# 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

[אפליקציה לתקשורת בין אנשים – שליחת הודעות, קבצים וביצוע שיחות - ChatEase 1](#_Toc136660289)

[תוכן עניינים 2](#_Toc136660290)

[1. מבוא 4](#_Toc136660291)

[1.1 תכולת ספר הפרויקט 4](#_Toc136660292)

[1.2 הרקע לפרויקט 4](#_Toc136660293)

[1.3 תהליך המחקר 4](#_Toc136660294)

[1.4 אתגרים מרכזיים 4](#_Toc136660295)

[2. מבנה / ארכיטקטורה 6](#_Toc136660296)

[2.1 Front-End 7](#_Toc136660297)

[2.1.1 Login Page 7](#_Toc136660298)

[2.1.2 Signup Page 8](#_Toc136660299)

[2.1.3 Reset Password Page 9](#_Toc136660300)

[2.1.4 האפליקציה עצמה – ChatEase 10](#_Toc136660301)

[2.2 Back-End 11](#_Toc136660302)

[3. מדריך למשתמש 14](#_Toc136660303)

[3.1 צד לקוח 14](#_Toc136660304)

[3.2 צד שרת 15](#_Toc136660305)

[4. בסיס נתונים 16](#_Toc136660306)

[5. מדריך למפתח 18](#_Toc136660307)

[5.1 צד לקוח 18](#_Toc136660308)

[המודול 18](#_Toc136660309)

[התיקייה 19](#_Toc136660310)

[מחוץ לתיקיות ולמודולים - 19](#_Toc136660311)

[5.2 צד שרת 21](#_Toc136660312)

[המודול DirectoryLock 21](#_Toc136660313)

[המודול ServerSecureSocket 21](#_Toc136660314)

[המודול SyncDB 21](#_Toc136660315)

[מחוץ למודולים - Other Files 22](#_Toc136660316)

[5.3 פרוטוקול התקשורת בין השרת ללקוח 23](#_Toc136660317)

[6. רפלקציה 24](#_Toc136660318)

[7. ביבליוגרפיה 25](#_Toc136660319)

[8. נספחים 26](#_Toc136660320)

[8.0 קישור לrepository של הGithub של הפרויקט 26](#_Toc136660321)

[8.1 צד שרת 26](#_Toc136660322)

[DirectoryLock 26](#_Toc136660323)

[\_\_init\_\_.py 26](#_Toc136660324)

[dirlock.py 26](#_Toc136660325)

[ServerSecureSocket 27](#_Toc136660326)

[\_\_init\_\_.py 27](#_Toc136660327)

[aes.py 27](#_Toc136660328)

[server\_encrypted\_protocol\_socket.py 28](#_Toc136660329)

[SyncDB 31](#_Toc136660330)

[\_\_init\_\_.py 31](#_Toc136660331)

[database.py 31](#_Toc136660332)

[file\_database.py 32](#_Toc136660333)

[sync\_database.py 35](#_Toc136660334)

[Other Files 42](#_Toc136660335)

[calls\_udp\_server.py 42](#_Toc136660336)

[server.py 48](#_Toc136660337)

[ServerGUI.py 91](#_Toc136660338)

[8.2 צד לקוח 97](#_Toc136660339)

[ClientSecureSocket 97](#_Toc136660340)

[\_\_init\_\_.py 97](#_Toc136660341)

[aes.py 97](#_Toc136660342)

[client\_encrypted\_protocol\_socket.py 97](#_Toc136660343)

[webroot 100](#_Toc136660344)

[ChatEase.css 100](#_Toc136660345)

[ChatEase.html 113](#_Toc136660346)

[ChatEase.js 115](#_Toc136660347)

[login&signup&reset.css 145](#_Toc136660348)

[login.html 149](#_Toc136660349)

[login.js 150](#_Toc136660350)

[messages.js 151](#_Toc136660351)

[reset\_password.html 154](#_Toc136660352)

[reset\_password.js 155](#_Toc136660353)

[selection.js – not in use 158](#_Toc136660354)

[signup.html 159](#_Toc136660355)

[signup.js 160](#_Toc136660356)

[Other Files 163](#_Toc136660357)

