכל מערכות ההפעלה העיקריות : Windows, macOS, Linux, ואפילו iOS ו Android-באמצעות דפדפנים או אפליקציות.
- **ניהול קבצים ותיקיות**: מנגנון לארגון קבצים ותיקיות, דומה למערכת הקבצים במערכות הפעלה מקומיות, שימוש ב Metadata-לניהול מידע על הקבצים כמו גודל, שם, יוצר, ותאריך שינוי.
- **תמיכה בנפחים גדולים**: אפשרות לניהול אחסון בנפחים משמעותיים עם מיטוב ביצועי**.**

3. **ניהול נתונים במסד נתונים -**

- **אחסון מבוסס מסד נתונים** :שמירת מידע על קבצים, תיקיות, משתמשים, הרשאות, ושיתופים בבסיס נתונים רלציוני (SQL) או.NoSQL ,ניהול גרסאות קבצים.(File Versioning)
- **אופטימיזציה**: שמירת נתונים דחוסים להקטנת נפח האחסון הנדרש. מנגנוני גיבוי ושחזור נתונים במקרה של תקלה.

4. **אבטחת מידע -**

- **הצפנה מקצה לקצה**: שמירה על פרטיות המשתמשים על ידי הצפנה של מידע בזמן העלאה ואחסון.
- **אימות משתמשים**: מנגנונים כמו סיסמאות מוצפנות, אימות דו-שלבי ,(2FA) וניהול הרשאות מותאם אישית.
- **ניהול גישה מבוסס תפקידים**:**(RBAC)** ניהול הרשאות פרטני לפי קובץ, תיקיה, ומשתמש.

<u>תחומים בהם המערכת לא מטפלת:</u>

1. **תחומי רשתות שאינם מכוסים -**

- **ניהול תשתית רשת**: המערכת אינה עוסקת בהגדרת שרתים, ניהול מתגים, או רכיבי חומרה אחרים של הרשת.
- **חיבוריות בין עננים**:**(Multi-Cloud)** המערכת אינה מציעה ניהול אחסון משולב בין ספקי ענן שונים) כגון Google Drive ו.(AWS-
- **אופטימיזציה של מהירות תעבורה**: המערכת לא כוללת מנגנונים אוטומטיים להאצת העלאות או הורדות, כמו רשתות.CDN

2. **תחומי מערכות הפעלה שאינם מכוסים -**

- **שילוב ישיר עם מערכת הקבצים**: אין אפשרות להריץ את המערכת ישירות על מערכת קבצים מקומית) כמו FUSE ב.(Linux-
- **גישה ללא אינטרנט**: המערכת מחייבת חיבור פעיל לאינטרנט ואינה מספקת פתרונות לגישה במצב לא מקוון.(Offline)
- **שילוב עם אפליקציות צד שלישי**: המערכת אינה תומכת באופן מובנה בשילוב עם תוכנות כמו Office.Adobe Reader

3. **תחומים עסקיים ותמיכת לקוח -**

- **שירותים מבוססי**:**AI** אין שימוש באלגוריתמים מבוססי בינה מלאכותית לניתוח נתונים (כגון ניתוח גודל שימוש, הצעות לניהול קבצים).
- **תמיכה טכנית אוטומטית**: המערכת אינה כוללת צ'אטבוט אוטומטי או פורטל תמיכה מתוחכם.

4. **תחומי אבטחה שאינם מכוסים -**

- **זיהוי איומים בזמן אמת**: המערכת אינה כוללת כלים לזיהוי מתקפות סייבר בזמן אמת.
- **מנגנוני אבטחה פיזיים**: לא מוצעים פתרונות לאבטחת שרתים פיזית או גיבויים חוץ-משרדיים.

המערכת מתמקדת בניהול קבצים ושיתוף מאובטח בענן, תוך שימוש בטכנולוגיות רשתות, אבטחת מידע, וניהול מערכות קבצים. עם זאת, היא אינה מתעסקת בתחומים טכניים עמוקים יותר כמו ניהול

רשתות פיזיות, שילוב מערכות קבצים מקומיות, או אינטגרציות רחבות עם שירותי צד שלישי. תיחום זה מאפשר לי להתמקד ביעילות ובשיפור המערכת בנקודות החוזקה שלה.

פירוט תיאור המערכת (אפיון):

תיאור מפורט של המערכת:

מערכת אחסון הענן פותחה כדי לספק פתרון מאובטח, גמיש ויעיל לניהול ושיתוף קבצים בין משתמשים. המערכת מאפשרת למשתמשים להעלות, להוריד, לשתף, לערוך ולנהל קבצים ותיקיות בצורה נוחה, תוך שמירה על אבטחת המידע באמצעות הצפנה ובקרת הרשאות. היא מבוססת על מודל שרת-לקוח, כאשר השרת מאחסן את הקבצים בצורה מסודרת במסד הנתונים, והלקוח מתחבר אליו לצורך ביצוע פעולות שונות.

פונקציונליות עיקרית:

העלאה, הורדה וניהול קבצים - המערכת מאפשרת למשתמשים להעלות קבצים לכל תיקייה שבבעלותם, להוריד קבצים קיימים ולמחוק קבצים מיותרים. קבצים נשמרים במסד נתונים במקום ישיר במערכת הקבצים, מה שמאפשר שליטה טובה יותר, גיבויים יעילים ואבטחה מוגברת. ניתן גם לשחזר העלאה או הורדה שנקטעה במקרה של התנתקות מהשרת, מה שמאפשר חוויית שימוש חלקה גם בתנאי רשת משתנים.

בקרת הרשאות ושיתוף קבצים - משתמשים יכולים לשתף קבצים עם אחרים ולהגדיר הרשאות גישה בהתאם לצרכים שלהם. המערכת מאפשרת שליטה ברמת כל קובץ או תיקייה, כך שניתן להגדיר הרשאות קריאה, כתיבה, מחיקה ושיתוף חוזר עבור כל משתמש בנפרד. ניתן גם להגדיר קישורי שיתוף זמניים שיפוגו לאחר פרק זמן מוגדר, מה שמעניק למשתמשים שליטה מלאה על הקבצים שהם משתפים.

אבטחה והצפנת תקשורת - המערכת עושה שימוש בהצפנה מתקדמת כדי להבטיח שמידע המשתמשים מוגן מפני גורמים חיצוניים. בעת יצירת החיבור בין הלקוח לשרת, מתבצעת החלפת מפתחות מוצפנים באמצעות פרוטוקול RSA, שלאחריה כל הנתונים מועברים כשהם מוצפנים עם AES. כך נמנעת אפשרות ליירוט מידע על ידי צד שלישי. בנוסף, הנתונים נשמרים בצורה מאובטחת במסד הנתונים, כאשר גישה ישירה לקבצים מתאפשרת רק למשתמשים המורשים.

ממשק משתמש דינמי וניהול נוח - הלקוח בנוי עם **PyQt6**, מה שמאפשר ממשק גרפי אינטואיטיבי שניתן להתאים לרזולוציות מסך שונות. הכפתורים והאלמנטים בממשק מתאימים את עצמם באופן דינמי כדי לספק חוויית משתמש מיטבית. המשתמשים יכולים לנווט בקלות בין תיקיות, לבצע פעולות גרירה

ושחרור (Drag & Drop) לניהול קבצים, ולערוך קובצי טקסט ותמונות ישירות מתוך המערכת מבלי להוריד אותם.

אימות משתמשים וניהול חשבונות - כל משתמש חייב להירשם ולהתחבר למערכת עם כתובת דוא"ל וסיסמה. בעת ההרשמה, המערכת שולחת קוד אימות למייל של המשתמש כדי לוודא את זהותו.
המערכת תומכת גם בשינוי שם משתמש, איפוס סיסמה והגדרת משתמשים ברמות שונות (כגון מנהלים בעלי הרשאות מורחבות).

סל מחזור ושחזור קבצים שנמחקו - כאשר משתמש מוחק קובץ, הוא אינו נמחק מידית מהשרת אלא מועבר ל"תיקיית נמחקו לאחרונה". משתמשים יכולים לשחזר קבצים שנמחקו בטעות או למחוק אותם לצמיתות לאחר זמן מוגדר.

תקשורת אסינכרונית ושיפור ביצועים - כדי למנוע קריסות והיתקעויות, המערכת משתמשת בגישה **אסינכרונית** לכל הבקשות לשרת. כך ניתן להעלות ולהוריד קבצים ברקע, תוך כדי המשך עבודה על משימות אחרות. כל התקשורת מתבצעת בפרוטוקול TCP, עם ניהול משאבים יעיל שמבטיח עבודה יציבה גם בעומסים גבוהים.

<u>ארכיטקטורת המערכת:</u>

צד השרת - השרת מנהל את כל נתוני המשתמשים, קבצי האחסון ומערכת ההרשאות. הוא מכיל את מסד הנתונים שבו נשמרים כל המידע והמטא-נתונים של הקבצים, ומטפל בבקשות מהלקוחות כגון העלאה, הורדה, מחיקה ושיתוף קבצים. התקשורת בין הלקוח לשרת מוצפנת, והשרת שומר על ניהול משאבים חכם כדי לתמוך במספר רב של חיבורים בו-זמנית.

צד הלקוח - הלקוח מספק ממשק גרפי למשתמשים ומאפשר להם לנהל את הקבצים שלהם בצורה נוחה. הוא כולל מנגנון התחברות מאובטח, יכולות תצוגה ועריכה של קבצים, מערכת הרשאות מתקדמת, וניהול חכם של הורדות והעלאות. כל הבקשות לשרת מבוצעות בצורה אסינכרונית כדי למנוע קיפאון של הממשק בזמן פעולות כבדות.

רשת ותקשורת - המערכת תומכת בגישה דרך רשתות מקומיות (LAN) וגם באינטרנט .(WAN) היא מזהה את כתובת ה IP-של השרת באופן **אוטומטי דרך** ,**DHCP** כך שאין צורך להגדיר כתובת ידנית.
המשתמשים יכולים להתחבר לשרת מכל מקום, כל עוד יש להם הרשאות מתאימות.

תשתית ואדריכלות מינימלית להפעלת המערכת:

כדי להפעיל את המערכת, נדרשת חומרה מינימלית שתתמוך בפעילות תקינה:

שרת :מעבד **Intel i5 / Ryzen 5**ומעלה, זיכרון ,**4GB RAM**אחסון **+10GB**פנויים, חיבור רשת מהיר 100Mbps) לפחות.(

לקוח :מחשב עם ,**Windows/Linux/macOS**מעבד **Intel i3**ומעלה ,זיכרון .**2GB+ RAM**

מערכת רשת :יציבות חיבור ,**TCP/IP**תמיכה בפרוטוקול **DHCP**או כתובת IP קבועה לשרת.

יתרונות המערכת

שליטה מלאה –כל הקבצים מנוהלים בשרת פרטי ללא תלות בספקי אחסון חיצוניים כמו Google Drive אוiCloud.

אבטחה גבוהה –הצפנה מקצה לקצה, אימות דו-שלבי ואפשרות לניהול הרשאות פרטני לכל קובץ.

סנכרון והמשכיות –ניתן להמשיך הורדה או העלאה לאחר ניתוק מהשרת, ללא צורך להתחיל מחדש.

שיתוף גמיש –הגדרת הרשאות שיתוף מותאמות אישית עם תוקף מוגדר מראש.

תמיכה במספר מכשירים –גישה נוחה מהמחשב האישי, עם אפשרות להתאמה

פירוט יכולות לכל משתמש:

המערכת מחולקת לשלושה סוגי משתמשים :משתמש רגיל, משתמש פרימיום ואדמין .לכל סוג משתמש יש גישה שונה לפונקציות ולמשאבי המערכת, בהתאם להרשאות שהוקצו לו.

משתמש רגיל:

משתמש רגיל הוא כל משתמש שנרשם למערכת ומקבל גישה לתכונות הבסיסיות של אחסון הקבצים.

משתמש רגיל יכול להעלות, להוריד, למחוק ולארגן קבצים בתוך שטח האחסון שהוקצה לו. קבצים שנמחקו מועברים לסל המחזור וניתן לשחזרם בתוך זמן מוגבל.

המשתמש יכול לשתף קבצים עם אחרים, אך מוגבל בכמות הקבצים שניתן לשתף ובזמן שבו הקישורים יהיו פעילים.

משתמש רגיל מקבל כמות אחסון קבועה מראש) למשל 5 .(GBאם הוא רוצה להעלות יותר קבצים, עליו למחוק קבצים ישנים או לשדרג לחשבון פרימיום.

המשתמש צריך להתחבר עם שם משתמש וסיסמה, ויכול להפעיל אימות דו-שלבי.

משתמש פרימיום:

משתמשי פרימיום הם משתמשים ששדרגו את החשבון שלהם ומשלמים עבור גישה לתכונות מתקדמות ושיפור חוויית השימוש במערכת.

משתמש פרימיום יכול להעלות, להוריד, למחוק ולנהל קבצים ללא מגבלות גודל קובץ מחמירות, וכן ליהנות מהעלאה והורדה מהירה יותר.

משתמשי פרימיום יכולים לשתף קבצים ללא מגבלות זמן, להגדיר הרשאות גישה מתקדמות (כגון עריכה או קריאה בלבד), ולהגן על קבצים משותפים באמצעות סיסמה.

המשתמש מקבל נפח אחסון מוגדל) למשל 100 GBאו יותר, בהתאם למסלול שלו.(

ניתן לשחזר גרסאות ישנות יותר של קובץ במקרה של מחיקה או שינוי לא רצוי.

ניתן להגדיר סנכרון ישיר בין מחשבים אישיים ושרת המערכת, כך שהקבצים יהיו זמינים תמיד מכל מכשיר מחובר.

משתמשי פרימיום נהנים מרמת אבטחה משופרת, כולל הצפנה מתקדמת יותר וגיבויים אוטומטיים של כל הקבצים בענן.

אדמין (מנהל מערכת):

אדמין הוא משתמש בעל הרשאות ניהול מלאות במערכת. הוא אחראי על ניהול המשתמשים, בקרת הנתונים, הגדרות השרת וניהול האבטחה.

האדמין יכול ליצור, למחוק או לשנות הרשאות לכל המשתמשים במערכת, כולל שדרוג חשבונות רגילים לחשבונות פרימיום.

יש לו גישה ליומן פעילות (Logs) שמאפשר לעקוב אחר העלאות, הורדות, מחיקות וניסיונות כניסה כושלים.

האדמין יכול לגשת לכל הקבצים במערכת, לערוך או למחוק קבצים ולנהל את האחסון הכללי של השרת.

האדמין שולט בהגדרות כגון גודל מקסימלי של העלאת קובץ, מספר חיבורים מקביליים, והקצאת שטח אחסון למשתמשים.

האדמין יכול לאכוף מדיניות אבטחה, לחסום משתמשים חשודים, לשנות הגדרות הצפנה ולנהל עדכוני מערכת.

האדמין יכול להפיק דוחות שימוש על שטח האחסון, עומסי השרת, ושימוש של כל משתמש כדי לייעל את ביצועי המערכת.

המערכת מעניקה למשתמשים רגילים גישה לניהול ושיתוף קבצים בצורה בסיסית עם הגבלות מסוימות. משתמשי פרימיום נהנים מתכונות מתקדמות כמו נפח אחסון גדול יותר, שיתוף ללא הגבלות, גיבויים וסנכרון אוטומטי. אדמין שולט על המערכת כולה, מנהל משתמשים, מנהל הרשאות, שולט באבטחה ומפקח על פעילות השרת. כך מתקבלת מערכת מאורגנת וגמישה שבה לכל משתמש יש את ההרשאות והכלים המתאימים לצרכיו.

פירוט הבדיקות המתוכננות:

בדיקות הן חלק מהותי בתהליך הפיתוח על מנת להבטיח שהמערכת פועלת כהלכה ועונה על הדרישות שהוגדרו. להלן רשימת הבדיקות המתוכננות, כולל תיאור המטרה של כל בדיקה ואופן הביצוע שלה.

1. שיחזור העלאה והורדה לאחר התנתקות/קריסה

מטרת הבדיקה - לוודא שהמשתמש יכול להמשיך בהעלאה או בהורדה של קובץ מהמקום שבו הפעולה נעצרה במקרה של ניתוק או קריסת התוכנה.

מה יבוצע בפועל - בהעלאת קובץ גדול (1GB) לשרת. לאחר שהועלו 50% מהנתונים, ניתוק החיבור באופן יזום. חידוש את ההתחברות ובדיקה אם ההעלאה ממשיכה מהמקום שבו הופסקה.

2. תקשורת אסינכרונית

מטרת הבדיקה - לוודא שהמערכת אינה קופאת כאשר מתבצעות העלאות והורדות של קבצים ברקע.

מה יבוצע בפועל - הפעלת העלאת קובץ תוך כדי ניווט בממשק המשתמש. בדיקה אם לחצנים אחרים בממשק עדיין מגיבים. בדיקה האם ניתן להתחיל פעולות נוספות במקביל.

3. שיתוף קבצים

מטרת הבדיקה - לוודא שמשתמשים יכולים לשתף קבצים עם אחרים, ושמערכת ההרשאות עובדת כראוי.

מה יבוצע בפועל - משתמש A שיתף קובץ עם משתמש B עם הרשאת "קריאה בלבד." משתמש B ינסה לפתוח את הקובץ ולערוך אותו. בדיקה אם אפשר לבטל שיתוף ולהחזיר אותו.

4. שמירת התחברות / התחברות אוטומטית

מטרת הבדיקה - לוודא שמשתמשים יכולים להתחבר אוטומטית אם בחרו באפשרות זו.

מה יבוצע בפועל - משתמש יתחבר עם שמירת התחברות. התוכנה נסגרה ונפתחה מחדש. בדקתי אם המשתמש נשאר מחובר ללא צורך בהזנת סיסמה מחדש.

5. הצפנה ותקשורת מאובטחת

מטרת הבדיקה - לוודא שכל הנתונים המועברים מוצפנים ושלא ניתן להאזין להם.

מה יבוצע בפועל - שליחת קובץ לשרת תוך שימוש ב Wireshark-כדי לבדוק האם התוכן מועבר בטקסט ברור. בדיקה אם השרת והלקוח מחליפים מפתחות מוצפנים לפני תחילת ההעברה.

6. עריכת קבצים וצפייה

מטרת הבדיקה - לוודא שמשתמשים יכולים לצפות ולערוך קבצים נתמכים ישירות מהמערכת.

מה יבוצע בפועל – פתיחת קבצים לעריכה ובדיקה האם העריכה נשמרת

7.הורדה מרובת קבצים כZIP-

מטרת הבדיקה - לוודא שמשתמשים יכולים לבחור מספר קבצים וליצור קובץ ZIP להורדה.

מה יבוצע בפועל - סימון מספר קבצים ולחיצה על "הורד כ."ZIP- לבדוק את תקינות קובץ ה ZIP- שנוצר.

<u>ניהול ותכנון לו"ז:</u>

שלבי פיתוח מרכזיים ("אבנים גדולות") -

1. **איסוף דרישות ותכנון ראשוני**

מטרה: להבין את הצרכים והדרישות מהמערכת, לתכנן ארכיטקטורה, ולבנות מפרט טכני - זמן מתוכנן: 2 שבועות.

2. **תכנון ועיצוב ממשק משתמש (UI/UX)**

מטרה: יצירת אב-טיפוס לממשק, עיצוב חוויית משתמש ברורה ונוחה - זמן מתוכנן: 2 שבועות.

3. **פיתוח צד שרת (Backend Development)**

מטרה: פיתוח ה ,API-מסד הנתונים, והלוגיקה העסקית - זמן מתוכנן: 4 שבועות.

4. **פיתוח צד לקוח (Frontend Development)**

מטרה: פיתוח ממשק המשתמש על פי העיצוב שהוגדר, כולל התממשקות עם הAPI - זמן מתוכנן: 4 שבועות.

5. **אינטגרציה ובדיקות מערכת**

מטרה: חיבור כל רכיבי המערכת יחדיו וביצוע בדיקות פונקציונליות, ביצועים, אבטחה, ותאימות - זמן מתוכנן: 3 שבועות.

6. **תיקונים ושיפורים**

מטרה: טיפול בבאגים ושיפור ביצועים בהתבסס על תוצאות הבדיקות - זמן מתוכנן: 2 שבועות.

7. **פריסה וסיום פרויקט**

מטרה: פריסת המערכת בשרתים וכתיבת תיעוד מלא לשימוש ותחזוקה - זמן מתוכנן: 2 שבועות.

תכנון לוח זמנים ראשוני:

משך (שבועות)	סיום מתוכנן	תחילת עבודה	שלב
2	שבוע 2	שבוע 1	איסוף דרישות ותכנון ראשוני
2	שבוע 4	שבוע 3	תכנון ועיצוב ממשק משתמש
4	שבוע 8	שבוע 5	פיתוח צד שרת
4	שבוע 12	שבוע 9	פיתוח צד לקוח
3	שבוע 15	שבוע 13	אינטגרציה ובדיקות מערכת
2	שבוע 17	שבוע 16	תיקונים ושיפורים
2	שבוע 19	שבוע 18	פריסה וסיום פרויקט

לוח זמנים בפועל:

משך (שבועות)	שלב
4	איסוף דרישות ותכנון ראשוני
3	תכנון ועיצוב ממשק משתמש
1	פיתוח צד שרת
2	פיתוח צד לקוח
0.5	אינטגרציה ובדיקות מערכת
3	תיקונים ושיפורים
1	פריסה וסיום פרויקט

ניהול סיכונים ודרכי התמודדות:

סיכון 1: עיכובים בפיתוח

מהות הסיכון :חלקים מהפיתוח עשויים להתעכב בשל בעיות טכניות, חוסר בתיאום בין צוותים, או מחסור במשאבים.

