

SyncSphere



פרוייקט גמר בהגנת סייבר ומערכות הפעלה

אייל פיקז

שם המורה - צבי שירן

תאריך הגשה - 29.04.2025



מבוא:	3
"יזום":	3
הגדרת הלקוח:	3
הגדרת יעדים:	4
סקירת פתרונות קיימים:	4
סקירת טכנולוגיית הפרויקט:	5
תיחום הפרויקט:	5
פירוט תיאור המערכת:	5
תיאור המערכת:	5
פירוט היכולות של המערכת:	6
פירוט הבדיקות שמתוכננות לפרויקט:	6
תכנון וניהול לוח זמנים:	9
ניהול הסיכונים בפרויקט:	10
תיאור תחום הידע:	11
צד לקוח:	11
מבנה/ארכיטקטורה של הפרויקט:	17
תיאור הארכיטקטורה של המערכת המוצעת:	17
תיאור החומרה:	17
תיאור הטכנולוגיה הרלוונטית:	18
תיאור זרימת המידע במערכת:	19
תיאור האלגוריתמים במערכת:	29
תיאור סביבת הפיתוח:	30
תיאור פרוטוקול התקשורת:	30
תיאור מסכי המערכת:	31
תיאור מבני הנתונים:	35
סקירת חולשות ואיומים:	37
מימוש הפרויקט:	38
סקירת המודולים, המחלקות וקשרי הגומלין ביניהם:	38
סקירת המודולים המיובאים:	38
מודולים שפיתחתי:	38
בעיות אלגוריתמיות - פירוט:	43
תיאור הבדיקות במערכת:	50
מדריך למשתמש:	55
פירוט כלל קבצי המערכת:	55
התקנת המערכת:	57
הפעלת המערכת:	57
סיכום אישי - רפלקציה:	1
ביבליוגרפיה:	1
נספחים:	1
קוד הפרויקט:	1
app.py :	1
auth.py:	1
config.py:	1
file_management_routes.py:	1

file_management.py:.....	1
friend_management_routes.py:.....	1
friend_management.py:.....	1
models.py:.....	1
sync.py:.....	1
grand_server.py:.....	1
templates/dashboard.html:.....	1
templates/friends_files.html:.....	1
templates/friends_list.html:.....	1
templates/incoming_requests.html:.....	1
templates/login.html:.....	1
templates/register.html:.....	1
static/css/auth.css:.....	1
static/css/dashboard.css:.....	1
של הפרויקט: github-קישור לעמוד ה	1

מבוא:

ייזום:

הפרוייקט שבחרתי לבצע הוא מערכת לגיבוי קבצים, שמירתם ושיתופם עם משתמשים אחרים במערכת. המוצר מאפשר למשתמש לאחסן קבצים על גבי שרת מרוחק ולהורידם במידת הצורך. המערכת מורכבת ממספר שרתים אזוריים (שאינם המשתמש יוצר אינטראקציה) ושרת אחד מרכזי האחראי על סנכרון הגיבויים בין השרתים האזוריים. בחרתי בפרוייקט זה כיוון שפרוייקט זה שם דגש על עבודה עם מערכת הקבצים של המחשב, תחום שאותי אישית מעניין ושאיני רוצה לעסוק בו. האתגרים שאני צופה בפרוייקט הם:

1. מימוש ממשק המשתמש (עליו ארחיב בהמשך).
2. ניהול עבודה עם מסד הנתונים בצורה חכמה ויעילה.
3. ניצול מערכת הקבצים בצורה מקסימלית ועבודה כמה שיותר נוחה.
4. ניהול שיתוף מידע בין שרתים אזוריים
5. מחשבה על בעיות - כגון, מה קורה במידה והשרת קורס.
6. ניהול הסנכרון בין השרתים האזוריים כך שפעולה המתבצעת בשרת אזורי תתבצע גם בשרת השרתים.
7. למידה עצמית של הנושאים הרלוונטיים.

הגדרת הלקוח:

המערכת מיועדת לאנשים שמחפשים פתרון מאובטח וידידותי למשתמש לניהול גיבויי קבצים ושיתופם עם משתמשים ספציפיים. המערכת מותאמת במיוחד לשיתוף קבצים בין שני אנשים שונים. אפשר לפעול עם המערכת ביותר מכך, אך פחות מומלץ לעשות זאת והמערכת אינה פועלת בצורה

אידיאלית. בנוסף לכך, המערכת יכולה להוות פתרון כסביבת עבודה לחברה/צוות שרוצה לחלוק בין חברי הצוות השונים.

הגדרת יעדים:

היעדים המרכזיים שהצבתי לעצמי בתחילת הפרויקט הם:

1. על המערכת לאפשר גיבוי של קבצים על שרת מרוחק, בנוסף למספר פעולות בסיסיות אחרות.
2. על המערכת לספק אפשרות להוספת חברים וצפייה בקבציהם.
3. על המערכת לספק פתרון למקרים שבו שרת של המערכת קורס/לא מתפקד.
4. פשטות - על המערכת להיות ידידותית למשתמש ופשוטה להבנה כך שגם אנשים שאינם משתמשים בטכנולוגיה על בסיס קבוע יצליחו להשתמש בצורה קלה יחסית.
5. אבטחה - על המערכת לאחסן את הקבצים של המשתמש בצורה בטוחה ומוצפנת כך שלא תתאפשר גישה לקבצים אלו. בנוסף לכך, על המערכת לשמור נתונים רגישים כגון סיסמאות בצורה מאובטחת שבמידה ונתונים אלו יודלפו, נתונים אלו לא יפגעו מהדלפה כזו.
6. זמינות - על המערכת להיות נגישה בכל זמן שהוא ובכל מקום שהוא, ואף במקרה של קריסות של השרתים/בעיות אחרות לאפשר גישה למערכת.

היעדים האופציונליים שהצבתי לעצמי בתחילת הפרויקט הם:

1. הוספת אפשרות לנהל כמה גרסאות של קובץ אחד.
2. הוספת אפשרות לשליחת הודעות בין חברים.
3. אסתטיות - על המערכת להיות ידידותית לעין.

בעיות תועלות וחסרונות:

הבעיה שהמערכת רוצה לספק פתרון היא אפשרות לגיבוי קבצים והעברה שלהם ממחשב למחשב במידת הצורך. בניגוד לשירותי גיבוי קבצים אחרים, על המערכת לספק אפשרות גם לשתף קבצים אלו עם משתמשים אחרים בצורה פשוטה וידידותית למשתמש.

1. על המערכת לספק פתרון פשוט לגיבוי קבצים.
2. על המערכת לספק אפשרות לשתף גישה לקבצים הנשמרים למשתמשים אחרים, ובמידת הצורך לספק גישה רק לחלקם.
3. על המערכת לסנכרן עם עצמה את השינויים המתבצעים בכל שרת ועדכון מסד הנתונים במקרה של כל פעולה מרכזית במערכת(עליהן ארחיב בהמשך).
4. על המערכת לשמור על קבצי ונתוני המשתמש בצורה מאובטחת.

סקירת פתרונות קיימים:

פתרונות קיימים הדומים לפרויקט שלי הם:

1. כלי של גוגל לשמירת קבצים ושיתופם עם אנשים אחרים במידת הצורך. - Google drive
השוני המרכזי בין הפרויקט שלי לכלי זה הוא שכלי זה מבוסס יותר לעבודה עם מוצרים של

גוגל ומותאם יותר לשירותי גוגל(כגון מסמכים של גוגל, ג'מייל ועוד) בעוד שהפרוייקט שלי עומד בפני עצמו ומספק פתרונות כלליים גם לקבצי מערכת שאינם קשורים לגוגל.

2. One drive - השוני - המאפשר העלאת, הורדת קבצים ושיתופם עם משתמשים אחרים. המרכזי בין הפרוייקט שלי לכלי זה הוא שכלי זה שם דגש על עבודה עם המערכת ווינדוס ועם הכלים של מיקרוסופט אופיס. בניגוד לכך, הפרוייקט שלי אינו ממוקד על מערכת הפעלה ספציפית והמשתמש לא באמת יחווה הבדל מרכזי אם הוא ייגש לפרוייקט שלי ממערכת (macos) הפעלה אחרת(כגון מערכות מבוססות לינוקס או).
3. github - השוני העיקרי - כלי המבוסס על גיט לשמירת קבצים וניהול גרסאות באמצעות גיט. הפרוייקט שלי אינו ממוקד על מערכת הפעלה ספציפית והמשתמש לא באמת יחווה הבדל מרכזי אם הוא ייגש לפרוייקט שלי ממערכת (macos) הפעלה אחרת(כגון מערכות מבוססות לינוקס או).

ניתן לראות שרוב התחלופות הפופולריות מתמקדות בנישה מסוימת. בניגוד לכך, הפרוייקט שלי מותאם לכל הנישות השונות במידה שווה. אמנם הפרוייקט שלי אינו ברמה גבוהה בכל נישה כגון התחלופות שהצעתי פה, אך הפרוייקט שלי עשוי להיות אלטרנטיבה לפתרונות אלו בכלי אחד.

סקירת טכנולוגיית הפרוייקט:

הפרוייקט שלי עוסק בטכנולוגיה קיימת. הטכנולוגיות העיקריות שהפרוייקט שלי עוסק בהן הן ירחבו בהמשך.

לפרוייקט ישנן מספר הגבלות שצריך לקחת בחשבון על מנת להשתמש בפרוייקט:

1. על המחשב להיות בעל חיבור לאינטרנט.
2. למרות הפשטות היחסית של הפרוייקט, מומלץ לעיין במדריך למשתמש.

תיאור הפרוייקט:

תוכנה:

1. כתיבה בשפה פייתון
2. כתיבה בשפות html, css על מנת לעבוד עם ממשק משתמש.
3. הצפנות.

רשתות:

4. תקשורת באמצעות סוקטים(שרת מרובה לקוחות).

אחר:

5. עבודה עם מסד נתונים בשימוש sqlite3
6. עבודה עם מערכת הקבצים של המחשב.

פירוט תיאור המערכת:

תיאור המערכת:

המערכת מורכבת ממספר חלקים עליהם אפרט מיד:

1. שרת אזורי - שרת שאיתו המשתמש מבצע אינטראקציה. השרת האזורי מאחסן אצלו את בנוסף לכך, השרת האזורי הוא שמספק את ממשק המשתמש
2. ממשק משתמש - בחרתי לבנות אתר שיספק כממשק משתמש. בהתחלה חשבתי להשתמש באפליקציה על שולחן העבודה, אך לאתר יש יתרונות שיותר מתאימים לפרוייקט מסוג

זה(לדוגמה, ניתן לגשת ממכשירים שונים וניתן לגשת בקלות ללא הורדה). ממשק המשתמש נבנה באמצעות הספרייה flask בפייתון באמצעות דפי html ו-css. במסד נתונים - מאחסן נתונים שבהם השרת האזורי משתמש. בין היתר, הנתונים שנשמרים במסד הנתונים הם:

- פרטים על קבצי המשתמש(כגון גודל הקובץ, שם הקובץ וכו').
 - פרטי משתמש(כגון שם המשתמש, סיסמה, אימייל וכו').
 - קשרים בין משתמשים(בקשות חברות, חברויות וכו').
4. שרת מרכזי - שרת מרכזי שכל השרתים האזוריים מתחברים אליו. שרת זה אחראי לאסוף כל 5 דקות את כל השינויים שנעשו בשרתים האזוריים בזמן הזה ולסנכרן את שאר השרתים האזוריים(לדוג', במידה ומעלים לשרת אזורי קובץ, השרת האזורי שולח זאת לשרת המרכזי שדואג לשלוח את זה לכל שאר השרתים האזוריים שמעדכנים זאת אצלם). שרת זה פותר 2 בעיות עיקריות:
- אין תלות עכשיו בשרת אזורי אחד(במידה ושרת אזורי קורס, אפשר להתחבר מחדש לשרת אחר ולבצע עליו פעולות שונות).
 - הזמינות של המערכת גדלה(ניתן לפרוס שרתים אזוריים במקומות שונים).

כל הרכיבים יחדיו מרכיבים מערכת שמספקת אפשרות גיבוי וניהול קבצים בכמה שפחות עבודה מצד המשתמש ושיתוף קבצים בין משתמשים.

פירוט היכולות של המערכת:

היכולות העיקריות של המערכת הן:

- התחברות והרשמה למערכת
- הוספת קובץ לשרת.
- הורדת קובץ מהשרת למכשיר שממנו מחוברים למערכת.
- מחיקת קובץ מהשרת.
- שינוי הרשאות גישה לקובץ(גישה רק למשתמש, גישה לחברים או גישה ציבורית).
- הוספת חבר.
- מחיקת חבר.

פירוט הבדיקות שמתוכננות לפרויקט:

מספר הבדיקה	שם הבדיקה	מה הבדיקה אמורה לבדוק	איך מתכננים לבדוק
1	התחברות והרשמה למערכת	האם ניתן להתחבר ולהירשם למערכת	הרשמה למערכת ולאחר מכן נסיון התחברות

2	טיפול בלקוח אחד	האם המערכת מסוגלת לטפל בפעולות הבסיסיות של לקוח אחד	ביצוע הפעולות הבאות - העלאת קובץ, מחיקת קובץ, שינוי הרשאת קובץ, הורדת קובץ, שליחת בקשת חברות, אישור בקשה וצפייה בקבצים של חבר
3	התחברות ממכשיר חיצוני	האם המערכת מסוגלת לטפל בלקוח שאינו מחובר על אותו מכשיר	ביצוע אותן פעולות על גבי מחשב שונה כאשר השרת רץ על אותו מכשיר
4	טיפול במסד הנתונים ומערכת הקבצים	האם השינויים שנעשו נשמרים במסד הנתונים ומערכת הקבצים	לאחר ביצוע הפעולות, אסגור את השרת ואבדוק אם הקבצים הרלוונטיים + הפעולות נשמרו
5	הצפנת המערכת	האם המערכת מסוגלת להצפין את התקשורת	הסנפה באמצעות wireshark של תעבורת המערכת
6	נסיונות	האם המערכת	אנסה להיכנס

	התחברות מתמשכים	מסוגלת להתמודד עם bruteforce	למערכת מספר פעמים ברציפות ואראה אם ישנה חסימה לאחר מספר נסיונות מסוים
7	סנכרון שינויים בין שרתים אזוריים	האם המערכת מצליחה לסנכרן שינויים בין מספר שרתים אזוריים הפועלים בו זמנית	אריץ שני שרתים אזוריים משני מכשירים שונים, אבצע שינוי באחד מהם ואראה אם הוא מתעדכן בשני
8	sql injection	האם המערכת מוגנת מ-sql injection	באמצעות כלי בשם sqlmap, אריץ שאליות sql זדוניות ואראה אם המערכת מוגנת מכך
9	קריסה של השרת המרכזי	האם השרתים האזוריים מסוגלים לתפקד במצב שבו השרת המרכזי קורס	אריץ את השרתים האזוריים ואכבה את השרת המרכזי, ולאחר מכן אבצע פעולות על מנת לראות אם השרת האזורי מתפקד

ביצוע מספר שינויים בשרת אזורי, ולאחר זמן מה הפעלה של שרת אזורי חדש. אראה אם השינויים שהתרחשו בשרת האזורי הוטמעו גם בשרת החדש	האם שרת אזורי שהתחבר באיחור/אחרי קריסה מסוגל להשלים את השינויים שהתרחשו בזמן זה	התחברות מאוחרת של שרת אזורי	10
---	--	------------------------------------	-----------

תכנון וניהול לוח זמנים:

תחילה התחלתי ממסמכים שונים המאפיינים את הפרויקט, דנים בחלק מן הבחירות שבהן ומסבירים בחירות שונות בארכיטקטורת הפרויקט.

המסמכים שבניתי הם:

- מסמך ייזום
- מסמך אפיון
- מסמך ניתוח ועיצוב

לאחר מסמכים אלו, התחלתי לעבוד על הפרויקט עצמו. בתחילת העבודה, השקעתי זמן רב במחשבה על מבנה הפרויקט עצמו. חשבתי בעיקר על מספר דרכים שונות לממש את העבודה עם מערכת הקבצים עצמה(עליה ארחיב בהמשך התיק). לאחר מכן, התחלתי עם הקוד של הפרויקט. תחילה התחלתי בפיתוח הלוגיקה של השרתים האזוריים, משם עברתי לממשק המשתמש וסיימתי בשרת המרכזי ובגיבוי בין השרתים. בתחילת הפרויקט, תכננתי להקצות פחות זמן בפועל ממה שהקצתי לבניית הממשק הגרפי, דבר שהוביל לקצת פחות זמן לעבוד על ההמשך לאחר מכן. חוץ מכך, אני מרגיש שפעלתי יחסית בצורה ללוח הזמנים שתכננתי בהתחלה ושבסך הכל עמדתי בו לאורך המטרות הגדולות. אבני הדרך שהצבתי לעצמי במהלך הפרויקט הן:

- בניית הלוגיקה המרכזית של השרת(הוספת קבצים, מחיקתם, פעולות שונות הקשורות לחברים ועוד).
- בניית הממשק הגרפי(אתר ועיצוב דפי html השונים בפרויקט).
- בניית הלוגיקה של הסנכרון בין השרתים האזוריים והשרת המרכזי.
- הוספת אבטחה במקומות הדרושים, תיקון באגים קטנים וליטוש עיצוב הדפים.

בנוסף לכך, היו 2 מטרות שהצבתי לעצמי אך לבסוף אני ויתרתי עליהן(שתיהן הרגישו לי לא לגמרי נחוצות ועקב אילוצי זמן ויתרתי עליהן):

- אפשרות של שליחת הודעות בין משתמשים
- אפשרות לשמור כמה גרסאות במקביל של אותו קובץ

ניהול הסיכונים בפרויקט:

הסיכונים העיקריים שחשבתי עליהם במהלך הפרויקט הם:

הסיכון	פירוט הסיכון	תיאור דרכים להתמודד עם הסיכון	מה בוצע בפועל
מערכת לא מוגנת	מערכת בעלת חולשות רבות, פגיעה למתקפות שונות	יישום אמצעי אבטחה נגד מתקפות ספציפיות, שימוש בהצפנה לפרוייקט	שימוש ב-tls להצפנה בפרוייקט ויישום הגנות נגד מתקפות ספציפיות (יורחב בהמשך)
מערכת לא יציבה	באגים רבים במערכת, ייתכנות לקריסת המערכת	תכנון נכון של הקוד ושימוש באמצעים כגון try & except	שימוש באמצעים כגון try & except, תכנון ראשוני של הקוד במידת הצורך שינוי הקוד
בעיה בסנכרון בין שרתים	אי יכולת לסנכרן שינויים בין שרתים שונים במערכת	בניית מנגנון סנכרון יעיל ויציב	בניית מנגנון סנכרון(עליו ארחיב בהמשך)

<p>הערכתי את הזמן שלי בצורה לא מספיק טובה, משימות שלא הקדשתי להן זמן רב בהתחלה לקחו לי הרבה יותר זמן משחשבתי ובפועל היו תקופות שהתקשתי להתמיד בפרויקט עקב סיבות שונות</p>	<p>ניהול הזמנים בצורה נכונה ופריסת העבודה בצורה חכמה</p>	<p>עיכוב בהגשת הפרויקט</p>	<p>אי עמידה בלוחות הזמנים</p>
---	--	----------------------------	-------------------------------

תיאור תחום הידע:

צד לקוח:

1. רישום משתמשים:

- מהות היכולת - יצירת חשבון חדש במערכת
- אוסף היכולות/פעולות הנדרשות למימוש היכולת:
 - תצוגת טופס הרשמה(שם משתמש, אימייל, סיסמה)
 - כפתור "הרשמה"
 - בדיקת תקינות הנתונים - שדות לא ריקים, כתובת אימייל תקינה, סיסמה עמידה בתנאים
 - הצפנה - hashing של הסיסמה, העברת הנתונים בצורה מאובטחת(שימוש ב-tls)
 - שליחה אל השרת - בקשת post

- קבלת הנתונים בשרת
- בדיקת תקינות בשרת - ייחודיות בפרטים הדרושים לכך
- כתיבת הנתונים במסד הנתונים - יצירת user חדש
- הצגת הודעת הצלחה/שגיאה מתאימה
- סנכרון האירוע - הוספתו לתור השומר את האירוע עד לבקשת הסנכרון של השרת המרכזי
- הפנייה לעמוד ההתחברות
- אובייקטים נחוצים - מסך הרשמה, מסד נתונים, פונקציית hash לשימוש, תקשורת בין השרת ללקוח, תעודת ssl ומפתח ציבורי, חיבור לשרת המרכזי.

2. התחברות ואימות משתמש:

- מהות היכולת - התחברות של משתמש קיים וקבלת session מאובטח.
- אוסף היכולות/פעולות הנדרשות למימוש היכולת:
- מסך התחברות(שדות להכניס אימייל וסיסמה)
- כפתור "התחברות"
- בדיקת תקינות הנתונים - שדות לא ריקים, כתובת אימייל תקינה/קיימת במערכת, סיסמה התואמת את האימייל
- הצפנה - העברת הנתונים בצורה מאובטחת(שימוש ב-tls)
- שליחה אל השרת - בקשת post
- קבלת הנתונים בשרת
- בדיקת תקינות בשרת - השוואת ערך ה-hash של הסיסמה הנשלחת לסיסמה הקיימת במסד הנתונים
- יצירת session token מתאים במידה והפרטים תקינים
- הפנייה לעמוד הראשי
- אובייקטים נחוצים - מסך התחברות, מסד נתונים, פונקציית hash לשימוש, תקשורת בין השרת ללקוח, תעודת ssl ומפתח ציבורי.

3. צפייה בקבצים:

- מהות היכולת - צפייה בקבצי המשתמש בדף הבית
- אוסף היכולות/פעולות הנדרשות למימוש היכולת:
- מסך בית
- שליפת הקבצים השייכים למשתמש ממסד הנתונים
- הצפנה - העברת הנתונים בצורה מאובטחת(שימוש ב-tls)
- הצגת הנתונים על המסך
- אובייקטים נחוצים - עמוד בית, מסד נתונים, תקשורת בין השרת ללקוח.

4. העלאת קבצים:

-מהות היכולת - העלאה מאובטחת של קובץ לשרת.
-אוסף היכולות/פעולות הנדרשות למימוש היכולת:

- עמוד בית
- כפתור "בחר קובץ"
- בדיקת תקינות קובץ - גודלו אינו גדול מדי, הסיומת של הקובץ מורשית
- הצפנה - העברת הנתונים בצורה מאובטחת(שימוש ב-tls)
- שליחת הקובץ לשרת
- קבלת הנתונים בשרת
- יצירת שם ייחודי
- שמירת הקובץ במערכת הקבצים
- הוספה למסד הנתונים - הוספת קובץ חדש, עדכון שדות בטבלת המשתמשים
- הצגת הודעת הצלחה/שגיאה מתאימה
- סנכרון האירוע - הוספתו לתור השומר את האירוע עד לבקשת הסנכרון של השרת המרכזי
- הצגת התוצאה במסך הבית - הוספת הקובץ לרשימת הקבצים
- אובייקטים נחוצים - מסד נתונים, מסך בית, תקשורת בין השרת ללקוח, תור השומר את השינויים, חיבור לשרת המרכזי.

5. הורדת קבצים:

-מהות היכולת - הורדה מאובטחת של קבצים מהשרת למכשיר הלקוח.
-אוסף היכולות/פעולות הנדרשות למימוש היכולת:

- עמוד בית
- כפתור "הורדה"
- הצפנה - העברת הנתונים בצורה מאובטחת(שימוש ב-tls)
- הורדת הקובץ למכשיר הלקוח
- הצגת הודעת הצלחה/שגיאה מתאימה
- אובייקטים נחוצים - תקשורת בין השרת ללקוח, מסך בית.

6. מחיקת קובץ:

-מהות היכולת - מחיקת קובץ מהשרת
-אוסף היכולות/פעולות הנדרשות למימוש היכולת:

- עמוד בית
- כפתור "מחיקה"
- שליחת הקובץ הרצוי למחיקה לשרת
- מחיקת הקובץ ממערכת הקבצים
- מחיקת הקובץ ממסד הנתונים
- סנכרון האירוע - הוספתו לתור השומר את האירוע עד לבקשת הסנכרון של השרת המרכזי
- הצגת הודעת הצלחה/שגיאה מתאימה

- הצגת התוצאה במסך הבית - מחיקת הקובץ מרשימת הקבצים
- אובייקטים נחוצים - תקשורת בין השרת ללקוח, מסך בית, תור השומר את השינויים, חיבור בין השרת לשרת המרכזי, מסד נתונים.

7. שינוי הרשאות של קובץ:

-מהות היכולת - שינוי הרשאות הגישה של משתמשים אחרים לקובץ של המשתמש ('פרטי', 'חברים בלבד', 'ציבורי').

-אוסף היכולות/פעולות הנדרשות למימוש היכולת:

- עמוד בית
- כפתור "שינוי הרשאה"
- הצפנה - העברת הנתונים בצורה מאובטחת(שימוש ב-tls)
- שליחת השינוי לשרת
- קבלת הנתונים בשרת
- עדכון מסד הנתונים - שינוי ההרשאות של הקובץ המתאים לכך
- הצגת הודעת הצלחה/שגיאה מתאימה
- הצגת התוצאה על המסך - שינוי ההרשאה ברשימת הקבצים
- סנכרון האירוע - הוספתו לתור השומר את האירוע עד לבקשת הסנכרון של השרת המרכזי
- אובייקטים נחוצים - תקשורת בין השרת ללקוח, תקשורת בין השרת לשרת המרכזי, מסך בית, מסד נתונים, תור השומר את השינויים.

8. שליחת בקשת חברות:

-מהות היכולת - שליחת בקשת חברות למשתמש אחר הקיים במערכת.

-אוסף היכולות/פעולות הנדרשות למימוש היכולת:

- עמוד שליחת בקשות חברות
- כפתור "שליחת בקשה"
- בדיקת תקינות הנתונים - הקלט אינו ריק
- הצפנה - העברת הנתונים בצורה מאובטחת(שימוש ב-tls)
- שליחת הנתונים לשרת
- קבלת הנתונים לשרת
- בדיקת תקינות הנתונים - המשתמש אינו חבר של המשתמש השני, לא קיימת בקשה הממכה לאישור
- עדכון מסד הנתונים - הוספה לטבלת בקשות החברות
- סנכרון האירוע - הוספתו לתור השומר את האירוע עד לבקשת הסנכרון של השרת המרכזי
- הצגת הודעת הצלחה/שגיאה מתאימה
- אובייקטים נחוצים - מסך בקשות חברות, תקשורת בין השרת ללקוח, תור השומר את השינויים, מסד נתונים, תקשורת בין השרת לשרת המרכזי.

9. הצגת בקשות החברות:

-מהות היכולת - הצגת בקשות החברות שנשלחו למשתמש
-אוסף היכולות/פעולות הנדרשות למימוש היכולת:

- עמוד בקשות חברות
 - שליפת הבקשות השייכות למשתמש ממסד הנתונים
 - הצפנה - העברת הנתונים בצורה מאובטחת(שימוש ב-tls)
 - שליחת הנתונים ללקוח
 - הצגת הנתונים במסך הלקוח
- אובייקטים נחוצים - מסך בקשות חברות, תקשורת בין השרת ללקוח, מסד נתונים.

10. אישור/דחיית בקשת חברות:

-מהות היכולת - אישור/דחייה של בקשת חברות הנשלחה למשתמש.
-אוסף היכולות/פעולות הנדרשות למימוש היכולת:

- עמוד בקשות חברות
 - הצגת בקשות החברות
 - כפתור "אישור ו"דחייה"
 - הצפנה - העברת הנתונים בצורה מאובטחת(שימוש ב-tls)
 - שליחת התשובה לשרת
 - קבלת הנתונים בשרת
 - עדכון מסד הנתונים - שינוי סטטוס הבקשה(ובעת הצורך להוריד אותה)
 - במידה והבקשה אושרה - עדכון מסד הנתונים - הוספה לטבלת החברויות הקיימות
 - סנכרון האירוע - הוספתו לתור השומר את האירוע עד לבקשת הסנכרון של השרת המרכזי
 - הצגת הודעת הצלחה/שגיאה מתאימה
 - מחיקת הבקשה ממסך הבקשות
- אובייקטים נחוצים - מסך בקשות חברות, תקשורת בין השרת ללקוח, מסד נתונים, תור השומר את השינויים, תקשורת בין השרת לשרת המרכזי.

11. צפייה ברשימת החברים:

-מהות היכולת - צפייה ברשימת החברים של המשתמש.
-אוסף היכולות/פעולות הנדרשות למימוש היכולת:

- עמוד חברים
 - שליפת החברים של המשתמש ממסד הנתונים
 - הצפנה - העברת הנתונים בצורה מאובטחת(שימוש ב-tls)
 - הצגת החברים על המסך
- אובייקטים נחוצים - מסך חברים, תקשורת בין השרת ללקוח, מסד נתונים.

12. צפייה בקבצים של חבר:

-מהות היכולת - צפייה בקבצים שחבר של המשתמש אישר גישה אליהם(מצב 'חברים בלבד' או 'ציבורי').

-אוסף היכולות/פעולות הנדרשות למימוש היכולת:

- כפתור "צפה בקבצים"
 - עמוד קבצים של החבר
 - שליפת הקבצים שמאושרת הגישה אליהם ממסד הנתונים
 - הצפנה - העברת הנתונים בצורה מאובטחת(שימוש ב-tls)
 - הצגת הקבצים על המסך
- אובייקטים נחוצים - מסך קבצים של החבר, תקשורת בין השרת ללקוח, מסד נתונים.

13. הסרת חבר:

-מהות היכולת - הסרת משתמש מרשימת החברים של המשתמש הקיים.

-אוסף היכולות/פעולות הנדרשות למימוש היכולת:

- עמוד חברים
 - כפתור "הסרת חבר"
 - הצפנה - העברת הנתונים בצורה מאובטחת(שימוש ב-tls)
 - שליחת הנתונים לשרת
 - קבלת הנתונים בשרת
 - שינוי במסד הנתונים - הסרת החבר מטבלת החברויות
 - סנכרון האירוע - הוספתו לתור השומר את האירוע עד לבקשת הסנכרון של השרת המרכזי
 - הצגת הודעת הצלחה/שגיאה מתאימה
 - מחיקת החבר ממסך החברים
- אובייקטים נחוצים - מסך חברים, תקשורת בין השרת ללקוח, מסד נתונים, תור השומר את השינויים, תקשורת בין השרת לשרת המרכזי.

14. התנתקות מהמערכת:

-מהות היכולת - התנתקות מהמערכת וחזרה למסך ההתחברות.

-אוסף היכולות/פעולות הנדרשות למימוש היכולת:

- כפתור התנתקות
 - שליחת הנתונים לשרת
 - קבלת הנתונים בשרת
 - הפניה לעמוד ההתחברות
- אובייקטים נחוצים - כפתור התנתקות, עמוד ההתחברות, תקשורת בין השרת ללקוח.