[calls\_udp\_client.py 163](#_Toc136660358)

[ChatEaseGUI.py 166](#_Toc136660359)

[communication.py 183](#_Toc136660360)

[photo\_tools.py 197](#_Toc136660361)

# 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.

**משתמש**

בקשת מידע/שליחת מידע

בקשה לשליחת מידע לשרת/בקשת מידע מהשרת

פלט, פקודות

התראות

עדכון הGUI

קבלת מידע

**Server – Back End**

**Python – Front End**

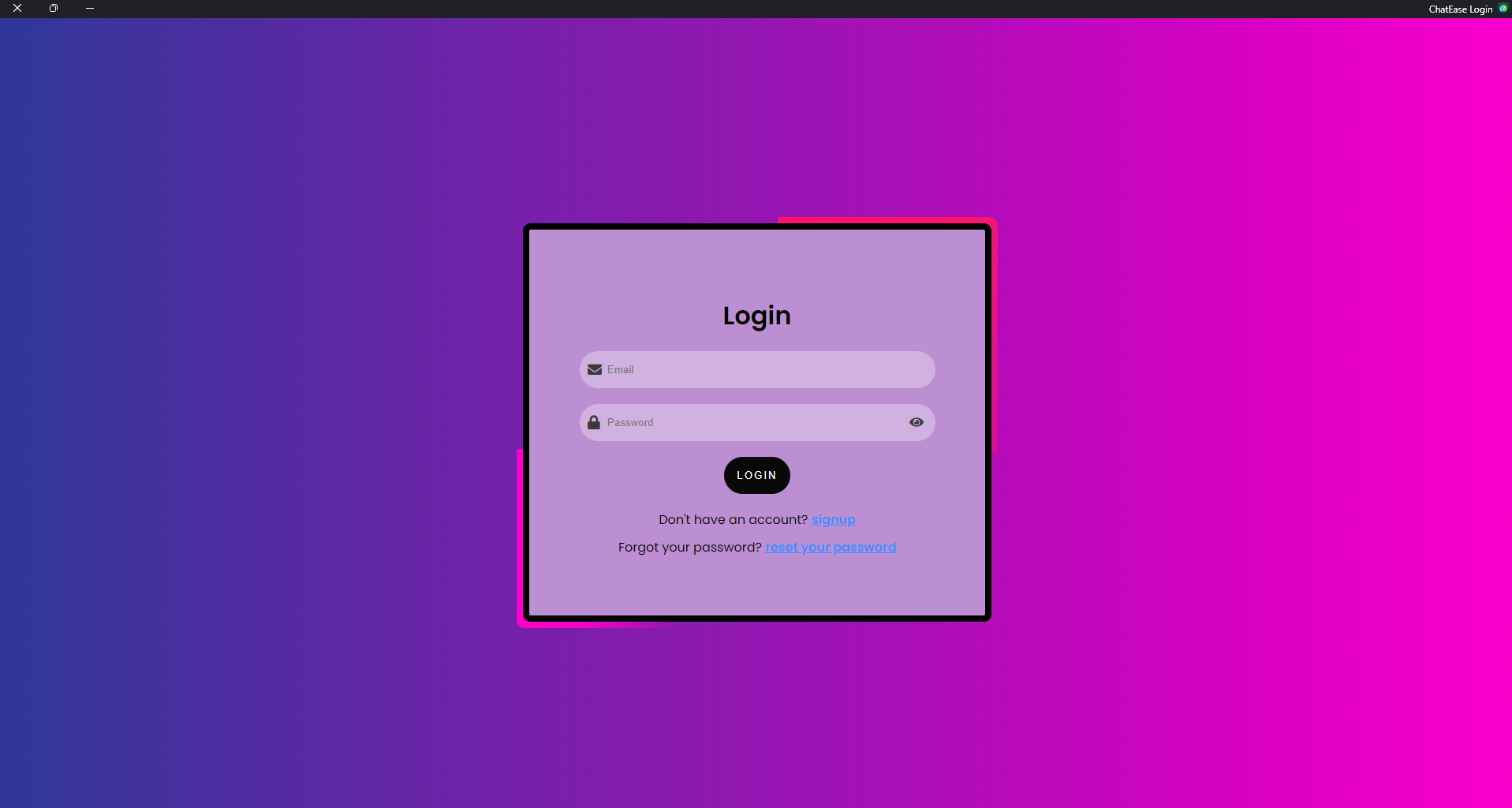