דרכים להתמודד:

- תכנון לוח זמנים ריאלי עם זמן "חיץ" למקרים לא צפויים.
- חלוקה של משימות גדולות למשימות קטנות ומעקב אחר התקדמותן.
- שימוש בכלים לניהול פרויקטים (Trello, Jira) כדי לשמור על סדר ומעקב.
- אם מתגלה עיכוב, תגבור כוח אדם או העברת משימות לצוותים אחרים.

מה בוצע בפועל:

- עדכון לוח הזמנים בעקבות עיכובים טכניים בשלבי האינטגרציה.
- הוספת מפתח נוסף לצוות הפיתוח במידת הצורך.

<u>סיכון 2: חוסר יציבות מערכת במהלך בדיקות</u>

מהות הסיכון :המערכת עלולה לקרוס במהלך בדיקות עומס או פונקציונליות עקב באגים או תכנון שגוי.

דרכים להתמודד:

- ביצוע בדיקות יחידה (Unit Tests) מוקדם ככל האפשר על רכיבים מרכזיים.
- ביצוע בדיקות אינטגרציה בשלבים מוקדמים כדי לאתר בעיות בחיבור בין חלקי המערכת.
- שימוש בסביבות בדיקה מבודדות לפני העלאה לייצור.

מה בוצע בפועל:

- הטמעה מוקדמת של בדיקות אוטומטיות לזיהוי בעיות בשלב מוקדם.
- תיקון באגים משמעותיים שהתגלו במהלך בדיקות עומס.

<u>סיכון 3: סיכוני אבטחה</u>

מהות הסיכון :דליפת מידע אישי או פריצה לשרתים עלולות לפגוע באמון המשתמשים.

דרכים להתמודד:

- שימוש בפרוטוקולים מאובטחים (TLS/SSL) לכל התעבורה.
- אחסון סיסמאות במאגר הנתונים בצורה מוצפנת.(Hashing)
- ביצוע סריקות אבטחה תקופתיות וזיהוי פרצות.
- הגבלת הרשאות גישה לפי תפקידים.

מה בוצע בפועל:

- פריסת מנגנון הצפנת תעבורה מלא ובדיקת חדירות.(Penetration Testing)
- תיקון פרצות קטנות שהתגלו במהלך סריקות אבטחה ראשוניות.

תיאור תחום הידע

פירוט מעמיק של יכולות המערכת:

יכולות בצד שרת:

ניהול משתמשים

מהות :ניהול מלא של משתמשים במערכת, כולל רישום, התחברות, שינוי פרטים, ושחזור סיסמה.

אוסף יכולות נדרשות:

יצירת חשבון חדש עם בדיקות לתקינות המידע (אימייל ייחודי, שם משתמש וסיסמה), אימות חשבון על ידי שליחת קוד אימות לאימייל, התחברות באמצעות אימות שם משתמש וסיסמה, שינוי פרטי משתמש, כולל שם משתמש וסיסמה, מנגנון לאיפוס סיסמה עם שליחת קוד זמני למייל, מחיקת חשבון תוך מחיקת כל המידע המשויך אליו מהמערכת.

אובייקטים נחוצים:

ממשק משתמש ליצירת חשבונות ולניהול פרטי המשתמש, תקשורת בין הלקוח לשרת לפרוטוקולי התחברות ורישום, בסיס נתונים לאחסון פרטי המשתמשים, הצפנה לשמירה על פרטיות הסיסמאות והמידע האישי.

ניהול קבצים ותיקיות

מהות :מתן אפשרות למשתמשים לבצע פעולות מגוונות על קבצים ותיקיות המאוחסנים במערכת.

אוסף יכולות נדרשות:

העלאת קבצים תוך שימוש במנגנון נעילה למניעת גישה כפולה, הורדת קבצים בשלבים (chunks) או כקובץZIP , מחיקת קבצים ותיקיות, תוך וידוא הרשאות מתאימות, שינוי שמות של קבצים ותיקיות, יצירת תיקיות חדשות, שיתוף קבצים ותיקיות עם משתמשים אחרים עם הרשאות מותאמות אישית, שחזור קבצים שנמחקו למיקומם המקורי.

אובייקטים נחוצים:

ממשק משתמש לניהול קבצים ותיקיות, מנגנון נעילה למניעת התנגשות בגישה, בסיס נתונים לניהול מטה-נתונים של קבצים ותיקיות, מנגנון הרשאות מותאם אישית לכל משתמש.

הרשאות וגישה

מהות :שליטה מלאה על גישת משתמשים לקבצים ותיקיות במערכת.

אוסף יכולות נדרשות:

הגדרת הרשאות לכל קובץ ותיקייה (קריאה, כתיבה, מחיקה, שיתוף, הורדה), זיהוי משתמשים אורחים ומניעת פעולות לא מורשות, בקרת גישה מותאמת אישית לכל משתמש.

אובייקטים נחוצים:

מנגנון הרשאות מבוסס נתונים, ממשק לניהול הרשאות עבור המשתמשים.

ביצועים ואבטחה

מהות :הבטחת אבטחת המידע וניהול משאבים יעיל במערכת.

אוסף יכולות נדרשות:

הצפנת התקשורת בין השרת ללקוח באמצעות RSA ומפתח משותף, מנגנון נעילה למניעת גישה כפולה לקבצים, הגבלת מהירויות העלאה והורדה לפי סוג המנוי, טיפול בניהול עומסים ואופטימיזציה של ביצועים.

אובייקטים נחוצים:

מערכת הצפנה ופענוח, בסיס נתונים לניהול פרטי מנויים והגבלות, מנגנון לניהול משאבי מערכת.

ממשק פרוטוקולים

מהות :ניהול פרוטוקולים מותאמים אישית לביצוע פעולות בין הלקוח לשרת.

אוסף יכולות נדרשות:

קבלת בקשות מהלקוח לפי קודים מותאמים (LOGIN, SIGNUP, UPLOAD וכו'), ניתוח בקשות והעברתם למודול המתאים בשרת, החזרת תגובה מסודרת ללקוח.

אובייקטים נחוצים:

מערכת תקשורת בין לקוח לשרת, מודול לעיבוד בקשות.

ניהול מנויים

מהות :מתן שירותי אחסון מותאמים אישית לפי רמות מנוי שונות.

אוסף יכולות נדרשות:

הגדרת רמות מנוי (חינמי, בסיסי, פרימיום, מקצועי), ניהול מגבלות כמו אחסון ומהירות העלאה והורדה לפי רמת מנוי, שדרוג מנוי תוך שמירה על המידע הקיים.

אובייקטים נחוצים:

בסיס נתונים לניהול פרטי מנויים, ממשק לניהול ושדרוג מנויים.

תחזוקה וניהול מערכת

מהות :שמירה על פעילות תקינה של המערכת וניהול תקלות.

אוסף יכולות נדרשות:

רישום לוגים של כל הפעולות במערכת, ניטור תעבורה ושיפור ביצועים, מנגנון לשחזור אוטומטי של חיבורים שנקטעו.

אובייקטים נחוצים:

מערכת ניטור ותיעוד, מודולים לשחזור ושיפור ביצועים.

יכולות נוספות

מהות :תמיכה בפעולות מתקדמות להגדלת היציבות והאמינות של המערכת.

אוסף יכולות נדרשות:

שחזור נתונים שנמחקו בטעות, תיקון שגיאות ואופטימיזציה של תהליכים.

אובייקטים נחוצים:

בסיס נתונים לגיבוי נתונים, מודול לניהול ותיקון שגיאות.

יכולות בצד לקוח:

ניהול משתמשים

מהות :מתן אפשרות למשתמשים לנהל את חשבונותיהם ישירות מהאפליקציה.

אוסף יכולות נדרשות:

ממשק משתמש אינטואיטיבי לרישום, התחברות ושינוי פרטי חשבון, תהליך רישום עם קליטת פרטים ושליחתם לשרת, התחברות לחשבון קיים באמצעות אימייל וסיסמה, מנגנון לשחזור סיסמה הכולל שליחת קוד זמני לאימייל, הצגת הודעות למשתמש על הצלחה או כישלון של פעולות.

אובייקטים נחוצים:

מסכי רישום והתחברות, חיבור מאובטח לשרת לשליחת פרטי המשתמש, מודול לניהול הודעות משתמש.(success/error)

ניהול קבצים ותיקיות

מהות :מתן אפשרות למשתמשים לנהל קבצים ותיקיות באחסון הענן דרך ממשק גרפי.

אוסף יכולות נדרשות:

הצגת רשימת הקבצים והתיקיות של המשתמש, העלאת קבצים ותיקיות לאחסון הענן, הורדת קבצים למחשב המקומי, יצירת תיקיות חדשות ישירות מתוך האפליקציה, שיתוף קבצים ותיקיות עם משתמשים אחרים על פי הרשאות, שינוי שמות של קבצים ותיקיות, מחיקת קבצים ותיקיות והעברתם לאזור "אשפה."

אובייקטים נחוצים:

ממשק משתמש להצגת קבצים ותיקיות, חיבור לשרת לקבלת עדכונים בזמן אמת על תוכן המשתמש, מנגנון לניהול הרשאות שיתוף מתוך האפליקציה.

<u>הרשאות וגישה</u>

מהות :ניהול הרשאות שיתוף וגישה לקבצים ותיקיות.

אוסף יכולות נדרשות:

תפריט שיתוף קבצים עם הגדרת הרשאות (קריאה, כתיבה, מחיקה), הצגת מצב שיתוף לכל קובץ ותיקייה בממשק, שינוי הרשאות שיתוף בזמן אמת, מנגנון להסרת שיתופים קיימים.

אובייקטים נחוצים:

מסך לניהול הרשאות, חיבור לשרת לעדכון הרשאות ותוכן.

<u>ביצועים ואבטחה</u>

מהות :שמירה על תקשורת מאובטחת עם השרת וניהול פעולות באופן יעיל.

אוסף יכולות נדרשות:

חיבור מוצפן לשרת לכל פעולה שמתבצעת דרך האפליקציה, ניהול זיכרון וטעינת תוכן בצורה אופטימלית, טיפול במקרים של חיבור איטי או ניתוק פתאומי, הצגת הודעות למשתמש על מצב התקשורת עם השרת.

אובייקטים נחוצים:

מערכת הצפנה ופענוח נתונים, מודול לניהול חיבורים ותקלות.

<u>ממשק פרוטוקולים</u>

מהות :ניהול הבקשות והתשובות בין הלקוח לשרת בצורה מסודרת.

אוסף יכולות נדרשות:

שליחת בקשות לשרת בפורמט אחיד ומאובטח, קבלת תשובות מהשרת וניתוחן, טיפול במקרי שגיאה והצגתן למשתמש בצורה ברורה.

אובייקטים נחוצים:

מודול לשליחת בקשות וקבלת תשובות, ממשק לטיפול בשגיאות ותקלות.

<u>ניהול מנויים</u>

מהות :הצגת פרטי המנוי של המשתמש ואפשרות לשדרוג.

אוסף יכולות נדרשות:

הצגת פרטי המנוי הנוכחי (אחסון זמין, מהירות העלאה והורדה), מתן אפשרות לשדרוג מנוי מתוך האפליקציה, ניהול תשלומים ומעקב אחר תקופת המנוי.

אובייקטים נחוצים:

מסך פרטי מנוי ושדרוג, חיבור לשרת לעדכון סטטוס המנוי, מודול לניהול תשלומים.

<u>תחזוקה וניהול מערכת</u>

מהות :שמירה על תקינות האפליקציה בצד הלקוח וניהול תקלות.

אוסף יכולות נדרשות:

טיפול במקרי קריסה ושמירה על יציבות האפליקציה, עדכון גרסאות האפליקציה באופן אוטומטי, מנגנון לדיווח על תקלות והצעת פתרונות למשתמש.

אובייקטים נחוצים:

מודול לניהול עדכונים, ממשק לדיווח תקלות, חיבור לשרת לקבלת מידע על תקלות ועדכונים.

מבנה/ארכיטקטורה של המערכת

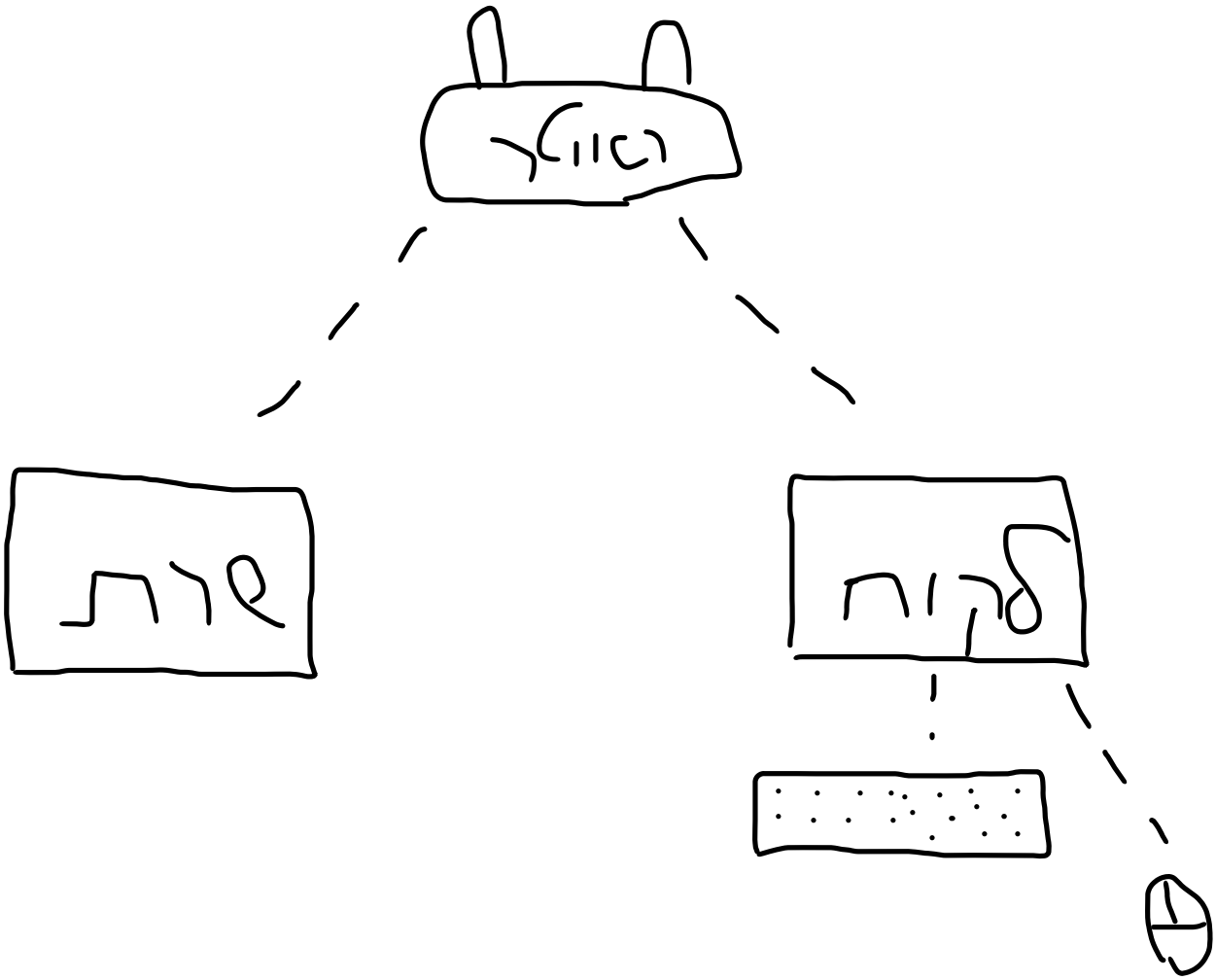
תיאור הארכיטקטורה של המערכת:

תיאור החומרה:

החומרה במערכת שלי מורכבת בבסיס משרת שרץ על מחשב ווינדוס כלשהו ומאחסן את הקבצים שמעלים אליו המשתמשים, כלומר השרת דורש מחשב מתפקד עם זיכרון, בנוסף אם נדרשת התחברות של לקוחות מרשת אחרת, יש לאפשר Port Forwarding במחשב השרת.

הלקוח גם הוא ירוץ על מחשב ווינדוס, אך בניגוד לשרת הלקוח הוא בעל ממשק משתמש גרפי, כלומר בכדי להשתמש במערכת, הלקוח יצטרך גם מקלדת ועכבר בשביל לנוע ולהשתמש בממשק. הלקוח לא צריך הרשאות מיוחדות בכדי להתחבר לשרת, גם כזה שלא נמצא באותה רשת פנימית.

שרטוט:



טכנולוגיה רלוונטית:

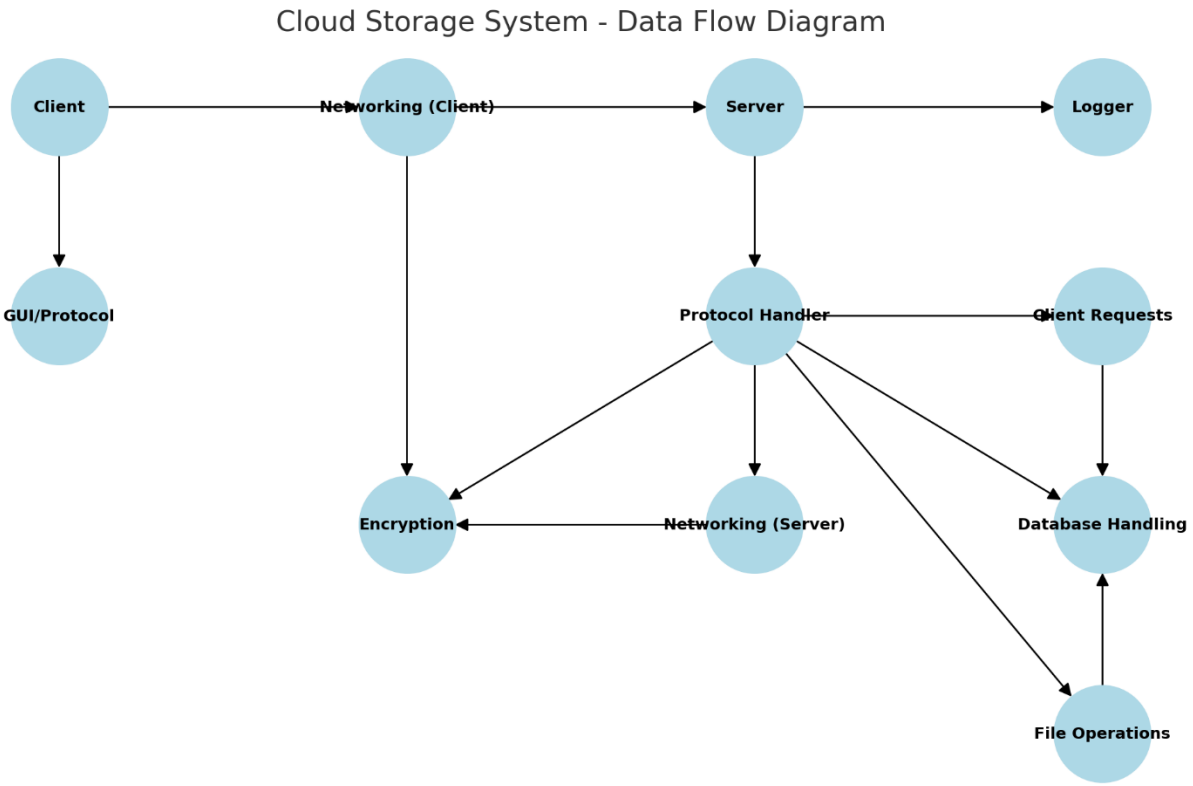
המערכת שלי בנויה בשפת התכנות פיתון, בחרתי לתכנת בפיתון מכמה סיבות. סיבה ראשונה היא שאני שולט בפיתון ומכיר את השפה טוב כך שאני יכול לבנות פרויקט ברמה גבוהה ולהצליח לפתור בעיות בקלות. בנוסף בפיתון יש המון ספריות שימושיות שעוזרות עם בניית הפרויקט. השתמשתי בספרייה PYQT6 לממשק המשתמש שלי מכיוון שהיא נוחה לשימוש, יש לה תוכנת עיצוב דפים (Designer) ואפילו ניתן להוסיף עיצוב עם CSS.