צד שרת:

15. שליחת השינויים לשרת:

- מהות היכולת - עדכון השינויים שהתרחשו בשרת האזורי לשרת המרכזי.
- אוסף היכולות/פעולות הנדרשות למימוש היכולת:
 - הצפנה - העברת הנתונים בצורה מאובטחת(שימוש ב-tls)
 - הוספת הנתונים לאחר כל אירוע לתור - changes_queue(מבנה נתונים מסוג תור)
 - קבלת הודעה מהשרת המרכזי המצביעה על סנכרון הנתונים
 - שליחת האירועים שבתור לשרת המרכזי
- אובייקטים נחוצים - תקשורת בין השרת לשרת המרכזי, מסד נתונים, תור השומר את השינויים, סוקטים

16. עדכון שרתים אזוריים בשינוי:

- מהות היכולת - עדכון השינויים שהתרחשו בשרת אזורי מסוים לשאר השרתים האזוריים.
- אוסף היכולות/פעולות הנדרשות למימוש היכולת:
 - הצפנה - העברת הנתונים בצורה מאובטחת(שימוש ב-tls)
 - הבחנה מאיזה שרת אזורי הגיע עדכון מסוים
 - שליחת העדכון לשאר השרתים האזוריים
 - קבלת הנתונים בשרתים האזוריים
 - עדכון מסד הנתונים בהתאם לסוג השינוי
- אובייקטים נחוצים - תקשורת בין השרת לשרת המרכזי, מסד נתונים, תור השומר את השינויים, סוקטים

מבנה/ארכיטקטורה של הפרויקט:

תיאור הארכיטקטורה של המערכת המוצעת:

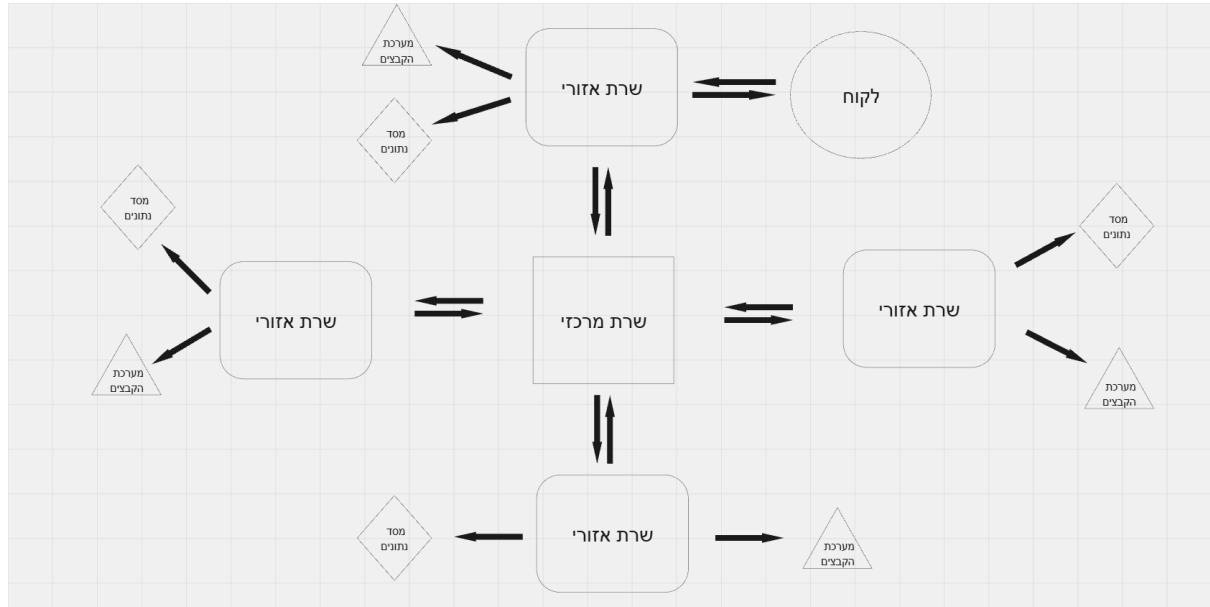
תיאור החומרה:

1. מסד נתונים
2. שרת אזורי - שרת web מבוסס על flask.

3. שרת מרכזי - שרת מבוסס סוקטים

4. תיקיית הקבצים - אחסון הקבצים במערכת

קשרים בין רכיבי החומרה - בין הלקוח לשרת האזורי יש קשר דו כיווני (כל צד מתקשר עם השני), וקיים קשר דו כיווני בין השרת המרכזי לשרתים האזוריים. בנוסף לכך, השרתים האזוריים עובדים מול תיקיות הקבצים ומסד הנתונים.



תיאור הטכנולוגיה הרלוונטית:

שפת תכנות - python 3.12

מערכת הפעלה - windows

תקשורת - תקשורת מבוססת סוקטים tcp, בנוסף לכך, בין השרת ללקוח ישנה תקשורת http

מבוססת tls (פרוטוקול https)

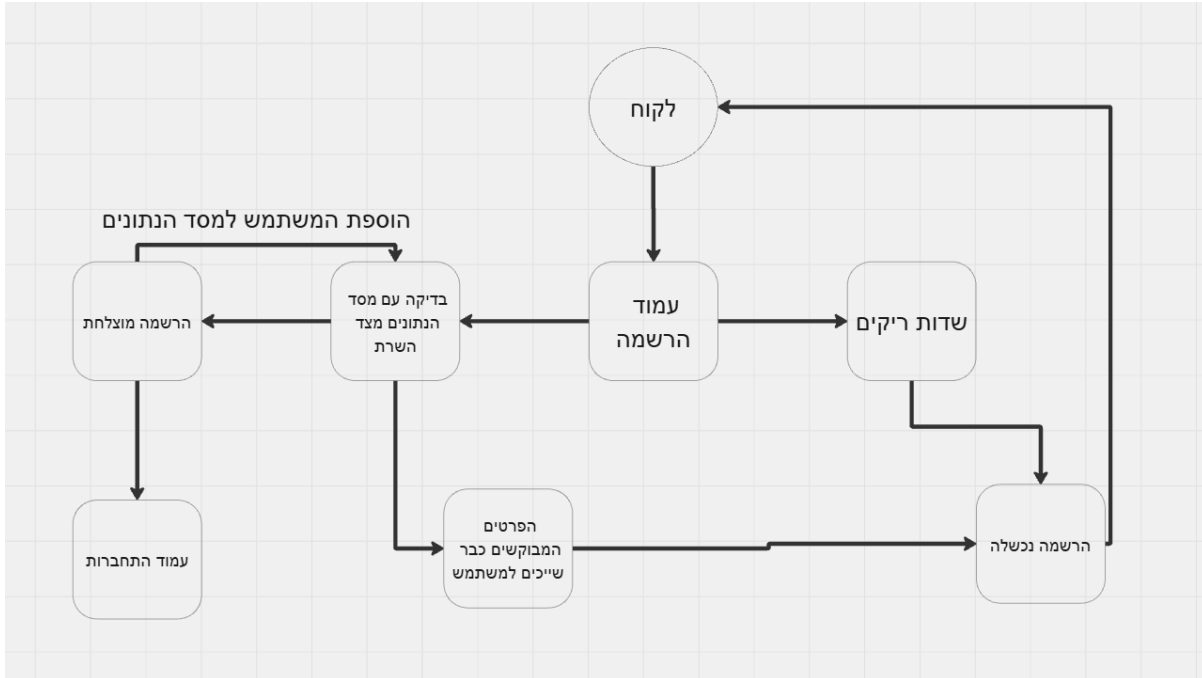
הצפנה - הצפנה עקב שימוש ב-tls(גרסה זו של הפרוטוקול משתמשת ב-RSA(הצפנה אסימטרית) על מנת לשתף את המפתחות ולאחר מכן ב-AES(הצפנה סימטרית)).

מסד נתונים - sqlite 3

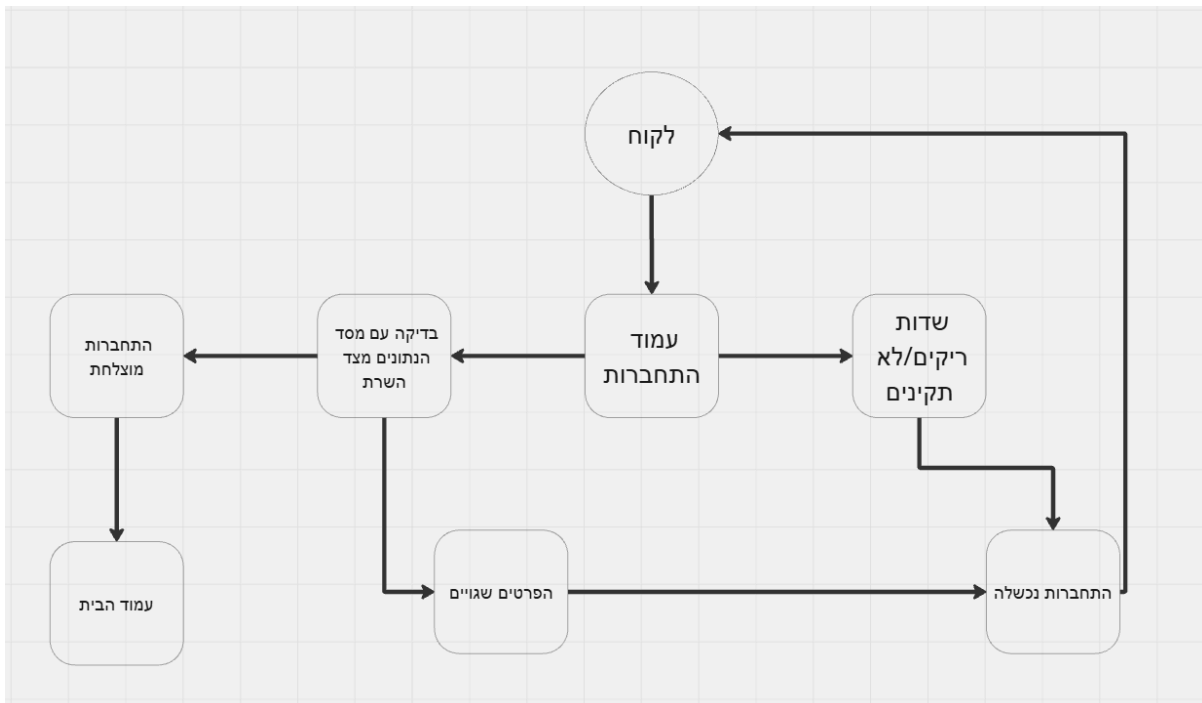
ממשק גרפי - שימוש ב-html, css

תיאור זרימת המידע במערכת:

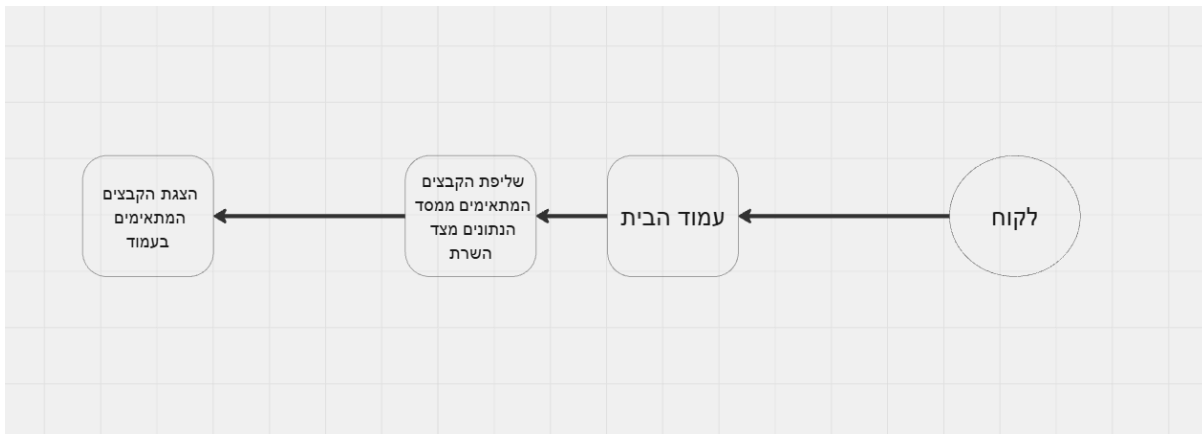
1. הרשמה למערכת



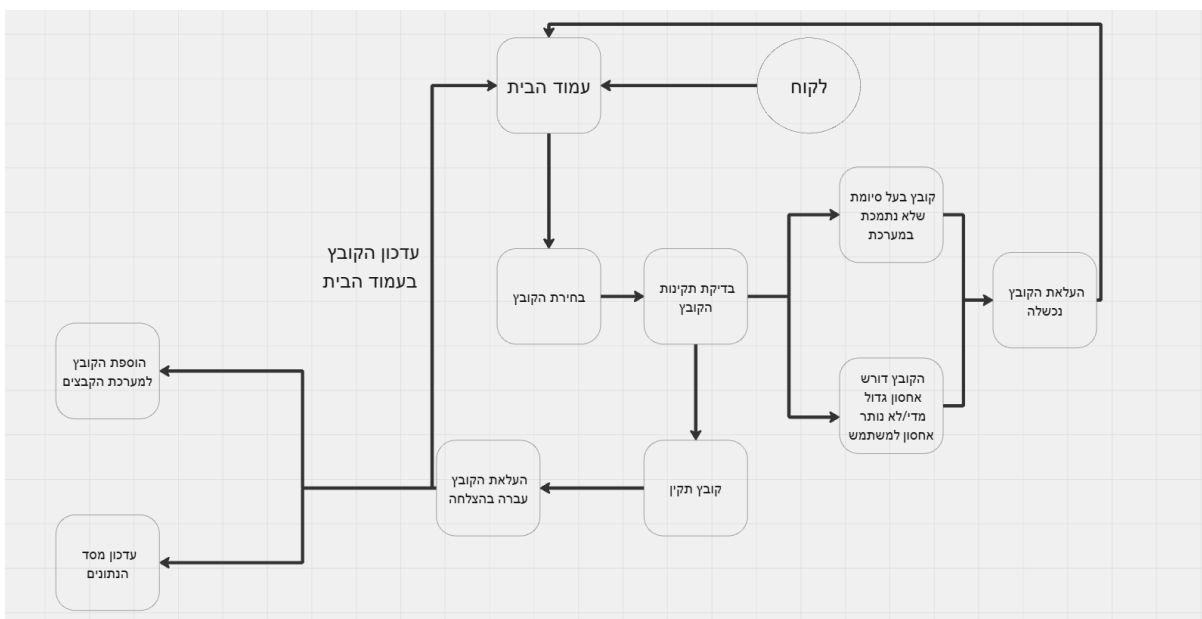
2. התחברות למערכת



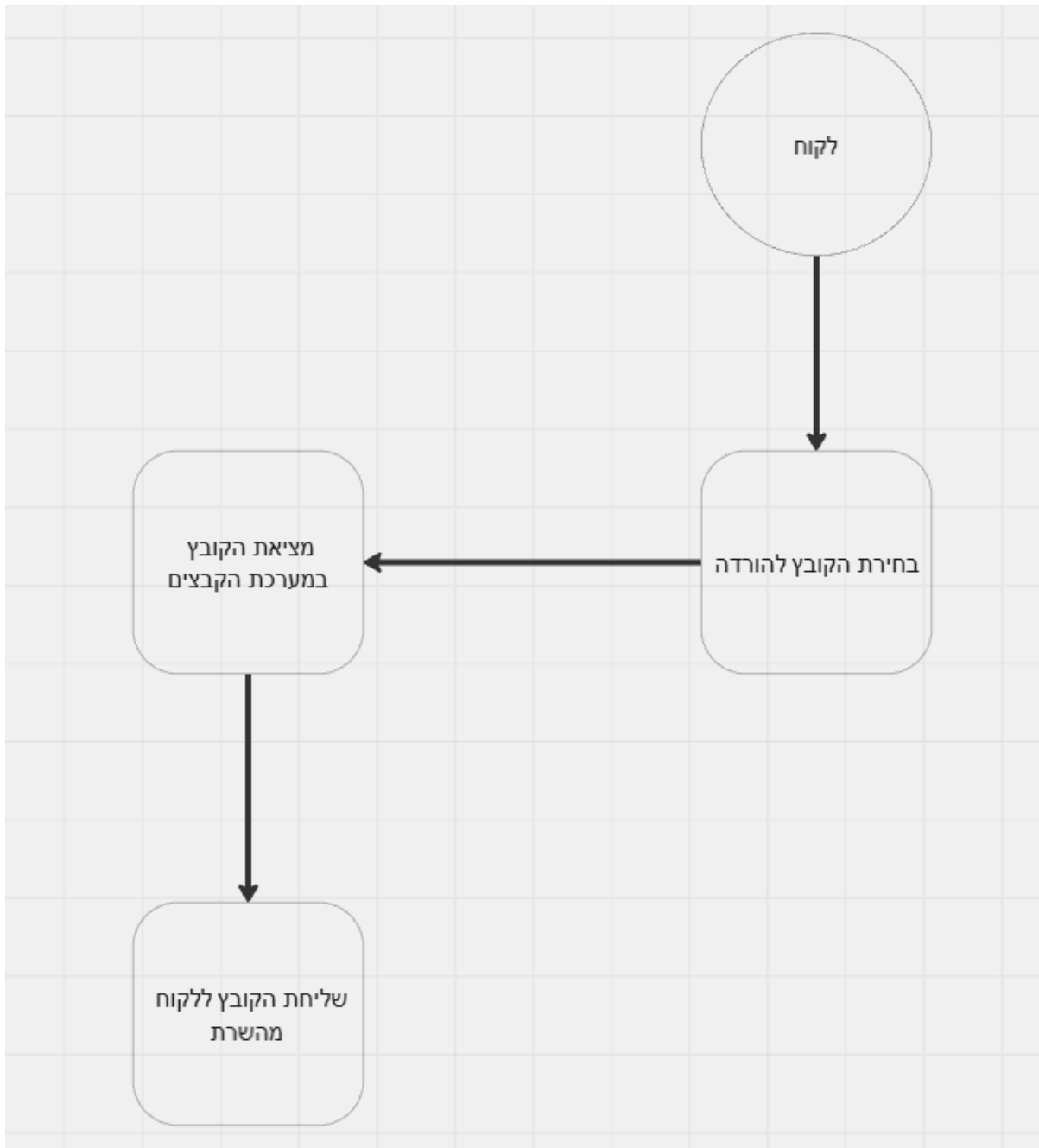
3. צפייה בקבצים



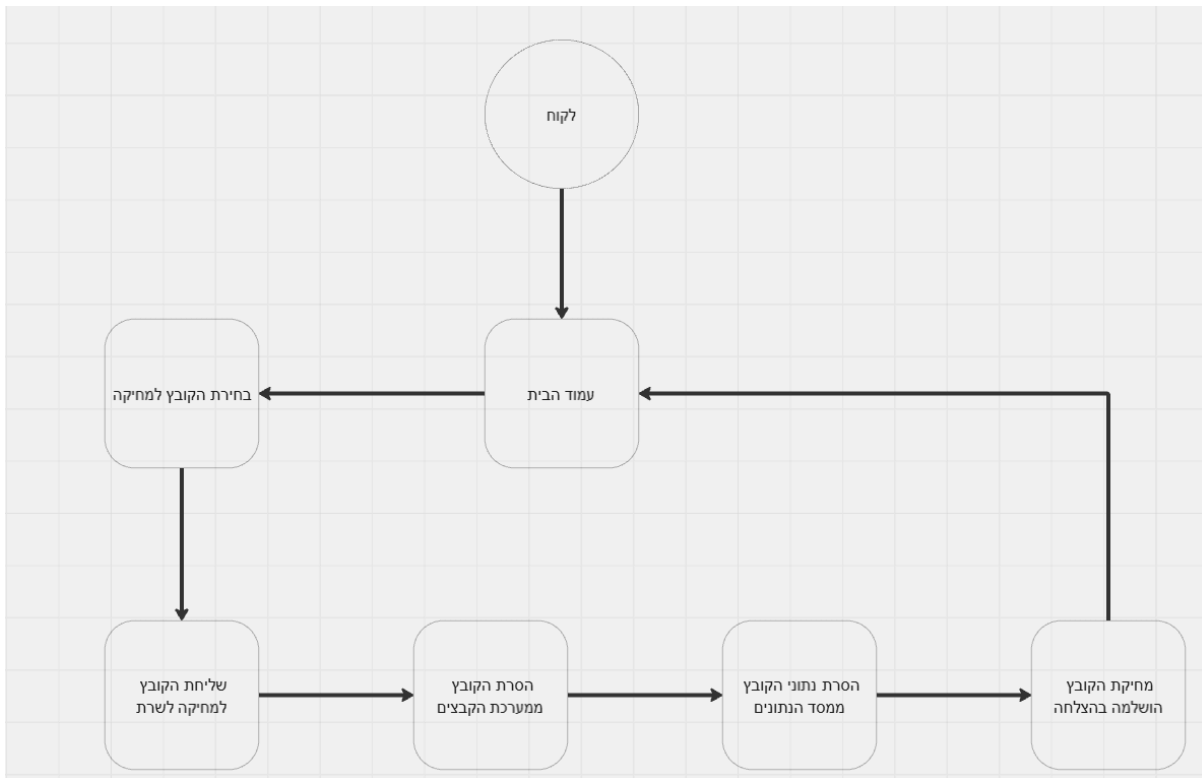
4. העלאת קובץ



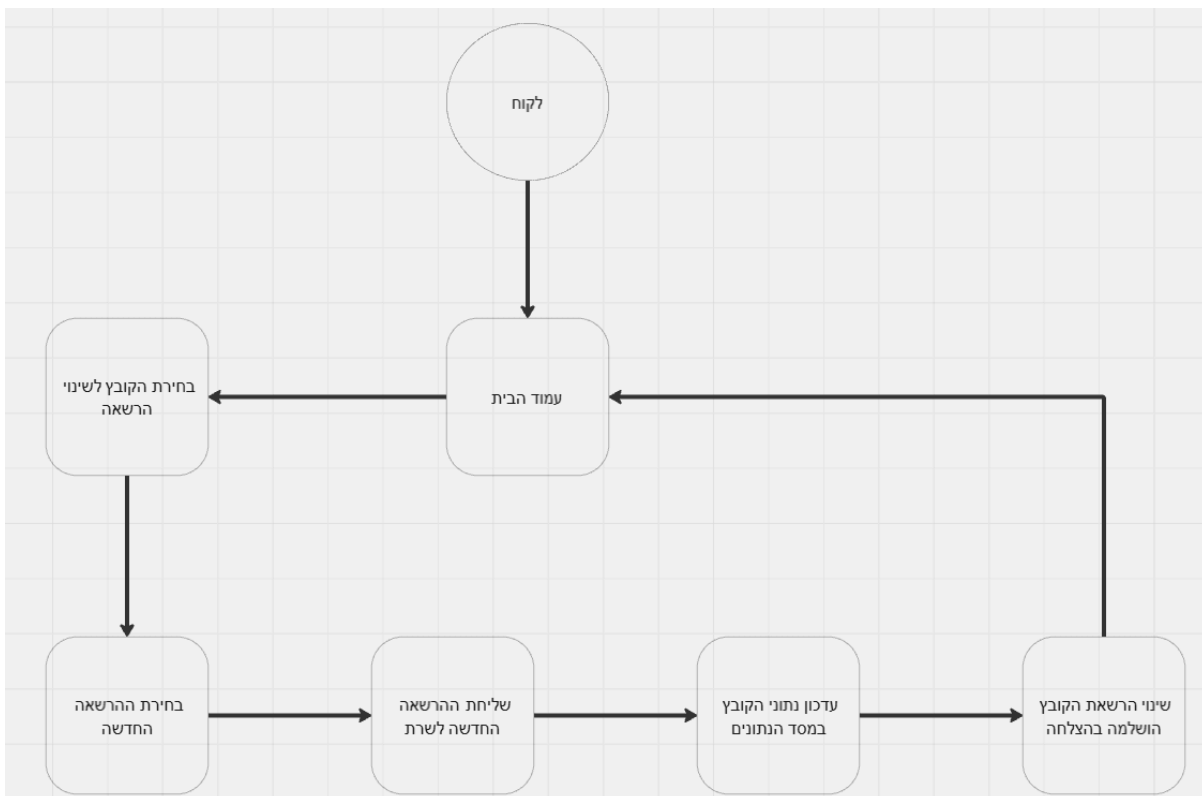
5. הורדת קובץ



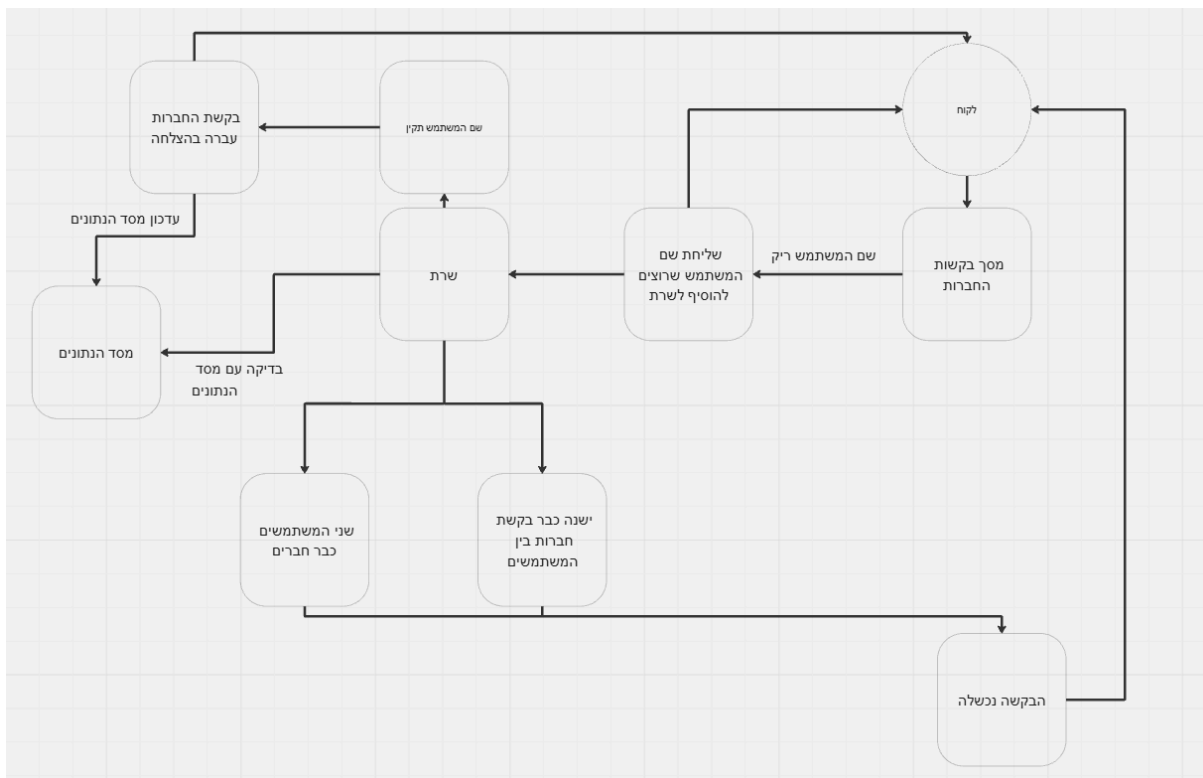
6. מחיקת קובץ



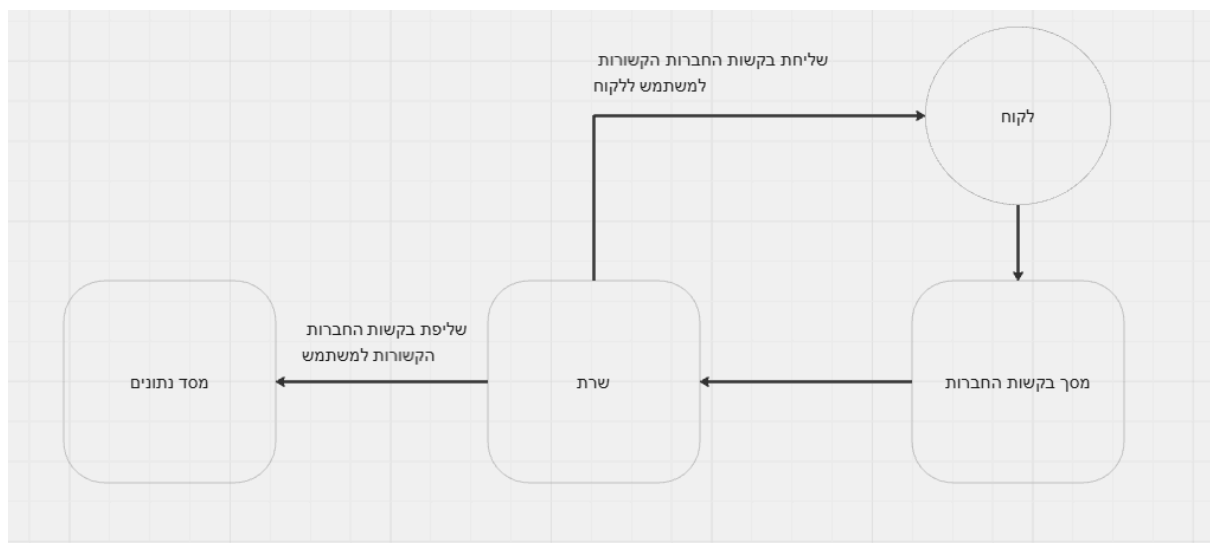
7. שינוי הרשאה של קובץ קיים



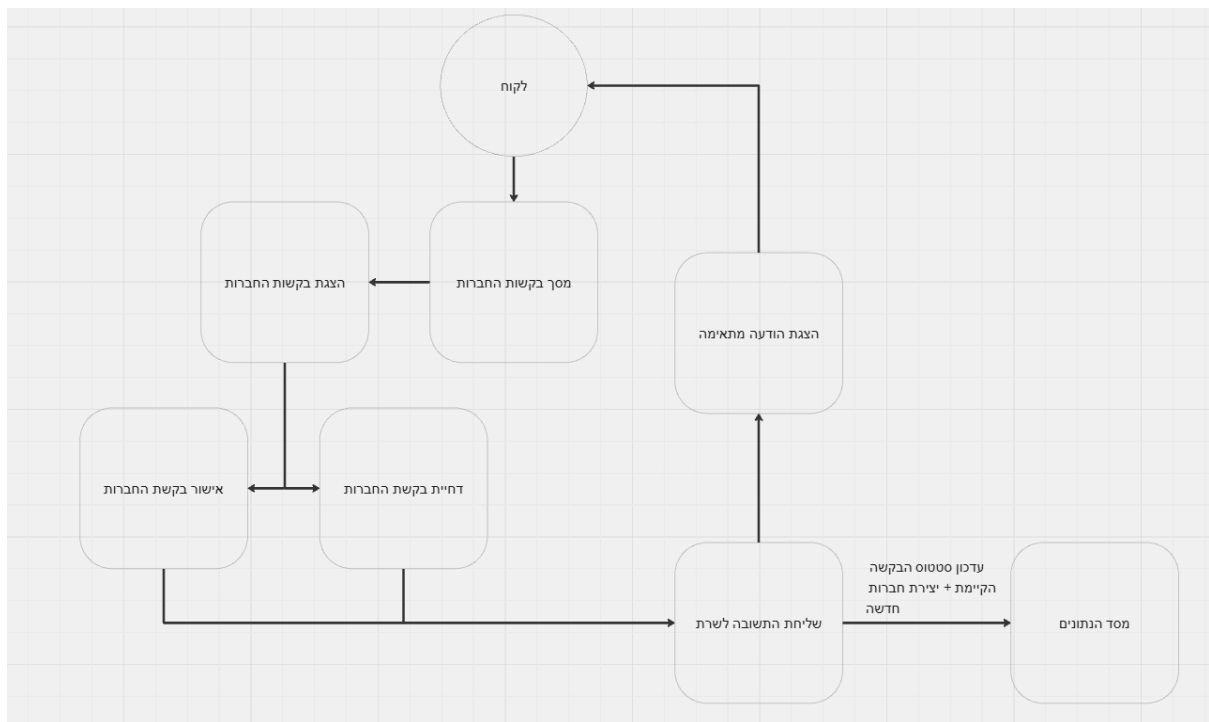
8. שליחת בקשת חברות



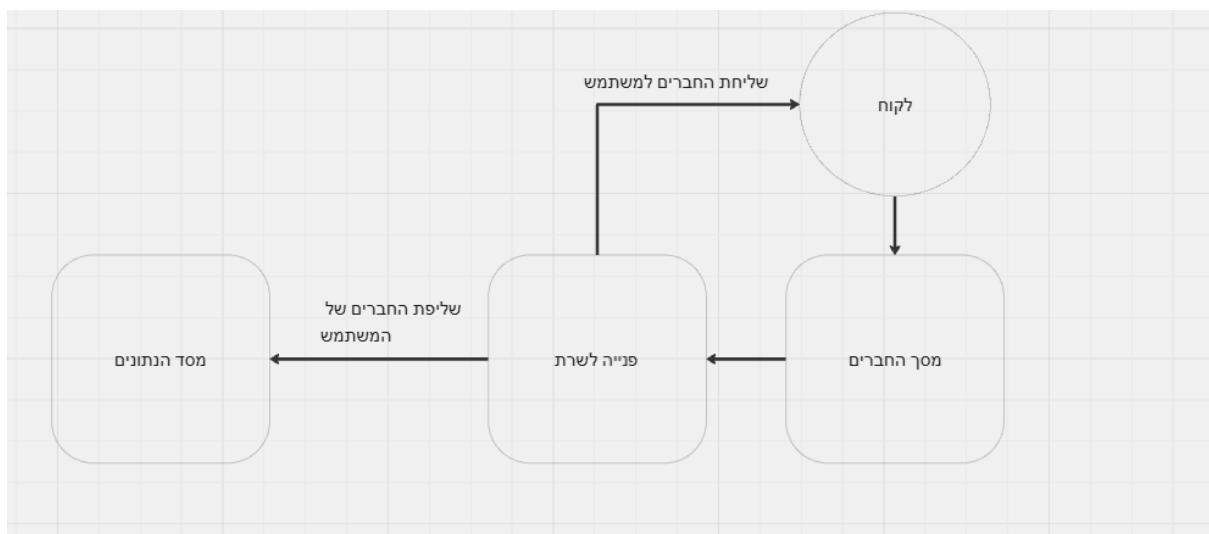
9. הצגת בקשות החברות



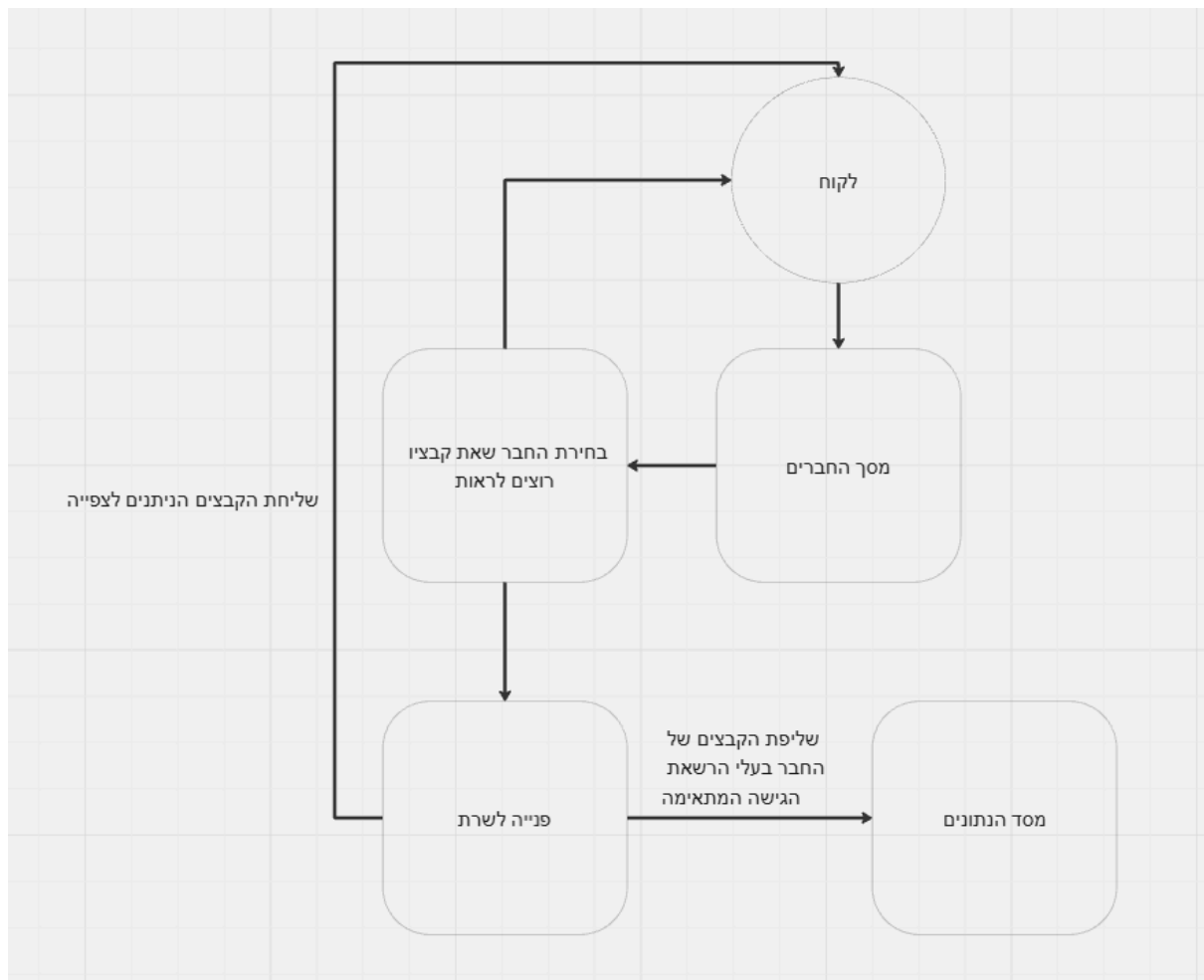
10. אישור/דחיית בקשת חברות



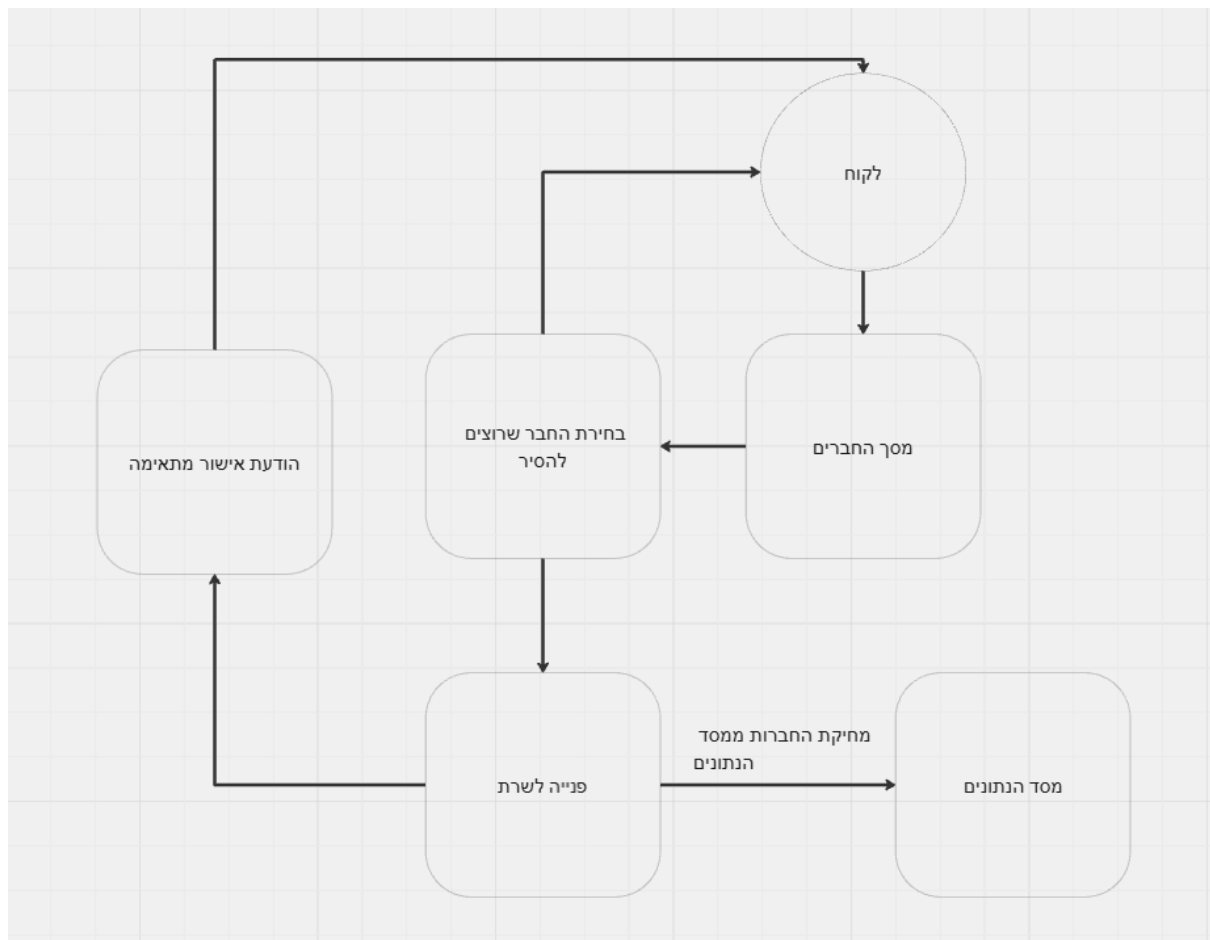
11. צפייה ברשימת החברים



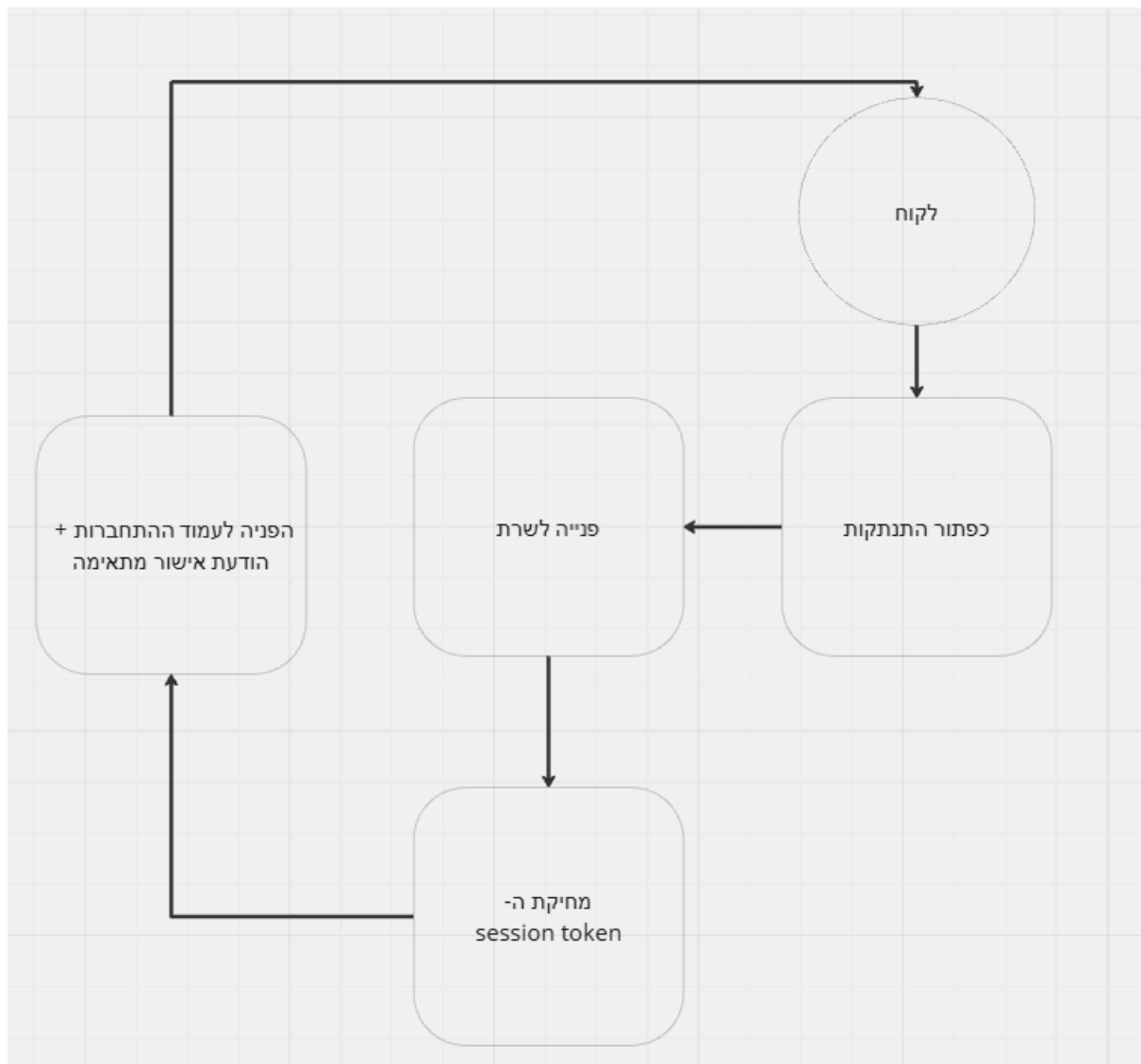
12. צפייה בקבצים של חבר



13. הסרת חבר

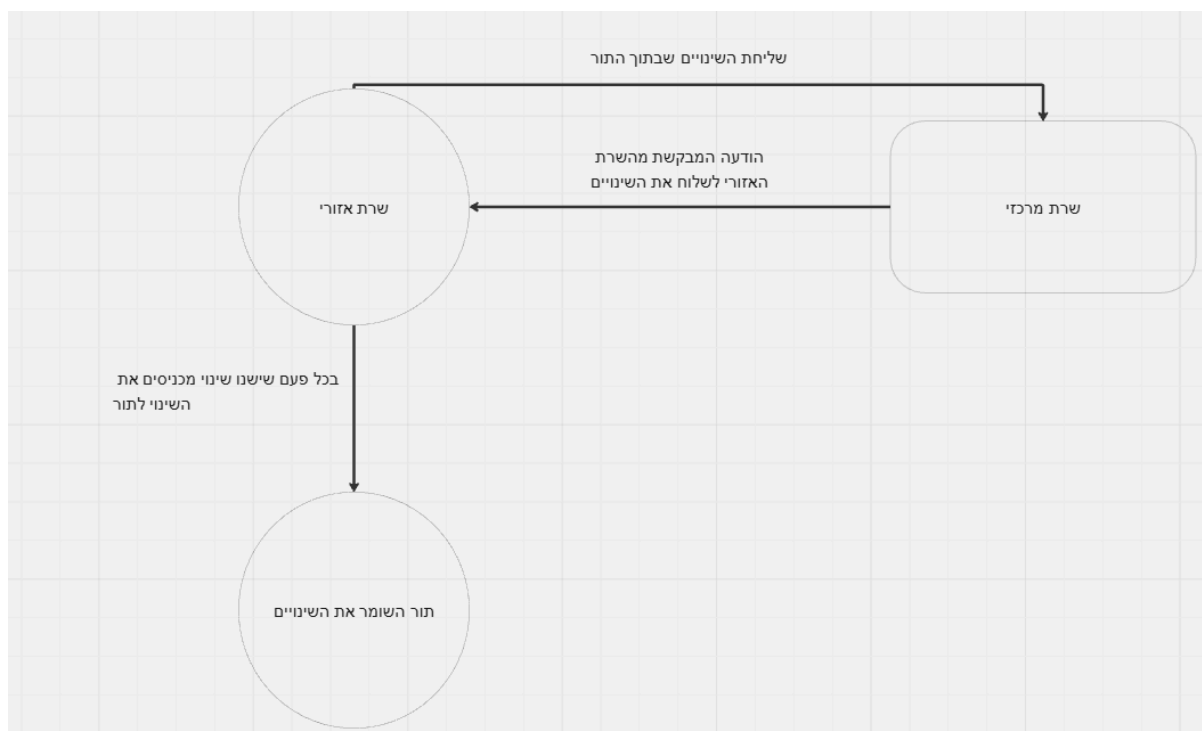


14. התנתקות מהמערכת

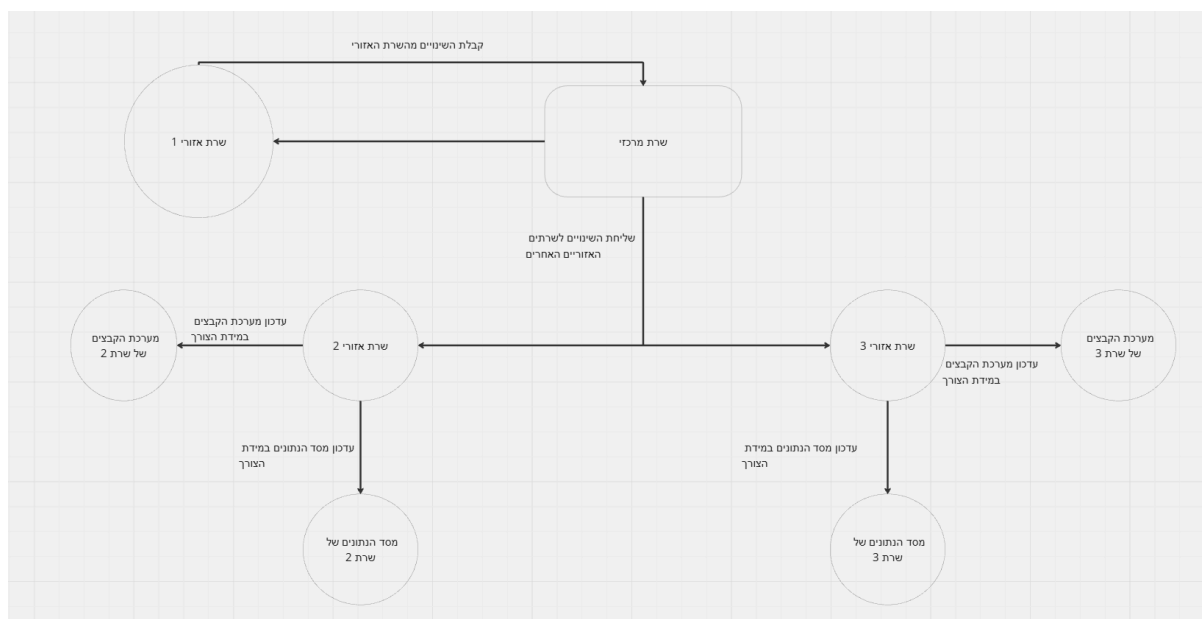


צד שרת:

15. שליחת השינויים לשרת



16. עדכון שרתים אזוריים בשינוי



תיאור האלגוריתמים במערכת:

1. מבנה מערכת הקבצים:

הבעיה - הפרויקט כולל עבודה מתמשכת עם מערכת הקבצים של השרתים. בניית מערכת קבצים בצורה נכונה בעלת השפעה בבטיחות המערכת ומהירות המערכת.

חלופה 1- יצירת תיקייה לכל משתמש. לחלופה זו יש מספר יתרונות. לדוגמה, הקבצים של כל משתמש מבודדים בתיקייה נפרדת כך שאם משהו קורה לאחת מן התיקיות, לא תהיה השפעה על קבצי שאר המשתמשים. בנוסף לכך, כל לעקוב אחר כל משתמש(אחר כמות הזכרון שנותרה לו לדוגמה) בחלופה זו. לעומת זאת, יש לפתרון זה גם מספר חסרונות. פתרון זה עלול להפוך את מערכת הקבצים למסורבלת מאוד ומלאה בתיקיות שונות.

חלופה 2- שמירת כל הקבצים בתיקייה אחת כאשר לכל קובץ נותנים שם ייחודי באמצעות uid (מזהה ייחודי שדואג להבטיח שלכל אובייקט יהיה מזהה ששייך רק לו). היתרון העיקרי של חלופה זו היא פשטות המימוש. בניגוד לחלופה השנייה, כמעט ולא צריך לנווט במערכת הקבצים במהלך ריצת הפרויקט. החסרון העיקרי של חלופה זו היא שהרבה יותר קשה לעקוב אחר משתמש ספציפי ולבצע פעולות הקשורות אליו(לדוגמה, במידה ורוצים למחוק משתמש מהמערכת, צריך לעבור על כל קובץ על מנת למחוק פשוט למחוק תיקייה מסוימת.

תחילה חשבתי שהחלופה הראשונה היא החלופה העדיפה לפרויקט, עקב היתרונות שלה בבניית מערכת בקנה מידה גדול יותר. אך עקב הפשטות של החלופה השנייה והגמישות שבפתרון זה(במידה ומתחילים לממש פתרון זה ורוצים להחליף פתרון הרבה יותר פשוט מאשר הפתרון השני).

2. תלות בשרת אחד:

הבעיה - עקב כך שהשרת בפרויקט צריך לשמור את הנתונים ואת הקבצים של כל המשתמשים, נוצרת תלות בשרת זה. קריסה של השרת/עומס של כמות לקוחות רבה על אותו שרת עלולה לפגוע בביצועי המערכת וליצור תלות בשרת זה.

חלופה 1- בניית שרת מרכזי האחראי על סנכרון בין הרבה שרתים אזוריים(סוג של שימוש ב-start topology). חלופה זו פותרת את הבעיה כך שבמידה ושרת מסוים קורס, עדיין ניתן לגשת למערכת ולהשתמש בשירותיה. היתרונות המרכזיים של חלופה זו היא שלמספר השרתים האזוריים אין כל כך השפעה על ביצועי המערכת(כל עוד השרת המרכזי יכול לטפל בכמות השרתים בו זמנית) כך שניתן במידת הצורך לפרוס מספר רב של שרתים אזוריים באזורים שונים(דבר היכול לשדרג את הזמינות). החסרונות המרכזיים של חלופה זו היא שנוצרת בעצם תלות בשרת המרכזי כך שאם הוא קורס, פתרון זה אינו פועל זמנית.

חלופה 2- שימוש ב-mesh topology(טופולוגית רשת המורכבת ממספר רב של צמתים המשמשים גם כמקור מידע וגם על מנת להעביר מידע משרת לשרת). חלופה זו מאפשרת מספר רב של שרתים הפרוסים וכך אינה תלות בשרת כלשהו. היתרון המרכזי של חלופה זו היא שאין תלות בשרת מסוים במערכת כך שבמידה ושרת מסוים קורס, המערכת תמשיך לתפקד באותה דרך בדיוק. החסרון העיקרי של חלופה זו הוא שככל שיש מספר רב יותר של שרתים, כך מבנה הרשת נהיה מסובך יותר ויותר.

החלופה שבחרתי במקרה זה היא החלופה הראשונה. אני חושב שהחסרון המרכזי של חלופה 1 הוא משהו שניתן להתגבר עליו(לדוגמה, בניית מספר שרתים מרכזיים המתקשרים ביניהם למקרה שאחד מן השרתים המרכזיים נופל). בניגוד לחלופה 2, ניתן להתגבר על החסרון של חלופה 1 בצורה יחסית פשוטה.

תיאור סביבת הפיתוח:

- שפת תכנות - python 3.12
- עיצוב ממשק המשתמש הגרפי - html, css
- כלי פיתוח - visual studio code
- מסד נתונים - sqlite 3

תיאור פרוטוקול התקשורת:

פרוטוקול התקשורת שיצרתי ממומש בעיקר בסנכרון הפעולות החדשות בין כל שרת. פרוטוקול התקשורת בנוי כך:
שדה 1: סוג ההודעה - מתחלק ל-2 סוגים שונים, הודעה שהשרת המרכזי שלח והודעה ששרת אזורי שלח. במידה והשרת המרכזי שלח את ההודעה, סוג ההודעה הוא:

- שלח (send) - סימן לשרת האזורי לשלוח את כל השינויים שהוא צבר במהלך ריצת השרת. אחרי הודעה זו לא יגיע כלום.
- קבל (receive) - סימן שהודעה זו כוללת את כל השינויים שעל השרת לעדכן. השדה הנוסף שיתקבל בסוג הודעה זו הוא 'events' והתוכן שלו הוא העדכונים שהשרת צריך לשנות(עליהם ארחיב בהמשך).

במידה והשרת האזורי שולח לשרת המרכזי עדכון מסוים, סוג ההודעה הוא סוג האירוע שעל השרתים האחרים לעדכן. לכל הודעה יש payload שהוא הנתונים הדרושים לביצוע השינוי. האירועים האפשריים הם:

- העלאת קובץ (file_upload) - קובץ חדש הועלה לשרת. ה-payload הוא id הקובץ, id המשתמש, שם הקובץ המקורי, שמו במערכת הקבצים, תאריך ההעלאה, גודל הקובץ, ההרשאה ותוכן הקובץ.
- מחיקת קובץ (file_delete) - קובץ נמחק מן השרת. ה-payload הוא id הקובץ שנמחק.
- שינוי הרשאה (permission_change) - שונתה הרשאה של קובץ בשרת. ה-payload הוא id הקובץ וההרשאה החדשה ששמים לו.
- בקשת חברות (friend_request) - נשלחה בקשת חברות חדשה. ה-payload הוא id הבקשה ו-id שני המשתמשים.
- חבר הוסף (friend_added) - אחד מן המשתמשים אישר את בקשת החברות ונוצרה חברות חדשה. ה-payload הוא id הבקשה וה-id של 2 המשתמשים.
- חבר נדחה (friend_rejected) - אחד מן המשתמשים דחה את בקשת החברות. ה-payload הוא id הבקשה וה-id של 2 המשתמשים.

- חבר הוסר (friend_deleted) - אחד מן המשתמשים הסיר את אחד מן החברים שלו. ה-payload הוא ה-id של שני המשתמשים.
- משתמש נוצר (user_create) - נוצר משתמש חדש באחד מן השרתים. ה-payload הוא id המשתמש, שם המשתמש, האימייל והסיסמה

בנוסף לכך, בכל הודעה נשלח גם הזמן הנוכחי ברגע שליחתה על מנת מעקב יותר קל אחר הודעות שנשלחות במערכת(אין באמת השפעה על ביצועי המערכת).

סוג ההודעה	נשלח מ-	נשלח אל	payload
send	השרת המרכזי	שרת אזורי	אין
receive	השרת המרכזי	שרת אזורי	אין
file_upload	שרת אזורי	השרת המרכזי	id, user_id, original_filename, stored_filename, upload_date, file_size, content
file_delete	שרת אזורי	השרת המרכזי	file_id
permission_change	שרת אזורי	השרת המרכזי	file_id, new_permission
friend_request	שרת אזורי	השרת המרכזי	request_id, from_id, to_id
friend_rejected	שרת אזורי	השרת המרכזי	request_id, from_id, to_id
friend_added	שרת אזורי	השרת המרכזי	request_id, from_id, to_id
friend_deleted	שרת אזורי	השרת המרכזי	request_id, from_id, to_id
user_create	שרת אזורי	השרת המרכזי	user_id, username, email, password

תיאור מסכי המערכת:

מסך ההרשמה - מאפשר ליצור משתמש חדש במערכת. העמוד בנוסף לכך מכיל הפניה למסך ההתחברות. במידה וההרשמה מוצלחת(פרטים אינם ריקים ותקינים), המשתמש יופנה לעמוד ההתחברות ויקבל הודעת הצלחה מתאימה.



SyncSphere

Create your account

Register

Already have an account? [Log In](#)

מסך ההתחברות- מאפשר להתחבר למערכת. עמוד זה מכיל הפניה לעמוד ההרשמה. עמוד זה הוא עמוד ברירת המחדל במידה ואין session פעיל במכשיר זה. במידה ובקשת ההתחברות תקינה(הפרטים אינם ריקים ומתאימים למשתמש קיים), המשתמש יופנה לעמוד הבית(עמוד הצפייה בקבצים).



Create your account

Already have an account? [Log In](#)

מסך הבית - מסך זה מאפשר צפייה בקבצי המשתמש וצפייה בנתונים הקשורים לכל קובץ. מסך זה הוא מסך ברירת המחדל במידה וקיים session במכשיר. בנוסף, מסך זה מכיל הפניות למסך בקשות החברות, מסך החברים וכפתור ההתנתקות. במסך זה ניתן להוריד קובץ, להעלות קובץ, לשנות את הרשאות הקובץ ולמחוק אותו מהמערכת.