**HTML CSS JS**

## 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.

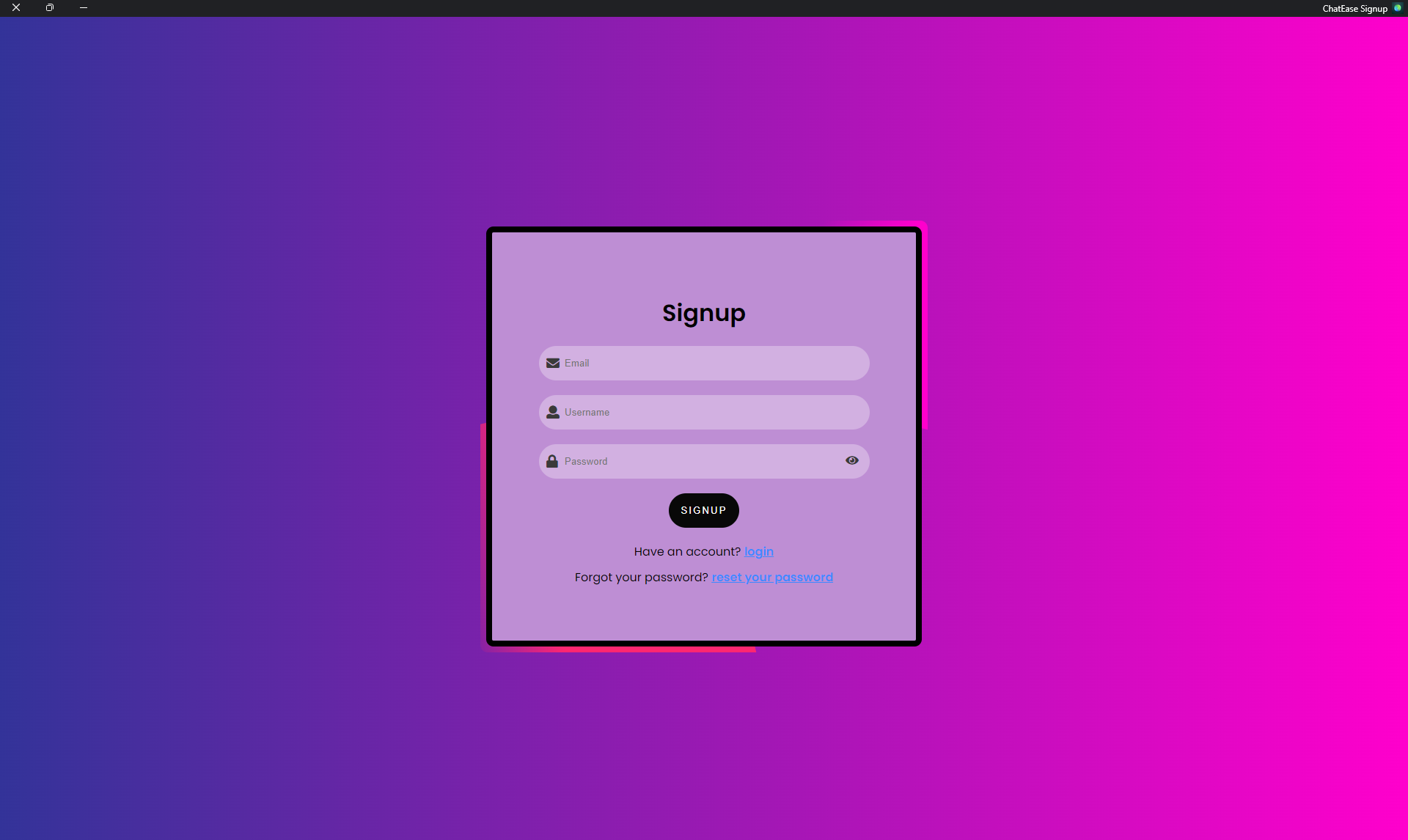
עבור כל אחד מהעמודים האלו יש קובץ html אחד, קובץ css אחד וקובץ js אחד או יותר.  
כל event שדורש תקשורת עם השרת או קבלת מידע חדש על מנת לעדכן את הGUI הjs מתקשר עם הpython ומקבל את מה שהוא צריך על מנת לבצע את הפעולה. זאת אומרת שעבור כל event כזה יש פונקציה גם בpython שתטפל בה וגם בjs.

