אפליקציה לתקשורת בין אנשים – שליחת הודעות, קבצים וביצוע שיחות - ChatEase

בית ספר: ויצו הדסים

מגיש: עומר דגרי

ת.ז: 215200908

שם המנחה: ניר דוויק

תאריך הגשה: 31.05.2023

תוכן עניינים

1. מבוא

1.1 תכולת ספר הפרויקט

ספר זה מתאר את הסיבות להכנת האפליקציה ChatEase, כיצד היא עובדת, תהליך המחקר והפיתוח, מדריך למשתמש ולמפתח וקוד עדכני לתאריך ההגשה, אני מוסיף פה קישור לrepository של הGitHub של הפרויקט אשר יכיל את הקוד המעודכן תמיד (מאחר ואני אמשיך לעבוד גם אחרי הגשת ספר הפרויקט עד הבגרות).
https://github.com/Omer-Dagry/ChatEase

1.2 הרקע לפרויקט

רציתי לבחור בפרויקט שיאתגר אותי, ושההיקף עבודה שלו יהיה גדול. אז חשבתי על כל מיני רעיונות ושאלתי את חברים שלי ואת ההורים שלי מה לדעתם יותר מעניין וכמעט כולם אמרו שפרויקט של שיחות וצאטים מעניין יותר משאר הרעיונות, אני גם חשבתי שזה יהיה מאתגר יותר משאר הרעיונות, לדוגמה רעיון אחר לפרויקט שרציתי היה מערכת קבצים עבור ארגונים אבל הפרויקט הזה סוג של מכיל אותו בפנים מאחר ולכל משתמש באפליקציה שלי יש את כל הקבצים שלו ולכל צאט יש קבצים והם מוגבלים רק למשתמשים בצאט אבל בנוסף לכל זה אני מאפשר גם שליחת הודעות וקבצים ושיחות אז בחרתי בפרויקט הזה כי ידעתי שהוא יהיה מאתגר יותר משאר הרעיונות.

1.3 תהליך המחקר

פה לא היה הרבה לעשות מאחר ואפליקציות צ'אטים זה משהו שאנחנו משתמשים בחיי היום יום ואנחנו מודעים לאפשרויות שיש בשוק כגון whatsapp, telegram, Instagram, facebook messenger ועוד.

יתרונות:
1. הפרויקט שלי מאחסן את המידע של הצ'אטים בשרת כך שניתן להתחבר מכל מכשיר חדש למשתמש והמידע ישמר ויהיה את כל ההיסטוריה בלי שזה יתאכסן בחשבון גוגל דרייב שלכם כמו בוואצאפ לדוגמה.
2. אצלי כל התקשורת מאובטחת והמידע נשמר בשרת לעומת וואצאפ שהאבטחה היא מקצה לקצה.
3. בנוסף הפרויקט שלי מציע GUI עם כל מיני אפקטים נחמדים בניגוד לשאר האפשרויות בשוק.
4. הפרויקט שלי מאפשר לבצע שיחות וועידה דרך המחשב דבר שעד לאחרונה לא היה אצל שאר האפליקציות בשוק (למיטב ידיעתי) חוץ מוואצאפ שהוסיפו בשנה האחרונה.

1.4 אתגרים מרכזיים

1. חיפוש GUI שיאפשר לי לעצב אותו בצורה שאני אהיה מרוצה ממנה, בסוף בחרתי בספרייה eel בפייתון אשר מאפשר לי לעבוד עם html, css, js ביחד עם הפייתון.
2. לחשוב על דרך מהירה לנעול את הקבצים של הצאט ברגע שאני משנה אותם בלי לנעול משתמשים שלא נמצאים בצאט או משתמשים שנמצאים בצאט אבל לא מנסים לשלוח הודעה לצאט הזה, על מנת למנוע איבוד הודעות.
3. כל הקטע של השיחות, מאחר שאני מאפשר לעשות שיחות וועידה (כלומר בין יוצר מ2 אנשים) הייתי צריך למצוא דרך לעשות את השרת של השיחות מהיר, מאחר ונגיד ומחוברים אליו 5 משתמשים עבור כל חבילה שהוא מקבל (והוא מקבל הרבה בסביבות ה10 עד 15 חבילות בשנייה מכל לקוח) הוא צריך לשלוח 4 חבילות (את אותה החבילה לכל שאר משתתפי השיחה) משמע עבור כל חבילה יש לו פי 4

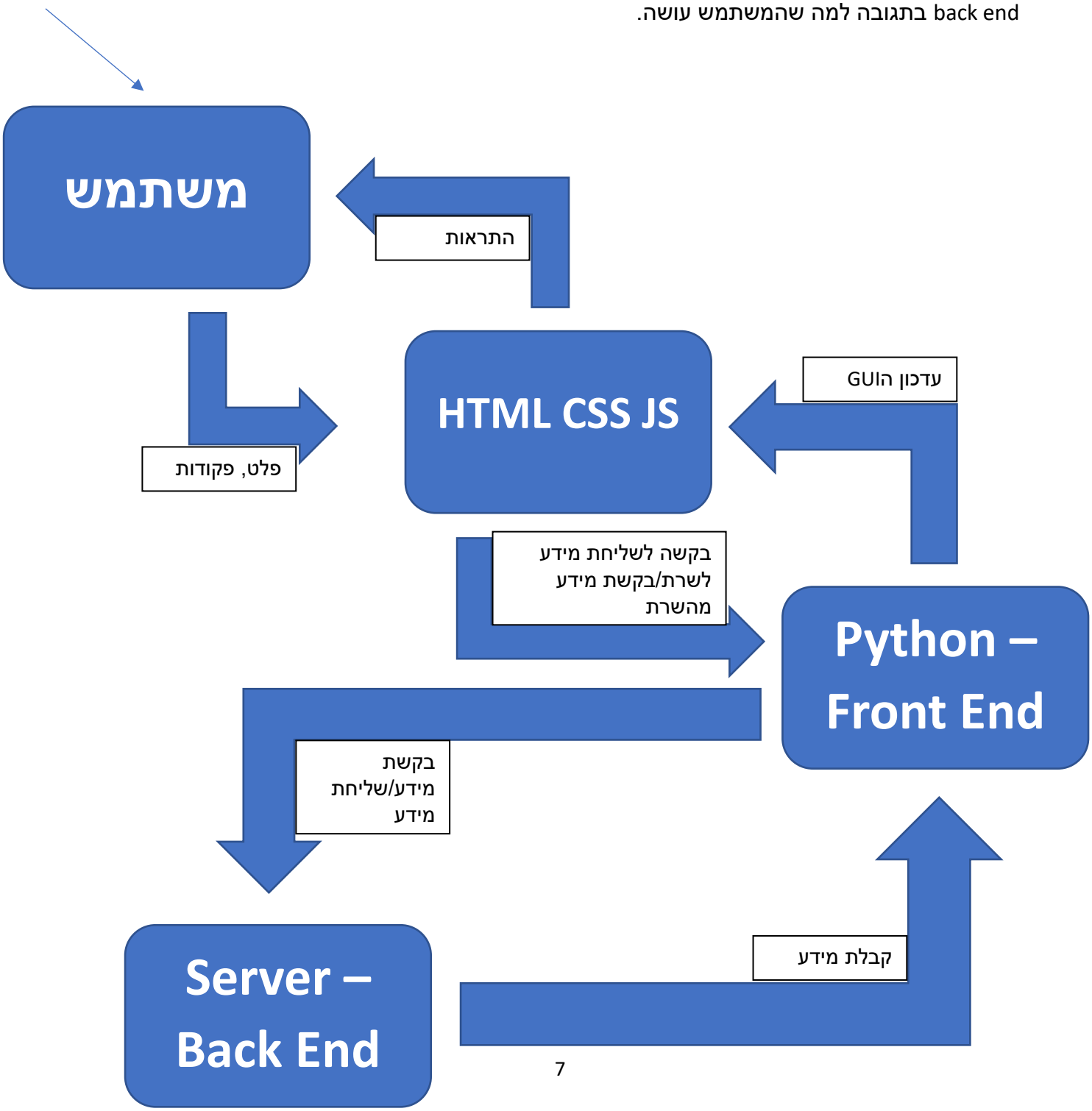
חבילות לשלוח במקרה הזה אז קרה מצב שאחרי כמה זמן הצטברו אצלי הרבה חבילות בגרסה הראשונה של השרת שיחות מאחר ולא עשיתי אותו מספיק מהיר ואז בתחילת השיחה הכל היה חלק אבל אחרי כמה שניות פתאום התחילו קטיעות (רק בשיחות וועידה).

4. בהתחלה היו לי בעיות עם הקישור של הפייתון לjs בעזרת הספרייה eel אבל אחרי ששיחקתי עם זה קצת ובדקתי באינטרנט (מהקצת מידע שיש על זה) הצלחתי לעבוד עם זה.

2. מבנה / ארכיטקטורה

הפרויקט מחולק ל2 חלקים, חלק ה front end וחלק ה back end, בנוסף חלק הfront end מכיל גם שני חלקים הפייתון וה html css js.

ה front end אחראי על האינטרקציה של המשתמש עם התוכנה וחלק ה back end שאחראי על הטיפול בכל הבקשות של המשתמשים.
המשתמש לא מתקשר ישירות עם חלק ה back end אלא חלק ה front end מתקשר בשבילו עם ה back end בתגובה למה שהמשתמש עושה.



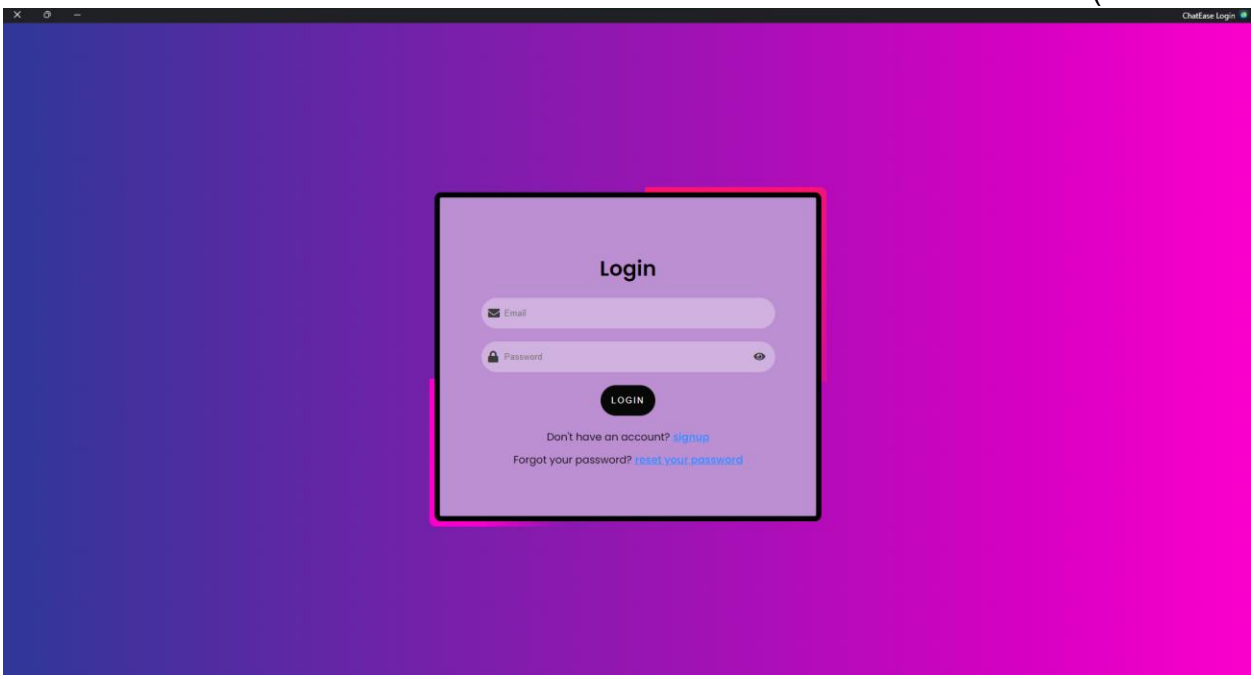
2.1 Front-End

חלק זה מכיל כמה קבצים שונים בשפות שונות.
יש את העמודים של הlogin, signup, reset password ואת העמוד של האפליקציה עצמה.

עבור כל אחד מהעמודים האלו יש קובץ html אחד, קובץ css אחד וקובץ js אחד או יותר.
כל event שדורש תקשורת עם השרת או קבלת מידע חדש על מנת לעדכן את הGUI הjs מתקשר עם הpython ומקבל את מה שהוא צריך על מנת לבצע את הפעולה. זאת אומרת שעבור כל event כזה יש פונקציה גם בpython שתטפל בה וגם בjs.

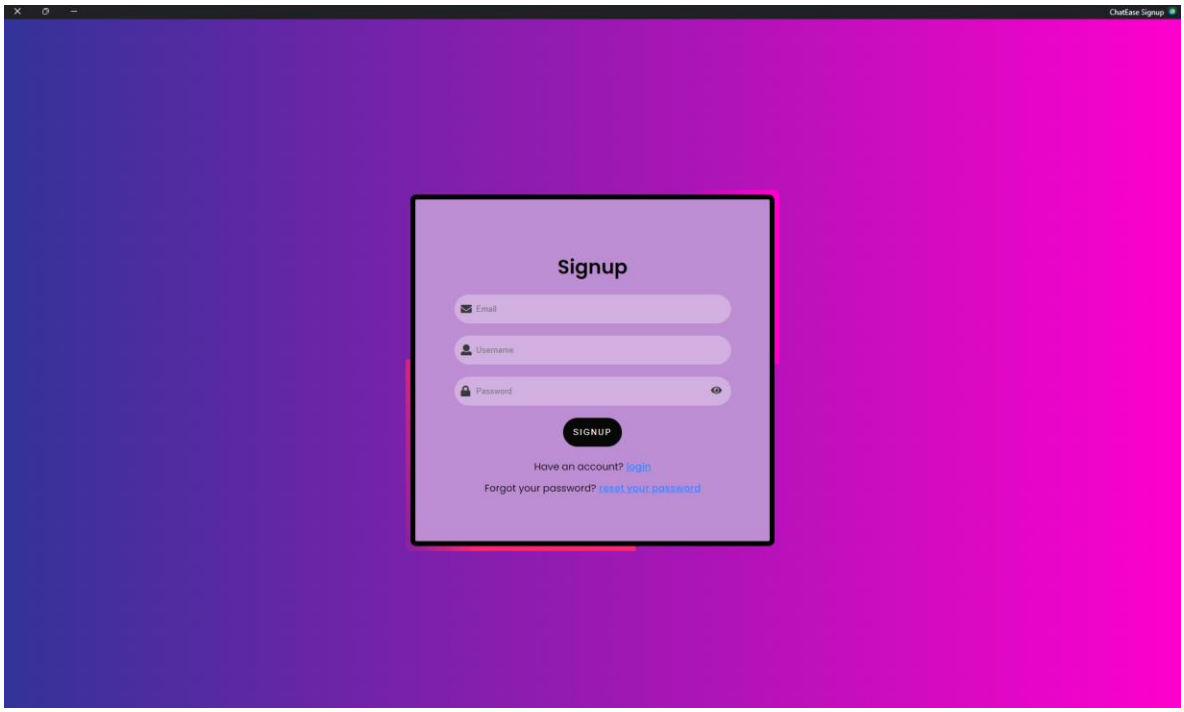
2.1.1 Login Page

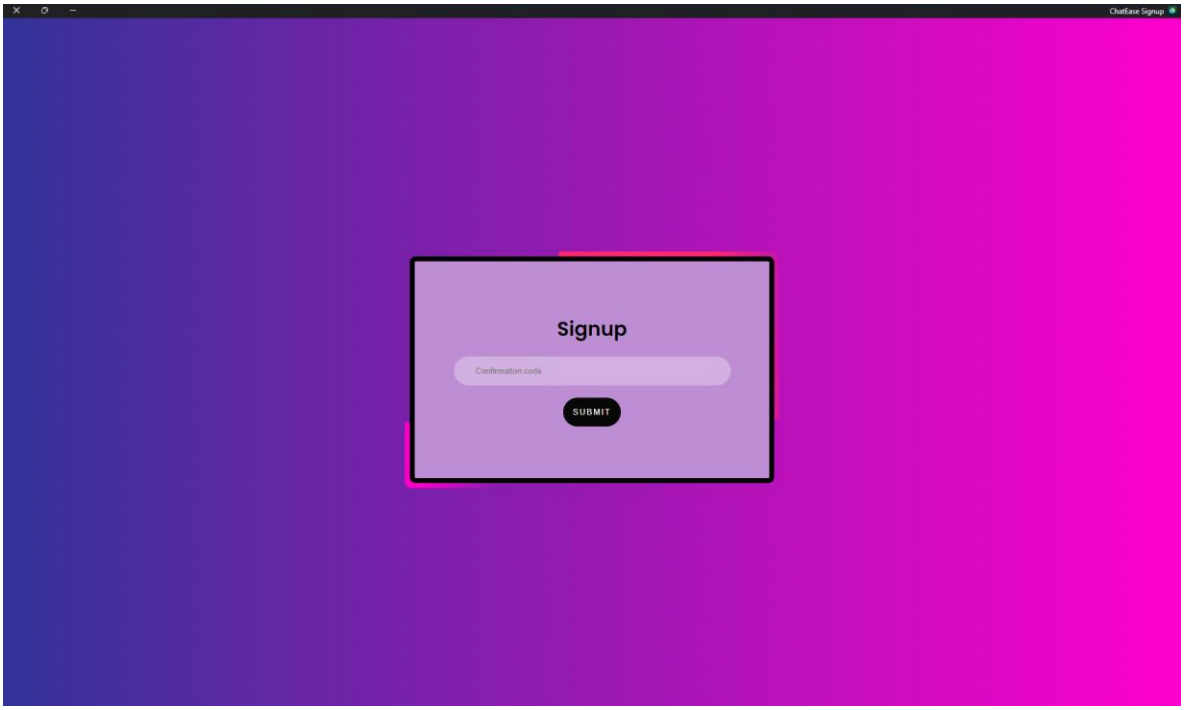
בדף זה ניתן להתחבר למשתמש שקיים או לעבור לאחד מהדפים האחרים - הרשמה או איפוס סיסמה.
בדף זה המשתמש מתבקש להכניס את האימייל והסיסמה שלו לאפליקציה (ניתן להסתיר או להראות את הסיסמה).



Signup Page 2.1.2

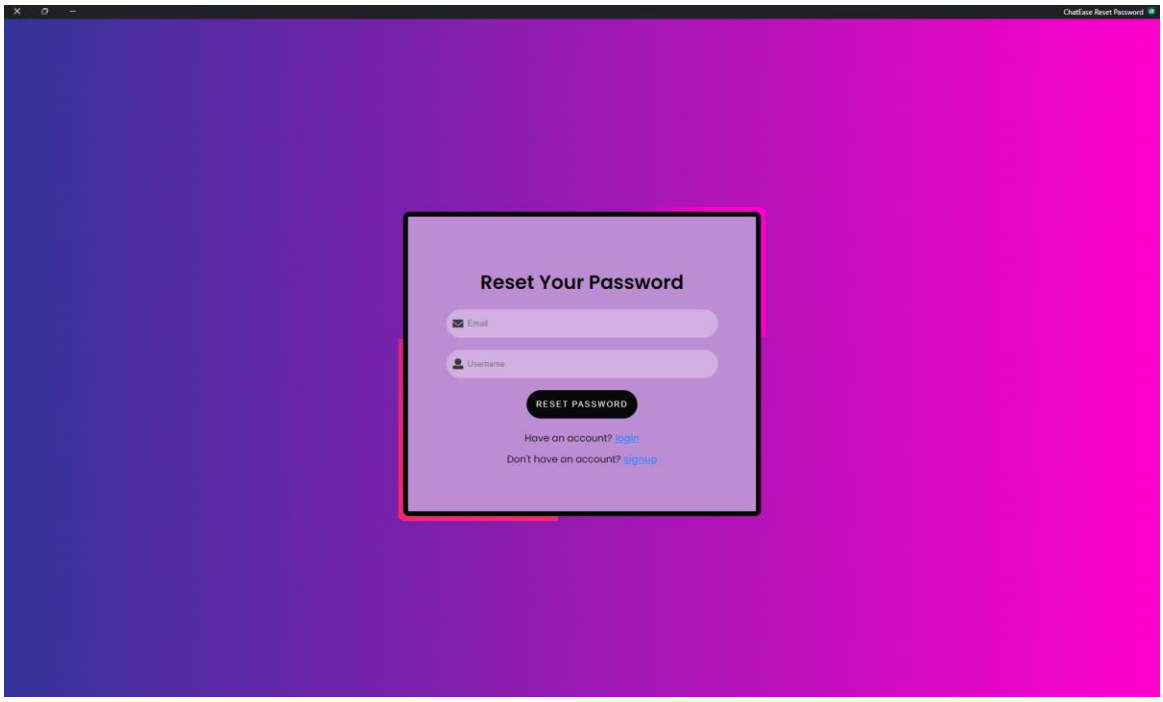
בדף זה ניתן ליצור משתמש חדש או לעבור לאחד מהדפים האחרים - התחברות או איפוס סיסמה.
בדף זה המשתמש מתבקש להכניס את האימייל, השם משתמש שהוא רוצה ואת הסיסמה שלו לאפליקציה (ניתן להסתיר או להראות את הסיסמה). ולאחר מכן ישלח לאימייל שלו קוד לאישור הרשמה ורק לאחר שיכניס אותו השרת יצור לו משתמש חדש (בתנאי שאין לאימייל משתמש קיים).

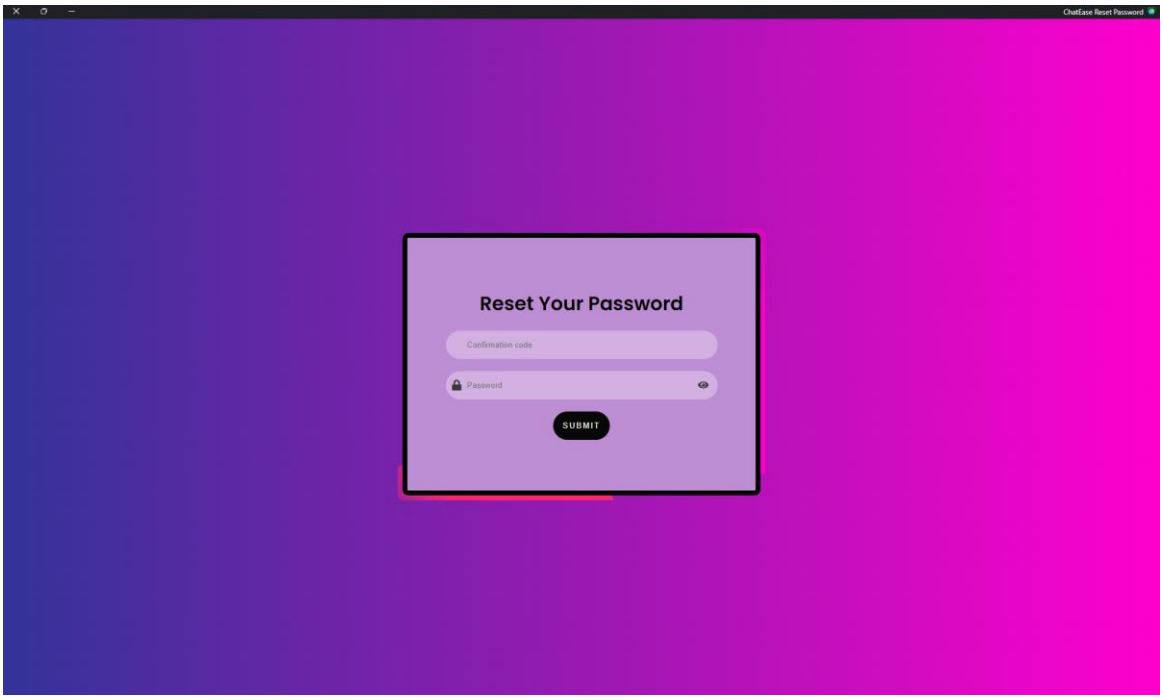




Reset Password Page 2.1.3

בדף זה ניתן לאפס סיסמה למשתמש קיים או לעבור לאחד מהדפים האחרים - התחברות או הרשמה.
בדף זה המשתמש מתבקש להכניס את האימייל שלו ואת השם משתמש ולאחר מכן ישלח לו לאימייל קוד לאישור איפוס הסיסמה ויתאפשר לו להכניס את הקוד והסיסמה החדשה אם הקוד נכון תופיע הודעה שהסיסמה אופסה בהצלחה אם לא תופיע הודעת שגיאה.

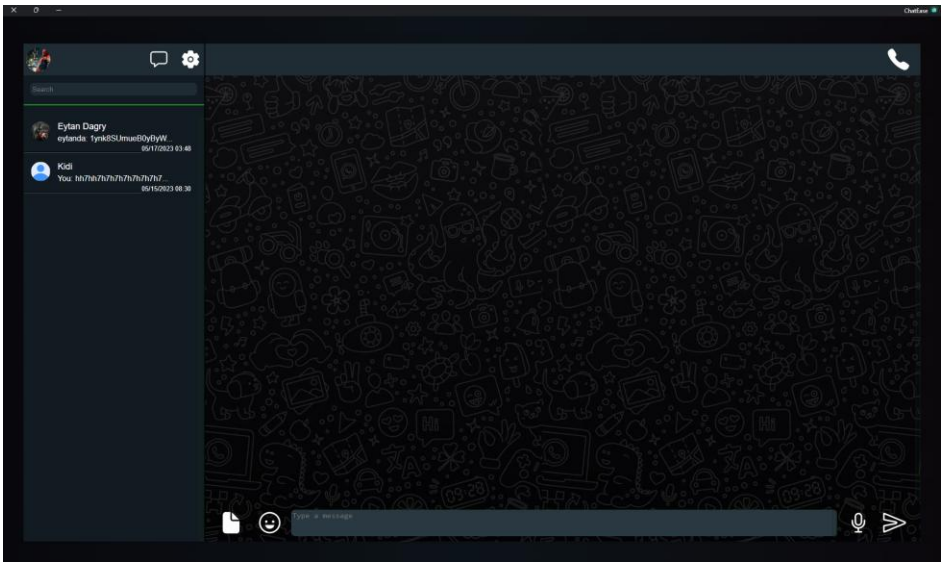


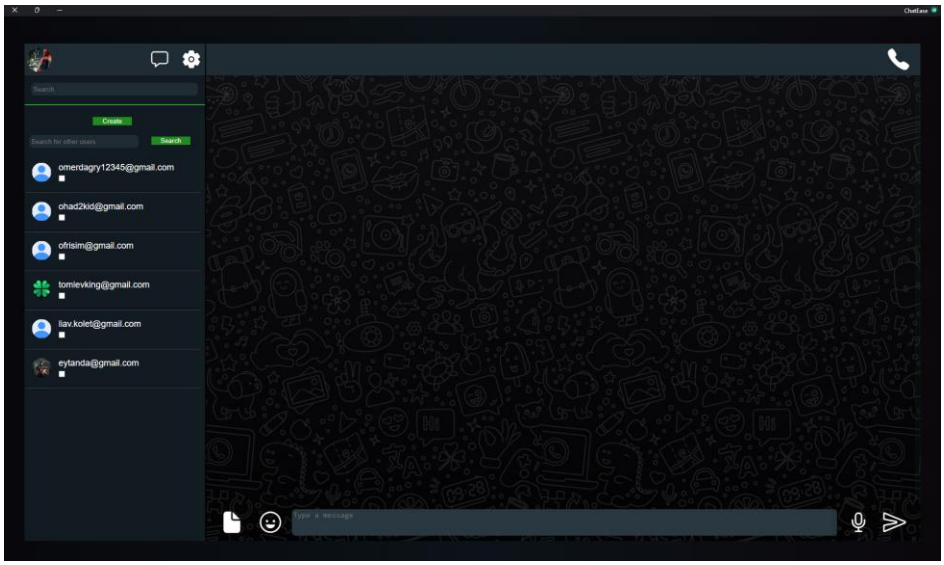


2.1.4 האפליקציה עצמה – ChatEase

בדף הזה המשתמש כבר מחובר ונמצא בתוך האפליקציה.
במצב הזה הוא יכול להיכנס לכל הצאטים שלו, לשנות תמונת פרופיל, ליצור צאט בינו לבין מישהו 1 או ליצור קבוצה חדשה, לשלוח הודעות, ואת כל סוגי הקבצים, תמונות יופיעו בתור תמונה וקבצים אחרים בתור שם קובץ ואם לוחצים זה פותח אותם (גם תמונות וגם קבצים אחרים), להקליט ולשלוח הודעות קוליות.

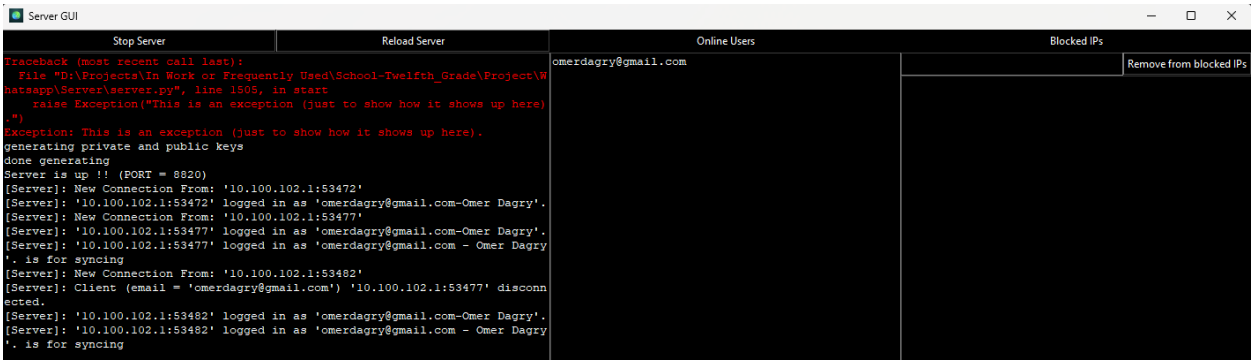
בכל צאט בין יחידים וגם בצאט של קבוצה יש אפשרות להתחיל שיחה, בצד ימין למעלה במסך יש כפתור של טלפון. ניתן לעשות שיחות וועידה ולא רק שיחות של 1 על 1.





Back-End 2.2

בצד של השרת יש את קובץ הGUI של השרת, את הקובץ העיקרי של השרת בנוסף יש 4 מודולים - מודול לשיחות, מודול "לנעילת" קבצים בצורה שלא מפריעה למשתמשים אחרים אלא אם הם מנסים לגשת לקובץ הספציפי הזה בזמן שהוא נעול, מודול לסוקט עבור הצד של השרת שהוא מאובטח ובעל פרוטוקול (שליחת אורך הודעה מאופס ל30 תווים ואז את ההודעה) ומודול לdatabase אשר מסונכרן בין threads ובין processes שזה היה עבודה שעשינו השנה בלימודים והחלטתי להשתמש בה לפרויקט (שיניתי כמה דברים).



זה הGUI של השרת, בGUI ניצן לעצור את השרת ולהפעיל, ניתן גם לעשות reload לקובצי שרת, זה היה בשבילי בשביל הפיתוח, יותר נוח ללחוץ על הכפתור ושזה יריץ את הגרסה החדשה של השרת מאשר לעצור את השרת ולהריץ בעצמי כל פעם.

הGUI הוא הconsole של השרת כלומר הוא מציג את הstdout וstderr, העלתי שגיאה בכוונה כדי להראות שזה גם מודפס פה.

ניתן גם לראות את כל המשתמשים המחוברים כפי שרואים בתמונה.

בנוסף, יש blocked ips, השרת חוסם IP מסוים למשך הזמן שמוגדר בקובץ השרת (5 דקות כרגע) במקרה שנגרמו יותר exception ממה שמוגדר שמותר בזמן מוקצב (כרגע 100 exceptions ב5 דקות), יש גם אפשרות להסיר IP מהblocked ips וגם שעוצרים את השרת עשיתי שזה מאפס את הblocked_ips.

לבדיקה של המנגנון חסימה ניתן להריץ את הקוד הבא אשר שולח לשרת 200 פעמים פקודה לא קיימת (חייבים לפתוח סוקט חדש כל פעם ששולחים את הפקודה מאחר וכשהשרת מקבל פקודה לא מוכרת הוא אוטומטית סוגרת את התקשורת עם הסוקט הזה):

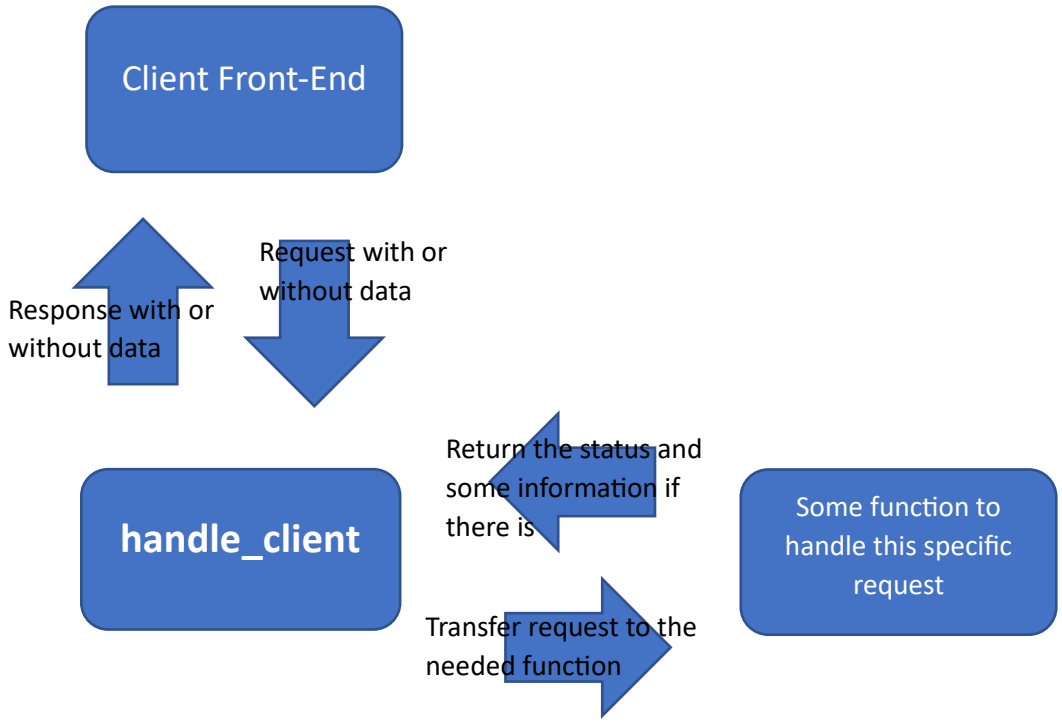
```
from ClientSecureSocket import ClientEncryptedProtocolSocket


for _ in range(200):
    sock = ClientEncryptedProtocolSocket()
    sock.connect(("127.0.0.1", 8820))
    sock.send_message(b"unknown")
    sock.close()
```

השרת מתחיל מיצירת כל הdatabaseים הנחוצים – אימייל סיסמה, אימייל שם משתמש, id של צאט וכל המשתמשים שבו, אימייל סטטוס (מחובר או לא ואם לא מתי היה לאחרונה). כל אלו משתמשים במודול של הdatabase.

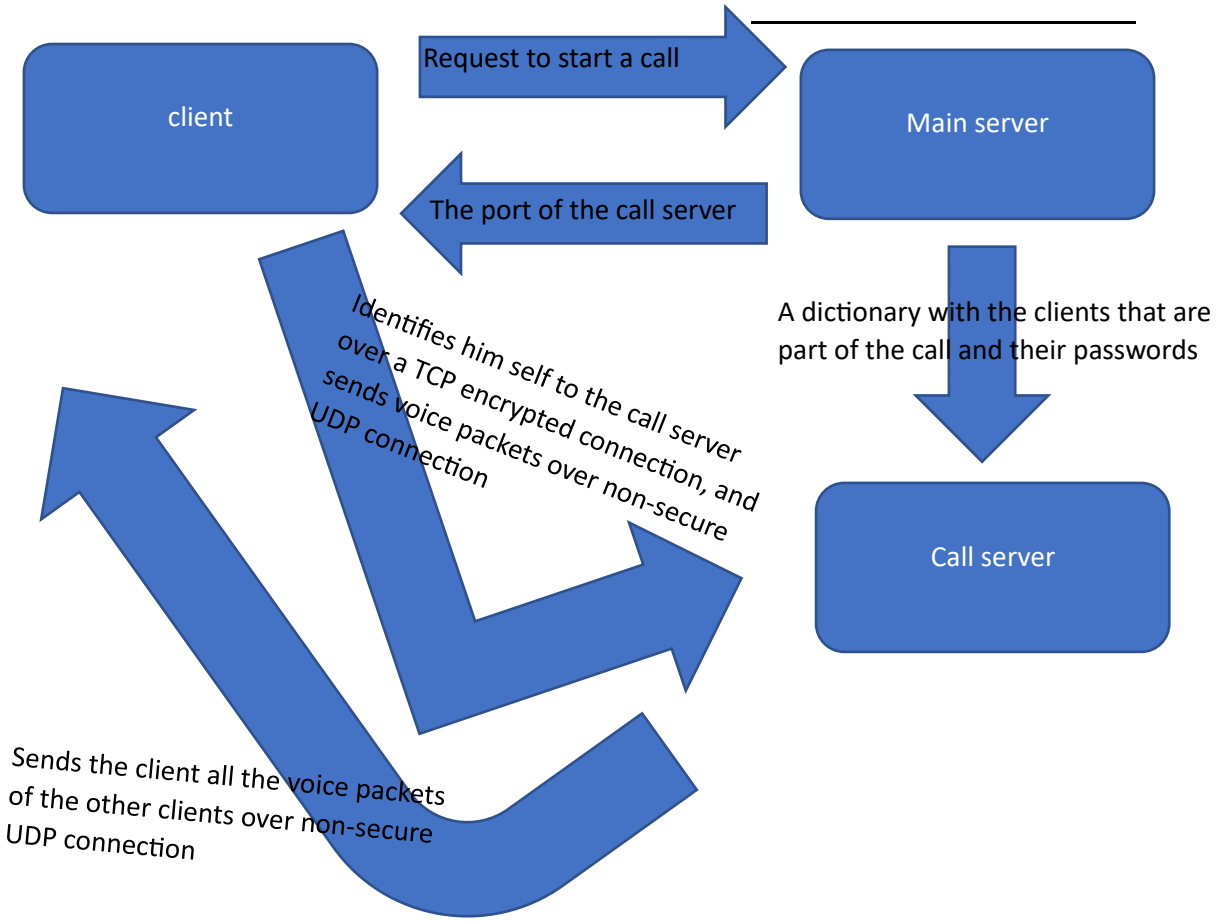
לאחר מכן הוא מוחק את כל המנעולים של המודול לנעילת קבצים במקרה והשרת נעצר על ידי KeyboardInterrupt ולא סיים כמו שצריך. אחרי זה אם יש לקוח שעדיין מסומן בdatabase שהוא מחובר (גם קורה בגלל KeyboardInterrupt) הוא משנה את זה.