SyncSphere

[My Files](#)
[Requests](#)
[Friends List](#)
[Logout](#)

Available Files

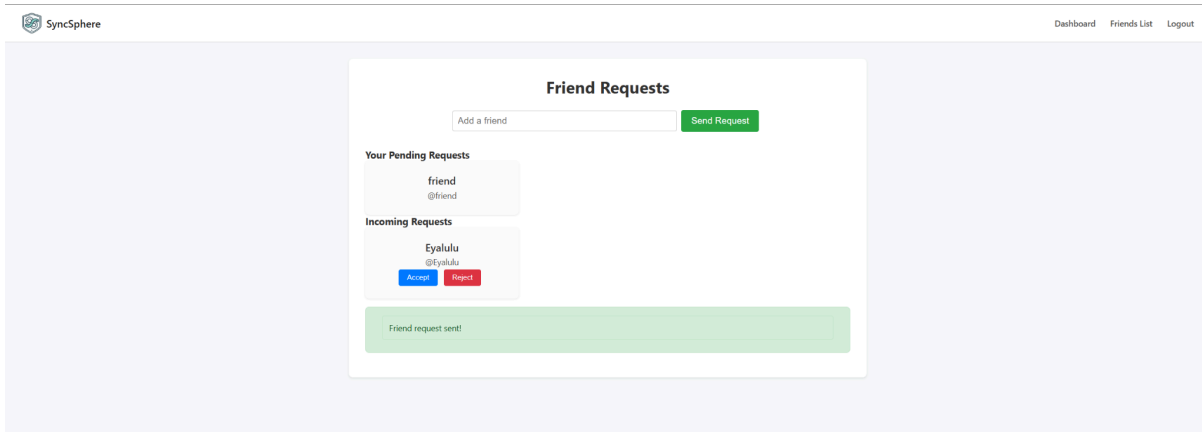
Filename	Upload Date	Size (bytes)	Actions
pdf	2025-04-25 18:54	460313	Download Delete Friends Only Update
_.pdf	2025-04-30 17:41	568735	Download Delete Private Update
data.txt	2025-04-30 17:48	81	Download Delete Private Update
data.txt	2025-04-30 17:50	81	Download Delete Private Update
suu.pdf	2025-04-30 18:56	89656	Download Delete Private Update

Effortless File Management

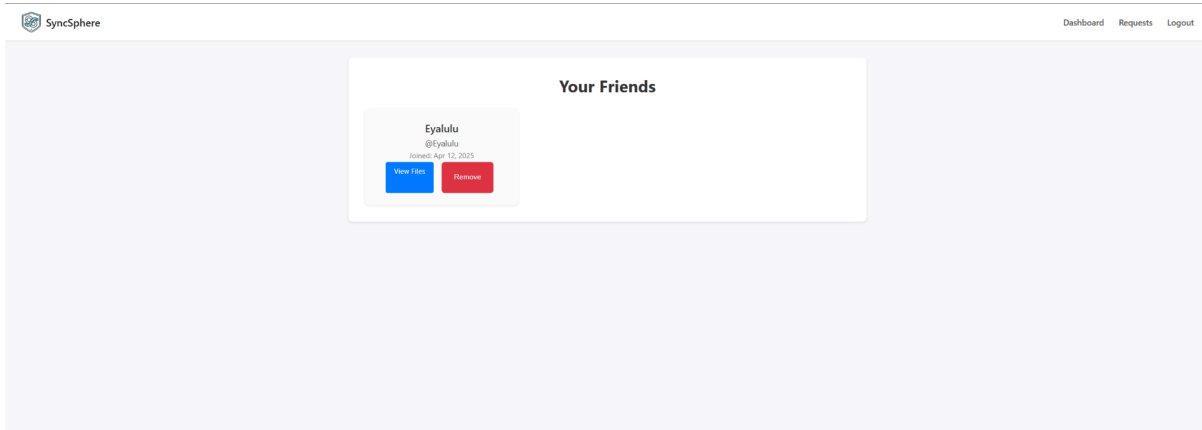
Easily upload and organize your files with SyncSphere. Simplify your workflow with quick actions.

No file chosen

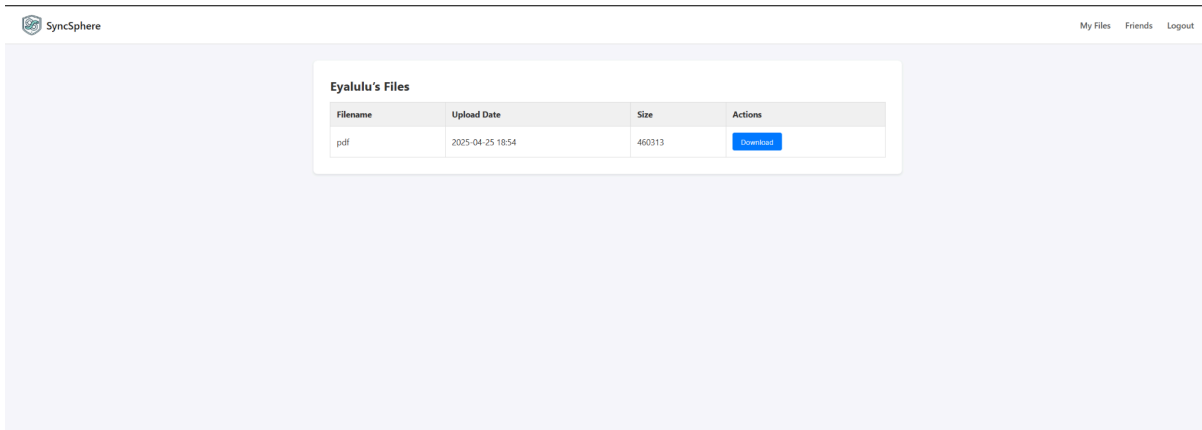
מסך בקשות החברות - מסך זה מאפשר צפייה בבקשות החברות הקיימות ושליחת בקשת חברות. מסך זה מכיל הפניות למסך הבית, מסך רשימת החברים וכפתור ההתנתקות. באמצעות מסך זה ניתן לאשר/לדחות בקשת חברות וניתן לשלוח בקשת חברות חדשה.

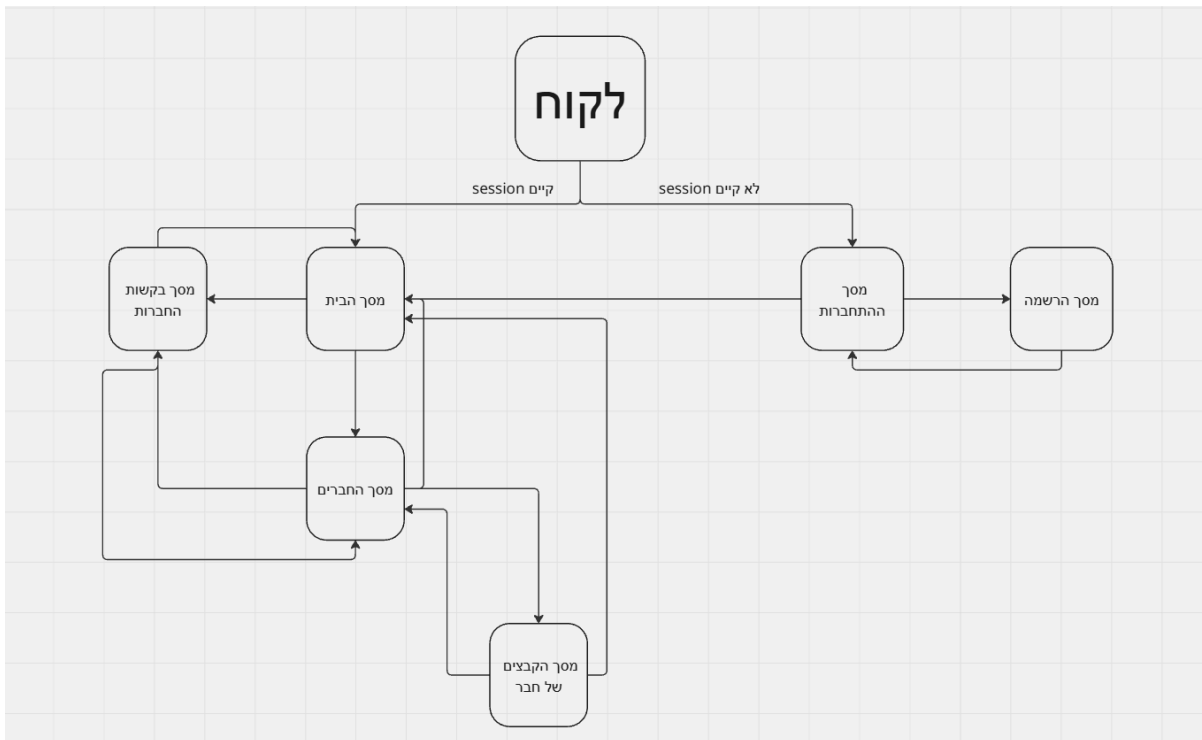


מסך רשימת החברים- מסך זה מאפשר צפייה בכל החברים של המשתמש. מסך זה מאפשר להסיר חבר ולצפות ברשימת הקבצים שהמשתמש אישר גישה אליהם. המסך מכיל הפניה למסך הבית, מסך בקשות החברות וכפתור ההתנתקות. במידה ולוחצים על כפתור צפייה ברשימת הקבצים של המשתמש, המשתמש מופנה לדף הקבצים של החבר.



מסך רשימת הקבצים של חבר- מסך זה מאפשר צפייה בכל הקבצים המורשים לצפייה של חבר מסוים. המשתמש יכול להוריד את הקובץ של החבר ולצפות בנתונים עליו. המסך מכיל הפניה למסך הבית, מסך החברים וכפתור התנתקות.





תיאור מבני הנתונים:

מסד נתונים: מסד הנתונים (sqlite3) של הפרויקט, שומר את כל הנתונים הנחוצים בשרתים האזוריים. שם מסד הנתונים הוא database.db
 הטבלאות הקיימות במסד הנתונים הן:
 משתמש (user) - אחראית על שמירת הנתונים הבאים הקשורים למשתמש:

- ה-id של המשתמש - מטיפוס int. לדוגמה, 1
- שם המשתמש - מטיפוס varchar (הערה להמשך, ב-sql, טיפוס זה הוא כמו string).
- לדוגמה, admin
- אימייל - מטיפוס varchar. לדוגמה, admin@gmail.com
- ערך ה-hash של הסיסמה - מטיפוס varchar. לדוגמה, 32768:8:1\$5NPoDCFYHRpdCQpp\$0c321a4760711a9fc8325dcbf2a99e6babffce55251f11e1c75389b9ce9ad3943b1fc823eedc7e4687bb1aa8e2b2dcd76ee e7442d663053af9a03a99077ced89 (ערך ה-hash למחרוזת password)
- תאריך יצירת המשתמש - מטיפוס datetime (תאריך). לדוגמה, 2025-04-11T20:20:50.179491
- האחסון המשומש על ידי המשתמש (בבייטים) - מטיפוס int. לדוגמה, 1000
- האחסון המקסימלי המתאפשר למשתמש - מטיפוס int. הערך הדיפולטיבי של שדה זה הוא 1 גיגהבייט (107374821 בייטים).

בנוסף לכך, לשדות מסוימים בטבלה הוגדרו תכונות מסוימות:

- הטבלה ממוינת לפי ה-id של המשתמשים כך שהמשתמש הראשון הוא בעל id 1, השני בעל id 2 וכן הלאה. הטבלה ממוינת כך על מנת לייעל את החיפוש כאשר מחפשים משתמש.

- השדות username, email מוגדרים כייחודיים. כלומר, לכל משתמש חייב להיות שם משתמש ואימייל ייחודי.

קובץ(file) - אחראית על שמירת הנתונים הקשורים לקבצים המאוחסנים במערכת:

- ה-id של הקובץ, - מטיפוס int
- ה-id של המשתמש שאליו שייך הקובץ - מטיפוס id. לדוגמה, 1
- שם הקובץ המקורי - מטיפוס varchar. לדוגמה, test.txt
- שם הקובץ באחסון - מטיפוס varchar(על הסיבה לשני שמות שונים לכל קובץ - המקורי ובאחסון יורחב בהמשך תיאור המערכת). לדוגמה, 08eef1e5d62345319b3600912cac35ff.pdf
- תאריך העלאה - מטיפוס datetime. לדוגמה, 18:54 2025-04-11
- גודל הקובץ(בבייטים) - מטיפוס int. לדוגמה, 1000
- הרשאת הצפייה בקובץ(public, friends_only, private) - מטיפוס varchar. לדוגמה, private

בנוסף לכך, לשדות מסוימים בטבלה הוגדרו תכונות מסוימות:

- הטבלה ממוינת לפי ה-id של הקבצים כך שהקובץ הראשון הוא בעל id 1, השני בעל id 2 וכן הלאה. הטבלה ממוינת כך על מנת לייעל את החיפוש כאשר עוברים על הקבצים.
- העמודה user_id בנוסף לכך מצביעה על עמודת ה-id בטבלת המשתמש(user). בנוסף לכך, כאשר משתמש נמחק מהטבלה user, כל הקבצים בעלי ה-id user יימחקו גם הם. הגדרנו זאת על מנת לקשר בקלות בין כל קובץ לבעליו.

בקשת חברות(friend_request) - אחראית על שמירת בקשות החברות במערכת ונתונים הקשורים אליהן:

- ה-id של הבקשה - מטיפוס int. לדוגמה, 1.
- ה-id של שולח הבקשה - מטיפוס int. לדוגמה, 1.
- ה-id של יעד הבקשה - מטיפוס int. לדוגמה, 1.
- סטטוס הבקשה('ממתינה', 'מאושרת', 'נדחתה') - מטיפוס varchar. לדוגמה, pending.
- תאריך יצירת הבקשה - מטיפוס datetime. לדוגמה, 18:54 2025-04-11.

בנוסף לכך, לשדות מסוימים בטבלה הוגדרו תכונות מסוימות:

- העמודות של שולח הבקשה, נמען הבקשה מצביעות על עמודת ה-id בטבלת המשתמשים(user). בנוסף לכך, כאשר משתמש נמחק מהטבלה user, כל בקשות החברות הקשורות אליו יימחקו גם הן. הגדרנו זאת על מנת לקשר בקלות בין כל בקשה למשתמשים הקשורים אליה.

חברות(friendship) - אחראית על שמירת החברויות במערכת:

- ה-id של החברות - מטיפוס int. לדוגמה, 1.
- ה-id של המשתמש הראשון - מטיפוס int. לדוגמה, 1.

- ה-id של המשתמש השני - מטיפוס int. לדוגמה, 1.
- תאריך אישור הבקשה - מטיפוס datetime. לדוגמה, 2025-04-11 18:54.

בנוסף לכך, לשדות מסוימים בטבלה הוגדרו תכונות מסוימות:

- הטבלה ממוינת לפי ה-id של בקשות החברות כך שבקשת החברות הראשונה הוא בעל id 1, השנייה בעל id 2 וכן הלאה. הטבלה ממוינת כך על מנת לייעל את החיפוש כאשר מחפשים עוברים על הבקשות.
- העמודות של שני המשתמשים החברים מצביעות על עמודת ה-id בטבלת המשתמשים (user). בנוסף לכך, כאשר משתמש נמחק מהטבלה user, כל החבריו הקשורות אליו יימחקו גם הן. הגדרנו זאת על מנת לקשר בקלות בין כל חברות למשתמשים הקשורים אליה.

סקירת חולשות ואיומים:

המתקפות/חולשות המרכזיות שלדעתי שוות התייחסות הן:

1. **sql injection** - במידה ולא מתגוננים מכך בהתאם, מבנה הנתונים עלול להיות רגיש ל-sql injection. עקב כך שאני משתמש ב-sql alchemy לעבודה עם מסד הנתונים בקוד, המערכת מוגנת מ-sql injection עקב כך ש-sql alchemy משתמש בשאילתות sql מובנות כך שהסיכון להזרקת שאילתה זדונית פוחת
2. **cross site request forgery** - התגוננתי ממתקפה זו באמצעות טוקן ייחודי. בכל בקשה לשרת, הטוקן הייחודי צריך להישלח לשרת. משמע, במידה ואתר זדוני מנסה לבצע פעולה במערכת, ללא הטוקן הייחודי(שאינו לו גישה אליו) לא ניתן לבצע פעולות בשרת.
3. **session hijacking** - על מנת למנוע מתקפה זו, הגדרתי את ה-session cookie כך שלא ניתן לגשת אליה מצד שלישי, לא ניתן לגשת אליה באמצעות קוד javascript(only) והן נשלחות רק על גבי https.
4. **MITM** - ברמה עקרונית, כיוון שהמערכת משתמשת ב-ssl, המערכת אמורה להיות מוגנת מ-mitm. עם זאת, עקב כך שתעודת ה-ssl לא מוכרת רשמית ע"י אף רשות(על מנת שהיא תהיה מוכרת זהו תהליך שצורך זמן וכסף) ובמהלך לחיצת היד ב-ssl לא מתבצע אימות שהשרת הוא אכן השרת האמיתי כך שניתן עדיין להתחזות לשרת. עם זאת, במידה ובאמת הייתי משתמש בתעודה מוכרת, הפרויקט היה מוגן ממתקפה זו.

5. **bruteforce** - במהלך תהליך ההתחברות, ישנה הגבלה של 5 ניסיונות התחברות שגויים. לאחר 5 ניסיונות, לא ניתן להתחבר למערכת מאותו מכשיר למשך 15 דקות. עקב מנגנון זה, מתקפת bruteforce אינה אפשרית.

6. **DDoS** - בניגוד לאיומים שהוצגו ממקודם, נגד איום זה לא מימשתי הגנה משמעותית כך שמתקפות מסוג זה הן בהחלט חולשה קיימת. בנוסף לכך, ניתן לבצע מספר פעולות רב במערכת(אין הגבלה של כמות פעולות לפרק זמן מסוים) כך שניתן לנצל את כל משאבי המערכת ולמנוע תפקוד תקין.

הצפנה במערכת - על מנת להצפין את התקשורת במערכת, השתמשתי ב-ssl. במהלך העברת המידע עצמו, משתמשים בהצפנה סימטרית(ספציפית ב-AES). על מנת לפתור את בעיית שיתוף המתפתחות, המערכת משתמשת בפרוטוקול דיפי הלמן.

מימוש הפרויקט:

סקירת המודולים, המחלקות וקשרי הגומלין ביניהם:

סקירת המודולים המיובאים:

- os - עבודה עם מערכת הקבצים(חיבור מיקום של קבצים, מציאת תיקיית העבודה הנוכחית וכו').
- threading - עבודה עם מספר threads במקביל
- flask(וכל תת המודולים תחתיו כגון werkzeug, flask-wtf וכו') - מסגרת עבודה לניהול ניתובים, html rendering, עיבוד טפסים, ניהול sessions וגיבוב סיסמאות.
- sqlalchemy - עבודה עם מסד נתונים
- datetime - טיפול בתאריכים וזמנים
- socket - יצירת חיבור בין 2 רכיבים במערכת(ספציפית השרת המרכזי ושרת אזורי)
- ssl - יצירת חיבור מאובטח בין 2 סוקטים
- json - שליחת מבני נתונים בין סוקטים(סריאליזציה)
- time - השהיית התכנית במקרים מסוימים(למנוע מלולאה לרוץ תמיד)
- uuid - יצירת מזהים ייחודיים לשמירת הקבצים במערכת

מודולים שפיתחתי:

1. User

תפקיד המחלקה - מייצגת משתמש קיים במערכת.

תכונות המחלקה:

- ❖ id - מזהה ייחודי של המשתמש.
- ❖ username - שם המשתמש של המשתמש, גם הוא ייחודי.

- ❖ email - האימייל של המשתמש. גם הוא ייחודי לכל משתמש.
- ❖ password_hash - ערך ה-hash של סיסמת המשתמש (המערכת שומרת את ערך ה-hash של הסיסמה במקום הסיסמה עצמה מטעמי אבטחה).
- ❖ created_at - זמן יצירת המשתמש (תאריך ושעה).
- ❖ used_storage - כמות האחסון שהמשתמש מנצל במערכת (נשמר בביתיים).
- ❖ storage_quota - מכסת האחסון שהמשתמש יכול לנצל.

פעולות המחלקה:

- ❖ set_password(self, password) - פעולה המקבלת סיסמה, מבצעת עליה את פונקציית password_hash ושומרת את הערך בתכונה password_hash.
- טענת כניסה - password, משתנה מסוג string שמייצג את הסיסמה של המשתמש במחלקה.
- טענת יציאה - הפעולה מבצעת על הסיסמה את פונקציית ה-hash ושומרת את הערך בתכונה password_hash. הפעולה לא מחזירה כלום.
- ❖ check_password(self, password) - פעולה המקבלת סיסמה ובודקת האם הסיסמה זהה לסיסמה של המשתמש.
- טענת כניסה - password, משתנה מסוג string שמייצג סיסמה שרוצים לבדוק האם היא של המשתמש.
- טענת יציאה - הפעולה מבצעת את פונקציית ה-hash על הסיסמה, משווה אותה לערך ה-hash של סיסמת המשתמש ומחזירה אמת במידה והסיסמאות זהות ושקר אם אחרת.

2. File

תפקיד המחלקה - מייצגת נתונים של קובץ מסוים (לא כולל תוכן הקובץ עצמו).

תכונות המחלקה:

- ❖ id - מזהה ייחודי של הקובץ.
- ❖ user_id - ה-id של בעל הקובץ.
- ❖ stored_filename - השם הייחודי של הקובץ במערכת הקבצים.
- ❖ original_filename - השם המקורי של הקובץ (השם שבו המשתמש העלה את הקובץ).
- ❖ upload_date - שעת ותאריך העלאת הקובץ למערכת.
- ❖ file_size - גודל הקובץ בביתיים.
- ❖ permissions - הרשאות הגישה לקובץ (פרטי, חברים בלבד, ציבורי).

פעולות המחלקה:

אין למחלקה פעולות מיוחדות.

3. FriendRequest

תפקיד המחלקה- מייצגת בקשת חברות בין 2 משתמשים במערכת.

תכונות המחלקה:

- ❖ id - מזהה ייחודי של הבקשה.
- ❖ from_user_id - ה-id של שולח הבקשה.
- ❖ to_user_id - ה-id של מקבל הבקשה.
- ❖ status - סטטוס הבקשה (ממתין לאישור, אושר, נדחה)
- ❖ created_at - שעת ותאריך יצירת שליחת בקשת החברות

פעולות המחלקה:

אין למחלקה פעולות מיוחדות.

4. Friendship

תפקיד המחלקה- מייצגת חברות בין 2 משתמשים במערכת.

תכונות המחלקה:

- ❖ id - מזהה ייחודי של החברות
- ❖ user_id - ה-id של המשתמש הראשון.
- ❖ friend_id - ה-id של המשתמש השני.
- ❖ created_at - שעת ותאריך יצירת החברות.

פעולות המחלקה:

אין למחלקה פעולות מיוחדות

5. FileManager

תפקיד המחלקה- מקבצת את ניהול הקבצים (שמירת קבצים, מחיקת קבצים, הרשאות גישה וכו') למחלקה.

תכונות המחלקה:

- ❖ upload_folder - נתיב התיקייה בה מאחסנים את הקבצים שמשתמשים מעלים למערכת.

פעולות המחלקה:

- ❖ allowed_file(self, filename) - פעולה המקבלת קובץ ובודקת אם סוג קובץ זה מותר.
 - טענת כניסה - filename, משתנה מסוג string המייצג שם של קובץ (כולל סיומת הקובץ).
 - טענת יציאה - הפעולה בודקת האם סיומת הקובץ מורשית במערכת (ישנן סיומות קבצים שלא מותרות במערכת מטעמי אבטחה) ובמידה והקובץ מורשה מחזירה אמת, אחרת שקר.

- ❖ `generate_unique_filename(self, filename)` - פעולה המקבלת שם של קובץ ומייצרת לו שם ייחודי.
 - טענת כניסה - `filename`, משתנה מסוג `string` המייצג שם של קובץ (כולל סיומת הקובץ).
 - טענת יציאה - הפעולה מייצרת לקובץ שם ייחודי (`uuid`), מוסיפה לשם הייחודי את הסיומת ומחזירה את השם הייחודי.
- ❖ `save_file(self, file_storage, user)` - פעולה המקבלת קובץ ומשתמש, שומרת את הקובץ
 - טענת כניסה - `file_storage`, אובייקט של הקובץ שהועלה דרך flask ו-`user`, אובייקט מסוג `user` המייצג את המשתמש שהעלה את הקובץ.
 - טענת יציאה - הפעולה שומרת את הקובץ בתיקייה, מעדכנת את מסד הנתונים (מוסיפה file חדש), מעדכנת את השדה `used_storage` אצל המשתמש, מכניסה את האירוע לתוך `changes_queue` ומחזירה אובייקט מסוג `file` של הקובץ
- ❖ `list_user_files(self, user)` - פעולה המקבלת משתמש ומחזירה את כל הקבצים השייכים למשתמש.
 - טענת כניסה - `user`, תם.
- ❖ `get_file_record(self, file_id)` - פעולה המקבלת `id` של קובץ ומחזירה את נתוני הקובץ ממסד הנתונים.
 - טענת כניסה - `file_id` משתנה מסוג `int` המייצג `id` של קובץ.
 - טענת יציאה - הפעולה מבצעת שאילתת `sql` על מנת לקבל את הקובץ שה-`id` שלו זהה ל-`id` המתבקש ומחזירה אובייקט מסוג `user` המייצג משתמש.
 - טענת יציאה - הפעולה מבצעת שאילתת `sql` על מנת לסנן את הקבצים השייכים למשתמש ומחזירה או את נתוני הקובץ ממסד הנתונים במידה והוא קיים, ו-`None` אם לא.
- ❖ `delete_file(self, file_record, user)` - פעולה המוחקת קובץ מהמערכת ומעדכנת את מסד הנתונים.
 - טענת כניסה - `file_record`, נתוני הקובץ ממסד הנתונים ו-`user`, אובייקט מסוג משתמש שמייצג משתמש שרוצה למחוק קובץ.
 - טענת יציאה - הפעולה מוחקת את הקובץ הקיים ממערכת הקבצים (במידה ולמשתמש יש גישה למחוק את הקובץ), מסירה את רשומת הקובץ ממסד הנתונים, מכניסה את האירוע לתוך `changes_queue` ומחזירה `true` במידה והמחיקה הצליחה.
- ❖ `update_permissions(self, file_record, new_permissions, user)` - פעולה המקבלת קובץ, משתמש הרוצה לשנות הרשאה של קובץ והרשאה חדשה ומעדכנת את הרשאת הקובץ.
 - טענת כניסה - `file_record`, נתוני הקובץ ממסד הנתונים, `user`, אובייקט מסוג משתמש שמייצג משתמש שרוצה לשנות את הרשאת הקובץ ו-`new_permissions`, משתנה מסוג `string` שמייצג את ההרשאה החדשה לקובץ.

- טענת יציאה - הפעולה משנה את הרשאת הקובץ (במידה ולמשתמש יש גישה לעשות זאת), מעדכנת את נתוני הקובץ במסד הנתונים, מכניסה את האירוע לתוך changes_queue ומחזירה את רשומת הקובץ המעודכנת.

❖ is_access_allowed(self, file_record, user) - פעולה הבודקת האם למשתמש יש גישה לקובץ.

- טענת כניסה - file_record, נתוני הקובץ ממסד הנתונים ו-user, אובייקט מסוג משתמש שמייצג משתמש.
- טענת יציאה - הפעולה בודקת אם למשתמש יש גישה לקובץ, מחזירה אמת במידה וכן ושקר אם אחרת.

6. FriendManager

תפקיד המחלקה - מקבצת את ניהול החברויות, בקשות החברות ופעולות הקשורות לכך (יצירת חברות, מחיקת חברות וכו').

תכונות המחלקה:

אין למחלקה זו תכונות.

פעולות המחלקה:

❖ send_request(self, from_user, to_user) - פעולה המקבלת שני משתמשים ושולחת בקשת חברות מאחד לשני.

- טענת כניסה - from_user, אובייקט מסוג user המייצג את שולח הבקשה ו-to_user, אובייקט מסוג user המייצג את מקבל הבקשה.
- טענת יציאה - הפונקציה בודקת שבקשת החברות חוקית (לא קיימת כבר בקשת חברות ביניהם, המשתמשים לא חברים, שני המשתמשים הם שונים), מעדכנת את מסד הנתונים, מכניסה את האירוע לתוך changes_queue ומחזירה מופע של friendrequest

❖ get_incoming_requests(self, user) - פעולה המקבלת משתמש ומחזירה רשימה של בקשות החברות שנשלחו אל המשתמש.

- טענת כניסה - user, אובייקט מסוג user המייצג את המשתמש שרוצים לראות את בקשות החברות שנשלחו אליו.
- טענת יציאה - הפעולה מבצעת שאילתת sql על מנת לקבל את כל בקשות החברות שהמשתמש שאליהן נשלח הבקשה הוא user ושהסטטוס שלהן הוא "ממתינות לאישור", ומחזירה את הבקשות הללו.

❖ get_outgoing_requests(self, user) - פעולה המקבלת משתמש ומחזירה רשימה של בקשות החברות שהמשתמש שלח.

- טענת כניסה - user, אובייקט מסוג user המייצג את המשתמש שרוצים לראות את בקשות החברות שהוא שלח.
- טענת יציאה - הפעולה מבצעת שאילתת sql על מנת לקבל את כל בקשות החברות שהמשתמש ששלח את הבקשה הוא user ושהסטטוס שלהן הוא "ממתינות לאישור", ומחזירה את הבקשות הללו.

- ❖ `respond_request(self, request_id, accept)` - פעולה המקבלת בקשת חברות והתשובה אליה ומשנה את הסטטוס של הבקשה.
 - טענת כניסה - `request_id`, משתנה מסוג `int` המייצג את ה-id של הבקשה שרוצים לשנות את הסטטוס שלה ו-`accept`, משתנה בוליאני המחזיק `true` במידה והבקשה אושרה ו-`false` במידה והיא נדחתה.
 - טענת יציאה - הפעולה מעדכנת את סטטוס הבקשה במסד הנתונים, במידה והבקשה אושרה מעדכנים את טבלת ה-`friendship`, מכניסה את האירוע לתוך `changes_queue` ומחזירה מופע של `friendrequest`.
- ❖ `get_friends(self, user)` - פעולה המקבלת משתמש ומחזירה את רשימת החברים שלו.
 - טענת כניסה - `user`, אובייקט מסוג `user` המייצג משתמש שרוצה את רשימת חבריו.
 - טענת יציאה - הפעולה מבצעת שאילתת `sql` על מנת לקבל את כל המשתמשים שחברים של המשתמש ומחזירה רשימה של כל החברים.
- ❖ `remove_friend(self, user, to_user)` - הפעולה מקבלת שני משתמשים ומסירה את החברות ביניהם.
 - טענת כניסה - `user`, אובייקט מסוג `user` המייצג את המשתמש שמסיר את הבקשה ו-`to_user`, אובייקט מסוג `user` המייצג את המשתמש שמוסר מהחברים.
 - טענת יציאה - הפעולה מוחקת ממסד הנתונים את החברות בין שני המשתמשים, מכניסה את האירוע לתוך `changes_queue` ומחזירה `true` במידה והחברות הוסרו בהצלחה.