### 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).  


### 2.1.2 Signup Page

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).

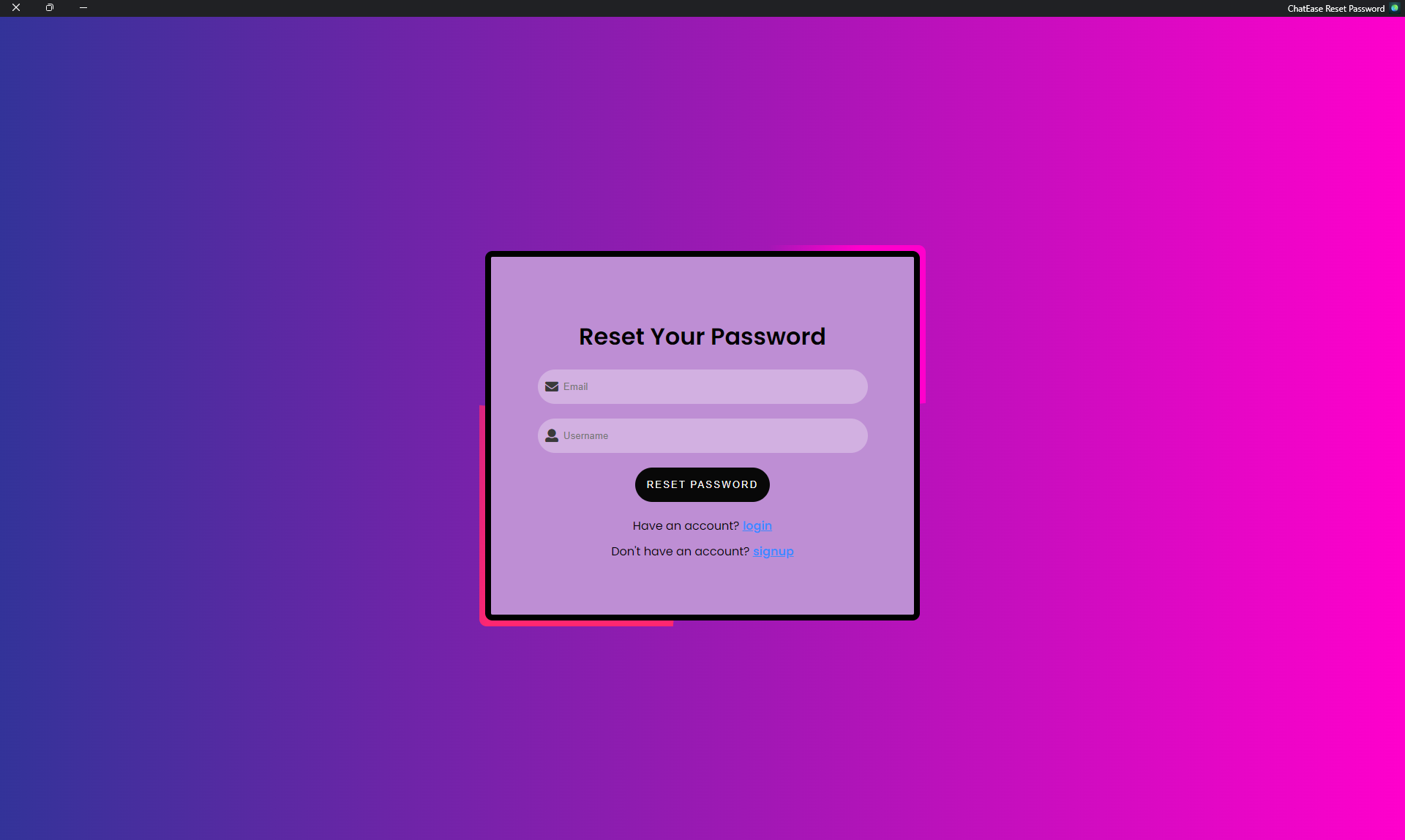