רק אחרי כל זה מתחילה הפונקציה main בפונקציה זו השרת מייצר לו מפתחות public וprivate לאחר מכן יוצר סוקט עבור השרת עם המודול ServerSecureSocket (המודול שהזכרתי למעלה) אחרי זה הוא שולח מייל עם הip החיצוני שלו אל מייל משותף שיצרתי למשתמשים (למשתמשים אין גישה ישירה רק לפייתון), עשיתי את זה בעיקר בגלל שכשבדקתי את האפליקציה עם חברים לא היה לי כוח כל הזמן להעביר להם את הip החיצוני שלי ושהם יתחילו לשנות בקוד וגם שיצרתי exe לא היה ניתן לשנות את הקוד אז זה היה יותר בשביל שהפייתון יוכל לבדוק לבד מה הip של השרת כרגע, אני מודע שזה לא הדבר הכי חכם כי זה אימייל שנמצא אצל כל המשתמשים אבל זה היה יותר בשביל השלב של הבדיקה ולא באמת בתור מטרה לשימוש כפיצ'ר.

בשלב זה השרת מחכה ללקוחות חדשים ועבור כל לקוח הוא פותח thread חדש שדואג לטפל בו וכך גם מאפשר לעטוף כל לקוח במקרה ויש איזה exception הוא לא ישפיע על האחרים.



זאת הארכיטקטורה של השרת הראשי, בנוסף אליו ברגע שלקוח מתחיל שיחה השרת הראשי יפתח עוד שרת משני עבור השיחה בין הלקוחות (הספציפיים האלו, לכל שיחה יש שרת משלה), השרתים של השיחות כוללים גם תקשורת מוצפנת בTCP כמו שאר הפרויקט וגם תקשורת לא מוצפנת בUDP עבור השיחות עצמן.
בתור התחלה השרת הראשי פותח שרת שיחות ומעביר את כל המשתמשים והסיסמאות שלהם לשרת המשני אז כל משתמש צריך להתחבר בTCP לשרת המשני ולהזדהות ורק אז הIP שלו מאושר בתקשורת של הUDP והוא יקבל ויוכל לשלוח פקטות קוליות.

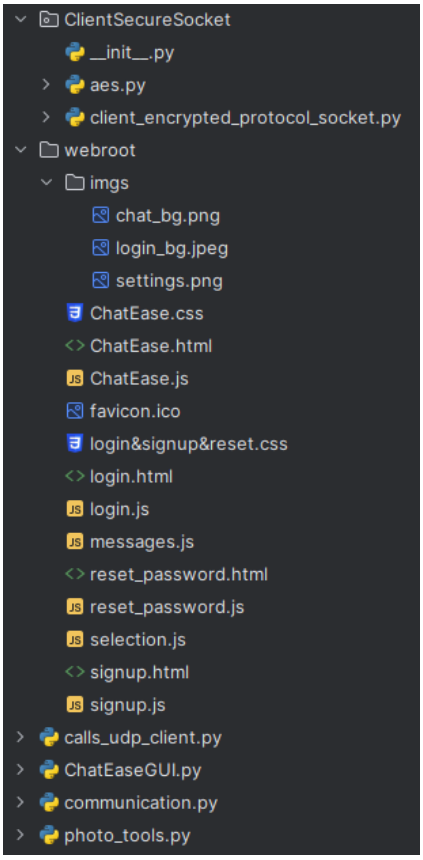


3. מדריך למשתמש

3.1 צד לקוח

אני יוצר exe של הצד של הלקוח אז כל מה שצריך זה לפתוח את הexe ליצור משתמש / לאפס סיסמה ואו להתחבר ולאחר מכן יש גישה לאפליקציה עצמה. כשמריצים את הexe זה כנראה יזהיר שזה יכול להיות ווירוס מאחר ואין לי דרך לחתום על הexe בתור חברה כי אין לי חברה, צריך ללחוץ על המשך בכל מקרה ורק אז האפליקציה תתחיל.
לדפים השונים של התחברות, הרשמה, איפוס סיסמה והאפליקציה עצמה ניתן לראות Front-End 2.1.

במקרה של שימוש ישירות בפייתון יש צורך בכל הקבצים הבאים כפי שהם מסודרים בתיקיות:



בנוסף בפייתון צריך את הספריות הבאות:

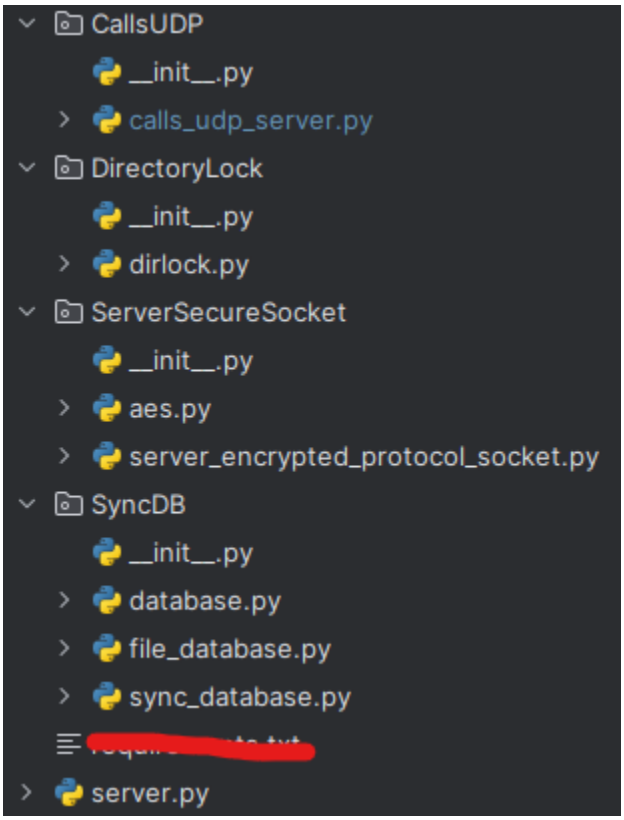
eel
rsa
numpy
pillow
pyaudio
easygui
pycryptodome

והגרסת פייתון צריכה להיות 3.11 ומעלה בעקבות הצורה של התיעוד של typesה בתוך הקבצי פייתון המאופשרת רק מגרסה זו ומעלה (ניתן גם בגרסאות קודמות אם מוסיפים בראש הקובץ from __future__ import annotations), יכול להיות שניתן גם בגרסה 3.10 אך אני לא בטוח.

3.2 צד שרת

גם פה, אני יוצר exe כך שאין צורך לעשות משהו מיוחד חוץ מלהריץ אותו.
לפירוט על התהליך של השרת ניתן לראות Back-End 2.2.

במקרה שמריצים ישירות את הפייתון יש צורך בכל הקבצים הבאים כפי שהם מסודרים בתיקיות:



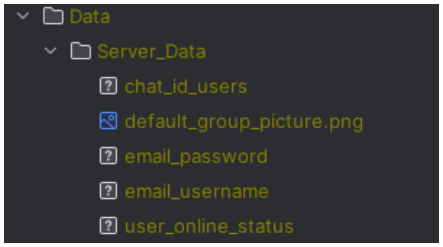
בנוסף בפייתון צריך את הספריות הבאות:
rsa
pycryptodome

והגרסת פייתון צריכה להיות 3.11 ומעלה בעקבות הצורה של התיעוד של typesה בתוך הקבצי פייתון המאופשרת רק מגרסה זו ומעלה (ניתן גם בגרסאות קודמות אם מוסיפים בראש הקובץ from __future__ import annotations), יכול להיות שניתן גם בגרסה 3.10 אך אני לא בטוח.

4. בסיס נתונים

הבסיסי נתונים נמצאים בצד של השרת וכאשר לקוח מתחבר מועבר אליו כל המידע שלו וכל הצ'אטים שלו.

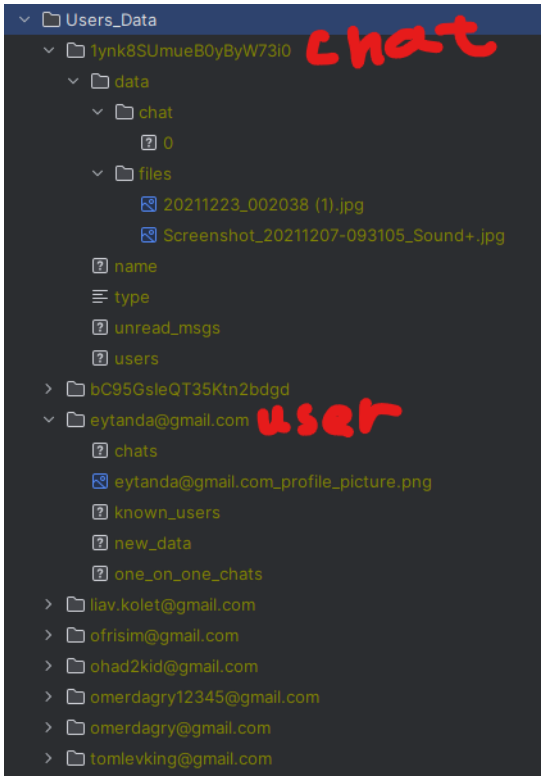
מידע עבור השרת על המשתמשים:



אלו כל הdatabaseים של השרת (בסופו של דבר זה dict אך זה מסונכרן בין threads ובין processes):

1. הid של כל הצ'אטים ועבור כל id כל המשתמשים בצ'אט. מפתח str ערך list[str]
2. אימייל של כל משתמש והסיסמה שלו (עוברת hash פעמיים, פעם אחת ישר אחרי שהמשתמש מכניס את הסיסמה בצד של הלקוח ועוד אחד ישר שהיא מגיעה לשרת כדי לא לשמור ישירות את הhash שאיתו ניתן להתחבר). מפתח str ערך str
3. אימייל של כל משתמש והשם משתמש שלו (כדי לאפשר לכמה אימייל להיות עם אותו של משתמש). מפתח str ערך str
4. האימייל של כל משתמש והסטטוס שלו (מחובר וכמה סוקטים שלו מחוברים או לא מחובר ומתי היה מחובר לאחרונה). מפתח str ערך list[str, datetime.datetime | int]

בנוסף המידע של כל הצ'אטים והמשתמשים נשמר בקבצים כפי שמתואר להלן:

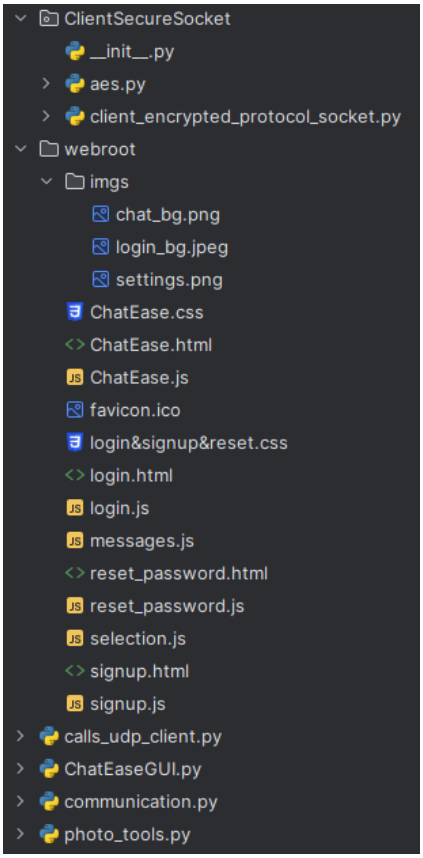


בצ'אטים: יש קובץ שמכיל את שם הצ'אט, קובץ שמכיל את סוג הצ'אט (יחיד או קבוצה) קובץ שמכיל את מספר ההודעות שלא נראו על ידי כל משתמש וקובץ שמכיל את כל המשתמשים שנמצאים בצ'אט. בנוסף יש את התיקייה שנקראת data ובתוכה את התיקיות chat וfiles בתיקייה של files פשוט יש את כל הקבצים שנשלחו בצ'אט, בתיקייה של הchat יש x קבצים (עבור כל 800 הודעות קובץ) כל קובץ כזה מכיל מילון שעבר serialization ו"נזרק" לקובץ המילון מכיל את הindex של ההודעה, את ההודעה עצמה וכל מיני meta data על ההודעה כגון מי ראה מי מחק עבור עצמה האם נמחקה לכולם (במקרה כזה ההודעה באמת נמחקת גם) הזמן שנשלחה ועוד.

במשתמשים: יש קובץ שמכיל את כל הצ'אטים שהם נמצאים בהם (chat id), קובץ של התמונת פרופיל שלהם, קובץ של האימיילים של כל המשתמשים שהם מכירים, קובץ של כל הצ'אטים שהם נמצאים בהם שהם לא קבוצות אלא אחד על אחד, הקובץ של הnew_data שמופיע בתמונה כבר לא קיים.

5. מדריך למפתח

5.1 צד לקוח



המודול ClientSecureSocket

1. __init__.py
2. aes.py

הקובץ מכיל מחלקה בשם AESCipher אשר משומשת על מנת להצפין ולפענח את כל התקשורת בין הלקוחות לשרת. במחלקה יש 4 פונקציות, ורק ל2 מהם צריך באמת לקרוא – encrypt, decrypt אשר מצפינות ומפענחות את המידע באמצעות הפרוטוקול AES השתיים האחרות הן _pad, _unpad אשר משומשות על מנת להשלים את האורך של המידע שיהיה באורך שמתאים להצפנה ובשביל להוריד את המידע המיותר שנוסף כדי להצפין.

3. *Client_encrypted_protocol_socket.py*

הקובץ מכיל מחלקה בשם ClientEncryptedProtocolSocket אשר מכילה 7 פונקציות לשימוש חיצוני – recv_message לקבלת הודעה 1 מהשרת, send_message לשליחת הודעה לשרת, connect בשביל להתחבר לשרת, set_timeout בשביל לשנות את הtimeout של הסוקט, get_timeout כדי לקבל את הtimeout שיש כרגע, getpeername כדי לקבל את הip והport שאליו אנחנו מחוברים, close על מנת לסגור את הסוקט.

ו2 פונקציות לשימוש פנימי - __recv_all על מנת לוודא קבלה של כל המידע, __exchange_aes_key אשר מבצעת החלפה עם השרת של המפתח הציבורי שהוא מייצר בהתחלה על מנת לשלוח לו בצורה

מוצפנת את המפתח להצפנת AES שתשמש אותנו להמשך התקשורת מאחר והצפנה זו מהירה יותר על מידע גדול באופן משמעותי.

התיקייה webroot

1. *התיקייה imgs* - מכילה 3 תמונות עבור הGUI
2. *ChatEase.css* - הקובץ עיצוב של האפליקציה עצמה
3. *ChatEase.html* - הקובץ הבסיסי לאפליקציה עצמה, לא מכיל יותר מידי רק את השלד של האתר כל השאר נעשה מהjs.
4. *ChatEase.js* - אשר מכיל את כל הפונקציות לטיפול בלחיצה על כל כפתור וכל הפונקציות לתקשורת עם הpython על מנת לקבל את כל הצ'אטים והמשתמשים המוכרים והתמונות וכו'.

5. *favicon.ico* - הicon של האתר
6. *login&signup&reset.css* – הקובץ עיצוב של העמודים של ההתחברות, הרשמה ואיפוס סיסמה.
7. *login.html* – *מכיל את הדף התחברות*
8. login.js – *מכיל כמה פונקציות בודדות אשר מעבירות לpython את השם משתמש והסיסמה כדי שיוכל להזדהות מול השרת.*
9. messages.js – *מכיל את כל סוגי ההודעות על מנת שיהיה אפשר לשכפל אותן כל פעם שיוצרים הודעה במקום ליצור מחדש כל פעם (גם נראה נקי יותר בקוד וגם אמור לעבוד מהר יותר השכפול מאשר היצירה).*
10. reset_password.html – *מכיל את הדף לאיפוס סיסמה.*
11. reset_password.js – *מכיל כמה פונקציות בודדות לתקשורת עם הpython.*
12. selection.js – not in use – *לא קיים יותר. היה בהתחלה כדי לא לאפשר לבחור טקסט ואז גיליתי שאפשר לעשות את זה מהcss.*
13. signup.html – *מכיל את הדף להרשמה.*
14. signup.js – *מכיל כמה פונקציות בודדות לתקשורת עם הpython.*

מחוץ לתיקיות ולמודולים - Other Files

1. *communication.py* – מכיל פונקציות להרשמה ואיפוס סיסמה, ומכיל מחלקה בשם Communication אשר משמשת לביצוע כל התקשורת עם השרת ברגע שהצד של הלקוח יודע מה השם משתמש והסיסמה של הלקוח, במחלקה זו ישנם 17 פונקציות:
   1.1. login – להתחברות רגילה.
   1.2. login_sync – להתחברות של סוקט המיודעת לסנכרון.
   1.3. sync – מחכה להודעת סנכרון מהשרת (כל פעם שיש מידע חדש) ושומרת את המידע החדש לקבצים.
   1.4. upload_file – פותחת thread ל_upload_file ומתחילה אותו.
   1.5. _upload_file - מעלה קובץ לשרת (בתור הודעה לצאט מסוים), לא אמורים לקרוא לה, רק לupload_file.
   1.6. send_message – בשביל לשלוח הודעה בצאט
   1.7. familiarize_user_with – בשביל לבדוק האם קיים משתמש ואם כן "להכיר" בינו לבין המשתמש הזה, בשביל לאפשר להתחיל איתו צאט.
   1.8. new_chat – על מנת ליצור צאט חדש (אחד על אחד).
   1.9. new_group – על מנת ליצור קבוצה חדשה.
   1.10. add_user_to_group – על מנת להוסיף משתמש לקבוצה מסוימת.

1.11. remove_user_to_group – על מנת להסיר משתמש מסוים מקבוצה מסוימת.
1.12. make_call – על מנת להתחיל שיחה עם צאט מסוים (שיחה קולית).
1.13. upload_profile_picture – על מנת להעלות תמונת פרופיל חדשה.
1.14. upload_group_picture – על מנת לשנות את התמונה של קבוצה מסוימת.
1.15. delete_message_for_me – על מנת למחוק הודעה מסוימת עבורי.
1.16. delete_message_for_everyone – על מנת למחוק הודעה מסוימת עבור כולם.
1.17. mark_as_seen – על מנת להודיע לשרת באיזה צאט המשתמש נמצא כדי שיוכל לסמן שכל ההודעות בצאט *הזה נקראו על ידי המשתמש הזה.*

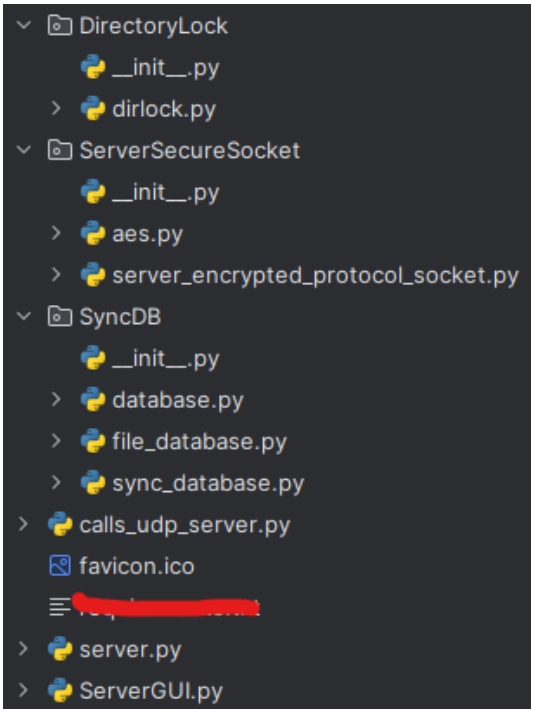
2. ChatEaseGUI.py – *מכיל בעיקר פונקציות שעוטפות את כל הפונקציות בcommunication כדי לחשוף אותן לjs באמצעות eel. בנוסף מכיל כמה פונקציות שחשופות לjs כדי לקבל מידע על צאט/משתמשים מוכרים ועוד פונקציה שנמצאת בלולאה של סנכרון.*

3. calls_udp_client.py - מכיל 3 פונקציות עיקריות ועוד פונקציה אחת שמתחילה את כולן, 2 פונקציות לשליחה וקבלה של חבילות UDP של אודיו ועוד פונקציה לשמירה על קשר TCP בנוסף המאפשר הזדהות וזיהוי בקלות יותר אם הלקוח עדיין מחובר (ניתן גם בלי).

4. photo_tools.py – *עבור בדיקת גודל תמונה (שיהיה לפחות 64x64) יש גם פונקציה שהופכת תמונה לעגולה אך היא לא בשימוש יותר.*

****** אני רוצה לציין שגם כל הפונקציות וגם המשתנים בקוד מתועדים אז לא נכנסתי ממש לעומק כאן אלא נתתי פירוט כללי של איזה פונקציות קיימות והשמות שלהן ברורים ******

5.2 צד שרת



המודול DirectoryLock

1. __init__.py
2. dirlock.py - מכיל 2 פונקציות – block, unblock אשר יוצר תיקייה על מנת לחסום משאב ומסיר את התיקייה על מנת לפנות את המשאב, מה שיקרה בעצם אם 2 ינסו לעשות block זה שאחד מהם יקבל exception שהתיקייה קיימת כבר וכך ידע שהמשאב חסום.

המודול ServerSecureSocket

1. __init__.py
2. aes.py – כמו בלקוח 5.1 צד לקוח.
3. server_encrypted_protocol_socket.py – גם כמו בלקוח רק הexchange של המפתח AESשונה ויש פונקציה של accept ואין של connect וכו' הבדלים כרגיל בין סוקט של שרת ושל לקוח.

המודול SyncDB

1. __init__.py
2. database.py – מכיל מחלקה אחת, הכי בסיסית, בסך הכל עוטפת את המילון של python.
3. file_database.py – מכיל מחלקה אחת אשר יורשת מהמחלקה בdatabse.py ומוסיפה כתיבה לקובץ וקריאה מקובץ על מנת לאפשר סנכרון בין threads ובין processes.
4. sync_database.py – מכיל מחלקה אשר מקבלת עצם של FileDatabase אשר נמצאת בקובץ file_database.py ומאפשרת set וגם get כמו על dictionary זאת אומרת בעזרת [] אך שימו לב מאחר וזה משותף בין threads ובין processes אם משנים ערך בתוך המילון צריך לעשות עליו set ולא משהו כמו:
database["5"].append(5)

מאחר וזה לא יגרום לטריגר של כתיבה לקובץ, במקום זאת צריך לעשות:
some_list = database["5"] ואז צריך
some_list.append(5) ואז
database["5"] = some_list

מחוץ למודולים - Other Files

calls_udp_server.py - מכיל 3 פונקציות עיקריות ועוד 2, אחת שעוטפת את זאת שמתחילה את 3 הפונקציות העיקריות ואחת שמתחילה את שלושת הפונקציות העיקריות. הפונקציות העיקריות – קבלה של חבילות מכל הלקוחות, הפצה של כל חבילה לכל הלקוחות חוץ מזה ששלך את החבילה, פונקציה לשמירה על חיבור TCP בנוסף לUDP על מנת הזדהות וזיהוי קל יותר של ניתוק.

server.py – מכיל את כל הפונקציות לטיפול בכל הבקשות של הלקוח ופונקציה לטיפול בconnection עם הלקוח – handle_client, עוד פונקציה לקבלת לקוחות ובשביל להתחיל את השרת – main ועוד כמה פונקציות לשימוש כללי של השרת. אני לא מפרט פה מאחר וכל הפונקציות מתועדות גם עם תיעוד במילים של מה הן עושות וגם type hints, כך שאין צורך לפרט עוד פעם.

ServerGUI.py – GUI נחמד לסרבר אשר מציג את כל הoutput ואת כל הerrors, בנוסף מציג את כל הלקוחות המחוברים ואת כל הIPs החסומים, מאפשר להתחיל ולעצור את השרת ובנוסף לעשות reload לקובץ של השרת זה היה בשביל שלב הפיתוח ככה נוח אם משנים משהו בקובץ של השרת פשוט ללחוץ על הכפתור וזה עוצר את השרת ומתחיל מחדש עם השינויים. מאפשר גם להסיר IP חסום מהרשימה.

****** אני רוצה לציין שגם כל הפונקציות וגם המשתנים בקוד מתועדים אז לא נכנסתי ממש לעומק כאן אלא נתתי פירוט כללי של איזה פונקציות קיימות והשמות שלהן ברורים ומתארים את מה שהן עושות ******

5.3 פרוטוקול התקשורת בין השרת ללקוח

הפרוטוקול לא מסובך, הפרוטוקול להלן:

cmd (padded to length of 30 chars) after the cmd without any separation if there is only one data argument then it will be straight after the cmd if there are multiple data arguments we will put the length of the argument (padded to 15 chars) and then the argument data, the last argument won't have length because it will just be the rest of the message.
after we create a request/response in this format we will send it with the special ServerSecureSocket module or ClientSecureSocket, which will send the length of the request/response (padded to 30 chars) and only then the request/response itself, when receiving we receive 30 chars that will tell us the length of the request/response and then the data itself.

יש גם הודעה של סנכרון שבמקרה הזה פשוט מעבירים מילונית דרך הסוקט (בעזרת pickle, אשר גם נעטפת ב30 תווים שאומרים מה האורך של כל ההודעה) והמילונית מכילה בתור מפתח path של קובץ ובתור ערך את הdata של הקובץ.

6. רפלקציה

מה הקשיים / אתגרים שעמדו בפניו

התקשורת בין השרת ללקוח מאובטחת.
בהתחלה עשיתי את הפרויקט עם סוקט של ssl ויש אצלהם בעיה שמעבירים הרבה מידע בפעם אחת זה פשוט נתקע אז הייתי צריך לממש את זה בעצמי, זה החלק של הaes.py ושל הclient_encrypted_protocol_socket.py ושל הsever_encrypted_protocol_socket.py, אשר מחליפים מפתח חיצוני ומעבירים את המפתח לAES להמשך התקשורת בצורה מוצפנת (עם RSA עם המפתח הציבורי של השרת) ואז כל התקשורת ממשיכה עם הצפנת AES, למדתי ככה עוד קצת על הצפנה.

בנוסף כל הקטע של השיחות (במיוחד השיחות וועידה) זה היה משהו שלא עשיתי אף פעם וידעתי שזה לא יהיה הכי פשוט אבל כשהתחלתי ממש את השיחות ראיתי שצריך למצוא דרך לעשות את השרת ממש מהיר ויעיל אחרת יהיה תקיעות. בגרסה הראשונה של השיחות שעשיתי השרת לא היה מספיק מהיר (עבור כמה משתמשים – שיחות וועידה) וראיתי שמה שקורה זה שבכמה שניות הראשונות זה עובד חלק ואחרי זה מתחיל להצטבר חבילות אצל השרת שהוא לא מספיק להפיץ לכל הלקוחות ואז מתחיל להיתקע.

מה הוא היה עושה אחרת לו היה מתחיל היום

אם הייתי מתחיל את הפרויקט מחדש היום עם כל הידע הייתי הולך ישר על html ועל css וגם לא הייתי משתמש בeel כדי לאפשר לגשת לזה דרך דפדפן בלי python אך בגלל שכשעברתי לhtml וlcss וjs השרת כבר היה מוכן והוא לא עובד בhttp אלא בפרוטוקול שאני עשיתי חיפשתי דרך לעבוד עם דרך הpython עם html css js כדי שלא אצטרך לשנות את כל השרת כי לא ידעתי אם יהיה לי זמן גם לזה.

מה אם היה קורה אחרת העבודה הייתה יעילה יותר עבורו

עשיתי גם ניתוח להסרת משקפיים לקראת הסוף של העבודה על הפרויקט וזה השבית אותי לשבוע שהיה קריטי אבל הסתדרתי.
בנוסף אם הייתי מחליט מראש ללכת על html css זה היה חוסך לי הרבה עבודה כי החלפתי לזה רק אחרי שכבר סיימתי את הGUI של המשתמשים וזה היה הרבה עבודה שבסוף הלכה לפח כי עברתי מtkinter לhtml וcss.

מה אתם מרגישים שהעבודה על הפרויקט נתנה לכם?
אני מרגיש שהבנתי יותר את כל הקטע של הצפנה.
והכי חשוב שלמדתי עוד 3 שפות תכנות שלא ידעתי לפני HTML, CSS, JS.
מצאתי בעצמי פתרונות לעוד כל מיני דברים כמו הנעילה של הצאטים ואיך ליעל את השרת של השיחות.
למדתי קצת על אודיו בpython.

7. ביבליוגרפיה

1. https://stackoverflow.com/questions/36894315/how-to-select-a-specific-input-device-with-pyaudio
עבור בחירת מיקרופון בשיחות, לא בטוח שאספיק לממש לזה GUI אז לא בטוח שזה יהיה בשימוש אבל זה כבר נמצא בקוד בתור comment.

2. https://docs.python.org/3/library/tkinter.messagebox.html#module-tkinter.messagebox
עבור הצגת הודעות שגיאה בחלונית קטנה.

3. https://stackoverflow.com/questions/51396841/how-to-change-a-python-thread-name-from-inside-the-thread-on-windows
עבור שינוי השם של הthread של הלקוחות שמתחברים לserver לאחר ביצוע login.

4. https://github.com/python-eel/Eel/issues/395
הייתה לי בעיה כשאר השתמשתי עם הeel וניסיתי לפתוח file dialog ומצאתי את הבעיה הזאת שמישהו העלה והפתרון שם פתר גם לי את הבעיה.

היו עוד כל מיני דברים שנעזרתי בהם אבל לא שמרתי את הקישורים להכל.

8. נספחים

8.0 קישור לrepository של הGithub של הפרויקט

אני מוסיף פה קישור לrepository של הGitHub של הפרויקט אשר יכיל את הקוד המעודכן תמיד (מאחר ואני אמשיך לעבוד גם אחרי הגשת ספר הפרויקט עד הבגרות).
https://github.com/Omer-Dagry/ChatEase

8.1 צד שרת

DirectoryLock

__init__.py

```
from .dirlock import block, unblock
```

dirlock.py

```
"""
##############################################
Author: Omer Dagry
Mail: omerdagry@gmail.com
Date: 30/05/2023 (dd/mm/yyyy)
##############################################
"""

import os
import time
import shutil


def block(path: str) -> bool:
    """ :return: True to signal the "lock" is acquired """
    """
        create a folder "folder_name" to block another thread
from touching a
        specific resource file of the user/chat because we are
reading the file
        and then writing back and if someone does it at the
same time with us
        something will be lost, so blocking like this allows
to block only for
        this specific chat and not block all the clients
threads, which is what we want
    """
    while True:
        try:
            os.makedirs(path, exist_ok=False)
            break
        except OSError:  # locked
            time.sleep(0.0005)
```

```
    return True


def unblock(path: str) -> bool:
    """ :return: False to signal the "lock" isn't acquired """
    if not os.path.isdir(path):
        raise ValueError(f"The 'lock' is already unlocked.
(path - '{path}')")
    shutil.rmtree(path)
    return False
```

ServerSecureSocket

__init__.py

```
from .server_encrypted_protocol_socket import
ServerEncryptedProtocolSocket
```

aes.py

```
import hashlib

from Crypto import Random
from Crypto.Cipher import AES


class AESCipher:
    """ a class to wrap the AES encryption and decryption """
    def __init__(self, key: str | bytes):
        self.bs = AES.block_size
        key = key.encode() if isinstance(key, str) else key
        self.key = hashlib.sha256(key).digest()

    def encrypt(self, raw: bytes) -> bytes:
        raw = self._pad(raw)
        iv = Random.new().read(AES.block_size)
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        # return base64.b64encode(iv + cipher.encrypt(raw))
        return iv + cipher.encrypt(raw)

    def decrypt(self, enc: bytes) -> bytes:
        # enc = base64.b64decode(enc)
        iv = enc[:AES.block_size]
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        return
self._unpad(cipher.decrypt(enc[AES.block_size:]))

    def _pad(self, s: bytes) -> bytes:
        return s + ((self.bs - len(s) % self.bs) * chr(self.bs
- len(s) % self.bs)).encode()
```

```
    @staticmethod
    def _unpad(s: bytes) -> bytes:
        return s[:-s[-1]]
```

server_encrypted_protocol_socket.py

```
"""
##############################################
Author: Omer Dagry
Mail: omerdagry@gmail.com
Date: 30/05/2023 (dd/mm/yyyy)
##############################################
"""

from __future__ import annotations

import rsa
import socket

from .aes import AESCipher


class AESKeyMissing(Exception):
    """ Raised when there is an attempt to send or receive
data and '__exchange_aes_key' wasn't called yet """


class ServerEncryptedProtocolSocket:
    """ a wrapped socket with encryption and special send &
recv """
    def __init__(self, my_public_key: rsa.PublicKey,
my_private_key: rsa.PrivateKey,
                 family: socket.AddressFamily | int = None,
type: socket.SocketKind | int = None,
                 proto: int = None, fileno: int | None = None,
sock: socket.socket = None):
        self.__my_public_key: rsa.PublicKey = my_public_key
        self.__my_private_key: rsa.PrivateKey = my_private_key
        #
        self.__aes_key: None | bytes = None
        self.__aes_cipher: None | AESCipher = None
        if sock is None:
            kwargs = {"family": family, "type": type, "proto":
proto, "fileno": fileno}
            kwargs = {key_word: arg for key_word, arg in
kwargs.items() if arg is not None}
```

```
            self.__sock = socket.socket(**kwargs)
        else:
            self.__sock = sock
            self.__exchange_aes_key()

    # Public:

    def recv_message(self, timeout: int = None) -> bytes:
        """ receive 1 full message """
        if self.__aes_cipher is None:
            raise AESKeyMissing("aes_cipher is None, please
call connect before calling recv_message")
        current_timeout = self.__sock.timeout
        self.settimeout(timeout)
        data_length = b""
        while len(data_length) != 30:
            try:
                res = self.__recvall(30 - len(data_length))
                data_length += res
                if res == b"":  # connection closed
                    return res
            except socket.timeout:
                if data_length == b"":
                    return b""
        data_length = int(data_length.decode().strip())
        data = b""
        while len(data) != data_length:
            try:
                res = self.__recvall(data_length - len(data))
                data += res
                if res == b"":  # connection closed
                    return res
            except socket.timeout:
                if data_length == b"":
                    return b""
        self.settimeout(current_timeout)
        return self.__aes_cipher.decrypt(data)

    def send_message(self, data: bytes) -> bool:
        """ send 1 message """
        if self.__aes_cipher is None:
            raise AESKeyMissing("aes_cipher is None, please
call connect before calling send_message")
        try:
            data = self.__aes_cipher.encrypt(data)

self.__sock.sendall(f"{len(data)}".ljust(30).encode())
```

```
            self.__sock.sendall(data)
        except ConnectionError:
            return False
        return True

    def bind(self, __address: tuple[str, int]) -> None:
        return self.__sock.bind(__address)

    def listen(self, __backlog: int = None) -> None:
        args = () if __backlog is None else (__backlog,)
        return self.__sock.listen(*args)

    def accept(self) -> tuple[ServerEncryptedProtocolSocket,
tuple[str, int]]:
        client_sock, client_addr = self.__sock.accept()
        return
ServerEncryptedProtocolSocket(self.__my_public_key,
self.__my_private_key, sock=client_sock), client_addr

    def settimeout(self, __value: float | None) -> None:
        return self.__sock.settimeout(__value)

    def get_timeout(self) -> float | None:
        return self.__sock.timeout

    def getpeername(self) -> tuple[str, int]:
        return self.__sock.getpeername()

    def close(self):
        self.__sock.close()

    # Private:

    def __recvall(self, buffsize: int) -> bytes:
        data = b""
        while len(data) < buffsize:
            res = self.__sock.recv(buffsize - len(data))
            data += res
            if res == b"":  # connection closed
                return res
        return data

    # Exchange the random aes key using server public key
    def __exchange_aes_key(self) -> None:
        """ send the connection this server public key and
then receive the AES encryption key """
        my_public_key_bytes =
```

```
self.__my_public_key.save_pkcs1("PEM")

self.__sock.sendall(f"{len(my_public_key_bytes)}".ljust(30).en
code() + my_public_key_bytes)
        aes_key_len = int(self.__recvall(30).decode().strip())
        aes_key_encrypted = self.__recvall(aes_key_len)
        self.__aes_key = rsa.decrypt(aes_key_encrypted,
self.__my_private_key)
        self.__aes_cipher = AESCipher(self.__aes_key)
```