בעיות אלגוריתמיות - פירוט

1. אתחיל דווקא מהבעיה השנייה (בעיית התלות בשרת אחד בודד). הפתרון שלי לבעיה זו היא יצירת שרת מרכזי אחד והקמת מספר שרתים אזוריים שאיתם הלקוח יוצר אינטראקציה. החיבור לשרת זה מתרחש כאשר מריצים את `app.py`, כאשר בקובץ זה התכנית מריצה גם את התוכן של `app.py` וגם את החיבור לשרת המרכזי (באמצעות `multithreading`).

```

3 # Launch the background sync thread exactly once
4 print("[Sync] Launching background sync thread...", flush=True)
5 threading.Thread(target=connect_to_server, daemon=True).start()
6
7 # Start the Flask HTTPS server (self-signed cert for dev)
8 app.run(
9     host='0.0.0.0',
10    debug=False,           # disable debugger in production
11    ssl_context=(certfile, keyfile)
12 )

```

בקובץ `config.py` (יוסבר עליו בהמשך במדריך למשתמש) מוגדר תור (queue) המכיל את השינויים השונים המתרחשים בשרת אזורי.

```
changes_queue = Queue()
```

לתור זה מכניסים אירוע שמתעדכן במסד הנתונים של השרת. האירועים הללו הם: העלאת קובץ, מחיקת קובץ, שינוי הרשאה של קובץ, יצירת משתמש, שליחת בקשת חברות, תגובה לבקשת חברות, הסרת חבר. במימוש של כל אחת מן הפעולות הללו, מכניסים את האירוע הרלוונטי לתור. לדוגמה, בפעולה להסרת הקובץ ממסד הנתונים:

```
def delete_file(self, file_record, user):
    """ Delete a file from disk and remove its DB record. Only the file owner may delete. Enqueues a file_delete sync event. """
    if file_record.user_id != user.id:
        # Prevent unauthorized deletions
        raise PermissionError("You are not authorized to delete this file.")
    # Remove file from disk
    file_path = os.path.join(self.upload_folder, file_record.stored_filename)
    if os.path.exists(file_path):
        os.remove(file_path)
    # Remove DB record
    db.session.delete(file_record)
    db.session.commit()

    # Notify other servers of deletion
    changes_queue.put({
        "type": "file_delete",
        "file_id": file_record.id,
        "timestamp": datetime.now().isoformat()
    })
    return True
```

ניתן לראות בתחתית הפונקציה, מכניסים לתוך התור את הנתונים הרלוונטים לשינוי זה. פעולה זו מתרחשת בכל אירוע הדורש שמירה במסד הנתונים במערכת. דוגמה נוספת לכך ניתן לראות בשינוי ההרשאות של קובץ:

```
def update_permissions(self, file_record, new_permissions, user):
    """ Change the permission of a file (private or public) Only the owner may change permissions. Enqueues a permission_change sync event. """
    if file_record.user_id != user.id:
        # Prevent unauthorized permission changes
        raise PermissionError("You are not authorized to change permissions for this file.")
    file_record.permissions = new_permissions
    db.session.commit()

    # Notify other servers of permission change
    changes_queue.put({
        "type": "permission_change",
        "file_id": file_record.id,
        "new_permissions": file_record.permissions,
        "timestamp": datetime.now().isoformat()
    })
    return file_record
```

ובהוספת קובץ למערכת:

```

# Read file content for sync event
with open(file_path, 'rb') as f:
    content = f.read()

# Enqueue a file_upload event for synchronization
changes_queue.put({
    "type": "file_upload",
    "payload": {
        "id": file_record.id,
        "user_id": file_record.user_id,
        "stored_filename": file_record.stored_filename,
        "original_filename": file_record.original_filename,
        "upload_date": file_record.upload_date.isoformat(),
        "file_size": file_record.file_size,
        "permissions": file_record.permissions,
        "content": content
    },
    "timestamp": datetime.now().isoformat()
})

```

בנוסף לשרת ה-http שרץ ע"י הרצת הקובץ app.py, רץ הקובץ sync.py שאחראי על תקשורת עם השרת המרכזי. בקובץ זה, השרת האזורי מחכה להודעה המורה עליו לשלוח לשרת המרכזי את השינויים שנעשו בשרת זה או הודעה המודיעה על השינויים שנערכו בשרתים אזוריים אחרים.

```

message_type = received_message.get('type')
if message_type == 'send':
    send_changes(sock)
elif message_type == 'receive':
    receive_changes(received_message)

```

אתחיל מהאפשרות הראשונה, המורה על שליחת השינויים לשרת. באפשרות זו, השרת האזורי עובר על כל אירוע שקרה מאז הסנכרון האחרון, מצרף אותו ל-list ושולח לשרת המרכזי. עם זאת, במידה והאירוע שקרה הוא קובץ חדש שהועלה, השרת צריך לבצע עוד מספר פעולות. כאשר שומרים את האירוע של הוספת קובץ בתור, שומרים את תוכן הקובץ כבייטים. על מנת לשלוח את המידע בנוגע לכל אירוע, בחרתי להשתמש בספרייה json, וספרייה זו עובדת עם מחרוזות. לפיכך, צריך להמיר את תוכן הקובץ למחרוזת של ביטים. בשביל כך, ממירים את תוכן הקובץ למחרוזת, מעדכנים את המידע על האירוע ושולחים את האירוע לשרת המרכזי. בסך הכל, פעולה זו נראת כך:

```
# If it's a new file upload, Base64-encode raw bytes for JSON transport
if change['type'] == 'file_upload':
    p = change['payload'].copy()
    raw = p.pop('content') # remove binary data from payload
    # encode bytes to UTF-8 string so it can be embedded in JSON
    p['content'] = base64.b64encode(raw).decode('utf-8')
    events.append({
        'type': change['type'],
        'payload': p,
        'timestamp': change['timestamp']
    })
else:
    # other event types can be sent as-is
    events.append(change)
```

עכשיו, נעבור לאפשרות השנייה, שהיא קבלת השינויים שחלו בשרתים אחרים. באפשרות זו השרת האזורי מקבל את השינויים מהשרת המרכזי (על תהליך זה ארחיב בהמשך), ובהתאם לכל שינוי מבצע את הפעולות הנדרשות. לדוגמה, במידה ונמחק קובץ, השרת האזורי מוחק את הקובץ ממסד הנתונים.

```
elif event_type == 'file_delete':
    rec = file_manager.get_file_record(ev['file_id'])
    if not rec:
        print(f"[Sync] Warn: no file record for deletion ID {ev['file_id']}", flush=True)
        continue
    user = User.query.get(rec.user_id)
    file_manager.delete_file(rec, user)
```

במקרה ושינוי מסוים הוא הוספה של קובץ, צריך לעשות את התהליך ההפוך מהתהליך שביצענו בשליחת השינויים לשרת המרכזי (צריך להמיר את תוכן הקובץ ממחרזת לביטים) וצריך ליצור את הקובץ גם במערכת הקבצים. תהליך זה נראה כך:

```
if event_type == 'file_upload':
    # Decode and write file bytes
    payload = ev['payload']
    data = base64.b64decode(payload['content'])
    dest = os.path.join(UPLOAD_FOLDER, payload['stored_filename'])
    os.makedirs(os.path.dirname(dest), exist_ok=True)
    with open(dest, 'wb') as f:
        f.write(data)
```

עכשיו, אעבור לדון בפתרון הבעיה מצד השרת המרכזי. מטרתו של השרת המרכזי הוא לסנכרן את השינויים המתרחשים בכל שרת אזורי כך שבמידה ומתבצעת פעולה בשרת מרכזי אחד, יהיה אפשרי לגשת לשרת אזורי אחר ואותה פעולה שהתבצעה שם תתבצע גם בשרת הזה. השרת המרכזי מאזין לשרתים אזוריים וכאשר אחד כזה מתחבר, נוצר thread חדש לשרת המרכזי המטפל בשרת אזורי זה.

```

while True:
    try:
        raw_conn, addr = server_sock.accept()
        # Perform TLS handshake on the new connection
        try:
            conn = context.wrap_socket(raw_conn, server_side=True) #Wraps the socket w
            print(f"[GrandServer] Regional connected (TLS): {addr}", flush=True)

        except ssl.SSLError as e:
            print(f"[GrandServer] SSL handshake failed with {addr}: {e}", flush=True)
            raw_conn.close()
            continue

        with clients_lock:
            clients.append(conn)
        # Replay missed events for reconnects
        send_history(conn)
        threading.Thread(target=client_handler, args=(conn,), daemon=True).start() #st

```

כאשר מגיע הזמן לסנכרן את העדכונים, השרת המרכזי שולח הודעה לכל השרתים המחוברים אליו המצביעה על כך. פעולה זו רצה בתוך thread נפרד שתפקידו הוא לדאוג לכך.

```

y.
time.sleep(SYNC_INTERVAL) #Sleeps until the next sync process.
print("[GrandServer] Requesting changes from all regionals...", flush=True)
packet = json.dumps({'type': 'send'}) + '\n'
data = packet.encode('utf-8')

with clients_lock:
    for conn in list(clients):
        try:
            conn.sendall(data)

```

בינתיים, שאר ה-threads מאזינים להודעות מהשרתים האזוריים הכוללות את השינויים.

```

"""Read incoming 'changes' messages
addr = conn.getpeername() #Gets the
f = conn.makefile('r') #Wraps the s
try:
    for line in f:
        #Continuously reads each in
        line = line.strip() #Ignori
        if not line:
            continue

```

התפקיד של השורה השנייה פה הוא לעטוף את הסוקט כסוג של קובץ. היתרונות של עטיפה זו היא יעילות במהלך קריאת נתונים במקום להשתמש ב-recv וניתן לעבור באיטרציה על כל שורה בקובץ. הסיבה שזה רלוונטי לנו היא שכאשר שרת אזורי שולח את השינויים, הוא מפריד כל שינוי בסימן חל שמייצג שורה חדשה. בשימוש בדרך זו ניתן בקלות להפריד בין כל שינוי ובצע פעולות הקשורות אליו. כאשר השרת מקבל את כל השינויים משרת אזורי, הוא שולח הודעה הדומה להודעת broadcast לשאר השרתים המחוברים לשרת המרכזי.

```

packet = json.dumps({'type': 'receive', 'events': events}) + '\n'
data = packet.encode('utf-8')

with clients_lock:
    for conn in list(clients):
        if conn is exclude:
            continue
        try:
            conn.sendall(data)

```

פתרון זה מקטין את התלות בשרת אזורי כלשהו. במידה ושרת אזורי מסוים קורס/מתנתק, עדיין אפשר להתחבר לשרת אזורי אחר ולהשתמש במערכת. על מנת ששרתים אזוריים יוכלו לרוץ גם לאחר שהם קרסו (בלי שיהיו חסרים בהם שינויים מסוימים), כאשר שרת אזורי מתחבר לשרת המרכזי, השרת המרכזי דואג לעדכן את השרת האזורי בכך. השרת המרכזי דואג לשמור כל שינוי שהתרחש ב-5 הסנכרונים האחרונים (מספר זה הוא שרירותי, ניתן לשנות אותו במידת הצורך). כאשר מתחבר שרת אזורי למערכת, השרת המרכזי דואג לשלוח לו את כל השינויים שקרו בזמן זה.


```
def send_history(conn):
    """Sends the last 5 events to a regional_server."""
    for events in history:
        packet = json.dumps({'type': 'receive', 'events': events}) + '\n'
        try:
            conn.sendall(packet.encode('utf-8'))
```

הבעיה המרכזית שנותרה לפתרון זה היא מקרה שבו השרת המרכזי קורס. במקרה זה, לא ניתן לסנכרן את השינויים בין השרתים האזוריים אך כל שרת אזורי פועל כרגיל חוץ ממה שנוגע לסנכרון השרתים האחרים. על מנת לפתור בעיה זו, ניתן ליצור מערכת של כמה שרתים מרכזיים שמתקשרים ביניהם, כך שבמידה ושרת מרכזי אחד נופל, השרתים המרכזיים האחרים יכולים לכפות עליהם. את הפתרון הזה לא מימשתי, אך פתרון זה הוא יחסית פשוט ודורש הקמה של רשת סוקטים קטנה (ולא עודד לוגיקה/אלגוריתם מיוחדת). הפונקציות המלאות:

השרת האזורי:

2. אעבור לבעיה שנשארה, שהיא מבנה מערכת הקבצים. בכל שרת אזורי ישנה תיקייה בשם uploads שכוללת את הקבצים הקיימים על השרת (הקבצים של כל המשתמשים מופיעים בתיקייה זו). על מנת שלא יהיו התנגשויות בין שמות קבצים (ייתכן מצב שלשני משתמשים שונים יש קובץ בעל אותו שם), לכל קובץ יש שם מיוחד. את השם המיוחד הזה מייצרים על ידי uuid (מזהה ייחודי לכל קובץ שאורכו 128 ביטים). הסיכוי להתנגשות בין שמות של שני קבצים שואף ל-0 (אחד לשניים בחזקת 128). בנוסף לכך, לכל קובץ מצרפים למזהה זה את סיומת הקובץ.

```
def generate_unique_filename(self, filename):
    """ Generate a unique filename using UUID4 and preserve the file extension
    ext = filename.rsplit('.', 1)[1] if '.' in filename else ''
    # Combine a random hex string with the original extension
    unique_name = f"{uuid.uuid4().hex}.{ext}" if ext else uuid.uuid4().hex
    return unique_name
```

כתוצאה מכך, כך תיקיית הקבצים עשויה להיראות:

```

v uploads
  3f8614b1efe5467cb6f0d2a38a01cd1f
  cc2057b9db484cafae50405b40affca4.pdf
```

כאשר רוצים לגשת לקובץ מסוים במערכת, במסד הנתונים רשום לכל קובץ גם את השם המקורי שלו וגם את השם שבו הוא נשמר בזיכרון, ולכן אין שום בעיה למצוא את השם המקורי של הקובץ באמצעות השם במערכת הקבצים ולהפך.

תיאור הבדיקות במערכת:

מספר הבדיקה	שם הבדיקה	מה הבדיקה אמורה לבדוק	איך מתכננים לבדוק	תוצאות הבדיקה
1	התחברות והרשמה למערכת	האם ניתן להתחבר ולהירשם למערכת	הרשמה למערכת ולאחר מכן ניסיון התחברות	עבד
2	טיפול בלקוח אחד	האם המערכת מסוגלת לטפל בפעולות הבסיסיות של לקוח אחד	ביצוע הפעולות הבאות - העלאת קובץ, מחיקת קובץ, שינוי הרשאת קובץ, הורדת קובץ, שליחת בקשת חברות, אישור בקשה וצפייה בקבצים של חבר	תחילה רוב הפעולות עברו בהצלחה חוץ מהפעולה של צפייה בקבצים של חבר. עקב כך נאלצתי לשנות את הבדיקה לאיזה קבצים משתמש יכול לגשת ולאחר זאת זה עבד
3	התחברות ממכשיר חיצוני	האם המערכת מסוגלת לטפל בפעולות הבסיסיות של לקוח אחד	ביצוע אותן פעולות על גבי מחשב שונה כאשר השרת רץ על אותו מכשיר	עבד

4	טיפול במסד הנתונים ומערכת הקבצים	האם השינויים שנעשו נשמרים במסד הנתונים ומערכת הקבצים	לאחר ביצוע הפעולות, אסגור את השרת ואבדוק אם הקבצים הרלוונטיים + הפעולות נשמרו	עבד
5	הצפנת המערכת	האם המערכת מסוגלת להצפין את התקשורת	הסנפה באמצעות wireshark של תעבורת המערכת	תחילה לא עבד עקב כך שהתעודה לא מהימנה והמחשב שלי חסם את האתר. על מנת לפתור את זה, הייתי צריך לאשר את התעודה במכשיר באמצעות שימוש ב-certmgr, msc. אישרתי את התעודה ואז המערכת חזרה לעבוד. לאחר הסנפה, ניתן היה לראות שהמערכת מוצפנת

6	נסיונות התחברות מתמשכים	האם המערכת מסוגלת להתמודד עם bruteforce	אנסה להיכנס למערכת מספר פעמים ברציפות ואראה אם ישנה חסימה לאחר מספר נסיונות מסוים	עבד
7	סנכרון שינויים בין שרתים אזוריים	האם המערכת מצליחה לסנכרן שינויים בין מספר שרתים אזוריים הפועלים בו זמנית	אריץ שני שרתים אזוריים משני מכשירים שונים, אבצע שינוי באחד מהם ואראה אם הוא מתעדכן בשני	תחילה הסנכרון לא עבד עקב כך שכל שרת אזורי התחבר פעמיים לשרת המרכזי. הסיבה לכך היא שה-reload r של flask מריץ 2 threads, וכתוצאה מכך נוצרים threads 2 שמתחברים לשרת המרכזי. כאשר מבטלים את ה-reloader, נוצר רק thread אחד והמערכת עובדת

8	sql injection	האם המערכת מוגנת מ-sql injection	באמצעות כלי בשם sqlmap, אריץ שאילות sql זדוניות ואראה אם המערכת מוגנת מכך	עבד
9	קריסה של השרת המרכזי	האם השרתים האזוריים מסוגלים לתפקד במצב שבו השרת המרכזי קורס	אריץ את השרתים האזוריים ואכבה את השרת המרכזי, ולאחר מכן אבצע פעולות על מנת לראות אם השרת האזורי מתפקד	עבד
10	התחברות מאוחרת של שרת אזורי	האם שרת אזורי שהתחבר באיחור/אחרי קריסה מסוגל להשלים את השינויים שהתרחשו בזמן זה	ביצוע מספר שינויים בשרת אזורי, ולאחר זמן מה הפעלה של שרת אזורי חדש. אראה אם השינויים שהתרחשו בשרת האזורי הוטמעו גם בשרת החדש	עבד

בדיקות נוספות שביצעתי למערכת:

מספר הבדיקה	שם הבדיקה	מה הבדיקה אמורה לבדוק	איך מתכננים לבדוק	תוצאות הבדיקה
11	הרשאות גישה לקובץ ציבורי	בדיקה האם משתמש שאינו הבעלים של קובץ יכול להוריד אותו אך לא יכול לשנות כלום לגביו	העלאת קובץ, הגדרתו כציבורי והתחברות ממשתמש אחר שהוא חבר של המשתמש	עבד
12	שימוש במערכת בטלפון	בדיקה האם מכשיר שהוא לא מחשב (כמו טלפון לדוגמה) יכול להתחבר למערכת	הרצת שרת אזורי על מחשב והתחברות למערכת עם טלפון	עבד, אך כמצופה המערכת לא עובדת בצורה אופטימלית עם טלפון (חוויית המשתמש אינה אופטימלית)

מדריך למשתמש:

פירוט כלל קבצי המערכת:

static	4/25/2025 6:18 PM	File folder	
templates	5/1/2025 11:02 PM	File folder	
uploads	5/1/2025 10:48 PM	File folder	
__init__.py	4/27/2025 12:21 AM	Python Source File	0 KB
app.py	5/1/2025 10:47 PM	Python Source File	4 KB
auth.py	5/1/2025 9:57 PM	Python Source File	5 KB
config.py	5/1/2025 10:00 PM	Python Source File	1 KB
database.db	5/1/2025 11:27 PM	Data Base File	28 KB
file_management.py	5/1/2025 11:18 PM	Python Source File	6 KB
file_management_routes.py	5/1/2025 11:09 PM	Python Source File	4 KB
friend_management.py	5/1/2025 11:18 PM	Python Source File	5 KB
friend_management_routes.py	5/1/2025 11:16 PM	Python Source File	5 KB
grand_server.py	5/1/2025 9:55 PM	Python Source File	7 KB
models.py	5/1/2025 9:48 PM	Python Source File	3 KB
server.crt	4/30/2025 4:39 PM	Security Certificate	2 KB
server.key	4/30/2025 4:39 PM	KEY File	2 KB
sync.py	5/2/2025 1:48 AM	Python Source File	7 KB

- app.py - הקובץ הראשי שמקים את המערכת (מקים את האתר, מגדיר את החיבור למסד הנתונים וכו'). מריץ בתוכו גם את sync.py.
- auth.py - אחראי על האותנטיקציה של משתמשים (הרשמה והתחברות).
- config.py - מרכז את הקבועים וההגדרות של המערכת.
- database.db - מסד הנתונים של המערכת.
- file_management.py - מכיל את המחלקה filemanager ואחראי על לוגיקת הקבצים.
- file_management_routes.py - ממפה את הפעולות על הקבצים ל-http.
- friend_management.py - מכיל את המחלקה friendmanager ואחראי על לוגיקת החברויות.
- friend_management_routes.py - ממפה את הפעולות הקשורות לחברויות ל-http.
- grand_server.py - השרת המרכזי האחראי על סנכרון המערכת.
- models.py - הגדרת מודלים כגון user, file וכו'.
- server.crt - תעודת ה-ssl של השרת.
- server.key - המפתח הציבורי של השרת.
- sync.py - אחראי על סנכרון השינויים עם השרת המרכזי.

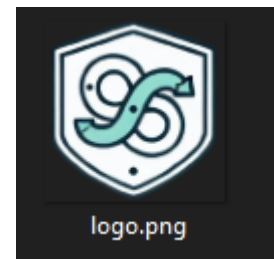
תוכן התיקייה static:

css	5/1/2025 11:19 PM	File folder
images	4/25/2025 6:18 PM	File folder

תוכן התיקייה CSS:

# auth.css	5/1/2025 11:25 PM	CSS Source File	4 KB
# dashboard.css	5/1/2025 11:27 PM	CSS Source File	5 KB

- auth.css - קובץ CSS המופיע בדפי ההתחברות וההרשמה.
- dashboard.css - קובץ CSS המופיע בשאר הדפים (אלה שאפשר לגשת אליהם רק לאחר התחברות).

תוכן התיקייה images:**תוכן התיקייה templates:**

dashboard.html	5/1/2025 11:22 PM	Chrome HTML Do...	4 KB
friends_files.html	5/1/2025 3:47 PM	Chrome HTML Do...	2 KB
friends_list.html	5/1/2025 10:56 PM	Chrome HTML Do...	3 KB
incoming_requests.html	5/1/2025 11:02 PM	Chrome HTML Do...	3 KB
login.html	5/1/2025 11:19 PM	Chrome HTML Do...	2 KB
register.html	5/1/2025 11:19 PM	Chrome HTML Do...	2 KB

- dashboard.html - דף ה-HTML של מסך הבית.
- friends_files.html - דף ה-HTML של מסך הקבצים של חבר.
- friends_list.html - דף ה-HTML של רשימת החברים.
- incoming_requests.html - דף ה-HTML של בקשות החברות.
- login.html - דף ה-HTML של מסך ההתחברות.
- register.html - דף ה-HTML של מסך ההרשמה.

תוכן התיקייה uploads הוא הקבצים המועלים למערכת. דוגמה:

3f8614b1efe5467cb6f0d2a38a01cd1f	5/1/2025 10:47 PM	File	297 KB
cc2057b9db484cafae50405b40affca4.pdf	5/1/2025 10:47 PM	Microsoft Edge P...	88 KB

התקנת המערכת:

על מנת להריץ את השרת, צריך לוודא את הדברים הבאים:

- מותקן code editor כגון visual studio code או pycharm.
- מותקן על המכשיר python 3.12 ומעלה.
- הספריות הבאות מותקנות על המכשיר:

❖ flask

❖ flask-wtf

❖ SQLAlchemy

❖ Werkzeug

ניתן להתקין את ספריות אלו באמצעות הרצת הפקודה בטרמינל:

```
pip install Flask Flask-WTF SQLAlchemy Werkzeug
```

- למחשב ישנו חיבור לאינטרנט.
- בנוסף לכך, מומלץ לשנות קבועים מסוימים ב-config.py כגון המיקום של תיקיית הקבצים, הכתובת של השרת המרכזי, מיקום תעודת ה-ssl והמפתח הציבורי, סוגי הקבצים המתאפשרים במערכת וכו'.

במידה ורוצים רק להתחבר למערכת (ולהסתמך על שרת אזורי שכבר רץ), הדרישות הללו לא רלוונטיות.

הפעלת המערכת:

כעת, צריך לוודא שיש שרת מרכזי הפועל. במידה ואין, ניתן להריץ אותו כאשר מריצים את הקובץ grand_server.py.

דרך visual studio code, ניתן להריץ את השרת באמצעות לחיצה ימנית על הקובץ ולחיצה על "run code" או כתיבת הפקודה ב-terminal:

```
python grand_server.py
```

Run Code	Ctrl+Alt+N
Open to the Side	Ctrl+Enter
Open With...	
Reveal in File Explorer	Shift+Alt+R
Open in Integrated Terminal	
Share	>
Select for Compare	
Open Timeline	
Cut	Ctrl+X
Copy	Ctrl+C
Copy Path	Shift+Alt+C
Copy Relative Path	Ctrl+K Ctrl+Shift+C
Run Tests	
Debug Tests	
Run Tests with Coverage	
Rename...	F2
Delete	Delete
Run Python File in Terminal	

במידה והכל רץ כשורה, יופיע ב-terminal:

```
[GrandServer] Listening on 0.0.0.0:9000
```

כעת, ניתן להריץ את אחד מן השרתים האזוריים. על מנת להריץ שרת זה, צריך להריץ את הקובץ `app.py`. ניתן להריצו באמצעות אותו כפתור ב-visual studio code (יש ללחוץ קליק ימני על הקובץ `app.py` כעת) או להריץ את השורה ב-terminal:

```
python app.py
```

במידה והכל התבצע כשורה, יופיע ב-terminal:

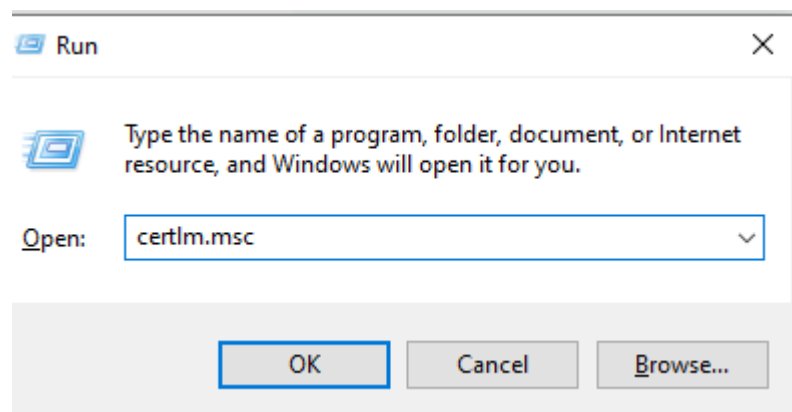
```
[Sync] Launching background sync thread...
* Serving Flask app 'app'
* Debug mode: off
[Sync] Connected to grand server
WARNING: This is a development server. Do not use it in a production deployment. Use
SGI server instead.
* Running on all addresses (0.0.0.0)
* Running on https://127.0.0.1:5000
* Running on https://10.0.0.3:5000
Press CTRL+C to quit
```

וב-terminal של השרת המרכזי יופיע:

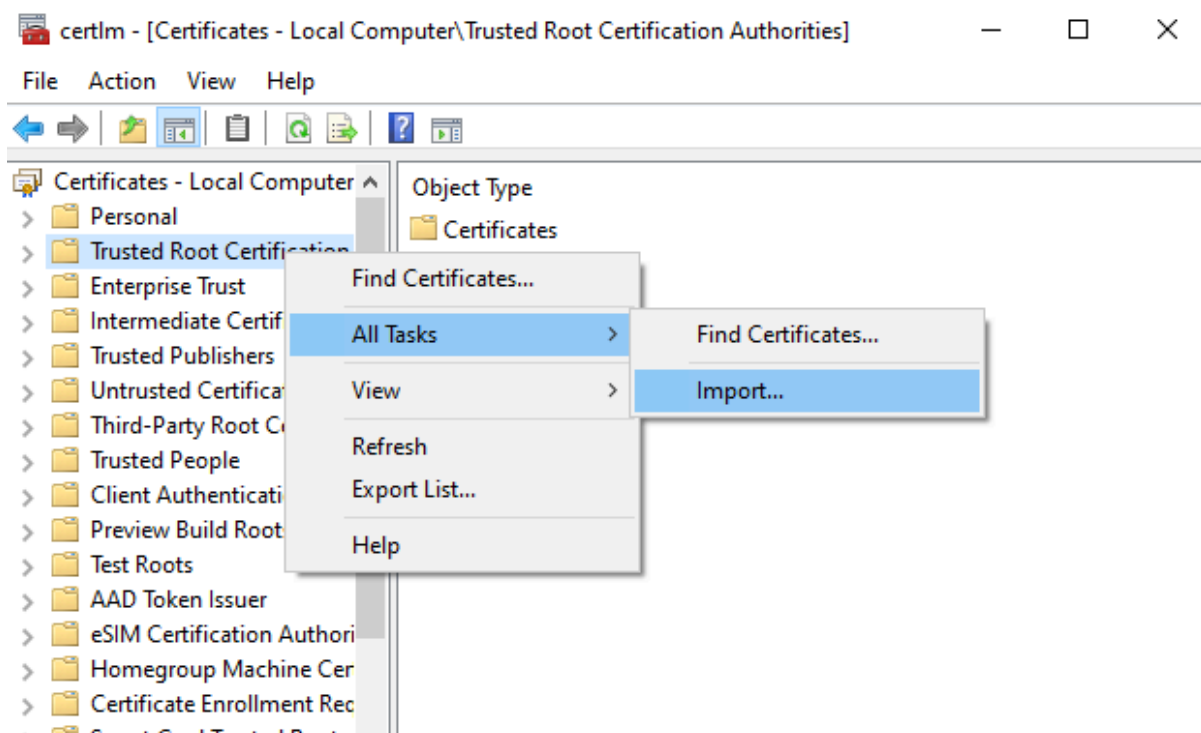
```
[GrandServer] Regional connected (TLS): ('10.0.0.3', 56279)
```

לאחר שבוצעו כל השלבים האלו, ניתן להתחבר למערכת באמצעות הדפדפן. קודם כל, כיוון שתעודת ה-ssl של השרת אינה רשמית, צריך לוודא שהתעודה מוכרת על ידי המכשיר שמשתמשים בו. במידה ולשרת יש תעודת ssl רשמית, ניתן לדלג על החלק הבא:

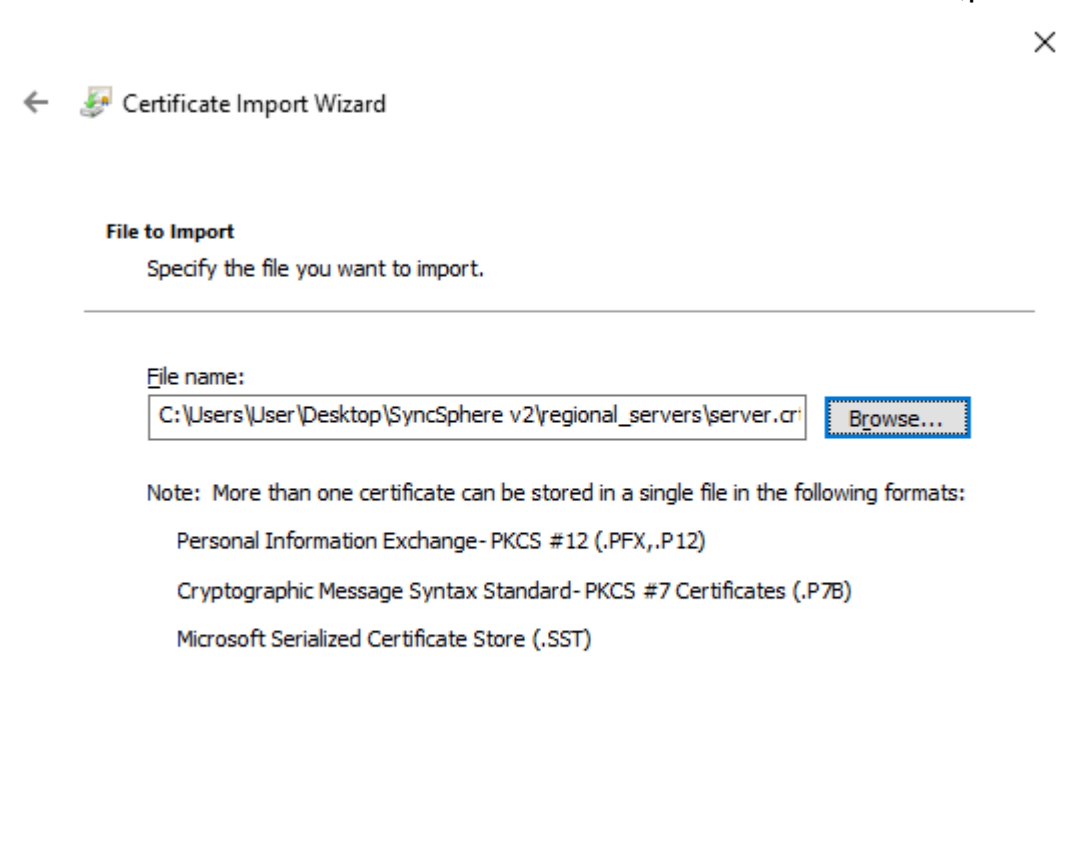
על מנת להתחבר במכשיר מסוים צריך את תעודת ה-ssl על המכשיר. ניתן ללחוץ עכשיו על + win R, בלחפש בלשונית החיפוש certlm.msc.