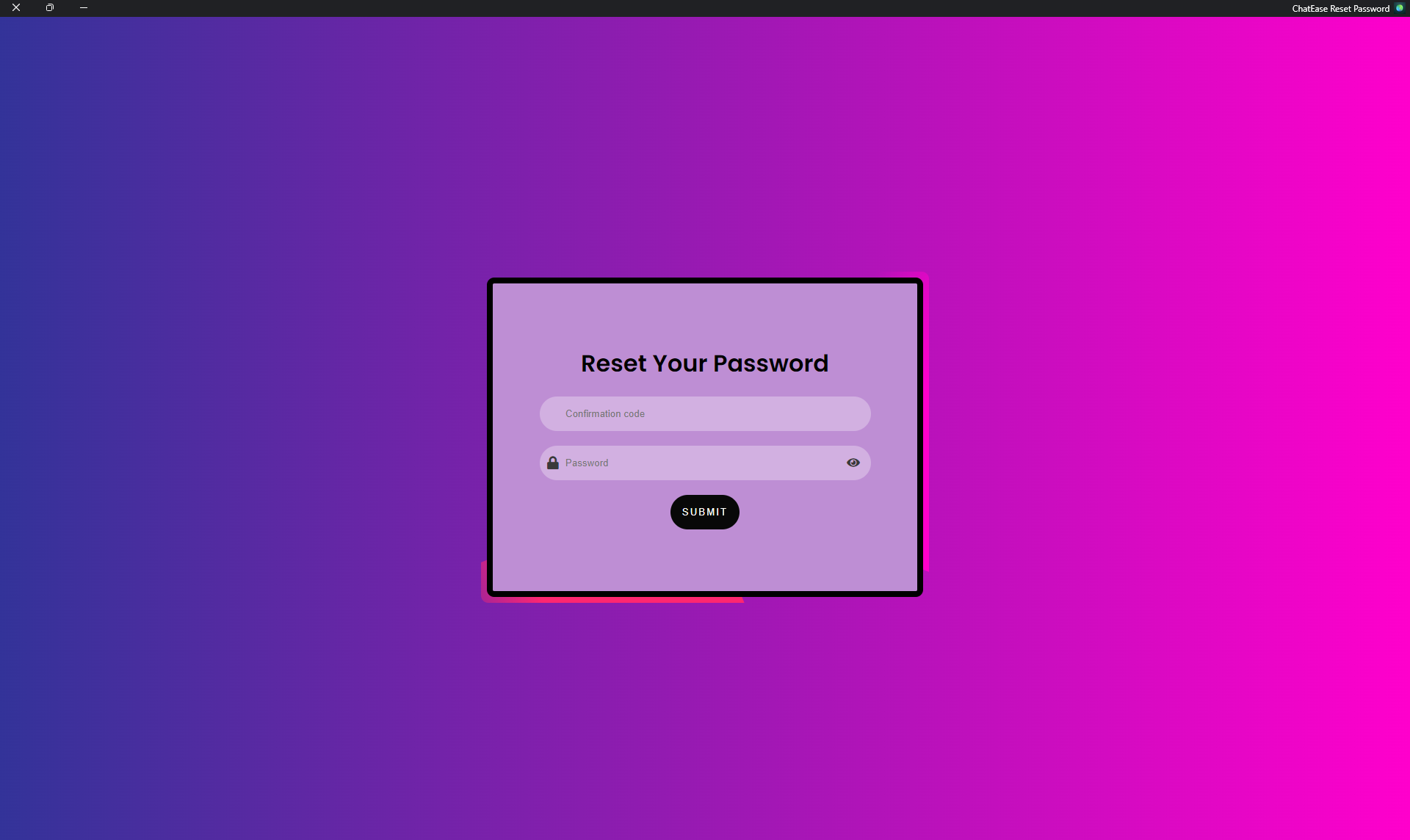
A screenshot of a computer

Description automatically generated with medium confidence

### 2.1.3 Reset Password Page

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.