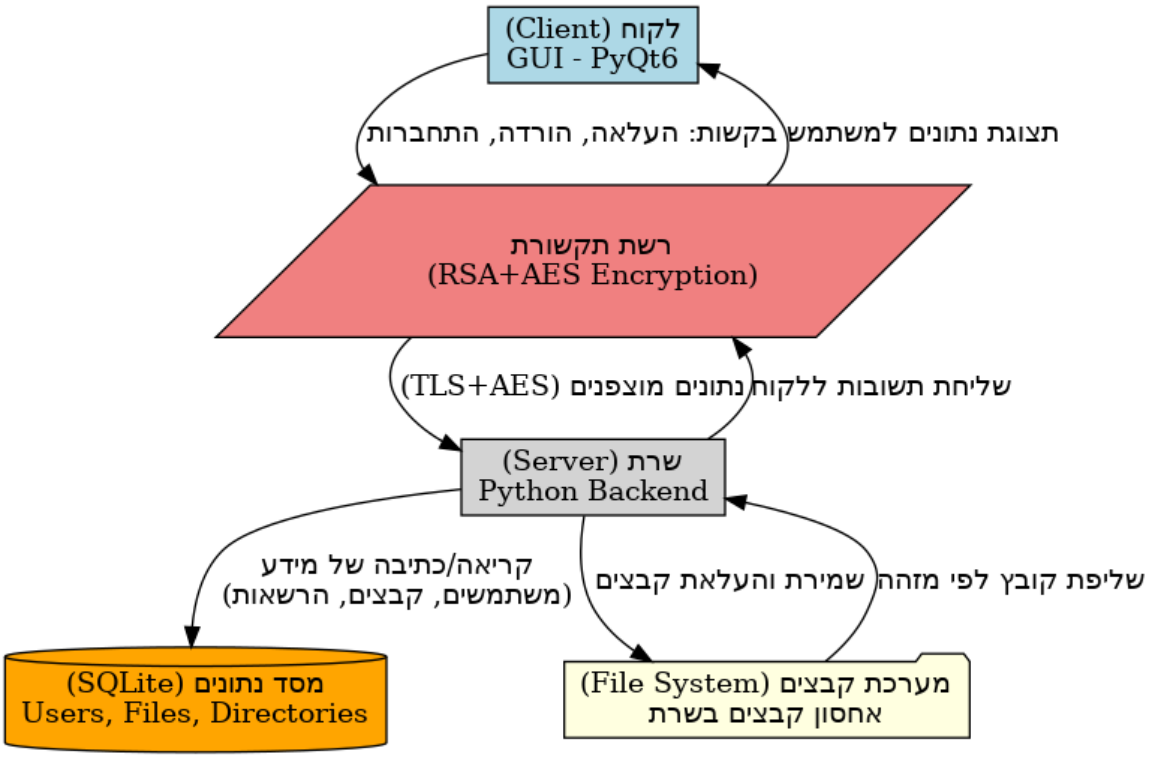
המערכת שלי רצה על מערכת ההפעלה ווינדוס. בחרתי לכתוב את המערכת כך שתתאים ווינדוס מכיוון שזוהי מערכת ההפעלה הנפוצה ביותר בעולם וכך תהיה גישה לקהל רחב יותר של משתמשים. בנוסף לכך, ישנם אמולטורים במערכות הפעלה אחרות כגון לינוקס ומאק אואס, שבעזרתם ניתן להריץ תוכנות ווינדוס כך שגם משתמש שאינו ווינדוס יוכל להשתמש במערכת.

בשביל התקשורת השתמשתי בתקשורת סוקטים אסינכרונית. כלומר השרת מפעיל סוקט שמאזין לבקשות, בכל פעם שמגיעה בקשה הוא מעבד אותה ומחזיר תשובה למקור. הלקוח עובד בצורה דומה, הוא מאזין כל הזמן לתשובות, וכאשר המשתמש מבקש, הוא שולח בקשות לשרת. התקשורת האסינכרונית תורמת לכך שבזמן שהלקוח מעלה קובץ גדול, הממשק לא יתקע והוא יוכל להמשיך לבקש דברים מהשרת, הוא לא יתקע עד לתשובה מהשרת.

אני בחרתי בפרויקט הזה גם מתוך תחומי העניין האישיים שלי. אני חושב שמערכת ענן היא דבר שימושי מאוד ומגניב, וביצירת מערכת כזו בעצמי אני יכול להוסיף פיצ'רים ואופציות מיוחדות שנראות לי מתאימות שלא בהכרח יש במערכות אחרות. בנוסף אני חושב שזה גם מאוד שימושי לשימוש האישי שלי, מערכת אחסון טובה ומהירה שלי אין בה הגבלה.

זרימת המידע במערכת:





תיאור האלגוריתמיקה בפרויקט:

ניסוח וניתוח של הבעיה האלגוריתמית:

תיאור אלגוריתמיקה של הפרויקט

ניסוח הבעיה האלגוריתמית

מערכת האחסון בענן עוסקת בניהול נתונים בצורה מאובטחת, יעילה ומבוזרת. הבעיה המרכזית שיש לפתור היא כיצד לספק למשתמשים גישה נוחה ושקופה לקבצים ותיקיות תוך הבטחת אבטחת מידע, ביצועים גבוהים ותמיכה בריבוי משתמשים בו זמנית. הבעיה מתפרקת למספר תתי-בעיות אלגוריתמיות עיקריות:

1. **ניהול משתמשים והרשאות** :כיצד לאמת משתמשים ולנהל את הרשאותיהם באופן דינמי.
2. **אחסון נתונים מבוזר ומאובטח** :כיצד לאחסן נתונים בצורה שמבטיחה גם גישה מהירה וגם הצפנה.
3. **תמיכה בעומסים גבוהים** :כיצד לנהל העלאות והורדות בו-זמנית עבור מספר רב של משתמשים.
4. **שיתוף קבצים** :כיצד ליישם מנגנון גמיש לשיתוף קבצים בין משתמשים עם בקרת הרשאות ברזולוציה גבוהה.

ניתוח הבעיה האלגוריתמית

1. ניהול משתמשים והרשאות

- **בעיה** :אימות זהות המשתמשים ושמירת נתונים רגישים כגון סיסמאות באופן מאובטח.
- **פתרון אלגוריתמי** :שימוש בהצפנה עם bcrypt לשמירת סיסמאות והצפנת תקשורת באמצעות RSAלהחלפת מפתחות ו AES-לתעבורת נתונים.

2. אחסון נתונים מבוזר ומאובטח

- **בעיה** :שמירה על זמינות גבוהה של נתונים תוך שמירה על פרטיות.
- **פתרון אלגוריתמי** :שימוש בבסיס נתונים מבוסס sqlite3 לניהול מטה-נתונים, ואחסון פיזי של קבצים במערכת קבצים עם מנגנוני גיבוי והצפנה.

3. תמיכה בעומסים גבוהים

- **בעיה** :טיפול בו זמנית בבקשות העלאה והורדה של משתמשים רבים.
- **פתרון אלגוריתמי** :יישום תורים אסינכרוניים (Asynchronous Queues) לניהול משימות ברקע, חלוקת עומסים דינמית (Load Balancing) וחיתוך קבצים לחתיכות קטנות (Chunks) כדי לשפר את ביצועי ההעברה.

4. שיתוף קבצים

- **בעיה** :ניהול שיתופים דינמיים עם הרשאות שונות (קריאה, כתיבה, מחיקה וכו').
- **פתרון אלגוריתמי** :ניהול טבלת הרשאות בבסיס הנתונים, ושימוש במנגנון היררכי לבדיקה האם למשתמש יש גישה ישירה או עקיפה (לדוגמה, דרך שיתוף).

דרישות ביצועים

- **אבטחת מידע** :זמן חישוב הצפנה/פענוח קצר, תוך עמידה בתקני אבטחה מודרניים.
- **סקלאביליות** :המערכת צריכה לתמוך באלפי משתמשים פעילים בו זמנית עם גישה מהירה לנתונים.
- **זמינות גבוהה** :המערכת צריכה להבטיח מינימום השבתות, עם מנגנוני התאוששות מהירה מתקלות.

<u>אלגוריתמים קיימים:</u>

1. הצפנה ואימות משתמשים

- **bcrypt**:אלגוריתם להצפנת סיסמאות על ידי שימוש במנגנון Hashing משופר המוסיף Salt לכל סיסמה כדי למנוע התקפות מילון.
- RSA: **RSA + AES**משמש להחלפת מפתחות בצורה מאובטחת, ולאחר מכן AES משמש להצפנה סימטרית של התקשורת בין הלקוח לשרת.

2. אחסון נתונים

- **SQLite**:מנוע בסיס נתונים קל משקל המאפשר ניהול מטה-נתונים של קבצים ותיקיות בצורה מאובטחת ויעילה.

- **מערכת קבצים מבוזרת** :שימוש במנגנונים לניהול הקבצים הפיזיים כולל גיבוי, אחסון מוצפן וניהול גרסאות.

3. טיפול בעומסים גבוהים

- **Asynchronous Task Queues**: ניהול משימות ברקע בצורה אסינכרונית עם תורים כדי למנוע עומסים על השרת.
- **Load Balancing**: חלוקת עומסים בין משאבים שונים במערכת כדי לשפר את הביצועים.

4. מנגנוני הרשאות ושיתוף

- **RBAC (Role-Based Access Control)**: ניהול הרשאות על בסיס תפקידים המאפשר גמישות בניהול גישה לקבצים ותיקיות.
- **טבלאות הרשאות בבסיס הנתונים** :כל קובץ או תיקייה מקושרים לרשימת הרשאות מותאמות אישית לכל משתמש.

<u>סקירת הפתרון הנבחר:</u>

הפתרון הנבחר משלב מספר אלגוריתמים קיימים לצד התאמות ייחודיות שנעשו לצורך מימוש המערכת.

נימוק הבחירה בפתרון

1. **שימוש באלגוריתמים מוכרים ומאובטחים:**
   - bcrypt להצפנת סיסמאות מספק שכבת אבטחה חזקה נגד התקפות brute-force.
   - שילוב של RSA ו AES- מאפשר תקשורת מוצפנת מאובטחת מבלי לפגוע בביצועים.
2. **שימוש במנגנון מבוזר לניהול עומסים:**
   - חלוקת הנתונים למקטעים קטנים (Chunks) מאפשרת עבודה במקביל ומשפרת ביצועים בעת העלאה והורדה.
   - Load Balancing מונע עומסים על השרת ומבטיח חוויית משתמש חלקה.
3. **ניהול הרשאות גמיש ויעיל:**

- RBAC (Role-Based Access Control) מאפשר קביעת הרשאות מותאמות אישית לכל משתמש.
- ניהול טבלאות הרשאות בבסיס הנתונים מספק פתרון גמיש לניהול שיתופי קבצים.

שלילת פתרונות אלטרנטיביים

1. שימוש בגיבוי קבצים בלבד ללא בסיס נתונים :פתרון זה היה מגביל את יכולת ניהול המטה-נתונים של הקבצים ולא מאפשר ניהול הרשאות מתקדם.
2. שימוש בהצפנה סימטרית בלבד :פתרון זה היה דורש העברת מפתחות פרטיים בין הלקוח לשרת, דבר שהיה מסכן את אבטחת המידע.
3. ניהול תהליכים סינכרוניים בלבד :היה גורם לעיכובים משמעותיים בעת העלאה והורדה של קבצים ולא מאפשר ביצועים גבוהים.

דרישות ביצועים

- אבטחת מידע :זמן חישוב הצפנה/פענוח קצר, תוך עמידה בתקני אבטחה מודרניים.
- סקלאביליות :המערכת צריכה לתמוך באלפי משתמשים פעילים בו זמנית עם גישה מהירה לנתונים.
- זמינות גבוהה :המערכת צריכה להבטיח מינימום השבתות, עם מנגנוני התאוששות מהירה מתקלות.

תיאור סביבת הפיתוח:

כלי הפיתוח הנדרשים:

הפרויקט נבנה בשפת Python, המספקת גמישות רבה ונוחות פיתוח, יחד עם ספריות מתקדמות לאבטחת מידע, ניהול תקשורת ועבודה עם ממשקי משתמש. לצורך פיתוח וניהול הקוד, נעשה שימוש בסביבת העבודה Visual Studio Code, המאפשרת עריכה נוחה, אינטגרציה עם בקרת גרסאות, ותמיכה בתוספים שימושיים לשיפור חוויית הפיתוח. Visual Studio Code מציע תמיכה רחבה בכתיבת קוד Python עם כלי בדיקה, ניפוי שגיאות, ותוספים לניתוח ביצועים. בנוסף, פלטפורמת הפיתוח כוללת את ספריות PyQt6 ליצירת ממשקי משתמש, SQLite, לניהול בסיס הנתונים, ו bcrypt-ו RSA-לאבטחת מידע. בחירת הכלים הללו נבעה מהצורך במערכת אמינה, מאובטחת וגמישה המסוגלת להתמודד עם דרישות מורכבות כמו הרשאות, שיתוף קבצים ותעבורה מוצפנת.

הסביבה והכלים הנדרשים לבדיקות:

לצורך בדיקות המערכת, נעשה שימוש בקובץ **log** לתיעוד פעולות קריטיות, איתור בעיות וניטור תקלות במהלך הריצה. כמו כן, שולבו **נקודות עצירה (breakpoints)** במהלך הפיתוח כדי לאפשר ניפוי שגיאות ידני ובדיקת זרימת הקוד בזמן אמת. הבדיקות בוצעו באמצעות בדיקות ידניות לכל אחת מהפונקציות המרכזיות, כולל אימות משתמשים, העלאת והורדת קבצים, ניהול הרשאות ושיתוף קבצים, כדי לוודא שכל רכיב פועל כמצופה.

בנוסף, נעשה שימוש ב **GitHub**-לניהול גרסאות, שמירת שינויים ובידוד משתנים חשובים, מה שמאפשר חזרה לגרסאות קודמות במקרה של תקלה. כך ניתן היה לבדוק כל שינוי באופן מבוקר לפני שילובו בקוד המרכזי.

פרוטוקול התקשורת:

תיאור מילולי של הפרוטוקול:

מבנה הפרוטוקול במערכת מבוסס על מבנה נתונים אחיד וקבוע, המאפשר תקשורת אמינה בין הלקוח לשרת. כל הודעה מורכבת ממספר חלקים עיקריים, המסודרים בפורמט קבוע, על מנת להבטיח עיבוד יעיל וקריא של הבקשות והתגובות.

הפרוטוקול בנוי כך שה**4 הבתים הראשונים** מוקדשים לאורך ההודעה, ומיוצגים כנתונים בינאריים. הדבר מאפשר לשרת וללקוח לדעת מראש כמה נתונים עליהם לקרוא מהזרם, ובכך למנוע קריאות חסרות או קריאות עודפות של מידע.

לאחר מכן, מופיע **קוד ההודעה** ,אשר מורכב מ **4-תווים גדולים באנגלית**)לדוגמה ,"LOGN" : "UPLD" .("DELT" ,קוד זה מאפשר לשרת לזהות במהירות את סוג הבקשה, ומקל על ניתוח ההודעה.

לאחר קוד ההודעה, מגיעים **השדות השונים של ההודעה** ,כאשר לכל סוג הודעה יש כמות שונה של שדות בהתאם לצרכיה. השדות מופרדים זה מזה באמצעות התו | ,המאפשר פענוח פשוט ומהיר. לדוגמה, הודעת התחברות ("LOGN") עשויה לכלול את שם המשתמש והסיסמה כשדות, ואילו הודעת העלאת קובץ ("UPLD") תכלול שם קובץ, גודל, ונתוני ההצפנה. בכל הודעה קודם כל מופיעים שדות החובה ולאחר מכן שדות הרשות.

מבנה זה מבטיח שהשרת והלקוח מסוגלים לתקשר בצורה עקבית ויעילה, תוך שמירה על גמישות בהוספת פקודות חדשות בעתיד.

פירוט כלל ההודעות הזורמות במערכת:

שם ההודעה	קוד ההודעה	כיוון	שדות חובה	שדות רשות
בקשת יציאה	EXIT	לקוח → שרת	-	-
אישור יציאה	EXTR	שרת → לקוח	-	-

בקשת התחברות	LOGN	לקוח → שרת	שם משתמש/אימייל, סיסמה	-
אישור התחברות	LOGS	שרת → לקוח	אימייל, שם משתמש, רמת מנוי	-
בקשת הרשמה	SIGU	לקוח → שרת	אימייל, שם משתמש, סיסמה, אימות סיסמה	-
אישור הרשמה	SIGS	שרת → לקוח	אימייל, שם משתמש, סיסמה	-
בקשת שליחת קוד שחזור סיסמה	FOPS	לקוח → שרת	אימייל	-
אישור שליחת קוד שחזור סיסמה	FOPR	שרת → לקוח	אימייל	-
בקשת שחזור סיסמה	PASR	לקוח → שרת	אימייל, קוד שחזור, סיסמה חדשה, אימות סיסמה חדשה	-
אישור שחזור סיסמה	PASS	שרת → לקוח	אימייל, סיסמה חדשה	-
בקשת יציאה	LOGU	לקוח → שרת	-	-
אישור יציאה	LUGR	שרת → לקוח	-	-
בקשת שליחת קוד אימות לחשבון	SVER	לקוח → שרת	אימייל	-

אישור שליחת קוד אימות לחשבון	VERS	שרת → לקוח	אימייל	-
בקשת אימות חשבון	VERC	לקוח → שרת	אימייל, קוד אימות	-
אישור אימות חשבון	VERR	שרת → לקוח	אימייל	-
בקשת מחיקת חשבון	DELU	לקוח → שרת	אימייל	-
אישור מחיקת חשבון	DELR	שרת → לקוח	אימייל	-
בקשת שינוי שם משתמש	CHUN	לקוח → שרת	שם משתמש חדש	-
אישור שינוי שם משתמש	CHUR	שרת → לקוח	שם משתמש חדש	-
בקשת נתוני קבצים בתיקייה	GETP	לקוח → שרת	מזהה תיקייה, כמות, סוג מיון, כיוון מיון	חיפוש
נתוני קבצים בתיקייה	PATH	שרת → לקוח	כמות כוללת, רשימת קבצים	-
בקשת נתוני תיקיות בתיקייה	GETD	לקוח → שרת	מזהה תיקייה, כמות, סוג מיון, כיוון מיון	חיפוש
נתוני תיקיות בתיקייה	PATD	שרת → לקוח	כמות כוללת, רשימת תיקיות	-

בקשת העלאת קובץ	FILS	לקוח → שרת	שם קובץ, תיקייה הורה, גודל קובץ, מזהה	-
אישור התחלת העלאת קובץ	FISS	שרת → לקוח	שם קובץ	-
שליחת נתוני קובץ (חלקי)	FILD	לקוח → שרת	מזהה קובץ, מיקום, נתונים	-
שליחת נתוני קובץ (אחרון)	FILE	לקוח → שרת	מזהה קובץ, מיקום, נתונים	-
אישור סיום העלאת קובץ	FILR	שרת → לקוח	שם קובץ	-
בקשת הורדת קובץ	DOWN	לקוח → שרת	מזהה קובץ	-
אישור הורדת קובץ	DOWR	שרת → לקוח	שם קובץ	-
שליחת נתוני קובץ שהורד	RILD	שרת → לקוח	מזהה קובץ, מיקום, נתונים	-
שליחת נתוני קובץ שהורד (אחרון)	RILE	שרת → לקוח	מזהה קובץ, מיקום, נתונים	-
בקשת יצירת תיקייה	NEWF	לקוח → שרת	שם תיקייה	-
אישור יצירת תיקייה	NEFR	שרת → לקוח	שם תיקייה	-

בקשת שינוי שם קובץ/תיקייה	RENA	לקוח → שרת	מזהה קובץ, שם חדש	-
אישור שינוי שם קובץ/תיקייה	RENR	שרת → לקוח	שם קובץ ישן, שם חדש	-
בקשת מחיקת קובץ/תיקייה	DELF	לקוח → שרת	מזהה קובץ/תיקייה	-
אישור מחיקת קובץ	DLFR	שרת → לקוח	שם קובץ	-
אישור מחיקת תיקייה	DFFR	שרת → לקוח	שם תיקייה	-
בקשת שיתוף קובץ	SHRS	לקוח → שרת	מזהה קובץ, שם משתמש	-
אישור שיתוף קובץ	SHRR	שרת → לקוח	מזהה קובץ, שם משתמש	הרשאות
בקשת עדכון הרשאות שיתוף	SHRP	לקוח → שרת	מזהה קובץ, שם משתמש, הרשאות	-
בקשת הסרת שיתוף	SHRE	לקוח → שרת	מזהה קובץ	-
אישור הסרת שיתוף	SHRM	שרת → לקוח	שם קובץ	-

מסכי המערכת:

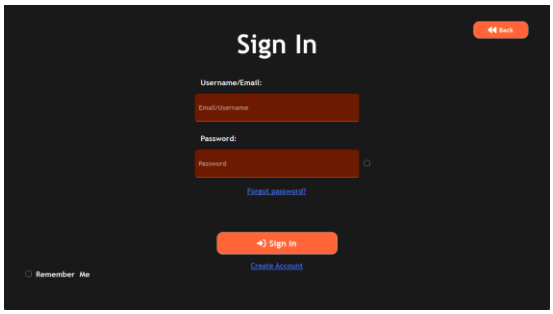
תיאור כל מסך:

מסך ראשי (Main Page)

המסך הראשי של האפליקציה שממנו המשתמש מתחיל את השימוש. במסך זה מופיע שם האפליקציה או הלוגו שלה, יחד עם שלושה כפתורים מרכזיים: התחברות, הרשמה ויציאה. המשתמש יכול לבחור אם להתחבר לחשבון קיים, ליצור חשבון חדש או לצאת מהאפליקציה. זהו המסך הראשוני שמוביל לכל הפעולות המרכזיות במערכת.