SyncDB

__init__.py

```
from .sync_database import SyncDatabase
from .file_database import FileDatabase
```

database.py

```
"""
##############################################
Author: Omer Dagry
Mail: omerdagry@gmail.com
Date: 30/05/2023 (dd/mm/yyyy)
##############################################
"""

from typing import *


class Database:
    __slots__ = ("__database", "__dict__")

    def __init__(self):
        self.__database = {}

    def set_database(self, dic: dict) -> bool:
        """ set self.__database """
        self.__database = dic
        return True

    def get_database(self) -> dict:
        """ get self.__database """
        return self.__database

    def __setitem__(self, key: Hashable, val: Any):
        """ add key: val """
        self.__database[key] = val

    def safe_set(self, key: Hashable, val: Any) -> bool:
```

```
        """ add key: val, only if key is not already in
database """
        if key in self.__database:
            return False
        self.__database[key] = val
        return True

    def __getitem__(self, key: Hashable) -> Any:
        """ get the val of key """
        if key in self.__database.keys():
            val = self.__database[key]
            return val
        else:
            raise KeyError(f"{key} isn't a key in the
__database.")

    def pop(self, key: Hashable) -> Any:
        """ remove key and self.__database[key] value """
        if key in self.__database.keys():
            return self.__database.pop(key)
        else:
            raise KeyError(f"{key} isn't a key in the
__database.")

    def get(self, key: Hashable) -> Any | None:
        """ get a value, if it doesn't exist, return None."""
        return self.__database.get(key)

    def __contains__(self, key: Hashable) -> bool:
        """ return True if key exists in __database else False
"""
        return key in self.__database


_ = Database()
_["hello"] = 5  # check set_value
assert _["hello"] == 5  # check get_value
assert _.pop("hello") == 5  # check pop_value
_.set_database({"hi": 6, "bye": 5})  # check set_database
assert _.get_database() == {"hi": 6, "bye": 5}  # check
get_database
del _
```

file_database.py

```
"""
##############################################
Author: Omer Dagry
Mail: omerdagry@gmail.com
```

```
Date: 30/05/2023 (dd/mm/yyyy)
##############################################
"""

import os
import pickle

from typing import *
from .database import Database


class FileDatabase(Database):
    __slots__ = ("__database_file_name",)

    def __init__(self, database_file_name: str,
ignore_existing: bool = False, clear_database: bool = False):
        super().__init__()
        if os.path.isfile(database_file_name) and not
ignore_existing:
            raise ValueError(f"The File '{database_file_name}'
Already Exists.")
        self.__database_file_name = database_file_name
        with open(self.__database_file_name, "wb" if
clear_database else "ab") as db:  # create the database file
            if clear_database:  # clear database if
clear_database
                db.write(b"")

    def write_database(self, read_after: bool = True):
        """ Write `self.__database` to a file using pickle """
        with open(self.__database_file_name, "wb") as
database_file:
            data = pickle.dumps(super().get_database())
            database_file.write(data)
        if read_after:
            self.read_database()

    def read_database(self):
        """ Read __database file with pickle and set
`self.__database` """
        with open(self.__database_file_name, "rb") as
database_file:
            try:
                dic = pickle.load(database_file)
            except EOFError:  # file is empty
                dic = {}
        ok = super().set_database(dic)
```

```
        while not ok:
            ok = super().set_database(dic)

    def set_database(self, dic: dict) -> bool:
        """ Override & write the __database after setting it
"""
        ok = super().set_database(dic)
        if ok:
            self.write_database()
        return ok

    def get_database(self) -> dict:
        """ Override & read from __database before returning
the __database"""
        self.read_database()
        return super().get_database()

    def __setitem__(self, key: Hashable, val: Any) -> bool:
        """ Override & add read & write __database with pickle
"""
        self.read_database()
        ok = super().__setitem__(key, val)
        self.write_database()
        return ok

    def safe_set(self, key: Hashable, val: Any) -> bool:
        """ Override & add read & write __database with pickle
"""
        self.read_database()
        ok = super().safe_set(key, val)
        self.write_database()
        return ok

    def __getitem__(self, key: Hashable) -> Any:
        """ Override & add read __database with pickle """
        self.read_database()
        return super().__getitem__(key)

    def pop(self, key: Hashable) -> Any:
        """ Override & add read & write __database with pickle
"""
        self.read_database()
        val = super().pop(key)
        self.write_database()
        return val

    def get(self, key: Hashable) -> Any | None:
```

```
        """ get a value, if it doesn't exist, return None."""
        self.read_database()
        return super().get(key)

    def __contains__(self, key: Hashable) -> bool:
        """ return True if key exists in __database else False
"""
        self.read_database()
        return super().__contains__(key)


# because each process has its own memory, this file will be
imported
# by x processes, so the file name of the __database can't be
the same
# for all the processes, or they will interfere with each
other in the
# asserts because every action affects the file of the
__database and there
# are x number of processes that will import this file
simultaneously
file_name = f"####test####{os.getpid()}"
try:
    _ = FileDatabase(database_file_name=file_name)
    _["hello"] = 5  # check set_value & write & read
    assert _["hello"] == 5  # check get_value & read
    assert _.pop("hello") == 5  # check pop_value & write &
read
    _.set_database({"hi": 6, "bye": 5})  # check set_database
& write
    assert _.get_database() == {"hi": 6, "bye": 5}  # check
get_database & read
    del _
    # check final result
    with open(file_name, "rb") as test_file:
        _ = pickle.load(test_file)
    assert _ == {"hi": 6, "bye": 5}
    del _
except BaseException as exception:
    raise exception
finally:
    os.remove(file_name)
    del file_name
```

sync_database.py

```
"""
##############################################
Author: Omer Dagry
```

```
Mail: omerdagry@gmail.com
Date: 30/05/2023 (dd/mm/yyyy)
##############################################
"""

import os
import pickle
import threading
import multiprocessing

from typing import *
from .file_database import FileDatabase


class SyncDatabase:
    __slots__ = ("__database", "__mode", "__max_reads_together",
"__edit_lock", "__semaphore", "__dict__")

    def __init__(self, database: FileDatabase, mode: bool,
max_reads_together: int = 10):
        """ mode: True -> multiprocessing, False -> threading
"""
        #
        # check that database is FileDatabase
        if not isinstance(database, FileDatabase):
            raise ValueError("the arg '__database' is an
instance of FileDatabase.")
        # check that the mode is valid
        if not isinstance(mode, bool):
            raise ValueError("the arg 'mode' can be True or
False, False for threads, True for processes.")
        # set params
        self.__database = database
        self.__mode = mode
        self.__max_reads_together = max_reads_together
        # set semaphores according to mode
        if self.__mode:
            self.__edit_lock = multiprocessing.Lock()
            self.__semaphore =
multiprocessing.Semaphore(self.__max_reads_together)
        else:
            self.__edit_lock = threading.Lock()
            self.__semaphore =
threading.Semaphore(self.__max_reads_together)

    def __acquire_all(self):
        # pickup edit lock
```

```
        self.__edit_lock.acquire()
        # acquire all the semaphores
        for _ in range(self.__max_reads_together):
            self.__semaphore.acquire()

    def __release_all(self):
        # release all the semaphores
        for _ in range(self.__max_reads_together):
            self.__semaphore.release()
        # release the edit lock
        self.__edit_lock.release()

    def keys(self):
        """ get all keys """
        self.__semaphore.acquire()
        keys = self.__database.get_database().keys()
        self.__semaphore.release()
        return keys

    def values(self):
        """ get all values """
        self.__semaphore.acquire()
        values = self.__database.get_database().values()
        self.__semaphore.release()
        return values

    def items(self):
        """ get all keys and values """
        self.__semaphore.acquire()
        items = self.__database.get_database().items()
        self.__semaphore.release()
        return items

    def __setitem__(self, key: Hashable, val: Any) -> bool:
        """ set a value """
        self.__acquire_all()
        ok = self.__database[key] = val
        self.__release_all()
        return ok

    def safe_set(self, key: Hashable, val: Any) -> bool:
        """ add key: val, only if key is not already in database
"""
        self.__acquire_all()
        if key in self.__database:
            result = False
        else:
```

```
            result = True
            self.__database[key] = val
        self.__release_all()
        return result

    def add(self, key: Hashable, val: Hashable) -> None:
        """
        use either this function if a set is the container of
the values or use 'append' if a list is

            adds val to the set

            key: {current_values, ...} -> key: {current_values,
..., val}
        """
        self.__acquire_all()
        current_val: set = self.__database[key]
        current_val.add(val)
        self.__database[key] = current_val
        self.__release_all()

    def update(self, key: Hashable, val: Iterable[Hashable]) ->
None:
        """
        use either this function if a set is the container of
the values or use 'extend' if a list is or 'db[] = ' if
        it's 1 value

            extends the set with val

            key: {current_values, ...} -> key: {current_values,
..., *val}
        """
        self.__acquire_all()
        current_val: set = self.__database[key]
        current_val.update(val)
        self.__database[key] = current_val
        self.__release_all()

    def remove_set(self, key: Hashable, val: Hashable) -> bool:
        """
        use either this function if a set is the container of
the values or use 'remove_list' if a set is or 'pop' if
        it's 1 value

            removes val from the set
```

```
            key: {current_values, ..., val} -> key:
{current_values, ...}
        """
        self.__acquire_all()
        current_val: set = self.__database[key]
        if val in current_val:
            result = True
            current_val.remove(val)
        else:
            result = False
        self.__database[key] = current_val
        self.__release_all()
        return result

    def append(self, key: Hashable, val: Any) -> None:
        """
        use either this function if a list is the container of
the values or use 'add' if a set is or 'db[] = ' if
        it's 1 value

            appends val to the list

            key: [current_values, ...] -> key: [current_values,
..., val]
        """
        self.__acquire_all()
        current_val: list = self.__database[key]
        current_val.append(val)
        self.__database[key] = current_val
        self.__release_all()

    def extend(self, key: Hashable, val: Iterable[Any]) -> None:
        """
        use either this function if you want a list as the
container of the values or use 'update' if you want a set

            extends the list of current_val with val

            key: [current_values, ...] -> key: [current_values,
..., *val]
        """
        self.__acquire_all()
        current_val = self.__database[key]
        current_val.extend(val)
        self.__database[key] = current_val
        self.__release_all()
```

```
    def remove_list(self, key: Hashable, val: Any) -> bool:
        """
        use either this function if a list is the container of
the values or use 'remove_set' if a set is or 'pop' if
        it's 1 value

            removes val from the list of current_val

            key: [current_values, ..., val] -> key:
[current_values, ...]

            :return: True if val was in the list of values and
was removed else False
        """
        self.__acquire_all()
        current_val = self.__database[key]
        if not isinstance(current_val, list):
            current_val = [current_val]
        if val not in current_val:
            result = False
        else:
            result = True
            current_val.remove(val)
            self.__database[key] = current_val
        self.__release_all()
        return result

    def __getitem__(self, key: Hashable) -> Any:
        """ get a value """
        self.__semaphore.acquire()
        try:
            val = self.__database[key]
        # make sure that the lock is released even if KeyError
raised
        except KeyError:
            self.__semaphore.release()
            raise
        # release the lock
        self.__semaphore.release()
        return val

    def pop(self, key: Hashable) -> Any:
        """ remove a value """
        self.__acquire_all()
        try:
            val = self.__database.pop(key)
        # make sure that the lock is released even if KeyError
```

```
raised
        except KeyError:
            self.__edit_lock.release()
            # release all the semaphore
            for _ in range(self.__max_reads_together):
                self.__semaphore.release()
            raise
        self.__release_all()
        return val

    def get(self, key: Hashable) -> Any | None:
        """ get a value, if it doesn't exist, return None."""
        self.__semaphore.acquire()
        result = self.__database.get(key)
        self.__semaphore.release()
        return result

    def __contains__(self, key: Hashable) -> bool:
        self.__semaphore.acquire()
        result = key in self.__database
        self.__semaphore.release()
        return result


# because each process has its own memory, this file will be
imported
# by x processes, so the file name of the __database can't be
the same
# for all the processes, or they will interfere with each other
in the
# asserts because every action affects the file of the
__database and there
# are x number of processes that will import this file
simultaneously
file_name = f"####test####{os.getpid()}"
__ = FileDatabase(file_name)
try:
    _ = SyncDatabase(__, True)
    _["hello"] = 5  # check set_value & write & read
    assert _["hello"] == 5  # check get_value & read
    assert _.pop("hello") == 5  # check pop_value & write & read
    _["hi"] = 6
    _["bye"] = 5
    assert _["hi"] == 6 and _["bye"] == 5
    del _
    # check final result
    with open(file_name, "rb") as test_file:
```

```
        _ = pickle.load(test_file)
    assert _ == {"hi": 6, "bye": 5}
    del _, __
except BaseException as exception:
    raise exception
finally:
    os.remove(file_name)
    del file_name
```

Other Files

calls_udp_server.py

```
"""
##############################################
Author: Omer Dagry
Mail: omerdagry@gmail.com
Date: 30/05/2023 (dd/mm/yyyy)
##############################################
"""

import sys
import time
import pickle
import socket
import logging
import hashlib
import traceback
import multiprocessing
import concurrent.futures

from multiprocessing.managers import DictProxy, SyncManager
from ServerSecureSocket import ServerEncryptedProtocolSocket

# Constants
# logging
LOG_DIR = 'log'
LOG_LEVEL = logging.DEBUG
LOG_FILE = LOG_DIR + "/ChatEase-Calls-Server.log"
LOG_FORMAT = "%(levelname)s | %(asctime)s | %(processName)s |
%(message)s"
# Others
CHUNK = 1024 * 8
BUFFER_SIZE = CHUNK * 4


def broadcast_audio(server_socket: socket.socket, data: bytes,
sent_from: tuple[str, int], clients: DictProxy):
```

```
    """ broadcast audio stream to all connected clients """
    try:
        time_ = time.perf_counter()
        for addr in clients.keys():  # type: tuple[str, int]
            try:
                # if this ip:port didn't send data for more than
5 seconds stop sending to him
                if (time_ - clients[addr]) > 5:
                    continue
                if addr != sent_from:
                    server_socket.sendto(data, addr)
            except Exception as e:
                # print(f"closed {addr}, ({str(e)})")
                pass
    except KeyboardInterrupt:
        pass


def receive_audio_and_broadcast(server_socket: socket.socket,
clients_ips: DictProxy, clients: DictProxy):
    """ receive audio from clients and submit work for the
broadcast func """
    try:
        with
concurrent.futures.ThreadPoolExecutor(max_workers=10) as x:
            while True:
                try:
                    data, sent_from =
server_socket.recvfrom(BUFFER_SIZE)
                    if sent_from in clients or sent_from[0] in
clients_ips:
                        # if addr not in clients: print(f'new
connection from {addr}')
                        clients[sent_from] = time.perf_counter()
                        x.submit(broadcast_audio, server_socket,
data, sent_from, clients)
                except BlockingIOError:
                    time.sleep(0.005)
                except (ConnectionError, socket.error,
TimeoutError):
                    pass
    except KeyboardInterrupt:
        pass


def accept(server_socket: ServerEncryptedProtocolSocket) \
        -> tuple[ServerEncryptedProtocolSocket, tuple[str, int]]
```

```
| tuple[None, None]:
    """ accept a client with a timeout of 1 sec """
    server_socket.settimeout(1)
    try:
        client, addr = server_socket.accept()
    except socket.timeout:
        return None, None
    return client, addr


def disconnect_client(client_sock:
ServerEncryptedProtocolSocket, addr: tuple[str, int],
clients_ips: DictProxy,
                      clients_socket_addr: dict,
client_sock_last_checkin: dict, port: int) -> None:
    """ close connection with client & remove from all databases
"""
    if (time.perf_counter() -
client_sock_last_checkin[client_sock]) > 12:
        if len(clients_ips[addr[0]]) == 1:
            clients_ips.pop(addr[0])
        else:
            ports: list = clients_ips[addr[0]]
            ports.remove(addr[1])
            clients_ips[addr[0]] = ports
        clients_socket_addr.pop(client_sock)
        client_sock_last_checkin.pop(client_sock)
        print(f"Calls server on {port = } - %s:%d Disconnected."
% addr)
        logging.info(f"Calls server on {port = } - %s:%d
Disconnected." % addr)


def handle_tcp_connections(tcp_server_socket:
ServerEncryptedProtocolSocket, clients_ips: DictProxy,
                           clients_passwords: dict[str, str],
port: int) -> None:
    """ handle TCP connections with clients """
    try:
        clients_socket_addr: dict[ServerEncryptedProtocolSocket,
tuple[str, int]] = {}
        client_sock_last_checkin:
dict[ServerEncryptedProtocolSocket, float] = {}
        last_msg = time.perf_counter()
        one_client: None | float = None
        while (time.perf_counter() - last_msg) < 20 and
(one_client is None or (time.perf_counter() - one_client) < 15):
```

```
            try:
                client, addr = accept(tcp_server_socket)
                if client is not None and addr is not None:
                    client: ServerEncryptedProtocolSocket
                    addr: tuple[str, int]
                    print(f"Calls server on {port = } - New
Connection From %s:%d" % addr)
                    logging.info(f"Calls server on {port = } -
New Connection From %s:%d" % addr)
                    # check password
                    client.settimeout(0.05)
                    try:
                        username, password =
pickle.loads(client.recv_message())  # type: str, str
                    except pickle.PickleError:
                        username = ""
                        password = ""
                    if username in clients_passwords and \
                            clients_passwords[username] ==
hashlib.md5(password.encode()).hexdigest().lower():
                        client.send_message(b"ok    ")
                        print(f"Calls server on {port = } -
%s:%d Connected as '{username}'." % addr)
                        logging.info(f"Calls server on {port = }
- %s:%d Connected as '{username}'." % addr)
                        # allow receiving UDP messages from this
ip
                        if addr[0] in clients_ips:
                            clients_ips[addr[0]] =
clients_ips[addr[0]] + [addr[1]]
                        else:
                            clients_ips[addr[0]] = [addr[1]]
                        clients_socket_addr[client] = addr
                        client_sock_last_checkin[client] =
time.perf_counter()
                    else:
                        logging.info(f"Calls server on {port = }
- %s:%d sent wrong username or password." % addr)
                        client.send_message(b"not ok")
                        client.close()
            except Exception as err:
                traceback.format_exception(err)
                #
            if one_client is None and len(clients_socket_addr)
== 1:
                one_client = time.perf_counter()
            elif len(clients_socket_addr) != 1:
```

```
                one_client = None
            for client, addr in
list(clients_socket_addr.items()):
                try:
                    msg = client.recv_message()
                    if msg == b"hi":
                        client_sock_last_checkin[client] =
time.perf_counter()
                    else:
                        disconnect_client(
                            client, addr, clients_ips,
clients_socket_addr, client_sock_last_checkin, port)
                    last_msg = time.perf_counter()
                except (socket.timeout, ConnectionError,
socket.error):
                    disconnect_client(client, addr, clients_ips,
clients_socket_addr, client_sock_last_checkin, port)
    except KeyboardInterrupt:
        pass


def main(tcp_server_sock: ServerEncryptedProtocolSocket, port:
int, clients_passwords: dict[str, str],
         clients_ips: DictProxy, clients: DictProxy):
    """ start all the processes and call handle_tcp_connections

    :param tcp_server_sock: the TCP socket of the server
    :param port: the port of the server
    :param clients_passwords: dictionary of username and
password (hashed twice)
    :param clients_ips: a dictionary that can be shared between
processes, will contain all
                        the ips of the verified TCP connections
    :param clients: a dictionary that can be shared between
processes, will contain all the
                    clients ip:port and the time of last msg
    """
    # server UDP socket
    server_socket = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
    server_socket.bind(("0.0.0.0", port))
    server_socket.settimeout(0.05)

    # the receiving process (also calls broadcast), 2 sounds
better
    receive_process = multiprocessing.Process(
        target=receive_audio_and_broadcast, args=(server_socket,
```

```
clients_ips, clients)
    )
    receive_process.start()
    receive_process2 = multiprocessing.Process(
        target=receive_audio_and_broadcast, args=(server_socket,
clients_ips, clients)
    )
    receive_process2.start()

    try:
        # the tcp connections to the clients
        handle_tcp_connections(tcp_server_sock, clients_ips,
clients_passwords, port)
    finally:
        receive_process.kill()
        receive_process2.kill()


def start_call_server(tcp_server_sock:
ServerEncryptedProtocolSocket,
                      port: int, clients_passwords: dict[str,
str], print_queue: multiprocessing.Queue):
    """ call this to start the server """
    if print_queue is not None:
        class STDRedirect:
            def __init__(self, std_type):
                assert std_type == "stdout" or std_type ==
"stderr"
                self.std_type = std_type

            def write(self, data):
                print_queue.put((self.std_type, data))

        sys.stdout = STDRedirect("stdout")
        sys.stderr = STDRedirect("stderr")
    # logging configuration
    logging.basicConfig(format=LOG_FORMAT, filename=LOG_FILE,
level=LOG_LEVEL)
    try:
        print(f"Call server starting on {port = }.")
        logging.info(f"Call server starting on {port = }.")
        with multiprocessing.Manager() as manager:  # type:
SyncManager
            main(tcp_server_sock, port, clients_passwords,
manager.dict(), manager.dict())
    except KeyboardInterrupt:
        pass
```

```
    finally:
        print(f"Call server on {port = } has ended.")
        logging.info(f"Call server on {port = } has ended.")


if __name__ == '__main__':
    s = socket.socket()
    s.bind(("0.0.0.0", 16400))
    s.listen()
    start_call_server(
        s, 16400, {"omer":
hashlib.md5(hashlib.md5("omer".encode()).hexdigest().lower().enc
ode()).hexdigest().lower()},
        print
    )
```