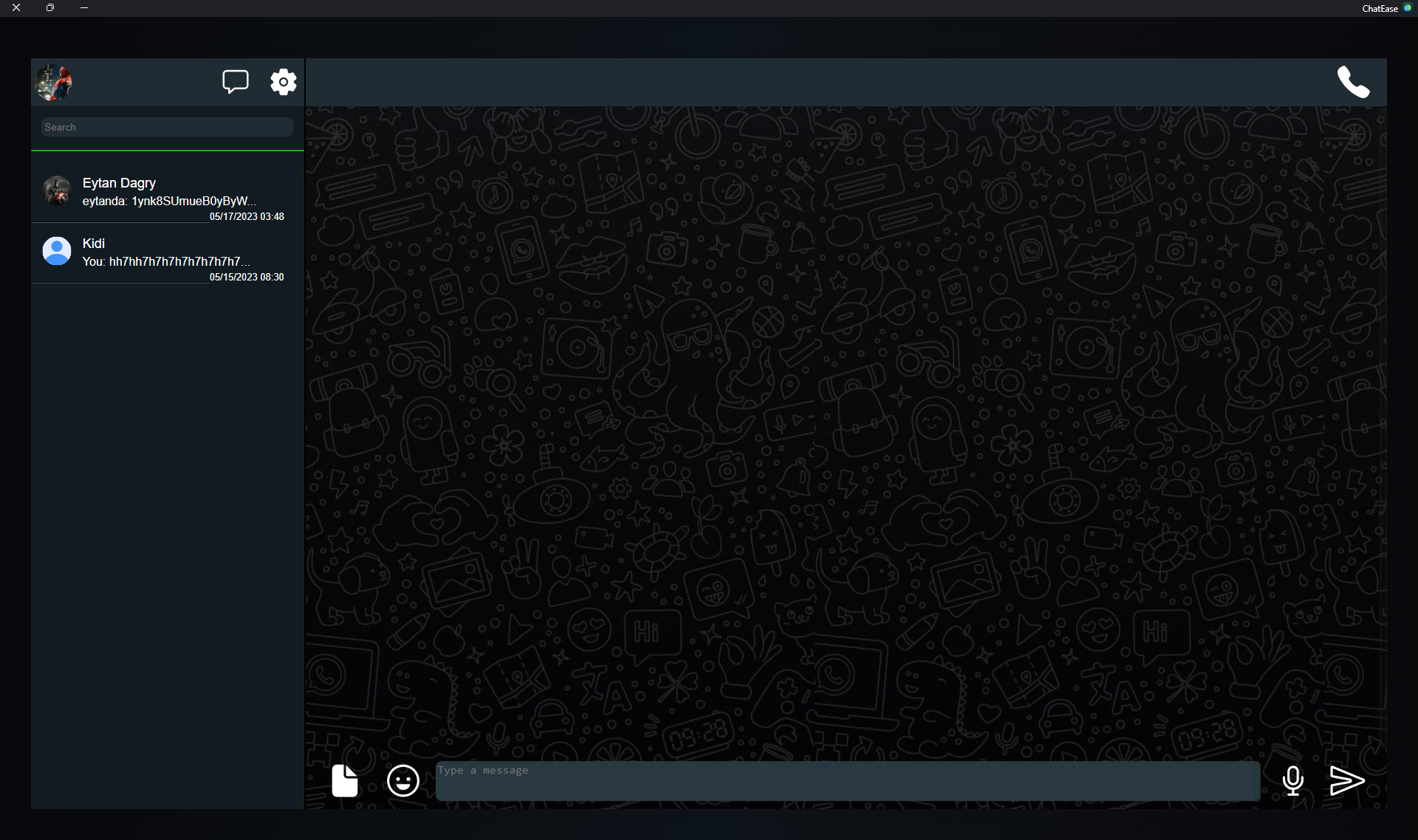




### 2.1.4 האפליקציה עצמה – ChatEase

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.**



A screenshot of a computer

Description automatically generated with medium confidence

## 2.2 Back-End

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).