לאחר שלוחצים על הכפתור 'ok' ומאפשרים לתוכנה לבצע שינויים במחשב, לוחצים מקש ימני על "Trusted root certification authorities", שמים את העכבר על "all tasks" ולוחצים על "import".



לאחר מכן, לוחצים על "next" ובלשונית החיפוש שמים את תעודת ה-ssl.



לאחר מכן, לוחצים פעמיים על "next" ולבסוף על "finish". סוגרים את כל החלונות שפתחנו ופותחים מחדש את הדפדפן.

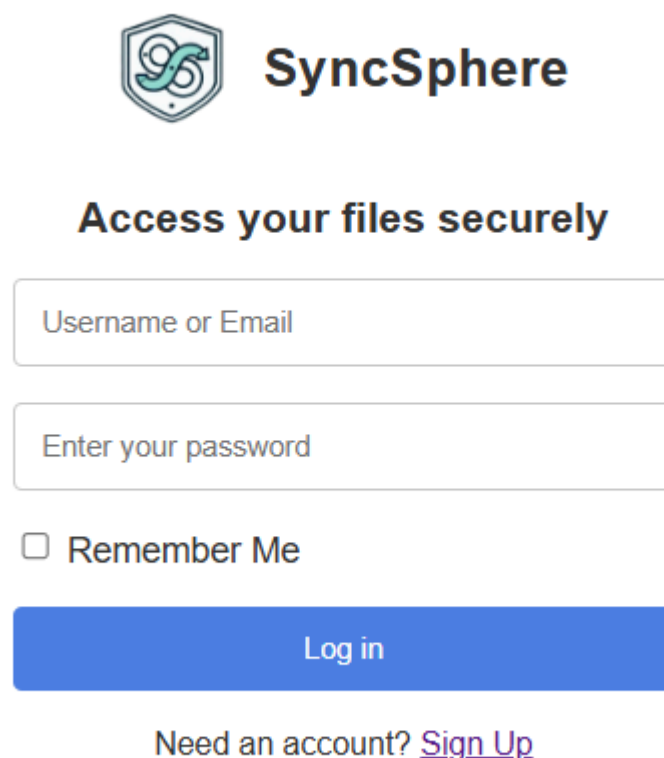
(מחלק זה ניתן להמשיך למי שיש תעודת ssl רשמית):

על מנת להתחבר, צריך לכתוב את ה-url הבא: `https://(server_ip):(server_port)` כאשר `server_ip` הוא כתובת ה-ip של השרת ו-`server_port` הוא הפורט שהשרת רץ עליו. דוגמה לכתובת כאשר ip השרת הוא 100.100.100.100 והפורט הוא 5000 יהיה:

<https://100.100.100.100:5000>

שימוש במערכת:

לאחר כניסה לכתובת ה-url, יופיע העמוד הבא:



The image shows the SyncSphere login page. At the top is the SyncSphere logo, which consists of a shield icon with a stylized 'S' and the text 'SyncSphere' next to it. Below the logo is the heading 'Access your files securely'. There are two input fields: the first is labeled 'Username or Email' and the second is labeled 'Enter your password'. Below these fields is a checkbox labeled 'Remember Me'. At the bottom of the login section is a blue button labeled 'Log in'. Below the button is the text 'Need an account? [Sign Up](#)'.

על מנת להשתמש במערכת, צריך משתמש פעיל. על מנת ליצור משתמש, ניתן ללחוץ על הכפתור "sign up".



SyncSphere

Access your files securely

☐ Remember Me

Log in

Need an account? [Sign Up](#)

לאחר לחיצה על כפתור זה, יופיע המסך הבא:



SyncSphere

Create your account

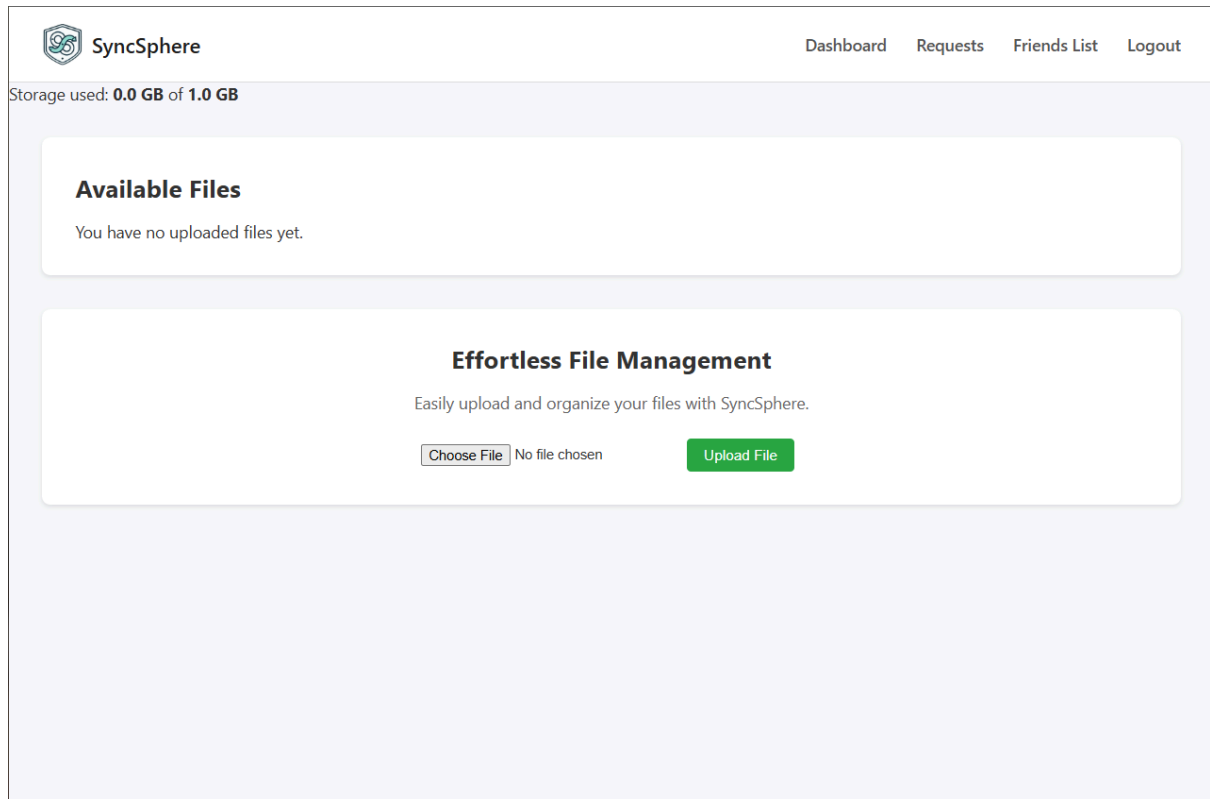
Register

Already have an account? [Log In](#)

עכשיו, על מנת ליצור משתמש יש למלא את הפרטים בשדות המתאימים לכך.

במידה והפרטים תקינים (כתובת אימייל תקינה וייחודית, שם משתמש ייחודי), המשתמש יוצר בהצלחה ומסך ההתחברות יקפוץ בחזרה. כעת, נותר למלא את הפרטים של המשתמש וללחוץ על כפתור ההתחברות.


במידה וההתחברות עברה בהצלחה, המשתמש יופנה לדף הבא:



למעלה בצד ימין, יש ארבעה כפתורים:

- dashboard לחזרה למסך הבית
- requests למעבר למסך בקשות החברות
- friends list למעבר למסך החברים
- logout להתנתקות מהמערכת

בצד העליון השמאלי של המסך, ניתן לראות את כמות האחסון שנוצלה במערכת ואת כמות האחסון המקסימלית.

 SyncSphere

DashboardRequestsFriends ListLogout

Storage used: 0.0 GB of 1.0 GB

Available Files


You have no uploaded files yet.

Effortless File Management

Easily upload and organize your files with SyncSphere.

No file chosen

כעת, על מנת להעלות קובץ למערכת, יש ללחוץ על הכפתור באמצע המסך "choose file".

 SyncSphere

DashboardRequestsFriends ListLogout

Storage used: 0.0 GB of 1.0 GB

Available Files

You have no uploaded files yet.

Effortless File Management

Easily upload and organize your files with SyncSphere.

No file chosen


כעת, ניתן למצוא את הקובץ הרצוי במערכת הקבצים. לאחר שמוצאים את הקובץ, צריך ללחוץ על הכפתור "upload file".

Effortless File Management

Easily upload and organize your files with SyncSphere.

test.txt

במידה והקובץ תקין (אין חריגה מכמות הזיכרון, סיומת מאושרת בשרת האזורי, קובץ קיים) תופיע הודעה מתאימה והקובץ יופיע על המסך.

 SyncSphere

DashboardRequestsFriends ListLogout

Storage used: 0.0 GB of 1.0 GB

Available Files

Filename	Upload Date	Size (bytes)	Actions
test.txt	2025-05-01 20:40	0	Download Delete Private Update

Effortless File Management

Easily upload and organize your files with SyncSphere.

[Choose File](#) No file chosen [Upload File](#)

File uploaded successfully!

- כעת, ניתן לראות את שם הקובץ, תאריך ההעלאה למערכת וגודל האחסון שהוא צורך. כעת, ניתן לבצע מספר פעולות הקשורות לקובץ:
 - ניתן להוריד את הקובץ למכשיר בלחיצה על הכפתור "download":

Available Files

Filename	Upload Date	Size (bytes)	Actions
test.txt	2025-05-01 20:40	0	Download Delete Private Update

- ניתן למחוק את הקובץ מן המערכת בלחיצה על הכפתור "delete":

Available Files

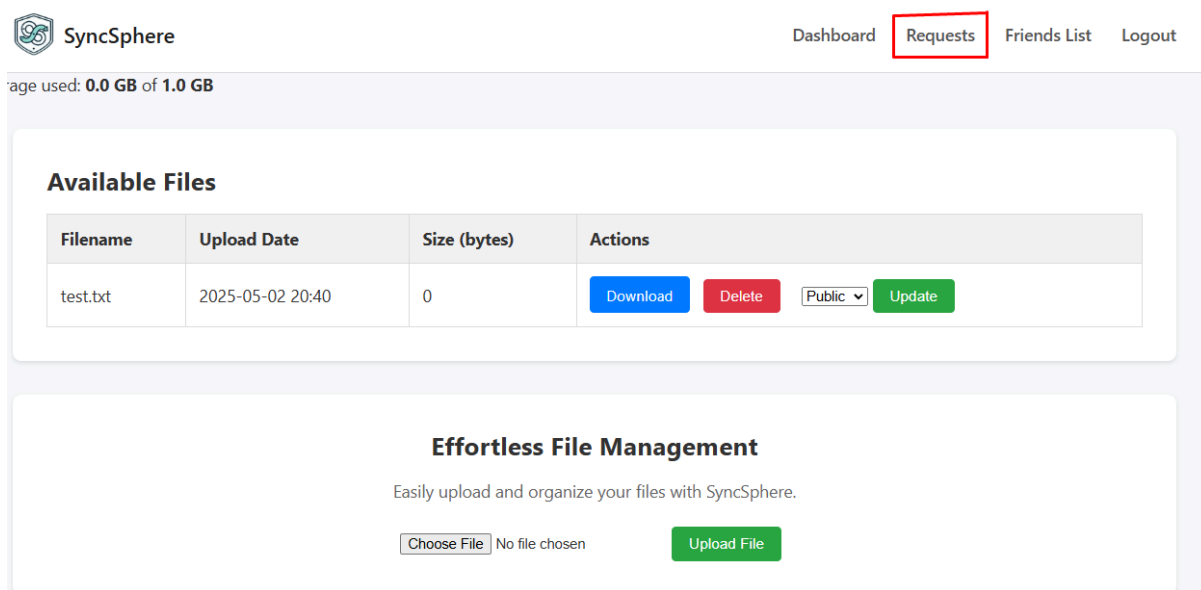
Filename	Upload Date	Size (bytes)	Actions
test.txt	2025-05-01 20:40	0	Download Delete Private Update

- ניתן לשנות את הרשאת הצפייה בקובץ בלחיצה על הלשונית ליד הרשאת הגישה, הגדרת ההרשאה הרצויה ואז הכפתור "update"

Available Files

Filename	Upload Date	Size (bytes)	Actions
test.txt	2025-05-01 20:40	0	Download Delete Private Update

במידה ורוצים להוסיף חבר למערכת, ניתן לעשות זאת באמצעות מעבר למסך בקשות החברות:



SyncSphere

Dashboard **Requests** Friends List Logout

Storage used: 0.0 GB of 1.0 GB

Available Files

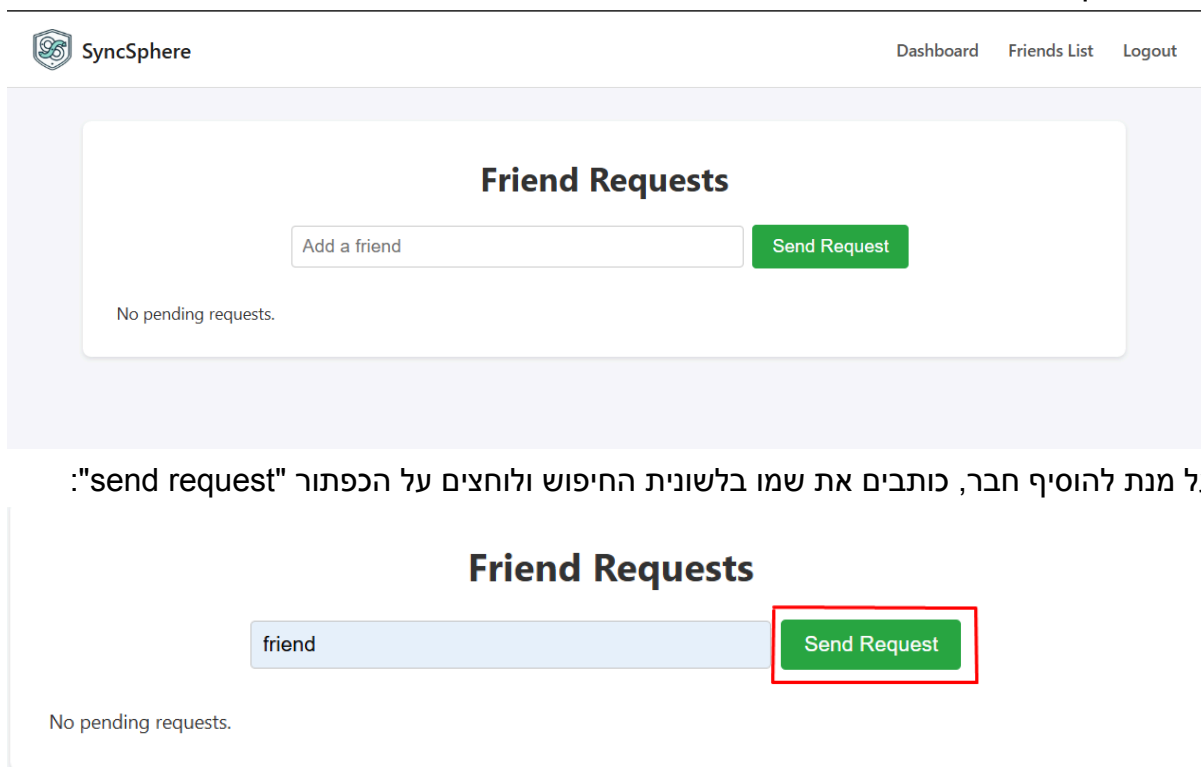
Filename	Upload Date	Size (bytes)	Actions
test.txt	2025-05-02 20:40	0	Download Delete Public Update

Effortless File Management

Easily upload and organize your files with SyncSphere.

No file chosen

כעת, המסך הבא יופיע:



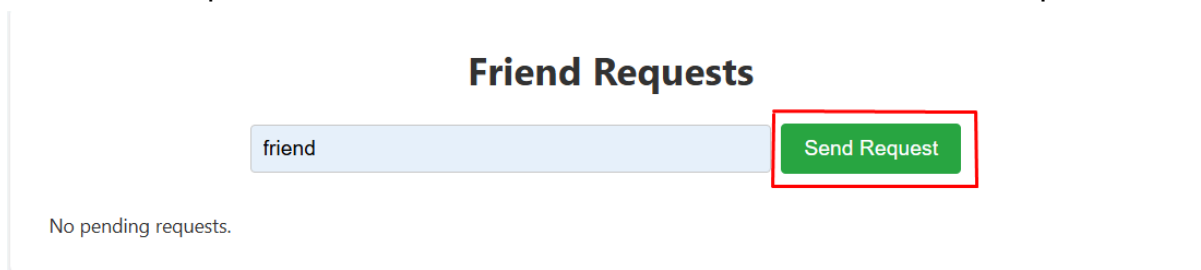
SyncSphere

Dashboard Friends List Logout

Friend Requests

No pending requests.

על מנת להוסיף חבר, כותבים את שמו בלשונית החיפוש ולוחצים על הכפתור "send request":



Friend Requests

No pending requests.

במידה והבקשה תקינה, תופיע על כך הודעה ובקשת החברות תופיע על המסך:

Friend Requests

Send Request

Your Pending Requests

test
@test

No pending requests.

Friend request sent!

בקשות חברות שנשלחו למשתמש יופיעו גם הן במסך זה. במידה ורוצים לאשר/לדחות בקשה, יש ללחוץ על הכפתור המתאים לכך:

Friend Requests

Send Request

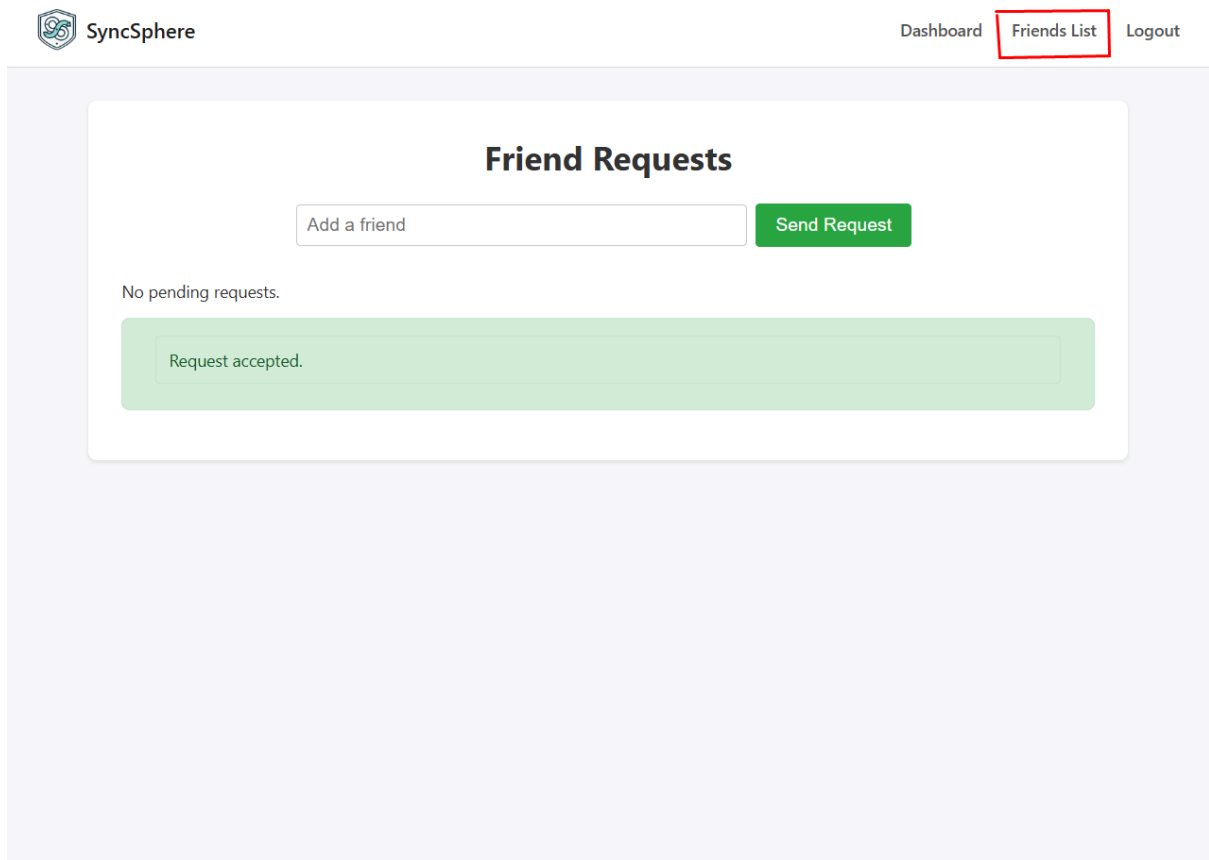
Incoming Requests

test
@test

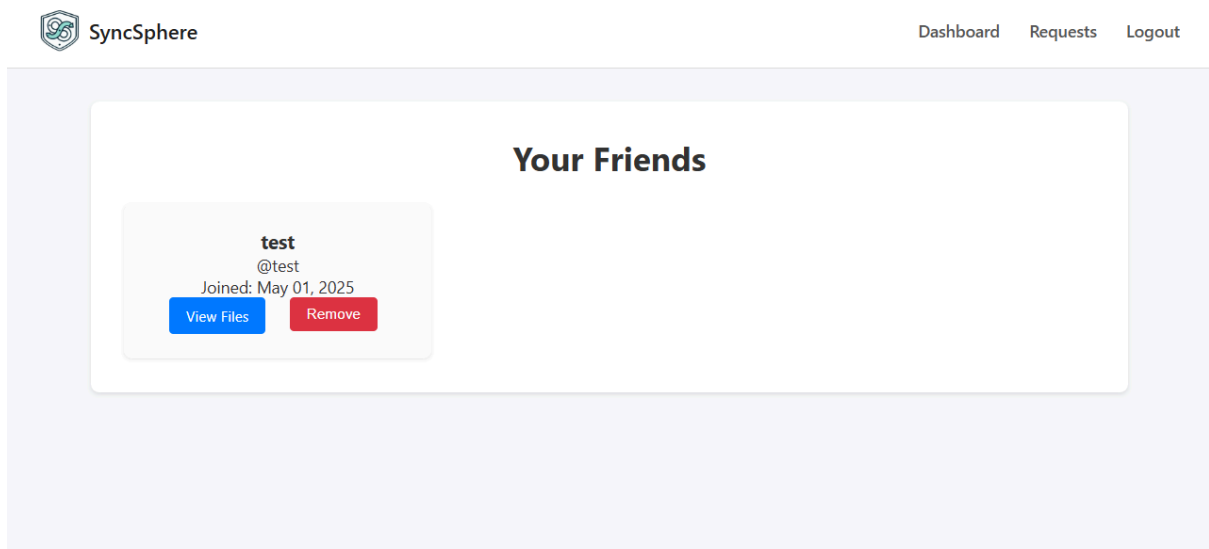
Accept

Reject

כעת, כדי לראות את החברים של המשתמש, ניתן לעבור למסך החברים המופיע בפינה הימנית למעלה בדף (ניתן לעבור לדף זה גם מדף הבית בלחיצה על אותו כפתור):



כעת, בדף זה יופיעו כל החברים של המשתמש:



על מנת להסיר חבר במידת הצורך, ניתן ללחוץ על כפתור ה-"remove" המופיע ליד שם החבר.

Your Friends

test

@test

Joined: May 01, 2025

View Files

Remove

על מנת לצפות בקבצים של חבר, ניתן ללחוץ על הכפתור "view files".

Your Friends

test


@test

Joined: May 01, 2025

View Files

Remove

לאחר לחיצה על כפתור זה, יופיעו קבצי המשתמש, נתונים הקשורים אליהם ואפשרות להורדה.

 SyncSphere

Dashboard Friends Logout

test's Files

Filename	Upload Date	Size	Actions
test.txt	2025-05-01 20:40	0	Download


שים לב - ניתן לראות רק את הקבצים שהרשאות הגישה שלהם היא "public".

במידה ורוצים להוריד את הקובץ, ניתן ללחוץ על הכפתור "download":

test's Files

Filename	Upload Date	Size	Actions
test.txt	2025-05-01 20:40	0	Download

במידה ורוצים להתנתק מהמערכת, ניתן ללחוץ על הכפתור "logout" המופיע בפינה הימנית העליונה של כל עמוד:

 SyncSphere

Dashboard Friends **Logout**

test's Files

Filename	Upload Date	Size	Actions
test.txt	2025-05-01 20:40	0	Download

כעת אפשר להשתמש במערכת ולתפעל אותה בפשטות!

סיכום אישי - רפלקציה:

אני אתחיל מהסוף, תהליך העבודה על הפרויקט היה מהנה ומאתגר בו זמנית. אתחיל מלציין שזוהי הפעם הראשונה שאני עובד על פרויקט בסדר גודל כזה, כך שתחילה לא ידעתי כיצד לגשת לכל התהליך. תחילה רציתי להקדיש זמן רב לתכנון המערכת והפרויקט ככלל, על מנת למנוע בזבוז זמן מיותר בעבודה על הפרויקט. עם זאת, אני חושב שתכנון הפרויקט שלי לקה בחסר. לא ידעתי באמת להעריך בצורה טובה פרויקט בסדר גודל כזה, כך שלא הקדשתי זמן רב לדברים שבפועל לקחו לי זמן גדול. לדוגמה, בתכנון הפרויקט הקדשתי הרבה פחות זמן ממה שלקח בפועל לעצב את ממשק המשתמש ואת דפי האינטרנט. דווקא לחלק זה כן התחברתי, ומצאתי את עצמי הרבה פעמים לא מסתפק בעיצוב מסוים לדף שלא מצא חן בעיניי. במהלך פרויקט זה נדרש ממני המון למידה עצמית בתחומים שונים שפחות הכרתי. הבולטים בהם היו שימוש בספרייה flask ובעיצוב דפי אינטרנט באמצעות html-ו css. אני מרגיש שהסיבה המרכזית לכך שהצלחתי ללמוד את נושא זה הייתה היעזרות בחברים המכירים כבר את הנושא (חלקם עבדו על עיצוב אתרים בכיתה י' לדוגמה) כך שבמידה ולא הבנתי משהו, הם הרגישו חופשי לעזור לי ולהסביר במידת הצורך. חוץ מכך, אני מרגיש שנהייתי מהלמידה העצמית, גם אם היו קשיים מסוימים בחלק מהנושאים (כגון עבודה עם מסד הנתונים). בסה"כ, אני מרגיש שהפרויקט תרם לי רבות בפיתוח יכולות למידה עצמית וחקר באינטרנט. דבר משמעותי שהוריד לי מוטיבציה במהלך העבודה על הפרויקט היה שאיפת יתר. בתכנון המערכת היו פיצ'רים מסוימים שרציתי לכלול חוץ מהחיוניים שקיימים במערכת, אך את חלקם לא הצלחתי/הספקתי לעבוד עליהם בצורה מספיק טובה על מנת להוסיף אותם. במהלך כתיבת הקוד עצמו, היו חלקים שיותר נהניתי ופחות נהניתי. אישית נהניתי ממחשבה על האלגוריתם של סנכרון השרתים השונים ונהניתי מלחשוב על אופטימיזציות שונות שיכולתי לבצע בקשר לאלגוריתם זה. תהליך כתיבת תיקי הפרויקט גם היה לי קשה בהתחלה, שכן כמו הפרויקט עצמו אין לי ניסיון רב בעיצוב תיקי פרויקט בסדר גודל כזה. מחלק זה דווקא פחות נהניתי שכן אני מרגיש שתהליך הכתיבה חזר על עצמו לעיתים קרובות והרגשתי שחזרתי על דברים שכתבתי בחלקים מוקדמים יותר של הפרויקט מספר פעמים. בראייה לאחור, אני חושב שהייתי מתכנן את הזמן שלי יותר בזהירות והייתי מקצה יותר זמן ממה שאני חושב שהיה לוקח לי למקרה והייתי נתקל בבעיות בדרך (כמו שקרה בפועל). אחד מן הכלים הכי חשובים שאני לוקח מהפרויקט הוא התמדה במשהו אחד לאורך זמן. בעבר כשעבדתי על פרויקטים עבודות שונות מספר פעמים רב עבדתי על כמה פרויקטים במקביל/הפרויקטים לקחו כמות זמן קצרה יחסית כך שאף פעם באמת לא יצא לי לעבוד על משהו מסוים לאורך כל כך הרבה זמן. בנוסף לכך, אני מרגיש שיכולות הלמידה העצמית והתכנון לטווח ארוך שלי השתפרו עקב העבודה בפרויקט. במידה מסוימת אני מרגיש שכן היה לי את הכישורים הללו ברמה מסוימת, אך עקב הגודל של הפרויקט איתגרתי את עצמי גם בתחומים אלה. במבט לאחור, אני חושב שהייתי מתחיל מלוגיקת המערכת לפני העבודה על ממשק המשתמש וכו'. אני אישית חושב שלוגיקת המערכת הוא החלק החשוב באמת והייתי צריך להסתכל על עיצוב ממשק המשתמש כמעין פרס או בונוס שיכולתי לבצע במקום לשים את זה כפוקוס מרכזי.

לסיכום, העבודה על הפרויקט הייתה בהחלט תהליך מיוחד וחדש בשבילי, ולמרות כל הקשיים שהיו בדרך (והיו קשיים), אני מאמין שאזכור את העבודה על הפרויקט כחווייה חיובית בסך הכל.

אני רוצה להודות למורה שלי צביקה שליווה אותי לאורך כל התהליך ולחבריי לכיתה שהיו קשובים אליי ועזרו לי במידת הצורך בכל בקשה שהיא הקשורה לפרויקט ומחוצה לו.

ביבליוגרפיה:

GeeksforGeeks. (2023, April 24). *Flask Tutorial*. GeeksforGeeks.

<https://www.geeksforgeeks.org/flask-tutorial/>

SQLAlchemy Documentation — SQLAlchemy 2.0 Documentation. (n.d.).

Docs.sqlalchemy.org. <https://docs.sqlalchemy.org/en/20/>

ssl — TLS/SSL wrapper for socket objects — Python 3.7.2 documentation.

(2018). Python.org. <https://docs.python.org/3/library/ssl.html>

SuperSimpleDev. (2022, February 5). *HTML & CSS Full Course - Beginner*

to Pro (2022). Wwww.youtube.com.