server.py

```
"""
##############################################
Author: Omer Dagry
Mail: omerdagry@gmail.com
Date: 30/05/2023 (dd/mm/yyyy)
##############################################
"""

import os
import ssl
import sys
import rsa
import time
import socket
import pickle
import random
import shutil
import string
import logging
import hashlib
import smtplib
import datetime
import threading
import traceback
import multiprocessing

from typing import *
from email.mime.text import MIMEText
from DirectoryLock import block, unblock
```

```
from SyncDB import SyncDatabase, FileDatabase
from calls_udp_server import start_call_server
from email.mime.multipart import MIMEMultipart
from multiprocessing.managers import DictProxy
from ServerSecureSocket import ServerEncryptedProtocolSocket


# Constants
# logging
LOG_DIR = 'log'
LOG_LEVEL = logging.DEBUG
LOG_FILE = LOG_DIR + "/ChatEase-Server.log"
LOG_FORMAT = "%(levelname)s | %(asctime)s | %(processName)s |
%(message)s"
# Others
# Chat id's possible characters
CHAT_ID_CHARS = [letter for letter in string.ascii_uppercase +
string.ascii_lowercase + string.digits]
random.shuffle(CHAT_ID_CHARS)
# Server email and special app password
SERVER_EMAIL = "project.twelfth.grade@gmail.com"
SERVER_EMAIL_APP_PASSWORD = "hbqbubnlppqxmupy"
# Paths
SERVER_DATA = "Data\\Server_Data\\"
USERS_DATA = "Data\\Users_Data\\"
# IP & Port
IP = "0.0.0.0"
PORT = 8820
SERVER_IP_PORT = (IP, PORT)
# Blocking Clients
BLOCK_TIME = 60 * 5
BLOCK_AFTER_X_EXCEPTIONS = 100
EXCEPTIONS_WINDOW_TIME = 60 * 5

# Create All Needed Directories
os.makedirs(f"{SERVER_DATA}", exist_ok=True)
os.makedirs(f"{USERS_DATA}", exist_ok=True)
os.makedirs(LOG_DIR, exist_ok=True)

# Globals
print_ = print
print_queue = None
# File DBs
# email_password_file_database -> {email: password, another
email: password, ...}
email_password_file_database =
FileDatabase(f"{SERVER_DATA}email_password",
```

```
ignore_existing=True)
# email_user_file_database -> {email: username, another email:
another username, ...}
email_user_file_database =
FileDatabase(f"{SERVER_DATA}email_username",
ignore_existing=True)
# chat_id_users_database -> {chat_id: [email, another_email],
another_chat_id: [email, another_email], ...}
chat_id_users_file_database =
FileDatabase(f"{SERVER_DATA}chat_id_users",
ignore_existing=True)
# user_online_status_database ->
# {email: ["Online", number_of_live_connection], email:
["Offline", last_seen - datetime.datetime], ...}
user_online_status_file_database =
FileDatabase(f"{SERVER_DATA}user_online_status",
ignore_existing=True)
# Sync DBs
# {email (str): password (str)}
email_password_database =
SyncDatabase(email_password_file_database, False,
max_reads_together=1000)
# {email (str), username (str)}
email_user_database = SyncDatabase(email_user_file_database,
False, max_reads_together=1000)
# {chat_id (str): users (set[str])}
chat_id_users_database =
SyncDatabase(chat_id_users_file_database, False,
max_reads_together=1000)
# {email (str): status (list[str, int | datetime.datetime])}
user_online_status_database =
SyncDatabase(user_online_status_file_database, False,
max_reads_together=1000)
# Others
clients_sockets = []
printing_lock = threading.Lock()
sync_sockets_lock = threading.Lock()
sync_sockets: dict[str, set[ServerEncryptedProtocolSocket]] = {}
# {email: [sync_sock, sync_sock, ...], ...}
received_exception_from: dict[str, set[datetime.datetime]] = {}
# {ip: {time of exception (for each exception)}}
blocked_ips: dict[str, datetime.datetime] = {}  # {ip: time of
block}
online_clients: dict[str, None] = {}  # {email: None, email2:
None, ...}
add_exception_lock = threading.Lock()
blocked_client_lock = threading.Lock()
```

```
ongoing_calls: dict[str, multiprocessing.Process] = {}  #
chat_id: the process of the call server
my_public_key: rsa.PublicKey | None = None
my_private_key: rsa.PrivateKey | None = None


def print(*values: object, sep: str | None = " ", end: str |
None = "\n"):
    """ a wrapper around print to ensure the prints won't get
mixed with each other """
    printing_lock.acquire()
    print_(*values, sep=sep, end=end)
    printing_lock.release()


def start_server(my_public_key: rsa.PublicKey, my_private_key:
rsa.PrivateKey) -> ServerEncryptedProtocolSocket:
    """ creates server socket binds it and returns it """
    server_socket = ServerEncryptedProtocolSocket(my_public_key,
my_private_key)
    try:
        server_socket.bind(SERVER_IP_PORT)
        print(f"Server is up !! ({PORT = })")
        logging.info(f"Server is up !! ({PORT = })")
        server_socket.listen()
    except OSError:
        logging.debug(f"The Port {PORT} Is Taken.")
        print(f"The Port {PORT} Is Taken.")
        sys.exit(1)
    return server_socket


def accept_client(server_socket: ServerEncryptedProtocolSocket)
\
        -> tuple[ServerEncryptedProtocolSocket | None,
tuple[str, int] | None]:
    """ except client with a timeout of 2 seconds, if there is
no connection in 2 seconds returns None"""
    global clients_sockets
    server_socket.settimeout(2)
    try:
        client_socket, client_addr = server_socket.accept()
    except (socket.error, ConnectionError):
        return None, None
    clients_sockets.append(client_socket)
    logging.info("[Server]: New Connection From: '%s:%s'" %
(client_addr[0], client_addr[1]))
```

```
    print("[Server]: New Connection From: '%s:%s'" %
(client_addr[0], client_addr[1]))
    return client_socket, client_socket.getpeername()


def write_to_file(file_path: str, mode: str, data: bytes | str)
-> None:
    """ write to file """
    with open(file_path, mode) as f:
        f.write(data)


def read_from_file(file_path: str, mode: str) -> str | bytes:
    """ read a file """
    with open(file_path, mode) as f:
        data: str | bytes = f.read()
    return data


def add_exception_for_ip(ip: str) -> None:
    """ Add the ip to a list of exceptions """
    add_exception_lock.acquire()
    if ip in received_exception_from:
        received_exception_from[ip].add(datetime.datetime.now())
    else:
        received_exception_from[ip] = {datetime.datetime.now()}
    logging.debug(f"[Server]: added the exception that was
received while handling '{ip}' to the list of exceptions")
    add_exception_lock.release()


def watch_exception_dict():
    """
    watch over the list of exceptions received from IPs, if an
IP has more than BLOCK_AFTER_X_EXCEPTIONS
    exception in the last EXCEPTIONS_WINDOW_TIME minutes block
that IP for BLOCK_TIME minutes

    open a thread for this function
    """
    while True:
        current_time = datetime.datetime.now()
        remove_ips = []
        for ip in received_exception_from:
            remove = []
            for ex_time in received_exception_from[ip]:
                if (current_time - ex_time).seconds >
```

```
EXCEPTIONS_WINDOW_TIME:
                    remove.append(ex_time)
            for ex_time in remove:
                received_exception_from[ip].remove(ex_time)
            # if we received more than BLOCK_AFTER_X_EXCEPTIONS
exception in the last EXCEPTIONS_WINDOW_TIME from
            # the same ip, block this ip for BLOCK_TIME
            if len(received_exception_from[ip]) >=
BLOCK_AFTER_X_EXCEPTIONS:
                blocked_client_lock.acquire()
                blocked_ips[ip] = current_time
                blocked_client_lock.release()
                msg = f"[Server]: the IP '{ip}' received more
than {BLOCK_AFTER_X_EXCEPTIONS} exception " \
                      f"in under than {EXCEPTIONS_WINDOW_TIME}
seconds. this IP is blocked for {BLOCK_TIME}"
                print(msg)
                logging.warning(msg)
                remove_ips.append(ip)  # can't change during
iteration
        for ip in remove_ips:
            received_exception_from.pop(ip)
        time.sleep(10)  # check every 10 seconds


def add_chat_id_to_user_chats(user_email: str, chat_id: str) ->
bool:
    """ add chat id to user chats file """
    if user_email not in email_password_database:
        return False
    block(f"{USERS_DATA}{user_email}\\chats block")
    try:
        if not
os.path.isfile(f"{USERS_DATA}{user_email}\\chats"):
            write_to_file(f"{USERS_DATA}{user_email}\\chats",
"wb", b"")
        try:
            chats_list: set =
pickle.loads(read_from_file(f"{USERS_DATA}{user_email}\\chats",
"rb"))
        except EOFError:
            chats_list = set()
        chats_list.add(chat_id)
        write_to_file(f"{USERS_DATA}{user_email}\\chats", "wb",
pickle.dumps(chats_list))
    finally:
        unblock(f"{USERS_DATA}{user_email}\\chats block")
```

```
    sync_new_data_with_client(user_email,
f"{USERS_DATA}{user_email}\\chats")
    return True


def remove_chat_id_from_user_chats(user_email: str, chat_id:
str) -> bool:
    """ remove chat id from user chats file """
    if user_email not in email_password_database:
        return False
    block(f"{USERS_DATA}{user_email}\\chats block")
    try:
        if not
os.path.isfile(f"{USERS_DATA}{user_email}\\chats"):
            write_to_file(f"{USERS_DATA}{user_email}\\chats",
"wb", b"")
        try:
            chats_set: set =
pickle.loads(read_from_file(f"{USERS_DATA}{user_email}\\chats",
"rb"))
        except EOFError:
            chats_set = set()
        if chat_id in chats_set:
            chats_set.remove(chat_id)
        write_to_file(f"{USERS_DATA}{user_email}\\chats", "wb",
pickle.dumps(chats_set))
    finally:
        unblock(f"{USERS_DATA}{user_email}\\chats block")
    sync_new_data_with_client(user_email,
f"{USERS_DATA}{user_email}\\chats")
    return True


def get_user_chats_file(email: str) -> set[str]:
    """ returns all the chat ids of a user """
    block(f"{USERS_DATA}{email}\\chats block")
    try:
        if not os.path.isfile(f"{USERS_DATA}{email}\\chats"):
            write_to_file(f"{USERS_DATA}{email}\\chats", "wb",
b"")
        try:
            chats_set: set =
pickle.loads(read_from_file(f"{USERS_DATA}{email}\\chats",
"rb"))
        except EOFError:
            chats_set = set()
    finally:
```

```
        unblock(f"{USERS_DATA}{email}\\chats block")
    return chats_set


def add_user_to_group_users_file(email: str, chat_id: str) ->
bool:
    """ add user to group users file, updates the new data """
    block(f"{USERS_DATA}{chat_id}\\users_block")
    try:
        if not os.path.isfile(f"{USERS_DATA}{chat_id}\\users"):
            write_to_file(f"{USERS_DATA}{chat_id}\\users", "wb",
b"")
        try:
            users_set: set =
pickle.loads(read_from_file(f"{USERS_DATA}{chat_id}\\users",
"rb"))
        except EOFError:
            users_set = set()
        users_set.add(email)
        write_to_file(f"{USERS_DATA}{chat_id}\\users", "wb",
pickle.dumps(users_set))
    finally:
        unblock(f"{USERS_DATA}{chat_id}\\users_block")
    sync_new_data_with_client(get_group_users(chat_id),
f"{USERS_DATA}{chat_id}\\users")
    return True


def remove_user_from_group_users_file(email: str, chat_id: str)
-> bool:
    """ remove user from group users file """
    block(f"{USERS_DATA}{chat_id}\\users_block")
    try:
        if not os.path.isfile(f"{USERS_DATA}{chat_id}\\users"):
            write_to_file(f"{USERS_DATA}{chat_id}\\users", "wb",
b"")
        try:
            users_set: set =
pickle.loads(read_from_file(f"{USERS_DATA}{chat_id}\\users",
"rb"))
        except EOFError:
            users_set = set()
        if email not in users_set:
            return False
        users_set.remove(email)
        write_to_file(f"{USERS_DATA}{chat_id}\\users", "wb",
pickle.dumps(users_set))
```

```
    finally:
        unblock(f"{USERS_DATA}{chat_id}\\users_block")
    sync_new_data_with_client(get_group_users(chat_id),
f"{USERS_DATA}{chat_id}\\users")
    return True


def get_group_users(chat_id: str) -> set[str]:
    """ get group users file """
    block(f"{USERS_DATA}{chat_id}\\users_block")
    try:
        if not os.path.isfile(f"{USERS_DATA}{chat_id}\\users"):
            write_to_file(f"{USERS_DATA}{chat_id}\\users", "wb",
b"")
        try:
            users_set: set =
pickle.loads(read_from_file(f"{USERS_DATA}{chat_id}\\users",
"rb"))
        except EOFError:
            users_set = set()
    finally:
        unblock(f"{USERS_DATA}{chat_id}\\users_block")
    return users_set


def is_user_in_chat(user_email: str, chat_id: str) -> bool:
    """ check if a user is in a chat
    :param user_email: the email of the user to check if he is
in the chat
    :param chat_id: the id of the chat that will be checked
    :return: True if the user is in the chat else False
    """
    if user_email not in email_password_database:
        return False
    if not os.path.isfile(f"{USERS_DATA}{user_email}\\chats"):
        return False
    try:
        chats_list: set =
pickle.loads(read_from_file(f"{USERS_DATA}{user_email}\\chats",
"rb"))
    except EOFError:
        chats_list = set()
    if chat_id not in chats_list:
        return False
    return True
```

```
def sync_new_data_with_client(emails: str | Iterable[str],
new_data_paths: str | Iterable[str]) -> None:
    """ send modified/new files to online users
    :param emails: the emails of the users to add the new data
to there new_data file
    :param new_data_paths: the paths to the new files / updated
files
    """
    if isinstance(emails, str):
        emails: set[str] = {emails}
    elif not isinstance(emails, set):  # some other type of
iterable
        emails: set[str] = set(emails)
    users_data_sync = False
    if isinstance(new_data_paths, str):
        if new_data_paths == "|users_data":
            users_data_sync = True
        new_data_paths: list[str] = [new_data_paths]
    for email in emails:
        if email not in email_password_database:
            continue
        if email in sync_sockets:
            for client_sync_socket in sync_sockets[email]:
                if not users_data_sync:
                    threading.Thread(target=sync, args=(email,
client_sync_socket, False, new_data_paths)).start()
                else:
                    threading.Thread(target=sync, args=(email,
client_sync_socket, False, [], True)).start()


def add_one_on_one_chat(email_1: str, email_2: str):
    """ adds a chat id (of type one on one) to users
one_on_one_chats file """
    for email in [email_1, email_2]:
        block(f"{USERS_DATA}{email}\\one_on_one_chats_block")
        try:
            if not
os.path.isfile(f"{USERS_DATA}{email}\\one_on_one_chats"):

write_to_file(f"{USERS_DATA}{email}\\one_on_one_chats", "wb",
b"")
            try:
                one_on_one_set: set =
pickle.loads(read_from_file(f"{USERS_DATA}{email}\\one_on_one_ch
ats", "rb"))
            except EOFError:
```

```
                one_on_one_set = set()
            one_on_one_set.add(email_2)

write_to_file(f"{USERS_DATA}{email}\\one_on_one_chats", "wb",
pickle.dumps(one_on_one_set))
        finally:

unblock(f"{USERS_DATA}{email}\\one_on_one_chats_block")
        sync_new_data_with_client(email,
f"{USERS_DATA}{email}\\one_on_one_chats")


def get_one_on_one_chats_list_of(email: str) -> set[str]:
    """ get all the chats (not groups) of a user """
    if email not in email_password_database:
        return set()
    block(f"{USERS_DATA}{email}\\one_on_one_chats_block")
    try:
        try:
            one_on_one_set: set =
pickle.loads(read_from_file(f"{USERS_DATA}{email}\\one_on_one_ch
ats", "rb"))
        except EOFError:
            one_on_one_set = set()
    finally:
        unblock(f"{USERS_DATA}{email}\\one_on_one_chats_block")
    return one_on_one_set


def known_to_each_other(emails: list[str]) -> None:
    """ mark emails as known to each other
    :param emails: the emails of the users that are known to
each other
    """
    for email in emails:
        if email not in email_user_database:
            continue
        block(f"{USERS_DATA}{email}\\known_users_block")
        try:
            if not
os.path.isfile(f"{USERS_DATA}{email}\\known_users"):

write_to_file(f"{USERS_DATA}{email}\\known_users", "wb", b"")
            try:
                known_to_user: set =
pickle.loads(read_from_file(f"{USERS_DATA}{email}\\known_users",
"rb"))
```

```
            except EOFError:
                known_to_user = set()
            for email_2 in emails:
                if email == email_2 or email_2 not in
email_user_database:
                    continue
                known_to_user.add(email_2)
                sync_new_data_with_client(
                    email, f"known user profile
picture|{USERS_DATA}{email_2}\\{email_2}_profile_picture.png")
            write_to_file(f"{USERS_DATA}{email}\\known_users",
"wb", pickle.dumps(known_to_user))
        finally:
            unblock(f"{USERS_DATA}{email}\\known_users_block")
        sync_new_data_with_client(email,
f"{USERS_DATA}{email}\\known_users")


def get_user_known_users(email: str) -> set[str]:
    """ get all the users known to a user """
    block(f"{USERS_DATA}{email}\\known_users_block")
    try:
        if not
os.path.isfile(f"{USERS_DATA}{email}\\known_users"):
            write_to_file(f"{USERS_DATA}{email}\\known_users",
"wb", b"")
        try:
            known_to_user: set =
pickle.loads(read_from_file(f"{USERS_DATA}{email}\\known_users",
"rb"))
        except EOFError:
            known_to_user = set()
    finally:
        unblock(f"{USERS_DATA}{email}\\known_users_block")
    return known_to_user


def create_new_chat(ip: str, user_created: str, with_user: str)
-> tuple[bool, str]:
    """ create a new chat (one on one, not group)
    :param ip: the ip of the clients
    :param user_created: the email of the user that created the
chat
    :param with_user: the email of the user that the chat is
created with
    :return: (True, chat_id) if the chat was created else
(False, "")
```

```
    """
    # check that the 2 users exist
    if user_created not in email_user_database:
        return False, ""
    if with_user not in email_user_database:
        return False, "User Doesn't Exist."
    user_created_username = email_user_database[user_created]
    with_user_username = email_user_database[with_user]
    user_created_one_on_one_chats =
get_one_on_one_chats_list_of(user_created)
    if with_user in user_created_one_on_one_chats:
        return False, "Chat Already Exists."
    chat_id = "".join(random.choices(CHAT_ID_CHARS, k=20))
    while not chat_id_users_database.safe_set(chat_id,
{user_created, with_user}):
        chat_id = "".join(random.choices(CHAT_ID_CHARS, k=20))
    try:
        os.makedirs(f"{USERS_DATA}{chat_id}\\data\\chat",
exist_ok=False)
        os.makedirs(f"{USERS_DATA}{chat_id}\\data\\files",
exist_ok=True)
    # if OSError is raised, that means that there is already a
chat with this chat id
    # and there shouldn't be according to the
chat_id_users_database
    except OSError:
        return False, ""
    # chat metadata
    write_to_file(f"{USERS_DATA}{chat_id}\\name", "wb",
pickle.dumps([user_created_username, with_user_username]))
    write_to_file(f"{USERS_DATA}{chat_id}\\type", "w", "chat")
    write_to_file(f"{USERS_DATA}{chat_id}\\users", "wb", b"")
    write_to_file(f"{USERS_DATA}{chat_id}\\unread_msgs", "wb",
                  pickle.dumps({user_created: 0, with_user: 0}))
    add_user_to_group_users_file(user_created, chat_id)
    add_user_to_group_users_file(with_user, chat_id)
    # add chat id to each user chats
    add_chat_id_to_user_chats(user_created, chat_id)
    add_chat_id_to_user_chats(with_user, chat_id)
    # add with_user to user_created one_on_one_chats file
    # add user_created to with_user one_on_one_chats file
    add_one_on_one_chat(user_created, with_user)
    #
    known_to_each_other([with_user, user_created])
    sync_new_data_with_client([user_created, with_user],
f"{USERS_DATA}{chat_id}")
    send_msg(ip, user_created, chat_id, f"{user_created} added
```

```
{with_user}.", add_message=True)
    return True, chat_id


def create_new_group(ip: str, user_created: str, users:
list[str], group_name: str) -> tuple[bool, str]:
    """ create a new group """
    users.append(user_created)
    #
    for email in set(users):
        if email not in email_user_database:
            return False, ""
    chat_id = "".join(random.choices(CHAT_ID_CHARS, k=20))
    while not chat_id_users_database.safe_set(chat_id,
set(users)):
        chat_id = "".join(random.choices(CHAT_ID_CHARS, k=20))
    try:
        os.makedirs(f"{USERS_DATA}{chat_id}\\data\\chat",
exist_ok=False)
        os.makedirs(f"{USERS_DATA}{chat_id}\\data\\files",
exist_ok=True)
    # if OSError is raised, that means that there is already a
chat with this chat id
    # and there shouldn't be according to the
chat_id_users_database
    except OSError:
        return False, ""
    # chat metadata
    write_to_file(f"{USERS_DATA}{chat_id}\\name", "wb",
pickle.dumps([group_name]))
    write_to_file(f"{USERS_DATA}{chat_id}\\type", "w", "group")
    write_to_file(f"{USERS_DATA}{chat_id}\\users", "wb", b"")
    write_to_file(
        f"{USERS_DATA}{chat_id}\\group_picture.png", "wb",

read_from_file(f"{SERVER_DATA}\\default_group_picture.png",
"rb")
    )
    write_to_file(f"{USERS_DATA}{chat_id}\\unread_msgs", "wb",
                  pickle.dumps(dict(((user_email, 0) for
user_email in users))))
    for email in users:
        add_chat_id_to_user_chats(email, chat_id)
        add_user_to_group_users_file(email, chat_id)
    known_to_each_other(users)
    sync_new_data_with_client(users, f"{USERS_DATA}{chat_id}")
    for user in users:
```

```
        if user != user_created:
            send_msg(ip, user_created, chat_id, f"{user_created}
added {user}.", add_message=True)
    return True, chat_id


def add_user_to_group(ip: str, from_user: str, add_user: str,
group_id: str) -> bool:
    """ add a user to group (all the messages from before will
be visible to him) """
    if from_user not in email_user_database or add_user not in
email_user_database or \
            not is_user_in_chat(from_user, group_id):
        return False
    group_users = get_group_users(group_id)  # without the new
user
    # update database
    chat_id_users_database.add(group_id, add_user)
    # update file of users
    add_user_to_group_users_file(add_user, group_id)
    #
    add_chat_id_to_user_chats(add_user, group_id)
    unread_msgs: dict =
pickle.loads(read_from_file(f"{USERS_DATA}{group_id}\\unread_msg
s", "rb"))
    unread_msgs[from_user] = 0
    write_to_file(f"{USERS_DATA}{group_id}\\unread_msgs", "wb",
pickle.dumps(unread_msgs))
    # make the entire chat as new data for the added user
    sync_new_data_with_client(add_user,
f"{USERS_DATA}{group_id}")
    # only update the users file for the others
    sync_new_data_with_client(group_users,
f"{USERS_DATA}{group_id}\\users")
    send_msg(ip, from_user, group_id, f"{from_user} added
{add_user}.", add_message=True)
    return True


def remove_user_from_group(ip: str, from_user: str, remove_user:
str, group_id: str) -> bool:
    """ remove a user from group (all the messages will be
deleted for him) """
    if from_user not in email_user_database or remove_user not
in email_user_database or \
            not is_user_in_chat(from_user, group_id):
        return False
```

```
    # update database
    chat_id_users_database.remove_set(group_id, remove_user)
    # update file of users
    remove_user_from_group_users_file(remove_user, group_id)
    #
    remove_chat_id_from_user_chats(remove_user, group_id)
    unread_msgs: dict =
pickle.loads(read_from_file(f"{USERS_DATA}{group_id}\\unread_msg
s", "rb"))
    unread_msgs.pop(from_user)
    write_to_file(f"{USERS_DATA}{group_id}\\unread_msgs", "wb",
pickle.dumps(unread_msgs))
    #
    sync_new_data_with_client(remove_user,
[f"{USERS_DATA}{remove_user}\\chats", f"remove -
{USERS_DATA}{group_id}"])
    sync_new_data_with_client(get_group_users(group_id),
f"{USERS_DATA}{group_id}\\users")
    send_msg(ip, from_user, group_id, f"{from_user} removed
{remove_user}.", remove_msg=True)
    return True


def send_msg(ip: str, from_user: str, chat_id: str, msg: str,
             file_msg: bool = False, remove_msg: bool = False,
add_message: bool = False) \
        -> bool | tuple[bool, tuple[set, list[str]]]:
    """ send message (to chat/group)
    :param ip: the ip of the client that sent the request
    :param from_user: the email of the user that sent the msg
    :param chat_id: the id of the chat that the msg is being
sent to
    :param msg: the msg
    :param file_msg: if a file was sent to a chat the send_file
                     function will call this function with
file_msg=True
                     and the msg will be the file location
    :param remove_msg: a message that will say 'x removed y' and
will be displayed different
    :param add_message: a message that will say 'x added y' and
will be displayed different
    """
    # 3 types: regular msg / file msg (if it's a file) / remove
msg (if someone removed someone)
    msg_type = "msg" if not file_msg and not remove_msg and not
remove_msg and not add_message else \
        "file" if file_msg else "remove" if remove_msg else
```

```
"add" if add_message else None
    if msg_type is None or (msg_type == "msg" and msg == ""):
        return False
    lock = block(f"{USERS_DATA}{chat_id}\\data\\not free")
    try:
        users_in_chat: set = chat_id_users_database.get(chat_id)
        if users_in_chat is None or not
is_user_in_chat(from_user, chat_id):
            return False
        list_of_chat_files =
os.listdir(f"{USERS_DATA}{chat_id}\\data\\chat\\")
        if list_of_chat_files:
            latest = int(max(list_of_chat_files))
            try:
                data: dict =
pickle.loads(read_from_file(f"{USERS_DATA}{chat_id}\\data\\chat\
\{latest}", "rb"))
            except EOFError:
                data = {}
            first_chat = False
        else:
            latest = -1
            data = {}
            first_chat = True
        if len(data) >= 800 or first_chat:
            #        index:             [from_user, msg,
msg_type, deleted_for, delete_for_all, seen by, time]
            time_formatted =
datetime.datetime.now().strftime("%m/%d/%Y %H:%M")
            data = {(latest + 1) * 800: [from_user, msg,
msg_type, [], False, [], time_formatted]}

write_to_file(f"{USERS_DATA}{chat_id}\\data\\chat\\{latest +
1}", "wb", pickle.dumps(data))
        else:
            #        index:              [from_user, msg,
msg_type, deleted_for, delete_for_all, seen by, time]
            time_formatted =
datetime.datetime.now().strftime("%m/%d/%Y %H:%M")
            data[max(data.keys()) + 1] = [from_user, msg,
msg_type, [], False, [], time_formatted]

write_to_file(f"{USERS_DATA}{chat_id}\\data\\chat\\{latest}",
"wb", pickle.dumps(data))
        block(f"{USERS_DATA}{chat_id}\\unread messages not
free")
        try:
```

```
            unread_msgs: dict =
pickle.loads(read_from_file(f"{USERS_DATA}{chat_id}\\unread_msgs
", "rb"))
        except EOFError:
            unread_msgs = {}
        for user in unread_msgs.keys():
            if user != from_user:
                unread_msgs[user] += 1
        write_to_file(f"{USERS_DATA}{chat_id}\\unread_msgs",
"wb", pickle.dumps(unread_msgs))
        unblock(f"{USERS_DATA}{chat_id}\\unread messages not
free")
        # when finished remove the folder
        lock = unblock(f"{USERS_DATA}{chat_id}\\data\\not free")
        # add the new file / updated file to the new data of all
the users in the chat
        latest = latest + 1 if len(data) >= 800 or first_chat
else latest
        sync_paths = [f"{USERS_DATA}{chat_id}\\unread_msgs",
f"{USERS_DATA}{chat_id}\\data\\chat\\{latest}"]
        if not file_msg:
            sync_new_data_with_client(users_in_chat, sync_paths)
        return True if not file_msg else (True, (users_in_chat,
sync_paths))
    except Exception as e:
        traceback.print_exception(e)
        add_exception_for_ip(ip)
        logging.warning(f"received exception while handling
'{ip}' exception: "

f"{''.join(traceback.format_exception(e))} (user: '{from_user}',
func: 'send_msg')")
        return False if not file_msg else (False, (set(), ""))
    finally:
        if lock:
            unblock(f"{USERS_DATA}{chat_id}\\data\\not free")


def send_file(ip: str, from_user: str, chat_id: str, file_data:
bytes, file_name: str) -> bool:
    """ send file (to chat/group)
    :param ip: the ip of the client that sent the request
    :param from_user: the email of the user that sent the file
    :param chat_id: the id of the chat that the file is being
sent to
    :param file_data: the data of the file
    :param file_name: the name of the file
```

```
    """
    try:
        users_in_chat: set = chat_id_users_database.get(chat_id)
        if users_in_chat is None or not
is_user_in_chat(from_user, chat_id):
            return False
        location = f"{USERS_DATA}{chat_id}\\data\\files\\"
        # if there is already a file with this name, create new
name
        if os.path.isfile(location + file_name):
            new_file_name = ".".join(file_name.split(".")[:-1])
+ "_1." + file_name.split(".")[-1]
            i = 2
            while os.path.isfile(location + new_file_name):
                new_file_name = ".".join(file_name.split(".")[:-
1]) + f"_{i}." + file_name.split(".")[-1]
                i += 1
        else:
            new_file_name = file_name
        # save the file
        with open(location + new_file_name, "wb") as file:
            file.write(file_data)
        #
remove USERS_DATA
        status, (users_in_chat, new_data) = \
            send_msg(ip, from_user, chat_id, "\\".join((location
+ new_file_name).split("\\")[2:]), file_msg=True)
        if status:
            sync_new_data_with_client(users_in_chat, [location +
new_file_name, *new_data])
            return True
        else:
            os.remove(location + new_file_name)
            return False
    except Exception as e:
        traceback.print_exception(e)
        add_exception_for_ip(ip)
        logging.warning(f"received while handling '{ip}'
exception: "

f"{''.join(traceback.format_exception(e))} (user: '{from_user}',
func: 'send_file')")
        return False


def delete_msg_for_me(ip: str, from_user: str, chat_id: str,
index_of_msg: int) -> bool:
```

```
    """ delete massage only for yourself (in chat/group) """
    users_in_chat: set = chat_id_users_database.get(chat_id)
    if users_in_chat is None or not is_user_in_chat(from_user,
chat_id):
        return False
    file_number = index_of_msg // 800  # there are 800 messages
per file
    if not
os.path.isfile(f"{USERS_DATA}{chat_id}\\data\\chat\\{file_number
}"):
        return False  # index_of_msg is invalid
    lock = block(f"{USERS_DATA}{chat_id}\\data\\not free")
    try:
        try:
            data: dict =
pickle.loads(read_from_file(f"{USERS_DATA}{chat_id}\\data\\chat\
\{file_number}", "rb"))
        except EOFError:
            data = {}
        # msg -> [from_user, msg, msg_type, deleted_for,
delete_for_all, seen by, time]
        msg = data.get(index_of_msg)
        if msg is not None:
            deleted_for = msg[3]
            if from_user not in deleted_for:
                deleted_for.append(from_user)
            msg[3] = deleted_for
            data[index_of_msg] = msg

write_to_file(f"{USERS_DATA}{chat_id}\\data\\chat\\{file_number}
", "wb", pickle.dumps(data))
        else:
            lock = unblock(f"{USERS_DATA}{chat_id}\\data\\not
free")
            return False
        lock = unblock(f"{USERS_DATA}{chat_id}\\data\\not free")
        sync_new_data_with_client(from_user,
f"{USERS_DATA}{chat_id}\\data\\chat\\{file_number}")
        return True
    except Exception as e:
        traceback.print_exception(e)
        add_exception_for_ip(ip)
        logging.debug(f"received while handling '{ip}'
exception: "
                      f"{''.join(traceback.format_exception(e))}
(user: '{from_user}', func: 'delete_msg_for_me')")
        return False
```

```
    finally:
        if lock:
            unblock(f"{USERS_DATA}{chat_id}\\data\\not free")


def delete_msg_for_everyone(ip: str, from_user: str, chat_id:
str, index_of_msg: int) -> bool:
    """ delete massage for everyone (in chat/group) """
    users_in_chat: set = chat_id_users_database.get(chat_id)
    if users_in_chat is None or not is_user_in_chat(from_user,
chat_id):
        return False
    file_number = index_of_msg // 800  # there are 800 messages
per file
    if not
os.path.isfile(f"{USERS_DATA}{chat_id}\\data\\chat\\{file_number
}"):
        return False  # index_of_msg is invalid
    lock = block(f"{USERS_DATA}{chat_id}\\data\\not free")
    try:
        try:
            data: dict =
pickle.loads(read_from_file(f"{USERS_DATA}{chat_id}\\data\\chat\
\{file_number}", "rb"))
        except EOFError:
            data = {}
        # msg -> [from_user, msg, msg_type, deleted_for,
delete_for_all, seen by, time]
        msg = data.get(index_of_msg)
        if msg is not None and msg[0] == from_user and not
msg[4]:
            path_to_file = msg[1]
            msg[1] = "This Message Was Deleted."
            msg[4] = True
            data[index_of_msg] = msg

write_to_file(f"{USERS_DATA}{chat_id}\\data\\chat\\{file_number}
", "wb", pickle.dumps(data))
            if msg[2] == "file":  # file msg, remove the file as
well
                #
file name
                os.remove(f"{USERS_DATA}{path_to_file}")
                # tell all the clients to remove this file on
their side
                sync_new_data_with_client(users_in_chat,
f"remove - {USERS_DATA}{path_to_file}")
```

```
        else:
            lock = unblock(f"{USERS_DATA}{chat_id}\\data\\not
free")
            return False
        lock = unblock(f"{USERS_DATA}{chat_id}\\data\\not free")
        sync_new_data_with_client(users_in_chat,
f"{USERS_DATA}{chat_id}\\data\\chat\\{file_number}")
        return True
    except Exception as e:
        traceback.print_exception(e)
        add_exception_for_ip(ip)
        logging.debug(f"received while handling '{ip}'
exception: {''.join(traceback.format_exception(e))} "
                      f"(user: '{from_user}', func:
'delete_msg_for_everyone')")
        return False
    finally:
        if lock:
            unblock(f"{USERS_DATA}{chat_id}\\data\\not free")


def mark_as_seen(chat_id: str, user_email: str) -> None:
    """ Mark all unread msgs in the chat that the user is
currently in as read """
    users_in_chat: set = chat_id_users_database.get(chat_id)
    if users_in_chat is None or not is_user_in_chat(user_email,
chat_id):
        return
    block(f"{USERS_DATA}{chat_id}\\unread messages not free")
    try:
        try:
            unread_msgs: dict =
pickle.loads(read_from_file(f"{USERS_DATA}{chat_id}\\unread_msgs
", "rb"))
        except EOFError:
            unread_msgs = {}
        unread_msgs_amount = unread_msgs[user_email]
        unread_msgs[user_email] = 0
        write_to_file(f"{USERS_DATA}{chat_id}\\unread_msgs",
"wb", pickle.dumps(unread_msgs))
        sync_new_data_with_client(user_email,
f"{USERS_DATA}{chat_id}\\unread_msgs") if unread_msgs_amount !=
0 \
            else None
    finally:
        unblock(f"{USERS_DATA}{chat_id}\\unread messages not
free")
```

```
    if unread_msgs_amount > 0:
        block(f"{USERS_DATA}{chat_id}\\data\\not free")
        try:
            chat_files =
os.listdir(f"{USERS_DATA}{chat_id}\\data\\chat")
            chat_files.sort(key=lambda x: int(x), reverse=True)
            current_file_pos = 0
            while unread_msgs_amount > 0 and current_file_pos !=
len(chat_files):
                # msgs -> {index: [from_user, msg, msg_type,
deleted_for, delete_for_all, seen by, time]}
                msgs: dict = pickle.loads(

read_from_file(f"{USERS_DATA}{chat_id}\\data\\chat\\{chat_files[
current_file_pos]}", "rb"))
                added_to_new_data = False
                for msg_index in msgs.keys():
                    if user_email not in msgs[msg_index][-2]:
                        # TODO: maybe if everyone read it just
change to True instead of a list?
                        msgs[msg_index][-2].append(user_email)
                        unread_msgs_amount -= 1
                        if not added_to_new_data:
                            added_to_new_data = True
                            sync_new_data_with_client(
                                users_in_chat,
f"{USERS_DATA}{chat_id}\\data\\chat\\{chat_files[current_file_po
s]}")
                current_file_pos += 1
        finally:
            unblock(f"{USERS_DATA}{chat_id}\\data\\not free")


def upload_profile_picture(email: str, picture_file: bytes) ->
bool:
    """ change your profile picture """

write_to_file(f"{USERS_DATA}{email}\\{email}_profile_picture.png
", "wb", picture_file)
    known_to_user = get_user_known_users(email)
    known_to_user.add(email)
    sync_new_data_with_client(email,
f"{USERS_DATA}{email}\\{email}_profile_picture.png")
    sync_new_data_with_client(
        known_to_user, f"known user profile
picture|{USERS_DATA}{email}\\{email}_profile_picture.png")
    return True
```

```
def update_group_photo(from_user: str, chat_id: str,
picture_file: bytes) -> bool:
    """ change group picture """
    users_in_chat: set = chat_id_users_database.get(chat_id)
    if users_in_chat is None or not is_user_in_chat(from_user,
chat_id):
        return False

write_to_file(f"{USERS_DATA}\\{chat_id}\\group_profile_picture.p
ng", "wb", picture_file)
    group_users = get_group_users(chat_id)
    sync_new_data_with_client(group_users,
f"{USERS_DATA}\\{chat_id}\\group_profile_picture.png")
    return True


def send_mail(to: str, subject: str, body: str, html: str = "")
-> None:
    """ send an email (for signup and password reset) """
    email_msg = MIMEMultipart('alternative')
    email_msg["From"] = SERVER_EMAIL
    email_msg["To"] = to
    email_msg["Subject"] = subject
    email_msg.attach(MIMEText(body, "plain"))
    if html != "":
        email_msg.attach(MIMEText(html, "html"))
    with smtplib.SMTP_SSL("smtp.gmail.com", 465,
context=ssl.create_default_context()) as smtp:
        smtp.login(SERVER_EMAIL, SERVER_EMAIL_APP_PASSWORD)
        smtp.sendmail(SERVER_EMAIL, to, email_msg.as_string())
    logging.info(f"sent email to {to}")


def signup(username: str, email: str, password: str,
client_sock: ServerEncryptedProtocolSocket) -> tuple[bool, str]:
    """  signup
    :param username: the username
    :param email: the email of the user
    :param password: the md5 hash of the password
    :param client_sock: the socket of the client
    """
    # check that the email isn't registered
    if email in email_user_database:
        return False, "Error"
    client_sock.send_message("signup".ljust(30).encode())
```

```
    # create confirmation code
    confirmation_code = random.choices(range(0, 10), k=6)
    confirmation_code = "".join(map(str, confirmation_code))
    # send confirmation code to the email
    send_mail(email, "Confirmation Code",
              f"Your code is: {confirmation_code}",
              f"<div style='color: rgb(18, 151, 228); font-size:
xx-large;'>Your code is: {confirmation_code}</div>")
    print(f"[Server]: A mail was sent to '{email}' with the
confirmation code '{confirmation_code}'")
    # send client a msg that says that we are waiting for a
confirmation code

client_sock.send_message(f"{'confirmation_code'.ljust(30)}".enco
de())
    # receive the response from the client and set a timeout of
5 minutes
    msg = client_sock.recv_message(timeout=60*5)
    # if response timed out
    if msg == b"":
        return False, "Request timeout."
    msg = msg.decode()
    # check the response
    if msg[: 30].strip() != "confirmation_code" or msg[30:] !=
confirmation_code:
        logging.info(f"signup attempt (for '{email}') failed - "
                     f"got wrong confirmation code from user
trying to signup as '{username}'")
        return False, "Confirmation code is incorrect."
    # if username doesn't exist
    if email not in email_user_database and len(username) <= 40:
        # add username and password to the
email_password_database & email_user_database
        email_password_database[email] =
hashlib.md5(password.encode()).hexdigest().lower()
        email_user_database[email] = username
        user_online_status_database[email] = ["Offline",
datetime.datetime.now()]
        logging.info(f"signup attempt successful - email:
'{email}', username: '{username}'")
        # create user directory and files
        os.makedirs(f"{USERS_DATA}{email}", exist_ok=True)
        write_to_file(f"{USERS_DATA}{email}\\chats", "wb", b"")
        write_to_file(f"{USERS_DATA}{email}\\known_users", "wb",
b"")
        write_to_file(f"{USERS_DATA}{email}\\one_on_one_chats",
"wb", b"")
```

```
        write_to_file(
            f"{USERS_DATA}{email}\\{email}_profile_picture.png",
"wb",

read_from_file(f"{SERVER_DATA}\\default_group_picture.png",
"rb")
        )
        print(f"Signed up successfully {email}-{username}")
        return True, "Signed up successfully."
    return False, "Email already registered."


def reset_password(email: str, username: str, client_sock:
ServerEncryptedProtocolSocket) -> tuple[bool, str]:
    """ reset password
    :param email: the email of the user that wants to reset
their password
    :param username: the username
    :param client_sock: the socket of the client
    """
    if email not in email_user_database:
        return False, ""
    if email_user_database[email] != username:
        return False, ""
    client_sock.send_message("reset
password".ljust(30).encode())
    username = email_user_database[email]
    # create confirmation code
    confirmation_code = random.choices(range(0, 10), k=6)
    confirmation_code = "".join(map(str, confirmation_code))
    # send confirmation code to the email
    send_mail(email, "Confirmation Code",
              f"Your code is: {confirmation_code}",
              f"<div style='color: rgb(18, 151, 228); font-size:
xx-large;'>Your code is: {confirmation_code}</div>")
    print(f"[Server]: A mail was sent to '{email}' with the
confirmation code '{confirmation_code}'")
    # send client a msg that says that we are waiting for a
confirmation code

client_sock.send_message(f"{'confirmation_code'.ljust(30)}".enco
de())
    # receive the response from the client and set a timeout of
5 minutes
    msg = client_sock.recv_message(timeout=60 * 5)
    # if response timed out
    if msg == b"":
```

```
        return False, "Request timeout."
    msg = msg.decode()
    # check the response
    if msg[: 30].strip() != "confirmation_code" or msg[30:] !=
confirmation_code:
        logging.info(f"reset password attempt (for '{email}')
failed - got wrong confirmation code from user")
        return False, "Confirmation code is incorrect."
    client_sock.send_message(f"{'reset
password'.ljust(30)}{'ok'.ljust(6)}".encode())
    # send client a msg that says that we are waiting for a new
password
    client_sock.send_message("new password".ljust(30).encode())
    # receive the response from the client and set a timeout of
5 minutes
    msg = client_sock.recv_message(timeout=60 * 5)
    # if response timed out
    if msg == b"":
        return False, "Request timeout."
    msg = msg.decode()
    # validate the response
    if msg[: 30].strip() != "new password":
        return False, ""
    # extract password
    password = msg[30:]  # the md5 of the password
    # set new password
    email_password_database[email] =
hashlib.md5(password.encode()).hexdigest().lower()  # md5 again
    print(f"reset password attempt successful - email:
'{email}', username: '{username}'.")
    logging.info(f"reset password attempt successful - email:
'{email}', username: '{username}'.")
    return True, "Password changed successfully."


def login(email: str, password: str) -> bool:
    """ login
    :param email: the email of the user
    :param password: the md5 hash of the password
    """
    time.sleep(random.random())  # prevent timing attack (sleep
between 0 and 1 seconds)
    if email not in email_user_database:
        return False
    user_password_hashed_hash = email_password_database[email]
    if user_password_hashed_hash !=
hashlib.md5(password.encode()).hexdigest().lower():
```

```
        return False
    block(f"{USERS_DATA}{email}\\online status")
    try:
        if email in user_online_status_database and
user_online_status_database[email][0] == "Online":
            user_online_status_database[email] = ["Online",
user_online_status_database[email][1] + 1]
        else:
            online_clients[email] = None
            user_online_status_database[email] = ["Online", 1]

sync_new_data_with_client(get_user_known_users(email),
f"|users_data")
    finally:
        unblock(f"{USERS_DATA}{email}\\online status")
    return True


def sync(email: str, client_sync_sock:
ServerEncryptedProtocolSocket,
         sync_all: bool = False, new_data_paths: list[str] =
None, sync_users_status: bool = False) -> None:
    """ send user all the requested files (those who are new or
changed / all his files) """
    block(f"{USERS_DATA}{email}\\sync")
    try:
        new_data = []
        if not sync_all:
            if new_data_paths is None:
                return
            if new_data_paths:  # if the list isn't empty, can
be empty when syncing users_status only
                new_data = new_data_paths
                add, remove = [], []
                for i in range(len(new_data)):
                    path = new_data[i]
                    if os.path.isdir(path):
                        # add all files in dir and sub-dirs
                        for path2, _, files in os.walk(path):
                            add.extend(map(lambda p:
os.path.join(path2, p), files))  # (can't change during
iteration)
                        remove.append(i)  # remove the folder
itself from new_data list (can't change during iteration)
                for index in reversed(remove):
                    # move the item we want to remove to the end
of the list and then pop it,
```

```
                    # better performance because if we remove an
item that isn't the last all the items
                    # after it need to move 1 index back, unless
we remove the last item in the list
                    if index != len(new_data) - 1:
                        new_data[-1], new_data[index] =
new_data[index], new_data[-1]
                    new_data.pop()
                new_data.extend(add)
                del add, remove
        else:
            # user metadata
            new_data = [f"{USERS_DATA}{email}\\{file}" for file
in os.listdir(f"{USERS_DATA}{email}\\")]
            chat_ids_set = get_user_chats_file(email)  # user
chats and groups
            for chat_id in chat_ids_set:
                for path2, _, files in
os.walk(f"{USERS_DATA}{chat_id}"):
                    new_data.extend(map(lambda p:
os.path.join(path2, p), files))
            known_to_user = get_user_known_users(email)
            for other_email in known_to_user:
                new_data.append(
                    f"known user profile
picture|{USERS_DATA}{other_email}\\{other_email}_profile_picture
.png")
        # make a dictionary -> {file_path: file_data}
        file_name_data: dict[str, bytes | str] = {}
        if sync_users_status or sync_all:
            known_to_user = get_user_known_users(email)
            users_status: dict[str, int | datetime.datetime |
str] = \
                dict(((user,
user_online_status_database.get(user)[1]) for user in
known_to_user))
            current_time = datetime.datetime.now()
            for user in users_status.keys():
                if not isinstance(users_status[user], int):
                    time_format = "%H:%M %m/%d/%Y" if
(current_time - users_status[user]).days >= 1 else "%H:%M"
                    users_status[user] = f'Last Seen
{users_status[user].strftime(time_format)}'
                else:
                    users_status[user] = "Online"
            file_name_data["users_status"] =
pickle.dumps(users_status)
```

```
        for file in new_data:
            # if it's a chat file, we need to lock it
            if file.count("\\") == 4:
                try:
                    chat_path = "\\".join(file.split("\\")[:-1])
# remove chat file, leave chat_id and data dir
                    if os.path.isdir(chat_path) and
chat_path.endswith("\\data"):
                        file_path_for_user =
"\\".join(file.split("\\")[2:])
                        chat_id =
file_path_for_user.split("\\")[0]
                        block(f"{USERS_DATA}{chat_id}\\data\\not
free")
                        try:
                            file_name_data[file_path_for_user] =
read_from_file(file, "rb")
                        finally:
                            unblock(f"{USERS_DATA}{chat_id}\\data\\not free")
                        continue
                except Exception as e:
                    traceback.format_exception(e)
            elif file.count("\\") == 3 and
file.endswith("unread_msgs"):
                try:
                    block_path = "\\".join(file.split("\\")[:-
1]) + "\\unread messages not free"
                    file_path_for_user =
"\\".join(file.split("\\")[2:])
                    block(block_path)
                    try:
                        file_name_data[file_path_for_user] =
read_from_file(file, "rb")
                    finally:
                        unblock(block_path)
                    continue
                except Exception as e:
                    traceback.format_exception(e)
            if os.path.isfile(file):
                #                           remove
Data\\Users_Data
                file_path_for_user =
"\\".join(file.split("\\")[2:])
                file_name_data[file_path_for_user] =
read_from_file(file, "rb")
            elif file.startswith("known user profile picture|"):
```

```
                real_file_path = "".join(file.split("known user
profile picture|")[1:])
                file_path_for_user =
"\\".join(real_file_path.split("\\")[3:])

file_name_data[f"profile_pictures\\{file_path_for_user}"] =
read_from_file(real_file_path, "rb")
            elif file.startswith("call|"):
                file_name_data[file] = file
            elif file.startswith("remove - "):
                #                           remove the "remove -
"      remove Data\\Users_Data
                file_name_data["\\".join(" - ".join(file.split("
- ")[1:]).split("\\")[2:])] = "remove"
            elif file == f"{USERS_DATA}{email}\\sync":
                pass
            else:
                logging.debug(f"[Server]: error in 'sync'
function, FileNotFound: '{file}'")
        # pickle the dictionary
        sync_res: bytes = pickle.dumps(file_name_data)
        # sync_res can be big, so it's more efficient to not
concat them
        client_sync_sock.send_message("sync".ljust(30).encode()
+ sync_res)
    except Exception as e:
        traceback.print_exception(e)
        try:
            ip, port = client_sync_sock.getpeername()
            add_exception_for_ip(ip)
        except (ConnectionError, socket.error):
            pass
        print(f"received exception in sync while handling
'{email}' ex: {traceback.format_exception(e)}")
        logging.warning(f"received exception in sync while
handling '{email}' ex: {traceback.format_exception(e)}")
    finally:
        unblock(f"{USERS_DATA}{email}\\sync")


def call_group(from_email: str, chat_id: str) -> int | None:
    """ make a call to all the users in the chat - chat_id

    :returns: the port of the call server
    """
    users_in_chat = get_group_users(chat_id)
    if from_email not in users_in_chat:
```

```
        return None
    if chat_id in ongoing_calls and
ongoing_calls[chat_id].is_alive():  # there is an active call
for this chat
        return None
    clients_passwords: dict[str, str] = dict(((email,
email_password_database[email]) for email in users_in_chat))
    #
    users_in_chat.remove(from_email)
    online = []
    for user in users_in_chat:
        if user in user_online_status_database and
user_online_status_database[user][0] == "Online":
            online.append(user)
    not_online = users_in_chat - set(online)
    #

    port = 16400
    tcp_server_sock =
ServerEncryptedProtocolSocket(my_public_key, my_private_key)
    while port <= 65535:
        try:
            tcp_server_sock.bind(("0.0.0.0", port))
            break
        except OSError:  # port taken
            tcp_server_sock.close()
            tcp_server_sock =
ServerEncryptedProtocolSocket(my_public_key, my_private_key)
            port += 1
    if port > 65535:
        return None
    tcp_server_sock.listen()

    p = multiprocessing.Process(target=start_call_server,
args=(tcp_server_sock, port, clients_passwords, print_queue))
    p.start()
    ongoing_calls[chat_id] = p
    print(f"New call server started on {port = }")

    with open(f"{USERS_DATA}{chat_id}\\name", "rb") as f:
        group_name = pickle.loads(f.read())
    group_name = group_name[0] if len(group_name) == 1 else
group_name[0] if group_name[0] == from_email \
        else group_name[1]
    sync_new_data_with_client(online,
f"call|{port}|{group_name}")
    if not_online:  # if there are users that aren't online
```

```
        threading.Thread(target=watch_for_offline_users,
args=(group_name, not_online, port, p), daemon=True).start()
    return port


def watch_for_offline_users(group_name: str, offline_users: set,
port: int,
                            call_server_process:
multiprocessing.Process) -> None:
    """
        As long as the call is active wait for all the
        offline users to come online to alert them about the
call
    """
    while call_server_process.is_alive() and offline_users:
        remove = []
        for user_email in offline_users:
            if user_online_status_database[user_email][0] ==
"Online":
                sync_new_data_with_client(user_email,
f"call|{port}|{group_name}")
                remove.append(user_email)
        for user_email in remove:
            offline_users.remove(user_email)
        time.sleep(1)


def login_or_signup_response(mode: str, status: str, reason:
str) -> bytes:
    """ make a response for login/signup according to the
protocol
    :param mode: login or signup
    :param status: 'ok' or 'not ok'
    :param reason: the reason
    """
    return
f"{mode.ljust(30)}{status.lower().ljust(6)}{reason}".encode()


def request_response(cmd: str, status: str, reason: str) ->
bytes:
    """ make a response according to the protocol
    :param cmd: the requested command
    :param status: 'ok' or 'not ok'
    :param reason: the reason
    """
    return
```

```
f"{cmd.ljust(30)}{status.lower().ljust(6)}{reason}".encode()


def handle_client(client_socket: ServerEncryptedProtocolSocket,
client_ip_port: tuple[str, int]) -> None:
    """ handle a client (each client gets a thread that runs
this function) """
    logged_in, signed_up, stop, email, username = False, False,
False, None, None
    try:
        # let client login / signup and login
        while not logged_in:
            client_socket.settimeout(5)  # add timeout of 5
seconds, if there is no request within this time, close con
            try:
                msg = client_socket.recv_message()
                if msg == b"":
                    break
            except socket.timeout:
                add_exception_for_ip(client_ip_port[0])

client_socket.send_message(login_or_signup_response("login",
"not ok", "Request Timed Out."))
                stop = True
                break
            client_socket.settimeout(None)
            msg = msg.decode()
            cmd = msg[: 30].strip()
            # check if the msg is signup msg or login, else
throw the msg
            # because the client hasn't logged in yet
            if cmd == "login":
                len_email = int(msg[30: 45].strip())
                email = msg[45: 45 + len_email].lower()
                password = msg[45 + len_email:]
                if not login(email, password):
                    client_socket.send_message(
                        login_or_signup_response("login", "not
ok", "Incorrect email or password !")
                    )
                    stop = True
                    break
                logged_in = True
                username = email_user_database[email]

client_socket.send_message(login_or_signup_response("login",
"ok", username))
```

```
                del msg, cmd, len_email, password
            elif cmd == "signup" and not signed_up:
                len_username = int(msg[30: 32].strip())
                username = msg[32: 32 + len_username]
                len_email = int(msg[32 + len_username: 47 +
len_username].strip())
                email = msg[47 + len_username: 47 + len_username
+ len_email].lower()
                password = msg[47 + len_username + len_email:]
                ok, reason = signup(username, email, password,
client_socket)
                if not ok:

client_socket.send_message(login_or_signup_response("signup",
"not ok", reason))
                    stop = True
                    break

client_socket.send_message(login_or_signup_response("signup",
"ok", ""))
                signed_up = True
                del msg, len_username, len_email, password,
reason, ok
            elif cmd == "signup" and signed_up:  # don't allow 1
connection to signup multiple times
                stop = True
                break
            elif cmd == "reset password":
                email_len = int(msg[30: 45])
                tmp_email = msg[45: 45 + email_len].lower()
                tmp_username = msg[45 + email_len:]
                status, reason = reset_password(tmp_email,
tmp_username, client_socket)
                client_socket.send_message(request_response(cmd,
"ok" if status else "not ok", reason))
                if not status:
                    stop = True
                    break
            else:
                add_exception_for_ip(client_ip_port[0])
                stop = True
                break
        if not stop and logged_in:
            if email not in email_user_database:  # double check
that this user exists
                logging.debug(f"[Server]: The email '{email}'
doesn't exists in the database.")
```

```
                raise ValueError(f"[Server]: The email '{email}'
doesn't exists in the database.")
            # set thread name                    email
number of connections (of this client)
            threading.current_thread().name = f"{email} (client
- {user_online_status_database[email][1]})"
            client_socket.settimeout(None)
            username = email_user_database[email]
            password = None  # no need to save the password, set
to None
            #
            msg = f"[Server]: '%s:%s' logged in as '{email}-
{username}'." % client_ip_port
            print(msg), logging.info(msg)
            #
            # handle client's requests until client disconnects
            stay_encoded = {"file", "upload profile picture",
"upload group picture", "new group"}
            while True:
                request: bytes
                request = client_socket.recv_message()
                if request == b"" or request == b"bye":
                    break
                cmd = request[: 30].decode().strip()
                # decode the request only if it's not a file
                request = request if cmd in stay_encoded else
request.decode()
                response = None
                if cmd == "user in chat":
                    request: str
                    chat_id = request[30:]
                    mark_as_seen(chat_id, email)
                elif cmd == "msg":
                    request: str
                    len_chat_id = int(request[30: 45].strip())
# currently 20
                    chat_id = request[45: len_chat_id + 45]
                    msg = request[len_chat_id + 45:]
                    if len(msg) < 5000:
                        ok = send_msg(client_ip_port[0], email,
chat_id, msg)
                    else:
                        ok = False
                    response = request_response(cmd, "ok" if ok
else "not ok", "")
                elif "this is a sync sock" in cmd:
                    msg = f"[Server]: '%s:%s' logged in as
```

```
'{email} - {username}'. is for syncing" % client_ip_port
                    print(msg), logging.info(msg)
                    if cmd.endswith("all"):
                        sync(email, client_socket,
sync_all=True)
                    # add client sync sock to sync_sockets dict
                    sync_sockets_lock.acquire()
                    if email in sync_sockets:
                        sync_sockets[email].add(client_socket)
                    else:
                        sync_sockets[email] = {client_socket}
                    sync_sockets_lock.release()
                elif cmd == "delete for everyone":
                    request: str
                    chat_id_len = int(request[30: 45].strip())
                    chat_id = request[45: 45 + chat_id_len]
                    message_index = int(request[45 +
chat_id_len:].strip())
                    delete_msg_for_everyone(client_ip_port[0],
email, chat_id, message_index)
                elif cmd == "file":
                    request: bytes
                    chat_id_len = int(request[30: 45].strip())
                    chat_id = request[45: 45 +
chat_id_len].decode()
                    file_name_len = int(request[45 +
chat_id_len: 60 + chat_id_len].strip())
                    file_name = request[60 + chat_id_len: 60 +
chat_id_len + file_name_len].decode()
                    file_data = request[60 + chat_id_len +
file_name_len:]
                    send_file(client_ip_port[0], email, chat_id,
file_data, file_name)
                elif cmd == "delete for me":
                    request: str
                    chat_id_len = int(request[30: 45].strip())
                    chat_id = request[45: 45 + chat_id_len]
                    message_index = int(request[45 +
chat_id_len:])
                    delete_msg_for_me(client_ip_port[0], email,
chat_id, message_index)
                elif cmd == "call":
                    request: str
                    chat_id = request[30:]
                    port = call_group(email, chat_id)
                    if port is None:
                        response = request_response(cmd, "not
```

```
ok", "Error")
                    else:
                        response = request_response(cmd, "ok",
f"{port}")
                elif cmd == "add user":
                    request: str
                    chat_id_len = int(request[30: 45].strip())
                    chat_id = request[45: 45 + chat_id_len]
                    other_user = request[45 + chat_id_len:]
                    ok = add_user_to_group(client_ip_port[0],
email, other_user, chat_id)
                    response = request_response(cmd, "ok" if ok
else "not ok", "")
                elif cmd == "remove user":
                    request: str
                    chat_id_len = int(request[30: 45].strip())
                    chat_id = request[45: 45 + chat_id_len]
                    other_user = request[45 + chat_id_len:]
                    ok =
remove_user_from_group(client_ip_port[0], email, other_user,
chat_id)
                    response = request_response(cmd, "ok" if ok
else "not ok", "")
                elif cmd == "upload profile picture":
                    request: bytes
                    status = upload_profile_picture(email,
request[30:])
                    response = request_response(cmd, "ok" if
status else "not ok", "")
                elif cmd == "upload group picture":
                    request: bytes
                    len_chat_id = int(request[30:
45].decode().strip())  # currently 20
                    chat_id = request[45: len_chat_id +
45].decode()
                    status = update_group_photo(email, chat_id,
request[len_chat_id + 45:])
                    response = request_response(cmd, "ok" if
status else "not ok", "")
                elif cmd == "familiarize user with":
                    request: str
                    other_email = request[30:]
                    if other_email in email_user_database:
                        known_to_each_other([email,
other_email])
                        response = request_response(cmd, "ok",
"")
```

```
                    else:
                        response = request_response(cmd, "not
ok", "user doesn't exists")
                elif cmd == "new chat":
                    request: str
                    other_email = request[30:]
                    status, chat_id =
create_new_chat(client_ip_port[0], email, other_email)
                    response = request_response(cmd, "ok" if
status else "not ok", chat_id)
                elif cmd == "new group":
                    request: bytes
                    group_name_len = int(request[30:
45].decode().strip())
                    group_name = request[45: 45 +
group_name_len].decode()
                    other_users_list: list[str] =
pickle.loads(request[45 + group_name_len:])
                    status, chat_id =
create_new_group(client_ip_port[0], email, other_users_list,
group_name)
                    response = request_response(cmd, "ok" if
status else "not ok", chat_id)
                else:
                    msg = f"[Server]: '%s:%s' Logged In As
'{email}-{username}' - sent unknown cmd '{cmd}'" % \
                          client_ip_port
                    print(msg)
                    logging.warning(msg)
                    continue
                # send the response
                if response is not None:
                    client_socket.send_message(response)
    except Exception as err:
        traceback.print_exception(err)
        add_exception_for_ip(client_ip_port[0])
        if not isinstance(err, ConnectionError):
            username = "Unknown username" if "username" not in
locals() else username
            logging.warning(f"[Server]: error while handling
'%s:%s' "
                            f"('{username}'):
{''.join(traceback.format_exception(err))}" % client_ip_port)
    finally:
        sync_sockets_lock.acquire()
        if email in sync_sockets and client_socket in
sync_sockets[email]:
```

```
            sync_sockets[email].remove(client_socket)
        sync_sockets_lock.release()
        block(f"{USERS_DATA}{email}\\online status")
        try:
            if email in user_online_status_database and
user_online_status_database[email][0] == "Online":
                if user_online_status_database[email][1] == 1:
                    if email in online_clients:
                        online_clients.pop(email)
                    user_online_status_database[email] =
["Offline", datetime.datetime.now()]  # last seen

sync_new_data_with_client(get_user_known_users(email),
f"|users_data")
                else:
                    # reduce online count by 1 (each connection
is 1)
                    user_online_status_database[email] =
["Online", user_online_status_database[email][1] - 1]
        finally:
            unblock(f"{USERS_DATA}{email}\\online status")
        #
        client_socket.close()
        username = "Unknown email" if email is None else email
        print(f"[Server]: Client ({email = }) '%s:%s'
disconnected." % client_ip_port)
        logging.info(f"[Server]: Client '%s:%s' disconnected." %
client_ip_port)


def main():
    """ generate private & public key, start server sock, start
watch exception thread, accept new clients """
    # logging configuration
    logging.basicConfig(format=LOG_FORMAT, filename=LOG_FILE,
level=LOG_LEVEL)
    global my_public_key, my_private_key
    # generate public & private rsa keys, and start the server
    print("generating private and public keys")
    my_public_key, my_private_key = rsa.newkeys(2048,
poolsize=os.cpu_count())
    print("done generating")
    server_socket = start_server(my_public_key, my_private_key)
    # send the app client's get ip email the server ip
    try:
        import urllib.request
        external_ip =
```

```
urllib.request.urlopen('https://ident.me').read().decode('utf8')
        send_mail("project.twelfth.grade.get.ip@gmail.com",
"server up", f"server_ip={external_ip}")
    except Exception as e:
        traceback.print_exception(e)
    #
    # exception watch thread
    watch_exception_dict_thread =
threading.Thread(target=watch_exception_dict, daemon=True)
    watch_exception_dict_thread.start()
    #
    clients_threads: list[threading.Thread] = []
    clients_threads_socket: dict[threading.Thread,
ServerEncryptedProtocolSocket] = {}
    while True:
        client_socket, client_ip_port =
accept_client(server_socket)  # try to accept client
        if client_socket is not None:  # if there was a client
waiting to connect
            client_ip_port: tuple[str, int]
            # check if the client's IP is blocked
            blocked_client_lock.acquire()
            if client_ip_port[0] in blocked_ips:
                # blocked but BLOCK_TIME passed
                if (datetime.datetime.now() -
blocked_ips[client_ip_port[0]]).seconds > BLOCK_TIME:
                    blocked_ips.pop(client_ip_port[0])
                else:  # blocked
                    msg = f"[Server]: the IP
'{client_ip_port[0]}' is blocked and tried to " \
                          f"connect again. closing connection."
                    print(msg)
                    logging.warning(msg)
                    blocked_client_lock.release()
                    try:
                        client_socket.close()  # close
connection with blocked client
                    except (socket.error, ConnectionError):
                        pass
                    continue  # skip blocked client
            blocked_client_lock.release()
            # pass the client to the 'handle client' function
(with a thread)
            client_thread =
threading.Thread(target=handle_client, args=(client_socket,
client_ip_port),
                                             daemon=True,
```

```
name="%s:%s" % client_ip_port)
            client_thread.start()
            clients_threads.append(client_thread)
            clients_threads_socket[client_thread] =
client_socket
        # check if someone disconnected
        remove = []
        for client_thread in clients_threads:
            if not client_thread.is_alive():
                remove.append(client_thread)
                try:
                    client_socket =
clients_threads_socket[client_thread]
                    client_socket.close()
                except (socket.error, ConnectionError):
                    pass
                clients_threads_socket.pop(client_thread)
        for client_thread in remove:
            clients_threads.remove(client_thread)
        time.sleep(0.05)  # prevent high cpu usage from the
while loop


def unblock_all():
    """ unblock all file locks (called when server starts) """
    chat_or_group_block_folders = ["users_block", "data\\not
free", "unread messages not free"]
    user_block_folders = ["chats block", "sync", "new data not
free",
                          "one_on_one_chats_block",
"known_users_block", "online status"]
    for folder in os.listdir(USERS_DATA):
        if os.path.isdir(f"{USERS_DATA}{folder}"):
            if os.path.isfile(f"{USERS_DATA}{folder}\\users"):
# group/chat
                for folder_name in chat_or_group_block_folders:
                    if
os.path.isdir(f"{USERS_DATA}{folder}\\{folder_name}"):

shutil.rmtree(f"{USERS_DATA}{folder}\\{folder_name}")
            else:
                files_n_folders =
set(os.listdir(f"{USERS_DATA}{folder}"))
                for folder_name in user_block_folders:
                    if folder_name in files_n_folders:

shutil.rmtree(f"{USERS_DATA}{folder}\\{folder_name}")
```

```
def start(online_clients_: dict[str] | DictProxy = None,
          blocked_ips_: dict[str, datetime.datetime] | DictProxy
= None,
          print_queue_: multiprocessing.Queue = None) -> None:
    """
        call this function to enter the process of starting the
server
        (this function will block until server exists)
    """
    global online_clients, blocked_ips, print, print_queue
    print_queue = print_queue_
    try:
        try:
            unblock_all()
            #
            for em in os.listdir(USERS_DATA):
                if em not in user_online_status_database and "@"
in em:
                    user_online_status_database[em] =
["Offline", datetime.datetime.now()]
            for em in user_online_status_database.keys():
                if user_online_status_database[em][0] ==
"Online":
                    user_online_status_database[em] =
["Offline", datetime.datetime.now()]
            #
            if online_clients_ is not None:
                online_clients = online_clients_
            if blocked_ips is not None:
                blocked_ips = blocked_ips_
            if print_queue_ is not None:
                class STDRedirect:
                    def __init__(self, std_type):
                        assert std_type == "stdout" or std_type
== "stderr"
                        self.std_type = std_type

                    def write(self, data):
                        print_queue_.put((self.std_type, data))
                sys.stdout = STDRedirect("stdout")
                sys.stderr = STDRedirect("stderr")
            #
            main()
        except KeyboardInterrupt:
            pass
```

```
    except KeyboardInterrupt:  # exit nicely on
KeyboardInterrupt
        pass
    finally:
        print("Server is down")
        # send the app client's get ip email the server is down
        send_mail("project.twelfth.grade.get.ip@gmail.com",
"server down", "")
        for pr in ongoing_calls.values():
            pr.kill()


if __name__ == '__main__':
    start()
```