A screenshot of a computer

Description automatically generated with medium confidence

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.

Client Front-End

Response with or without data

Request with or without data

Return the status and some information if there is

Transfer request to the needed function

**handle\_client**

Some function to handle this specific request

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.

Identifies him self to the call server over a TCP encrypted connection, and sends voice packets over non-secure UDP connection

Sends the client all the voice packets of the other clients over non-secure UDP connection

A dictionary with the clients that are part of the call and their passwords

The port of the call server

Request to start a call

Call server

Main server

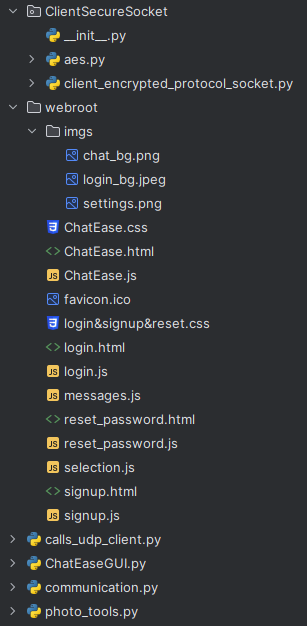
client

# 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 [2.1 Front-End](#_2.1_Front-End).

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



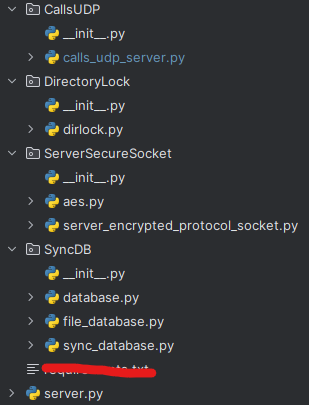
בנוסף בפייתון צריך את הספריות הבאות:  
eel  
rsa  
numpy  
pillow  
pyaudio  
easygui  
pycryptodome

והגרסת פייתון צריכה להיות 3.11 ומעלה בעקבות הצורה של התיעוד של הtypes 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 [2.2 Back-End](#_2.2_Back-End).

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 הtypes 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:  
A screenshot of a computer

Description automatically generated with medium confidence

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:  
A screenshot of a computer

Description automatically generated

**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

A screenshot of a computer program

Description automatically generated with medium confidence

המודול [ClientSecureSocket](#_ClientSecureSocket)  
1. [\_\_init\_\_.py](#___init__.py_3)  
2. [aes.py](#_aes.py_1)  
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](#_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](#_webroot)  
*1. התיקייה imgs* - מכילה 3 תמונות עבור הGUI  
*2. [ChatEase.css](#_ChatEase.css)* - הקובץ עיצוב של האפליקציה עצמה  
*3.* [*ChatEase.html*](#_ChatEase.html) - הקובץ הבסיסי לאפליקציה עצמה, לא מכיל יותר מידי רק את השלד של האתר כל השאר נעשה מהjs.  
*4.* [*ChatEase.js*](#_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*](#_login&signup&reset.css) – הקובץ עיצוב של העמודים של ההתחברות, הרשמה ואיפוס סיסמה.  
*7.* [*login.html*](#_login.html) *– מכיל את הדף התחברות*8. [login.js](#_login.js)*– מכיל כמה פונקציות בודדות אשר מעבירות לpython את השם משתמש והסיסמה כדי שיוכל להזדהות מול השרת.*9. [messages.js](#_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](#_reset_password.html)*– מכיל את הדף לאיפוס סיסמה.*11. [reset\_password.js](#_reset_password.js)*– מכיל כמה פונקציות בודדות לתקשורת עם הpython.*12. [selection.js – not in use](#_selection.js_–_not)*– לא קיים יותר. היה בהתחלה כדי לא לאפשר לבחור טקסט ואז גיליתי שאפשר לעשות את זה מהcss.*13. [signup.html](#_signup.html)*– מכיל את הדף להרשמה.*14. [signup.js](#_signup.js) *– מכיל כמה פונקציות בודדות לתקשורת עם הpython.*