<https://www.youtube.com/watch?v=G3e-cpL7ofc>

נספחים:

קוד הפרויקט:

: [app.py](#)

```

import os
import threading
from datetime import timedelta

from flask import Flask, render_template, session,
    redirect, url_for, flash
from flask_wtf import CSRFProtect

    from models import db
    from auth import auth_bp
    from file_management_routes import files_bp
    from friend_management_routes import friends_bp
    from config import certfile, keyfile, basedir,
        db_file, secret_key
    from sync import connect_to_server

    (__app = Flask(__name

Secret key for signing session cookies and CSRF #
    tokens
    app.config['SECRET_KEY'] = secret_key

(SQLite database URI (absolute path to file #
    app.config['SQLALCHEMY_DATABASE_URI'] =
        "{f"sqlite:/// {db_file

```

```

app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] =
    False # disable event system to save memory

    })app.config.update
    Only send session cookie over HTTPS #
        , 'SESSION_COOKIE_SECURE': True'
    Prevent JavaScript access to the cookie #
        , 'SESSION_COOKIE_HTTPONLY': True'
    Mitigate CSRF by restricting cross-site #
        sending of cookies
        , 'SESSION_COOKIE_SAMESITE': 'Lax'
    Lifetime of a "permanent" session #
        PERMANENT_SESSION_LIFETIME': '
            (timedelta(days=7
                ({

    Initialize the SQLAlchemy extension #
        (db.init_app(app

    Wrap the app in Flask-WTF's CSRFProtect to #
        auto-validate tokens on all POST forms
        (csrf = CSRFProtect(app

        .../Authentication routes: /auth #
        app.register_blueprint(auth_bp,
            ('url_prefix='/auth
        .../File management routes: /files #
        app.register_blueprint(files_bp,
            ('url_prefix='/files
        .../Friend management routes: /friends #

```

```

        app.register_blueprint(friends_bp,
                               ('url_prefix='/friends

                               ('/') app.route@
                               :()def dashboard

    Ensure we have a user_id in session (1 #
        ('uid = session.get('user_id
                               :if not uid

    (('return redirect(url_for('auth.login

Lazy imports to avoid circular dependencies #
        from models import User
        from file_management import FileManager

    Load current user from DB using (2 #
        Session.get() to avoid legacy warning
        (user = db.session.get(User, uid
                               :if user is None

    Stale or invalid session → clear and #
                               force re-login
                               ()session.clear

    ("flash("Please log in again.", "error
    (('return redirect(url_for('auth.login

    Instantiate FileManager and fetch the (3 #
                               user's files
                               file_manager =
    FileManager(upload_folder=os.path.join(basedir,
                               (('uploads
    (files = file_manager.list_user_files(user

```

```

Render the dashboard template with user (4 #
                                data
                                )return render_template
                                , 'dashboard.html'
                                , username=user.username
                                , files=files
                                , current_user=user
                                (
                                : '___if ___name___ == '___main
Create database tables if they don't exist #
                                : ()with app.app_context
                                ()db.create_all

Launch the background sync thread exactly #
                                once
                                print("[Sync] Launching background sync
                                (thread...", flush=True
                                threading.Thread(target=connect_to_server,
                                ()daemon=True).start

Start the Flask HTTPS server (self-signed #
                                (cert for dev
                                )app.run
                                , 'host='0.0.0.0
                                debug=False,                                # disable
                                debugger in production
                                (ssl_context=(certfile, keyfile
                                (

```

auth.py:

```

from flask import Blueprint, render_template,
request, redirect, url_for, flash, session
from werkzeug.security import
generate_password_hash, check_password_hash
from sqlalchemy import or_
from models import db, User
from config import changes_queue
from datetime import datetime, timedelta

# Create a Blueprint for all auth-related routes
(register, login, logout)
auth_bp = Blueprint('auth', __name__)

# Lockout policy: max failed attempts before
temporary block
MAX_LOGIN_ATTEMPTS = 5
LOCKOUT_TIME_MINUTES = 15

def _is_locked_out():
    """ Check if the current session is under
    lockout. Returns True if lockout hasn't expired;
    otherwise clears lockout state."""
    lockout_until = session.get('lockout_until')
    if lockout_until and
datetime.fromisoformat(lockout_until) >
datetime.now():
        return True

```

```

        # Lockout expired or not set → remove any
        stale counters
        session.pop('lockout_until', None)
        session.pop('login_attempts', None)
        return False

@auth_bp.route('/register', methods=['GET',
'POST'])
def register():
    """ Handle user registration."""
    if request.method == 'POST':
        session.clear() # Prevent session
        fixation attacks

        username =
request.form['username'].strip()
        email     = request.form['email'].strip()
        password = request.form['password']

        # Basic input validation
        if not username or not email or not
password:
            flash('Please fill in all fields!',
'error')
            return
        redirect(url_for('auth.register'))

        # Check for existing user by username OR
email (safe ORM filter, avoids SQLi)
        existing = User.query.filter(

```

```

        or_(User.username == username,
User.email == email)
    ).first()
    if existing:
        flash('Username or Email already
exists!', 'error')
        return
    redirect(url_for('auth.register'))

    # Create and hash the new user's password
    new_user = User(username=username,
email=email)
    new_user.set_password(password)
    db.session.add(new_user)
    db.session.commit()

    # Notify other regional servers of the new
user
    changes_queue.put({
        "type": "user_create",
        "user_id": new_user.id,
        "username": new_user.username,
        "email": new_user.email,
        "password": password,
        "timestamp":
datetime.now().isoformat()
    })

    flash('Registration successful! Please log
in.', 'success')

```

```

        return redirect(url_for('auth.login'))

    # GET request → render form template
    return render_template('register.html')

@auth_bp.route('/login', methods=['GET', 'POST'])
def login():
    """ Handle user login. """
    if _is_locked_out():
        flash('Too many failed attempts. Try again later.', 'error')

        # Render the login template without
        # redirect to preserve lockout message
        return render_template('login.html')

    if request.method == 'POST':
        session.clear() # Prevent session
        # fixation on every login attempt

        identifier =
request.form['username_or_email'].strip()
        password = request.form['password']
        remember = 'remember_me' in request.form
        # Checkbox presence

        # Lookup by username OR email using safe
        # ORM parameter binding
        user = User.query.filter(

```



```

        or_(User.username == identifier,
User.email == identifier)
    ).first()

    # Verify password using Werkzeug's
constant-time check
    if user and user.check_password(password):
        # Reset failed-attempt counters on
success
        session.pop('login_attempts', None)
        session.pop('lockout_until', None)
        # Store minimal user info in session
        session['user_id'] = user.id
        session['username'] = user.username
        session.permanent = remember #
honor "Remember Me"
        return redirect(url_for('dashboard'))

    # Failed login: increment counter
    attempts = session.get('login_attempts',
0) + 1
    session['login_attempts'] = attempts

    # If too many fails, impose lockout window
    if attempts >= MAX_LOGIN_ATTEMPTS:
        until = datetime.now() +
timedelta(minutes=LOCKOUT_TIME_MINUTES)
        session['lockout_until'] =
until.isoformat()

```

```

        flash(f'Too many failed attempts. Try
again in {LOCKOUT_TIME_MINUTES} minutes.',
'error')
    else:
        flash('Invalid credentials, please try
again.', 'error')

    return redirect(url_for('auth.login'))

# GET request → render the login form
return render_template('login.html')

@auth_bp.route('/logout')
def logout():
    """ Log the user out by clearing the session
and redirecting to login."""
    session.clear()
    flash('Logged out successfully!', 'success')
    return redirect(url_for('auth.login'))

```

config.py:

```

import os
from queue import Queue

# Hostname or IP of the Grand Server
GRAND_HOST = "10.0.0.3"
GRAND_PORT = 9000

BIND_HOST = "0.0.0.0"

```

```

basedir =
os.path.abspath(os.path.dirname(__file__)) #The
location of the system.
certfile = os.path.join(basedir, 'server.crt')
#The location of the certificate.
keyfile = os.path.join(basedir, 'server.key') #The
location of the public key
secret_key =
"dfcc0ab22a9913f9f19c758feabe1c2c56d0e5be670647ef4
f131666c36f0572" #The secret key of the project.

db_file = os.path.join(basedir, 'database.db')
UPLOAD_FOLDER = os.path.join(os.getcwd(),
'uploads')

changes_queue = Queue()

ALLOWED_EXTENSIONS = {'txt', 'pdf', 'png', 'jpg',
'jpeg', 'gif'}

```

file_management_routes.py:

```

from flask import Blueprint, request, flash,
redirect, url_for, send_from_directory, session
from models import User
from file_management import FileManager
import os

```

```

# Blueprint for file operations, all routes under
# /files
files_bp = Blueprint('files', __name__)

# Determine absolute path to the uploads directory
basedir =
os.path.abspath(os.path.dirname(__file__))
upload_dir = os.path.join(basedir, 'uploads')

# Instantiate FileManager with the upload folder
# path
file_manager =
FileManager(upload_folder=upload_dir)

def get_current_user():
    """ Helper to retrieve the logged-in User from
    session. Returns None if no valid user_id in
    session. """
    user_id = session.get('user_id')
    return User.query.get(user_id) if user_id else
    None

@files_bp.route('/upload', methods=['POST'])
def upload():
    """ Handle file upload: """
    user = get_current_user()
    if not user:
        flash("Please log in first.", "error")

```

```

        return redirect(url_for('auth.login'))

    # Get the FileStorage object from the form
    file_storage = request.files.get('file')
    if not file_storage or not
file_storage.filename:
        flash("No file selected for upload.",
"error")
        return redirect(url_for('dashboard'))

    try:
        # Attempt to save file; may flash quota or
type errors internally
        file_manager.save_file(file_storage, user)
        flash("File uploaded successfully!",
"success")
    except ValueError as e:
        # Known validation error from FileManager
        flash(str(e), "error")
    except Exception:
        # Unexpected error
        flash("An unexpected error occurred while
uploading.", "error")

    return redirect(url_for('dashboard'))

@files_bp.route('/download/<int:file_id>')
def download(file_id):

```

```

        """ Serve a file download if the user has
permission. """
        user = get_current_user()
        file_record =
file_manager.get_file_record(file_id)
        if not file_record:
            flash("File not found.", "error")
            return redirect(url_for('dashboard'))

        # Ensure user owns the file or it's public
        if not
file_manager.is_access_allowed(file_record, user):
            flash("Access not allowed.", "error")
            return redirect(url_for('dashboard'))

        # Send the stored file under its original
filename
        return send_from_directory(
            file_manager.upload_folder,
            file_record.stored_filename,
            as_attachment=True,

download_name=file_record.original_filename
        )

@files_bp.route('/delete/<int:file_id>',
methods=['POST'])
def delete(file_id):

```

```

        """ Delete a file record and its physical
file. """
        user = get_current_user()
        file_record =
file_manager.get_file_record(file_id)
        if not file_record:
            flash("File not found.", "error")
            return redirect(url_for('dashboard'))

        try:
            file_manager.delete_file(file_record,
user)

            flash("File deleted successfully.",
"success")
        except Exception as e:
            flash(str(e), "error")

        return redirect(url_for('dashboard'))

@files_bp.route('/permissions/<int:file_id>',
methods=['POST'])
def change_permissions(file_id):
    """Update a file's access permissions (private
or public). """
    user = get_current_user()
    if not user:
        flash("Please log in first.", "error")
        return redirect(url_for('auth.login'))

```

```

        file_record =
file_manager.get_file_record(file_id)
        if not file_record:
            flash("File not found.", "error")
            return redirect(url_for('dashboard'))

        # Only the owner can change permissions
        if file_record.user_id != user.id:
            flash("Access not allowed.", "error")
            return redirect(url_for('dashboard'))

        new_perm = request.form.get('permissions')
        try:

file_manager.update_permissions(file_record,
new_perm, user)
            flash("Permissions updated!", "success")
        except Exception as e:
            flash(str(e), "error")

        return redirect(url_for('dashboard'))

```

file_management.py:

```

import os
import uuid
from werkzeug.utils import secure_filename
from models import db, File
from config import changes_queue
from datetime import datetime
from config import ALLOWED_EXTENSIONS

```



```

class FileManager:
    #Encapsulates file operations.
    def __init__(self, upload_folder):
        """ Initialize FileManager with a
        directory to store uploaded files. Creates the
        directory if it doesn't exist. """
        self.upload_folder = upload_folder
        os.makedirs(self.upload_folder,
exist_ok=True)

    def allowed_file(self, filename):
        """ Check if the file has an allowed
        extension. Returns True if extension is in
        ALLOWED_EXTENSIONS. """
        return '.' in filename and
filename.rsplit('.', 1)[1].lower() in
ALLOWED_EXTENSIONS

    def generate_unique_filename(self, filename):
        """ Generate a unique filename using UUID4
        and preserve the file extension. """
        ext = filename.rsplit('.', 1)[1] if '.' in
filename else ''
        # Combine a random hex string with the
        original extension
        unique_name = f"{uuid.uuid4().hex}.{ext}"
if ext else uuid.uuid4().hex
        return unique_name

```

```

def save_file(self, file_storage, user):
    """ Save an uploaded file to disk and
create a corresponding DB record. Also updates the
user's used_storage and enqueues a sync event. """
    # Validate file presence
    if not file_storage or
file_storage.filename == '':
        raise ValueError("No file provided.")
    # Validate extension
    if not
self.allowed_file(file_storage.filename):
        raise ValueError("File type not
allowed.")

    # Secure the original filename and
generate a unique stored name
    original_filename =
secure_filename(file_storage.filename)
    unique_filename =
self.generate_unique_filename(original_filename)
    file_path =
os.path.join(self.upload_folder, unique_filename)

    # Save to disk and measure file size
    file_storage.save(file_path)
    file_size = os.path.getsize(file_path)

    # Enforce user storage quota
    if user.used_storage + file_size >
user.storage_quota:

```

```
        os.remove(file_path)
        raise ValueError("Storage quota
exceeded.")

# Create DB record for the file
file_record = File(
    user_id=            user.id,
    stored_filename=    unique_filename,
    original_filename=  original_filename,
    file_size=          file_size,
    permissions=        'private'
)
db.session.add(file_record)

# Update user's used storage and commit
both changes
user.used_storage += file_size
db.session.add(user)
db.session.commit()

# Read file content for sync event
with open(file_path, 'rb') as f:
    content = f.read()

# Enqueue a file_upload event for
synchronization
changes_queue.put({
    "type":      "file_upload",
    "payload": {
```

```

        "id":
file_record.id,
        "user_id":
file_record.user_id,
        "stored_filename":
file_record.stored_filename,
        "original_filename":
file_record.original_filename,
        "upload_date":
file_record.upload_date.isoformat(),
        "file_size":
file_record.file_size,
        "permissions":
file_record.permissions,
        "content":          content
    },
    "timestamp":
datetime.now().isoformat()
    })

    return file_record

    def list_user_files(self, user):
        """ Return a list of File records
        belonging to the given user. """
        return
File.query.filter_by(user_id=user.id).all()

    def get_file_record(self, file_id):

```

```

        """ Retrieve a single File record by its
        ID. Returns None if not found.
        """
        return File.query.get(file_id)

    def delete_file(self, file_record, user,
enqueue=True):
        """ Delete a file from disk and remove its
        DB record. Only the file owner may delete.
        Enqueues a file_delete sync event. """
        if file_record.user_id != user.id:
            # Prevent unauthorized deletions
            raise PermissionError("You are not
authorized to delete this file.")
            # Remove file from disk
            file_path =
os.path.join(self.upload_folder,
file_record.stored_filename)
            if os.path.exists(file_path):
                os.remove(file_path)
            # Remove DB record
            db.session.delete(file_record)
            db.session.commit()

            # Notify other servers of deletion
            if enqueue:
                changes_queue.put({
                    "type": "file_delete",
                    "file_id": file_record.id,

```

```

        "timestamp":
datetime.now().isoformat()
    })
    return True

    def update_permissions(self, file_record,
new_permissions, user, enqueue=True):
        """ Change the permission of a file
(private or public) Only the owner may change
permissions. Enqueues a permission_change sync
event. """
        if file_record.user_id != user.id:
            # Prevent unauthorized permission
changes
            raise PermissionError("You are not
authorized to change permissions for this file.")
        file_record.permissions = new_permissions
        db.session.commit()

        # Notify other servers of permission
change
        if enqueue:
            changes_queue.put({
                "type":
"permission_change",
                "file_id":         file_record.id,
                "new_permissions":
file_record.permissions,
                "timestamp":
datetime.now().isoformat()

```

```

        })

        return file_record

    def is_access_allowed(self, file_record,
user):
        """ Check if the given user may access the
file. Owners always allowed, otherwise only public
files. """
        if file_record.user_id == user.id:
            return True
        return file_record.permissions == 'public'

```

friend_managment_routes.py:

```

from flask import Blueprint, render_template,
request, redirect, url_for, flash, session
from models import User
from friend_management import FriendManager
from file_management import FileManager

# Blueprint for friend-related routes under
/friends
friends_bp = Blueprint('friends', __name__)
# Manager instances for business logic
friend_manager = FriendManager()
file_manager =
FileManager(upload_folder='uploads')

def get_current_user():

```

```

    """ Retrieve the logged-in user from the
session. Returns None if no user is logged in. """
    user_id = session.get('user_id')
    return User.query.get(user_id) if user_id else
None

@friends_bp.route('/requests', methods=['GET',
'POST'])
def view_requests():
    """ GET: Show incoming & outgoing friend
requests, and a form to send new ones. POST:
Handle submission of a new friend request by
username. """
    user = get_current_user()
    if not user:
        return redirect(url_for('auth.login'))

    # Handle new request submission
    if request.method == 'POST' and 'to_username'
in request.form:
        to_username = request.form['to_username']
        to_user =
User.query.filter_by(username=to_username).first()
        if not to_user:
            flash("User not found.", "error")
        else:
            try:
                friend_manager.send_request(user,
to_user)

```



```

        flash("Friend request sent!",
"success")

        except ValueError as e:
            flash(str(e), "error")

        return
redirect(url_for('friends.view_requests'))

    # Build lists for template
    raw_in =
friend_manager.get_incoming_requests(user)
    incoming = [
        {'req': fr, 'sender':
User.query.get(fr.from_user_id)}
        for fr in raw_in
    ]

    raw_out =
friend_manager.get_outgoing_requests(user)
    outgoing = [ User.query.get(fr.to_user_id) for
fr in raw_out ]

    return render_template(
        'incoming_requests.html',
        incoming=incoming,
        outgoing=outgoing
    )

@friends_bp.route('/requests/respond/<int:rq_id>',
methods=['POST'])

```

```

def respond_request(rq_id):
    """ Handle acceptance or rejection of a friend
    request. """
    user = get_current_user()
    action = request.form.get('action')
    try:
        friend_manager.respond_request(rq_id,
        accept=(action == 'accept'))
        flash(f"Request {action}ed.", "success")
    except ValueError as e:
        flash(str(e), "error")
    return
redirect(url_for('friends.view_requests'))

@friends_bp.route('/list')
def list_friends():
    """ Display the current user's friend list.
    """
    user = get_current_user()
    if not user:
        return redirect(url_for('auth.login'))
    friends = friend_manager.get_friends(user)
    return render_template('friends_list.html',
    friends=friends)

@friends_bp.route('/remove/<username>',
methods=['POST'])
def remove(username):

```

```

        """ Remove an existing friendship. """
        user = get_current_user()
        if not user:
            return redirect(url_for('auth.login'))

        to_user =
User.query.filter_by(username=username).first()
        if not to_user:
            flash("User not found.", "error")
        else:
            try:
                friend_manager.remove_friend(user,
to_user)

                flash("Friend removed.", "success")
            except ValueError as e:
                flash(str(e), "error")

        return
redirect(url_for('friends.list_friends'))

@friends_bp.route('/<username>/files')
def view_friend_files(username):
    """ View files shared by a specific friend.
Only files with 'public' permission or owned by
the friend are shown.
    """
    user = get_current_user()
    if not user:
        return redirect(url_for('auth.login'))

```

```
        friend =
User.query.filter_by(username=username).first()
        if not friend:
            flash("User not found.", "error")
            return
redirect(url_for('friends.list_friends'))

        friends_list =
friend_manager.get_friends(user)
        if friend not in friends_list:
            flash("You may only view files of your
friends.", "error")
            return redirect(url_for('dashboard'))

        all_files      =
file_manager.list_user_files(friend)
        # Filter out files the current user isn't
allowed to see
        allowed_files = [f for f in all_files if
file_manager.is_access_allowed(f, user)]

        return render_template(
            'friends_files.html',
            files=allowed_files,
            friend=friend
        )
```

friend_management.py:

```
from models import db, User, FriendRequest,
Friendship
from config import changes_queue
from datetime import datetime

class FriendManager:
    #Encapsulates friend request and friendship
    operations, including sending requests,
    responding, listing, and removal.

    def send_request(self, from_user, to_user,
enqueue=True):
        """ Send a friend request from 'from_user'
to 'to_user'. """
        # Prevent sending a request to oneself
        if from_user.id == to_user.id:
            raise ValueError("Cannot friend
yourself.")

        # Check existing pending request in either
direction
        existing = FriendRequest.query.filter_by(
            from_user_id=from_user.id,
            to_user_id=to_user.id,
            status='pending'
        ).first()
        if not existing:
            existing =
FriendRequest.query.filter_by(
```

```

        from_user_id=to_user.id,
        to_user_id=from_user.id,
        status='pending'
    ).first()
    if existing:
        raise ValueError("Friend request
already pending.")

    # Check if users are already friends
    fri = Friendship.query.filter_by(
        user_id=from_user.id,
        friend_id=to_user.id
    ).first()
    if not fri:
        fri = Friendship.query.filter_by(
            user_id=to_user.id,
            friend_id=from_user.id
        ).first()
    if fri:
        raise ValueError("You are already
friends.")

    # Create and persist the new friend
request
    fr = FriendRequest(
        from_user_id=from_user.id,
        to_user_id=to_user.id
    )
    db.session.add(fr)
    db.session.commit()

```

```

        # Enqueue change for synchronization with
other nodes
        if enqueue:
            changes_queue.put({
                "type":      "friend_request",
                "request_id": fr.id,
                "from_user": fr.from_user_id,
                "to_user":   fr.to_user_id,
                "timestamp":
datetime.now().isoformat()
            })
        return fr

    def get_incoming_requests(self, user):
        """ Retrieve all pending friend requests
where 'user' is the recipient. """
        return FriendRequest.query.filter_by(
            to_user_id=user.id,
            status='pending'
        ).all()

    def get_outgoing_requests(self, user):
        """ Retrieve all pending friend requests
sent by 'user'. """
        return FriendRequest.query.filter_by(
            from_user_id=user.id,
            status='pending'
        ).all()

```

```

def respond_request(self, request_id,
accept=True, enqueue=True):
    """ Accept or reject a friend request by
    ID. If accepted, creates reciprocal Friendship
    entries. """
    # Fetch the request object
    fr = FriendRequest.query.get(request_id)
    if not fr:
        raise ValueError("Friend request not
found.")
    # Update status
    fr.status = 'accepted' if accept else
'rejected'
    if accept:
        # Create mutual friendship records
        db.session.add(Friendship(user_id=fr.from_user_id,
friend_id=fr.to_user_id))
        db.session.add(Friendship(user_id=fr.to_user_id,
friend_id=fr.from_user_id))
        db.session.commit()

    # Enqueue corresponding sync event
    if enqueue:
        if accept:
            changes_queue.put({
                "type": "friend_added",
                "request_id": fr.id,

```



```

        "timestamp":
datetime.now().isoformat()
    })
    else:
        changes_queue.put({
            "type":
"friend_rejected",
            "request_id": fr.id,
            "from_user": fr.from_user_id,
            "to_user": fr.to_user_id,
            "timestamp":
datetime.now().isoformat()
        })
    return fr

def get_friends(self, user):
    """ List all users who are friends with
'user'. """
    # Query one direction, then load User for
each friendship
    fs =
Friendship.query.filter_by(user_id=user.id).all()
    return [User.query.get(f.friend_id) for f
in fs]

def remove_friend(self, user, to_user,
enqueue=True):
    """ Remove an existing friendship between
'user' and 'to_user'. """
    # Find both directions of the friendship

```

```

        f1 =
Friendship.query.filter_by(user_id=user.id,
friend_id=to_user.id).first()
        f2 =
Friendship.query.filter_by(user_id=to_user.id,
friend_id=user.id    ).first()
        if not f1 and not f2:
            raise ValueError("Friendship not
found.")
        # Delete whichever records exist
        if f1:
            db.session.delete(f1)
        if f2:
            db.session.delete(f2)
        db.session.commit()

        # Enqueue removal event
        if enqueue:
            changes_queue.put({
                "type":      "friend_removed",
                "user_id":   user.id,
                "friend_id": to_user.id,
                "timestamp":
datetime.now().isoformat()
            })
        return True

```

[models.py](#):

```
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime
from werkzeug.security import
generate_password_hash, check_password_hash

db = SQLAlchemy()

class User(db.Model):
    #A class that represents a user in the system.
    id = db.Column(db.Integer, primary_key=True)
    #The id of the user.
    username = db.Column(db.String(80),
unique=True, nullable=False) #The username.
    email = db.Column(db.String(120), unique=True,
nullable=False) #The email of the user.
    password_hash = db.Column(db.String(128),
nullable=False) #The hashed password of the user.
    created_at = db.Column(db.DateTime,
default=datetime.now) #The creation date of the
user.
    used_storage = db.Column(db.Integer,
nullable=False, default=0) #The used storage of
the user.
    storage_quota = db.Column(db.Integer,
nullable=False, default=1073741824) #The max
storage of the user.

    def set_password(self, password):
```

```

        """Generating the hash value of the
password."""
        self.password_hash =
generate_password_hash(password)

    def check_password(self, password):
        """Checking if the password is the same as
the user's password. """
        return
check_password_hash(self.password_hash, password)

class File(db.Model):
    #Represents the metadata of a file in the
system.
    id = db.Column(db.Integer, primary_key=True)
    #The id of the file
    user_id = db.Column(db.Integer,
db.ForeignKey('user.id'), nullable=False) #The id
of the file's owner.
    stored_filename = db.Column(db.String(128),
nullable=False) #The name of the file in the file
system.
    original_filename = db.Column(db.String(128),
nullable=False) #The original name of the file.
    upload_date = db.Column(db.DateTime,
default=datetime.now) #The creation date of the
file.
    file_size = db.Column(db.Integer) #The file's
size.

```

```

        permissions = db.Column(db.String(32),
default='private') #The file's viewing permission.

class FriendRequest(db.Model):
    #Represents a friend request in the system.
    id = db.Column(db.Integer, primary_key=True)
#The id of the request.
    from_user_id = db.Column(db.Integer,
db.ForeignKey('user.id'), nullable=False) #The id
of the sender.
    to_user_id = db.Column(db.Integer,
db.ForeignKey('user.id'), nullable=False) #The id
of the receiver.
    status = db.Column(db.String(16),
default='pending') #The status of the friend
request('pending', 'accepted', 'rejected')
    created_at = db.Column(db.DateTime,
default=datetime.now) #The creation date of the
friend request.

class Friendship(db.Model):
    #Represents a friendship in the system.
    id = db.Column(db.Integer, primary_key=True)
#The id of the friendship.
    user_id = db.Column(db.Integer,
db.ForeignKey('user.id'), nullable=False) #The id
of one of the user.
    friend_id = db.Column(db.Integer,
db.ForeignKey('user.id'), nullable=False) #The id
of the other user.

```

```

        created_at = db.Column(db.DateTime,
                                default=datetime.now) #The creation date of the
friendship.

```

sync.py:

```

import os
import socket
import json
import base64
import time
import ssl
from datetime import datetime
from queue import Empty

from config import GRAND_HOST, GRAND_PORT,
changes_queue, UPLOAD_FOLDER

def send_changes(sock):
    """ Drain the local changes_queue, encode file
    contents in Base64 when needed, and send a single
    'changes' packet to the Grand Server over the
    socket."""
    events = []
    # Pull all pending events without blocking
    while True:
        try:
            change = changes_queue.get_nowait()
        except Empty:

```

```

        break

        # If it's a new file upload, Base64-encode
raw bytes for JSON transport
        if change['type'] == 'file_upload':
            payload_copy =
change['payload'].copy()
            raw = payload_copy.pop('content') #
remove binary data from payload
            # encode bytes to UTF-8 string so it
can be embedded in JSON
            payload_copy['content'] =
base64.b64encode(raw).decode('utf-8')
            events.append({
                'type': change['type'],
                'payload': payload_copy,
                'timestamp': change['timestamp']
            })
        else:
            # other event types can be sent as-is
            events.append(change)

        # If no events to send, do nothing
        if not events:
            return

        # Build and send the JSON packet, ending with
newline for framing
        packet = {'type': 'changes', 'events': events}

```

```

        sock.sendall((json.dumps(packet) +
'\n').encode('utf-8'))

def receive_changes(message):
    """ Apply incoming events from Grand Server
inside a fresh Flask application context. This
allows us to modify the database and file system
safely."""
    from app import app
    with app.app_context():
        from file_management import FileManager
        from friend_management import
FriendManager
        from models import db, User, File

        file_manager =
FileManager(upload_folder=UPLOAD_FOLDER)
        friend_manager = FriendManager()

        for sync_event in message.get('events',
[]):
            event_type = sync_event.get('type')
            try:
                if event_type == 'file_upload':
                    # Decode and write file bytes
                    payload =
sync_event['payload']
                    data =
base64.b64decode(payload['content'])

```



```

        dest =
os.path.join(UPLOAD_FOLDER,
payload['stored_filename'])

os.makedirs(os.path.dirname(dest), exist_ok=True)
        with open(dest, 'wb') as f:
            f.write(data)

        # Merge into DB (insert or
update existing record by ID)
        rec = File(
            id=payload['id'],

user_id=payload['user_id'],

stored_filename=payload['stored_filename'],

original_filename=payload['original_filename'],

file_size=payload['file_size'],

permissions=payload['permissions'],

upload_date=datetime.fromisoformat(payload['upload
_date']))

        )
        db.session.merge(rec)
        db.session.commit()

elif event_type == 'file_delete':

```

```

        rec =
file_manager.get_file_record(sync_event['file_id']
)
        if not rec:
            print(f"[Sync] Warn: no
file record for deletion ID
{sync_event['file_id']}\"", flush=True)
            continue
        user =
User.query.get(rec.user_id)
        file_manager.delete_file(rec,
user, enqueue=False)

        elif event_type ==
'permission_change':
            rec =
file_manager.get_file_record(sync_event['file_id']
)
            if not rec:
                print(f"[Sync] Warn: no
file for permission change ID
{sync_event['file_id']}\"", flush=True)
                continue
            user =
User.query.get(rec.user_id)

file_manager.update_permissions(rec,
sync_event['new_permissions'], user,
enqueue=False)

```

```

        elif event_type == 'user_create':
            # Merge new user record,
preserves existing if present
            u = User(
                id=sync_event['user_id'],

username=sync_event['username'],
                email=sync_event['email']
            )

u.set_password(sync_event['password'])
            db.session.merge(u)
            db.session.commit()

        elif event_type ==
'friend_request':
            friend_manager.send_request(

User.query.get(sync_event['from_user']),

User.query.get(sync_event['to_user']),
                enqueue=False
            )

        elif event_type == 'friend_added':

friend_manager.respond_request(sync_event['request
_id'], accept=True, enqueue=False)

```

```

        elif event_type ==
'friend_rejected':

friend_manager.respond_request(sync_event['request
_id'], accept=False, enqueue=False)

        elif event_type ==
'friend_removed':

            friend_manager.remove_friend(

User.query.get(sync_event['user_id']),

User.query.get(sync_event['friend_id']),
                enqueue=False
            )

        else:

            # Unknown event type – log for
debugging

            print(f"[Sync] Unknown event
type: {event_type}", flush=True)

    except Exception as e:

        # Roll back any partial DB changes
on error

        db.session.rollback()

        print(f"[Sync] Error applying
'{event_type}' event: {e}", flush=True)

```

```

def sync_changes(sock):
    """ Read newline-delimited JSON commands from
    Grand Server and dispatch to send_changes or
    receive_changes handlers."""
    buffer = sock.makefile('r') # wrap socket in
    file-like object for line reads
    for line in buffer:
        try:
            received_message = json.loads(line)
        except json.JSONDecodeError:
            continue

        message_type =
received_message.get('type')
        if message_type == 'send':
            send_changes(sock)
        elif message_type == 'receive':
            receive_changes(received_message)

        time.sleep(0) # yield to other threads

    # When connection closes, clean up
    sock.close()
    print("[Sync] Connection closed", flush=True)

def connect_to_server():
    """ Establish a TLS-wrapped TCP connection to
    the Grand Server, then enter the sync loop."""