מסך התחברות (Login Page)

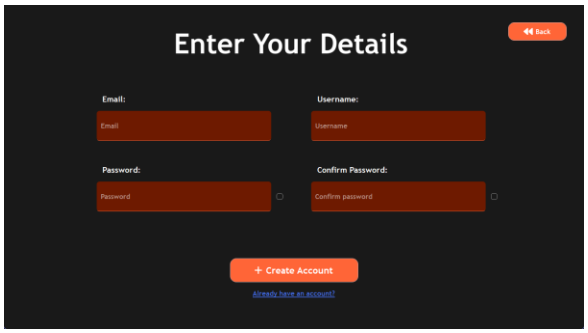
במסך זה המשתמש יכול להזין שם משתמש או אימייל וסיסמה כדי להתחבר למערכת. המסך כולל גם אפשרות לסימון "זכור אותי" כדי שהמשתמש לא יצטרך להזין מחדש את פרטיו בפעם הבאה. אם המשתמש שכח את הסיסמה, הוא יכול ללחוץ על כפתור לשחזור סיסמה. בנוסף, קיימת אפשרות למעבר מהיר למסך ההרשמה עבור משתמשים חדשים.



מסך הרשמה (Signup Page)

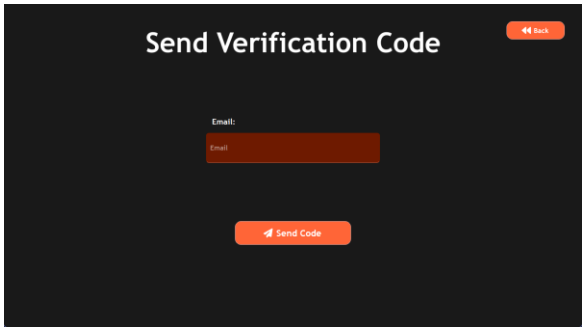
מסך זה מיועד למשתמשים חדשים שרוצים ליצור חשבון. המשתמש מתבקש להזין כתובת אימייל, שם משתמש, סיסמה ואימות סיסמה כדי לוודא שאין טעויות בהקלדה. אם למשתמש כבר יש חשבון, הוא

יכול לעבור ישירות למסך ההתחברות. לאחר מילוי הפרטים ולחיצה על כפתור ההרשמה, המשתמש יקבל קוד אימות למייל כדי לוודא שהכתובת תקינה.



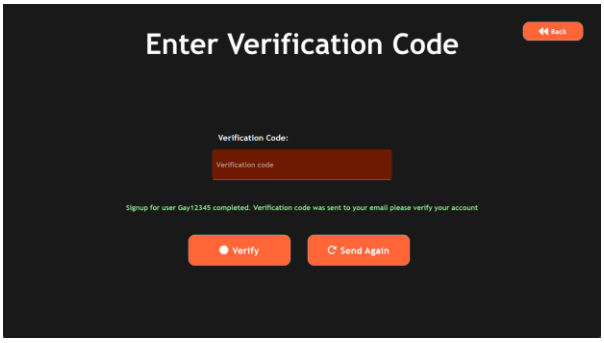
מסך שליחת קוד אימות (Send Verification Page)

כאשר משתמש חדש נרשם אך לא קיבל את קוד האימות למייל, הוא יכול להיכנס למסך זה ולהזין את כתובת האימייל שלו. לאחר מכן, המערכת תשלח שוב את קוד האימות כדי שהוא יוכל להשלים את תהליך ההרשמה.



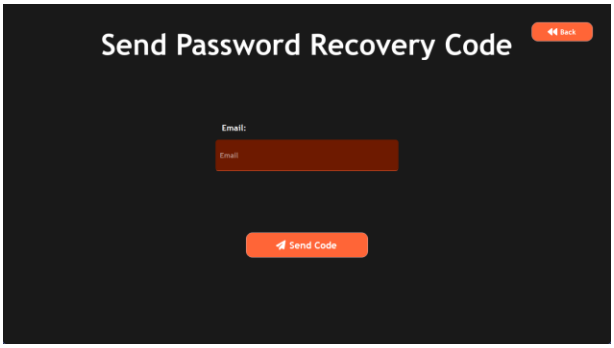
מסך אימות חשבון (Verification Page)

לאחר שהמשתמש קיבל את קוד האימות שנשלח למייל, הוא יכול להזין אותו במסך זה כדי לאמת את חשבונו. רק לאחר אימות מוצלח הוא יוכל להשתמש בכל האפשרויות של המערכת, כגון העלאת קבצים ושיתוף עם אחרים.



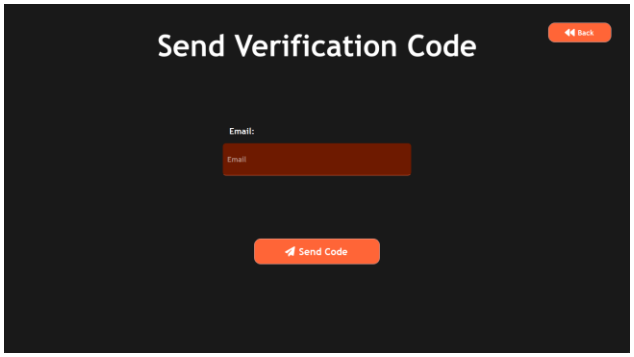
מסך איפוס סיסמה (Forgot Password Page)

במקרה שהמשתמש שכח את הסיסמה שלו, הוא יכול להשתמש במסך זה כדי לבקש קוד שחזור. עליו להזין את כתובת האימייל המקושרת לחשבון שלו, ולאחר מכן המערכת תשלח לו קוד שחזור סיסמה שיאפשר לו להגדיר סיסמה חדשה.



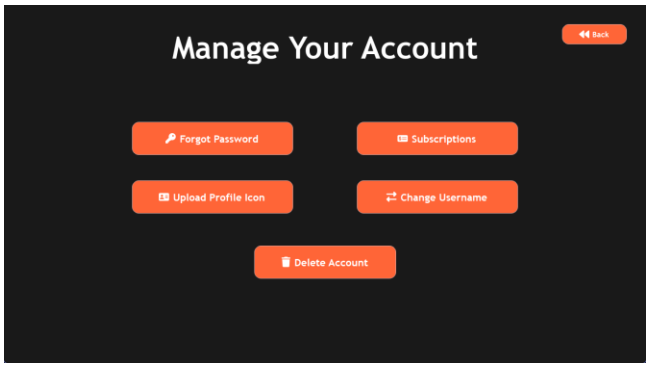
מסך הזנת קוד שחזור סיסמה (Recovery Page)

בשלב זה, לאחר שהמשתמש קיבל קוד שחזור למייל, הוא יכול להזין אותו במסך זה יחד עם סיסמה חדשה ואימות סיסמה. לאחר ההזנה המוצלחת, חשבונו יעודכן עם הסיסמה החדשה והוא יוכל להתחבר שוב.



מסך ניהול חשבון (Manage Account Page)

מסך זה מספק למשתמש אפשרויות לניהול חשבונו. הוא יכול לשנות את שם המשתמש שלו, להעלות תמונת פרופיל חדשה, לבקש איפוס סיסמה אם שכח אותה, או אף למחוק את החשבון לחלוטין אם הוא מחליט לעזוב את השירות.

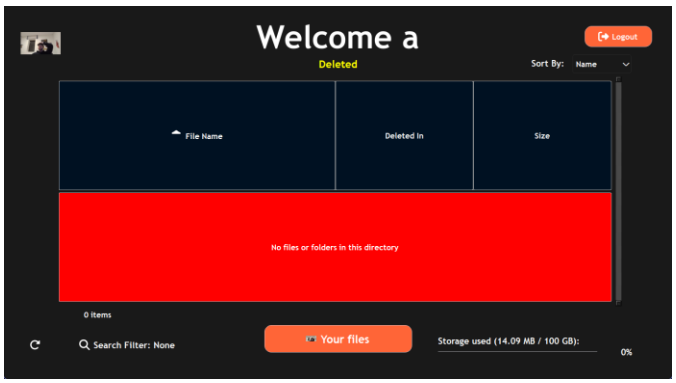


מסך קבצים (User Page)

זהו המסך המרכזי שבו המשתמש יכול לנהל את הקבצים והתיקיות שלו בענן. הוא יכול להעלות קבצים חדשים, להוריד קבצים קיימים, לשנות את שמם, למחוק אותם, או לשתף אותם עם משתמשים אחרים. בנוסף, מוצג מידע על נפח האחסון שנמצא בשימוש מתוך המקסימום המותר לו לפי המנוי שלו.

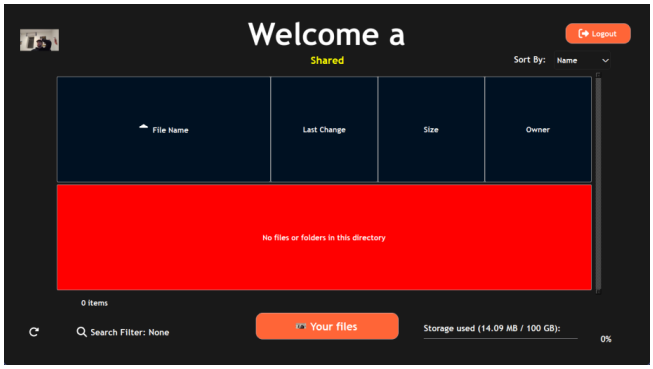
מסך קבצים שנמחקו (Deleted Files Page)

במסך זה המשתמש יכול לראות את כל הקבצים והתיקיות שמחק לאחרונה. כל קובץ שנמחק מועבר למסך זה לפרק זמן מוגבל, שבמהלכו ניתן לשחזר אותו לתיקייה המקורית. אם המשתמש רוצה למחוק את הקובץ לצמיתות, הוא יכול לעשות זאת מכאן.



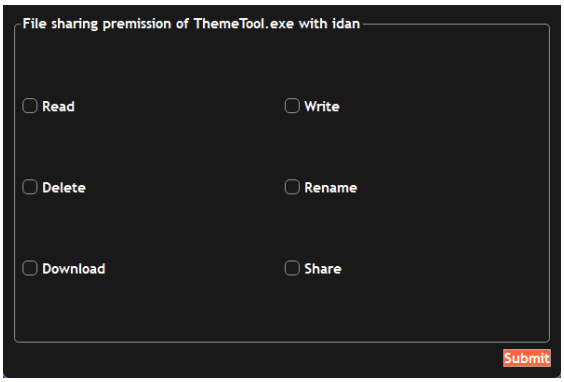
מסך קבצים משותפים (Shared Files Page)

מסך זה מציג את הקבצים שהמשתמש שיתף עם אחרים, כמו גם את הקבצים שאחרים שיתפו איתו. מכאן ניתן לנהל את השיתופים, להסיר גישה של משתמשים מסוימים או להוסיף הרשאות חדשות למשתמשים אחרים.



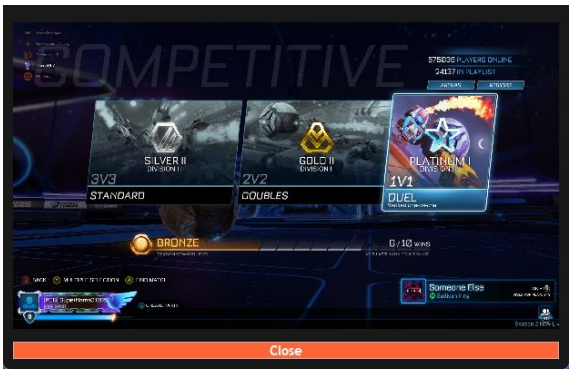
מסך ניהול שיתוף קבצים (File Sharing Options Page)

כאשר משתמש רוצה לשתף קובץ עם מישהו אחר, הוא יכול להיכנס למסך זה ולבחור אילו הרשאות להעניק. ניתן לקבוע האם המשתמש האחר יוכל רק לצפות בקובץ, לערוך אותו, למחוק אותו, לשנות את שמו, להוריד אותו או לשתף אותו מחדש עם אחרים.



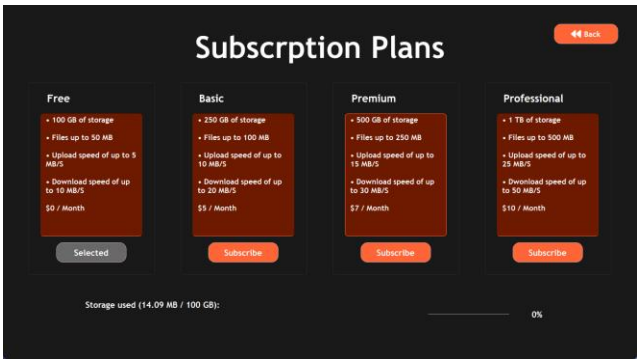
מסך צפייה בקובץ (File Viewer Page)

כאשר המשתמש פותח קובץ מסוים, הוא יכול לצפות בו ישירות בתוך האפליקציה. אם מדובר בקובץ טקסט, תמונה או מסמך מסוג מסוים, התוכן יוצג בתוך האפליקציה. ניתן גם לערוך קובצי טקסט ישירות מתוך המסך הזה.



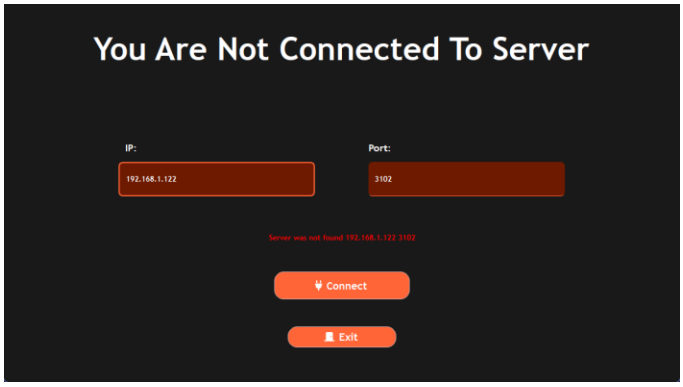
מסך שדרוג מנוי (Subscription Page)

במסך זה המשתמש יכול לבחור את חבילת האחסון המתאימה לו. כל חבילה מציעה נפח אחסון שונה ומהירויות שונות להעלאה והורדה. המשתמש יכול לעבור מחבילה חינמית לחבילה בתשלום לפי צרכיו.



מסך ניתוק מהשרת (Not Connected Page)

כאשר אין חיבור פעיל לשרת, מוצג המסך הזה. המשתמש יכול להזין כתובת IP ופורט חדשים ולנסות להתחבר מחדש.



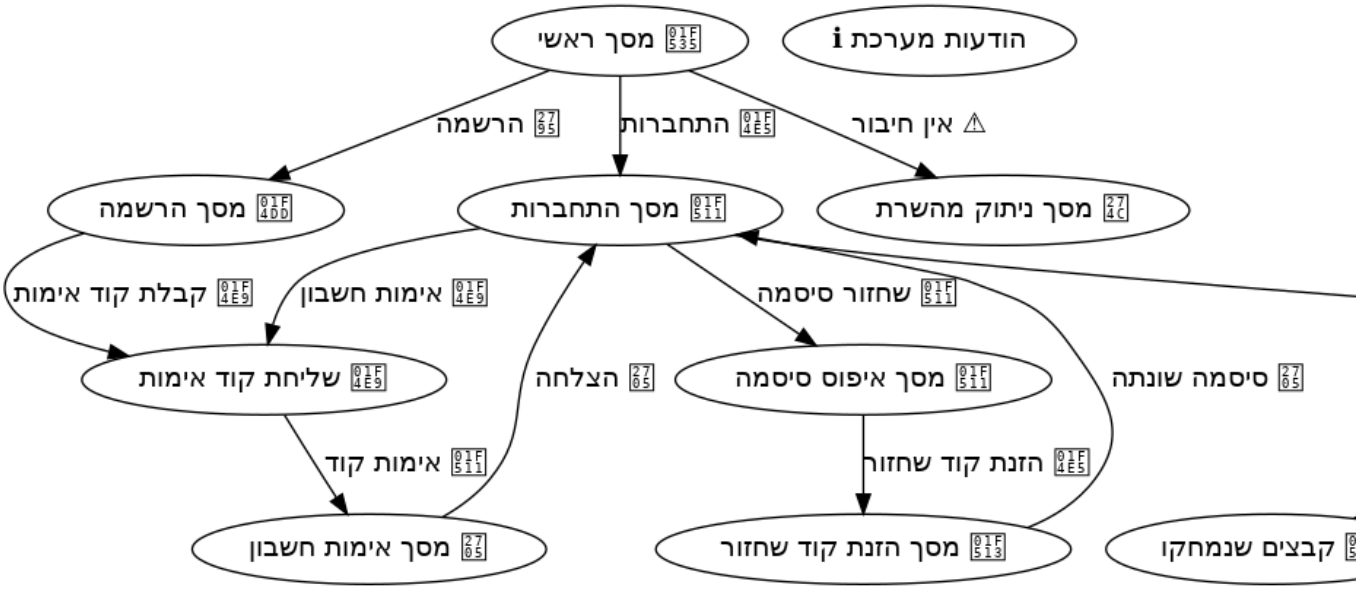
מסך שינוי שם קובץ (Rename File Dialog)

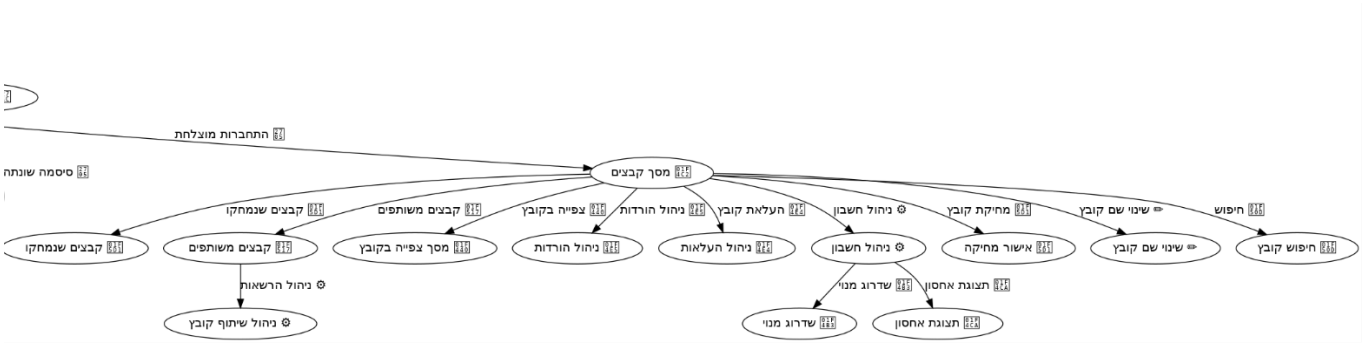
אם המשתמש רוצה לשנות את שמו של קובץ או תיקייה, הוא יכול להזין שם חדש במסך זה ולשלוח בקשה לשינוי שם.

מסך חיפוש קבצים (Search File Dialog)

במסך זה המשתמש יכול להקליד מילות חיפוש כדי למצוא קבצים או תיקיות בתוך המערכת.

תרשים מסכים:


מסך ראשי
הודעות מערכת
הרשמה
התחברות
אין חיבור
מסך הרשמה
מסך התחברות
מסך ניתוק מהשרת
קבלת קוד אימות
אימות חשבון
שחזור סיסמה
שליחת קוד אימות
הצלחה
מסך איפוס סיסמה
סיסמה שונתה
אימות קוד
הזנת קוד שחזור
מסך אימות חשבון
מסך הזנת קוד שחזור
קבצים שנמחקו


התחברות מוצלחת
סיסמה שונתה
מסך קבצים
קבצים שנמחקו
קבצים משותפים
מסך צפייה בקובץ
צפייה בקובץ
ניהול הורדות
העלאת קובץ
ניהול חשבון
מחיקת קובץ
שינוי שם קובץ
חיפוש
קבצים שנמחקו
קבצים משותפים
מסך צפייה בקובץ
ניהול הורדות
ניהול העלאות
ניהול חשבון
אישור מחיקה
שינוי שם קובץ
חיפוש קובץ
ניהול הרשאות
ניהול שיתוף קובץ
שדרוג מנוי
תצוגת אחסון
שדרוג מנוי
תצוגת אחסון

מבני הנתונים:

פירוט מבני הנתונים:

המערכת מבוססת על **ארבעה סוגי מבני נתונים עיקריים**:

מסד נתונים – **(SQLite)** מנוהל בקובץ database.db

מערכת קבצים – **(File System)** אחסון קבצים בשרת בהתאם למשתמשים ולתיקיות.

מבני נתונים בזיכרון – **(Data Structures in Memory)** מנוהלים בשרת לצורך שיפור ביצועים, חיבורי משתמשים ופעולות דינמיות.

מאגר נתונים מקומי בלקוח – **(Client-Side Data Store)** שמירת מידע זמני או קבוע במחשב הלקוח, כגון קובצי Cookie והיסטוריית פעולות.