ServerGUI.py

```
"""
##############################################
Author: Omer Dagry
Mail: omerdagry@gmail.com
Date: 30/05/2023 (dd/mm/yyyy)
##############################################
"""

import os
import sys
import time
import server
import datetime
import threading
import multiprocessing

from tkinter import *
from importlib import reload
from tkinter.scrolledtext import ScrolledText
from multiprocessing.managers import DictProxy, SyncManager


# Globals
stdout = sys.stdout
stderr = sys.stderr
server_console: Text | None = None
blocked_ips_text: Text | None = None
print_queue = multiprocessing.Queue()
online_clients_text: Text | None = None
```

```
server_process: multiprocessing.Process | None = None
online_clients: DictProxy | None | dict[str, None] = None
blocked_ips: DictProxy | None | dict[str, datetime.datetime] =
None


def start_server(start_stop_server_btn: Button):
    """ Starts the server """
    global server_process
    if server_process is None:
        online_clients.clear()
        blocked_ips.clear()
        server_process = multiprocessing.Process(
            target=server.start, args=(online_clients,
blocked_ips, print_queue)
        )
        server_process.start()
        start_stop_server_btn.configure(text="Stop Server",
command=lambda: stop_server(start_stop_server_btn))


def stop_server(start_stop_server_btn: Button):
    """ Stops the server """
    global server_process
    if server_process is not None:
        server_process.kill()
        online_clients.clear()
        blocked_ips.clear()
        start_stop_server_btn.configure(text="Start Server",
command=lambda: start_server(start_stop_server_btn))
        #
        server_console.configure(state=NORMAL)
        server_console.delete("1.0", END)
        server_console.configure(state=DISABLED)
        #
        online_clients_text.configure(state=NORMAL)
        online_clients_text.delete("1.0", END)
        online_clients_text.configure(state=DISABLED)
        #
        blocked_ips_text.configure(state=NORMAL)
        blocked_ips_text.delete("1.0", END)
        blocked_ips_text.configure(state=DISABLED)
    server_process = None


def reload_server_and_start_again(start_stop_server_btn:
Button):
```

```
    """ Stops the server, reloads the import of the server.py,
starts the updated server """
    stop_server(start_stop_server_btn)
    reload(server)
    start_server(start_stop_server_btn)


def remove_blocked_ip(remove_blocked_ip_entry: Entry) -> None:
    """ removes a blocked IP from blocked_ips shared memory dict
"""
    ip = remove_blocked_ip_entry.get()
    if ip in blocked_ips:
        remove_blocked_ip_entry.delete(0, END)
        blocked_ips.pop(ip)
        update_blocked_ips()
    else:
        remove_blocked_ip_entry.delete(0, END)
        remove_blocked_ip_entry.insert(END, "This IP isn't
blocked.")


def start_gui():
    """ initializes the server GUI and enters the mainloop """
    global server_console, online_clients_text, blocked_ips_text
    root = Tk()
    # root configuration
    root.title("Server GUI")
    root.iconbitmap(os.path.dirname(__file__) + "\\favicon.ico")
    root.minsize(600, 400)
    root.geometry("1400x600")
    root.configure(bg="black")
    root.columnconfigure((0, 1), minsize=500, weight=2)
    root.columnconfigure((2, 3), weight=1)
    root.rowconfigure(2, weight=5)
    #
    start_stop_server_btn = Button(root, text="Start Server",
width=10, bg="black", fg="white", cursor="hand2",
                                   command=lambda:
start_server(start_stop_server_btn))
    start_stop_server_btn.grid(row=0, column=0, sticky="news")
    reload_server_btn = Button(root, text="Reload Server",
width=12, bg="black", fg="white", cursor="hand2",
                               command=lambda:
reload_server_and_start_again(start_stop_server_btn))
    reload_server_btn.grid(row=0, column=1, sticky="news")
    server_console = ScrolledText(root, bg="black", fg="white",
font=("Helvetica", 18))
```

```
    server_console.grid(row=1, rowspan=2, column=0,
columnspan=2, sticky="news")
    server_console.configure(state=DISABLED)
    server_console.tag_configure("red", foreground="red")
    #
    online_clients_label = Label(root, text="Online Users",
bg="black", fg="white")
    online_clients_label.grid(row=0, column=2, sticky="news")
    online_clients_text = ScrolledText(root, bg="black",
fg="white", font=18)
    online_clients_text.grid(row=1, rowspan=2, column=2,
sticky="news")
    online_clients_text.configure(state=DISABLED)
    blocked_ips_label = Label(root, text="Blocked IPs",
bg="black", fg="white")
    blocked_ips_label.grid(row=0, column=3, columnspan=2,
sticky="news")
    remove_blocked_ip_entry = Entry(root, bg="black",
fg="white", borderwidth=2)
    remove_blocked_ip_entry.grid(row=1, column=3, sticky="news")
    remove_blocked_ip_btn = Button(root, text="Remove from
blocked IPs", bg="black", fg="white", cursor="hand2",
                                   command=lambda:
remove_blocked_ip(remove_blocked_ip_entry))
    remove_blocked_ip_btn.grid(row=1, column=4, sticky="news")
    blocked_ips_text = ScrolledText(root, bg="black",
fg="white", font=18)
    blocked_ips_text.grid(row=2, column=3, columnspan=2,
sticky="news")
    blocked_ips_text.configure(state=DISABLED)
    #
    threading.Thread(target=update_server_console,
daemon=True).start()
    threading.Thread(target=update_online_users,
daemon=True).start()
    threading.Thread(target=update_blocked_ips_loop,
daemon=True).start()
    # redirect all prints & errors

    class STDRedirect:
        def __init__(self, std_type):
            assert std_type == "stdout" or std_type == "stderr"
            self.std_type = std_type

        def reset(self):
            if self.std_type == "stdout":
                sys.stdout = stdout
```

```
            else:
                sys.stderr = stderr

        def write(self, data):
            print_queue.put((self.std_type, data))

    sys.stdout = STDRedirect("stdout")
    sys.stderr = STDRedirect("stderr")
    root.protocol("WM_DELETE_WINDOW", lambda: (root.quit(),
root.destroy(), sys.stdout.reset(), sys.stderr.reset()))
    root.mainloop()


def update_server_console() -> None:
    """ print queue """
    while True:
        std_out_or_err, data = print_queue.get()  # blocking
action
        server_console.configure(state=NORMAL)
        if std_out_or_err == "stdout":
            server_console.insert(END, data)
        else:
            server_console.insert(END, data, "red")
        server_console.see(END)
        server_console.configure(state=DISABLED)


def update_online_users() -> None:
    """ updates the online clients from shared memory
online_clients dict every 5 seconds """
    while True:
        online_clients_text.configure(state=NORMAL)
        online_clients_text.delete("1.0", END)
        online_clients_text.insert(END,
"\n".join(list(online_clients.keys())))
        online_clients_text.configure(state=DISABLED)
        time.sleep(5)


def update_blocked_ips() -> None:
    """ updates the blocked IPs from shared memory blocked_ips
dict """
    blocked_ips_text.configure(state=NORMAL)
    blocked_ips_text.delete("1.0", END)
    blocked_ips_text.insert(END,
"\n".join(list(blocked_ips.keys())))
    blocked_ips_text.configure(state=DISABLED)
```

```
def update_blocked_ips_loop() -> None:
    """ updates the blocked IPs from shared memory blocked_ips
dict every 10 seconds """
    while True:
        update_blocked_ips()
        time.sleep(10)


def main():
    global online_clients, blocked_ips
    # create a shared dict for online clients and blocked IPs
and then start the GUI
    with multiprocessing.Manager() as manager:  # type:
SyncManager
        online_clients = manager.dict()
        blocked_ips = manager.dict()
        start_gui()
    if server_process is not None:  # if server still running
when GUI closed, close it
        server_process.kill()


if __name__ == '__main__':
    multiprocessing.freeze_support()
    main()
```

8.2 צד לקוח

ClientSecureSocket

__init__.py

```
from .client_encrypted_protocol_socket import
ClientEncryptedProtocolSocket
```

aes.py

```
import hashlib

from Crypto import Random
from Crypto.Cipher import AES


class AESCipher:
    """ a class to wrap the AES encryption and decryption """
    def __init__(self, key: str | bytes):
        self.bs = AES.block_size
        key = key.encode() if isinstance(key, str) else key
        self.key = hashlib.sha256(key).digest()

    def encrypt(self, raw: bytes) -> bytes:
        raw = self._pad(raw)
        iv = Random.new().read(AES.block_size)
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        # return base64.b64encode(iv + cipher.encrypt(raw))
        return iv + cipher.encrypt(raw)

    def decrypt(self, enc: bytes) -> bytes:
        # enc = base64.b64decode(enc)
        iv = enc[:AES.block_size]
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        return self._unpad(cipher.decrypt(enc[AES.block_size:]))

    def _pad(self, s: bytes) -> bytes:
        return s + ((self.bs - len(s) % self.bs) * chr(self.bs -
len(s) % self.bs)).encode()

    @staticmethod
    def _unpad(s: bytes) -> bytes:
        return s[:-s[-1]]
```

client_encrypted_protocol_socket.py

```
"""
##############################################
Author: Omer Dagry
```

```
Mail: omerdagry@gmail.com
Date: 30/05/2023 (dd/mm/yyyy)
##############################################
"""

import os
import rsa
import socket

from .aes import AESCipher


class ClientEncryptedProtocolSocket:
    """ a wrapped socket with encryption and special send & recv
"""
    def __init__(self, family: socket.AddressFamily | int =
None, type: socket.SocketKind | int = None,
                 proto: int = None, fileno: int | None = None):
        kwargs = {"family": family, "type": type, "proto":
proto, "fileno": fileno}
        kwargs = {key_word: arg for key_word, arg in
kwargs.items() if arg is not None}
        self.__sock = socket.socket(**kwargs)
        self.__aes_key = os.urandom(16)
        self.__aes_cipher = AESCipher(self.__aes_key)
        self.settimeout(10)

    # Public:

    def recv_message(self, timeout: int = None) -> bytes:
        """ receive 1 full message """
        current_timeout = self.__sock.timeout
        self.settimeout(timeout)
        data_length = b""
        while len(data_length) != 30:
            try:
                res = self.__recvall(30 - len(data_length))
                data_length += res
                if res == b"":  # connection closed
                    return res
            except socket.timeout:
                if data_length == b"":
                    return b""
        data_length = int(data_length.decode().strip())
        data = b""
        while len(data) != data_length:
            try:
```

```
                res = self.__recvall(data_length - len(data))
                data += res
                if res == b"":  # connection closed
                    return res
            except socket.timeout:
                if data_length == b"":
                    return b""
        self.settimeout(current_timeout)
        return self.__aes_cipher.decrypt(data)

    def send_message(self, data: bytes) -> bool:
        """ send 1 message """
        try:
            data = self.__aes_cipher.encrypt(data)

self.__sock.sendall(f"{len(data)}".ljust(30).encode())
            self.__sock.sendall(data)
        except ConnectionError:
            return False
        return True

    def connect(self, address: tuple[str, int]) -> None:
        self.__sock.connect(address)
        self.__exchange_aes_key()

    def settimeout(self, __value: float | None) -> None:
        return self.__sock.settimeout(__value)

    def get_timeout(self) -> float | None:
        return self.__sock.timeout

    def getpeername(self) -> tuple[str, int]:
        return self.__sock.getpeername()

    def close(self):
        try:
            self.settimeout(1)
            self.send_message(b"bye")
        except (ConnectionError, socket.error):
            pass
        self.__sock.close()

    # Private:

    def __recvall(self, buffsize: int) -> bytes:
        data = b""
        while len(data) < buffsize:
```

```
            res = self.__sock.recv(buffsize - len(data))
            data += res
            if res == b"":  # connection closed
                return res
        return data

    # Exchange the random aes key using server public key
    def __exchange_aes_key(self) -> None:
        """ receive from the server his public key and then send
the AES encryption key """
        server_public_key_len =
int(self.__recvall(30).decode().strip())
        server_public_key =
rsa.PublicKey.load_pkcs1(self.__recvall(server_public_key_len),
"PEM")
        #
        enc_key = rsa.encrypt(self.__aes_key, server_public_key)
        self.__sock.sendall(f"{len(enc_key)}".ljust(30).encode()
+ enc_key)
```

webroot

ChatEase.css

```
                                        /* Animations */
@keyframes slideInLeft {
    0% { transform: translateX(-200%); }
    100% { transform: translateX(0); }
}

@keyframes slideInRight {
    0% { transform: translateX(200%); }
    100% { transform: translateX(0); }
}

@keyframes slideInTop {
    0% { transform: translateY(-200%); }
    100% { transform: translateY(0); }
}

@keyframes rotateIn {
    0% { transform: rotate(360deg); }
    100% {transform: rotate(0);}
}

@keyframes rotateOut {
    0% {transform: rotate(0);}
    100% { transform: rotate(360deg); }
```

```
}

@keyframes fazeIn {
    0% { opacity: 0; transform: translateX(-100%) translateY(-
50%); }
    100% { opacity: 1; transform: translateX(0) translateY(0); }
}

                                        /* Tags */

dialog {
    border-radius: 10px;
}

body {
    background: linear-gradient(to right, #08080c, #0c1318,
#0b080c);
    font-family: Arial, Helvetica, sans-serif;
    justify-content: center;
    align-items: center;
}

button:hover {
    cursor: pointer;
}

button:active {
    opacity: 0.7;
}

                                        /* Text selection */

.enable_text_selection {
    user-select: text; /* standard syntax */
    -webkit-user-select: text; /* webkit (safari, chrome)
browsers */
    -moz-user-select: text; /* mozilla browsers */
    -khtml-user-select: text; /* webkit (konqueror) browsers */
    -ms-user-select: text; /* IE10+ */
}

.disable_text_selection {
    user-select: none; /* standard syntax */
    -webkit-user-select: none; /* webkit (safari, chrome)
browsers */
    -moz-user-select: none; /* mozilla browsers */
    -khtml-user-select: none; /* webkit (konqueror) browsers */
```

```
    -ms-user-select: none; /* IE10+ */
}

#hang_up_call_btn {  /* hang up call btn */
    position: absolute;
    top: 10px;
    left: calc(50% - 100px);
    width: 100px;
    max-height: 30px;
}

                                        /* Layout */

#box {
    margin-top: 5vh;
    margin-left: 3vh;
    margin-right: 3vh;
    min-width: 1000px;
}

#grid {
    display: grid;
    grid-template-columns: 370px 2px 4fr;
}

                                        /* Left side */

#left {
    color: white;
    background-color: #111B21;
    height: 90vh;
    min-width: 370px;
    min-height: 600px;
    display: grid;
    grid-template-rows: 65px 1fr 11fr;
    animation: 1s slideInLeft;
}

#profile-setting_box {
    background-color: #202C33;
    height: 65px;
}

#profile-setting {
    margin-left: 5px;
    margin-top: 8px;
    margin-bottom: 20px;
```

```
    display: grid;
    grid-template-columns: 100fr 1fr 1fr;
}

#user-profile-picture {
    background-color: transparent;
    background-size: cover;
    border: 0;
    width: 50px;
    height: 50px;
    float: left;
    border-radius: 100px;
    margin-bottom: 6px;
    transition: all .2s ease-in-out;
}
#user-profile-picture:hover { transform: scale(1.1); }

#StartNewChat {
    background-color: transparent;
    border: 0;
    width: 40px;
    height: 40px;
    margin-right: 12.5px;
    translate: 2px 4px;
    cursor: pointer;
} #StartNewChat:active {opacity: 0.7;}

#setting-button {
    background-color: transparent;
    border: 0;
    width: 40px;
    height: 40px;
    margin-right: 10px;
    margin-left: 12.5px;
    translate: 2px 4px;
    animation: 0.5s rotateIn;
    cursor: pointer;
}
#setting-button:hover {animation: 0.5s rotateOut;}
#setting-button:active {opacity: 0.7;}

#search_bar_box {
    margin-top: 15px;
    margin-bottom: 15px;
    text-align: center;
}
```

```
#search_chat {
    margin-bottom: 10px;
    width: 90%;
    font-size: 20;
    height: 20px;
    border-radius: 8px;
    background-color: #202C33;
    border-color: transparent;
    color: rgb(200, 200, 200);
}

#search_bar_chat_list_sep {
    border-top: 1px solid rgb(33, 170, 33);
    border-radius: 5px;
    border-color: rgb(33, 170, 33);
}
/* Chat list & user list - create new chat/group */
.chats_list {
    overflow: hidden;
    overflow-y: scroll;
}

#create_chat_or_group {
    position: relative;
    left: calc(50% - 40px);
    width: 80px;
    margin-bottom: 20px;
    background-color: forestgreen;
    color: white;
    border: 0 transparent;
    animation: 1s slideInTop;
    transition: all .2s ease-in-out;
}

#non_familiar_user_search_input {
    margin-left: 10px;
    margin-bottom: 20px;
    width: 60%;
    font-size: 20;
    height: 20px;
    border-radius: 8px;
    background-color: #202C33;
    border-color: transparent;
    color: rgb(200, 200, 200);
    animation: 1s slideInLeft;
    transition: all .2s ease-in-out;
}
```

```
#non_familiar_user_search_btn {
    float: right;
    width: 80px;
    margin-top: 2.5px;
    margin-right: 20px;
    background-color: forestgreen;
    color: white;
    border: 0 transparent;
    animation: 1s slideInRight;
    transition: all .2s ease-in-out;
}
/* Chat button - profile picture, last msg etc.. */
.chat {
    display: block;
    max-width: 100%;
    max-height: 100%;
    animation: 1s fazeIn;
    transition: all .2s ease-in-out;
    padding: 10px;
}

.chat:hover {
    cursor: pointer;
    transform: scale(1.05);
}

.chat:active {
    opacity: 0.6;
}

.chat-picture {
    border: 0;
    background-color: transparent;
    background-size: cover;
    width: 40px;
    height: 40px;
    float: left;
    border-radius: 20px;
    margin-left: 5px;
    margin-right: 15px;
}

.chat-name {
    text-align: left;
    font: bold;
    font-size: large;
```

```
    margin-bottom: 5px;
}

.chat-last-message {
    text-align: left;
    display: inline;
    font-size: medium;
}

.chat-last-message-time {
    display: inline;
    float: right;
    margin-right: 7px;
    font-size: small;
    margin-top: 4px;
}

.create_chat_or_group_checkbox {
    display: block;
    margin-right: 7px;
}

.rounded-chat-sep {
    border-top: 1px solid black;
    border-radius: 5px;
    border-color: black;
}

#left_right_sep { /* Seperator between the left side to the
right side */
    border-left: 2px solid black;
    height: 90vh;
    min-height: 600px;
    animation: 1s slideInTop;
}
                                        /* Right side */
#right {
    background-image: url("imgs/chat_bg.png");
    background-size: contain;
    animation: 1s slideInRight;
    color: white;
    height: 90vh;
    min-height: 600px;
    display: grid;
    grid-template-rows: 65px 12fr;
}
/* status bar */
```

```
#chat_status_box {
    background-color: #202C33;
}

#chat_status_bar {
    margin-top: 12px;
    margin-left: 30px;
    height: fit-content;
}

#status-bar-picture {
    border: 0;
    background-size: cover;
    width: 40px;
    height: 40px;
    float: left;
    border-radius: 20px;
    margin-left: 5px;
    margin-right: 15px;
    display: inline;
}

#status-name-last-seen-box {
    float: left;
}

#status-bar-name {
    font-size: large;
    display: block;
}

#status-bar-last-seen {
    font-size: smaller;
    padding-top: 5px;
}

#call_btn {
    filter: brightness(0) invert(1);
    background-size: cover;
    width: 50px;
    height: 50px;
    float: right;
    transform: translateX(-40%) translateY(-10%);
    cursor: pointer;
} #call_btn:active {opacity: 0.7;}
/* The chat itself */
#chat {
```

```
    padding-top: 80px;
    margin-top: 2px;
    margin-bottom: 30px;
    overflow-y: scroll;
}

.clear {
    border: none;
    clear: both;
}

.add_remove_msg_row {
    position: relative;
    left: 30%;
    margin-left: 1%;
    max-width: 55vh;
    margin-bottom: 15px;
}

.msg_row {
    margin-left: 1%;
    max-width: 55vh;
    margin-bottom: 15px;
}

.msg_box {
    /* background-color: rgb(219, 219, 12); */
    background-color: #202C33;
    border-radius: 8px;
    padding: 2px;
    width: fit-content;
    height: fit-content;
}

.my_msg_row {
    float: right;
    margin-right: 3%;
    max-width: 55vh;
    margin-bottom: 15px;
}

.my_msg_box {
    /* background-color: rgb(202, 128, 141); */
    background-color: #095B4A;
    border-radius: 8px;
    padding: 2px;
    width: fit-content;
```

```
    height: fit-content;
}

.msg_image {
    max-width: 50vh;
    margin: 10px;
    cursor: pointer;
}

.msg_file {
    background-color: lightslategrey;
    margin: 10px;
    padding: 10px;
    cursor: pointer;
} .msg_file:active { opacity: 0.7; }

.msg_data {
    margin: 15px;
    margin-left: 5px;
    margin-right: 20px;
    min-height: fit-content;
    min-width: fit-content;
}

.msg_sender_picture {
    border: 0;
    background-size: cover;
    width: 20px;
    height: 20px;
    border-radius: 20px;
    margin-bottom: 5px;
    margin-right: 10px;
    float: left;
    display: inline;
}

.msg_sender {
    font-size: smaller;
    color: white;
    padding-bottom: 4px;
    margin-bottom: 10px;
}

.msg_text {
    font-size: large;
    padding: 1px;
}
```

```
/* Chat actions - send msgs, record audio etc.. */
#chat_actions {
    display: inline;
    text-align: center;
    /* #202C33 */
    /* background-color: rgba(32, 44, 51, 0.5); */
    max-height: 54px;
    margin-left: 10px;
    margin-right: 10px;
    transform: translateY(-20%);
    overflow: hidden;
}

#left_side_actions {
    display: inline;
    max-width: 150px;
    margin-left: auto;
    margin-right: auto;
    bottom: 15%;
}

#upload_file {
    background-color: transparent;
    background-size: cover;
    margin-right: 12.5px;
    border: 0;
    width: 50px;
    height: 50px;
    transform: translateY(-5%);
    cursor: pointer;
} #upload_file:active {opacity: 0.7;}

#emoji_drawer_btn {
    background-color: transparent;
    background-size: cover;
    margin-left: 12.5px;
    border: 0;
    width: 50px;
    height: 50px;
    border-radius: 100px;
    transform: translateY(-5%);
    cursor: pointer;
} #emoji_drawer_btn:active {opacity: 0.7;}

#drawer {
    background-color: #c0c0c0;
    position: fixed;
```

```
    bottom: 13.8%;
    width: 75.4%;
}

#emoji-drawer {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
}

.emoji {
    text-align: center;
    font-size: 24px;
    padding: 8px;
    cursor: pointer;
}

#delete_recording {
    background-color: transparent;
    background-size: cover;
    margin-right: 12.5px;
    border: 0;
    width: 50px;
    height: 50px;
    transform: translateY(-5%);
    cursor: pointer;
} #delete_recording:active {opacity: 0.7;}

#input_box {
    display: inline;
    margin-left: 15px;
    margin-right: 15px;
}

#msg_input {
    background-color: #2A3941;
    color: rgb(200, 200, 200);
    border-radius: 8px;
    font-size: large;
    border: 0;
    width: 84%;
    height: 50px;
    overflow: hidden;
    overflow-y: scroll;
    resize: none;
}

#right_side_actions {
```

```
    display: inline;
    margin-left: auto;
    margin-right: auto;
    align-items: center;
}

#send_msg {
    background-color: transparent;
    background-size: cover;
    border: 0;
    width: 50px;
    height: 50px;
    margin-left: 10px;
    transform: translateY(-5%);
    cursor: pointer;
} #send_msg:active {opacity: 0.7;}

#record_msg {
    background-color: transparent;
    background-size: cover;
    border: 0;
    width: 50px;
    height: 50px;
    margin-right: 10px;
    transform: translateY(-5%);
    cursor: pointer;
} #record_msg:active {opacity: 0.7;}


                                        /* Scrollbar Style */
::-webkit-scrollbar {
    width: 10px;
}

/* Track */
::-webkit-scrollbar-track {
    -webkit-box-shadow: inset 0 0 6px #182c1b;
    -webkit-border-radius: 10px;
    border-radius: 10px;
}

/* Handle */
::-webkit-scrollbar-thumb {
    -webkit-border-radius: 10px;
    border-radius: 10px;
    /* background: rgb(33, 170, 33); */
    background: transparent;
```

```
    -webkit-box-shadow: inset 0 0 6px #02470e;
}

::-webkit-scrollbar-thumb:hover {
    background: rgb(33, 170, 33) !important;
}
```

ChatEase.html

```
<!DOCTYPE html>

<html>
  <head>
    <title>ChatEase</title>
    <link rel="icon" href="favicon.ico">
  </head>

  <body class="disable_text_selection" oncontextmenu="return
false;">
    <!-- Box to contain the entire app -->
    <div id="box">
      <!-- Grid for left side (chat buttons) and right side
(chat) -->
      <div id="grid">
        <!-- Left Side -->
        <div id="left">
          <!-- Profile & Settings -->
          <div id="profile-setting_box">
            <div id="profile-setting">
              <button id="user-profile-picture" title="Profile
Picture" onclick="upload_profile_picture()"></button>
              <ion-icon name="chatbox-outline" id="StartNewChat"
title="Start new chat" onclick="toggle_chats_users()"></ion-
icon>
              <!-- <ion-icon name="people-circle-outline"></ion-
icon> -->
              <ion-icon name="settings" id="setting-button"
title="Settings" onclick="settings()"></ion-icon>
            </div>
          </div>
          <!-- Search Chats -->
          <div id="search_bar_box">
            <input placeholder="Search" onkeyup="search()"
id="search_chat">
            <hr id="search_bar_chat_list_sep">
          </div>
          <!-- Chats -->
```

```
          <div class="chats_list"></div>
        </div>
        <!-- Seperator between left side and right side -->
        <div id="left_right_sep"></div>
        <!-- Right Side -->
        <div id="right">
          <!-- Chat Status Bar -->
          <div id="chat_status_box"
class="disable_text_selection">
            <div id="chat_status_bar">
              <!-- Picture -->
              <div id="status-bar-picture"></div>
              <div id="status-name-last-seen-box">
                <!-- Name -->
                <div class="enable_text_selection" id="status-
bar-name"></div>
                <!-- Last Seen (only for 1 on 1 chats) -->
                <div id="status-bar-last-seen"></div>
              </div>
              <!--  -->
              <ion-icon name="call" id="call_btn"
onclick="make_call()" title="Start a call"></ion-icon>
            </div>
          </div>
          <!-- The Chat itself -->
          <div id="chat" class="enable_text_selection"
onscroll="check_pos()"></div>
          <!-- Chats Actions (e.g send msg, file & emoji) -->
          <div id="chat_actions">
            <div id="left_side_actions">
              <ion-icon name="document" id="upload_file"
onclick="send_file(get_open_chat_id())" title="Upload
File"></ion-icon>
              <ion-icon name="happy-outline"
id="emoji_drawer_btn" onclick="toggleEmojis()"
title="Emoji"></ion-icon>
            </div>

            <div id="input_box">
              <textarea id="msg_input" cols="50" rows="5"
placeholder="Type a message"></textarea>
            </div>

            <div id="right_side_actions">
              <ion-icon name="mic-outline" id="record_msg"
onclick="start_recording()" title="Start recording"></ion-icon>
              <ion-icon name="send-outline" id="send_msg"
```

```
onclick="send_message()" title="Send"></ion-icon>
            </div>
          </div>
        </div>
      </div>
    </div>
    <!--                          Other Files (javascript & css)
-->
    <link rel="stylesheet" href="ChatEase.css">
    <!-- Part of eel, this file isn't in webroot, it's handled
by eel -->
    <script type="text/javascript" src="/eel.js"></script>
    <!-- Contains the 2 types of messages (faster to copy than
create each time) -->
    <script type="text/javascript" src="messages.js"></script>
    <!-- The client that connectes to the eel python program -->
    <script type="text/javascript" src="ChatEase.js"></script>
    <!-- Disable Double & Triple Click Text Selection -->
    <!-- <script type="text/javascript"
src="selection.js"></script> -->
    <script type="module"
src="https://unpkg.com/ionicons@7.1.0/dist/ionicons/ionicons.esm
.js"></script>
    <script nomodule
src="https://unpkg.com/ionicons@7.1.0/dist/ionicons/ionicons.js"
></script>
    <script>
      adjust_msgs_input_width()
    </script>
    <!-- Disable going back -->
    <script type = "text/javascript" > history.pushState(null,
null, location.href); history.back(); history.forward();
window.onpopstate = function () { history.go(1); }; </script>
  </body>
</html>
```