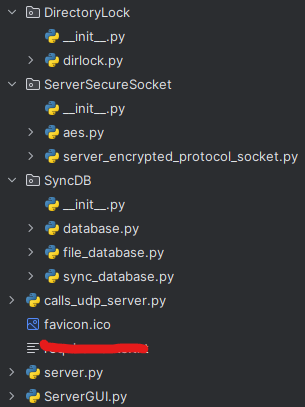
מחוץ לתיקיות ולמודולים - [Other Files](#_Other_Files)  
*1. [communication.py](#_communication.py)*– מכיל פונקציות להרשמה ואיפוס סיסמה, ומכיל מחלקה בשם Communication אשר משמשת לביצוע כל התקשורת עם השרת ברגע שהצד של הלקוח יודע מה השם משתמש והסיסמה של הלקוח, במחלקה זו ישנם 17 פונקציות:  
1.1. login – להתחברות 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 לupload\_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](#_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](#_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](#_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](#_DirectoryLock)

1. [\_\_init\_\_.py](#___init__.py_2)
2. [dirlock.py](#_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](#_ServerSecureSocket)

1. [\_\_init\_\_.py](#___init__.py_1)
2. [aes.py](#_aes.py)– כמו בלקוח [5.1 צד לקוח](#_5.1_צד_לקוח).
3. [server\_encrypted\_protocol\_socket.py](#_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](#_SyncDB)

1. [\_\_init\_\_.py](#___init__.py)
2. [database.py](#_database.py)– מכיל מחלקה אחת, הכי בסיסית, בסך הכל עוטפת את המילון של python.
3. [file\_database.py](#_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](#_sync_database.py)– מכיל מחלקה אשר מקבלת עצם של FileDatabase אשר נמצאת בקובץ file\_database.py ומאפשרת set וget 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](#_Other_Files_1)

[calls\_udp\_server.py](#_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](#_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](#_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 ולcss 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](https://docs.python.org/3/library/tkinter.messagebox.html" \l "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).encode() + 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().encode()).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\_chats", "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\_chats", "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\_msgs", "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\_msgs", "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\_pos]}")  
 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.png", "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)}".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()  
 # 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)}".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()  
 # 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/${other\_user\_email}\_profile\_picture.png")`;  
 }  
 changed\_buttons.push(chat\_box\_left(picture\_path, chat\_name, last\_message, time, chat\_id, chat\_type, users));  
 }  
 if (changed) chat\_search(true, changed\_buttons);  
 }  
 setTimeout(load\_chat\_buttons, 100); // update again in 100 milliseconds  
}  
async function load\_users\_buttons() {  
 // calls user\_box\_left for every user  
 users\_list.innerHTML = "";  
 users\_list.appendChild(create\_new\_chat\_or\_group);  
 users\_list.appendChild(search\_for\_non\_familiar\_user);  
 let known\_to\_user = JSON.parse(await eel.get\_known\_to\_user()());  
 let other\_email;  
 for (let index in known\_to\_user) {  
 other\_email = known\_to\_user[index];  
 let picture\_path = `url("${email}/profile\_pictures/${other\_email}\_profile\_picture.png")`;  
 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\_picture.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.backgroundColor = "#232323";  
 else msg\_row.getElementsByClassName("msg\_box")[0].style.backgroundColor = "#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].cloneNode(true);  
 my\_msg\_box.getElementsByClassName("msg\_text")[0].innerHTML = msg;  
 msg\_row.appendChild(my\_msg\_box);  
 msg\_row.getElementsByClassName("my\_msg\_box")[0].style.backgroundColor = "#232323";  
 } else {  
 msg\_row.getElementsByClassName("msg\_box")[0].remove();  
 let msg\_box = window.msg\_row.getElementsByClassName("msg\_box")[0].cloneNode(true);  
 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.backgroundColor = "#232323";  
 msg\_row.getElementsByClassName("msg\_box")[0].style.backgroundColor = "#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.png")`;  
 }  
 // 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(recording\_options\_box);  
 else file\_row.getElementsByClassName("msg\_box")[0].appendChild(recording\_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].innerHTML = 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\_search\_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\_search\_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;600&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\_msgs\_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.\_\_email}{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}".encode() + \  
 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\_email}".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\_email}".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}".encode() + 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\_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 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}{message\_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