פירוט מאגרי המידע של המערכת:

מאגר קבצים (File System)

תיקיית אחסון ראשית/server/cloud :

מבנה האחסון לפי משתמשים:

```
cloud/
│── user_123456/
│   ├── photos/
│   │   ├── photo1.jpg
│   │   ├── photo2.png
│   ├── documents/
│   │   ├── report.docx
│   ├── project.zip
```

שדות עיקריים במאגר הקבצים:

שם השדה	תיאור	טיפוס נתונים	אורך/גודל	דוגמאות לערכים

user_id	מזהה המשתמש	מחרוזת (UUID)	32תווים	"user_123456"
file_id	מזהה ייחודי לקובץ	מחרוזת (UUID)	32תווים	"file_abcdef123456"
directory	שם התיקייה שבה הקובץ מאוחסן	מחרוזת	255תווים	"מסמכים"
file_name	שם הקובץ המקורי	מחרוזת	255תווים	"report.docx"
server_name	שם ייחודי בשרת	מחרוזת (UUID)	32תווים	"abcd1234efgh5678"
size	גודל הקובץ בבייטים	מספר שלם	עד 8 בתים	1048576 (1MB)
last_modified	תאריך עדכון אחרון	תאריך/זמן	פורמט YYYY-MM-DD HH:MM:SS	"2024-01-01 15:30:00"

פעולות אפשריות על מאגר הקבצים:

- העלאה, הורדה, מחיקה, שינוי שם
- שיתוף וניהול הרשאות
- מעקב אחרי שינויים

מאגר נתונים בזיכרון(In-Memory Data Store)

מאגר זה מנוהל ישירות בשרת ומספק מידע דינמי מבלי לגשת למסד הנתונים בכל בקשה.

שדות עיקריים במאגר הנתונים בזיכרון:

שם השדה	תיאור	טיפוס נתונים	דוגמאות לערכים

clients	רשימת משתמשים מחוברים	מילון ({id: Client})	{ "user_123456": ClientObject }
bytes_received	כמות הנתונים שהמשתמש קיבל	מספר שלם	15728640 (15MB)
bytes_sent	כמות הנתונים שהמשתמש שלח	מספר שלם	5242880 (5MB)
files_uploading	קבצים בתהליך העלאה	רשימה(List)	["file_abc123", "file_xyz789"]
files_in_use	קבצים שנפתחו לצפייה	רשימה(List)	["file_123456", "file_654321"]
active_sessions	חיבורים פעילים לשרת	מילון ({id: Session})	{ "user_789012": SessionObject }

שימושים עיקריים:

- ניהול חיבורים למערכת
- שמירה על סטטוס של העלאות והורדות
- מניעת מחיקת קבצים בזמן צפייה

מאגר נתונים מקומי בלקוח(Client-Side Data Store)

קובצי Cookie ואחסון מקומי:

שם השדה	תיאור	טיפוס נתונים	אורך/גודל	דוגמאות לערכים
cookie	מזהה משתמש לשמירת התחברות	מחרוזת	512תווים	"cookie_data_abc123"

cookie_expiration	תאריך תפוגה של קובץ ה-Cookie	תאריך/זמן	פורמט YYYY-MM-DD HH:MM:SS	"2025-01-01 12:00:00"

קובצי מטמון והיסטוריה:(/cache)

שם השדה	תיאור	טיפוס נתונים	אורך/גודל	דוגמאות לערכים
uploading_files.json	נתונים על קבצים בתהליך העלאה	JSON	עד 1MB	{ "file_123": { "path": "C:/file.txt", "progress": 50 } }
downloading_files.json	נתונים על קבצים בתהליך הורדה	JSON	עד 1MB	{ "file_456": { "path": "C:/download.txt", "progress": 75 } }

משתנים דינמיים בלקוח:(gui.py)

שם השדה	תיאור	טיפוס נתונים	דוגמאות לערכים
user	מידע על המשתמש המחובר	מילון (dict)	{ "username": "idan_hazay", "email": "user@example.com" }
files	רשימת הקבצים בתיקייה הפעילה	רשימה (List)	["file_123", "file_456"]
directories	רשימת התיקיות בתיקייה הפעילה	רשימה (List)	["dir_abc", "dir_xyz"]
files_downloading	קבצים בתהליך הורדה	מילון (dict)	{ "file_654": { "progress": 20 } }

שימושים עיקריים של מאגר הנתונים בלקוח:

- שמירת מידע על קבצים שהורדו או הועלו
- ניהול חוויית המשתמש מבלי לדרוש גישה מתמדת לשרת
- שיפור ביצועים על ידי טעינת נתונים מהזיכרון

<u>מסד הנתונים:</u>

למסד הנתונים קוראים database.db

טבלאות עיקריות במסד הנתונים:

שם הטבלה	תיאור
Users	טבלת המשתמשים, שומרת פרטי משתמשים כולל אימייל, סיסמה, סטטוס אימות, רמת מנוי וכו.'
Files	טבלת הקבצים, כוללת פרטי קובץ כגון מזהה, שם קובץ, גודל, תיקיית הורה, תאריך עדכון אחרון.
Directories	טבלת התיקיות, מאחסנת תיקיות ומיקומן במערכת.
Permissions	טבלת הרשאות, מנהלת את ההרשאות של כל קובץ ותיקייה למשתמשים שונים.
Deleted	טבלת הקבצים שנמחקו, הכוללת מידע על קבצים שהועברו למחיקה ומועד מחיקתם הצפוי.

מבנה מפורט של הטבלאות:

טבלת המשתמשים (Users)

תיאור :מכילה מידע על כל המשתמשים במערכת.

מפתח ראשי) id :(Primary Key) מזהה משתמש ייחודי(

שם השדה	טיפוס נתונים	אורך	תיאור	ערכים לדוגמה

id	TEXT (PK)	32 תווים	מזהה ייחודי של המשתמש	"a1b2c3d4e5f6..."
email	TEXT (UNIQUE)	255	כתובת אימייל ייחודית	"user@example.com"
username	TEXT (UNIQUE)	50	שם משתמש	"idan_hazay"
password	TEXT	255	סיסמה מוצפנת	"5f4dcc3b5aa765..."
salt	TEXT	255	מלח (Salt) להצפנת סיסמה	"random_salt"
verified	BOOLEAN	-	האם המשתמש אומת?	True / False
subscription_level	INTEGER	1	רמת מנוי (0=חינם, 1=בסיסי, 2=מתקדם, 3=פרו)	0 / 1 / 2
cookie	TEXT (UNIQUE)	512	מזהה cookie לשמירת התחברות	"cookie_data..."
cookie_expiration	TEXT	50	תאריך תפוגה של העוגייה	"2025-01-01 12:00:00"

טבלת הקבצים (Files)

תיאור :מידע על כל הקבצים שהועלו למערכת.

מפתח ראשי (Primary Key): id

שם השדה	טיפוס נתונים	אורך	תיאור	ערכים לדוגמה
id	TEXT (PK)	32 תווים	מזהה קובץ ייחודי	"file123456abc"

sname	TEXT (UNIQUE)	255	שם ייחודי בשרת	"abcd1234efgh5678"
fname	TEXT	255	שם קובץ שהמשתמש רואה	"דוח_שנתי.pdf"
parent	TEXT	32	מזהה התיקייה המכילה	"dir5678xyz"
owner_id	TEXT	32	מזהה המשתמש שהעלה	"user123456"
size	INTEGER	-	גודל הקובץ בבייטים	1048576 (1MB)
last_edit	TEXT	50	תאריך עדכון אחרון	"2024-01-01 15:30:00"

טבלת התיקיות (Directories)

מבנה התיקיות של המשתמשים.

מפתח ראשי (Primary Key): id

שם השדה	טיפוס נתונים	אורך	תיאור	ערכים לדוגמה
id	TEXT (PK)	32	מזהה תיקייה ייחודי	"dir98765xyz"
name	TEXT	255	שם התיקייה	"תמונות"
parent	TEXT	32	מזהה תיקיית ההורה	"dir5678xyz"
owner_id	TEXT	32	מזהה המשתמש	"user123456"

טבלת ההרשאות (Permissions)

שומרת הרשאות שיתוף בין משתמשים שונים.

מפתח ראשי (Primary Key): id

שם השדה	טיפוס נתונים	אורך	תיאור	ערכים לדוגמה
id	TEXT (PK)	32	מזהה הרשאה ייחודי	"perm876543"
file_id	TEXT	32	מזהה קובץ/תיקייה	"file123456abc"
owner_id	TEXT	32	המשתמש ששיתף	"user123456"
user_id	TEXT	32	המשתמש המקבל הרשאה	"user789012"
read	BOOLEAN	-	הרשאת קריאה	True / False
write	BOOLEAN	-	הרשאת כתיבה	True / False
delete	BOOLEAN	-	הרשאת מחיקה	True / False
rename	BOOLEAN	-	הרשאת שינוי שם	True / False
download	BOOLEAN	-	הרשאת הורדה	True / False
share	BOOLEAN	-	האם המשתמש יכול לשתף הלאה	True / False

טבלת קבצים שנמחקו (Deleted)

שומרת מידע זמני על קבצים שנמחקו לפני מחיקה לצמיתות.

מפתח ראשי id :(Primary Key)

שם השדה	טיפוס נתונים	אורך	תיאור	ערכים לדוגמה
id	TEXT (PK)	32	מזהה קובץ שנמחק	"file789xyz"
owner_id	TEXT	32	המשתמש שמחק את הקובץ	"user123456"
time_to_delete	TEXT	50	תאריך מחיקה סופי	"2024-02-01 00:00:00"

סקירת חולשות ואיומים:

הצפנת סיסמאות ואבטחת פרטי משתמשים

המערכת מאחסנת את סיסמאות המשתמשים באמצעות הצפנת bcrypt יחד עם Salt ייחודי לכל סיסמה. שיטה זו מבטיחה שגם אם מסד הנתונים ייחשף, לא ניתן יהיה לשחזר את הסיסמאות בקלות . Bcrypt בנוי כך שהוא מכניס עיכוב מובנה בחישוב, מה שמקשה על מתקפות כוח גס .(Brute Force) בעת התחברות, הסיסמה שהוזנה מוצפנת מחדש ומושווית לערך השמור במסד הנתונים, כך שאין צורך לשמור סיסמאות בפורמט טקסטואלי.

מניעת SQL Injection והגבלת גישה למסד הנתונים

כדי למנוע מתקפות ,SQL Injection כל הבקשות למסד הנתונים משתמשות בפרמטרים מוכנים מראש .(Prepared Statements)שימוש בטכניקה זו מונע הכנסת שאילתות זדוניות שעלולות למחוק או לגשת לנתונים רגישים. בנוסף, הרשאות במסד הנתונים הוגדרו כך שמשתמשי המערכת יכולים לבצע רק פעולות רלוונטיות עבורם, והגישה לקריאה וכתיבה מוגבלת בהתאם.

ניהול הרשאות ואבטחת גישה לקבצים

כל קובץ במערכת משויך למשתמש שהעלה אותו, והגישה אליו מוגבלת בהתאם להרשאות המוגדרות בטבלת .Permissionsמשתמשים יכולים להגדיר האם קובץ יהיה נגיש רק לקריאה, לכתיבה, למחיקה או לשיתוף. לפני שליחת קובץ למשתמש, המערכת מבצעת בדיקה לוודא שיש לו גישה מתאימה. כך נמנעת גישה בלתי מורשית לקבצים פרטיים.

הצפנת נתונים בתקשורת בין הלקוח לשרת

כדי להגן על נתוני המשתמשים בעת התקשורת בין השרת ללקוח, כל הנתונים מוצפנים באמצעות הצפנת .AES-256 בנוסף, המערכת משתמשת בפרוטוקול החלפת מפתחות RSA כך שהתקשורת הראשונית בין השרת ללקוח מתבצעת בערוץ מאובטח. הצפנה זו מונעת יירוט של נתונים במהלך התקשורת.

אבטחת עוגיות ושמירת התחברות

כדי למנוע חטיפת חשבון באמצעות גניבת עוגיות ,(Session Hijacking) כל עוגיית התחברות מוגדרת עם חתימה. בנוסף, כל עוגיה מוגבלת בזמן תפוגה קבוע, כך שאפילו אם היא נגנבה, היא תהיה חסרת תוקף לאחר זמן קצר.

הגבלת משאבי מערכת לפי סוג מנוי

כדי להבטיח שמשתמשים לא ינצלו משאבים באופן בלתי הוגן, למשתמשים ברמות מנוי שונות יש הגבלות שונות על מהירות העלאה והורדה. משתמשים בחשבון חינמי מקבלים מהירות העלאה נמוכה יותר, בעוד שלמשתמשים במנוי מתקדם יש עדיפות על משאבי השרת.

מימוש הפרויקט

סקירת כל המודולים והמחלקות המרכיבים את המערכת והקשרים ביניהם:

מודולים מיובאים:

מודולים מיובאים בשרת

שם המודול	ייעוד המודול
os	מספק פונקציות לניהול קבצים ותיקיות במערכת ההפעלה.
sqlite3	משמש לניהול מסד הנתונים מסוג SQLite.
traceback	מאפשר מעקב אחר חריגות ולוגים של שגיאות.
datetime	משמש לניהול תאריכים ושעות, כולל חישובי זמנים.
hashlib	מספק פונקציות להצפנת מידע באמצעות אלגוריתמים שונים כגון SHA-256.
rsa	משמש להצפנה א-סימטרית עבור חילופי מפתחות מאובטחים.
struct	מאפשר עבודה עם נתונים בבינארי, כולל קידוד ופענוח מידע ברשת.
Crypto	מספק הצפנת AES לצורך אבטחת תקשורת בין השרת ללקוח.
socket	מאפשר יצירת חיבורי רשת TCP/IP לתקשורת בין השרת ללקוח.
psutil	מספק מידע על תהליכים וסטטיסטיקות מערכת, כולל ניהול חיבורים.
base64	משמש לקידוד ופענוח נתונים בטקסט מוצפן, כמו בהצפנת קבצים.
functools	מספק פונקציות עזר לניהול תהליכים, כולל תזמון קריאות פונקציה.

מודולים מיובאים בלקוח

שם המודול	ייעוד המודול
sys	משמש לניהול פרמטרים של המערכת ולשליטה בביצוע הקוד.
traceback	משמש למעקב אחר שגיאות ולתיעוד חריגות בעת ביצוע הקוד.

socket	מאפשר יצירת חיבורי רשת TCP/IP בין הלקוח לשרת.
struct	מסייע בהמרת נתונים לפורמט בינארי לצורך שליחה ברשת.
hashlib	משמש להצפנת מידע, כמו יצירת מפתחות לשמירת סיסמאות.
rsa	מספק הצפנה א-סימטרית לביצוע חילופי מפתחות מאובטחים.
Crypto	משמש להצפנת AES כדי להגן על התקשורת עם השרת.
base64	מאפשר קידוד נתונים, בעיקר לצורך הצפנת קבצים.
uuid	משמש ליצירת מזהים ייחודיים עבור קבצים ומשתמשים.
time	מספק פונקציות למדידת זמן ולתזמון פעולות.
re	משמש לטיפול בביטויים רגולריים.(Regular Expressions)
json	משמש לקריאה וכתיבה של קובצי ,JSON כגון קובצי מטמון ושמירת נתונים זמניים.
PyQt6	מספק את הממשק הגרפי של הלקוח, כולל תצוגה אינטראקטיבית.
docx	מאפשר פתיחה וקריאה של קובצי Word בתוך היישום.

מודולים שפתחתי:

מחלקות בצד השרת

מחלקה Server :

תפקיד המחלקה:

מנהל את השרת הראשי, כולל חיבורי משתמשים, ניהול בקשות, ותקשורת עם מסד הנתונים.

תכונות המחלקה

שם התכונה	תפקיד ושימוש
clients	מילון של חיבורי לקוחות פעילים({id: Client})
database	מופע של מחלקת ניהול מסד הנתונים

פעולות במחלקה

שם פעולה	טענת כניסה	טענת יציאה
start_server	אין	מפעיל את השרת וממתין לחיבורים
handle_client	client_socket, address	מטפל בבקשות הלקוח ושולח תשובות מתאימות
stop_server	אין	סוגר את כל החיבורים ומכבה את השרת

מחלקה DatabaseHandler :

תפקיד המחלקה:

ניהול פעולות קריאה וכתיבה במסד הנתונים.

תכונות המחלקה

שם התכונה	תפקיד ושימוש
connection	חיבור פעיל למסד הנתונים
cursor	אובייקט ביצוע שאילתות

פעולות במחלקה

שם פעולה	טענת כניסה	טענת יציאה
get_user	user_id	מחזיר את פרטי המשתמש לפי המזהה
add_file	file_data	מוסיף רשומה חדשה לטבלת הקבצים
remove_file	file_id	מוחק קובץ ממסד הנתונים

מחלקה ProtocolServer :

תפקיד המחלקה:

מטפל בפרוטוקול התקשורת עם הלקוחות, מעבד פקודות ומבצע את הפעולות המתאימות.

תכונות המחלקה

שם התכונה	תפקיד ושימוש
server	מופע של מחלקת השרת
database	מופע של מחלקת מסד הנתונים

פעולות במחלקה

שם פעולה	טענת כניסה	טענת יציאה
parse_command	command, client_socket	מזהה את סוג הבקשה ומפנה לביצוע המתאים
send_response	client_socket, response	שולח תשובה מתאימה ללקוח

מחלקה Encryption :

תפקיד המחלקה:

מטפלת בהצפנה ופענוח של נתונים בתקשורת.

תכונות המחלקה

שם התכונה	תפקיד ושימוש
block_size	גודל בלוק הצפנה עבורAES

פעולות במחלקה

שם פעולה	טענת כניסה	טענת יציאה
encrypt	plain_text, key	מחזירה את הטקסט המוצפן
decrypt	encrypted_text, key	מחזירה את הטקסט המקורי

מחלקות בצד הלקוח

מחלקה Application :

תפקיד המחלקה:

מנהל את ממשק המשתמש והחיבור לרשת.

תכונות המחלקה

שם התכונה	תפקיד ושימוש
qtapp	מופע שלQApplication
network	מופע של מחלקת התקשורת

פעולות במחלקה

שם פעולה	טענת כניסה	טענת יציאה
start_app	אין	מפעיל את ממשק המשתמש והחיבור לרשת
handle_reply	reply	מעבד את תגובת השרת ומעדכן את הGUI-

מחלקה Network :

תפקיד המחלקה:

אחראית על יצירת חיבור לרשת ושליחת נתונים לשרת.

תכונות המחלקה

שם התכונה	תפקיד ושימוש
sock	מופע של socket לניהול חיבור רשת
shared_secret	מפתח הצפנהAES

פעולות במחלקה

שם פעולה	טענת כניסה	טענת יציאה
send_data	data, encryption	שולח נתונים לשרת עם הצפנה או בלעדיה
recv_data	אין	מחזיר את הנתונים שהתקבלו מהשרת

מחלקה ProtocolClient :

תפקיד המחלקה:

מטפלת בשליחת פקודות לשרת ופענוח תשובות.

תכונות המחלקה

שם התכונה	תפקיד ושימוש
network	מופע של מחלקת Network

פעולות במחלקה

שם פעולה	טענת כניסה	טענת יציאה
send_command	command	שולח פקודה לשרת ומחכה לתשובה
parse_reply	reply	מזהה את הפקודה ומעביר להמשך עיבוד

מחלקה FileSending :

תפקיד המחלקה:

מטפלת בהעלאת קבצים לשרת וניהול קבצים בתהליך העלאה.

תכונות המחלקה

שם התכונה	תפקיד ושימוש
active_threads	רשימה של תהליכי העלאה פעילים
file_queue	רשימה של קבצים שממתינים להעלאה

פעולות במחלקה

שם פעולה	טענת כניסה	טענת יציאה
send_files	cmd, file_id, resume_file_id	מתחיל העלאת קובץ לשרת
resume_files_upload	id, progress	ממשיך העלאה של קובץ מהנקודה שבה נעצר

מחלקה FileViewer :

תפקיד המחלקה:

מציג קבצים שנבחרו לצפייה בתוך ממשק המשתמש.