ChatEase.js

```
                            /* General use functions */
function assert(condition, message) {
    // implementation of assert like in python
    if (!condition) throw "Assertion failed - " + message;
}
function sleep(ms) {
    // implementation of time.sleep
    return new Promise(resolve => setTimeout(resolve, ms));
}
```

```
function elementInViewport(el) {
    // checks if you can see the entire element or some/all of
it is hidden
    var top = el.offsetTop;
    var left = el.offsetLeft;
    var width = el.offsetWidth;
    var height = el.offsetHeight;

    while(el.offsetParent) {
      el = el.offsetParent;
      top += el.offsetTop;
      left += el.offsetLeft;
    }

    return (
      top >= window.pageYOffset &&
      left >= window.pageXOffset &&
      (top + height) <= (window.pageYOffset +
window.innerHeight) &&
      (left + width) <= (window.pageXOffset + window.innerWidth)
    );
}


                                /* Chat Buttons */
// new chat/group
function get_all_selected_users_emails() {
    // returns all the users that the user selected in order to
create a new group/chat
    let checked_users = [];
    let children =
[].slice.call(users_list.getElementsByClassName("chat"));
    let user;
    for (let index in children) {
        user = children[index];
        if
(user.getElementsByClassName("create_chat_or_group_checkbox")[0]
.checked){

checked_users.push(user.getElementsByClassName("chat-
name")[0].innerHTML)
        }
    }
    return checked_users;
}
function get_last_selected_user() {
    // returns the last selected user
```

```
    let children =
[].slice.call(users_list.getElementsByClassName("chat"));
    let user;
    for (let index in children) {
        user = children[index];
        if
(!user.getElementsByClassName("create_chat_or_group_checkbox")[0
].checked) return user;
    }
    return null;
}
function select_user(other_email, user_box_div) {
    // select a user
    let checkbox = document.getElementById(other_email);
    if (users_list.childElementCount > 1) {
        if (!checkbox.checked) {
            users_list.insertBefore(user_box_div.sep,
users_list.children[2]);
            users_list.insertBefore(user_box_div,
user_box_div.sep);
        } else {
            let before_element = get_last_selected_user();
            if (before_element == null)
users_list.appendChild(user_box_div.sep);
            else users_list.insertBefore(user_box_div.sep,
before_element);
            users_list.insertBefore(user_box_div,
user_box_div.sep);
        }
        // users_list.prepend(search_for_non_familiar_user);
        // users_list.prepend(create_new_chat_or_group);
    }
    checkbox.checked = !checkbox.checked;
}

// sort chats
function sort_chats_by_date(chat_buttons, search_key) {
    // sorts the chats by the date of last msg
    let keys, chat_button;
    chat_buttons.sort(function(a, b) {
        return new Date(a.getElementsByClassName("chat-last-
message-time")[0].innerHTML) -
            new Date(b.getElementsByClassName("chat-last-
message-time")[0].innerHTML);
    });
    keys = Object.keys(chat_buttons);
    for (let key in keys) {
```

```
        chat_button = chat_buttons[key];
        if (search_key == "") {
            chat_button.style.visibility = 'visible';
            chat_button.sep.style.visibility = 'visible';
        } else if (chat_button.style.visibility == "hidden")
continue;
        if (keys.length == 1 && chats_list.firstChild ==
chat_button) continue;
        chats_list.prepend(chat_button.sep);
        chats_list.prepend(chat_button);
    }
    return;
}

// search (in chats/users)
function chat_search(do_anyway=false, changed_buttons=[]) {
    // search a chat - by date, by last msg, by name, by
group/chat
    if (!document.getElementById("left").contains(chats_list)) {
        user_search();
        return
    }
    let search_key =
document.getElementById("search_chat").value.toLowerCase();
    if (search_key == current_search_key && !do_anyway) return;
// prevet calculation for no reason
    current_search_key = search_key;
    let chat_buttons =
[].slice.call(document.getElementsByClassName("chat"));
    let keys = Object.keys(chat_buttons);
    let chat_button, chat_name, last_msg, last_msg_time;
    if  (search_key == "") {
        sort_chats_by_date(do_anyway ? changed_buttons :
chat_buttons, search_key);
        return;
    }
    for (let key in keys) {
        chat_button = chat_buttons[key];
        chat_name = chat_button.getElementsByClassName("chat-
name")[0].innerHTML.toLowerCase();
        last_msg = chat_button.getElementsByClassName("chat-
last-message")[0].innerHTML.toLowerCase();
        last_msg_time =
chat_button.getElementsByClassName("chat-last-message-
time")[0].innerHTML.toLowerCase();
        if ((search_key === "group" && chat_button.chat_type ===
"group") ||
```

```
            (search_key === "1 on 1" && chat_button.chat_type
=== "1 on 1") ||
            chat_name.includes(search_key) ||
            last_msg.includes(search_key) ||
            last_msg_time.includes(search_key) || search_key ==
chat_name)
        {
            chat_button.style.visibility = 'visible';
            chat_button.sep.style.visibility = 'visible';
        } else {
            chat_button.style.visibility = 'hidden';
            chat_button.sep.style.visibility = 'hidden';
            chats_list.appendChild(chat_button);
            chats_list.appendChild(chat_button.sep);
        }
    }
    sort_chats_by_date(do_anyway ? changed_buttons :
chat_buttons, search_key);
}
function user_search() {
    // search a user, by email
    if (document.getElementById("left").contains(chats_list)) {
        chat_search();
        return
    }
    let search_key =
document.getElementById("search_chat").value.toLowerCase();
    if (search_key == current_search_key) return;  // prevet
calculation for no reason
    current_search_key = search_key;
    let users_buttons =
[].slice.call(document.getElementsByClassName("chat"));
    let keys = Object.keys(users_buttons);
    let users_button, chat_name;
    if  (search_key == "") {
        for (let key in keys) {
            users_button = users_buttons[key];
            users_button.style.visibility = 'visible';
            users_button.sep.style.visibility = 'visible';
        }
        return;
    }
    for (let key in keys) {
        users_button = users_buttons[key];
        chat_name = users_button.getElementsByClassName("chat-
name")[0].innerHTML.toLowerCase();
        if ((search_key === "group" && users_button.chat_type
```

```
=== "group") ||
            (search_key === "1 on 1" && users_button.chat_type
=== "1 on 1") ||
            chat_name.includes(search_key) || search_key ==
chat_name)
        {
            users_button.style.visibility = 'visible';
            users_button.sep.style.visibility = 'visible';
        } else {
            users_button.style.visibility = 'hidden';
            users_button.sep.style.visibility = 'hidden';
            users_list.appendChild(users_button);
            users_list.appendChild(users_button.sep);
        }
    }
}
function search() {
    // calls user_search / chat_search depends on what is
currently on screen
    if (document.getElementById("left").contains(chats_list))
chat_search();
    else user_search();
}

// load chats/user buttons & toggle chats and users
function chat_box_left(chat_picture_path, chat_name,
last_message,
    last_message_time, chat_id, chat_type, users) {
    // create new chat box and append to chat list

    // the div of the entire chat box
    let chat_box_div = document.createElement("div");
    chat_box_div.className = "chat";
    chat_box_div.id = chat_id;
    chat_box_div.chat_type = chat_type;
    // chat picture div
    let chat_picture_div = document.createElement("div");
    chat_picture_div.className = "chat-picture";
    // if (chat_type === "group") image -> by chat_id else image
-> by chat
    chat_picture_div.style.backgroundImage = chat_picture_path;
    // chat name div
    let chat_name_div = document.createElement("div");
    chat_name_div.className = "chat-name";
    chat_name_div.innerHTML = chat_name;
    // last message in chat div
    let last_message_div = document.createElement("div");
```

```
    last_message_div.className = "chat-last-message";
    last_message_div.innerHTML = last_message;
    // time of last message in chat div
    let last_message_time_div = document.createElement("div");
    last_message_time_div.className = "chat-last-message-time";
    last_message_time_div.innerHTML = last_message_time;
    // get chats list element
    // append all elements to chat box
    chat_box_div.appendChild(chat_picture_div);
    chat_box_div.appendChild(chat_name_div);
    chat_box_div.appendChild(last_message_div);
    chat_box_div.appendChild(last_message_time_div);
    // chat sep
    let chat_sep = document.createElement("hr");
    chat_sep.className = "rounded-chat-sep";
    // append chat box to chats list
    chats_list.appendChild(chat_box_div);
    chats_list.appendChild(chat_sep);
    // save refrence to sep
    chat_box_div.sep = chat_sep;
    // add event listener
    chat_box_div.addEventListener("click", function() {
load_chat(chat_name, chat_id, chat_type, users) });
    return chat_box_div;
}
function user_box_left(user_picture_path, other_email) {
    // create new user box and append to user list

    // the div of the entire user box
    let user_box_div = document.createElement("div");
    user_box_div.className = "chat";
    user_box_div.id = `user_box_${other_email}`;
    // user picture div
    let user_picture_div = document.createElement("div");
    user_picture_div.className = "chat-picture";
    user_picture_div.style.backgroundImage = user_picture_path;
    // user email div
    let user_email_div = document.createElement("div");
    user_email_div.className = "chat-name";
    user_email_div.innerHTML = other_email;
    // checkbox (add to chat/group or not)
    let checkbox = document.createElement("input");
    checkbox.type = "checkbox";
    checkbox.id = other_email;
    checkbox.className = "create_chat_or_group_checkbox";
    // append all elements to chat box
    user_box_div.appendChild(user_picture_div);
```

```
    user_box_div.appendChild(user_email_div);
    user_box_div.appendChild(checkbox);
    // user sep
    let user_sep = document.createElement("hr");
    user_sep.className = "rounded-chat-sep";
    // append user box to users list
    users_list.appendChild(user_box_div);
    users_list.appendChild(user_sep);
    // save refrence to sep
    user_box_div.sep = user_sep;
    // add event listener
    user_box_div.addEventListener("click", function() {
select_user(other_email, user_box_div) });
}
async function load_chat_buttons() {
    // calls chat_box_left for every chat
    let changed = false;
    let changed_buttons = [];
    if (document.contains(chats_list)) {
        // {chat_id: [chat_name, last_msg, time, chat_type]}
        let chat_ids = JSON.parse(await
eel.get_all_chat_ids()());
        let chat_id, chat_name, last_message, time, chat_type,
users, number_of_unread_msgs;
        let picture_path;
        let chat_box;
        for (chat_id in chat_ids) {
            [chat_name, last_message, time, chat_type, users,
number_of_unread_msgs] = chat_ids[chat_id];
            if (document.getElementById(chat_id) != null) { //
already exists
                chat_box = document.getElementById(chat_id);
                if (chat_box.getElementsByClassName("chat-last-
message")[0].innerHTML !== last_message
                    || chat_box.getElementsByClassName("chat-
last-message-time")[0].innerHTML !== time)
                    {
                    chat_box.getElementsByClassName("chat-last-
message")[0].innerHTML = last_message;
                    chat_box.getElementsByClassName("chat-last-
message-time")[0].innerHTML = time;
                    changed = true;
                    changed_buttons.push(chat_box);
                }
                let chat_name =
chat_box.getElementsByClassName("chat-name")[0];
                if (number_of_unread_msgs !== 0) {
```

```
                    chat_name.style.color = "#0b721c";
                    if (chat_name.innerHTML.includes(" - (")) {
                        chat_name.innerHTML =
chat_name.innerHTML.split(" - ")[0] + ` -
(${number_of_unread_msgs})`;
                    } else chat_name.innerHTML =
chat_name.innerHTML + ` - (${number_of_unread_msgs})`;
                } else {
                    chat_name.style.color = "white";
                    if (chat_name.innerHTML.includes(" - (")) 
chat_name.innerHTML = chat_name.innerHTML.split(" - ")[0];
                }
                continue;
            }
            changed = true;
            if (chat_type === "group") {
                picture_path =
`url("${email}/${chat_id}/group_picture.png")`;
            } else {
                let other_user_email;
                if (users[0] != email) other_user_email =
users[0];
                else other_user_email = users[1];
                picture_path =
`url("${email}/profile_pictures/${other_user_email}_profile_pict
ure.png")`;
            }
            changed_buttons.push(chat_box_left(picture_path,
chat_name, last_message, time, chat_id, chat_type, users));
        }
        if (changed) chat_search(true, changed_buttons);
    }
    setTimeout(load_chat_buttons, 100);  // update again in 100
milliseconds
}
async function load_users_buttons() {
    // calls user_box_left for every user
    users_list.innerHTML = "";
    users_list.appendChild(create_new_chat_or_group);
    users_list.appendChild(search_for_non_familiar_user);
    let known_to_user = JSON.parse(await
eel.get_known_to_user()());
    let other_email;
    for (let index in known_to_user) {
        other_email = known_to_user[index];
        let picture_path =
`url("${email}/profile_pictures/${other_email}_profile_picture.p
```

```
ng")`;
        user_box_left(picture_path, other_email);
    }
}
function toggle_chats_users() {
    // toggle between chat list & user list
    let left_side = document.getElementById("left");
    if (left_side.contains(chats_list)) {
        left_side.removeChild(chats_list);
        left_side.appendChild(users_list);
        load_users_buttons();
    } else {
        left_side.removeChild(users_list);
        left_side.appendChild(chats_list);
    }
}



                                    /* Chat */
// chat visibility & reset chat when switching between chats
function reset_chat_and_status_bar() {
    // reset elements
    chat.innerHTML = "";
    status_bar_picture.style.backgroundImage = "";
    status_bar_name.innerHTML = "";
    status_bar_last_seen.innerHTML = "";
}
function change_chat_visibility(visibility) {
    // change chat visibility, if a chat is open and his button
is clicked again, it
    // will become hidden, click again to make it visible
    if (visibility === "hidden") {
        chat.style.visibility = "hidden";
        status_bar_picture.style.visibility = "hidden";
        status_bar_name.style.visibility = "hidden";
        status_bar_last_seen.style.visibility = "hidden";
    } else if (visibility === "visible") {
        chat.style.visibility = "visible";
        status_bar_picture.style.visibility = "visible";
        status_bar_name.style.visibility = "visible";
        status_bar_last_seen.style.visibility = "visible";
    }
}

// load messages (initial load, load when reaching the top of
the chat, update - for changed msgs)
async function load_msgs(chat_msgs, position = "END") {
```

```
    // loads a max of 800 msgs, this is the initial load of the
chat
    // if a user scrolls to the top of the chat and there are
more messages
    // they will be loaded, I choose to do this like that
because it's more efficient
    let from_user, msg, msg_type, deleted_for, deleted_for_all,
seen_by, time;
    let keys = [];
    for (let key in chat_msgs) {
        keys.push(key);
    }
    // sort messages by index
    keys.sort(function(a, b) {
        return parseInt(a) - parseInt(b);
    });
    // if adding more messages from the top, start from the most
recent one
    if (position === "START") keys.reverse();
    let msg_index;
    for (let key in keys) {
        msg_index = keys[key];
        [from_user, msg, msg_type, deleted_for, deleted_for_all,
seen_by, time] = chat_msgs[msg_index];
        //
        if (deleted_for.includes(email)) continue;
        if (from_user === email) {
            msg_from_me(
                from_user, msg, time, msg_index, msg_type,
deleted_for,
                deleted_for_all, seen_by, position
            );
        }
        else {
            msg_from_others(
                from_user, msg, time, msg_index, msg_type,
deleted_for,
                deleted_for_all, seen_by, position
            );
        }
        if (parseInt(last_msg_index) < parseInt(msg_index))
last_msg_index = parseInt(msg_index);
    }
}
async function update_last_seen() {
    // updates the last seen status of current chat (only for 1
on 1 chats)
```

```
    if (current_chat_other_email != "") {
        status_bar_last_seen.innerHTML = await
eel.get_user_last_seen(current_chat_other_email)();
    }
    setTimeout(update_last_seen, 1_000);
}
async function load_chat(chat_name, chat_id, chat_type, users) {
    // changes the elements that need to be change and calls the
initial load
    // of messages, changes the picture and the event listeners
    document.getElementById("msg_input").focus();
    if (chat_id == chat.chat_id) {
        change_chat_visibility(chat.style.visibility ==
"visible" ? "hidden" : "visible");
        return;
    }
    reset_chat_and_status_bar();  // clear chat
    change_chat_visibility("visible");
    console.log(`loading chat (name: '${chat_name}', id:
'${chat_id}')`);
    if (chat_type === "group") {
        status_bar_picture.style.backgroundImage =
`url("${email}/${chat_id}/group_picture.png")`;
        status_bar_last_seen.innerHTML = "";
        current_chat_other_email = "";
        status_bar_picture.onclick = function () {
upload_group_picture(chat_id) };
        status_bar_picture.style.cursor = "pointer";
    }
    else {
        let other_user_email;
        if (users[0] != email) other_user_email = users[0];
        else other_user_email = users[1];
        status_bar_picture.style.backgroundImage =
`url("${email}/profile_pictures/${other_user_email}_profile_pict
ure.png")`;
        status_bar_last_seen.innerHTML = await
eel.get_user_last_seen(other_user_email)();
        current_chat_other_email = other_user_email;  // in
order to update every 1 second
        status_bar_picture.onclick = null;
        status_bar_picture.style.cursor = "context-menu";
    }
    last_msg_index = 0;
    chat.chat_id = chat_id;
    let chat_msgs = JSON.parse(await
eel.get_chat_msgs(chat_id)());
```

```
    if (chat_msgs === {}) return;  // chat is empty
    await load_msgs(chat_msgs);
    status_bar_name.innerHTML = chat_name;
    chat.scrollTo(0, chat.scrollHeight);
    setTimeout(function() { chat.scrollTo(0, chat.scrollHeight);
}, 200);
    await eel.mark_as_seen(get_open_chat_id())();
}
async function load_more_msgs() {
    // if the user scrolled to the top of the chat this function
will be called
    // it will ask the python for older messages in this chat,
if there are
    // it will load another 800 messages
    chat.scrollBy(0, 20);
    let chat_msgs = JSON.parse(await eel.get_more_msgs()());
    if (Object.keys(chat_msgs).length === 0) return;  // no more
messages
    let first_msg = chat.firstChild;
    let chat_id = chat.chat_id;
    console.log(`loading more messages (id: '${chat_id}')`);
    await load_msgs(chat_msgs, "START");
    chat.onscroll = check_pos;  // re-allow loading more msgs
    chat.scrollTo(0, chat.scrollHeight);
    first_msg.scrollIntoView(true);
    chat.scrollBy(0, -200);  // show some of the new loaded
messages
}
function check_pos() {
    // when the chat is scrolled this function is called
    // when it reaches the top it will call load_more_msgs
    if (chat.scrollTop == 0) {
        chat.onscroll = null;  // disable until finished loading
all new msgs
        load_more_msgs();
    }
}
// eel.expose
function update(chat_id, chat_msgs) {
    // if there is a new msg after the loading of
    // the chat this function is responsible to
    // adding that message, also if a msg was edited
    // this function will change the msg
    if (chat_id !== chat.chat_id) return null;
    chat_msgs = JSON.parse(chat_msgs);
    let from_user, msg, msg_type, deleted_for, deleted_for_all,
seen_by, time, msg_row;
```

```
    let new_messages = {};
    let scrollToBottom = false;
    if (chat.scrollHeight - chat.scrollTop - chat.offsetHeight
<= 200) scrollToBottom = true;
    for (let msg_index in chat_msgs) {
        [from_user, msg, msg_type, deleted_for, deleted_for_all,
seen_by, time] = chat_msgs[msg_index];
        if (parseInt(msg_index) > parseInt(last_msg_index)) {
// new messages
            new_messages[msg_index] = [from_user, msg, msg_type,
deleted_for, deleted_for_all, seen_by, time];
            continue;
        }
        msg_row = document.getElementById(`msg_${msg_index}`);
        // old message that has been changed
        if (deleted_for.includes(email)) {
            if (msg_row !== null) {
                msg_row.remove();
            }
        } else if (deleted_for_all && msg_type === "msg") {
            if (msg_row !== null) {

msg_row.getElementsByClassName("msg_text")[0].innerHTML = msg;
// This message was deleted.
                if (from_user === email)
msg_row.getElementsByClassName("my_msg_box")[0].style.background
Color = "#232323";
                else
msg_row.getElementsByClassName("msg_box")[0].style.backgroundCol
or = "#232323";
            }
        } else if (msg_row !== null && deleted_for_all &&
msg_type === "file" &&

msg_row.getElementsByClassName("msg_image").length === 1) {

msg_row.getElementsByClassName("msg_image")[0].remove();
            if (from_user === email) {

msg_row.getElementsByClassName("my_msg_box")[0].remove();
                let my_msg_box =
window.my_msg_row.getElementsByClassName("my_msg_box")[0].cloneN
ode(true);

my_msg_box.getElementsByClassName("msg_text")[0].innerHTML =
msg;
                msg_row.appendChild(my_msg_box);
```

```
msg_row.getElementsByClassName("my_msg_box")[0].style.background
Color = "#232323";
            } else {

msg_row.getElementsByClassName("msg_box")[0].remove();
                let msg_box =
window.msg_row.getElementsByClassName("msg_box")[0].cloneNode(tr
ue);

msg_box.getElementsByClassName("msg_text")[0].innerHTML = msg;

msg_box.getElementsByClassName("msg_sender")[0].innerHTML =
from_user;
                msg_row.appendChild(msg_box);

msg_row.getElementsByClassName("msg_box")[0].style.backgroundCol
or = "#232323";

msg_row.getElementsByClassName("msg_box")[0].style.backgroundCol
or = "#232323";
            }
        }
    }
    load_msgs(new_messages);
    if (scrollToBottom) setTimeout(function() { chat.scrollTo(0,
chat.scrollHeight); }, 200);;
}

function adjust_msgs_input_width() {
    // adjust the input bar width according to the size of the
window
    let send_btn = document.getElementById("send_msg");
    let msgs_input = document.getElementById("msg_input");
    let upload_file = document.getElementById("upload_file");
    let width = 45;
    while (elementInViewport(send_btn) &&
elementInViewport(upload_file) && width < 78) {
        width++;
        msgs_input.style.width = `${width}%`;
    }
    while (!elementInViewport(send_btn) &&
elementInViewport(upload_file) && width > 45) {
        width--;
        msgs_input.style.width = `${width}%`;
    }
    msgs_input.style.width = `${width - 1}%`;
```

```
}

// eel.expose
function get_open_chat_id() {
    // rerturns the current chat id
    return chat.chat_id;
}


                                    /* Messages */

function message_options(msg_index, full_sender, seen_by,
deleted_for_all) {
    // messages options - delete for me/all , read receipts
    let popup = document.createElement("dialog");
    let delete_for_me = document.createElement("button");
    delete_for_me.innerHTML = "Delete for me";
    delete_for_me.addEventListener("click", async function() {
        popup.close();

document.getElementsByTagName("body")[0].removeChild(popup);
        await eel.delete_message_for_me(get_open_chat_id(),
msg_index)();
    });
    popup.appendChild(delete_for_me);
    if (full_sender === email && !deleted_for_all) {
        let delete_for_all = document.createElement("button");
        delete_for_all.innerHTML = "Delete for all";
        delete_for_all.addEventListener("click", async
function() {
            popup.close();

document.getElementsByTagName("body")[0].removeChild(popup);
            await
eel.delete_message_for_everyone(get_open_chat_id(),
msg_index)();
        });
        popup.appendChild(delete_for_all);
    }
    let read_receipts = document.createElement("button");
    read_receipts.innerHTML = "Read receipts";  // TODO:
implement read receipts
    popup.appendChild(read_receipts);
    //
    let close = document.createElement("button");
    close.innerHTML = "Close";
    close.addEventListener("click", function() { popup.close();
```

```
});
    popup.appendChild(close);
    //
    document.getElementsByTagName("body")[0].prepend(popup);
    popup.showModal();
}
function handle_msg_length(msg) {
    // adds \n if needed to limit msg row length
    let max_chars = 65;
    if (msg.length < max_chars) return msg;
    let row_length = 0;
    let i = 0;
    while (i < msg.length) {
        if (msg[i] === " " && row_length > max_chars * 0.6) {
            msg = msg.slice(0, i) + "\n" + msg.slice(i);
            row_length = 0;
        }
        else if (row_length >= max_chars) {
            msg = msg.slice(0, i) + "\n" + msg.slice(i);
            row_length = 0;
            i++;
        }
        row_length += 1;
        i++;
    }
    return msg + "\n\n";
}


function append_to_chat(position, element) {
    // append a message to the chat, initial load will use END
    // load more msgs will use START
    if (position === "END") {
        chat.appendChild(element);
        chat.appendChild(window.clear.cloneNode(true));
    } else {
        chat.prepend(window.clear.cloneNode(true));
        chat.prepend(element);
    }
}
/*
TODO: add time -
      when this msg time (date) is different from
      the last one (maybe with 'sticky' position in css)
*/
function add_msg(from, sender, msg, time, msg_index, msg_type,
deleted_for,
```

```
    deleted_for_all, seen_by, position="END") {
        // create a new message from all types
        // types:
        // This message was deleted.
        // msg from you, msg from others
        // file msg from you, file msg from others
        // add and remove message (add user and remove user from
group)
        assert(
            position === "END" || position === "START",
            `msg_from_me: param position must be either 'END' or
'START', got '${position}'`
        );
        assert(
            msg_type == "msg" || msg_type == "file" || msg_type
== "remove" || msg_type == "add",
            `msg_from_me: param msg_type must be either 'msg' or
'file' or 'remove' or 'add', got '${msg_type}'`
        );
        let full_sender = sender;
        sender = sender.split("@");
        sender = sender.slice(0, sender.length - 1).join("@") +
":";
        if (deleted_for_all) {
            // This message was deleted.
            let this_msg_row = from == "me" ?
window.my_msg_row.cloneNode(true) :
window.msg_row.cloneNode(true);
            this_msg_row.id = `msg_${msg_index}`;

this_msg_row.getElementsByClassName("msg_text")[0].innerHTML =
msg;
            let msg_box = full_sender === email ?
this_msg_row.getElementsByClassName("my_msg_box")[0] :
this_msg_row.getElementsByClassName("msg_box")[0];
            msg_box.style.backgroundColor = "#232323";
            // append msg row to chat
            append_to_chat(position, this_msg_row);
            // add event listener
            this_msg_row.addEventListener("contextmenu",
function() { message_options(msg_index, full_sender, seen_by,
deleted_for_all); }) // right click
        } else if (deleted_for.includes(email)) {
            return;
        } else if (msg_type === "msg") {
            msg = handle_msg_length(msg);
            let this_msg_row = from == "me" ?
```

```
window.my_msg_row.cloneNode(true) :
window.msg_row.cloneNode(true);
            this_msg_row.id = `msg_${msg_index}`;

this_msg_row.getElementsByClassName("msg_text")[0].innerHTML =
msg;
            // msg_picture.style.backgroundImage =
`url("${email}/${email}_profile_picture.png")`;
            if (full_sender !== email) {

this_msg_row.getElementsByClassName("msg_sender")[0].innerHTML =
sender;
                let msg_picture =
this_msg_row.getElementsByClassName("msg_sender_picture")[0];
                msg_picture.style.backgroundImage =
`url("${email}/profile_pictures/${full_sender}_profile_picture.p
ng")`;
            }
            // append msg row to chat
            append_to_chat(position, this_msg_row);
            // add event listener
            this_msg_row.addEventListener("contextmenu",
function() { message_options(msg_index, full_sender, seen_by,
deleted_for_all); }) // right click
        } else if (msg_type === "file") {
            msg = msg.replaceAll("\\", "/");
            let display_file = false;
            for (let index in image_types) {
                if
(msg.toLowerCase().endsWith(image_types[index])) {
                    display_file = true;
                    break;
                }
            }
            let voice_recording =
msg.toLowerCase().endsWith(".wav") ? true : false;
            if (display_file) {
                let photo_row = from == "me" ?
window.my_photo_msg_row.cloneNode(true) :
window.photo_msg_row.cloneNode(true);

photo_row.getElementsByClassName("msg_image")[0].src =
`${email}/${msg}`;
                photo_row.id = `msg_${msg_index}`;

photo_row.getElementsByClassName("msg_image")[0].onclick = async
function () { await eel.start_file(`${email}/${msg}`); };
```

```
                append_to_chat(position, photo_row);
                photo_row.addEventListener("contextmenu",
function() { message_options(msg_index, full_sender, seen_by,
deleted_for_all); }) // right click
            } else if (voice_recording) {
                let file_row = from == "me" ?
window.my_photo_msg_row.cloneNode(true) :
window.photo_msg_row.cloneNode(true);

file_row.getElementsByClassName("msg_image")[0].remove();
                file_row.id = `msg_${msg_index}`;
                let recording_options_box =
document.createElement("audio");
                recording_options_box.controls = true;
                recording_options_box.id = "audio_player";
                let audio_file =
document.createElement("source");
                audio_file.src = `${email}/${msg}`;
                audio_file.id = "audio_file";
                audio_file.type = "audio/wav";
                recording_options_box.appendChild(audio_file);
                if (from == "me")
file_row.getElementsByClassName("my_msg_box")[0].appendChild(rec
ording_options_box);
                else
file_row.getElementsByClassName("msg_box")[0].appendChild(record
ing_options_box);

                append_to_chat(position, file_row);
                file_row.addEventListener("contextmenu",
function() { message_options(msg_index, full_sender, seen_by,
deleted_for_all); }) // right click
            } else {
                let file_row = from == "me" ?
window.my_photo_msg_row.cloneNode(true) :
window.photo_msg_row.cloneNode(true);

file_row.getElementsByClassName("msg_image")[0].className =
"msg_file";
                file_row.id = `msg_${msg_index}`;
                let file_name = msg.split("/");
                file_name = file_name[file_name.length - 1];

file_row.getElementsByClassName("msg_file")[0].alt = file_name;

file_row.getElementsByClassName("msg_file")[0].onclick = async
function () { await eel.start_file(`${email}/${msg}`); };
```

```
                append_to_chat(position, file_row);
                file_row.addEventListener("contextmenu",
function() { message_options(msg_index, full_sender, seen_by,
deleted_for_all); }) // right click
            }
        } else if (msg_type === "remove" || msg_type === "add")
{
            let add_remove_msg_row =
window.add_remove_msg_row.cloneNode(true);

add_remove_msg_row.getElementsByClassName("msg_text")[0].innerHT
ML = msg;
            append_to_chat(position, add_remove_msg_row);
        }
}
function msg_from_me(sender, msg, time, msg_index, msg_type,
deleted_for,
    deleted_for_all, seen_by, position="END") {
        // call add message on a message that you sent
    add_msg("me", sender, msg, time, msg_index, msg_type,
deleted_for,
    deleted_for_all, seen_by, position);
}
function msg_from_others(sender, msg, time, msg_index, msg_type,
deleted_for,
    deleted_for_all, seen_by, position="END") {
        // call add_msg on a message that was sent by someone
else
    add_msg("others", sender, msg, time, msg_index, msg_type,
deleted_for,
    deleted_for_all, seen_by, position);
}


                            /* Window active & inactive */

function window_active() {
                    /* Change Color Of Search Sep */
    // remove search bar sep
    let search_bar_box =
document.getElementById("search_bar_box");
    let sep =
document.getElementById("search_bar_chat_list_sep");
    search_bar_box.removeChild(sep);
    // create new one with green color
    sep = document.createElement("hr");
    sep.style.borderTop = "1px solid rgb(33, 170, 33)";
```

```
    sep.style.borderRadius = "5px";
    sep.style.backgroundColor = "rgb(33, 170, 33)";
    sep.style.borderColor = "rgb(33, 170, 33)";
    sep.id = "search_bar_chat_list_sep";
    // append it
    search_bar_box.appendChild(sep);
}

function window_inactive() {
                    /* Change Color Of Search Sep */
    // remove search bar sep
    let search_bar_box =
document.getElementById("search_bar_box");
    let sep =
document.getElementById("search_bar_chat_list_sep");
    search_bar_box.removeChild(sep);
    // create new one with black color
    sep = document.createElement("hr");
    sep.style.borderTop = "1px solid black";
    sep.style.borderRadius = "5px";
    sep.style.backgroundColor = "black";
    sep.style.borderColor = "black";
    sep.id = "search_bar_chat_list_sep";
    // append it
    search_bar_box.appendChild(sep);
}

                                    /* Emoji */
function addEmoji(emoji) {
    // add the emoji that was pressed
    document.getElementById('input_bar').value += emoji;
}

                                /* Necessary Data */
async function ask_for_email() {
    email = await eel.get_email()();
}


async function ask_for_username() {
    username = await eel.get_username()();
}


                                /* Communication */
// send message/file
async function send_file(chat_id, file_path="") {
```

```
    await eel.send_file(chat_id, file_path)();
}
async function send_message() {
    let input_bar = document.getElementById("msg_input");
    let msg = input_bar.value;
    input_bar.value = "";  // clear input bar
    input_bar.focus();
    let ok = await eel.send_message(msg, get_open_chat_id())();
    if (!ok && input_bar.value === "") input_bar.value = msg;
}

async function familiarize_user_with() {
    // checks if user exists, if it does it will make
    // him "known" to you and add him to users list
    // so you can create a new chat/group with him
    let user_search_input =
document.getElementById("non_familiar_user_search_input");
    let other_email = user_search_input.value;
    user_search_input.value = "";
    if (other_email != "" && other_email.includes("@") &&
        other_email.length > 2 && other_email.includes(".") &&
        !other_email.includes(" ")
        ) {
        let exists = await
eel.familiarize_user_with(other_email)();
        await sleep(1000);
        if (exists) { toggle_chats_users();
toggle_chats_users(); }
    } else alert("Invalid user, user is an email, needs to have
'@' & '.' and can't contain spaces.")
}

// new chat/group
async function new_chat(other_email) {
    await eel.new_chat(other_email)();
}
async function new_group(other_emails, group_name) {
    await eel.new_group(other_emails, group_name)();
}
function new_group_or_chat() {
    // checks if a new group should be created or a new chat
    // if group asks for group name
    // finally toggles between users list to chats list
    let checked_users = get_all_selected_users_emails();
    if (checked_users.length == 0) ;
    else if (checked_users.length == 1)
new_chat(checked_users[0]);
```

```
    else {
        // get group name, can't start with spaces, can't be ""
and can't contain only spaces
        let group_name = "";
        while (group_name != null && (group_name == "" ||
                !group_name.replace(/\s/g, '').length ||
group_name[0] == " ")) {
            group_name = prompt('Please Enter Group Name: ');
        }
        if (group_name != null) new_group(checked_users,
group_name);
    }
    toggle_chats_users();
}


async function add_user_to_group() {  // TODO create a button
for this and implement function
    // popup with all known users - group users and select the
user to add
    // await eel.add_user_to_group(other_email,
get_open_chat_id());
}


async function remove_user_from_group() {  // TODO create a
button for this and implement function
    // popup with all group users and select the user to remove
    // await eel.remove_user_from_group(other_email,
get_open_chat_id());
}


// recordings functions
async function start_recording() {
    let ok = await eel.start_recording(get_open_chat_id())();
    if (ok) {
        let record_btn = document.getElementById("record_msg");
        record_btn.name = "stop-outline";
        record_btn.onclick = stop_recording;
        record_btn.title = "Stop recording";
    }
}
async function stop_recording() {
    // stop recording and let user decide if he wants to send
    // the recorded audio or delete it (he can listen to it)
    let ok = await eel.stop_recording()();
```

```
    if (ok) {
        let record_btn = document.getElementById("record_msg");
        record_btn.name = "mic-outline";
        record_btn.onclick = start_recording;
        record_btn.title = "Start recording";
    }
}
function restore_input() {
    // resotre input after displaying recording options
    let audio = chat_actions.getElementsByTagName("audio")[0];
    audio.getElementsByTagName("source")[0].remove();
    audio.remove();
    document.getElementById("delete_recording").remove();
    chat_actions.insertBefore(input_bar_box,
document.getElementById("right_side_actions"));
    document.getElementById("send_msg").onclick = send_message;
}
async function delete_recording(rec_file_path) {
    restore_input();
    await eel.delete_recording(rec_file_path)();
}
function send_recording(rec_file_path, chat_id) {
    restore_input();
    send_file(chat_id, rec_file_path);
}
// eel.expose
function display_recording_options(rec_file_path, chat_id) {
    // display the recording options - delete, send & listen to
audio
    if (chat_actions.contains(input_bar_box)) {
        chat_actions.removeChild(input_bar_box);
    } else {
        let delete_btn =
document.getElementById("delete_recording");
        delete_btn.click();
        chat_actions.removeChild(input_bar_box);
    }
    let recording_options_box = document.createElement("audio");
    recording_options_box.controls = true;
    recording_options_box.id = "audio_player";
    let audio_file = document.createElement("source");
    audio_file.src = rec_file_path;
    audio_file.id = "audio_file";
    audio_file.type = "audio/wav";
    recording_options_box.appendChild(audio_file);
    chat_actions.insertBefore(recording_options_box,
document.getElementById("right_side_actions"));
```

```
    let delete_btn = document.createElement("ion-icon");
    delete_btn.name = "trash-outline";
    delete_btn.id = "delete_recording";
    delete_btn.onclick = (rec_file_path) =>
{delete_recording(rec_file_path)};
    chat_actions.insertBefore(delete_btn,
recording_options_box);
    document.getElementById("send_msg").onclick = function()
{send_recording(rec_file_path, chat_id)};
}
// end of recordings functions

async function add_hang_up_btn_and_check_call_status() {
    // add a button on the top that allows to hang up
    // also checks if the call ended, if so removes the button
    // and alerts the user
    let hang_up_call_btn = document.createElement("button");
    hang_up_call_btn.innerHTML = "Hang Up";
    hang_up_call_btn.id = "hang_up_call_btn";
    hang_up_call_btn.addEventListener("click", async function()
{
        await eel.hang_up_call()();
        hang_up_call_btn.remove();
    });
    let body = document.getElementsByTagName("body")[0];
    body.prepend(hang_up_call_btn);
    async function check_call_status() {
        // checks if the call is still running
        let is_running = await eel.check_ongoing_call()();
        if (is_running) setTimeout(check_call_status, 2500);
        else {
            let hang_up_call_btn =
document.getElementById("hang_up_call_btn");
            if (hang_up_call_btn !== null) {
                hang_up_call_btn.remove();
                alert("Call ended");
            }
        }
    }
    check_call_status();
}

async function make_call() {
    // start a call with the current chat/group
    if (document.getElementById("hang_up_call_btn") !== null) {
        alert("Already in a call, hang up to start a new one.");
        return;
```

```
    }
    let chat_id = get_open_chat_id();
    if (chat_id !== "") {
        let status = await eel.make_call(chat_id)();
        if (status) add_hang_up_btn_and_check_call_status();
        else alert("Call failed.");
    }
}

function ongoing_call(group_name, port) {
    // show a popup and let the user decide
    // if he wants to answer the call or ignore
    let popup = document.createElement("dialog");
    let group_name_label = document.createElement("h1");
    group_name_label.innerHTML = group_name;
    let answer = document.createElement("button");
    answer.innerHTML = "Answer Call";
    answer.addEventListener("click", async function() {
        popup.close();
        popup.remove();
        if (document.getElementById("hang_up_call_btn") !==
null) {  // there is an ongoing call
            let what_to_do = prompt("Already in a call, do you
want to hang up? [y/n] ");
            while (what_to_do != null &&
                   (what_to_do == "" ||
!what_to_do.replace(/\s/g, '').length) &&
                   what_to_do !== "y" && what_to_do !== "Y" &&
what_to_do !== "yes" && what_to_do !== "YES" &&
                   what_to_do !== "n" && what_to_do !== "N" &&
what_to_do !== "no" && what_to_do !== "NO"
            ) {
                what_to_do = prompt("Already in a call, do you
want to hang up? [y/n] ");
            }
            if (what_to_do !== "y" && what_to_do !== "Y" &&
what_to_do !== "yes" && what_to_do !== "YES"){
                return;
            } else eel.hang_up_call()();
        }
        await eel.answer_call(port)();
        add_hang_up_btn_and_check_call_status();
    });
    let ignore = document.createElement("button");
    ignore.innerHTML = "Ignore Call";
    ignore.addEventListener("click", function() { popup.close();
popup.remove(); });
```

```
    popup.appendChild(group_name_label);
    popup.appendChild(answer);
    popup.appendChild(ignore);
    document.getElementsByTagName("body")[0].prepend(popup);
    popup.showModal();
}

async function upload_profile_picture() {
    await eel.upload_profile_picture()();
}

async function upload_group_picture(chat_id) {
    await eel.upload_group_picture(chat_id)();
}


                                        /* Main Setup */


// eel.expose
async function main() {
    // main setup - start app
    console.log("main");
    // start the python updater
    await eel.start_app()();
    // ask for email & username
    await ask_for_email();
    await ask_for_username();
    window.title += ` - ${username}`

    // profile picture
    let user_profile_picture = document.getElementById("user-
profile-picture");
    user_profile_picture.style.backgroundImage =
`url("${email}/${email}_profile_picture.png")`;

    // load all chat buttons
    load_chat_buttons();
    // adjust input width on loadup
    adjust_msgs_input_width();

    // start last seen updater, call it once and it will call
it-self
    await update_last_seen();

    // Event listeners
    // window resize, resize input width
```

```
    window.addEventListener("resize", adjust_msgs_input_width);
    // window active & inactive event listeners
    window.addEventListener('focus', window_active);
    window.addEventListener('blur', window_inactive);
    // current state
    if (document.hasFocus()) window_active();
    else window_inactive();
    window.addEventListener("beforeunload", function () {
eel.close_program()(); })

    // TODO: uncomment the next lines

    // block special keys
    // document.onkeydown = function (e) {
    //     if (e.key === "F1" || e.key === "F3" || e.key ===
"F5" ||
    //         e.key === "F7" || e.key === "F12") {
    //         return false;
    //     }
    // };

    // main finish log
    console.log("main setup finished successfully");
}


                                /* Globals */
var image_types = [".jpeg", ".webp", ".gif", ".png", ".apng",
".svg", ".bmp", ".ico", ".jpg"];
var email;  // email
var username; // username
//
var last_msg_index;  // the index number of the most recent msg
in current chat
var current_search_key;  // current search input (of chat
buttons)
// the chat
var chat = document.getElementById("chat");  // the chat div
chat.chat_id = "";
// chat status bar
var status_bar_name = document.getElementById("status-bar-
name");  // chat name
var status_bar_picture = document.getElementById("status-bar-
picture");  // chat picture
var status_bar_last_seen = document.getElementById("status-bar-
last-seen");  // chat lst seen
var current_chat_other_email = "";  // if one on one chat, it
```

```
will contaim the email of the other user
// chat actions & input
var chat_actions = document.getElementById("chat_actions");  //
chat actions (file, emoji, send)
var input_bar_box = document.getElementById("input_box");  //
input message
// chat list
var chats_list =
document.getElementsByClassName("chats_list")[0];  // list of
chats/groups
// new chat/group
var users_list = document.createElement("div");  // list of
users (for creating chats/groups)
users_list.className = "chats_list";
var create_new_chat_or_group = document.createElement("button");
// create new chat/group btn
create_new_chat_or_group.id = "create_chat_or_group";
create_new_chat_or_group.onclick = new_group_or_chat;
create_new_chat_or_group.innerHTML = "Create";
var search_for_non_familiar_user =
document.createElement("div");  // search user (for new
chat/group) box
search_for_non_familiar_user.id =
"search_for_non_familiar_user";
let non_familiar_user_search_input =
document.createElement("input");  // input of username to search
non_familiar_user_search_input.id =
"non_familiar_user_search_input";
non_familiar_user_search_input.placeholder = "Search for other
users";
search_for_non_familiar_user.appendChild(non_familiar_user_searc
h_input);
let non_familiar_user_search_btn =
document.createElement("button");
non_familiar_user_search_btn.id =
"non_familiar_user_search_btn";
non_familiar_user_search_btn.innerHTML = "Search";
non_familiar_user_search_btn.onclick = familiarize_user_with;
search_for_non_familiar_user.appendChild(non_familiar_user_searc
h_btn);  // button to trigger search

var message_options_window;
var call_options_window;

// eel
eel.expose(get_open_chat_id);
eel.expose(update);
```

```
eel.expose(display_recording_options);
eel.expose(ongoing_call);

main();


/*                                    TODOS
1. need to add buttons for adding & removing users when a chat
is 'group'
   and implement the functions to select the users to remove/add
2. messages options - seen list?
*/
```