```

```

        # Create SSL context for client with no
verification (dev only)
        context =
ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
        context.check_hostname = False
        context.verify_mode = ssl.CERT_NONE

        raw_sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        sock = context.wrap_socket(raw_sock,
server_hostname=GRAND_HOST)

        sock.connect((GRAND_HOST, GRAND_PORT))
        print("[Sync] Connected to grand server",
flush=True)
        sync_changes(sock)

```

grand_server.py:

```

import socket
import threading
import json
import time
import ssl

from config import BIND_HOST, GRAND_PORT,
certfile, keyfile

```

```

# How often (in seconds) to prompt regionals for
changes
SYNC_INTERVAL = 30  # 1 sync every 30 seconds

# We want to keep all batches from the last 5 sync
intervals (5×SYNC_INTERVAL seconds)
HISTORY_WINDOW = SYNC_INTERVAL * 5

# history holds dicts {"ts": timestamp, "events":
[...]}
history = []

# Global list of connected regional sockets
clients = []
clients_lock = threading.Lock()  # ensure one
thread at a time touches clients list

def send_history(conn):
    """
    Replay every batch of events from the last
HISTORY_WINDOW seconds
    to this newly connected regional_server.
    """
    cutoff = time.time() - HISTORY_WINDOW
    # send each batch whose timestamp is within
the window
    for entry in history:
        if entry["ts"] < cutoff:
            continue

```

```

        packet = json.dumps({'type': 'receive',
                              'events': entry["events"]}) + '\n'
        try:
            conn.sendall(packet.encode('utf-8'))
        except Exception as e:
            print(f"[GrandServer] Failed to replay
history to {conn.getpeername()}: {e}", flush=True)

def accept_loop(server_sock, context):
    """Continuously accept new regional
connections and wrap them with TLS."""
    while True:
        try:
            raw_conn, addr = server_sock.accept()
            # Perform TLS handshake on the new
connection
            try:
                conn =
context.wrap_socket(raw_conn, server_side=True)
                print(f"[GrandServer] Regional
connected (TLS): {addr}", flush=True)
            except ssl.SSLError as e:
                print(f"[GrandServer] SSL
handshake failed with {addr}: {e}", flush=True)
                raw_conn.close()
                continue

            # add to clients under lock
            with clients_lock:

```



```

        clients.append(conn)

        # replay missed-history batches
        send_history(conn)

        # start handler thread

threading.Thread(target=client_handler,
args=(conn,), daemon=True).start()

    except Exception as e:
        print(f"[GrandServer] accept_loop
error: {e}", flush=True)
        time.sleep(1)

def client_handler(conn):
    """Read incoming 'changes' messages and
rebroadcast them."""
    addr = conn.getpeername()
    f = conn.makefile('r') # treat socket as file
    for line-based reading
    try:
        for line in f:
            line = line.strip()
            if not line:
                continue

            try:
                msg = json.loads(line)

```

```

        except json.JSONDecodeError as e:
            print(f"[GrandServer] Invalid JSON
from {addr}: {e}", flush=True)
            continue

        mtype = msg.get('type')
        if mtype == 'changes':
            events = msg.get('events', [])
            print(f"[GrandServer] Received
{len(events)} events from {addr}", flush=True)
            broadcast(events, exclude=conn)
        else:
            print(f"[GrandServer] Unknown
message type from {addr}: {mtype}", flush=True)

    except Exception as e:
        print(f"[GrandServer] Connection error
from {addr}: {e}", flush=True)
    finally:
        # remove disconnected client
        with clients_lock:
            if conn in clients:
                clients.remove(conn)

        try:
            conn.close()
        except:
            pass

        print(f"[GrandServer] Regional
disconnected: {addr}", flush=True)

```

```

def sync_loop():
    """Every SYNC_INTERVAL seconds, send 'send' to
    all live regionals."""
    while True:
        try:
            time.sleep(SYNC_INTERVAL)
            print("[GrandServer] Requesting
changes from all regionals...", flush=True)
            packet = json.dumps({'type': 'send'})
+ '\n'

            data = packet.encode('utf-8')

            with clients_lock:
                for conn in list(clients):
                    try:
                        conn.sendall(data)
                    except Exception as e:
                        addr = None
                        try:
                            addr =
conn.getpeername()

                        except:
                            pass
                        print(f"[GrandServer]
Error sending sync request to {addr}: {e}",
flush=True)

                    clients.remove(conn)
                    try:
                        conn.close()

```

```

        except:
            pass

    except Exception as e:
        print(f"[GrandServer] sync_loop error: {e}", flush=True)
        time.sleep(1)

def broadcast(events, exclude=None):
    """
    Broadcast received events to every regional
    except the sender,
    and record each batch with a timestamp for
    history-window replay.
    """
    # record this batch with the current time
    history.append({
        "ts": time.time(),
        "events": events
    })
    # purge any entries older than HISTORY_WINDOW
    cutoff = time.time() - HISTORY_WINDOW
    history[:] = [h for h in history if h["ts"] >= cutoff]

    packet = json.dumps({'type': 'receive',
        'events': events}) + '\n'
    data = packet.encode('utf-8')

```

```

with clients_lock:
    for conn in list(clients):
        if conn is exclude:
            continue
        try:
            conn.sendall(data)
        except Exception as e:
            addr = None
            try:
                addr = conn.getpeername()
            except:
                pass
            print(f"[GrandServer] Broadcast
error to {addr}: {e}", flush=True)
            clients.remove(conn)
            try:
                conn.close()
            except:
                pass

def main():
    # Create TCP listening socket
    server_sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    # allow quick reuse after restart
    server_sock.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, 1)
    server_sock.bind((BIND_HOST, GRAND_PORT))
    server_sock.listen()

```

```

    print(f"[GrandServer] Listening on
{BIND_HOST}:{GRAND_PORT}", flush=True)

    # Create TLS context and load certificate/key
    context =
ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
    context.load_cert_chain(certfile=certfile,
keyfile=keyfile)

    # start accept and sync loops in background
threads
    threading.Thread(target=accept_loop,
args=(server_sock, context), daemon=True).start()
    threading.Thread(target=sync_loop,
daemon=True).start()

    try:
        # keep main thread alive
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        print("[GrandServer] Shutting down.",
flush=True)
    finally:
        server_sock.close()

if __name__ == '__main__':
    main()

```

templates/dashboard.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>SyncSphere - Dashboard</title>
  <meta name="viewport"
content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="{{
url_for('static', filename='css/dashboard.css')
}}">
</head>
<body>
  <header class="navbar">
    <div class="navbar-left">
      <a href="{{ url_for('dashboard') }}"
class="logo-link" style="display:flex;
align-items:center; text-decoration:none;">
        
        <span class="app-name">SyncSphere</span>
      </a>
    </div>
    <nav class="navbar-right">
      <a href="{{ url_for('dashboard')
}}">Dashboard</a>
      <a href="{{ url_for('friends.view_requests')
}}">Requests</a>
```

```

    <a href="{{ url_for('friends.list_friends')
}}">Friends List</a>

    <a href="{{ url_for('auth.logout')
}}">Logout</a>

</nav>
</header>

<section class="storage-usage">
    {% set used_gb = (current_user.used_storage /
(1024*1024*1024)) %}
    {% set quota_gb = (current_user.storage_quota
/ (1024*1024*1024)) %}
    <p>Storage used: <strong>{{ used_gb|round(2)
}} GB</strong> of <strong>{{ quota_gb|round(2) }}
GB</strong></p>
</section>

<main>
    <section class="files-section">
        <h2>Available Files</h2>
        {% if files %}
            <table class="files-table">
                <thead>
                    <tr>
                        <th>Filename</th>
                        <th>Upload Date</th>
                        <th>Size (bytes)</th>
                        <th>Actions</th>
                    </tr>
                </thead>

```



```

        <tbody>
            {% for file in files %}
                <tr>
                    <td>{{ file.original_filename
}}</td>

                    <td>{{
file.upload_date.strftime("%Y-%m-%d %H:%M")
}}</td>

                    <td>{{ file.file_size }}</td>
                    <td>
                        <a href="{{
url_for('files.download', file_id=file.id) }}"
class="btn btn-download">Download</a>

                        <form action="{{
url_for('files.delete', file_id=file.id) }}"
method="post" style="display:inline;">
                            <input type="hidden"
name="csrf_token" value="{{ csrf_token() }}">
                            <button type="submit"
class="btn btn-delete" onclick="return
confirm('Are you sure?')">Delete</button>
                        </form>

                        <form action="{{
url_for('files.change_permissions',
file_id=file.id) }}" method="post"
style="display:inline; margin-left:0.5rem;">
                            <input type="hidden"
name="csrf_token" value="{{ csrf_token() }}">
                            <select name="permissions"
class="permission-select">

```

```

        <option value="private" {%
if file.permissions=='private' %}selected{% endif
%}>Private</option>

        <option value="public" {% if
file.permissions=='public' %}selected{% endif
%}>Public</option>

        </select>
        <button type="submit"
class="btn btn-upload">Update</button>
    </form>
</td>
</tr>
{% endfor %}
</tbody>
</table>
{% else %}
    <p class="no-files">You have no uploaded
files yet.</p>
{% endif %}
</section>

<section class="upload-cta">
    <h2>Effortless File Management</h2>
    <p>Easily upload and organize your files
with SyncSphere.</p>
    <form action="{{ url_for('files.upload') }}"
method="post" enctype="multipart/form-data"
class="upload-form">
        <input type="hidden" name="csrf_token"
value="{{ csrf_token() }}">

```

```

        <input type="file" name="file" required>
        <button type="submit" class="btn
btn-upload">Upload File</button>
    </form>
</section>

    {% with messages =
get_flashed_messages(with_categories=True) %}
    {% if messages %}
        <div class="flash-messages">
            <ul>
                {% for category, message in messages
%}
                    <li class="{{ category }}">{{
message }}</li>
                {% endfor %}
            </ul>
        </div>
    {% endif %}
    {% endwith %}
</main>
</body>
</html>

```

templates/friends_files.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">

```

```

<title>{{ friend.username }}'s Files</title>
<meta name="viewport"
content="width=device-width, initial-scale=1.0">
<link rel="stylesheet" href="{{
url_for('static', filename='css/dashboard.css')
}}">
</head>
<body>
  <header class="navbar">
    <div class="navbar-left">
      <a href="{{ url_for('dashboard') }}"
class="logo-link" style="display:flex;
align-items:center; text-decoration:none;">
        
        <span class="app-name">SyncSphere</span>
      </a>
    </div>
    <nav class="navbar-right">
      <a href="{{ url_for('dashboard')
}}">Dashboard</a>
      <a href="{{ url_for('friends.list_friends')
}}">Friends</a>
      <a href="{{ url_for('auth.logout')
}}">Logout</a>
    </nav>
  </header>

  <main class="main-content">

```

```

<section class="files-section">
  <h2>{{ friend.display_name or
friend.username }}'s Files</h2>
  {% if files %}
    <table class="files-table">
      <thead>
        <tr>
          <th>Filename</th>
          <th>Upload Date</th>
          <th>Size</th>
          <th>Actions</th>
        </tr>
      </thead>
      <tbody>
        {% for file in files %}
          <tr>
            <td>{{ file.original_filename
}}</td>
            <td>{{
file.upload_date.strftime("%Y-%m-%d %H:%M")
}}</td>
            <td>{{ file.file_size }}</td>
            <td><a href="{{
url_for('files.download', file_id=file.id) }}"
class="btn btn-download">Download</a></td>
          </tr>
        {% endfor %}
      </tbody>
    </table>
  {% else %}

```

```

        <p class="no-files">No files
available.</p>
        {% endif %}
    </section>
</main>
</body>
</html>

```

templates/friends_list.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>SyncSphere - Your Friends</title>
    <meta name="viewport"
content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{
url_for('static', filename='css/dashboard.css')
}}">
</head>
<body>
    <header class="navbar">
        <div class="navbar-left">
            <a href="{{ url_for('dashboard') }}"
class="logo-link" style="display:flex;
align-items:center; text-decoration:none;">
                

```

```

        <span class="app-name">SyncSphere</span>
    </a>
</div>

<nav class="navbar-right">
    <a href="{{ url_for('dashboard') }}">Dashboard</a>
    <a href="{{ url_for('friends.view_requests') }}">Requests</a>
    <a href="{{ url_for('auth.logout') }}">Logout</a>
</nav>
</header>

<main class="main-content">
    <section class="list-section">
        <h2>Your Friends</h2>

        {% if friends %}
            <div class="friends-grid">
                {% for friend in friends %}
                    <div class="friend-card">
                        <div class="friend-info">
                            <h3>{{ friend.display_name or
friend.username }}</h3>
                            <p>@{{ friend.username }}</p>
                            <p>Joined: {{
friend.created_at.strftime('%b %d, %Y') }}</p>
                        </div>
                        <div class="friend-actions">

```

```

        <a href="{{
url_for('friends.view_friend_files',
username=friend.username) }}"
        class="btn btn-download">View
Files</a>

        <form action="{{
url_for('friends.remove',
username=friend.username) }}"
        method="post"
        style="display:inline;
margin-left:0.5rem;">
            <input type="hidden"
name="csrf_token" value="{{ csrf_token() }}">
            <button type="submit"
                class="btn btn-remove"
                onclick="return
confirm('Remove friend?')">
                Remove
            </button>
        </form>
    </div>
</div>
{% endfor %}
</div>
{% else %}
    <p class="no-friends">You have no friends
yet.</p>
{% endif %}

```



```

        {% with messages =
get_flashed_messages(with_categories=True) %}
        {% if messages %}
            <div class="flash-messages">
                <ul>
                    {% for category, message in messages
%}
                        <li class="{{ category }}">{{
message }}</li>
                    {% endfor %}
                </ul>
            </div>
        {% endif %}
    {% endwith %}
</section>
</main>
</body>
</html>

```

templates/incoming_requests.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>SyncSphere - Friend Requests</title>
    <meta name="viewport"
content="width=device-width, initial-scale=1.0">

```

```

<link rel="stylesheet" href="{{
url_for('static', filename='css/dashboard.css')
}}">
</head>
<body>
  <header class="navbar">
    <div class="navbar-left">
      <a href="{{ url_for('dashboard') }}"
class="logo-link" style="display:flex;
align-items:center; text-decoration:none;">
        
        <span class="app-name">SyncSphere</span>
      </a>
    </div>
    <nav class="navbar-right">
      <a href="{{ url_for('dashboard')
}}">Dashboard</a>
      <a href="{{ url_for('friends.list_friends')
}}">Friends List</a>
      <a href="{{ url_for('auth.logout')
}}">Logout</a>
    </nav>
  </header>

  <main class="main-content">
    <section class="list-section">
      <h2>Friend Requests</h2>

```

```

        <form method="post" action="{{
url_for('friends.view_requests') }}"
class="action-form">
            <input type="hidden" name="csrf_token"
value="{{ csrf_token() }}">
            <input type="text" name="to_username"
placeholder="Add a friend" required>
            <button type="submit" class="btn
btn-upload">Send Request</button>
        </form>

{% if outgoing %}
    <h3>Your Pending Requests</h3>
    <div class="friends-grid">
        {% for u in outgoing %}
            <div class="friend-card">
                <div class="friend-info">
                    <h3>{{ u.display_name or
u.username }}</h3>
                    <p>@{{ u.username }}</p>
                </div>
            </div>
        {% endfor %}
    </div>
{% endif %}

{% if incoming %}
    <h3>Incoming Requests</h3>
    <div class="friends-grid">
        {% for item in incoming %}

```

```

        <div class="friend-card">
            <div class="friend-info">
                <h3>{{ item.sender.display_name or
item.sender.username }}</h3>
                <p>@{{ item.sender.username }}</p>
            </div>
            <div class="friend-actions">
                <form method="post" action="{{
url_for('friends.respond_request',
rq_id=item.req.id) }}">
                    <input type="hidden"
name="csrf_token" value="{{ csrf_token() }}">
                    <button name="action"
value="accept" class="btn
btn-download">Accept</button>
                    <button name="action"
value="reject" class="btn
btn-delete">Reject</button>
                </form>
            </div>
        </div>
    {% endfor %}
</div>
{% else %}
    <p class="no-requests">No pending
requests.</p>
{% endif %}

    {% with messages =
get_flashed_messages(with_categories=True) %}

```

```

        {% if messages %}
        <div class="flash-messages">
            <ul>
                {% for category, msg in messages %}
                <li class="{{ category }}">{{ msg
}}</li>

                {% endfor %}
            </ul>
        </div>
        {% endif %}
    {% endwith %}
</section>
</main>
</body>
</html>

```

templates/[login](#).html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>SyncSphere - Login</title>
    <meta name="viewport"
content="width=device-width, initial-scale=1.0">
    <link rel="preconnect"
href="https://fonts.googleapis.com">
    <link rel="stylesheet" href="{{
url_for('static', filename='css/auth.css') }}">
</head>

```

```

<body>
  <div class="login-container">
    <div class="logo">
      
      <h1>SyncSphere</h1>
    </div>
    <h2>Access your files securely</h2>
    <form action="{{ url_for('auth.login') }}"
method="post">
      <input type="hidden" name="csrf_token"
value="{{ csrf_token() }}">
      <input type="text" name="username_or_email"
placeholder="Username or Email" required>
      <input type="password" name="password"
placeholder="Enter your password" required>
      <label>
        <input type="checkbox" name="remember_me"
value="on"> Remember Me
      </label>
      <button type="submit">Log in</button>
    </form>
    <p class="sign-up-text">
      Need an account? <a href="{{
url_for('auth.register') }}">Sign Up</a>
    </p>
    {% with messages =
get_flashed_messages(with_categories=True) %}
      {% if messages %}

```

```

        <div class="flash-messages">
            <ul>
                {% for category, message in messages
%}
                    <li class="{{ category }}">{{
message }}</li>
                {% endfor %}
            </ul>
        </div>
    {% endif %}
{% endwith %}
</div>
</body>
</html>

```

templates/[register](#).html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>SyncSphere - Register</title>
    <meta name="viewport"
content="width=device-width, initial-scale=1.0">
    <link rel="preconnect"
href="https://fonts.googleapis.com">
    <link rel="stylesheet" href="{{
url_for('static', filename='css/auth.css') }}">
</head>
<body>

```

```

<div class="login-container">
  <div class="logo">
    
    <h1>SyncSphere</h1>
  </div>
  <h2>Create your account</h2>
  <form action="{{ url_for('auth.register') }}"
method="post">
    <input type="hidden" name="csrf_token"
value="{{ csrf_token() }}">
    <input type="text" name="username"
placeholder="Choose a username" required>
    <input type="email" name="email"
placeholder="Your email address" required>
    <input type="password" name="password"
placeholder="Create a password" required>
    <button type="submit">Register</button>
  </form>
  <p class="sign-up-text">
    Already have an account? <a href="{{
url_for('auth.login') }}">Log In</a>
  </p>
  {% with messages =
get_flashed_messages(with_categories=True) %}
    {% if messages %}
      <div class="flash-messages">
        <ul>

```



```

        {% for category, message in messages
%}

        <li class="{{ category }}">{{
message }}</li>

        {% endfor %}

    </ul>

</div>

{% endif %}

{% endwith %}

</div>
</body>
</html>

```

static/css/auth.css:

```

:root {
    --primary-color: #4f80e1;
    --text-color: #333;
    --bg-color: #fff;
    --font-family: 'Poppins', sans-serif;
}

* {
    box-sizing: border-box;
}

body {
    margin: 0;
    padding: 0;
}

```

```
font-family: var(--font-family);
background-color: var(--bg-color);
color: var(--text-color);
}

.navbar {
  display: flex;
  align-items: center;
  justify-content: space-between;
  height: 60px;
  padding: 0 2rem;
  background-color: #fff;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.navbar .logo {
  display: flex;
  align-items: center;
}

.navbar .logo img {
  height: 32px;
  margin-right: 8px;
}

.navbar .logo h1 {
  font-size: 1.5rem;
  margin: 0;
}
```

```
.navbar nav ul {
  list-style: none;
  display: flex;
  gap: 1rem;
  margin: 0;
  padding: 0;
}

.navbar nav ul li a {
  text-decoration: none;
  color: var(--text-color);
  font-weight: 500;
  transition: color 0.3s ease;
}

.navbar nav ul li a:hover {
  color: var(--primary-color);
}

.container {
  width: 90%;
  max-width: 1100px;
  margin: 0 auto;
  padding: 2rem 0;
}

.files-section {
  text-align: center;
  margin-bottom: 3rem;
}
```

```
.files-section h1 {  
  font-size: 2rem;  
  margin-bottom: 1rem;  
}  
  
.file-actions {  
  margin-top: 1rem;  
}  
  
.file-actions button {  
  background-color: var(--primary-color);  
  color: #fff;  
  border: none;  
  padding: 0.75rem 1.5rem;  
  margin: 0.5rem;  
  border-radius: 4px;  
  cursor: pointer;  
  font-size: 1rem;  
  transition: opacity 0.3s ease;  
}  
  
.file-actions button:hover {  
  opacity: 0.9;  
}  
  
.hero-section {  
  text-align: center;  
  padding: 3rem 2rem;  
  background-color: #f8f9fc;
```

```
border-top: 1px solid #e2e2e2;
}

.hero-section h2 {
  font-size: 2rem;
  margin-bottom: 1rem;
}

.hero-section p {
  font-size: 1.1rem;
  line-height: 1.6;
  margin-bottom: 2rem;
  max-width: 600px;
  margin-left: auto;
  margin-right: auto;
}

.hero-section button {
  background-color: var(--primary-color);
  color: #fff;
  border: none;
  padding: 1rem 2rem;
  border-radius: 4px;
  cursor: pointer;
  font-size: 1.1rem;
  transition: opacity 0.3s ease;
}

.hero-section button:hover {
  opacity: 0.9;
```

```
}

.footer {
  text-align: center;
  padding: 1rem;
  font-size: 0.9rem;
  background-color: #fff;
  border-top: 1px solid #e2e2e2;
}

.btn {
  background-color: var(--primary-color);
  color: #fff;
  border: none;
  padding: 0.75rem 1.5rem;
  border-radius: 4px;
  cursor: pointer;
}

.btn:hover {
  opacity: 0.9;
}

.login-container {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  height: 100vh;
  width: 100%;
}
```

```
}

.login-container .logo {
  display: flex;
  align-items: center;
  gap: 1rem;
  margin-bottom: 1rem;
}

.login-container .logo img {
  width: 50px;
}

.login-container .logo h1 {
  font-size: 1.5rem;
  margin: 0;
}

.login-container h2 {
  font-size: 1.2rem;
  margin: 1rem 0;
}

.login-container form {
  display: flex;
  flex-direction: column;
  width: 300px;
  gap: 1rem;
}
```

```
.login-container form input {
  padding: 0.75rem;
  border: 1px solid #ccc;
  border-radius: 4px;
}

.login-container form button {
  background-color: var(--primary-color);
  color: #fff;
  border: none;
  padding: 0.75rem;
  border-radius: 4px;
  cursor: pointer;
  transition: opacity 0.3s ease;
}

.login-container form button:hover {
  opacity: 0.9;
}

.sign-up-text {
  margin-top: 1rem;
  font-size: 0.9rem;
}

.flash-messages ul {
  list-style: none;
  padding: 0;
  margin: 1rem 0;
}
```



```
.flash-messages li.success {
  background-color: #d4edda;
  border: 1px solid #c3e6cb;
  color: #155724;
  padding: 0.75rem;
  border-radius: 4px;
  margin-bottom: 0.5rem;
}

.flash-messages li.error {
  background-color: #ffe7e7;
  border: 1px solid #ffa3a3;
  color: #d8000c;
  padding: 0.75rem;
  border-radius: 4px;
  margin-bottom: 0.5rem;
}
```

static/css/dashboard.css:

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
```

```
    font-family: "Segoe UI", Tahoma, Geneva,  
Verdana, sans-serif;  
    background-color: #f6f8fa;  
    color: #333;  
}  
  
.navbar {  
    display: flex;  
    align-items: center;  
    justify-content: space-between;  
    background-color: #fff;  
    padding: 1rem 2rem;  
    border-bottom: 1px solid #ddd;  
}  
  
.navbar-left {  
    display: flex;  
    align-items: center;  
}  
  
.logo {  
    height: 40px;  
    margin-right: 0.5rem;  
}  
  
.app-name {  
    font-size: 1.25rem;  
    font-weight: 600;  
    color: #222;  
}
```

```
.navbar-right a {
  margin-left: 1.5rem;
  text-decoration: none;
  color: #555;
  font-weight: 500;
}

.navbar-right a:hover {
  color: #000;
}

main {
  max-width: 1200px;
  margin: 2rem auto;
  padding: 0 2rem;
}

.files-section {
  background-color: #fff;
  padding: 2rem;
  border-radius: 8px;
  margin-bottom: 2rem;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.files-section h2 {
  font-size: 1.5rem;
  margin-bottom: 1rem;
}
```

```
.files-table {  
  width: 100%;  
  border-collapse: collapse;  
  margin-top: 1rem;  
}  
  
.files-table thead {  
  background-color: #f0f0f0;  
}  
  
.files-table th,  
.files-table td {  
  padding: 0.75rem;  
  text-align: left;  
  border: 1px solid #ddd;  
}  
  
.no-files {  
  margin-top: 1rem;  
}  
  
.btn {  
  display: inline-block;  
  padding: 0.5rem 1rem;  
  margin-right: 0.5rem;  
  text-decoration: none;  
  font-size: 0.875rem;  
  border-radius: 4px;  
  cursor: pointer;
```

```
border: none;
}

.btn-download {
  background-color: #007BFF;
  color: #fff;
}

.btn-download:hover {
  background-color: #0056b3;
}

.btn-delete {
  background-color: #dc3545;
  color: #fff;
}

.btn-delete:hover {
  background-color: #c82333;
}

.upload-cta {
  text-align: center;
  background-color: #fff;
  padding: 2rem;
  border-radius: 8px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.upload-cta h2 {
```

```
    font-size: 1.5rem;
    margin-bottom: 1rem;
}

.upload-cta p {
    color: #666;
    margin-bottom: 1.5rem;
}

.upload-form {
    display: inline-block;
    text-align: left;
}

.upload-input {
    margin-bottom: 1rem;
}

.btn-upload {
    background-color: #28a745;
    color: #fff;
}

.btn-upload:hover {
    background-color: #218838;
}

.btn-remove {
    background-color: #dc3545;
    color: #fff;
}
```

```
}

.btn-remove:hover {
  background-color: #c82333;
}

.flash-messages {
  max-width: 1200px;
  margin: 1rem auto;
  padding: 1rem 2rem;
  background-color: #d4edda;
  border: 1px solid #c3e6cb;
  border-radius: 8px;
  color: #155724;
}

.flash-messages ul {
  list-style: none;
}

.flash-messages li.success {
  background-color: #d4edda;
  border: 1px solid #c3e6cb;
  color: #155724;
  padding: 0.75rem;
  border-radius: 4px;
  margin-bottom: 0.5rem;
}

.flash-messages li.error {
```

```
background-color: #d4edda;
border: 1px solid #c3e6cb;
color: #155724;
padding: 0.75rem;
border-radius: 4px;
margin-bottom: 0.5rem;
}

.action-section .action-form {
  display: flex;
  gap: 0.5rem;
}

.action-section .action-form
input[name="to_username"] {
  flex: 1;
  min-width: 0;
  height: 2.5rem;
  font-size: 1.1rem;
  padding: 0.4rem 0.6rem;
  border-radius: 4px;
  border: 1px solid #ccc;
}

.action-section .action-form button.btn-upload {
  height: 2.6rem;
  padding: 0 1.2rem;
  font-size: 1.1rem;
  border-radius: 4px;
}
```



```
.list-section {
  background-color: #fff;
  padding: 2rem;
  border-radius: 8px;
  max-width: 1000px;
  margin: 2rem auto;
  box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

.list-section h2 {
  font-size: 2rem;
  margin-bottom: 1.5rem;
  text-align: center;
}

.friends-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill,
minmax(240px,1fr));
  gap: 1.5rem;
}

.friend-card {
  background-color: #fdfdfd;
  padding: 1.5rem;
  border-radius: 8px;
  text-align: center;
  box-shadow: 0 1px 3px rgba(0,0,0,0.1);
  display: flex;
```

```
flex-direction: column;
align-items: center;
}

.friend-info .friend-name {
  font-size: 1.25rem;
  font-weight: 600;
  margin: 0;
}

.friend-info .friend-username {
  font-size: 1rem;
  color: #555;
  margin: 0.25rem 0;
}

.friend-info .friend-details {
  font-size: 0.875rem;
  color: #777;
  margin: 0.2rem 0;
}

.friend-actions {
  margin-top: auto;
  display: flex;
  gap: 0.5rem;
}

.no-friends {
  text-align: center;
```

```
font-size: 1.1rem;
color: #555;
}

.list-section .action-form {
  display: flex;
  justify-content: center;
  align-items: center;
  gap: 0.5rem;
  margin-bottom: 2rem;
  max-width: 600px;
  margin-left: auto;
  margin-right: auto;
}

.list-section .action-form
input[name="to_username"] {
  flex: 2;
  min-width: 200px;
  height: 2.5rem;
  font-size: 1.1rem;
  padding: 0.4rem 0.6rem;
  border-radius: 4px;
  border: 1px solid #ccc;
}

.list-section .action-form button.btn-upload {
  height: 2.6rem;
  padding: 0 1.2rem;
  font-size: 1.1rem;
```

```
border-radius: 4px;  
}
```

קישור לעמוד ה-github של הפרויקט:

<https://github.com/eyalp7/SyncSphere>