תכונות המחלקה

שם התכונה	תפקיד ושימוש
file_path	הנתיב לקובץ הנבחר
title	כותרת החלון

פעולות במחלקה

שם פעולה	טענת כניסה	טענת יציאה
open_in_native_app	אין	פותח את הקובץ באפליקציה ברירת המחדל של המערכת
file_viewer_dialog	אין	מציג את הקובץ בתוך חלון פנימי

קטעי קוד ופיתוחים מיוחדים:

שיחזור העלאה והורדה של קבצים לאחר התנתקות/קריסה

אחת הבעיות המרכזיות במערכות אחסון בענן היא קטיעת חיבור בעת העלאת או הורדת קובץ, מה שעלול לגרום לאובדן נתונים ולהתחלת הפעולה מחדש. כדי לפתור זאת, המערכת כוללת מנגנון חידוש חיבור (Resumable Upload/Download) ששומר את מצב ההעלאה או ההורדה במסד הנתונים או בקובץ זמני. במקרה של ניתוק, כאשר המשתמש מתחבר מחדש, המערכת מזהה את הנקודה שבה הופסקה הפעולה וממשיכה אותה מאותה נקודה במקום להתחיל מחדש. כך, חוויית המשתמש משתפרת, נחסך זמן, ונמנע בזבוז של רוחב פס.

```
elif code == "RESR":  # File upload resumed
    file_id, progress = fields[1], fields[2]
    self.window.file_sending.resume_files_upload(file_id, progress)
    to_show = f"File upload of file {file_id} continued at {progress}"

elif code == "RUSR":  # File download resumed
    file_id, progress = fields[1], fields[2]
    to_show = f"Resumed download of file {file_id} from byte {progress}"
```

תקשורת אסינכרונית

כאשר מערכת אחסון בענן מבצעת פעולות כגון העלאה, הורדה או בקשות לשרת, שימוש בגישה מסונכרנת עלול לגרום לתקיעות ולחוסר תגובתיות בממשק המשתמש. כדי למנוע זאת, התקשורת מתבצעת באופן אסינכרוני – כלומר, הבקשות נשלחות ברקע, והמשתמש יכול להמשיך להשתמש במערכת מבלי לחכות לתשובה מהשרת.

```
class ReceiveThread(QThread):
        # Define a signal to emit data received from recv_data
        reply_received = pyqtSignal(bytes)

        def __init__(self, network):
            super().__init__()
            self.running = True  # Add a flag to control the thread loop
            self._pause_event = threading.Event()  # Event to manage pausing
            self._pause_event.set()  # Initially, the thread is not paused
            self.network = network

        def run(self):
            while self.running:
                # Wait for the thread to be resumed if paused
                self._pause_event.wait()

                # Simulate receiving data
                reply = self.network.recv_data()  # Assume this method exists and returns bytes
                if reply:
                    self.reply_received.emit(reply)  # Emit the received reply to the main thread
```

שיתוף קבצים

המערכת מאפשרת למשתמשים לשתף קבצים עם אחרים, תוך שליטה ברמות ההרשאה לכל קובץ. שיתוף יכול להתבצע באמצעות קישורים ישירים או על ידי הגדרת הרשאות למשתמשים ספציפיים (כגון צפייה בלבד, עריכה, מחיקה, או שיתוף מחדש). הנתונים על ההרשאות נשמרים במסד הנתונים, וניתן לעדכן או לבטל את השיתוף בכל עת. המערכת גם מספקת אפשרות לייצר קישורי שיתוף מוגבלים בזמן, כך שהגישה לקובץ תפוג לאחר פרק זמן מוגדר.

```
elif code == "SHRR":  # Sharing options retrieved
    file_id, user_cred, file_name = fields[1], fields[2], fields[3]
    if len(fields) == 3:
        self.window.share_file(file_id, user_cred, file_name)
    else:
        self.window.share_file(file_id, user_cred, file_name, *fields[4:])
    to_show = "Sharing options received"
```

שמירת התחברות / התחברות אוטומטית

כדי לחסוך מהמשתמש את הצורך להזין את פרטי ההתחברות שלו בכל כניסה, המערכת מאפשרת שמירה של אישורי הגישה באופן מאובטח. בעת ההתחברות, נוצר **Token** ייחודי הנשמר בקובץ cookies מוצפן או במסד הנתונים, והוא משמש לאימות המשתמש בעת כניסה מחדש. במידה שהמשתמש מתנתק באופן ידני או לאחר זמן ארוך של חוסר פעילות, ה Token-יימחק או יפוג כדי למנוע גישה לא מורשית.

```
def send_cookie(self):
    """Sends stored user authentication cookie to the server."""
    try:
        with open(COOKIE_PATH, "r") as f:
            cookie = f.read()
            self.send_data(b"COKE|" + cookie.encode())
    except:
        print("Cookie file not found")
```

הצפנה - החלפת מפתחות עם הצפנה ואז תקשורת מאובטחת

אבטחת התקשורת בין הלקוח לשרת היא קריטית כדי למנוע גישה בלתי מורשית לנתונים. לשם כך, המערכת משתמשת בפרוטוקול RSA להחלפת מפתחות מוצפנים בין השרת ללקוח, ולאחר מכן כל התקשורת מוצפנת באמצעות AES. בצורה זו, גם אם צד שלישי יירט את התקשורת, הוא לא יוכל

לפענח את הנתונים ללא המפתח המתאים. שיטה זו מגנה על פרטיות המשתמשים ומבטיחה שהקבצים שלהם מוגנים מפני התקפות.

```
def rsa_exchange(self):
    try:
        self.network.send_data_wrap(b"RSAR", False)
        s_public_key = self.recv_rsa_key()
        shared_secret = self.send_shared_secret(s_public_key)
        return shared_secret
    except:
        print(traceback.format_exc())
```

שמירת קבצים במסד נתונים

במקום לשמור קבצים ישירות במערכת הקבצים של השרת, המערכת מאחסנת אותם במסד נתונים. גישה זו משפרת את האבטחה בכך שהיא מונעת שינוי בלתי מורשה של קבצים, מאפשרת גיבויים נוחים יותר, ומקלה על חיפוש ומעקב אחר קבצים. כמו כן, היא מאפשרת שליטה טובה יותר על הגישה לקבצים, שכן כל בקשת קובץ חייבת לעבור דרך שכבת האימות של המערכת.

```
def add_file(self, file_dict):
    """
    Adds a new file entry to the database.
    """
    conn = sqlite3.connect(self.database)
    cursor = conn.cursor()
    columns = ', '.join(file_dict.keys())  # Extract column names
    values = ', '.join(['?'] * len(file_dict))  # Create placeholders for values
    sql = f"INSERT INTO {self.files_table} ({columns}) VALUES ({values})"

    try:
        cursor.execute(sql, list(file_dict.values()))  # Execute the insert query
        conn.commit()
    except sqlite3.IntegrityError:
        print("Key values already exist in table")  # Log integrity constraint error
    conn.close()
```

ממשק משתמש דינמי

המערכת כוללת ממשק שמתאים את עצמו בצורה חכמה לגודל החלון, מספר הקבצים המוצגים ורזולוציית המסך. כך, משתמשים במכשירים שונים – ממחשבים נייחים ועד טלפונים ניידים – יוכלו

לעבוד עם המערכת בצורה נוחה. ההתאמה הדינמית מתבצעת באמצעות פונקציה שכתבתי שמשנה את גודל הממשק עם שינוי גודל המסך.

```
def resizeEvent(self, event):
    """Dynamically resizes widgets based on the new window size."""
    new_width, new_height = self.width(), self.height()
    width_ratio, height_ratio = new_width / self.original_width, new_height / self.original_height

    for widget in self.findChildren(QWidget):
        if widget in self.original_sizes:
            original_geometry = self.original_sizes[widget]['geometry']
            original_font_size = self.original_sizes[widget]['font_size']

            new_x = int(original_geometry.x() * width_ratio) if width_ratio ≠ 1 else original_geometry.x()
            new_width = int(original_geometry.width() * width_ratio) if width_ratio ≠ 1 else original_geometry.width()

            new_y = int(original_geometry.y() * height_ratio) if height_ratio ≠ 1 else original_geometry.y()
            new_height = int(original_geometry.height() * height_ratio) if height_ratio ≠ 1 else original_geometry.height()

            widget.setGeometry(new_x, new_y, new_width, new_height)
            widget.updateGeometry()

            new_font_size = max(int(original_font_size * (width_ratio + height_ratio) / 2), 8)
            font = widget.font()
            font.setPointSize(new_font_size)
            widget.setFont(font)

            if isinstance(widget, QPushButton):
                icon = widget.icon()
                if not icon.isNull():
                    base = 60 if widget.text() == "" else 16
                    new_icon_size = int(base * (width_ratio + height_ratio) / 2)
                    widget.setIconSize(QSize(new_icon_size, new_icon_size))
```

קבלת אייפי משרת לא ידוע (DHCP)

במקרים שבהם השרת מתארח ברשת עם כתובת **IP דינמית**, הלקוח לא יכול להתחבר אליו באמצעות כתובת IP קבועה. כדי לפתור זאת, המערכת משתמשת במנגנון **UDP Broadcast** שבו השרת משדר את כתובתו לכל הלקוחות ברשת המקומית. הלקוחות מאזינים לשידור הזה, מזהים את הכתובת הנכונה, ומתחברים אליה באופן אוטומטי.

```
def search_server(self):
    """Broadcasts a search request to locate an available server on the network."""
    try:
        search_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
        search_socket.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
        search_socket.settimeout(SOCK_TIMEOUT)

        netmask, ip = self.get_subnet_mask()  # Get subnet information
        broadcast_address = self.get_broadcast_address(ip, netmask)
        search_socket.sendto(b"SEAR", (broadcast_address, 31026))  # Broadcast search request

        response, addr = search_socket.recvfrom(1024)
        response = response.decode().split("|")
        if response[0] == "SERR":
            ip, port = response[1], response[2]
            return ip, int(port)

    except TimeoutError:
        print("No server found")
        return SAVED_IP, SAVED_PORT
    except:
        print(traceback.format_exc())
        return SAVED_IP, SAVED_PORT
```

אימות משתמש עם מייל

בעת הרשמה למערכת, המשתמש מתבקש להזין את כתובת הדוא"ל שלו, ולאחר מכן נשלח אליו קוד אימות חד-פעמי. המשתמש נדרש להזין את הקוד במערכת כדי להשלים את הרישום. תהליך זה מונע יצירת חשבונות מזויפים, מאבטח את הגישה למערכת, ומוודא שהמשתמש אכן בעל השליטה על כתובת הדוא"ל שסופקה.

```python
def send_verification(self, email):
    """
    Sends an account verification email with a randomly generated 6-digit code.
    Stores the code in the database with a 30-minute expiration.
    """
    id = self.db.get_user_id(email)  # Retrieve user ID from email
    code = random.randint(100000, 999999)  # Generate verification code
    valid_until = str(timedelta(minutes=30) + datetime.now())  # Set expiration time
    self.db.update_user(id, ["last_code", "valid_until"], [code, valid_until])  # Store code in database

    em = EmailMessage()  # Build email
    em["From"] = self.gmail
    em["To"] = email
    em["Subject"] = "Account Verification"
    body = f"Your account verification code is: {code}\nCode is valid for 30 minutes"
    em.set_content(body)
    self.send_mail(em, email)  # Send email
```

הגבלות הורדה והעלאה למשתמשים, רמות משתמש

לא כל המשתמשים במערכת הם בעלי אותן הרשאות – חלקם יכולים רק להוריד קבצים, אחרים יכולים גם להעלות, לערוך או לשתף. בנוסף, לכל משתמש מוקצה נפח אחסון ורוחב פס מוגדרים מראש, בהתאם לרמתו במערכת. ניהול ההרשאות מתבצע באמצעות מסד הנתונים, שבו לכל משתמש מוקצות הרשאות גישה בהתאם לתפקידו או למנוי שלו.

```python
class Limits:
    """
    Users networking and files limitations, based on subscription
    """
    def __init__(self, level):
        level = int(level)
        if (level == 0):
            self.max_storage = 100_000
            self.max_file_size = 50
            self.max_upload_speed = 5
            self.max_download_speed = 10
        elif (level == 1):
            self.max_storage = 250_000
            self.max_file_size = 100
            self.max_upload_speed = 10
            self.max_download_speed = 20
        elif (level == 2):
            self.max_storage = 500_000
            self.max_file_size = 250
            self.max_upload_speed = 15
            self.max_download_speed = 30
        elif (level == 3):
            self.max_storage = 1_000_000
            self.max_file_size = 500
            self.max_upload_speed = 25
            self.max_download_speed = 50
        else:
            raise Exception
```

סימון כמה קבצים ותיקיות בו זמנית, הורדה כ- ZIP

כאשר משתמשים רוצים להוריד מספר קבצים יחד, המערכת מאפשרת להם לבחור כמה קבצים או תיקיות ולהוריד אותם כקובץ ZIP אחד. גישה זו מקלה על המשתמשים ומאפשרת הורדה מהירה ויעילה יותר. בעת בחירת הקבצים, המערכת יוצרת קובץ ZIP דינמי ושולחת אותו להורדה במקום לשלוח כל קובץ בנפרד.

```python
def zip_files(self, ids):
    """
    Creates a zip file containing the specified files and directories.
    """
    zip_buffer = io.BytesIO()  # Create an in-memory zip buffer

    with zipfile.ZipFile(zip_buffer, 'w', zipfile.ZIP_DEFLATED) as zf:
        for file_id in ids:
            if self.get_file_sname(file_id) is not None:
                # It's a file
                file_path = self.server_path + "\\cloud\\" + self.get_file_sname(file_id)
                file_name = self.get_file_fname(file_id)
                zf.write(file_path, file_name)  # Add file to zip archive
            elif self.get_dir_name(file_id) is not None:
                # It's a directory, use zip_directory to add contents
                directory_buffer = self.zip_directory(file_id)
                with zipfile.ZipFile(directory_buffer, 'r') as dir_zip:
                    for name in dir_zip.namelist():
                        dir_name = self.get_dir_name(file_id)
                        zf.writestr(f"{dir_name}/{name}", dir_zip.read(name))  # Maintain folder structure

    zip_buffer.seek(0)  # Reset buffer position
    return zip_buffer  # Return zip file

def zip_directory(self, directory_id):
    """
    Creates a zip archive containing all files and directories within the specified directory.
    """
    directory_contents = self.db.get_directory_contents(directory_id)  # Retrieve directory contents
    zip_buffer = io.BytesIO()  # Create in-memory zip buffer

    with zipfile.ZipFile(zip_buffer, 'w', zipfile.ZIP_DEFLATED) as zf:
        for full_path, relative_path in directory_contents:
            zf.write(full_path, relative_path)  # Add files to zip archive

    zip_buffer.seek(0)  # Reset buffer position
    return zip_buffer  # Return zip file buffer
```

מסמך בדיקות מלא:

בדיקות שתוכננו בשלב האפיון:

בדיקות התבצעו על מערכת אחסון הענן במטרה לוודא שהיא פועלת כמצופה, הן מבחינת פונקציונליות והן מבחינת אבטחה ויציבות. הבדיקות כוללות בדיקות יחידה (Unit Testing), בדיקות אינטגרציה, בדיקות עומסים (Stress Testing) ובדיקות אבטחה.

1. שיחזור העלאה והורדה לאחר התנתקות/קריסה

מטרת הבדיקה - לוודא שהמשתמש יכול להמשיך בהעלאה או בהורדה של קובץ מהמקום שבו הפעולה נעצרה במקרה של ניתוק או קריסת התוכנה.

מה בוצע בפועל - התחלתי בהעלאת קובץ גדול (1GB) לשרת. לאחר שהועלו 50% מהנתונים, ניתקתי את החיבור באופן יזום. חידשתי את ההתחברות ובדקתי אם ההעלאה ממשיכה מהמקום שבו הופסקה. ביצעתי בדיקות דומות להורדה של קובץ גדול.

תוצאות הבדיקה - ההעלאה וההורדה חזרו לנקודה שבה נפסקו ללא צורך להתחיל מחדש.

בעיות שהתגלו וכיצד נפתרו - בגרסה הראשונה, כאשר החיבור נותק, המערכת לא שמרה את מצב ההעלאה בזמן אמת, ולכן הנתונים שהועלו עד הניתוק אבדו.
פתרון :הוספתי מנגנון שמירה במסד הנתונים על מצב ההעלאה.

2. תקשורת אסינכרונית

מטרת הבדיקה - לוודא שהמערכת אינה קופאת כאשר מתבצעות העלאות והורדות של קבצים ברקע.

מה בוצע בפועל - הפעלתי העלאת קובץ תוך כדי ניווט בממשק המשתמש. בדקתי אם לחצנים אחרים בממשק עדיין מגיבים. בדקתי האם ניתן להתחיל פעולות נוספות במקביל.

תוצאות הבדיקה - המערכת המשיכה לפעול כרגיל בזמן ההעלאה.

בעיות שהתגלו וכיצד נפתרו - במקרים מסוימים, בעת העלאה של קובץ גדול, התוכנה קפאה לזמן קצר.
פתרון :העברתי את כל פעולות הרשת לסביבת **Thread נפרד** ולתקשורת אסינכרונית.

3. שיתוף קבצים

מטרת הבדיקה - לוודא שמשתמשים יכולים לשתף קבצים עם אחרים, ושמערכת ההרשאות עובדת כראוי.

מה בוצע בפועל - משתמש A שיתף קובץ עם משתמש B עם הרשאת "קריאה בלבד." משתמש B ניסה לפתוח את הקובץ ולערוך אותו. בדקתי אם אפשר לבטל שיתוף ולהחזיר אותו.

תוצאות הבדיקה - מערכת ההרשאות עבדה כמצופה.

בעיות שהתגלו וכיצד נפתרו - בתחילה, כאשר קובץ נמחק, הוא עדיין הופיע למשתמש השני.
פתרון :הוספתי בדיקה במסד הנתונים שמסירה קבצים משותפים כאשר הבעלים מוחק אותם.

4. שמירת התחברות / התחברות אוטומטית

מטרת הבדיקה - לוודא שמשתמשים יכולים להתחבר אוטומטית אם בחרו באפשרות זו.

מה בוצע בפועל - משתמש התחבר עם שמירת התחברות. התוכנה נסגרה ונפתחה מחדש. בדקתי אם המשתמש נשאר מחובר ללא צורך בהזנת סיסמה מחדש.

תוצאות הבדיקה - ההתחברות האוטומטית עבדה כצפוי.

בעיות שהתגלו וכיצד נפתרו - בגרסה הראשונה, כאשר Token נשמר, הוא לא היה מוצפן.
פתרון :שמרתי את ה Token-בקובץ cookiesמוצפן עם .AES

5. הצפנה ותקשורת מאובטחת

מטרת הבדיקה - לוודא שכל הנתונים המועברים מוצפנים ושלא ניתן להאזין להם.

מה בוצע בפועל - שלחתי קובץ לשרת תוך שימוש ב Wireshark-כדי לבדוק האם התוכן מועבר בטקסט ברור. בדקתי אם השרת והלקוח מחליפים מפתחות מוצפנים לפני תחילת ההעברה.

תוצאות הבדיקה - הנתונים הוצפנו בהצלחה והפרוטוקול RSA הופעל כראוי.

בעיות שהתגלו וכיצד נפתרו - בתחילה, חיבור הלקוח נכשל לעיתים בגלל חישוב מפתח שגוי.
פתרון :השתמשתי באותו גנרטור מפתחות בכל הצדדים למניעת חוסר תאימות.

6. עריכת קבצים וצפייה

מטרת הבדיקה - לוודא שמשתמשים יכולים לצפות ולערוך קבצים נתמכים ישירות מהמערכת.

מה בוצע בפועל - פתחתי קובץ txtלעריכה. בדקתי את הצגת תמונות בפורמט .jpg בדקתי תצוגת מסמכי .pdf

תוצאות הבדיקה - כל הקבצים הוצגו ונערכו בהצלחה.

בעיות שהתגלו וכיצד נפתרו - קבצי pdf גדולים נטענו לאט מאוד.
פתרון :השתמשתי בטעינה דינמית – הצגת העמודים תוך כדי טעינה.

7.הורדה מרובת קבצים כZIP-

מטרת הבדיקה - לוודא שמשתמשים יכולים לבחור מספר קבצים וליצור קובץ ZIP להורדה.

מה בוצע בפועל - סימנתי מספר קבצים ולחצנו על "הורד כ."ZIP- בדקתי את תקינות קובץ ה ZIP-שנוצר.

תוצאות הבדיקה - קובץ ה ZIP-נוצר וניתן היה לפתוח אותו.

בעיות שהתגלו וכיצד נפתרו - אם נבחרו יותר מ-100 ,MBההורדה הייתה איטית מאוד.
פתרון :בצעתי דחיסת ZIP ישירות בשרת ולא בלקוח.

<u>בדיקות נוספות:</u>

1.מחיקת קבצים ושחזור מסל המחזור

מטרת הבדיקה -לוודא שקבצים שנמחקו מועברים לתיקיית "נמחקו לאחרונה" וניתנים לשחזור או למחיקה סופית.

מה בוצע בפועל - מחקתי מספר קבצים מתיקיות שונות. בדקתי אם הם מופיעים בתיקיית "נמחקו לאחרונה." ניסיתי לשחזר קובץ ולהחזירו למיקומו המקורי. ניסיתי למחוק קובץ לצמיתות ובדקתי אם הוא נעלם ממערכת הקבצים וממסד הנתונים.

תוצאות הבדיקה -המחיקה עבדה כראוי, הקבצים הופיעו בסל המחזור, הצלחתי לשחזר אותם ולמחוק אותם סופית.

בעיות שהתגלו וכיצד נפתרו - קבצים ששוחזרו הופיעו בתיקייה הלא נכונה.
פתרון :הוספתי מעקב אחר מיקום הקובץ לפני המחיקה כדי לשחזר אותו למיקומו המקורי.

2.ניהול הרשאות משתמשים

מטרת הבדיקה -לוודא שמשתמשים יכולים לקבל ולנהל הרשאות מתאימות לקבצים ולתיקיות.

מה בוצע בפועל - יצרתי קובץ ושיתפתי אותו עם משתמש אחר עם הרשאת קריאה בלבד. המשתמש השני ניסה לערוך או למחוק את הקובץ. בדקתי האם המשתמש יכול לשנות את הרשאות השיתוף. ביטלתי את השיתוף ובדקתי אם המשתמש השני מאבד גישה.