login&signup&reset.css

```
                                        /* Fonts */

@import
url('https://fonts.googleapis.com/css2?family=Poppins:wght@400;6
00&display=swap');

                                        /* Animations */

@keyframes box_animation {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}

@keyframes box_animation_back {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(-360deg); }
}

                                        /* Tags */

body {
    font-family: 'Poppins', sans-serif;
    min-height: 100vh;
    background: linear-gradient(to right, #333399, #ff00cc);
    display: flex;
    justify-content: center;
    align-items: center;
    overflow-y: hidden;
}

button:hover {
    cursor: pointer;
```

```
}

button:active {
    opacity: 0.7;
}

                                        /* Text selection */

.enable_text_selection {
    user-select: text; /* standard syntax */
    -webkit-user-select: text; /* webkit (safari, chrome)
browsers */
    -moz-user-select: text; /* mozilla browsers */
    -khtml-user-select: text; /* webkit (konqueror) browsers */
    -ms-user-select: text; /* IE10+ */
}

.disable_text_selection {
    user-select: none; /* standard syntax */
    -webkit-user-select: none; /* webkit (safari, chrome)
browsers */
    -moz-user-select: none; /* mozilla browsers */
    -khtml-user-select: none; /* webkit (konqueror) browsers */
    -ms-user-select: none; /* IE10+ */
}

                            /* Box (for login/signup/reset
password */

#box_wrap::before {
    content: '';
    position: absolute;
    width: 594px;
    height: 505px;
    top: -50%;
    left: -50%;
    background: linear-gradient(0deg, transparent, transparent,
#ff00cc, #ff00cc, #ff00cc);
    transform-origin: bottom right;
    animation: box_animation 6s linear infinite;
    z-index: 1;
}

#box_wrap {
    position: relative;
    background: transparent;
    border-radius: 10px;
```

```
    padding: 8px;
    overflow: hidden;
}

#box_wrap::after {
    content: '';
    position: absolute;
    width: 594px;
    height: 505px;
    top: -50%;
    left: -50%;
    background: linear-gradient(0deg, #ff2770, #ff2770, #ff2770,
transparent, transparent);
    transform-origin: bottom right;
    animation: box_animation_back 6s linear infinite;
    z-index: 1;
}

#box {
    width: 450px;
    background: #BE8ED4;
    padding: 4rem;
    border-radius: 10px;
    border: 8px solid black;
    position: relative;
    text-align: center;
    z-index: 5;
}
/* boxes for inputs */
#password_box, #email_box, #username_box, #confirmation_code_box
{
    width: fit-content;
    height: fit-content;
    display: block;
    position: relative;
}
/* icons for inputs */
.fa.fa-lock, .fa.fa-envelope, .fa.fa-user {
    position: absolute;
    top: 20.5%;
    left: 10px;
    font-size: 18px;
    color: rgb(58, 58, 58);
}
/* inputs */
#username_input, #password_input, #email_input,
#confirmation_code_input {
```

```
    border: none;
    outline: none;
    background: rgba(255, 255, 255, .3);
    padding: 1rem;
    border-radius: 100px;
    width: 400px;
    margin-bottom: 20px;
    display: inline;
    flex-direction: column;
    margin-left: auto;
    margin-right: auto;
    padding-left: 35px;
}
/* icons for changing password visibility */
.fa.fa-eye, .fa.fa-eye-slash {
    position: absolute;
    top: 21%;
    right: 15px;
    font: 18px;
    color: rgb(58, 58, 58);
}
/* buttons for login/signup/reset password/submitting
confirmation code */
#login_btn, #signup_btn, #reset_password_btn, #submit {
    border: none;
    margin-bottom: 20px;
    background: rgb(8, 8, 8);
    color: white;
    padding: 1rem;
    border-radius: 100px;
    text-align: center;
    text-transform: uppercase;
    letter-spacing: 2px;
}
/* links to other pages */
.other_pages {
    margin-bottom: 10px;
}
a {
    /* text-decoration: none; */
    color: #3f89ff;
    font-weight: bold;
} a:hover { text-decoration: none; }
```

login.html

```
<!DOCTYPE html>

<html>
  <head>
    <title>ChatEase Login</title>
    <link rel="icon" href="favicon.ico">
  </head>

  <body class="disable_text_selection" oncontextmenu="return
false;">
    <div id="box_wrap">
      <div id="box">
          <h1>Login</h1>
          <div id="email_box">
            <i class="fa fa-envelope" title="email"></i>
            <input id="email_input" type="email"
placeholder="Email">
          </div>
          <div id="password_box">
            <i class="fa fa-lock" title="password"></i>
            <input id="password_input" type="password"
placeholder="Password">
            <i class="fa fa-eye" id="eye" title="Show Password"
onclick="toggle_password_visibility()"></i>
          </div>
          <button id="login_btn"
onclick="login()">Login</button>
          <div class="other_pages">
              Don't have an account? <a
href="signup.html">signup</a>
          </div>
          <div class="other_pages">
              Forgot your password? <a
href="reset_password.html">reset your password</a>
          </div>
      </div>
    </div>
    <!--                          Other Files (javascript & css)
-->
    <link rel="stylesheet" href="login&signup&reset.css">
    <!-- Part of eel, this file isn't in webroot, it's handled
by eel -->
    <script type="text/javascript" src="/eel.js"></script>
    <!-- The client that connectes to the eel python program -->
    <script type="text/javascript" src="login.js"></script>
    <!-- Disable Double & Triple Click Text Selection -->
```

```
    <!-- <script type="text/javascript"
src="selection.js"></script> -->
    <script type="module"
src="https://unpkg.com/ionicons@7.1.0/dist/ionicons/ionicons.esm
.js"></script>
    <script nomodule
src="https://unpkg.com/ionicons@7.1.0/dist/ionicons/ionicons.js"
></script>
    <link rel="stylesheet"
href="https://use.fontawesome.com/releases/v5.0.7/css/all.css">
    <!-- Disable going back -->
    <script type = "text/javascript" > history.pushState(null,
null, location.href); history.back(); history.forward();
window.onpopstate = function () { history.go(1); }; </script>
  </body>
</html>
```

login.js

```
function toggle_password_visibility() {
    let password_input =
document.getElementById("password_input");
    password_input.type = password_input.type === "password" ?
"text" : "password";
    let eye = document.getElementById("eye");
    eye.className = eye.className === "fa fa-eye" ? "fa fa-eye-
slash" : "fa fa-eye";
    eye.title = eye.title === "Show Password" ? "Hide Password"
: "Show Password";
}

async function login() {
    // login request
    let login_btn = document.getElementById("login_btn");
    login_btn.onclick = null;
    let email_input = document.getElementById("email_input");
    let password_input =
document.getElementById("password_input");
    let email = email_input.value, password =
password_input.value;
    let [status, reason] = await eel.login(email, password)();
    if (!status && reason != "Already Logged In") {
        let error_msg = document.getElementById("error_msg");
        if (error_msg === null) {
            error_msg = document.createElement("div");
            error_msg.id = "error_msg";
            error_msg.style.color = "rgb(145, 52, 60)";
```

```
            error_msg.style.marginBottom = "22px";
            let box = document.getElementById("box");
            box.insertBefore(error_msg, login_btn);
        }
        error_msg.innerHTML = reason;
    } else {
        window.location = "ChatEase.html";
    }
    login_btn.onclick = async function() { await login(); };
}


document.onkeydown = function (e) {
    if (e.key === "F1" || e.key === "F3" || e.key === "F5" ||
        e.key === "F7" || e.key === "F12") {
        return false;
    }
};
```

messages.js

```
/* Create an global msg (from yourself) because it's faster to
copy it when creating a new msg */
// msg row
var my_msg_row = document.createElement("div");
my_msg_row.className = "my_msg_row";
// msg box
var my_msg_box = document.createElement("div");
my_msg_box.className = "my_msg_box";
// msg text and time
var my_text_and_time = document.createElement("div");
my_text_and_time.className = "msg_data";
// // msg sender picture
// var my_msg_sender_picture = document.createElement("div");
// my_msg_sender_picture.className = "msg_sender_picture";
// // my_msg_sender_picture.style.backgroundImage = "";
// // msg sender
// var my_msg_sender = document.createElement("div");
// my_msg_sender.className = "msg_sender";
// // my_msg_sender.innerHTML = sender;
// msg text
var my_msg_text = document.createElement("div");
my_msg_text.className = "msg_text";
// my_msg_text.innerHTML = msg;
// msg time
// var my_msg_time = document.createElement("div");
// my_msg_time.className = "msg_time";
```

```
// my_msg_time.innerHTML = time;
// append all elements to msg row
// // my_text_and_time.appendChild(my_msg_sender_picture);
// // my_text_and_time.appendChild(my_msg_sender);
my_text_and_time.appendChild(my_msg_text);
// my_text_and_time.appendChild(my_msg_time);
my_msg_box.appendChild(my_text_and_time);
my_msg_row.appendChild(my_msg_box);

window.my_msg_row = my_msg_row;

/* ----------------------------------------------------------------
----- */

/* Create an global msg (from others) because it's faster to
copy it when creating a new msg */
// msg row
var msg_row = document.createElement("div");
msg_row.className = "msg_row";
// msg box
var msg_box = document.createElement("div");
msg_box.className = "msg_box";
// msg text and time
var text_and_time = document.createElement("div");
text_and_time.className = "msg_data";
var msg_sender_picture = document.createElement("div");
msg_sender_picture.className = "msg_sender_picture";
// my_msg_sender_picture.style.backgroundImage = "";
// msg sender
var msg_sender = document.createElement("div");
msg_sender.className = "msg_sender";
// msg_sender.innerHTML = sender;
// msg text
var msg_text = document.createElement("div");
msg_text.className = "msg_text";
// msg_text.innerHTML = msg;
// msg time
// var msg_time = document.createElement("div");
// msg_time.className = "msg_time";
// msg_time.innerHTML = time;
// append all elements to msg row
text_and_time.appendChild(msg_sender_picture);
text_and_time.appendChild(msg_sender);
text_and_time.appendChild(msg_text);
// text_and_time.appendChild(msg_time);
msg_box.appendChild(text_and_time);
msg_row.appendChild(msg_box);
```

```
window.msg_row = msg_row;

/* ----------------------------------------------------------------
----- */

var clear = document.createElement("div");
clear.className = "clear";
window.clear = clear;

/* ----------------------------------------------------------------
----- */

var my_photo_msg_row = document.createElement("div");
my_photo_msg_row.className = "my_msg_row";
var my_photo_msg_box = document.createElement("div");
my_photo_msg_box.className = "my_msg_box";
var my_photo = document.createElement("img");
my_photo.className = "msg_image";
my_photo_msg_box.appendChild(my_photo);
my_photo_msg_row.appendChild(my_photo_msg_box);

window.my_photo_msg_row = my_photo_msg_row;

/* ----------------------------------------------------------------
----- */

var photo_msg_row = document.createElement("div");
photo_msg_row.className = "msg_row";
var photo_msg_box = document.createElement("div");
photo_msg_box.className = "msg_box";
var photo = document.createElement("img");
photo.className = "msg_image";
photo_msg_box.appendChild(photo);
photo_msg_row.appendChild(photo_msg_box);

window.photo_msg_row = photo_msg_row;

/* ----------------------------------------------------------------
----- */

var add_remove_msg_row = document.createElement("div");
add_remove_msg_row.className = "add_remove_msg_row";
var add_remove_msg_box = document.createElement("div");
add_remove_msg_box.className = "msg_box";
var add_remove_msg_data = document.createElement("div");
add_remove_msg_data.className = "msg_data";
```

```
var add_remove_msg_text = document.createElement("div");
add_remove_msg_text.className = "msg_text";
add_remove_msg_data.appendChild(add_remove_msg_text);
add_remove_msg_box.appendChild(add_remove_msg_data);
add_remove_msg_row.appendChild(add_remove_msg_box);

window.add_remove_msg_row = add_remove_msg_row;

/* ----------------------------------------------------------------
----- */
```

reset_password.html

```
<!DOCTYPE html>

<html>
  <head>
    <title>ChatEase Reset Password</title>
    <link rel="icon" href="favicon.ico">
  </head>

  <body class="disable_text_selection" oncontextmenu="return
false;">
    <div id="box_wrap">
      <div id="box">
          <h1>Reset Your Password</h1>
          <div id="email_box">
            <i class="fa fa-envelope" title="email"></i>
            <input id="email_input" type="email"
placeholder="Email">
          </div>
          <div id="username_box">
            <i class="fa fa-user" title="username"></i>
            <input id="username_input" type="text"
placeholder="Username">
          </div>
          <button id="reset_password_btn"
onclick="reset_password_request()">Reset Password</button>
          <div class="other_pages" id="have_account">
              Have an account? <a href="login.html">login</a>
          </div>
          <div class="other_pages" id="no_account">
              Don't have an account? <a
href="signup.html">signup</a>
          </div>
      </div>
    </div>
```

```
    <!--                            Other Files (javascript & css)
-->
    <link rel="stylesheet" href="login&signup&reset.css">
    <!-- Part of eel, this file isn't in webroot, it's handled
by eel -->
    <script type="text/javascript" src="/eel.js"></script>
    <!-- The client that connectes to the eel python program -->
    <script type="text/javascript"
src="reset_password.js"></script>
    <!-- Disable Double & Triple Click Text Selection -->
    <!-- <script type="text/javascript"
src="selection.js"></script> -->
    <script type="module"
src="https://unpkg.com/ionicons@7.1.0/dist/ionicons/ionicons.esm
.js"></script>
    <script nomodule
src="https://unpkg.com/ionicons@7.1.0/dist/ionicons/ionicons.js"
></script>
    <link rel="stylesheet"
href="https://use.fontawesome.com/releases/v5.0.7/css/all.css">
    <!-- Disable going back -->
    <script type = "text/javascript" > history.pushState(null,
null, location.href); history.back(); history.forward();
window.onpopstate = function () { history.go(1); }; </script>
  </body>
</html>
```

reset_password.js

```
function toggle_password_visibility() {
    let password_input =
document.getElementById("password_input");
    password_input.type = password_input.type === "password" ?
"text" : "password";
    let eye = document.getElementById("eye");
    eye.className = eye.className === "fa fa-eye" ? "fa fa-eye-
slash" : "fa fa-eye";
    eye.title = eye.title === "Show Password" ? "Hide Password"
: "Show Password";
}

function display_msg(msg_, type="error") {
    // display message about the status of the reset password
    let msg = document.getElementById("msg");
    if (msg === null) {
        msg = document.createElement("div");
        msg.id = "msg";
```

```
        msg.style.color = type === "error" ? "rgb(145, 52, 60)"
: "rgb(65, 180, 102)";
        msg.style.marginBottom = "22px";
        if (reset_password_btn != null) box.insertBefore(msg,
reset_password_btn);
        else box.insertBefore(msg, submit_btn);
    }
    msg.innerHTML = msg_;
}

async function reset_password_request() {
    // make a request to reset password
    reset_password_btn.onclick = null;
    let email = email_input.value, username =
username_input.value;
    let status = await eel.reset_password_stage1(email,
username)();
    if (!status) {
        display_msg("Error");
    } else {
        email_box.remove();
        username_box.remove();
        reset_password_btn.remove();
        link_login.remove();
        link_signup.remove();
        let msg = document.getElementById("msg");
        if (msg != null) msg.remove();
        box.appendChild(confirmation_code_box);
        box.appendChild(password_box);
        box.appendChild(submit_btn);
    }
    reset_password_btn.onclick = async function() { await
reset_password_request(); };
}

async function reset_password_confirmation_and_send_pass() {
    // send the confirmation code and new password
    submit_btn.onclick = null;
    let confirmation_code = confirmation_code_input.value,
password = password_input.value;
    let status = await
eel.reset_password_stage2(confirmation_code, password)();
    confirmation_code_box.remove();
    password_box.remove();
    submit_btn.remove();
    box.appendChild(email_box);
    box.appendChild(username_box);
```

```
    box.appendChild(reset_password_btn);
    box.appendChild(link_login);
    box.appendChild(link_signup);
    let msg = document.getElementById("msg");
    if (msg != null) msg.remove();
    if (!status) display_msg("Reset password failed !");
    else display_msg("Password has been reset successfully !",
type="regular");
    submit_btn.onclick = async function() { await
reset_password_confirmation_and_send_pass(); };
}


document.onkeydown = function (e) {
    if (e.key === "F1" || e.key === "F3" || e.key === "F5" ||
        e.key === "F7" || e.key === "F12") {
        return false;
    }
};


                                /* Globals */
var email_box = document.getElementById("email_box");
var username_box = document.getElementById("username_box");
var reset_password_btn =
document.getElementById("reset_password_btn");
var link_login = document.getElementById("have_account");
var link_signup = document.getElementById("no_account");

/* Confirmation code stage */

var confirmation_code_box = document.createElement("div");
confirmation_code_box.id = "confirmation_code_box";
var confirmation_code_input = document.createElement("input");
confirmation_code_input.id = "confirmation_code_input";
confirmation_code_input.type = "text";
confirmation_code_input.placeholder = "Confirmation code";
confirmation_code_box.appendChild(confirmation_code_input);

var password_box = document.createElement("div");
password_box.id = "password_box";
var password_icon = document.createElement("i");
password_icon.className = "fa fa-lock";
password_icon.title = "password";
password_box.appendChild(password_icon);
var password_input = document.createElement("input");
password_input.id = "password_input";
```

```
password_input.type = "password";
password_input.placeholder = "Password";
password_box.appendChild(password_input);
var password_eye = document.createElement("i");
password_eye.className = "fa fa-eye";
password_eye.id = "eye";
password_eye.title = "Show Password";
password_eye.onclick = toggle_password_visibility;
password_box.appendChild(password_eye);

var submit_btn = document.createElement("button");
submit_btn.id = "submit";
submit_btn.innerHTML = "Submit";
submit_btn.onclick = reset_password_confirmation_and_send_pass;
var box = document.getElementById("box");
```

selection.js – not in use

```
// var _tripleClickTimer = 0;
// var _mouseDown = false;

// document.onmousedown = function() {
//     _mouseDown = true;
// };

// document.onmouseup = function() {
//     _mouseDown = false;
// };

// document.ondblclick = function DoubleClick(evt) {
//     ClearSelection();
//     window.clearTimeout(_tripleClickTimer);

//     //handle triple click selecting whole paragraph
//     document.onclick = function() {
//         ClearSelection();
//     };

//     _tripleClickTimer =
window.setTimeout(RemoveDocumentClick, 100);
// };

// function RemoveDocumentClick() {
//     if (!_mouseDown) {
//         document.onclick = null;
//         return true;
//     }
```

```
//      _tripleClickTimer =
window.setTimeout(RemoveDocumentClick, 100);
//      return false;
// }

// function ClearSelection() {
//      if (window.getSelection)
//          window.getSelection().removeAllRanges();
//      else if (document.selection)
//          document.selection.empty();
// }
```

signup.html

```
<!DOCTYPE html>

<html>
  <head>
    <title>ChatEase Signup</title>
    <link rel="icon" href="favicon.ico">
  </head>

  <body class="disable_text_selection" oncontextmenu="return
false;">
    <div id="box_wrap">
      <div id="box">
          <h1>Signup</h1>
          <div id="email_box">
            <i class="fa fa-envelope" title="email"></i>
            <input id="email_input" type="email"
placeholder="Email">
          </div>
          <div id="username_box">
            <i class="fa fa-user" title="username"></i>
            <input id="username_input" type="text"
placeholder="Username">
          </div>
          <div id="password_box">
            <i class="fa fa-lock" title="password"></i>
            <input id="password_input" type="password"
placeholder="Password">
            <i class="fa fa-eye" id="eye" title="Show Password"
onclick="toggle_password_visibility()"></i>
          </div>
          <button id="signup_btn"
onclick="signup_request()">Signup</button>
```

```
          <div class="other_pages" id="have_account">
              Have an account? <a href="login.html">login</a>
          </div>
          <div class="other_pages" id="forgot_pass">
              Forgot your password? <a
href="reset_password.html">reset your password</a>
          </div>
      </div>
    </div>
    <!--                          Other Files (javascript & css)
-->
    <link rel="stylesheet" href="login&signup&reset.css">
    <!-- Part of eel, this file isn't in webroot, it's handled
by eel -->
    <script type="text/javascript" src="/eel.js"></script>
    <!-- The client that connectes to the eel python program -->
    <script type="text/javascript" src="signup.js"></script>
    <!-- Disable Double & Triple Click Text Selection -->
    <!-- <script type="text/javascript"
src="selection.js"></script> -->
    <script type="module"
src="https://unpkg.com/ionicons@7.1.0/dist/ionicons/ionicons.esm
.js"></script>
    <script nomodule
src="https://unpkg.com/ionicons@7.1.0/dist/ionicons/ionicons.js"
></script>
    <link rel="stylesheet"
href="https://use.fontawesome.com/releases/v5.0.7/css/all.css">
    <!-- Disable going back -->
    <script type = "text/javascript" > history.pushState(null,
null, location.href); history.back(); history.forward();
window.onpopstate = function () { history.go(1); }; </script>
  </body>
</html>
```

signup.js

```
function toggle_password_visibility() {
    let password_input =
document.getElementById("password_input");
    password_input.type = password_input.type === "password" ?
"text" : "password";
    let eye = document.getElementById("eye");
    eye.className = eye.className === "fa fa-eye" ? "fa fa-eye-
slash" : "fa fa-eye";
    eye.title = eye.title === "Show Password" ? "Hide Password"
: "Show Password";
}
```

```
function display_msg(msg_, type="error") {
    // display message about the status of the signup
    let msg = document.getElementById("msg");
    if (msg === null) {
        msg = document.createElement("div");
        msg.id = "msg";
        msg.style.color = type === "error" ? "rgb(145, 52, 60)"
: "rgb(65, 180, 102)";
        msg.style.marginBottom = "22px";
        if (signup_btn != null) box.insertBefore(msg,
signup_btn);
        else box.insertBefore(msg, submit_btn);
    }
    msg.innerHTML = msg_;
}

async function signup_request() {
    // signup request (first stage)
    signup_btn.onclick = null;
    let email = email_input.value, password =
password_input.value, username = username_input.value;
    let [status, reason] = await eel.signup_stage1(email,
password, username)();
    if (!status) {
        display_msg(reason);
    } else {
        email_box.remove();
        username_box.remove();
        password_box.remove();
        signup_btn.remove();
        link_login.remove();
        link_reset.remove();
        let msg = document.getElementById("msg");
        if (msg != null) msg.remove();
        box.appendChild(confirmation_code_box);
        box.appendChild(submit_btn);
    }
    signup_btn.onclick = async function () { await
signup_request(); };
}

async function signup_confirmation_code() {
    // signup confirmation code stage
    submit_btn.onclick = null;
    let confirmation_code = confirmation_code_input.value;
    let status = await eel.signup_stage2(confirmation_code)();
    confirmation_code_box.remove();
```

```
    submit_btn.remove();
    box.appendChild(email_box);
    box.appendChild(username_box);
    box.appendChild(password_box);
    box.appendChild(signup_btn);
    box.appendChild(link_login);
    box.appendChild(link_reset);
    let msg = document.getElementById("msg");
    if (msg != null) msg.remove();
    if (!status) display_msg("Signup failed !");
    else display_msg("Signed up successfully !",
type="regular");
    submit_btn.onclick = async function() { await
signup_confirmation_code(); };
}


document.onkeydown = function (e) {
    if (e.key === "F1" || e.key === "F3" || e.key === "F5" ||
        e.key === "F7" || e.key === "F12") {
        return false;
    }
};


                                /* Globals */
var email_box = document.getElementById("email_box");
var username_box = document.getElementById("username_box");
var password_box = document.getElementById("password_box");
var signup_btn = document.getElementById("signup_btn");
var link_login = document.getElementById("have_account");
var link_reset = document.getElementById("forgot_pass");

/* Confirmation code stage */

var confirmation_code_box = document.createElement("div");
confirmation_code_box.id = "confirmation_code_box";
var confirmation_code_input = document.createElement("input");
confirmation_code_input.id = "confirmation_code_input";
confirmation_code_input.type = "text";
confirmation_code_input.placeholder = "Confirmation code";
confirmation_code_box.appendChild(confirmation_code_input);

var submit_btn = document.createElement("button");
submit_btn.id = "submit";
submit_btn.innerHTML = "Submit";
```

```
submit_btn.onclick = signup_confirmation_code;
var box = document.getElementById("box");
```

Other Files

calls_udp_client.py

```
"""
##############################################
Author: Omer Dagry
Mail: omerdagry@gmail.com
Date: 30/05/2023 (dd/mm/yyyy)
##############################################
"""

import time
import pickle
import socket
import hashlib
import pyaudio
import threading

from ClientSecureSocket import ClientEncryptedProtocolSocket


# Constants
# PyAudio
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 44100
# Others
CHUNK = 1024 * 8
BUFFER_SIZE = CHUNK * 4

# Globals
stop = False
connected = False


def get_sound(stream: pyaudio.Stream, client_socket:
socket.socket):
    """ a loop to get UDP audio packets from the server """
    global stop
    data = b""
    while not stop:
        try:
            # write audio received from server to the stream
            data = client_socket.recvfrom(BUFFER_SIZE)[0]
```

```
        except socket.timeout:
            continue
        except (ConnectionError, socket.error):
            stop = True
            break
        except KeyboardInterrupt:
            pass
        if data != b"":
            stream.write(data)


def send_sound(stream: pyaudio.Stream, client_socket:
socket.socket, server_addr: tuple[str, int]):
    """ a loop to send UDP audio packets to the server """
    global stop
    while not stop:
        # Read audio data from the stream
        data = stream.read(CHUNK)
        try:
            # Send the audio data to the server
            client_socket.sendto(data, server_addr)
        except socket.timeout:
            pass
        except (ConnectionError, socket.error):
            stop = True
            break
        except KeyboardInterrupt:
            pass
        time.sleep(0.02)


def handle_tcp_connection(server_addr: tuple[str, int], email:
str, password: str):
    """ the TCP connection to the server """
    global connected, stop
    tcp_sock = ClientEncryptedProtocolSocket()
    tcp_sock.connect(server_addr)
    connected = True
    try:
        tcp_sock.send_message(pickle.dumps([email, password]))
        if tcp_sock.recv_message() != b"ok    ":
            print("Failed to connect to call.")
            raise ConnectionError
        print("Connected to call.")
        while True:
            tcp_sock.send_message(b"hi")
            time.sleep(5)
```

```
    except (ConnectionError, socket.error):
        pass
    except KeyboardInterrupt:
        pass
    finally:
        stop = True
        tcp_sock.close()


def join_call(server_addr: tuple[str, int], email: str,
password: str):
    """
        calls all the needed functions and start the PyAudio
stream
        open a process when calling this function
    """
    global stop
    #
    tcp_connection_thread = threading.Thread(
        target=handle_tcp_connection, args=(server_addr, email,
password,), daemon=True
    )
    tcp_connection_thread.start()
    #
    while not connected:
        if stop:
            pass  # TODO: display error of connection to server
        time.sleep(0.1)
    #
    client_socket = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
    # Create PyAudio stream for recording and playing audio
    p = pyaudio.PyAudio()
    # info = p.get_host_api_info_by_index(0)
    # device_count = info.get('deviceCount')
    # devices = [p.get_device_info_by_host_api_device_index(0,
i) for i in range(device_count)]
    # input_devices = [device for device in devices if
"maxInputChannels" in device and device["maxInputChannels"] > 0]
    # output_devices = [device for device in devices
    #                   if "maxOutputChannels" in device and
device["maxOutputChannels"] > 0]
    stream = p.open(format=FORMAT, channels=CHANNELS, rate=RATE,
input=True, output=True, frames_per_buffer=CHUNK)
    send_sound_thread = threading.Thread(target=send_sound,
args=(stream, client_socket, server_addr), daemon=True)
    try:
```

```
        client_socket.settimeout(0.1)
        send_sound_thread.start()
        get_sound(stream, client_socket)
    except KeyboardInterrupt:
        pass
    finally:
        stop = True
        send_sound_thread.join(1)
        # Clean up PyAudio and close the connection
        stream.stop_stream()
        stream.close()
        p.terminate()
        client_socket.close()


if __name__ == '__main__':
    join_call(("127.0.0.1", 16400), "omer",
hashlib.md5("omer".encode()).hexdigest().lower())
```

ChatEaseGUI.py

```
"""
##############################################
Author: Omer Dagry
Mail: omerdagry@gmail.com
Date: 30/05/2023 (dd/mm/yyyy)


bindings between the communication.py to the eel website
and some more function to get chats & users data
##############################################
"""

import io
import multiprocessing
import os
import sys
import wave
import time
import json
import socket
import shutil
import pickle
import easygui
import hashlib
import imaplib
import pyaudio
```

```
import threading
import traceback
import email as email_lib
# for .exe
if not os.path.dirname(__file__).endswith("Client - PC html css
js"):
    os.chdir(os.path.dirname(__file__))  # change working dir to
were the .exe was unpacked in
    # no stderr & stdout because the .exe is created without a
console, so redirect it
    logfile = io.StringIO()
    sys.stdout = logfile
    sys.stderr = logfile
# import eel only after handling stdout and stderr
import eel

from calls_udp_client import join_call
from communication import Communication as Com
from ClientSecureSocket import ClientEncryptedProtocolSocket
from communication import signup_request,
send_confirmation_code, reset_password_request,
reset_password_choose_password


# Constants
SERVER_PORT = 8820

# Globals
email: None | str = None
username: None | str = None
password: None | str = None
communication: None | Com = None
sock: None | ClientEncryptedProtocolSocket = None
sync_sock: None | ClientEncryptedProtocolSocket = None
waiting_for_confirmation_code_reset: bool = False
waiting_for_confirmation_code_signup: bool = False
sync_thread: threading.Thread | None = None
first_time_sync_all: bool = True
open_chat_files_lock = threading.Lock()
open_chat_files: set[str] = set()
chat_folder: str = ""
stop_rec: bool = True
stop: bool = False
send_file_active: list[bool] = [False]
call_process: multiprocessing.Process | None = None
```

```
"""                                                         Chat
"""


@eel.expose
def get_all_chat_ids() -> str:
    """ returns all chat ids as json dict, {chat_ids:
[chat_name, last_msg, last_msg_time, chat_type, users]} """
    if not os.path.isdir(f"webroot\\{email}"):
        return json.dumps({})
    chat_ids = [chat_id for chat_id in
os.listdir(f"webroot\\{email}") if
os.path.isdir(f"webroot\\{email}\\{chat_id}")]
    if "profile_pictures" in chat_ids:
        chat_ids.remove("profile_pictures")
    if "recordings" in chat_ids:
        chat_ids.remove("recordings")
    # {chat_id,
    #  [chat_name, last_message, time, chat_type - group or the
email of the other user, users, num_of_unread_msgs]
    #  }
    chat_id_last_msg_and_time: dict[str, list[str, str, str,
str]] = {}
    for chat_id in chat_ids:
        try:
            with open(f"webroot\\{email}\\{chat_id}\\name",
"rb") as f:
                chat_name = pickle.loads(f.read())
            chat_type = "group" if len(chat_name) == 1 else "1
on 1"
            chat_name = chat_name[0] if len(chat_name) == 1 else
chat_name[0] if chat_name[0] != username \
                else chat_name[1]
            with open(f"webroot\\{email}\\{chat_id}\\users",
"rb") as f:
                users = list(pickle.loads(f.read()))
            latest_chat_msgs_file_name =
max(os.listdir(f"webroot\\{email}\\{chat_id}\\data\\chat"))
            with
open(f"webroot\\{email}\\{chat_id}\\data\\chat\\{latest_chat_msg
s_file_name}", "rb") as f:
                last_chat_msgs = pickle.loads(f.read())
            msgs_index = set(last_chat_msgs.keys())
            last_msg_index = max(msgs_index)
            last_msg = last_chat_msgs[last_msg_index]
            if email not in last_msg[3]:
                sender = last_msg[0].split('@')[0] if
```

```
last_msg[0] != email else "You"
                msg = f"{sender}: {last_msg[1]}"
                msg = msg if len(msg) <= 25 else msg[:25] +
"..."
            else:
                msg = ""
            msg_time = last_msg[-1]
            number_of_unread_msgs = 0
            if
os.path.isfile(f"webroot\\{email}\\{chat_id}\\unread_msgs"):
                with
open(f"webroot\\{email}\\{chat_id}\\unread_msgs", "rb") as f:
                    try:
                        unread_msgs_dict: dict =
pickle.loads(f.read())
                    except EOFError:
                        unread_msgs_dict = {}
                if email in unread_msgs_dict:
                    number_of_unread_msgs =
unread_msgs_dict[email]
            chat_id_last_msg_and_time[chat_id] = [chat_name,
msg, msg_time, chat_type, users, number_of_unread_msgs]
        except FileNotFoundError:
            pass
    return json.dumps(chat_id_last_msg_and_time)


@eel.expose
def get_user_last_seen(user_email: str) -> str:
    """ returns the time 'user_email' was last seen or 'Online'
if he is online """
    try:
        with open(f"webroot\\{email}\\users_status", "rb") as f:
            try:
                users_status: dict = pickle.loads(f.read())
            except EOFError:
                users_status = {}
    except FileNotFoundError:
        users_status = {}
    return users_status.get(user_email, "")


@eel.expose
def get_chat_msgs(chat_id: str) -> str:
    """ returns the last file of msgs + the file before it (max
1600 msgs) """
    global chat_folder, open_chat_files
```

```
    latest_file_path =
f"webroot\\{email}\\{chat_id}\\data\\chat\\"
    latest_file = max(os.listdir(latest_file_path))
    data: dict = {}
    open_chat_files = set()
    if latest_file != "0":
        with open(f"{latest_file_path}{int(latest_file) - 1}",
"rb") as f:
            data = pickle.loads(f.read())

open_chat_files.add(f"{latest_file_path}{int(latest_file) - 1}")
    latest_file_path += latest_file
    with open(latest_file_path, "rb") as f:
        data.update(pickle.loads(f.read()))
    open_chat_files_lock.acquire()
    open_chat_files.add(latest_file_path)
    chat_folder = os.path.dirname(latest_file_path)
    open_chat_files_lock.release()
    return json.dumps(data)


@eel.expose
def get_more_msgs() -> str:
    """ checks if there is an older chat file that isn't already
loaded, if there is it returns the msgs (800 msgs) """
    open_chat_files_lock.acquire()
    if chat_folder + "\\0" not in open_chat_files:  # no more
chat files to load
        with open(chat_folder +
f"\\{int(min(list(open_chat_files))) - 1}") as f:
            data = json.dumps(pickle.loads(f.read()))
    else:
        data = json.dumps({})
    open_chat_files_lock.release()
    return data


@eel.expose
def get_known_to_user() -> str:
    """ returns all the users that are known to the user """
    with open(f"webroot\\{email}\\known_users", "rb") as f:
        try:
            known_users: list[str] =
list(pickle.loads(f.read()))
        except EOFError:
            known_users = list()
    return json.dumps(dict(((i, user_email) for i, user_email in
```

```
enumerate(list(known_users)))))


"""                                                    Get User Info
"""


@eel.expose
def get_email() -> str:
    return email


@eel.expose
def get_username() -> str:
    return username


"""                                Sync With Server & Update Open
Chats In GUI                                                    """


def update(first_time_sync_mode: bool) -> None:
    """ syncs with the sever and updates the GUI """
    global sync_sock, stop
    while not stop:
        try:
            # sends nothing, waits for sync msg from server
            new_data, modified_files, deleted_files,
ongoing_calls = communication.sync(sync_sock)
        except (ConnectionError, socket.error, UnicodeError):
            sync_sock.close()
            status, sync_sock, reason = \
                communication.login_sync(verbose=False,
sync_mode="all" if first_time_sync_mode else "new")
            if not status:
                # TODO: display error (reason)
                break
            continue
        if new_data:
            try:
                if first_time_sync_mode:
                    raise AttributeError
                open_chat_id = eel.get_open_chat_id()()
            except AttributeError:  # GUI haven't loaded up yet
                open_chat_id = ""
            if open_chat_id != "":
                # let the server know that the user saw all the
```

```
messages in the chat
                mark_as_seen(open_chat_id)
                for file_path in modified_files:
                    if open_chat_id == file_path.split("\\")[2]
and file_path in open_chat_files:
                        try:
                            with open(file_path, "rb") as f:
                                data =
json.dumps(pickle.loads(f.read()))
                            eel.update(open_chat_id, data)()
                        except (pickle.UnpicklingError,
AttributeError):
                            pass
            # delete a file, if the user who sent it deleted the
msg
            for file_path in deleted_files:
                if os.path.isfile(file_path):
                    os.remove(file_path)
            if ongoing_calls.keys():
                # update GUI about new calls
                for group_name, port in ongoing_calls.items():
                    print(group_name, port)
                    eel.ongoing_call(group_name, port)()
                    print("called ongoing call")
        if first_time_sync_mode and new_data:
            os.makedirs(f"webroot\\{email}\\first sync done",
exist_ok=True)
            first_time_sync_mode = False


"""                                     Communication Wrapper
Functions                                                    """


@eel.expose
def mark_as_seen(open_chat_id: str) -> None:
    """ let the server know that the user saw all the messages
in the chat """
    if sync_sock is not None and open_chat_id is not None and
open_chat_id != "":
        communication.mark_as_seen(sync_sock, open_chat_id)


@eel.expose
def login(email_: str, password_: str) -> tuple[bool, str]:
    """ login & start sync """
    global communication, sock, email, password, username, sock,
```

```
first_time_sync_all, sync_sock
    if email_ is None or email_ == "" or password_ is None or
password_ == "":
        return False, ""
    if sock is not None:
        try:
            sock.close()
        except (ConnectionError, socket.error):
            pass
    sock = None
    password_ =
hashlib.md5(password_.encode()).hexdigest().lower()
    communication = Com(email_, password_, SERVER_IP_PORT)
    status, regular_sock, username_or_reason =
communication.login(verbose=False)
    if status:
        sock = regular_sock
        username = username_or_reason
        email = email_
        password = password_
        start_app()  # start sync thread
        while not os.path.isdir(f"webroot\\{email}\\first sync
done"):  # wait for first sync to finish
            time.sleep(0.01)
        return True, ""
    communication = None
    return False, username_or_reason


@eel.expose
def signup_stage1(email_: str, password_: str, username_: str) -
> tuple[bool, str]:
    """ make a request to signup """
    if email_ is None or username_ is None or email_ == "" or
username_ == "" or password_ is None or password_ == "":
        return False, ""
    global sock, email, password, username,
waiting_for_confirmation_code_signup
    password_ =
hashlib.md5(password_.encode()).hexdigest().lower()
    status, regular_sock, reason = signup_request(username_,
email_, password_, SERVER_IP_PORT, return_status=True)
    if status:
        sock = regular_sock
        username = username_
        email = email_
        password = password_
```

```
        waiting_for_confirmation_code_signup = True
        return True, reason
    return False, reason


@eel.expose
def signup_stage2(confirmation_code: str) -> bool:
    """ confirmation code for signup request """
    global waiting_for_confirmation_code_signup, sock
    if sock is None or not waiting_for_confirmation_code_signup
or confirmation_code is None or confirmation_code == "":
        return False
    waiting_for_confirmation_code_signup = False
    status = send_confirmation_code(sock, confirmation_code,
False, "signup")
    if not status:
        sock = None
    return status


@eel.expose
def reset_password_stage1(email_: str, username_: str) -> bool:
    """ make a request to reset password """
    if email_ is None or username_ is None or email_ == "" or
username_ == "":
        return False
    global sock, waiting_for_confirmation_code_reset
    status, regular_sock = reset_password_request(username_,
email_, SERVER_IP_PORT)
    if status:
        waiting_for_confirmation_code_reset = True
        sock = regular_sock
    return status


@eel.expose
def reset_password_stage2(confirmation_code: str, password_:
str) -> bool:
    """ confirmation code and password reset """
    global waiting_for_confirmation_code_reset, sock
    if sock is None or not waiting_for_confirmation_code_reset
or \
            password_ is None or confirmation_code is None or
password == "" or confirmation_code == "":
        return False
    waiting_for_confirmation_code_reset = False
    status = send_confirmation_code(sock, confirmation_code,
```

```
False, "reset")
    if not status:
        sock = None
        return False
    password_ =
hashlib.md5(password_.encode()).hexdigest().lower()
    status = reset_password_choose_password(sock, password_)
    if not status:
        sock = None
    return status


@eel.expose
def send_file(chat_id: str, file_path: str) -> None:
    """ send a file """
    global communication, send_file_active
    if send_file_active[0]:
        return None
    send_file_active[0] = True
    if chat_id == "" or chat_id is None:
        return None
    if os.path.isfile(file_path):
        communication.upload_file(chat_id, filename=file_path,
send_file_active=send_file_active)
        return None
    file_path = f"webroot\\{file_path}"
    if os.path.isfile(file_path):
        communication.upload_file(chat_id, filename=file_path,
send_file_active=send_file_active)
    elif file_path == "webroot\\":
        communication.upload_file(chat_id,
send_file_active=send_file_active)
    return None


@eel.expose
def send_message(message: str, chat_id: str) -> bool:
    """ send a message """
    global sock
    if chat_id == "" or message == "" or message is None or
chat_id is None:
        return False
    res = communication.send_message(chat_id, message, sock)
    if not res:
        sock.close()
        status, sock, reason =
communication.login(verbose=False)
```

```
        res = communication.send_message(chat_id, message, sock)
        if not res:
            pass
            # TODO: display error
    return res


@eel.expose
def familiarize_user_with(other_email: str) -> bool:
    """ check if other_email exists and if it does make him
"known" to this user """
    return communication.familiarize_user_with(other_email,
sock)


@eel.expose
def new_chat(other_email: str) -> bool:
    """ create a new chat (one on one) """
    return communication.new_chat(other_email, sock)


@eel.expose
def new_group(other_emails: list[str], group_name: str) -> bool:
    """ create a new group """
    print(other_emails, group_name)
    return communication.new_group(other_emails, group_name,
sock)[0]


@eel.expose
def add_user_to_group(other_email: str, chat_id: str) -> bool:
    """ add a user to a group """
    return communication.add_user_to_group(other_email, chat_id,
sock)


@eel.expose
def remove_user_from_group(other_email: str, chat_id: str) ->
bool:
    """ remove a user from a group """
    return communication.remove_user_from_group(other_email,
chat_id, sock)


@eel.expose
def make_call(chat_id: str) -> bool:
    """ start a call """
```

```
    global call_process
    call_server_port = communication.make_call(chat_id) if
chat_id != "" else None
    if call_server_port is None:
        return False
    if call_process is not None:
        call_process.kill()
        call_process = None
    call_process = multiprocessing.Process(
        # TODO:             change to SERVER_IP
        target=join_call, args=((SERVER_IP, call_server_port),
email, password,), daemon=True
    )
    call_process.start()
    return True


@eel.expose
def answer_call(port: int) -> None:
    global call_process
    if call_process is not None:
        call_process.kill()
        call_process = None
    call_process = multiprocessing.Process(
        # TODO:             change to SERVER_IP
        target=join_call, args=((SERVER_IP, port), email,
password,), daemon=True
    )
    call_process.start()


@eel.expose
def check_ongoing_call():
    """ returns True if there is an ongoing call otherwise False
"""
    global call_process
    if call_process is not None and call_process.is_alive():
        return True
    call_process = None
    return False


@eel.expose
def hang_up_call() -> None:
    """ exit call """
    global call_process
    if call_process is not None:
```

```
        call_process.kill()
        call_process = None


@eel.expose
def upload_profile_picture() -> bool:
    """ upload a new profile picture """
    return communication.upload_profile_picture()


@eel.expose
def upload_group_picture(chat_id: str) -> bool:
    """ upload a new picture for a group """
    return communication.upload_group_picture(chat_id)


@eel.expose
def delete_message_for_me(chat_id: str, message_index: int) ->
bool:
    """ delete a message for yourself """
    return communication.delete_message_for_me(chat_id,
message_index, sock)


@eel.expose
def delete_message_for_everyone(chat_id: str, message_index:
int) -> bool:
    """ delete a message for everyone """
    return communication.delete_message_for_everyone(chat_id,
message_index, sock)


"""                                                Recording
"""


@eel.expose
def start_recording(chat_id: str) -> bool:
    """ start audio recording """
    global stop_rec
    if stop_rec:
        stop_rec = False
        recording_thread = threading.Thread(target=record_audio,
args=(chat_id,), daemon=True)
        recording_thread.start()
        return True
    return False
```

```
def record_audio(chat_id: str) -> None:
    """ this function is the actual function the records audio
"""
    global stop_rec
    skip = False
    os.makedirs(f"webroot\\{email}\\recordings", exist_ok=True)
    num = max([int(num.split(".")[0]) for num in
os.listdir(f"webroot\\{email}\\recordings")] + [0])
    recording_file_path = f"webroot\\{email}\\recordings\\{num +
1}.wav"
    try:
        audio = pyaudio.PyAudio()
        stream = audio.open(format=pyaudio.paInt16, channels=1,
rate=44100, input=True, frames_per_buffer=1024)
        frames = []
        while not stop_rec:
            data = stream.read(1024)
            frames.append(data)
        stream.stop_stream()
        stream.close()
        audio.terminate()
        with open(recording_file_path, "wb") as f:
            sound_file = wave.open(f, "wb")
            sound_file.setnchannels(1)

sound_file.setsampwidth(audio.get_sample_size(pyaudio.paInt16))
            sound_file.setframerate(44100)
            sound_file.writeframes(b''.join(frames))
            sound_file.close()
    except Exception as e:  # in case there is no microphone
        # TODO: display error message
        skip = True
    finally:
        stop_rec = True
    if not skip:
        time.sleep(1)
        eel.display_recording_options(recording_file_path[8:],
chat_id)()
        # send_file(chat_id, recording_file_path)


@eel.expose
def stop_recording() -> bool:
    """ stop recording """
    global stop_rec
```

```
    if not stop_rec:
        stop_rec = True
        time.sleep(2)
        return True
    return False


@eel.expose
def delete_recording(recording_file_path: str):
    """ delete recording """
    if os.path.isfile(f"webroot\\{recording_file_path}"):
        os.remove(f"webroot\\{recording_file_path}")


"""                                                 Other Functions
"""


@eel.expose
def start_file(file_path: str) -> bool:
    """ open a file """
    file_path = file_path.replace("/", "\\")
    if os.path.isfile(file_path):
        os.startfile(file_path)
        return True
    elif os.path.isfile(f"webroot\\{file_path}"):
        os.startfile(f"webroot\\{file_path}")
        return True
    return False


@eel.expose
def close_program():
    """ called when there is a refresh / a redirect in order to
restart the sync """
    global sync_thread, stop, call_process
    stop = True
    if sync_thread is not None:
        sync_thread.join()
        if sync_sock is not None:
            sync_sock.close()
        sync_thread = None
        if call_process is not None:
            call_process.kill()
        call_process = None
```

```
def get_server_ip() -> str | None:
    """ try to get the server IP from the email that is shared
between all the clients """
    try:
        connection = imaplib.IMAP4_SSL("imap.gmail.com")

connection.login("project.twelfth.grade.get.ip@gmail.com",
"wkqakclcvgfwyitn")
        connection.select()
        result, data = connection.uid('search', None, "ALL")
        if result == 'OK':
            for num in reversed(data[0].split()):
                result, data = connection.uid('fetch', num,
'(RFC822)')
                if result == 'OK':
                    email_message =
email_lib.message_from_bytes(data[0][1])
                    from_email = str(email_message['From'])
                    if from_email !=
"project.twelfth.grade@gmail.com":
                        continue
                    subject = str(email_message['Subject'])
                    if subject == "server up":
                        content =
str(email_message.get_payload()[0])
                        return content.split('server_ip=')[-
1].strip()
                    elif subject == "server down":
                        return None
        connection.close()
        connection.logout()
    except Exception as e:
        traceback.format_exception(e)  # returns the formatted
exception
        return None


"""                                    Connect To Server & Start
GUI & Sync                                                    """


@eel.expose
def start_app() -> None:
    """ start sync """
    global sync_thread, sync_sock, first_time_sync_all, stop
    os.makedirs(f"webroot\\{email}\\", exist_ok=True)
    close_program()
```

```
    if sync_sock is not None:
        sync_sock.close()
    status, sync_sock, reason = \
        communication.login_sync(verbose=False, sync_mode="all"
if first_time_sync_all else "new")
    if not status:
        pass  # TODO: display error
        print("error restarting sync sock")
    stop = False
    if sync_thread is None or not sync_thread.is_alive():
        # Start sync thread
        sync_thread = threading.Thread(target=update,
args=(first_time_sync_all,), daemon=True)
        sync_thread.start()
        first_time_sync_all = False


def main():
    """ launch eel """
    # Launch GUI
    try:
        if os.path.isdir(f"webroot\\{email}\\first sync done"):
            shutil.rmtree(f"webroot\\{email}\\first sync done")
        eel.init("webroot")
        port = 8080
        while True:
            try:
                with socket.socket() as s:
                    s.bind(("127.0.0.1", port))
                break
            except OSError:  # port taken
                if port < 65535:
                    port += 1
                else:
                    raise Exception("Couldn't find an open port
for GUI local host.")
        eel.start("login.html", port=port, cmdline_args=["-
incognito"])
    except (Exception, BaseException) as e:
        if not isinstance(e, SystemExit) and not isinstance(e,
KeyboardInterrupt):
            traceback.print_exception(e)
    finally:
        if os.path.isdir(f"webroot\\{email}"):
            shutil.rmtree(f"webroot\\{email}")
            # shutil.rmtree(f"webroot\\{email}\\recordings")
            # shutil.rmtree(f"webroot\\{email}\\first sync
```

```
done")
        try:
            if sync_sock is not None:
                sync_sock.close()
        except (ConnectionError, socket.error):
            pass
        try:
            if sock is not None:
                sock.close()
        except (ConnectionError, socket.error):
            pass


if __name__ == '__main__':
    # More Constants
    # Server IP - try to get through clients shared email, if
not ask from user
    SERVER_IP = None  # get_server_ip()
    while SERVER_IP != "no" and \
            (SERVER_IP is None or SERVER_IP.count(".") != 3 or
not
            all((i.isnumeric() and -1 < int(i) < 256 for i in
SERVER_IP.split("")))):
        SERVER_IP = easygui.enterbox("Please Enter Server IP: ",
"Server IP")
    if SERVER_IP == "no":  # cancel run
        sys.exit(1)
    assert SERVER_IP.count(".") == 3 and all((i.isnumeric() and
-1 < int(i) < 256 for i in SERVER_IP.split("."))), \
        "Invalid Server IP"
    SERVER_IP_PORT = (SERVER_IP, SERVER_PORT)
    main()
```

communication.py

```
"""
##############################################
Author: Omer Dagry
Mail: omerdagry@gmail.com
Date: 30/05/2023 (dd/mm/yyyy)
##############################################
"""

import os
import pickle
import shutil
import socket
```

```
from tkinter import *
from typing import Literal
from threading import Thread
from tkinter import messagebox
from photo_tools import check_size
from tkinter.filedialog import askopenfilename
from ClientSecureSocket import ClientEncryptedProtocolSocket

# Constants
CALL = "call|"
REMOVE = "remove"
USER_STATUS = "update users_status"
SYNC_CODE = "sync".ljust(30).encode()


def showerror(title: str | None, message: str | None, **options)
-> None:
    """ display a little error window """
    print(f"Show error: {title = }: {message = }")
    Thread(target=messagebox.showerror, args=(title, message,),
kwargs=options).start()


def signup_request(username: str, email: str, password: str,
server_ip_port: tuple[str, int],
                   sock: ClientEncryptedProtocolSocket | None =
None, return_status: bool = False) \
        -> tuple[bool, None | ClientEncryptedProtocolSocket] |
tuple[bool, None | ClientEncryptedProtocolSocket, str]:
    """ signup first step """
    # signup (length 30)|len username (max 40)|username|len
email (length 10)|
    # email|password (fixed length - md5 hash length)
    signup_msg =
f"{'signup'.ljust(30)}{str(len(username)).ljust(2)}" \

f"{username}{str(len(email)).ljust(15)}{email}{password}".encode
()
    if sock is None:
        sock = ClientEncryptedProtocolSocket()
        sock.connect(server_ip_port)
    if not sock.send_message(signup_msg):
        sock.close()
        if not return_status:
            showerror("Signup Error", "Could not send signup
request, lost connection to server.")
```

```
            return False, None
        return False, None, "Lost connection to server !"
    try:
        response = sock.recv_message().decode()
    except (ConnectionError, socket.error):
        if not return_status:
            return True, sock
        return False, None, "Lost connection to server."
    if response != "signup".ljust(30):
        if not return_status:
            return False, None
        return False, None, response[36:]
    if not return_status:
        return True, sock
    return True, sock, ""


def send_confirmation_code(sock: ClientEncryptedProtocolSocket,
confirmation_code: str,
                           verbose: bool, signup_or_reset:
Literal["signup", "reset"]) -> bool:
    """ confirmation code (for signup and reset password) """
    try:
        confirmation_code_msg = sock.recv_message().decode()
    except (ConnectionError, socket.error):
        return False
    if confirmation_code_msg.strip() == "confirmation_code":
        if not
sock.send_message(f"{'confirmation_code'.ljust(30)}{confirmation
_code}".encode()):
            showerror(
                "Signup Error" if signup_or_reset == "signup"
else "Reset Password Error",
                "Could not send confirmation code, lost
connection to server."
            )
            sock.close()
            return False
    else:
        return False
    # signup (length 30)   status (length 6)   reason
    # reset password (length 30)   status (length 6)   reason
    try:
        response = sock.recv_message().decode()
    except (ConnectionError, socket.error):
        return False
    if (response[:30].strip() != "signup" and signup_or_reset ==
```

```
"signup") or \
            (response[:30].strip() != "reset password" and
signup_or_reset == "reset"):
        return False
    response = response[30:]
    if response[:6].strip() != "ok":
        response = response[6:]
        if verbose:
            print("Signup" if signup_or_reset == "Reset
Password" else f" Failed, Server Sent: {response}")
        return False
    return True


def signup(username: str, email: str, password: str,
server_ip_port: tuple[str, int], verbose: bool = True,
           sock: ClientEncryptedProtocolSocket | None = None,
login_after: bool = True) \
        -> tuple[bool, None | ClientEncryptedProtocolSocket]:
    """ signup full process """
    status, sock = signup_request(username, email, password,
server_ip_port, sock)
    if not status:
        return False, None
    status = send_confirmation_code(sock, input("Please Enter
The Confirmation Code: "), verbose, "signup")
    if not status:
        return False, None
    if verbose:
        print("Signed up Successfully.")
    if login_after:
        communication = Communication(email, password,
server_ip_port)
        ok, sock, username = communication.login(verbose, sock)
        if not ok:
            return False, None
    return True, sock


def reset_password_request(username: str, email: str,
server_ip_port: tuple[str, int]) \
        -> tuple[bool, ClientEncryptedProtocolSocket | None]:
    """ reset password first step """
    sock = ClientEncryptedProtocolSocket()
    sock.connect(server_ip_port)
    if not sock.send_message(f"{'reset
password'.ljust(30)}{str(len(email)).ljust(15)}{email}{username}
```

```
".encode()):
        showerror(
            "Reset Password Error", "Could not send reset
password request, lost connection to server.")
        sock.close()
        return False, None
    try:
        response = sock.recv_message().decode()
    except (ConnectionError, socket.error):
        return False, None
    if response != "reset password".ljust(30):
        sock.close()
        return False, None
    return True, sock


def reset_password_choose_password(sock:
ClientEncryptedProtocolSocket, password: str) -> bool:
    """ reset password last step """
    try:
        new_password_msg = sock.recv_message()
    except (ConnectionError, socket.error):
        return False
    if new_password_msg != f"{'new
password'.ljust(30)}".encode():
        sock.close()
        return False
    sock.send_message(f"{'new
password'.ljust(30)}{password}".encode())
    try:
        reset_password_status = sock.recv_message()
    except (ConnectionError, socket.error):
        return False
    if "not ok" in reset_password_status.decode() or
reset_password_status == b"":
        sock.close()
        return False
    return True


def reset_password(username: str, email: str, server_ip_port:
tuple[str, int], verbose: bool) \
        -> tuple[bool, ClientEncryptedProtocolSocket | None]:
    """ reset password full process, the returned socket isn't
logged in !! """
    status, sock = reset_password_request(username, email,
server_ip_port)
```

```
    if not status:
        return False, None
    status = send_confirmation_code(
        sock, input('Please enter your confirmation code (sent
to your email): '), verbose, "reset"
    )
    if not status:
        return False, None
    status = reset_password_choose_password(sock, input('Please
enter your new password: '))
    return status, sock if status else None


class Communication:
    """ a class that contains all the communications that
require a username and password """
    def __init__(self, email: str, password: str,
server_ip_port: tuple[str, int]) -> None:
        """
        :param email: the username
        :param password: the md5 hash of the real password
        :param server_ip_port: a tuple of the server IP and port
        """
        self.__email = email
        self.__password = password
        self.__server_ip_port = server_ip_port

    def login(self, verbose: bool = True, sock:
ClientEncryptedProtocolSocket | None = None) \
            -> tuple[bool, None | ClientEncryptedProtocolSocket,
str]:
        """ login """
        # login (length 30)     len email (length 10)   email
password (fixed length - md5 hash length)
        login_msg =
f"{'login'.ljust(30)}{str(len(self.__email)).ljust(15)}{self.__e
mail}{self.__password}".encode()
        if sock is None:
            sock = ClientEncryptedProtocolSocket()
            try:
                sock.connect(self.__server_ip_port)
            except (ConnectionError, socket.error):
                return False, None, "Can't reach the server."
        if not sock.send_message(login_msg):
            showerror("Login Error", "Could not send login
request, lost connection to server.")
            sock.close()
```

```
            return False, None, "Lost connection to server."
        # login (length 30)   status (length 6)   reason
        try:
            response = sock.recv_message().decode()
        except (ConnectionError, socket.error):
            return False, None, "Lost connection to server."
        if response[:30].strip() != "login":
            return False, None, "Error"
        response = response[30:]
        if response[:6].strip() != "ok":
            response = response[6:]
            if verbose:
                print(f"Login Failed, Server Sent: {response}")
            sock.close()
            return False, None, response
        if verbose:
            print("Logged in Successfully.")
        return True, sock, response[6:]

    def login_sync(self, verbose: bool = True, sock:
ClientEncryptedProtocolSocket | None = None,
                   sync_mode: str = "all") -> tuple[bool, None |
ClientEncryptedProtocolSocket, str]:
        """ login & let the server know this connection is to
sync data """
        status, sock, reason = self.login(verbose=verbose,
sock=sock)
        if status:
            sync_sock_notify_msg = f"{f'this is a sync sock
{sync_mode}'.ljust(30)}".encode()
            if not sock.send_message(sync_sock_notify_msg):
                sock.close()
                return False, None, "Error notifying the server
about sync sock"
        return status, sock, reason

    def sync(self, sock: ClientEncryptedProtocolSocket) ->
tuple[bool, list[str], list[str], dict[str, int]]:
        """ sync once
        :param sock: this sock must be logged in using
login_sync
        :return: True if new data received else False, list of
the modified/new files, list of deleted files/folders
        """
        response = sock.recv_message(timeout=1)
        #                           cmd                  {}
str     bytes | str
```

```
        # response -> f"{'sync new/all'.ljust(30)}{empty-
dict/dict[file_name, file_data]}"
        if response[:30] != SYNC_CODE:
            return False, [], [], {}
        try:
            files_dict = pickle.loads(response[30:])
        except EOFError:
            files_dict = {}
        if files_dict:
            deleted_files_path: list[str | os.PathLike] = []
            modified_files_path: list[str | os.PathLike] = []
            ongoing_calls: dict[str, int] = {}
            for file_path, file_data in files_dict.items():
                file_data: bytes
                if file_path.startswith(self.__email):
                    file_path = f"webroot\\{file_path}"
                else:
                    file_path =
f"webroot\\{self.__email}\\{file_path}"
                # if it's a not remove message
                # a remove message will be after a request of a
client
                # to delete message for everyone, if the message
is a file
                # in order to delete the file on the clients
side
                if file_data != REMOVE and CALL not in
file_path:
                    modified_files_path.append(file_path)
                    for _ in range(2):
                        try:
                            with open(file_path, "wb") as f:
                                f.write(file_data)
                            break
                        except FileNotFoundError:

os.makedirs("\\".join(file_path.split("\\")[:-1]))
                elif CALL in file_path:
                    file_data: str

ongoing_calls["|".join(file_data.split("|")[2:])] =
int(file_data.split("|")[1])
                elif os.path.isfile(file_path):  # a file was
deleted in a chat
                    deleted_files_path.append(file_path)
                    os.remove(file_path)
                elif os.path.isdir(file_path):  # the user was
```

```
removed from the chat
                    deleted_files_path.append(file_path)
                    shutil.rmtree(file_path)
            return True, modified_files_path,
deleted_files_path, ongoing_calls
        else:
            return False, [], [], {}

    def upload_file(self, chat_id: str | int, filename: str =
"", root: Tk = None,
                    delete_file: bool = False, send_file_active:
list[bool] = None) -> None:
        """ upload a file """
        upload_thread = Thread(
            target=self.upload_file_, args=(str(chat_id),
filename, delete_file, send_file_active), daemon=True
        )
        upload_thread.start()
        if root is not None:
            root.destroy()

    def upload_file_(self, chat_id: str, filepath: str,
delete_file: bool, send_file_active: list[bool]) -> None:
        """ upload a file (don't call this func, call
upload_file_) """
        if filepath == "":
            root = Tk()
            root.attributes('-topmost', True)  # Display the
dialog in the foreground.
            root.iconify()  # Hide the little window.
            filepath = askopenfilename(parent=root)
            root.destroy()
            if send_file_active:
                send_file_active[0] = False
            if filepath == "" or filepath is None or not
os.path.isfile(filepath):
                return
        elif send_file_active:
            send_file_active[0] = False
        ok, sock, _ = self.login(verbose=False)
        if not ok:
            raise ValueError("email or password incorrect, could
not login to upload file")
        with open(filepath, "rb") as f:
            file_data = f.read()
        file_name = filepath.split("/")[-1]
        file_name = file_name.split("\\")[-1]
```

```
        request =
f"{'file'.ljust(30)}{str(len(chat_id)).ljust(15)}{chat_id}" \

f"{str(len(file_name)).ljust(15)}{file_name}".encode() +
file_data
        if not sock.send_message(request):
            showerror("Failed to upload file", "Could not upload
the file, lost connection to server.")
            return
        if delete_file:
            os.remove(filepath)
        sock.close()

    @staticmethod
    def send_message(chat_id: str | int, msg: str, sock:
ClientEncryptedProtocolSocket) -> bool:
        """ send a message """
        if len(msg) > 5000:
            showerror("Message To Long", f"Message length is
{len(msg)}, and the maximum is 4999")
            return False
        chat_id = str(chat_id)
        request =
f"{'msg'.ljust(30)}{str(len(chat_id)).ljust(15)}{chat_id}{msg}".
encode()
        if not sock.send_message(request):
            showerror("Failed to send message", f"Could not send
the message, lost connection to server.")
            return False
        try:
            status_msg = sock.recv_message().decode()
        except (ConnectionError, socket.error):
            return False
        if "not ok" in status_msg or status_msg == "":
            showerror("Failed to send message", f"Could not send
the message, server error.")
            return False
        return True

    @staticmethod
    def familiarize_user_with(other_email: str, sock:
ClientEncryptedProtocolSocket) -> bool:
        """ search for a user that isn't "known" to me, and make
him "known" to me """
        request = f"{'familiarize user
with'.ljust(30)}{other_email}".encode()
        if not sock.send_message(request):
```

```
            showerror(f"Failed to familiarize user", "lost
connection to server.")
            return False
        try:
            response = sock.recv_message()
        except (ConnectionError, socket.error):
            return False
        if "not ok" in response.decode() or response == b"":
            # showerror(f"Failed to familiarize user",
response.split(b"not ok")[1].decode())
            return False
        return True

    @staticmethod
    def new_chat(other_email: str, sock:
ClientEncryptedProtocolSocket) -> bool:
        """ create a new chat (1 on 1) """
        request = f"{'new
chat'.ljust(30)}{other_email}".encode()
        if not sock.send_message(request):
            showerror(f"Failed to create new chat with
'{other_email}'", "lost connection to server.")
            return False
        try:
            response = sock.recv_message()
        except (ConnectionError, socket.error):
            return False
        if "not ok" in response.decode() or response == b"":
            showerror(f"Failed to create new chat with
'{other_email}'", "server error.")
            return False
        return True

    @staticmethod
    def new_group(other_emails: list[str], group_name: str,
sock: ClientEncryptedProtocolSocket) -> tuple[bool, str]:
        """ create new group """
        request = f"{'new
group'.ljust(30)}{str(len(group_name)).ljust(15)}{group_name}".e
ncode() + \
                  pickle.dumps(other_emails)
        if not sock.send_message(request):
            showerror(f"Failed to create new group", "lost
connection to server.")
            return False, ""
        try:
            response = sock.recv_message().decode()
```

```
        except (ConnectionError, socket.error):
            return False, ""
        if "not ok" in response or response == "":
            showerror(f"Failed to create new group", "server
error.")
            return False, ""
        chat_id = response.split("ok")[-1].strip()
        return True, chat_id

    @staticmethod
    def add_user_to_group(other_email: str, chat_id: str, sock:
ClientEncryptedProtocolSocket) -> bool:
        """ add a user to group """
        request = f"{'add
user'.ljust(30)}{str(len(chat_id)).ljust(15)}{chat_id}{other_ema
il}".encode()
        if not sock.send_message(request):
            showerror(f"Failed to add '{other_email}' to group",
"lost connection to server.")
            return False
        try:
            status_msg = sock.recv_message()
        except (ConnectionError, socket.error):
            return False
        if "not ok" in status_msg.decode() or status_msg == b"":
            showerror(f"Failed to add '{other_email}' to group",
"server error.")
            return False
        return True

    @staticmethod
    def remove_user_from_group(other_email: str, chat_id: str,
sock: ClientEncryptedProtocolSocket) -> bool:
        """ remove a user from the group """
        request = f"{'remove
user'.ljust(30)}{str(len(chat_id)).ljust(15)}{chat_id}{other_ema
il}".encode()
        if not sock.send_message(request):
            showerror(f"Failed to remove '{other_email}' from
group", "lost connection to server.")
            return False
        try:
            status_msg = sock.recv_message()
        except (ConnectionError, socket.error):
            return False
        if "not ok" in status_msg.decode() or status_msg == b"":
            showerror(f"Failed to remove '{other_email}' from
```

```
group", "server error.")
            return False
        return True

    def make_call(self, chat_id: str) -> int | None:
        """
        send a request to the server to start a server for this
call and
        notify the other users in the chat, and return the port
for this call
        """
        ok, sock, _ = self.login(verbose=False)
        if not ok:
            raise ValueError("email or password incorrect, could
not login to upload file")
        if not
sock.send_message(f"{'call'.ljust(30)}{chat_id}".encode()):
            showerror(f"Failed to make a call", "lost connection
to server.")
            return None
        try:
            port_message = sock.recv_message().decode()
        except (ConnectionError, socket.error):
            return None
        if "not ok" in port_message or port_message == "":
            showerror(f"Failed to make a call", "server error.")
            return None
        return int(port_message.split("ok")[1].strip())

    def upload_profile_picture(self, path_to_picture:
os.PathLike | str = None) -> bool:
        """ upload profile picture """
        if path_to_picture is None:  # ask for file
            file_types = [("PNG", "*.png"), ("JPG", "*.jpg"),
("JPEG", "*.jpeg")]
            root = Tk()
            root.attributes('-topmost', True)  # Display the
dialog in the foreground.
            root.iconify()  # Hide the little window.
            path_to_picture =
askopenfilename(filetypes=file_types)
            root.destroy()
            if path_to_picture == "" or path_to_picture is None
or not os.path.isfile(path_to_picture):
                return False
        if not check_size(path_to_picture):  # check image size
            showerror("Profile Picture", "Image size is
```

```
invalid,\nmust be at least 64x64.")
            return False
        ok, sock, _ = self.login(verbose=False)
        if not ok:
            raise ValueError("email or password incorrect, could
not login to upload file")
        with open(path_to_picture, "rb") as f:
            file_data = f.read()
        request = f"{'upload profile
picture'.ljust(30)}".encode() + file_data
        if not sock.send_message(request):
            showerror(
                "Upload Profile Picture Error", "Could not
upload the file, lost connection to server.")
            return False
        sock.close()
        return True

    def upload_group_picture(self, chat_id: str,
path_to_picture: os.PathLike | str = None) -> bool:
        """ upload group picture """
        if path_to_picture is None:  # ask for file
            file_types = [("PNG", "*.png"), ("JPG", "*.jpg"),
("JPEG", "*.jpeg")]
            root = Tk()
            root.attributes('-topmost', True)  # Display the
dialog in the foreground.
            root.iconify()  # Hide the little window.
            path_to_picture =
askopenfilename(filetypes=file_types)
            root.destroy()
            if path_to_picture == "" or path_to_picture is None
or not os.path.isfile(path_to_picture):
                return False
        if not check_size(path_to_picture):  # check image size
            showerror("Group Picture", "Image size is
invalid,\nmust be at least 64x64.")
            return False
        ok, sock, _ = self.login(verbose=False)
        if not ok:
            raise ValueError("email or password incorrect, could
not login to upload file")
        with open(path_to_picture, "rb") as f:
            file_data = f.read()
        request = f"{'upload group
picture'.ljust(30)}{str(len(chat_id)).ljust(15)}{chat_id}".encod
e() + file_data
```

```
        if not sock.send_message(request):
            showerror(
                "Upload Group Picture Error", "Could not upload
the file, lost connection to server.")
            return False
        sock.close()
        return True

    @staticmethod
    def delete_message_for_me(chat_id: str, message_index: int,
sock: ClientEncryptedProtocolSocket) -> bool:
        """ delete message for me """
        request = f"{'delete for
me'.ljust(30)}{str(len(chat_id)).ljust(15)}{chat_id}{message_ind
ex}"
        if not sock.send_message(request.encode()):
            showerror("Delete Message For Me Error", "Could not
delete the message.")
            return False
        return True

    @staticmethod
    def delete_message_for_everyone(chat_id: str, message_index:
int, sock: ClientEncryptedProtocolSocket) -> bool:
        """ delete message for everyone """
        request = f"{'delete for
everyone'.ljust(30)}{str(len(chat_id)).ljust(15)}{chat_id}{messa
ge_index}"
        if not sock.send_message(request.encode()):
            showerror("Delete Message For Me Error", "Could not
delete the message.")
            return False
        return True

    @staticmethod
    def mark_as_seen(sock: ClientEncryptedProtocolSocket,
chat_id: str) -> None:
        """ mark all the messages in the chat as seen """
        request = f"{'user in chat'.ljust(30)}{chat_id}"
        if not sock.send_message(request.encode()):
            showerror("User in chat Error", "Lost connection to
server.")
```

photo_tools.py

```
import os
import numpy as np
```

```
from PIL import Image, ImageDraw


def format_photo(path: os.PathLike | str):
    """ resizes the image and makes it round """
    # --------------------------- make image round -------------
--------------
    img = Image.open(path).convert("RGB")
    np_image = np.array(img)
    alpha = Image.new('L', img.size, 0)
    draw = ImageDraw.Draw(alpha)
    draw.pieslice(((0, 0), img.size), 0, 360, fill=255)
    np_alpha = np.array(alpha)
    np_image = np.dstack((np_image, np_alpha))
    # --------------------------- resize image -----------------
----------
    img = Image.fromarray(np_image)
    img.thumbnail((64, 64), Image.Resampling.LANCZOS)
    path = ".".join(path.split(".")[:-1]) + ".png"
    img.save(path, "png")


def check_size(path: os.PathLike | str) -> bool:
    """ :return: True if size is valid, otherwise False """
    img = Image.open(path)
    if img.size[0] >= 64 <= img.size[1]:
        return True
    return False


if __name__ == '__main__':
    pass
```