תוצאות הבדיקה -הרשאות הגישה עבדו כראוי, המשתמש השני לא הצליח לבצע פעולות שאינן מורשות עבורו.

בעיות שהתגלו וכיצד נפתרו - כאשר הרשאת עריכה הוסרה, המשתמש השני עדיין יכל לערוך עד הרענון הבא.
פתרון :הוספתי בדיקת הרשאות בזמן אמת בעת ניסיון לשמור שינויים.

3. אימות משתמש עם מייל

מטרת הבדיקה -לוודא שאימות המשתמשים באמצעות דוא"ל מתבצע בהצלחה ומונע יצירת חשבונות מזויפים.

מה בוצע בפועל - נרשמתי למערכת עם כתובת דוא"ל חוקית. בדקתי אם מתקבל מייל עם קוד אימות. הזנתי קוד שגוי ובדקתי שהמערכת מזהה זאת. הזנתי את הקוד הנכון ואישרתי את החשבון.

תוצאות הבדיקה -מערכת האימות עבדה בהצלחה, רק קודים תקינים התקבלו.

בעיות שהתגלו וכיצד נפתרו - לעיתים קוד האימות לא נשלח מיד.
פתרון :הוספתי תהליך בדיקה מחדש ושליחה חוזרת במקרה שהמשתמש לא קיבל את הקוד לאחר 30 שניות.

4. הגבלת גודל קבצים להעלאה

מטרת הבדיקה -לוודא שהמערכת מונעת העלאת קבצים מעבר לגודל המרבי המותר.

מה בוצע בפועל - ניסיתי להעלות קובץ קטן בגודל MB.10 ניסיתי להעלות קובץ בגודל GB, 1מעבר למגבלת MB.500 בדקתי אם מתקבלת הודעה מתאימה במקרה של חריגה מהגודל המותר.

תוצאות הבדיקה -העלאת קבצים עד MB 500עבדה, ניסיון להעלות קובץ גדול יותר נחסם עם הודעה מתאימה.

בעיות שהתגלו וכיצד נפתרו - בהעלאת קובץ גדול, ההודעה לא הוצגה מיד.
פתרון :הוספתי בדיקת גודל קובץ לפני תחילת ההעלאה במקום רק לאחר סיום שליחת הנתונים.

5. ריבוי חיבורים לשרת (בדיקות עומסים)

מטרת הבדיקה -לבדוק כיצד המערכת מתפקדת תחת עומס כאשר מספר רב של משתמשים מתחברים ומעלים קבצים במקביל.

מה בוצע בפועל - פתחתי 50 חיבורים בו-זמנית באמצעות סקריפט סימולציה. כל חיבור התחבר לחשבון משתמש וביצע העלאות והורדות סימולטניות. בדקתי את זמני התגובה של השרת ואת ניצול הזיכרון והמעבד.

תוצאות הבדיקה -השרת הצליח לטפל בעד 40 חיבורים במקביל ללא האטה משמעותית.

בעיות שהתגלו וכיצד נפתרו - מעל 50 חיבורים, זמן התגובה של השרת גדל משמעותית.

מדריך למשתמש

קבצי המערכת:

צד השרת -

🗁 **server/**

│-- 🗁 **cloud/** *(ניהול אחסון ושיתוף קבצים)*

│-- 🗁 **database/** *(נתוני משתמשים, קבצים והרשאות)*

│ │-- 🗎 **database.db** *(מסד נתונים מרכזי)*

│-- 🗁 **keys/** *(מפתחות הצפנה לאבטחת התקשורת)*

│-- 🗁 **modules/** *(מודולים מרכזיים לפונקציונליות השרת)*

│ │-- 🗎 **client_requests.py** *(טיפול בבקשות הלקוח)*

│ │-- 🗎 **config_s.py** *(הגדרות שרת כגון פורטים וכתובות IP)*

│ │-- 🗎 **database_handling.py** *(ניהול הנתונים במסד הנתונים)*

│ │-- 🗎 **encrypting_s.py** *(הצפנת נתונים ותעבורה)*

│ │-- 🗎 **errors.py** *(ניהול שגיאות ותקלות במערכת)*

│ │-- 🗎 **limits.py** *(הגבלת גודל העלאה, מספר חיבורים וכו')*

│ │-- 🗎 **logger.py** *(רישום פעולות וטעויות ביומן השרת)*

│ │-- 🗎 **networking_s.py** *(ניהול תקשורת עם הלקוחות)*

│ │-- 🗎 **protocol_s.py** *(פרוטוקול תקשורת בין השרת ללקוח)*

│ │-- 🗎 **validity.py** *(אימות נתונים ובדיקות תקינות)*

│-- 🗁 **user_icons/** *(אייקונים למשתמשים)*

│-- 🗎 **server.py** *(הקובץ הראשי להפעלת השרת)*

צד הלקוח –

🗁 **client/**

│-- 🗁 **assets/** *(משאבים גרפיים כגון סמלים ותמונות)*

│-- 🗁 **cache/** *(מטמון נתונים להאצת ביצועים)*

│-- 🗁 **cookies/** *(אחסון אישורי התחברות אוטומטית)*

│-- 🗁 **gui/** *(קבצי ממשק המשתמש)*

│-- 🗁 **modules/** *(מודולים לניהול הקבצים והתקשורת)*

│ │-- 🗎 **config.py** *(הגדרות הלקוח)*

│ │-- 🗎 **dialogs.py** *(חלונות הודעה ואינטראקציה עם המשתמש)*

│ │-- 🗎 **encrypting.py** *(הצפנת נתונים בצד הלקוח)*

│ │-- 🗎 **file_send.py** *(שליחת קבצים לשרת)*

│ │-- 🗎 **file_viewer.py** *(תצוגה ועריכה של קבצים מקומיים)*

│ │-- 🗎 **gui.py** *(ניהול ממשק המשתמש)*

│ │-- 🗎 **helper.py** *(פונקציות עזר שונות)*

│ │-- 🗎 **limits.py** *(הגבלת פעולות המשתמש)*

│ │-- 🗎 **logger.py** *(רישום יומן אירועים ושגיאות בצד הלקוח)*

│ │-- 🗎 **networking.py** *(ניהול תקשורת עם השרת)*

│ │-- 🗎 **protocol.py** *(יישום פרוטוקול התקשורת עם השרת)*

│ │-- 🗎 **receive.py** *(קבלת קבצים מהשרת וניהולם בצד הלקוח)*

│-- 🗎 **client.pyw** *(הקובץ הראשי להפעלת ממשק המשתמש של הלקוח)*

התקנת המערכת:

הסביבה הנדרשת:

הסביבה הנדרשת להרצת המערכת היא מחשב בעל מערכת הפעלה ווינדוס, המערכת ניתנת לכיוון לקובץ exe אחד עם נספחי תמונות וקבצים אחרים כך שאין שום צורך בהתקנת פייתון על המחשב.

הכלים הנדרשים:

מכיוון שאפשר לשים את המערכת בתוך קובץ תוכנה אחד, אין כלי כלשהו שנדרש להרצת המערכת. אם כן רוצים להריץ את המערכת דרך קבצי py, נדרשת גרסת פייתון 3.12 או יותר, pip וכמובן הורדת הספריות החיצוניות הדרושות.

מיקומי קבצים:

לאחר כיווץ המערכת לקובץ אחד, המערכת תהיה מורכבת מקובץ הexe, תיקיית assets, cookies, gui וקובץ הlog. בצד השרת הקבצים הם כמו בעץ הקבצים.

נתונים התחלתיים:

כל משתמש חדש שרוצה להתחבר למערכת חייב לפתוח חשבון, בשביל לפתוח חשבון חובה אימייל מתפקד בשביל לקבל את קוד האימות, שם משתמש באורך 4 או יותר, וסיסמה שעונה על הדרישות: לפחות אות גדולה, מספר ואורך של 8 או יותר. לאחר הרישום יהיה ניתן להתחבר למערכת.

רשת:

מהלקוח נדרש חיבור אינטרנט תקין, ואם הוא אינו נמצא באותה הרשת הפנימית כמו השרת גם את האייפי והפורט שהשרת רץ עליו. אם הם באותה הרשת הוא יקבל זאת אוטומטית.

ארכיטקטורה נדרשת:

דרישות המערכת הפיזית הנדרשות מהלקוח הן מחשב ווינדוס 10/11, לפחות 100MB אחסון פנוי, 8GB זיכרון RAM, חיבור לרשת (פנימית או חיצונית) ומעבד בסיסי. דרישות המערכת מהשרת הן גבוהות יותר, מערכת ההפעלה זהה אך כמות האחסון, הזיכרון וחוזק המעבד תלויים בדרישות מהשרת, ככל שהחומרה תהיה חזקה יותר כך השרת יוכל לטפל ביותר לקוחות בו זמנית.

משתמשי המערכת:

אופן הפעלת המשתמש:

משתמש קצה (Regular User)

תיאור:

משתמש קצה הוא **משתמש רגיל** אשר משתמש במערכת לניהול קבצים בענן. לכל משתמש יש **חשבון אישי** עם גישה לקבצים שהוא העלה ולקבצים ששיתפו איתו.

אופן הפעלת המערכת:

1. המשתמש פותח את האפליקציה ומתחבר עם שם משתמש וסיסמה.
2. הוא מקבל תצוגת קבצים אישית, כולל אפשרות ליצירת תיקיות והעלאת קבצים חדשים.
3. ניתן להעלות, להוריד ולמחוק קבצים בהתאם להרשאות שהוגדרו.
4. אפשר לשתף קבצים עם משתמשים אחרים ולהגדיר הרשאות לכל קובץ (קריאה, כתיבה, מחיקה).
5. המשתמש יכול לנהל אחסון בענן ולשדרג את המנוי בהתאם לצרכים.

משתמש שיתופי (Shared User)

תיאור:

משתמש זה אינו הבעלים של הקבצים, אלא **משתמש שקיבל גישה** לקובץ או תיקייה ממשתמש אחר.

אופן הפעלת המערכת:

1. המשתמש מתחבר למערכת ונכנס לתצוגת הקבצים המשותפים.
2. בהתאם להרשאות שניתנו לו, הוא יכול לצפות, להוריד, לערוך או למחוק קובץ.
3. אם יש לו הרשאת "שיתוף", הוא יכול לשתף את הקובץ עם אחרים.
4. אם ההרשאות מוגבלות לקריאה בלבד, המשתמש לא יוכל לערוך או למחוק את הקובץ.

מנהל מערכת (Admin User)

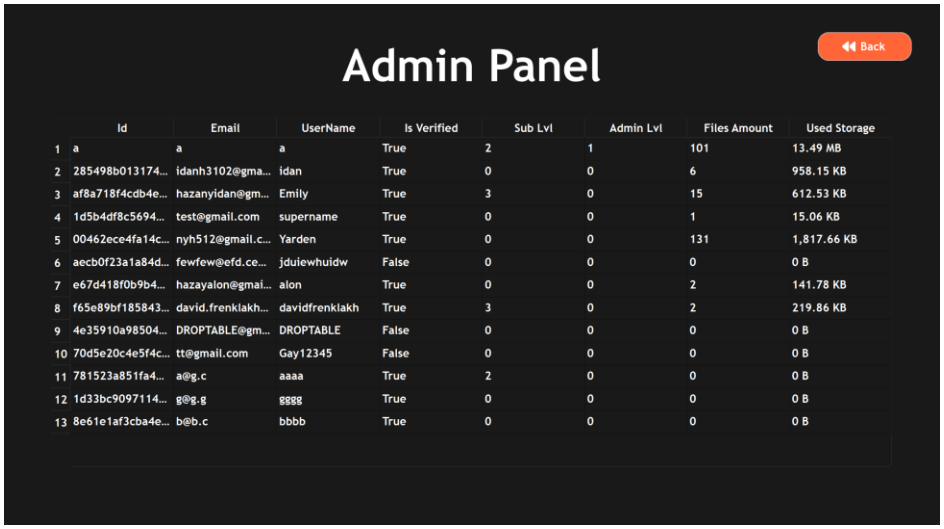
תיאור:

מנהל המערכת אחראי **על ניטור המשתמשים, בקרת נתונים ושמירה על תקינות השרת**.

אופן הפעלת המערכת:

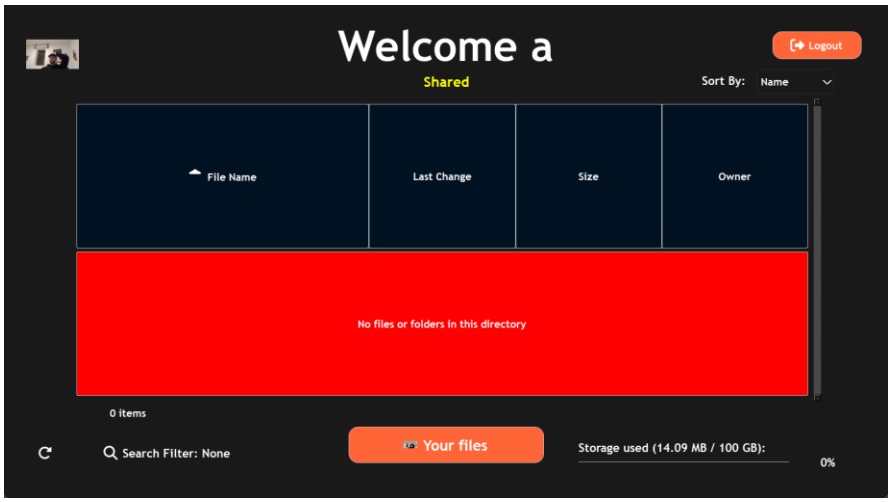
1. המנהל מתחבר עם הרשאות מיוחדות ומקבל **לוח בקרה מנהלי**.
2. ניתן לראות **רשימת משתמשים** ,לבדוק פעילות חשודה ולנהל גישות.
3. ניתן **למחוק או להשעות חשבון** של משתמש במקרה של הפרת מדיניות.
4. אפשרות לנטר **תעבורה, שימוש באחסון וניהול עומסי שרת**.
5. קבלת **התראות על שגיאות במערכת** וטיפול בתקלות.
6. גישה ל**מערכת גיבויים** לצורך שחזור נתונים שנמחקו.

<u>צילומי מסכי הפרויקט הרלוונטיים:</u>

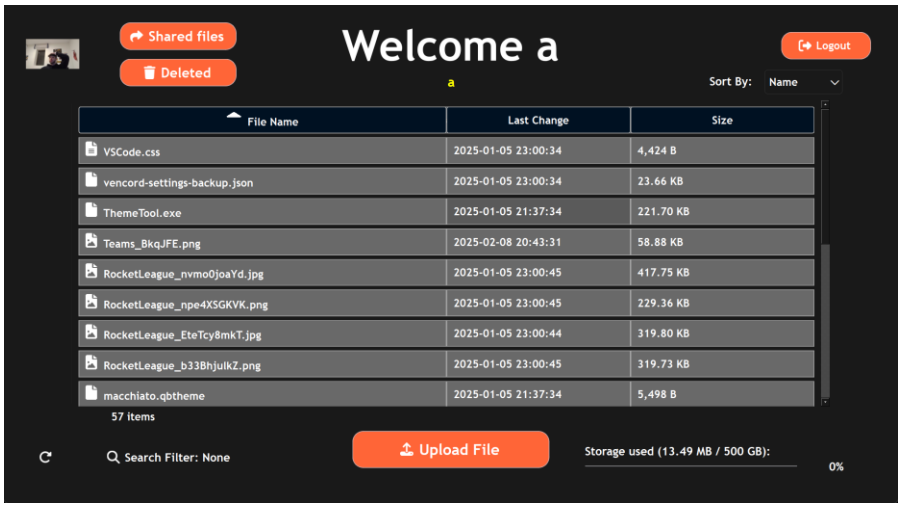


	Id	Email	UserName	is Verified	Sub Lvl	Admin Lvl	Files Amount	Used Storage
1	a	a	a	True	2	1	101	13.49 MB
2	285498b013174...	idanh3102@gma...	idan	True	0	0	6	958.15 KB
3	af8a718f4cdb4e...	hazanyidan@gm...	Emily	True	3	0	15	612.53 KB
4	1d5b4df8c5694...	test@gmail.com	supername	True	0	0	1	15.06 KB
5	00462ece4fa14c...	nyh512@gmail.c...	Yarden	True	0	0	131	1,817.66 KB
6	aecb0f23a1a84d...	fewfew@efd.ce...	jduiewhuidw	False	0	0	0	0 B
7	e67d418f0b9b4...	hazayalon@gmai...	alon	True	0	0	2	141.78 KB
8	f65e89bf185843...	david.frenklakh...	davidfrenklakh	True	3	0	2	219.86 KB
9	4e35910a98504...	DROPTABLE@gm...	DROPTABLE	False	0	0	0	0 B
10	70d5e20c4e5f4c...	tt@gmail.com	Gay12345	False	0	0	0	0 B
11	781523a851fa4...	a@g.c	aaaa	True	2	0	0	0 B
12	1d33bc9097114...	g@g.g	gggg	True	0	0	0	0 B
13	8e61e1af3cba4e...	b@b.c	bbbb	True	0	0	0	0 B

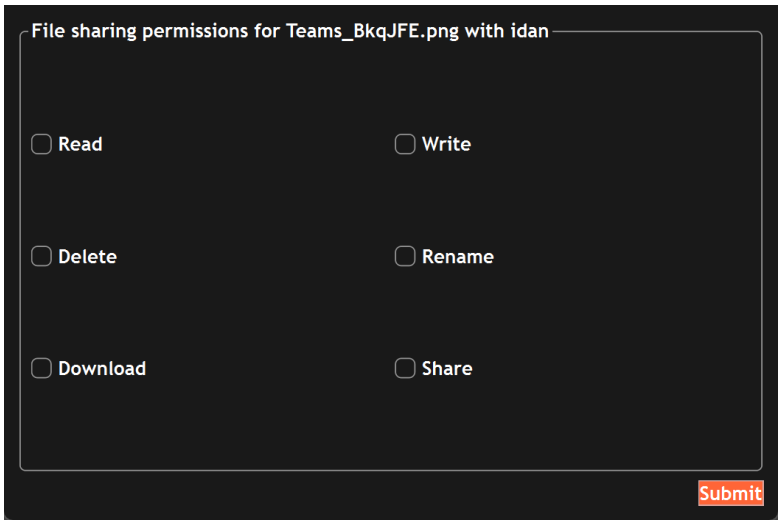
מסך מנהל המערכת בו הוא יכול לראות את הנתונים על כל החשבונות הרשומים במערכת, לחפש ביניהם ולערוך את פרטיהם.



מסך הקבצים המשותפים בו המשתמש השיתופי יכול לראות את הקבצים ששותפו איתו.



מסך הקבצים הראשי בו כל משתמש יכול לראות את הקבצים שלו, להעלות קבצים חדשים, למחוק, לשתף ולערוך.



מסך השיתוף בו משתמש יכול לשתף קובץ עם משתמש אחר ולהחליט אילו גישות יהיו לו.

רפלקציה

תהליך העבודה:

תהליך העבודה על הפרויקט

העבודה שלי על הפרויקט התבצעה בצורה סדורה ומחולקת לשלבים, כאשר בכל שלב תכננתי, פיתחתי, בדקתי והתאמתי את המערכת לפי הצורך. בתחילת הדרך הגדרתי את הדרישות העיקריות של המערכת, כולל ניהול קבצים בענן, אבטחת נתונים, תקשורת בין הלקוח לשרת ושיתוף קבצים עם משתמשים אחרים. לאחר מכן חילקתי את העבודה בין פיתוח צד השרת, צד הלקוח ומסד הנתונים.

בשלב הראשון בניתי את הארכיטקטורה של המערכת, שכללה את הגדרת מבנה הנתונים, תכנון מסד הנתונים ומערכת הקבצים, ובחירת הפרוטוקולים המתאימים לתקשורת מאובטחת. לאחר מכן מימשתי את השרת, כולל ניהול חיבורים, טיפול בבקשות המשתמשים והצפנת התקשורת. במקביל, פיתחתי את צד הלקוח עם ממשק משתמש אינטואיטיבי באמצעות PyQt6. בהמשך שילבתי מנגנוני ניהול הרשאות ושיתוף קבצים, וביצעתי בדיקות כדי לוודא את יציבות המערכת.

ההצלחות בפרויקט

אחת ההצלחות המשמעותיות שלי הייתה היכולת לבנות מערכת יציבה ומודולרית, שבה כל רכיב עובד בצורה עצמאית אך משתלב באופן מלא עם יתר הרכיבים. השימוש בפרוטוקול הצפנה מתקדם-AES 256 ו-RSA הבטיח תקשורת מאובטחת בין הלקוח לשרת.
נוסף על כך, תכננתי את מערכת ההרשאות והשיתוף כך שניתן יהיה להגדיר הרשאות מדויקות לכל קובץ ולוודא שרק משתמשים מורשים יוכלו לגשת אליו. הצלחה נוספת הייתה ניהול משאבי השרת, באמצעות מנגנונים שמגבילים משתמשים מלבצע פעולות רבות בזמן קצר וכך מונעים עומס על המערכת.

האתגרים בתהליך הפיתוח

אחד האתגרים המרכזיים שנתקלתי בהם היה ניהול התקשורת המאובטחת בין השרת ללקוח. מאחר שהמערכת דורשת אבטחת מידע ברמה גבוהה, היה צורך לממש הצפנה חזקה תוך שמירה על ביצועים טובים. התמודדתי עם כך באמצעות שילוב של RSA להחלפת מפתחות ו AES-להצפנת הנתונים בפועל.
אתגר נוסף היה התמודדות עם נפחי קבצים גדולים והבטחת העלאה והורדה יציבה. במהלך הפיתוח גיליתי קשיים כאשר קבצים גדולים הועברו בחתיכות, ולכן היה צורך לממש מנגנון העלאה חלקית (Resume Upload), שאפשר להמשיך העלאה או הורדה מהנקודה שבה היא נעצרה.

הקשיים ודרכי הפתרון

במהלך העבודה על המערכת נתקלתי במספר קשיים, ואחד הבולטים שבהם היה מנגנון ניהול קבצים מחוקים. בתחילה הקבצים נמחקו מיד, אך הדבר גרם למקרים שבהם משתמשים איבדו מידע חשוב בטעות. כדי לפתור זאת, יצרתי טבלת קבצים שנמחקו, שבה הקבצים נשמרים לתקופה של 30 יום לפני מחיקה סופית.

קושי נוסף היה התגובה האיטית של ממשק המשתמש כאשר בוצעו פעולות כבדות, כגון העלאת קובץ גדול. כדי לפתור זאת, הפעלתי משימות ברקע (Threads) בצד הלקוח, כך שהמשתמש יוכל להמשיך לעבוד ללא עיכובים בזמן העלאת קבצים לשרת.

במהלך בדיקות המערכת גיליתי ניסיונות להעמיס על השרת על ידי שליחת בקשות מרובות, ולכן הטמעתי Rate Limiting, שמונע ממשתמשים לשלוח יותר מדי בקשות בפרק זמן קצר.

תהליך הלמידה:

במהלך העבודה על הפרויקט למדתי רבות על אבטחת מידע, ניהול תקשורת בין לקוח לשרת, עבודה עם מסדי נתונים ושיפור ביצועים .הלמידה התבצעה תוך כדי יישום מעשי, פתרון בעיות ושימוש בתיעוד רלוונטי. בכל שלב בפרויקט נדרשתי להעמיק בתחומים חדשים, להבין כיצד לממש פתרונות בצורה אופטימלית, ולשפר את המערכת כך שתהיה יציבה, מהירה ומאובטחת.

ניהול תקשורת בין לקוח לשרת בפרוטוקול מותאם אישית

בתהליך הפיתוח הייתי צריך ליצור פרוטוקול מותאם אישית שיאפשר שליחת פקודות וקבלת תשובות בצורה מסודרת ומאובטחת .למדתי כיצד TCP עובד ברמת הבסיס ,איך לחלק נתונים להודעות קטנות (Packetization), ואיך להתמודד עם בעיות של אובדן נתונים או עיכובים בתקשורת .הבנתי גם כיצד לבנות מערכת ניהול בקשות יעילה ,כך שכל הודעה מעובדת בצורה אסינכרונית מבלי לעכב את שאר המערכת.

הצפנה ואבטחת מידע

נושא האבטחה היה אחד התחומים שבהם העמקתי בצורה משמעותית. למדתי כיצד לשלב הצפנה סימטרית (AES-256) וא-סימטרית (RSA), כך שהתהליך יהיה גם מאובטח וגם יעיל מבחינת ביצועים . התמודדתי עם אתגר של ניהול מפתחות הצפנה בצורה בטוחה ,כולל החלפת מפתחות דינמית בתחילת החיבור. בנוסף, למדתי כיצד להשתמש בחתימות דיגיטליות (HMAC) כדי להבטיח שהנתונים שנשלחים לא עברו שינוי.

ניהול מסדי נתונים ב SQLite-והבטחת שלמות הנתונים

עבודה עם מסד הנתונים דרשה ממני להבין כיצד לשמור על ביצועים טובים גם כאשר יש כמויות גדולות של נתונים .למדתי כיצד לייעל שאילתות ,להשתמש באינדקסים לשיפור מהירות החיפוש, ולהבטיח שלמות נתונים באמצעות טרנזקציות .בנוסף, חקרתי כיצד למנוע SQL Injection על ידי שימוש ב-Prepared Statements, וכיצד לנהל גישה מבוקרת לטבלאות רגישות כמו משתמשים והרשאות.

העלאת והורדת קבצים גדולים ללא עומס על השרת

כשהתחלתי לממש את מערכת העלאת הקבצים, הבנתי שלא ניתן לשלוח קבצים גדולים במכה אחת, כי זה גורם לעומס על השרת ולבעיות בזיכרון. למדתי כיצד לחלק קובץ לחלקים קטנים (Chunking), ולשלוח כל חלק בנפרד, תוך שימוש במנגנון חידוש העלאה (Resume Upload) למקרה שהחיבור מתנתק. זה איפשר למערכת להמשיך הורדות והעלאות מנקודת העצירה, מה שמשפר את חוויית המשתמש ומונע אובדן נתונים.

שימוש במנגנוני ניהול עומסים ומניעת ניצול משאבי מערכת

בשלב הבדיקות הבנתי שמשתמשים יכולים לשלוח בקשות רבות מאוד בזמן קצר, מה שעלול להעמיס על השרת. כדי להתמודד עם זה, למדתי כיצד להטמיע Rate Limiting, כלומר, הגבלה על כמות הבקשות שכל משתמש יכול לשלוח בפרק זמן מסוים. בנוסף, למדתי כיצד לזהות בקשות חריגות או מתקפות DDoS, ולחסום אותן לפני שהן פוגעות במערכת.

עבודה עם ממשק גרפי PyQt6 והתאמתו למערכת דינמית

למדתי כיצד לעבוד עם PyQt6 כדי לבנות ממשק משתמש מודרני ודינמי, שתומך בתהליכי רקע (Threads) מבלי להקפיא את ה-UI. התמודדתי עם אתגרים כמו עדכון אוטומטי של תצוגת הקבצים בזמן אמת, והבנתי כיצד להפריד בין הלוגיקה של הממשק ללוגיקה של העיבוד כדי לשמור על קוד קריא ותחזוקתי.

כלים נלקחים להמשך:

לאורך העבודה על הפרויקט למדתי טכנולוגיות חדשות ,שיטות עבודה מסודרות ,ופתרונות טכניים מתקדמים שיכולים לשמש אותי בפרויקטים עתידיים. ישנם מספר כלים ומיומנויות משמעותיים שאני לוקח להמשך, הן ברמה הטכנית והן ברמה הפרקטית של ניהול פרויקטים.

ניהול תקשורת מאובטחת בין לקוח לשרת

במהלך הפרויקט רכשתי הבנה מעמיקה בפרוטוקולי תקשורת ,במיוחד בנוגע להצפנה, שלמות נתונים ואבטחה בזמן אמת .הכלים המרכזיים שאקח איתי:

- שימוש ב RSA-לחילופי מפתחות ו AES-256-להצפנת נתונים בצורה יעילה
- מניעת מתקפות Man-in-the-Middle באמצעות חתימות דיגיטליות ואימות נתונים
- תכנון פרוטוקול תקשורת מותאם אישית שמבוסס על TCP ומסוגל להתמודד עם עומסים ושגיאות

עבודה עם מסדי נתונים ואופטימיזציה של שאילתות

עבדתי עם SQLite ולמדתי כיצד לייעל ביצועים של שאילתות ולנהל מסד נתונים בצורה מאובטחת ויעילה .הכלים המרכזיים שאמשיך להשתמש בהם:

- שימוש באינדקסים לשיפור מהירות חיפושי נתונים
- הגבלת גישה למסד הנתונים באמצעות הרשאות נכונות ושימוש ב Prepared Statements-למניעתSQL Injection

עבודה עם קבצים גדולים ושיפור ביצועים בהעלאות והורדות

למדתי כיצד להתמודד עם העברת קבצים גדולים בצורה יעילה ,תוך שמירה על חוויית משתמש טובה וללא עומס על השרת .הכלים המרכזיים להמשך:

- חלוקת קובץ לחלקים (Chunking) כדי לאפשר העברה בשלבים ולשמור על מהירות יציבה

- מנגנון חידוש העלאה/הורדה (Resume Upload/Download) למניעת אובדן נתונים במקרה של ניתוק
- ניהול עומסים בשרת בזמן העלאות גדולות ,כך שמשתמש אחד לא יגרום לקריסת המערכת

ניהול עומסים והגבלת משאבי מערכת

למדתי כיצד להתמודד עם שימוש אינטנסיבי במערכת ,ולמנוע ניצול יתר של המשאבים. הכלים להמשך:

- Rate Limiting להגבלת מספר הבקשות שמשתמש יכול לשלוח בפרק זמן מסוים
- מעקב אחר תעבורה חשודה וזיהוי בקשות לא חוקיות לפני שהן מגיעות לשירותים הקריטיים
- איזון עומסים (Load Balancing) וניהול חיבורים מרובים כך שהשרת יתמודד טוב יותר עם משתמשים רבים

פיתוח ממשק משתמש דינמי עם PyQt6

למדתי כיצד לפתח ממשק משתמש שמתעדכן בזמן אמת, תומך בתהליכי רקע, ומאפשר עבודה חלקה . הכלים המרכזיים:

- שימוש ב Threads-למניעת הקפאת הממשק בזמן פעולות כבדות כמו העלאה והורדה
- עדכון דינמי של תצוגת הנתונים ללא צורך ברענון ידני של המשתמש
- שימוש בחלונות דיאלוג חכמים שמאפשרים אישור פעולות וניהול קבצים בצורה נוחה

פיתוח קוד מודולרי ותחזוקתי

במהלך העבודה הקפדתי על עיצוב קוד נקי, מחולק למודולים, קל לתחזוקה ולשדרוג עתידי .הכלים שאמשיך להשתמש בהם:

- הפרדת לוגיקה עסקית מממשק משתמש ,כך שהמערכת תהיה גמישה לשינויים עתידיים
- שימוש במחלקות ומודולים עצמאיים ,שמאפשרים לעבוד על רכיבים שונים בנפרד
- תיעוד מסודר של קוד ולוגים ,כדי להקל על ניתוח תקלות ושיפורים

תובנות מהתהליך:

במהלך העבודה על הפרויקט הגעתי למספר תובנות חשובות לגבי ניהול פיתוח מערכת מורכבת, למידה עצמאית ושיתוף פעולה עם אחרים.

אחת התובנות המרכזיות היא החשיבות של תכנון מסודר לפני תחילת הכתיבה .בתחילת הדרך ניסיתי ליישם פתרונות תוך כדי תנועה, אך עם הזמן הבנתי שעדיף לשרטט את הארכיטקטורה מראש, לתכנן את מבנה מסד הנתונים ולהגדיר את הפרוטוקול בצורה מדויקת .זה חסך הרבה עבודה חוזרת ותקלות בהמשך הדרך.

בנוסף, נוכחתי לדעת שלמידה עצמאית היא מיומנות קריטית ,במיוחד כאשר עובדים עם טכנולוגיות שלא הכרתי לעומק. נאלצתי לקרוא תיעוד רשמי, מאמרים ופורומים טכניים ,ולפעמים אפילו לנסות כמה גישות שונות עד שמצאתי את הדרך האופטימלית ליישום פתרון מסוים.

שיתוף מידע ולמידת עמיתים הוכחו ככלים משמעותיים לשיפור העבודה. כאשר נתקלתי בבעיות מסוימות ,לשאול אנשים עם ניסיון, לבדוק פתרונות קיימים ולשתף תובנות עם אחרים חסכו לי זמן רב ואפשרו לי למצוא פתרונות טובים יותר.

גם קבלת עזרה ממומחים הייתה חשובה בשלבים מסוימים, בעיקר בנושאי אבטחת מידע והצפנה . פידבק מאנשים מנוסים יותר עזר לי להבין כיצד לממש תקשורת מוצפנת ביעילות ולמנוע בעיות אבטחה שעלולות להופיע במערכת מבוססת רשת.

למדתי גם שהיכולת להתמודד עם בעיות ולהמשיך לחפש פתרונות היא חיונית. לעיתים, שגיאות וקשיים נראו בלתי פתירים, אבל כשלקחתי צעד אחורה, ניסיתי להבין את הבעיה מהבסיס, וביצעתי בדיקות שיטתיות – הצלחתי להגיע לפתרון.

בסופו של דבר, תהליך הפיתוח היה חוויית למידה משמעותית ,הן מבחינה טכנית והן מבחינת שיטות עבודה נכונות, שיתוף ידע ושיפור יכולות פתרון בעיות.

ראייה לאחור:

במהלך העבודה על הפרויקט הצלחתי לפתח מערכת יציבה, מאובטחת ויעילה, אך לאחר שסיימתי וניתחתי את תהליך העבודה, ישנם מספר דברים שהייתי עושה בצורה שונה אם הייתי מתחיל מחדש. חלק מהשינויים נובעים מתוך תובנות שלמדתי תוך כדי הפיתוח ,וחלקם קשורים לשיפור המבנה והביצועים של המערכת.

מבנה הפרויקט – הפרדה טובה יותר בין רכיבי השרת

בשלב הפיתוח הראשוני, חלק מהמודולים בצד השרת כללו תלות הדדית גבוהה מדי ,מה שגרם לקושי בהפרדת רכיבים לשינויים עתידיים. אם הייתי מתחיל מחדש, הייתי מפריד את השרת למודולים נפרדים בצורה ברורה יותר –למשל, מודול עצמאי לטיפול בקבצים, מודול עצמאי לניהול הרשאות, ומודול תקשורת שמבודד את כל ההיבטים של שליחת וקליטת נתונים. זה היה מקל על תיקונים עתידיים ומשפר את קריאות הקוד.

שימוש בפרוטוקול תקשורת קיים במקום פיתוח מותאם אישית

במקום לפתח פרוטוקול תקשורת מותאם אישית לחלוטין ,הייתי שוקל להשתמש בפרוטוקול סטנדרטי יותר, כמוWebSocket או HTTP עם REST API, לפחות לחלק מהפעולות. היתרון של פתרונות מוכנים הוא יכולת להשתמש בכלים קיימים לניהול עומסים ואבטחה ,במקום לטפל בכל הפרטים הטכניים של יצירת פרוטוקול מאפס.

שיפור אופן ניהול ההרשאות ושיתוף הקבצים

מערכת ההרשאות שיצרתי עובדת בצורה יעילה, אך היא מנוהלת דרך מסד הנתונים בלבד ,וכל בדיקת גישה דורשת שאילתות מרובות .בראייה לאחור, הייתי משתמש במטמון (Caching) עבור הרשאות שנבדקות לעיתים קרובות. כך ניתן לצמצם את מספר הפניות למסד הנתונים ולשפר את ביצועי המערכת, במיוחד במקרים של משתמשים עם הרבה קבצים משותפים.

הפרדת תהליכים כבדים לשירותים חיצוניים

כדי לשפר את היעילות, הייתי שוקל להפעיל חלק מהתהליכים הכבדים כרכיבים נפרדים ,כמו שירות העלאת קבצים עצמאי ,שפועל בנפרד מהשרת המרכזי. כך השרת לא יטפל ישירות בכל קובץ שמועלה, אלא יעביר את הבקשה לשירות צדדי שמתמקד בהעלאה ובשיפור ביצועים.

שימוש בתשתית מסד נתונים מתקדמת יותר

SQLite היה בחירה טובה לפיתוח הראשוני ,אך אם המערכת הייתה צריכה לתמוך ביותר משתמשים ובקצבים גבוהים ,הייתי שוקל לעבור למסד נתונים מתקדם יותר, כמו PostgreSQL. היתרונות של מעבר כזה כוללים ניהול טוב יותר של חיבורים במקביל, ביצועים משופרים בשאילתות מסובכות , ואפשרויות מתקדמות לניהול הרשאות וטרנזקציות.

שיפור חוויית המשתמש והיעילות של ממשק הלקוח

בצד הלקוח, הייתי משפר את חוויית המשתמש מבחינת תגובתיות ,(Responsiveness) במיוחד בפעולות כמו טעינת קבצים מרובים בו זמנית. למשל, שימוש באנימציות והודעות סטטוס יותר ברורות יכול היה לשפר את חוויית המשתמש. כמו כן, אם הייתי מתחיל מחדש, הייתי משתמש ב QThread- בצורה רחבה יותר ,כדי להבטיח שכל הפעולות הכבדות יפעלו במקביל לממשק המשתמש ,בלי לגרום להקפאות זמניות.

איך הייתי משפר:

אם היו לי משאבים נוספים, הייתי משפר את ביצועי המערכת, מוסיף תכונות מתקדמות ומשפר את חוויית המשתמש.

הייתי משתמש במסד נתונים מבוזר כמו PostgreSQL או MongoDB ,כדי להתמודד עם עומסים גבוהים ולשפר את מהירות הביצוע של שאילתות מורכבות .בנוסף, הייתי מפעיל שרתים מבוזרים (Load Balancing) כדי לחלק את העומס בצורה טובה יותר בין כמה שרתים ולאפשר תמיכה במספר רב של משתמשים במקביל.

בצד התקשורת, הייתי עובר לשימוש ב WebSocket-במקום TCP מותאם אישית, כדי לאפשר תקשורת רציפה ודינמית יותר בין הלקוח לשרת, מה שהיה משפר עדכוני נתונים בזמן אמת.

בצד הלקוח, הייתי משפר את חוויית המשתמש על ידי שיפור עיצוב הממשק ,הוספת אנימציות אינטראקטיביות ,ושימוש בתהליכי רקע משופרים (QThread) כדי שהפעולות יפעלו בצורה חלקה יותר.

מבחינת אבטחה, הייתי משקיע בשימוש באימות דו-שלבי (2FA) מבוסס אפליקציה ,ולא רק דרך מייל, כדי לחזק את ההגנה על חשבונות המשתמשים.

לבסוף, אם היו לי משאבים נרחבים, הייתי מפתח אפליקציה למובייל כדי לאפשר למשתמשים לנהל את הקבצים שלהם גם מהטלפון ,ובכך להרחיב את הגישה למערכת.

שאלות חקר עצמי:

1. כיצד ניהלתי את הזמן שלי במהלך הפיתוח, והאם היה אפשר לשפר זאת?
2. אילו בעיות טכניות נתקלתי בהן, וכיצד פתרתי אותן?
3. איזה חלק בפרויקט היה הכי מאתגר עבורי, ולמה?
4. האם התכנון הראשוני של הפרויקט היה נכון, או שהייתי צריך לשנות אותו תוך כדי העבודה?
5. מה למדתי על תהליכי פיתוח תוכנה במהלך הפרויקט?
6. אם הייתי מתחיל את הפרויקט מחדש, מה הייתי עושה אחרת?
7. כיצד ניתן לשפר את חוויית המשתמש ומה ניתן ללמוד מפרויקטים דומים?
8. איך העבודה על הפרויקט חיזקה את היכולות הטכנולוגיות שלי?
9. כיצד ניתן לשדרג את הפרויקט בעתיד מבחינת ביצועים ואבטחה?
10. מה למדתי על עבודה עם מערכות מבוזרות ותקשורת מאובטחת?

תודות:

אני רוצה להוקיר תודה לאופיר המורה שלי שעזר לי עם בעיות ותמך בי לאורך כל הדרך, לחברים הטובים שעזרו לי להמשיך גם כשהיה קשה ותמיד עזרו לי עם בעיות שנתקלתי בהם, ובמיוחד לדוד פרנקלך, ניר יוסף, ענבר באלין, עידן טל, ויואב שבתאי וכמובן לחברה שלי אמילי, שבלעדיה לא הייתי מצליח להגיע לתוצר כזה.

ביבליוגרפיה

AES Encryption. (n.d.). *Advanced Encryption Standard (AES) - Overview & Implementation*. Retrieved from https://www.ibm.com/docs/en/linux-on-systems?topic=security-advanced-encryption-standard-aes

Huss, M. (2021). *Python for the Busy Developer: Fast, Scalable, and Secure Web Apps* (1st ed.). Apress. Retrieved from https://link.springer.com/book/10.1007/978-1-4842-6622-6

PyQt Documentation. (n.d.). *PyQt6 Reference Guide*. Retrieved from https://www.riverbankcomputing.com/static/Docs/PyQt6/

RSA Encryption. (n.d.). *RSA Algorithm Explained with Examples*. Retrieved from https://www.geeksforgeeks.org/rsa-algorithm-cryptography

SQLite. (n.d.). *SQLite Documentation*. Retrieved from https://www.sqlite.org/docs.html

Van Rossum, G., & Drake, F. L. (2009). *The Python Language Reference Manual*. Network Theory Ltd. Retrieved from https://docs.python.org/3/reference/

Various authors. (n.d.). *Python Documentation*. Retrieved from https://docs.python.org/3/

נספחים

מסמך הקוד המלא הכולל את כל קוד הפרויקט, המחלקות והערות.

סדר הקבצים:

client.pyw, client_requests.py, config.py, config_s.py, database_handling.py, dialogs.py, encrypting.py, encrypting_s.py, errors.py, file_send.py, file_viewer.py, gui.py, helper.py, limits.py, limits_s.py, logger.py, logger_s.py, networking.py, networking_s.py, protocol.py, protocol_s.py, receive.py, server.py, update_release.py, validity.py